MAE207 Applications of complex analysis

Stefan LLEWELLYN SMITH

Spring quarter 2007

In memory of David Crighton.

Introduction (05/29/2008)

0.1 Rationale

Many solution techniques in applied mathematics use complex variable techniques, in many cases leading to closed-form but difficult-to-use solutions. The development of computers in the last 50 years has led to brute-force approaches to many of these problems. However modern computer software for symbolic and technical computing like Maple and Matlab can also be used to evaluate and generalize the analytical solutions. In these notes, we will cover some of these approaches using actual examples and as much of the underlying theory as needed.

0.2 Prerequisites

Some prior knowledge of complex numbers, up to integration using residues, and of linear algebra, including solution of systems of linear equations and eigenvalue problems, is desirable. These notes have a distinctly applied flavor: very few, if any, proofs will appear, and methods will be justified by their reasonableness rather than by theorems and lemmas. Nevertheless, I will attempt not to write down any outright lies and to 'give conditions that are mathematically correct.

Some background in numerical analysis would be helpful, but is probably not absolutely necessary. As mentioned above, the goal here is to use existing software to solve problems using complex analysis, not to derive fundamental results of numerical analysis. Some experience is useful in giving an idea of where the tools that are being used come from and why they work.

0.3 Computer resources

The computer system that I will be using is Matlab[1] There are three main reasons for this:

1. I expect most students to have access to Matlab.
2. It can do what I want. If it can't in its basic form, it probably can using the Maple extensions built in or the symbolic algebra package. Beyond this is more difficult.

[1] In these notes, I will not worry about trademarks and the like. In a real book, there would be ®, © and ™ signs floating around.

We may occasionally require something more, in which case we will use Maple. I dislike Mathematica. And if that's no good, we'll use FORTRAN.

3. I like Matlab.[2]

Matlab calls itself "The Language of Technical Computing" and fulfills the role of an easy-to-use language to do things using mathematical objects. Its syntax is not perfect, but is straightforward if you have seen C previously. Its graphics capabilities are good. Its toolboxes are outstanding. Its extensions like Simulink and the like are pointless.

However, Matlab is not always fast enough. This is because it is an interpreted language. One can compile Matlab, but this is a fairly pointless exercise. Note that linear algebra routines like matrix inversion in Matlab are *fast*: it is the control structures and boring day-to-day operations that slow things down. Hence there will be an occasional need for something faster, in which case FORTRAN is the language of choice. This may seem slightly perverse, given the venerability of FORTRAN and its problems, as well as the fact that Matlab is C-based. Nevertheless, FORTRAN is still the best choice there is to draw upon the existing base of (free) numerical software.

My programming style is not quite as spare as that of Prof. Ierley. However, I recommend using command-line summonings of Matlab and Maple. The GUI rarely adds much value, and Emacs is a better editor than anything else.

0.4 Resources

0.4.1 Books

You may find the following useful:

- Ablowitz, M. J. & Fokas, A. S. *Complex variables: introduction and applications.* Cambridge University Press, Cambridge, 2003. Probably the best advanced undergraduate/graduate level applied complex variable book. Good section on Riemann–Hilbert problems, which is unusual in books at this level.

- Abramowitz. M. & Stegun, I. A., ed. *Handbook of mathematical functions with formulas, graphs, and mathematical tables.* Dover, New York, 1965. If you want to do this stuff, you need to own this. If you want to do this stuff well, you need to read this for fun. Some people like the hardbound government edition. Dover edition cheap, of course.

- Bornemann, F., Laurie, D., Wagon, S. & Waldvogel, J. *The SIAM 100-digit challenge.* SIAM, Philadelphia, 2004. In 2002, Nick Trefethen issued a challenge to readers of SIAM News: solve 10 problems, each to 10 digits. Rather more people met the challenge than he expected. This book presents a fascinating discussion of these problems, using them to introduce a variety of topics in computational mathematics.

[2]Note for UCSD students: one of the developers of Matlab, Loren Shure, was a graduate student in IGPP at SIO. Bob Parker, Professor Emeritus of geophysics, had previously written something very similar for his own use.

Highly recommended, and I don't say that just because you'll find my name in it as a prize winner.

- Carrier, G. F, Krook, M. & Pearson, C. E. *Functions of a complex variable; theory and technique.* SIAM, Philadelphia. New ed., 2005. A classic which has been reprinted. A good source to learn about "advanced" applied complex analysis.

- Duffy, D. G. *Transform methods for solving partial differential equations.* Chapman & Hall/CRC, Boca Raton. 2nd ed., 2004. A detailed, not to say over-detailed exposition of transforms and integrals. Lots of complex analysis of course. Valuable bibliography.

- Gradshteyn, I. S. & Ryzhik, I. M. *Tables of integrals, series and products.* Academic Press, San Diego. 6th ed., 2000. A number of other people have contributed to this. This has most integrals that you'll need in life. A laudable goal is to be featured on the online errata page.[3] There is also a version on CD.

- Henrici, P. *Applied and computational complex analysis.* Wiley, New York, 1974–1986. Three volumes written by a distinguished numerical analyst who combines in a very effective manner theory and applications. Probably the closest book in spirit to these notes.

- Ierley, G. R. *I'm sorry, wrong number.* Unpublished manuscript, 2007. These notes by Prof. Ierley are remarkable on extrapolation and worth reading for an insight on how to get the most bang in precision for your computing buck.

- Press, W. H., Flannery, B. P., Teukolsky, S. A. & Vetterling, W. T. *Numerical recipes...* Cambridge University Press. There are a number of editions and also a web site (see below). A great resource, and can be read online. Watch out though, there are mistakes. Often most useful when combined with canned software to do the boring bits (who wants to type in a Gaussian elimination subroutine?). I'm used to the FORTRAN 2nd edition.

- Trefethen, L. N. *Spectral methods in MATLAB.* SIAM, Philadelphia, 2000. Exceptionally clear introduction to modern spectral methods, using MATLAB programs brilliantly. I will try not to copy this approach too obviously. Good value paperback.

- Needham, T. *Visual complex analysis.* Clarendon Press, Oxford, 1997. An unusual book which emphasizes the geometrical aspects of complex analysis. I'm not sure it works for me, but the pictures are interesting.

0.4.2 Web resources

- `http://cernlib.web.cern.ch/cernlib/` CERNLIB is a library of numerical routines developed at CERN. It used to be proprietary but is now freely available. Good

[3] Guess who is...

for some of the more abstruse manifestations of special functions, such as Bessel functions of imaginary order.

- `http://www.cpc.cs.qub.ac.uk/` CPC Program Library.
- `http://www.mathtable.com/` Daniel Zwillinger's web site. He is probably the most active special function and mathematical tables person out there. Useful errata, e.g. for G&R.
- `http://mathworks.com/` Has the Matlab documentation, which is useful if you don't like running Matlab in a GUI. Also contains contributed Matlab software which can be useful. There is also an extensive support site, which can be useful if you're having technical difficulties with linking Matlab and FORTRAN and so on, but not really otherwise.
- `http://tonic.physics.sunysb.edu/docs/num_meth.html` Good links to numerical methods resources on the Internet.
- `http://www.gnu.org/software/gsl` The GNU Scientific Library.
- `http://www.nr.com/` The Numerical Recipes web site.
- `http://www.olemiss.edu/sciencenet/benet/` Boundary Element Resources Network contains lots of useful information about boundary element methods (BEMs). We will not touch on these much, but they are closely linked to some of the things we will be discussing.
- `http://dlmf.nist.gov/` Digital Library of Mathematical Functions. The successor project to A&S. Seems stalled at the moment.
- `http://gams.nist.gov/` Excellent search engine and search tree linking to many different public domain codes.
- `http://www.netlib.org/` Most venerable numerical software repository. `gams` links to it, and so on.

0.4.3 Software

This can be broken down into two main groups. First, integrated packages/suites/-technical languages. These break down into computer algebra-type programs, which range from extremely abstruse and special-purpose (MACAULEY) to what most people would recognize as symbolic algebra programs (Mathematica, Maple, Reduce) to more numerically-oriented programs (Matlab, IDL). The distinctions are blurred however. Mathematica and Maple have extensive numerical capabilities, and Matlab includes symbolic algebra facilities.

Second, we have the general purpose computer language (FORTRAN, C, C++ and so on) combined with appropriate software. In general, you need to be fairly eccentric these days to write your own low-level mathematical software (Gaussian elimination,

quadrature, etc...), although I know plenty of people who have done so. It is a *good* thing to know how to do so, and in most cases these people have a very good reason for doing so. As mentioned above, I use FORTRAN. You can use C or C++; don't expect me to debug it.

The list of mathematical software libraries and packages is too long to mention, ranging from specialized and application-oriented (all the way up to very expensive commercial codes like FLUENT) to general and low-level. We won't be dealing with the former. From the latter, you should be aware of libraries like BLAS, LAPACK, ARPACK, SLATEC, QUADPACK, CERNLIB.

Warning 1 [4]

At a fundamental level, computers operate on numbers (integers in fact: you should be aware that a lot of effort has gone on behind the scenes to deal with real numbers in a sensible way). Computers have no concept of infinity or limiting processes. Therefore, anything we do will boil down to solving sets of linear equations. If your computer doesn't do that fast, you will have trouble as things become more accurate and more complicated.

Obtaining very high-precision results can be important for certain applications. Standard IEEE double precisions gives about 16 digits of accuracy. To do better, one needs multiple or arbitrary precision software. Most symbolic algebra packages use exact arithmetic, or as much as possible in the case of radicals, and then have user-configurable precision (50, 100, ...digits). There are also FORTRAN multiple-precision package: MP and ARPREC, which are worth playing with. There is also the GNU MP Bignum library, with which I am not familiar.

0.5 Overview of the material

The following is a quick overview of the 20 chapters. I have tried to make each chapter a little lecture, and also to have the chapters follow in a logical order. I may have failed on both fronts; in particular there may be too much material for 20 lectures.

1. Introduction and review of complex numbers
2. Branch cuts
3. Finding zeros
4. ODEs in the complex plane
5. Integral transforms
6. FFTs

[4] Warning are things we would do well to remember. Forgetting them won't kill you as you sit in front of your computer, but the output might be wrong, or might take forever to compute. The former might kill someone else in the long run if you have a real job; the latter might kill your job in the long run if you're a scientist.

7. Solving ODEs using integral transforms
8. Laplace transforms
9. Multiple transforms: Cagniard-de Hoop method
10. Multiple transforms: absolute and convective instability
11. Wiener–Hopf method
12. Matrix factorizations
13. Integral equations and principal-value integrals
14. WKB and ray theory
15. Laplace's equation
16. Conformal maps
17. Elliptic functions
18. Special functions
19. Chebyshev expansions
20. Orthogonal polynomials

Sprinkled into the text you will find theorems, quasitheorems, exercises and examples. Quasitheorems are theorems that I have stated somewhat imprecisely. The results are not wrong, but often conditions for their validity are lacking. Full versions can be found in the references above.

Chapter 1

Introduction (04/14/2008)

We rapidly review complex numbers and more particularly complex analysis. Our discussion is brief, and concentrates on building up as quickly as possible the tools that we will use. In lieu of extended discussions, we try and provide one example of everything.

1.1 Complex numbers

1.1.1 Definition

We start be reviewing complex numbers. We define two operations, "addition" and "multiplication", on pairs (x, y) of reals as follows:

$$\begin{aligned} (x_1, y_1) + (x_2, y_2) &= (x_1 + y_1, x_2 + y_2), & (1.1) \\ (x_1, y_1) \times (x_2, y_2) &= (x_1 x_2 - y_1 y_2, x_1 y_2 + x_2 y_1). & (1.2) \end{aligned}$$

One can verify that $(0, 0)$ is the identity for addition, and every element (x, y) has the inverse $(-x, -y)$ under addition. Hence subtraction is defined. The element $(1, 0)$ is the identity under multiplication. Every element except for $(0, 0)$ has an inverse as can be seen from

$$(x, y) \times \left(\frac{x}{x^2 + y^2}, -\frac{y}{x^2 + y^2} \right) = (1, 0). \quad (1.3)$$

We see that $(\mathbb{C}, +, \times)$ form a field, and drop the $\times$ sign from now on. By identifying $\mathbb{R} \times \mathbb{R}$ with $\mathbb{C}$ under the map $(x, y) \to z = x + \mathrm{i}y$, we have defined the complex numbers. We identify elements of the form $(x, 0)$ with real numbers. While i in this approach can initially be viewed as an abstract symbol, considering the element $(0, 1)$ shows that $\mathrm{i}^2 = -1$ shows that i is the complex unit.

The reason i was introduced is that the set $\mathbb{C}$ is algebraically closed, i.e. every polynomial has a zero in $\mathbb{C}$. (This is of course not true in $\mathbb{R}$, as shown by the polynomial $x^2 + 1$.) This is a theorem, given the above definition: the fundamental theorem of algebra.

Note that $\mathbb{C}$ is also a vector space under multiplication by reals.

1.1.2 Geometry

The identification of a complex number with a pair of real numbers leads immediately to the Argand diagram, where $z = x + iy$ is associated with the point in the plan with coordinates (x, y). Some of the algebraic operations acquire geometric meanings then: $-z$ is the reflection of z through the origin, $z_1 + z_2$ is the vector sum of the two vectors associated with points (arrowhead at the point, tail at the origin), and we can now think about distances and angles.

We write the distance from the origin of the point z as $|z| \equiv \sqrt{x^2 + y^2}$; this is the modulus of z (sometimes written as r when polar coordinates are being used). Defining the complex conjugate of z, $\bar{z}$, by $\bar{z} = x - iy$ gives $|z|^2 = z\bar{z}$. The angle made by the line (vector, whatever) to z is the argument of z; clearly $(x, y) = |z|(\cos\theta, \sin\theta)$. Then $\tan\theta = y/x$ in general, but one has to be careful about cases like $x = 0$ and what happens in different quadrants of the complex plane. These may seem like trivial issues, but can lead to real problem in numerical implementations.

Warning 2 *An equation that is true or "obvious", as a relation between complex entities, can fail as part of a numerical implementation.*

The argument θ is safe whenever it appears inside a sine, cosine, or exponential function. Recalling the series for these, we see that in polar coordinates

$$z = r\cos\theta + ir\sin\theta = re^{i\theta}. \tag{1.4}$$

Now we blithely use the results we know about the exponential function, obtaining some trigonometric identities on the way. This shows that multiplication is a "twist and scale" operation:

$$z_1 z_2 = (r_1\cos\theta_1, r_1\sin\theta_1)(r_2\cos\theta_2, r_2\sin\theta_2) = r_1 r_2 e^{i(\theta_1+\theta_2)}. \tag{1.5}$$

The modulus of the product is the produce of the moduli. The argument of the product is the sum of the arguments (watch out for the modulo 2π aspect).

It is sometimes useful to think of the complex plane as a simply-connected set, i.e. as a sphere. Then the point at infinity is part of this sphere, which is called the Riemann sphere. This geometric picture can be very useful when thinking about multivalued functions.

Exercise 1.1 *Write a two-paragraph discussion of the Riemann sphere.*

1.2 Analyticity

A central notion in this book will be that of analyticity. A number of the techniques that we consider will explicitly be concerned with constructing analytic functions. This will usually be the most difficult part of the problem. Once a certain function has been constructed with the appropriate analyticity properties, one is usually home and dry.

To begin, we need the notion of a function. A function $f(z)$ is a rule that to the complex number z associates another complex number f. We will return to this issue when talking

about multivalued functions (sometimes called multifunctions). The range of f is the subset of $\mathbb{C}$ of values $f(z)$; f is one-to-one if $f(z_1) = f(z_2) \Rightarrow z_1 = z_2$.

We will often write $f = u + \mathrm{i}v$. We will also be alarmingly slapdash about the arguments. When it suits us, f will be a function of z, of x and y, or of z and $\bar{z}$.

1.2.1 Differentiation

The complex function f is differentiable at the point z if the limit

$$f'(z) \equiv \lim_{h\to 0} \frac{f(z+h) - f(z)}{h} \tag{1.6}$$

exists *for all complex h as $h \to 0$*. We have not defined limits and so on: these notions are taken to be understood, if only informally, for real numbers, and we are discussing the extension to the complex case.

The fact that (1.6) has to hold for all h, i.e. in all directions about the origin, is fantastically important. It leads to entirely unexpected consequences; to be brief we can say that complex differentiable functions are nice.

We can use this property to derive in a very simple manner the Cauchy–Riemann equations. First take h to be real. Then $f(z+h) = f(x+h, y)$ and $\mathrm{d}f/\mathrm{d}z = \partial/\partial x f(x,y)$. Similarly is we write $h = \mathrm{i}k$, $\mathrm{d}f/\mathrm{d}z = \partial/\partial(\mathrm{i}y) f(x,y)$. Identifying real and imaginary parts gives the Cauchy–Riemann equations:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}. \tag{1.7}$$

Theorem 1 *If f is analytic, then (1.7) hold. Conversely, if u and v are continuous and (1.7) hold, then f is analytic.*

It is sometimes useful to consider z and $\bar{z}$ as independent variables. We can change variables for an arbitrary function by our usual abuse of notation, $f(z, \bar{z}) = f(x, y)$. Then the chain rule gives, at least formally,

$$\frac{\partial}{\partial x} = \frac{\partial}{\partial z} + \frac{\partial}{\partial \bar{z}}, \qquad \frac{\partial}{\partial y} = \mathrm{i}\frac{\partial}{\partial z} - \mathrm{i}\frac{\partial}{\partial \bar{z}}. \tag{1.8}$$

Note that this leads to

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} = 4\frac{\partial^2}{\partial z \partial \bar{z}}. \tag{1.9}$$

The term analytic is sometimes used interchangeably with differentiable, and sometimes not. So is the word holomorphic. We will just use analytic from now on. So the complex function f is analytic in a subset of the complex plane S is (1.6) holds at all points of S.

We can check analyticity by hand to build intuition:

Example 1.1 *The function $f(z) = z^m$, with m integer, is differentiable everywhere in $\mathbb{C}$. We write*

$$\frac{(z+h)^m - z^m}{h} = mz^{m-1} + \frac{m(m-1)}{2}hz^{m-2} + \cdots \to mz^{m-1}, \tag{1.10}$$

which holds for all z.

We can see where it fails in harder situations:

Example 1.2 *The function $f(z) = |z|^2$ is differentiable only at the origin. We have*

$$\frac{(z+h)(\bar{z}+\bar{h}) - z^2}{h} = \bar{z} + \frac{\bar{h}}{h}z. \tag{1.11}$$

This does not have a unique limit as $h \to 0$ (since the second term depends on the argument of h), except when $z = 0$.

However, in most cases we use the following shortcut:

Tip 1 *The obvious functions that we know and love (polynomials, exponential, sine, logarithm, etc...) are analytic except at points where there's an obvious problem.*

1.2.2 Taylor and Laurent series

An equivalent definition of analyticity is by use of Taylor series: the function f is analytic at z_0 if it has a convergent expansion of the form

$$f(z_0 + h) = a_0 + a_1 h + \tfrac{1}{2}a_2 h^2 + \cdots, \tag{1.12}$$

where we identify a_1 with $f'(z_0)$. Given our definition (1.6), this is a theorem. We do not prove it.

Unlike the real-variable case, we have another, very useful, kind of series: the Laurent series. If the function f is analytic in the annulus $0 < |z - z_0| < d$, then it can be written as

$$f(z_0 + h) = \cdots + a_{-1}h^{-1} + a_0 + a_1 h + \tfrac{1}{2}a_2 h^2 + \cdots, \tag{1.13}$$

where the sum can in principle extend infinitely far to the left. In that case the point z_0 is called an essential singularity. Otherwise, it is a pole.

This definition excludes points of non-analyticity where are not isolated. These includes points of accumulation such as $1/(\sin z^{-1})$ which is non-analytic at $z = (n\pi)^{-1}$ for all integer n and branch points. We will have more to say about the latter.

An entire function has no singularities in the complex plane (excluding the point at infinity). This means no poles, no non-isolated singularities, no branch points.

1.3 Multivalued functions

Our definition in words of function as a rule implies that to each z we associate one $f(z)$. This is the usual procedure, and certainly necessary for computer implementation. However, there are times when a more general definition is useful.

A multivalued function or multifunction is a rule that to each z associates one or more $f(z)$. For example, we can define the multifunction $\sqrt{z}$ to be one of the complex number that, when squared, give z. The usual real-valued definition of $\sqrt{x}$ picks the positive root of $y^2 = x$. However for complex numbers, we want square roots of all z.

Dealing with multifunctions leads naturally to the idea of Riemann surfaces: $\mathbb{C}$ is mapped to a more complicated geometric structure, with different copies of $\mathbb{C}$ stitched together.

From a practical perspective, we will not use multifunctions in their raw form (it can be done), but instead introduce branch cuts to make the functions single-valued. We pay a price: functions are no longer continuous in places where they were previously. For now, we defer the discussion to Chapter 2.

1.4 Analytic continuation

In $|z| < 1$, we write

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots, \tag{1.14}$$

and the sum converges. We can think of this another way: reading from left to right, we see that the sum is the same as the rational function in the unit disk. The two sides are just different ways of writing the same function. Hence we use the equality to extend the sum to the entire complex plane, nothing that the point 1 has issues. This is essentially analytic continuation in a nutshell.

Quasitheorem 1 *If f_1 and f_2 are defined over subregions D_1 and D_2 of $\mathbb{C}$ and $f_1(z) = f_2(z)$ in $D_1 \cap D_2$, and this region is* enough of the complex plane*, namely a non-trivial curve or set, then f_1 provides an analytic continuation of f_2 to D_1.*

Again, we have been vague. This is only interesting if D_1 is bigger than D_2 (it may contain D_2.) In the previous example, we had $D_1 = \mathbb{C} \setminus \{1\}$ and D_2 the unit disk.

Example 1.3 *Consider the integral*

$$F(z) = \int_0^\infty \mathrm{e}^{-tz}\,\mathrm{d}t = \frac{1}{z} \tag{1.15}$$

when Re $z > 0$ (otherwise the integral does not converge). The function F is analytic in this domain of definition, and the equality holds there. Hence z^{-1} is its analytic continuation to the $\mathbb{C} \setminus \{0\}$. This kind of argument is very common with Mellin and Laplace transforms.

1.5 Liouville's theorem

Theorem 2 *A bounded entire function is constant.*

This is amazing. More generally, a function satisfying $|f(z)| < c|z|^m$ for some integer m and z sufficiently large is a polynomial.

Exercise 1.2 *Use Liouville's theorem to prove the fundamental theorem of algebra: every polynomial has a root in* $\mathbb{C}$.

1.6 Integration

1.6.1 Cauchy's theorem

Integrals in the complex plane can be defined using parameterizations. We shall not worry about distinctions between curves, contours, or smoothness. All the obvious properties of integration based on linearity work.

Example 1.4 *This is a useful integral:* $I_m = \int_C z^m \, \mathrm{d}z$. *when* C *is the unit circle. Parameterize the unit circle by* $z = \mathrm{e}^{\mathrm{i}\theta}$, *so that* $\mathrm{d}z = \mathrm{i}\mathrm{e}^{\mathrm{i}\theta} \, \mathrm{d}\theta$. *Then*

$$I_m = \int_0^{2\pi} \mathrm{i}\mathrm{e}^{\mathrm{i}(m+1)\theta} \, \mathrm{d}\theta = 0 \qquad (m \neq -1) \tag{1.16}$$

by the properties of trigonometric functions. Clearly $I_{-1} = 2\pi\mathrm{i}$.

The fundamental theorem of calculus also works, and if one thinks about applying it to a closed curve, one would like to argue that the result always vanishes. This is almost true: the correct version is Cauchy's theorem.

Quasitheorem 2 *If the function* f *is analytic inside a closed curve* C, *then*

$$\int_C f(z) \, \mathrm{d}z = 0. \tag{1.17}$$

This is a quasitheorem because we have not worried about the smoothness of the curve C. The conditions on f are fine (note that Cauchy's original proof required f to be analytic on the curve; Darboux showed this was unnecessary).

We can hence deform contours in regions where the integrand is analytic: adding connecting lines and checking appropriate cancellations (running along the same arc in two different directions) shows that deforming a contour makes no difference. Hence the result above, $I_m = 2\pi\mathrm{i}\delta_{m,-1}$, is valid for any contour enclosing the origin.

Exercise 1.3 *Prove Green's theorem in the plane from Cauchy's theorem.*

1.6.2 Integration by residues

The fact that we can deform contours except when the integrand has a pole inside it and the result for I_{-1} above leads to the following result:

Quasitheorem 3

$$\int_C f(z)\,\mathrm{d}z = 2\pi\mathrm{i}\sum_n Res(f; z_n), \tag{1.18}$$

where $Res(f; z_0)$ denotes the residue, that is the -1 term in the Laurent series, at the singularity z_n.

If the integral exists, one doesn't have to worry too much about the behavior of f along C. I have sidestepped issues with non-isolated singularities, which will not crop up here.

For a function $f(z)/g(z)$ that has a simple pole at z_0, the residue is given by $f(z_0)/g'(z_0)$ (assuming that common terms have been cancelled out).

We use this to compute real integrals. The name of the game is to find a contour such that the parameterization of the integral over part of the contour gives what the desired real integral, while the rest of the integral vanishes or can be dealt with somehow.

The classic example is the following. Note that versions of this which require dealing with a simple pole at the origin, or that use Cauchy principal values, are missing the point.

Example 1.5 *Compute*

$$I = \int_0^\infty \frac{\sin x}{x}\,\mathrm{d}x. \tag{1.19}$$

First note that we can extend the range to the whole real line by adding a factor of $\frac{1}{2}$. The next step is to come up with a contour integral. The natural candidate is

$$I_1 = \tfrac{1}{2}\int_C \frac{\sin z}{z}\,\mathrm{d}z. \tag{1.20}$$

Unfortunately, the function $\sin z$ blows for large positive and negative imaginary values, so any semicircular contour will have an unbounded integrand. Exponentials are better behaved, so let's use the relation $\sin z = (\mathrm{e}^{\mathrm{i}z} - \mathrm{e}^{-\mathrm{i}z})/(2\mathrm{i})$.The real problem is the contour then, since $\mathrm{e}^{\mathrm{i}z}/z$ blows up at the origin. However, the choice of contour for the original function is immaterial, so we can deform the contour off the real axis, either up or down. We take it up to C_+. We now work with

$$I_2 = \int_{C_+} \frac{\mathrm{e}^{\mathrm{i}z} - \mathrm{e}^{-\mathrm{i}z}}{4\mathrm{i}z}\,\mathrm{d}z = \int_{C_+} \frac{\mathrm{e}^{\mathrm{i}z}}{4\mathrm{i}z}\,\mathrm{d}z - \int_{C_+} \frac{\mathrm{e}^{-\mathrm{i}z}}{4\mathrm{i}z}\,\mathrm{d}z. \tag{1.21}$$

The first integral on the right-hand side can be closed in the upper half-plane and has no poles inside the contour, hence it vanishes. The second can be closed in the lower half-plane. There is a simple pole at the origin with residue 1. Hence $I_2 = \pi/2$. Note the minus sign from the fact that the contour in the lower half-plane is described in the negative sense. Now $I = I_2 +$ stuff that goes away, so we have $I = \pi/2$.

As a lesson from that example, we note the following:

Warning 3 *Analyticity in $\mathbb{C}$ is very nice. However, functions sometimes have different behavior in the complex plane compared to the real line. Sine and cosine are no longer bounded (this is a classic mistake).*

It's worth practising contour integration. Be efficient:

Tip 2 *Don't worry about detailed proofs that contributions along parts a contour vanish. Use orders of magnitude. Large circles or semicircles with radius R behave OK provided the integrand is $o(R^{-1})$. The only case where one has to be careful is $O(R^{-1})$ along a semicircle. There one actually has to use Jordan's lemma, which I do not include here on principle.*

This tip uses order symbols. These are all valid in some limit as z tends to something, so they hold for z close enough to that point. Briefly, $f = O(g)$ if f/g is bounded and $f = o(g)$ if $f/g \to 0$.

One can do many more integrals this way. Note the use we made of symmetry in the integral there. Clearly the integral of $x \sin x/(1+x^4)$ would work similarly.

Exercise 1.4 *What about the integral of* $\sin x/(1+x^3)$*? This is no longer elementary and you will need special functions.*

Example 1.6 *Finally, we look at*

$$L(z) = \int_1^z \frac{\mathrm{d}u}{u}, \tag{1.22}$$

when z is imaginary. The path of integration doesn't matter, provided it doesn't go through the origin. Take it to be from 1 to $|z|$ and then along an arc from $|z|$ to $z = |z|\mathrm{e}^{\mathrm{i}\theta}$. Using the obvious parameterizations, we find

$$L(z) = \int_1^{|z|} \frac{\mathrm{d}x}{x} + \int_0^{\theta} \mathrm{i}\,\mathrm{d}\theta = \log|z| + \mathrm{i}\theta. \tag{1.23}$$

The logarithm above is the usual real logarithm. The presence of θ should immediately alert us to the possibility of multivaluedness: we have not specified how many times around the origin our path goes and there are many possibilities differing by multiples of $2\pi\mathrm{i}$. Clearly $L(z)$ is the complex logarithm, and we have shown that it is a multifunction. The principal branch has $-\pi < \theta \leq \theta$. We shall return to this later. Note that the complex logarithm is the inverse of the complex exponential function, which is invariant under the addition of $2\pi\mathrm{i}$. The integral I_{-1} is an integral of $L(z)$ around a closed curve encircling the origin. Then θ varies smoothly and changes by 2π, giving the required answer.

1.6.3 Cauchy's integral formula

Theorem 3 *If $f(z)$ is analytic inside and on the contour C, then for any interior point of z,*

$$f(z) = \frac{1}{2\pi\mathrm{i}} \int_C \frac{f(\zeta)}{\zeta - z}\, dd\zeta. \tag{1.24}$$

This formula can be differentiated as often as we wish. Hence an analytic formula is differentiable infinitely many times, which is definitely not the case for a real differentiable function (e.g. $|x|x$).

We will be very interested in integrals of the form

$$F(z) \int_C \frac{\varphi(\zeta)}{\zeta - z} \, \mathrm{d}\zeta, \tag{1.25}$$

known as Cauchy-type integrals, in what follows.

1.6.4 Cauchy principal value integrals

We have glossed over the definition of improper integrals. Formally we have for a real integral

$$\int_{-\infty}^{\infty} f(x) \, \mathrm{d}x = \lim_{M_- \to \infty, M_+ \to \infty} \int_{-M_-}^{M_+} f(x) \, \mathrm{d}x \tag{1.26}$$

when the limit exists. The Cauchy principal value corresponds to taking $M_- = M_+$ in the limit. It exists when the integral exists, and sometimes when it does not. Similarly, when $f(x)$ has a simple pole along the contour, say at the origin, we can do the same thing and write

$$⨍_a^b f(x) \, \mathrm{d}x = \left(\int_a^\epsilon + \int_\epsilon^b \right) f(x) \, \mathrm{d}x. \tag{1.27}$$

Often we will detour around simple poles and the limit becomes a Cauchy principal value integral. These integrals are not limited to integration along the real axis.

Example 1.7 *If $f(z)$ is analytic at the origin and decays at infinity, then*

$$⨍_{-\infty}^{\infty} \frac{f(z)}{z} \, \mathrm{d}z = ⨍_{-\infty}^{\infty} \frac{f(z) - f(0)}{z} \, \mathrm{d}z + f(0) ⨍_{-\infty}^{\infty} \frac{\mathrm{d}z}{z} = \int_{-\infty}^{\infty} \frac{f(z) - f(0)}{z} \, \mathrm{d}z. \tag{1.28}$$

The first integral on the right-hand side is a Cauchy principal value integral because of the behavior at infinity. The second integral is zero:

$$⨍_{\infty}^{\infty} \frac{\mathrm{d}z}{z} = \lim_{\epsilon \to 0, M \to \infty} \left(\int_{-M}^{-\epsilon} + \int_\epsilon^M \right) \frac{\mathrm{d}z}{z} = \log \frac{|-\epsilon|}{|-M|} + \log \frac{M}{\epsilon} = 0. \tag{1.29}$$

1.7 Plemelj formulae

Consider what happens in a Cauchy-type integral $F(z)$ as in (1.25) as z tends to the contour of integration C. Write F^+ for the result as z approaches C from the left (with respect to the direction of integration) and F^- for the result as z approaches C from the right.

$$F^{\pm}(z) = \pm \tfrac{1}{2} \varphi(z) + \frac{1}{2\pi \mathrm{i}} ⨍_C \frac{\varphi(\zeta)}{\zeta - z} \, \mathrm{d}z. \tag{1.30}$$

These formulae underlie a lot of results on integral equations and Riemann–Hilbert problems.

1.8 The principle of the argument

Consider the integral

$$J = \frac{1}{2\pi \mathrm{i}} \int_C \frac{f'(z)}{f(z)} \, \mathrm{d}z = [\arg(f(z))]_C. \tag{1.31}$$

If f is well-behaved along the contour, this will be a multiple of 2πi. The result will depend on the number of singularities of the function inside the contour. When f has a simple zero at z, we have $f(z+h) = hf'(z) + \cdots$, so $f'(z)/f(z) \sim h^{-1} + \cdots$. If f has a zero of order n, $f'(z)/f(z) \sim nh^{-1} + \cdots$ The same argument with poles gives minus signs, so equating the two sides, we have

$$\frac{1}{2\pi \mathrm{i}} \int_C \frac{f'(z)}{f(z)} \, \mathrm{d}z = Z - P, \tag{1.32}$$

where Z and P are the numbers of zeros and poles, counted with multiplicity.

Exercise 1.5 *Discuss the integral*

$$\int_C z \frac{f'(z)}{f(z)} \, \mathrm{d}z. \tag{1.33}$$

See Chapter 3

1.9 Physical applications

Complex numbers are unjustly tarred with the appellation imaginary, and one could ascribe their creation to the mathematical aim of solving $x^2 + 1 = 0$. However, their origin lies in the very practical goal of solving cubic equations and obtaining three real solutions, two of which involve complex arithmetic when using Cardan's formula. Complex analysis is of enormous help in solving a number of physical application, which we describe briefly.

1.9.1 Waves

The definition of the complex exponential in terms of trigonometric functions shows that complex analysis is uniquely suited to describing periodic systems and waves, which after all are periodic disturbances in a medium.

The use of complex numbers ranges from efficient ways of solving *LRC* circuits, where differentiation maps to multiplication by an imaginary number, through Fourier transforms and FFTs, which underlie modern signal processing, to quantum mechanics and nonlinear scattering theory in integrable systems.

1.9.2 Integral transforms

Fourier's theorem states that any periodic function can be decomposed into a sum of harmonics. This generalizes to Fourier transforms, which apply to (almost) any real functions. We write

$$F(k) = \int_{-\infty}^{\infty} f(x) \mathrm{e}^{-\mathrm{i}kx} \, \mathrm{d}x, \qquad f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(k) \mathrm{e}^{\mathrm{i}kx} \, \mathrm{d}k. \tag{1.34}$$

I will try and stick to this notation (x and k) for transforms in space, and use t and ω for transforms in time. It can be useful to use the opposite convection in the latter case. Sometimes I may use α as the transform variable.

Many other integral transforms exist. All rely on some version or extension of Fourier's inversion theorem to obtain an inverse transform. Examples are Laplace transforms (one-sided in our case), Mellin transforms, Hankel transforms, Kantorovich–Lebedev and others.

1.9.3 Laplace's equation

The other main application of complex analysis comes from (1.9): harmonic functions, i.e. functions that satisfy Laplace's equation, can be obtained from analytic functions. Boundary conditions on the harmonic functions become conditions on the analytic functions at the boundary.

Laplace's equation occurs in fluid mechanics (potential flow), heat transfer (steady-state solutions), elasticity, potential theory and many others. The biharmonic equation, $\nabla^4 \phi = 0$, can be attacked using complex methods. It occurs in Stokes flow and linear elasticity.

These approaches are limited to two dimensions usually.

Chapter 2

Branch cuts (04/10/2008)

We have introduced multifunctions previously. We now treat branch cuts.

2.1 Branch cuts

First, we call a branch point of f a point in $\mathbb{C}$ around which the function f does not return to its original value as we trace out a closed curve. Typical example: the origin for $\sqrt{z}$ or $\log z$. In fact infinity is also a branch point for both these functions, and for geometrical reasons, there has to be more than one branch point for a function. This notion of treating infinity on a par with other points has already come up when the Riemann sphere was mentioned.

Quasitheorem 4 *Anything that can be done with a function $f(z)$ at a point z in $\mathbb{C}$ can be done with the point at infinity by setting $t = z^{-1}$ and proceeding as usual.*

Then a branch cut is an arbitrary curve in $\mathbb{C}$ joining two branch points. We have enormous freedom in how we pick a cut, so we have replaced a multifunction by an infinite family of functions with different branch cut structures. In addition to the curve, we must pick a branch by specifying the value of the function at a point somewhere. From a computer science perspective, a branch of a function is an abstract data structure including f, the topology of the cuts and the value chosen for $f(z_0)$. This is more than we want to deal with usually, so we will try to be efficient.

Note that in physical problems, the complex plane is a place we visit to find a solution that must be expressed in terms of real numbers.[1] Hence the answer we find cannot depend on the particular choice of branch cut topology we use.

Warning 4 *If the final, physical, answer you find depends on the branch cuts you have picked, you have probably made a mistake. This does not mean that different parts of your solution can depend on the branch cuts cannot individually depend on the branch cuts (leaky waves and so on). The final answer cannot.*

[1] This is an approximation of course: continuum mechanics does not hold at the molecular scale and below. More abstrusely, it is sometimes suggested that all lengths in the universe or time intervals are discrete multiples of some fantastically small fundamental units. Who knows?

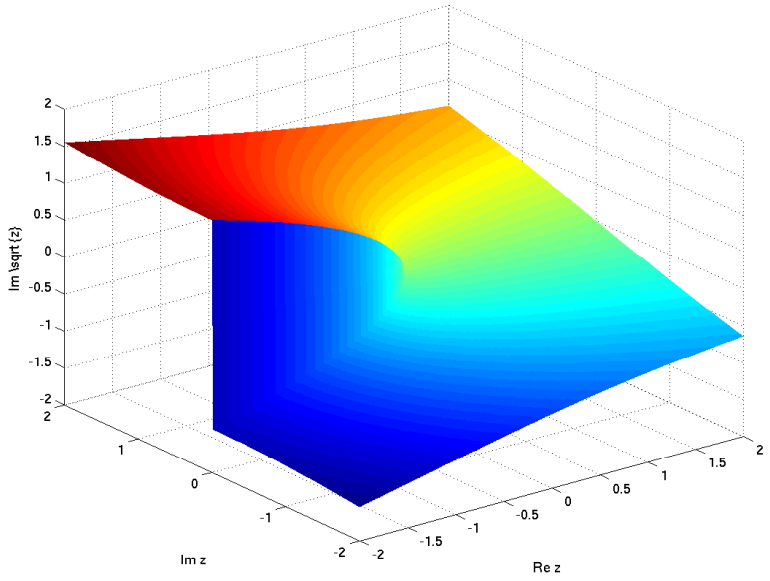

Figure 2.1: Contour plot of the imaginary part of $\sqrt{z}$ using the native Matlab branch cut.

This also does not hold if one is using a complex function to represent a discontinuous real function. But there the branch cut was not picked arbitrarily as part of the method of solution; it came from the physics.

2.2 Matlab implementation

The prototypical multifunctions are the square root and logarithm functions. They are of course related, since $\sqrt{z} = \exp\left(\frac{1}{2}\log z\right)$. Let's look at the the imaginary part of $\sqrt{z}$ in Matlab. The code of program `p21.m`: calculates $\sqrt{z}$ in a square of the complex plane and produces a surface plot of the output, as shown in Figure 2.1.

```
% p21: square root branch cut; 04/02/2008 SGLS
x = -2:.01:2; y = x;
[x,y] = meshgrid(x,y); z = x + i*y;
f = sqrt(z);
surf(x,y,imag(f)); shading flat
xlabel('Re z'); ylabel('Im z'); zlabel('Im \sqrt {z}')
```

We see a discontinuity along the negative real. In fact, Matlab is using the principal branch of the logarithm, which can't be seen in Figure 2.1. To verify this, type `log(-1)` in Matlab and obtain `0 + 3.1416i`, corresponding to the principal branch of axis and values between $-\pi$ and π.

This is fine, but what if we want a different branch cut? Straight branch cuts are very easy. The point about the Matlab branch is that the cut is along $z = -x$ for positive real

x. Hence the function $g(z) = e^{-i\alpha/2}\sqrt{ze^{i\alpha}}$, which is certainly a solution of $g(z)^2 = z$, has a branch cut when $z = -xe^{-i\alpha}$, i.e. along a ray making angle $-\alpha$ with the real axis. So if we replace `sqrt(z)` by `exp(i*pi/5)*sqrt(z*exp(-2*i*pi/5))`, we obtain a branch cut at making an angle of $2\pi/5$.

For curved branch cuts, we are out of luck. One has to write a program that sets the argument of the logarithm or other multifunction by hand.

2.3 More complicated branch cut structures

The more branch points, the more branch cuts and the more ways of joining them up:

Exercise 2.1 *How many ways are there of choosing branch cuts that are topologically distinct for n branch points?*

Certain functions recur again and again in physical problems, the most common being $\beta(z) = \sqrt{z^2-1} = \sqrt{z-1}\sqrt{z+1}$ and $\gamma(z) = \sqrt{z^2+1} = \sqrt{z-i}\sqrt{z+i}$. Actually these equalities are not necessarily true in Matlab, in the sense that one can calculate left-hand and right-hand sides and obtain different answers. Let's have some new notation.

Warning 5 *We will use the symbol $\overset{\mathbb{C}}{=}$ to denote an equality that is true for complex numbers or functions, but not necessarily for the results of a Matlab calculation; e.g.*

$$\log xy \overset{\mathbb{C}}{=} \log x + \log y. \tag{2.1}$$

Let's look at $\beta(z) = sqrtz^2 - 1$. The way to handle this is to think of a length of string tied to nails at ± 1. Start on the real axis at $x > 1$. We need to pick a branch and the sensible one is real and positive. Then we go around the branch cut, as if we had a pencil keeping the string drawn taut. The complex number $z-1$ can be thought of as a vector from 1 to z, so on the positive real axis it initially has argument $\theta_1 = 0$. The complex number $z+1$ is a vector from -1 to z, once again initially with argument $\theta_{-1} = 0$. As the tip of the pencil moves to be just above the branch cut, the argument of $z-1$ increases to 2π. As it moves further left beyond the branch cut, the argument of $z+1$ now changes to π. The pencil tip now loops around to be under the branch cut: the argument of $z+1$ has increased to 2π. Finally the pencil tip reappears to the right and the argument of $z-1$ increases to 2π. We use the result $\beta(z) = \sqrt{|z^2-1|}e^{i(\theta_1+\theta_{-1})/2}$. Combining the results we get positive real values on the positive real axis $x > 1$, negative real values on the negative real axis $x < -1$, positive imaginary above the branch cut, negative imaginary below the real axis. Note that this function depends only on z^2, but $f(z) \neq f(-z)$ for real $|z| > 1$.

Warning 6 *Complex functions with branch cuts do not always have the symmetry properties you might expect.*

The Matlab code in `p22.m` gives the picture in Figure 2.2. The top row uses the function `sqrt(z.^2-1)` which is not the correct branch cut. Solving $z^2 - 1 = -x^2$ gives

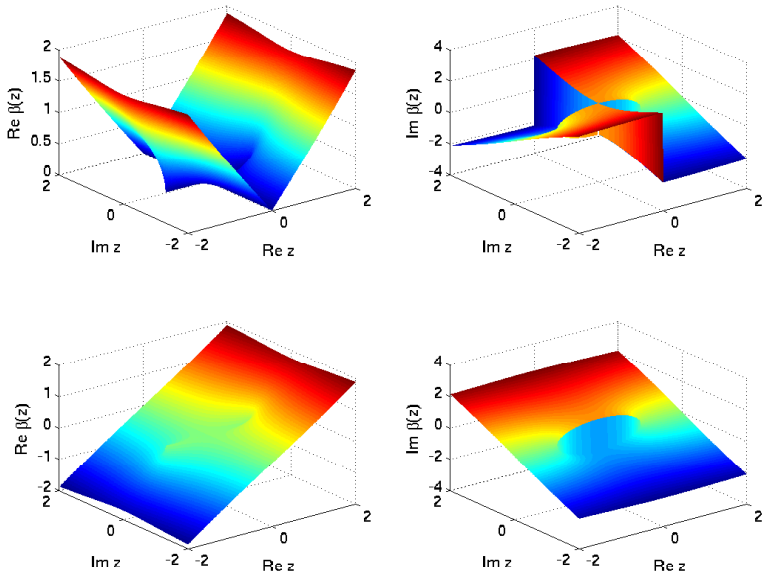

Figure 2.2: Contour plot of the real and imaginary part of $\beta(z)$ using several different branch cuts.

$z = \sqrt{1-x^2}$; this is for $x > 0$, so the branch cut contains parts of the imaginary axis as well as the real axis. The bottom row uses `sqrt(z-1).*sqrt(z+1)`. Now the branch cuts are at $z \pm 1 = -x^2$, and the portion left of -1 cancels, leaving the branch cut between -1 and 1 as desired. The right-hand column shows the imaginary part, which vanishes on the real axis outside the branch cuts and does not show the antisymmetry on the real axis so obviously.

The functions $\beta(z)$ and $\gamma(z)$ is commonly encountered in wave propagation problems for monochromatic waves (Helmholtz-type equations). If the function under consideration is an inverse transform, the presence of singularities on the inversion contour, the real axis, is usually a warning of some kind of causality or radiation condition. It is usually safer to add a small imaginary part to take the singularity off the real axis.

Example 2.1 *The function* $|z|_\epsilon = \lim_{\epsilon\to 0}\sqrt{z^2+\epsilon^2}$ *is encountered quite often in the solution of Laplace's equation. Of course* $|z$ *is a non-analytic function (check the Cauchy–Riemann equations or go from the definition). However, one can use it by thinking of the limit. This function is analytic in the strip* $-\epsilon < Im\ z < \epsilon$*, and then arguments of analyticity are OK in that strip.*

By analogy with `p22.m`, a branch of the function $\gamma(z)$ can be chosen which is real and positive on the positive real axis and has a branch cut between $-$i and i. This uses the code `z.*sqrt(1+1./z.^2)`. However this is not the appropriate branch to deal with an inverse transform, since it intersects the inversion contour. Possible branch cut structures are cuts along the imaginary axis with $|\text{Im } z| > 1$, using `sqrt(z.^2+1)`, and cuts parallel to the real axis in the left half-plane, using `sqrt(z-i).*sqrt(z+i)`. Finally one can construct very strange cuts with code like `sqrt(z+i).*sqrt(z).*sqrt(-i./z+1)`: imaginary axis between -1 and 1, negative real axis, horizontal line going left from -1.. These possibilities are shown in `p23.m` and in Figure 2.3.

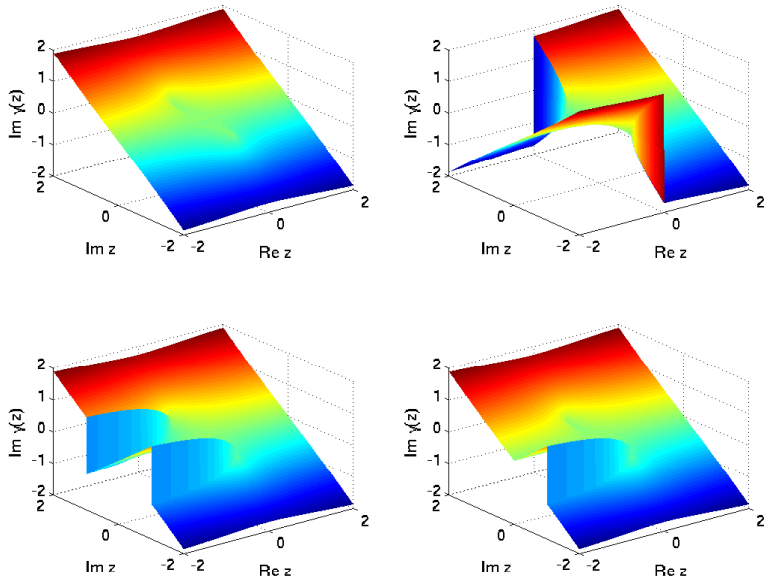

Figure 2.3: Contour plot of the imaginary part of $\gamma(z)$ using two different branch cuts.

As seen in `p22.m`, branch cuts can cancel in a nice way. Typical examples occur in linear elasticity problems:

Example 2.2 *The function* $f(z) = \sqrt{z^2+1}\sqrt{z^2+k_0^2}$, *where* $k_0 > 1$, *has a nice branch cut structure.*

IN CLASS

Exercise 2.2 *Work out the branch cut structure of*

$$f(z) = \sqrt{z + \sqrt{z^2+1}}. \tag{2.2}$$

Be as explicit as possible. Think about how the Riemann sheet structure stitches together.

2.4 Integrals

2.4.1 Keyholes and hamburgers

Branch cuts are critical in calculating certain integrals using residues.[2]

The following integral can be done without branch cuts,but we'll use a cut here:

$$I = \int_0^\infty \frac{x^2}{1+x^5}\,\mathrm{d}x. \tag{2.3}$$

[2]Richard Feynmann's proud claim that he could do any integral without using residues always struck me as depressing: contour integrals are so enjoyable.

The integrand has no good symmetry properties on the real axis, although it does well along the ray making an angle $2\pi/5$ from the axis. So one approach is to go out along the real axis and come back along the arc making angle $2\pi/5$ with the real axis. The result is $\pi/5 \sin(2\pi/5)$.

Consider

$$I_C = \int_C \frac{z^2 \log z}{1+z^5}\,\mathrm{d}z. \tag{2.4}$$

We have introduced a multifunction. Take the branch cut along the positive real axis, with $\log z$ real along the top of the real axis (and hence with imaginary part $2\pi\mathrm{i}$ along the bottom of the positive real axis. The "keyhole" contour C goes along the top of the positive real axis to R, swings around a large circle, and then back along the bottom of the positive real axis. Since $\log z$ is integrable near the origin, we need not bother with little circles about the origin in our policy of efficiency. The integrand is $O(R^{-5})$ for large R, which immediately shows that the contribution from the large circle vanishes. There are simple poles at the five fifth roots of -1: $\mathrm{e}^{\pi\mathrm{i}/5}$, $\mathrm{e}^{3\pi\mathrm{i}/5}$, $\mathrm{e}^{5\pi\mathrm{i}/5}$ $\mathrm{e}^{7\pi\mathrm{i}/5}$ and $\mathrm{e}^{9\pi\mathrm{i}/5}$ (the choice of arguments is set by the choice of branch). Hence

$$\begin{aligned} I_C &= 2\pi\mathrm{i}\left(\mathrm{e}^{2\pi\mathrm{i}/5}\frac{\pi\mathrm{i}/5}{5\mathrm{e}^{2\pi\mathrm{i}/5}} + \mathrm{e}^{6\pi\mathrm{i}/5}\frac{3\pi\mathrm{i}/5}{5\mathrm{e}^{6\pi\mathrm{i}/5}} + \mathrm{e}^{10\pi\mathrm{i}/5}\frac{5\pi\mathrm{i}/5}{5\mathrm{e}^{10\pi\mathrm{i}/5}} + \mathrm{e}^{14\pi\mathrm{i}/5}\frac{7\pi\mathrm{i}/5}{5\mathrm{e}^{14\pi\mathrm{i}/5}}\right. \\ &\quad \left. + \mathrm{e}^{18\pi\mathrm{i}/5}\frac{9\pi\mathrm{i}/5}{5\mathrm{e}^{18\pi\mathrm{i}/5}}\right) \\ &= 2\pi\mathrm{i}\left(\frac{\pi\mathrm{i}}{25}\right)[20c^2 + 10c - 5 + 8\mathrm{i}s(1-c)], \end{aligned} \tag{2.5}$$

where we have written s and c for the sine and cosine of $2\pi/5$ respectively. Taking appropriate limits and nothing that $\log z = \log|z| + 2\pi\mathrm{i}$ along the underside of the real axis, we find

$$I_C = \int_0^R \frac{x^2 \log x}{1+x^5}\,\mathrm{d}x + \int_R^0 \frac{x^2(\log x + 2\pi\mathrm{i})}{1+x^5}\,\mathrm{d}x + O(R^{-4}) \tag{2.6}$$

for large R. The logarithmic terms cancel and leave us with $-2\mathrm{i}\pi I$ as $R \to \infty$. Equating these results gives

$$I = -\frac{\pi\mathrm{i}}{25}[20c^2 + 10c - 5 + 8\mathrm{i}s(1-c)]. \tag{2.7}$$

Now we need to simplify this result. The imaginary part vanishes, so $c^2 = -\frac{1}{2}c + \frac{1}{4}$. Hence

$$I = \frac{8\pi}{25}s(1-c) = \frac{8\pi}{25s}(1-c^2)(1-c) = \frac{8\pi}{25s}(\tfrac{3}{4} + \tfrac{1}{2}c)(1-c) = \frac{8\pi}{25s}(\tfrac{3}{4} - \tfrac{1}{4}c - \tfrac{1}{2}c^2) = \frac{\pi}{5s} \tag{2.8}$$

as before.

Now consider

$$I = \int_1^\infty \frac{\mathrm{d}x}{(x-\frac{1}{2})\sqrt{x(x-1)}}. \tag{2.9}$$

The singularity at $x = 1$ is integrable. Consider

$$I_C = \int_C \frac{\mathrm{d}z}{(z-\frac{1}{2})z\sqrt{z(z-1)}}\,\mathrm{d}z \tag{2.10}$$

with branch cuts from 1 to infinity along the positive real axis and from 0 to minus infinity along the negative real axis. We take the branch that is real and positive on top of the positive real axis. The contour goes out from 1 along the top of the real axis to R, loops around a semicircle to $-R$, back along the top of the negative real axis to 0, back out along the bottom of the negative real axis to $-R$, semicircle back to R and returns to 1 along the bottom of the positive real axis. This is the "hamburger" contour. We don't bother with little circles or indentations around 1 and 0: the singularity is integrable so there will be on contribution. Similarly, for large R, the contributions from the large semicircle is $O(R^{-1})$ and goes away. Paying close attention to the cuts, we find

$$I_c = 2\int_1^\infty \frac{\mathrm{d}x}{(x-\frac{1}{2})\sqrt{x(x-1)}} + 2\int_0^\infty \frac{\mathrm{d}y}{(y+\frac{1}{2})\sqrt{y(y+1)}} = 4I \tag{2.11}$$

after a simple change of variable. Now there is a simple pole at $z = \frac{1}{2}$, with residue 2. Hence $I = \pi$. Note that if we move that pole, the result cannot be obtained this way because there is no longer the required symmetry to express I_c purely in terms of I.

2.4.2 Branch cuts and integrating around infinity

Consider

$$I = \int_{-1}^{1} \frac{1}{(1+x^2)\sqrt{1-x^2}}\,\mathrm{d}x. \tag{2.12}$$

Consider I_c with the obvious integrand, branch cut between -1 and 1 with positive real values along the top of the branch cut. This is non-standard. Take the contour to go around the branch cut in the positive sense. Again we don't bother indenting around the ends of the branch cut since the singularities are integrable. Along the top of the cut, we obtain $\mathrm{i}I$, and ditto along the bottom. Now think about deforming the contour. As it grows larger, we have an estimate $O(R^{-2})$. Hence the integral around infinity vanishes. However, there are two poles in the complex plane at $z = \pm\mathrm{i}$ that are moved through to deform the contour to infinity. Hence

$$I = \tfrac{1}{2}I_c = \pi\mathrm{i}\left(\frac{1}{2\mathrm{i}\sqrt{2}} + \frac{1}{(-2\mathrm{i})\mathrm{i}(-\sqrt{2})}\right) = \frac{\pi}{\sqrt{2}}, \tag{2.13}$$

where one has to be careful with the arguments of the square roots. This technique of deforming the contour to go around infinity also works well for more complicated integrals.

Chapter 3

Finding zeros (04/09/2008)

Finding the zeros of an analytic function is required for many physical problems. This is particularly the case for problems solved using integral transforms, where the behavior of complex frequency and wavenumber leads to complex dispersion relations linking the two.

In many wave propagation problems, for example surface gravity waves, we have some physical insight into the problem, and can usually identify propagating waves and exponentially damped waves (a boundary is required for these). The dispersion relation takes the form of a transcendental relation and zeros are pure real or pure imaginary, so that a few judicious substitutions give a real equation to solve. Intrinsically complex cases are more difficult.

Exercise 3.1 *Find the real and imaginary roots of the dispersion relation*

$$\nu - k \tanh k. \tag{3.1}$$

This is a problem that deals with real numbers, not complex numbers.

3.1 Multi-dimensional zero finding

In general, this is a difficult and unrewarding task. In one dimension, we have a good intuitive picture of what is going on: to solve $f(x) = 0$, we draw a graph of $y = f(x)$ and see where it intersects the x-axis. Even better, if our function $f(x)$ is continuous (and there are cases where it won't be), the intermediate value theorem tells us that between points where f has opposite sign, there must be a zero of f.

There are many algorithms to find zeros, as in secant, Newton–Raphson, more complicated versions, and so on. There is no shame in using a packaged routine. The programmer probably had more time and experience than you. Matlab has `fzero` which is good. Note that it can be used in two modes: by specifying an initial guess, and by specifying an interval. In the second case, `fzero` can tell you if there is no root (assuming your function is continuous). If there are several roots in your interval, you might not find the one you expect. No guarantee that there is a root in the first case, and again the zero you find might not be the one you expect. In FORTRAN, SLATEC's `dfzero` does a good job.

For polynomials, you should be using an algorithm designed for polynomials. Matlab has `roots`.

Even if you are using a canned package, some care is required. In all cases, you should look at what is going on graphically. Unless everything is very spiky, the main point of running the algorithm is often to get good accuracy. Nevertheless, you should check your results. Acton (1970) has some wise words on this and on many other thins. To extend on the words of Boyd (2001; Definition 16),

Definition 3.1 *An idiot is someone who obtains a numerical result and does not check it. Graphically, physically, by doubling the resolution, etc...*

In higher dimensions, things are nastier. Instead of finding the intersection of a curve and a straight line, we are finding the intersection of hypersurfaces and we may not be looking at points any longer; for example the zeros of $f(x,y) = x^2 - y^2$ are no longer points (zero dimensions) but curves (one dimension). If this is really what you are doing, grit your teeth and use something like `fsolve` in Matlab or a FORTRAN routine.

The complex function $f(z)$ have one advantage when it comes to finding zeros: two unknowns (real and imaginary parts of z) and two equations to satisfy (real and imaginary parts of f). We have a good graphic pictures of what is going on, as in Figure 3.1 for an essentially arbitrary function $f(z)$.

```
% p31: contour plot for zeros; 04/07/2008 SGLS
x = -10:00.1:10; y = x;
[x,y] = meshgrid(x,y); z = x + i*y;
f = z.^2+1 - sin(z).*z;
contour(x,y,real(f),[0 0],'r');
hold on; contour(x,y,imag(f),[0 0],'b'); hold off
xlabel('Re z'); ylabel('Im z');
```

Another advantage is that the algebraic (and analytic structure) of complex numbers means that we can use algorithms that were developed for the one-dimensional cases, in particular iterative methods.

Exercise 3.2 *Find the zeros of $f(z) = z^2 + 1 - z\sin z$. Discuss what is meant by "all zeros". How many do you think would be useful? Can you find approximate values for the larger ones?*

3.2 Iterative methods

3.2.1 Fixed-point iteration

The idea here is to turn our zero-finding problem into an equation of the form $f(z) = z$ and then solve the iterative process

$$z_{n+1} = f(z_n), \tag{3.2}$$

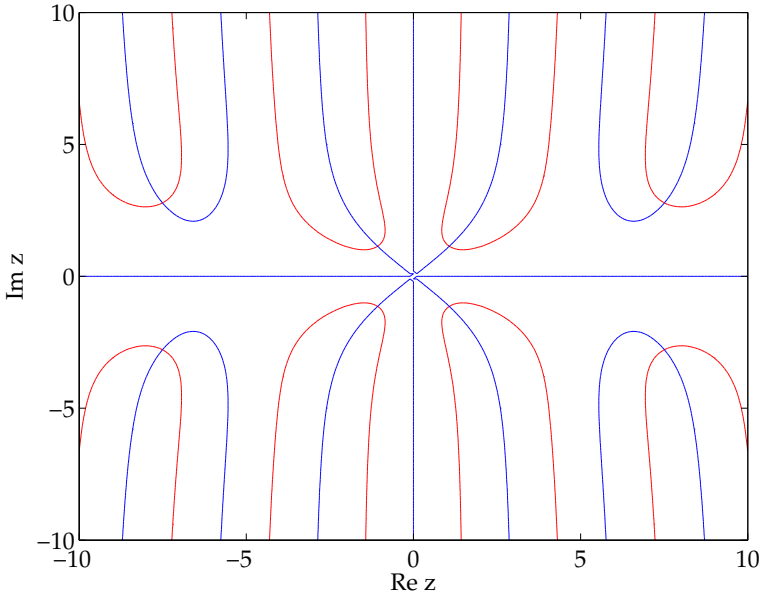

Figure 3.1: Contour of real (red) and imaginary (blue) part of the function $f(z) = z^2 + 1 - z\sin z$.

with some initial guess z_0.The condition for convergence is simple to obtain. Write the root of $f(z) = z$ as z^*, so that $z^* = f(z^*)$. Define the error as $e_n = z_n - z^*$ and subtract:

$$e_{n+1} = f(z^* + e_n) - f(z^*) = e_n f'(z^*) + O(|e_n|^2). \tag{3.3}$$

If we are close enough to the z^*, then this is almost a geometric process and the convergence will depend on $|f'(z^*)|$.

Quasitheorem 5 *The fixed-point iteration $z_{n+1} = f(z_n)$ converges near the root z^* if $|f'(z^*)| < 1$.*

However, if initial guess is poor, the result may be poor. Worse than that, it may be unpredictable. This is the cue for lots of lovely pictures of Julia sets and so on, as in Figure 3.2.

```
% p32: iterations and fractals; 04/07/2008 SGLS
x = -1.2:0.005:1.2; y = x;
[x,y] = meshgrid(x,y); z = x + i*y;
c = 0.285 + 0.01*i;
for j = 1:length(x);
  for k = 1:length(y);
    zn = z(j,k);  n = 0;
    while (abs(zn)<10 & n<100)
      zn = zn.^2 + c; n = n + 1;
    end
```

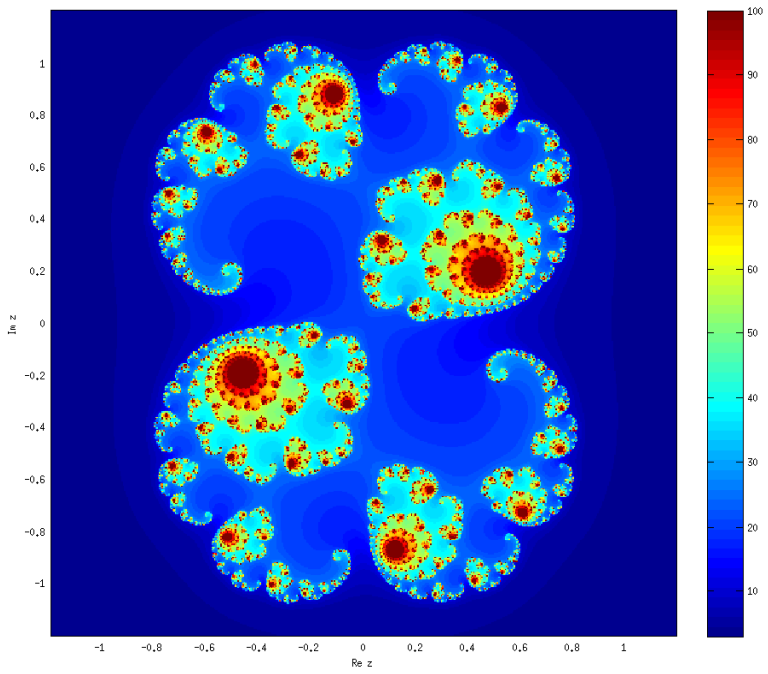

Figure 3.2: Level sets of the basin attraction of infinity for the map $z_{n+1} = z_n^2 + c$, where $c = 0.285 + 0.01\mathrm{i}$. The color gives the number of the iteration at which $|z_n| = 10$ with a maximum of 100.

```
    f(j,k) = n;
  end
end
pcolor(x,y,f); shading flat; axis square; colorbar
xlabel('Re z'); ylabel('Im z');
```

3.2.2 Newton–Raphson

Fixed point iteration is limited by the condition on $|f'(z^*)|$. Worse, it may not be obvious how to produce an iteration from the original equation except in an artificial way. Newton–Raphson[1] iteration converges quadratically and looks particularly simple for complex functions because the derivative is a complex quantity rather than a mapping as in the multidimensional case. So no Jacobians.

The iteration is

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}. \tag{3.4}$$

The geometric interpretation in one dimension is that one draws from $f(x_n)$ the tangent to the curve $y = f(x)$ and finds the intersection of the tangent with the x-axis as the next

[1]Question: who was Raphson? Answer: Newton's programmer. Originally heard from Arieh Iserles. In reality Raphson was an obscure English mathematician – approximate dates 1648–1715.

iterate. In the complex plane, this isn't so clear. Once again the choice of starting point can be crucial, and we have Newton fractals, etc... Watch our for multiple roots.

3.3 Principle of the argument

The methods above are effective and useful. We can use the analytic structure of complex differentiability to obtain methods that are guaranteed to find zeros in a domain of the complex plane. The function has to be analytic, and multiple roots can be confusing, but the method is extremely useful.

We know that

$$\int_C \frac{f'(z)}{f(z)}\,\mathrm{d}z = Z - P, \tag{3.5}$$

where Z and P are the number of zeros and poles of $f(z)$ in the domain, counted with multiplicity. So if we have a function of $f(z)$ with no poles inside C, we can see if it has a zero. Checking to see if a function has poles is usually very easy, and we can almost always get rid of the poles.

Now we can show similarly that for a function that has no poles inside C,

$$\int_C z\frac{f'(z)}{f(z)}\,\mathrm{d}z = \sum_n z_n, \tag{3.6}$$

where the z_n are the zeros we are looking for. If $Z = 1$, we have found the zero. If $Z = 2$, we can use

$$\int_C z^2\frac{f'(z)}{f(z)}\,\mathrm{d}z = \sum_n z_n^2 \tag{3.7}$$

to solve for z_1 and z_2. Obviously this can be extended to more and more zeros, but it is usually easier to change C.

So by computing an integral, we guarantee to find the zero (or zeros) inside C. This can be very advantageous if one is unsure where the zeros are or if one is having trouble with starting values for an iterative method. To do this, one needs to compute complex integrals numerically. How does one do this?

3.4 An aside on numerical integration and contour integrals

Whole books can be written on the topic of numerical integration. Many have, some more interesting than Piessens *et al.* (1983) which is a slight elaboration of the manual for QUADPACK. We will limit ourselves to one quick discussion,

Numerical integration is based on replacing integrals by sums, as in the original definition of the Riemann integral.

Tip 3 *I have never met a physical problem that could not be solved using Riemann integrals, except for probability integrals where Stieltjes is useful. I have never used a Lebesgue integral.*

The workhorse of most automatic integration is the compound trapezium rule:

$$\int_0^1 f(z)\,\mathrm{d}z = h\left(\tfrac{1}{2}f(0) + f(h) + f(2h) + \cdots + f((n-1)h) + \tfrac{1}{2}f(h)\right), \tag{3.8}$$

where $nh = 1$. More general intervals of integration are obvious. Clever ways of applying this recursively lead to efficient algorithms with potentially high accuracy. Note that if f isn't smooth enough, this accuracy will not be reached. This is the classic trade-off between order and smoothness. Matlab actually uses adaptive Simpson rule (`quad`) and adaptive Lobatto quadrature (`quadl`), and clearly states in the help page for the latter: "Function QUAD may be more efficient with low accuracies or nonsmooth integrands."

Formally the error for (3.8) is $O(h^2)$. Integrating the simple function $f(x) = x/(4x^2 + 1)$ between 0 and 1 shows this,as indicated in Figure 3.3(a). In fact, the compound trapezium rule does well compared to the Matlab functions, but efficiency has not been factored into this calculation. The same integrand is integrated around the unit circle in (b), and the difference is startling. The compound trapezoid rule is now exponentially accurate. This can be proved for periodic functions, and all the closed contour integrals we examine will be periodic. Note that for periodic integrals the rule can be written as

$$\int_0^1 f(z)\,\mathrm{d}z = h\sum_{j=1}^{n} f(jh). \tag{3.9}$$

```
% p33: numerical integration in the complex plane; 04/07/2008 SGLS
f = inline('x./(4*x.^2+1)'); Iex = log(5)/8;
n = 10.^(1:6); h = 1./n;
for j = 1:length(n)
  x = h(j)*(0:n(j));
  I1(j) = h(j)*( 0.5*f(x(1)) + sum(f(x(2:end-1))) + 0.5*f(x(end)) );
  I2(j) = quad(f,0,1,h(j));
  I3(j) = quadl(f,0,1,h(j));
end
subplot(2,2,1)
loglog(h,abs(I1-Iex),'.',h,h.^2/12,h,abs(I2-Iex),'o',h,abs(I3-Iex),'s');

g = inline('exp(2*pi*i*t)./(4*exp(4*pi*i*t)+1)*2*pi*i.*exp(2*pi*i*t)');
Ic = 0.5*pi*i;
for j = 1:length(n)
  t = 2*pi*h(j)*(1:n(j));
  z = exp(i*t); dz = 2*pi*i*z;
  I4(j) = h(j)*sum(f(z).*dz);
  I5(j) = quad(g,0,1,h(j));
  I6(j) = quad(g,0,1,h(j));
end
subplot(2,2,2)
loglog(h,abs(I4-Ic),'.',h,h.^2/12,h,abs(I5-Ic),'o',h,abs(I6-Ic),'s')
```

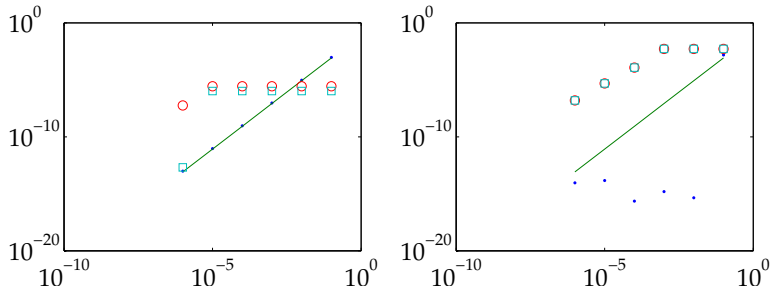

Figure 3.3: (a) Absolute errors in the integral of $f(x)$ between 0 and 1 as a function of h for the trapezium (dots) and the Matlab routines `quad` (circles) and `quadl` (squares). The solid line shows $h^2/12$. (b) Same for the integral of $f(z)$ around the unit circle.

To be fair, at least Matlab can integrate complex integrands directly. FORTRAN programs need separate real and imaginary parts for most canned integration packages. In addition, infinite integrals do not have this wonderful property, so Matlab routines are usually faster then.

Some care is required with singular integrands. If the singularity is integrable (fractional powers between -1 and 0 or logarithms), a change of variable is required for quadrature rules that use the endpoints to avoid returning `NaN`'s. From a mathematical perspective it's also a good thing to do, since the quadrature routines usually require a certain degree of smoothness to achieve optimal convergence. There are FORTRAN subroutines that take care of singular integrals automatically (`dqagi`, `dqawf`, etc...). They are mostly changes of variable as discussed, with two important exceptions. First, infinite integration with an oscillatory integral – this is specialized, so use the routines. Finally, numerical evaluation of Cauchy principal value integrals is also specialized so use the routines.

Tip 4 `NaN` *and* `Inf` *are our friends. They show us we've screwed up. We should be grateful to the IEEE standard. They're also useful in plotting routines to remove part of a field that we're contouring.*

3.5 An example using Matlab

In a paper dealing with elastic plates, Abrahams *et al.* (2008) require the complex zeros of the numerator and denominator, denoted μ_n and λ_n respectively, of the following function:

$$D = \frac{k^4[4 + (3+\nu)(1-\nu)\sinh^2 k + (1-\nu)^2 k^2]}{\sinh^2 k - k^2}. \tag{3.10}$$

These zeros are related to the Papkovich–Fadle eigenfunctions for a semi-infinite elastic plate.

By symmetry, we see that if k is a zero (of numerator or denominator), so is $-k$. Similarly, if k is a zero, so is $\bar{k}$. Hence we can restrict ourselves to the first quadrant, and label the desired zeros in an obvious way, moving away from the origin ($\mu_0 = \lambda_0 = 0$).

It is a good policy to study the behavior of D in various limits For small $|k|$, $D \sim 12$, a finite limit. For large $|k|$ with Re $k > 0$, the hyperbolic sine terms dominate and $D \sim (3+\nu)(1-\nu)k^4$. Along the imaginary axis, the hyperbolic sine term is bounded and for large $|k|$, $D \sim -(1-\nu)^2 k^4$.

We start with the denominator. First, can we find a first approximation to the zeros? We can see there are no pure real roots and no pure imaginary roots except at the origin (k does not intersect $\sin k$ or $\sinh k$ except at the origin). When $|k|$ is big and Re $k > 0$, $\sinh k \sim \frac{1}{2}e^k$, which grows must faster than k. Hence for the two to be equal, the imaginary part of k must be such as to make e^k smaller, i.e. near $\pi/2 + j\pi$ where j is an integer. Substituting into $\sinh k = \pm k$ gives

$$\tfrac{1}{2}e^{x+i(\pi/2+j\pi)+\cdots} = \tfrac{1}{2}(-1)^j i e^x + \cdots = \pm[x + i(\pi/2 + 2j\pi) + \cdots]. \tag{3.11}$$

Exploiting the ambiguity in sign and solving for the real part gives $k \sim \log(\pi + 2j\pi) + i(\pi/2 + j\pi)$ as a first guess. This is in fact a very good guess, and starting from it, Matlab has no trouble finding the roots, as shown in Figure 3.4. Note the use of the `fsolve` function and the change from complex to real and back in the implementation.

```
function p34

n = 20; k = pd(n); nn = 1:n; kg = log(pi+2*nn*pi) + i*(pi/2+nn*pi);
plot([0 ; k],'r.'); hold on; plot([0 kg],'o'); hold off
xlabel('Re k'); ylabel('Im k')

function k = pd(M)

options = optimset('Display','off','TolFun',eps,'TolX',eps);
k = zeros(M,1);
for j = 1:M
  kg = fsolve(@(k) crel(k),[log(pi+2*j*pi) pi/2+j*pi],options);
  k(j) = kg(1) + i*kg(2);
end

function f = crel(k)

k = k(1) + i*k(2);
b = sinh(k)^2-k^2;
f(1) = real(b);
f(2) = imag(b);
```

The approach to finding the μ_n is similar, including the initial guess. However, convergence is more delicate, particularly for $\nu > 1/4$. In that case, the code starts with $\nu = 1/4$ and increases ν gradually, ensuring convergence. This is an example of a continuation procedure: start from a case that works and change a parameter (gently). The iteration can converge to negative real values, and these are removed.

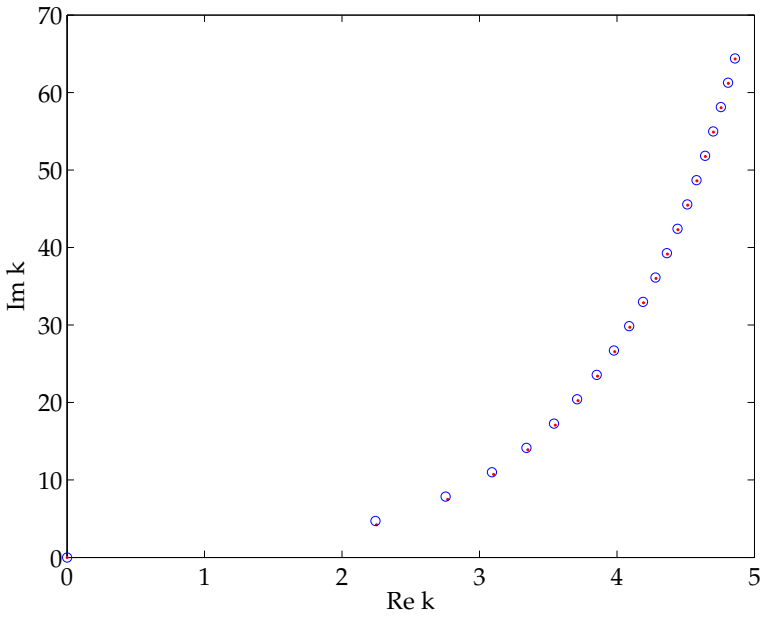

Figure 3.4: Dots: zeros λ_n of the denominator of (3.10). unit circle; circles: initial guesses.

Note also the special case considered at the beginning. There is one zero, k_i, that does not have the same asymptotic behavior as the others. It is purely imaginary. It can be seen on the imaginary axis in Figure 3.5. The different values of ν do not seem to make much difference to the final plots, but, as mentioned previously, can cause problems numerically.

Exercise 3.3 *Justify the initial guess for k_i in* `p35.m`.

```
function p35

subplot(2,2,1)
k = pn(20,0); plot([0 ; k],'.')
xlabel('Re k'); ylabel('Im k')
subplot(2,2,2)
k = pn(20,0.25); plot([0 ; k],'.')
xlabel('Re k'); ylabel('Im k')
subplot(2,2,3)
k = pn(20,0.3); plot([0 ; k],'.')
xlabel('Re k'); ylabel('Im k')
subplot(2,2,4)
k = pn(20,0.5); plot([0 ; k],'.')
xlabel('Re k'); ylabel('Im k')


function k = pn(M,nu)
```

```
options = optimset('Display','off','TolFun',eps,'TolX',eps);
k = zeros(M+1,1);
kim = fsolve(@(k) crel(k,nu),[0 2/(1-nu)],options);
for j = 1:M
  kg = [log(2*j*pi*sqrt((1-nu)/(3+nu))) j*pi];
  if (nu>0.25 & j==1)
    for nn = 0.25:0.05:nu
      kg = fsolve(@(k) crel(k,nn),kg,options);
    end
  end
  kg = fsolve(@(k) crel(k,nu),kg,options);
  k(j+1) = kg(1) + i*kg(2);
end
k = [k(imag(k)<kim(2)) ; i*kim(2) ; k(imag(k)>kim(2))];
k = k(1:M);

function c = crel(k,nu)

k = k(1) + i*k(2);
b = 4 + (3+nu)*(1-nu)*sinh(k)^2 + (1-nu)^2*k^2;
c(1) = real(b);
c(2) = imag(b);
```

Warning 7 *The physically acceptable values for ν lie in the range $(0, 1/2)$ (materials with $-1 < \nu < 0$ are thermodynamically possible but do not occur naturally). The code* `p35.m` *has only been tested in that range of ν and attempting to use it for other values of ν will probably result in disaster.*

3.6 An example using the principle of the argument

In a problem treating with groundwater flow using Laplace transforms, it is necessary to find the zeros of

$$D = z \tan z + K, \tag{3.12}$$

where K is complex. One approach used the principle of the argument.

The code is long and is relegated to the appendix. The critical issue is the presence of an unexpected zero for complex K in certain parts of the K-plane. The location of this zero causes problems with iterative methods or canned packages, because it is hard to know where it is except in limiting cases, and it leads to poor convergence for the other roots unless the initial guesses are very good.

Figure 3.6 shows roots for (a) $K < 0$, (b) K pure imaginary, (c) $K > 0$ and (d) K with large positive real part but not pure real. The new root appears in case (d) and $z \sim \mathrm{i}K$.

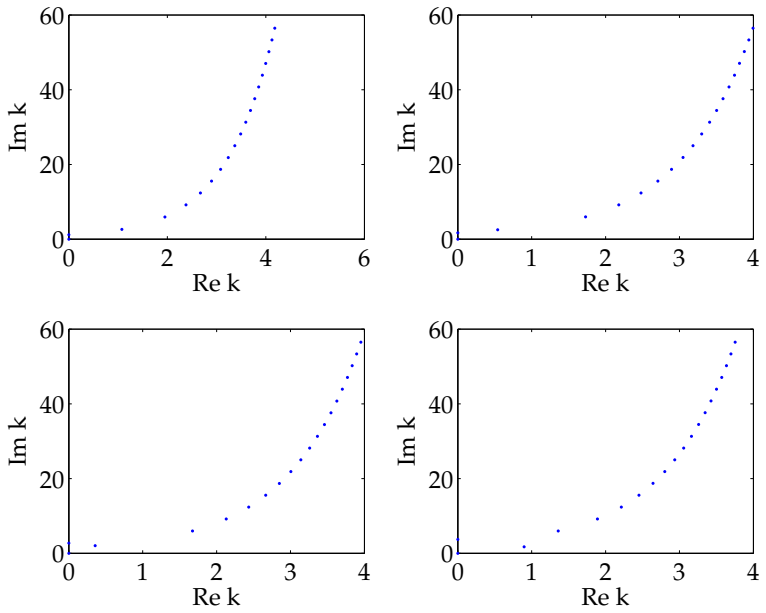

Figure 3.5: Zeros λ_n of the numerator of (3.10). (a) $\nu = 0$, (b) $\nu = 0.3$, (c) $\nu = 0.4$, (d) $\nu = 1/2$.

The numerical code uses a series of rectangles in the complex plane, since something special is obviously happening at Re $z = \pi/2 + j\pi$. The top and bottom of the boxes are at $\pm \max(2\pi, 2\text{abs}(\text{Re } K))$ respectively: the second value is dictated by a scaling argument as Im z increases, while the first is there to ensure that roots still fit in the box for small $|\text{Re } K|$. The roots are polished after being found crudely using the contour integral procedure. This polishing is carried out using a complex version of (3.12) and there are (inconsequential) imaginary parts in case (a), where the zeros are in fact real. The numerically found zeros in (c) are real as they should be.

Exercise 3.4 *Derive the relation $z \sim \mathrm{i}K$ for this new root. When does it exist?*

Exercise 3.5 *Explain why there are 9 points plotted in case (c) of Figure 3.6.*

3.7 Other approaches

There are other ways of finding zeros. Sometimes the zeros are related to an underlying Sturm–Liouville problem. Recalling the Rayleigh–Ritz method, which gives the eigenvalue as a ratio of integrals, one can use a trial function to obtain a good first approximation and then iterate. See Porter & Chamberlain (1999).

References

Books:

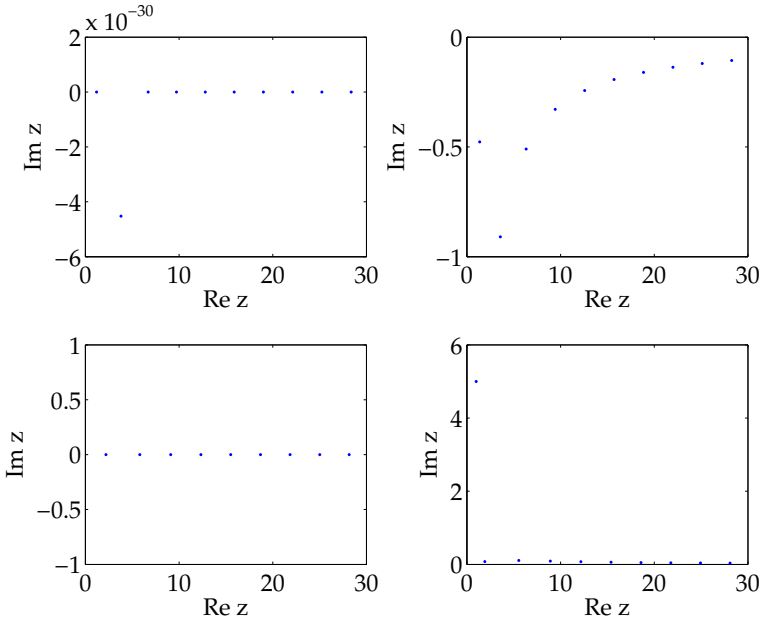

Figure 3.6: The first 10 zeros ranked by (positive) real part of (3.12) in the first quadrant for (a) $K = -3$, (b) $K = 3\mathrm{i}$, (c) $K = 3$ (only 9 zeros plotted) and (d) $K = 5 - \mathrm{i}$.

- Abrahams, I. D., Davis, A. M. J. & Llewellyn Smith, S. G. Matrix Wiener–Hopf approximation for a partially clamped plate. *Q. J. Mech. Appl. Math.*, doi: 10.1093/qjmam/hbn004.
- Acton, F. S. *Numerical methods that work.* Harper & Row, New York, 1970.
- Boyd, J. P. *Chebyshev and Fourier spectral methods.* Dover, New York, 2nd ed., 2001.
- Chamberlain, P. G. & Porter, D. On the solution of the dispersion relation for water waves. *Appl. Ocean Res.*, **21**, 161–166, 1999.
- Piessens, R. *et al. Quadpack: a subroutine package for automatic integration.* Springer-Verlag, Berlin. 1983.

Web sites:

- http://en.wikibooks.org/wiki/Fractals/Iterations_in_the_complex_plane/Julia_set
- http://en.wikipedia.org/wiki/Joseph_Raphson
- http://en.wikipedia.org/wiki/Newton_fractal

Chapter 4

ODEs in the complex plane (04/17/08)

4.1 Motivation

We start with a problem from fluid mechanics, stability to be precise (see Llewellyn Smith 1996). The governing equations for inviscid two-dimensional fluid flow can be written in terms of ψ, the streamfunction such that $(u, v) = (\psi_y, -\psi_x)$ – this is the geophysical convention which is the opposite from the classical convection – or $(u_r, u_\theta) = -r^{-1}(\psi_\theta, \psi_r)$ in polar coordinates. The streamfunction satisfies the vorticity equation

$$\frac{\mathrm{D}q}{\mathrm{D}t} = \frac{\partial q}{\partial t} + J(\psi, q) = 0, \qquad q = \nabla^2 \psi. \tag{4.1}$$

Here q is the scalar vorticity and the Jacobian takes the form $J(a, b) = a_x b_y - a_y b_x$ in Cartesian coordinates and $J(a, b) = r^{-1}(a_r b_\theta - a_\theta b_r)$ in polar coordinates.

We wish to investigate the stability of unidirectional shear flows, $\psi = \Psi(y)$, and swirling flows, $\psi = \Psi(r)$. These flows are exact, steady, solutions of (4.1). We hence write $\psi = \Psi + \psi'$ (and hence $q = Q + q'$) and retain only linear terms in the resulting equation. The result is

$$\frac{\partial q'}{\partial t} + J(\Psi, q') + J(\psi', Q) = 0, \qquad q' = \nabla^2 \psi'. \tag{4.2}$$

In the Cartesian case, the coefficients are independent of x, so we may consider modes of the form $\mathrm{e}^{\mathrm{i}kx}$. The linearized equation reduces to

$$\left(\frac{\partial}{\partial t} + \mathrm{i}kU\right)\left[\frac{\partial^2 \psi'}{\partial y^2} - k^2 \psi'\right] + \mathrm{i}kQ'\psi' = 0, \tag{4.3}$$

where $U = \Psi'(y)$ is the background velocity in the x-direction and Q' is the derivative of Q with respect to y. In the cylindrical case, the appropriate mode structure is $\mathrm{e}^{\mathrm{i}n\theta}$ and the corresponding result is

$$\left(\frac{\partial}{\partial t} + \mathrm{i}n\Omega\right)\left[\frac{1}{r}\left(r\frac{\partial \psi'}{\partial r}\right) - \frac{n^2}{r^2}\psi'\right] - \frac{\mathrm{i}nQ'}{r} = 0 \tag{4.4}$$

with $\Omega = r^{-1}\Psi'(r)$ the angular velocity. Rayleigh's equation, properly speaking, probably refers to the Fourier transform in time of (4.3). Boundary conditions are $\psi' = 0$ on the boundaries $y = \pm a$ in the Cartesian case; $\psi' = 0$ at $r = a$ and finiteness at $r = 0$ in the polar case. (These are replaced by appropriate decay conditions if the domain is unbounded.) On physical grounds $Q'(0) = 0$ and $\Omega = \Omega_0 + O(r^2)$.

Eigenmodes of the system come from substituting the time dependence $e^{i\omega t}$. If there are solutions for ω with negative imaginary part for any k, the basic state is linearly unstable since an arbitrary perturbation will contain some component at the unstable k, and perturbations grow. If ω is real for all k, the basic state is neutrally stable: the perturbation neither grows nor decays in time.

This is an eigenvalue problem. Second-order eigenvalue problems are very common in physical situations, and so we concentrate on second-order equations here. Matlab has a suite of ODE solvers (`ode45.m`, etc...) that solve initial-value problems for systems of first-order equations. This is not a limitation since such systems are equivalent to n-order systems. We will limit ourselves to linear ODEs in this chapter. These are hard enough for the second-order case. For the first-order case, we can find an explicit solution using integrating factors.

4.2 Radial Rayleigh equation

We consider the radial Rayleigh equation written in self-adjoint form:

$$(-r\phi')' + \left[\frac{n^2}{r^2} + \frac{nQ'}{n\Omega - \omega}\right]\phi = 0. \tag{4.5}$$

Clearly something strange can happen if the denominator of the second fraction vanishes, which corresponds to a neutral mode since ω has to be real. The eigenvalue problem is a boundary value problem, since one has a natural[1] boundary condition at the origin and something at large r.

We shall return to (4.5) and its Cartesian counterpart later. Let us consider a more general problem for now, one in which there is some extra physical effect of small amplitude that enters the governing vorticity equation (e.g. the beta-effect) and in which we are solving an initial-value problem based on (4.4). If some different physical balance holds far from the origin, we need to understand how well-behaved solutions of the Laplace-transformed version of (4.5) behave for large r. This is a connection problem (see Olver 1974 for a general presentation).

The problem is hence to solve

$$(-r\phi')' + \left[\frac{n^2}{r^2} + \frac{inQ'}{p + in\Omega}\right]\phi = 0 \tag{4.6}$$

with ϕ well-behaved at the origin and obtain the behavior of ϕ for large r.

[1] Jargon for a condition that is imposed by the equation, so to speak, rather than coming from outside. Common in discussions of the finite element method.

4.2.1 Frobenius expansions

First we have to be able to solve near the origin. We recall the classification of the point a of the linear second-order ODE

$$y'' + p(x)y' + q(x)y = 0. \tag{4.7}$$

Ordinary point: $p(a)$ and $q(a)$ well behaved. Otherwise: singular point. Regular singular point: $(x-a)p(x)$ and $(x-a)^2q(x)$ bounded as $x \to a$. Irregular singular point: otherwise.

A series solution of the form $\sum_{n=0}^{\infty} a_n(x-a)^{n+\sigma}$ with non-vanishing a_0 always exists for a regular singular point. Plug into (4.6) near $r = x = 0$ to obtain

$$(-\sigma^2 + n^2)a_0 r^{\sigma-2} + O(r^{\sigma-1}) = 0. \tag{4.8}$$

Hence near the origin, $\phi \sim r^{\pm n}$. Obviously only the plus sign is physically acceptable. The fact that the indices differ by $2n$, an integer, means that there may be complications to do with logarithms. These complications do not affect us here.

What initial condition do we use for our vector $(\phi(0), \phi'(0))$? For $n > 1$, this vector vanishes and our ODE solver cannot start. This is a typical problem for regular singular points. There are (at least) three possible solutions in this case:

1. Start integrating at $r = \delta$ small with $\phi(\delta) = \delta^n$ and $\phi'(\delta) = n\delta^{n-1}$. This is just forcing the solution to be the Frobenius expansion for small r. I dislike introducing an error into the problem.

2. Work with $g = r^{-n}\phi$. Then $g(0) = 1$ and $g'(0) = 0$. The ODE for g is

$$g'' + \frac{2n+1}{r}g' - \frac{\mathrm{i}nQ'}{r(p + \mathrm{i}n\Omega)}g = 0. \tag{4.9}$$

   Note that when calculating $g''(0)$, which is required numerically, one has to use l'Hopital's rule. Since $Q'(0) \sim qr$, we find

$$g''(0) + (2n+1)g''(0) = \frac{\mathrm{i}nq}{p + \mathrm{i}n\Omega_0}. \tag{4.10}$$

3. Work with $h = \phi^{1/n}$. This can be unpleasant in the complex case, because of branch cuts (especially when $0 < n < 1$ which is not physically possible for the Rayleigh equation unless one is working only in a sector which makes no sense for the basic state).

4.2.2 Irregular singular points

Frobenius series only work at regular singular points. At irregular singular points, there is no nice theory. The best approach is to use the substitution $y = \mathrm{e}^{S(x)}$ and solve as an asymptotic series. For (4.6), this is unnecessary and using $\Omega = O(r^{-2})$ and $Q = O(r^{-\infty})$[2], we find $\phi = O(r^n)$ for large r.

[2]Decays faster than any power of r.

Exercise 4.1 *Work out the behavior near the irregular singular point at infinity for*

$$f'' + \frac{1}{r}f' + \left[(1 + \mathrm{e}^{-r^2})k^2 - \frac{n^2}{r^2}\right] f = 0. \tag{4.11}$$

4.2.3 Connection coefficients

The connection coefficient Δ is the coefficient of the r^n term at large r. It is shown for the Gaussian vortex, $Q(r) = \mathrm{e}^{-r^2}$ in Figure 4.1. The code uses strategy 1.

```
% p41.m 04/10/08 SGLS
function [nn,res]=p41
global p n
nn=0:0.1:4;
n=2;
pp=1+i*nn;
r0=1e-4;
rmax=100;
for j=1:size(pp,2)
  p=pp(j);
  [t,y]=ode45(@f,[0 rmax],[1 0 0 0]);
  res(1,j)=y(end,1);
  res(2,j)=y(end,3);
end
plot(nn,res)
xlabel('Im p'); ylabel('\Delta')

function f=f(r,y)
global p n
f = zeros(4,1);
f(1) = y(2);
f(3) = y(4);
if (r==0)
  temp = i*n*2/(p+i*n*0.5)*(y(1)+i*y(3));
  f(2) = real(temp)/(2*n+2);
  f(4) = imag(temp)/(2*n+2);
else
  temp = i*n*qp(r)/r/(p+i*n*om(r))*(y(1)+i*y(3));
  f(2) = -(2*n+1)/r*y(2) + real(temp);
  f(4) = -(2*n+1)/r*y(4) + imag(temp);
end

function om=om(t)
om=(1-exp(-t.^2))/2./t.^2;
```

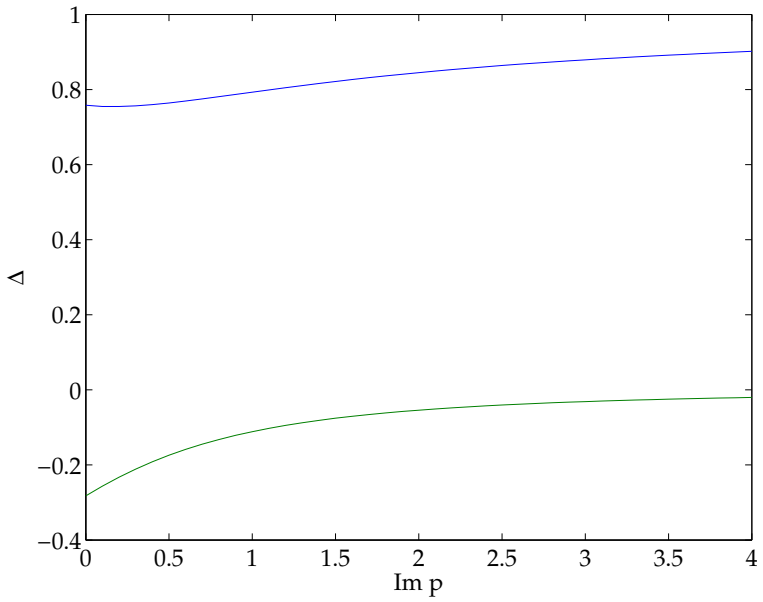

Figure 4.1: Real (blue) and imaginary (green) parts of the connection coefficient for the Gaussian vortex with $n = 2$.

```
function qp=qp(t)
qp=-2*t.*exp(-t.^2);
```

Exercise 4.2 *Obtain the expressions for $\Omega(r)$ and $Q'(r)$ for the Gaussian vortex.*

Exercise 4.3 *Rewrite* `p41.m` *to use both other strategies to deal with the regular singular point. Which of the three is easiest?*

Exercise 4.4 *Discuss the connection problem for (4.11).*

4.3 Integrating along complex paths

4.3.1 A problem of Boyd

We have looked at eigenvalue problems for complex-value functions, for which the complex attribute was mostly an algebraic property, and for which the special properties of complex analyticity were not used. We now consider a situation where it is very useful to solve an ODE along a path in the complex plane.

The following problem is discussed in Boyd (2001; § 7.11). Find the eigenvalues λ of

$$f_{yy} + \left(\frac{1}{y} - \lambda\right) = 0 \qquad \text{with } f(a) = f(b) = 0, a < 0 \text{ and } b > 0. \tag{4.12}$$

Here y is our real independent variable. The singularity at the origin means that this is not a regular Sturm–Liouville problem, and the solutions are singular. The remedy is to use a path in the complex plane. After stretching the interval trivially onto $[-1, 1]$, write

$$y = x + \mathrm{i}\Delta(x^2 - 1). \tag{4.13}$$

The solutions $f(y)$ has a branch point at the origin in the complex y-plane, and the choice of whether the path goes above or below the origin is given by the physics of the problem. Here we must go below the real axis, so $\Delta > 0$.

This is now a boundary-value problem. These are more difficult to program. Luckily, Matlab has one: `bvp4c`. However functions like `bvp4c` need an initial guess. One could divide the complex λ-plane up into little boxes and see what eigenvalue was returned by the initial guess, but this would be painful. Continuation won't help much because there is no parameter to vary.

Exercise 4.5 *Try this approach.*

Instead we look at methods to return all the eigenvalues. Note also that the path in the complex plane doesn't give a very useful estimate for the eigenfunctions, which are physical for real y.

4.3.2 Pseudospectral methods in one paragraph

One can classify numerical methods to solve differential equations chronologically: finite differences finite elements, spectral. Spectral methods are particularly good at obtaining high accuracy in certain problems. One subcategory of spectral methods, often called pseudospectral, enforces equations at certain points in the domain.

A new strand of the literature developed pointing out the intimate connection with linear algebra. One can represent a function $f(x)$ as a vector $\mathbf{f}$ of the values of $f(x_i)$ where the x_i are discretization points. Then the derivative $f'(x)$ can be represented at those points by a differentiation matrix D acting on $\mathbf{f}$. There is hence a very simple representation for differential equations.

Excellent books on this topic are Fornberg (1996), Boyd (2001) and Trefethen (2000), which is particularly pedagogical and also has Matlab code that can be downloaded. I tend to use the Matlab toolbox of Weideman & Reddy (2000), because it is more complete.

4.3.3 Solution to Boyd's problem

We solve (4.12) using the toolbox of Weideman & Reddy (2000). The x-variable is discretized at the appropriate Chebyshev points with $N = 40$, and the differential equation is turned into a generalized eigenvalue problem. Accuracy is outstanding: the first 7 eigenvalues

```
ans =

  0.125053784915595 + 0.284985539605466i
```

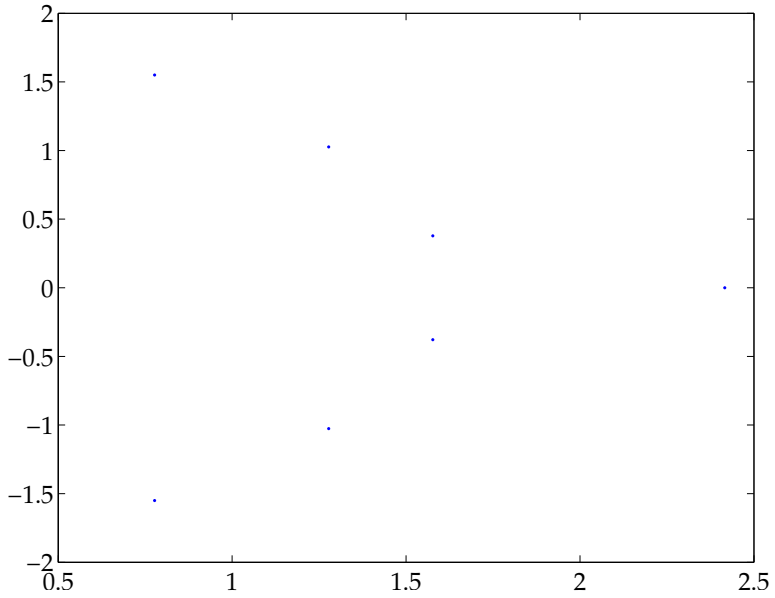

Figure 4.2: First 7 eigenvalues of (4.12) solved using a pseudospectral method.

```
-0.296730301617421 + 0.520320308706465i
-0.638802533479911 + 0.000261422784429i
-1.217501201556951 + 0.563348974818304i
-1.601895926942392 + 0.007822859918900i
-2.726011668004498 + 0.489965769815868i
-3.101209886006712 + 0.070299331841965i
```

are plotted in Figure 4.2 are essentially good to machine accuracy.

```
% p43.m Solve Boyd problem using pseudospectral method SGLS 04/10/08
N=42;
D=0.5;
a=-6;b=6;
[x,DM]=chebdif(N,2);
s=x+i*D*(x.^2-1);
y=a+(b-a)/2*(s+1);
D2=DM(:,:,2);
D1=DM(:,:,1);
M=4/(b-a)^2*( diag(1./(1+i*2*D*x).^2)*D2 - diag(i*2*D./(1+i*2*D*x).^3)*D1 ) + diag(1./y
v=eig(M(2:N-1,2:N-1));
[y,k]=sort(-real(v));
v=v(k);
plot(v(1:7),'.')
```

Boyd (2001; §7.5) has an excellent discussion of how to determine which zeros obtained from a pseudospectral approach are accurate, based on their behavior as N is increased.

4.3.4 Orr–Sommerfeld and Rayleigh equations

Both Trefethen (2000) and Weideman & Reddy (2000) provide codes to solve the Orr–Sommerfeld equation. The Orr–Sommerfeld equation is the viscous counterpart of the Rayleigh equation, which is strictly speaking only valid for Poiseuille-Couette basic flow. The Trefethen code `p40.m` is extremely simple. The corresponding figure are shown in Figure 4.3. One eigenvalue is just in the right half-plane.

```
% p40.m - eigenvalues of Orr-Sommerfeld operator (compare p38.m)

  R = 5772; clf, [ay,ax] = meshgrid([.56 .04],[.1 .5]);
  for N = 40:20:100

    % 2nd- and 4th-order differentiation matrices:
    [D,x] = cheb(N); D2 = D^2; D2 = D2(2:N,2:N);
    S = diag([0; 1 ./(1-x(2:N).^2); 0]);
    D4 = (diag(1-x.^2)*D^4 - 8*diag(x)*D^3 - 12*D^2)*S;
    D4 = D4(2:N,2:N);

    % Orr-Sommerfeld operators A,B and generalized eigenvalues:
    I = eye(N-1);
    A = (D4-2*D2+I)/R - 2i*I - 1i*diag(1-x(2:N).^2)*(D2-I);
    B = D2-I;
    ee = eig(A,B);
    i = N/20-1; subplot('position',[ax(i) ay(i) .38 .38])
    plot(ee,'.','markersize',12)
    grid on, axis([-.8 .2 -1 0]), axis square
    title(['N = ' int2str(N) '    \lambda_{max} = ' ...
        num2str(max(real(ee)),'%16.12f')]), drawnow
  end
```

Project 1 *Examine the irregular neutral modes of the Orr–Sommerfeld equation using paths in the complex planes. Relevant references: Engevik, Lin.*

4.4 References

Publications:

- Boyd, J. P. *Chebyshev and Fourier spectral methods.* Dover, New York, 2nd ed, 2001.

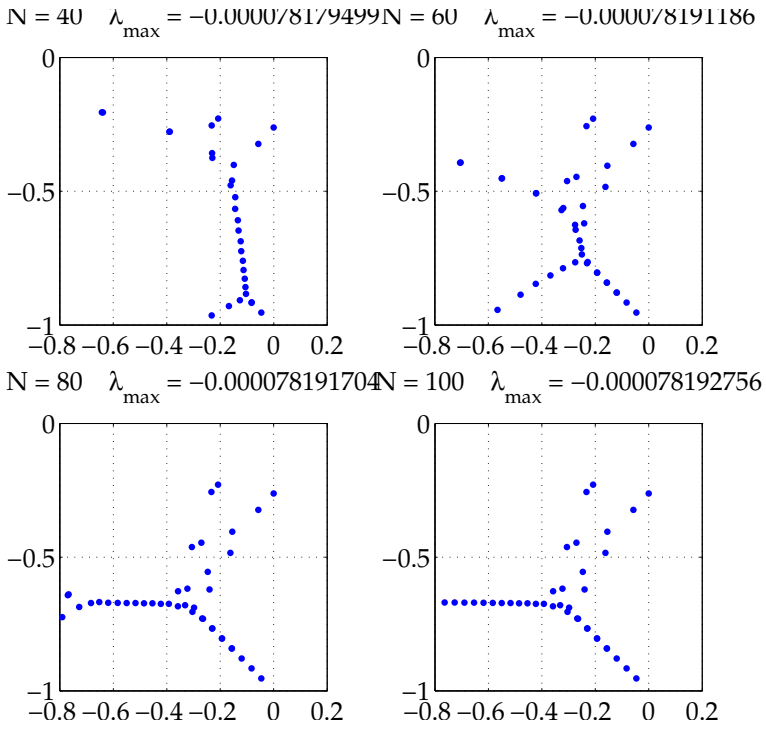

Figure 4.3: Eigenvalues of the Orr–Sommerfeld equation for $R = 5772$, Outout 40 of Trefethen (2000).

- Fornberg, B. *A practical guide to pseudospectral methods.* Cambridge University Press, Cambridge, 1998.
- Llewellyn Smith, S. G. *Vortices and Rossby-wave radiation on the beta-plane.* Ph.D. thesis, University of Cambridge, 1996.
- Olver, F. W. J. *Asymptotics and special functions.* A K Peters, Natick, 1997.
- Weideman, J. A. C. & Reddy, S. C. A MATLAB differentiation matrix suite. *ACM Trans. Math. Software*, **25**, 465–519, 2000.

Chapter 5

Integral transforms (04/23/2008)

5.1 Fourier series

5.1.1 Definition

Fourier series are the natural language to describe periodic functions. In fact, their use is more general. If $f(x)$ is a function of the real variable x, we can write

$$f(x) = \tfrac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos nx + \sum_{n=1}^{\infty} b_n \sin nx, \tag{5.1}$$

where the coefficients in the series are given by

$$a_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos nx\,\mathrm{d}x, \qquad b_n = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin nx\,\mathrm{d}x. \tag{5.2}$$

Note that (5.1) must be viewed as holding in the range $0 < x < 2\pi$. The equality holds under certain conditions on f, and investigating this and similar questions was a major priority of mathematicians in the first half of the 20th century. We will be as expeditious as usual. Note that if $f(x)$ has a discontinuity at x_0, the series on the right-hand side of(5.1) tends to the average of $f(x_0^-)$ and $f(x_0^+)$.

There is a natural extension of (5.1) from trigonometric to complex exponentials:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n \mathrm{e}^{\mathrm{i}nx}. \tag{5.3}$$

If $f(x)$ is real then $c_{-n} = \bar{c_n}$. The coefficients are given by

$$c_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)\mathrm{e}^{-\mathrm{i}nx}\,\mathrm{d}x. \tag{5.4}$$

Exercise 5.1 *Work out the relation between a_n and b_n in (5.1) and c_n in (5.3).*

Example 5.1 *Let us solve Laplace's equation $\nabla^2 u = 0$ in the circle $r < a$ with boundary condition $u = f(\theta)$ on $r = a$. The coefficients of the Laplacian in polar coordinates are independent of θ, since*

$$\nabla^2 u = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2} = 0. \tag{5.5}$$

Hence we expand in a Fourier series in θ:

$$u = \sum_{n=-\infty}^{\infty} u_n(r)\mathrm{e}^{\mathrm{i}n\theta}. \tag{5.6}$$

Periodicity means this goes through well and we obtain

$$\frac{\mathrm{d}^2 u_n}{\mathrm{d}r^2} + \frac{\mathrm{d}u_n}{\mathrm{d}r} - \frac{n^2}{r^2}u_n = 0. \tag{5.7}$$

We decompose the boundary condition in the same way as $f(\theta) = \sum_{n=-\infty}^{\infty} f_n \mathrm{e}^{\mathrm{i}n\theta}$. At the origin, we require u_n to be finite. Hence solving the ODE (5.7) with the appropriate boundary conditions gives $u = f_n(r/a)^{|n|}$. Putting this altogether, we obtain

$$\begin{aligned} u(r,\theta) &= \sum_{n=-\infty}^{\infty} f_n \left(\frac{r}{a}\right)^n \mathrm{e}^{\mathrm{i}n\theta} = \frac{1}{2\pi}\int_0^{2\pi} f(\phi) \sum_{n=-\infty}^{\infty} \mathrm{e}^{\mathrm{i}n(\theta-\phi)} \left(\frac{r}{a}\right)^n \mathrm{d}\phi \\ &= \frac{1}{2\pi}\int_0^{2\pi} f(\phi)\left[1 + \sum_{n=1}^{\infty} \mathrm{e}^{\mathrm{i}n(\theta-\phi)}(r/a)^n + \sum_{n=1}^{\infty} \mathrm{e}^{-\mathrm{i}n(\theta-\phi)}(r/a)^n\right] \mathrm{d}\phi \\ &= \frac{r^2 - a^2}{2\pi}\int_0^{2\pi} \frac{f(\phi)}{a^2 - 2ar\cos(\phi-\theta) + r^2}\,\mathrm{d}\phi. \end{aligned} \tag{5.8}$$

The geometric series converge for $r < a$. This is Poisson's integral formula for the Dirichlet problem in a circle. Figure 5.1 shows the value of u as a function of ϕ inside the circle. We use the trapezium rule since this is a periodic integral.

```
% p51.m Poisson's integral formula SGLS 04/14/08
a = 1.5; r = 0.5;
P = inline('sin(3*phi).*exp(-(phi-1).^2)./(a^2 - 2*a*r*cos(phi-theta)+r^2)','phi','thet
n = 100; h = 1./n;
phi = 2*pi*h*(1:n); dphi = 2*pi;
th = (0:0.01:1)*2*pi;
for j = 1:length(th)
  theta = th(j);
  u(j) = h*sum(P(phi,theta,r,a).*dphi);
end
plot(th,u,th,cos(3*th).*exp(-(th-1).^2))
xlabel('\theta'); ylabel('u')
```

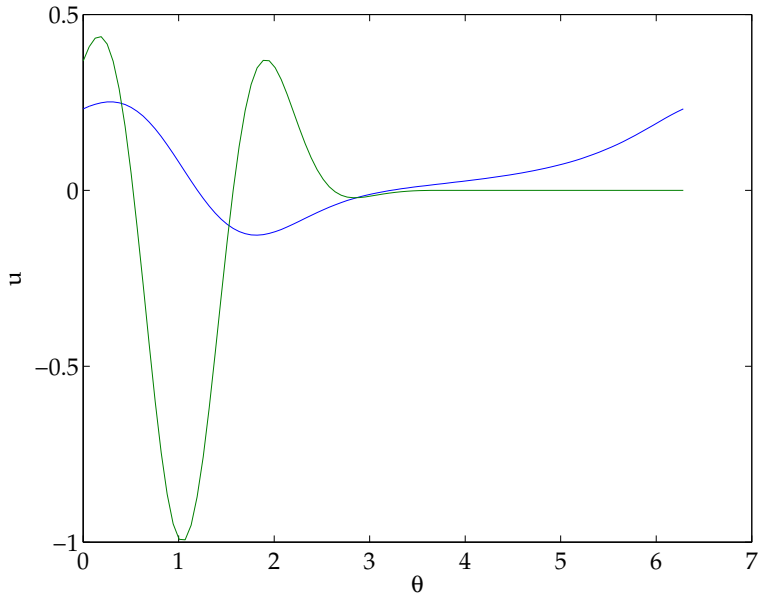

Figure 5.1: Plot of $u(\phi, 0.5)$ (blue) and $u(\phi, 1.5)$ (green) where u solves the Laplace equation in the circle $r < 1.5$ and boundary condition $u(\phi, 1.5) = \sin 3\phi e^{-(\phi-1)^2}$.

Exercise 5.2 *Work out the solution to the Neumann problem in the disk $r < a$ with $\partial u/\partial = g(\theta)$ on the boundary. Write a Matlab program to plot the output.*

Warning 8 *When writing a function $f(x)$ as a series of expansion functions $f_n(x)$, as for a Fourier series, always multiply the equation by $f_n(x)$ and integrate. This approach can cope when the boundary values of $f(x)$ and $f_n(x)$ do not match up.*

Example 5.2

Sine and cosine series are just special cases of general Fourier series and are not pursued here. One can think about how to generalize Fourier series appropriately. One way is to characterize expansion function by their zeros, another is by the way the Sturm–Liouville problem they solve. A final way is to think about the "continuum limit", in which there are more and more modes. We continue along these lines.

5.2 Fourier transforms

Fourier series are defined on an internal and have a countably infinite number of modes (discrete sum over integer). One can think about taking a large and large interval, and more and more modes. Intuitive reasoning about the Riemann sum takes us in a cavalier fashion to the Fourier transform and inverse transform

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-ikx}\, dx, \qquad f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(k) e^{ikx}\, dk. \tag{5.9}$$

The existence of an inverse transform, which is the manifestation of Fourier's theorem is critical to the use of Fourier transforms. Abstractly, we Fourier transform a problem, solve for the Fourier transform, and return to the original variable.

One of the most important facts about Fourier transforms for our purposes are their analyticity properties. The Fourier transform defined in (5.9) is complex because of the exponential, but the presence of the integral sign in it and in particular in the inverse transform lead to results that make use of analyticity. Again we do not worry about the conditions on $f(x$ for these: provided f is well-enough behaved, f is analytic for large enough $|k|$, although there may be branch cuts extending to infinity.

Most Fourier transforms that one comes across in textbooks can be evaluated by hand. The inverse transforms are not so easy, but often succumb to contour integration.

Example 5.3 *Calculate the inverse Fourier transform of*

$$F(k) = \frac{1}{k^4+1}. \tag{5.10}$$

We need to compute

$$I = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{\mathrm{e}^{\mathrm{i}kx}}{k^4+a^4}\,\mathrm{d}k. \tag{5.11}$$

For $x > 0$, we close in the upper half-plane, where the integrand has simple poles at $\mathrm{e}^{\mathrm{i}\pi/4}$ and $\mathrm{e}^{3\mathrm{i}\pi/4}$. Hence

$$I = \mathrm{i}\left(\frac{\mathrm{e}^{\mathrm{i}x(1+\mathrm{i})/\sqrt{2}}}{4\mathrm{e}^{3\mathrm{i}\pi/4}} + \frac{\mathrm{e}^{\mathrm{i}x(-1+\mathrm{i})/\sqrt{2}}}{4\mathrm{e}^{9\mathrm{i}\pi/4}}\right) = \tfrac{1}{2}\mathrm{e}^{-x/\sqrt{2}}\cos\left(x+\pi/4\right). \tag{5.12}$$

For negative x, we can use the symmetry properties of the Fourier transform to obtain the general result with $|x|$ replacing x.

This is an example which can be carried out by hand. Let's try something more esoteric using a little numerical calculation. The following function is certainly not the result of computing a Fourier transform:

$$F(k) = \frac{\mathrm{e}^{-1/(k^2+1)^2}}{k^2+1}. \tag{5.13}$$

This has essential singularities at $\pm\mathrm{i}$. Note that for large $|k|$, the exponential term looks like 1 and the denominator ensures convergence for $x = 0$. For $x > 0$ we can compute the inverse transform just by doing the integral around a little circle about the pole. The results are shown in Figure 5.2. Note that the imaginary part of $f(x)$ is zero to machine accuracy.

```
% p52.m inverse Fourier transform of essential singularity SGLS 04/16/08
a = 1.5; r = 0.5;
F = inline('exp(i*k*x-1./(k.^2+1).^2)./(k.^2+1)','k','x');
n = 100; h = 1./n;
```

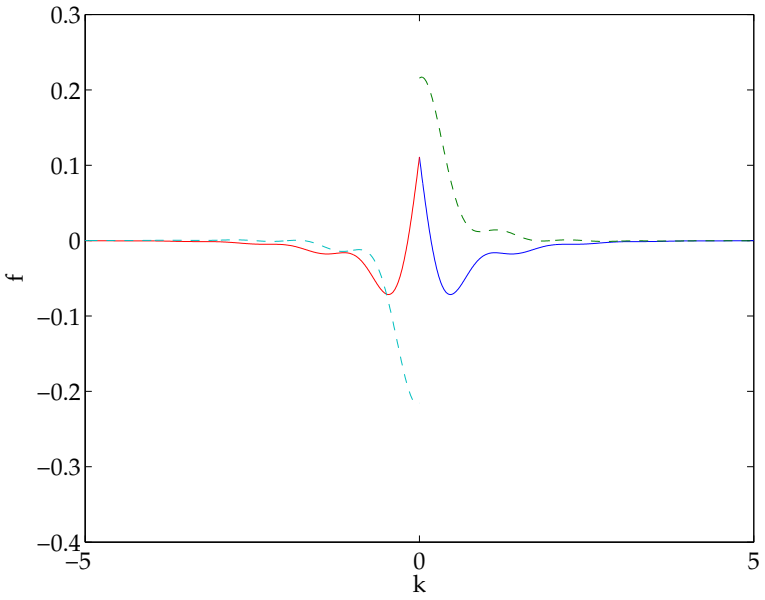

Figure 5.2: Solution to (4.12) computed using Real (solid) and imaginary (dashed) parts of the inverse transform $f(x)$ of (5.13).

```
k = i + 0.5*exp(i*2*pi*h*(1:n)); dk = 0.5*2*pi*i*exp(i*2*pi*h*(1:n));
xp = 0:.01:5;
for j = 1:length(xp)
  fp(j) = h*sum(F(k,xp(j)).*dk);
end
k = -i + 0.5*exp(i*2*pi*h*(1:n)); dk = 0.5*2*pi*i*exp(i*2*pi*h*(1:n));
xm = -5:.01:0;
for j = 1:length(xm)
  fm(j) = -h*sum(F(k,xm(j)).*dk);
end
plot(xp,real(fp),xp,imag(fp),'--',xm,real(fm),xm,imag(fm),'--')
xlabel('k'); ylabel('f')
```

Warning 9 *The Riemann–Lebesgue lemma guarantees that the Fourier transform of a bounded function tends to* 0 *as* $k \to \infty$ *(this is immediately obvious along the imaginary axis). Hence if you are dealing with a function* $F(k)$ *that does not have this property, it is probably unphysical or wrong.*

Exercise 5.3 *Explain when inverse Fourier transforms are discontinuous.*

Exercise 5.4 *Obtain the inverse transform of* $F(k)$ *as an infinite sum (maybe even a doubly infinite sum). Can you sum it numerically in an efficient way?*

5.3 Using Fourier transforms

The critical property of Fourier transforms from a practical perspective is that they enable one to replace differentiation, an analytic concept, with multiplication, an algebraic concept. Fourier transforms are appropriate for equations with constant coefficients, or with very special and simple coefficients, and with appropriate decay properties.

We have (formally)

$$f'(x) \to \mathrm{i}kF(k), \qquad xf(x) \to \mathrm{i}F'(k). \tag{5.14}$$

The proof is simple: integrate by parts and wave your hands.

As an example, we solve the ODE

$$f'' + \frac{1}{x}f' - a^2 f = 0 \tag{5.15}$$

where a is real and positive. Multiply by x and apply the rules (5.14) to obtain

$$\mathrm{i}\frac{\mathrm{d}}{\mathrm{d}k}[(-a^2 - k^2)F] + \mathrm{i}kF = 0. \tag{5.16}$$

This has solution $F(k) = A(a^2 + k^2)^{-1/2}$. We have a formal solution, which contains a branch cut. Pick $A = \pi$ so that $F(0) = \int_{-\infty}^{\infty} f(x)\,\mathrm{d}x = \pi a^{-1}$. One can show that this solution is the one usually written $K_0(ax)$. Note that we have only found one solution. This is because all the solutions grow exponentially.

We can't express $f(x)$ as an integral around a singularity any longer. We need first to pick a branch. The transform $F(k)$ has branch points at $\pm cia$. We can't have branch cuts cross the inversion contour. The natural cuts are from $\pm a$ to $\pm\infty$, with $a^2 + k^2$ real and positive on the real axis. Consider now $x > 0$. We can deform the integration contour up and around the branch cut, so that

$$f(x) = \frac{1}{2\pi}\int_1^\infty \frac{2\pi \mathrm{e}^{-ux}}{\mathrm{i}(u^2 - a^2)^{1/2}}\mathrm{i}\,\mathrm{d}u = \int_1^\infty \frac{\mathrm{e}^{-ux}}{(u^2 - a^2)^{1/2}}\,\mathrm{d}u. \tag{5.17}$$

Using this cut has led to exponential convergence in the integral. We can see that $f(x)$ is singular as $x \to 0$. This is rapid enough to ensure that essentially any change of variable works. To use a general-purpose integration package, it is advisable to remove the integrable singularity at $u = 1$ by writing $u = 1 + v^2$. Figure 5.3 shows $f(x)$ for positive x and $a = 1$ using the Matlab `quad` function. Because of the exponential decay, we truncate the integral at $v = 50/x$, where the exponential in the integrand is approximate $\mathrm{e}^{-50} \approx 2 \times 10^{-22}$, which is comfortably smaller than the error estimate for `quad` (10^{-6} by default).

```
% p53.m inverse Fourier transform for K_0 SGLS 04/14/08

fint = inline('exp(-(1+v.^2)*x)./sqrt(2+v.^2)*2','v','x');
x = 0.01:.01:5;
```

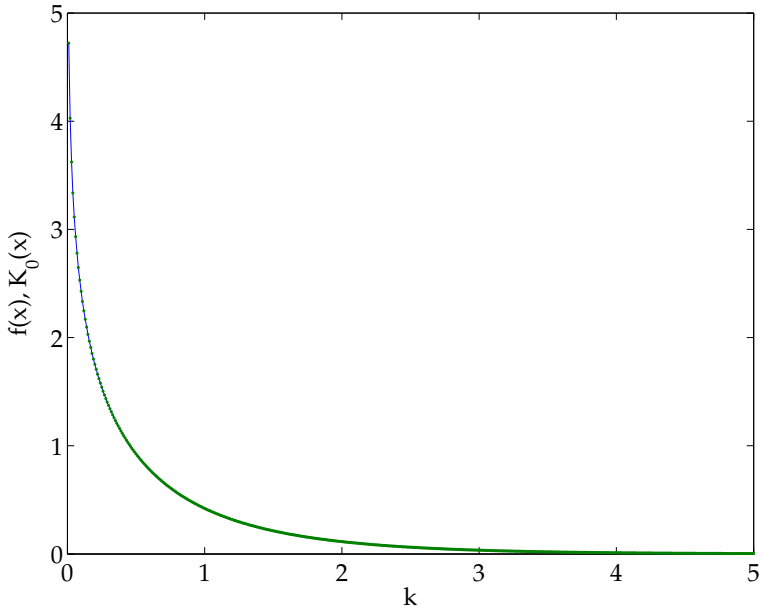

Figure 5.3: Inverse Fourier transform solution to (4.12) (blue); $K_0(x)$ (green).

```
for j = 1:length(x)
  f(j) = quad(@(v) fint(v,x(j)),0,50/sqrt(x(j)));
end
plot(x,f,x,besselk(0,x),'.');
xlabel('k'); ylabel('f(x), K_0(x)')
```

Exercise 5.5 *From (5.17), discuss the singularity in $K_0(x)$ as $x \to 0$.*

Exercise 5.6 *From (5.15) discuss the behavior of the solutions near the irregular singular point at infinity.*

Most wave problems that can be dealt with have constant coefficients, so these kinds of ODEs in k will not appear. There can still be branch cuts associated with the continuous spectrum, which we gloss over for now.

5.4 h-transforms and other transforms

In our inexorable pursuit of generalization, we arrive at the idea of h-transforms (Bleistein & Handelsman 1986). We will use s and k as the transform variables, more or less according to tradition. The h-transform of $f(x)$ is given by

$$F(\lambda) = \int f(x)h(\lambda x)\,\mathrm{d}x. \tag{5.18}$$

The range of integration depends on the function h. Our notation is left deliberately vague. The inversion integral is not specifically mentioned here. In general, all inversion integrals come from the Fourier inversion theorem, which is the result underlying (5.9). This indicates that the majority of these transforms are just the Fourier transform by another name.

In this way, we have the (one-sided) Laplace transform and inverse

$$F(s) = \int_0^\infty f(x)\mathrm{e}^{-sx}\,\mathrm{d}x, \qquad f(x) = \frac{1}{2\pi\mathrm{i}}\int_\Gamma F(s)\mathrm{e}^{sx}\,\mathrm{d}x. \tag{5.19}$$

where Γ is the Bromwich contour, a vertical contour parallel to the imaginary axis and to the right of all singularities of $F(s)$. For large $|k|$ with positive real part, $F(s)$ is analytic. In fact, the inverse transform returns 0 for $x < 0$, so be careful with Laplace transforms of functions which are non-zero for $x < 0$.

We also have the Mellin transform

$$F(s) = \int_0^\infty f(x)x^{s-1}\,\mathrm{d}x, \qquad f(x) = \frac{1}{2\pi\mathrm{i}}\int_{c-\mathrm{i}\infty}^{c+\mathrm{i}\infty} F(s)s^{-x}\,\mathrm{d}s, \tag{5.20}$$

which can be obtained from the Laplace transform by a change of variable. The Mellin transform is not an h-transform. The domain of analyticity of the Mellin transform is usually between two vertical lines in the complex s-plane.

Less immediately obvious is the Hankel transform

$$F(k) = \int_0^\infty f(x)J_\nu(kx)x\,\mathrm{d}x, \qquad f(x) = \int_0^\infty F(k)J_\nu(kx)k\,\mathrm{d}k, \tag{5.21}$$

which can be motivated for $n = 0$ by the two-dimensional Fourier transform of a radially symmetric function. We requires $\nu \geq -1/2$. There are some other definitions that include square roots.

The generalized Stieltjes transform is

$$F(s) = s^\nu \int_0^\infty \frac{f(x)}{(1+sx)^\nu}\,\mathrm{d}x. \tag{5.22}$$

Again there are other definitions. I don't even know the inverse transform.

The Kontorovich–Lebedev transform is

$$F(s) = \int_0^\infty f(x)K_{\mathrm{i}x}(s)\,\mathrm{d}x, \qquad f(x) = \frac{2}{\pi^2 x}\int_0^\infty F(s)K_{\mathrm{i}s}(x)\sinh(\pi s)s\,\mathrm{d}x. \tag{5.23}$$

This is unusual because the index of the modified Bessel function (sometimes called Macdonald function) occurs, rather than its argument.

Sneddon (1972) is a detailed and useful exposition of most of the integral transforms one can think of. An exhaustive list can be found in Zayed (1996), while Debnath & Bhatta (2006) is slightly less general. Davies (2002) is a clear pedagogical work that is good on complex analysis.

We do not give examples of the use of these transforms here, but will do so in Chapter 8. In most cases, the choice of transform to use is dictated by the geometry of the system (Fourier for straight boundaries, Mellin in wedges for the Laplace equation in two dimensions, Kontorovich–Lebedev in wedges for the Laplace equation in three dimensions).

5.5 Hilbert transforms

The Hilbert transform has a slightly different origin, and is related very closely to the notion of analyticity. We start with an example; solve $\nabla^2 u = 0$ in the half-plane $y > 0$ with $u(x,0) = g(x)$ and $u \to 0$ as $y \to \infty$. Then we can Fourier transform in x and obtain

$$\frac{\mathrm{d}^2 U}{\mathrm{d}y^2} - k^2 U = 0, \tag{5.24}$$

with $U(0) = G$, the Fourier transform of $g(x)$, and $U \to 0$ as $y \to \infty$. The solution is clearly $U = G\mathrm{e}^{-|k|y}$, where the modulus sign can be viewed as a placeholder for $||_\epsilon$ (see Chapter 1). Then the inverse Fourier transform of this result gives

$$f = \frac{1}{2\pi}\int_{-\infty}^{\infty} G(k)\mathrm{e}^{-|k|y}\,\mathrm{d}k = \frac{y}{\pi}\int_{-\infty}^{\infty}\frac{g(\xi)}{(x-\xi)^2+y^2}. \tag{5.25}$$

We have used the convolution theorem, which states that the Fourier transform of

$$f * g(x) = \int_{-\infty}^{\infty} f(\xi)g(x-\xi)\,\mathrm{d}\xi \tag{5.26}$$

is $F(k)G(k)$.

Exercise 5.7 *Prove the convolution theorem.*

Exercise 5.8 *Show that the inverse Fourier transform of $\pi\mathrm{e}^{-|k|y}$ is $y(x^2+y^2)^{-1}$.*

Exercise 5.9 *Show that as $y \to 0$, $u(x,y) \to g(x)$. The best approach is to consider different regions of the integral separately.*

UNDER CONSTRUCTION (I.E. WRONG) Since u is a harmonic function, it can be written as the real part of an analytic function $f(z)$. Taking $g(x)$ to be real, we see immediately see that

$$f(z) = \frac{1}{\pi}\int_{-\infty}^{\infty}\frac{g(\xi)}{z-\xi}\,\mathrm{d}\xi, \tag{5.27}$$

which is just Cauchy's integral formula. The imaginary part of this gives a new function

$$v(x,y) = \frac{y}{\pi}\int_{-\infty}^{\infty}\frac{g(\xi)}{(x-\xi)^2+y^2}\,\mathrm{d}\xi. \tag{5.28}$$

With a nod to (5.27) we define the Hilbert transform as

$$\mathcal{H}(k) = \frac{1}{\pi}\fint_{-\infty}^{\infty}\frac{h(x)}{x-k}\,\mathrm{d}x. \tag{5.29}$$

Note that the Hilbert transform is not an h-transform. The inversion formula is

$$h(x) = -\frac{1}{\pi}\fint_{-\infty}^{\infty}\frac{\mathcal{H}(k)}{k-x}\,\mathrm{d}k. \tag{5.30}$$

The transform of $h(x)$ can be viewed as the convolution of $h(x)$ with $(\pi x)^{-1}$. The Fourier transform of this function is $\mathrm{i}\,\mathrm{sgn}\,k$.

Exercise 5.10 *Show that the Fourier transform of* $(\pi x)^{-1}$ *is* $\operatorname{sgn} k$.

Exercise 5.11 *Show that applying the Hilbert transform twice to* $f(x)$ *returns* $-f(x)$ *(use the convolution theorem).*

from u. Using Fourier transforms, we can show that $V = \mathrm{i} \operatorname{sgn} kU$. The Hilbert transform is used extensively in signal processing.

Project 2 *Write a report on the use of Hilbert transforms in signal processing, including examples using some of the complex techniques we have been discussing.*

5.6 References

Publications:

- Bleistein, N. & Handelsman, R. A. *Asymptotic expansions of integrals.* Dover, New York, 1986.
- Davies, B. *Integral transforms and their applications.* Springer, New York, 2002.
- Debnath, L, & Bhatta, D. *Integral transforms and their applications.* Chapman & Hall/CRC, Boca Raton, 2007.
- Sneddon, I. N. *The use of integral transforms.* Mc-Graw Hill, New York, 1972.
- Zayed, A. I. *Handbook of function and generalized function transformations.* CRC, Boca Raton, 1996.

Chapter 6

The Fast Fourier Transform (04/22/08)

6.1 Fourier coefficients

The Fast Fourier Transform (FFT) is a transform by name, but one that takes a finite set of numbers (e.g. data at points) to a finite set of coefficients. As such it is more like a Fourier series (function to coefficients) than a Fourier transform (function to function). The Fast refers to efficiency, not to the definition of the transform, and one can argue that the FFT is just an implementation of the Discrete Fourier Transform (DFT), where Discrete refers to its working on values at points.

Hence we start by considering Fourier coefficients in Fourier series: how to compute them and some of their properties. In textbook cases, we can find the coefficients by hand. (It would be easier to work in the interval $(-\pi, \pi)$ for some of the integrals, but for the DFT the original interval $(0, 2\pi)$ is convenient.)

Example 6.1 *Compute the Fourier series of* $f(x) = -\operatorname{sgn}(x - \pi)$, *i.e.* 1 *for* $0 < x < \pi$ *and* -1 *for* $\pi < x < 2\pi$ *and then repeated periodically. From the definition of* c_n, *we have*

$$\begin{aligned} c_n &= \frac{1}{2\pi}\int_0^{2\pi} f(x)\mathrm{e}^{-\mathrm{i}nx}\,\mathrm{d}x = \frac{1}{2\pi}\int_0^{\pi} \mathrm{e}^{-\mathrm{i}nx}\,\mathrm{d}x - \frac{1}{2\pi}\int_\pi^{2\pi} \mathrm{e}^{-\mathrm{i}nx}\,\mathrm{d}x \\ &= \frac{2}{n\pi} \end{aligned} \tag{6.1}$$

Note the n^{-1} *decay for large* n. *The Fourier sum with* n *ranging from* -50 *to* 50 *is shown in Figure 6.1, along with a filtered version where the amplitude of the higher modes is reduced.*

```
% p61.m Fourier series for -sgn(x-pi) SGLS 04/16/08
nn = (-50:50)';
c = 2/pi*(-1).^((nn-1)/2)./nn; c(1:2:end) = 0;
d = c.*exp(-abs(nn).^2/500);
x =(-1:0.01:1)*2*pi;
f = sum(exp(i*nn*x).*(c*ones(size(x))));
g = sum(exp(i*nn*x).*(d*ones(size(x))));
plot(x,f,x,g)
xlabel('x'); ylabel('f')
```

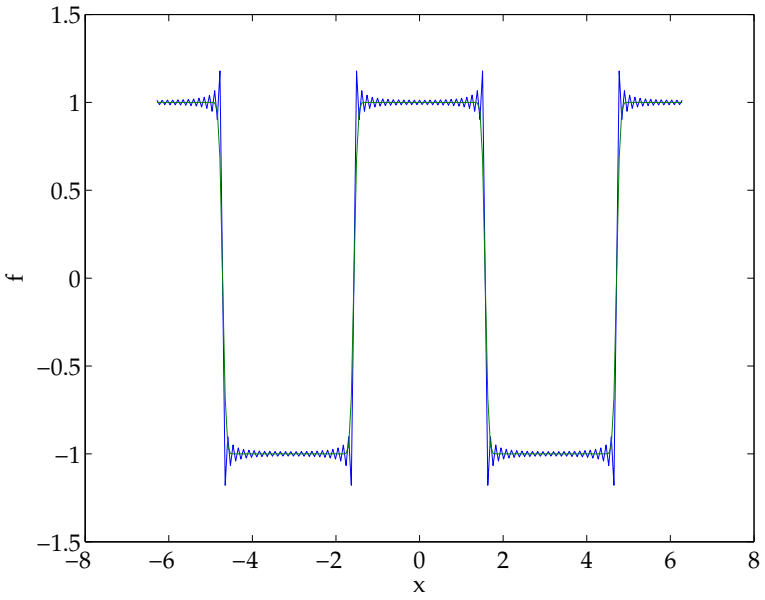

Figure 6.1: Fourier coefficients of a step function with and without filtering.

Note that the discontinuities in $f(x)$ leads to oscillations near the discontinuities. There is pointwise convergence, but for a finite truncation of the sum, there is always an overshoot, which moves closer to the point of discontinuity. This is the Gibbs phenomenon. To use these functions in real applications, one usually needs to filter, and some care is required to avoid losing too much of the high-wavenumber energy.

Quasitheorem 6 *The Fourier coefficients of a function $f(x)$ with a discontinuity in $f^{(j)}(x)$ behave like n^{-j-1} for large n.*

The Fourier coefficients of $\delta(x-a)$ are simply $c_n = \mathrm{e}^{-ina}/(2\pi)$, i.e. they do not decay with n. Progressively spikier generalized functions, $\delta'(x-a)$, and so on, have coefficients that go like n^j where j is the number of derivatives on δ. All this can (and should) be done properly; see for example Lighthill (1959) for a brief and dare I say challenging account.

Now let us try a more complicated function from the previous chapter: $f(x) = x(4x^2+1)^{-1}$. A closed form expression can be found for the coefficients c_n, but we want to compute it numerically. The function $f(x)$ is not periodic, so we do not quite have the nice properties of the trapezoid rule as applied to periodic functions. However, the exponential term is pure real at the end points, so the imaginary part is periodic. We use the periodic version of the the rule, thereby committing a small error. Our 101-term sum is a good approximation, as shown in Figure 6.2.

```
% p62: numerical integration for Fourier coefficients 04/16/2008 SGLS
f = inline('x./(4*x.^2+1)');
n = 100; h = 1./n;
c = zeros(101,1);
```

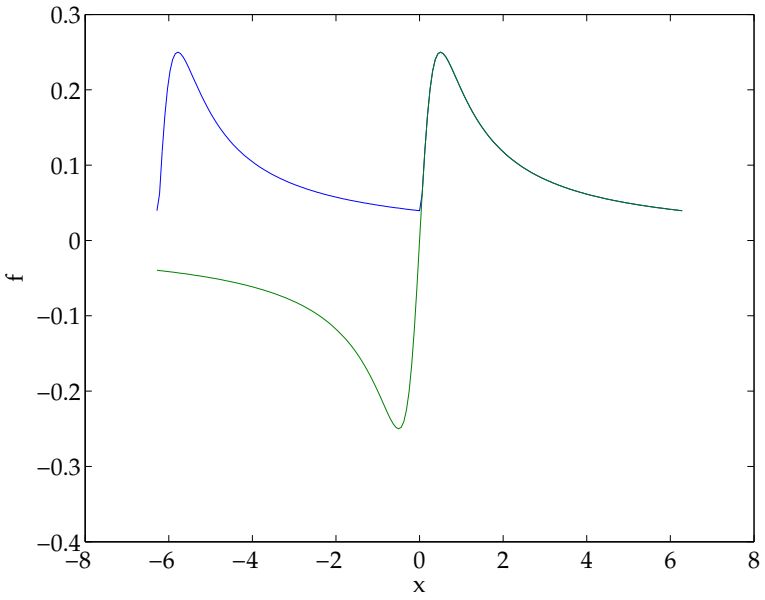

Figure 6.2: Fourier series and exact values of $x(4x^2+1)^{-1}$.

```
x = 2*pi*h*(1:n);
for n = -50:50
  c(n+51) = h*sum(f(x).*exp(-i*n*x));
end
x =(-1:0.01:1)*2*pi;
fs = sum(exp(i*nn*x).*(c*ones(size(x))));
plot(x,fs,x,f(x))
xlabel('x'); ylabel('f')
```

Exercise 6.1 *Find a closed form for the c_n of* `p62.m`*. You will end up with special functions. Write a Matlab program to compare the error in the numerical integration of* `p62.m`*; examine how the error depends on the number of terms in the quadrature rule and in the Fourier sum.*

We now argue in the following way. If we knew the function, or data, only at equispaced points along the real axis: $h, \ldots, 2\pi$ with values $f_m = f(x_m)$, $m = 1 \ldots M$, then we could view these data as approximating a period function with $f(0) = f(2\pi)$. The trapezoidal rule would then become exponentially accurate. We would have found the coefficients as a linear combination of the data,

$$c_m = \sum_{n=1}^{N} f_n \mathrm{e}^{-\mathrm{i}mnh}, \tag{6.2}$$

and the function can be represented as the series

$$f_n = \sum_{m=-M}^{M} c_m e^{imhn}. \tag{6.3}$$

The DFT is a formalization of this idea, showing that it all works and tidying up the notation (we have not used the optimal intervals above).

Trefethen (2000; Chapter 3) has a very clear discussion of the FFT. He uses a slightly different notation and considers both matrix-based and FFT approaches to representing functions on periodic grids.

6.2 The DFT

One of the major problems with the DFT and the FFT is notation and order of indexing. Matlab has a self-consistent approach, which is not necessarily intuitive. We follow Matlab's definition.

```
>> help fft
 FFT Discrete Fourier transform.
    FFT(X) is the discrete Fourier transform (DFT) of vector X.  For
    matrices, the FFT operation is applied to each column. For N-D
    arrays, the FFT operation operates on the first non-singleton
    dimension.

    FFT(X,N) is the N-point FFT, padded with zeros if X has less
    than N points and truncated if it has more.

    FFT(X,[],DIM) or FFT(X,N,DIM) applies the FFT operation across the
    dimension DIM.

    For length N input vector x, the DFT is a length N vector X,
    with elements
                     N
       X(k) =       sum  x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.
                    n=1
    The inverse DFT (computed by IFFT) is given by
                     N
       x(n) = (1/N) sum  X(k)*exp( j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.
                    k=1

    See also FFT2, FFTN, FFTSHIFT, FFTW, IFFT, IFFT2, IFFTN.

    Overloaded functions or methods (ones with the same name in other directories)
       help uint8/fft.m
```

```
help uint16/fft.m
help gf/fft.m
help iddata/fft.m
```

Ignore the multi-dimensional and zero-padding aspects. We have a transform that goes from a vector with entries x_n to a vector with entries X_k, according to

$$X_k = \sum_{n=1}^{\infty} x_n e^{-2\pi i(k-1)(n-1)/N}, \qquad 1 \le k \le N, \tag{6.4}$$

$$x_n = \frac{1}{N} \sum_{k=1}^{\infty} X_k e^{2\pi i(k-1)(n-1)/N}, \qquad 1 \le n \le N. \tag{6.5}$$

Compare this to (6.2) and (6.3). The notation is designed to deal with Matlab's suffix convention that starts at 1.

The first thing to do is check that (6.5) and (6.5) really are self-inverse. Substitute the first into the second; this gives the double sum

$$\begin{aligned} x_m &\stackrel{?}{=} \frac{1}{N} \sum_{k=1}^{\infty} \left(\sum_{n=1}^{\infty} x_n e^{-2\pi i(k-1)(n-1)/N} \right) e^{2\pi i(k-1)(m-1)/N} \\ &= \frac{1}{N} \sum_{n=1}^{\infty} x_n \left(\sum_{j=0}^{N-1} e^{2\pi i j(m-n)/N} \right). \end{aligned} \tag{6.6}$$

The final sum is a geometric progression with ratio $e^{2\pi i(m-n)/N}$. If $m \neq n$, the sum yields a fraction with numerator $1 - e^{2\pi i(m-n)} = 0$ since m and n are integers. If $m = n$, every element in the sum is 1 and there are N terms in the sum. Hence

$$x_m \stackrel{?}{=} \frac{1}{N} \sum_{n=1}^{\infty} x_n N \delta_{mn} = x_n \tag{6.7}$$

and we are done.

WRONG: ISSUES WITH COEFFICIENTS. One of the reasons this is useful is how it relates to derivatives. If we view X_n as representing a function $f(x)$, with the function being sampled at points $x = 2\pi(n-1)/N$.then

$$f(x) = \frac{1}{N} \sum_{k=1}^{\infty} X_k e^{i(k-1)x} \tag{6.8}$$

with the Matlab convention. We can now differentiate:

$$f'(x) = \frac{1}{N} \sum_{k=1}^{\infty} i(k-1) X_k e^{2\pi i(k-1)x}, \tag{6.9}$$

so the coefficients of the derivative $f'(x)$ are $Y_k = i(k-1)X_k$. This is tremendously useful.

Clearly the FFT of a data vector is related to the Fourier coefficients as the length of the vector increases. It also has intrinsic interest in signal processing, which we will not discuss here. Our interest lies in applications so continuum problems.

6.3 FFT

The FFT proper refers to a fast way to calculate the DFT of a data vector x_n of size N. There are N terms to calculate, and each is a sum of N terms, so the naive estimate is that

the process takes N^2 operations. In fact, the transform can be done in $N \log N$ operations, as pointed out by Cooley & Tukey (1965)[1]. It turns out that they had been anticipated by Gauss. This is not a course about algorithms, so we will not explain the algorithm here, but to be brief it relies on a recursive way of computing the terms in the sum, once it has been written in a clever way. Hence the logarithmic dependence. This recursion requires a certain amount of overhead, and it is not useful to continue it down to its final value, which is a sum of 1 term, since it is straightforward and fast to compute the DFT of a small vector.

The question of what is meant by small is a good one. The answer will depend on the architecture and memory (cache especially) of the machine being used. Platform-dependent implementation exist and those are fast. A particularly nice approach is FFTW, which runs test cases before executing a calculation and determines the fastest approach. Matlab now uses FFTW.

Exercise 6.2 *Write a Matlab program to examine the speed of FFTW in matlab. Investigate the command* `fftw`.

6.4 Solving linear PDEs with FFTs

We can use some programs from Trefethen (2000) to illustrate this. The numerical derivatives of two simple functions is shown in Figure 6.3.

```
% p5.m - repetition of p4.m via FFT
%        For complex v, delete "real" commands.

% Differentiation of a hat function:
  N = 24; h = 2*pi/N; x = h*(1:N)';
  v = max(0,1-abs(x-pi)/2); v_hat = fft(v);
  w_hat = 1i*[0:N/2-1 0 -N/2+1:-1]' .* v_hat;
  w = real(ifft(w_hat)); clf
  subplot(3,2,1), plot(x,v,'.-','markersize',13)
  axis([0 2*pi -.5 1.5]), grid on, title('function')
  subplot(3,2,2), plot(x,w,'.-','markersize',13)
  axis([0 2*pi -1 1]), grid on, title('spectral derivative')

% Differentiation of exp(sin(x)):
  v = exp(sin(x)); vprime = cos(x).*v;
  v_hat = fft(v);
  w_hat = 1i*[0:N/2-1 0 -N/2+1:-1]' .* v_hat;
  w = real(ifft(w_hat));
  subplot(3,2,3), plot(x,v,'.-','markersize',13)
  axis([0 2*pi 0 3]), grid on
  subplot(3,2,4), plot(x,w,'.-','markersize',13)
```

[1] I don't know if anyone reads this paper any more. I never have.

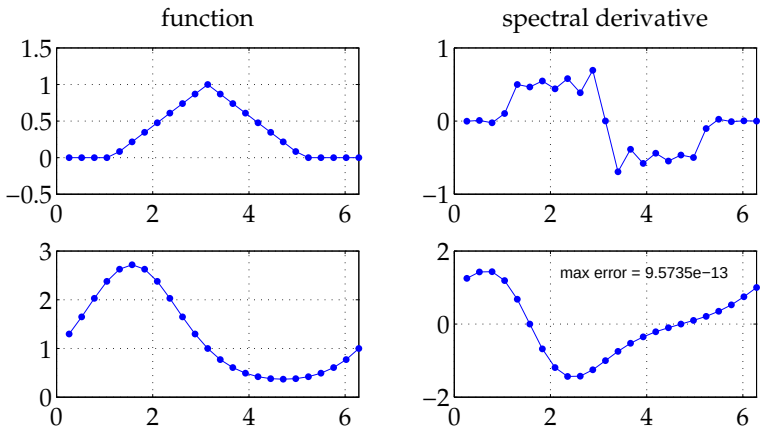

Figure 6.3: Spectral differentiation of two functions. Output 4 of Trefethen (2000).

```
  axis([0 2*pi -2 2]), grid on
  error = norm(w-vprime,inf);
  text(2.2,1.4,['max error = ' num2str(error)])
```

Exercise 6.3 *Explain the code in* `p5.m`

From this to solving simple linear PDEs is but a small step. Figure 6.4 shows the solution of the variable coefficient wave equation

$$u_t + \left[\frac{1}{5} + \sin^2(x-2)\right] u_x = 0. \tag{6.10}$$

The time-integration uses leapfrog. There's nothing wrong with leapfrog in general, but careful numerical analysis textbooks (e.g. Iserles 1996) would warn you to be very careful: there's no guarantee that coupling some spatial differentiation strategy with a time-stepping routine will not end in disaster. And these books are right. However, people tend to be more casual in physics and engineering, and the proof of the pudding is in the eating: it works here.

```
% p6.m - variable coefficient wave equation

% Grid, variable coefficient, and initial data:
  N = 128; h = 2*pi/N; x = h*(1:N); t = 0; dt = h/4;
  c = .2 + sin(x-1).^2;
  v = exp(-100*(x-1).^2); vold = exp(-100*(x-.2*dt-1).^2);

% Time-stepping by leap frog formula:
  tmax = 8; tplot = .15; clf, drawnow, set(gcf,'renderer','zbuffer')
  plotgap = round(tplot/dt); dt = tplot/plotgap;
```

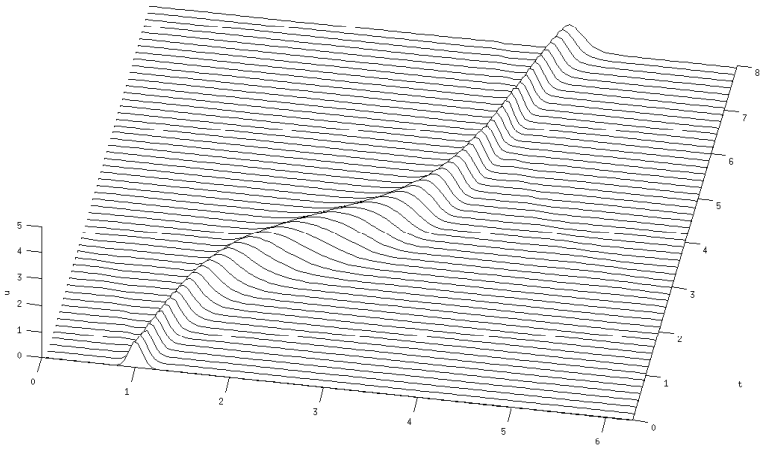

Figure 6.4: Spectral differentiation of two functions. Output 5 of Trefethen (2000).

```
nplots = round(tmax/tplot);
data = [v; zeros(nplots,N)]; tdata = t;
for i = 1:nplots
  for n = 1:plotgap
    t = t+dt;
    v_hat = fft(v);
    w_hat = 1i*[0:N/2-1 0 -N/2+1:-1] .* v_hat;
    w = real(ifft(w_hat));
    vnew = vold - 2*dt*c.*w; vold = v; v = vnew;
  end
  data(i+1,:) = v; tdata = [tdata; t];
end
waterfall(x,tdata,data), view(10,70), colormap(1e-6*[1 1 1]);
axis([0 2*pi 0 tmax 0 5]), ylabel t, zlabel u, grid off
```

6.5 Interpolation

Balmforth, Llewellyn Smith & Young (2001) study critical layers in a two-dimensional vortex. In the critical layer, the streamfunction takes the form

$$\psi \equiv -(y^2/2) + \varphi(\theta, t), \qquad \varphi(\theta, t) \equiv \hat{\varphi}(t)\mathrm{e}^{2\mathrm{i}\theta} + \hat{\varphi}^*(t)\mathrm{e}^{-2\mathrm{i}\theta}, \tag{6.11}$$

and the vorticity advection equation is

$$\zeta_t + \frac{\partial(\psi, \zeta + \beta y)}{\partial(\theta, y)} = \zeta_t + y\zeta_\theta + \varphi_\theta \zeta_y + \beta\varphi_\theta = 0. \tag{6.12}$$

The evolution of $\hat{\varphi}(t)$ is then obtained from

$$\mathrm{i}\hat{\varphi}_t = \chi + \langle \mathrm{e}^{-2\mathrm{i}\theta}\zeta\rangle, \qquad \text{where } \langle\cdots\rangle \text{ is} \qquad \langle f\rangle \equiv \fint \mathrm{d}y \oint \frac{\mathrm{d}\theta}{2\pi} f(\theta, y, t). \tag{6.13}$$

The principal value integral in (6.13) is necessary because $\zeta \propto y^{-1}$ as $|y| \to \infty$. However, henceforth we will omit the dash on the understanding that we take the principal values of all integrals to assure convergence.

When written in terms of $F = \zeta + \beta y$, the equation for ζ reduces to the advection-diffusion equation

$$F_t + yF_\theta + \varphi_\theta F_y = 0. \tag{6.14}$$

(F can be used instead of ζ interchangeably in the $\langle\rangle$ term since $\langle y\rangle = 0$.) Balmforth *et al.* (2001) solved the equations with an operator splitting scheme based on the algorithm developed by Cheng & Knorr (1976) for the Vlasov equation and subject to the boundary condition in y: $\zeta \to 0$ as $y \to \pm\infty$. In order to save computing the evolution of unneeded angular harmonics, they considered a periodicity in θ of π rather than 2π.

To avoid dealing with an infinite domain, the range is truncated in y and the vorticity equation is solved over the region $-L < y < L$. This requires boundary conditions at $y = \pm L$. To increase the accuracy of this truncation of the domain, they used the asymptotic solution for large y. The slow algebraic decay of the vorticity complicates the numerical scheme over the original Cheng & Knorr algorithm.

An integration step over the interval $[t, t+\tau]$ is divided into three stages:

1. Advect in θ for half a time step. This amounts to solving

   $$F_t + yF_\theta = 0 \tag{6.15}$$

   over $[t, t+\tau/2]$, leaving $\hat{\varphi}$ unchanged. The exact solution is $F^\alpha(y,\theta) = F(y, \theta - y\tau/2, t)$. This shift in θ is computed using Fourier interpolation.

2. Advect in y for a complete time step. This is done using a spline method and you are referred to the original paper.

3. Advect in θ for another half time step. This gives $F(y,\theta,t+\tau) = F^\beta(y, \theta - y\tau/2)$ as the new vorticity distribution. Except at the first and final time steps, this step can be combined with the first one and a shift in θ is only carried out once per time step.

The main practical limitation on the time step (other than $\tau \ll 1$ for the sake of precision) is that one cannot shift in y over more than one grid point. To avoid this situation, it is easiest to limit the time step to ensure that shifts over more than one grid point do not take place.

COEFFICIENT PROBLEMS AGAIN. The step that we focus on is the Fourier interpolation. The y-dependence is parametric, and what we care about is obtaining $f(x-a)$, knowing the coefficients X_k. Once again, we write $f(x-a)$ in terms of the coefficients:

$$f(x-a) = \frac{1}{N}\sum_{k=1}^{\infty} X_k \mathrm{e}^{\mathrm{i}(k-1)(x-a)}. \tag{6.16}$$

Hence the coefficients of the shifted function are $X_k \mathrm{e}^{-a\mathrm{i}(k-1)}$.

Exercise 6.4 *Write a Matlab program to carry out interpolation using the FFT. Examine the accuracy of this procedure as a function of the number of points by comparing the numerical answer to the exact shifted function.*

6.6 References

Publications:

- Balmforth, N. J., Llewellyn Smith, S. G. & Young, W. R. Disturbing vortices. *J. Fluid Mech.*, **426**, 95–133, 2001.
- Cheng, C. Z. & Knorr, G. The integration of the Vlasov equation in configuration space. *J. Comp. Phys.*, **22**, 330–351, 1976.
- Cooley, J. W & Tukey, J. W. An algorithm for the machine computation of the complex Fourier series. *Math. Comp.*, **19**, 297–301, 1965.
- Iserles, A. *A first course in the numerical analysis of differential equations.* Cambridge University Press, Cambridge, 1996.
- Lighthill, M. J. *An introduction to Fourier analysis and generalised functions.* Cambridge University Press, Cambridge, 1958.

Web sites:

- http://www.fftw.org

Chapter 7

Solving differential equations using transforms (04/22/08)

7.1 General procedure

This is an approach that is discussed in surprisingly few books[1]. It is equivalent to a standard transform in many cases, but gives some insight into the analytic properties of the solution that is obtained. We will also be able to obtain more than just one solution, which was a problem with the Fourier transform example of Chapter 5.

For a given differential equation for the function $f(z)$, consider the formal solution for the function:

$$f(z) = \int_C K(z,t)h(t)\,\mathrm{d}t. \tag{7.1}$$

We allow z and t to be complex, and C is a path in the complex plane to be discussed later. The function $K(z,t)$ is the kernel. Typical examples are the Fourier kernel, $\mathrm{e}^{\mathrm{i}zt}$, the Laplace kernel, e^{zt}, the Euler kernel, $(t-z)^{-\nu}$, the Mellin kernel, t^{-z}, and so on. The choice is dictated by the problem at hand.

We now substitute (7.1) into the governing equation for f. In general, we haven't made any progress, unless we can operate on the integrand using integration by parts to obtain a simpler equation for $g(z)$. In the Fourier case, we could remove derivatives and terms of the form $tf(t)$. The result is an integral in closed form, $[g(z,t)]_C$, that we annihilate by choosing C appropriately. There are two possibilities: C is a closed contour containing all the singularities of $K(z,t)h(t)$, or C is an open contour beginning and ending at zeros of $g(z,t)$. This is most easily illustrated using an example.

Example 7.1 [2] *We look at Hermite's equation*

$$f'' - 2zf + 2\nu f = 0. \tag{7.2}$$

Use the Laplace kernel and substitute into the ODE. Integrate by parts to find

$$\int_C \mathrm{e}^{zt}[(t^2 + 2\nu + 2)h(t) + 2th'(t)]\,\mathrm{d}t - 2[\mathrm{e}^{zt}th(t)]_C = 0. \tag{7.3}$$

[1] I've seen it in one or two, but I can't remember where now.

[2] From my Part II Lecture notes.

This equation can be satisfied by choosing $h(t)$ so that is satisfies the ODE in t inside the integral. Hence

$$h(t) = t^{-\nu-1}\mathrm{e}^{-t^2/4}. \tag{7.4}$$

Next we pick C so that $[t^{-\nu}\mathrm{e}^{zt-t^2/4}]_C = 0$. The position of the zeros of this express depends on the value of ν. The integrand vanishes at infinity in the sectors $|\arg t| < \pi/4$ and $|\pi - \arg t| < \pi/4$. It also vanishes at the origin if $\nu < 0$. The final expression for $f(z)$ is

$$f(z) = \int_C t^{-\nu-1}\mathrm{e}^{zt-t^2/4}\,\mathrm{d}t. \tag{7.5}$$

If ν is not an integer, there is a branch cut, that we can take along the negative real axis with the usual branch. If ν is a positive integer or 0, there is a pole at the origin around which we can integrate to get a non-trivial answer. If ν is not a negative integer or zero, we have one path from left to right, and from left back to itself looping around the cut. If $\nu < 0$ one can also take a contour starting at the origin. If ν is a negative integer, one can take paths starting from the origin and going to left or right. If ν is a positive integer, there is only path circling the origin and one from left to right. When ν is an integer, there is no cut.

This example works because the ODE for $h(t)$ is second order. This is usually the requirement for this technique to be useful. In some cases, one can find more than two paths. Then only two of them can give linearly independent solutions.

The representation (7.5) gives an explicit way to calculate Hermite functions. Let us work out the two linearly independent solutions for $\nu = 1/3$ (essentially an arbitrary value). We carry out the integral numerically; the exponential convergence means we can just truncate. This is true irrespective of the value of z, and this representation works just fine for complex z. Maple knows about Hermite functions, so we use the Maple function from within Matlab. The first integral is parallel to the real axis. The second loops around the branch cut and is proportional to the Maple Hermite function. In general, it is not terribly clear from Figure 7.1 which solutions have been computed and how one can distinguish them. A quick analysis of the irregular singular point at infinity, writing $f = \mathrm{e}^S$, shows that

$$S'' + S'^2 - 2zS' + 2\nu = 0. \tag{7.6}$$

The dominant balances for large $|z|$ are $S \sim z^2 - (\nu+1)\log z$ and $S \sim \nu \log z$, corresponding to

$$f \sim z^{-\nu-1}z^{-\nu-1}\mathrm{e}^{z^2}, \qquad f \sim z^{\nu}. \tag{7.7}$$

From Figure 7.1, the function H_1 looks like it will grow exponentially, while H_2 will not.

```
% 71.m Hermite functions SGLS 04/22/08

f1 = inline('exp(-(t+1i).^2/4+z*(t+1i)).*(t+1i).^(-nu-1)','t','nu','z');
f2 = inline('exp(-(1+1i*y).^2/4+z*(1+1i*y)).*(1+1i*y).^(-nu-1)*1i','y','nu','z');
nu = 1/3; z = -1.5:.05:1.5;
for j = 1:length(z)
  H1(j) = quad(@(t) f1(t,nu,z(j)),-20,20);
```

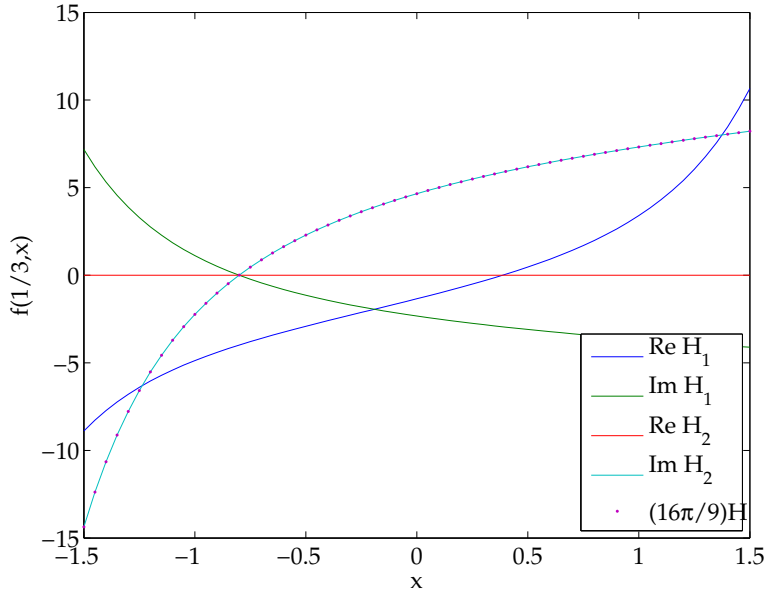

Figure 7.1: Hermite functions $f(1/3,(x)$ for x real. For comparison the function $(16\pi/9)H(x)$ from Matlab is plotted.

```
  H2(j) = quad(@(t) f1(t,nu,z(j)),-20,1);
  H2(j) = conj(H2(j)) - H2(j) + quad(@(t) f2(t,nu,z(j)),-1,1);
end
fH = pi*16/9*mfun('HermiteH',nu,z);
plot(z,real(H1),z,imag(H1),z,real(H2),z,imag(H2),z,fH,'.')
legend('Re H_1','Im H_1','Re H_2','Im H_2','(16\pi/9)H','Location','SouthEast')
xlabel('x'); ylabel('f(1/3,x)')
```

Exercise 7.1 *Compute the solutions for integer ν.*

Exercise 7.2 *Try larger values of z and complex z.*

7.2 The method of steepest descents

We would like to be able to understand the behavior of a function defined by an integral representation from the integral itself. The ISP analysis does not tell you which of the possible behaviors your particular integral will have. The results we find are not convergent series solutions, but asymptotic expansions, which are just fine for our purposes. Remember that given a series with term $a_n\delta_n(x)$, where the $\delta_n(x)$ are gauge functions, that is functions that are getting smaller in an asymptotic sense, we say that the series converges if the answer gets better, for all x, if we take more and more terms. For an

asymptotic expansion, the answers gets better if x gets larger say (or closer to x_0); ultimately the answer gets worse if we add more and more terms. Note that all convergent series are asymptotic.

Tip 5 *Some people would say you only ever need the first term in any asymptotic expansion. Others say calculate as many as you like.*

Many of the integral representations use the Laplace kernel, and a lot of machinery has been developed to analyze such integrals. Good introductions can be found in Bender & Orszag (1978) and Hinch (1991). A more general approach is given in Bleistein & Handelsman (1986) using Mellin transforms.

7.2.1 Watson's lemma

Consider the integral

$$F(x) = \int_0^M \mathrm{e}^{-xt} f(t), \mathrm{d}t, \tag{7.8}$$

where M may be infinite if the integral exists. Then close to the origin, f has the asymptotic behavior $f(t) \sim a_{r_1} t^{s_1} + a_{r_2} t^{s_2} + \cdots$, with $-1 < r_1 < r_2 < \cdots$, then

$$F(x) \sim a_{r_1} \Gamma(s_1 + 1) x^{-s_1 - 1} + a_{r_2} \Gamma(s_2 + 2) x^{-s_2 - 1} + \cdots, \qquad \text{as } x \to \infty, \tag{7.9}$$

an asymptotic expansion. Hence the behavior of $F(x)$, which looks like a Laplace transform, for large x depends only on the behavior of $f(t)$ for small t. This is usually the easy way of doing things. The harder way is to find the behavior for small x, knowing the behavior for large t.

Exercise 7.3 *Prove Watson's lemma for loop integrals:*

$$F(x) = \int_{-\infty}^{0+} \mathrm{e}^{xt} f(t) \, \mathrm{d}t \sim \sum \frac{2\pi \mathrm{i} a_{r_i} x^{-s_i - 1}}{\Gamma(-s_i)} \tag{7.10}$$

with the same notation as before for $f(t)$; the contour starts from minus infinity below the real axis, loops anticlockwise around the real axis and returns above the real axis to minus infinity. You will need to use Hankel's representation of the gamma function.

Definition 7.1 *The Gamma function is defined for Re $z > 0$ by*

$$\Gamma(z) = \int_0^\infty \mathrm{e}^{-t} t^{z-1} \, \mathrm{d}t. \tag{7.11}$$

From the recurrence relation $\Gamma(z+1) = z\Gamma(z)$, one can extend this definition to the whole complex plane excluding negative integers.

For a lovely little book on the Gamma function, see Artin (1964). The Gamma function is the natural extension of factorials to non-integer numbers.

Theorem 4 *(Bohr–Mellerup) The only continuous log-convex function satisfying the recurrence relation $f(x+1) = xf(x)$ and $f(1) = 1$ is the Gammma function.*

Exercise 7.4 *Prove the recurrence relation for the Gamma function. Discuss the nature of the singularities of the Gamma function: location, nature, residue...*

7.2.2 Laplace's method

Laplace's method, simply stated, is Watson's lemma associated to a change of variable. We are looking at integrals of the form

$$F(x) = \int_0^A \mathrm{e}^{-h(t)x} f(t)\,\mathrm{d}t \tag{7.12}$$

(A can be infinite). For large x, the exponential term wipes everything out, except where $h'(t)$ vanishes (if that happens). We can try the change of variable $u = h(t)$, with the chain rule yielding $\mathrm{d}t = \mathrm{d}u/h'[h^{-1}(u)]$. This change of variable will fail when $h'(t) = 0$: these are critical points.

Practically speaking then, we work out where h' vanishes, and consider these points and the endpoints. The contribution from t_0 goes like some power of t multiplied by $\mathrm{e}^{-h(t_0)x}$. so we only need to consider the smallest values of $h(t_0)$ (in magnitude). To obtain more terms, it is best to do the change of variable and use Watson's lemma. For the leading term, one can calculate it by hand (usually true for the first correction term too).

Watson's lemma provides the formal justification, but one can also treat this method informally using arguments on what is small and what is big: the crucial issue is the size of the region that contributes to the dominant behavior. This is especially useful if the variable x enters the integrand not only in the exponential. Note finally that the functions $f(t)$ can be slightly badly behaved: discontinuous, not infinitely differentiable and so on. Provided the integral exists, the method works.

Example 7.2 *Find the behavior of the Gamma function for large positive x. This is a classic. Start by writing*

$$\Gamma(x) = \int_0^\infty \mathrm{e}^{-t+x\log t} t^{-1}\,\mathrm{d}t. \tag{7.13}$$

This doesn't look right, but if we differentiate everything in the exponential, we get $-1 + xt^{-1}$ which vanishes at $t = x$. Hence the action happens when $t = O(x)$. One can continue informally, but we might as well make the change of variable $t = xu$. Then

$$\Gamma(x) = x^{x-1}\int_0^\infty \mathrm{e}^{x(-u+\log u)} u^{-1}\,\mathrm{d}u \sim x^{x-1}\int_0^\infty \mathrm{e}^{-x+(u-1)\times 0-\frac{1}{2}(u-1)^2}\,\mathrm{d}u. \tag{7.14}$$

The u^{-1} has been pulled out and replaced by its value at the critical point, 1. Take the e^{-x} out, do the Gaussian integral after changing the endpoints (exponential error) and ignore the dots. The results is

$$\Gamma(x) \sim \sqrt{2\pi} x^{x-1/2}\mathrm{e}^{-x}. \tag{7.15}$$

This is Stirling's approximation.

To get the whole series, one needs to carry out the change of variable mentioned, so that

$$\Gamma(x) = x^{x-1}\int_0^\infty \mathrm{e}^{-xv} u^{-1}\frac{\mathrm{d}u}{\mathrm{d}v}\,\mathrm{d}v. \tag{7.16}$$

There is a good way to do this: use Lagrange's inversion theorem (Bellman 2003):

Theorem 5 *Let $f(z)$ and $g(z)$ be analytic functions of z around the point $z = a$ with*

$$u = a + \epsilon g(u). \tag{7.17}$$

If $|\epsilon g(z)| < |z - a|$, (7.17) has one root, $u = u(a)$, in the neighborhood of $z = a$, and

$$f(u) = f(a) + \sum_{n=1}^{\infty} \frac{\epsilon^n}{n!} \left(\frac{\mathrm{d}}{\mathrm{d}a} \right)^{n-1} [f'(a)g(a)''n]. \tag{7.18}$$

Exercise 7.5 *Find the first three terms in the asymptotic expansion of the Gamma function. Use Lagrange's theorem.*

To see how well we are doing, let us compute the Gamma function for large x. The results show that for $x > 1$ we are doing a very good job both numerically and using the approximation (7.15).

```
% p72.m Gamma function asymptotics SGLS 04/21/08

f = inline('exp(-t).*t.^(x-1)','t','x');
x = logspace(0,1,21);
for j = 1:length(x)
  g(j) = quad(@(t) f(t,x(j)),0,50);
end
gas = sqrt(2*pi)*x.^(x-0.5).*exp(-x);
loglog(x,g,x,gamma(x),'.',x,abs((gas-gamma(x))./gas),x,1/12./x,'.')
xlabel('x'); ylabel('\Gamma(x) and approximations')
```

7.2.3 The method of stationary phase

A natural question is what happens with the Fourier kernel. A similar approach, due originally to Kelvin, carries through. It was first derived to obtain the free surface of surface gravity waves at large distances form the source and is still very useful for such problems. The method is slightly more delicate than Laplace's method, because the amplifying effect of the exponential is not exponential, so to speak, but depends on rapid cancellation of trigonometric functions. Usually all zeros have to be taken, and higher-order corrections are hard to obtain, because the contributions from the end points can be important. Once again, the method works for fairly general integrals of the form

$$G(x) = \int_a^b \mathrm{e}^{\mathrm{i}xt} f(t)\, ddt. \tag{7.19}$$

We won't go through this technique. It is important however, particularly since it can be viewed as underpinning ray theory. See Lighthill (1979) for an interesting account. If you really want to get into ray theory, see Bleistein (1984).

Project 3 *Develop an Eulerian ray solver using the ideas of Sethian, Osher and others that computes amplitudes for situations with background flows where the energy is no longer conserved. Warning: this is non-trivial.*

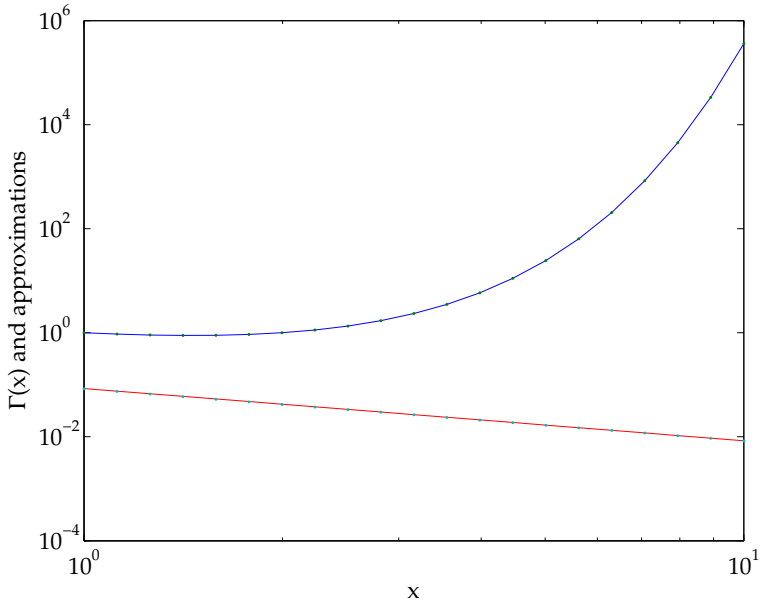

Figure 7.2: Euler's integral (7.11) (blue) and Matlab $\Gamma(x)$ (green dots). $x^{-x+1/2}e^x\Gamma(x) - 1$ (red) and $1/(12x)$ (blue dots).

7.2.4 Steepest descents

The method of steepest descents can be viewed as the generalization of the former two methods to the case of complex z and $h(t)$. Ironically, however, it requires the functions to be analytic which is more stringent than the previous conditions. This is not a problem when using integral transforms, and the method is very powerful. The method works on integral of the form

$$H(x) = \int_C e^{zh(t)} f(t)\, dt, \tag{7.20}$$

where C is some contour in the complex plane.

The basic idea is to deform the contour of integration to one along which we make the argument of the integral real, giving us the exponential increase and decrease of Laplace's method. Obviously one has to be careful about deforming contours through poles. There are critical points where $h'(t) = 0$ and one then finds a nice combination of subcontours that go from startpoint to endpoint. Here surface plots are extremely useful in understanding the topology of the complex plane.

Equations for these subcontours are curves of constant phase, since we want the real part to change. The correct curves are curves of steepest descent: the real part rises to a maximum at the critical point. One doesn't even need to compute the exact contours, because all that really matters is the topology of the real and imaginary parts. Near the critical points, all that matters are local approximations. There will usually be a question of which phase in a fractional power to use: this corresponds to the correct steepest descent and not ascent curve. The standard critical points have two curves going through them (saddle); higher order points will have more (monkey saddle).

Example 7.3 *I made up the following:*

$$F(z,a) = \int_C \mathrm{e}^{zs - s^3/3} s^a \, \mathrm{d}s. \tag{7.21}$$

If a is not an integer there is a branch cut along the negative real axis; we take the principal branch. We could rescale s to isolate the z term to recover the canonical form, but we shan't bother. There are critical points at $s = z^{1/2}$ (two of them) and the equations for the path of constant phase are ($s = \xi + \mathrm{i}\eta$, $z = x + \mathrm{i}y$)

$$Im\ (zs - s^3/3) = Im\ (2/3) z^{3/2}. \tag{7.22}$$

These curves and critical points are shown in Figure 7.3 for different values of z. Note that there are different curves going through the different critical points, except when z is real. The presence of the branch cut does not affect the curves on this figure, but means that when we deform contours, we may have to veer off the curves of constant phase to avoid going through a branch cut.

```
% p73.m Paths of constant phase SGLS 04/22/08

xi = -5:0.1:5; eta = xi; [xi,eta] = meshgrid(xi,eta); s = xi + 1i*eta;
subplot(2,2,1)
z = 1.2; h = z*s - 1/3*s.^3; sc = sqrt(z); ihz = imag(2/3*sc^3);
contour(xi,eta,imag(h),[ihz -ihz])
hold on; plot(real(sc)*[1 -1],imag(sc)*[1 -1],'k*'); hold off
xlabel('\xi'); ylabel('\eta');
subplot(2,2,2)
z = -0.8*i; h = z*s - 1/3*s.^3; sc = sqrt(z); ihz = imag(2/3*sc^3);
contour(xi,eta,imag(h),[ihz -ihz])
hold on; plot(real(sc)*[1 -1],imag(sc)*[1 -1],'k*'); hold off
xlabel('\xi'); ylabel('\eta');
subplot(2,2,3)
z = 2.4-0.3*i; h = z*s - 1/3*s.^3; sc = sqrt(z); ihz = imag(2/3*sc^3);
contour(xi,eta,imag(h),[ihz -ihz])
hold on; plot(real(sc)*[1 -1],imag(sc)*[1 -1],'k*'); hold off
xlabel('\xi'); ylabel('\eta');
subplot(2,2,4)
z = -0.4+2.2*i; h = z*s - 1/3*s.^3; sc = sqrt(z); ihz = imag(2/3*sc^3);
contour(xi,eta,imag(h),[ihz -ihz])
hold on; plot(real(sc)*[1 -1],imag(sc)*[1 -1],'k*'); hold off
xlabel('\xi'); ylabel('\eta');
```

Let us now discuss C. From the exponential, we see that C can start and end in $|\arg s| < \pi/3$ and the wedges obtained by rotating this sector by $2\pi/3$. We can manufacture two linearly independent solutions by specifying the start and end sectors (if both are the same, the integral is zero by Cauchy' theorem). Let us start in third quadrant

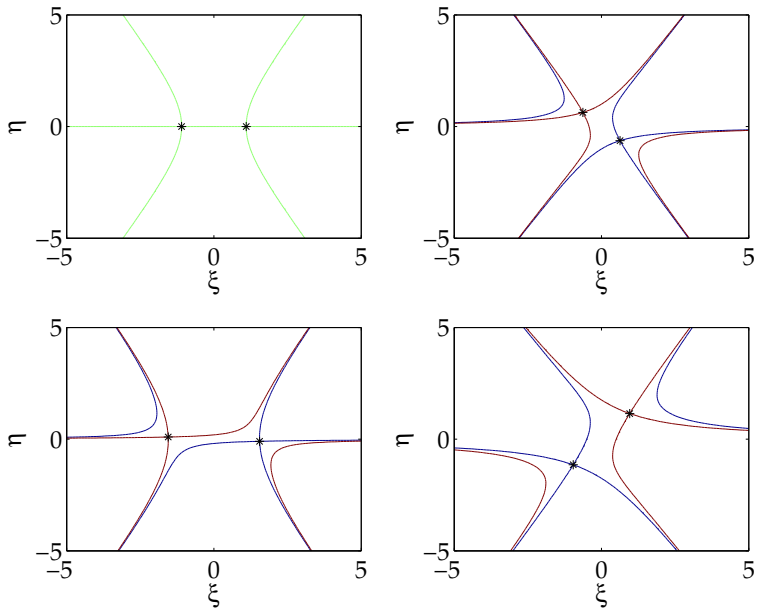

Figure 7.3: Curves of constant phase of $zs - s^3$. (a) $z = 1.2$; (b) $z = -0.8\mathrm{i}$; (c) $z = 2.4 - 0.3\mathrm{i}$; (d) $z = -0.4 + 2.2\mathrm{i}$.

and finish along the real axis. Then in (a) we can deform the contour to one that goes through both critical points $\pm z^{1/2}$. In (b) and (c) we go along the blue curve and only through the critical point in the fourth quadrant. In (d) we go through the critical point in the third quadrant out to infinity in the second quadrant, then back along the red curve through the other critical point out to toward the real axis.

We do case (b) so that z is negative imaginary. The critical point is $s_c = z^{1/2} = \sqrt{0.8}\mathrm{e}^{-\mathrm{i}\pi/4}$ (we are treating the square root here as a function, not as a multifunction). There are no problems with branch cuts when we deform on the blue contour. All that matters is the local behavior of the argument of the exponential near $z^{1/2}$, which is given by

$$h(s) = h(s_c) + (s - s_c)h'(s_c) + \tfrac{1}{2}(s - s_c)^2 h''(s_c) + \cdots = (2/3)z^{3/2} - 3z^{1/2}(s - z^{1/2})^2 + \cdots, \tag{7.23}$$

where we just keep the quadratic term. Doing the integrals gives

$$F(z, a) \sim \sqrt{\pi/3}\mathrm{e}^{2/3z^{3/2}} z^{a/2-1/2}. \tag{7.24}$$

Figure 7.4 compares this approximation to the numerically computed integral. Notice how the integral has been evaluated using two straight lines avoiding the origin.

```
% p72.m Steepest descent asymptotics SGLS 04/25/08

f1 = inline('exp(z*exp(-2*pi*i/3)*u-u.^3/3).*(u*exp(-2*pi*i/3)).^a*exp(-2*pi*i/3)','u',
f2 = inline('exp(z*(u-i)-(u-i).^3/3).*(u-i).^(a-0.5)','u','z','a');
```

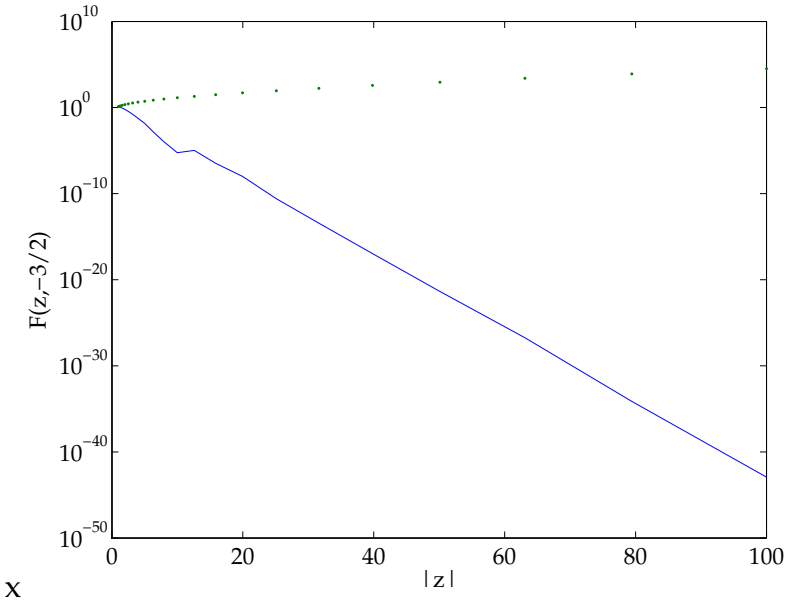

Figure 7.4: INCORRECT GRAPH?

```
a = 1.5; z = -i*logspace(0,2,21); zc = sqrt(z);
for j = 1:length(z)
  F(j) = quad(@(u) f1(u,z(j),a),10,2/sqrt(3));
  F(j) = F(j) + quad(@(u) f2(u,z(j),a),-1/sqrt(3),10);
end
Fas = sqrt(pi/3)*zc.^(a-0.5).*exp(2/3*zc.^1.5);
semilogy(abs(z),abs(F),abs(z),abs(Fas),'.')
xlabel('|z|'); ylabel('F(z,-3/2)')
```

Exercise 7.6 *Work out the behavior for the other three cases.*

Exercise 7.7 *Find the ODE satisfied by $F(z,a)$ from (7.21).*

Exercise 7.8 *Analyze the behavior of the Hankel function defined by*

$$H_\nu^{(1)} = \frac{1}{\pi \mathrm{i}} \int_{-\infty}^{\infty+\pi\mathrm{i}} \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}t \tag{7.25}$$

with $|\arg z| < \frac{1}{2}\pi$.

7.3 Stokes' phenomenon

Figure 7.3 shows that the topology of critical points and steepest descent curves changes as z changes. Hence the form of the asymptotic expansion can change in different regions

of the z-plane as one crosses lines called Stokes lines. This is called Stokes' phenomenon. It is real and important: it means that we do not have enough flexibility in our expansion functions to represent the asymptotic expansion of our function in a single formula. For nonlinear problems, one needs a more general approach.

Warning 10 *Watch out for notation. There are two sets of lines: Stokes and anti-Stokes, and authors use them to mean opposite things.*

In fact, the asymptotic expansion does not change discontinuously across as Stokes line. There is a transition region, first pointed out by Berry (1989). Some authors have used Stokes lines to understand complex ray theory (Chapman *et al.* 1999).

7.4 References

Publications:

- Artin, E. *The gamma function.* Holt, Rhinehart and Winston, New York, 1964.
- Bellman, R. *Perturbation techniques in mathematics, engineering & physics.* Dover, Mineola, 2003.
- Bender, C. M. & Orszag, S. A. *Advanced mathematical methods for scientists and engineers.* McGraw-Hill, ?, 1978.
- Berry, M. V. Uniform asymptotic smoothing of Stokes' discontinuities. *Proc. R. Soc. Lond. A* , **422**, 7–21, 1989.
- Bleistein, N. *Mathematical methods for wave phenomena.* Academic Press, Orlando, 1984.
- Bleistein, N. & Handelsman, R. A. *Asymptotic expansions of integrals.* Dover, New York, 1986.
- Chapman, S. J., Lawry, J. M. H., Ockendon, J. R. & Tew, R. H. On the theory of complex rays. *SIAM Rev.*, **41**, 417–509, 1999.
- Hinch, E. J. *Perturbation methods.* Cambridge University Press, Cambridge, 1991.
- Lighthill, M. J. *Waves in fluids.* Cambridge University Press, Cambridge, 1978.

Chapter 8

Laplace transforms (05/12/08)

8.1 Definition and inverse

We have already seen the definition of the Laplace transform:

$$F(p) = \int_0^\infty \mathrm{e}^{-px} f(x)\,\mathrm{d}x. \tag{8.1}$$

If $f(x) = O(\mathrm{e}^{ax})$ for large x, the integral converges for Re $p > a$ (and maybe for Re $p = a$) and the Laplace transform $F(p)$ is hence defined and analytic for Re $p > a$. The function defined by (8.1) can often be continued analytically outside this region. The function $f(x)$ also has to be civilized near the origin.

Example 8.1 *Consider the transform of x^ν. We have*

$$F(p) = \int_0^\infty \mathrm{e}^{-px} x^\nu\,\mathrm{d}x = p^{-\nu-1}\int_0^\infty \mathrm{e}^{-u} u^\nu\,\mathrm{d}u = \Gamma(\nu+1)p^{-\nu-1} \tag{8.2}$$

on making the change of variable $px = u$. From the behavior at the left endpoint, the transform is defined for $\nu > -1$. The analytic properties in terms of p from the definition show that $F(p)$ is defined for Re $p > 0$. For Re $p = 0$, there is convergence at infinity only if $\nu < -1$ which is not allowed. However, the result $\Gamma(\nu+1)p^{-\nu-1}$ can be extended over the whole complex plane, taking into account possible branch cuts and singularities at the origin. The function $F(p)$ of (8.2) exists for $\nu < -1$, but is not the transfor of a "normal" furction.

The definition (8.1) is formally that of the one-sided Laplace transform, for which only the behavior of $f(x)$ for $x > 0$ is of interest. This is built explicitly into the inversion formula

$$f(x) = \frac{1}{2\pi\mathrm{i}}\int_{c-\mathrm{i}\infty}^{c+\mathrm{i}\infty} F(p)\mathrm{e}^{px}\,\mathrm{d}p, \tag{8.3}$$

where c is a real number such that all the singularities of $F(p)$ have real part smaller than c. Then for $x < 0$, one can close the contour in the right half-plane and obtain zero. For poles, the $2\pi\mathrm{i}$ prefactor cancels nicely with that term in the residue theorem.

Warning 11 *If you are using the one-sided Laplace transform of $f(x)$, don't try to use negative values of x. The transform-inverse pair will always return 0, even if you thought $f(x)$ was non-zero for $x < 0$.*

The main point of the Laplace transform comes from the formula for the transform of a derivative, which nicely builds in the initial condition. The transform of $f'(x)$ is $pF(p) - f(0)$. Formulas for $f''(x)$ and so on can be obtained straightforwardly. Let us also remind ourselves of the convolution theorem.

Theorem 6 *(Convolution theorem). If $f(x)$ and $g(x)$ have Laplace transforms $F(p)$ and $G(p)$ respectively, then the inverse Laplace transform of $F(p)G(p)$ is the convolution $(f * g)(x)$, where the convolution is defined by*

$$(f * g)(x) = \int_0^x f(\xi)g(x - \xi)\,\mathrm{d}x. \tag{8.4}$$

We discussed inverting Fourier transforms using the calculus of residues and the same process works here, although if branch cuts happen, one has to be more careful. Let us do an example that comes from a PDE to learn something about Fourier series at the same time. We wish to solve the problem

$$c_t = c_{xx} \qquad \text{with } c_x = 0 \text{ on } x = 0 \text{ and } c_x = f(t) \text{ on } x = a. \tag{8.5}$$

This is a one-dimensional diffusion equation with imposed flux boundary conditions.

We start using a Fourier series approach. Write

$$c(t) = \sum_{n=0}^{\infty} c_n(t) \cos(n\pi x/a). \tag{8.6}$$

How do we know this? Well we don't really at this point, although clearly we are doing just fine with the left-hand boundary condition. If we keep sines and cosines, we will have trouble there. However, we definitely have non-zero c_x at the right-hand boundary, which doesn't seem to mesh with the behavior of the cosine function. We heed warning8 and multiply (8.5) by $\cos(m\pi x/a)$ and integrate. The rule is one can substitute the expansion as soon as it is no longer differentiated, and then use the orthogonality relation $\int_0^a \cos(m\pi x/a)\cos(n\pi x/a)\,\mathrm{d}x = a\epsilon_m^{-1}\delta_{mn}$, where ϵ_j is the Neumann symbol, defined to be 0 for $j = 0$ and 2 otherwise. One of the boundary terms does not cancel and the result is

$$\begin{aligned}
&\int_0^a \cos(m\pi x/a) \sum_{n=0}^{\infty} \dot{c}_n \cos(n\pi x/a)\,\mathrm{d}x = a\epsilon_m^{-1}\dot{c}_m \\
= \;& [\cos(m\pi x/a)c_x]_0^a - \int_0^a \cos(m\pi x/a) \sum_{n=0}^{\infty} (-n\pi/a)^2 \cos(n\pi x/a)\,\mathrm{d}x \\
= \;& (-1)^m f(t) + a\epsilon_m^{-1}(m\pi/a)^2 c_m.
\end{aligned} \tag{8.7}$$

We hence obtain the inhomogeneous equation

$$\dot{c}_m + (m\pi/a)^2 c_m = (-1)^m \frac{\epsilon_m}{a} f(t). \tag{8.8}$$

This can be solved using an integrating factor, yielding

$$c_m = (-1)^m \frac{\epsilon_m}{a} \int_0^t \mathrm{e}^{-(m\pi/a)^2(t-\tau)} f(\tau)\,\mathrm{d}\tau. \tag{8.9}$$

Note that this equation is in convolution form and can be viewed as the Green's function response convolved with the forcing.

The second approach is the Laplace transform (8.5) in time. Writing the resulting dependent variable as $C(p,x)$, we have

$$pC = C_{xx} \qquad \text{with } C_x = 0 \text{ on } x = 0 \text{ and } C_x = F(t) \text{ on } x = a. \tag{8.10}$$

The solution to this equation is

$$C = \frac{\cosh \sqrt{p}x}{\sqrt{p}\sinh \sqrt{p}a} F(p). \tag{8.11}$$

Now let us invert this transform. By the convolution theorem, the answer is $g * f$, where g is the inverse transform of the fraction in (8.11). This fraction does not in fact have a branch cut (check by expanding). It does have lots of singularities where $\sqrt{p}a = n\pi\mathrm{i}$ for integer n. These singularities are simple poles and are hence at $p = -(n\pi/a)^2$. This raises the ugly question of which square root to take in the root, but the answer is that it doesn't matter; we just taken one. The singularities with $n > 0$ are clearly simple poles; the singularities at $p = -(n\pi/a)^2$ can be shown to be simple poles. We have

$$g = \frac{1}{a} + \sum_{n=1}^{\infty} \frac{\cosh(n\pi\mathrm{i}x/a)}{\frac{1}{2}a\cosh(n\pi\mathrm{i})} \mathrm{e}^{-(n\pi/a)^2 t} = \frac{1}{a} + \sum_{n=1}^{\infty} (-1)^m \frac{2}{a} \cos(n\pi x/a) \mathrm{e}^{-(n\pi/a)^2 t}. \tag{8.12}$$

which matches with (8.9).

Figure 8.1 shows $c(t,x)$ for different values of t for the (arbitrary) forcing function $f(t) = E_1(t)$, where $E_1(t)$ is the exponential integral. The solution is computed by carrying out the convolution integral numerically and summing the Fourier series. The answer is acceptable, but the program is very slow. The behavior near $x = 1$ look suspiciously flat too; presumably more modes are needed to synthesize the non-zero slope there.

```
% p81: numerical integration for Laplace solution 04/25/2008 SGLS
f = inline('exp(-(m*pi/a)^2*(t-tau)).*expint(tau)','tau','t','m','a');
a = 1.4; MMAX = 20;
n = 100; h = 1./n;
x = a*h*(0:n); tt = 0.5:0.5:1;
c = zeros(size(x));
clf; hold on
for j = 1:length(tt)
  t = tt(j)
  for k = 1:length(x)
    c(k) =  quad(@(tau) f(tau,t,0,a),eps,t)/a;
```

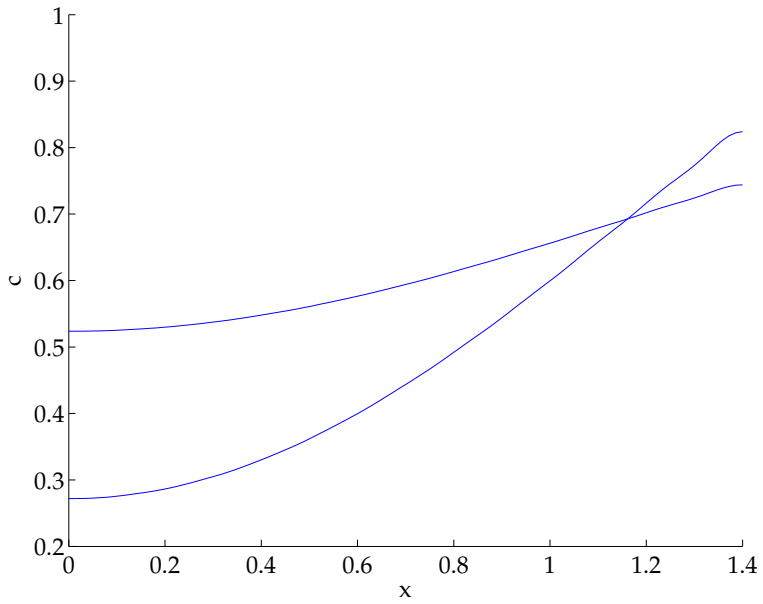

Figure 8.1: Solution $c(x,t)$ to (8.5) for $t = 0.5, 1$. The integral in (8.9) was computed numerically and terms up to $m = 20$ were retained in the Fourier series.

```
    for m = 1:MMAX
      c(k) = c(k) + quad(@(tau) f(tau,t,m,a),eps,t)*cos(m*pi*x(k)/a)*2/a*(-1)^m;
    end
  end
  plot(x,c); drawnow
end
hold off
xlabel('x'); ylabel('c')
```

Exercise 8.1 *Repeat this calculation with boundary condition $c = 0$ at $x = 0$.*

Project 4 *Pelloni claims that a recent development by Fokas can deal with more complicated boundary conditions for linear equations like the one above. Investigate, write up and consider examples with numerical calculations.*

8.2 Solving integral equation using the Laplace transform

Integral equations take a variety forms. Commonly encountered are (linear) Fredholm and Volterra equations of the first and second kind. In general, Volterra are easier than Fredholm and second kind is easier than first kind. A Fredholm equations of the second kind is

$$f(x) = \lambda \int_0^1 k(x,t) f(t)\,\mathrm{d}t + g(x) \tag{8.13}$$

and a Volterra equation of the first kind is

$$g(x) = \lambda \int_0^x k(x,t) f(t)\, \mathrm{d}t, \tag{8.14}$$

where $f(t)$ is an unknown function, to be found, in both cases. F1 and V2 are left to the reader. The function $k(x,t)$ is called the kernel. There is a close relation between integral equations and linear algebra (for a good treatment see Pipkin 1991). As a result, their numerical solution can be relatively easy. This is a good thing, because in most cases that is the only way of solving them. For a clear introduction to numerical techniques for solving Volterra equations, see Linz (1985).

A special kind of integral equation has a convolution kernel, i.e. one that depends only on the difference between the two variables so that $K(x,t) = K(x-t)$. For the Volterra equation above, we see immediately that

$$G(p) = K(p)F(p), \tag{8.15}$$

which can be inverted with derisory ease.

A particularly nice application is the Abel equation:

$$g(t) = \int_0^t \frac{f(\tau)}{\sqrt{\tau - t}}\, \mathrm{d}\tau. \tag{8.16}$$

Note that $g(0) = 0$ seems like a good idea, although it isn't quite obvious in some limiting cases. We need the Laplace transform of $t^{-1/2}$, which we found earlier to be $\Gamma(\frac{1}{2})p^{-1/2}$. Hence $F(p) = \pi - 1/2 p^{1/2} G(p)$. We invert this using the convolution theorem, but we should be careful of the fact that $p^{1/2}$, which does not decay at large real p, cannot be the transform of an ordinary function. Hence we write $p^{1/2}G(p) = p(p^{-1/2}G(p))$ and obtain

$$f(t) = \frac{1}{\pi}\frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \frac{g(\tau)}{\sqrt{\tau - t}}\, \mathrm{d}\tau. \tag{8.17}$$

An example is enlightening. First take $g(t) = 1$. Then

$$f(t) = \frac{1}{\pi}\frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \frac{1}{\sqrt{\tau - t}}\, \mathrm{d}\tau = \frac{2}{\pi\sqrt{t}}. \tag{8.18}$$

Note that $f(t)$ is singular as $t \to 0$, which might not have been obvious initially.

Exercise 8.2 *Solve the Abel equation for $f(t) = \log t$ (by hand) and $f(t) = \log(t)/(1+t^2)$ (numerically).*

8.3 Numerical inversion

8.3.1 Overview

As seen above, the solution to a fair number of problems can be written down as a Laplace Transform. Carrying out the transform by hand may be impossible or may be computationally expensive (e.g. returning an infinite sum). It would be useful to carry out the

inversion numerically. Algorithms to carry out inverse Laplace transforms have been around for a long time, and are reviewed in a number of places, including Davies (2002) and Duffy (2004).

The algorithms we consider start from the exact Laplace transform and invert that expression. The problem of going backward and forward from time variable to Laplace variable is rather different and difficult. The only real possibility is to adapt the FFT, but the result looks rather like an FFT with some strong filtering. For some reason, this algorithm has only been used practically in one or two areas, notably structural engineering (ref).

We will not review the different possible algorithms but only mention a few before concentrating on one in particular. One can try and calculate the Bromwich integral directly; this becomes an exercise in integration of what can be a very oscillator function (e.g. the Laplace transform of the harmless function $H(x)H(1-x)$ (the $H(x)$ is automatic really), namely $(1-\mathrm{e}^{-p})p^{-1}$, oscillates wildly going up toward $\pm\infty$). There are expansion approaches using Laguerre polynomials or Chebyshev polynomials.

The approach we detail goes back to Talbot (1979). In its simplest form, it is very straightforward. Deform the Bromwich contour to something that essentially looks like a parabola around the real axis going off into the left half-plane. Crucially, one needs to do some simple shifting and scaling transformations beforehand. Then replace the integral by the simplest trapezoid approximation.

The accuracy of this method can be investigated quite carefully. Talbot (1979) knew this well, but really only offered semi-empirical bounds. More recently, Trefethen, Weideman & Schmelzer (2006) and Weideman & Trefethen (2007) gave more sophisticated analyses relating the algorithm to approximation problems.

The algorithm is very easy to code. The major part of Talbot (1979) lies in a method to find the smallest number of points that will return a certain accuracy. For many applications, this is unnecessary because the algorithm is still blindingly fast with more points than strictly needed. However, even today, there are problems where it is worth using the smallest number of points that one can get away with. We will present only the simple version of the algorithm with a fixed number of points. Murli & Rizzardi (1990) wrote a publicly available version. I have not used it. I have my own matlab and Fortran version; the Matlab version is included below.

8.3.2 Talbot's method

We replace the Bromwich contour by an equivalent contour L going to $-\infty$ in the left halfplan. This replacement is permissible if L encloses all singularities of $F(p)$ and $|F(p)| \to 0$ uniformly in Re $p \leq \gamma_0$ as $|p| \to \infty$. This second condition may not hold for the original function, but can generally be made to hold by introducing a scaling parameter λ and shift parameter σ and considering $F(\lambda p + \sigma)$. The inversion formula becomes

$$f(x) = \frac{\lambda \mathrm{e}^{\sigma x}}{2\pi \mathrm{i}} \int_L F(\lambda p + \sigma)\mathrm{e}^{\lambda \sigma x}\,\mathrm{d}s \tag{8.19}$$

for $x > 0$.

We define the new contour by a change of variable $z = S_\nu(p)$. Many functions work, but Talbot emphasizes

$$S(z) = \frac{z}{1 - \mathrm{e}^{-z}} + \frac{\nu - 1}{2} z, \tag{8.20}$$

where ν is a positive parameter. This function maps the interval $(-2\pi\mathrm{i}, 2\pi\mathrm{i})$ on the imaginary axis to L. In terms of θ, we have

$$S_\nu(z) = \theta \cot\theta + \mathrm{i}\nu\theta, \qquad S'_\nu(z) = \frac{\nu}{2} + \frac{\mathrm{i}}{2}[\theta + \cot\theta(\cot\theta - 1)]. \tag{8.21}$$

Then we replace the integral by the trapezoidal rule and obtain

$$\tilde{f}(x) = \frac{\lambda \mathrm{e}^{\sigma x}}{n} {\sum_{k=-n+1}^{n-1}}' S'_\nu(z_k) \mathrm{e}^{\lambda S_\nu(z_k)} F(\lambda S_\nu(z_k) + \sigma). \tag{8.22}$$

If $f(x)$ is real, we can take the real part of the sum evaluated along the portion of S in the upper half-plane.

The freedom to choose ν is crucial in being able to go around all the singularities if they are complex. The algorithm is simple to code, once one has worked out λ, σ, ν and n. Doing so forms the bulk of Talbot's paper.

I have the sneaking suspicion there's a bug in `talbot.m` in the sum: have I taken $1/2$ the last term properly? The arguments of `talbot.m` are $F(p)$, `D`, the requested digits of accuracy, the location of the singularities, and their order (take 0 for a branch cut).

```
function [f,n] = talbot(fun,tt,D,sj,mult)

% Use the error analysis on page 108b
phat = max(real(sj));
sigma0 = max(0,phat);
spj = sj-sigma0;
if (~isreal(spj))
  ss = spj(imag(spj)~=0);
  r = imag(ss)./angle(ss);
  [rr,k] = find(max(r));
  sd = ss(k);
  qd = imag(sd);
  thetad = angle(sd);
  multd = mult(k);
else
  qd = 0;
  thetad = 0;
  multd = 0;
end
% 15 significant digits in Matlab
c = 15;
```

```
% test Talbot S.P. value
%c= 20;
% Talbot DP
%c = 27;

warning off
m = length(feval(fun,0));
warning on
f = zeros(length(tt),m);
for j = 1:length(tt)
  t = tt(j);

  % determine lambda, sigma, nu
  v = qd*t;
  omega = min(0.4*(c+1)+v/2,2*(c+1)/3);
  if (v<=omega*thetad/1.8)
    lambda = omega/t;
    sigma = sigma0;
    nu = 1;
  else
    kappa = 1.6 + 12/(v+25);
    phi = 1.05 + 1050/max(553,800-v);
    mu = (omega/t+sigma0-phat)/(kappa/phi-cot(phi));
    lambda = kappa*mu/phi;
    sigma = phat - mu*cot(phi);
    nu = qd/mu;
  end
  tau = lambda*t;
  a = (nu-1)/2;
  sigmap = sigma-sigma0;

  % determine n
  % n0
  Dd = D + min(2*multd-2,2) + floor(multd/4);
  if (v>omega*thetad/1.8 | multd==0 | isreal(spj))
    % Case 2 or Real singularities or not poles
    n0 = 0;
  else
    % Dominant singularity is a pole
    sstar = (sd-sigma0)/lambda;
    p = real(sstar);
    q = imag(sstar);
    r = abs(sstar);
    theta = angle(sstar);
    y = fzero(@(y) utal(y,sstar,q,r,theta),2*pi - 13/(5-2*p-q-0.45*exp(p)));
```

```
    % iteration blows up with negative im pole, so use bogus positive one
    uds = log((y-q)/r/sin(y-theta));
    n0 = floor((2.3*Dd+(phat-sigma0)*t)/uds)+1;
  end
  % n1
  e = (2.3*D+omega)/tau;
  if (e<=4.4)
    rho = (24.8-2.5*e)/(16+4.3*e);
  elseif (e<=10)
    rho = (129/e-4)/(50+3*e);
  else
    rho = (256/e+0.4)/(44+19*e);
  end
  n1 = floor(tau*(a+1/rho))+1;
  % n2
  gamma = sigmap/lambda;
  if (~isreal(spj))
    Dp = Dd;
  else
    Dp = max(D + min(2*mult-2,2) + floor(mult/4));
  end
  y = v/1000;
  eta = (1.09-0.92*y+0.8*y^2)*min(1.78,1.236+0.0064*(1.78)^Dp);
  n2 = floor(eta*nu*(2.3*Dp+omega)/(3+4*gamma+exp(-gamma)))+1;
  n = max(n0,max(n1,n2));
%  [lambda sigma nu n]
  % sum over both branches of paraboloid
  k=((-n+1):(n-1))';
  % vectorise sum over curve
  theta=k*pi/n;
  warning off
  alpha=theta.*cot(theta);
  alpha(n)=1;
  snu=alpha+nu*i*theta;
  s=lambda*snu+sigma;
  beta=theta+alpha.*(alpha-1)./theta;
  beta(n)=0;
  warning on
  % plug in actual function here
  ff=feval(fun,s);
%  keyboard
  temp=diag((nu+i*beta))*diag(exp(snu*tau))*ff;
  f(j,:)=f(j,:)+sum(temp);
  f(j,:)=f(j,:)*lambda*exp(sigma*t)/n/2;
end
```

```
function utal=utal(y,sstar,q,r,theta)

u = log((y-q)/r/sin(y-theta));
z = -u+i*y;
utal = real(sstar*(1-exp(-z))-z);
```

8.3.3 Numerical solution to model problem

For practical purposes, one needs a function $f(t)$ with a known Laplace transform if one is to solve (8.5) using a numerical inverse Laplace transform. For (8.5), $f(t)$ was carefully chosen to give $F(p) = p^{-1}\log(1+p)$. We use (8.11) in `talbot.m`; we don't bother passing all the singularities to the routine, just the one furthest to the right at the origin which is a pole of order two (an extra factor of p^{-1} from $F(p)$).

The solution obtained using this techniques is plotted in Figure 8.2. It is much smoother and better behaved. The program also runs much faster. The behavior at the right-hand edge of the interval looks right, since

```
>> expint(tt)

ans =

    1.2227    0.7024    0.4544    0.3106    0.2194
```

which matches with the observed slopes.

```
% p82: numerical inversion for Laplace solution 04/25/2008 SGLS
F = inline('1./p.*log(1+p).*cosh(sqrt(p)*x)./sqrt(p)./sinh(sqrt(p)*a)','p','x','a');
a = 1.4;
n = 100; h = 1./n;
x = a*h*(0:n); tt = 0.1:0.1:1;
c = zeros(length(x),length(tt));
clf; hold on
for k = 1:length(x)
  [f,n] = talbot(@(p) F(p,x(k),a),tt,6,0,1);
  c(k,:) = f;
end
plot(x,c); drawnow
hold off
xlabel('x'); ylabel('c')
```

Exercise 8.3 *Is the Matlab code actually correct or is there a factor of* $1/2$ *missing? Will this matter much?*

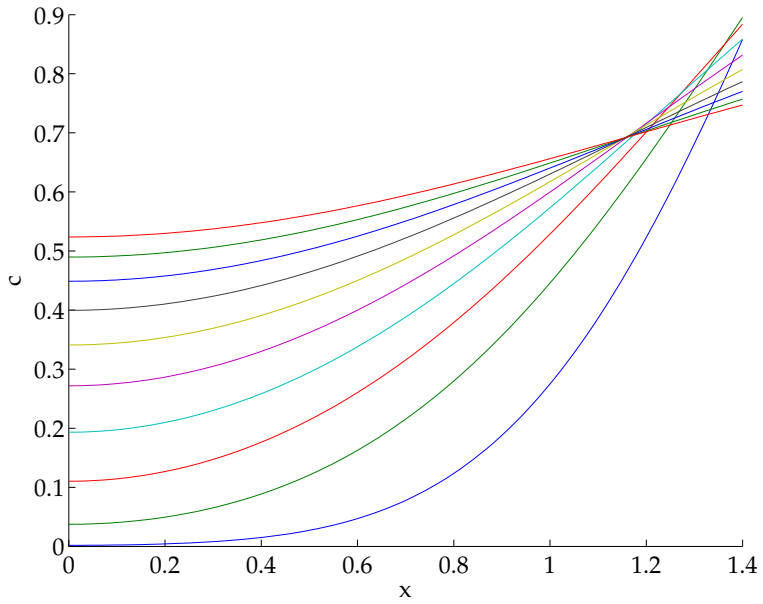

Figure 8.2: Solution $c(x,t)$ to (8.5) for $t = 0.1{:}0.1{:}1$ using Talbot's algorithm. Six digits of accuracy were requested.

8.4 Diffusion out of a well

This is a problem I have been looking at for Andrew Kummel in Chemistry and Biochemistry that is relevant for the design of a new platform called "Single Cell Hyper Analyzer for Secretants" (SiCHAS).

8.4.1 Formulation

We have a cylindrical well of depth d and radius a connected to a semi-infinite space. The well and half-space are filled with fluid in which a protein diffuses. The concentration of protein obeys the diffusion equation

$$c_t = D\nabla^2 c \tag{8.23}$$

with D an appropriate diffusion coefficient. All the boundaries are impermeable to the protein, so $c_n = 0$ on the boundaries. As initial condition, we have a uniform layer of depth l with concentration c_i at the bottom of the well.

This is an axisymmetric problem. Use cylindrical polar coordinates with $z = 0$ at the top of the well and z pointing down. Non-dimensionalize lengths by a, time by a^2/D, and define $h = d/a$. Then the diffusion equation becomes

$$c_t = \nabla^2 c = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial c}{\partial r}\right) + \frac{\partial^2 c}{\partial z^2}. \tag{8.24}$$

Boundary conditions are $c_r = 0$ on $r = 1$, $0 < z < h$, $c_z = 0$ on $r < 1$, $z = h$ and $r > 1$, $z = 0$, and decay as $(r^2 + z^2)^{1/2} \to \infty$, $z < 0$.

The diffusion equation is linear so the scaling of c is arbitrary. Take the initial condition

$$c_0(z) = \begin{cases} \epsilon^{-1} & \text{for } r < 1,\ h - \epsilon < z < h, \\ 0 & \text{elsewhere.} \end{cases} \tag{8.25}$$

This means that $l = a\epsilon$ and that the initial integrated c, i.e. the mass, is π.

We wish to find $\bar{c}$, the mass in the well, as a function of time. The parameters of the problem are h and ϵ. Suggested values in microns of 10 for a, 100 for d and 1 for l give $h = 10$ and $\epsilon = 0.1$. If $D = 10^{-6}$ cm^{2}s^{-1}, then the time scale is 1 second.

8.4.2 Solution

This is an extension of Miles (2002). Take the Laplace transform of (8.24):

$$pC - c_0(z) = \nabla^2 C. \tag{8.26}$$

Define the function $F = C_z$ on $z = 0$; this is the nondimensional flux out of the well. Expand it in a Fourier–Bessel series as

$$F(r) = \sum_n \frac{F_n}{I_n} J_0(k_n r), \qquad \text{with} \qquad F_n = \int_0^1 F(r) J_0(k_n r) r \, \mathrm{d}r. \tag{8.27}$$

The numbers k_n are roots of the equation $J_0'(k_n) = -J_1(k_n) = 0$. We have $k_0 = 0$. The numbers I_n are given by

$$I_n = \int_0^1 J_0^2(k_n r) r \, \mathrm{d}r = \frac{1}{2} J_0^2(k_n). \tag{8.28}$$

Note that $I_0 = 1/2$.

Now work inside the well. Separate variables by writing $C = \sum_n J_0(k_n r) Z_n(z)$, multiply (8.26) by $r J_0(k_m r)$ and integrate from 0 to 1. This gives

$$p I_m Z_m - c_0(z) \int_0^1 r J_0(k_m r) \, \mathrm{d}r = I_m Z_m'' - k_m^2 I_m Z_m. \tag{8.29}$$

This equation can be rewritten as

$$Z_n'' - (p + k_n^2) Z_n = -\frac{J_1(k_n)}{k_n I_n} c_0(z) = -\frac{\delta_{n0}}{\epsilon} H(z + \epsilon - h) \tag{8.30}$$

in $0 < z < h$. Define $\mu_n = (p + k_n^2)^{1/2}$. We find

$$Z = \begin{cases} A_n \cosh \mu_n (h - z) + \frac{\delta_{n0}}{p\epsilon} \cosh \sqrt{p}(h - \epsilon - z) & \text{for } 0 < z < h - \epsilon, \\ A_n \cosh \mu_n (h - z) + \frac{\delta_{n0}}{p\epsilon} & \text{for } h - \epsilon < z < h. \end{cases} \tag{8.31}$$

We can now write the solution for C on $z = 0$ as

$$C = \sum_{n \geq 1} J_0(k_n) A_n \cosh \mu_n h + A_0 \cosh \sqrt{p} h + \frac{1}{p\epsilon} \cosh \sqrt{p}(h - \epsilon). \tag{8.32}$$

In the reservoir, we have

$$C = \int_0^1 F(\eta)\eta \,\mathrm{d}\eta \int_0^\infty J_0(kr)J_0(k\eta)\mathrm{e}^{\mu z}\frac{k}{\mu}\,\mathrm{d}k \tag{8.33}$$

where $\mu = (p + k^2)^{1/2}$. At $z = 0$, this is

$$C = \int_0^1 F(\eta)\eta \,\mathrm{d}\eta \int_0^\infty J_0(kr)J_0(k\eta)\frac{k}{\mu}\,\mathrm{d}k = \sum_n \frac{F_n}{I_n}\int_0^\infty Q_n(k)J_0(kr)\frac{k}{\mu}\,\mathrm{d}k, \tag{8.34}$$

where

$$Q_n(k) = \int_0^1 J_0(k_n r)J_0(kr)r\,\mathrm{d}r = \frac{k}{k^2 - k_n^2}J_0(k_n)J_1(k). \tag{8.35}$$

Match (8.32) at $z = 0$ and (8.34) and use orthogonality:

$$\tfrac{1}{2}\left[A_0\cosh\sqrt{p}h + \frac{1}{p\epsilon}\cosh\sqrt{p}(h-\epsilon)\right] = \sum_n \frac{F_n}{I_n}\int_0^\infty Q_n(k)Q_0(k)\frac{k}{\mu}\,\mathrm{d}k, \tag{8.36}$$

$$A_m I_m \cosh\mu_m h = \sum_n \frac{F_n}{I_n}\int_0^\infty Q_n(k)Q_m(k)\frac{k}{\mu}\,\mathrm{d}k \quad (m \geq 1). \tag{8.37}$$

We can relate the coefficients A_n to F_n using

$$A_0\sqrt{p}\sinh\sqrt{p}h + \frac{\sqrt{p}}{p\epsilon}\sinh\sqrt{p}(h-\epsilon) = -2F_0, \tag{8.38}$$

$$A_m\mu_m I_m \sinh\mu_m h = -F_m \quad (m \geq 1). \tag{8.39}$$

We have obtained an infinite set of coupled linear equations for the F_n:

$$F_0\frac{\coth\sqrt{p}h}{\sqrt{p}} + \sum_n \frac{F_n}{I_n}\int_0^\infty Q_0(k)Q_n(k)\frac{k}{\mu}\,\mathrm{d}k = \frac{\sinh\sqrt{p}\epsilon)}{2p\epsilon\sinh\sqrt{p}h}, \tag{8.40}$$

$$\frac{F_m}{\mu_m}\coth\mu_m h + \sum_n \frac{F_n}{I_n}\int_0^\infty Q_m(k)Q_n(k)\frac{k}{\mu}\,\mathrm{d}k = 0 \quad (m \geq 1). \tag{8.41}$$

The quantity we seek is $\bar{c} = \int_W c\,\mathrm{d}V$, the mass of protein inside the well. At the initial instant, $\bar{c} = \pi$. We can obtain an equation for $\bar{c}$ by integrating (8.26) over the well, giving

$$p\bar{C} - \int_W c_0(z)\,\mathrm{d}V = \int_{S_W}\nabla C\cdot\mathbf{n}\,\mathrm{d}S = -2\pi\int_0^1 Fr\,\mathrm{d}r = -2\pi F_0. \tag{8.42}$$

Physically this equation just says that the change in mass is due to the flux of matter out of the top of the well. We can transform this into

$$\bar{C} = p^{-1}\pi(1 - 2F_0). \tag{8.43}$$

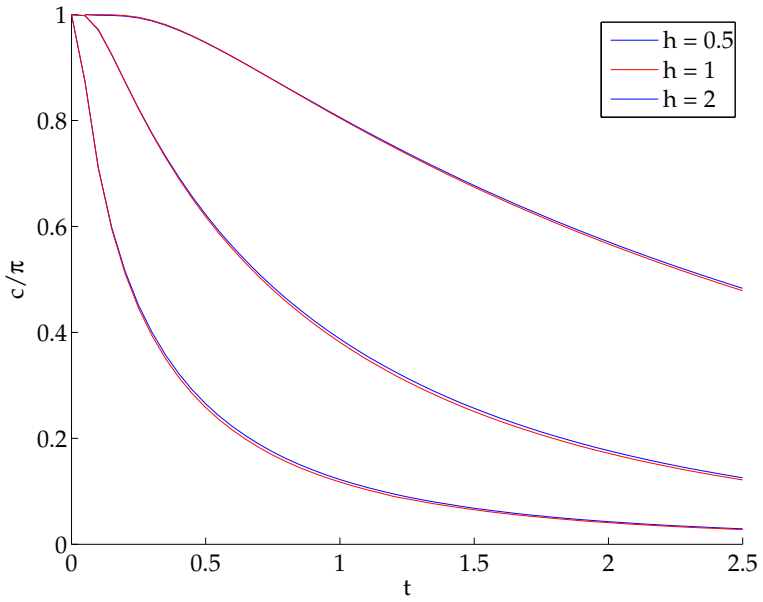

Figure 8.3: Concentration in well $\bar{c}/\pi$. The blue curves correspond to $M = 0$, the red to $M = 5$. The differences are very small.

8.4.3 Numerical solution

We compute the inverse Laplace transform of $\bar{C}$ using Talbot's algorithm. The function $F_0(p)$ is calculated by truncating the set of linear equations of (8.40–8.41) at finite order. The Fortran code is relegated to the Appendix.

Figure 8.3 shows the evolution of $\bar{c}/\pi$ as a function of time for different values of h, while ϵ is fixed to be 0.1. We see that the depth of the well affects how fast protein escapes from it

```
% p83.m Concentration in well for SiCHAS 04/24/08 SGLS
hh = [0.5 1 2];
e = 0.1;
clf; hold on
for j = 1:length(hh)
  [f,c,t] = Kdo(hh(j),e,0,2.5,.05,6);
  t = [0 ; t]; c = [pi ; c];  plot(t,c/pi,'b'); drawnow
  [f,c,t] = Kdo(hh(j),e,5,2.5,.05,6);
  t = [0 ; t]; c = [pi ; c];  plot(t,c/pi,'r'); drawnow
end
hold off
axis([0 2.5 0 1]); legend('h = 0.5','h = 1','h = 2')
xlabel('t'); ylabel('c/\pi')
```

8.5 References

Publications:

- Davies, B. *Integral transforms and their applications.* Springer, New York, 2002.
- Duffy, D. G. *Transform methods for solving partial differential equations.* Chapman & Hall/CRC, Boca Raton, 2004.
- Linz, P. *Analytical and numerical methods for Volterra equations.* SIAM, Philadelphia, 1985. SIAM, 1985
- Miles, J. W. 2002. Gravity waves in a circular well. *J. Fluid Mech.*, **460**, 177–180.
- Murli, M. & Rizzardi, M. Talbot's method for the Laplace inversion problem. *ACM Trans. Math. Software*, **16**, 158–168, 1990.
- Pipkin, A. C. *A course on integral equations.* Springer, New York, 1991.
- Talbot, A. The accurate numerical inversion of Laplace transforms. *J. Inst. Math. Appl.*, **23**, 97–120, 1979.
- Trefethen, L. N., Weideman, J. A. C. & Schmelzer, T. Talbot quadratures and rational approximations. *BIT Num. Math.*, **46**, 65-3-670, 2006.
- Weideman, J. A. C. & Trefethen, L. N. Parabolic and hyperbolic contours for computing the Bromwich integral. *Math. Comp.*, **76**, 1341–1356, 2007.

Chapter 9

Multiple transforms: Cagniard–de Hoop method (04/29/08)

9.1 Multiple integrals

Since we have not defined integrals formally, we have no trouble not defining multiple integrals and just using them. Essentially everything carries through and one can brazenly change orders of integration and so on in the finite case. However there are a few little surprises in store with infinite integrals and especially principal value integrals.

Let us remind ourselves how to change the order of integration and how to change variables in multiple integrals. Re-expressing the area/volume/hypersurface/hypervolume in the integral is merely a matter of reparameterizing the governing equations defining the boundary. To change variables, one uses the formula

$$\mathrm{d}x_1\,\mathrm{d}x_2\ldots\,\mathrm{d}x_{n-1}\,\mathrm{d}x_n = \left|\frac{\partial(x_1,x_2,\ldots,x_{n-1},x_n)}{\partial(y_1,y_2,\ldots,y_{n-1},y_n)}\right|\,\mathrm{d}y_1\,\mathrm{d}y_2\ldots\,\mathrm{d}y_{n-1}\,\mathrm{d}y_n. \tag{9.1}$$

The big fraction is the determinant of the $n \times n$ matrix made up of the partial derivatives.

Example 9.1 *Spherical polar coordinates. We have Cartesian components x, y, z and spherical polar coordinates r, θ, ϕ. Note that for the latter coordinates, in that order, to form a right-handed basis, θ must be co-latitude (0 at the North pole, π at the South pole). The relationship between the two coordinate systems is*

$$x = r\sin\theta\cos\phi, \qquad y = r\sin\theta\sin\phi, \qquad z = r\cos\theta. \tag{9.2}$$

Then

$$\left|\frac{\partial(x,y,z)}{\partial(r,\theta,\phi)}\right| = \begin{vmatrix} \sin\theta\cos\phi & r\cos\theta\cos\phi & -r\sin\theta\sin\phi \\ \sin\theta\sin\phi & r\cos\theta\sin\phi & r\sin\theta\cos\phi \\ \cos\theta & -r\sin\theta & 0 \end{vmatrix} = r^2\sin\theta \tag{9.3}$$

as you already know. The area of a sphere is

$$\int_0^{\pi}\int_0^{2\pi} r^2\sin\theta\,\mathrm{d}\theta\,\mathrm{d}\phi = 2\pi r^2[-\cos\theta]_0^{\pi} = 4\pi r^2. \tag{9.4}$$

Theorem 7 *(Poincaré–Bertrand) Let $f(x,t)$ be Hölder-continuous with respect to both its variables. Then*

$$\fint_a^b \frac{1}{x-y}\left[\fint_a^b \frac{f(x,t)}{t-x}\,\mathrm{d}t\right]\,\mathrm{d}x = -\pi^2 f(y,y) + \fint_a^b \left[\fint_a^b \frac{f(x,t)}{(x-y)(t-x)}\,\mathrm{d}x\right]\,\mathrm{d}t. \tag{9.5}$$

This theorem extends to complex integrals with no trouble.

Exercise 9.1 *Track down the original references and construct a clear proof. Discuss applications. See* `http://eom.springer.de/p/p072980.htm` *for references.*

Let us test this formula numerically. Calculating principal value integrals is not entirely straightforward. Quadpack has the routine `dqawc`. In Matlab, one can consider either writing a rule from scratch, deforming the contour away from the pole (which will require inserting the πi residue contribution by hand), or subtracting out the singular part, as in

$$\begin{aligned}\fint_a^b \frac{f(x,t)}{t-x}\,\mathrm{d}t &= \fint_a^b \frac{f(x,t)-f(x,x)}{t-x}\,\mathrm{d}t + f(x,x)\fint_a^b \frac{1}{t-x} \\ &= \fint_a^b \frac{f(x,t)-f(x,x)}{t-x}\,\mathrm{d}t + f(x,x)\log\left|\frac{b-x}{a-x}\right|.\end{aligned} \tag{9.6}$$

This will only work if f is well-behaved near $t = x$. It is also more involved for the right-hand side of (9.5). The result is a rather crude program written in Fortran which could almost certainly be made much more elegant.

```
c     p91.f Poincare-Bertrand 04/29/08 SGLS
c     g77 -O -o p91 p91.f -lslatec -Wl,-framework -Wl,vecLib
      program p91

      implicit double precision (a-h,p-z)
      parameter (epsabs=1d-6,epsrel=0d0)
      parameter (limit=10000,lenw=limit*4)
      dimension iwork(limit),work(lenw)
      common a,b,tc,xc,yc
      external Flo,Fro

      call xsetf(0)
      a = 0d0
      b = 1d0
      pi = 4d0*atan(1d0)
      do j = 0,10
         yc = j*0.1d0
         call dqawc(Flo,a,b,yc,epsabs,epsrel,r1,abserr1,
     $           neval1,ier1,limit,lenw,last,iwork,work)
         write (6,*) 'lhs',r1,ier1,neval1,abserr1
```

```
      call dqawc(Fro,a,b,yc,epsabs,epsrel,r2,abserr2,
     $           neval2,ier2,limit,lenw,last,iwork,work)
      write (6,*) 'rhs',r2,ier2,neval2,abserr2
      write (6,*) 'PB',yc,r1,-pi*pi*f(yc,yc)+r2
    enddo
    end


    double precision function f(x,t)

    implicit double precision (a-h,p-z)

    f = x*x + t*t + log(1d0 + (1d0+x*x)/(1d0+t*t))
    end


    double precision function fl(t)

    implicit double precision (a-h,p-z)
    common a,b,tc,xc,yc

    fl = f(xc,t)
    end


    double precision function fr(x)

    implicit double precision (a-h,p-z)
    common a,b,tc,xc,yc

    fr = f(x,tc)
    end


    double precision function Flo(xin)

    implicit double precision (a-h,p-z)
    parameter (epsabs=1d-6,epsrel=0d0)
    parameter (limit=10000,lenw=limit*4)
    dimension iwork(limit),work(lenw)
    common a,b,tc,xc,yc
    external fl

    xc = xin
    if (a.eq.xc.or.b.eq.xc) then
```

```
         Flo = 0d0
         return
      endif
      call dqawc(fl,a,b,xc,epsabs,epsrel,result,abserr,
     $     neval,ier,limit,lenw,last,iwork,work)
      Flo = result
      end


      double precision function Fro(tin)

      implicit double precision (a-h,p-z)
      parameter (epsabs=1d-6,epsrel=0d0)
      parameter (limit=10000,lenw=limit*4)
      dimension iwork(limit),work(lenw)
      common a,b,tc,xc,yc
      external fr

      tc = tin
      if (a.eq.tc.or.b.eq.tc) then
         Fro = 0d0
         return
      endif
      call dqawc(fr,a,b,yc,epsabs,epsrel,r1,abserr,
     $     neval,ier,limit,lenw,last,iwork,work)
      call dqawc(fr,a,b,tc,epsabs,epsrel,r2,abserr,
     $     neval,ier,limit,lenw,last,iwork,work)
      Fro = r1 - r2
      end
```

```
> p91
 lhs  0. 6 0  0.
 rhs  0. 6 0  0.
 PB  0.  0. -6.84108846
 lhs -1.34216557 0 1425  9.8550233E-07
 rhs  5.69631497 0 1425  9.85498413E-07
 PB  0.1 -1.34216557 -1.34216558
 lhs -2.82517836 0 1385  9.80385068E-07
 rhs  4.80547845 0 1385  9.80383278E-07
 PB  0.2 -2.82517836 -2.82517837
 lhs -3.91783477 0 1345  9.83306328E-07
 rhs  4.69978248 0 1345  9.83305798E-07
 PB  0.3 -3.91783477 -3.91783477
 lhs -5.01960816 0 1335  8.24348636E-07
 rhs  4.97975371 0 1335  8.24348424E-07
```

```
PB  0.4 -5.01960816 -5.01960816
lhs -6.20534062 0 1335  8.32939443E-07
rhs  5.57055004 0 1335  8.32938877E-07
PB  0.5 -6.20534062 -6.20534062
lhs -7.44370949 0 1365  8.629954E-07
rhs  6.50349414 0 1365  8.62995162E-07
PB  0.6 -7.44370949 -7.4437095
lhs -8.59730377 0 1375  8.19080127E-07
rhs  7.915997 0 1375  8.19078694E-07
PB  0.7 -8.59730377 -8.59730378
lhs -9.28631154 0 1415  8.06449584E-07
rhs  10.1878706 0 1415  8.06445658E-07
PB  0.8 -9.28631154 -9.28631154
lhs -8.11656248 0 1455  8.25652154E-07
rhs  14.7132851 0 1455  8.2564037E-07
PB  0.9 -8.11656248 -8.11656248
lhs  0. 6 0  0.
rhs  0. 6 0  0.
PB  1.  0. -26.5802973
```

Ioakimidis (1985) considers constructs interesting quadrature rules for double integrals from the Poincaré–Bertrand formula, claiming s they can be used to solve Cauchy-type singular integral equations but does not consider any applications.

Exercise 9.2 *Find an application for Ioakimidis (1985). Maybe more appropriate for Chapter 13. This is slightly beyond the level of detail we have gone into for quadrature so far.*

9.2 Multiple transforms

Since integral transforms are essentially integrals, we can apply transforms repeatedly and obtain multiple transforms. Multiple Fourier transforms, one for each spatial dimension, are common. Throwing in a Laplace transform for the temporal dependence is also common. For example, studying the linearized stability of a system using the Laplace-Fourier transform lies behind ideas in convective and absolute instability, as we shall see later. The Poincaré–Bertrand theorem warns us to be careful, however, with Hilbert transforms.

Example 9.2 *The Hankel transform of order* 0 *can be derived from the two-dimensional Fourier transform. Let the function $f(x,y)$ depend only in fact on r, the polar coordinate. Transform to $F(k,l)$ and back; F will be a function only of κ, the polar coordinate in the (k,l) plane. Then*

$$F(k,l) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\mathrm{e}^{-\mathrm{i}(kx+ly)}\,\mathrm{d}x\,\mathrm{d}y = \int_0^{\infty} f(r)\mathrm{e}^{-\mathrm{i}\kappa r\cos(\theta-\phi)}\,\mathrm{d}\theta r\,\mathrm{d}r = \pi\int_0^{\infty} f(r)J_0(\kappa r)r\,\mathrm{d}r. \tag{9.7}$$

The angular coordinates are θ in the (x, y) plane and ϕ in the (k, l) plane, but cancel. Doing the same for the inverse transform and tidying up gives the transform pair

$$g(\kappa) = \int_0^\infty f(r) J_0(\kappa r) r \, \mathrm{d}r, \qquad f(r) = \int_0^\infty g(\kappa) J_0(\kappa r) \kappa \, \mathrm{d}\kappa. \tag{9.8}$$

Numerically the same issues about computing numerical transforms exist: one has to find a good quadrature rule and so on. There is a serious new problem though: how to deal with singularities that depend on the previous integral transform. This will be addressed in Chapter 13. We turn now to a situation where one uses the double integral to our advantage.

9.3 Cagniard–de Hoop method

9.3.1 Basic idea

Cagniard originally studied seismological problems and was the first to consider initial-value problems. His analysis was later extended by de Hoop. Note that Cagniard's original work (Cagniard 1939) mentions the axisymmetric case, not just the two-dimensional case. We will present the method, then go through two worked examples.

Consider a Laplace–Fourier transform $f^{\mathrm{LF}}(p, k)$. Carry out the inverse Fourier transform to get

$$f^{\mathrm{L}} = \frac{1}{2\pi} \int_{-\infty}^{\infty} f^{\mathrm{LF}}(p, k) \mathrm{e}^{\mathrm{i}kx} \, \mathrm{d}k. \tag{9.9}$$

The game now is to turn this into something that looks like a Laplace transform. This is accomplished by deforming the contour in the complex plane to get a curve parameterized by a new real variable t. Then we're done and we can read off the original function.

9.3.2 An impact problem

Adda-Bedia & Llewellyn Smith (2006) consider the LF transform:

$$\varphi^{\mathrm{LF}}(k, z, s) = -\frac{{u_C}^{\mathrm{LF}}(k, s)}{s\sqrt{k^2+1}} \mathrm{e}^{-sz\sqrt{k^2+1}}, \tag{9.10}$$

where the function $u_C(k, s)$ comes from transforming

$$u_C(x, t) = (t - x^2/2) H(t - x^2/2), \tag{9.11}$$

where H is the Heaviside step function.

We define branch cuts in the k-plane extending from $\pm\mathrm{i}$ to $\pm\mathrm{i}\infty$, with $\sqrt{k^2+1}$ positive and real for k on the real axis. Hence the transformed pressure, which is what is physically interesting, is

$$p^{\mathrm{LF}}(k, z, s) = -s^2 \varphi^{\mathrm{LF}}(k, z, s) = \sqrt{2\pi} \frac{s^{-3/2}}{\sqrt{k^2+1}} \mathrm{e}^{-sk^2/2 - sz\sqrt{k^2+1}}. \tag{9.12}$$

Exercise 9.3 *Show that*

$$u_C{}^{\mathrm{LF}} = \sqrt{2\pi} s^{-5/2} \mathrm{e}^{-sk^2/2}. \tag{9.13}$$

We now follow the Cagniard–de Hoop procedure. The Laplace transform of the pressure on the boundary $z = 0$ is given by

$$p^{\mathrm{L}}(x, 0, s) = \frac{s^{-1/2}}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{\mathrm{e}^{-s(k^2/2 + \mathrm{i}kx)}}{\sqrt{k^2 + 1}} \, \mathrm{d}k. \tag{9.14}$$

Note that we have rescaled the integration variable k, assuming that s is positive and real, or that the integration limits can be moved back to the real axis if s is complex. This enables us to write

$$p^{\mathrm{L}}(x, 0, s) = \pi^{-1/2} s^{-1/2} g^{\mathrm{L}}(x, s), \tag{9.15}$$

so as to isolate the x-dependence in the integral into the exponential alone.

We now define a new variable τ so that the integral in (9.14) becomes a Laplace integral. Hence we set

$$\tau = \tfrac{1}{2} k^2 + \mathrm{i}kx. \tag{9.16}$$

There are two regimes that we label supersonic and subsonic.

Supersonic region

When $|x| < \sqrt{2\tau}$, the relation (9.16) may be inverted to give

$$k(x, \tau) = -\mathrm{i}x \pm \sqrt{2\tau - x^2}. \tag{9.17}$$

We now move the Fourier contour to $\mathrm{Im}\, k = -x$, initially with $|x| < 1$ so as not to involve the branch cut. We then change the variable of integration from k to τ. The resulting horizontal contour is described as t takes the values $\frac{1}{2}x^2$ to ∞ (there are two branches of k corresponding to the two possible signs of the square root: k_+ in the right half-plane and k_- in the left half-plane). Hence we may write

$$g^{\mathrm{L}}(x, s) = \frac{1}{\sqrt{2}} \int_{\infty}^{x^2/2} \frac{\mathrm{e}^{-s\tau}}{\sqrt{k_-^2 + 1}} \frac{\mathrm{d}k_-}{\mathrm{d}\tau} \, \mathrm{d}\tau + \frac{1}{\sqrt{2}} \int_{x^2/2}^{\infty} \frac{\mathrm{e}^{-s\tau}}{\sqrt{k_+^2 + 1}} \frac{\mathrm{d}k_+}{\mathrm{d}\tau} \, \mathrm{d}\tau. \tag{9.18}$$

Since $k_-^2 = (k_+^2)^*$ and

$$\frac{\mathrm{d}k_+}{\mathrm{d}\tau} = -\frac{\mathrm{d}k_-}{\mathrm{d}\tau} = \frac{1}{\sqrt{2\tau - x^2}}, \tag{9.19}$$

we obtain

$$g^{\mathrm{L}}(x, s) = \int_0^{\infty} H(\tau - x^2/2) \frac{1}{\sqrt{\tau - x^2/2}} \mathrm{Re}\, \frac{1}{\sqrt{k^2 + 1}} \mathrm{e}^{-s\tau} \, \mathrm{d}\tau. \tag{9.20}$$

The choice of the branch of k is now irrelevant. The expression (9.20) has the form of a Laplace transform, so τ is in fact the time variable t. From now on we replace τ by t, in particular in (9.17). Now (9.20) may readily be inverted to give

$$g(x, t) \equiv g_1(x, t) = H(t - x^2/2) \frac{1}{\sqrt{t - x^2/2}} \mathrm{Re}\, \frac{1}{\sqrt{k^2(x, t) + 1}}, \tag{9.21}$$

where $k(x,t)$ is defined by (9.17) with t replacing τ. Hence in the supersonic regime, the pressure on the boundary $z = 0$ takes the form

$$p(x,0,t) \equiv p_S(x,t) = \frac{1}{\pi} t^{-1/2} * g_1(x,t) \,, \tag{9.22}$$

where $*$ is the time-convolution operator.

Wave region

For $|x| > 1$, there is still the contribution $g_1(x,t)$ from the horizontal contour, but the contour of integration must now also detour around the branch cut, and so there is an extra contribution, $g_2(x,t)$, to add which corresponds to $|x| > \sqrt{2t}$. The appropriate way to write $k(x,t)$ is now

$$k(x,t) = -\mathrm{i}x + \mathrm{i}\,\mathrm{sgn}\,x\sqrt{x^2 - 2t}. \tag{9.23}$$

The new section of the contour corresponds to values of t in the range $(t_a, \frac{1}{2}x^2)$, where $t_a(x) \equiv |x| - \frac{1}{2}$ is the equation of the ray propagating at the acoustic velocity that passes through the transonic point. We hence obtain

$$g_2^{\mathrm{L}}(x,s) = \frac{1}{\sqrt{2}} \int_{x^2/2}^{t_a} \frac{\mathrm{e}^{-st}}{(\mathrm{i}\,\mathrm{sgn}\,x)|k^2+1|^{1/2}} \frac{\mathrm{d}k}{\mathrm{d}t}\,\mathrm{d}t + \frac{1}{\sqrt{2}} \int_{t_a}^{x^2/2} \frac{\mathrm{e}^{-st}}{(-\mathrm{i}\,\mathrm{sgn}\,x)|k^2+1|^{1/2}} \frac{\mathrm{d}k}{\mathrm{d}t}\,\mathrm{d}t. \tag{9.24}$$

This leads to

$$g_2(x,t) = H(x^2-1)H(x^2/2-t)H(t-t_a(x)) \frac{|k^2+1|^{-1/2}}{\sqrt{x^2/2-t}}. \tag{9.25}$$

The pressure hence takes the form

$$p_S(x,t) = \frac{1}{\pi} t^{-1/2} * [g_1(x,t) + g_2(x,t)]. \tag{9.26}$$

Figure 9.1 shows $p_S(x,t)$. The curve marked 'subsonic' is not the physical pressure, since the correct physical boundary condition (9.11) is not satisfied outside the contact area. Determining the true pressure in the contact region in the subsonic regime requires substantially more work. Note that the convolution integrals (9.22) and (9.26) have integrable singularities at the endpoints $t = x^2/2$ corresponding to the contact line.

```
% p92.m Impact CdH convolution integrals 04/29/08 SGLS
function p92

xx = 0:0.001:2; tt = [0.1 0.5 1];
[f,f2,f3] = psup(xx,tt,2);
plot(xx,-f);
xlabel('x'); ylabel('p'); legend('t = 0.1','t = 0.5','t = 1')
```

```
function [f,f2,f3]=psup(xx,tt,n)

f2=zeros(length(tt),length(xx));
f3=zeros(length(tt),length(xx));
for j=1:length(tt)
    t=tt(j);
    for k=1:length(xx)
        x=xx(k);
        if (t>x^2/2)
            f2(j,k)=quad(@phi2int,0,pi/2,[],[],x,t,n);
        end
        ta=x-0.5;
        if (t>ta & x>1)
            f3(j,k)=quad(@phi3int,0,pi/2,[],[],x,t,n);
        end
    end
end
f=f2+f3;

function f=phi2int(theta,x,t,n)

tp=x^2/2+(t-x^2/2)*sin(theta).^4;
k=-i*x+sqrt(2*tp-x^2);
r=sqrt(k.^2+1);
snz=find(theta~=0);
tnz=theta(snz);
f=zeros(size(theta));
f(snz)=-4*sin(tnz)./sqrt(1+sin(tnz).^2).*real(1./r(snz))/pi;
if (x==1 & 2*t~=x^2)
    sz=find(theta==0);
    f(sz)=-4*(2*t-x^2)^(-0.25)/2/sqrt(x)/pi;
end
if (n==1)
    f=2*(t-tp).*f;
elseif (n==0)
    f=4/3*(t-tp).^2.*f;
end

function f=phi3int(theta,x,t,n)

ta=x-0.5;
f=zeros(size(theta));
snz=find(theta~=0);
tnz=theta(snz);
sz=find(theta==0);
```

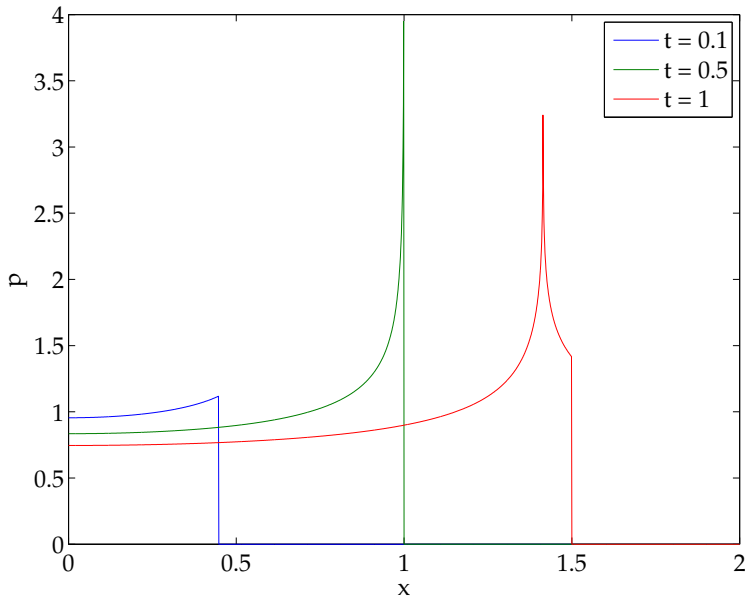

Figure 9.1: "Pressure" $p_S(x,t)$ at times $t = 0.1$ (supersonic), $t = \frac{1}{2}$ (transonic) and $t = 1$ (subsonic).

```
if (ta<t & t<x^2/2 & x>1)
    tp=ta+(t-ta)*sin(theta).^2;
    k=-i*(x-sqrt(x^2-2*tp(snz)));
    f(snz)=-2*sin(tnz)*sqrt(t-ta)./sqrt(x^2/2-tp(snz)).*abs(1./sqrt(k.^2+1))/pi;
    f(sz)=-2/sqrt(x^2/2-ta)*sqrt(x-1)*sqrt(0.5)/pi;
elseif (ta<x^2/2 & t>x^2/2 & x>1)
    tp=ta+(x^2/2-ta)*sin(theta).^2;
    k=-i*(x-sqrt(x^2-2*tp(snz)));
    f(snz)=-2*sin(tnz)*sqrt(x^2/2-ta)./sqrt(t-tp(snz)).*abs(1./sqrt(k.^2+1))/pi;
    f(sz)=-2*sqrt(x^2/2-ta)/sqrt(t-ta)/sqrt(x-1)/pi;
else
    return
end
if (n==1)
    f=2*(t-tp).*f;
elseif (n==0)
    f=4/3*(t-tp).^2.*f;
end
```

Project 5 *Carry out this calculation for an elastic solid with shear waves.*

9.4 Wavefronts using Cagniard–de Hoop

Craster (1996) obtains the Laplace transform for the pressure in a fluid overlying an elastic half-space in the form

$$\begin{aligned}\sigma_{yy}(x,y,p) &= \frac{k^4k']\epsilon F(p)}{2\pi}\int_{-\infty}^{\infty}\frac{p\gamma'(\zeta)}{c_d\gamma_0'(\zeta)} \\ &\quad\times\frac{\exp\{(p/c)d\}}{(2\zeta^2+k^2)^2-4\zeta^2\gamma_d'(\zeta)\gamma_s'(\zeta)+\epsilon k^4k'\gamma_d'(\zeta)/\gamma_0'(\zeta)}\,\mathrm{d}\zeta \qquad (9.27)\end{aligned}$$

(ζ is the Fourier transform variable). From now, we limit ourselves to $F(p) = p^{-1}$ and decorate σ_{yy} with H. More complicated forcings can be considered, for which the final solution is constructed using the convolution theorem.

The functions $\gamma_q'(\zeta)$ are defined by $\gamma_q'(\zeta) = (\zeta^2 + k_q^2)^{1/2}$, where the waveumber corresponding to the dilational wave is $k_d = 1$, the wavenumber corresponding to the shear wave is $k_s = k = c_d/c_s$ and the wavenumber corresponding to the acoustic wave in the fluid is $k_0 = k' = c_d/c_0$.

The denominator in the integrand is the elastic Rayleigh function

$$R(\zeta) = (2\zeta^2+k^2)^2 - 4\zeta^2\gamma_d'(\zeta)\gamma_s'(\zeta). \qquad (9.28)$$

The Rayleigh function has simple zeros at $\pm\mathrm{i}k_r$ where $k_r = c_d/c_r$, with c_r the Rayleigh wave speed. The Schölte function is

$$S(\zeta) = R(\zeta) + \epsilon k^4k'\gamma_d'(\zeta)/\gamma_0'(\zeta). \qquad (9.29)$$

This function has two roots at $\pm\mathrm{i}k_s$. It is quite different from the Rayleigh wave speed.

Now we consider the region $x > 0$ and adopt polar coordinates with $y = -r\cos\theta$ and $x = r\sin\theta$. Then we can solve $c_dt = \gamma_0'(\zeta)r\cos\theta + \mathrm{i}\zeta r\sin\theta$ to give

$$\zeta(t) = -\frac{\mathrm{i}c_dt}{r}\sin\theta \pm k'\left(\frac{c_0^2t^2}{r^2}-1\right)^{1/2}\cos\theta \qquad \text{for } r/c_0 \le t. \qquad (9.30)$$

This path cuts the imaginary axis at $-\mathrm{i}k'\sin\theta$. If $0 \le \theta \le \theta_{dcr}$, this is below the branch cuts of the function. Otherwise, the path must also include a section around the branch cuts with

$$\zeta_h(t) = -\frac{\mathrm{i}c_dt}{r}\sin\theta \pm \mathrm{i}k'\left(1-\frac{c_0^2t^2}{r^2}\right)^{1/2}\cos\theta \qquad \text{for } t \le r/c_0. \qquad (9.31)$$

We find

$$\sigma_{yy}^H(x,y,p) = \frac{\epsilon k'k^4}{\pi c_d}\mathrm{Re}\int_{r/c_0}^{\infty}\frac{\gamma_d'(\zeta(t))}{\gamma_0'(\zeta(t))S(\zeta(t))}\frac{\mathrm{d}\zeta(t)}{\mathrm{d}t}\mathrm{e}^{-pt}\,\mathrm{d}t \qquad \text{for } \theta_{dcr} \le \theta \le \tfrac{1}{2}\pi. \qquad (9.32)$$

Now the inverse Laplace transform is obvious. The general answer is

$$\begin{aligned}\sigma_{yy}^H(x,y,t) &= \frac{\epsilon k'k^4}{\pi c_d}\left\{H(t-r/c_0)\mathrm{Re}\left(\frac{\gamma_d'(\zeta(t))}{\gamma_0'(\zeta(t))S(\zeta(t))}\right)\right. \\ &\quad\left.+[H(t-t_d)-H(t-r/c_0)]\mathrm{Re}\left(\frac{\gamma_d'(\zeta_h(t))}{\gamma_0'(\zeta_h(t))S(\zeta_h(t))}\right)\right\}. \qquad (9.33)\end{aligned}$$

One can now find the wavefronts in the fluid explicitly. The hardest part in this process is finding the zeros of the Rayleigh and Schölte functions, and keeping track of real and imaginary parts.

Exercise 9.4 *For realistic parameter values (see Craster 1996), find the zeros of the $R(\zeta)$ and $S(\zeta)$. Be careful if you square equations that your roots actually work.*

Project 6 *Meteotsunamis are small tidal waves that are not generated by displacements of the seafloor caused by earthquakes, as is the case for normal tsunamis (for a review, see Monserrat* et al. *2006). Instead a propagating storm generates waves on the ocean surface that are amplified when their phase speed is approximately equal to the speed of motion of the forcing. Vennell (2007) examined a one-dimensional model. Investigate the two-dimensional case (Mercer* et al. *2002) using Cagniard–de Hoop (the topography will still be one-dimensional).*

9.5 References

Publications:

- Adda-Bedia, M. & Llewellyn Smith, S. G. Supersonic and subsonic states of dynamic contact between elastic bodies. *Proc.. R. Soc. Lond.* A, **463**, 759–786, 2006.
- Cagniard, L. *Réflexion et réfraction des ondes séismiques progressives.* Gauthier-Villars, Paris, 1939. (A 1962 English translation exists.)
- Craster, R. V. Wavefront expansions for pulse scattering by a surface inhomogeneity. *Q. J. Mech. Appl. Math.*, **49**, 657–674.
- Ioakimidis, N. I. Application of quadrature rules for Cauchy-type integrals to the generalized Poincaré–Bertrand formula. *Mathematics of Computing*, **44**, 199–206, 1985.
- Mercer, D., Sheng, J., Greatbatch, R. J. & Bobanovič, J. Barotropic waves generated by storms moving rapidly over shallow water. *J. Geophys. Res.*, **107**, 3152, 2002.
- Monserrat, S. Vilibić, I. & Rabinovich, A. B. Meteotsunamis: atmospherically induced destructive ocean waves in the tsunami frequency band. *Nat. Hazards Earth Syst. Sci.*, **6**, 1035–1051, 2006.
- Venell, R. Long barotropic waves generated by a storm crossing topography. *J. Phys. Oceanogr.*, **37**, 2809–2823, 2007.

Chapter 10

Multiple integrals: absolute and convective instability (05/06/08)

10.1 Case's analysis

We have mentioned Rayleigh's equation previously. As pointed out in Case (1960), previous analyses of Couette flow had omitted some modes and were unclear about the completeness and time dependence of the solutions. Completeness here refers to being able to represent any disturbance as a sum of normal modes. The set of all trigonometric functions is complete on the interval $(0, 2\pi)$, but if one remove one of them, this is no longer true. We will go through Case's analysis and plot some actual results. In the next section, we will consider the more general problem.

We consider a homogeneous, incompressible, inviscid fluid flowing between infinite parallel plates at $y = 0$ and 1. We limit ourselves to the two-dimensional problem. Linearizing about the background flow $u = y$ and writing the time- and x-dependence as $\mathrm{e}^{\mathrm{i}(kx+\sigma t)}$, we can reduce the problem to a single equation for the vertical perturbation velocity v_1 in the form

$$(\sigma + ky)\left[\frac{\mathrm{d}^2 v_1}{\mathrm{d}y^2} - k^2 v_1\right] = 0, \tag{10.1}$$

with $v_1 = 0$ at $y = 0$ and 1. This equation has no solution, so we are stuck.

Let us now try an initial-value approach. Define Fourier and Laplace transforms by

$$v^{\mathrm{L}} = \int_0^\infty \mathrm{e}^{-pt} v \,\mathrm{d}t, \qquad v^{\mathrm{F}} = \int_{-\infty}^\infty \mathrm{e}^{-\mathrm{i}kx} v \,\mathrm{d}x \tag{10.2}$$

respectively. We find

$$\left[\frac{\mathrm{d}^2}{\mathrm{d}y^2} - k^2\right] v^{\mathrm{LF}} = \frac{1}{p + \mathrm{i}ky}\left[\frac{\mathrm{d}^2}{\mathrm{d}y^2} - k^2\right] {v_i}^{\mathrm{F}}(y), \tag{10.3}$$

where ${v_0}^{\mathrm{F}}(y)$ is the Fourier transform of the initial value of $v_1(x, y)$. We can invert the Laplace transform to obtain

$$\left[\frac{\mathrm{d}^2}{\mathrm{d}y^2} - k^2\right] v^{\mathrm{F}} = \mathrm{e}^{-\mathrm{i}kyt}\left[\frac{\mathrm{d}^2}{\mathrm{d}y^2} - k^2\right] {v_i}^{\mathrm{F}}(y). \tag{10.4}$$

This equation can be interpreted using the vorticity ζ^{F}, which is proportional to $\mathrm{d}^2 v^{\mathrm{F}}/\mathrm{d}y2 - k^2 v^{\mathrm{F}}$, so that

$$\zeta^{\mathrm{F}} = \mathrm{e}^{-\mathrm{i}kyt}\zeta^{\mathrm{F}}(y,0). \tag{10.5}$$

The Fourier transform of the vorticity merely evolves by a phase factor. This is sometimes called the Orr mechanism.

We can solve(10.4) using a Green's function. The solution is

$$v^{\mathrm{F}}(y,t) = \int_0^1 G(y,y_0)\mathrm{e}^{-\mathrm{i}ky_0t}[\left[\frac{\mathrm{d}^2}{\mathrm{d}y^2} - k^2\right] v_i{}^{\mathrm{F}}(y_0)\,\mathrm{d}y_0, \tag{10.6}$$

where the Green's function is given explicitly as

$$G(y,y_0) = -\frac{1}{k\sinh k}\sinh ky_< \sinh k(1-y_>), \tag{10.7}$$

where $y_<$ and $y_>$ are the lesser and greater of y and y_0 respectively.

Case then carries out a long-time analysis based on an expansion of $v_0(y)$ as a Fourier series. The result, that the perturbation vertical velocity goes to 0 like t^{-1} seems incorrect. To see this, we compute the solution numerically for a simple initial condition in the channel, $v_0 = y(1-y)$. This discrepancy in decay rate is not mentioned in Drazin & Reid (1981).

```
% p101.m Case solution SGLS 04/30/08
k = 1;
f = inline(['-1/k/sinh(k)*sinh(min(y,y0)).*sinh(k*(1-max(y,y0))).*' ...
            'exp(-i*k*y0*t).*(-2-k^2*y0.*(1-y0))'],'y','k','t','y0');
tt = [0 logspace(0,2,21)];
yy = 0:0.01:1;
clf; figure(1); hold on
for l = 1:length(tt)
  t = tt(l)
  for j = 1:length(yy)
    v(j) = quad(@(y0) f(yy(j),k,t,y0),0,1);
  end
  vm(l) = max(abs(v));
  plot(real(v),yy,imag(v),yy); drawnow
end
hold off
xlabel('v'); ylabel('y')
figure(2)
loglog(tt,vm,'.',tt,1./tt); xlabel('t'); ylabel('max(v^F)')
```

As Case points out, there are two classes of solutions. There are discrete solutions that satisfy

$$\frac{\mathrm{d}^2 v^{\mathrm{F}}}{\mathrm{d}y^2} - k^2 v^{\mathrm{F}} = 0 \tag{10.8}$$

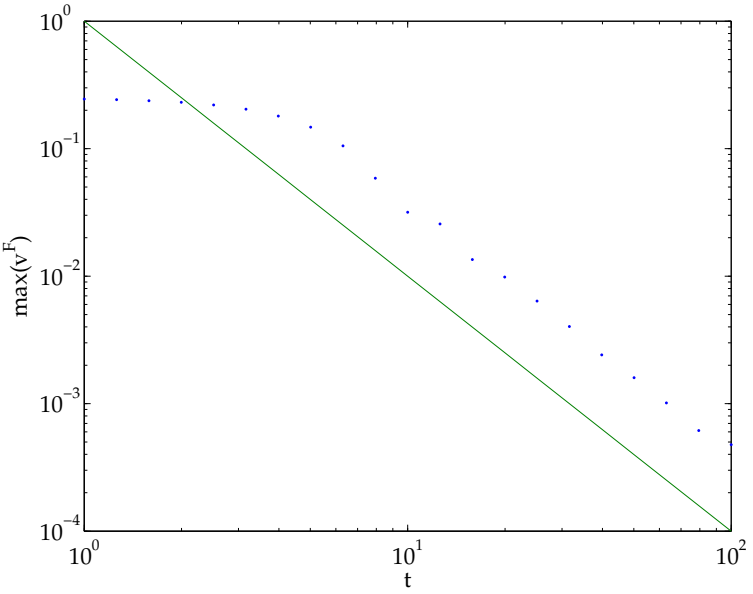

Figure 10.1: Maximum of $|v^{\mathrm{F}}|$ (dots); t^{-2} (green line).

and the boundary conditions (an empty set for Couette flow). Then there are those that satisfy

$$\frac{\mathrm{d}^2 v^{\mathrm{F}}}{\mathrm{d}y^2} - k^2 v^{\mathrm{F}} = \delta(\sigma/k + y), \tag{10.9}$$

which are in fact $G(y, -\sigma/k)$. These latter solutions form the continuous spectrum.

Case then treats more general background flows $u_0(y)$. The analysis is the same, but one has to construct a Green's function and invert a Laplace transform numerically. We now turn to the general problem of the long-time behavior of similar systems, not just the Rayleigh equation.

10.2 Convective and absolute instability

We follow the pedagogical treatment of Drazin (2002), which is very similar to the review article of Huerre & Monkewitz (1990). We are dealing with linearized stability. A background flow is said to be absolutely unstable if there exists an initial perturbation whose subsequent evolution satisfies

$$|\mathbf{u}'(\mathbf{x}, t)| \to \infty \qquad \text{as } t \to \infty \tag{10.10}$$

for all fixed $\mathbf{x}$. The flow is convectively unstable if the perturbation does not grow above some limit at any fixed point of the flow but does grow at a moving point. Then

$$|\mathbf{u}'(\mathbf{x}, t)| \to 0 \qquad \text{as } t \to \infty, \tag{10.11}$$

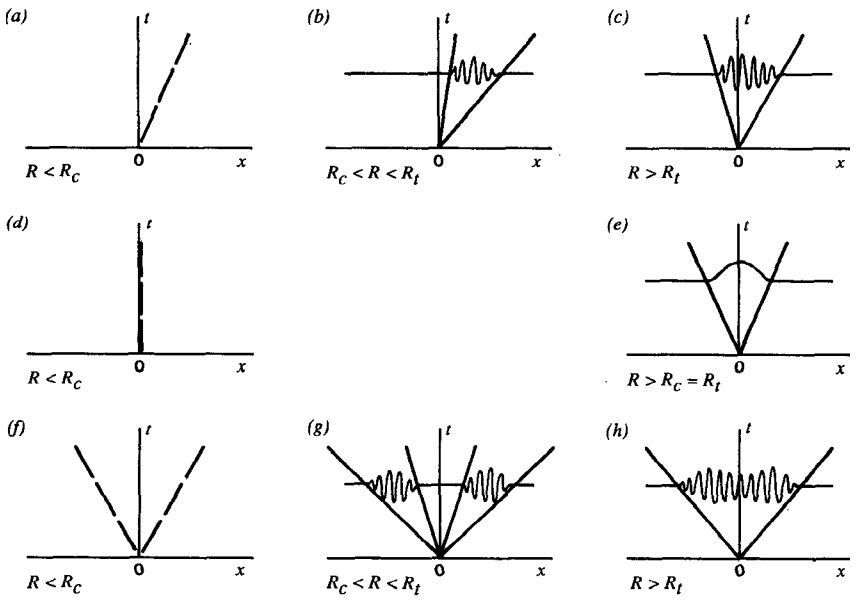

Figure 1 Sketches of typical impulse responses. Single traveling wave: (*a*) stable, (*b*) convectively unstable, (*c*) absolutely unstable. Stationary mode: (*d*) stable, (*e*) absolutely unstable. Counterpropagating traveling waves: (*f*) stable, (*g*) convectively unstable, (*h*) absolutely unstable.

Figure 10.2: From Huerre & Monkewitz (1990).

but there exists $\mathbf{V}$ such that

$$|\mathbf{u}'(\mathbf{x} + \mathbf{V}t, t)| \to \infty \qquad \text{as } t \to \infty. \tag{10.12}$$

Otherwise the flow is stable. These definitions are illustrated in Figure 10.2. These ideas are only interesting if the flow is open, so that fluid particles enter and leave through the boundaries of the domain.

This is not the same as non-normal stability (Schmid 2007) or nonlinear instability (Chomaz 2005), although there are links. Chomaz (2005) also details more recent thinking about global modes. Figure 10.3 shows a prettier color version of Figure 10.2.

We can turn the above limits into complex analysis by considering the Green's function associated with the dispersion relation $D(k, \omega; R) = 0$, where R is some control parameter. We are limiting ourselves to a one-dimensional view with $u'(x,t) = \text{Re}\,[A\mathrm{e}^{\mathrm{i}(kx-\omega t)}]$, so the dispersion relation can be identified with the differential operator in the form $D(-\mathrm{i}\partial_x, \mathrm{i}\partial_t; R) = 0$.

First we review temporal and spatial instability. In temporal stability, we consider k real and solve $D(k, \omega; R) = 0$ for ω. If, for some k, ω has positive imaginary part, the system is unstable. In spatial stability, we consider ω real (e.g. we are exciting a fixed-frequency disturbance somewhere in the flow) and solve $D(k, \omega; R) = 0$ for k. If k has negative imaginary part, the disturbance grows downstream.

Example 10.1 *This is a toy example from Drazin (2002):*

$$u'_t + Wu'_x = u'_{xx} + (R - R_c)u'. \tag{10.13}$$

The dispersion relation is $D = -\mathrm{i}\omega + \mathrm{i}Wk + k^2 - (R - R_c)$. If $R < R_c$, the system is temporally stable. If $R > R_c$ and $W \neq 0$, disturbances grow and propagate, so we have convective instability. If $R > R_c$ and $W = 0$, we have absolute instability. Note that the group velocity is given by $c_g = \partial_k \omega = W - 2\mathrm{i}k$. See the plots in Figure 10.4. Note the use of matrix exponentials in `p102.m`.

```
% p 102.m Model instability problem. See p6.m SGLS 05/06/08

N = 128; h = 2*pi/N; x = h*(1:N); t = 0;
dR = [-1 0 4 4]; W = [4 4 4 0];
tmax = 1; tplot = .1; clf, drawnow, %set(gcf,'renderer','zbuffer')
nplots = round(tmax/tplot);
dk = 1i*[0:N/2-1 0 -N/2+1:-1];
v0 = exp(-100*(x-2).^2);
v_hat = fft(v0);

for j = 1:4
 subplot(2,2,j)
 t = 0;
 data = [v0; zeros(nplots,N)]; tdata = t;
 for i = 1:nplots
   v = real(ifft(exp((-dk*W(j)+.01*dk.^2+dR(j))*t).*v_hat));
   data(i+1,:) = v; tdata = [tdata; t];
   t = t + tplot
 end
 waterfall(x,tdata,data), view(10,70), colormap(1e-6*[1 1 1]);
 axis([0 2*pi 0 tmax 0 5]), ylabel t, zlabel u, grid off; drawnow
end
```

The procedure is to look at the behavior of the Green's function defined by

$$D(-\mathrm{i}\partial_x, \mathrm{i}\partial_t; R) = \delta(x)\delta(t) \tag{10.14}$$

as $t \to \infty$ for fixed $V = x/t$. Then stability corresponds to $G \to 0$ for all V and instability to $G \to \infty$ for at least one V. For convective instability, $G \to 0$ for $V = 0$; for absolute instability $G \to \infty$ for $V = 0$. These ideas were developed earlier in the field of plasma (Briggs 1964) and then moved into fluids later.

The standard thinking is as follows. Fourier and Laplace transforming (10.14) gives

$$G(x,t) = \frac{1}{(2\pi)^2} \int_F \int_L \frac{\mathrm{e}^{\mathrm{i}(kx-\omega t)}}{D(k,\omega;R)}\, \mathrm{d}\omega\, \mathrm{d}k. \tag{10.15}$$

(Note that this is not quite a standard Laplace transform: we have $\omega = \mathrm{i}p$.) The contour L in the complex frequency plane is a straight horizontal line located above all the singularities of the integral to satisfy causality. The path F in the complex wavenumber plane is initially taken along the real axis. These paths are shown in Figure 10.5.

If the dispersion relation has a single discrete mode, one can carry out a residue calculation in the ω-plane at $\omega = \omega(k)$. This leads to

$$G(x,t) = -\frac{\mathrm{i}}{(2\pi)^2} H(t) \int_{-\infty}^{\infty} \frac{\mathrm{e}^{\mathrm{i}(kx-\omega(k)t)}}{\partial_\omega D(k,\omega(k);R)}\,\mathrm{d}k. \tag{10.16}$$

Now we can use steepest descents on the Fourier integral at large t and fixed x/t. Critical points occur at $\partial_k\omega = x/t$, and problems must be studied on a case-by-case basis. If there is a single critical point k_* and the contour of integral can be deformed onto a steepest descent path, we have

$$G(x,t) \sim -(2\pi)^{1/2}\mathrm{e}^{\mathrm{i}\pi/4} \frac{\mathrm{e}^{\mathrm{i}(k_* x-\omega(k_*)t)}}{\partial_\omega D(k_*,\omega(k_*);R)[\mathrm{d}^2\omega(k_*)/\mathrm{d}k^2 t]^{1/2}}. \tag{10.17}$$

The temporal growth rate is $\omega(k_*) - (x/t)k_*$.

INCOMPLETE. Then things become rather complicated. As the path L in the complex $\omega-$plane s moved up, the roots $k(\omega)$ of the dispersion relation move in the complex k-plane. As L approaches a root $\omega(k)$, one of the branches approaches the real axis. According to B/B, it cannot cross the axis.

It would be nice to have some simple numerical calculations of $G(x,t)$ to see how this works.

Exercise 10.1 *Solve the toy example numerically using the variables x and t.*

Exercise 10.2 *Solve the toy example numerically using inverse Fourier and Laplace transforms.*

10.3 A dissenting view

Cumberbatch argues that the method of analysis that has been described, which he calls B/B, for Briggs and Bers, gives contradictory results for a water wave problem.

Example 10.2 *This is due to Briggs, and is based on the dispersion relation*

$$D = (\omega - k)^2 - 1 + 4\mathrm{i}\omega. \tag{10.18}$$

The associated PDE is

$$(\partial_t + \partial_x)^2 u + u - 4u_t = \delta(x)\delta(t). \tag{10.19}$$

The solution to this problem is

$$u = -t[\pi(t-x)^3]^{1/2}\mathrm{e}^{(t-3x)(t+x)/4(t-x)} H(t)H(t-x). \tag{10.20}$$

This grows exponentially between the rays $t = 3x$ and $t = -x$, corresponding to absolute instability. This contradicts B/B.

Exercise 10.3 *Derive (10.20). (This is not that easy.)*

Figure 10.6 displays a contour plot of the solution (10.20), and absolute instability is clear.

```
% p103.m Briggs counter-example SGLS 05/01/08
x = -2:0.001:2; t = 0:0.001:1; [xx,tt] = meshgrid(x,t);
u = real(-tt.*(pi*(tt-xx)).^0.5.*exp((tt-3*xx).*(tt+xx)*0.25./(tt-xx)).* ...
    (tt>0).*(tt>xx));
pcolor(x,t,u); shading flat; colorbar
hold on; plot(x(x>0),3*x(x>0),'k',x(x<0),-x(x<0),'k',[0 0],[0 1],'w--'); hold off
xlabel('x'); ylabel('t')
```

Crighton & Oswell (1991) consider a problem which has similar issues to do with instability.

10.4 References

Publications:

- Briggs, R. J. *Electron-stream interactions in plasmas.* MIT Press, Boston, 1964.
- Case, K. M. Stability of inviscid plane Couette flow. *Phys. Fluids*, **3**, 143–148, 1960.
- Chomaz, J.-M. Global instabilities in spatially developing flows: Non-normality and nonlinearity. *Ann. Rev. Fluid Mech.*, **37**, 357–392, 2005.
- Crighton, D. G. & Oswell, J. E. Fluid loading with mean flow. I. Response of an elastic plate to localized excitation. *Proc. R. Soc. Lond.* A, **335**, 557–592, 1991.
- Cumberbatch, E. Wave patterns via multiple Fourier transforms. *Unpublished manuscript.*
- Huerre, P. & Monkewitz, P. A. Local and global instabilities in spatially developing flows. *Ann. Rev. Fluid Mech.*, **22**, 473–537, 1990.
- Drazin, P. G. Introduction to hydrodynamic stability. *Cambridge University Press*, Cambridge, 2002.
- Drazin, P. G. & Reid, W. H. Hydrodynamic stability. *Cambridge University Press*, Cambridge, 1981.
- Schmid, P. J. Nonmodal stability theory. *Ann. Rev. Fluid Mech.*, **39**, 129–162, 2007.

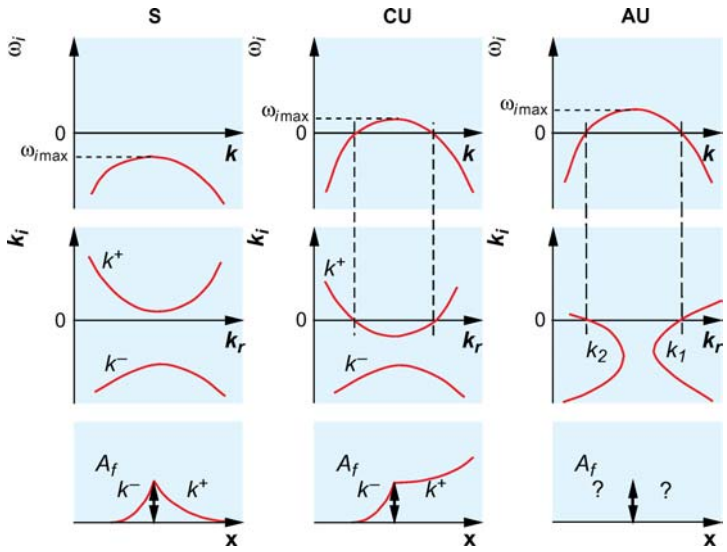

Figure 2 Classical linear stability theory for stable (S), convectively unstable (CU) and absolutely unstable (AU) flow. The first row of sketches illustrates the temporal stability theory with k real (temporal growth rate ω_i versus real wavenumber k). The second row illustrates the spatial stability theory with ω real (locus of the complex wavenumber k for varying real frequency ω) that describes the response to a harmonic forcing localized at $x = 0$. For a stable flow, the response to forcing, schematically shown in the last row, is damped and the k-branches that lie above the real k-axis propagate to the right of the forcing station. They are therefore labeled with a $+$ sign, whereas k-branches that lie below the real k-axis propagate to the left and are labeled with a $-$ sign. When a k^+-branch crosses the real k-axis the flow becomes unstable to a particular range of forcing frequencies. For a frequency in this unstable range, the spatial response to forcing is amplified downstream of the forcing station because, by a simple continuation argument, the k^+-branch keeps propagating downstream. When a k^+-branch pinches with a k^--branch, such an identification by continuity of the direction of propagation is no longer possible, the flow becomes absolutely unstable, and the response to a localized forcing cannot be defined. Any initial transient will then be amplified in situ because the wave with zero group velocity is temporally growing and overwhelms any other signal.

Figure 10.3: From Chomaz (2005).

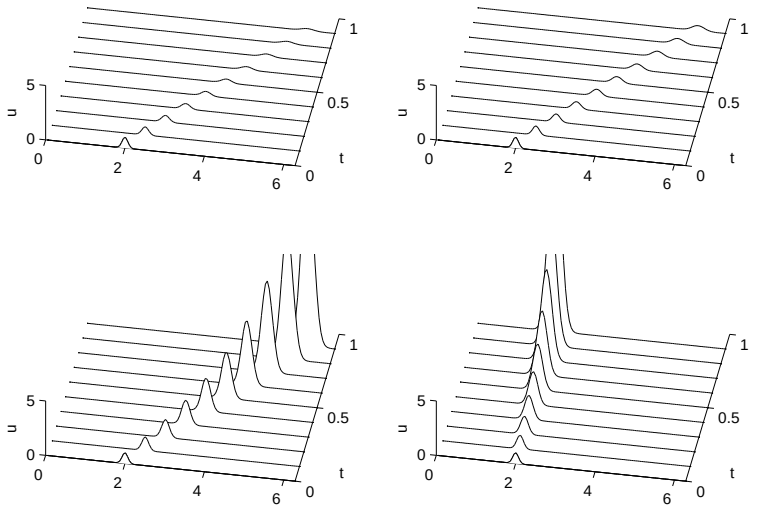

Figure 10.4: Evolution of u. $W = 2$ for (a–c), $W = 0$ for (d). $(R - R_c) = -1, 0, 4, 4$ respectively. We have stability, stability, convective instability and absolute instability.

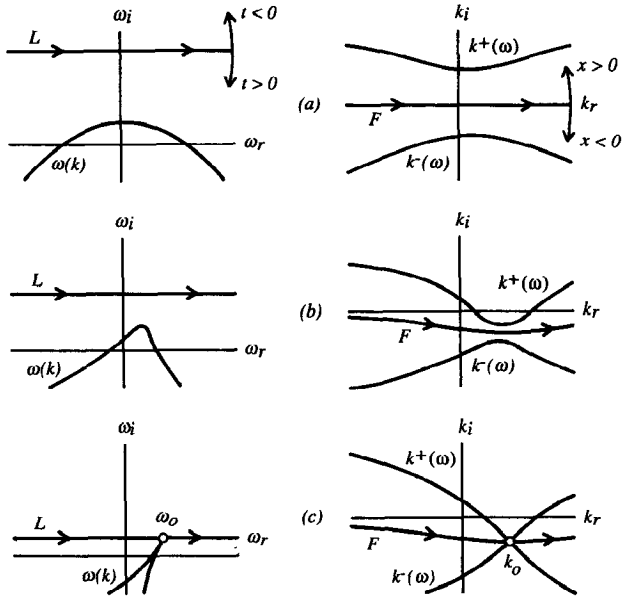

Figure 2 Loci of spatial branches $k^+(\omega)$ and $k^-(\omega)$ as L-contour is displaced downward in the complex ω-plane. (*a*), (*b*), and (*c*) refer to different stages of the pinching process.

Figure 10.5: From Huerre & Monkewitz (1990).

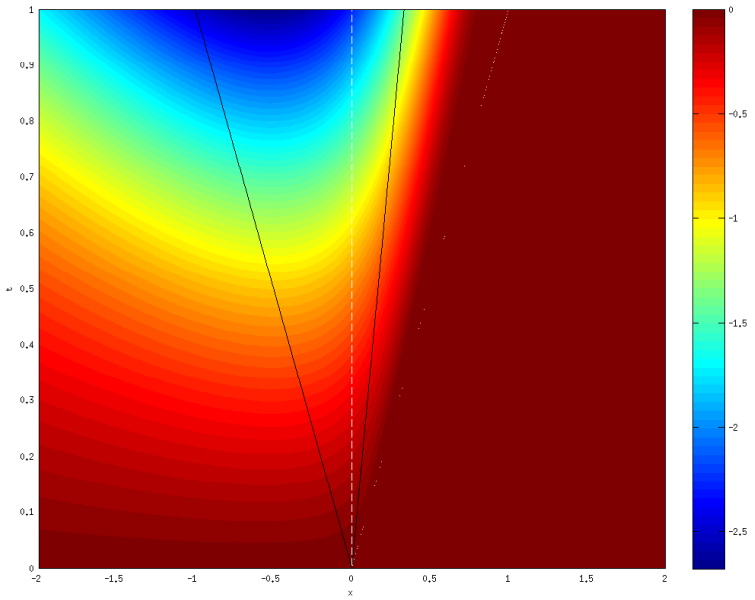

Figure 10.6: Solution to Briggs' counter-example from (10.20).

Chapter 11

The Wiener–Hopf method

11.1 The semi-infinite screen

11.1.1 The easy bit

The semi-infinite screen is always the example used to present the Wiener–Hopf method and we shall follow tradition. The classic reference is Noble (1988). Consider a rigid semi-infinite plate occupying $x < 0$, $y = 0$ immersed in a compressible fluid. A plane wave with velocity potential

$$\phi_i = \exp\left[-\mathrm{i}k_0(x\cos\Theta + y\sin\Theta) - \mathrm{i}\omega t\right] \tag{11.1}$$

with $0 < \Theta < \pi$ is incident upon this plate. The frequency ω and wavenumber k_0 are related by the dispersion relation $\omega = k_0 c$. The total velocity potential can be decomposed into $\phi_t = \phi + \phi_s$, where ϕ satisfies the Helmholtz equation

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + k_0^2\phi = 0, \tag{11.2}$$

The wavenumber will be taken to have a small imaginary part when necessary to ensure causality. We now suppress the time-dependence.

We have the following boundary conditions on the screen:

$$\frac{\partial\phi_t}{\partial y} = 0, \qquad \frac{\partial\phi}{\partial y} = \mathrm{i}k_0\sin\Theta\exp\left(-\mathrm{i}k_0 x\cos\Theta\right) \qquad \text{on } y = 0,\ x \le 0, \tag{11.3}$$

while we require ϕ_t and hence ϕ to be continuous on $y = 0$, $0 < x$. Finally $\partial\phi_t/\partial y$ and hence $\partial\phi/\partial y$ are continuous on $y = 0$. We also need some assumptions concerning the behavior of ϕ at infinity and near the edge of the screen at the origin.

Physically we know that there are three regions in this problem. In region (1), ϕ consists of a diffracted wave and a reflected wave. In region (2), ϕ consists of a diffracted wave minus the incident wave. In region (3), ϕ consists of a diffracted wave only. The reflected wave is $\exp\left[-\mathrm{i}k_0(x\cos\Theta - y\sin\Theta)\right]$. The diffracted wave behaves like $H_0^{(1)}(kr) \sim r^{-1/2}\mathrm{e}^{\mathrm{i}k_0 r}$ by causality. Hence for any fixed y,

$$|\phi| = O(\exp\left[\mathrm{Im}\ k_0(x\cos\Theta - |y|\sin\Theta)\right]) \qquad \text{for } x < -|y|\cot\Theta. \tag{11.4}$$

This says that in regions where ϕ contains a diffracted or reflected wave, it is of that order. In the region with only a diffracted wave,

$$|\phi| = O(\exp[-\text{Im } k_0 r]) \qquad \text{for } -|y|\cot\Theta < x. \tag{11.5}$$

Near the edge of the screen, we assume that the velocity potential is bounded as $x \to 0$, while $\partial\phi_t/\partial y$ is allowed to have an inverse square root singularity as $x \to +0$ on $y = 0$.

There are a number of ways of solving this problem. The original Wiener–Hopf technique was devised to solve coupled integral equations in radiation transport theory. The version we use is due to Jones, and presented in Noble (1988) very clearly.

We introduce a new object: the half-range Fourier transform. This is defined by

$$\Phi_+(k,y) = \int_0^\infty \phi e^{ikx}\,dx, \qquad \Phi_-(k,y) = \int_{-\infty}^0 \phi e^{ikx}\,dx. \tag{11.6}$$

(Note the sign convention.) The usual Fourier transform is given by the sum of the two: $\Phi = \Phi_+ + \Phi_0$. One can show that Φ_+ is analytic for Im $k >$ −Im k_0 and Φ_- is analytic for Im $k <$ Im $k_0 \cos\Theta$. As a result Φ is analytic in the strip −Im $k_0 <$ Im $k <$ Im $k_0 \cos\Theta$.

We Fourier transform the governing equation (11.2) and obtain

$$\frac{d\Phi}{dy^2} - (k^2 - k_0^2)\Phi = 0. \tag{11.7}$$

Write $\gamma = (k^2 - k_0^2)$. Then the solution is

$$\begin{aligned} \Phi &= A_1(k)e^{-\gamma y} + B_1(k)e^{-\gamma y} \qquad (y \geq 0), &(11.8)\\ &= A_2(k)e^{-\gamma y} + B_2(k)e^{-\gamma y} \qquad (y \leq 0). &(11.9) \end{aligned}$$

since Φ is discontinuous across $y = 0$. Using the fact that Φ decays for large $|y|$ and the usual definition of the branch cut leads to $B_1 = A_2 = 0$. Since $\partial\phi/\partial y$ is continuous across $y = 0$, we have

$$\Phi = \text{sgn}\, y A -_(k)e^{-\gamma|y|}. \tag{11.10}$$

Now write $\Phi(0)$ for Φ on the x-axis and so on. The continuity conditions and relations above lead to

$$\begin{aligned} \Phi_+(0) + \Phi_(+ 0) &= A(k), &(11.11)\\ \Phi_+(0) + \Phi_(- 0) &= -A(k), &(11.12)\\ \Phi'_+(0) + \Phi'_(0) &= -\gamma A(k). &(11.13) \end{aligned}$$

Note that

$$\Phi'_-(0) = \int_{-\infty}^0 (ik_0 \sin\Theta e^{-ik_0 x\cos\Theta})e^{-ikx}\,dx = \frac{k_0\sin\Theta}{k - k_0\cos\Theta}. \tag{11.14}$$

Some algebra now leads to

$$\begin{aligned} \Phi_+(0) &= -S_-, &(11.15)\\ \Phi'_+(0) + \frac{k_0\sin\Theta}{k - k_0\cos\Theta} &= -\gamma D_-. &(11.16) \end{aligned}$$

The functions S_- and D_- are unknown sum and difference functions, analytic in Im $k <$ Im $k_0 \cos\Theta$. We have two equations in four unknowns, which looks to be a problem.

11.1.2 Splits

Both (11.15) and (11.16) are in the form

$$R(k)\Phi_+(k) + S(k)\Psi_-(k) + T(k) = 0, \tag{11.17}$$

where R, S and T are given, and $+$ and $-$ functions are analytic in upper and lower half-planes with a non-trivial intersection.

The critical step in the Wiener–Hopf procedure (remember we are solving one equation with two unknown functions) is to find $K_+(k)$ and $K_-(k)$ analytic and non-zero in upper and lower half-planes such that

$$\frac{R(k)}{S(k)} = \frac{K+(k)}{K_-(k)}. \tag{11.18}$$

This is a multiplicative decomposition. Then we can rewrite (11.17) as

$$K_+(k)\Phi_+(k) + K_-(k)\Psi_-(k) + K_-(k)\frac{T(k)}{S(k)} = 0. \tag{11.19}$$

Now we carry out an additive split:

$$K_-(k)\frac{T(k)}{S(k)} = T+(k) + T_-(k). \tag{11.20}$$

We can now rewrite the equation with $-$ functions on one side and $+$ functions on the other side:

$$J(k) = K_+(k)\Phi_+(k) + C_+(k) = K_-(k)\Phi_-(k) + C_-(k). \tag{11.21}$$

We have two expressions that are analytic in upper and lower half-planes respectively, and that overlap over a non-trivial strip. Hence they are the same function, $J(k)$, which is entire. If we can bound the second and third terms appropriately, the extended form of Liouville's theorem tells us that $J(k)$ is a polynomial. There are a finite number of unknowns in the polynomial that come from physical conditions (holding near the edge).

Sometimes both of the splits can be done by hand. It's always a good idea to try and factorize expressions and cancel out singularities. If not, there is a formal way of obtaining any multiplicative or additive split:

Theorem 8 *Let $f(k)$ be an analytic function of $k = \sigma + \mathrm{i}\tau$, analytic in the strip $\tau_- < \tau < \tau_+$, such that $|f(k)| < C|\sigma|^{-p}$, $p > 0$, for $|\sigma| \to \infty$, the inequality holding uniformly for all τ in the strip $\tau_- + \epsilon <\leq \tau \leq \tau_+ + \epsilon$, $\epsilon > 0$. Then for $\tau_- < c < \tau < d < \tau_+$,*

$$f(k) = f +_(k) + f_-(k), \tag{11.22}$$

$$f_+(k) = \frac{1}{2\pi\mathrm{i}}\int_{-\infty+\mathrm{i}c}^{\infty+\mathrm{i}c} \frac{f(\zeta)}{\zeta - k}\,\mathrm{d}\zeta, \qquad f_-(k) = -\frac{1}{2\pi\mathrm{i}}\int_{-\infty+\mathrm{i}d}^{\infty+\mathrm{i}d} \frac{f(\zeta)}{\zeta - k}\,\mathrm{d}\zeta. \tag{11.23}$$

Theorem 9 *If $\log K(k)$ satisfies the conditions of the previous theorem, which implies in particular that $K(k)$ is analytic and non-zero in a strip $\tau_- < \tau < \tau_+$ and $K(k) \to 1$ as $\sigma \to \pm\infty$ in the strip, then we can write $K(k) = K_+(k)K_-(k)$ where $K_+(k)$, $K_-(k)$ are analytic, bounded and non-zero in $\tau < \tau_-$, $\tau < \tau_+$ respectively.*

11.1.3 Solution

We can carry out the multiplicative split in (11.16) by hand:

$$\frac{\Phi'_+(0)}{(k+k_0)^{1/2}} + \frac{k_0 \sin\Theta}{(k+k_0)^{1/2}(k-k_0\cos\Theta)} = -(k-k_0)^{1/2} D_-. \tag{11.24}$$

The second term is not regular in either half-plane. We deal with it in a very standard way, giving

$$\begin{aligned} & \frac{k_0 \sin\Theta}{(k+k_0)^{1/2}(k-k_0\cos\Theta)} \\ = \quad & \frac{k_0 \sin\Theta}{(k-k_0\cos\Theta)} \left[\frac{1}{(k+k_0)^{1/2}} - \frac{1}{(k_0+k_0\cos\Theta)^{1/2}} \right] + \frac{k_0 \sin\Theta}{(k_0+k_0\cos\Theta)^{1/2}(k-k_0\cos\Theta)} \qquad (11.25) \\ = \quad & H_+(k) + H_-(k). \qquad (11.26) \end{aligned}$$

Rearrange to obtain

$$J(k) = (k+k_0)^{1/2}\Phi'_+(0) + H_+(k) = -(k-k_0)^{1/2} D_- - H_-(k). \tag{11.27}$$

Some analysis of the edge conditions shows that $J(k)$ tends to zero for large k. This is a relief, since we had no other conditions left to fix terms in the polynomial. So by Liouville's theorem, we obtain

$$\begin{aligned} \Phi'_+(0) &= -(k+k_0)^{1/2} H_+(k), & (11.28) \\ D_- &= = -(k-k_0)^{1/2} H_-(k). & (11.29) \end{aligned}$$

Putting this together leads to

$$\phi = \mp \frac{1}{2\pi} (k_0 - k_0\cos\Theta)^{1/2} \int_{-\infty+\mathrm{i}a}^{\infty+\mathrm{i}a} \frac{\mathrm{e}^{-\mathrm{i}kx \mp \gamma y}}{(k-k_0)^{1/2}(k-k_0\cos\Theta)}\, \mathrm{d}k, \tag{11.30}$$

where $-\mathrm{Im}\ k_0 < a < \mathrm{Im}\ k_0\cos\Theta$ and with the upper and lower signs referring to $y \geq 0$ and $y \leq 0$ respectively. We can do the same thing with (11.15) but we don't need to.

Exercise 11.1 *Work out the large-r behavior of (11.30) using the method of steepest descents.*

The form (11.30) is special, because it can be turned into a Fresnel integral. It turns out that for $-\pi \leq \theta \leq \pi$, with θ giving the angle with respect to the x-axis,

$$\begin{aligned} \phi_t \quad = \quad & \pi^{-1/2} \mathrm{e}^{-\mathrm{i}\pi/4} [\mathrm{e}^{-\mathrm{i}k_0 r\cos(\theta-\Theta)} F\{(-2k_0 r)^{1/2} \cos\tfrac{1}{2}(\theta-\Theta)\} \\ & + \mathrm{e}^{-\mathrm{i}k_0 r\cos(\theta+\Theta)} F\{(2k_0 r)^{1/2} \cos\tfrac{1}{2}(\theta+-\Theta)\}]. \end{aligned} \tag{11.31}$$

The resulting field is shown in Figure 11.1.

```
% p111.m Fresnel solution for semi-infinity screen SGLS 05/05/08
k = 1.2; kF = k*sqrt(2/pi);
T = 0.45;
x = -10:0.25:10; y = x; [xx,yy] = meshgrid(x,y);
r = sqrt(xx.^2+yy.^2); t = atan2(yy,xx);
Fmr = 0.5 - mfun('FresnelC',-(2*kF*r).^0.5.*cos(0.5*(t-T)));
Fmi = 0.5 - mfun('FresnelS',-(2*kF*r).^0.5.*cos(0.5*(t-T)));
Fm = sqrt(pi/2)*(Fmr + i*Fmi);
Fpr = 0.5 - mfun('FresnelC',(2*kF*r).^0.5.*cos(0.5*(t+T)));
Fpi = 0.5 - mfun('FresnelS',(2*kF*r).^0.5.*cos(0.5*(t+T)));
Fp = sqrt(pi/2)*(Fpr + i*Fpi);
phi = pi^(-0.5)*exp(-i*pi/4)*(exp(-i*k*r.*cos(t-T)).*Fm + exp(-i*k*r.*cos(t+T)).*Fp);
subplot(2,2,1)
contour(x,y,real(phi)); colorbar; hold on; plot([-5 0],[0 0],'k'); hold off
xlabel('x'); ylabel('y')
subplot(2,2,2)
contour(x,y,imag(phi)); colorbar; hold on; plot([-5 0],[0 0],'k'); hold off
xlabel('x'); ylabel('y')
subplot(2,2,3)
contour(x,y,abs(phi)); colorbar; hold on; plot([-5 0],[0 0],'k'); hold off
xlabel('x'); ylabel('y')
subplot(2,2,4)
contour(x,y,angle(phi)); colorbar; hold on; plot([-5 0],[0 0],'k'); hold off
xlabel('x'); ylabel('y')
```

Exercise 11.2 *Derive (11.31). Work out the large-r behavior of (11.31) using the method of steepest descents.*

11.2 Numerical splits

The split that we used in the semi-infinite screen case was easy: γ could essentially be factorized by hand by looking at its poles and branch points. The additive split was just a case of removing a pole. In many cases, this is not possible. The Rayleigh and Schölte functions of Chapter 9 are cases where this is not possible.

We must then resort to carrying out the split numerically. This is possible from Theorems (8) and (9). The resulting numerically evaluated functions are then put back into the Fourier inversion algorithms or used as needed.

In Abrahams *et al.* (2008), a paper mentioned previously, one can find the following equation

$$r^+(k)r^-(k)\cosh[\Delta(k)(\theta^+(k)+\theta^-(k))] = (1+\frac{2}{3}k^2)g - \Delta\sqrt{f^2+e^2}, \tag{11.32}$$

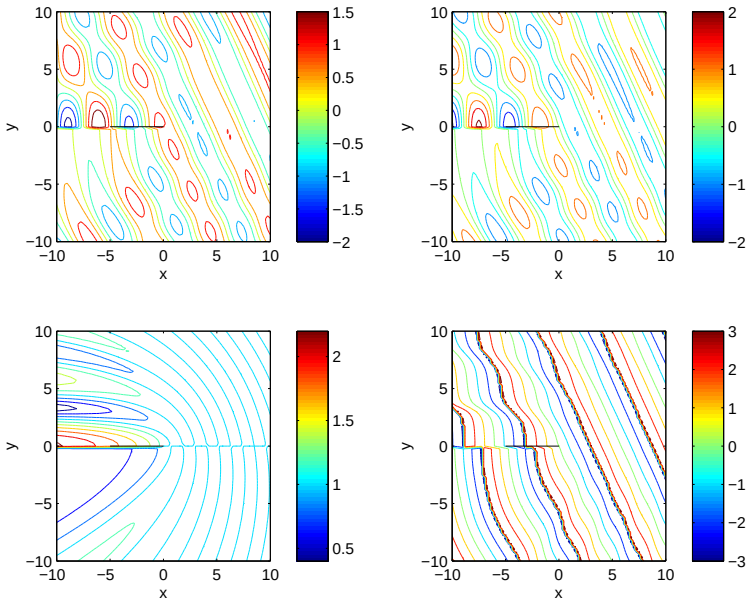

Figure 11.1: Total field ϕ_t calculated from (11.31) for $k = 1.2$ and $\Theta = 0.45$ (meaningless values). Contour plots of (a) real part; (b) imaginary part; (c) absolute value; (d) argument.

$$r^+(k)r^-(k)\sinh[\Delta(k)(\theta^+(k)+\theta^-(k))] = (1+\frac{2}{3}k^2)\sqrt{f^2+e^2} - \Delta g, \tag{11.33}$$

where

$$e(k) = k\left(1+\nu+\frac{2k^2}{\sinh^2 k - k^2}\right), \tag{11.34}$$

$$f(k) = \frac{2k^2}{\sinh^2 k - k^2}, \tag{11.35}$$

$$g(k) = \frac{k\sinh 2k}{\sinh^2 k - k^2}, \tag{11.36}$$

$$\Delta(k) = (1+k^2)^{1/2}. \tag{11.37}$$

We can separate the two equations to give

$$[r^+(k)r^-(k)]^2 = \frac{1}{3}k^2(1+\frac{4}{3}k^2)(g^2-f^2-e^2), \tag{11.38}$$

$$\tanh[\Delta(k)(\theta^+(k)+\theta^-(k))] = \frac{(1+\frac{2}{3}k^2)\sqrt{f^2+e^2}-\Delta g}{(1+\frac{2}{3}k^2)g - \Delta\sqrt{f^2+e^2}}. \tag{11.39}$$

Then from Theorem 9,

$$\theta^+(k) = \frac{1}{2\pi i}\int_{-\infty}^{\infty}\frac{1}{\Delta(\zeta)}\tanh^{-1}\left\{\frac{(1+\frac{2}{3}\zeta^2)\sqrt{f^2(\zeta)+e^2(\zeta)}-\Delta(\zeta)g(\zeta)}{(1+\frac{2}{3}\zeta^2)g(\zeta)-\Delta(\zeta)\sqrt{f^2(\zeta)+e^2(\zeta)}}\right\}\frac{d\zeta}{\zeta-k}$$

$$= \frac{k}{\pi i}\int_0^\infty \frac{1}{\Delta(\zeta)}\tanh^{-1}\left\{\frac{(1+\frac{2}{3}\zeta^2)\sqrt{f^2(\zeta)+e^2(\zeta)}-\Delta(\zeta)g(\zeta)}{(1+\frac{2}{3}\zeta^2)g(\zeta)-\Delta(\zeta)\sqrt{f^2(\zeta)+e^2(\zeta)}}\right\}\frac{d\zeta}{\zeta^2-k^2}, \qquad (11.40)$$

valid for $\mathrm{Im}(k) > 0$. The last result is true because the integrand is even in ζ, which further implies that

$$\theta^-(k) = \theta^+(-k) \qquad \text{for } k \text{ in the lower half-plane.} \qquad (11.41)$$

Actually, any path parallel to the real axis suffices for the full range integral and so if $\theta^+(k)$ is required for real k then the first integral must be indented below the chosen k. The existence of the integral representations in (11.40) is confirmed by noting that the right-hand side of (11.39) is $O(k^2)$ as $k \to 0$, $\sim \frac{1}{2}(1+\nu)$ as $|k| \to \infty$ and is bounded in the lower half-plane. Hence this representation is ideal for computing $\theta^\pm(k)$ and can be directly coded for numerical evaluation.

This procedure has to be modified for $[r^+(k)r^-(k)]^2$ in (11.38) whose right-hand side tends to 4 as $k \to 0$ and $\sim (3+\nu)(1-\nu)4k^6/9$ as $|k| \to \infty$ in the lower half-plane. The latter behaviour is not suitable for direct application of the product decomposition formula,which requires a function that tends to the value unity at infinity. This is simply circumvented by applying a suitable divisor to (11.38), employing the standard factorization formula, and then decomposing the divisor into upper and lower half functions by inspection. From

$$\frac{1}{3}k^2(g^2-f^2-e^2) = \left[\frac{k^2\sinh^2 k}{3(\sinh^2 k - k^2)}\right]\left\{\frac{k^2}{\sinh^2 k}[4+(1-\nu)^2k^2]+(3+\nu)(1-\nu)k^2\right\},$$

we find

$$\begin{aligned} r^+(k) &= (1-2ik/\sqrt{3})^{1/2}(1-ik/\sqrt{3})^{1/2}[2-ik(3+\nu)^{1/2}(1-\nu)^{1/2}]^{1/2}\times \\ &\quad \exp\left\{\frac{1}{4\pi i}\int_{-\infty}^{\infty}\log\left[\frac{[g^2(\zeta)-f^2(\zeta)-e^2(\zeta)]\zeta^2}{(\zeta^2+3)[4+(3+\nu)(1-\nu)\zeta^2]}\right]\frac{d\zeta}{\zeta-k}\right\} \end{aligned} \qquad (11.42)$$

for $\mathrm{Im}(k) > 0$ and indentation of the contour below k is taken if k is real. Note that the exponential function in this expression may be re-expressed as

$$\exp\left\{\frac{k}{2\pi i}\int_0^\infty \log\left[\frac{[g^2(\zeta)-f^2(\zeta)-e^2(\zeta)]\zeta^2}{(\zeta^2+3)[4+(3+\nu)(1-\nu)\zeta^2]}\right]\frac{d\zeta}{\zeta^2-k^2}\right\},$$

where convergence of this and the above integral are now ensured. The function $r^-(k)$, analytic in the lower half plane, is again, due to the symmetry, simply obtained from

$$r^-(-k) = r^+(k) \qquad \text{for } k \text{ in the lower half-plane.} \qquad (11.43)$$

The numerical program `p112.m` is long. Figure 11.2 shows computed minus and plus functions, as well as their product and sums compared to the original functions. One can't tell from this diagram that the functions are analytic, although this can be seen by taking values of Im k that go through singularities of $\theta(k)$ and $r(k)$.

```
% p112.m Numerical split SGLS 05/0608

function p112

global nu

nu = 0.3; ai = 0.2;;
aa = (-1:.025:1) - ai*i;
aa = (-10:.5:10) - ai*i;
aa = (-10:.5:10) - ai*i;
ax = real(aa);
t = tex(aa);
r = rex(aa);
r = (1+4*aa.^2/3).^0.5.*(1+aa.^2/3).^0.5.*(4+(3+nu)*(1-nu)*aa.^2).^0.5.*exp(r);
[tm,tp,rm,rp] = trdo(aa);

subplot(2,2,1)
plot(ax,real(tm),ax,imag(tm),'--',ax,real(tp),ax,imag(tp),'--')
subplot(2,2,2)
plot(ax,real(t),ax,imag(t),'--',ax,real(tm+tp),ax,imag(tm+tp),'--')
subplot(2,2,3)
plot(ax,real(rm),ax,imag(rm),'--',ax,real(rp),ax,imag(rp),'--')
subplot(2,2,4)
plot(ax,real(r),ax,imag(r),'--',ax,real(rm.*rp),ax,imag(rm.*rp),'--')


function [tm,tp,rm,rp] = trdo(aa)

global nu

tm = zeros(size(aa));
rm = zeros(size(aa));
tp = zeros(size(aa));
rp = zeros(size(aa));

for j = 1:length(aa)
  al = aa(j);
  i1 = quadl(@(z) tcv(z,al),-1,1);
  i2 = quadl(@(zeta) ti(zeta,al),-1,1);
  tm(j) = i1 + i2;
  if (imag(al)>=0)
    tm(j) = tm(j) - 2*pi*i*tex(al);
  end
  tp(j) = i1 + i2;
```

```
  if (imag(al)<0)
    tp(j) = tp(j) + 2*pi*i*tex(al);
  end
  i1 = quadl(@(z) rcv(z,al),-1,1);
  i2 = quadl(@(zeta) ri(zeta,al),-1,1);
  rm(j) = i1 + i2;
  if (imag(al)>=0)
    rm(j) = rm(j) - 2*pi*i*rex(al);
  end
  rp(j) = i1 + i2;
  if (imag(al)<0)
    rp(j) = rp(j) + 2*pi*i*rex(al);
  end
end
tm = -tm/(2*pi*i);
tp =  tp/(2*pi*i);
rm = -rm/(2*pi*i);
rp =  rp/(2*pi*i);
rm = (1+i*2*aa/sqrt(3)).^0.5.*(1+i*aa/sqrt(3)).^0.5.*(2+i*sqrt((3+nu)*(1-nu))*aa).^0.5.*
rp = (1-i*2*aa/sqrt(3)).^0.5.*(1-i*aa/sqrt(3)).^0.5.*(2-i*sqrt((3+nu)*(1-nu))*aa).^0.5.*

function tt = tex(al)

n = (1+2/3*al.^2).*sqrt(f2(al).^2+e2(al).^2)-Del(al).*g(al).*al.^2;
d = (1+2/3*al.^2).*g(al).*al.^2 - Del(al).*sqrt(f2(al).^2+e2(al).^2);
tt = atanh(n./d)./Del(al);
tt(al==0) = 0;

function rr = rex(al)

global nu

warning off
n = (g(al).^2-f(al).^2-e(al).^2).*al.^2;
d = (3+al.^2).*(4 + (3+nu)*(1-nu)*al.^2);
rr = 0.5*log(n./d);
rr(al==0) = 0;
warning on

function e = e(al)

global nu

e = al.*(1 + nu + f(al));
```

```
function e = e2(al)

global nu

e = al.^3.*(1 + nu) + al.*f2(al);

function f = f(al)

%f = 2*al.^2./(sinh(al).^2 - al.^2);
warning off
im = real(al)<0;
al(im) = -al(im);
f = zeros(size(al));
ib = real(al)>1; ab = al(ib);
f(ib) = ab.^2*8.*exp(-2*ab)./((1-exp(-2*ab)).^2-4*ab.^2.*exp(-2*ab));
is = real(al)<=1; as = al(is);
f(is) = 2./((sinh(as)./as).^2-1);
warning on

function f = f2(al)

%f = 2*al.^2./(sinh(al).^2 - al.^2);
warning off
im = real(al)<0;
al(im) = -al(im);
f = zeros(size(al));
ib = real(al)>1; ab = al(ib);
f(ib) = ab.^2*8.*exp(-2*ab)./((1-exp(-2*ab)).^2-4*ab.^2.*exp(-2*ab));
is = real(al)<=1; as = al(is);
f(is) = 2./((sinh(as)./as).^2-1);
f = f.*al.^2;
f(al==0) = 6;
warning on

function g = g(al)

%g = al.*sinh(2*a)./(sinh(al).^2 - al.^2);
warning off
im = real(al)<0;
al(im) = -al(im);
g = zeros(size(al));
ib = real(al)>1; ab = al(ib);
g(ib) = ab*2.*(1-exp(-4*ab))./((1-exp(-2*ab)).^2-4*ab.^2.*exp(-2*ab));
is = real(al)<=1; as = al(is);
g(is) = (sinh(2*as)./as)./((sinh(as)./as).^2-1);
```

```
warning on

function tt = ti(x,al)

ya = imag(al);
ym = 0.5;
if (abs(ya)>ym)
  y = 0;
elseif (ya>=0)
  y = ya-ym;
else
  y = ya+ym;
end
zeta = x + i*y;
tt = tex(zeta)./(zeta-al);

function tt = tcv(t,al)

inz = find(t~=0);
tnz = t(inz);
tt = zeros(size(t));
tt(inz) = ti(1./tnz,al)./tnz.^2;

function rr = ri(x,al)

ya = imag(al);
ym = 0.5;
if (abs(ya)>ym)
  y = 0;
elseif (ya>=0)
  y = ya-ym;
else
  y = ya+ym;
end
zeta = x + i*y;
rr = rex(zeta)./(zeta-al);

function rr = rcv(t,al)

inz = find(t~=0);
tnz = t(inz);
rr = zeros(size(t));
rr(inz) = ri(1./tnz,al)./tnz.^2;
```

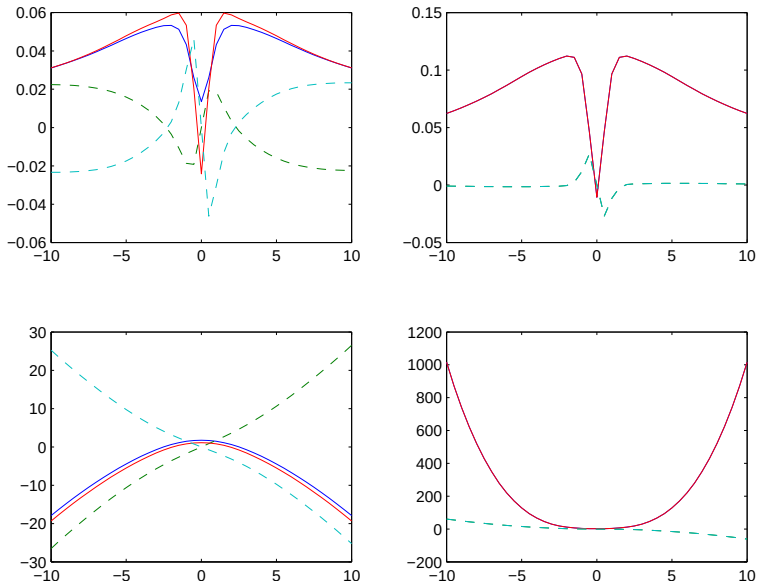

Figure 11.2: Numerical splits for Im $k = -0.2$. (a) $\theta^-(k)$ and $\theta^+(k)$ (imaginary parts dashed); (b) their sum and $\theta(k)$; (c) (a) $r^-(k)$ and $r+(k)$ (imaginary parts dashed); (b) their product and $r(k)$.

11.3 Mixed boundary-value problems

The Wiener–Hopf method is useful for solving a certain class of mixed boundary-value problems, in which ϕ is given over part of the boundary and $\partial\phi/\partial_n$ is given over the rest of the boundary (or some other combination of boundary conditions). Such problems are usually harder to solve than Dirichlet or Neumann problems.

Sneddon (1966) is a clear and useful reference. Many techniques use coupled integral equations. In some cases these can be solved by hand. In other cases, particularly when there are more than two boundary conditions, numerical approaches are necessary. Often the solution is expressed along different parts of the contours using different expansion functions, for example Chebyshev polynomials, and one obtains an infinite set of linear equations that are then truncated and solved numerically. This approach is sometimes combined with the Wiener–Hopf technique.

There have been developments since 1966, and a particularly interesting approach is outlined in Fabrikant (1991). Very elegant formulas can be obtained directly for some problems.

11.4 References

Publications:

- Abrahams, I. D., Davis, A. M. J. & Llewellyn Smith, S. G. Matrix Wiener–Hopf approximation for a partially clamped plate. *Q. J. Mech. Appl. Math.*, doi: 10.1093/qj-

mam/hbn004.

- Fabrikant, V. I. *Mixed boundary value problems of potential theory and their applications in engineering*, Kluwer, Dordrecht, 1991.
- Noble, B. *Methods based on the Wiener–Hopf technique*, Chelsea, New York, 1988.
- Sneddon, I. M. *Mixed boundary value problems in potential theory*, North-Holland, Amsterdam, 1966.

Chapter 12

Matrix factorizations

12.1 Introduction

The transverse oscillations of a thin elastic plate, which may be statically or dynamically loaded, are governed by a differential equation, fourth order in space and second order in time, that is a standard example in texts devoted to separation of variables and Fourier transform techniques. A classic unsolved problem in bending plate theory is the displacement of a uniformly loaded infinite strip having one edge clamped and the other clamped or free on two semi-infinite intervals. The parallel lines' geometry suggests the use of the Wiener–Hopf technique but the advantage of the parallel edges in creating constant bending profiles far away in either direction is offset by the appearance of a matrix Wiener–Hopf system.

The Wiener–Hopf technique has, since its invention in 1931, proved immensely valuable in determining exact solutions to a huge variety of problems arising in physics, engineering, and applied mathematics. Whilst the method is straightforward to apply for scalar equations, a key step fails in general for matrix systems; see a detailed discussion on this point below. Here this classic problem is tackled using a matrix Wiener–Hopf formulation. The solution is achieved by an approximate rather than exact procedure via application of Padé approximants.

12.2 The Wiener–Hopf Problem

In terms of Cartesian coordinates (x, y), the static displacement $w(x, y)$ of a plate, situated at $-\infty < x < \infty$, $0 \leq y \leq 1$, is governed by the fourth order equation

$$\nabla^4 w = \frac{q}{D}, \tag{12.1}$$

where q is a uniform load applied normal to the plane of the plate and D is the constant flexural rigidity. Physical choices of boundary conditions at a plate edge, include a clamped edge:

$$w = 0 = \frac{\partial w}{\partial y} \tag{12.2}$$

and a free edge:

$$\nu\frac{\partial^2 w}{\partial x^2}+\frac{\partial^2 w}{\partial y^2}=0, \qquad (2-\nu)\frac{\partial^3 w}{\partial x^2\partial y}+\frac{\partial^3 w}{\partial y^3}=0, \tag{12.3}$$

in which ν is Poisson's ratio ($0\le\nu\le 1/2$). Thus, for a plate clamped at $y=0,1$,

$$w=\frac{q}{24D}y^2(1-y)^2 \tag{12.4}$$

while, for a plate clamped at $y=0$ but free at $y=1$,

$$w=\frac{q}{24D}y^2(6-4y+y^2). \tag{12.5}$$

Consider a plate clamped at $y=0$, all x, and $y=1$, $x<0$ but free at $y=1$, $x>0$. Then, in terms of two sets of Papkovitch–Fadle eigenfunctions whose details are not needed here, (12.4,12.5) yield

$$\begin{aligned} w &= \frac{q}{24D}y^2(1-y)^2+\sum_{n\neq 0}A_n e^{\lambda_n x}\phi_n(y) \qquad (x<0),\\ w &= \frac{q}{24D}y^2(6-4y+y^2)+\sum_{n\neq 0}B_n e^{-\mu_n x}\psi_n(y) \qquad (x>0). \end{aligned} \tag{12.6}$$

Now, in view of (12.4), it is advantageous to write the total displacement as

$$w_{tot}=\frac{q}{24D}y^2(1-y)^2+\frac{q}{D}W, \tag{12.7}$$

where W satisfies (12.2) at $y=0$ and $y=1$, $x<0$ and, from (12.1) and (12.3), W is biharmonic and

$$\nu\frac{\partial^2 W}{\partial x^2}+\frac{\partial^2 W}{\partial y^2}=-\frac{1}{12}e^{-\epsilon x}, \quad (2-\nu)\frac{\partial^3 W}{\partial x^2\partial y}+\frac{\partial^3 W}{\partial y^3}=-\frac{1}{2}e^{-\epsilon x} \quad (y=1, x>0). \tag{12.8}$$

Here the exponential factors have been introduced for mathematical convenience, with ϵ a small positive number that will revert to zero after application of the Wiener–Hopf technique.

In terms of the Fourier transform

$$\Phi(k,y)=\int_{-\infty}^{\infty}W(x,y)e^{ikx}dx, \tag{12.9}$$

the boundary conditions (12.2), (12.8) yield

$$\Phi(k,0)=0=\Phi_y(k,0), \tag{12.10}$$

$$\int_{-\infty}^{0}W(x,1)e^{ikx}dx=\Phi^-(k,1)=0=\Phi_y^-(k,1)=\int_{-\infty}^{0}W_y(x,1)e^{ikx}dx, \tag{12.11}$$

$$\int_{0}^{\infty}[W_{yy}-\nu k^2 W]e^{ikx}dx=\Phi_{yy}^+(k,1)-\nu k^2\Phi^+(k,1)=\frac{-i}{12(k+i\epsilon)}, \tag{12.12}$$

$$\int_0^\infty [W_{yyy} - (2-\nu)k^2 W_y] e^{ikx} dx = \Phi^+_{yyy}(k,1) - (2-\nu)k^2 \Phi^+_y(k,1) = \frac{-i}{2(k+i\epsilon)}. \quad (12.13)$$

Convergence of the above Fourier full and half-range transforms is ensured if k lies in an infinite strip containing the real line, here and henceforth referred to as $\mathcal{D}$, with its width limited from below by the singularity at $k = -i\epsilon$. Evidently the unknown pairs of (half-range transform) functions $\Phi^+(k,1)$, $\Phi^+_y(k,1)$ and $\Phi^-_{yy}(k,1)$,$\Phi^-_{yyy}(k,1)$ are regular in the region above and including $\mathcal{D}$, denoted $\mathcal{D}^+$, and the region below and including $\mathcal{D}$, denoted $\mathcal{D}^-$, respectively. Thus, $\mathcal{D}^+ \cap \mathcal{D}^- \equiv \mathcal{D}$.

The displacement W is bounded at $x = \pm\infty$, so the biharmonic equation can be Fourier transformed (12.9) to give

$$\left(\frac{d^2}{dy^2} - k^2\right)^2 \Phi = 0 \quad (12.14)$$

and hence a general solution which satisfies (12.10) is

$$\Phi(k,y) = A(k)ky \sinh ky + B(k)(ky \cosh ky - \sinh ky). \quad (12.15)$$

Application of the conditions (12.11) now yields:

$$\begin{pmatrix} A(k) \\ B(k) \end{pmatrix} = \frac{1}{\sinh^2 k - k^2} \times \begin{pmatrix} k\sinh k & -(k\cosh k - \sinh k) \\ -(k\cosh k + \sinh k) & k\sinh k \end{pmatrix} \begin{pmatrix} \Phi^+(k,1) \\ k^{-1}\Phi^+_y(k,1) \end{pmatrix}. \quad (12.16)$$

Then the use of the conditions (12.12,12.13) facilitates the deduction of the following **matrix** Wiener-Hopf equation

$$\begin{pmatrix} -\Phi^-_{yyy}(k,1) \\ \Phi^-_{yy}(k,1) \end{pmatrix} - \frac{i}{2(k+i\epsilon)} \begin{pmatrix} -1 \\ 1/6 \end{pmatrix} = \mathbf{K}(k) \begin{pmatrix} \Phi^+(k,1) \\ \Phi^+_y(k,1) \end{pmatrix}, \quad (12.17)$$

where

$$\mathbf{K}(k) = \begin{pmatrix} k^2[g(k)+f(k)] & -ke(k) \\ -ke(k) & g(k)-f(k) \end{pmatrix}, \quad (12.18)$$

$$e(k) = k\left(1+\nu+\frac{2k^2}{\sinh^2 k - k^2}\right), \quad (12.19)$$

$$f(k) = \frac{2k^2}{\sinh^2 k - k^2}, \quad (12.20)$$

$$g(k) = \frac{k\sinh 2k}{\sinh^2 k - k^2}. \quad (12.21)$$

The determinant of the kernel is

$$|\mathbf{K}(k)| = \frac{k^4[4+(3+\nu)(1-\nu)\sinh^2 k + (1-\nu)^2 k^2]}{\sinh^2 k - k^2}, \quad (12.22)$$

and the complex numbers $\{\mu_n, \lambda_n; n \geq 1\}$ appearing in (12.6) are the zeros in the first quadrant of the numerator and denominator respectively. Negative values of n denote complex conjugates.

It remains to consider the neighbourhood of $(0,1)$, where the edge condition changes. In terms of the polar coordinate representation $x = -r\cos\theta$, $y = 1 - r\sin\theta$, a biharmonic function $w = r^{q+1}F(\theta)$ $(0 < \theta < \pi)$ is required such that $w = 0 = w_y$ at $\theta = 0$ and $\nu w_{xx} + w_{yy} = 0 = (2-\nu)w_{xxy} + w_{yyy}$ at $\theta = \pi$. Thus

$$F = A[\cos(q+1)\theta - \cos(q-1)\theta] + B\left[\frac{\sin(q+1)\theta}{q+1} - \frac{\sin(q-1)\theta}{q-1}\right],$$

where

$$2A\cos q\pi + (1+\nu)B\frac{\sin q\pi}{q-1} = 0, \qquad (1+\nu)(q-1)A\sin q\pi - 2B\cos q\pi = 0.$$

Hence the eigenvalue equation is

$$4\cos^2 q\pi + (1+\nu)^2\sin^2 q\pi = 0,$$

whose solutions are

$$q = \left(n - \frac{1}{2}\right) \pm iK, \qquad \tanh K\pi = \frac{1+\nu}{2},$$

for any integer n. But, for the displacement gradient to remain finite, $\mathrm{Re}(q) \geq 0$ and so the lowest admissible value of n is 1. Therefore, w is of order $r^{3/2}$ as $r = [x^2 + (1-y)^2]^{1/2} \to 0$.

12.3 Factorization of the kernel

12.3.1 Introduction and overview of the factorization procedure

In the previous section the matrix Wiener–Hopf equation was derived, in which the kernel, $\mathbf{K}(k)$, is written in (12.18). The aim of this section is to factorize $\mathbf{K}(k)$ into a product of two matrices

$$\mathbf{K}(k) = \mathbf{K}^-(k)\mathbf{K}^+(k), \tag{12.23}$$

one containing those singularities of $\mathbf{K}(k)$ lying in the lower half-plane, referred to as $\mathbf{K}^+(k)$, and $\mathbf{K}^-(k)$ which is analytic in the lower half-plane, $\mathcal{D}^-$, and hence contains the singularities of $\mathbf{K}(k)$ lying above the strip $\mathcal{D}$. Note that $[\mathbf{K}^+(k)]^{-1}$ and $[\mathbf{K}^-(k)]^{-1}$ are also analytic in the regions $\mathcal{D}^+$ and $\mathcal{D}^-$, respectively. Further, it is necessary for successful completion of the Wiener–Hopf procedure that $\mathbf{K}^\pm(k)$ are at worst of algebraic growth. Unfortunately, although matrix kernel factorization with the requisite growth behaviour has been proven to be possible for a wide class of kernels, to which the kernel (12.18) belongs, no constructive method has yet been found to complete this in general. There are classes of matrices for which product factorization can be achieved explicitly, the

most important of which are those amenable to Hurd's method and Khrapkov-Daniele commutative matrices The present problem yields a kernel which falls outside of the classes permitting an exact factorization and so an approximate decomposition will be performed here. Essentially, the procedure is to rearrange the kernel into an appropriate form, namely to resemble a Khrapkov (commutative) matrix, and then to replace a scalar component of it by a function which approximates it accurately in the strip of analyticity $\mathcal{D}$. The new approximate kernel is able to be factorized exactly (into an explicit non-commutative decomposition) and, in the previous cases cited above, strong numerical evidence was offered for convergence of the resulting approximate factors to the exact ones as the scalar approximant is increased in accuracy.

12.3.2 Conditioning of the matrix kernel

The matrix $\mathbf{K}(k)$ is characterized by its elements $e(k)$, $f(k)$, $g(k)$, given in (12.19)–(11.36), and in particular by their behavior for large and small k. Evidently,

$$e(k) \sim k(1+\nu), \qquad f(k) \sim 0, \qquad g(k) \sim 2|k| \text{ as } |k| \to \infty, k \in \mathcal{D} \tag{12.24}$$

and

$$f(k) \sim \frac{6}{k^2}, \qquad g(k) - f(k) \sim \frac{2}{3}k^2 f(k), \qquad e(k) \sim k f(k) \text{ as } k \to 0. \tag{12.25}$$

It is appropriate to arrange the kernel (12.18) so that the diagonal elements are equal to the same even function, and the off-diagonal terms are odd as $k \to \infty$. This is achieved by the rearrangement

$$\mathbf{K}(k) = \frac{1}{2}\begin{pmatrix} 0 & -k \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & -1 \\ i & i \end{pmatrix}\mathbf{L}(k)\begin{pmatrix} i & 1 \\ i & -1 \end{pmatrix}\begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}, \tag{12.26}$$

where $\mathbf{L}(k)$ takes the form

$$\mathbf{L}(k) = g(k)\mathbf{I} + \begin{pmatrix} 0 & f(k)+ie(k) \\ f(k)-ie(k) & 0 \end{pmatrix}, \tag{12.27}$$

with $\mathbf{I}$ the identity. The definition of $\mathbf{K}(k)$ in (12.18) easily establishes the subsequently useful properties

$$\mathbf{K}(-k) = \mathbf{K}(k) = [\mathbf{K}(k)]^T, \tag{12.28}$$

where T denotes the transpose.

Now $\mathbf{L}(k)$ is arranged in Khrapkov form, having the square of the second matrix term in (12.27) equal to a scalar polynomial in k times the identity. This is achieved by removing the factor $\sqrt{f^2(k)+e^2(k)}$ from this matrix in (12.27). However, as

$$f(k) \pm ie(k) \sim (1 \pm ik)f(k), \qquad k \to 0, \tag{12.29}$$

it is more effective to write $\mathbf{L}(k)$ as

$$\mathbf{L}(k) = g(k)\mathbf{I} + \sqrt{\frac{f^2(k)+e^2(k)}{1+k^2}}\mathbf{J}(k), \tag{12.30}$$

$$\mathbf{J}(k) = \begin{pmatrix} 0 & d(k)(1+ik) \\ d^{-1}(k)(1-ik) & 0 \end{pmatrix}, \tag{12.31}$$

in which

$$d(k) = \sqrt{\left(\frac{f(k)+ie(k)}{f(k)-ie(k)}\right)\left(\frac{1-ik}{1+ik}\right)}. \tag{12.32}$$

Evidently, a branch of $d(k)$ can be chosen to be regular in $\mathcal{D}$, equal to unity at $k=0$ and, because of (12.24), tend to unity at infinity in the strip. Thus the factor $(1-ik)/(1+ik)$ in $d(k)$ not only ensures an order k^2 deviation from unity but also causes $\arg[d(k)]$ to tend to zero as $k \to 0$ and $k \to \pm\infty$. Note that the function $d(k)$ has an infinite sequence of finite branch-cuts at symmetric locations in the upper and lower half-planes. The matrix $\mathbf{L}(k)$ now appears to be in Khrapkov form, because

$$\mathbf{J}^2(k) = \Delta^2(k)\mathbf{I}, \qquad \Delta^2(k) = 1+k^2 \tag{12.33}$$

but $\mathbf{J}(k)$ fails to be entire, having the infinite sequences of finite branch-cuts associated with $d(k)$. These will have to be considered once the partial Khrapkov decomposition is complete, but for the present will be ignored.

12.3.3 Partial decomposition of $\mathbf{K}(k)$

Before performing the Khrapkov factorization on $\mathbf{L}(k)$, it is necessary to note, from (12.25), (12.27), that as $k \to 0$,

$$|\mathbf{L}(k)| = g^2 - f^2 - e^2 \sim \frac{12}{k^2} + O(1), \tag{12.34}$$

which is singular at the origin, contrary to the original assumption of regularity in $\mathcal{D}$. This removable singularity, due to the use of $\mathbf{L}(k)$ instead of $\mathbf{K}(k)$, is conveniently handled by introducing the resolvent matrix, $\mathbf{R}(k)$, defined by

$$\mathbf{R}^{-1}(k) = (1+\frac{2}{3}k^2)\mathbf{I} - \mathbf{J}(k), \tag{12.35}$$

which commutes with $\mathbf{L}(k)$. The combined matrix

$$\mathbf{T}(k) = \mathbf{R}^{-1}(k)\mathbf{L}(k) \tag{12.36}$$

has determinant value 4 at $k=0$ which allows $\mathbf{T}(k)$ to be factorized instead of $\mathbf{L}(k)$. The subsequent factoring of $\mathbf{R}(k)$ will not pose any difficulty.

Since any matrices of the form $\alpha\mathbf{I} + \beta\mathbf{J}(k)$, like $\mathbf{R}^{-1}(k)$ and $\mathbf{T}(k)$, will commute, the product factors of $\mathbf{T}(k)$ may be posed in the form

$$\mathbf{T}^{\pm}(k) = r^{\pm}(k)\left(\cosh[\Delta(k)\theta^{\pm}(k)]\mathbf{I} + \frac{1}{\Delta(k)}\sinh[\Delta(k)\theta^{\pm}(k)]\mathbf{J}(k)\right), \tag{12.37}$$

where $r^{\pm}(k)$, $\theta^{\pm}(k)$ are scalar functions of k with the analyticity property indicated by their superscript. The function $\Delta(k)$, given by (12.33), generates no branch-cuts because

(12.37) contains only even powers of $\Delta(k)$. The scalar factors $r^{\pm}(k)$, $\theta^{\pm}(k)$ are deduced by equating

$$\mathbf{T}(k) = \mathbf{T}^{+}(k)\mathbf{T}^{-}(k), \tag{12.38}$$

which yields the decomposition discussed in Chapter 11. Hence $\mathbf{T}^{\pm}(k)$ have been determined (12.37, 11.40, 11.42) in a form which can be evaluated directly and these are analytic in their indicated half-planes, $\mathcal{D}^{\pm}$, except for the singularities occurring in $\mathbf{K}(k)$ (due to $d(k)$) which have yet to be resolved.

The inverse of $\mathbf{R}(k)$, introduced above in order to improve the convergence of $\mathbf{L}(k)$, was chosen in a form that now allows $\mathbf{R}(k)$ to be easily constructed and directly factorized. First, $\mathbf{R}^{-1}(k)$ may, by inspection, be written in the form

$$\frac{1}{2}[(1 + ik/\sqrt{3})\mathbf{I} - \mathbf{J}(k)][(1 - ik/\sqrt{3})\mathbf{I} - \mathbf{J}(k)], \tag{12.39}$$

where both matrices are entire save for the finite cuts in the scalar function $d(k)$ contained within $\mathbf{J}(k)$. It is then readily apparent that the partial decomposition of the resolvent matrix is

$$\mathbf{R}(k) = 2\mathbf{R}^{+}(k)\mathbf{R}^{-}(k), \tag{12.40}$$

where

$$\mathbf{R}^{\pm}(k) = \frac{3}{4k(k \pm i\sqrt{3}/2)}\left[(1 \mp ik/\sqrt{3})\mathbf{I} + \mathbf{J}(k)\right], \tag{12.41}$$

i.e. each matrix is analytic in its indicated half-plane except for poles at $k = 0$ and the finite branch-cuts in $d(k)$. Note that $\mathbf{R}^{\pm}(k)$ commute with each other and with $\mathbf{T}^{\pm}(k)$. Hence this completes the partial product factorization of $\mathbf{K}(k)$, namely, from (12.26, 12.36, 12.38, 12.40),

$$\mathbf{K}(k) = \mathbf{Q}^{-}(k)\mathbf{Q}^{+}(k), \tag{12.42}$$

where

$$\mathbf{Q}^{-}(k) = \begin{pmatrix} -ik & -ik \\ 1 & -1 \end{pmatrix}\mathbf{R}^{-}(k)\mathbf{T}^{-}(k), \tag{12.43}$$

$$\mathbf{Q}^{+}(k) = \mathbf{T}^{+}(k)\mathbf{R}^{+}(k)\begin{pmatrix} ik & 1 \\ ik & -1 \end{pmatrix}. \tag{12.44}$$

Note that $\mathbf{Q}^{\pm}(k)$ are without a pole singularity at $k = 0$ even though $\mathbf{R}^{\pm}(k)$ contain this singularity (verified later in section 3.4). All that remains is to (approximately) remove the residual singularities appearing in $\mathbf{J}(k)$.

12.3.4 Approximate factorization

There is no exact procedure known for eliminating the finite branch-cuts in $d(k)$ from the upper (lower) half-planes of the matrix factor $\mathbf{Q}^{+}(k)$ ($\mathbf{Q}^{-}(k)$). To obtain an approximate factorization the original matrix $\mathbf{K}(k)$ is replaced by a new one, $\mathbf{K}_N(k)$, where

$$\mathbf{K}_N(k) = \frac{1}{2}\begin{pmatrix} 0 & -k \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & -1 \\ i & i \end{pmatrix}\mathbf{L}_N(k)\begin{pmatrix} i & 1 \\ i & -1 \end{pmatrix}\begin{pmatrix} k & 0 \\ 0 & 1 \end{pmatrix}, \tag{12.45}$$

$$\mathbf{L}_N(k) = g(k)\mathbf{I} + \sqrt{\frac{f^2(k)+e^2(k)}{1+k^2}}\mathbf{J}_N(k) \tag{12.46}$$

and $\mathbf{J}_N(k)$ is as given in (12.31) but with a modified scalar $d(k) \to d_N(k)$, i.e.

$$\mathbf{J}_N(k) = \begin{pmatrix} 0 & d_N(k)(1+ik) \\ d_N^{-1}(k)(1-ik) & 0 \end{pmatrix}. \tag{12.47}$$

The scalar $d_N(k)$ is any function which approximates $d(k)$ accurately in the strip $\mathcal{D}$, and for efficacy of the following method it is most convenient to use a rational function approximation

$$d_N(k) = \frac{P_N(k)}{Q_N(k)}, \tag{12.48}$$

where $P_N(k)$, $Q_N(k)$ are polynomial functions of order N. Note that the order of each polynomial is the same as it is required that $d_N(k) \to 1$ as $|k| \to \infty$. There is a variety of ways of generating the coefficients of these polynomials, and the simplest and perhaps most justifiable (in terms of its analyticity properties) is to use Padé approximant. Care is required to ensure that $d_N(k)$ does not introduce spurious singularities into the strip of analyticity $\mathcal{D}$ and consequently produce an inaccurate factorization. One-point Padé approximants, if they exist, are determined uniquely from the Taylor series expansion of the original function at any point of regularity and accurately serve our current purposes because of the rapid decay at large real k that is present in the Fourier transforms (12.98) and (12.99). The polynomials $P_N(k)$ and $Q_N(k)$ are determined uniquely from $\sum_{n=0}^{\infty} f_n k^n - P_N(k)/Q_N(k) = O(k^{2N+1})$ where f_n are the coefficients of the Maclaurin series expansion of $d_N(k)$.

The definition (12.32) displays the symmetry, $d(-k) = 1/d(k)$, which must be reflected in the similar approximant behaviour:

$$d_N(-k) = 1/d_N(k). \tag{12.49}$$

Hence $Q_N(k) = P_N(-k)$ in (12.48), since the approximant is assumed to be in its lowest form. Then N must be even ($N = 2M$) for the limit value unity to be attained at the origin and at infinity in $\mathcal{D}$. Thus, for example,

$$d_4(k) = \frac{1 + \frac{13}{15}k^2 + \frac{1+\nu}{6}k^3(\frac{i}{2}+k) - \frac{197}{1575}k^4}{1 + \frac{13}{15}k^2 + \frac{1+\nu}{6}k^3(-\frac{i}{2}+k) - \frac{197}{1575}k^4}, \tag{12.50}$$

with errors $O(k^9)$ as $k \to 0$ and $O(k^{-1})$ as $|k| \to \infty$ in $\mathcal{D}$. After writing $[(f+ie)/(1+ik)]^{1/2}$ as a power series in ik, suitably truncated, an equivalent procedure that exploits the numerator/denominator symmetry in $d(k)$ is cross-multiplication. Then, for (12.50), only the terms in ik, ik^3, ik^5, ik^7 need be considered and suffice to determine the four coefficients in $d_4(k)$.

Note that the approximation of just $d(k)$ does not change the form of the scalar Khrapkov factors (11.40), (11.42), and so a partial decomposition of $\mathbf{K}_N(k)$ is simply

$$\mathbf{K}_N(k) = \mathbf{Q}_N^-(k)\mathbf{Q}_N^+(k) \tag{12.51}$$

in which $\mathbf{Q}_N^{\pm}(k)$ are given by (12.43), (12.44), with $\mathbf{R}^{\pm}(k)$, $\mathbf{T}^{\pm}(k)$ replaced respectively by $\mathbf{R}_N^{\pm}(k)$ and $\mathbf{T}_N^{\pm}(k)$, for which the subscript N denotes that $\mathbf{J}_N(k)$, given by (12.47), everywhere replaces $\mathbf{J}(k)$. Thus, the factorization of $\mathbf{K}_N(k)$ has been accomplished apart from sequences of poles, arising from the zeros and poles of $d_N(k)$ occurring in both half-planes exterior to $\mathcal{D}$. The removal of these singularities will then achieve an explicit exact factorization of $\mathbf{K}_N(k)$ which approximates the actual factors $\mathbf{K}^{\pm}(k)$ in their regions of analyticity.

The exact factorization of $\mathbf{K}_N(k)$, given by (12.45), may be written as

$$\mathbf{K}_N(k) = \mathbf{K}_N^-(k)\mathbf{K}_N^+(k), \tag{12.52}$$

$$\mathbf{K}_N^-(k) = \mathbf{Q}_N^-(k)\mathbf{M}(k), \qquad \mathbf{K}_N^+(k) = \mathbf{M}^{-1}(k)\mathbf{Q}_N^+(k) \tag{12.53}$$

in which $\mathbf{M}(k)$ must be a meromorphic matrix chosen to eliminate the poles of $\mathbf{Q}_N^-(k)$ in the lower half-plane and the poles of $\mathbf{Q}_N^+(k)$ in $\mathcal{D}^+$. A (non-unique) ansatz for $\mathbf{M}(k)$ can be posed after noting certain symmetry properties of $\mathbf{Q}_N^{\pm}(k)$. From (12.49),

$$\mathbf{J}_N(-k) = [\mathbf{J}_N(k)]^T, \tag{12.54}$$

where the superscript denotes the transpose, and so by inspection of (12.41),

$$\mathbf{R}_N^+(-k) = [\mathbf{R}_N^-(k)]^T. \tag{12.55}$$

Similarly, from (11.41), (11.43) and the obvious evenness of $\Delta(k)$ in (12.33), changing k to $-k$ in (12.37) reveals

$$\mathbf{T}_N^+(-k) = [\mathbf{T}_N^-(k)]^T. \tag{12.56}$$

Hence it is found (see (12.44)) that

$$\mathbf{Q}_N^+(-k) = [\mathbf{T}_N^-(k)]^T[\mathbf{R}_N^-(k)]^T \begin{pmatrix} -ik & -ik \\ 1 & -1 \end{pmatrix}^T = [\mathbf{Q}_N^-(k)]^T \tag{12.57}$$

and thus the second equation in (12.53) gives:

$$\mathbf{K}_N^+(-k) = \mathbf{M}^{-1}(-k)[\mathbf{Q}_N^-(k)]^T. \tag{12.58}$$

Symmetry properties dictate, by comparison with the first equation in (12.53), that a suitably scaled $\mathbf{M}(k)$ can be constructed so that $\mathbf{M}^{-1}(-k) = [\mathbf{M}(k)]^T$. After this is achieved, it suffices to eliminate poles of $\mathbf{K}_N^-(k)$ in the lower half plane.

Evidently, $d_{2M}(k)$ is a ratio of polynomials in (ik) with real coefficients so the zeros of $P_{2M}(k)$ are pure imaginary and/or pairs of the form $\pm k_1 + ik_2$. Moreover, the vanishing of the k term in $P_{2M}(k)$ implies that the sum of the inverses of the zeros is zero. Hence at least one zero lies in each (upper/lower) half-plane. Suppose now that $d_{2M}(k)$ has N_p $(0 < N_p < 2M)$ poles in the upper half-plane at $k = ip_n$, $n = 1, 2, \ldots, N_p$ ($ip_n \notin \mathcal{D}^-$) and $N_q (= 2M - N_p)$ poles in the region below the strip at $k = -iq_n$, $n = 1, 2, \ldots, N_q$. These are the zeros of $P_{2M}(-k)$, that is, $P_{2M}(k)$ has $2M (= N)$ simple zeros at

$$k = -ip_n, \quad n = 1, 2 \ldots, N_p; \qquad k = iq_n, \quad n = 1, 2 \ldots, N_q \tag{12.59}$$

in lower and upper regions respectively. Thus, $d_N(k)$ and its inverse may be expressed as Mittag-Leffler expansions:

$$d_{2M}(k) = 1 + \sum_{n=1}^{N_p} \frac{\alpha_n}{p_n + ik} + \sum_{n=1}^{N_q} \frac{\beta_n}{q_n - ik}, \tag{12.60}$$

$$\frac{1}{d_{2M}(k)} = 1 + \sum_{n=1}^{N_p} \frac{\alpha_n}{p_n - ik} + \sum_{n=1}^{N_q} \frac{\beta_n}{q_n + ik}. \tag{12.61}$$

The coefficients α_n, β_n are easily determined from the coefficients of the polynomial $P_{2M}(k)$. By inspection of the location of $d_N(k)$ in $\mathbf{Q}_N^-(k)$, the ansatz for $\mathbf{M}(k)$ is now posed

$$\begin{aligned} M_{11}(k) &= \frac{1}{\sqrt{2}} + \sum_{n=1}^{N_p} \frac{A_n}{p_n + ik} + \sum_{n=1}^{N_q} \frac{B_n}{q_n - ik}, \\ M_{21}(k) &= \frac{1}{\sqrt{2}} + \sum_{n=1}^{N_p} \frac{C_n}{p_n - ik} + \sum_{n=1}^{N_q} \frac{D_n}{q_n + ik}, \end{aligned} \tag{12.62}$$

and

$$\begin{aligned} M_{22}(k) &= \frac{1}{\sqrt{2}} + \sum_{n=1}^{N_p} \frac{E_n}{p_n - ik} + \sum_{n=1}^{N_q} \frac{F_n}{q_n + ik}, \\ -M_{12}(k) &= \frac{1}{\sqrt{2}} + \sum_{n=1}^{N_p} \frac{G_n}{p_n + ik} + \sum_{n=1}^{N_q} \frac{H_n}{q_n - ik}, \end{aligned} \tag{12.63}$$

where $A_n, B_n, C_n, D_n, E_n, F_n, G_n, H_n$ are as yet undetermined constants. The construction of $\mathbf{M}^{-1}(k)$ involves

$$|\mathbf{M}(k)| = M_{11}(k)M_{22}(k) - M_{12}(k)M_{21}(k), \tag{12.64}$$

whose poles are eliminated by satisfying

$$A_m M_{22}(ip_m) + G_m M_{21}(ip_m) = 0, \qquad M_{11}(-ip_m)E_m - M_{12}(-ip_m)C_m = 0 \qquad (1 \le m \le N_p),$$

$$B_m M_{22}(-iq_m) + H_m M_{21}(-iq_m) = 0, \qquad M_{11}(iq_m)F_m - M_{12}(iq_m)D_m = 0 \qquad (1 \le m \le N_q).$$

Comparison of (12.62) and (12.63) shows that this is achieved by setting

$$M_{22}(k) = M_{11}(-k), \qquad -M_{12}(k) = M_{21}(-k), \tag{12.65}$$

i.e., $E_n = A_n, G_n = C_n (1 \le n \le N_p)$, $F_n = B_n, H_n = D_n (1 \le n \le N_q)$, and requiring that

$$A_m M_{11}(-ip_m) + C_m M_{21}(ip_m) = 0 \qquad (1 \le m \le N_p), \tag{12.66}$$

$$B_m M_{11}(iq_m) + D_m M_{21}(-iq_m) = 0 \qquad (1 \le m \le N_q). \tag{12.67}$$

By pre-multiplying $\mathbf{M}(k)$ by $\mathbf{R}_N^-(k)\mathbf{T}_N^-(k)$, and eliminating poles in the lower half-plane, conditions relating the coefficients A_n, B_n, C_n, D_n can be found. From (12.37) and (12.41), it is known that

$$\left[(1+ik/\sqrt{3})\mathbf{I}+\mathbf{J}_N(k)\right]\left[\cosh[\Delta(k)\theta^-(k)]\mathbf{I}+\frac{1}{\Delta(k)}\sinh[\Delta(k)\theta^-(k)]\mathbf{J}_N(k)\right]\mathbf{M}(k)$$

must be analytic in $\mathcal{D}^-$ and so, according to (12.47), poles in the lower half-plane are to be removed from

$$\mathbf{W}(k)=\begin{pmatrix} a^-(k) & b^-(k)(1+ik)d_N(k) \\ b^-(k)(1-ik)/d_N(k) & a^-(k)\end{pmatrix}\mathbf{M}(k), \tag{12.68}$$

where

$$a^\pm(k) = (1\mp ik/\sqrt{3})\cosh[\Delta(k)\theta^\pm(k)]+\Delta(k)\sinh[\Delta(k)\theta^\pm(k)], \tag{12.69}$$

$$b^\pm(k) = \cosh[\Delta(k)\theta^\pm(k)]+\frac{1\mp ik/\sqrt{3}}{\Delta(k)}\sinh[\Delta(k)\theta^\pm(k)] \tag{12.70}$$

are scalar functions analytic in the indicated regions and such that

$$a^-(k)=a^+(-k), \qquad b^-(k)=b^+(-k). \tag{12.71}$$

From (12.68),

$$W_{11}(k)=a^-(k)M_{11}(k)+b^-(k)(1+ik)d_{2M}(k)M_{21}(k), \tag{12.72}$$

which contains simple poles in the lower half-plane only at $k=-iq_n$, $n=1,2,\ldots,N_q$ because, by comparison of (12.61) and (12.62), $d_{2M}(k)M_{21}(k)$ is regular at the poles of $M_{21}(k)$. The matrix element in (12.72) is forced to remain finite at $k=-iq_n$ by setting

$$a^-(-iq_m)B_m+b^-(-iq_m)\beta_m(1+q_m)M_{21}(-iq_m)=0, \quad m=1,\ldots,N_q. \tag{12.73}$$

Similarly, in (12.68), $W_{21}(k)$ contains no poles in the lower half-plane if and only if

$$a^-(-ip_m)C_m+b^-(-ip_m)\alpha_m(1-p_m)M_{11}(-ip_m)=0, \quad m=1,\ldots,N_p. \tag{12.74}$$

The second column of (12.68) yields similar equations when suppressing the remaining poles in the lower half-plane. Thus, after use of (12.65),

$$a^-(-ip_m)A_m-b^-(-ip_m)\alpha_m(1-p_m)M_{21}(ip_m)=0, \quad m=1,\ldots,N_p, \tag{12.75}$$

$$-a^-(-iq_m)D_m+b^-(-iq_m)\beta_m(1+q_m)M_{11}(iq_m)=0, \quad m=1,\ldots,N_q. \tag{12.76}$$

By inspection, (12.74), (12.75) imply (12.66) and (12.73), (12.76) imply (12.67). Then $|\mathbf{M}(k)|$ is entire and, since it takes the value unity at infinity, must by Liouville's theorem be given by

$$|\mathbf{M}(k)|=1. \tag{12.77}$$

Moreover, this ensures that $\mathbf{M}^{-1}(-k)=[\mathbf{M}(k)]^T$, as anticipated above, and that the inverses of $\mathbf{K}_N^\pm(k)$ are also free of singularities in $\mathcal{D}^\pm$ respectively. Confirmation that these

equations suffice to eliminate poles of $\mathbf{K}^+(k)$ in the upper-half plane is obtained by showing that $\mathbf{M}^{-1}(k)\mathbf{T}^+(k)\mathbf{R}^+(k) = [\mathbf{W}(-k)]^T$. The sets of equations (12.73)–(12.76) constitute a linear system of $2M$ equations for the $2M$ unknowns A_m, B_m, C_m, D_m and are easily solved to determine their values. Note that it may transpire that $1 + q_m$ or $1 - p_m$ is zero for particular choices of m, N etc. in which case B_m, D_m or A_m, C_m would vanish.

The explicit approximate factorization of $\mathbf{K}(k)$ is complete, having obtained an exact noncommutative matrix product decomposition of $\mathbf{K}_N(k)$. The factors $\mathbf{K}_N^{\pm}(k)$ are constructed from (12.53), with $\mathbf{Q}_N^{\pm}(k)$ given from (12.47), (12.43), (12.44), (12.41) and (12.37). The meromorphic matrix takes the explicit form (12.62) in which the coefficients satisfy algebraic equations (12.73)–(12.76). As $N = 2M$ increases it is expected that $\mathbf{K}_N^{\pm}(k)$ will converge rapidly to the exact factors $\mathbf{K}^{\pm}(k)$ and this will be borne out by numerical results given in section 5. All that remains here is to verify that the apparent pole at $k = 0$ in $\mathbf{R}^{\pm}(k)$ is removed and to give the behaviours of $\mathbf{K}_N^{\pm}(k)$ for large $|k|$ in $\mathcal{D}^{\pm}$.

As $k \to 0$, $d_N(k) \to 1$ by virtue of the function $d(k)$ in (12.32), and hence $\mathbf{R}_N^{\pm}(k)$ behaves as, from (12.31) and (12.41),

$$\mathbf{R}_N^{\pm}(k) = \frac{\mp i\sqrt{3}}{2k}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + O(1). \tag{12.78}$$

Therefore

$$\begin{pmatrix} -ik & -ik \\ 1 & -1 \end{pmatrix}\mathbf{R}_N^{-}(k) \sim \sqrt{3}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} + O(1), \quad k \to 0. \tag{12.79}$$

Now, $\mathbf{T}_N^{\pm}(k)$, from their definitions, are bounded at the origin and, by inspection, so also is $\mathbf{M}(k)$ in (12.62). Hence, from (12.53) and (12.79) it may be deduced that

$$\mathbf{K}_N^{-}(k) = O(1), \qquad k \to 0. \tag{12.80}$$

Similarly, from above,

$$\mathbf{R}_N^{+}(k)\begin{pmatrix} ik & 1 \\ ik & -1 \end{pmatrix} = O(1), \quad k \to 0 \tag{12.81}$$

and so $\mathbf{K}_N^{+}(0)$ is bounded too.

As $|k|$ tends to infinity it is a straightforward matter to deduce the asymptotic behaviour of the product factors. First, by inspection of (12.41),

$$\mathbf{R}_N^{\pm}(k) \sim \frac{3i}{4k}\begin{pmatrix} \mp 1/\sqrt{3} & 1 \\ -1 & \mp 1/\sqrt{3} \end{pmatrix} \tag{12.82}$$

in view of the fact that $d_N(k)$ is **defined** in $\mathbf{J}_N(k)$ to behave as

$$d_N(k) \to 1, \qquad |k| \to \infty. \tag{12.83}$$

Second, the asymptotic form of the Krapkhov decomposition elements $r^{\pm}(k)$ can be deduced from its integral definition written in (11.42), whose exponent has an integral that is $O(k^{-1})$ for large $|k|$. Hence, by inspection,

$$r^{\pm}(k) = [(3+\nu)(1-\nu)4/9]^{1/4} e^{\mp 3i\pi/4} k^{3/2} + O(k^{1/2}), \quad |k| \to \infty, k \in \mathcal{D}^{\pm}. \tag{12.84}$$

Third, the asymptotic form of the Krapkhov decomposition elements $\theta^{\pm}(k)$ cannot, due to $\mathbf{L}(k)$ not being diagonally dominant as $|k| \to \infty$ in $\mathcal{D}$, be similarly deduced from their integral definition written in (11.40) because the right-hand side of (11.39) is $O(1)$ for large $|k|$. Hence it is necessary to first subtract $\tanh^{-1}[(1+\nu)/2]$ before using the sum-split formula and then employ the standard decomposition

$$\frac{1}{\Delta(k)} = \frac{2}{\pi}(1+k^2)^{-1/2}\arctan\left(\frac{i-k}{i+k}\right)^{1/2} + \frac{2}{\pi}(1+k^2)^{-1/2}\arctan\left(\frac{i+k}{i-k}\right)^{1/2}, \quad (12.85)$$

in which

$$2\arctan\left(\frac{i-k}{i+k}\right)^{1/2} = \frac{\pi}{2} + i\ln[\sqrt{1+k^2}+k],$$

to show that

$$\theta^{\pm}(k) \sim \pm\left[\ln\left(\frac{3+\nu}{1-\nu}\right)\frac{i\ln|k|}{2\pi k} + \frac{\chi}{k}\right], \qquad |k| \to \infty, k \in \mathcal{D}^{\pm}, \quad (12.86)$$

where

$$\chi = \frac{i}{\pi}\int_0^{\infty}\left[\tanh^{-1}\left\{\frac{(1+\frac{2}{3}\zeta^2)\sqrt{f^2(\zeta)+e^2(\zeta)} - \Delta(\zeta)g(\zeta)}{(1+\frac{2}{3}\zeta^2)g(\zeta) - \Delta(\zeta)\sqrt{f^2(\zeta)+e^2(\zeta)}}\right\} - \tanh^{-1}\frac{1+\nu}{2}\right]\frac{d\zeta}{\Delta(\zeta)}. \quad (12.87)$$

Thus, from (12.37), $\mathbf{T}_N^{\pm}(k) = O(k^{3/2})$. Last, the meromorphic matrix (12.62) has the large $|k|$ form

$$\mathbf{M}(k) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \quad (12.88)$$

and therefore $\mathbf{Q}_N^{\pm}(k)$ in (12.43), (12.44) can be estimated. Finally, the asymptotic growth of $\mathbf{K}_N^{\pm}(k)$ in (12.53) is found to be:

$$\begin{aligned} \mathbf{K}_N^{-}(k) &\sim k^{1/2}\begin{pmatrix} -ik & -ik \\ 1 & -1 \end{pmatrix}\begin{pmatrix} O(1) & O(1) \\ O(1) & O(1) \end{pmatrix}, \\ \mathbf{K}_N^{+}(k) &\sim \begin{pmatrix} O(1) & O(1) \\ O(1) & O(1) \end{pmatrix} k^{1/2}\begin{pmatrix} ik & 1 \\ ik & -1 \end{pmatrix}. \end{aligned} \quad (12.89)$$

Hence

$$[\mathbf{K}_N^{-}(k)]^{-1} \sim \begin{pmatrix} O(1) & O(1) \\ O(1) & O(1) \end{pmatrix} k^{-1/2}\begin{pmatrix} ik^{-1} & 1 \\ ik^{-1} & -1 \end{pmatrix} \quad (12.90)$$

and the kernel decomposition is now complete.

12.4 Solution of the Wiener–Hopf equation

Having obtained an approximate factorization of $\mathbf{K}(k)$ it is now a straightforward matter to complete the solution of the Wiener–Hopf equation (12.17). Dropping the subscript N

in the factorization (12.52) for brevity, the Wiener–Hopf equation can be recast into the form

$$[\mathbf{K}^-(k)]^{-1}\begin{pmatrix} -\Phi^-_{yyy}(k,1) \\ \Phi^-_{yy}(k,1)\end{pmatrix} - \frac{i}{2(k+i\epsilon)}\left\{[\mathbf{K}^-(k)]^{-1} - [\mathbf{K}^-(-i\epsilon)]^{-1}\right\}\begin{pmatrix} -1 \\ 1/6\end{pmatrix} =$$
$$\mathbf{E}(k) = \mathbf{K}^+(k)\begin{pmatrix} \Phi^+(k,1) \\ \Phi^+_y(k,1)\end{pmatrix} + \frac{i}{2(k+i\epsilon)}[\mathbf{K}^-(-i\epsilon)]^{-1}\begin{pmatrix} -1 \\ 1/6\end{pmatrix}, \tag{12.91}$$

where $k \in \mathcal{D}$. The left-hand side is analytic in $\mathcal{D}^-$, whereas the right-hand side is regular in $\mathcal{D}^+$. Thus, the equation has been arranged so that the two sides offer analytic continuation into the whole complex k-plane which must therefore be equal to an entire function, say $\mathbf{E}(k)$, that is determined by examining the growth at infinity of both sides of (12.91) in their respective half-planes of analyticity. This requires the large k behaviour of $\Phi^+(k,1)$, $\Phi^+_y(k,1)$, $\Phi^-_{yy}(k,1)$, $\Phi^-_{yyy}(k,1)$, which relate directly to the values of the untransformed physical variables near to $(0,1)$, where the edge condition changes. For example, a function which behaves like x^n, $x \to 0+$, has a half-range (0 to ∞) Fourier transform which decays like $O(k^{-n-1})$, $k \to \infty$ in the upper half plane. Hence $w = O(r^{3/2})$ as $r = [x^2 + (1-y)^2]^{1/2} \to 0$ implies that

$$\Phi^+(k,1) = O(k^{-5/2}), \quad \Phi^+_y(k,1) = O(k^{-3/2}) \tag{12.92}$$

as $|k| \to \infty$, $k \in \mathcal{D}^+$ and

$$\Phi^-_{yy}(k,1) = O(k^{-1/2}), \quad \Phi^-_{yyy}(k,1) = O(k^{1/2}), \quad |k| \to \infty, k \in \mathcal{D}^-. \tag{12.93}$$

These are used, together with the asymptotic forms (12.89), (12.90) to reveal that both elements of the left-hand side of (12.91) decay as $O(k^{-1})$ in the lower half plane and similarly the right-hand side has the form $o(1)$ as $|k| \to \infty$ in the upper half plane. Hence, $\mathbf{E}(k)$ is an entire function which decays to zero at infinity, and so, by Liouville's theorem,

$$\mathbf{E}(k) \equiv 0. \tag{12.94}$$

Thus, the solution of the Wiener–Hopf equation is

$$\begin{pmatrix} -\Phi^-_{yyy}(k,1) \\ \Phi^-_{yy}(k,1)\end{pmatrix} = \frac{i}{2(k+i\epsilon)}\left\{\mathbf{I} - \mathbf{K}^-(k)[\mathbf{K}^-(-i\epsilon)]^{-1}\right\}\begin{pmatrix} -1 \\ 1/6\end{pmatrix} \tag{12.95}$$

or, equivalently,

$$\begin{pmatrix} \Phi^+(k,1) \\ \Phi^+_y(k,1)\end{pmatrix} = -\frac{i}{2(k+i\epsilon)}[\mathbf{K}^+(k)]^{-1}[\mathbf{K}^-(-i\epsilon)]^{-1}\begin{pmatrix} -1 \\ 1/6\end{pmatrix}. \tag{12.96}$$

From this, the coefficients $A(k), B(k)$ are readily deduced from (12.16) and hence $\Phi(k,y)$ is established for all y, $0 < y < 1$, from (12.15). Finally, on setting the convergence factor ϵ equal to zero in (12.95), the additional displacement due to the semi-infinite free edge is

$$\frac{q}{D}W = \frac{q}{2\pi D}\int_{-\infty}^{\infty}\Phi(k,y)e^{-ikx}dk, \tag{12.97}$$

where the integral path runs along the real line indented above the origin, and

$$\Phi(k,y) = \frac{-i}{2(\sinh^2 k - k^2)} \begin{pmatrix} y\sinh ky \\ y\cosh ky - k^{-1}\sinh ky \end{pmatrix}^T \times$$

$$\begin{pmatrix} k\sinh k & -(\cosh k - k^{-1}\sinh k) \\ -(k\cosh k + \sinh k) & \sinh k \end{pmatrix} [\mathbf{K}^+(k)]^{-1}[\mathbf{K}^-(0)]^{-1} \begin{pmatrix} -1 \\ 1/6 \end{pmatrix}. \tag{12.98}$$

It is a straightforward matter to verify that, when this solution is substituted into (12.7), the total displacement satisfies the fourth-order equation (12.1) and the boundary conditions (12.2), (12.3). Moreover, W evidently has the structure predicted by (12.6) for $x < 0$, in which case the contour in (12.97) is completed in $\mathcal{D}^+$ and residues at the zeros of $\sinh^2 k - k^2$ are obtained. An alternative form for (12.98) is obtained by replacing $[\mathbf{K}^+(k)]^{-1}$ by $[\mathbf{K}(k)]^{-1}\mathbf{K}^-(k)$, according to (12.23), and substituting (12.18) and (12.22), whence

$$\Phi(k,y) = \frac{-i}{2k^2[4 + (3+\nu)(1-\nu)\sinh^2 k + (1-\nu)^2k^2]} \begin{pmatrix} y\sinh ky \\ y\cosh ky - k^{-1}\sinh ky \end{pmatrix}^T$$

$$\times \begin{pmatrix} [(1-\nu)\cosh k + (1+\nu)k^{-1}\sinh k] & -[(1-\nu)k\sinh k - 2\cosh k] \\ -[(1-\nu)\sinh k + 2k^{-1}\cosh k] & [(1-\nu)k\cosh k - (1+\nu)\sinh k] \end{pmatrix}$$

$$\times \mathbf{K}^-(k)[\mathbf{K}^-(0)]^{-1} \begin{pmatrix} -1 \\ 1/6 \end{pmatrix}. \tag{12.99}$$

Then, for $x > 0$, completion in $\mathcal{D}^-$ of the contour in (12.97) yields $qy^2(5-2y)/24D$ from the pole at the origin and residues at the zeros of $[4 + (3+\nu)(1-\nu)\sinh^2 k + (1-\nu)^2k^2]$, which include a pair on the imaginary axis determined by $k = i\kappa, 4 - (1-\nu)^2\kappa^2 = (3+\nu)(1-\nu)\sin^2\kappa$. Thus the structure of W predicted by (12.6) for $x < 0$ is demonstrated.

12.5 Numerical computations

The calculations are carried out with MATLAB, except for the evaluation of the Padé approximants d_{2M}, which are obtained using Maple. The resulting fractions are then converted to floating points (16 digit accuracy) and returned to MATLAB. Accuracy is low unless $N = 2M$ with M even, to take account of symmetries about both axes. Maximum accuracy occurs at about $N = 2M = 16$. This could be increased by using variable precision arithmetic. The poles $ip_m, -iq_m$ and coefficients α_m, β_m in (12.60) are then readily determined, followed by $\mathbf{J}_N(k)$ and $\mathbf{R}_N^+(k)$. Evaluation of $r_N^+(k), \theta_N^+(k)$ yields $\mathbf{T}_N^+(k)$ and hence $\mathbf{Q}_N^+(k)$, given by (12.44). The scalar Wiener–Hopf decomposition of r and θ is accomplished using standard MATLAB numerical integration. The default relative accuracy of 10^{-6} was amply sufficient, since as in many contour integrals of analytic functions, higher accuracy was actually obtained. Finally $\mathbf{K}_N^+(k)$ is constructed. The displacement, $W(x,y)$, is evaluated as a sum of the residues of (12.98) over the first 50 poles in the upper half-plane for $x < 0$. Note that once the residues are known, the inverse Fourier transform

(12.97) can be computed for different x for essentially no further cost. The companion matrix function $\mathbf{K}_N^-(k)$, needed in (12.99) for $x > 0$, is constructed similarly. The oscillatory case is algebraically more complicated but structurally the same.

The numerical evaluation of the approximation $[\mathbf{K}_N^+(k)]^{-1}$ to $[\mathbf{K}^+(k)]^{-1}$ in (12.98) depends on the accurate determination of the coefficients $a^+(ip_m), \ldots$ in the sets of equations (12.73)–(12.76). These are given by (12.69), (12.70), in which $\Delta(k)$ appears analytically but branch 5Bcuts may arise from the presence of $\theta^{\pm}(k)$ in the sinh functions. By factoring $\Delta(\zeta)$ from the numerator of the fraction in (11.40), it is evident that the branch cuts created by the approximate factorization arise solely from the square root in the definition (12.30) of $\mathbf{L}(k)$.

Very high accuracy would require variable precision arithmetic and a large number of terms in the residue sum, especially when computing the deflection near the transition between the free and clamped edges.

Values of ν for naturally occurring materials are constrained thermodynamically to lie between 0 and $1/2$. Examples are cork ($\nu = 0$), concrete ($\nu = 0.2$), stainless steel ($\nu = 0.3$) and rubber ($\nu = 0.5$). A typical example of a Padé approximant for the steady $\nu = 1/2$ case is shown in Figure 12.1 for real values of k. The behaviour near the origin of the error is of the appropriate power and the error decreases for large k. Note that the relative error decreases slowly for large k, but this is unimportant. The sum over the residues at the zeroes and poles of $|\mathbf{K}|$ requires evaluating the Padé approximant off the real axis. The curves analogous to those of Figure 12.1 become very confusing close to branch cuts, where the smooth, single-valued Padé approximant lacks the significant jumps in d near its artificially created cuts. Nevertheless, the computed residues approach the exact ones. The behaviour for other values of ν and in the oscillatory case is qualitatively the same.

Figure 12.2 shows the plate displacement for the steady cases with $\nu = 0.3$ for $N = 16$. The difference between these results and those for ν in the range $(0, 1/2)$ are small and so other results are not shown. The relative difference in W between $\nu = 0$ and $\nu = 1/2$, except on the boundaries, are 15% or smaller and decay very rapidly for $x < 0$. The surface has an imperceptible defect at $x = 0$: the values of $W(0, y)$ computed using (12.98) and (12.99) are not exactly the same in the Padé approximant technique, but would be the same for the exact matrix $\mathbf{K}$. This discrepancy provides an estimate of the error, which is found to decrease with N until at least $N = 16$. The plate is deformed substantially in the region $0 < x < 1$ and then attains its asymptotic value smoothly.

12.6 Conclusion

The Padé approximant technique for matrix Wiener–Hopf equations yields accurate numerical results for a classic plate deflection problem, whether the forcing be static or dynamic. The theory is complicated by the need for successive modifications $\mathbf{L}$, $\mathbf{T}$ of the kernel and $\mathbf{M}$, $\mathbf{M}^{-1}$ of the matrix factors $\mathbf{Q}^-$, $\mathbf{Q}^+$ in order to achieve the required analyticity of $\mathbf{K}^-$ and $\mathbf{K}^+$. The numerical implementation is conceptually straightforward, but, like many problems in numerical complex analysis, requires careful attention to the analyticity properties of the functions being used. The technique provides a constructive scheme to obtain the physical solution without the difficulties involved in matching the

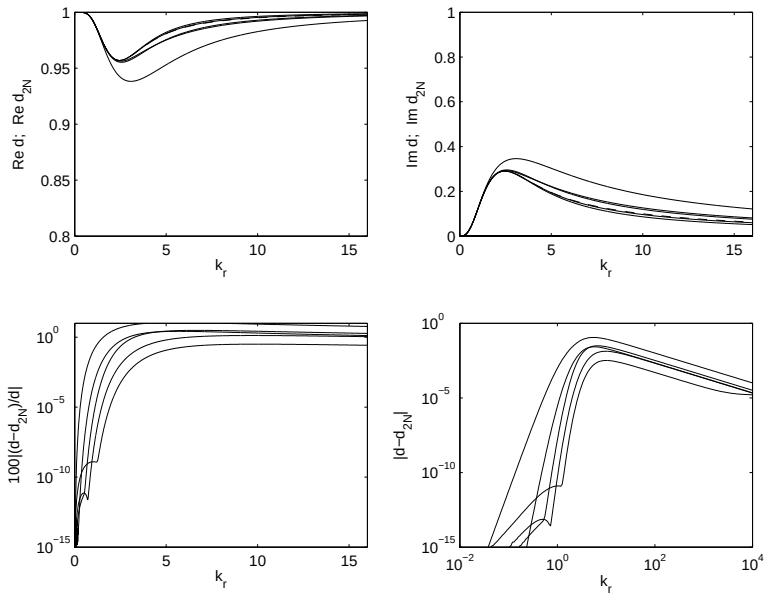

Figure 12.1: Padé approximant to $d(k)$ (12.32) in the steady case with $\nu = 1/2$ for $N = 4$, 8, 12, 16. Curves with larger N cluster together. (a) Real and (b) imaginary parts. (c) Percentage relative error in modulus on a logarithmic scale. (d) Absolute error.

biorthogonal Papkovich–Fadle eigenfunctions.

Physically, the reason that the displacement is insensitive to the value of ν appears to be that the imposed load and the clamped boundary condition do not give much of a role to ν, which relates compression in one direction to expansion/compression in the other, at least in the steady case. Of course the actual dependence on ν of the displacement can only be determined by carrying out the calculation. Unfortunately, there doesn't appear to be a special value of ν for which everything can be solved by hand. This suggests that for similar problems, a single value of ν can be used to obtain typical results.

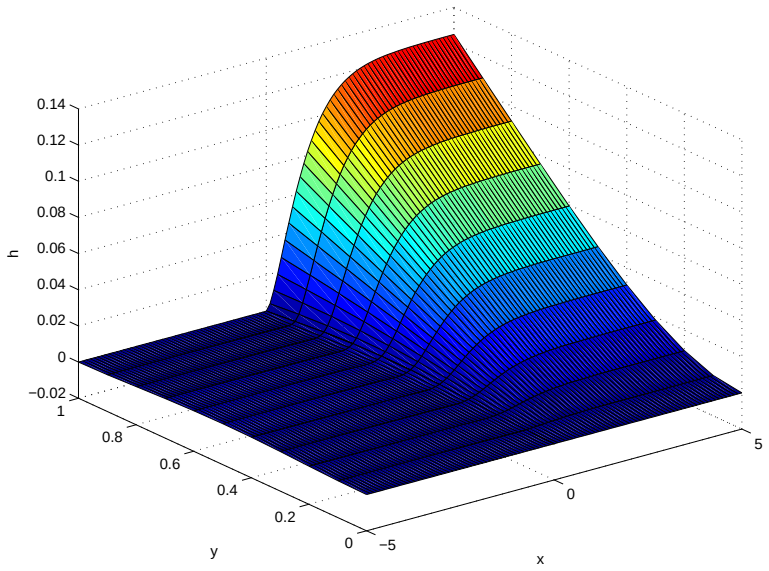

Figure 12.2: Displacement $W(x, y)$ for $\nu = 0.3$ with $N = 16$.

Chapter 13

Integral equations (05/13/08)

13.1 Types of equations

We have already met linear integral equations, Fredholm

$$f(x) = \lambda \int_0^1 k(x,t) f(t)\,\mathrm{d}t + g(x) \tag{13.1}$$

and Volterra

$$g(x) = \lambda \int_0^x k(x,t) f(t)\,\mathrm{d}t, \tag{13.2}$$

where $f(t)$ is an unknown function, to be found, in both cases. The above are equations of the second and first kind (F2 and V1 respectively).

The Wiener–Hopf technique of the previous two chapters can also be phrased in terms of coupled integral equations on semi-infinite intervals. Now seems like a good time to return to integral equations. Let's solve some that are not in convolution form, and then turn to integrals with singular kernels, i.e. in which the integral is a principal value integral. These provide beautiful applications of complex variable ideas.

Solving integral equations numerically is treated quite well in *Numerical Recipes*, both for Volterra and Fredholm equations. Be careful though: the midpoint? method suggested for V2 is said to work fine for V1, but a look at Linz (1985) will show that this is incorrect.

13.2 Non-singular equations

We can show very simply how V2 reduces to a system of linear equations. Remember that V2 is

$$f(x) = \lambda \int_0^x k(x,t) f(t)\,\mathrm{d}t + g(x). \tag{13.3}$$

Discretize the interval into segments with endpoints 0, h, $2h$, $\ldots x$, with $x = Nh$. This implies it's easiest to increase x in units of h to avoid fractional segments. Now evaluate

(13.3) at the point $(m+\frac{1}{2})h$, discretizing the integral by the midpoint rule:

$$f_m = \lambda \sum_{n=0}^{m} \tfrac{1}{2} k_{mn} f_n + g_m, \tag{13.4}$$

where $f_m = f((m+\frac{1}{2})h)$, $k_{mn} = k(((m+\frac{1}{2})h, (n+\frac{1}{2})h)$ and $g_m = g((m+\frac{1}{2})h)$. This is now a lower-diagonal set of linear equations that can be solved by back-substitution.

One can use the trapezium rule instead. The result is almost as simple, but one has to be slightly more careful. The same method clearly makes sense for V1 and Fredholm equations. For the latter case, the linear system is no longer lower-diagonal, but this is no impediment. For the former, there is a technical problem.

More sophisticated methods exist, including means for dealing with integrable singularities in the kernel. These include integrating out the singularity or using more sophisticated quadrature rules. This leads onto the topic of Gaussian integration, which we may see again briefly when talking about orthogonal polynomials in Chapter 19.

Another improvement for V1 is the algorithm described by Linz (1985) which lends itself well to Richardson extrapolation (known as the deferred approach to the limit in the old days). It is just based on the ... quadrature rule, which has known error $h^2/12$ (remember this always includes an unknown prefactor depending on the exact function being integrated, but independent of h). Figure 13.1 shows the results of this method applied to the following test problems in Linz (1985):

$$\int_0^t \cos(t-s) f(s)\,\mathrm{d}s = 1 - \cos t, \tag{13.5}$$

$$\int_0^t [1 + 2(t-s)] f(s)\,\mathrm{d}s = t, \tag{13.6}$$

$$\int_0^t \begin{pmatrix} \cos(t-s) & 0 & 0 \\ 0 & 1+2(t-s) & 0 \\ 1 & \mathrm{e}^{-(t-s)} & 1 \end{pmatrix} f(s)\,\mathrm{d}s = \begin{pmatrix} 1-\cos t \\ t \\ \frac{1}{2}t^2 \end{pmatrix}. \tag{13.7}$$

Note that the third equation is a matrix V1 equation with the first two lines reproducing the first two problems.

The routine `p131.m` includes specific provision for convolution kernels, which should lead to a slight gain in efficiency, as well as for Richardson extrapolation. The plots of Figure 13.1 clearly show the $O(h^2)$ convergence for the bare algorithm, compared to $O(h^4)$ convergence for the algorithm with extrapolation.

Tip 6 *Richardson extrapolation. The idea is that you have a numerical approximation whose error you know. Say it goes like $h^2/12$. Compute f_1, the approximation with step size h. Then if you carry out the procedure again with a step size of $h/2$, the error for $f_{1/2}$ is $h^2/48$. Now compute $f_1 - 4f_2$: the $h^2/12$ term in the error cancels and you are left with something smaller. The result: the order of the error has gone down without having to come up with a new method and with only one extra calculation. This is gold. Use it when you can.*

```
% p131.m Linz algorithm SGLS 05/12/08
```

```
function p131
hh = 10.^(0:-0.5:-2);
e = zeros(length(hh),4);

subplot(2,2,1)
for j=1:length(hh)
  hh(j)
  [t,y] = volt1m(@k91,@g91,10,hh(j),0,0);
  e(j,1:2) = [norm(y-t) norm(y-t,inf)];
  [t,y] = volt1m(@k91,@g91,10,hh(j),1,0);
  e(j,3:4) = [norm(y-t) norm(y-t,inf)];
end
loglog(hh,e,'.',hh,hh.^2,hh,hh.^4/24)
xlabel('h'); ylabel('e')

subplot(2,2,2)
for j=1:length(hh)
  hh(j)
  [t,y] = volt1m(@k91c,@g91,10,hh(j),0,1);
  e(j,1:2) = [norm(y-t) norm(y-t,inf)];
  [t,y] = volt1m(@k91c,@g91,10,hh(j),1,1);
  e(j,3:4) = [norm(y-t) norm(y-t,inf)];
end
loglog(hh,e,'.',hh,hh.^2,hh,hh.^4/24)
xlabel('h'); ylabel('e')

subplot(2,2,3)
for j=1:length(hh)
  hh(j)
  [t,y] = volt1m(@k932,@g932,10,hh(j),0,0);
  e(j,1:2) = [norm(y-exp(-2*t)) norm(y-exp(-2*t),inf)];
  [t,y] = volt1m(@k932,@g932,10,hh(j),1,0);
  e(j,3:4) = [norm(y-exp(-2*t)) norm(y-exp(-2*t),inf)];
end
loglog(hh,e,'.',hh,hh.^2,hh,hh.^4/24)
xlabel('h'); ylabel('e')

subplot(2,2,4)
for j=1:length(hh)
  hh(j)
  [t,y] = volt1m(@ktest,@gtest,10,hh(j),0,0);
  y = y(:,3); yex = exp(-t)-2*exp(-2*t);
  e(j,1:2) = [norm(y-yex) norm(y-yex,inf)];
  [t,y] = volt1m(@ktest,@gtest,10,hh(j),1,0);
  y = y(:,3); yex = exp(-t)-2*exp(-2*t);
```

```
  e(j,3:4) = [norm(y-yex) norm(y-yex,inf)];
end
loglog(hh,e,'.',hh,hh.^2,hh,hh.^4/24)
xlabel('h'); ylabel('e')

function [t,y] = volt1m(k,g,tmax,h,Ri,Co)

if (Ri==0)
  t = h:h:tmax+0.5*h;
  N = length(t);
  for n = 1:N
    r = g(t(n))/h;
    if (n==1)
      m = length(r);
      y = zeros(m,N);
      if (Co==1)
        K = zeros(m,m,N);
      end
    end
    if (Co==1)
      K(:,:,n) = k((n-0.5)*h);
      for j = 1:n-1
        r = r - K(:,:,n-j+1)*y(:,j);
      end
      y(:,n) = K(:,:,1)\r;
    else
      for j = 1:n-1
        K = k(t(n),t(j)-0.5*h);
        r = r - K*y(:,j);
      end
      K = k(t(n),t(n)-0.5*h);
      y(:,n) = K\r;
    end
  end
  t = (t-0.5*h)';
  y = y.';
else
  [t,y1] = volt1m(k,g,tmax,h,0,Co);
  [t2,y2] = volt1m(k,g,tmax,h/3,0,Co);
  y = y2(2:3:end,:) + 1/8*(y2(2:3:end,:)-y1);
end

function k = k91(t,s)

k = cos(t-s);
```

```
function k = k91c(t)

k = cos(t);

function g = g91(t)

g = 1 - cos(t);

function k = k932(t,s)

k = 1 + 2*(t-s);

function g = g932(t)

g = t;

function k = ktest(t,s)

k = [cos(t-s) 0 0 ; 0 1+2*(t-s) 0 ; 1 exp(-(t-s)) 1];

function g = gtest(t)

g = [1-cos(t) ; t ; 0.5*t^2];
```

Exercise 13.1 *Solve (13.5) and (13.6) using Laplace transforms and Talbot's algorithm from (8).*

13.3 Singular integral equations: Tuck's method

There is a crucial difference between equations of the first and second kind. The first tend to be ill-posed.

Definition 13.1 *A problem is well-posed if its solution is uniquely determined and not arbitrarily sensitive to small changes in e.g. boundary conditions.*

There is a physical reason why equations of the first kind tend to be ill-posed: integration is a smoothing operation. So in (13.2) we could add to $f(x)$ some wildly oscillating function and the oscillations would be wiped out by the integral. Hence there may be other solutions to the equation very close to the correct solution. In equations of the first kind, the presence of the unknown function $f(x)$ outside an integral removes this problem.

Returning to the linear algebra perspective, the ill-posedness is reflected by an increasingly worse condition number of the resulting matrix. The condition number measures

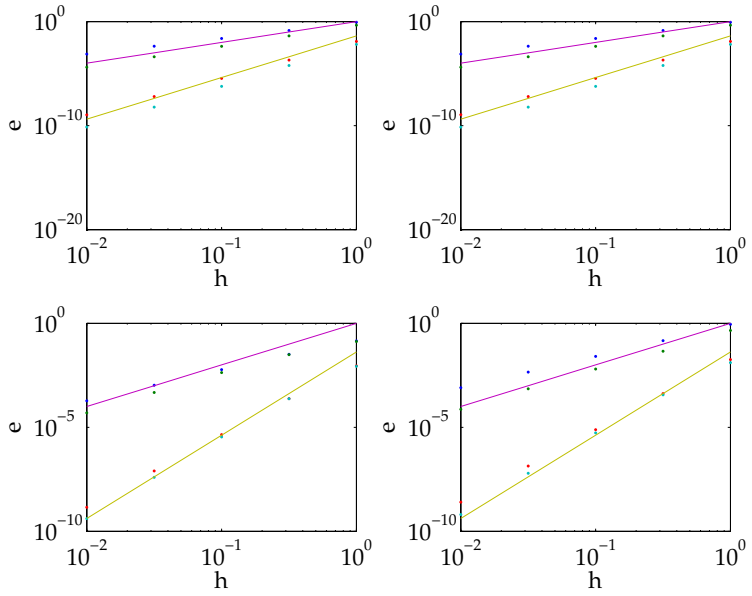

Figure 13.1: Linz algorithm applied to test cases.

the spread in eigenvalues. If it is infinite, there is a zero eigenvalue and the matrix is no longer invertible.

However, as the kernel inside the integral becomes more singular, it smooths less and less, so this problem goes away. The most extreme limit corresponds to $k(s,t) = \delta(s-t)$; then the integral just becomes $f(x)$ and the answer is trivial. We also saw that the Abel integral equation of Chapter 8 had a unique solution, when it existed. As a result, singular integral equations are more likely to have unique solutions than integral equations with smooth kernels.However, we have to be careful when discretizing them so as to deal with the singularities appropriately.

One important class of singular integral equations corresponds to kernels with $(s-t)^{-1}$ singularities. Then the integral itself must be viewed as a principal value integral. Such integral equations arise very naturally in the solution of Laplace's equation, and this leads to the field of Boundary Integral Methods (for a review of numerical methods see e.g. Pozrikidis 2002). We will limit ourselves to a simple discussion based on Tuck (1980).

We can base pretty much everything on what is known as the airfoil equation[1]

$$\frac{1}{\pi} \int_{-1}^{1} \frac{f(t)}{t-x}\, \mathrm{d}t = g(x) \tag{13.8}$$

(notice the change in notation). This equation has the critical property that its solution is not unique. We can see this by computing

$$\int_{-1}^{1} \frac{(1-t^2)^{-1/2}}{t-x}\, \mathrm{d}t = 0. \tag{13.9}$$

[1]Richard Courant is said to have called this the most important piece of mathematics of the 20th century, in the sense that it was an application of pure mathematics to a very concrete problem. I may be misquoting.

Physically this corresponds to needing to fix the circulation about a contour to define the irrotational flow about the contour uniquely. We hence have to supplement (13.8) with some extra condition, usually $\int_{-1}^{1} f(t)\,\mathrm{d}t = 0$ or a requirement of finiteness at one of the endpoints.

Warning 12 *Integral equations may have unique solutions, or at least, a finite number of arbitrary constants (one for the airfoil equations). However, the solutions often have square root singularities at the endpoints. These integrable singularities are usually physically meaningful but require care in the mathematical procedure.*

Exercise 13.2 *Prove (13.9).*

Tuck (1908) suggests integrating the airfoil equation to obtain an integral equation with a logarithmic kernel:

$$\frac{1}{\pi}\int_{-1}^{1} \log|t-x| f(t)\,\mathrm{d}t = h(x). \tag{13.10}$$

One can show that this equation has a unique solution. Of course, $h(x)$ is no longer unique and fixing it corresponds to removing the non-uniqueness as mentioned previously. Figure 13.2 shows the solution to (13.8) with zero right-hand side. This is hence the homogeneous solution, so an extra condition has to be applied. The right-hand side of $h(x)$ is taken to be 1, which leads to the exact solution $\log 2/\pi\sqrt{1-x^2}$. The numerical procedure performs very well, even though the solution is singular at both endpoints.

The details of the procedure, briefly, are as follows. The unknown function $f(x)$ is pulled out of the integral and evaluated at the midpoints of the intervals. The resulting logarithmic integral is computed explicitly. Hence this is a modified midpoint technique. For increase accuracy near the endpoints, a stretched (Chebyshev) mesh is taken. The extension to different $g(x)$ is trivial, although some thought is required with respect to the constant of integration in $h(x)$. More complicated kernels are possible, but they need to be integrable. Subtracting out the logarithmic singularity is the way to go. For an example, see Llewellyn Smith & Young (2003).

```
% p132.m Fredholm singular integral equation SGLS 05/12/08

l=4; M=40;
xi=-l*cos(pi*(0:M)/M);
x=(xi(1:end-1)+xi(2:end))/2;
x=x';
hp=ones(M,1)*xi(2:end)-x*ones(1,M);
hm=ones(M,1)*xi(1:end-1)-x*ones(1,M);
K=hp.*(log(abs(hp))-1)-hm.*(log(abs(hm))-1);
dxi=ones(M,1)*(xi(2:end)-xi(1:end-1));
g=ones(M,1);
gamma=K\g;
plot(x,gamma,x,1/pi/log(l/2)./sqrt(l^2-x.^2),'.',[-l l],[1 1])
xlabel('x'); ylabel('\gamma')
```

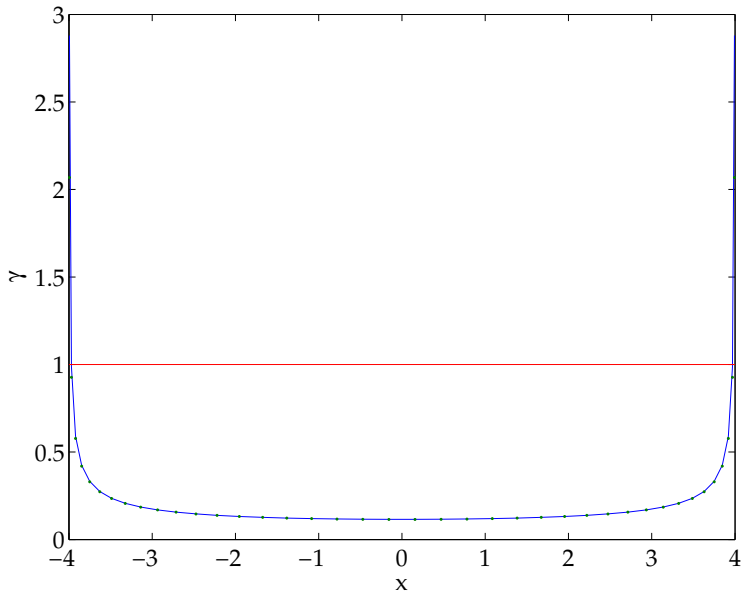

Figure 13.2: Tuck's method applied to the airfoil equation (13.8) with $g = 0$.

Exercise 13.3 *What condition was used to make the solution plotted in Figure 13.2 unique?*

Exercise 13.4 *Solve (13.8) with the right-hand sides* $\cos x$ *and* $(1 - x^2)^{-1/2}$.

13.4 Singular integral equations: complex variable approaches

Remarkably, some singular integral equations can be solved exactly. The exact details are too long to enter into here, but an excellent discussion can be found in Pipkin (1991). A by-product of this method is being able to compute a number of complex integrals almost trivially.

Exercise 13.5 *Show that the solution to the coupled integral equations (Pétrélis, Llewellyn Smith & Young 2006)*

$$\frac{8\pi}{3} = \fint_{-1}^{1} \frac{S_\pm(t)}{t - x}\,\mathrm{d}t \mp \frac{\pi S_\pm(x)}{\sqrt{3}} \tag{13.11}$$

is

$$S_\pm(x) = \frac{4(x \mp 1/3)}{\sqrt{3}(1 - x^2)^{1/3}(1 \mp x)^{1/3}}. \tag{13.12}$$

(If you can do this, you are ready to teach this class.)

13.5 References

Publications:

- Linz, P. *Analytical and numerical methods for Volterra equations.* SIAM, Philadelphia, 1985. SIAM, 1985
- Llewellyn Smith, S. G. & Young, W. R. Tidal conversion at a very steep ridge. *J. Fluid Mech.*, **495**, 175–191, 2003.
- Pipkin, A. C. *A course on integral equations.* Springer, New York, 1991.
- Pétrélis, F., Llewellyn Smith, S. G. & Young, W. R. Tidal conversion at a submarine ridge. *J. Phys. Oceanogr.*, **32**, 1053–1071, 2006.
- Pozrikidis, C. *A practical guide to boundary element methods with the software library BEMLIB*, Chapman & Hall/CRC, Boca Raton, 2002.
- Tuck. E. O. Applications and solutions of Cauchy-singular integral equations. In *The application and numerical solution of integral equations*, (eg. R. S. Anderssen, F. R.de Hoog & M. A. Lukas), pp. 21–50, Sitjoff and Noordhoff, Alphen aan den Rijn, 1980.

Chapter 14

WKB and ray theory (05/14/08)

14.1 Liouville–Green approximants

When dealing with irregular singular points, we have already seen the use of the substitution $w = \mathrm{e}^S$, after which we usually solve for S as an asymptotic series. Usually, the leading-order behavior is given by $S'^2 \sim -q$ (it's always best to avoid writing $S'^2 + q \sim 0$ for technical reasons). The form e^S is sometimes called the Liouville–Green approximant to the function. It is a useful approximation close to a point x (which could be infinity). However it does not usually do that much over the rest of the range.

Example 14.1 *Consider the Bessel equation*

$$f'' + \frac{1}{x} f' + \left(k^2 - \frac{n^2}{x^2}\right) f = 0 \tag{14.1}$$

close to the irregular singular point at infinity. This is an LG expansion, with

$$S'' + S'^2 + x^{-1} S' + k^2 - n^2 x^{-2} = 0. \tag{14.2}$$

The dominant behavior comes from $S'^2 \sim -k^2$, so $S \sim \pm \mathrm{i}kx$. Then the next order term comes from writing $S = \pm \mathrm{i}kx + T$, with $|T| \ll x$ and similar relations on the derivatives of T. The governing equation for T is with

$$T'' - k^2 \pm 2\mathrm{i}kT' + T'^2 \pm \mathrm{i}kx^{-1} + x^{-1}T' - n^2 x^{-2} = 0. \tag{14.3}$$

Now the leading-order balance is $2T' \sim -x^{-1}$, so $T \sim -\frac{1}{2}\log x$ and $y \sim x^{-1/2}\mathrm{e}^{\pm \mathrm{i}x}$.

Exercise 14.1 *Work out the LG approximants for the equation*

$$f'' + \frac{1}{x^2} f - \frac{k^2}{x^2} f = 0. \tag{14.4}$$

Note that we can turn any linear second-order ODE $y'' + ay + b = 0$ into Liouville form $y'' + qy = 0$ by an appropriate substitution. This leaves the S'' and S'^2 terms in the LG expansion, so a series approach is still necessary.

Exercise 14.2 *Obtain the function q in the Liouville normal form in terms of a and b.*

Cheng (2007) has a nice discussion of ISPs, including some tricks to get the governing behavior quickly.

14.2 WKB(J)

The LG expansions are useful but not uniformly valid in x. If there is a small parameter in the problem, we can aim to find asymptotic expansions using that parameter valid everywhere (almost everywhere). Guided by the presence of the S'^2 term above, we want the oscillatory term to be multiplied by a small coefficient. We are hence led to the standard from

$$\epsilon^2 y'' + q(x)y = 0. \tag{14.5}$$

A small term multiplies the highest derivative, which is usually the sign of a singular perturbation (think of boundary layers). Here, however, setting $\epsilon = 0$ cannot provide a good approximation except under unusual circumstances. Instead, if $q > 0$, the function oscillates over the whole interval. If $q < 0$, there can be exponentially decaying or growing solutions that are more boundary-layer like. If q changes sign, some more care is needed.

Exercise 14.3 *Work out the transformation need to turn $f'' + af + b = 0$ into $y'' + qy = 0$.*

Guided by the LG expansion, we write

$$y = \mathrm{e}^{Q/\epsilon}, \qquad Q = Q_0/\epsilon + Q_1 + Q_2\epsilon + \cdots. \tag{14.6}$$

The higher terms in the expansion can be taken out of the exponential if desired. Substituting in expanding gives

$$\epsilon^2(Q'' + Q'^2) + q = 0. \tag{14.7}$$

We find $Q_0 = \pm\mathrm{i}\int^x q^{1/2}\,\mathrm{d}x$ where $q > 0$ and $Q_0 = \pm\int^x (-q)^{1/2}\,\mathrm{d}x$ where $q < 0$. Subsequently $Q_1 = -\frac{1}{4}\log|q|$.

Many high-frequency physical systems can be solved for using WKB ideas. Schrödinger's equation contains $\hbar$ and classical physics can be viewed as the limit as $\hbar \to 0$. The resulting equations give the Hamilton–Jacobi equations of mechanics. The Helmholtz equation that governs monochromatic wave propagation reduces in the limit of small wavelength to the eikonal equation (to be discussed in what follows). For an isotropic medium, this equation has light travelling in straight lines and we recover geometric optics. In fact the LG approximants are sometimes called the physical optics and geometric optics approximations to the solution, depending on the number of terms kept (see Bender & Orszag 1978).

The nomenclature WKB(J) stands for Wentzel, Kramers and Brillouin (and Jeffreys). These physicists worked on semi-classical limits of the quantum mechanics and were concerned with obtaining quantization rules. In fact, the first version of quantum mechanics was just classical mechanics with quantization rules. Their contribution was to the problem where $q(x)$ vanishes, a turning point. At such points, the WKB form derived above is no longer accurate. The solution looks like an Airy function, oscillatory on one side and exponentially decreasing or increasing on the other. One can match this Airy function to the WKB solutions on both sides, giving rise to a phase shift. There is a real subtlety to do with the direction in which matching is allowed, that is still a source of contention. The phase shift is quite real, and a beautiful and important piece of work

gives quantization rules purely from the topology of the domain. This is EBK (Einstein–Brillouin–Keller) quantization. These ideas can be applied more general to PDEs, as in Keller & Rubinow (1960).

The primary use of WKB in these cases is to derive eigenvalue conditions for bound states, and excellent results are obtained. The finite case is quite simple. Take (14.5) with $q(x) > 0$ on the interval $(0, 1)$ with homogeneous boundary conditions $y(0) = y(1) = 0$. This is an eigenvalue problem and the WKB solution is just

$$y(x) = Cq(x)^{-1/4} \sin\left(\frac{1}{\epsilon}\int_0^x q(x)^{1/2}\,\mathrm{d}x\right). \tag{14.8}$$

The condition at the origin is satisfied and that at 1 leads to the eigenvalue relation

$$\frac{1}{\epsilon}\int_0^1 q(x)^{1/2}\,\mathrm{d}x = n\pi \tag{14.9}$$

from the properties of the sine function. In this sense the problem is trivial because the integral above is just a number. Often one solves the problem

$$\epsilon^2 y'' + q(x)y = Ey, \tag{14.10}$$

for which the dependence of the integral on E is less trivial. Note that if the boundary conditions are on the derivatives of y, one neglects the prefactor when carrying out the derivative since it is asymptotically smaller and one ends up with cosines.

Example 14.2 *Find the eigenvalues ϵ_n of the equation*

$$\epsilon^2 y'' + 4\operatorname{sech}^{1/2} xy = 0 \tag{14.11}$$

on the interval $(0, 1)$ with homogeneous boundary conditions. We solve the problem numerically using Chebyshev collocation and compare to the WKB prediction. For this we need the numerical result $\int_0^1 2\operatorname{sech}^{1/4} x\,\mathrm{d}x = 1.926183$.

```
% p141.m WBK eigenvalue problem on bounded interval SGLS 05/15/08

N = 40;
qi = inline('2*sech(x).^0.25');
p = quadl(qi,0,1,eps);
eas = p./((1:N)'*pi);
[x,D] = chebdif(N+2,2); x = 0.5*(x+1); D2 = 4*D(2:end-1,2:end-1,2);
A = diag(4*sech(x).^0.5); A = A(2:end-1,2:end-1);
e = sqrt(-sort(eig(A,D2)));
semilogy(1:N,e,'.',1:N,eas,'o')
```

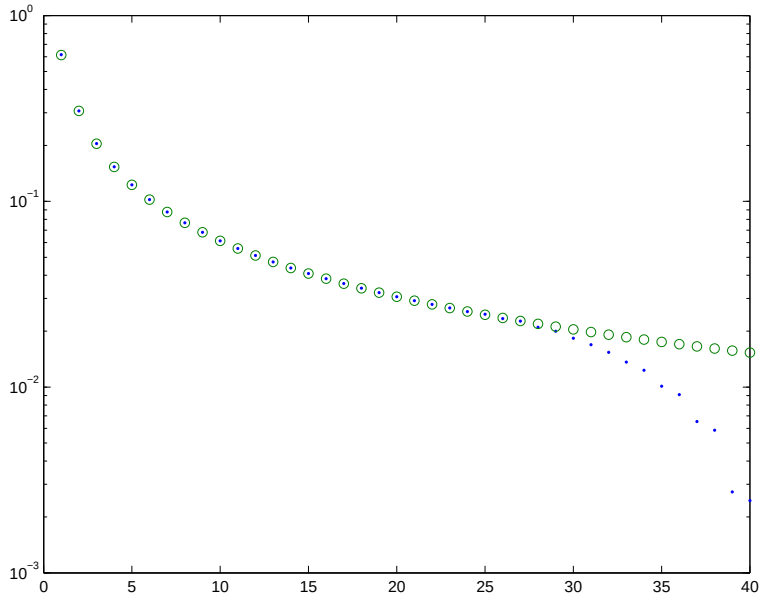

Figure 14.1: Eigenvalues of (14.11) from numerical calculations and WKB solution.

14.3 Ray theory: from expansions

Ray theory can be viewed as the multidimensional generalization of WKB. There are two ways of obtaining the governing equations, which are essentially the same. The first way is to take the governing equation and expand it in the relevant small or large parameter. We will illustrate this with the Helmholtz equation, following Bleistein (1984). The second way is to start with the dispersion relation for the waves and derive it from it the ray equations directly. This procedure is carried out in Lighthill (1979), Hinch (1991) and many other places.

Start with the Helmholtz equation

$$\nabla^2 u + (\omega^2/c^2(\mathbf{x}))u = 0, \tag{14.12}$$

where ω is in some sense large. Assume a solution of the form

$$u \sim \omega^\beta \mathrm{e}^{\mathrm{i}\omega\tau(\mathbf{x})} \sum_{j=0}^{\infty} \frac{A_j(\mathbf{x})}{(\mathrm{i}\omega)^2}. \tag{14.13}$$

We don't know β at this point and in fact it can only come from matching the solution to the prescribed data, so we ignore it for now. Now substitute and expand. we get

$$\begin{aligned} (\nabla\tau)^2 &= c^{-2}, & (14.14)\\ 2\nabla\tau\cdot\nabla A_0 + A_0\nabla^2\tau &= 0, & (14.15)\\ 2\nabla\tau\cdot\nabla A_j + \nabla^2\tau &= -\nabla^2 A_{j-1}. & (14.16) \end{aligned}$$

The first equation in the hierarchy, (14.14), is the eikonal equation. The second, (14.15), is the transport equation. The rest we ignore. The function $\tau(\mathbf{x})$ is has the units of time.

The eikonal equation (14.14) is a first-order nonlinear PDE. It can be solved by the method of characteristics (in fact using Charpuit's method – see Ockendon *et al.* 1999). This method solves for everything along the rays. Write $p = 1/c$ and $p_j = \partial\tau/\partial x_j$; the latter is the slowness vector. Then the eikonal equation becomes $\sum_{j=1}^{3} p_j p_j = p^2$. The characteristic equations for the eikonal equation become the ray equations

$$\frac{\mathrm{d}x_j}{\mathrm{d}\sigma} = \lambda p_j, \qquad \frac{\mathrm{d}p_j}{\mathrm{d}\sigma} = \lambda \frac{\partial p}{\partial x_j}, \qquad \frac{\partial \tau}{\partial \sigma} = \lambda p^2. \tag{14.17}$$

The new variable σ parameterizes the ray. Rays are directed along the gradient of τ.

Now the transport equation. Multiply (14.15) by A_0 to obtain

$$\nabla\cdot[A_0^2 \nabla\tau] = 0. \tag{14.18}$$

This is a conversation law. The quantity $A_0^2 p = A_0^2/c$ is preserved in ray tubes. We can turn (14.15) into an ODE along characteristics by substituting (14.29) into (14.15) and obtaining

$$\frac{\mathrm{d}A_0^2}{\mathrm{d}\sigma} = -A_0^2 \lambda \nabla^2 \tau. \tag{14.19}$$

If we know a ray solution for τ, the right-hand side is known. However, the ray equations presented so far are insufficient: we need gradients of τ along the ray. Obtaining these requires solving 12 more ODEs along the rays.

There are a number of choices possible for σ, given by λ. First, $\lambda = 1$, which is the simplest to use in analytical solutions. Second, $\lambda = c = 1/p$, in which case σ is arc length. Third, $\lambda = c^2 = 1/p^2$, in which case τ can be used as the parameter along the ray.

Ray theory has many uses; the Helmholtz equation is used is seismology to model travel times of seismic waves. There is some unpleasantness, since the different shear and dilational modes couple at boundaries and interfaces where the wave speeds have discontinuities.

For more general equations, it would be nice to have a rapid approach, and there is one. However, there are time when only the expansion procedure will do. This is particularly true when dealing with systems with background flow, where it is not obvious what the dispersion relation is from the beginning.

14.4 Ray theory: from dispersion relations

We consider a scalar wave field that takes the form of a single wave propagating with slowly varying amplitude $a(\mathbf{X}, t)$ and slow varying phase $\theta = \epsilon^{-1}\Theta(\mathbf{X}, t)$. We can define a local frequency $\omega = -\Theta_T$ and a local wavenumber $\mathbf{k} = \partial\Theta/\partial\mathbf{X}$. This gives two consistency relations

$$\nabla_X \times \mathbf{k} = \mathbf{0}, \qquad \nabla_X \omega + \frac{\partial \mathbf{k}}{\partial T} = 0. \tag{14.20}$$

We will have a local dispersion relation $\omega = \Omega(\mathbf{k}, \mathbf{X}, T)$, where the last two arguments denote the dependence on the medium. The group velocity is given by $\mathbf{f}_g = \nabla_k \Omega$. We

obtain

$$\frac{\mathrm{d}\omega}{\mathrm{d}T} = \frac{\partial\Omega}{\partial T}, \qquad \frac{\mathrm{d}\mathbf{k}}{\mathrm{d}T} = -\nabla_X\Omega, \tag{14.21}$$

where total derivatives indicate differential along ray with $\mathrm{d}/\mathrm{d}T = \partial/\partial T + \mathbf{f}_g\cdot\nabla$. We see that in a time-independent medium, frequency is conserved along a ray. In a spatially homogeneous medium, wavenumber is conserved along a ray. If the medium properties depend only on z, the horizontal wavenumber is conserved along a ray. These equations are the same as were obtained earlier when solving the eikonal equation using the method of characteristics.

One finds that a scalar quantity is also conserved along ray tubes, but it is no longer energy. Instead the quantity E/ω, called wave action, is conserved. If the frequency is constant, one recovers conservation of energy.

14.5 Eulerian eikonal solvers

The eikonal equation is nonlinear whereas the original equation that gave rise to it was linear. This is slightly unusual and has the consequence that one has to be careful when talking about adding solutions. For example, there may be several rays passing through one point, or none (shadow). These rays are separate solutions to the eikonal equation.

One problem with ray approaches is that one cannot control in advance where the rays go and one end up getting poor coverage in certain regions. Of course in the shadows one gets nothing. One can get around this by integrating backward from the observation point, but then one has a boundary-value problem since one needs to find the ray that came in from the right direction.

The idea of Eulerian solvers is hence very tempting: one would obtain the solution to the eikonal equation at each point on a grid. Although since the physical field is the superposition of a number of rays, what would this mean? It turns out that one can construct such methods, and they have the property (if one does it right) that they pick out the first travel time solution, i.e. the first ray to reach the grid point. These ideas, as well as a discussion of efficient implementations, can be found in Sethian (1999).

Obtaining the full field is much more difficult, and is still to some extent an open problem. One needs to augment the dimension of the problem to deal with shading regions and caustics, and we are being taken too far afield.

Project 7 *Implement an Eulerian ray method that produces the full physical field for an arbitrary underlying PDE. (This is not a realistic project.)*

14.6 Ray theory: example

These are notes, mostly by Joe Keller, on the problem of acoustic scattering by a vortex.

14.6.1 Formulation

The Euler equations for the density $\rho(\mathbf{x}, t)$ and velocity $\mathbf{u}(\mathbf{x}, t)$ of an isentropic flow are

$$\rho_t + \nabla \cdot (\rho \mathbf{u}) = 0, \qquad \rho[\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u}] = -\nabla p(\rho). \tag{14.22}$$

When the solution depends upon a parameter ε, the derivative of (14.22) yields the acoustic equation for $\dot{\rho}$ and $\dot{\mathbf{u}}$, the derivatives of the solution with respect to ε:

$$\dot{\rho}_t + \nabla\cdot(\dot{\rho}\mathbf{u} + \rho\dot{\mathbf{u}}) = 0, \qquad \dot{\rho}[\mathbf{u}_t + (\mathbf{u}\cdot\nabla)\mathbf{u}] + \rho[\dot{\mathbf{u}}_t + (\dot{\mathbf{u}}\cdot\nabla)\mathbf{u} + (\mathbf{u}\cdot\nabla)\dot{\mathbf{u}}] = -\nabla(c^2\dot{\rho}). \tag{14.23}$$

Here we have introduced the sound speed c defined by $c^2 = p_\rho(\rho)$.

A solution of (14.22) is the axially symmetric vortex flow

$$\rho = \text{constant}, \qquad \mathbf{u} = (0, ca\, r\Omega(r)). \tag{14.24}$$

The velocity $\mathbf{u}$ is written in cylindrical coordinates r, θ and ca is the vortex strength. Using (14.24) in (14.23) yields

$$\dot{\rho}_t + \mathbf{u} \cdot \nabla\dot{\rho} + \rho\nabla \cdot \dot{\mathbf{u}} = 0, \qquad \dot{\mathbf{u}}_t + (\mathbf{u}\cdot\nabla)\dot{\mathbf{u}} + (\dot{\mathbf{u}}\cdot\nabla)\mathbf{u} = -\rho^{-1}c^2\nabla\dot{\rho}. \tag{14.25}$$

We seek a solution of (14.25) corresponding to an incident time-harmonic plane wave with wavenumber k and angular frequency $\omega = kc$ coming from $x = -\infty$. With $\mathbf{f}_x$ denoting a unit vector in the x-direction, we can write it as

$$\dot{\rho}_{\text{inc}} = \rho B_0 \mathrm{e}^{\mathrm{i}(kx-\omega t)}, \qquad \mathbf{u}_{\text{inc}} = cB_0\mathbf{f}_x\mathrm{e}^{\mathrm{i}(kx-\omega t)}. \tag{14.26}$$

We write the solution in the following form, which is appropriate for $ka \gg 1$:

$$\dot{\rho}_{\text{inc}} = \mathrm{e}^{\mathrm{i}(k\varphi(\mathbf{x})-\omega t)}\rho[B(\mathbf{x}) + (\mathrm{i}k)^{-1}B_1(\mathbf{x}) + O(k^{-2})], \tag{14.27}$$

$$\dot{\mathbf{u}}_{\text{inc}} = \mathrm{e}^{\mathrm{i}(k\varphi(\mathbf{x})-\omega t)}c[\mathbf{A}(\mathbf{x}) + (\mathrm{i}k)^{-1}\mathbf{A}_1(\mathbf{x}) + O(k^{-2})]. \tag{14.28}$$

The solution (14.28) must tend to the incident wave (14.26) as $x \to -\infty$, and otherwise must contain only outgoing waves.

To determine φ, B and $\mathbf{A}$ we substitute (14.29) into (14.25), and retain term of order k^1 and k^0. Equating to zero the coefficient of k^1 yields

$$(-c + \mathbf{u}\cdot\nabla\varphi)B + c\mathbf{A}\cdot\nabla\varphi = 0, \qquad (-c + \mathbf{u}\cdot\nabla\varphi)\mathbf{A} + cB\nabla\varphi = \mathbf{0}. \tag{14.29}$$

Then equating to zero terms of order k^0 yields

$$(-c + \mathbf{u}\cdot\nabla\varphi)B_1 + c\mathbf{A}_1\cdot\nabla\varphi = -\mathbf{u}\cdot\nabla B - c\nabla\cdot\mathbf{A}, \tag{14.30}$$

$$(-c + \mathbf{u}\cdot\nabla\varphi)\mathbf{A}_1 + cB_1\nabla\varphi = -(\mathbf{u}\cdot\nabla)\mathbf{A} - (\mathbf{A}\cdot\nabla)\mathbf{u} - c\nabla B. \tag{14.31}$$

Now we must solve (14.29) and (14.31) subject to the condition at $x = -\infty$ corresponding to the incident wave

$$\varphi \to x = r\cos\theta, \quad B \to B_0, \quad \mathbf{A} \to B_0\mathbf{f}_x \qquad \text{as } x \to -\infty. \tag{14.32}$$

14.6.2 Eikonal equation and rays

The solution of the second equation in (14.29) for $\mathbf{A}$ in terms of B and φ is

$$\mathbf{A} = (c - \mathbf{u}\cdot\nabla\varphi)^{-1} cB\nabla\varphi = (1 - a\Omega\varphi_\theta)^{-1} B\nabla\varphi. \tag{14.33}$$

We use (14.33) in the first equation of (14.29) and require that $B \neq 0$ to obtain

$$(\nabla\varphi)^2 - (1 - a\Omega\varphi_\theta)^2 = 0. \tag{14.34}$$

This is a first-order partial differential equation for φ called the eikonal equation which can be solved by the method of rays or characteristics.

To use this method we introduce a parameter τ and a curve $r(\tau)$, $\theta(\tau)$, $p_1(\tau)$, $p_2(\tau)$ where $p_1(\tau) = \varphi_r[r(\tau),\theta(\tau)]$ and $p_2(\tau) = \varphi_\theta[r(\tau),\theta(\tau)]$. We define the Hamiltonian $H(p_1, p_2, r, \theta)$ to be half the left side of (14.34):

$$H(p_1, p_2, r, \theta) = \frac{1}{2}p_1^2 + \frac{1}{2r^2}p_2^2 - \frac{1}{2}(1 - a\Omega p_2)^2. \tag{14.35}$$

Then Hamilton's equations for the curves are

$$\begin{aligned}
p_{1\tau} &= -H_r = r^{-3}p_2^2 - a\Omega' p_2(1 - a\Omega p_2), && (14.36)\\
p_{2\tau} &= -H_\theta = 0, && (14.37)\\
r_\tau &= H_{p_1} = p_1, && (14.38)\\
\theta_\tau &= H_{p_2} = r^{-2}p_2 + a\Omega(1 - a\Omega p_2). && (14.39)
\end{aligned}$$

From (14.37) we see that $p_2 =$ constant along a ray. Then we use (14.38) to write the left-hand side of (14.36) as $p_{1\tau} = r_{\tau\tau}$, and (14.36) becomes a second-order equation for r. We multiply it by r_τ and integrate to get, with E a constant of integration,

$$\frac{1}{2}r_\tau^2 = -\frac{1}{2r^2}p_2^2 + \frac{1}{2}(1 - a\Omega p_2)^2 + E. \tag{14.40}$$

We set $E = 0$ and then (14.40) is just the equation $H = 0$ obtained from (14.34) and (14.35) with $p_1 = r_\tau$. Now we solve (14.40) for $1/r_\tau = \mathrm{d}\tau/\mathrm{d}r$ and integrate to get

$$\tau = \pm\int_{r_0}^{r} [(1 - a\Omega p_2)^2 - r^{-2}p_2^2]^{-1/2}\,\mathrm{d}r. \tag{14.41}$$

We choose $\tau = 0$ when $r = r_0$. The minus sign holds for $\tau < 0$, the plus sign for $\tau > 0$. The lower limit r_0 is the minimum value of r on the ray. At it the bracketed expression in the integrand vanishes.

Once $r(\tau)$ is determined from (14.41) then $\theta(t)$ can be found by integrating (14.39). To find $\varphi(\tau) = \varphi[r(\tau),\theta(\tau)]$, we write

$$\varphi(\tau) = \varphi_r r_\tau + \varphi_\theta\theta_\tau = p_1 r_\tau + p_2\theta_\tau = r_\tau^2 + p_2\theta_\tau. \tag{14.42}$$

Here we have used the definitions of p_1 and p_2 as well as (14.38). We integrate (14.42) from τ_0 to τ go get

$$\varphi(\tau) - \varphi(\tau_0) = p_2\theta(\tau) - p_2\theta(\tau_0) + \int_{\tau_0}^{\tau}(r_\tau^2 - 1)\,\mathrm{d}\tau + \tau - \tau_0. \tag{14.43}$$

As $\tau_0 \to -\infty$, (14.32) shows that $\varphi(\tau_0) \to x = r\cos\theta \sim -r$ since $\theta \to \pi$ and (14.41) shows that $\tau_0 \sim -r$. Then as $\tau_0 \to -\infty$, (14.43) yields the following expression

$$\varphi(\tau, p_2) = \tau + p_2[\theta(\tau) - \pi] + \int_{-\infty}^{\tau} [(1 - a\Omega p_2)^2 - r^{-2}p_2^2 - 1]\,\mathrm{d}\tau. \tag{14.44}$$

The parameter p_2 is constant on a ray. By using the definition $p_2 = \varphi_\theta$ with the asymptotic form of φ at $x = -\infty$, we get $p_2 = \partial_\theta(r\cos\theta) = -r\sin\theta = -y$. Then p_2 is minus the impact parameter, i.e. the distance from the origin to the asymptote to the incident ray.

14.6.3 Transport equation for the amplitude

Now we consider (14.31) which is a system of inhomogeneous linear algebraic equations for $\mathbf{A}_1$ and B_1. The homogeneous form of these equations, (14.17), has a nontrivial solution, so the coefficient matrix is singular. Therefore (14.31) will have a solution only if the inhomogeneous terms satisfy a solvability condition. To obtain it we solve the second equation of (14.31) for $\mathbf{A}_1$ in terms of B_1 and use the solution in the first equation to eliminate $\mathbf{A}_1$. The coefficient of B_1 vanishes, leaving the equation

$$\nabla\varphi\cdot[(\mathbf{A}\cdot\nabla)c^{-1}\mathbf{u} + (c^{-1}\mathbf{u}\cdot\nabla)\mathbf{A}] + \nabla\varphi\cdot\nabla B = (c^{-1}\mathbf{u}\cdot\nabla\varphi - 1)(c^{-1}\mathbf{u}\cdot\nabla B + \nabla\cdot\mathbf{A}). \tag{14.45}$$

This is the solvability condition into which we now substitute (14.33) for $\mathbf{A}$ to get

$$(1 - c^{-1}\mathbf{u}\cdot\varphi)^{-1}\nabla\varphi\cdot[(\nabla\varphi\cdot\nabla)c^{-1}\mathbf{u}]B + \nabla\varphi\cdot\nabla B + (1 - c^{-1}\mathbf{u})[] \tag{14.46}$$

14.6.4 Implementation

Figure 14.2 shows the result of tracing rays numerically through a vortex with Gaussian streamfunction. Note the crowding of rays and caustics. The Mach number is $a = 0.1$

```
% p142.m Ray tracing for Gaussian vortex SGLS 05/15/08

function p142
xmin=-100; dy=0.1; a=0.1; fn=@vel;

xt=ones(400,601)*NaN;
yt=xt; phit=xt; xgt=xt; ygt=xt; xdt=xt; ydt=xt; alphat=xt; tt=xt;
lt=zeros(1,201);

j=0;
for yy=-20:dy:20
   j=j+1;
   [t,h]=doray(500,xmin,yy,0,1,a,1e-6,fn);

   l=size(t,1);
   lt(j)=l;
```

```
   tt(1:l,j)=t;
   xt(1:l,j)=h(:,3);
   yt(1:l,j)=h(:,4);
   phit(1:l,j)=h(:,5);
   xgt(1:l,j)=h(:,9);
   ygt(1:l,j)=h(:,10);
   [u,v,ux,vx,uy,vy,uxx,vxx,uyy,vyy,uxy,vxy]=feval(fn,h(:,3),h(:,4),a);
   alphat(1:l,j)=1-u.*h(:,1)-v.*h(:,2);
   for k=1:size(t,1)
      hp=rayeq(0,h(k,:)',a,fn);
      xdt(k,j)=hp(3);
      ydt(k,j)=hp(4);
   end
   [yy l]
end

lm=max(lt);
t=tt(1:lm,1:j); x=xt(1:lm,1:j); y=yt(1:lm,1:j); phi=phit(1:lm,1:j);
xg=xgt(1:lm,1:j); yg=ygt(1:lm,1:j); xd=xdt(1:lm,1:j); yd=ydt(1:lm,1:j);
alpha=alphat(1:lm,1:j);
J=(xd.*yg - yd.*xg);

alpha0=ones(lm,1)*alpha(1,:);
J0=ones(lm,1)*J(1,:);
B=alpha./alpha0.*sqrt(abs(J0./J));

clear tt xt yt phit xgt ygt xdt ydt

figure(2)
clf
contour(x,y,phi,(0:0.5:200)*pi)
hold on
contour(x,y,J,[0 0],'k')
hold off
axis equal


function [t,h]=doray(tmax,x,y,psi,dir,a,tol,rayfn)
%DORAY Compute ray trajectory
%   [T,H]=DORAY(TMAX,X,Y,PSI,DIR,A,TOL,RAYFN)
%   integrates the ray equations for the velocity field
%   defined by rayfn.
%
%   Input:
%       tmax:  maximum time
```

```
%       x:     initial x location
%       y:     initial y location
%       psi:   angle of ray to horizontal
%       dir:   positive corresponds to larger p1 (e.g. towards +x)
%       a:     velocity amplitude (Mach number)
%       tol:   integration tolerance
%       rayfn: function defining velocity field
%
%   Output:
%       t     : time
%       h     : [p1 p2 x y phi 0 p1g p2g xg yg]

% Set up initial condition for p1 and p2
[u,v,ux,vx,uy,vy,uxx,vxx,uyy,vyy,uxy,vxy]=feval(rayfn,x,y,a);
a1=sin(psi)*(1-u^2)+u*v*cos(psi);
a2=cos(psi)*(1-v^2)+u*v*sin(psi);
be=(u*sin(psi)-v*cos(psi))/a2;
A=1-u^2+a1^2/a2^2*(1-v^2)-2*u*v*a1/a2;
B=2*(u + v*(a1/a2-u*be) + be*a1/a2*(1-v^2));
C=be^2*(1-v^2)-1+2*v*be;
if (dir>=0)
   p1=max(roots([A B C]));
else
   p1=min(roots([A B C]));
end
p2=a1/a2*p1+be;
%H=0.5*(p1^2+p2^2)-0.5*(1-u*p1-v*p2)^2;

% Set up initial conditon for normal derivative equations
a1y=-2*u*uy*sin(psi)+(uy*v+u*vy)*cos(psi);
a2y=-2*v*vy*cos(psi)+(uy*v+u*vy)*sin(psi);
bey=(uy*sin(psi)-vy*cos(psi))/a2 - a2y/a2^2*(u*sin(psi)-v*cos(psi));
Ay=-2*u*uy + (2*a1y*a1*a2^2-2*a2y*a2*a1^2)/a2^4*(1-v^2)* - 2*a1^2/a2^2*v*vy -2*(uy*v*+u
By=2*(uy +vy*(a1/a2-u*be) +v*( (a1y*a2-a2y*a1)/a2^2 -uy*be - u*bey) + bey*a1/a2*(1-v^2)
Cy=2*be*bey*(1-v^2)-2*be^2*v*vy +2*(vy*be+v*bey);
p1y=-(Ay*p1^2+By*p1+Cy)/(2*A*p1+B);
p2y=(a1y*a2-a2y*a1)/a2^2*p1+a1/a2*p1y+bey;

h0=[p1 ; p2 ; x ; y ; 0 ; 0 ; p1y ; p2y ; 0 ; 1];

options=odeset('RelTol',tol,'AbsTol',tol);
[t,h]=ode45(@(t,h) rayeq(t,h,a,rayfn),[0 tmax],h0,options);



function hdot = rayeq(t,h,a,rayfn)
```

```
%RAYEQ Return derivative in ray equations
%   HDOT=RAYEQ(T,H,A,RAYFN)
%
%   Input:
%       t:     time
%       h:     vector [p1 p2 x y phi 0 p1g p2g xg yg]
%       a:     velocity amplitude (Mach number)
%       rayfn: function defining velocity field
%
%   Output:
%       hdot:  derivative of h

p1=h(1);
p2=h(2);
x=h(3);
y=h(4);

[u,v,ux,vx,uy,vy,uxx,vxx,uyy,vyy,uxy,vxy]=feval(rayfn,x,y,a);
alpha=1-u*p1-v*p2;
hdot(1)=-(ux*p1+vx*p2)*alpha;
hdot(2)=-(uy*p1+vy*p2)*alpha;
hdot(3)=p1+u*alpha;
hdot(4)=p2+v*alpha;
hdot(5)=p1*hdot(3)+p2*hdot(4);
hdot(6)=0;

Hxx=uxx*p1+vxx*p2-(uxx*v+2*ux*vx+u*vxx)*p1*p2-(ux^2+u*uxx)*p1^2-(vx^2+v*vxx)*p2^2;
Hxy=uxy*p1+vxy*p2-(uxy*v+ux*vy+uy*vx+u*vxy)*p1*p2-(ux*uy+u*uxy)*p1^2-(vx*vy+v*vxy)*p2^2
Hxp1=ux-(ux*v+u*vx)*p2-2*u*ux*p1;
Hxp2=vx-(ux*v+u*vx)*p1-2*v*vx*p2;
Hyy=uyy*p1+vyy*p2-(uyy*v+2*uy*vy+u*vyy)*p1*p2-(uy^2+u*uyy)*p1^2-(vy^2+v*vyy)*p2^2;
Hyp1=uy-(uy*v+u*vy)*p2-2*u*uy*p1;
Hyp2=vy-(uy*v+u*vy)*p1-2*v*vy*p2;
Hp1p1=1-u^2;
Hp1p2=-u*v;
Hp2p2=1-v^2;

hdot(7)=-Hxp1*h(7)-Hxp2*h(8)-Hxx*h(9)-Hxy*h(10);
hdot(8)=-Hyp1*h(7)-Hyp2*h(8)-Hxy*h(9)-Hyy*h(10);
hdot(9)=Hp1p1*h(7)+Hp1p2*h(8)+Hxp1*h(9)+Hyp1*h(10);
hdot(10)=Hp1p2*h(7)+Hp2p2*h(8)+Hxp2*h(9)+Hyp2*h(10);

hdot=hdot';
```

```
function [u,v,ux,vx,uy,vy,uxx,vxx,uyy,vyy,uxy,vxy]=vel(x,y,a)
%GAUSSPSI Return velocity and derivatives for Gaussian streamfunction
%   [U,V,UX,VX,UY,VY,UXX,VXX,UYY,VYY,UXY,VXY]=VEL(X,Y,A)
%
%   Input:
%       x:     x coordinate
%       y:     y coordinate
%       a:     velocity amplitude (Mach number)
%
%   Output:
%       u     : x-component of velocity
%       v     : y-component of velocity
%       ux    : u_x
%       vx    : v_x
%       uy    : u_y
%       vy    : v_y
%       uxx   : u_xx
%       vxx   : v_xx
%       uyy   : u_yy
%       vyy   : v_yy
%       uxy   : u_xy
%       vxy   : v_xy

r=sqrt(x.^2+y.^2);
theta=atan2(y,x);
c=cos(theta);
s=sin(theta);

Om=a*(-2)*exp(-r.^2);
Omp=a*(4*r).*exp(-r.^2);
Ompp=a*(4-8*r.^2).*exp(-r.^2);

u=-y.*Om;
v=x.*Om;
ux=-y.*c.*Omp;
uy=-Om-y.*s.*Omp;
vx=Om+x.*c.*Omp;
vy=x.*s.*Omp;

uxx=-y.*c.^2.*Ompp-s.^3.*Omp;
uxy=-c.*Omp-y.*c.*s.*Ompp+s.^2.*c.*Omp;
uyy=-2.*s.*Omp-y.*s.^2.*Ompp-s.*c.^2.*Omp;
vxx=2.*c.*Omp+x.*c.^2.*Ompp+s.^2.*c.*Omp;
vxy=s.*Omp+x.*c.*s.*Ompp-c.^2.*s.*Omp;
```

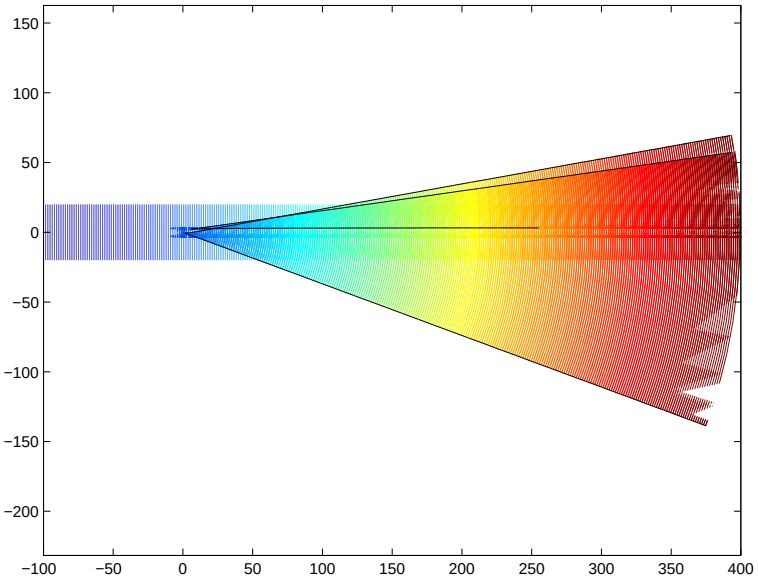

Figure 14.2: Wave fronts for scattering by a Gaussian vortex using numerical ray tracing.

```
vyy=x.*s.^2.*Ompp+c.^3.*Omp;
```

14.7 Complex Rays

One can investigate how the ansatz (14.13) does in shadow regions. A fascinating approach is that of Chapman *et al.* (1999), in which Stokes lines figure prominently.

Exercise 14.4 *Read this paper. Write a two-paragraph summary.*

14.8 References

Publications:

- Bender, C. M. & Orszag, S. A. *Advanced mathematical methods for scientists and engineers.* McGraw-Hill, ?, 1978.
- Bleistein, N. *Mathematical methods for wave phenomena.* Academic Press, Orlando, 1984.
- Chapman, S. J., Lawry, J. M. H., Ockendon, J. R. & Tew, R. H. On the theory of complex rays. *SIAM Rev.*, **41**, 417–509, 1999.
- Cheng, H. *Advanced analytic methods in applied mathematics, science, and engineering.* LuBan, Boston, 2007.

- Hinch, E. J. *Perturbation methods.* Cambridge University Press, Cambridge, 1991.
- Lighthill, M. J. *Waves in fluids.* Cambridge University Press, Cambridge, 1978.
- Ockendon, J. R., Howison, S. D., Lacey, A. A. & Movchan, A. B *Applied partial differential equations*, Oxford University Press, Oxford, 1999.
- Keller, J. B. & Rubinow, S. I. Asymptotic solution of eigenvalue problems. *Ann. Phys.*, 9, 24–75, 1960.
- Sethian, J. A. *Level set methods and fast marching methods.* 2nd ed., Cambridge University Press, Cambridge, 1999.

Chapter 15

Laplace's equation (05/21/08)

15.1 Introduction

From a complex variable perspective, we know that the real and imaginary parts of an analytic function $f(z) = u(x,y) + \mathrm{i}v(x,y)$ satisfy Laplace's equation and are hence harmonic. This is simply from the Cauchy–Riemann equations

$$u_x = v_y, \qquad u_y = -v_x, \tag{15.1}$$

cross-differentiating. If f is analytic, it is certainly smooth enough to allow taking extra derivatives.

Graphically we have seen that this corresponds to no maxima or minima in a closed domain. Every point looks (locally) like a saddle: if it increases in one direction, it decreases perpendicular to it. In one dimension, it's hard to do that, so the equivalent of Laplace's equation, namely $f_{xx} = 0$, means that the function is linear or constant. In three dimensions or more, various directions can be involved, provided the total curvatures add up to zero.

It's worth plotting an example of a solution to Laplace's equation to remember what this looks like. We already solved Laplace's equation and found the Poisson integral in Chapter 5. Here we sum functions in a square.

Solve Laplace's equation $\nabla^2 u = 0$ in the square $0 \le x, y \le 1$ with $u = 0$ on $x = 0$, $u = 0$ on $y = 0$, $u = x$ on $y = 1$ and $u = y^2$ on $x = 1$. Note that the boundary conditions are compatible here: the limit of u along one edge is the same as the limit along another edge that meets the first. This is not always necessary, but tends to make things converge well. The natural way to solve this problem is by using Fourier series. It isn't clear whether to transform in x or in y, and which form to use, so let's try both ways with the most general possible expansion.

Write

$$u = \sum_n u_n(y) f_n(x), \tag{15.2}$$

where we require the f_n to be orthogonal and satisfy $f_n'' = -a_n^2 f_n$, but do not specify their precise form yet. Now multiply Laplace's equation by $f_m(x)$ and integrate by parts, giving

$$I_m(u_m'' - a_m^2 u_m) + [f_m u' - f_m' u]_0^1 = 0, \tag{15.3}$$

where $I_m = \int_0^1 f_m^2 \,\mathrm{d}x f$. We do not know u' on the boundaries, so we take $f_m = 0$ there. Hence $f_n = \sin n\pi x$, $a_n = n\pi$ and $I_n = \frac{1}{2}$. We find

$$u_n'' - (n\pi)^2 u_n = 2f_n'(1)u(1,y) = 2n(-1)^n y^2. \tag{15.4}$$

We can solve this using the boundary condition at $y = 0$ to find

$$u_n = b_n \sinh n\pi y + \frac{2(-1)^{n+1}}{n\pi^2}\left[y^2 + \frac{2}{(n\pi)^2}(1 - \mathrm{e}^{-n\pi y})\right]. \tag{15.5}$$

The boundary condition at $y = 1$ gives

$$b_n \sinh n\pi + \frac{2(-1)^{n+1}}{n\pi^2}\left[1 + \frac{2}{(n\pi)^2}(1 - \mathrm{e}^{-n\pi})\right] = 2\int_0^1 x \sin n\pi x \,\mathrm{d}x = \frac{2(-1)^{n+1}}{n\pi}, \tag{15.6}$$

which is an equation for the b_n.

Alternatively we expand in y. The working is slightly simpler. Start with

$$u = \sum_n v_n(x) f_n(y). \tag{15.7}$$

Now

$$v_n'' - (n\pi)^2 v_n = 2f_n'(1)u(x,1) = 2n(-1)^n x. \tag{15.8}$$

We can solve this using the boundary condition at $x = 0$ to find

$$v_n = c_n \sinh n\pi x + \frac{2(-1)^{n+1}}{n\pi^2} x. \tag{15.9}$$

The boundary condition at $x = 1$ gives

$$c_n \sinh n\pi + \frac{2(-1)^{n+1}}{n\pi^2} = 2\int_0^\pi y^2 \sin n\pi y \,\mathrm{d}y = \begin{cases} -\dfrac{2}{n\pi} & \text{for } n \text{ even}, \\ \dfrac{2(n^2\pi^2 - 4)}{n^3\pi^3} & \text{for } n \text{ odd}, \end{cases} \tag{15.10}$$

which is an equation for the c_n.

Figure 15.1 shows surface plots of these two solutions. The convergence of the Fourier series is hampered by Gibbs' phenomenon. The series in y looks frankly wrong. So while expanding in Fourier series may seem natural, it does not always give good results.

```
% p151.m Fourier series solution of Laplace's equation SGLS 05/20/08
x = 0:0.01:1; y = 0:0.02:1; [x,y] = meshgrid(x,y);
u = zeros(size(x)); v = u;
for n = 1:50
  b = 1/sinh(n*pi)*( 2*(-1)^(n+1)/(n*pi^2)*(pi - (1 + 2/(n*pi)^2*(1 - exp(-n*pi)))));
  u = u + sin(n*pi*x).*(b*sinh(n*pi*y) + 2*(-1)^(n+1)/(n*pi^2)*(y.^2 + 2/(n*pi)^2*(1 -
  if (mod(n,2)==0)
    c = 1/sinh(n*pi)*( -2*(-1)^(n+1)/(n*pi^2) - 2/(n*pi) );
```

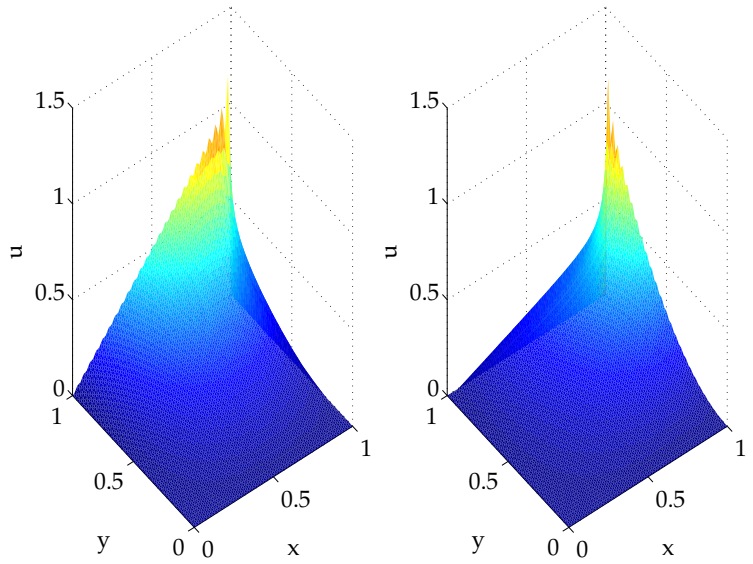

Figure 15.1: Solution to Laplace's equation using separation of variables.

```
  else
    c = 1/sinh(n*pi)*( -2*(-1)^(n+1)/(n*pi^2) + 2*(n^2*pi^2-4)/(n*pi)^3 );
  end
  v = v + sin(n*pi*y).*(c*sinh(n*pi*x) + 2*(-1)^(n+1)/(n*pi^2)*x);
end
subplot(1,2,1)
surf(x,y,u); shading flat
xlabel('x'); ylabel('y'); zlabel('u')
subplot(1,2,2)
surf(x,y,v); shading flat
xlabel('x'); ylabel('y'); zlabel('u')
```

Exercise 15.1 *Why is there a difference between the two plots? Check whether the code and algebra are correct.*

This is a typical example of a solution to Laplace's equation that was obtained in closed form, but which requires evaluation on a computer. There are many ways of finding solutions to Laplace's equation: by finding an appropriate analytic function (including using conformal maps), by separating variables, by transform methods and by finite difference/finite element/spectral methods. We leave the first approach to Chapter 16. The third has been investigated already, so we concentrate on the other approaches.

Laplace's equation is the canonical[1] elliptic PDE. It has no characteristics along which information propagates, and its solutions are smooth. There are some interesting twists

[1]Fancy mathematical word for typical or original.

that would take us some way beyond standard elliptic problems: free surfaces can give waves and hyperbolic systems. The natural generalization to $\nabla^4 u = 0$ is the biharmonic equation, which has interesting properties in terms of analyticity. Codimension-two problems have the position of the boundary as one of the unknowns (Howison, Morgan & Ockendon 1997). These problems are sometimes called or free boundary problems. The latter name is more general, since it includes problems in which an interface evolves, such as freezing and melting problems. In codimension-one problems, it is not evolution that is important, but the solution of some equilibrium problem, typically with one boundary condition over the known parts of the boundary, and two boundary conditions over the remainder of the boundary, which is one of the quantities to be found.

15.2 Applications

Laplace's equation is one of the ubiquitous and most important equations of mathematical physics. It occurs in potential theory whenever one has a conservative solenoidal field. Fields satisfying Laplace's equation include the gravitational potential outside matter, the electrical potential in free space, the velocity potential in fluid mechanics, various potentials in elasticity, the displacement of a membrane. In all these cases, the field is trying abstractly to be as smooth as possible, in the sense of having no maxima or minima. The membrane example provides a graphical interpretation, since the actual surface displacement (e.g. of a drum) is the solution to the equation.

Many other equations of mathematical physics such as the diffusion equation, the wave equation of Helmholtz's equation contain the Laplacian operator. In general, this term dominates close to singularities and hence understanding the behavior of the Green's functions of the Laplace equation is particularly useful.

15.3 Pseudospectral methods

The term "pseudospectral" is potentially misleading. The approach is spectral in the sense that the unknown function is represented as an expansion, and the word spectral usually refers to some kind of Sturm–Liouville problem that leads to eigenfunctions. The "pseudo" refers to the fact that the governing equations are solved at certain collocation points. Other spectral approaches, namely the Galerkin and the tau method, don't use physical space as part of the algorithm, although of course to plot solutions one normally needs to return to the physical variable. Do not confuse the pseudo here with the option of computing nonlinear terms in physical space: that can be done (trivially) using pseudospectral methods or in the other approaches, at the cost of transforming a few times. This is historically how the first spectral methods were devised.

Modern pseudospectral methods using Matlab have made solving Laplace's equation in two dimensions very simple indeed. Figure 15.2 shows a contour plot of u for the same boundary conditions as before. The code is simple; Neumann or mixed boundary conditions can be incorporated easily by changing the lines corresponding to `L(bx0,:)` and so on.

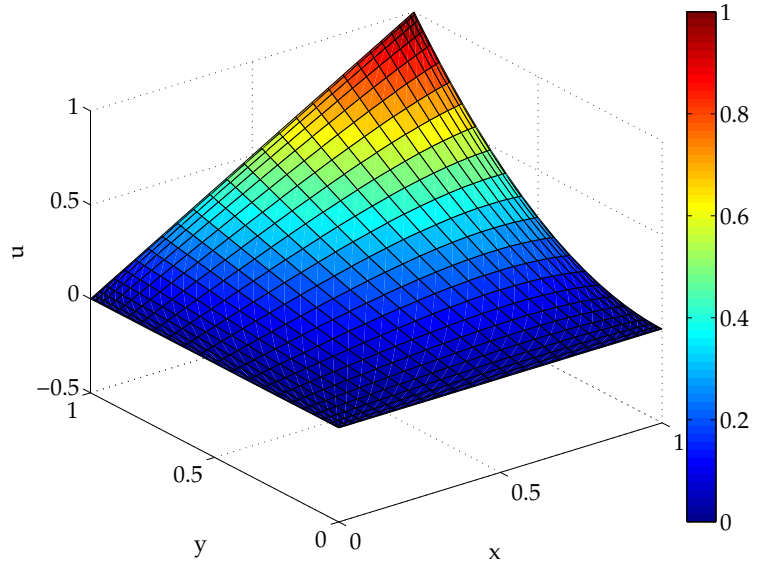

Figure 15.2: Solution to Laplace's equation using Chebyshev collocation.

```
% p152.m Laplace's equation using pseudospectral methods SGLS 05/19/08
M = 30; N = 25; MN = M*N;
[x,DM] = chebdif(M,2); x = (x+1)*0.5; D2x = DM(:,:,2)*4;
[y,DM] = chebdif(N,2); y = (y+1)*0.5; D2y = DM(:,:,2)*4;
[xx,yy] = meshgrid(x,y); xx = xx(:); yy = yy(:);
bx0 = find(xx==0); bx1 = find(xx==1);
by0 = find(yy==0); by1 = find(yy==1);
Ix = eye(M); Iy = eye(N); I = eye(MN);
L = kron(D2x,Iy) + kron(Ix,D2y);
L(bx0,:) = I(bx0,:); L(bx1,:) = I(bx1,:);
L(by0,:) = I(by0,:); L(by1,:) = I(by1,:);
b = zeros(MN,1); b(bx1) = y.^2; b(by1) = x;
u = L\b; u = reshape(u,N,M);
surf(x,y,u); colorbar
xlabel('x'); ylabel('y'); zlabel('u')
```

Exercise 15.2 *Solve Laplace's equation in three dimensions with the same boundary conditions as previously, but with* $u = 0$ *on* $z = 0$ *and* $u = 1$ *on* $z = 1$.

For another approach called the method of particular solutions, see Betcke & Trefethen (2005). This method has the historical feature of being the first to produce solutions of Laplace's equation in L-shaped domains, as in the Matlab logo. A related method can be used to deal with problems where one of the boundaries is now known in advance.

15.4 A pseudospectral approach for a mixed boundary value problem

We look at an example due to Read (2008). The hydraulic potential $\phi(x,y)$ satisfies Laplace's equation with boundary conditions

$$\phi(0,y) = h_1, \qquad \phi(s,y) = h_2 \tag{15.11}$$

along the vertical boundaries at $x = 0$ and $x = s$. Along the base of the aquifer, we have

$$\frac{\partial \phi}{\partial y}(x,0) = 0. \tag{15.12}$$

The condition at the soil surface located at $y = f(x)$ is mixed:

$$\begin{aligned} \phi(x,f(x)) &= f(x), \qquad 0 \le x < a, & (15.13)\\ \frac{\partial \phi}{\partial y}(x,f(x)) - f'(x)\frac{\partial \phi}{\partial x}(x,f(x)) &= R(x), \qquad a < x \le s, & (15.14) \end{aligned}$$

where $R(x)$ is the known vertical recharge. This condition can be rewritten in terms of the conjugate stream function $\psi(x,y)$, yielding

$$\begin{aligned} \phi(x,f(x)) &= f(x), \qquad 0 \le x < a, & (15.15)\\ \psi(x,f(x)) &= -\int R(x)\,\mathrm{d}x = r(x), \qquad a < x \le s, & (15.16) \end{aligned}$$

We can write solutions for ϕ and ψ that satisfy side and bottom boundary conditions:

$$\begin{aligned} \phi &= h_1 + \frac{h_2 - h_1}{s}x + \sum_{n=1}^{\infty} A_n \cosh\frac{n\pi y}{s}\sin\frac{n\pi x}{s}, & (15.17)\\ \psi &= A_0 + \frac{h_2 - h_1}{s}y + \sum_{n=1}^{\infty} A_n \sinh\frac{n\pi y}{s}\cos\frac{n\pi x}{s}, & (15.18) \end{aligned}$$

The unknown constant term A_0 represents the mass flux through the aquifer at the dam wall. Now define the following functions:

$$F(x,y) = \begin{cases} f(x) - h_1 - \dfrac{h_2 - h_1}{s}x, & 0 \le x < a,\\ f(x) - \dfrac{h_2 - h_1}{s}y, & a < x \le s \end{cases} \tag{15.19}$$

and

$$U_n(x,y) = \begin{cases} \cosh\dfrac{n\pi y}{s}\sin\dfrac{n\pi x}{s}, & 0 \le x < a,\\ \sinh\dfrac{n\pi y}{s}\cos\dfrac{n\pi x}{s}, & a < x \le s \end{cases} \tag{15.20}$$

with

$$U_0(x,y) = \begin{cases} 0, & 0 \le x < a,\\ 1, & a < x \le s. \end{cases} \tag{15.21}$$

The mixed boundary condition becomes

$$F^t(x) = F(x, f(x)) = \sum_{n=0}^{\infty} A_n U_n(x, f(x)) = \sum_{n=0}^{\infty} A_n U_n^t(x). \tag{15.22}$$

We truncate the sum at $n = N$ and enforce the condition at $N + 1$ collocation points

$$F^t(x_i) = F_i^t = \sum_{n=0}^{N} A_n U_n^t(x_i). \tag{15.23}$$

Read (2008) uses discrete least squares, taking more collocation points than unknowns.

Figure 15.3 shows ϕ and ψ on the upper boundary for the parameter values $h_1 = 1$, $h_2 = 1.5$, $s = 10$, $a = 5$ and $f(x) = 1 + (1 - \cos \pi x/10)/8$. The value of $N = 300$, $M = 2N$ used in Read (2008) failed here; these results are for $N = 80$. The results look comparable.

```
% p153.m Mixed Laplace bvp SGLS 05/20/08
N = 80; M = 2*N; n = 0:N;
h1 = 1; h2 = 1.5; s = 10; a = 5;
x = linspace(0,10,M+1)';  f = 1 + (1-cos(pi*x/10))/8;
F = (f-h1-(h2-h1)/s*x).*(x<a) - (h2-h1)/s*f.*(a<=x);
[n,xx] = meshgrid(n,x); ff = 1 + (1-cos(pi*xx/10))/8;
U = cosh(n*pi.*ff/s).*sin(n*pi.*xx/s).*(xx<a) + sinh(n*pi.*ff/s).*cos(n*pi.*xx/s).*(a<=
U(:,1) = (a<=xx(:,1));
A = U\F;
phi = h1 + (h2-h1)/s*x + (cosh(n*pi.*ff/s).*sin(n*pi.*xx/s))*A;
psi = (h2-h1)/s*f + (sinh(n*pi.*ff/s).*cos(n*pi.*xx/s))*A;
subplot(2,1,1)
plot(x,phi)
xlabel('x'); ylabel('\phi')
subplot(2,1,2)
plot(x,psi)
xlabel('x'); ylabel('\psi')
```

Exercise 15.3 *Investigate why* `p153.m` *seems to fail around* $N = 80$*, far below Read's value.*

15.5 The Boundary Integral Method

Another approach for solving Laplace's equation is the Boundary Integral Method (BIM). A detailed exposition can be found in Pozrikidis (2002). This method also works for the Helmholtz equation, Stokes flow problem, and so on. The method is too lengthy to describe in detail. Figure 15.4 shows some results using it.

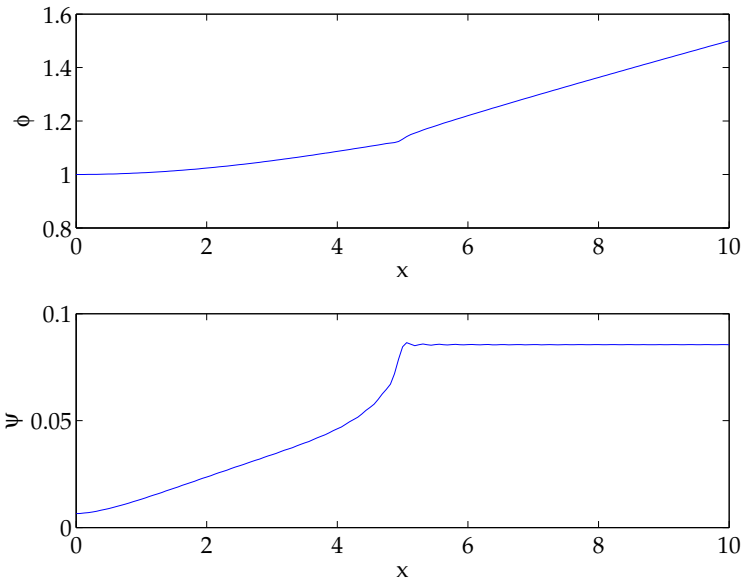

Figure 15.3: Solution to mixed boundary-value problem using collocation.

15.6 References

Publications:

- Betcke, T. & Trefethen, L. N. Reviving the method of particular solutions. *SIAM Rev.*, **47**, 469–491. 2005.
- Howison, S. D., Morgan, J. D. & Ockendon, J. R. A class of codimension-two free boundary problems. *SIAM Rev.*, **39**, 221–253, 1997.
- Pozrikidis, C. *A practical guide to boundary element methods with the software library BEMLIB*, Chapman & Hall/CRC, Boca Raton, 2002.
- Read, W. W. An analytic series method for Laplacian problems with mixed boundary conditions. *J. Comp. Appl. Math.*, **209**, 22–32.

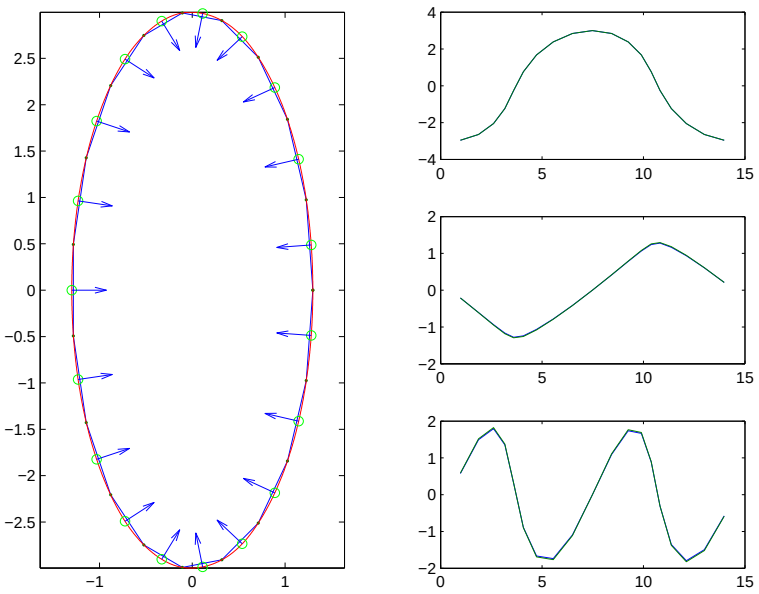

Figure 15.4: Added mass coefficients computed for a two-dimensional elliptical body using the BIM.

Chapter 16

Conformal mapping (05/22/08)

16.1 Conformal maps

We can think of complex-valued functions of complex arguments as mappings from one complex number to another: $z \mapsto f(z)$. The class of complex mappings is obviously slightly more restricted than that of real mappings, since f is a function of z, and not x and y separately.

Consider the origin without loss of generality. Then the angle between the images of the two points z_1 and z_2 is the argument of the ratio

$$\frac{f(z_2)}{f(z_1)} = \frac{z_2 f'(0) + \cdots}{z_1 f'(0) + \cdots} = \frac{z_2}{z_1} + \cdots, \tag{16.1}$$

assuming that f is analytic at the origin. But this means that the angle between $f(z_2)$ and $f(z_1)$ is the same as the angle between z_2 and z_1 unless $f'(0) = 0$. Hence analytic functions give maps that preserve angle and are called conformal.

At points where f' vanishes, the angle can change. This is crucial, since any interesting conformal map needs to change the shape of some boundary or curve, which can only happen at such points.

There are of course infinitely many conformal maps. The ones that are commonly used boil down to polynomials, powers, Möbius maps, which are really combinations of linear maps and the function z^{-1}, exponential and logarithms, and trigonometric and hyperbolic functions, which are just elaborations of exponentials. To start, remember that adding a constant is a translation, while multiplication by a complex number a is a stretch-scaling: points are rotated about the origin by $\arg z$ and their magnitude is multiplied by $|a|$.

I only know one Möbius map, but all Möbius maps are basically the same

$$w = \frac{1 - z}{1 + z}. \tag{16.2}$$

One has to know that Möbius maps take circles to circles, lines to lines, and circles that go through singularities to lines and vice versa. They form a group under composition, and their effect can be determined by knowing just the images of three points. Here 1 goes

to 0, 0 goes to -1 and -1 goes to infinity. It's also convenient to know that i and $-$i are mapped onto themselves. The picture is two counter-rotating disks: the area in the first quadrant outside the unit disk gets squeezed into the quarter-circle inside the unit disk in the first quadrant, that area goes one hop to the left into the unit disk in the second quadrant, which goes to the remainder of the unit disk, which finally "rotates" about i back to the first quadrant. The lower half-plane is the symmetric image of this.

The exponential map $z \mapsto \mathrm{e}^z = \mathrm{e}^x(\cos y + \mathrm{i}\sin y)$ is periodic in y and conformal everywhere in the finite plane. Strips with height 2π are mapped onto the entire plane, so the map is one-to-many. The logarithmic map $z \mapsto \log z = \log r + \mathrm{i}\theta$ requires choosing a cut. The entire complex plane is mapped onto a strip with height 2π.

Anything more complicated requires practice. It is extremely useful to have Matlab to plot the result of mappings.

```
% p161.m Conformal maps examples SGLS 05/21/08
clf
for j = -10:10
  x = j*0.5; y = -10:.01:10; z1 = x + i*y;
  y = j*0.5; x = -10:.01:10; z2 = x + i*y;
  f1(1,:) = 1./z1; f2(1,:) = 1./z2;
  f1(2,:) = (z1-1)./(z1+1); f2(2,:) = (z2-1)./(z2+1);
  f1(3,:) = exp(z1); f2(3,:) = exp(z2);
  f1(4,:) = log(z1); f2(4,:) = log(z2);
  for k = 1:4
    subplot(2,2,k)
    hold on; plot(real(f1(k,:)),imag(f1(k,:)),real(f2(k,:)),imag(f2(k,:)),'+.'); hold o
    axis([-5 5 -5 5])
  end
end
```

Laplace's equation is conformally invariant. This is obvious from the complex variable framework. We can compose analytic functions as we like and construct solutions to Laplace's equation in this fashion.

16.2 Examples

16.2.1 Potential flow

Potential flow provides a great opportunity to use conformal maps. The typical problem is to find the flow past an obstacle of a certain shape in an oncoming stream with unit velocity in the x-direction. As a quick refresher, we remind ourselves that in potential flow, the fluid velocity is irrotational, so that there exists a velocity potential ϕ with $\mathbf{u} = \nabla\phi$. If the flow is also incompressible, $\nabla\cdot\mathbf{u} = \nabla^2\phi = 0$. Hence an analytic function $f = \phi + \mathrm{i}\psi$ gives a velocity field. The imaginary part of f is the streamfunction, which is

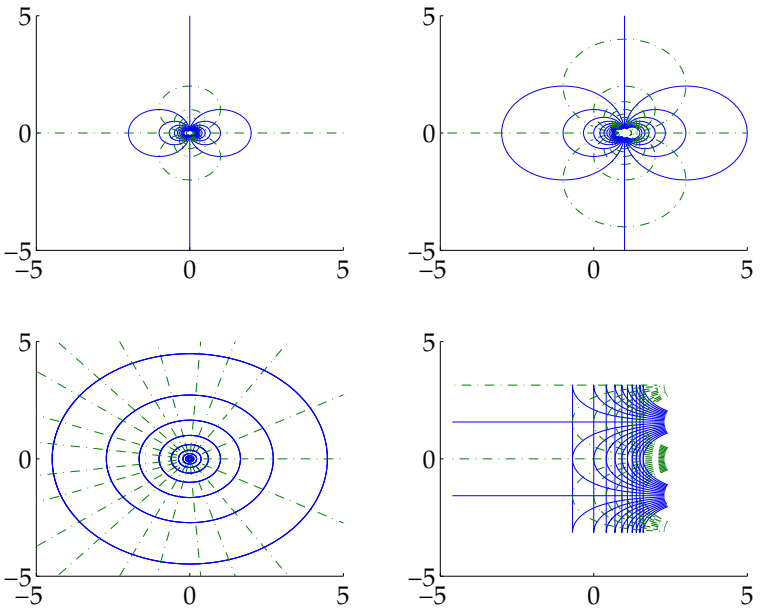

Figure 16.1: Examples of complex mappings. Shown are the images of horizontal and vertical lines in the z-plane. (a) $z \mapsto 1/z$; (b) $z \mapsto (1-z)/(1+z)$; (c) $z \mapsto \exp z$; (d) $z \mapsto \log z$.

constant along lines with no-normal flow, such as solid boundaries. The flow is taken to be inviscid, otherwise we have an extra boundary condition and viscosity to worry about.

This is useful if we know solutions to this problem in a simple geometry, and we do. In the upper (or lower) half-plane, z is the analytic potential corresponding to uniform flow from left to right. For inviscid flow we need only satisfy no normal flow, but the velocity field is just $(1,0)$ so this is automatic. Flow past a cylinder is also simple. The velocity potential is $w(z) = z + 1/z$. We can see this work without any real work. At infinity, the potential looks like z – check. Along the unit circle, $w = \mathrm{e}^{\mathrm{i}\theta} + \mathrm{e}^{-\mathrm{i}\theta} = 2\cos\theta$, which is real; hence the streamfunction is zero along the unit circle, which can hence be a material surface – check.

An excellent source for conformal maps and potential flow is Milne-Thomson (1996). Note that the definition of complex potential has a minus sign in front of it compared to the standard modern use.

16.2.2 Flow through an aperture

It can be useful to consider the inverse mapping going from $w = f(z)$ to z. For example. $z = c\cosh w$ gives

$$x = c\cosh\phi\cos\psi, \qquad y = c\sinh\phi\sin\psi. \tag{16.3}$$

We can eliminate ϕ and obtain

$$\frac{x^2}{c^2\cos^2\phi} - \frac{y^2}{c^2\sin^2\phi} = 1, \tag{16.4}$$

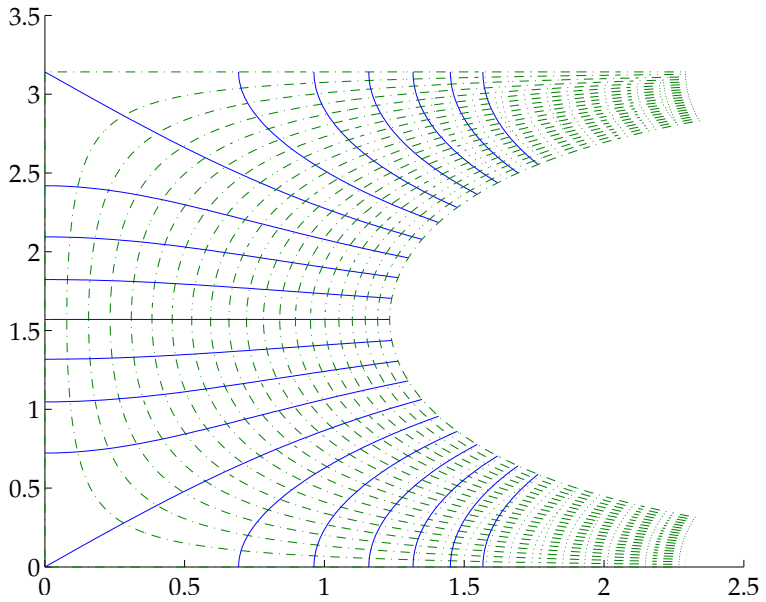

Figure 16.2: Streamlines and isopotentials for flow through an aperture.

so the streamlines are confocal hyperbolas with foci at $(\pm c, 0)$. Any streamline can again be taken as a solid boundary. Figure 16.2 illustrates the streamlines. Note that because of Matlab's (sensible) convention in computing inverse hyperbolic functions, we only get the right half of the picture.

```
% p162.m aperture conformal map SGLS 05/22/08
clf
for j = 0:20
  x = (j-10)*0.5; y = pi*(0:.01:1); z1 = x + i*y;
  y = j*pi/20; x = -10:.01:10; z2 = x + i*y;
  f1 = acosh(z1/2); f2 = acosh(z2/2);
  hold on; plot(real(f1),imag(f1),real(f2),imag(f2),'-.'); hold off
end
```

Exercise 16.1 *Obtain the flow past an elliptic cylinder using the mapping* $z = c \cos w$.

16.2.3 Problems with branch cuts

At this point, one could consider Zhukovsky/Joukowski (Cyrillic transliteration issues) mappings, but instead let's look at the map $w = a\pi U \coth a\pi/z$. Explicitly,

$$w = a\pi U \coth\left(\frac{a\pi x}{r^2} - \frac{a\pi \mathrm{i} y}{r^2}\right). \tag{16.5}$$

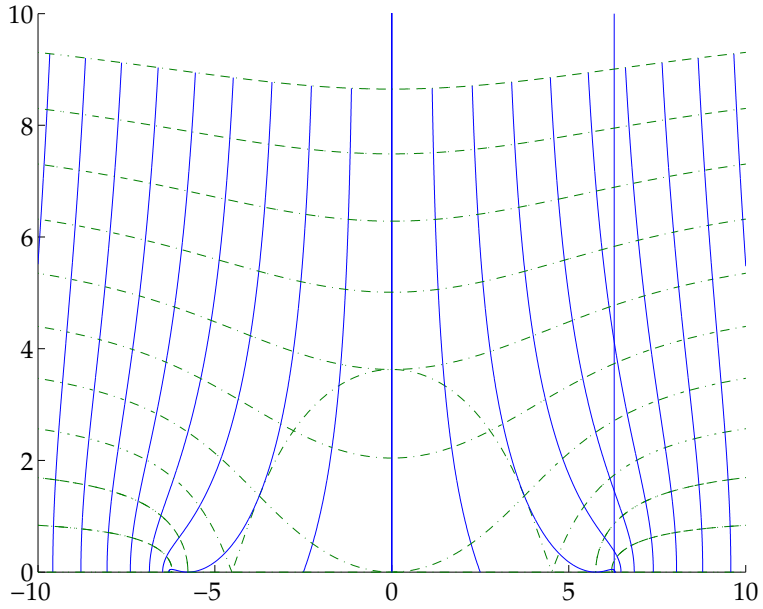

Figure 16.3: Streamlines and isopotentials for flow over a log.

When $y = 0$ or $a\pi y/r^2 = \pi/2$, the argument of coth is real and w is real: these are streamlines. They corresponds to the x-axis and to the circle of radius a with center $(0, a)$. For large values of $|z|$, $w \sim Uz$, i.e. uniform oncoming flow. Hence we should be seeing flow over a log so to speak.

Figure 16.3 shows isopotentials and streamlines in the upper half-plane. Clearly something is wrong since we have intersecting streamlines.

```
% p163.m log conformal map SGLS 05/22/08
clf
for j = -10:10
  x = j; y = -0:.01:10; z1 = x + i*y;
  y = j; x = -10:.01:10; z2 = x + i*y;
  f1 = 2*pi*coth(2*pi./z1); f2 = 2*pi*coth(2*pi./z2);
  hold on; plot(real(f1),imag(f1),real(f2),imag(f2),'-.'); hold off
    axis([-10 10 0 10])
end
```

Exercise 16.2 *Find the problem and fix it.*

16.2.4 Free surface flows

Free surface flows are industrially important. There is an extensive literature on inviscid, potential, free surface flows. The standard references are Birkhoff & Zarantonello (1957) and Gurevich (1965). These approaches depend crucially on conformal maps. The free

surface is unknown and has to be found as part of the solution. One has a kinematic and a dynamic boundary condition on the surface. These reduce to $|\mathrm{d}w/\mathrm{d}z| =$ constant on the free boundary.

For flows bounded by flat plates and free streamlines, one can us the hodograph, which is locus of values of the complex velocity $\mathrm{d}w/\mathrm{d}z$. On free streamlines, it must be on the unit circle. The theory for non-flat boundaries is much more complicated. Free streamlines are used to compute Helmholtz flows, in which there is a region of stagnant fluid behind obstacles.

The results tend to be messy and hard to visualise, especially because they are usually given for the hodograph.

Exercise 16.3 *Take the simple solution (8) of p. 29 of Birkhoff & Zarantonello (1957) and plot it. This may require solving some ODEs.*

A classic free surface flow is the Borda mouthpiece flow. Figure 16.4 shows the streamlines. The relevant mapping is

$$F(z) = 1 + g(z) + \log g(z), \qquad g(z) = z^2 + z\sqrt{z^2+1} \tag{16.6}$$

and the streamlines are given by the rays with $\arg z$ constant.

However, this is a notorious example[1] (see `http://www-personal.umich.edu/~williams/archive/f` of a calculation that can be wrecked by a poor choice of branch cuts. The issue is actually more technical than that, and involves the concepts of $+0.0$ and -0.0 that are supposed to be supported by the IEEE754 standard.

Warning 13 *This is a good opportunity to remind ourselves of the danger of poorly programmed elementary functions, logarithms and the use of extra digits accuracy. Go to W. Kahan's home page and read the documents there.*

```
% p164.m Borda mouthpiece SGLS 05/22/08
r = logspace(-3,3,61);
clf
for j = -10:10
  t = pi*j*0.05;
  z = r*exp(i*t);
  g = z.^2 + z.*sqrt(z.^2+1);
  F = 1 + g + log(g);
  hold on; plot(real(F),imag(F)); hold off
end
axis([-4 8 -7 7]); axis square
```

Exercise 16.4 *Why does this work when the website cited implies it shouldn't?*

[1] Notorious for those who care about such things – there may be 5 or 6 of us.

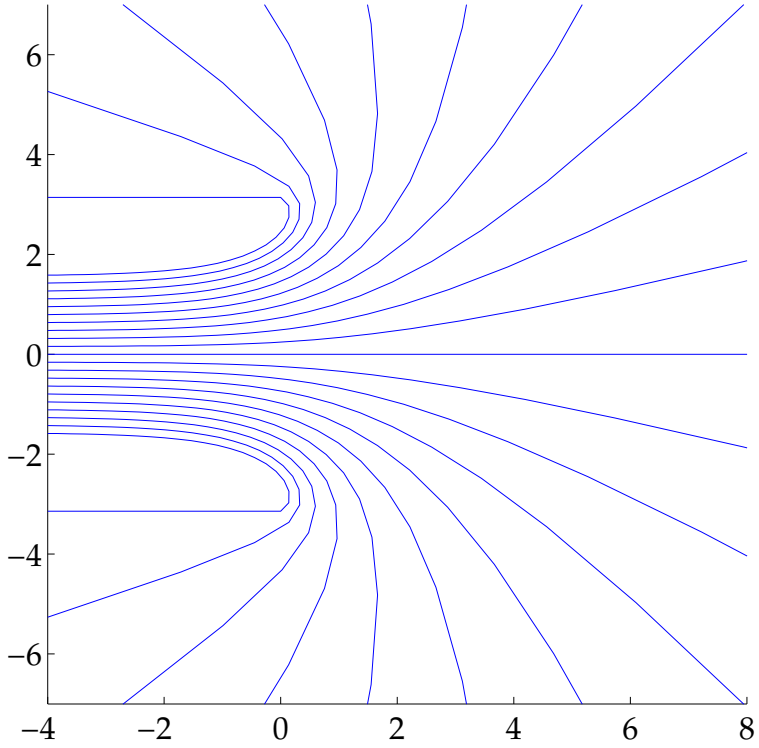

Figure 16.4: Streamlines for flow into the Borda mouthpiece.

16.2.5 The Schwarz formula

An interesting application of conformal maps can be found in Llewellyn Smith, Michelin & Crowdy (2008). The classical theory of the motion of a solid through an inviscid fluid was developed by Kirchhoff and subsequently by Thomson and Tait. The coupled fluid-body system, which in its primary formulation requires solving the Euler equations and Newton's equations for the body, can be replaced by a system of ordinary differential equations that take into account the forces and couples exerted on the solid by the fluid. The flow is taken to be irrotational, which is the case if the body starts from rest, for example. It can be shown that the nature of the far-field motion depends crucially on an object called the virtual mass tensor.

The symmetry properties of the virtual mass tensor are discussed in Lamb (1945; § **126**), where it is pointed out that "as might be anticipated from the complexity of the question, the physical meaning of the results is not easily grasped". In Lamb's Cartesian notation we can take $w = p = q = 0$, so that the coefficients corresponding to m_{16} and m_{26} are G and F respectively. All other "cross-terms" (combining linear and angular velocity) vanish. Then, considering the two-dimensional body as being extended indefinitely in the third dimension, F and G vanish in the following cases (the headings are taken from Lamb):

3°. If the body has two axes of symmetry at right angles.

4° and 5°. If the body is a circle or a regular polygon.

7°. If the body is symmetric under a rotation by a right angle.

It is important to point out that these are sufficient, but not necessary, conditions. Can we find a sufficient and necessary condition? Since this is a problem concerning shapes of two-dimensional bodies, it is natural to use conformal mapping theory.

We can reduce the problem to constructing a conformal map $z = f(\zeta)$ from the interior of the unit disk to the outside of a simply connected body. The complex potential $w = \phi + \mathrm{i}\psi$ satisfies the following boundary value problem

$$\mathrm{Re}[-\mathrm{i}W(\zeta)] = \psi = -\frac{\Omega}{2} z\overline{z} - \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} z + \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} \overline{z} \qquad \text{on } |\zeta| = 1. \tag{16.7}$$

Since $W(\zeta)$ must be analytic in $|\zeta| \leq 1$, this is just the Schwarz problem for the unit disk and its solution is given by the Poisson integral formula:

$$-\mathrm{i}W(\zeta) = \frac{1}{2\pi\mathrm{i}} \oint_{|\zeta'|=1} \frac{\mathrm{d}\zeta'}{\zeta'} \left(\frac{\zeta' + \zeta}{\zeta' - \zeta} \right) \left[-\frac{\Omega}{2} z\overline{z} - \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} z + \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} \right]. \tag{16.8}$$

The condition that there be no dipole field at infinity becomes

$$\oint_{|\zeta'|=1} \frac{\mathrm{d}\zeta'}{\zeta'^2} \left[-\frac{\Omega}{2} z\overline{z} - \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} z + \frac{\mathrm{i}Ue^{\mathrm{i}\alpha}}{2} \right] = 0. \tag{16.9}$$

One can construct mappings satisfying this condition, but a critical point is that the mappings need to be univalent, which means meromorphic and single-valued, so that $f(z_1) = f(z_2)$ implies $z_1 = z_2$. This an important problem in advanced complex analysis.

16.3 The Bieberbach conjecture

The most famous such problem is the Bieberbach conjecture. Consider the class Σ_0 of functions that are univalent and analytic in $|z| < 1$, normalized to have Taylor series expansions for the form

$$f(z)z + \sum_{n=2}^{\infty} a_n z^n. \tag{16.10}$$

The Bieberbach conjecture states that $|a_n| \leq n$ for $n \geq 2$. This conjecture was stated in 1916 by Bieberbach, and proved by de Branges in 1984.

The ideas developed to prove the conjecture are very sophisticated, but many aspects of the proof are fairly elementary. A good account is given in Henrici, vol. III. The original proof used the Askey–Gasper theorem, which is a result in the theory of orthogonal polynomials. We shall meet orthogonal polynomials in Chapter 19.

16.4 Schwarz–Christoffel mappings

Henrici, vol. III, also discusses the numerical construction of conformal maps. There are situations where one knows a conformal map exists, but its explicit form can only be found numerically, for example by solving an integral equation.

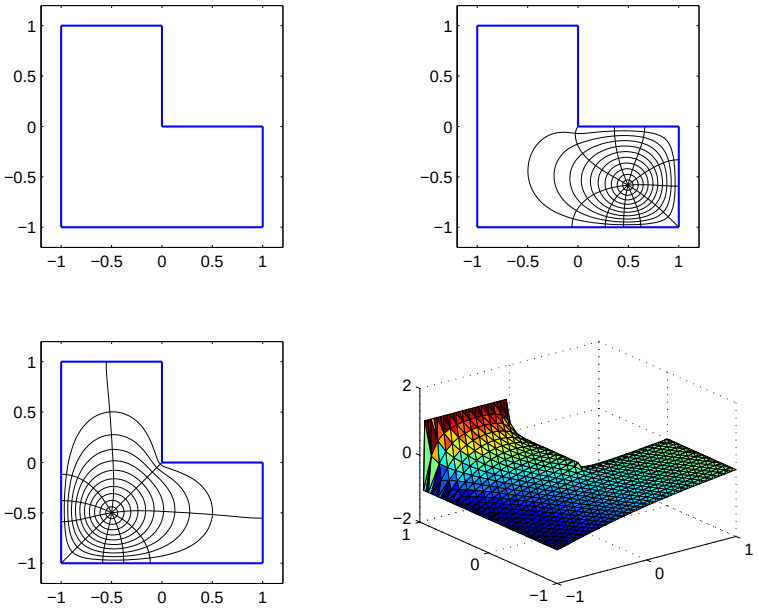

Figure 16.5: Example of use of the SC toolbox.

A very useful class of mappings is given by the Schwarz–Christoffel formula. This gives a recipe for a conformal map from the upper half-plane to the interior of a polygon with vertices $w_1, \ldots, w_n$. The interior angles at the vertices are $\alpha_1\pi, \ldots, \alpha_n\pi$. The pre-images of the vertices (prevertices) are along the real axis at $z_1 < z_2 < \cdots < z_n$. We can also have vertices at infinity. The Schwarz—Christoffel formula is

$$f(z) = f(z_0) + c \int_{z_0}^{z} \prod_{j=1}^{n-1} (\zeta - z_j)^{\alpha_j - 1} \, \mathrm{d}\zeta. \tag{16.11}$$

The problem is that the prevertices cannot usually be computed analytically.

The Schwarz–Christoffel (SC) toolbox written by Toby Driscoll that constructs these maps. It consists of a number of Matlab programs. The toolbox can also solve Laplace's equation in such domains. Figure 16.5 shows an example of using the toolbox.

```
% p165.m SC toolbox example (based on TD example) SGLS 05/22/08
p = polygon([i -1+i -1-i 1-i 1 0])
subplot(2,2,1); plot(p)
subplot(2,2,2); f = diskmap(p); f, plot(f)
subplot(2,2,3); f = center(f,-0.5-0.5i), plot(f)
subplot(2,2,4); phi = lapsolve(p,[1 -1 NaN 0 NaN 0]');
[tri,x,y] = triangulate(p); trisurf(tri,x,y,phi(x+i*y));
```

16.5 Fuchsian differential equation

Things can get even more complicated. See Craster (1996) for a discussion of mapping a curvilinear quadrangle to a half-plane. One needs to solve things called Fuchsian differential equations.

16.6 References

Publications:

- Birkhoff, G. & Zarantonello, E. H. *Jets, wakes, and cavities*, Academic, New York, 1957.
- Craster, R. V. Conformal mappings involving curvilinear quadrangles. *IMA J. Appl. Math.*, **57**, 181–191; 1996.
- Llewellyn Smith, S. G., Michelin, S. & Crowdy, D. G. The dipolar field of rotating bodies in two dimensions. In press, *J. Fluid Mech.*, 2008.
- Gurevich, M. I. *Theory of jets in ideal fluids*, Academic, New York, 1965.
- Lamb, H. *Hydrodynamics*, 6th ed, Dover, New York, 1945.
- Milne-Thomson, L. M. *Theoretical hydrodynamics*, 5th ed, Dover, New York, 1996.

Web sites:

- http://www.cs.berkeley.edu/~wkahan/
- http://www.math.udel.edu/~driscoll/software/SC/
- http://www-personal.umich.edu/~williams/archive/forth/complex/borda.html

Chapter 17

Elliptic functions (05/27/08) [15 min. short]

17.1 Problem 10 of the Hundred-dollar, Hundred-digit Challenge

The last problem of the Hundred-dollar, Hundred-digit challenge was the following:

> A particle at the center of a 10×1 rectangle undergoes Brownian motion (i.e., two-dimensional random walk with infinitesimal step lengths) until it hits the boundary. What is the probability that it hits at one of the ends rather than at one of the sides?

Here is what the team of Glenn Ierley, Stefan Llewellyn Smith and Bob Parker wrote in their submission to Nick Trefethen :

> By well-known theory, the probability is the value of a harmonic function in the center of the rectangle with boundary conditions of zero on long edges and unity on the short ones. There are several ways to solve this classic problem: one method, suggested by Bill Young, uses the analytic solution in a semi-infinite rectangle, and the method of images. Another is the closed-form result via the Schwarz–Christoffel transformation (Morse and Feshbach, 1953). Only one edge is at unity here, hence the initial factor of two:
>
> $$P = 2(2/\pi)Im\ln(\mathrm{sn}[(K+iK')/2,k]) = \frac{2}{\pi}\arcsin(k')$$
>
> where $k' = \sqrt{1-k^2}$, k is the modulus, and K is the quarter-period of the Jacobi elliptic function $sn(u,k)$ associated with the nome $q = \exp(-\pi/10)$; K' is the complementary quarter period. If $Q = \exp(-10\pi)$ we find the explicit infinite product:
>
> $$k' = 4\sqrt{Q}\prod_{n=1}^{\infty}[(1+Q^{2n})/(1+Q^{2n-1})]^4$$

which converges at a more than adequate rate, since $Q \approx 2.27 \times 10^{-14}$. Here some old-fashioned analysis and a hand calculator are sufficient. In any event,

$$\begin{aligned} P &= 10^{-6} \times .383758797925122610340713318620483910079300559407 \\ &\quad 250956903002279917343660685274327650084284564726991 0 \end{aligned}$$

Notice the reference to SC.

Let us go through a number of approaches and see how they do and then discuss elliptic functions. Bornemann *et al.* (2004) have a through discussion that is well worth reading. This chapter is more about this problem than elliptic functions.

We can find the probability $u_h(x, y)$ that a particle starting at an arbitrary lattice point (x, y) and taking random steps of length h left, right, up or down, reaches the ends:

$$u_h(x, y) = \frac{1}{4}[u_h(x + h, y) + u_h(x - h, y) + u_h(x, y + h) + u_h(x, y - h)] \tag{17.1}$$

with $u_h = 1$ at the ends and $u_h = 0$ along the edges. Now we take the limit $h \to 0$ and find $\nabla^2 u = 0$ with boundary conditions $u = 1$ at the ends and $u = 0$ along the edges. What we are looking for is then $p = u(0, 0)$.

17.2 Fourier solution

As in Chapter 15, we can solve this using Fourier series. The result is in Bornemann *et al.* (2004):

$$u(x, y) = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \frac{\cosh[(k + 1/2)\pi x/b]}{\cosh[(k + 1/2)\pi a/b]} \cos[(k + 1/2)\pi y/b], \tag{17.2}$$

where the result is for the rectangle $(-a, a) \times (-b, b)$. At the origin this gives

$$p = \frac{4}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \operatorname{sech}[(2k + 1)\pi\rho/2], \tag{17.3}$$

where $\rho = a/b = 10$ here. This series converges very very well:

$$p = \frac{4}{\pi} \operatorname{sech} 5\pi - \frac{4}{3\pi} \operatorname{sech} 15\pi + \cdots = 3.837587979251258 \times 10^{-7} - 2.905183386488490 \times 10^{-21} + \cdots \tag{17.4}$$

We are basically done with one term, and what's more we then have

$$p \approx \frac{4}{\pi} \operatorname{sech} 5\pi \approx \frac{8}{\pi} e^{-5\pi} = 3.837587979251345 \times 10^{-7} \tag{17.5}$$

to 13 significant digits.

Exercise 17.1 *Derive (17.2).*

17.3 Pseudospectral solution

This is just a simple adaptation of the approach of Chapter 15. The size of the matrices is limited by memory size. One can do much better by using a direct solver for the problem and not constructing the large $MN \times MN$ matrix explicitly. However, even with less than the largest size of matrix that fits in memory, we can obtain essentially machine accuracy for p. The changes from the approximate result (17.5) are of the order of machine accuracy.

```
% p172.m 100$100d using pseudospectral methods SGLS 05/26/08
for j = 1:10
  M = 20*j+1, N = 2*j+1,; MN = M*N
  [x,DM] = chebdif(M,2); x = (x+1)*5; D2x = DM(:,:,2)*0.04;
  [y,DM] = chebdif(N,2); y = (y+1)*0.5; D2y = DM(:,:,2)*4;
  [xx,yy] = meshgrid(x,y); xx = xx(:); yy = yy(:);
  bx0 = find(xx==0); bx1 = find(xx==10);
  by0 = find(yy==0); by1 = find(yy==1);
  Ix = speye(M); Iy = speye(N); I = speye(MN);
  L = kron(D2x,Iy) + kron(Ix,D2y);
  L(bx0,:) = I(bx0,:); L(bx1,:) = I(bx1,:);
  L(by0,:) = I(by0,:); L(by1,:) = I(by1,:);
  b = zeros(MN,1); b(bx0) = 1; b(bx1) = 1;
  u = L\b; u = reshape(u,N,M);
  u((N+1)/2,(M+1)/2)
  u((N+1)/2,(M+1)/2) - 8/pi*exp(-5*pi)
  whos
end
surf(x,y,u); colorbar
xlabel('x'); ylabel('y'); zlabel('u')
```

17.4 Method of images

This is the solution used in our 100$100d entry. We used Maple and the Fortran MP package. The latter has been superseded by the Arprec package, but still works just fine. See below for the relevant web site. David H. Bailey is the person to thank for this outstanding software. Getting both packages to work is a good test for your Fortran 77, Fortran 90 and C++ compilers. This reflects poorly on most compilers, not on the packages. Note that Mpfun77 requires a re-entrant Fortran compiler, and that gfortran chokes (or at least used to choke) on Arprec; g95 does OK.

The mathematical basis for this approach is simple. The solution in the semi-infinite domain $x > 0$, $0 < y < 1$ with boundary condition $u(0,y) = 1$ is simple to obtain. Probably the easiest way is to consider the analytic function

$$f = \frac{2}{\pi} \tanh \log \pi z \tag{17.6}$$

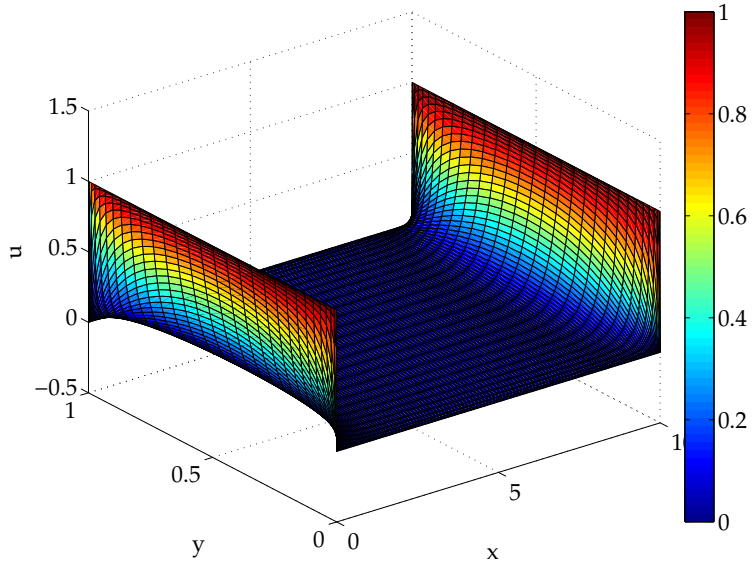

Figure 17.1: Solution to Laplace's equation for Problem 10 using the pseudospectral method.

This solution does not satisfy the boundary condition at $x = 10$ of course. But we can add $u(20 - x, y)$ or something. Now the boundary condition at $x = 10$ is OK but the boundary condition at $x = 0$ is wrong. Fix it by adding $u(-20 + x, y)$, so now we've messed up the boundary condition at $x = 10$. And so on. This is a rapidly converging series.

Exercise 17.2 *Tidy up the argument above.*

The Maple program is very simple and runs quickly. The last value output by it is

```
f :=
0.383758797925122610340713318620483910079300559407250956903002279917 34\
    366068527432765008428456472699101535338727783178038311554466024291385 50\
    885041034413877481608759295735796764777307732929517401839843039313801 64\
    791007249511812009415206805319597352138098534521313987700776507826197 31\
    493901775746888998844446809863262645992082227863113899932239658258145 41\
    304704619981090190291217606459112130890215289906014907236375669781752 07\
    616848068011174956830494480432650358233414269466258803728261620508826 06\
    388095332287999044152330344651386604646042111243751975386985732606495 92\
    385738647751091878362605260951809030770838782403065485925222816074543 10\
    473227442385515420967787023851279873869420019901074412797715216621155 02\
    028226638041673315386936093582553902408902176199164972194717846326025 86\
    337758107419192413974681745158181231843202203416063899323259130536841 27\
    735824884531580310731812130696089510387417482494827793439666780933960 72\
    568353941095950812406441092575993038188872862194446236425749926009229 47\
    153363285561594410413234561576168439159723323849298274286426095614228 20\
    436858059610372083132011650519264785853520358038355033539991113234496 53\
```

```
462687211593228661953150802027165314947468839691970311566385599305377 11\
922238562772098876560539673874560114374142911118927007360702336221969 00\
698646583618130008010449397943037934407697219631179407400200841144126 72\
354708735740829408717285324975240586801053974269678520283564844115474 98\
237394238272629968237617924876537506151488084048607496766811382294672 02\
542418375871379079979761209600451211448686637289444345102156219105705 56\
508108065091822949873429501764049479016466409786690510310784470379023 02\
050326935184466633194574502329979031396970386339603363674811197881317 03\
853958190084458079353530498674753846839745266812041707955356314355704 47\
310005491064577967313017391007734112915771327973875055197069948919325 80\
144301452874832271956807294877003716312687330907907768144468950988074 50\
680074771056230518061061950439411079950576411633117899679229690442573 06\

                   -6
963964346150118 10
```

```
# Maple solution of 100$100d problem SGLS 05/26/08
Digits:=2000;
L:=10;
f:=2*evalf(coeff(evalc(2/Pi*log(tanh(Pi*(L/2+I/2)/2))),I));
for j from 1 to 150 do
   f:=f+2*(-1)^j*evalf(coeff(evalc(2/Pi*log(tanh(Pi*(L/2+j*L+I/2)/2))),I));
od;
```

The MP Fortran program is also simple. For interest, its MP translated version is also included. It does not run particularly fast. The last value returned is

```
10 ^        -7 x  3.837587979251226103407133186204839100793005594072509569030 0
227991734366068527432765008428456472699101535338727783178038311554466024291385
508850410344138774816087592957357967647773077329295174018398430393138016479100
724951181200941520680531959735213809853452131398770077650782619731493901775746
888998844446809863262645992082227863113899932239658258145413047046199810901902
912176064591121308902152899060149072363756697817520761684806801117495683049448
043265035823341426946625880372826162050882606388095332287999044152330344651386
604646042111243751975386985732606495923857386477510918783626052609518090307708
387824030654859252228160745431047322744238551542096778702385127987386942001990
107441279771521662115502028226638041673315386936093582553902408902176199164972
194717846326025863377581074191924139746817451581812318432022034160638993232591
305368412773582488453158031073181213069608951038741748249482779343966678093396
072568353941095950812406441092575993038188872862194446236425749926009229471533
632855615944104132345615761684391597233238492982742864260956142282043685805961
037208313201165051926478585352035803835503353999111323449653462687211593228661
953150802027165314947468839691970311566385599305377119222385627720988765605396
738745601143741429111189270073607023362219690069864658361813000801044939794303
793440769721963117940740020084114412672354708735740829408717285324975240586801
```

0539742696785202835648441154749823739423827262996823761792487653750615148808404860749676681138229467202542418375871379079979761209600451211448686637289444345102156219105705565081080650918229498734295017640494790164664097866905103107844703790230205032693518446663319457450232997903139697038633960336367481119788131703853958190084458079353530498674753846839745266812041707955356314355704473100054910645779673130173910077341129157713279738750551970699489193258014430145287483227195680729487700371631268733090790776814446895098807450680074771056230518061061950439411079950576411633117899679229690442573069639643461501208059982174767014766705410365029230935018522109401684181408228826020356515662085313913703345581635833915610975466097955736369711500452638574537951816688922476673008815420573004734226770092375385436642969302790812741068492324010880265192917007711330673991892618510246540764889920892470120114930293433022394430764300881501268861618182222878047926487402311948763153004087250289587842463725517015830095240397904461348489307910337109487509090959525706413453440240782130680700341415228841173220757650111290406013967993269834955060476450004923439248806524326880078473346801359325200468451592475829831270961407018670088872005759149971574640143256675659668789597474668751795722075345582481513155925308912263470449274628561755344515237893670759671169646330135519519363829429465251011014890994245077042774933300062722779520418548836778836022639573225625712583138782140249161109344476234978194918510156104568137260251480859149682771854897156793949780168404272814814784951942448913091684326832799649275174000736097991993462092713972053054966237898824332152918911875508981741685361726245276788718332170826797195765668580119326542975744300677006892079133872144504847811170132390988274262441112833304913832233667080275859732626288920121139257513788356044965289788551863186312127601080353258051536875339532643369416027702852674822985615951453811982875457002407470361595299159382435991338202109773077899072834354848026647755030701712497738702342319177923796877108674926099819031812582116110606572879248970738186802525400542018381988691891007150642184436889934067219396731906381656652677102756427601883117425917050602622211183734341519925163409816554277797392886992253166752388713447859068191194164640038639870120860910979909771800123285235803190596952081857349347995602010004369680911842563705988930570208791004552928027203341684142831226658791845593834413893463364778075653025215404560224517044378920469135715158935837407368039438142338962249960119782168317027359889449183961389268287143310562767567752609593795900770831639570040851794653232696713197967292029935640426602603710401638218705784077017233019238898980904808544582180596568616232136851559489680780530766919364210866283443180645843971597540484224880890184596388142055795743859527270551007874807558980641370522523000836234539567722469464603517544775140767575126957909520613514155654777972397255117781865624646734969332136316565488686294753197306478900144097926572075521096926096108350837481611117776460044982575901520944085697955294156392815194835343730996192079325351247023960686968217072294228189205503602901091214550344403839730393194505939788430302409589849608024436632722537218225354018735008495353224842497102233902660578161358255467364092668769038208963103079498896770410367530238807876413176752902338423393552857988256823694979292490670911300318908818135947417831642904191579007118618932464601597823412418524509295318927794880719201075389559814

741544435891957512005201332573332950455786206875519010098795434706738359230172
839398729498205091747665027608221089579372059716906661240870745292877246695041
685792821791951347029422095500456320247997440486760109985915983893486523487474
712918126979714253643222439345846068439608502483338521641759650135087367895565
227775670531701010610512050952318520451449897746610200236918960828278615211943
601147657922032172995884801059364503760533245769540079136430798865410589056188
013796408043764383784752833056930967597416210273116148959327991407403747165016
506717230029055599654218661602334929203352939117039448324293760852247805540546
992787525423842119910114068164112790210983163595657316260207596608041626933207
051338704454215232562324969346604296114710411699539008400173967573525593551717
225304474901068609413953754821790909442236636631991136198428210058671631343392
804179459138371156047620046142665220953238925227133041289183115388523876916613
798399566042545664465506317065265027977820444757006994563239492477610787321319
459737493390942034461152253854738223782017099102379579800069406781302102085691
841774989071361576346808622817707154364861320806240508319982008060029626644035
030045179794343581154002771275145577588743113500118199978372748043743160537434
138408502955832369297153909338705169946407425694316229890025457653413453216888
028256015004659717723461537038227959060329884183942985875415480451237171271541
220241750444378921762423204018910910394765293358088305901303452530469835893835
318688063000647761422541947265419052400574162546112972048874487367277878174253
757462662403761955994835310469056779759925426048315352166095416813711713457347
661804955313989345882361636796987070466479156964237278419130359343513794156569
204681344174783380540393351429269458347418581259182241552480428265627804456439
674910516833812214721269535047308127152277779476017032686584677165699944999159
543628779838040656567106755907351374349311683657037373511524959003610369780088
089351960979831119161145262118377181156363340800557382872954215659580576710980
567118341797115143473713582109744673175911725279807383488084489899516792 77,

```
c     p175.f 100$100d problem 10 in MP fortran SGLS 05/26/08
c     transmp < p175.f >! p175mp.f
c     g77 -O -o p175mp p175mp.f mpfun.o
c     p175mp >! p175.out &
cmp+  precision level 7000
cmp+  mixed mode safe
cmp+  output precision 7000
cmp+  scratch space 20000

      program prob10
cmp+  implicit multip real (a-h,p-z)
      implicit double precision (a-h,p-z)
cmp+  multip complex c,s,ta
      double complex c,s,ta,ione

      xl=10.0d0+0
      pi=(4.0d0+0)*atan(1.0d0+0)
```

```
      s2=sqrt(2.0d0+0)
      ione=(0,1)
      s=sinh(pi*xl/(4.0d0+0))/s2 + ione*cosh(pi*xl/(4.0d0+0))/s2
      c=cosh(pi*xl/(4.0d0+0))/s2 + ione*sinh(pi*xl/(4.0d0+0))/s2
      ta=s/c
      f=(4.0d0+0)/pi*atan2(dpimag(ta),dpreal(ta))
      write (6,*) f
      mone=-1
      do j=1,150
         s=sinh(pi*(1+2*j)*xl/(4.0d0+0))/s2 +
     $        ione*cosh(pi*(1+2*j)*xl/(4.0d0+0))/s2
         c=cosh(pi*(1+2*j)*xl/(4.0d0+0))/s2 +
     $        ione*sinh(pi*(1+2*j)*xl/(4.0d0+0))/s2
         ta=s/c
         f=f+mone**j*(4.0d0+0)/pi*atan2(dpimag(ta),dpreal(ta))
         write (6,*) f
      enddo
      end


      function dpreal(c)
      double precision dpreal
      double complex c
      dpreal = dble(c)
      return
      end


      function dpimag(c)
      double precision dpreal
      double complex c
      dpimag = dimag(c)
      return
      end
```

```
c     p175.f 100$100d problem 10 in MP fortran SGLS 05/26/08
c     transmp < p175.f >! p175mp.f
c     g77 -O -o p175mp p175mp.f mpfun.o
cmp+  precision level 7000
cmp+  mixed mode safe
cmp+  output precision 7000
cmp+  scratch space 20000

      program prob10
```

```
cmp+  implicit multip real (a-h,p-z)
      implicit double precision (a-h,p-z)
cmp+  multip complex c,s,ta
CMP>    double complex c,s,ta,ione
      double complex ione
CMP<

      REAL C(1946)
      REAL S(1946)
      REAL TA(1946)
      REAL XL(973)
      CHARACTER*15 MPA2
      CHARACTER*1 MPA1(7101)
      INTEGER MPI1
      REAL MPM1(973)
      REAL MPJ1(973)
      REAL MPM2(973)
      REAL PI(973)
      REAL MPM3(973)
      REAL MPM4(973)
      REAL S2(973)
      REAL MPZ1(1946)
      DOUBLE PRECISION MPD1
      DOUBLE PRECISION MPD2
      REAL MPZ2(1946)
      REAL MPZ3(1946)
      REAL F(973)
      REAL MPJ2(973)
      REAL MPJ3(973)
      INTEGER MPNWQ, MPNW4
      REAL MPL02, MPL10, MPPIC
      COMMON /MPTCON/ MPNWQ, MPNW4, MPL02(974), MPL10(974),
     $  MPPIC(974)
      REAL MPSS
      COMMON /MPCOM3/ MPSS(   20000)
C
      CALL MPSETP ('NW',    970)
      CALL MPSETP ('IMS',    20000)
      MPNWQ =    969
      MPNW4 =    973
      CALL MPDMC (2.D0, 0, MPM1)
      CALL MPLOG (MPM1, MPL02, MPL02)
      CALL MPDMC (10.D0, 0, MPM1)
      CALL MPLOG (MPM1, MPL02, MPL10)
      CALL MPPI (MPPIC)
```

```
      CALL MPSETP ('NW',    969)
CMP<
CMP>    xl=10.0d0+0
      MPA2 = '10^0 x 10.0'
      READ (MPA2, '( 11A1)' ) (MPA1(MPI1), MPI1 = 1,  11)
      CALL MPINPC (MPA1,  11, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM2)
      CALL MPEQ (MPM2, xl)
CMP<
CMP>    pi=(4.0d0+0)*atan(1.0d0+0)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM2)
      MPA2 = '10^0 x 1.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM3)
      CALL MPDMC (1.D0, 0, MPM4)
      CALL MPANG (MPM4, MPM3, MPPIC, MPM1)
      CALL MPMUL (MPM2, MPM1, MPM3)
      CALL MPEQ (MPM3, pi)
CMP<
CMP>    s2=sqrt(2.0d0+0)
      MPA2 = '10^0 x 2.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM2)
      CALL MPSQRT (MPM2, MPM1)
      CALL MPEQ (MPM1, s2)
CMP<
      ione=(0,1)
CMP>    s=sinh(pi*xl/(4.0d0+0))/s2 + ione*cosh(pi*xl/(4.0d0+0))/s2
```

```
      CALL MPMUL (pi, xl, MPM1)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM2)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM2, MPJ1, MPM3)
      CALL MPDIV (MPM1, MPM3, MPM2)
      CALL MPCSSH (MPM2, MPL02, MPM3, MPM1)
      CALL MPDIV (MPM1, s2, MPM2)
      CALL MPMUL (pi, xl, MPM1)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM3)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM3, MPJ1, MPM4)
      CALL MPDIV (MPM1, MPM4, MPM3)
      CALL MPCSSH (MPM3, MPL02, MPM1, MPM4)
      MPD1 = DREAL (ione)
      MPD2 = DIMAG (ione)
      CALL MPMULD (MPM1, MPD1, 0, MPZ1)
      CALL MPMULD (MPM1, MPD2, 0, MPZ1(MPNWQ+5))
      CALL MPMPCM (MPNW4, MPZ1, MPM1, MPM3)
      CALL MPDIV (MPM1, s2, MPZ2)
      CALL MPDIV (MPM3, s2, MPZ2(MPNWQ+5))
      CALL MPDMC (0.D0, 0, MPM1)
      CALL MPMMPC (MPM2, MPM1, MPNW4, MPZ3)
      CALL MPCADD (MPNW4, MPZ3, MPZ2, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, s)
CMP<
CMP>    c=cosh(pi*xl/(4.0d0+0))/s2 + ione*sinh(pi*xl/(4.0d0+0))/s2
      CALL MPMUL (pi, xl, MPM1)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM2)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM2, MPJ1, MPM3)
      CALL MPDIV (MPM1, MPM3, MPM2)
      CALL MPCSSH (MPM2, MPL02, MPM1, MPM3)
      CALL MPDIV (MPM1, s2, MPM2)
```

```
      CALL MPMUL (pi, xl, MPM1)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM3)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM3, MPJ1, MPM4)
      CALL MPDIV (MPM1, MPM4, MPM3)
      CALL MPCSSH (MPM3, MPL02, MPM4, MPM1)
      MPD1 = DREAL (ione)
      MPD2 = DIMAG (ione)
      CALL MPMULD (MPM1, MPD1, 0, MPZ1)
      CALL MPMULD (MPM1, MPD2, 0, MPZ1(MPNWQ+5))
      CALL MPMPCM (MPNW4, MPZ1, MPM1, MPM3)
      CALL MPDIV (MPM1, s2, MPZ2)
      CALL MPDIV (MPM3, s2, MPZ2(MPNWQ+5))
      CALL MPDMC (0.D0, 0, MPM1)
      CALL MPMMPC (MPM2, MPM1, MPNW4, MPZ3)
      CALL MPCADD (MPNW4, MPZ3, MPZ2, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, c)
CMP<
CMP>    ta=s/c
      CALL MPCDIV (MPNW4, s, c, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, ta)
CMP<
CMP>    f=(4.0d0+0)/pi*atan2(dpimag(ta),dpreal(ta))
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM2)
      CALL MPDIV (MPM2, pi, MPM1)
      CALL MPMPCM (MPNW4, ta, MPM3, MPM2)
      CALL MPEQ (ta, MPM3)
      CALL MPANG (MPM3, MPM2, MPPIC, MPM4)
      CALL MPMUL (MPM1, MPM4, MPM2)
      CALL MPEQ (MPM2, f)
CMP<
CMP>    write (6,*) f
      CALL MPOUT (6, f, 7000, MPA1)
CMP<
      mone=-1
```

```
      do j=1,150
CMP>       s=sinh(pi*(1+2*j)*xl/(4.0d0+0))/s2 +
CMP>   $         ione*cosh(pi*(1+2*j)*xl/(4.0d0+0))/s2
      MPA2 = '10^0 x 1'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      MPA2 = '10^0 x 2'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ2)
      MPD1 = j
      CALL MPMULD (MPJ2, MPD1, 0, MPJ3)
      CALL MPINFR (MPJ3, MPJ3, MPM1)
      CALL MPADD (MPJ1, MPJ3, MPJ2)
      CALL MPINFR (MPJ2, MPJ2, MPM1)
      CALL MPMUL (pi, MPJ2, MPM1)
      CALL MPMUL (MPM1, xl, MPM2)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM3)
      CALL MPDIV (MPM2, MPM3, MPM1)
      CALL MPCSSH (MPM1, MPL02, MPM3, MPM2)
      CALL MPDIV (MPM2, s2, MPM1)
      MPA2 = '10^0 x 1'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      MPA2 = '10^0 x 2'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ2)
      MPD1 = j
      CALL MPMULD (MPJ2, MPD1, 0, MPJ3)
      CALL MPINFR (MPJ3, MPJ3, MPM2)
      CALL MPADD (MPJ1, MPJ3, MPJ2)
      CALL MPINFR (MPJ2, MPJ2, MPM2)
      CALL MPMUL (pi, MPJ2, MPM2)
      CALL MPMUL (MPM2, xl, MPM3)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM2)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
```

```
      CALL MPADD (MPM2, MPJ1, MPM4)
      CALL MPDIV (MPM3, MPM4, MPM2)
      CALL MPCSSH (MPM2, MPL02, MPM3, MPM4)
      MPD1 = DREAL (ione)
      MPD2 = DIMAG (ione)
      CALL MPMULD (MPM3, MPD1, 0, MPZ1)
      CALL MPMULD (MPM3, MPD2, 0, MPZ1(MPNWQ+5))
      CALL MPMPCM (MPNW4, MPZ1, MPM2, MPM3)
      CALL MPDIV (MPM2, s2, MPZ2)
      CALL MPDIV (MPM3, s2, MPZ2(MPNWQ+5))
      CALL MPDMC (0.D0, 0, MPM2)
      CALL MPMMPC (MPM1, MPM2, MPNW4, MPZ3)
      CALL MPCADD (MPNW4, MPZ3, MPZ2, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, s)
CMP<
CMP>       c=cosh(pi*(1+2*j)*xl/(4.0d0+0))/s2 +
CMP>   $        ione*sinh(pi*(1+2*j)*xl/(4.0d0+0))/s2
      MPA2 = '10^0 x 1'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      MPA2 = '10^0 x 2'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ2)
      MPD1 = j
      CALL MPMULD (MPJ2, MPD1, 0, MPJ3)
      CALL MPINFR (MPJ3, MPJ3, MPM1)
      CALL MPADD (MPJ1, MPJ3, MPJ2)
      CALL MPINFR (MPJ2, MPJ2, MPM1)
      CALL MPMUL (pi, MPJ2, MPM1)
      CALL MPMUL (MPM1, xl, MPM2)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM1, MPJ1, MPM3)
      CALL MPDIV (MPM2, MPM3, MPM1)
      CALL MPCSSH (MPM1, MPL02, MPM2, MPM3)
      CALL MPDIV (MPM2, s2, MPM1)
      MPA2 = '10^0 x 1'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      MPA2 = '10^0 x 2'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
```

```
      CALL MPINPC (MPA1,   8, MPJ2)
      MPD1 = j
      CALL MPMULD (MPJ2, MPD1, 0, MPJ3)
      CALL MPINFR (MPJ3, MPJ3, MPM2)
      CALL MPADD (MPJ1, MPJ3, MPJ2)
      CALL MPINFR (MPJ2, MPJ2, MPM2)
      CALL MPMUL (pi, MPJ2, MPM2)
      CALL MPMUL (MPM2, xl, MPM3)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM2)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ1)
      CALL MPADD (MPM2, MPJ1, MPM4)
      CALL MPDIV (MPM3, MPM4, MPM2)
      CALL MPCSSH (MPM2, MPL02, MPM4, MPM3)
      MPD1 = DREAL (ione)
      MPD2 = DIMAG (ione)
      CALL MPMULD (MPM3, MPD1, 0, MPZ1)
      CALL MPMULD (MPM3, MPD2, 0, MPZ1(MPNWQ+5))
      CALL MPMPCM (MPNW4, MPZ1, MPM2, MPM3)
      CALL MPDIV (MPM2, s2, MPZ2)
      CALL MPDIV (MPM3, s2, MPZ2(MPNWQ+5))
      CALL MPDMC (0.D0, 0, MPM2)
      CALL MPMMPC (MPM1, MPM2, MPNW4, MPZ3)
      CALL MPCADD (MPNW4, MPZ3, MPZ2, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, c)
CMP<
CMP>       ta=s/c
      CALL MPCDIV (MPNW4, s, c, MPZ1)
      CALL MPCEQ (MPNW4, MPZ1, ta)
CMP<
CMP>       f=f+mone**j*(4.0d0+0)/pi*atan2(dpimag(ta),dpreal(ta))
      MPD1 = mone
      CALL MPDMC (MPD1, 0, MPJ2)
      CALL MPNPWR (MPJ2, j, MPJ1)
      CALL MPINFR (MPJ1, MPJ1, MPM1)
      MPA2 = '10^0 x 4.0'
      READ (MPA2, '( 10A1)' ) (MPA1(MPI1), MPI1 = 1,  10)
      CALL MPINPC (MPA1,  10, MPM1)
      MPA2 = '10^0 x 0'
      READ (MPA2, '(  8A1)' ) (MPA1(MPI1), MPI1 = 1,   8)
      CALL MPINPC (MPA1,   8, MPJ2)
      CALL MPADD (MPM1, MPJ2, MPM2)
```

```
      CALL MPMUL (MPJ1, MPM2, MPM1)
      CALL MPDIV (MPM1, pi, MPM2)
      CALL MPMPCM (MPNW4, ta, MPM3, MPM1)
      CALL MPEQ (ta, MPM3)
      CALL MPANG (MPM3, MPM1, MPPIC, MPM4)
      CALL MPMUL (MPM2, MPM4, MPM1)
      CALL MPADD (f, MPM1, MPM2)
      CALL MPEQ (MPM2, f)
CMP<
CMP>       write (6,*) f
      CALL MPOUT (6, f, 7000, MPA1)
CMP<
      enddo
      end


      function dpreal(c)
      double complex c
      dpreal = dble(c)
      return
      end


      function dpimag(c)
      double complex c
      dpimag = dimag(c)
      return
      end
```

Note that the Fourier series of Bornemann *et al.* (2004) is much better. I'm not perfect.

17.5 SC Toolbox solution

One can also try and solve this using the SC Toolbox. The naive approach leads to failure: $p = 1.000000000000001$ and the very unhealthy Figure 17.2. A smarter approach given in Bornemann *et al.* (2004) gives $3.837587979011753 \times 10^{-7}$ and 11 digits.

```
% p174.m 100$100d problem using SC toolbox SGLS 05/26/08
p = polygon([0 10 10+i i])
phi = lapsolve(p,[0 1 0 1]);
phi(5+0.5*i)
[tri,x,y] = triangulate(p); trisurf(tri,x,y,phi(x+i*y));
p = polygon([10+i -10+i -10-i 10-i]);
```

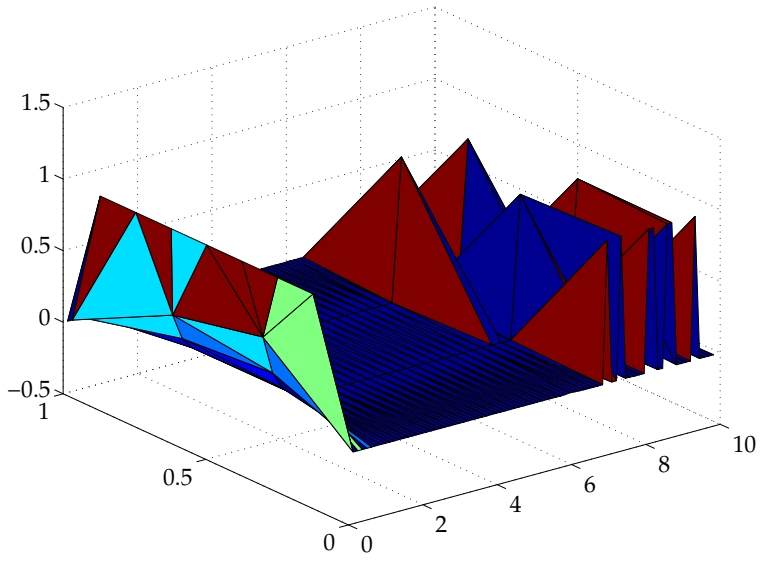

Figure 17.2: Poor solution to Laplace's equation for Problem 10 using SC Toolbox.

```
f = center(crdiskmap(p,scmapopt('Tolerance',1e-11)),0);
prevert = get(diskmap(f),'prevertex'); angle(prevert(1))/pi
```

17.6 Elliptic function solution

Where do elliptic functions come from? There are three kinds of elliptic functions really: elliptic integrals, Weierstrassian elliptic functions and Jacobi elliptic functions. For the problem here, we get elliptic integrals through the the Schwartz–Christoffel mapping from the unit disk in the z-plane to the rectangle with vertices $(a+\mathrm{i}b, -a+\mathrm{i}b, -a-\mathrm{i}b, a-\mathrm{i}b)$:

$$f(z) = 2c \int_0^z \frac{\mathrm{d}\zeta}{\sqrt{(1-\mathrm{e}^{-\mathrm{i}p\pi}\zeta^2)(1-\mathrm{e}^{\mathrm{i}p\pi}\zeta^2)}}. \tag{17.7}$$

The unknown parameter c is real and the integral can be taken along any path from 0 to z in the unit disk. From this one can deduce

$$a+\mathrm{i}b = 2c\mathrm{e}^{\mathrm{i}p\pi/2}K(\mathrm{e}^{\mathrm{i}p\pi}) \tag{17.8}$$

where $K(k)$ denotes the complete elliptic integral of the first kind with modulus k:

$$K(k) = \int_0^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}}. \tag{17.9}$$

One can take the argument of both sides of (17.8) and solve the following transcendental equation for p:

$$\cot^{-1} p = \frac{p\pi}{2} + \arg K(\mathrm{e}^{\mathrm{i}p\pi}). \tag{17.10}$$

This is a simple equation to solve in Matlab because it is very easy to compute $K(k)$.

The arithmetic-geometric mean is an algorithm due to Gauss. It computes the quantity $M(a,b)$ as the limit as $n \to \infty$ of the iteration

$$a_0 = a, \quad b_0 = b, \qquad a_{n+1} = \frac{a_n + b_n}{2}, \quad b_{n+1} = \sqrt{a_n b_n}. \tag{17.11}$$

This iteration converges quadratically. The complete elliptic integral is then given by

$$K(k) = \frac{\pi}{2M(1, \sqrt{1-k^2})}. \tag{17.12}$$

Using various identities about elliptic functions, we can obtain the closed form solution of § 17.1. There are a lot of formulas and results about elliptic functions.

17.7 Elliptic functions and integrals

The simplest way to see where the elliptic integrals come from is to work out the perimeter of an ellipse with semi-major and semi-minor axes a and b respectively. Parameterize the ellipse using $x = a\cos\theta, y = b\sin\theta$ so that $\mathrm{d}s^2 = (a^2\sin^2\theta + b^2\cos^2\theta)\mathrm{d}\theta^2$. Then

$$P = \int \mathrm{d}s = 4\int_0^{\pi/2} \sqrt{a^2\sin^2\theta + b^2\cos^2\theta}\,\mathrm{d}\theta = 4a\int_0^1 (1-t^2)^{-1/2}(1-mt^2)^{1/2}\,\mathrm{d}t = 4aE(m), \tag{17.13}$$

where $m = b^2/a^2$ and we take $m < 1$ in this derivation. This is in fact a complete elliptic integral of the second kind. "Complete" refers to the upper limit. If we were computing the perimeter up to some other angle, it would just be an elliptic integral of the first kind. For $m = 0$ we have $E(0) = \pi/2$ and we recover trigonometric functions for the incomplete kind. See AS and GR for more details.

Exercise 17.3 *The oscillations of a pendulum are given by*

$$\ddot{\theta} + \frac{g}{l}\sin\theta = 0. \tag{17.14}$$

The usual approach linearizes this equation by replacing $\sin\theta$ *by* θ*. Solve the equation exactly. Start by integral once to give*

$$\tfrac{1}{2}\dot{\theta}^2 + \frac{g}{l}\cos\theta = E. \tag{17.15}$$

The Weierstrass and Jacobi elliptic functions can also be found in GR and AS. A clear description can be found in Whittaker & Watson (1963). Elliptic function are the canonical functions that are periodic with respect to two numbers (whose ratio is not real), so that

$$f(z + 2\omega_1) = f(z), \qquad f(z + 2\omega_2) = f(z). \tag{17.16}$$

There are two kinds of functions that satisfy such relations. First, those with one double pole in each cell: these are Weierstrass functions. Second, those with two simple poles in

each cell: these are Jacobi functions. The latter can also be viewed as inverse functions of elliptic integrals, in the same way that the trigonometric functions can be viewed as inverse functions of integrals of the form $(1-t^2)^{-1/2}$. There are also Theta functions that are related to both; see (again) Whittaker & Watson (1963) or Bellman (1961).[1]

17.8 An amazing result

Bornemann *et al.* (2004) show that using results due to Ramanujan one can find p in the closed form

$$p = \frac{2}{\pi}\sin^{-1}\left[(3-2\sqrt{2})^2(2+\sqrt{5})^2(\sqrt{10}-3)^2(5^{1/4}-\sqrt{2})^4\right] \quad (17.17)$$

$$= \frac{2}{\pi}\sin^{-1}\left[\frac{1}{(3+2\sqrt{2})^2(2+\sqrt{5})^2(3+\sqrt{10})^2(\sqrt{2}+5^{1/4})^4}\right]. \quad (17.18)$$

The second formula is numerically stabler. This really is an exact closed form result.

17.9 References

Publications:

- Bellman, R. E. *A brief introduction to theta functions*, Rinehart and Winston, New York, 1961.
- Eagle, A. *The elliptic functions as they should be*, Galloway & Porter, Cambridge, 1958.
- Morse P. M. & Feshbach, H. *Methods of theoretical physics*, Mc-Graw-Hill, New York, 1953.
- Whittaker, E. T. & Watson, G. N. *A course of modern analysis*, 4th ed, Cambridge University Press, Cambridge, 1963.

Web sites:

- http://crd.lbl.gov/~dhbailey/mpdist/
- http://www.math.udel.edu/~driscoll/software/SC/

[1] An unusual book on elliptic functions is Eagle (1958), who comes up with a whole new notation. It's touching but a little sad to think of so much effort expended for nought. Granted, the standard notation is not that transparent.

Chapter 18

Special functions (06/04/08) [15 min. short]

18.1 Introduction

The very name "special function" is a little perplexing. Why should these functions be any more special than polynomials, exponentials or trigonometric functions beyond our being used to these latter "elementary functions"? People are wont to say that Bessel functions are just like sines and cosines in radial geometry, with a little bit of extra decay to make it all work, but this explanation seems unsatisfactory, frankly.

One sometimes sees the name "higher transcendental functions" in older books (e.g. Whittaker & Watson 1963). Higher seems OK, but transcendental[1] seems strange, since trigonometric and exponential functions are critically concerned with transcendental numbers. Remember that the proofs that e and π were not algebraic numbers, were milestones of classical analysis.

So what is a special function? It's not clear there is a universal definition. A tentative definition would be a function that requires the tools of analysis to define and or describe it, and that has entered the lexicon and toolbox of working mathematicians, physicists and engineers. I'm not sure this is a perfect definition and it would be interesting to come up with a better one.

Whittaker & Watson (1963) is a two-part book. The first half is a very detailed introduction to real analysis, still useful today. The second half discusses transcendental functions using the tools derived. So the transcendental functions are at the same time motivation and application.

There are many excellent books on special functions. AS and GR have of course been mentioned before as have the comprehensive books of Olver (1974), Whittaker & Watson (1963) and the focused books of Bellman (1961) and Artin (1964). Other books include the Bateman Manuscript Project (1953–1955), Magnus, Oberhettinger & Soni (1966; the German connection), Luke (1969), Lebedev (1972), Sneddon (1980), Temme (1996), Andrews *et al.* (1999) and Mathai & Haubold (2008). There is also a journal, *Integral transforms and special functions*.

[1] Not algebraic, i.e. not the solution of a polynomial equations with integer coefficients.

Good websites have also been cited previously. Note that Stephen Wolfram of Mathematica fame is a special-functions enthusiast in his own words (see http://www.stephenwolfram.com/p

However, the book on special functions that outshines all others is not even about all special functions. It's just about Bessel functions, but it is extraordinary and even today one would be hard pressed to see how to improve on its scholarship. This book is of course Watson (1944).

A brief comment on Batterman (2007) is in order. The title of this paper, published in the British Journal for the Philosophy of Science is "On the Specialness of Special Functions (The Nonrandom Effusions of the Divine Mathematician). The paper discusses what, in the author's opinion, makes special functions special.

One can argue that special functions only make sense in the context of a clear understanding and synthesis of the singularity structure and definitions by ODEs or sum of these functions. This makes them a late 19th century construct. In fact, complex analysis is crucial since power series are intimately tied to the complex plane. Many of the formulas can only be obtained using complex analysis. An interesting exception is Artin (1964), which uses only real analysis. In consequence, much use is made of functional equations and difference equations. It's beautifully done, and it works for the Gamma function, but it probably wouldn't work for anything else.

The history of special functions in the 20th century follows two strands. On the one hand, a proliferation of specialized special functions in particular fields (e.g. the Theodorson function in aerodynamics). On the other hand, efforts at greater generalizations, leading to Meier's G-function and Heun's equation. This brings back the psychological aspect of the subject: the hypergeometric function due to Gauss, with three singularities in the complex plane, seems to be about as much as most people are prepared to consider. More general functions just seem too vague and lack character.

18.2 Classification of special functions

There is on single classification. That of AS given in Figures 18.2–18.2 is useful and certainly shapes my thinking of special functions, although I would put Gamma functions before exponential integrals. You can find AS online at http://www.math.sfu.ca/~cbm/aands/; I'm not sure about the copyright issues.

This list is interesting. Gamma and error functions are universal: they pop up all over mathematics. The exponential integral is fairly straightforward and turns up occasionally, but doesn't seem to have a deep meaning. Lengendre, Bessel (Struve is an offshoot), Mathieu and Spheroidal wave functions come from separating variables in Laplace's equation and Helmholtz's equation. Hpergeometric functions are a natural framework for many functions. Coulomb wave functions come from quantum mechanics. Elliptic functions occur naturally in complex analysis and in integrating ODEs. Parabolic cylinder functions come up in asymptotics and WKB. Orthogonal polynomials appear naturally in Sturm–Liouville theory and have bee applied all over mathematical physics and numerical analysis.

The heart of the subject is really the special functions of mathematical physics, and the other functions that crop up in their asymptotic description (Gamma, parabolic cylinder

N. C. METROPOLIS
J. B. ROSSER
H. C. THACHER, Jr.
JOHN TODD
C. B. TOMPKINS
J. W. TUKEY.

VI

Contents

Handbook of Mathematical Functions
with
Formulas, Graphs, and Mathematical Tables
Edited by Milton Abramowitz and Irene A. Stegun

1. Introduction

The present Handbook has been designed to provide scientific investigators with a comprehensive and self-contained summary of the mathematical functions that arise in physical and engineering problems. The well-known Tables of Functions by E. Jahnke and F. Emde has been invaluable to workers in these fields in its many editions[1] during the past half-century. The present volume extends the work of these authors by giving more extensive and more accurate numerical tables, and by giving larger collections of mathematical properties of the tabulated functions. The number of functions covered has also been increased.

The classification of functions and organization of the chapters in this Handbook is similar to that of An Index of Mathematical Tables by A. Fletcher, J. C. P. Miller, and L. Rosenhead.[2] In general, the chapters contain numerical tables, graphs, polynomial or rational approximations for automatic computers, and statements of the principal mathematical properties of the tabulated functions, particularly those of computational importance. Many numerical examples are given to illustrate the use of the tables and also the computation of function values which lie outside their range. At the end of the text in each chapter there is a short bibliography giving books and papers in which proofs of the mathematical properties stated in the chapter may be found. Also listed in the bibliographies are the more important numerical tables. Comprehensive lists of tables are given in the Index mentioned above, and current information on new tables is to be found in the National Research Council quarterly Mathematics of Computation (formerly Mathematical Tables and Other Aids to Computation).

The mathematical notations used in this Handbook are those commonly adopted in standard texts, particularly Higher Transcendental Functions, Volumes 1-3, by A. Erdélyi, W. Magnus, F. Oberhettinger and F. G. Tricomi (McGraw-Hill, 1953-55). Some alternative notations have also been listed. The introduction of new symbols has been kept to a minimum, and an effort has been made to avoid the use of conflicting notation.

---

functions).

18.3 Computation of special functions

There are naturally books specifically about computing special functions. Examples are Thompson (1997), Zhang & Jin (1996) and Gil, Segura & Temme (2007). I mean to read them soon.

There are many different ways of getting values for special functions. We won't go through all of them.

18.3.1 Rational approximation

Many complicated functions can be represented very accurately over a certain range using a rational approximation. Many chapters in AS include appropriate approximations over a number of ranges. Note that this method is also extremely useful for calculating elementary functions (see e.g. Muller 1997). The Wolfram lecture mentioned above indicates that this method is used extensively in Mathematica, with the hard work being done in advance to establish the approximations.

18.3.2 Summing a series

Many special functions can be defined in terms of an infinite series that converges in a certain portion of the complex plane. The hypergeometric series is a typical example. A natural approach is then to sum this series. This requires some care, because convergent series can exhibit extreme cancellation between early terms before settling down and converging. An implementation is shown in `genHyper.m` in the Appendix. This is a straight translation to Matlab of a Fortran algorithm due to Nardin & Bhalla.

18.3.3 Solving an ODE

Many special functions satisfy a known ODE. Solving that ODE accurately gives the result. The functions `cerfa.m` and `cerfb.m` from Weideman & Reddy (2000) solve the ODE defining the error function using a pseudospectral method.

```
function y = cerfa(t,N)

% The function y = cerfa(t,N) computes y(t) = exp(t^2) erfc(t)
% for t > 0 using an NxN Chebyshev differentiation matrix.
% The boundary condition is y = 0 at t = infty.
% The input parameter may be a scalar or vector.

% J.A.C. Weideman, S.C. Reddy 1998

c  = 3.75;                              % Initialize parameter
```

```
[x, D] = chebdif(N+1,1);                  % Compute Chebyshev points,
D = D(2:N+1,2:N+1);                       % assemble differentiation matrix,
x = x(2:N+1);                             % and incorporate boundary condition

A = diag((1-x).^3)*D-diag(4*c^2*(1+x));  % Coefficient matrix
b = 4*c/sqrt(pi)*(x-1);                   % Right-hand side
y = A\b;                                  % Solve system

y = chebint([0; y], (t-c)./(t+c));        % Interpolate
```

```
function y = cerfb(t,N)

% The function y = cerfb(t,N) computes y(t) = exp(t^2) erfc(t)
% for t > 0 using an NxN Chebyshev differentiation matrix.
% The boundary condition is y = 1 at t = 0.
% The input parameter may be a scalar or vector.

% J.A.C. Weideman, S.C. Reddy 1998

c  = 3.75;                                % Initialize parameter

[x, D] = chebdif(N+1,1);                  % Compute Chebyshev points,

A = diag((1-x).^3)*D-diag(4*c^2*(1+x));  % Coefficient matrix
b = 4*c/sqrt(pi)*(x-1);                   % Right-hand side

a1 = A(1:N,N+1); b = b(1:N);
A = A(1:N,1:N);
y = A\(b-a1);                             % Solve system

y = chebint([y; 1], (t-c)./(t+c));        % Interpolate
```

NR describes a related technique. Integrate the ODE to where one wants the result z, from some starting point z_0 in the complex plane. Sometimes one knows $f(z_0)$ and we're done. Otherwise, pick z_0 to be within the radius of convergence of a power series expansion for f and sum the series to find $f(z_0)$.

18.3.4 Recurrence relations

This tends to apply to functions such as orthogonal polynomials that satisfy a relation, so that knowing p_{n-1} and p_n, we can compute p_{n+1}. These are usually easy to program, but one has to be careful with numerical stability when solving recursion relations.

This approach is also useful for functions like Bessel functions, where there are recurrence relations connecting functions of different orders. One still has to compute J_0 and J_1 to start the recurrence, however,

18.3.5 Matlab

Matlab has a certain number of built-in special functions.

```
>> help specfun
  Specialized math functions.

  Specialized math functions.
    airy        - Airy functions.
    besselj     - Bessel function of the first kind.
    bessely     - Bessel function of the second kind.
    besselh     - Bessel functions of the third kind (Hankel function).
    besseli     - Modified Bessel function of the first kind.
    besselk     - Modified Bessel function of the second kind.
    beta        - Beta function.
    betainc     - Incomplete beta function.
    betaln      - Logarithm of beta function.
    ellipj      - Jacobi elliptic functions.
    ellipke     - Complete elliptic integral.
    erf         - Error function.
    erfc        - Complementary error function.
    erfcx       - Scaled complementary error function.
    erfinv      - Inverse error function.
    expint      - Exponential integral function.
    gamma       - Gamma function.
    gammainc    - Incomplete gamma function.
    gammaln     - Logarithm of gamma function.
    psi         - Psi (polygamma) function.
    legendre    - Associated Legendre function.
```

and other irrelevant functions. These other functions are not really irrelevant, but they are discrete number theoretic functions and coordinate transformations, which are not good applications of complex analysis.

Watch out. Some of the implementations of these functions are only defined for real arguments. They are of course also limited to double precision.

The driver program for evaluating error functions, `evacuee.m`, is a typical example. Depending on the value of x, different rational approximations are used. The argument x has to be real.

```
function result = erfcore(x,jint)
%ERFCORE Core algorithm for error functions.
%   erf(x) = erfcore(x,0)
```

```
%   erfc(x) = erfcore(x,1)
%   erfcx(x) = exp(x^2)*erfc(x) = erfcore(x,2)

%   C. Moler, 2-1-91.
%   Copyright 1984-2005 The MathWorks, Inc.
%   $Revision: 5.15.4.4 $  $Date: 2006/12/15 19:28:21 $

%   This is a translation of a FORTRAN program by W. J. Cody,
%   Argonne National Laboratory, NETLIB/SPECFUN, March 19, 1990.
%   The main computation evaluates near-minimax approximations
%   from "Rational Chebyshev approximations for the error function"
%   by W. J. Cody, Math. Comp., 1969, PP. 631-638.

%   Note: This M-file is intended to document the algorithm.
%   If a MEX file for a particular architecture exists,
%   it will be executed instead, but its functionality is the same.
%#mex

%{
    if ~isreal(x),
       error('MATLAB:erfcore:ComplexInput', 'Input argument must be real.')
    end
    result = repmat(NaN,size(x));
%
%   evaluate  erf  for  |x| <= 0.46875
%
    xbreak = 0.46875;
    k = find(abs(x) <= xbreak);
    if ~isempty(k)
        a = [3.1611237438705656e00; 1.1386415415105016e02;
             3.7748523768530202e02; 3.2093775891384694e03;
             1.8577706184603153e-1];
        b = [2.3601290952344121e01; 2.4402463793444417e02;
             1.2826165260773723e03; 2.8442368334391706e03];

            y = abs(x(k));
            z = y .* y;
            xnum = a(5)*z;
            xden = z;
            for i = 1:3
               xnum = (xnum + a(i)) .* z;
               xden = (xden + b(i)) .* z;
            end
            result(k) = x(k) .* (xnum + a(4)) ./ (xden + b(4));
            if jint ~= 0, result(k) = 1 - result(k); end
```

```
            if jint == 2, result(k) = exp(z) .* result(k); end
    end
%
%   evaluate  erfc  for 0.46875 <= |x| <= 4.0
%
    k = find((abs(x) > xbreak) &  (abs(x) <= 4.));
    if ~isempty(k)
        c = [5.64188496988670089e-1; 8.88314979438837594e00;
             6.11919063714162950e01; 2.98635138197400131e02;
             8.81952221241769090e02; 1.71204761263407058e03;
             2.05107837782607147e03; 1.23033935479799725e03;
             2.15311535474403846e-8];
        d = [1.57449261107098347e01; 1.17693950891312499e02;
             5.37181101862009858e02; 1.62138957456669019e03;
             3.29079923573345963e03; 4.36261909014324716e03;
             3.43936767414372164e03; 1.23033935480374942e03];

            y = abs(x(k));
            xnum = c(9)*y;
            xden = y;
            for i = 1:7
               xnum = (xnum + c(i)) .* y;
               xden = (xden + d(i)) .* y;
            end
            result(k) = (xnum + c(8)) ./ (xden + d(8));
            if jint ~= 2
               z = fix(y*16)/16;
               del = (y-z).*(y+z);
               result(k) = exp(-z.*z) .* exp(-del) .* result(k);
            end
    end
%
%   evaluate  erfc  for |x| > 4.0
%
    k = find(abs(x) > 4.0);
    if ~isempty(k)
        p = [3.05326634961232344e-1; 3.60344899949804439e-1;
             1.25781726111229246e-1; 1.60837851487422766e-2;
             6.58749161529837803e-4; 1.63153871373020978e-2];
        q = [2.56852019228982242e00; 1.87295284992346047e00;
             5.27905102951428412e-1; 6.05183413124413191e-2;
             2.33520497626869185e-3];

            y = abs(x(k));
            z = 1 ./ (y .* y);
```

```
            xnum = p(6).*z;
            xden = z;
            for i = 1:4
               xnum = (xnum + p(i)) .* z;
               xden = (xden + q(i)) .* z;
            end
            result(k) = z .* (xnum + p(5)) ./ (xden + q(5));
            result(k) = (1/sqrt(pi) -  result(k)) ./ y;
            if jint ~= 2
               z = fix(y*16)/16;
               del = (y-z).*(y+z);
               result(k) = exp(-z.*z) .* exp(-del) .* result(k);
               k = find(~isfinite(result));
               result(k) = 0*k;
            end
    end
%
%   fix up for negative argument, erf, etc.
%
    if jint == 0
            k = find(x > xbreak);
            result(k) = (0.5 - result(k)) + 0.5;
            k = find(x < -xbreak);
            result(k) = (-0.5 + result(k)) - 0.5;
    elseif jint == 1
            k = find(x < -xbreak);
            result(k) = 2. - result(k);
    else  % jint must = 2
            k = find(x < -xbreak);
            z = fix(x(k)*16)/16;
            del = (x(k)-z).*(x(k)+z);
            y = exp(z.*z) .* exp(del);
            result(k) = (y+y) - result(k);
    end
%}
```

Matlab translations of the Zhang & Jin (1996) algorithms due to Barrowes may be found at the Mathworks web site. Some of the comments imply there are bugs.

Matlab can also access some of Maple's special functions.

```
>> mfunlist
 MFUNLIST Special functions for MFUN.
    The following special functions are listed in alphabetical order
    according to the third column. n denotes an integer argument,
    x denotes a real argument, and z denotes a complex argument. For
```

```
   more detailed descriptions of the functions, including any
   argument restrictions, see the Reference Manual, or use MHELP.

bernoulli      n       Bernoulli Numbers
bernoulli      n,z     Bernoulli Polynomials
BesselI        x1,x    Bessel Function of the First Kind
BesselJ        x1,x    Bessel Function of the First Kind
BesselK        x1,x    Bessel Function of the Second Kind
BesselY        x1,x    Bessel Function of the Second Kind
Beta           z1,z2   Beta Function
binomial       x1,x2   Binomial Coefficients
EllipticF -    z,k     Incomplete Elliptic Integral, First Kind
EllipticK -    k       Complete Elliptic Integral, First Kind
EllipticCK -   k       Complementary Complete Integral, First Kind
EllipticE -    k       Complete Elliptic Integrals, Second Kind
EllipticE -    z,k     Incomplete Elliptic Integrals, Second Kind
EllipticCE -   k       Complementary Complete Elliptic Integral, Second Kind
EllipticPi -   nu,k    Complete Elliptic Integrals, Third Kind
EllipticPi -   z,nu,k  Incomplete Elliptic Integrals, Third Kind
EllipticCPi -  nu,k    Complementary Complete Elliptic Integral, Third Kind
erfc           z       Complementary Error Function
erfc           n,z     Complementary Error Function's Iterated Integrals
Ci             z       Cosine Integral
dawson         x       Dawson's Integral
Psi            z       Digamma Function
dilog          x       Dilogarithm Integral
erf            z       Error Function
euler          n       Euler Numbers
euler          n,z     Euler Polynomials
Ei             x       Exponential Integral
Ei             n,z     Exponential Integral
FresnelC       x       Fresnel Cosine Integral
FresnelS       x       Fresnel Sine Integral
GAMMA          z       Gamma Function
harmonic       n       Harmonic Function
Chi            z       Hyperbolic Cosine Integral
Shi            z       Hyperbolic Sine Integral
GAMMA          z1,z2   Incomplete Gamma Function
W              z       Lambert's W Function
W              n,z     Lambert's W Function
lnGAMMA        z       Logarithm of the Gamma function
Li             x       Logarithmic Integral
Psi            n,z     Polygamma Function
Ssi            z       Shifted Sine Integral
Si             z       Sine Integral
```

```
Zeta            z       (Riemann) Zeta Function
Zeta            n,z     (Riemann) Zeta Function
Zeta            n,z,x   (Riemann) Zeta Function

       Orthogonal Polynomials (Extended Symbolic Math Toolbox only)
T      n,x             Chebyshev of the First Kind
U      n,x             Chebyshev of the Second Kind
G      n,x1,x          Gegenbauer
H      n,x             Hermite
P      n,x1,x2,x       Jacobi
L      n,x             Laguerre
L      n,x1,x          Generalized Laguerre
P      n,x             Legendre

   See also MFUN, MHELP.
```

18.3.6 Maple

I've never checked if all of Maple's special functions are available in Matlab. Of course in Maple they are available with arbitrary precision.

18.3.7 Mathematica

If we are to believe Stephen Wolfram, Mathematica does a good job with special functions. I still don't like Mathematica and I don't use it.[2] See also the Wolfram Functions Site `http://functions.wolfram.com/`.

18.3.8 Fortran libraries

I mentioned a number of these in the Introduction. Installing them can sometimes be a pain, but most are high quality. SLATEC is probably the most useful. The GAMS interface is also useful in linking to ACM algorithms which are not part of libraries.

A typical routine is shown in `derf.f` for the error function. Once again a rational approximation is used.

```
*DECK DERF
      DOUBLE PRECISION FUNCTION DERF (X)
C***BEGIN PROLOGUE  DERF
C***PURPOSE  Compute the error function.
C***LIBRARY   SLATEC (FNLIB)
```

[2] I don't think the following story is libellous. A colleague once taught a class on numerical methods and used Mathematica to implement algorithms. He wanted to compare what he was teaching to the inbuilt routines in Mathematica and contacted the company to ask them for information about these routines, but they wouldn't give him any. I view this as unhelpful.

```
C***CATEGORY  C8A, L5A1E
C***TYPE      DOUBLE PRECISION (ERF-S, DERF-D)
C***KEYWORDS  ERF, ERROR FUNCTION, FNLIB, SPECIAL FUNCTIONS
C***AUTHOR  Fullerton, W., (LANL)
C***DESCRIPTION
C
C DERF(X) calculates the double precision error function for double
C precision argument X.
C
C Series for ERF        on the interval  0.          to  1.00000E+00
C                                        with weighted error   1.28E-32
C                                         log weighted error  31.89
C                               significant figures required  31.05
C                                    decimal places required  32.55
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  D1MACH, DCSEVL, DERFC, INITDS
C***REVISION HISTORY  (YYMMDD)
C   770701  DATE WRITTEN
C   890531  Changed all specific intrinsics to generic.  (WRB)
C   890531  REVISION DATE from Version 3.2
C   891214  Prologue converted to Version 4.0 format.  (BAB)
C   900727  Added EXTERNAL statement.  (WRB)
C   920618  Removed space from variable name.  (RWC, WRB)
C***END PROLOGUE  DERF
      DOUBLE PRECISION X, ERFCS(21), SQEPS, SQRTPI, XBIG, Y, D1MACH,
     1  DCSEVL, DERFC
      LOGICAL FIRST
      EXTERNAL DERFC
      SAVE ERFCS, SQRTPI, NTERF, XBIG, SQEPS, FIRST
      DATA ERFCS(  1) / -.4904612123 4691808039 9845440333 76 D-1     /
      DATA ERFCS(  2) / -.1422612051 0371364237 8247418996 31 D+0     /
      DATA ERFCS(  3) / +.1003558218 7599795575 7546767129 33 D-1     /
      DATA ERFCS(  4) / -.5768764699 7674847650 8270255091 67 D-3     /
      DATA ERFCS(  5) / +.2741993125 2196061034 4221607914 71 D-4     /
      DATA ERFCS(  6) / -.1104317550 7344507604 1353812959 05 D-5     /
      DATA ERFCS(  7) / +.3848875542 0345036949 9613114981 74 D-7     /
      DATA ERFCS(  8) / -.1180858253 3875466969 6317518015 81 D-8     /
      DATA ERFCS(  9) / +.3233421582 6050909646 4029309533 54 D-10    /
      DATA ERFCS( 10) / -.7991015947 0045487581 6073747085 95 D-12    /
      DATA ERFCS( 11) / +.1799072511 3961455611 9672454866 34 D-13    /
      DATA ERFCS( 12) / -.3718635487 8186926382 3168282094 93 D-15    /
      DATA ERFCS( 13) / +.7103599003 7142529711 6899083946 66 D-17    /
      DATA ERFCS( 14) / -.1261245511 9155225832 4954248533 33 D-18    /
      DATA ERFCS( 15) / +.2091640694 1769294369 1705002666 66 D-20    /
```

```
      DATA ERFCS( 16) / -.3253973102 9314072982 3641600000 00 D-22    /
      DATA ERFCS( 17) / +.4766867209 7976748332 3733333333 33 D-24    /
      DATA ERFCS( 18) / -.6598012078 2851343155 1999999999 99 D-26    /
      DATA ERFCS( 19) / +.8655011469 9637626197 3333333333 33 D-28    /
      DATA ERFCS( 20) / -.1078892517 7498064213 3333333333 33 D-29    /
      DATA ERFCS( 21) / +.1281188399 3017002666 6666666666 66 D-31    /
      DATA SQRTPI / 1.772453850 9055160272 9816748334 115D0 /
      DATA FIRST /.TRUE./
C***FIRST EXECUTABLE STATEMENT  DERF
      IF (FIRST) THEN
         NTERF = INITDS (ERFCS, 21, 0.1*REAL(D1MACH(3)))
         XBIG = SQRT(-LOG(SQRTPI*D1MACH(3)))
         SQEPS = SQRT(2.0D0*D1MACH(3))
      ENDIF
      FIRST = .FALSE.
C
      Y = ABS(X)
      IF (Y.GT.1.D0) GO TO 20
C
C ERF(X) = 1.0 - ERFC(X)  FOR  -1.0 .LE. X .LE. 1.0
C
      IF (Y.LE.SQEPS) DERF = 2.0D0*X*X/SQRTPI
      IF (Y.GT.SQEPS) DERF = X*(1.0D0 + DCSEVL (2.D0*X*X-1.D0,
     1  ERFCS, NTERF))
      RETURN
C
C ERF(X) = 1.0 - ERFC(X) FOR ABS(X) .GT. 1.0
C
 20   IF (Y.LE.XBIG) DERF = SIGN (1.0D0-DERFC(Y), X)
      IF (Y.GT.XBIG) DERF = SIGN (1.0D0, X)
C
      RETURN
      END
```

The CERNLIB notoriously had the only routine to compute modified Bessel functions of imaginary order. Now it's freely avaiable, and there are other packages, so it isn't so bad. Another database is `http://www.cpc.cs.qub.ac.uk/`.

18.3.9 Multiple precision

Maple has already been mentioned. Arprec contains the following functions: $I_0(t)$, $\Gamma(z)$, erf, erfc, and multiple zeta functions. Carefully written basic algorithms can of course be written in multiple precision.

18.4 Example: error function

Figure 18.1 shows the difference between the error function computed to 1000 digits in Maple and a variety of other methods. No curve corresponds to zero difference to machine precision. The ODE method `cerfa.m` is not great; the rest are fine to machine accuracy. Note that the Fortran implementation is not totally efficient: the numerical part is fast, but it's been implemented as a scalar function in Matlab. It could be vectorized easily.

```
% p181.m Special function example SGLS 05/29/08
x = logspace(-2,2,81);
tic; e(1,:) = erf(x); t(1) = toc;
tic; e(2,:) = cerfa(x,10)'; e(2,:) = 1 - exp(-x.^2).*e(2,:); t(2) = toc;
tic; e(3,:) = cerfa(x,100)'; e(3,:) = 1 - exp(-x.^2).*e(3,:); t(3) = toc;
tic; for j = 1:length(x); e(4,j) = serf(x(j)); end; t(4) = toc;
maple restart; tic; e(5,:) = mfun('erf',x); t(5) = toc;
maple restart; maple('Digits:=1000'); tic; e(6,:) = mfun('erf',x); t(6) = toc;
t
loglog(x,abs(e-ones(6,1)*e(6,:)))
xlabel('x'); ylabel('erf(x)');
legend('Matlab','cerfa, N= 10','cerfa, N = 100','SLATEC','Default Maple')
```

```
C     serf.f error function using Fortran SGLS 05/29/08
C     mex -O -v -fortran -lslatec serf.f
C
C     The gateway routine.
      subroutine mexFunction(nlhs, plhs, nrhs, prhs)
C-----------------------------------------------------------------

      integer plhs(*), prhs(*)
      integer mxGetPr, mxCreateDoubleMatrix
C-----------------------------------------------------------------

      real*8 x, e

C     Check for proper number of arguments.
      if (nrhs .ne. 1) then
         call mexErrMsgTxt('One input required.')
      elseif (nlhs .ne. 1) then
         call mexErrMsgTxt('One output required.')
      endif

      call mxCopyPtrToReal8(mxGetPr(prhs(1)),x,1)
      e = derf(x)
      plhs(1) = mxCreateDoubleMatrix(1,1,1)
```

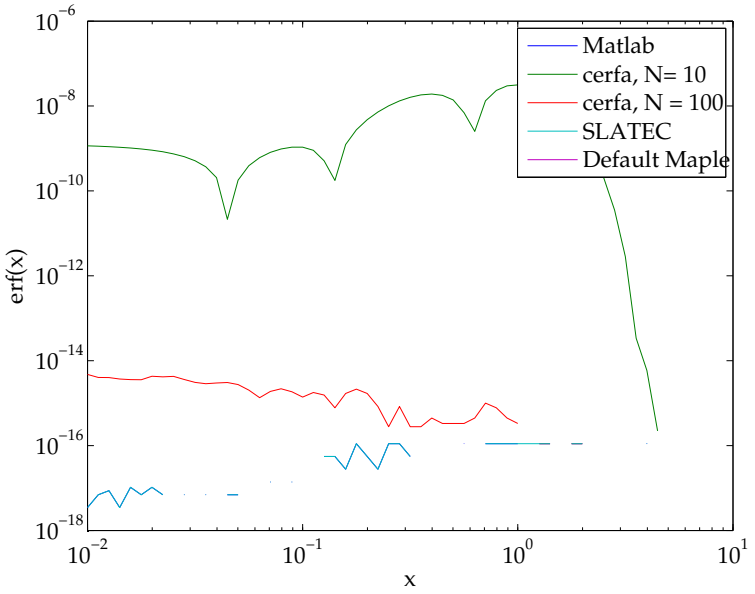

Figure 18.1: Accuracy of error function calculations.

```
    call mxCopyReal8ToPtr(e,mxGetPr(plhs(1)),1)
    end
```

18.5 A few other special functions

Special functions not in AS include polylogarithms, Lambert's W-function (a complete misnomer and really a Maple invention – see Hayes 2005), Meijer G-functions, generalized hypergeometric functions (Slater 1966), Painlevé transcendents, a number of number-theoretical functions (Dirichlet), and no doubt many others.

Other functions that I would not really classify as special functions are the Ackermann function (computer science) and various pathological counter-examples (Weierstrass functions).

18.6 References

Publications:

- Andrews, G. E., Askey, R. & Roy, R. *Special functions*, Cambridge University Press, Cambridge, 1999.
- Artin, E. *The gamma function.* Holt, Rhinehart and Winston, New York, 1964.
- Bateman Manuscript Project. *Higher transcendental functions*, McGraw-Hill, New York, 1953–1955.

- Batterman, R. W. On the specialness of special functions (The nonrandom effusions of the divine mathematician). *Brit. J. Phil. Sci.*, **58**, 263–286, 2007.
- Bellman, R. E. *A brief introduction to theta functions*, Rinehart and Winston, New York, 1961.
- Gil, A., Segura, J. & Temme, N. M. *Numerical methods for special functions*, SIAM, Philadelphia, 2007.
- Hayes, B. Why W?, *Amer. Scientist*, **93**, 104–108, 2005.
- Lebedev, N. N. *Special functions and their applications*, Dover, New York, 1972.
- Luke, Y. l. *The special functions and their approximations*, Academic, New York, 1969.
- Mathai, A. *Special functions for applied scientists*, Springer, Berlin, 2008.
- Magnus, W., Oberhettinger, F. & Soni, R. P. *Formulas and theorems for the special functions of mathematical physics*, 3rd ed., Springer, Berlin, 1966.
- Muller, J.-M. *Elementary functions: algorithms and implementations*, Birkhäuser, Boston, 1997.
- Slater, L. J. *Generalized hypergeometric functions*, Cambridge University Press, Cambridge, 1966.
- Sneddon, I. N. *Special functions of mathematical physics and chemistry*, Longman, London, 1980.
- Thompson, W. J. *Atlas for computing mathematical functions: an illustrated guide for practitioners with programs in Fortran 90 and Mathematics*, Wiley, New York, 1997.
- Timme, N. M. *Special functions: an introduction to the classical functions of mathematical physics*, Wiley, New York, 1996.
- Watson, G. N. *A treatise on the theory of Bessel functions*, 2nd ed,. Cambridge University Press, Cambridge, 1944.
- Whittaker, E. T. & Watson, G. N. *A course of modern analysis*, 4th ed, Cambridge University Press, Cambridge, 1963.
- Zhang, S. & Jin, J. M. *Computation of Special Functions*, Wiley, New York, 1996.

Web sites:

- http://www.cpc.cs.qub.ac.uk/
- http://en.wikipedia.org/wiki/Special_functions
- http://functions.wolfram.com/

- http://jin.ece.uiuc.edu/specfun.html
- http://www.informaworld.com/smpp/title~content=t71364368
- http://www.math.ku.dk/kurser/2006-08/blok2/klasan
- http://www.math.sfu.ca/~cbm/aands/
- http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectid=6218
- http://www.stephenwolfram.com/publications/talks/specialfunctions/

Chapter 19

Orthogonal polynomials (06/03/08) [10 min. short]

19.1 Introduction

Orthogonal polynomials span a wide range of applications, from the proof of the Bieberbach conjecture through Sturm–Liouville theory to numerical analysis. We will cover a few of these aspects. There has been a lot of recent work on orthogonal polynomials from a pure mathematical perspective, and some of this is covered in Andrews *et al.* (1999).

19.2 Three-term recurrence relations

Definition 19.1 *A sequence of orthogonal polynomials* $\{p_n(x)\}$*, where* $p_n(x)$ *has degree* n*, is orthogonal with respect to the weight function* $w(x)$ *if*

$$\int_a^b p_n(x)p_n(x)w(x)\,\mathrm{d}x = h_n\delta_{mn}. \tag{19.1}$$

One can be more general and talk about orthogonality with respect to a distribution $\mathrm{d}W(x)$. This definition implies the following

Theorem 10 *A sequence of orthogonal polynomials satisfies*

$$p_{n+1}(x) = (A_n x + B_n)p_n(x) - C_n p_{n-1}(x) = 0 \tag{19.2}$$

for $n \geq 0$ *and we set* $p_{-1}(x) = 0$*. The real constants* A_n*,* B_n *and* C_n *satisfy* $A_{n-1}A_nC_n > 0$*.*

Favard's theorem is the converse of this theorem: if the $\{p_n(x)\}$ satisfy a three-term recurrence relation, then they are orthogonal with respect to some weight function.

From our experience with Fourier series, which satisfy a simple orthogonality condition, we see that orthogonal polynomials may be useful in expanding functions and calculating series such that the approximation is always improved by adding more terms. In fact, we can obtain one family of orthogonal polynomials exactly by a simple change

of variable. Define the polynomials $T_n(x) = \cos\{n\cos^{-1}(x)\}$. Note that $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$ trivially. Then

$$\int_{-1}^{1} T_n(x)T_m(x)\frac{\mathrm{d}z}{\sqrt{1-x^2}} = \int_0^{\pi} \cos n\theta \cos m\theta \,\mathrm{d}\theta = \pi\epsilon_n^{-1}\delta_{nm}, \tag{19.3}$$

where ϵ_n is the Neumann symbol defined in Chapter 8. Hence these polynomials are orthogonal with respect to the weight function $(1-x^2)^{-1/2}$. These are the Chebyshev polynomials of the first kind.

One can go down the infinite-dimensional vector space road as far as one wishes, but we shan't bother, since we don't learn anything new. It is not obvious[1] how to go from the orthogonality condition to the recurrence relation with no extra information. Finding the ODE satisfied by the polynomials makes this fairly clear, but one then has to find the ODE. Luckily, AS has a list. It also contains explicit expressions for the coefficients. Usually the ODE satisfied by a sequence of orthogonal polynomials is not very enlightening in its own right. However, some orthogonal polynomials (called special orthogonal polynomials in Andrews *et al.* 1999) come from interesting problems, i.e. mathematical physics.

19.3 Sturm–Liouville problems

Physically it is not surprising that one generates sequences of orthogonal functions from separating variables in Helmholtz's equation for example, since linear wave systems have independent modes that do not exchange energy and are hence orthogonal. The functions are not always polynomials; for instance in cylindrical coordinates one always finds trigonometric functions from the angular dependence and these are not polynomials. Sturm–Liouville theory is the theory of second-order linear eigenvalue problems and these are the underlying eigenvalue problems that give rise to these orthogonal sequences:

$$-(p(x)f')' + q(x)f = \lambda r(x)f. \tag{19.4}$$

In a number of cases however, the functions take the form of a polynomial multiplied by another factor. A typical example comes from the Hermite polynomials, which occur for instance when solving problems in the equatorial wave guide in geophysical fluid dynamics (Matsuno 1966). The normal modes are then Hermite functions $H_n(x)\mathrm{e}^{-x^2}$, where the $H_n(x)$ are Hermite polynomials satisfying the orthogonality condition

$$\int_{-\infty|}^{\infty} \mathrm{e}^{-x^2} H_n(x)H_m(x)\,\mathrm{d}x = 2^n n!\sqrt{\pi}\delta_{mn}. \tag{19.5}$$

This is a case where the exact formula is simpler to write down than the recurrence relation:

$$H_n(x) = (-1)^n \mathrm{e}^{x^2}\frac{\mathrm{d}^n \mathrm{e}^{-x^2}}{\mathrm{d}x^n}. \tag{19.6}$$

[1] Read: I don't know.

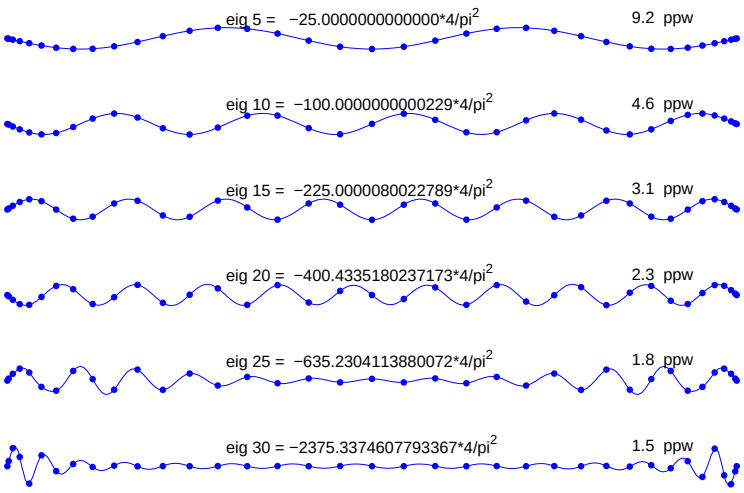

Figure 19.1:

Solving Sturm–Liouville problems numerically has been touched on before. For non-singular problems with non-periodic boundary conditions, one can happily use pseudopectral methods as in Figure 19.1 taken from Trefethen (2000). The accuracy is excellent.

```
% p15.m - solve eigenvalue BVP u_xx = lambda*u, u(-1)=u(1)=0

  N = 36; [D,x] = cheb(N); D2 = D^2; D2 = D2(2:N,2:N);
  [V,Lam] = eig(D2); lam = diag(Lam);
  [foo,ii] = sort(-lam);          % sort eigenvalues and -vectors
  lam = lam(ii); V = V(:,ii); clf
  for j = 5:5:30                  % plot 6 eigenvectors
    u = [0;V(:,j);0]; subplot(7,1,j/5)
    plot(x,u,'.','markersize',12), grid on
    xx = -1:.01:1; uu = polyval(polyfit(x,u,N),xx);
    line(xx,uu), axis off
    text(-.4,.5,sprintf('eig %d =%20.13f*4/pi^2',j,lam(j)*4/pi^2))
    text(.7,.5,sprintf('%4.1f  ppw', 4*N/(pi*j)))
  end
```

Singular Sturm–Liouville problems are significantly more complicated. Boyd () gives some examples in his unmistakable style. Pryce (1993) is an excellent account of the numerical solution of Sturm–Liouville problems, including singular problems. The emphasis is on methods designed expressly for Sturm–Liouville problems as opposed to general methods. The most widely used algorithm is probably SLEIGN which can be found at GAMS (SLEIGN2 is supposed to be more recent but appears to be limited to single precision). SLEIGN is built into gsl (the GNU scientific library). See the example below.

```
c     p191.f SLEIGN SGLS 06/03/08
c     g77 -O -o p191 p191.f sleign.f
      program p191
      implicit double precision (a-h,p-z)
      dimension slfun(110)

      a=-1d0
      b=0d0
      intab=1
      p0ata=-1.0
      qfata=1.0
      p0atb=-1.0
      qfatb=1.0
      a1=1.0
      a2=0.0
      b1=1.0
      b2=0.0

      numeig=1
      eig=0.0
      tol=1d-6
      islfun=101
      do j=0,100
         slfun(j+10)=j*.01-1.0
      enddo

      call sleign(a,b,intab,p0ata,qfata,p0atb,qfatb,a1,a2,b1,b2,
     $     numeig,eig,tol,iflag,islfun,slfun)
      write (6,*) iflag,eig
      write (6,*) (slfun(j),j=10,110)

      end

      function p(x)
      implicit double precision (a-h,p-z)

      p=1d0
      end

      function q(x)
      implicit double precision (a-h,p-z)

      q=0d0
      end
```

```
      function r(x)
      implicit double precision (a-h,p-z)

      r=1.1-0.1*exp(x*10)
      end
```

```
tako:~/Teaching/MAE207_2008 8:04> p191
 1  8.99546276
  0.  0.0424749863  0.0849079472  0.127256899  0.169480193  0.211535549
  0.253381609  0.294976972  0.336280481  0.377251271  0.417849048  0.458033158
  0.497764088  0.537002529  0.575709657  0.613847179  0.65137736  0.688263288
  0.72446803  0.759955989  0.794692055  0.828641863  0.861771825  0.894049357
  0.925442143  0.955919322  0.985450745  1.0140072  1.04156044  1.06808337
  1.09354943  1.11793361  1.14121179  1.16336094  1.18435918  1.20418573
  1.22282112  1.24024669  1.25644534  1.27140108  1.28509912  1.29752595
  1.30866937  1.31851826  1.32706298  1.33429512  1.34020758  1.34479455
  1.34805159  1.34997548  1.35056443  1.34981792  1.34773678  1.34432316
  1.33958055  1.33351374  1.32612885  1.31743335  1.30743596  1.29614674
  1.28357698  1.26973937  1.25464778  1.2383173  1.2207644  1.20200666
  1.18206284  1.16095307  1.13869849  1.11532137  1.09084531  1.06529485
  1.03869563  1.01107431  0.982458812  0.952877816  0.92236097  0.890939142
  0.85864379  0.825507487  0.791563455  0.756845869  0.72138968  0.685230406
  0.648404465  0.610948697  0.572900702  0.534298629  0.495180944  0.455586754
  0.415555429  0.375126653  0.334340455  0.293236949  0.25185637  0.210239007
  0.168425086  0.126454635  0.0843674526  0.0422029616  9.43449738E-08
```

Exercise 19.1 *What problem is being solved in* `p191.f`*?*

19.4 Interpolation

Note the irony of `p15.m`: the exact solutions are sines and cosines and yet we carefully use Chebyshev polynomials to represent our solutions, not trigonometric functions. Why is this? We need to talk about interpolation.

The building block of polynomial interpolation is Lagrange interpolation. Given the n points x_i at which the function to be interpolated takes the value f_i, the interpolating polynomial is

$$p_n(x) = \sum_{i=1}^{n} \frac{p(x) f_i}{p'(x_i)(x - x_i)}, \tag{19.7}$$

where $p(x) = (x - x_1) \dots (x - x_n)$. (Watch out for the order of p_n: n or $n - 1$?) There is nothing wrong with this formula as far as reproducing the f_i values goes. However, as shown in Figure 19.2, the resulting polynomial can do a very poor job when it is used with equispaced interpolation points, even inside the interval covered by the interpolation

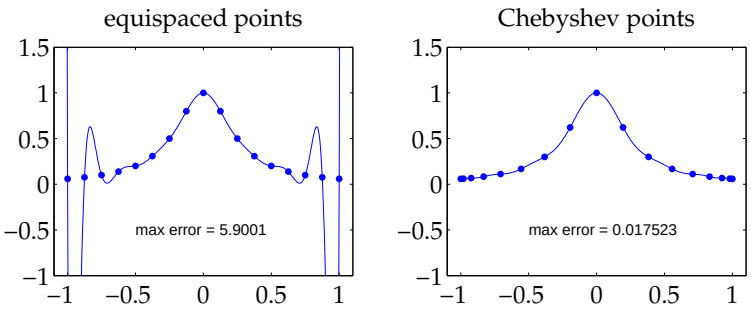

Figure 19.2: The Runge phenomenon. From Trefethen (2000).

points. This is known as Runge's phenomenon, and should warn us against equispaced points. Chebyshev points, also shown in Figure 19.2, are OK.

```
% p9.m - polynomial interpolation in equispaced and Chebyshev pts

  N = 16;
  xx = -1.01:.005:1.01; clf
  for i = 1:2
    if i==1, s = 'equispaced points'; x = -1 + 2*(0:N)/N; end
    if i==2, s = 'Chebyshev points';  x = cos(pi*(0:N)/N); end
    subplot(2,2,i)
    u = 1./(1+16*x.^2);
    uu = 1./(1+16*xx.^2);
    p = polyfit(x,u,N);              % interpolation
    pp = polyval(p,xx);              % evaluation of interpolant
    plot(x,u,'.','markersize',13)
    line(xx,pp)
    axis([-1.1 1.1 -1 1.5]), title(s)
    error = norm(uu-pp,inf);
    text(-.5,-.5,['max error = ' num2str(error)])
  end
```

The reasons for the Runge phenomenon are too long to go into here. There is a nice physical interpretation in terms of equipotential curves and the reader is referred to Boyd (2000) and Trefethen (2000), from which Figure 19.3 is taken.

```
% p10.m - polynomials and corresponding equipotential curves

  N = 16; clf
  for i = 1:2
    if i==1, s = 'equispaced points'; x = -1 + 2*(0:N)/N; end
    if i==2, s = 'Chebyshev points';  x = cos(pi*(0:N)/N); end
```

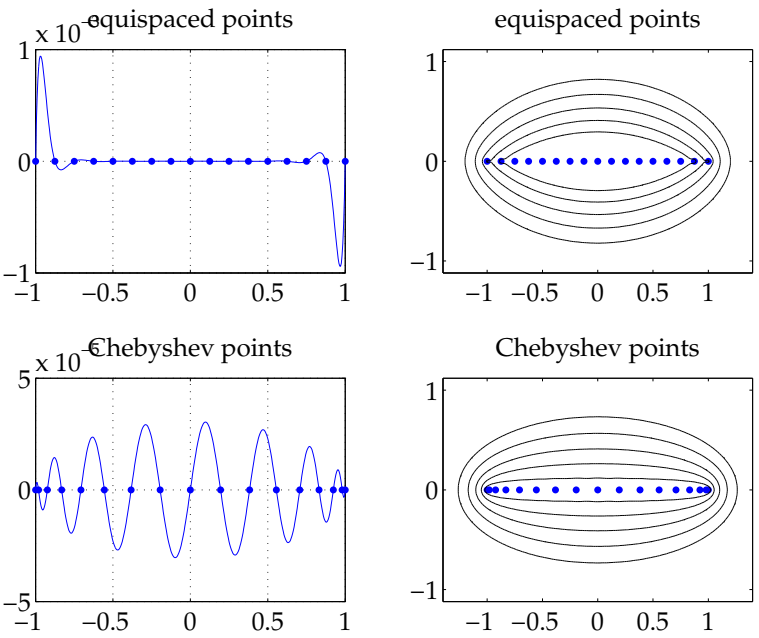

Figure 19.3: Equipotential curves for equispaced points and Chebyshev points.

```
  p = poly(x);

  % Plot p(x) over [-1,1]:
  xx = -1:.005:1; pp = polyval(p,xx);
  subplot(2,2,2*i-1)
  plot(x,0*x,'.','markersize',13), hold on
  plot(xx,pp), grid on
  set(gca,'xtick',-1:.5:1), title(s)

  % Plot equipotential curves:
  subplot(2,2,2*i)
  plot(real(x),imag(x),'.','markersize',13), hold on
  axis([-1.4 1.4 -1.12 1.12])
  xgrid = -1.4:.02:1.4; ygrid = -1.12:.02:1.12;
  [xx,yy] = meshgrid(xgrid,ygrid); zz = xx+1i*yy;
  pp = polyval(p,zz); levels = 10.^(-4:0);
  contour(xx,yy,abs(pp),levels), title(s), colormap(1e-6*[1 1 1]);
end
```

19.5 Approximation

This is a vast subject. Essentially we are trying to find simple ways of representing a complicated function f by a simple function p, e.g. a polynomial or a rational function (the latter case corresponding to Padé approximation). A good introductory discussion for the former case (in the context of approximating what we usually consider to be elementary functions) can be found in Muller (1997). The nature of the error is very important. We could try and minimize the "average error" over an interval (a, b). The typical version of this is least-squares approximation, where one minimizes

$$\|p - f\|_2 = \sqrt{\int_a^b w(x)[f(x) - p(x)]^2 \, \mathrm{d}x}, \tag{19.8}$$

where $w(x)$ is a continuous (positive) weight function. One can see that this problem will naturally be solved using expansions in terms of orthogonal polynomials. We could also try and solve the maximum error

$$\|p - f\|_\infty = \max_{a \le x \le b} |p(x) - f(x)|. \tag{19.9}$$

Interpolation can be viewed approximation with a weight function made up of sums of delta functions.

19.5.1 Minimax approximation

Working with polynomials, we can now look for the polynomial p^* of degree n that satisfies (19.9); this is the minimax polynomial. Chebyshev showed that the minimax degree-n polynomial p^* on $[a, b]$ is characterized by $n + 2$ points x_i in the interval with equal values of $\|f - p^*\}$. The largest approximation is reached at $n + 2$ point and also alternates in sign. This is the basis of the Remez algorithm that computes p^* for a continuous function.

The Matlab Signal Processing Toolbox includes the Remez algorithm as part of `firpm`, a function that designs linear-phase FIR filters. This is out my domain of expertise, but I put together something that I thought would use this to produce a minimax approximation. The results is shown in Figure 19.4. Obviously I don't know what I'm doing.

```
% p192.m Remez algorihm for inverse Gamma function SGLS 06/03/08
x1 = 0:0.1:0.9; x2 = sin(0.5*pi*x1); ig1 = 1./gamma(x1); ig2 = 1./gamma(x2);
b1 = firpm(17,x1,ig1); b2 = firpm(3,x2,ig2);
[h1,w1] = freqz(x1,1); [h2,w2] = freqz(x2,1);
plot(w1/pi,abs(h1),x1,ig1,w2/pi,abs(h2),x2,ig2)
xlabel('x'); ylabel('p(x)')
```

One can download a more helpful algorithm and obtain Figure 19.5. Note that this package attracts unjust scornful comments on the web site, since it does not produce a filter. However, it commits the *faux pas* of using the same names as already existing Matlab functions.

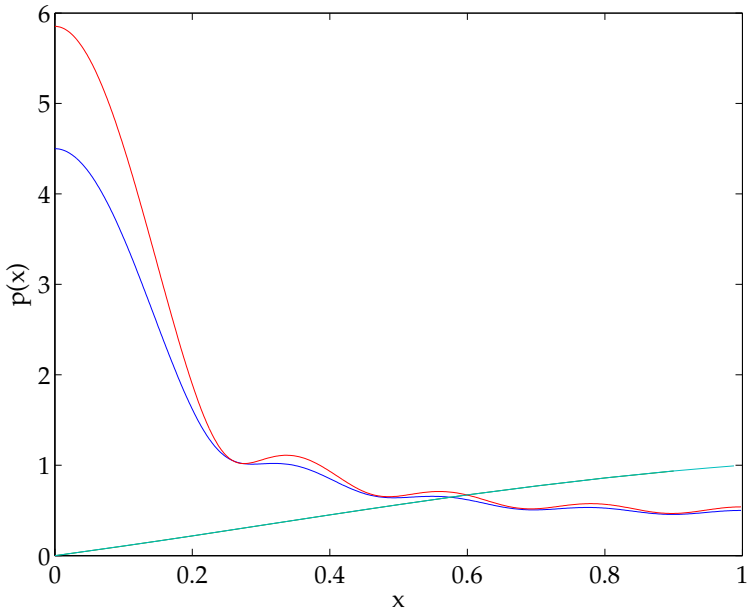

Figure 19.4: Supposedly minimax approximation to $1/\Gamma(x)$ on $(0,1)$.

```
% p193.m Remez algorithm SGLS 06/03/08
function p193

f = inline('1./gamma(x)');
fp = inline('-psi(x).*gamma(x)');
p = remez0(f,fp,[0 1],3);
p1 = p(1:end-1)
E = p(end)

x = (0:0.01:1)';
e = err(x,f,p,0);
plot(x,f(x),x,f(x)+e)
xlabel('x'); ylabel('f(x) and p(x)')

% By Sherif A. Tawfik, Faculty of Engineering, Cairo University
function A=remez0(fun, fun_der,interval,order)

powers=ones(order+2,1)*([0:order]);% the powers of the polynomial repeated in rows (ord
coeff_E =(-1).^[1:order+2];
coeff_E=coeff_E(:);     % the coefficients of the E as a column array
t=1:order;
t=t(:); % the powers of the polynomial starting from 1 in a column. This is used when di
%the polynomial
```

```
y=linspace(interval(1),interval(2),order+2); % the first choice of the (order+2) points

for i=1:10
    y=y(:); % make the points array a column array
    h=(y-interval(1))*ones(1,order+1); % repeat the points column minus the start of the
                                       %(order +1) times
    coeff_h=h.^powers;   % raise the h matrix by the power matrix elementwise
    M=[coeff_h coeff_E]; % the matrix of the LHS of the linear system of equations
    N= feval(fun,y); % the column vector of the RHS of the linear system of equations
    A=M\N;  % solution of the linear system of equations, first (order +1) element are
            % coefficients of the polynomial. Last element is the value of
            % the error at these points
    A1=A(1:end-1); % the coefficients only
    A_der=A(2:end-1).*t;   % the coeffcients of the derivative of the polynomial

    z(1)=interval(1);  % z(1) is the start point of the interval
    z(order+3)=interval(2);   % z(order+3) is the end point of the interval
    % in between we fill in with the roots of the error function
    for k=1: order+1
            z(k+1)=findzero0(@err,y(k),y(k+1),fun,A1,interval(1));
    end

    % between every two points in the array z, we seek the point that
    % maximizes the magnitude of the error function. If there is an extreme
    % point (local maximum or local minimum) between such two points of the
    % z array then the derivative of the error function is zero at this
    % extreme point. We thus look for the extreme point by looking for the
    % root of the derivative of the error function between these two
    % points. If the extreme point doesn't exist then we check the value of the  error
    % at the two current points of z and pick the one that gives maximum
    % magnitude

    for k=1:order+2
        if sign(err(z(k),fun_der,A_der,interval(1) ))~=sign(err(z(k+1),fun_der,A_der,in
            y1(k)=findzero0(@err,z(k),z(k+1),fun_der,A_der,interval(1)); % the extreme
            v(k)=abs(err(y1(k),fun,A1,interval(1))); % the value of the error function
        else  % if there is no change in sign therefore there is no extreme point and w
            v1=abs(err(z(k),fun,A1,interval(1))); % magnitude of the error function at
            v2=abs(err(z(k+1),fun,A1,interval(1))); % magnitude of the error function a
            % pick the larger of the two
            if v1>v2
                y1(k)=z(k);
                v(k)=v1;
            else
```

```
                y1(k)=z(k+1);
                v(k)=v2;
            end
        end
    end
    [mx ind]=max(v); % search for the point in the extreme points array that gives maxi
    % if the difference between this point and the corressponding point in
    % the old array is less than a certain threshold then quit the loop
    if abs(y(ind)-y1(ind)) <2^-30
        break;
    end
    % compare it also with the following point if it is not the last point
    if ind<length(y) & abs(y(ind+1)-y1(ind))  < 2^-30
        break
    end
    % replace the old points with the new points
    y=y1;
end

% By Sherif A. Tawfik, Faculty of Engineering, Cairo University
function e= err(x,fun, A, first)

% the polynomial coefficients array , make it a column array
A=A(:);

% the argument array , make it a column array
x=x(:);

% order of the polynomial is equal to the number of coefficients minus one
order=length(A)-1;

% the powers out in a row and repeated for each argument to form a matrix
% for example if the order is 2 and we have 3 arguments in x then
%         [0 1 2]
% powers= [0 1 2]
%         [0 1 2]
powers=ones(length(x),1)*[0:order];

% each argument is repeated a number of times equal to the number of
% coefficients to form a row then each element of the resulting row is
% raised with the corresponding power in the powers matrix
temp=((x-first)*ones(1,order+1)).^powers;

% multiply the resulting matrix with the coefficients table in order to
% obtain a column array. Each element of the resulting array is equal to
```

```
% the polynomial evaluated at the distance between the corresponding
% argument and the start of the interval
temp=temp*A;

% the error vector is then given as the difference between the function
% evaluated at the argument array and the polynomial evaluated at the
% argument array
e=feval(fun,x)-temp;

% By Sherif A. Tawfik, Faculty of Engineering, Cairo University
function y=findzero0(fun,x0,x1,varargin)
% fun is the function that we need to compute its root.
% x0 and x1 are two arguments to the function such that the root that we
% seek lie between them and the function has different sign at these two
% points
% varargin are the other arguments of the function

% the value of the function at the first point
f0=feval(fun,x0,varargin{:});

%the value of the function at the second point
f1=feval(fun,x1,varargin{:});

% check that the sign of the function at x0 and x1 is different. Otherwise
% report an error
if sign(f0)==sign(f1)
    error('the function at the two endpoints must be of opposite signs');
end

%find a closer point to the root using the method of chords. In the method
%of chords we simply connect the two points (x0, f(x0))  and (x1, f(x1))
%using a straight line and compute the intersection point with this line
%and the horizontal axis. This new point is closer to the desired root
x=x0 - f0 * ((x1-x0)/(f1-f0));

%evaluate the function at this new point
f=feval(fun,x,varargin{:});

% enter this root as long as the difference between the two points that
% sandwitch the desired root is larger than a certain threshold
while abs(f)>2^-52
    % we keep one of the two old points that has a different sign than the
    % new point and we overwrite the other old point with the new point
    if sign(f)==sign(f0)
        x0=x;
```

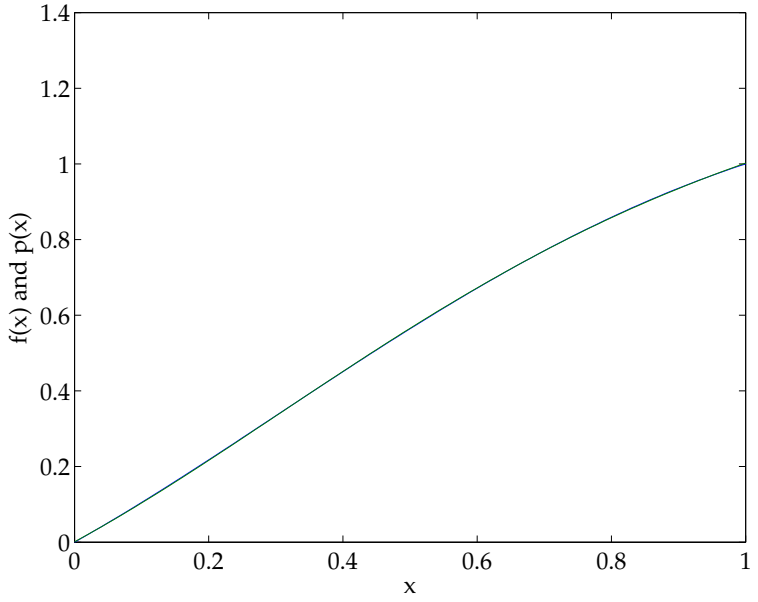

Figure 19.5: Good minimax approximation to $1/\Gamma(x)$ on $(0,1)$.

```
        f0=f;
    else
        x1=x;
        f1=f;
    end
    x=x0 - f0 * ((x1-x0)/(f1-f0));
    f=feval(fun,x,varargin{:});
end
% at the end of the loop we reach the root with the desired precision and
% it is given by x
y=x;
```

Exercise 19.2 *Why does* `p193.m` *need derivatives?*

19.5.2 Moving to the complex plane

Problem 5:

> Let $f(z) = 1/\Gamma(z)$, where $\Gamma(z)$ is the Gamma function, and let $p(z)$ be the cubic polynomial that best interprets $f(z)$ on the unit disk in the supremum norm $\|\cdot\|_\infty$. What is $\|f - p\|_\infty$?

Here was our answer:

Here is something with which to contend. There are some mathematical preliminaries: the coefficients of the desired cubic are purely real and the maxima will all be attained *on* the unit circle, rather than in its interior. Now one has to decide how to cope with this complex case for the Remez algorithm. The traditional idea for real-value functions of an equal ripple fit having precisely N maxima has to be abandoned. In the present case, while the fit is cubic, the error curve in fact attains five maxima. While one can make a good bit of progress with naïve application of a robust minimizer (e.g. **praxis**), one tends to fall into one or another of the many basins of attraction and settle upon a local minimum (e.g. with a *quartet* of errors around 0.217). The coefficient space is finely divided: the local apparently "optimal" solutions lie relatively close to each other, and in fact all lie close to the coefficients of the traditional series expansion about the origin, namely $(0, 1, \gamma, \gamma^2/2 - \pi^2/12)$. This problem appears the most challenging one for any significant increase in accuracy (probably followed next by problem 3). This task requires an accurate routine for the complex Psi function (e.g. Fullerton's). The MATLAB routines available at http://www.math.mu-luebeck.de/workers/modersitzki/COCA/coca5.htm lead to a result correct to 14 digits. Then a quick method is to compute all five error maxima, e_k, from the wholly symmetric error function:

$$E = \sum_{j=1}^{5} \sum_{k=j+1}^{5} \left(e_k(\mathbf{c}) - e_j(\mathbf{c})\right)^2 ,$$

and minimize E with Newton's method. Sensitivity of the problem means that care is needed in the choice of test increments in the coefficients if accurate finite first and second partial derivatives of E are to be found. It helps to realize that one maximum occurs at $\theta = \pi$, and symmetry of the others means that really only two more must first be located as a function of θ. Turning the crank leads to five maxima given by:

$$\begin{aligned} e_1 = e_5 &= 0.21433523459045964112541 64 \\ e_2 = e_4 &= 0.21433523459045964112541 66 \\ e_3 &= 0.21433523459045964112541 59 \end{aligned}$$

with accompanying coefficient values of:

$$\begin{aligned} c_0 &= 0.00554195085502078858444463 0869 \\ c_1 &= 1.01976185314727096802115509 7573 \\ c_2 &= 0.62521191659681903197885769 2800 \\ c_3 &= -0.60334322028589078858432807 8751. \end{aligned}$$

The Matlab output of the Coca run is given below and in Figure 19.6. The error is very flat indeed. We didn't bother enforcing symmetries, but the code do that itself. The program takes a fair while to run because we're obsessing about accuracy. One can also

probably use the Maple gamma function to get complex arguments. This version shows how to interface Matlab and Fortran in what seems to be a crude way, but which can in fact be very useful.

```
End COCA
         t     alpha         r
   0.360068528822733  -1.318823661558743   0.154252213355154
   0.500000000000000   3.141592653589793   0.303014482965043
   0.360068566101441  -1.318822407477413   0.331362413662508
   0.223326071337856   0.534002050116118   0.027976460680531
   0.223326086322160   0.534002489042060   0.183394429336764

lambda
   0.005541952172971
   1.019761855674338
   0.625211919123886
  -0.603343218967941


PARA =

                real_coef: 1
           critical_points: []
    initial_extremal_points: 'random'
                  exchange: 'single'
                  stepsize: 50
      relative_error_bound: 4.440892098500626e-16
                iterations: 57
            max_iterations: 100
        initial_clustering: 0.010000000000000
      clustering_tolerance: 1.000000000000000e-03
          newton_tolerance: 1.000000000000000e-10
           newton_stepsize: 1.000000000000000e-05
         newton_iterations: 0
     newton_max_iterations: 0
                   output: 'medium'
                     plot: 'always: abserror+boundary'
                   mywork: 'call cremez'
                  oldwork: ''
             as_subroutine: 0
                       FUN: [1x1 struct]
                     BASIS: [1x1 struct]
                  BOUNDARY: [1x1 struct]
                     C_dim: 4
                     R_dim: 4
```

```
                        t: [5x1 double]
                    alpha: [5x1 double]
                        r: [5x1 double]
              lower_bound: 0.214335234590460
                   lambda: [4x1 double]
               error_norm: 0.214335234590460
           relative_error: 2.589921873429980e-16


ans =

   0.005541952172971
   1.019761855674338
   0.625211919123886
  -0.603343218967941
```

```
clear  all
format long

% set parameters to default values
[PARA] = set_defaults;

    % Approximate                   f(z)     = 1/gamma(z)
    % by                            phi(j,z) = z^j, j=0,1,2,3
    %                               on the unit circle.

    FUN.name     = 'fgammar';          % name of approximant
    BASIS.name   = 'monom';         % name of basis function
    BASIS.choice = [0:3];           % this is the special choice
    BASIS.C_dim  = length(BASIS.choice);
    BOUNDARY.name= 'gcircle';       % name of boundary

    % there are no points, at which the partial derivative
    % does not exists
    critical_points = []';

    % the function MONOM.M has no additional parameter, so the
    % complex dimension can be defined by the length of the arguments
    C_dim       = BASIS.C_dim;

    % we will assume, that the coefficients are real
    real_coef = 1;

    % define special parameter
    % for further information on parameters see SET_PARA.M
```

```
        PARA.relative_error_bound = 2*eps;
        PARA.max_iterations       = 100;

        % for very extended output and pauses
        PARA.output = 'medium';
        % plots are shown at each iteration step
        PARA.plot   = 'always: abserror+boundary';

    % addparameter specific parameters
    PARA = set_defaults(PARA,FUN,BASIS,BOUNDARY,...
                        critical_points,real_coef,C_dim);

    FUN
    BASIS
    BOUNDARY
    PARA
    pause

    % call the COCA package
    [PARA] = coca(PARA);

    % do some output
    disp( [ '         t', '     alpha','         r'] )
    disp([PARA.t, PARA.alpha, PARA.r]);
    disp(['lambda']);
    disp([PARA.lambda]);
%=========================================================================
PARA
PARA.lambda
```

```
function [f,df]=fgammar(dummy, gamma)

n=size(gamma,1);
x=real(gamma);
y=imag(gamma);
z=zeros(2*n,1);
z(1:2:end)=x;
z(2:2:end)=y;
save dummy n z -ascii -double
!fgdo < dummy > /dev/null
load fgout
df=fgout(:,1)+i*fgout(:,2);

z=gamma;
```

```
t1=z.^2;
t2=t1.^2;
t3=t2.*z;
t4=t2.^2;
t5=t4.^2;
t10=t2.*t1;
t13=t1.*z;
t21=t2.*t13;
t30=-0.369680561864220 6D-11.*t5.*t3+0.7782263439905071D-11.*t5.*t2+0.5100370287454476D-
t33=t4.*z;
t40=t4.*t1;
t52=-0.4219773455554434D-1.*t10+0.72189432466631D-2.*t4-0.1165167591859065D-2.*t33-0.20
gamrecip=t30+t52;
f=gamrecip;
df=-df.*f;
```

```
      program fgdo
      implicit double precision (a-h,p-z)

      double complex zz,zpsi,zgrp

      open (10,file='fgout')
      read (5,*) fn
      do j=1,int(fn)
         read (5,*) zr
         read (5,*) zi
         zz=zr+(0,1)*zi
         zgrp=zpsi(zz)
         write (10,9000) dble(zgrp),dimag(zgrp)
      enddo
      close(10)
 9000 format (f60.40,1x,f60.40)
      end
```

19.6 Quadrature

One of the important uses of orthogonal polynomials is in developing efficient quadrature routines by playing with both weights and nodes. This is discussed extensively in NR, as well as in Andrews *et al.* (1999).

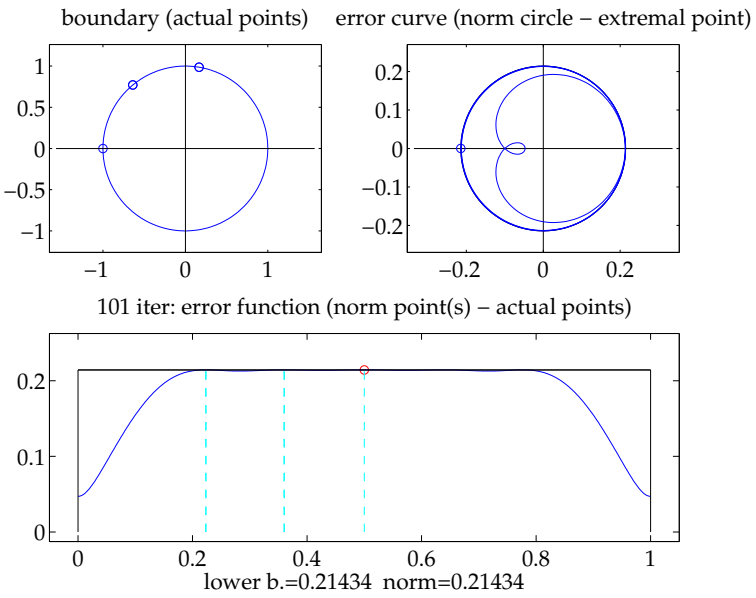

Figure 19.6: Output of COCA for Problem 5.

19.7 References

Publications:

- Andrews, G. E., Askey, R. & Roy, R. *Special functions*, Cambridge University Press, Cambridge, 1999.
- Matsuno, T. Quasigeostrophic motions in the equatorial area. *J. Met. Soc. Japan*, **44**, 25–43, 1966.
- Muller, J.-M. *Elementary functions: algorithms and implementations*, Birkhäuser, Boston, 1997.
- Price, J .D. *Numerical solution of Sturm–Liouville problems*, Clarendon Press, Oxford, 1993.

Web sites:

- http://www.math.mu-luebeck.de/modersitzki/COCA/coca5.html
- http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=8094&objectTyp

Chapter 20

Hypergeometric functions (06/04/08)

20.1 Hypergeometric series

We all know the exponential series

$$\exp z = 1 + z + \frac{z^2}{2} + \frac{z^3}{3!} + \cdots, \tag{20.1}$$

converging everywhere in the finite complex plane, and the binomial series

$$\frac{1}{1-z} = 1 + z + z^2 + z^3 + \cdots, \tag{20.2}$$

converging inside the unit disk $|z| < 1$ as well as on the unit circle $|z| = 1$ but not at $z = 1$. In fact the radius of convergence is just the distance to the nearest singularity in the complex plane.

One can think of a simple generalization to the above, with

$$\begin{aligned} {}_2F_1(a,b;c;z) &= 1 + \frac{ab}{c}z + \frac{a(a+1)b(b+1)}{c(c+1)}\frac{z^2}{2!} + \cdots \\ &\quad + \frac{a(a+1)\ldots(a+n-1)b(b-1)\ldots(b+n-1)}{c(c+1)\ldots(c+n-1)}\frac{z^3}{n!} + \cdots, \end{aligned} \tag{20.3}$$

at least in a formal sense. There are 2 parameters in the numerator (symmetric in a and b) and 1 in the denominator. We can simplify notation by defining the Pochhammer symbol

$$(a)_n = a(a+1)\ldots(a+n-1), \qquad (a)_0 = 1. \tag{20.4}$$

Note that $(1)_n = n!$, so 1 in the numerator parameters cancels with the factorial on the bottom. When $a = c$ or $b = c$, the two parameters cancel and we obtain

$${}_2F_1(a,b;b;z) = {}_1F_0(a;;z) = 1 + az + (a)_2\frac{z}{2} + \cdots + (a)_n\frac{z^n}{n!} + \cdots = (1-z)^{-a}, \tag{20.5}$$

Then ${}_1F_0(1;;z) = (1-z)^{-1}$, a function we have seen before. The exponential function on the other hand is clearly ${}_0F_0(z)$. Call the ${}_2F_1$ functions Gaussian. Anything with more than two parameters is usually called a generalized hypergeometric series.

Let us think about convergence. For ${}_2F_1$, the ratio test involves

$$\frac{(a+n)(b+n)z}{(c+n)n} \to z \quad \text{as } n \to \infty, \tag{20.6}$$

so the series converges in the disk $|z| < 1$. If a or b are negative integers, the series terminates. If c is a negative integer, the series is not defined, unless one of the numerator parameters is also a negative integer such that the series terminates first. Varying the number of parameters, ${}_1F_0$ functions have the same convergence criteria, while ${}_0F_0$ functions converge everywhere in the complex plane. We can synthesize all this.

Theorem 11 *The hypergeometric function ${}_pF_q(\ldots;z)$ converges in the entire complex plane if $p < q+1$, in the unit disk if $p = q+1$ and only at the origin if $p > q+1$. The behavior on the unit circle in the second case has to be looked at carefully.*

From a computational perspective, we can think about summing power series for $p \leq q+1$, but we face the usual difficulties of serious cancellation during this process. This is the approach of `genHyper.m` as already mentioned. It is not entirely clear how we can compute anything sensible for the $p > q+1$ case, since the series converges only at the origin. To make more progress, we will need an ODE or integral representation.

In the mean time, though, we find that Matlab links to Maple's implementation of the hypergeometric function. Figure 20.1 shows plots of the resulting complex values, which are unilluminating. The running times are very illuminating: avoid `genHyper.m` like the plague. It also works only within the radius of convergence. Maple is fast and works everywhere.

```
% p201.m Hypergeometric SGLS 06/04/08
x1 = 0:.05:6; x2 = 0:0.05:0.95; p = exp(i*pi/7);
subplot(2,2,1)
maple restart; tic, f1 = hypergeom(0.2,[],x1*p); t1 = toc
tic, f2 = zeros(size(x2));
for j = 1:length(x2)
  f2(j) = genHyper(0.2,[],x2(j)*p,0,0,15);
end
t2 = toc
plot(x1,real(f1),x1,imag(f1),x2,real(f2),x2,imag(f2))
xlabel('x'); ylabel('_1F_0'); title(['t_1 = ',num2str(t1),', t_2 = ',num2str(t2)])
subplot(2,2,2)
maple restart; tic, f1 = hypergeom([],-1/3,x1*p); t1 = toc
tic, f2 = zeros(size(x2));
for j = 1:length(x1)
  f2(j) = genHyper([],-1/3,x1(j)*p,0,0,15);
end
t2 = toc
plot(x1,real(f1),x1,imag(f1),x1,real(f2),x1,imag(f2))
xlabel('x'); ylabel('_0F_1'); title(['t_1 = ',num2str(t1),', t_2 = ',num2str(t2)])
```

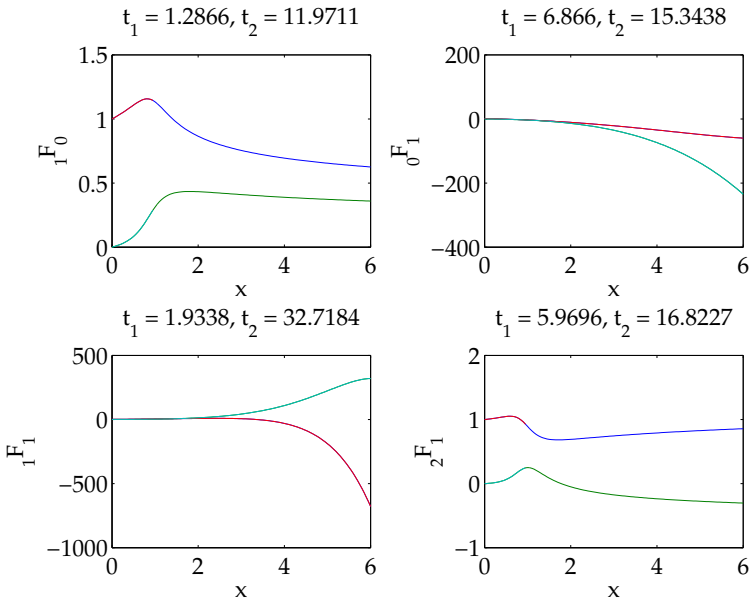

Figure 20.1: Evaluating hypergeometric functions using different Matlab and Maple functions. In all cases z is along a ray making an angle of $\pi/7$ radians with the x-axis; (a) ${}_1F_0(0.2;;z)$, (b) ${}_0F_1(;-1/3;z)$, (c) ${}_1F_1(1/3;1/7;z)$, (d) ${}_2F_1(-1/6,2/7;-2/3;z)$.

```
subplot(2,2,3)
maple restart; tic, f1 = hypergeom(1/3,1/7,x1*p); t1 = toc
tic, f2 = zeros(size(x1));
for j = 1:length(x1)
  f2(j) = genHyper(1/3,1/7,x1(j)*p,0,0,15);
end
t2 = toc
plot(x1,real(f1),x1,imag(f1),x1,real(f2),x1,imag(f2))
xlabel('x'); ylabel('_1F_1'); title(['t_1 = ',num2str(t1),', t_2 = ',num2str(t2)])
subplot(2,2,4)
maple restart; tic, f1 = hypergeom([-1/6,2/7],-2/3,x1*p); t1 = toc
tic, f2 = zeros(size(x2));
for j = 1:length(x2)
  f2(j) = genHyper([-1/6,2/7],-2/3,x2(j)*p,0,0,15);
end
t2 = toc
plot(x1,real(f1),x1,imag(f1),x2,real(f2),x2,imag(f2))
xlabel('x'); ylabel('_2F_1'); title(['t_1 = ',num2str(t1),', t_2 = ',num2str(t2)])
```

20.2 The hypergeometric equation and Riemann's equation

The hypergeometric differential equation is

$$z(1-z)w'' + [c-(a+b+1)z]w' - abw = 0. \tag{20.7}$$

It has three regular singular points at 0, 1 and ∞. Provided none of the numbers c, $c-a-b$ and $a-b$ are equal to an integer, one can write down 24 different solutions. These are related by a number of linear transformation formulae of the form

$$F(a,b;c;z) = (1-z)^{c-a-b}F(c-a,c-b;c;z). \tag{20.8}$$

There are also quadratic transformation formulas.

The hypergeometric differential equation (20.7) is a special case of Riemann's differential equation with three regular points at a, b, and c:

$$w'' + \left[\frac{1-\alpha-\alpha'}{z-a} + \frac{1-\beta-\beta'}{z-b} + \frac{1-\gamma-\gamma'}{z-c}\right] w'$$
$$+ \left[\frac{\alpha\alpha'(a-b)(a-c)}{z-a} + \frac{\beta\beta'(b-c)(b-a)}{z-b} + \frac{\gamma\gamma'(c-a)(c-b)}{z-c}\right] \frac{w}{(z-a)(z-b)(z-c)} \tag{20.9}$$

We have the condition

$$\alpha+\alpha'+\beta+\beta'+\gamma+\gamma' = 0. \tag{20.10}$$

The complete set of solutions to (20.9) is

$$w = P\left\{\begin{array}{cccc} a & b & c & \\ \alpha & \beta & \gamma & z \\ \alpha' & \beta' & \gamma' & \end{array}\right\}. \tag{20.11}$$

The special case of the hypergeometric function becomes

$$w = P\left\{\begin{array}{cccc} 0 & \infty & 1 & \\ 0 & a & 0 & z \\ 1-c & b & c-a-b & \end{array}\right\}. \tag{20.12}$$

Let us try to solve (20.7) on the interval $[0,1]$ in a non-degenerate case with $a = 1/3$, $b = -1/5$ and various c, with boundary conditions $w = 1$ at $z = 0$ and $w = 2$ at $z = 1$. Since the endpoints are regular singular points, we need to be slightly careful. The exponents at the singular points are 0 and $1-c$ at 0, 0 and $c-a-b$ at 1 and a and b at infinity. The result is shown in Figure 20.2. Evidently something is wrong, even when there is a solution. Some of the cases have no solution, because the second solution is singular near singular points.

```
% p202.m Solve hypergeometric equation SGLS 06/05/08
a = 1/3; b = -0.2; cc = [0.5 -0.5 1.5 a+b];
for j = 1:4
```

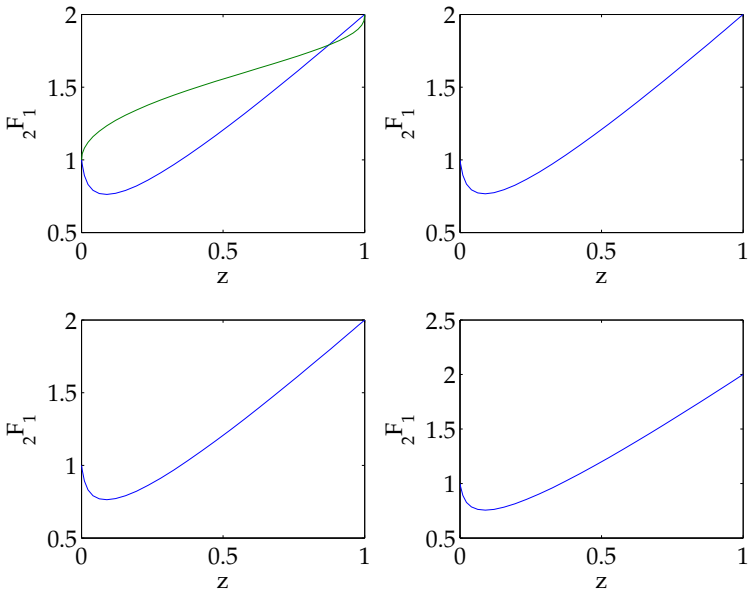

Figure 20.2: Numerical solution of hypergeometric ODE.

```
  c = cc(j)
  N = 30; [x,D] = chebdif(N+2,2); z = diag(0.5*(x+1)); I = eye(N+2);
  D1 = 2*D(:,:,1); D2 = 4*D(:,:,2);
  M = z*(1-z)*D2 + (c-(a+b+1)*z)*D1 - a*b*I;
  M(1,:) = [1 zeros(1,N+1)]; M(N+2,:) = [zeros(1,N+1) 1];
  r = [2 ; zeros(N,1) ; 1];
  w = M\r;
  z = diag(z);
  w0 = hypergeom([a,b],c,z);
  w1 = z.^(1-c).*hypergeom([a-c+1,b-c+1],2-c,z);
  C = [w0(end) w1(end) ; w0(1) w1(1)], C = C\[1 ; 2]
  wex = C(1)*w0 + C(2)*w1;
  subplot(2,2,j)
  plot(z,w,z,wex)
  xlabel('z'); ylabel('_2F_1')
end
```

Exercise 20.1 *Fix* `p202.m`.

20.3 Integral representations

The standard representation, valid for $\operatorname{Re} c > \operatorname{Re} b > 0$ is

$$F(a,b;c;z) = \frac{\Gamma(c)}{\Gamma(b)\Gamma(c-b)} \int_0^1 t^{b-1}(1-t)^{c-b-1}(1-tz)^{-a}\,\mathrm{d}t. \tag{20.13}$$

This gives an analytic function with a cut along the real z-axis from 1 to $infty$. This is hence an analytic continuation of the definition (20.3).

Exercise 20.2 *Show that*

$$F(a,b;c;1) = \frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)}, \tag{20.14}$$

c is not $0, -1, \ldots$, and Re $(c-a-b) > 0$.

The Mellin–Barnes integral is

$$F(a,b;c;z) = \frac{\Gamma(c)}{2\pi\mathrm{i}\Gamma(a)\Gamma(b)} \int_{-\mathrm{i}\infty}^{\mathrm{i}\infty} \frac{\Gamma(a+s)\Gamma(b+s)\Gamma(-s)}{\Gamma(c+s)} (-z)^s \,\mathrm{d}s. \tag{20.15}$$

The path of integration is chosen to go to the right of the poles of $\Gamma(a+s)$ and $\Gamma(b+s$, which are at the points $s = -a-n$ and $s = -b-m$ for integer n and m, and to the left of the poles of $\Gamma(-s)$, namely $s = 0, 1, \ldots$. The branch of $(-z)^2$ with $\pi < \arg(-z) < \pi$ is chosen. Cases in which $-a$, $-b$, $-c$ are negative integers are excluded; ditto for $a-b$ integer. This representation seems hard to swallow at first, but is the basis for many important results, as shown in Paris & Kaminski (2001). In particular it provides a natural way to generalize Gauss's hypergeometric function to arbitrary combinations of parameters (see § 20.5 below).

20.4 Confluent hypergeometric functions

Confluence means flowing together. We can make two of the singularities come together by taking an appropriate limit of the ODE or the sum. The result is the confluent hypergeometric equation. The standard version is to move the singularity to infinity to make an irregular singularity at infinity. The result is

$$zw'' + (b-z)w' - aw = 0. \tag{20.16}$$

There are now two parameters rather than three.

There are two independent solutions to this equation. The one that is equal to one at the origin is usually denoted

$$M(a,b,z) = 1 + \frac{az}{b} + \cdots + \frac{(a)_n z^n}{(b)_n n!} + \cdots \tag{20.17}$$

and is called Kummer's function. The second is written $U(z)$. The usual caveats about integer parameters apply and are not discussed here.

Again there are integral representations

$$M(a,b,z) = \frac{\Gamma(b)}{\Gamma(b-z)\Gamma(a)} \int_0^1 \mathrm{e}^{zt} t^{a-1} (1-t)^{b-a-1} \,\mathrm{d}t. \tag{20.18}$$

and

$$M(a,b,z) = \frac{\Gamma(b)}{2\pi\mathrm{i}\Gamma(a)} \int_{c-\mathrm{i}\infty}^{c+\mathrm{i}\infty} \frac{\Gamma(a+s)\Gamma(-s)}{\Gamma(b+s)} (-z)^s \,\mathrm{d}s. \tag{20.19}$$

The discussion of (20.19) in AS says that now $|\arg(-z) < \frac{1}{2}\pi$ and that c is finite. The presence of c is probably a red herring: one can always pull the contour back to the imaginary axis up and down the imaginary axis. The condition on the argument is less obvious, but watch out: real positive z is a problem.

Barnes–Mellin integrals are rarely used numerically, which seems a shame since the Gamma function decays agreeably quickly going up or down the imaginary axis: we have

$$\Gamma(\mathrm{i}y)\Gamma(-\mathrm{i}y) = |\Gamma(\mathrm{i}y)|^2 = \frac{\pi}{y \sinh \pi y}. \tag{20.20}$$

The condition on the contour is a pain if $a < 0$, since the contour has to wind around poles. Let's try $a = 1/3$ and $b = -2/5$. It's probably safer to use logarithms to avoid unfortunate cancellations for large $|y|$. Figure 20.3 shows the answer. The code is disappointingly slow, presumably because of the calculation of gamma functions of complex argument. Worse the results disagree, as shown in Figure 20.3. This seems to be theme for this chapter.

```
% p203.m Kummer function via BM integral SGLS 06/05/08
function [U,Um,z] = p203
a = 1/3; b = -2/5;
ym = 20; z = 0:-0.1:-1; U = zeros(size(z)); Um = U;
for j = 1:length(z)
  z(j)
  U(j) = gamma(b)/(gamma(a)*2*pi)*quad(@(y) Uint(y,a,b,z(j)),-ym,ym);
  maple('f:=KummerU',a,b,z(j));
  Um(j) = str2num(maple('evalf(f)'));
end
plot(z,real(U),z,imag(U),'--',z,real(Um),z,imag(Um),'--')
xlabel('z'); ylabel('U(z)')


function f = Uint(y,a,b,z)

s = -0.1 + i*y;
f = mfun('lnGAMMA',-s)+mfun('lnGAMMA',a+s)-mfun('lnGAMMA',b+s);
if (z~=0)
  f = f+s*log(-z);
end
f = exp(f);
```

Exercise 20.3 *From (20.18) explain the limiting procedure to go from Gauss's hypergeometric function to Kummer's function.*

Exercise 20.4 *Why do the results in Figure 20.3 disagree?*

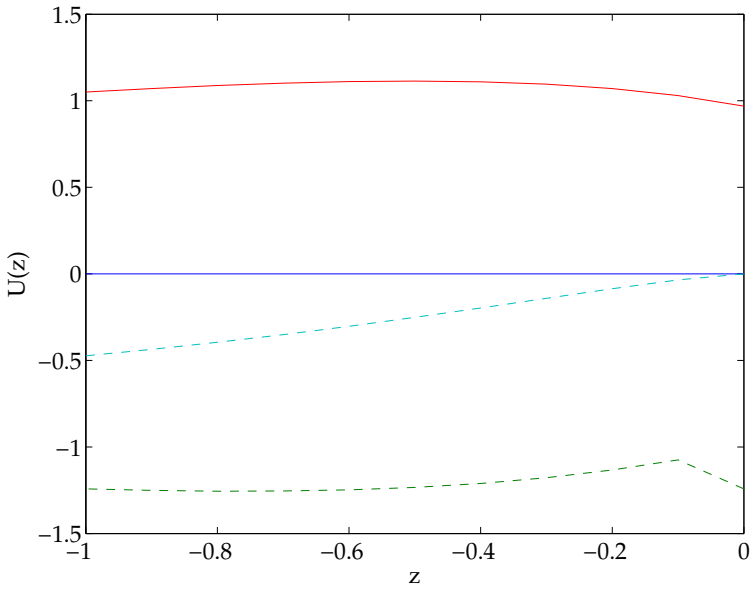

Figure 20.3: Kummer's function calculated using the Barnes–Mellin integral.

Another pair of linearly independent solutions of the confluent hypergeometric equation come the solutions $M_{\kappa,\mu}(z)$ and $W_{\kappa,\mu}(z)$ of Whittaker's equation

$$w'' + \left[-\frac{1}{4} + \frac{\kappa}{z} + \frac{\frac{1}{4} - \mu^2}{z^2} \right] w = 0. \tag{20.21}$$

The relation between the two families is

$$M_{\kappa,\mu}(z) = \mathrm{e}^{-\frac{1}{2}z} z^{\frac{1}{2}+\mu} M(\tfrac{1}{2} + \mu - \kappa, 1 + 2\mu, z) \tag{20.22}$$

$$W_{\kappa,\mu}(z) = \mathrm{e}^{-\frac{1}{2}z} z^{\frac{1}{2}+\mu} U(\tfrac{1}{2} + \mu - \kappa, 1 + 2\mu, z). \tag{20.23}$$

20.5 Meijer G-functions

This is a very general function indeed, and related to the generalized hypergeometric function. Mathai (1993) and Mathai & Haubold (2008) are quite enamored of it and of its applications to statistics and astrophysics problems. One starts from a Barnes–Mellin type integral

$$G^{m,n}_{p,q}(a_1, \ldots a_p; b_1, \ldots b_q; z) = \frac{1}{2\pi \mathrm{i}} \int_L \frac{\left\{ \prod_{j=1}^{m} \Gamma(b_j + s) \right\} \left\{ \prod_{j=1}^{n} \Gamma(1 - a_j - s) \right\}}{\left\{ \prod_{j=m+1}^{q} \Gamma(1 - b_j - s) \right\} \left\{ \prod_{j=n+1}^{p} \Gamma(a_j + s) \right\}} \, \mathrm{d}s, \tag{20.24}$$

where the contour L separates the poles of $\Gamma(b_j + s)$, $j = 1, \ldots, m$ (which extend off into the left half-plane) from the poles of $\Gamma(1 - a_j - s)$, $j = 1, \ldots, n$ (which extend off into

the right half-plane). The poles of the Gamma functions in the denominator are not a problem. The conditions for existence of the function are detailed carefully in Mathai (1993), but essentially we have existence for all non-zero z for $q \geq 1$, $q > p$ or $p \geq 1$, $p > q$, for $|z| < 1$ for $q \geq 1$, $q = p$, and for $|z| > 1$ for $p \geq 1$, $p = q$.

One can be even more general and talk about something called Fox's H-function. We shan't.

20.6 Conformal mappings

Hypergeometric functions enter naturally when one seeks conformal maps from curvilinear polygons to the upper half-plane or the unit disk. These mappings arise naturally in free-boundary problems in porous medium flow (Polubarinova-Kochina 1962).

Suppose that we have a curvilinear polygon, that is to say a polygon bounded by arcs of a circle

$$\operatorname{Im} \frac{k_s + l_s w}{m_s + n_s w} = 0, \tag{20.25}$$

for $s = 0, 1, \ldots, \nu + 1$, where w is a complex variable. The angles at the vertices are $\pi\alpha_1, \ldots, \pi\alpha_{\nu+1}$. we want to map this polygon to the upper half-plane ζ. The vertices of the polygon map onto points $a_1, \ldots a_{\nu+1}$ on the real axis with $a_{\nu+1} = \infty$. The mapping is constructed from two linearly independent solutions to the second-order ODE

$$w'' + \left[\frac{1-\alpha_1}{\zeta - a_1} + \cdots + \frac{1-\alpha_\nu}{\zeta - a_\nu}\right] w' + \frac{\alpha'_{\nu+1}\alpha''_{\nu+1}(\zeta - \lambda_1)\ldots(\zeta - \lambda_{\nu-2})}{(\zeta - a_1)\ldots(\zeta - z_\nu)} w = 0. \tag{20.26}$$

We have $\alpha'_{\nu+1} - \alpha''_{\nu+1} = \alpha_{\nu+1}$. We must also satisfy

$$\alpha_1 + \cdots + \alpha_\nu + \alpha'_{\nu+1} + \alpha''_{\nu+1} = \nu - 1. \tag{20.27}$$

The quantities $\lambda_1, \ldots, \lambda_{\nu-2}$ are additional parameters that are not known in advance. For a triangle, they do not exist. This corresponds to curvilinear triangles. If the linearly independent solutions of (20.26) are written $U_s(\zeta)$ and $V_s(\zeta)$, the mapping is given by

$$w = \frac{AU_S + BV_s}{CU_s + DV_S}, \tag{20.28}$$

where A, B, C, D are constants.

We see that for triangles, this procedure can be carried out explicitly. For polygons with four or more sides, there is a problem. Craster (1997) has an interesting approach that works in certain degenerate cases using the Mellin transform. This would lead us too far astray.

Exercise 20.5 *Verify that (20.25) does in fact correspond to an arc of a circle.*

20.7 References

Publications:

- Craster, R. V¿ The solution of a class of free boundary problems. *Proc. R. Soc. Lond.* A, **453**, 607–630.
- Mathias, A. M. *A handbook of generalized special functions for statistical and physical sciences*, Clarendon Press, Oxford, 1993.
- Mathai, A. M. & Haubold, H. J. *Special functions for applied scientists*, Springer, Berlin, 2008.
- Paris, R. B. & Kaminski, D. *Asymptotics and Mellin–Barnes integrals*, Cambridge University Press, Cambridge, 2001.
- Polubarinova-Kochina, P. Y. *Theory of ground water movement*, Princeton University Press, Princeton, 1962.
- Slater, L. J. *Confluent hypergeometric functions*, Cambridge University Press, Cambridge, 1960.
- Slater, L. J. *Generalized hypergeometric functions*, Cambridge University Press, Cambridge, 1966.

Appendix A

Listings (05/20/2008)

A.1 Graphical setup

```
% gsteup.m Graphics setup 04/08/08 SGLS
close all
clf
set(gcf,'DefaultAxesFontName','Palatino ','DefaultAxesFontSize',16)
```

A.2 Programs

A.2.1 Kdo.f

```
c     g77 -O -o Kdo Kdo.f talbot.f -lslatec -Wl,-framework -Wl,vecLib
      program Kdo

      implicit double precision (a-h,p-z)
      double complex p,fun,Fc
      parameter (NN=1000)
      dimension g(0:NN+1),Fc(0:NN+1),zk(0:NN)
      common zk,p,h,e,m,n,MM
      external dbesj1,fun

      read (5,*) h,e
      read (5,*) MM,tmax,tstep,n

      pi = 4d0*atan(1d0)
      zk(0) = 0d0
      do m = 1,MM
         b = (m - 0.25d0)*pi
         c = (m + 0.75d0)*pi
         r = (m + 0.25d0)*pi
```

```
         call dfzero(dbesj1,b,c,r,0d0,1d-15,iflag)
         write (11,*) 'dfzero',m,b,iflag
         zk(m) = b
      enddo

      t = 0d0
100   if (abs(t-tmax).gt.1d-6) then
         t = t + tstep
         call stalbot(fun,t,n,dcmplx(0d0,0d0),0,1,0d0,MM+2,Fc,g)
         write (6,1000) t,(g(j),j=0,MM+1)
         goto 100
      endif
1000  format (5000(1x,e32.26))
      end


      double complex function fun(s,Fc)

      implicit double precision (a-h,p-z)
      parameter (NN=1000)
      double complex s,Fc(0:NN+1)
      double complex p,mu,T(0:NN,0:NN),F(0:NN)
      dimension ipiv(0:NN),zk(0:NN)
      parameter (epsabs=1d-6,epsrel=0d0)
      parameter (limit=10000,lenw=limit*4)
      dimension iwork(limit),work(lenw)
      common zk,p,h,e,m,n,MM
      external Trint,Tiint

      p = s
      write (11,*) p,h,e,MM
      call xsetf(0)
      do m = 0,MM
         if (m.eq.0) then
            mu = sqrt(p)
            F(m) = 0.5d0/p/e*exp(mu*(e-h))*(1d0-exp(-2d0*mu*e))/
     $           (1d0-exp(-2d0*mu*h))
         else
            F(m) = (0d0,0d0)
         endif
         do n = 0,m
            call dqagi(Trint,0d0,1,epsabs,epsrel,rr,abserr,
     $           neval,ier,limit,lenw,last,iwork,work)
            call out(m,n,rr,abserr,neval,ier,lst,iwork,1)
            call dqagi(Tiint,0d0,1,epsabs,epsrel,ri,abserr,
```

```
$           neval,ier,limit,lenw,last,iwork,work)
       call out(m,n,ri,abserr,neval,ier,lst,iwork,1)
       T(m,n) = dbesj0(zk(m))*dbesj0(zk(n))*dcmplx(rr,ri)
    enddo
    write (11,*)
 enddo
 do m = 0,MM
    do n = m+1,MM
       T(m,n)  = T(n,m)
    enddo
 enddo
 do m = 0,MM
    do n = 0,MM
       T(m,n) = 2d0/dbesj0(zk(n))**2*T(m,n)
    enddo
    mu = sqrt(p + zk(m)**2)
    T(m,m) = T(m,m) + 1d0/mu
$        *(1d0+exp(-2d0*mu*h))/(1d0-exp(-2d0*mu*h))
 enddo

 do m = 0,MM
    write (11,*) 'T',(T(m,n),n=0,MM),' F',F(m)
 enddo
 call zgesv(MM+1,1,T,NN+1,ipiv,F,NN+1,info)
 write (11,*) 'zgesv',info
 write (11,*) 'F ',(F(m),m=0,MM)

 do m = 0,MM
    Fc(m) = F(m)
 enddo
 pi = 4d0*atan(1d0)
 Fc(MM+1) = pi*(1d0-2d0*F(0))/p
 end


 double complex function Tint(v)

 implicit double precision (a-h,p-z)
 parameter (NN=1000)
 double complex p,mu
 double precision zk(0:NN)
 common zk,p,h,e,m,n,MM
 external dbesj1

 if (v.ne.0d0.and.v.ne.zk(m).and.v.ne.zk(n)) then
```

```
         Tint = v**3/(v**2-zk(m)**2)/(v**2-zk(n)**2)
     $        *dbesj1(v)**2/sqrt(p+v**2)
      else
         Tint = (0d0,0d0)
      endif
      end


      subroutine out(m,n,result,abserr,neval,ier,lst,iwork,j)

      implicit double precision (a-h,p-z)
      dimension iwork(*)

      if (j.eq.2) then
         if (ier.eq.1.or.ier.eq.4.or.ier.eq.7) then
            write (11,1010) m,n,result,abserr,neval,ier,lst,
     $           (iwork(k),k=1,lst)
         else
            write (11,1000) m,n,result,abserr,neval,ier,lst
         endif
      else
         write (11,1000) m,n,result,abserr,neval,ier
      endif
 1000 format(2(i2,1x),2(e24.16,1x),i6,1x,i1,1x,i4)
 1010 format(2(i2,1x),2(e24.16,1x),i6,1x,i1,1x,i4,1x,100(i2,1x))
      end


      double precision function Trint(v)

      implicit double precision (a-h,p-z)
      double complex Tint

      Trint = dble(Tint(v))
      end


      double precision function Tiint(v)

      implicit double precision (a-h,p-z)
      double complex Tint

      Tiint = dimag(Tint(v))
      end
```

A.2.2 talbot.f

```
      double precision function talbot(fun,t,D,sj,mult,ns,aa)
      implicit double precision (a-h,p-z)

      double complex sj(ns),spj
      dimension mult(ns)
      logical rsing
      integer D,Dp,Dd,Dj,c
      double precision kappa,lambda,mu,nu,omega
      double complex fun,sd,sstar,snu,s,ak,b1,b2,bk
      parameter (c=15)
      common /talvar/ lambda, sigma, nu, n0, n1, n2, n
      external fun

      pi = 4d0*atan(1d0)
      rsing = .true.
      phat = dble(sj(1))
      do j = 1,ns
         pj = dble(sj(j))
         if (phat.le.pj) phat = pj
      enddo
      sigma0 = max(0d0,phat)
      qd = 0d0
      thetad = pi
      multd = 0
      rmax = 0d0
      do j = 1,ns
         if (dimag(sj(j)).ne.0d0) then
            rsing = .false.
            spj = sj(j) - phat
            r = dimag(spj)/atan2(dimag(spj),dble(spj))
            if (r.gt.rmax) then
               sd = sj(j)
               qd = abs(dimag(sd))
               thetad = atan2(qd,dble(spj))
               multd = mult(j)
            endif
         endif
      enddo

      v = qd*t
      if (aa.eq.0d0) then
         omega = min(0.4d0*(c+1d0)+v/2d0,2d0*(c+1d0)/3d0)
      else
```

```
         omega = 5d0*(c-1)/13d0 + aa*t/30d0
      endif
      if (v.le.omega*thetad/1.8d0) then
         lambda = omega/t
         sigma = sigma0
         nu = 1d0
      else
         kappa = 1.6d0 + 12d0/(v+25d0)
         phi = 1.05d0 + 1050d0/max(553d0,800d0-v)
         mu = (omega/t + sigma0 - phat)/(kappa/phi - 1d0/tan(phi))
         lambda = kappa*mu/phi
         sigma = phat - mu/tan(phi)
         nu = qd/mu
      endif
      tau = lambda*t
      a = (nu-1d0)*0.5d0
      sigmap = sigma - sigma0
c      write (6,*) 'lambda sigma nu',lambda,sigma,nu

c     determine n
c     n0
      Dd = D + min(2*multd-2,2) + int(multd*0.25d0)
      if (v.gt.omega*thetad/1.8d0.or.multd.eq.0.or.rsing) then
         n0 = 0
      else
         sstar = (sd-sigma0)/lambda
         p = dble(sstar)
         q = dimag(sstar)
         r = abs(sstar)
         theta = atan2(q,p)
         y = 2d0*pi-13d0/(5d0-2d0*p-q-0.45d0*exp(p))
 100     ul = (y-q)/(r*sin(y-theta))
         if (ul.le.0d0) then
            n0 = 0
         else
            u = log(ul)
            g = dble(sstar)*(1-exp(u)*cos(y))
     $           - dimag(sstar)*exp(u)*sin(y) + u
            dy = (q-y)*g/(1d0-2d0*r*exp(u)*cos(y-theta)+r**2*exp(2*u))
            y = y + dy
            if (abs(dy).ge.1e-4) goto 100
            ppd = phat - sigma0
            n0 = int((2.3d0*Dd+ppd*t)/u)
         endif
      endif
```

```
c     n1
      e = (2.3d0*D+omega)/tau
      if (e.le.4.4d0) then
         rho = (24.8d0-2.5d0*e)/(16d0+4.3d0*e)
      elseif (e.le.10d0) then
         rho = (129d0/e-4d0)/(50d0+3d0*e)
      else
         rho = (256d0/e+0.4d0)/(44d0+19d0*e)
      endif
      n1 = int(tau*(a+1d0/rho))+1
c     n2
      gamma = sigmap/lambda
      if (rsing) then
         Dp = 0
         do j = 1,ns
            Dj = D + min(2*mult(j)-2,2) + int(mult(j)*0.25d0)
            if (Dj.gt.Dp) Dp = Dtemp
         enddo
      else
         Dp = Dd
      endif
      y = 1d-3*v
      eta = (1.09d0-0.92d0*y+0.8d0*y**2)*min(1.78d0,
     $     1.236d0+0.0064d0*1.78d0**Dp)
      n2 = int(eta*nu*(2.3d0*Dp+omega)/(3d0+4d0*gamma+exp(-gamma)))+1
      n = max(n0,max(n1,n2))
c      write (6,*) 'n n0 n1 n2',n,n0,n1,n2

c     sum over paraboloid
      psi = tau*nu*pi/n
      u = 2d0*cos(psi)
      b2 = 0d0
      b1 = 0d0
      do k = n-1,1,-1
         theta = k*pi/n
         alpha = theta/tan(theta)
         snu = dcmplx(alpha,nu*theta)
         s = lambda*snu+sigma
         beta = theta+alpha*(alpha-1d0)/theta
         ak = exp(alpha*tau)*dcmplx(nu,beta)*fun(s)
         bk = ak + u*b1 - b2
         b2 = b1
         b1 = bk
      enddo
      s = lambda+sigma
```

```
      ak = exp(tau)*nu*fun(s)
      talbot = lambda*exp(sigma*t)/n*
     $     dble( 0.5d0*(ak + u*b1) - b2
     $     + dcmplx(0d0,1d0)*b1*sin(psi) )
 1000 format (f30.16,1x,f30.16)
      return
      end


      double complex function ztalbot(fun,t,D,sj,mult,ns,aa)
      implicit double precision (a-h,p-z)

      double complex sj(ns),spj
      dimension mult(ns)
      logical rsing
      integer D,Dp,Dd,Dj,c
      double precision kappa,lambda,mu,nu,omega
      double complex fun,sd,sstar,snu,s,ak,b1,b2,bk,am,bm1,bm2,bm
      parameter (c=15)
      common /talvar/ lambda, sigma, nu, n0, n1, n2, n
      external fun

      pi = 4d0*atan(1d0)
      rsing = .true.
      phat = dble(sj(1))
      do j = 1,ns
         pj = dble(sj(j))
         if (phat.le.pj) phat = pj
      enddo
      sigma0 = max(0d0,phat)
      qd = 0d0
      thetad = pi
      multd = 0
      rmax = 0d0
      do j = 1,ns
         if (dimag(sj(j)).ne.0d0) then
            rsing = .false.
            spj = sj(j) - phat
            r = dimag(spj)/atan2(dimag(spj),dble(spj))
            if (r.gt.rmax) then
               sd = sj(j)
               qd = abs(dimag(sd))
               thetad = atan2(qd,dble(spj))
               multd = mult(j)
            endif
```

```
         endif
      enddo

      v = qd*t
      if (aa.eq.0d0) then
         omega = min(0.4d0*(c+1d0)+v/2d0,2d0*(c+1d0)/3d0)
      else
         omega = 5d0*(c-1)/13d0 + aa*t/30d0
      endif
      if (v.le.omega*thetad/1.8d0) then
         lambda = omega/t
         sigma = sigma0
         nu = 1d0
      else
         kappa = 1.6d0 + 12d0/(v+25d0)
         phi = 1.05d0 + 1050d0/max(553d0,800d0-v)
         mu = (omega/t + sigma0 - phat)/(kappa/phi - 1d0/tan(phi))
         lambda = kappa*mu/phi
         sigma = phat - mu/tan(phi)
         nu = qd/mu
      endif
      tau = lambda*t
      a = (nu-1d0)*0.5d0
      sigmap = sigma - sigma0
c      write (6,*) 'lambda sigma nu',lambda,sigma,nu

c     determine n
c     n0
      Dd = D + min(2*multd-2,2) + int(multd*0.25d0)
      if (v.gt.omega*thetad/1.8d0.or.multd.eq.0.or.rsing) then
         n0 = 0
      else
         sstar = (sd-sigma0)/lambda
         p = dble(sstar)
         q = dimag(sstar)
         r = abs(sstar)
         theta = atan2(q,p)
         y = 2d0*pi-13d0/(5d0-2d0*p-q-0.45d0*exp(p))
 100     ul = (y-q)/(r*sin(y-theta))
         if (ul.le.0d0) then
            n0 = 0
         else
            u = log(ul)
            g = dble(sstar)*(1-exp(u)*cos(y))
     $           - dimag(sstar)*exp(u)*sin(y) + u
```

```
            dy = (q-y)*g/(1d0-2d0*r*exp(u)*cos(y-theta)+r**2*exp(2*u))
            y = y + dy
            if (abs(dy).ge.1e-4) goto 100
            ppd = phat - sigma0
            n0 = int((2.3d0*Dd+ppd*t)/u)
         endif
      endif
c     n1
      e = (2.3d0*D+omega)/tau
      if (e.le.4.4d0) then
         rho = (24.8d0-2.5d0*e)/(16d0+4.3d0*e)
      elseif (e.le.10d0) then
         rho = (129d0/e-4d0)/(50d0+3d0*e)
      else
         rho = (256d0/e+0.4d0)/(44d0+19d0*e)
      endif
      n1 = int(tau*(a+1d0/rho))+1
c     n2
      gamma = sigmap/lambda
      if (rsing) then
         Dp = 0
         do j = 1,ns
            Dj = D + min(2*mult(j)-2,2) + int(mult(j)*0.25d0)
            if (Dj.gt.Dp) Dp = Dtemp
         enddo
      else
         Dp = Dd
      endif
      y = 1d-3*v
      eta = (1.09d0-0.92d0*y+0.8d0*y**2)*min(1.78d0,
     $     1.236d0+0.0064d0*1.78d0**Dp)
      n2 = int(eta*nu*(2.3d0*Dp+omega)/(3d0+4d0*gamma+exp(-gamma)))+1
      n = max(n0,max(n1,n2))
c      n = 50
c      write (6,*) 'n n0 n1 n2',n,n0,n1,n2

c     sum over paraboloid
      psi = tau*nu*pi/n
      u = 2d0*cos(psi)
      b2 = 0d0
      b1 = 0d0
      bm2 = 0d0
      bm1 = 0d0
      do k = n-1,1,-1
         theta = k*pi/n
```

```
         alpha = theta/tan(theta)
         snu = dcmplx(alpha,nu*theta)
         s = lambda*snu+sigma
         beta = theta+alpha*(alpha-1d0)/theta
         ak = exp(alpha*tau)*dcmplx(nu,beta)*fun(s)
         bk = ak + u*b1 - b2
         b2 = b1
         b1 = bk
         am = exp(alpha*tau)*dcmplx(nu,-beta)*fun(conjg(s))
         bm = am + u*bm1 - bm2
         bm2 = bm1
         bm1 = bm
      enddo
      s = lambda+sigma
      ak = exp(tau)*nu*fun(s)
      am = ak
      ztalbot = lambda*exp(sigma*t)/n*0.5d0*
     $     ( 0.5d0*(ak + am + u*(b1+bm1)) - b2 - bm2
     $     + dcmplx(0d0,1d0)*b1*sin(psi)
     $     - dcmplx(0d0,1d0)*bm1*sin(psi) )
 1000 format (f30.16,1x,f30.16)
      return
      end


      subroutine stalbot(fun,t,D,sj,mult,ns,aa,Nd,Fc,res)
      implicit double precision (a-h,p-z)

      double complex sj(ns),spj,Fc(*)
      dimension mult(ns),res(*)
      logical rsing
      integer D,Dp,Dd,Dj,c
      double precision kappa,lambda,mu,nu,omega
      parameter (NN=50000)
      double complex fun,sd,sstar,snu,s,ak(NN),b1(NN),b2(NN),bk(NN)
      parameter (c=15)
      common /talvar/ lambda, sigma, nu, n0, n1, n2, n
c      common /taldat/ Nd, Fc(Nd),res(Nd)
      external fun

      if (Nd.gt.NN) stop
      pi = 4d0*atan(1d0)
      rsing = .true.
      phat = dble(sj(1))
      do j = 1,ns
```

```
         pj = dble(sj(j))
         if (phat.le.pj) phat = pj
      enddo
      sigma0 = max(0d0,phat)
      qd = 0d0
      thetad = pi
      multd = 0
      rmax = 0d0
      do j = 1,ns
         if (dimag(sj(j)).ne.0d0) then
            rsing = .false.
            spj = sj(j) - phat
            r = dimag(spj)/atan2(dimag(spj),dble(spj))
            if (r.gt.rmax) then
               sd = sj(j)
               qd = abs(dimag(sd))
               thetad = atan2(qd,dble(spj))
               multd = mult(j)
            endif
         endif
      enddo

      v = qd*t
      if (aa.eq.0d0) then
         omega = min(0.4d0*(c+1d0)+v/2d0,2d0*(c+1d0)/3d0)
      else
         omega = 5d0*(c-1)/13d0 + aa*t/30d0
      endif
      if (v.le.omega*thetad/1.8d0) then
         lambda = omega/t
         sigma = sigma0
         nu = 1d0
      else
         kappa = 1.6d0 + 12d0/(v+25d0)
         phi = 1.05d0 + 1050d0/max(553d0,800d0-v)
         mu = (omega/t + sigma0 - phat)/(kappa/phi - 1d0/tan(phi))
         lambda = kappa*mu/phi
         sigma = phat - mu/tan(phi)
         nu = qd/mu
      endif
      tau = lambda*t
      a = (nu-1d0)*0.5d0
      sigmap = sigma - sigma0
c      write (6,*) 'lambda sigma nu',lambda,sigma,nu
```

```
c     determine n
c     n0
      Dd = D + min(2*multd-2,2) + int(multd*0.25d0)
      if (v.gt.omega*thetad/1.8d0.or.multd.eq.0.or.rsing) then
         n0 = 0
      else
         sstar = (sd-sigma0)/lambda
         p = dble(sstar)
         q = dimag(sstar)
         r = abs(sstar)
         theta = atan2(q,p)
         y = 2d0*pi-13d0/(5d0-2d0*p-q-0.45d0*exp(p))
 100     ul = (y-q)/(r*sin(y-theta))
         if (ul.le.0d0) then
            n0 = 0
         else
            u = log(ul)
            g = dble(sstar)*(1-exp(u)*cos(y))
     $           - dimag(sstar)*exp(u)*sin(y) + u
            dy = (q-y)*g/(1d0-2d0*r*exp(u)*cos(y-theta)+r**2*exp(2*u))
            y = y + dy
            if (abs(dy).ge.1e-4) goto 100
            ppd = phat - sigma0
            n0 = int((2.3d0*Dd+ppd*t)/u)
         endif
      endif
c     n1
      e = (2.3d0*D+omega)/tau
      if (e.le.4.4d0) then
         rho = (24.8d0-2.5d0*e)/(16d0+4.3d0*e)
      elseif (e.le.10d0) then
         rho = (129d0/e-4d0)/(50d0+3d0*e)
      else
         rho = (256d0/e+0.4d0)/(44d0+19d0*e)
      endif
      n1 = int(tau*(a+1d0/rho))+1
c     n2
      gamma = sigmap/lambda
      if (rsing) then
         Dp = 0
         do j = 1,ns
            Dj = D + min(2*mult(j)-2,2) + int(mult(j)*0.25d0)
            if (Dj.gt.Dp) Dp = Dtemp
         enddo
      else
```

```
         Dp = Dd
      endif
      y = 1d-3*v
      eta = (1.09d0-0.92d0*y+0.8d0*y**2)*min(1.78d0,
     $     1.236d0+0.0064d0*1.78d0**Dp)
      n2 = int(eta*nu*(2.3d0*Dp+omega)/(3d0+4d0*gamma+exp(-gamma)))+1
      n = max(n0,max(n1,n2))
c      write (6,*) 'n n0 n1 n2',n,n0,n1,n2

c     sum over paraboloid
      psi = tau*nu*pi/n
      u = 2d0*cos(psi)
      do j = 1,Nd
         b2(j) = 0d0
         b1(j) = 0d0
      enddo
      do k = n-1,1,-1
         theta = k*pi/n
         alpha = theta/tan(theta)
         snu = dcmplx(alpha,nu*theta)
         s = lambda*snu+sigma
         beta = theta+alpha*(alpha-1d0)/theta
         dummy = fun(s,Fc)
         do j = 1,Nd
            ak(j) = exp(alpha*tau)*dcmplx(nu,beta)*Fc(j)
            bk(j) = ak(j) + u*b1(j) - b2(j)
            b2(j) = b1(j)
            b1(j) = bk(j)
         enddo
      enddo
      s = lambda+sigma
      dummy = fun(s,Fc)
      do j = 1,Nd
         ak(j) = exp(tau)*nu*Fc(j)
         res(j) = lambda*exp(sigma*t)/n*
     $        dble( 0.5d0*(ak(j) + u*b1(j)) - b2(j)
     $        + dcmplx(0d0,1d0)*b1(j)*sin(psi) )
      enddo
 1000 format (f30.16,1x,f30.16)
      return
      end


      subroutine sztalbot(fun,t,D,sj,mult,ns,aa,Nd,Fc,res)
      implicit double precision (a-h,p-z)
```

```
      double complex sj(ns),spj,Fc(*),res(*)
      dimension mult(ns)
      logical rsing
      integer D,Dp,Dd,Dj,c
      double precision kappa,lambda,mu,nu,omega
      parameter (NN=50000)
      double complex fun,sd,sstar,snu,s,ak(NN),b1(NN),b2(NN),bk(NN),
     $     am(NN),bm1(NN),bm2(NN),bm(NN)
      parameter (c=15)
      common /talvar/ lambda, sigma, nu, n0, n1, n2, n
c      common /taldat/ Nd, Fc(Nd),res(Nd)
      external fun

      if (Nd.gt.NN) stop
      pi = 4d0*atan(1d0)
      rsing = .true.
      phat = dble(sj(1))
      do j = 1,ns
         pj = dble(sj(j))
         if (phat.le.pj) phat = pj
      enddo
      sigma0 = max(0d0,phat)
      qd = 0d0
      thetad = pi
      multd = 0
      rmax = 0d0
      do j = 1,ns
         if (dimag(sj(j)).ne.0d0) then
            rsing = .false.
            spj = sj(j) - phat
            r = dimag(spj)/atan2(dimag(spj),dble(spj))
            if (r.gt.rmax) then
               sd = sj(j)
               qd = abs(dimag(sd))
               thetad = atan2(qd,dble(spj))
               multd = mult(j)
            endif
         endif
      enddo

      v = qd*t
      if (aa.eq.0d0) then
         omega = min(0.4d0*(c+1d0)+v/2d0,2d0*(c+1d0)/3d0)
      else
```

```
         omega = 5d0*(c-1)/13d0 + aa*t/30d0
      endif
      if (v.le.omega*thetad/1.8d0) then
         lambda = omega/t
         sigma = sigma0
         nu = 1d0
      else
         kappa = 1.6d0 + 12d0/(v+25d0)
         phi = 1.05d0 + 1050d0/max(553d0,800d0-v)
         mu = (omega/t + sigma0 - phat)/(kappa/phi - 1d0/tan(phi))
         lambda = kappa*mu/phi
         sigma = phat - mu/tan(phi)
         nu = qd/mu
      endif
      tau = lambda*t
      a = (nu-1d0)*0.5d0
      sigmap = sigma - sigma0
c      write (6,*) 'lambda sigma nu',lambda,sigma,nu

c     determine n
c     n0
      Dd = D + min(2*multd-2,2) + int(multd*0.25d0)
      if (v.gt.omega*thetad/1.8d0.or.multd.eq.0.or.rsing) then
         n0 = 0
      else
         sstar = (sd-sigma0)/lambda
         p = dble(sstar)
         q = dimag(sstar)
         r = abs(sstar)
         theta = atan2(q,p)
         y = 2d0*pi-13d0/(5d0-2d0*p-q-0.45d0*exp(p))
 100     ul = (y-q)/(r*sin(y-theta))
         if (ul.le.0d0) then
            n0 = 0
         else
            u = log(ul)
            g = dble(sstar)*(1-exp(u)*cos(y))
     $           - dimag(sstar)*exp(u)*sin(y) + u
            dy = (q-y)*g/(1d0-2d0*r*exp(u)*cos(y-theta)+r**2*exp(2*u))
            y = y + dy
            if (abs(dy).ge.1e-4) goto 100
            ppd = phat - sigma0
            n0 = int((2.3d0*Dd+ppd*t)/u)
         endif
      endif
```

```
c     n1
      e = (2.3d0*D+omega)/tau
      if (e.le.4.4d0) then
         rho = (24.8d0-2.5d0*e)/(16d0+4.3d0*e)
      elseif (e.le.10d0) then
         rho = (129d0/e-4d0)/(50d0+3d0*e)
      else
         rho = (256d0/e+0.4d0)/(44d0+19d0*e)
      endif
      n1 = int(tau*(a+1d0/rho))+1
c     n2
      gamma = sigmap/lambda
      if (rsing) then
         Dp = 0
         do j = 1,ns
            Dj = D + min(2*mult(j)-2,2) + int(mult(j)*0.25d0)
            if (Dj.gt.Dp) Dp = Dtemp
         enddo
      else
         Dp = Dd
      endif
      y = 1d-3*v
      eta = (1.09d0-0.92d0*y+0.8d0*y**2)*min(1.78d0,
     $     1.236d0+0.0064d0*1.78d0**Dp)
      n2 = int(eta*nu*(2.3d0*Dp+omega)/(3d0+4d0*gamma+exp(-gamma)))+1
      n = max(n0,max(n1,n2))
c      write (6,*) 'n n0 n1 n2',n,n0,n1,n2

c     sum over paraboloid
      psi = tau*nu*pi/n
      u = 2d0*cos(psi)
      do j = 1,Nd
         b2(j) = 0d0
         b1(j) = 0d0
      enddo
      do k = n-1,1,-1
         theta = k*pi/n
         alpha = theta/tan(theta)
         snu = dcmplx(alpha,nu*theta)
         s = lambda*snu+sigma
         beta = theta+alpha*(alpha-1d0)/theta
         dummy = fun(s,Fc)
         do j = 1,Nd
            ak(j) = exp(alpha*tau)*dcmplx(nu,beta)*Fc(j)
            bk(j) = ak(j) + u*b1(j) - b2(j)
```

```
            b2(j) = b1(j)
            b1(j) = bk(j)
         enddo
         dummy = fun(conjg(s),Fc)
         do j = 1,Nd
            am(j) = exp(alpha*tau)*dcmplx(nu,-beta)*Fc(j)
            bm(j) = am(j) + u*bm1(j) - bm2(j)
            bm2(j) = bm1(j)
            bm1(j) = bm(j)
         enddo
      enddo
      s = lambda+sigma
      dummy = fun(s,Fc)
      do j = 1,Nd
         ak(j) = exp(tau)*nu*Fc(j)
         res(j) = lambda*exp(sigma*t)/n*0.5d0*
     $        ( 0.5d0*(ak(j) + am(j) + u*(b1(j)+bm1(j))) - b2(j) -bm2(j)
     $        + dcmplx(0d0,1d0)*b1(j)*sin(psi)
     $        - dcmplx(0d0,1d0)*bm1(j)*sin(psi) )
      enddo
 1000 format (f30.16,1x,f30.16)
      return
      end
```

A.2.3 p154.m

```
% p153.m BIM method SGLS 05/20/08
function p153
global h N x0 y0 nx ny

a = 1.3; b = 3;
ie = input('-1 for exterior, 1 for interior: ');
m = input('resolution : ')
s = 3;
xc = 1; yc = -0.5;  xc = 0; yc = 0;
tt = linspace(0,1,m)*2*pi; tt = fliplr(tt); x = a*cos(tt')+xc; y = b*sin(tt')+yc;
tt = linspace(0,1,m)*2*pi; x = a*cos(tt')+xc; y = b*sin(tt')+yc;
[f,fp,A,B,P,S,zb,vm,fi,fpi,xfi,xfpi,I] = bem(x,y,ie,s);
P
S
zb
```

```
vm
I
figure(1)
subplot(1,2,1)
splot(x,y); hold on; quiver(x0,y0,nx,ny,0.25); hold off
th = atan2((y0-yc)/b,(x0-xc)/a);
if (ie==1)
  fex = [x0-xc y0-yc (a^2-b^2)/(a^2+b^2)*(x0-xc).*(y0-yc)];
else
  fex = -[b*cos(th) a*sin(th) 0.25*(a^2-b^2)*sin(2*th)];
end
s = cumsum(h);
for j = 1:3
  subplot(3,2,2*j)
  plot(s,f(:,j),s,fex(:,j))
end
[k,e] = ellipke(1-min(a,b)^2/max(a,b)^2); Pex = 4*max(a,b)*e
Sex = pi*a*b
if (ie==1)
  vmex = [pi*a*b 0 0 ; 0 pi*a*b 0 ; 0 0 0.25*(a^2-b^2)^2/(a^2+b^2)*pi*a*b]
else
  vmex = [pi*b^2 0 0 ; 0 pi*a^2 0 ; 0 0 0.125*pi*(a^2-b^2)^2]
end
Iex = vm(:,1:2) + Sex*[1 0 ; 0 1 ; -yc xc]


function [f,fp,A,B,P,S,zb,vm,fi,fpi,xfi,xfpi,I] = bem(x,y,ie,s)

global N x0 y0 nx ny

glinit
pspline(x,y,ie)
[A,B] = buildmat(x,y,ie,s);

I = eye(N);
fp = [nx ny -nx.*y0 + ny.*x0];
r = rank(B-0.5*I);
if (r<N)
  f = pinv(B-0.5*I)*A*fp;
else
  f = (B-0.5*I)\A*fp;
end

nx = -nx; ny = -ny; fp = -fp;
g1 = [ones(N,1) nx ny]; g2 = [ones(N,1) x0 y0];
```

```
vm = zeros(3,3); fi = vm; fpi = vm; xfi = vm; xfpi = vm;
for i = 1:3
  for j = 1:3
    vm(i,j) = doint(f(:,i).*fp(:,j));
    fi(i,j) = doint(f(:,i).*g1(:,j));
    fpi(i,j) = doint(fp(:,i).*g1(:,j));
    xfi(i,j) = doint(f(:,i).*g2(:,j));
    xfpi(i,j) = doint(fp(:,i).*g2(:,j));
  end
end
I = fi(:,2:3)-xfpi(:,2:3);

P = doint(ones(N,1));
S = ie*[doint(x0.*nx) doint(y0.*ny)];
zb = ie*[doint(0.5*x0.^2.*nx)/S(1) doint(0.5*y0.^2.*ny)/S(2)];


function glinit

global xi w xil wl

xi = zeros(20,1);
w = zeros(20,1);
xi(1) = -0.993128599185094924786;
xi(2) = -0.963971927277913791268;
xi(3) = -0.912234428251325905868;
xi(4) = -0.839116971822218823395;
xi(5) = -0.746331906460150792614;
xi(6) = -0.636053680726515025453;
xi(7) = -0.510867001950827098004;
xi(8) = -0.373706088715419560673;
xi(9) = -0.227785851141645078080;
xi(10)= -0.076526521133497333755;
xi(11) = -xi(10);
xi(12) = -xi(9);
xi(13) = -xi(8);
xi(14) = -xi(7);
xi(15) = -xi(6);
xi(16) = -xi(5);
xi(17) = -xi(4);
xi(18) = -xi(3);
xi(19) = -xi(2);
xi(20) = -xi(1);
w(1) = 0.017614007139152118312;
w(2) = 0.040601429800386941331;
```

```
w(3) = 0.062672048334109063570;
w(4) = 0.083276741576704748725;
w(5) = 0.10193011981724043503 7;
w(6) = 0.11819453196151841731 2;
w(7) = 0.13168863844917662689 8;
w(8) = 0.14209610931838205132 9;
w(9) = 0.14917298647260374678 8;
w(10)= 0.15275338713072585069 8;
w(11) = w(10);
w(12) = w(9);
w(13) = w(8);
w(14) = w(7);
w(15) = w(6);
w(16) = w(5);
w(17) = w(4);
w(18) = w(3);
w(19) = w(2);
w(20) = w(1);


for N = 20
  j = (0:2*N);
  alpha = 0.5*ones(1,2*N-1);
  warning off
  beta = 0.25./(4-1./j.^2);
  nu = (-1).^j./j./(j+1).*exp(2*gammaln(j+1)-gammaln(2*j+1));
  warning on
  nu(1) = 1;
  sig = zeros(2*N+1,2*N+1);
  sig(2,2:2*N+1) = nu(1:2*N);
  a = zeros(N,1); b = zeros(N,1);
  a(1) = alpha(1) + nu(2)/nu(1);
  b(1) = 0;
  for k = 3:N+1
    for l = k:2*N-k+3
      sig(k,l) = sig(k-1,l+1) + (alpha(l-1)-a(k-2))*sig(k-1,l) - ...
          b(k-2)*sig(k-2,l) + beta(l-1)*sig(k-1,l-1);
    end
    a(k-1) = alpha(k-1) + sig(k,k+1)/sig(k,k) - sig(k-1,k)/sig(k-1, ...
                                                    k-1);
    b(k-1) = sig(k,k)/sig(k-1,k-1);
  end
  J = diag(a) + diag(sqrt(b(2:end)),1) + diag(sqrt(b(2:end)),-1);
  [v,d] = eig(J);
  xil = diag(d);
```

```
  wl = v(1,:).^2';
end

function pspline(x,y,ie)

global x0 y0 nx ny a b c h N

N = length(x)-1;
x = x(:); y = y(:);
dx = diff(x); dy = diff(y);
h = sqrt(dx.^2+dy.^2);
z = [x y];
a = zeros(N,2); b = zeros(N+1,2); c = zeros(N,2);
A = diag(2/3*(h(1:N-1)+h(2:N))) + diag(h(2:N-1)/3,-1) + diag(h(2:N-1)/3,1);
for j = 1:2
  xG = z(:,j);
  b1 = 0;
  rhs = diff(xG(2:N+1))./h(2:N) - diff(xG(1:N))./h(1:N-1);
  bb = A\rhs;
  F0 = 2/3*(h(1)+h(N))*b1 + 1/3*h(1)*bb(1) + 1/3*h(N)*bb(N-1) - ...
       (xG(2)-xG(1))/h(1) + (xG(N+1)-xG(N))/h(N);
  bb = A\(rhs - 1/3*([h(1) ; zeros(N-3,1) ; h(N)]));
  b1 = 1;
  F1 = 2/3*(h(1)+h(N))*b1 + 1/3*h(1)*bb(1) + 1/3*h(N)*bb(N-1) - ...
       (xG(2)-xG(1))/h(1) + (xG(N+1)-xG(N))/h(N);
  b1 = -F0/(F1-F0);
  bb = A\(rhs - b1/3*([h(1) ; zeros(N-3,1) ; h(N)]));
  b(:,j) = [b1 ; bb ; b1];
  a(1:N,j) = diff(b(:,j))/3./h;
  c(1:N,j) = diff(xG)./h - 1/3*h.*(b(2:N+1,j) + 2*b(1:N,j));
end
b = b(1:N,:);

hm = 0.5*h;
x0 = a(:,1).*hm.^3 + b(:,1).*hm.^2 + c(:,1).*hm + x(1:N);
y0 = a(:,2).*hm.^3 + b(:,2).*hm.^2 + c(:,2).*hm + y(1:N);
hn = sqrt( (3*a(:,1).*hm.^2 + 2*b(:,1).*hm + c(:,1)).^2 + ...
           (3*a(:,2).*hm.^2 + 2*b(:,2).*hm + c(:,2)).^2 );
nx = -1./hn.*(3*a(:,2).*hm.^2 + 2*b(:,2).*hm + c(:,2))*ie;
ny =  1./hn.*(3*a(:,1).*hm.^2 + 2*b(:,1).*hm + c(:,1))*ie;

function [A,B] = buildmat(x,y,ie,s)

global xi w xil wl x0 y0 a b c h N
```

```
A = zeros(N); B = zeros(N);
for i = 1:N
  hh = 0.5*(xi+1).*h(i);
  hi = sqrt( (3*a(i,1)*hh.^2 + 2*b(i,1)*hh + c(i,1)).^2 + ...
             (3*a(i,2)*hh.^2 + 2*b(i,2)*hh + c(i,2)).^2 );
  nx = -1./hi.*(3*a(i,2)*hh.^2 + 2*b(i,2)*hh + c(i,2))*ie;
  ny =  1./hi.*(3*a(i,1)*hh.^2 + 2*b(i,1)*hh + c(i,1))*ie;
  xk = a(i,1).*hh.^3 + b(i,1).*hh.^2 + c(i,1).*hh + x(i);
  yk = a(i,2).*hh.^3 + b(i,2).*hh.^2 + c(i,2).*hh + y(i);
  for j = 1:N
    [G,Gx,Gy] = Gfn(xk,yk,x0(j),y0(j));
    B(j,i) = 0.5*h(i)*sum(w.*hi.*(Gx.*nx+Gy.*ny));
    if (i~=j)
      A(j,i) = 0.5*h(i)*sum(w.*hi.*G);
    else
      if (s==1)
        As(1) = 0.5*h(i)*sum(w.*hi.*G);
      elseif (s==2)
        h0 = 0.5*h(i);
        Gr = G + log(abs(hh-h0))/(2*pi);
        hi0 = sqrt( (3*a(i,1)*h0.^2 + 2*b(i,1)*h0 + c(i,1)).^2 + ...
                    (3*a(i,2)*h0.^2 + 2*b(i,2)*h0 + c(i,2)).^2 );
        As(2) = 0.5*h(i)*sum(w.*hi.*Gr) - 1/(2*pi)*( ...
            0.5*h(i)*sum(w.*log(abs(hh-h0)).*(hi-hi0)) + ...
            hi0*( (h(i)-h0)*(log(h(i)-h0) - 1) + h0*(log(h0) -1) ...
                  ) );
        elseif (s==3)
          h0 = 0.5*h(i);
          Gr = G + log(abs(hh-h0))/(2*pi);
          As(3) = 0.5*h(i)*sum(w.*hi.*Gr);
          dh = 0.5*h(i);
          hm = h0 - 0.5*(xi+1)*0.5*h(i); hl = h0 - xil*0.5*h(i);
          him = sqrt( (3*a(i,1)*hm.^2 + 2*b(i,1)*hm + c(i,1)).^2 + ...
                      (3*a(i,2)*hm.^2 + 2*b(i,2)*hm + c(i,2)).^2 );
          hil = sqrt( (3*a(i,1)*hl.^2 + 2*b(i,1)*hl + c(i,1)).^2 + ...
                      (3*a(i,2)*hl.^2 + 2*b(i,2)*hl + c(i,2)).^2 );
          As(3) = As(3) + (dh*sum(wl.*hil) - 0.5*dh*log(dh)*sum(w.*him))/(2*pi);
          hm = h0 + 0.5*(xi+1)*0.5*h(i); hl = h0 + xil*0.5*h(i);
          him = sqrt( (3*a(i,1)*hm.^2 + 2*b(i,1)*hm + c(i,1)).^2 + ...
                      (3*a(i,2)*hm.^2 + 2*b(i,2)*hm + c(i,2)).^2 );
          hil = sqrt( (3*a(i,1)*hl.^2 + 2*b(i,1)*hl + c(i,1)).^2 + ...
                      (3*a(i,2)*hl.^2 + 2*b(i,2)*hl + c(i,2)).^2 );
          As(3) = As(3) +  (dh*sum(wl.*hil) - 0.5*dh*log(dh)*sum(w.*him))/(2*pi);
      end
%      As
```

```
      A(j,i) = As(s);
    end
  end
end

function [G,Gx,Gy] = Gfn(x,y,x0,y0)

r2 = (x-x0).^2 + (y-y0).^2;
G = -1/(4*pi)*log(r2);
Gx = -1/(2*pi)*(x-x0)./r2;
Gy = -1/(2*pi)*(y-y0)./r2;

function I = doint(g)

global xi w a b c h N

I = 0;
for i = 1:N
  hh = 0.5*(xi+1).*h(i);
  hi = sqrt( (3*a(i,1)*hh.^2 + 2*b(i,1)*hh + c(i,1)).^2 ...
          + (3*a(i,2)*hh.^2 + 2*b(i,2)*hh + c(i,2)).^2 );
  I = I + 0.5*h(i)*sum(w.*hi)*g(i);
end

function splot(x,y)

global a b c h N x0 y0

plot(x,y,x,y,'.')
hold on
for j = 1:N
  hh = h(j)*(0:0.01:1);
  xs = a(j,1).*hh.^3 + b(j,1).*hh.^2 + c(j,1).*hh + x(j);
  ys = a(j,2).*hh.^3 + b(j,2).*hh.^2 + c(j,2).*hh + y(j);
  plot(xs,ys,'r')
end
plot(x0,y0,'go')
hold off
axis equal
```

A.2.4 genHyper.m

```
function [pfq]=genHyper(a,b,z,lnpfq,ix,nsigfig);
% function [pfq]=genHyper(a,b,z,lnpfq,ix,nsigfig)
% Description : A numerical evaluator for the generalized hypergeometric
%               function for complex arguments with large magnitudes
%               using a direct summation of the Gauss series.
%               pFq isdefined by (borrowed from Maple):
%   pFq = sum(z^k / k! * product(pochhammer(n[i], k), i=1..p) /
%         product(pochhammer(d[j], k), j=1..q), k=0..infinity )
%
% INPUTS:       a => array containing numerator parameters
%               b => array containing denominator parameters
%               z => complex argument (scalar)
%           lnpfq => (optional) set to 1 if desired result is the natural
%                    log of pfq (default is 0)
%              ix => (optional) maximum number of terms in a,b (see below)
%         nsigfig => number of desired significant figures (default=10)
%
% OUPUT:      pfq => result
%
% EXAMPLES:     a=[1+i,1]; b=[2-i,3,3]; z=1.5;
%               >> genHyper(a,b,z)
%               ans =
%                         1.02992154295955 +     0.106416425916656i
%               or with more precision,
%               >> genHyper(a,b,z,0,0,15)
%               ans =
%                         1.02992154295896 +     0.106416425915575i
%               using the log option,
%               >> genHyper(a,b,z,1,0,15)
%               ans =
%                        0.0347923403326305 +     0.102959427435454i
%               >> exp(ans)
%               ans =
%                         1.02992154295896 +     0.106416425915575i
%
%
% Translated from the original fortran using f2matlab.m
%  by Ben E. Barrowes - barrowes@alum.mit.edu, 7/04.
%

%% Original fortran documentation
%     ACPAPFQ.  A NUMERICAL EVALUATOR FOR THE GENERALIZED HYPERGEOMETRIC
%
```

```
%     1  SERIES.  W.F. PERGER, A. BHALLA, M. NARDIN.
%
%     REF. IN COMP. PHYS. COMMUN. 77 (1993) 249
%
%     ****************************************************************
%     *                                                              *
%     *    SOLUTION TO THE GENERALIZED HYPERGEOMETRIC FUNCTION       *
%     *                                                              *
%     *                           by                                 *
%     *                                                              *
%     *                      W. F. PERGER,                           *
%     *                                                              *
%     *               MARK NARDIN  and ATUL BHALLA                   *
%     *                                                              *
%     *                                                              *
%     *            Electrical Engineering Department                 *
%     *            Michigan Technological University                 *
%     *                  1400 Townsend Drive                         *
%     *                Houghton, MI  49931-1295   USA                *
%     *                     Copyright 1993                           *
%     *                                                              *
%     *               e-mail address: wfp@mtu.edu                    *
%     *                                                              *
%     *  Description : A numerical evaluator for the generalized     *
%     *    hypergeometric function for complex arguments with large  *
%     *    magnitudes using a direct summation of the Gauss series.  *
%     *    The method used allows an accuracy of up to thirteen      *
%     *    decimal places through the use of large integer arrays    *
%     *    and a single final division.                              *
%     *    (original subroutines for the confluent hypergeometric    *
%     *    written by Mark Nardin, 1989; modifications made to cal-  *
%     *    culate the generalized hypergeometric function were       *
%     *    written by W.F. Perger and A. Bhalla, June, 1990)         *
%     *                                                              *
%     *  The evaluation of the pFq series is accomplished by a func- *
%     *  ion call to PFQ, which is a double precision complex func-  *
%     *  tion.  The required input is:                               *
%     *  1. Double precision complex arrays A and B.  These are the  *
%     *     arrays containing the parameters in the numerator and de-*
%     *     nominator, respectively.                                 *
%     *  2. Integers IP and IQ.  These integers indicate the number  *
%     *     of numerator and denominator terms, respectively (these  *
%     *     are p and q in the pFq function).                        *
%     *  3. Double precision complex argument Z.                     *
%     *  4. Integer LNPFQ.  This integer should be set to '1' if the *
```

```
%     *     result from PFQ is to be returned as the natural logaritm*
%     *     of the series, or '0' if not.  The user can generally set*
%     *     LNPFQ = '0' and change it if required.                    *
%     *  5. Integer IX.  This integer should be set to '0' if the     *
%     *     user desires the program PFQ to estimate the number of    *
%     *     array terms (in A and B) to be used, or an integer        *
%     *     greater than zero specifying the number of integer pos-   *
%     *     itions to be used.  This input parameter is escpecially   *
%     *     useful as a means to check the results of a given run.    *
%     *     Specificially, if the user obtains a result for a given   *
%     *     set of parameters, then changes IX and re-runs the eval-  *
%     *     uator, and if the number of array positions was insuffi-  *
%     *     cient, then the two results will likely differ.  The rec-*
%     *     commended would be to generally set IX = '0' and then set*
%     *     it to 100 or so for a second run.  Note that the LENGTH   *
%     *     parameter currently sets the upper limit on IX to 777,    *
%     *     but that can easily be changed (it is a single PARAMETER  *
%     *     statement) and the program recompiled.                    *
%     *  6. Integer NSIGFIG.  This integer specifies the requested    *
%     *     number of significant figures in the final result.  If    *
%     *     the user attempts to request more than the number of bits*
%     *     in the mantissa allows, the program will abort with an    *
%     *     appropriate error message.  The recommended value is 10.  *
%     *                                                               *
%     *     Note: The variable NOUT is the file to which error mess-  *
%     *           ages are written (default is 6).  This can be       *
%     *           changed in the FUNCTION PFQ to accomodate re-       *
%     *           of output to another file                           *
%     *                                                               *
%     *  Subprograms called: HYPER.                                   *
%     *                                                               *
%     *****************************************************************
%
%
%
%

if nargin<6
 nsigfig=10;
elseif isempty(nsigfig)
 nsigfig=10;
end
if nargin<5
 ix=0;
elseif isempty(ix)
```

```
 ix=0;
end
if nargin<4
 lnpfq=0;
elseif isempty(lnpfq)
 lnpfq=0;
end
ip=length(a);
iq=length(b);

global  zero   half   one   two   ten   eps;
[zero , half , one , two , ten , eps]=deal(0.0d0,0.5d0,1.0d0,2.0d0,10.0d0,1.0d-10);
global  nout;
%
%
%
%
a1=zeros(2,1);b1=zeros(1,1);gam1=0;gam2=0;gam3=0;gam4=0;gam5=0;gam6=0;gam7=0;hyper1=0;h
argi=0;argr=0;diff=0;dnum=0;precis=0;
%
%
i=0;
%
%
%
nout=6;
if ((lnpfq~=0) & (lnpfq~=1)) ;
 ' error in input arguments: lnpfq ~= 0 or 1',
 error('stop encountered in original fortran code');
end;
if ((ip>iq) & (abs(z)>one)) ;
 ip , iq , abs(z),
 %format [,1x,'ip=',1i2,3x,'iq=',1i2,3x,'and abs(z)=',1e12.5,2x,./,' which is greater t
 error('stop encountered in original fortran code');
end;
if (ip==2 & iq==1 & abs(z)>0.9) ;
 if (lnpfq~=1) ;
  %
  %      Check to see if the Gamma function arguments are o.k.; if not,
  %
  %      then the series will have to be used.
  %
  %
  %
  %      PRECIS - MACHINE PRECISION
```

```
%
%
precis=one;
precis=precis./two;
dnum=precis+one;
while (dnum>one);
 precis=precis./two;
 dnum=precis+one;
end;
precis=two.*precis;
for i=1 : 6;
 if (i==1) ;
  argi=imag(b(1));
  argr=real(b(1));
 elseif (i==2);
  argi=imag(b(1)-a(1)-a(2));
  argr=real(b(1)-a(1)-a(2));
 elseif (i==3);
  argi=imag(b(1)-a(1));
  argr=real(b(1)-a(1));
 elseif (i==4);
  argi=imag(a(1)+a(2)-b(1));
  argr=real(a(1)+a(2)-b(1));
 elseif (i==5);
  argi=imag(a(1));
  argr=real(a(1));
 elseif (i==6);
  argi=imag(a(2));
  argr=real(a(2));
 end;
 %
 %       CASES WHERE THE ARGUMENT IS REAL
 %
 %
 if (argi==0.0) ;
  %
  %        CASES WHERE THE ARGUMENT IS REAL AND NEGATIVE
  %
  %
  if (argr<=0.0) ;
   %
   %         USE THE SERIES EXPANSION IF THE ARGUMENT IS TOO NEAR A POLE
   %
   %
   diff=abs(real(round(argr))-argr);
```

```
     if (diff<=two.*precis) ;
      pfq=hyper(a,b,ip,iq,z,lnpfq,ix,nsigfig);
      return;
     end;
    end;
   end;
  end;
  gam1=cgamma(b(1),lnpfq);
  gam2=cgamma(b(1)-a(1)-a(2),lnpfq);
  gam3=cgamma(b(1)-a(1),lnpfq);
  gam4=cgamma(b(1)-a(2),lnpfq);
  gam5=cgamma(a(1)+a(2)-b(1),lnpfq);
  gam6=cgamma(a(1),lnpfq);
  gam7=cgamma(a(2),lnpfq);
  a1(1)=a(1);
  a1(2)=a(2);
  b1(1)=a(1)+a(2)-b(1)+one;
  z1=one-z;
  hyper1=hyper(a1,b1,ip,iq,z1,lnpfq,ix,nsigfig);
  a1(1)=b(1)-a(1);
  a1(2)=b(1)-a(2);
  b1(1)=b(1)-a(1)-a(2)+one;
  hyper2=hyper(a1,b1,ip,iq,z1,lnpfq,ix,nsigfig);
  pfq=gam1.*gam2.*hyper1./(gam3.*gam4)+(one-z).^(b(1)-a(1)-a(2)).*gam1.*gam5.*hyper2./(
  return;
 end;
end;
pfq=hyper(a,b,ip,iq,z,lnpfq,ix,nsigfig);
return;


%     ****************************************************************
%     *                                                              *
%     *                   FUNCTION BITS                              *
%     *                                                              *
%     *                                                              *
%     *  Description : Determines the number of significant figures  *
%     *    of machine precision to arrive at the size of the array   *
%     *    the numbers must be stored in to get the accuracy of the  *
%     *    solution.                                                 *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
```

```
function [bits]=bits;
%
bit2=0;
%
%
%
bit=1.0;
nnz=0;
nnz=nnz+1;
bit2=bit.*2.0;
bit=bit2+1.0;
while ((bit-bit2)~=0.0);
 nnz=nnz+1;
 bit2=bit.*2.0;
 bit=bit2+1.0;
end;
bits=nnz-3;


%     ****************************************************************
%     *                                                              *
%     *                    FUNCTION HYPER                            *
%     *                                                              *
%     *                                                              *
%     *  Description : Function that sums the Gauss series.          *
%     *                                                              *
%     *  Subprograms called: ARMULT, ARYDIV, BITS, CMPADD, CMPMUL,   *
%     *                      IPREMAX.                                *
%     *                                                              *
%     ****************************************************************
%
function [hyper]=hyper(a,b,ip,iq,z,lnpfq,ix,nsigfig);
%
%
% PARAMETER definitions
%
sumr=[];sumi=[];denomr=[];denomi=[];final=[];l=[];rmax=[];ibit=[];temp=[];cr=[];i1=[];c
length=0;
length=777;
%
%
global  zero   half   one   two   ten   eps;
global  nout;
%
%
```

```
%
%
accy=0;ai=zeros(10,1);ai2=zeros(10,1);ar=zeros(10,1);ar2=zeros(10,1);ci=zeros(10,1);ci2=
%
cdum1=0;cdum2=0;final=0;oldtemp=0;temp=0;temp1=0;
%
%
%
i=0;i1=0;ibit=0;icount=0;ii10=0;ir10=0;ixcnt=0;l=0;lmax=0;nmach=0;rexp=0;
%
%
%
goon1=0;
foo1=zeros(length+2,1);foo2=zeros(length+2,1);bar1=zeros(length+2,1);bar2=zeros(length+
%
%
zero=0.0d0;
log2=log10(two);
ibit=fix(bits);
rmax=two.^(fix(ibit./2));
sigfig=two.^(fix(ibit./4));
%
for i1=1 : ip;
 ar2(i1)=real(a(i1)).*sigfig;
 ar(i1)=fix(ar2(i1));
 ar2(i1)=round((ar2(i1)-ar(i1)).*rmax);
 ai2(i1)=imag(a(i1)).*sigfig;
 ai(i1)=fix(ai2(i1));
 ai2(i1)=round((ai2(i1)-ai(i1)).*rmax);
end;
for i1=1 : iq;
 cr2(i1)=real(b(i1)).*sigfig;
 cr(i1)=fix(cr2(i1));
 cr2(i1)=round((cr2(i1)-cr(i1)).*rmax);
 ci2(i1)=imag(b(i1)).*sigfig;
 ci(i1)=fix(ci2(i1));
 ci2(i1)=round((ci2(i1)-ci(i1)).*rmax);
end;
xr2=real(z).*sigfig;
xr=fix(xr2);
xr2=round((xr2-xr).*rmax);
xi2=imag(z).*sigfig;
xi=fix(xi2);
xi2=round((xi2-xi).*rmax);
%
```

```
%     WARN THE USER THAT THE INPUT VALUE WAS SO CLOSE TO ZERO THAT IT
%     WAS SET EQUAL TO ZERO.
%
for i1=1 : ip;
 if ((real(a(i1))~=0.0) & (ar(i1)==0.0) & (ar2(i1)==0.0));
  i1,
 end;
 %format (1x,'warning - real part of a(',1i2,') was set to zero');
 if ((imag(a(i1))~=0.0) & (ai(i1)==0.0) & (ai2(i1)==0.0));
  i1,
 end;
 %format (1x,'warning - imag part of a(',1i2,') was set to zero');
end;
for i1=1 : iq;
 if ((real(b(i1))~=0.0) & (cr(i1)==0.0) & (cr2(i1)==0.0));
  i1,
 end;
 %format (1x,'warning - real part of b(',1i2,') was set to zero');
 if ((imag(b(i1))~=0.0) & (ci(i1)==0.0) & (ci2(i1)==0.0));
  i1,
 end;
 %format (1x,'warning - imag part of b(',1i2,') was set to zero');
end;
if ((real(z)~=0.0) & (xr==0.0) & (xr2==0.0)) ;
 ' warning - real part of z was set to zero',
 z=complex(0.0,imag(z));
end;
if ((imag(z)~=0.0) & (xi==0.0) & (xi2==0.0)) ;
 ' warning - imag part of z was set to zero',
 z=complex(real(z),0.0);
end;
%
%
%     SCREENING OF NUMERATOR ARGUMENTS FOR NEGATIVE INTEGERS OR ZERO.
%     ICOUNT WILL FORCE THE SERIES TO TERMINATE CORRECTLY.
%
nmach=fix(log10(two.^fix(bits)));
icount=-1;
for i1=1 : ip;
 if ((ar2(i1)==0.0) & (ar(i1)==0.0) & (ai2(i1)==0.0) &(ai(i1)==0.0)) ;
  hyper=complex(one,0.0);
  return;
 end;
 if ((ai(i1)==0.0) & (ai2(i1)==0.0) & (real(a(i1))<0.0));
  if (abs(real(a(i1))-real(round(real(a(i1)))))<ten.^(-nmach)) ;
```

```
   if (icount~=-1) ;
    icount=min([icount,-round(real(a(i1)))]);
   else;
    icount=-round(real(a(i1)));
   end;
  end;
 end;
end;
%
%     SCREENING OF DENOMINATOR ARGUMENTS FOR ZEROES OR NEGATIVE INTEGERS
%     .
%
for i1=1 : iq;
 if ((cr(i1)==0.0) & (cr2(i1)==0.0) & (ci(i1)==0.0) &(ci2(i1)==0.0)) ;
  i1,
  %format (1x,'error - argument b(',1i2,') was equal to zero');
  error('stop encountered in original fortran code');
 end;
 if ((ci(i1)==0.0) & (ci2(i1)==0.0) & (real(b(i1))<0.0));
  if ((abs(real(b(i1))-real(round(real(b(i1)))))<ten.^(-nmach)) &(icount>=-round(real(b
   i1,
   %format (1x,'error - argument b(',1i2,') was a negative',' integer');
   error('stop encountered in original fortran code');
  end;
 end;
end;
%
nmach=fix(log10(two.^ibit));
nsigfig=min([nsigfig,fix(log10(two.^ibit))]);
accy=ten.^(-nsigfig);
l=ipremax(a,b,ip,iq,z);
if (l~=1) ;
 %
 %     First, estimate the exponent of the maximum term in the pFq series
 %     .
 %
 expon=0.0;
 xl=real(l);
 for i=1 : ip;
  expon=expon+real(factor(a(i)+xl-one))-real(factor(a(i)-one));
 end;
 for i=1 : iq;
  expon=expon-real(factor(b(i)+xl-one))+real(factor(b(i)-one));
 end;
 expon=expon+xl.*real(log(z))-real(factor(complex(xl,0.0)));
```

```
 lmax=fix(log10(exp(one)).*expon);
 l=lmax;
 %
 %     Now, estimate the exponent of where the pFq series will terminate.
 %
 temp1=complex(one,0.0);
 creal=one;
 for i1=1 : ip;
  temp1=temp1.*complex(ar(i1),ai(i1))./sigfig;
 end;
 for i1=1 : iq;
  temp1=temp1./(complex(cr(i1),ci(i1))./sigfig);
  creal=creal.*cr(i1);
 end;
 temp1=temp1.*complex(xr,xi);
 %
 %     Triple it to make sure.
 %
 l=3.*l;
 %
 %     Divide the number of significant figures necessary by the number
 %     of
 %     digits available per array position.
 %
 %
 l=fix((2.*l+nsigfig)./nmach)+2;
end;
%
%     Make sure there are at least 5 array positions used.
%
l=max([l,5]);
l=max([l,ix]);
%     write (6,*) ' Estimated value of L=',L
if ((l<0) | (l>length)) ;
 length,
 %format (1x,['error in fn hyper: l must be < '],1i4);
 error('stop encountered in original fortran code');
end;
if (nsigfig>nmach) ;
 nmach,
 %format (1x,' warning--the number of significant figures requ','ested',./,'is greater
end;
%
sumr(-1+2)=one;
sumi(-1+2)=one;
```

```
numr(-1+2)=one;
numi(-1+2)=one;
denomr(-1+2)=one;
denomi(-1+2)=one;
for i=0 : l+1;
 sumr(i+2)=0.0;
 sumi(i+2)=0.0;
 numr(i+2)=0.0;
 numi(i+2)=0.0;
 denomr(i+2)=0.0;
 denomi(i+2)=0.0;
end;
sumr(1+2)=one;
numr(1+2)=one;
denomr(1+2)=one;
cnt=sigfig;
temp=complex(0.0,0.0);
oldtemp=temp;
ixcnt=0;
rexp=fix(ibit./2);
x=rexp.*(sumr(l+1+2)-2);
rr10=x.*log2;
ir10=fix(rr10);
rr10=rr10-ir10;
x=rexp.*(sumi(l+1+2)-2);
ri10=x.*log2;
ii10=fix(ri10);
ri10=ri10-ii10;
dum1=(abs(sumr(1+2).*rmax.*rmax+sumr(2+2).*rmax+sumr(3+2)).*sign(sumr(-1+2)));
dum2=(abs(sumi(1+2).*rmax.*rmax+sumi(2+2).*rmax+sumi(3+2)).*sign(sumi(-1+2)));
dum1=dum1.*10.^rr10;
dum2=dum2.*10.^ri10;
cdum1=complex(dum1,dum2);
x=rexp.*(denomr(l+1+2)-2);
rr10=x.*log2;
ir10=fix(rr10);
rr10=rr10-ir10;
x=rexp.*(denomi(l+1+2)-2);
ri10=x.*log2;
ii10=fix(ri10);
ri10=ri10-ii10;
dum1=(abs(denomr(1+2).*rmax.*rmax+denomr(2+2).*rmax+denomr(3+2)).*sign(denomr(-1+2)));
dum2=(abs(denomi(1+2).*rmax.*rmax+denomi(2+2).*rmax+denomi(3+2)).*sign(denomi(-1+2)));
dum1=dum1.*10.^rr10;
dum2=dum2.*10.^ri10;
```

```
cdum2=complex(dum1,dum2);
temp=cdum1./cdum2;
%
%     130 IF (IP .GT. 0) THEN
goon1=1;
while (goon1==1);
 goon1=0;
 if (ip<0) ;
  if (sumr(1+2)<half) ;
   mx1=sumi(l+1+2);
  elseif (sumi(1+2)<half);
   mx1=sumr(l+1+2);
  else;
   mx1=max([sumr(l+1+2),sumi(l+1+2)]);
  end;
  if (numr(1+2)<half) ;
   mx2=numi(l+1+2);
  elseif (numi(1+2)<half);
   mx2=numr(l+1+2);
  else;
   mx2=max([numr(l+1+2),numi(l+1+2)]);
  end;
  if (mx1-mx2>2.0) ;
   if (creal>=0.0) ;
    %        write (6,*) ' cdabs(temp1/cnt)=',cdabs(temp1/cnt)
    %
    if (abs(temp1./cnt)<=one) ;
     [sumr,sumi,denomr,denomi,final,l,lnpfq,rmax,ibit]=arydiv(sumr,sumi,denomr,denomi,f
     hyper=final;
     return;
    end;
   end;
 end;
else;
 [sumr,sumi,denomr,denomi,temp,l,lnpfq,rmax,ibit]=arydiv(sumr,sumi,denomr,denomi,temp,
 %
 %      First, estimate the exponent of the maximum term in the pFq
 %      series.
 %
 expon=0.0;
 xl=real(ixcnt);
 for i=1 : ip;
  expon=expon+real(factor(a(i)+xl-one))-real(factor(a(i)-one));
 end;
 for i=1 : iq;
```

```
  expon=expon-real(factor(b(i)+xl-one))+real(factor(b(i)-one));
 end;
 expon=expon+xl.*real(log(z))-real(factor(complex(xl,0.0)));
 lmax=fix(log10(exp(one)).*expon);
 if (abs(oldtemp-temp)<abs(temp.*accy)) ;
  [sumr,sumi,denomr,denomi,final,l,lnpfq,rmax,ibit]=arydiv(sumr,sumi,denomr,denomi,fina
  hyper=final;
  return;
 end;
 oldtemp=temp;
end;
if (ixcnt~=icount) ;
 ixcnt=ixcnt+1;
 for i1=1 : iq;
  %
  %      TAKE THE CURRENT SUM AND MULTIPLY BY THE DENOMINATOR OF THE NEXT
  %
  %      TERM, FOR BOTH THE MOST SIGNIFICANT HALF (CR,CI) AND THE LEAST
  %
  %      SIGNIFICANT HALF (CR2,CI2).
  %
  %
  [sumr,sumi,cr(i1),ci(i1),qr1,qi1,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(sumr,sumi,cr
  [sumr,sumi,cr2(i1),ci2(i1),qr2,qi2,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(sumr,sumi,c
  qr2(l+1+2)=qr2(l+1+2)-1;
  qi2(l+1+2)=qi2(l+1+2)-1;
  %
  %      STORE THIS TEMPORARILY IN THE SUM ARRAYS.
  %
  %
  [qr1,qi1,qr2,qi2,sumr,sumi,wk1,l,rmax]=cmpadd(qr1,qi1,qr2,qi2,sumr,sumi,wk1,l,rmax);
 end;
 %
 %
 %     MULTIPLY BY THE FACTORIAL TERM.
 %
 foo1=sumr;
 foo2=sumr;
 [foo1,cnt,foo2,wk6,l,rmax]=armult(foo1,cnt,foo2,wk6,l,rmax);
 sumr=foo2;
 foo1=sumi;
 foo2=sumi;
 [foo1,cnt,foo2,wk6,l,rmax]=armult(foo1,cnt,foo2,wk6,l,rmax);
 sumi=foo2;
 %
```

```
%     MULTIPLY BY THE SCALING FACTOR, SIGFIG, TO KEEP THE SCALE CORRECT.
%
for i1=1 : ip-iq;
 foo1=sumr;
 foo2=sumr;
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 sumr=foo2;
 foo1=sumi;
 foo2=sumi;
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 sumi=foo2;
end;
for i1=1 : iq;
 %
 %      UPDATE THE DENOMINATOR.
 %
 %
 [denomr,denomi,cr(i1),ci(i1),qr1,qi1,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(denomr,d
 [denomr,denomi,cr2(i1),ci2(i1),qr2,qi2,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(denomr
 qr2(l+1+2)=qr2(l+1+2)-1;
 qi2(l+1+2)=qi2(l+1+2)-1;
 [qr1,qi1,qr2,qi2,denomr,denomi,wk1,l,rmax]=cmpadd(qr1,qi1,qr2,qi2,denomr,denomi,wk1,l
end;
%
%
%     MULTIPLY BY THE FACTORIAL TERM.
%
foo1=denomr;
foo2=denomr;
[foo1,cnt,foo2,wk6,l,rmax]=armult(foo1,cnt,foo2,wk6,l,rmax);
denomr=foo2;
foo1=denomi;
foo2=denomi;
[foo1,cnt,foo2,wk6,l,rmax]=armult(foo1,cnt,foo2,wk6,l,rmax);
denomi=foo2;
%
%     MULTIPLY BY THE SCALING FACTOR, SIGFIG, TO KEEP THE SCALE CORRECT.
%
for i1=1 : ip-iq;
 foo1=denomr;
 foo2=denomr;
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 denomr=foo2;
 foo1=denomi;
 foo2=denomi;
```

```
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 denomi=foo2;
end;
%
%     FORM THE NEXT NUMERATOR TERM BY MULTIPLYING THE CURRENT
%     NUMERATOR TERM (AN ARRAY) WITH THE A ARGUMENT (A SCALAR).
%
for i1=1 : ip;
 [numr,numi,ar(i1),ai(i1),qr1,qi1,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(numr,numi,ar
 [numr,numi,ar2(i1),ai2(i1),qr2,qi2,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(numr,numi,a
 qr2(l+1+2)=qr2(l+1+2)-1;
 qi2(l+1+2)=qi2(l+1+2)-1;
 [qr1,qi1,qr2,qi2,numr,numi,wk1,l,rmax]=cmpadd(qr1,qi1,qr2,qi2,numr,numi,wk1,l,rmax);
end;
%
%     FINISH THE NEW NUMERATOR TERM BY MULTIPLYING BY THE Z ARGUMENT.
%
[numr,numi,xr,xi,qr1,qi1,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(numr,numi,xr,xi,qr1,q
[numr,numi,xr2,xi2,qr2,qi2,wk1,wk2,wk3,wk4,wk5,wk6,l,rmax]=cmpmul(numr,numi,xr2,xi2,q
qr2(l+1+2)=qr2(l+1+2)-1;
qi2(l+1+2)=qi2(l+1+2)-1;
[qr1,qi1,qr2,qi2,numr,numi,wk1,l,rmax]=cmpadd(qr1,qi1,qr2,qi2,numr,numi,wk1,l,rmax);
%
%     MULTIPLY BY THE SCALING FACTOR, SIGFIG, TO KEEP THE SCALE CORRECT.
%
for i1=1 : iq-ip;
 foo1=numr;
 foo2=numr;
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 numr=foo2;
 foo1=numi;
 foo2=numi;
 [foo1,sigfig,foo2,wk6,l,rmax]=armult(foo1,sigfig,foo2,wk6,l,rmax);
 numi=foo2;
end;
%
%     FINALLY, ADD THE NEW NUMERATOR TERM WITH THE CURRENT RUNNING
%     SUM OF THE NUMERATOR AND STORE THE NEW RUNNING SUM IN SUMR, SUMI.
%
foo1=sumr;
foo2=sumr;
bar1=sumi;
bar2=sumi;
[foo1,bar1,numr,numi,foo2,bar2,wk1,l,rmax]=cmpadd(foo1,bar1,numr,numi,foo2,bar2,wk1,l
sumi=bar2;
```

```
  sumr=foo2;

  %
  %     BECAUSE SIGFIG REPRESENTS "ONE" ON THE NEW SCALE, ADD SIGFIG
  %     TO THE CURRENT COUNT AND, CONSEQUENTLY, TO THE IP ARGUMENTS
  %     IN THE NUMERATOR AND THE IQ ARGUMENTS IN THE DENOMINATOR.
  %
  cnt=cnt+sigfig;
  for i1=1 : ip;
   ar(i1)=ar(i1)+sigfig;
  end;
  for i1=1 : iq;
   cr(i1)=cr(i1)+sigfig;
  end;
  goon1=1;
 end;
end;
[sumr,sumi,denomr,denomi,final,l,lnpfq,rmax,ibit]=arydiv(sumr,sumi,denomr,denomi,final,l
%     write (6,*) 'Number of terms=',ixcnt
hyper=final;
return;


%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ARADD                            *
%     *                                                              *
%     *                                                              *
%     *  Description : Accepts two arrays of numbers and returns     *
%     *    the sum of the array.  Each array is holding the value    *
%     *    of one number in the series.  The parameter L is the      *
%     *    size of the array representing the number and RMAX is     *
%     *    the actual number of digits needed to give the numbers    *
%     *    the desired accuracy.                                     *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [a,b,c,z,l,rmax]=aradd(a,b,c,z,l,rmax);
%
%
global  zero   half   one   two   ten   eps;
%
```

```
%
%
ediff=0;i=0;j=0;
%
%
for i=0 : l+1;
 z(i+2)=0.0;
end;
ediff=round(a(l+1+2)-b(l+1+2));
if (abs(a(1+2))<half | ediff<=-l) ;
 for i=-1 : l+1;
  c(i+2)=b(i+2);
 end;
 if (c(1+2)<half) ;
  c(-1+2)=one;
  c(l+1+2)=0.0;
 end;
 return;
else;
 if (abs(b(1+2))<half | ediff>=l) ;
  for i=-1 : l+1;
   c(i+2)=a(i+2);
  end;
  if (c(1+2)<half) ;
   c(-1+2)=one;
   c(l+1+2)=0.0;
  end;
  return;
 else;
  z(-1+2)=a(-1+2);
  goon300=1;
  goon190=1;
  if (abs(a(-1+2)-b(-1+2))>=half) ;
   goon300=0;
   if (ediff>0) ;
    z(l+1+2)=a(l+1+2);
   elseif (ediff<0);
    z(l+1+2)=b(l+1+2);
    z(-1+2)=b(-1+2);
    goon190=0;
   else;
    for i=1 : l;
     if (a(i+2)>b(i+2)) ;
      z(l+1+2)=a(l+1+2);
      break;
```

```
    end;
    if (a(i+2)<b(i+2)) ;
     z(l+1+2)=b(l+1+2);
     z(-1+2)=b(-1+2);
     goon190=0;
    end;
   end;
  end;
  %
 elseif (ediff>0);
  z(l+1+2)=a(l+1+2);
  for i=l : -1: 1+ediff ;
   z(i+2)=a(i+2)+b(i-ediff+2)+z(i+2);
   if (z(i+2)>=rmax) ;
    z(i+2)=z(i+2)-rmax;
    z(i-1+2)=one;
   end;
  end;
  for i=ediff : -1: 1 ;
   z(i+2)=a(i+2)+z(i+2);
   if (z(i+2)>=rmax) ;
    z(i+2)=z(i+2)-rmax;
    z(i-1+2)=one;
   end;
  end;
  if (z(0+2)>half) ;
   for i=l : -1: 1 ;
    z(i+2)=z(i-1+2);
   end;
   z(l+1+2)=z(l+1+2)+1;
   z(0+2)=0.0;
  end;
 elseif (ediff<0);
  z(l+1+2)=b(l+1+2);
  for i=l : -1: 1-ediff ;
   z(i+2)=a(i+ediff+2)+b(i+2)+z(i+2);
   if (z(i+2)>=rmax) ;
    z(i+2)=z(i+2)-rmax;
    z(i-1+2)=one;
   end;
  end;
  for i=0-ediff : -1: 1 ;
   z(i+2)=b(i+2)+z(i+2);
   if (z(i+2)>=rmax) ;
    z(i+2)=z(i+2)-rmax;
```

```
    z(i-1+2)=one;
   end;
  end;
  if (z(0+2)>half) ;
   for i=l : -1: 1 ;
    z(i+2)=z(i-1+2);
   end;
   z(l+1+2)=z(l+1+2)+one;
   z(0+2)=0.0;
  end;
 else;
  z(l+1+2)=a(l+1+2);
  for i=l : -1: 1 ;
   z(i+2)=a(i+2)+b(i+2)+z(i+2);
   if (z(i+2)>=rmax) ;
    z(i+2)=z(i+2)-rmax;
    z(i-1+2)=one;
   end;
  end;
  if (z(0+2)>half) ;
   for i=l : -1: 1 ;
    z(i+2)=z(i-1+2);
   end;
   z(l+1+2)=z(l+1+2)+one;
   z(0+2)=0.0;
  end;
 end;
 if (goon300==1) ;
  i=i; %here is the line that had a +1 taken from it.
  while (z(i+2)<half & i<l+1);
   i=i+1;
  end;
  if (i==l+1) ;
   z(-1+2)=one;
   z(l+1+2)=0.0;
   for i=-1 : l+1;
    c(i+2)=z(i+2);
   end;
   if (c(1+2)<half) ;
    c(-1+2)=one;
    c(l+1+2)=0.0;
   end;
   return;
  end;
  for j=1 : l+1-i;
```

```
  z(j+2)=z(j+i-1+2);
 end;
 for j=l+2-i : l;
  z(j+2)=0.0;
 end;
 z(l+1+2)=z(l+1+2)-i+1;
 for i=-1 : l+1;
  c(i+2)=z(i+2);
 end;
 if (c(1+2)<half) ;
  c(-1+2)=one;
  c(l+1+2)=0.0;
 end;
 return;
end;
%
if (goon190==1) ;
 if (ediff>0) ;
  for i=l : -1: 1+ediff ;
   z(i+2)=a(i+2)-b(i-ediff+2)+z(i+2);
   if (z(i+2)<0.0) ;
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
  for i=ediff : -1: 1 ;
   z(i+2)=a(i+2)+z(i+2);
   if (z(i+2)<0.0) ;
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
 else;
  for i=l : -1: 1 ;
   z(i+2)=a(i+2)-b(i+2)+z(i+2);
   if (z(i+2)<0.0) ;
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
 end;
 if (z(1+2)>half) ;
  for i=-1 : l+1;
   c(i+2)=z(i+2);
  end;
```

```
   if (c(1+2)<half) ;
    c(-1+2)=one;
    c(l+1+2)=0.0;
   end;
   return;
  end;
  i=1;
  i=i+1;
  while (z(i+2)<half & i<l+1);
   i=i+1;
  end;
  if (i==l+1) ;
   z(-1+2)=one;
   z(l+1+2)=0.0;
   for i=-1 : l+1;
    c(i+2)=z(i+2);
   end;
   if (c(1+2)<half) ;
    c(-1+2)=one;
    c(l+1+2)=0.0;
   end;
   return;
  end;
  for j=1 : l+1-i;
   z(j+2)=z(j+i-1+2);
  end;
  for j=l+2-i : l;
   z(j+2)=0.0;
  end;
  z(l+1+2)=z(l+1+2)-i+1;
  for i=-1 : l+1;
   c(i+2)=z(i+2);
  end;
  if (c(1+2)<half) ;
   c(-1+2)=one;
   c(l+1+2)=0.0;
  end;
  return;
 end;
end;
%
if (ediff<0) ;
 for i=l : -1: 1-ediff ;
  z(i+2)=b(i+2)-a(i+ediff+2)+z(i+2);
  if (z(i+2)<0.0) ;
```

```
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
  for i=0-ediff : -1: 1 ;
   z(i+2)=b(i+2)+z(i+2);
   if (z(i+2)<0.0) ;
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
 else;
  for i=l : -1: 1 ;
   z(i+2)=b(i+2)-a(i+2)+z(i+2);
   if (z(i+2)<0.0) ;
    z(i+2)=z(i+2)+rmax;
    z(i-1+2)=-one;
   end;
  end;
 end;
end;
%
if (z(1+2)>half) ;
 for i=-1 : l+1;
  c(i+2)=z(i+2);
 end;
 if (c(1+2)<half) ;
  c(-1+2)=one;
  c(l+1+2)=0.0;
 end;
 return;
end;
i=1;
i=i+1;
while (z(i+2)<half & i<l+1);
 i=i+1;
end;
if (i==l+1) ;
 z(-1+2)=one;
 z(l+1+2)=0.0;
 for i=-1 : l+1;
  c(i+2)=z(i+2);
 end;
 if (c(1+2)<half) ;
  c(-1+2)=one;
```

```
  c(l+1+2)=0.0;
 end;
 return;
end;
for j=1 : l+1-i;
 z(j+2)=z(j+i-1+2);
end;
for j=l+2-i : l;
 z(j+2)=0.0;
end;
z(l+1+2)=z(l+1+2)-i+1;
for i=-1 : l+1;
 c(i+2)=z(i+2);
end;
if (c(1+2)<half) ;
 c(-1+2)=one;
 c(l+1+2)=0.0;
end;


%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ARSUB                            *
%     *                                                              *
%     *                                                              *
%     *  Description : Accepts two arrays and subtracts each element *
%     *    in the second array from the element in the first array   *
%     *    and returns the solution.  The parameters L and RMAX are  *
%     *    the size of the array and the number of digits needed for *
%     *    the accuracy, respectively.                               *
%     *                                                              *
%     *  Subprograms called: ARADD                                   *
%     *                                                              *
%     ****************************************************************
%
function [a,b,c,wk1,wk2,l,rmax]=arsub(a,b,c,wk1,wk2,l,rmax);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
i=0;
```

```
%
%
for i=-1 : l+1;
 wk2(i+2)=b(i+2);
end;
wk2(-1+2)=(-one).*wk2(-1+2);
[a,wk2,c,wk1,l,rmax]=aradd(a,wk2,c,wk1,l,rmax);


%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ARMULT                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Accepts two arrays and returns the product.   *
%     *    L and RMAX are the size of the arrays and the number of   *
%     *    digits needed to represent the numbers with the required  *
%     *    accuracy.                                                 *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [a,b,c,z,l,rmax]=armult(a,b,c,z,l,rmax);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
b2=0;carry=0;
i=0;
%
%
z(-1+2)=(abs(one).*sign(b)).*a(-1+2);
b2=abs(b);
z(l+1+2)=a(l+1+2);
for i=0 : l;
 z(i+2)=0.0;
end;
if (b2<=eps | a(1+2)<=eps) ;
 z(-1+2)=one;
 z(l+1+2)=0.0;
```

```
else;
 for i=l : -1: 1 ;
  z(i+2)=a(i+2).*b2+z(i+2);
  if (z(i+2)>=rmax) ;
   carry=fix(z(i+2)./rmax);
   z(i+2)=z(i+2)-carry.*rmax;
   z(i-1+2)=carry;
  end;
 end;
 if (z(0+2)>=half) ;
  for i=l : -1: 1 ;
   z(i+2)=z(i-1+2);
  end;
  z(l+1+2)=z(l+1+2)+one;
  if (z(1+2)>=rmax) ;
   for i=l : -1: 1 ;
    z(i+2)=z(i-1+2);
   end;
   carry=fix(z(1+2)./rmax);
   z(2+2)=z(2+2)-carry.*rmax;
   z(1+2)=carry;
   z(l+1+2)=z(l+1+2)+one;
  end;
  z(0+2)=0.0;
 end;
end;
for i=-1 : l+1;
 c(i+2)=z(i+2);
end;
if (c(1+2)<half) ;
 c(-1+2)=one;
 c(l+1+2)=0.0;
end;


%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE CMPADD                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Takes two arrays representing one real and    *
%     *    one imaginary part, and adds two arrays representing      *
%     *    another complex number and returns two array holding the  *
%     *    complex sum.                                              *
```

```
%     *              (CR,CI) = (AR+BR, AI+BI)                          *
%     *                                                                *
%     *  Subprograms called: ARADD                                     *
%     *                                                                *
%     ******************************************************************
%
function [ar,ai,br,bi,cr,ci,wk1,l,rmax]=cmpadd(ar,ai,br,bi,cr,ci,wk1,l,rmax);
%
%
%
%
%
%
[ar,br,cr,wk1,l,rmax]=aradd(ar,br,cr,wk1,l,rmax);
[ai,bi,ci,wk1,l,rmax]=aradd(ai,bi,ci,wk1,l,rmax);


%
%
%     ******************************************************************
%     *                                                                *
%     *                 SUBROUTINE CMPSUB                              *
%     *                                                                *
%     *                                                                *
%     *  Description : Takes two arrays representing one real and      *
%     *    one imaginary part, and subtracts two arrays representing   *
%     *    another complex number and returns two array holding the    *
%     *    complex sum.                                                *
%     *              (CR,CI) = (AR+BR, AI+BI)                          *
%     *                                                                *
%     *  Subprograms called: ARADD                                     *
%     *                                                                *
%     ******************************************************************
%
function [ar,ai,br,bi,cr,ci,wk1,wk2,l,rmax]=cmpsub(ar,ai,br,bi,cr,ci,wk1,wk2,l,rmax);
%
%
%
%
%
%
[ar,br,cr,wk1,wk2,l,rmax]=arsub(ar,br,cr,wk1,wk2,l,rmax);
[ai,bi,ci,wk1,wk2,l,rmax]=arsub(ai,bi,ci,wk1,wk2,l,rmax);
```

```
%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE CMPMUL                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Takes two arrays representing one real and    *
%     *    one imaginary part, and multiplies it with two arrays     *
%     *    representing another complex number and returns the       *
%     *    complex product.                                          *
%     *                                                              *
%     *  Subprograms called: ARMULT, ARSUB, ARADD                    *
%     *                                                              *
%     ****************************************************************
%
function [ar,ai,br,bi,cr,ci,wk1,wk2,cr2,d1,d2,wk6,l,rmax]=cmpmul(ar,ai,br,bi,cr,ci,wk1,w
%
%
%
%
i=0;
%
%
[ar,br,d1,wk6,l,rmax]=armult(ar,br,d1,wk6,l,rmax);
[ai,bi,d2,wk6,l,rmax]=armult(ai,bi,d2,wk6,l,rmax);
[d1,d2,cr2,wk1,wk2,l,rmax]=arsub(d1,d2,cr2,wk1,wk2,l,rmax);
[ar,bi,d1,wk6,l,rmax]=armult(ar,bi,d1,wk6,l,rmax);
[ai,br,d2,wk6,l,rmax]=armult(ai,br,d2,wk6,l,rmax);
[d1,d2,ci,wk1,l,rmax]=aradd(d1,d2,ci,wk1,l,rmax);
for i=-1 : l+1;
 cr(i+2)=cr2(i+2);
end;


%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ARYDIV                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Returns the double precision complex number   *
%     *    resulting from the division of four arrays, representing  *
%     *    two complex numbers.  The number returned will be in one  *
```

```
%     *    of two different forms:  either standard scientific or as *
%     *    the log (base 10) of the number.                          *
%     *                                                              *
%     *  Subprograms called: CONV21, CONV12, EADD, ECPDIV, EMULT.    *
%     *                                                              *
%     ****************************************************************
%
function [ar,ai,br,bi,c,l,lnpfq,rmax,ibit]=arydiv(ar,ai,br,bi,c,l,lnpfq,rmax,ibit);
%
%
cdum=[];ae=[];be=[];ce=[];n1=[];e1=[];n2=[];e2=[];n3=[];e3=[];
global  zero   half   one   two   ten   eps;
%
%
%
%
ae=zeros(2,2);be=zeros(2,2);ce=zeros(2,2);dum1=0;dum2=0;e1=0;e2=0;e3=0;n1=0;n2=0;n3=0;pl
cdum=0;
%
dnum=0;
ii10=0;ir10=0;itnmax=0;rexp=0;
%
%
%
rexp=fix(ibit./2);
x=rexp.*(ar(l+1+2)-2);
rr10=x.*log10(two)./log10(ten);
ir10=fix(rr10);
rr10=rr10-ir10;
x=rexp.*(ai(l+1+2)-2);
ri10=x.*log10(two)./log10(ten);
ii10=fix(ri10);
ri10=ri10-ii10;
dum1=(abs(ar(1+2).*rmax.*rmax+ar(2+2).*rmax+ar(3+2)).*sign(ar(-1+2)));
dum2=(abs(ai(1+2).*rmax.*rmax+ai(2+2).*rmax+ai(3+2)).*sign(ai(-1+2)));
dum1=dum1.*10.^rr10;
dum2=dum2.*10.^ri10;
cdum=complex(dum1,dum2);
[cdum,ae]=conv12(cdum,ae);
ae(1,2)=ae(1,2)+ir10;
ae(2,2)=ae(2,2)+ii10;
x=rexp.*(br(l+1+2)-2);
rr10=x.*log10(two)./log10(ten);
ir10=fix(rr10);
rr10=rr10-ir10;
```

```
x=rexp.*(bi(l+1+2)-2);
ri10=x.*log10(two)./log10(ten);
ii10=fix(ri10);
ri10=ri10-ii10;
dum1=(abs(br(1+2).*rmax.*rmax+br(2+2).*rmax+br(3+2)).*sign(br(-1+2)));
dum2=(abs(bi(1+2).*rmax.*rmax+bi(2+2).*rmax+bi(3+2)).*sign(bi(-1+2)));
dum1=dum1.*10.^rr10;
dum2=dum2.*10.^ri10;
cdum=complex(dum1,dum2);
[cdum,be]=conv12(cdum,be);
be(1,2)=be(1,2)+ir10;
be(2,2)=be(2,2)+ii10;
[ae,be,ce]=ecpdiv(ae,be,ce);
if (lnpfq==0) ;
 [ce,c]=conv21(ce,c);
else;
 [ce(1,1),ce(1,2),ce(1,1),ce(1,2),n1,e1]=emult(ce(1,1),ce(1,2),ce(1,1),ce(1,2),n1,e1);
 [ce(2,1),ce(2,2),ce(2,1),ce(2,2),n2,e2]=emult(ce(2,1),ce(2,2),ce(2,1),ce(2,2),n2,e2);
 [n1,e1,n2,e2,n3,e3]=eadd(n1,e1,n2,e2,n3,e3);
 n1=ce(1,1);
 e1=ce(1,2)-ce(2,2);
 x2=ce(2,1);
 %
 %      TENMAX - MAXIMUM SIZE OF EXPONENT OF 10
 %
 %      THE FOLLOWING CODE CAN BE USED TO DETERMINE TENMAX, BUT IT
 %
 %      WILL LIKELY GENERATE AN IEEE FLOATING POINT UNDERFLOW ERROR
 %
 %      ON A SUN WORKSTATION.  REPLACE TENMAX WITH THE VALUE APPROPRIATE
 %
 %      FOR YOUR MACHINE.
 %
 %
 tenmax=320;
 itnmax=1;
 dnum=0.1d0;
 itnmax=itnmax+1;
 dnum=dnum.*0.1d0;
 while (dnum>0.0);
  itnmax=itnmax+1;
  dnum=dnum.*0.1d0;
 end;
 itnmax=itnmax-1;
 tenmax=real(itnmax);
```

```
 %
 if (e1>tenmax) ;
  x1=tenmax;
 elseif (e1<-tenmax);
  x1=0.0;
 else;
  x1=n1.*(ten.^e1);
 end;
 if (x2~=0.0) ;
  phi=atan2(x2,x1);
 else;
  phi=0.0;
 end;
 c=complex(half.*(log(n3)+e3.*log(ten)),phi);
end;


%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE EMULT                            *
%     *                                                              *
%     *                                                              *
%     *  Description : Takes one base and exponent and multiplies it *
%     *    by another numbers base and exponent to give the product  *
%     *    in the form of base and exponent.                         *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [n1,e1,n2,e2,nf,ef]=emult(n1,e1,n2,e2,nf,ef);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
nf=n1.*n2;
ef=e1+e2;
if (abs(nf)>=ten) ;
 nf=nf./ten;
 ef=ef+one;
end;
```

```
%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE EDIV                             *
%     *                                                              *
%     *                                                              *
%     *  Description : returns the solution in the form of base and  *
%     *    exponent of the division of two exponential numbers.      *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [n1,e1,n2,e2,nf,ef]=ediv(n1,e1,n2,e2,nf,ef);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
nf=n1./n2;
ef=e1-e2;
if ((abs(nf)<one) & (nf~=zero)) ;
 nf=nf.*ten;
 ef=ef-one;
end;


%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE EADD                             *
%     *                                                              *
%     *                                                              *
%     *  Description : Returns the sum of two numbers in the form    *
%     *    of a base and an exponent.                                *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [n1,e1,n2,e2,nf,ef]=eadd(n1,e1,n2,e2,nf,ef);
```

```
%
%
global  zero   half   one   two   ten   eps;
%
ediff=0;
%
%
ediff=e1-e2;
if (ediff>36.0d0) ;
 nf=n1;
 ef=e1;
elseif (ediff<-36.0d0);
 nf=n2;
 ef=e2;
else;
 nf=n1.*(ten.^ediff)+n2;
 ef=e2;
 while (1);
  if (abs(nf)<ten) ;
   while ((abs(nf)<one) & (nf~=0.0));
    nf=nf.*ten;
    ef=ef-one;
   end;
   break;
  else;
   nf=nf./ten;
   ef=ef+one;
  end;
 end;
end;


%
%     ****************************************************************
%     *                                                              *
%     *                 SUBROUTINE ESUB                              *
%     *                                                              *
%     *                                                              *
%     *  Description : Returns the solution to the subtraction of    *
%     *    two numbers in the form of base and exponent.             *
%     *                                                              *
%     *  Subprograms called: EADD                                    *
%     *                                                              *
%     ****************************************************************
%
```

```
function [n1,e1,n2,e2,nf,ef]=esub(n1,e1,n2,e2,nf,ef);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
[n1,e1,dumvar3,e2,nf,ef]=eadd(n1,e1,n2.*(-one),e2,nf,ef);


%
%
%     ****************************************************************
%     *                                                              *
%     *                 SUBROUTINE CONV12                            *
%     *                                                              *
%     *                                                              *
%     *  Description : Converts a number from complex notation to a  *
%     *    form of a 2x2 real array.                                 *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [cn,cae]=conv12(cn,cae);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
%
%
cae(1,1)=real(cn);
cae(1,2)=0.0;
while (1);
 if (abs(cae(1,1))<ten) ;
  while (1);
   if ((abs(cae(1,1))>=one) | (cae(1,1)==0.0)) ;
    cae(2,1)=imag(cn);
    cae(2,2)=0.0;
    while (1);
     if (abs(cae(2,1))<ten) ;
      while ((abs(cae(2,1))<one) & (cae(2,1)~=0.0));
       cae(2,1)=cae(2,1).*ten;
```

```
       cae(2,2)=cae(2,2)-one;
      end;
      break;
     else;
      cae(2,1)=cae(2,1)./ten;
      cae(2,2)=cae(2,2)+one;
     end;
    end;
    break;
   else;
    cae(1,1)=cae(1,1).*ten;
    cae(1,2)=cae(1,2)-one;
   end;
  end;
  break;
 else;
  cae(1,1)=cae(1,1)./ten;
  cae(1,2)=cae(1,2)+one;
 end;
end;


%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE CONV21                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Converts a number represented in a 2x2 real   *
%     *    array to the form of a complex number.                    *
%     *                                                              *
%     *  Subprograms called: none                                    *
%     *                                                              *
%     ****************************************************************
%
function [cae,cn]=conv21(cae,cn);
%
%
%
global  zero   half   one   two   ten   eps;
global  nout;
%
%
%
dnum=0;tenmax=0;
```

```
itnmax=0;
%
%
%     TENMAX - MAXIMUM SIZE OF EXPONENT OF 10
%
itnmax=1;
dnum=0.1d0;
itnmax=itnmax+1;
dnum=dnum.*0.1d0;
while  (dnum>0.0);
 itnmax=itnmax+1;
 dnum=dnum.*0.1d0;
end;
itnmax=itnmax-2;
tenmax=real(itnmax);
%
if (cae(1,2)>tenmax | cae(2,2)>tenmax) ;
 %      CN=CMPLX(TENMAX,TENMAX)
 %
 itnmax,
 %format (' error - value of exponent required for summation',' was larger',./,' than th
 error('stop encountered in original fortran code');
elseif (cae(2,2)<-tenmax);
 cn=complex(cae(1,1).*(10.^cae(1,2)),0.0);
else;
 cn=complex(cae(1,1).*(10.^cae(1,2)),cae(2,1).*(10.^cae(2,2)));
end;
return;


%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ECPMUL                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Multiplies two numbers which are each         *
%     *    represented in the form of a two by two array and returns *
%     *    the solution in the same form.                            *
%     *                                                              *
%     *  Subprograms called: EMULT, ESUB, EADD                       *
%     *                                                              *
%     ****************************************************************
%
```

```
function [a,b,c]=ecpmul(a,b,c);
%
%
n1=[];e1=[];n2=[];e2=[];c2=[];
c2=zeros(2,2);e1=0;e2=0;n1=0;n2=0;
%
%
[a(1,1),a(1,2),b(1,1),b(1,2),n1,e1]=emult(a(1,1),a(1,2),b(1,1),b(1,2),n1,e1);
[a(2,1),a(2,2),b(2,1),b(2,2),n2,e2]=emult(a(2,1),a(2,2),b(2,1),b(2,2),n2,e2);
[n1,e1,n2,e2,c2(1,1),c2(1,2)]=esub(n1,e1,n2,e2,c2(1,1),c2(1,2));
[a(1,1),a(1,2),b(2,1),b(2,2),n1,e1]=emult(a(1,1),a(1,2),b(2,1),b(2,2),n1,e1);
[a(2,1),a(2,2),b(1,1),b(1,2),n2,e2]=emult(a(2,1),a(2,2),b(1,1),b(1,2),n2,e2);
[n1,e1,n2,e2,c(2,1),c(2,2)]=eadd(n1,e1,n2,e2,c(2,1),c(2,2));
c(1,1)=c2(1,1);
c(1,2)=c2(1,2);



%
%
%     ****************************************************************
%     *                                                              *
%     *                  SUBROUTINE ECPDIV                           *
%     *                                                              *
%     *                                                              *
%     *  Description : Divides two numbers and returns the solution. *
%     *    All numbers are represented by a 2x2 array.               *
%     *                                                              *
%     *  Subprograms called: EADD, ECPMUL, EDIV, EMULT               *
%     *                                                              *
%     ****************************************************************
%
function [a,b,c]=ecpdiv(a,b,c);
%
%
b2=[];c2=[];n1=[];e1=[];n2=[];e2=[];n3=[];e3=[];
global  zero   half   one   two   ten   eps;
%
b2=zeros(2,2);c2=zeros(2,2);e1=0;e2=0;e3=0;n1=0;n2=0;n3=0;
%
%
b2(1,1)=b(1,1);
b2(1,2)=b(1,2);
b2(2,1)=-one.*b(2,1);
b2(2,2)=b(2,2);
[a,b2,c2]=ecpmul(a,b2,c2);
```

```
[b(1,1),b(1,2),b(1,1),b(1,2),n1,e1]=emult(b(1,1),b(1,2),b(1,1),b(1,2),n1,e1);
[b(2,1),b(2,2),b(2,1),b(2,2),n2,e2]=emult(b(2,1),b(2,2),b(2,1),b(2,2),n2,e2);
[n1,e1,n2,e2,n3,e3]=eadd(n1,e1,n2,e2,n3,e3);
[c2(1,1),c2(1,2),n3,e3,c(1,1),c(1,2)]=ediv(c2(1,1),c2(1,2),n3,e3,c(1,1),c(1,2));
[c2(2,1),c2(2,2),n3,e3,c(2,1),c(2,2)]=ediv(c2(2,1),c2(2,2),n3,e3,c(2,1),c(2,2));


%     ****************************************************************
%     *                                                              *
%     *                    FUNCTION IPREMAX                          *
%     *                                                              *
%     *                                                              *
%     *  Description : Predicts the maximum term in the pFq series   *
%     *    via a simple scanning of arguments.                       *
%     *                                                              *
%     *  Subprograms called: none.                                   *
%     *                                                              *
%     ****************************************************************
%
function [ipremax]=ipremax(a,b,ip,iq,z);
%
%
%
global  zero   half   one   two   ten   eps;
global  nout;
%
%
%
%
%
expon=0;xl=0;xmax=0;xterm=0;
%
i=0;j=0;
%
xterm=0;
for j=1 : 100000;
 %
 %      Estimate the exponent of the maximum term in the pFq series.
 %
 %
 expon=zero;
 xl=real(j);
 for i=1 : ip;
  expon=expon+real(factor(a(i)+xl-one))-real(factor(a(i)-one));
 end;
```

```
 for i=1 : iq;
  expon=expon-real(factor(b(i)+xl-one))+real(factor(b(i)-one));
 end;
 expon=expon+xl.*real(log(z))-real(factor(complex(xl,zero)));
 xmax=log10(exp(one)).*expon;
 if ((xmax<xterm) & (j>2)) ;
  ipremax=j;
  return;
 end;
 xterm=max([xmax,xterm]);
end;
' error in ipremax--did not find maximum exponent',
error('stop encountered in original fortran code');


%     ***************************************************************
%     *                                                             *
%     *                   FUNCTION FACTOR                           *
%     *                                                             *
%     *                                                             *
%     *  Description : This function is the log of the factorial.   *
%     *                                                             *
%     *  Subprograms called: none.                                  *
%     *                                                             *
%     ***************************************************************
%
function [factor]=factor(z);
%
%
global  zero   half   one   two   ten   eps;
%
%
%
pi=0;
%
if (((real(z)==one) & (imag(z)==zero)) | (abs(z)==zero)) ;
 factor=complex(zero,zero);
 return;
end;
pi=two.*two.*atan(one);
factor=half.*log(two.*pi)+(z+half).*log(z)-z+(one./(12.0d0.*z)).*(one-(one./(30.d0.*z.*


%     ***************************************************************
%     *                                                             *
```

```
%     *                    FUNCTION CGAMMA                            *
%     *                                                               *
%     *                                                               *
%     *  Description : Calculates the complex gamma function.  Based *
%     *     on a program written by F.A. Parpia published in Computer*
%     *     Physics Communications as the 'GRASP2' program (public   *
%     *     domain).                                                  *
%     *                                                               *
%     *                                                               *
%     *  Subprograms called: none.                                    *
%     *                                                               *
%     *****************************************************************
function [cgamma]=cgamma(arg,lnpfq);
%
%
%
%
%
global  zero   half   one   two   ten   eps;
global  nout;
%
%
%
argi=0;argr=0;argui=0;argui2=0;argum=0;argur=0;argur2=0;clngi=0;clngr=0;diff=0;dnum=0;e
%
first=0;negarg=0;cgamma=0;
i=0;itnmax=0;
%
%
%
%----------------------------------------------------------------------*
%     *
%     THESE ARE THE BERNOULLI NUMBERS B02, B04, ..., B14, EXPRESSED AS *
%     RATIONAL NUMBERS. FROM ABRAMOWITZ AND STEGUN, P. 810.            *
%     *
fn=[1.0d00,-1.0d00,1.0d00,-1.0d00,5.0d00,-691.0d00,7.0d00];
fd=[6.0d00,30.0d00,42.0d00,30.0d00,66.0d00,2730.0d00,6.0d00];
%
%----------------------------------------------------------------------*
%
hlntpi=[1.0d00];
%
first=[true];
%
tenth=[0.1d00];
```

```
%
argr=real(arg);
argi=imag(arg);
%
%     ON THE FIRST ENTRY TO THIS ROUTINE, SET UP THE CONSTANTS REQUIRED
%     FOR THE REFLECTION FORMULA (CF. ABRAMOWITZ AND STEGUN 6.1.17) AND
%     STIRLING'S APPROXIMATION (CF. ABRAMOWITZ AND STEGUN 6.1.40).
%
if (first) ;
 pi=4.0d0.*atan(one);
 %
 %      SET THE MACHINE-DEPENDENT PARAMETERS:
 %
 %
 %      TENMAX - MAXIMUM SIZE OF EXPONENT OF 10
 %
 %
 itnmax=1;
 dnum=tenth;
 itnmax=itnmax+1;
 dnum=dnum.*tenth;
 while (dnum>0.0);
  itnmax=itnmax+1;
  dnum=dnum.*tenth;
 end;
 itnmax=itnmax-1;
 tenmax=real(itnmax);
 %
 %      EXPMAX - MAXIMUM SIZE OF EXPONENT OF E
 %
 %
 dnum=tenth.^itnmax;
 expmax=-log(dnum);
 %
 %      PRECIS - MACHINE PRECISION
 %
 %
 precis=one;
 precis=precis./two;
 dnum=precis+one;
 while (dnum>one);
  precis=precis./two;
  dnum=precis+one;
 end;
 precis=two.*precis;
```

```
 %
 hlntpi=half.*log(two.*pi);
 %
 for i=1 : 7;
  fn(i)=fn(i)./fd(i);
  twoi=two.*real(i);
  fn(i)=fn(i)./(twoi.*(twoi-one));
 end;
 %
 first=false;
 %
end;
%
%     CASES WHERE THE ARGUMENT IS REAL
%
if (argi==0.0) ;
 %
 %      CASES WHERE THE ARGUMENT IS REAL AND NEGATIVE
 %
 %
 if (argr<=0.0) ;
  %
  %       STOP WITH AN ERROR MESSAGE IF THE ARGUMENT IS TOO NEAR A POLE
  %
  %
  diff=abs(real(round(argr))-argr);
  if (diff<=two.*precis) ;
   ,
   argr , argi,
   %format (' argument (',1p,1d14.7,',',1d14.7,') too close to a',' pole.');
   error('stop encountered in original fortran code');
  else;
   %
   %        OTHERWISE USE THE REFLECTION FORMULA (ABRAMOWITZ AND STEGUN 6.1
   %        .17)
   %        TO ENSURE THAT THE ARGUMENT IS SUITABLE FOR STIRLING'S
   %
   %        FORMULA
   %
   %
   argum=pi./(-argr.*sin(pi.*argr));
   if (argum<0.0) ;
    argum=-argum;
    clngi=pi;
   else;
```

```
   clngi=0.0;
  end;
  facneg=log(argum);
  argur=-argr;
  negarg=true;
  %
 end;
 %
 %       CASES WHERE THE ARGUMENT IS REAL AND POSITIVE
 %
 %
else;
 %
 clngi=0.0;
 argur=argr;
 negarg=false;
 %
end;
%
%      USE ABRAMOWITZ AND STEGUN FORMULA 6.1.15 TO ENSURE THAT
%
%      THE ARGUMENT IN STIRLING'S FORMULA IS GREATER THAN 10
%
%
ovlfac=one;
while (argur<ten);
 ovlfac=ovlfac.*argur;
 argur=argur+one;
end;
%
%      NOW USE STIRLING'S FORMULA TO COMPUTE LOG (GAMMA (ARGUM))
%
%
clngr=(argur-half).*log(argur)-argur+hlntpi;
fac=argur;
obasq=one./(argur.*argur);
for i=1 : 7;
 fac=fac.*obasq;
 clngr=clngr+fn(i).*fac;
end;
%
%      INCLUDE THE CONTRIBUTIONS FROM THE RECURRENCE AND REFLECTION
%
%      FORMULAE
%
```

```
 %
 clngr=clngr-log(ovlfac);
 if (negarg) ;
  clngr=facneg-clngr;
 end;
 %
else;
 %
 %      CASES WHERE THE ARGUMENT IS COMPLEX
 %
 %
 argur=argr;
 argui=argi;
 argui2=argui.*argui;
 %
 %      USE THE RECURRENCE FORMULA (ABRAMOWITZ AND STEGUN 6.1.15)
 %
 %      TO ENSURE THAT THE MAGNITUDE OF THE ARGUMENT IN STIRLING'S
 %
 %      FORMULA IS GREATER THAN 10
 %
 %
 ovlfr=one;
 ovlfi=0.0;
 argum=sqrt(argur.*argur+argui2);
 while (argum<ten);
  termr=ovlfr.*argur-ovlfi.*argui;
  termi=ovlfr.*argui+ovlfi.*argur;
  ovlfr=termr;
  ovlfi=termi;
  argur=argur+one;
  argum=sqrt(argur.*argur+argui2);
 end;
 %
 %      NOW USE STIRLING'S FORMULA TO COMPUTE LOG (GAMMA (ARGUM))
 %
 %
 argur2=argur.*argur;
 termr=half.*log(argur2+argui2);
 termi=atan2(argui,argur);
 clngr=(argur-half).*termr-argui.*termi-argur+hlntpi;
 clngi=(argur-half).*termi+argui.*termr-argui;
 fac=(argur2+argui2).^(-2);
 obasqr=(argur2-argui2).*fac;
 obasqi=-two.*argur.*argui.*fac;
```

```
 zfacr=argur;
 zfaci=argui;
 for i=1 : 7;
  termr=zfacr.*obasqr-zfaci.*obasqi;
  termi=zfacr.*obasqi+zfaci.*obasqr;
  fac=fn(i);
  clngr=clngr+termr.*fac;
  clngi=clngi+termi.*fac;
  zfacr=termr;
  zfaci=termi;
 end;
 %
 %      ADD IN THE RELEVANT PIECES FROM THE RECURRENCE FORMULA
 %
 %
 clngr=clngr-half.*log(ovlfr.*ovlfr+ovlfi.*ovlfi);
 clngi=clngi-atan2(ovlfi,ovlfr);
 %
end;
if (lnpfq==1) ;
 cgamma=complex(clngr,clngi);
 return;
end;
%
%     NOW EXPONENTIATE THE COMPLEX LOG GAMMA FUNCTION TO GET
%     THE COMPLEX GAMMA FUNCTION
%
if ((clngr<=expmax) & (clngr>=-expmax)) ;
 fac=exp(clngr);
else;
 ,
 clngr,
 %format (' argument to exponential function (',1p,1d14.7,') out of range.');
 error('stop encountered in original fortran code');
end;
resr=fac.*cos(clngi);
resi=fac.*sin(clngi);
cgamma=complex(resr,resi);
%
return;
```