TEN DIGIT ALGORITHMS

Lloyd N. Trefethen
Oxford University Computing Laboratory

1. INTRODUCTION

For twenty-five years I have been developing numerical algorithms, teaching numerical analysis courses, and helping people solve numerical problems at Stanford, NYU, MIT, Cornell, and Oxford. This proposal is an outgrowth of that experience. Of course, any ideas related to computing must germinate in a changing world. During my quarter-century we have seen the arrival of workstations and laptops, of ubiquitous graphics and visualization, of the World Wide Web, and of MATLAB and Maple and Mathematica; also the decline of numerical software libraries and an astonishing increase in the speed of everything we do. [1]

Over the years perhaps a thousand people have come to me for numerical advice. On half of these occasions, inevitably, I have little to contribute. As for the other half, however, it is remarkable how often the customer and I end up sitting down at the computer together and getting good numbers at surprisingly high speed. That experience motivates this essay. So does my experience in teaching. Since arriving at Oxford in 1997, I have adopted the habit of including an online demonstration in each lecture. I hand out hardcopies of the code for the students to look at as we go, and it is a matter of pride that the listing always fits on one page. Of course, the program is also made available electronically, so they can download it later as a template for their own work.

A view has come into focus for me of a kind of computing which I summarize by the notion of a "ten digit algorithm". I believe that ten digit algorithms can be useful for education, for communication, and for research. Many of the programs I've written in recent years are in this mode, and if you spend your time computing with numbers, I urge you too to make ten digit algorithms a part of your operating practice.

This paper was presented as the A. R. Mitchell Lecture at the 2005 Dundee Biennial Conference on Numerical Analysis, 27 June 2005

2. THE DEFINITION

Ten digits,
Five seconds,
And just one page.

A ten digit algorithm is a little gem of a program to compute something numerical. The jingle summarizes the three defining conditions. The program can be at most one page long, and it has to solve your problem to at least ten digits of accuracy on your machine in less than five seconds.

For me, the programs are usually in MATLAB, but this is not part of the definition. Maple and Mathematica, for example, are impressive alternatives.

Strictly speaking, the term "ten digit program" might be more correct. I chose "algorithm" because it is more interesting and serves as a reminder that although all aspects of a code are important, the preeminent one is the underlying numerical method.

Of course, the three conditions are artificial, at least in their precise formulation if not in their general idea. But I do not think this must count against them. After all, the net in tennis is artificial, as are the days of the week and the convention that a sonnet has 14 lines. Nevertheless, such constraints can guide our activities in fruitful ways.

3. WHY FIVE SECONDS?

In computing with humans, response time is everything. If a program runs in less than five seconds, its author will effortlessly adjust, improve, and experiment. The process of scientific exploration becomes interactive and pleasurable. If the program runs for a minute or an hour, on the other hand, it is a very different situation from the human point of view. Of course, for some problems that's unavoidable, but one's likelihood of getting the science right falls quickly as one loses the ability to steer the computation on a human time scale. I've seen this over and over again. Visitors to my office who can't run their programs fast often get numbers they aren't sure of, and numbers you aren't sure of are often wrong.

Many people believe that only "toy" problems can be solved in five seconds. Indeed, this is the more or less universal objection I encounter when I first tell people about ten digit algorithms. But I think it is mistaken. Of course, there are many large-scale computations that will eventually have to be carried out in "production mode", but the fact is, that is just a small part of the story of how numerical programmers can and should spend their time. First of all, many

computational problems can be solved without any of that kind of computing. Secondly, even if production mode will be needed for the final results, it does not follow that the programmer doing the development should spend most of his or her development hours in this mode. I've seen vast amounts of time wasted in that fashion.

We can put it this way. Maybe 90% or even 99% of numerical machine cycles are destined to be spent in big production runs. From the programmer's point of view, nevertheless, 90% of the jobs that get run should be quick ones.

Five seconds is not an absolute criterion; its significance changes as machines advance. This is intentional. Computing is a human activity, and the point is to keep humans in the loop.

4. WHY ONE PAGE?

Again it's a matter of linking to people. A code less than a page long can be read and studied. I am convinced that for a good fraction of problems, studying the code is feasible and valuable. And of course, a code that you study is one that you polish and perfect. Print out a copy, step away from your computer, and grab a red pen! The moment the listing spills onto page 2, the chance that you will look at it carefully cuts in half.

The matter of communication is overwhelmingly important. It is often said that there are three kinds of science: theory, experiment, and computation. Analogously, there are four kinds of scientific communication: words, figures, mathematics, and programs. If you rely on the first three alone, you are handicapped. For communicating algorithmic ideas, nothing can touch a short program in its precision and conciseness. If you are like me, you have wasted exasperating hours studying journal articles, trying to figure out exactly how the author's algorithm works and what parameter choices he or she prefers, when a short code would have answered these questions immediately. Code segments appear in rather few journal articles these days, and this is a trend that I hope we can persuade publishers to reverse.

Like "five seconds", "one page" is not an absolute criterion; its significance varies with the programming language. The development of high-level languages like MATLAB has enabled us to do more than was possible with Fortran or C. Thanks to these high-level languages, there is not much need any more to communicate with the old tools of flow charts and pseudocodes. We can use the executable language directly.

5. WHY TEN DIGITS?

The reasoning here is different. It's a matter of linking not to humans, but to the physical world.

To oversimplify a bit, there are two kinds of accuracy. Engineering accuracy means two or three digits. This is generally ample for the end user like the designer of a bridge or an airplane or a medical trial or an evaluation scheme for pricing financial options. Getting more than two or three digits for end-user applications will typically be impossible, because the problem is too complex, and meaningless, since it is not defined precisely enough.

Scientific accuracy, by contrast, means five or ten digits. This is what one aims for in solving fundamental problems upon which other ideas or computations will be built. For example, the flow of air over a Boeing 767 is an engineering application, but the flow through an infinite idealized circular pipe is a scientific application. Computing the natural frequencies of a symphony orchestra kettledrum is an engineering problem, but computing the eigenvalues of an L-shaped region is scientific. One's expectations can and should be different.

In physics, almost nothing is known to more than 10 digits. We know the gravitational constant to about 5 digits, Planck's constant to about 7, the speed of light to 8 or 9. These precisions are not increasing very fast. In twenty years our computers will be a million times more powerful, but physics will still not know much to more than 10 digits.

Ten digits is very different from three in the type of thinking and algorithms it forces upon you. To get ten digits of accuracy in five seconds of computing, you have to have really "nailed" your problem, and in particular, to have worked around any singularities it may have. It's a challenging and exciting type of computing.

6. CHALLENGE AND ENJOYMENT

This brings me to a key feature of ten digit algorithms. Ten digits, five seconds, one page — this is a challenge! By struggling to meet the challenge, we sharpen and improve our ideas and our codes. This is the best possible route to understanding a problem, and besides, it's *fun*. Computing is one of the highest creative activities that humans indulge in, and it should be fun. How else to induce people to engage their imaginations at the highest level?

7. MUST A TEN DIGIT ALGORITHM BE EASILY READABLE?

The one-page restriction is designed to encourage gems of programming, and one would hope that a ten digit algorithm will be as clear and easy to read as possible. Certainly one hopes for attractive structure and for useful, well laid-out comments. Nevertheless, readability is not one of the defining conditions of a ten digit algorithm, for in the end, power is more important. Think of the analogy of poetry. Poems are gems of verbal expression. Some

can be read and appreciated instantly, and that is a wonderful thing. Nevertheless, nobody would say that immediate accessibility is a requirement of a good poem. On the contrary, what matters is that it repay repeated reading and reflection.

8. SCRIPTS AND SUBROUTINES

Ultimately your program may be turned into a software tool or a production code. We are concerned here not with that stage, but with the earlier phases of development and exploration.

There's a simple principle that makes a big difference to your productivity, and it surprises me how many people don't use it. The principle (expressed here in MATLAB terminology) is, when developing a numerical code, always drive your computation through a script. Give it a short name like p49.m or stokes.m and keep a window on your screen with this file open for editing. To run it, you just type "p49" or "stokes". Everything you need, such as plotting commands, will be in this program, and any detail can be quickly changed without the need for retyping the rest. Don't work at the command line!

Of course your code will often be a ten digit algorithm, so you can see most of it at once on the screen and run it over and over again quickly, improving it at every step. And you can test and test and vary and plot and test and test again. When things go wrong, you have immediate access to all the variables in your workspace.

Functions and subroutines are powerful and important. But these are for operations you've finished debugging, black boxes, not for development work.

9. ACADEMIC NUMERICAL ANALYSIS

Years ago, numerical analysts were the main people trained to solve numerical problems on computers. Indeed, there was a time when anyone at Oxford who wanted to use the computer had to get permission from Leslie Fox, the Professor of Numerical Analysis! Since the 1960s, however, computation has spread to all scientists and engineers, and the field of numerical analysis has grown mature and academic. Most papers published in this field concentrate on advancing some particular algorithm for some particular problem, often by proving a new theorem about convergence or accuracy. These are genuine advances, a necessary part of our continuing progress into the future, but there is no denying that such papers are of interest mainly to specialists. Now in part, this specialization of our publications is an inevitable consequence of the maturing of a discipline, but something uninevitable has also happened along the way: many numerical analysts have also grown specialized in their interests

and abilities. If you have a seemingly ordinary enough problem to solve and you ask a numerical analyst for help, too often you'll go away disappointed.

Imagine if the medical establishment had only specialists, no general practitioners! I don't recommend the creation of a new class of GP numerical analysts, but it would be a good thing for our field if each of us developed our "GP side" a little further. If ten digit algorithms become a familiar signpost, they will encourage that kind of breadth.

10. PROOFS AND "ENGINEERING PROOFS"

Roughly speaking, numerical analysts like to establish correctness of a computation by a theorem, whereas practitioners like to do it by numerical tests such as varying parameters to confirm that the computed answer doesn't change. There is no doubt that ten digit algorithms have a natural link to this second kind of correctness test, thanks to the five-second condition. When a program runs fast, you will inevitably run it in various modes and with various parameter choices. It would be a sloppy person indeed who did not examine the performance of a ten digit algorithm from so many angles as to acquire good confidence in its correctness. By contrast, even a careful programmer will find it daunting to test a program that runs for hours.

My view is that all people doing numerical computations, including numerical analysts, should routinely perform this kind of testing. In other words, though there is a place for the theorist who doesn't compute, there is no place for a computing person who doesn't check his or her results. This is a sine qua non. In addition, since numerical analysis is a branch of mathematics, we can also have recourse to theorems, and the best algorithms will be developed in the light of both practice and theory.

Most ten digit algorithms will contain parameters that have been tuned. One might decide to use a grid spacing of 1/150 since 1/100 is not good enough, or 60 terms in a series expansion because 50 will not do. The testing that leads to such parameter choices will have been done by the programmer in the course of developing the program, and that is fine. By the time you show anybody your t.d.a., you will have run it a hundred times in various forms. Thus there is no requirement that the final ten digit algorithm must explore dependence on parameters or demonstrate that ten digits have been achieved, though in many cases, where time and space permit, you may choose to include such features.

11. NUMERICAL ANALYSIS AND APPLIED MATHEMATICS

One of the core methodologies of applied mathematics is asymptotic analysis, a tool that springs from the observation that in many scientific problems, the important phenomena are dominated by a nearby singularity. By

understanding the singular behaviour, one may solve the problem to sufficient accuracy, and even more important, one may understand the scientific essence of the matter.

Much of numerical analysis has taken more or less the opposite line: singularities and asymptotics are fine in their place, but it is our business to design general tools that do not depend on special features of the problem at hand.

I think ten digit algorithms can help to build a bridge between these two outstandingly successful traditions. Very often, a certain kind of method proves effective for ten digit algorithms. This is: eliminate the singularities, then do something relatively simple for the rest of the problem, e.g. involving a least-squares fit with some free expansion parameters. For conformally mapping a polygon, for example, you don't always need the Schwarz-Christoffel formula; sometimes you can "roll your own" elementary solution that's equally effective (see **`trapmap`**). Other t.d.a.'s in this booklet that fit this pattern include **`two_disks`**, **`many_disks`**, and **`jaisson`**.

12. IN CONCLUSION

Ten digit algorithms can

- Improve our publications
- Speed up program development
- Make our numerical methods faster
- Make our scientific results more reliable
- Facilitate comparisons of ideas and results
- Add focus to the classroom
- Add zest to our field

The challenge of designing these codes raises our standards and raises our expectations. It's good for the academics, and it opens the doors wider to non-academics. And it's fun!

The attached examples, 32 M-files from various areas I have been interested in, range from little demonstrators of routine ideas to advanced programs that can compete with any others in their particular domains. They can be downloaded from my web page at Oxford. I can't imagine teaching students about Gauss quadrature any more, say, without sharing with them a code like **`cheb_vs_gauss`**. At the more advanced level, I hope you will agree with me that some of the other programs in this collection solve some very serious computational problems. If ten digit algorithms are this powerful today, what will they be capable of tomorrow?

EXAMPLE TEN DIGIT ALGORITHMS

SPECIAL NUMBERS AND FUNCTIONS
pi_via_agm
richardson
bernoulli
gamma_talbot
euler_const
gamma_cmv

EXPANSIONS AND APPROXIMATIONS
taylor_coeffs
cheb_coeffs
barycentric
pade
real_cf_approx

INTEGRALS
integral1D
integral2D
cheb_vs_gauss

LINEAR ALGEBRA
pseudospectra
cuberootA
operator_norm

NONLINEAR DYNAMICS
planets

LAPLACE PROBLEMS
two_disks
many_disks
rayleigh_disks
jaisson

EIGENMODES OF DRUMS
circular_drum
Ldrum
isodrum

CONFORMAL MAPPING
trapmap
schwarz_christoffel

SPECTRAL METHODS
mathieu
orr_sommerfeld
OScritical

NONLINEAR PDE
kuramoto_siv
kdv

+ TWO FUNCTIONS CALLED BY
SEVERAL OF THESE CODES:
cheb.m
gauss.m

```
% pi_via_agm.m  Compute pi by arithmetic-geometric mean iteration
%               L. N. Trefethen 7/03
%
% The AGM iteration for computing elliptic integrals goes back to
% Gauss.  In 1976 Brent (J. ACM) and independently Salamin (Math.
% Comp.) published AGM algorithms converging quadratically to pi.
% This code implements a variant due to Borwein and Borwein (SIAM
% Review 1984) given as Algorithm 2.1 in their 1987 book "Pi and
% the AGM".  The results are more dramatic in extended precision!

  y = sqrt(sqrt(2));
  x = (y+1/y)/2;
  p = 2+sqrt(2);
  for i = 1:6
    fprintf('%21.16g\n',p)
    p = p*(1+x)/(1+y);
    s = sqrt(x);
    y = (y*s+1/s)/(1+y);
    x = (s+1/s)/2;
  end
```

```
% richardson.m  Richardson extrapolation for Archimedes' polygons
%               L. N. Trefethen 2/05
%
% This little code applies Richardson extrapolation to Archimedes'
% classic problem of approximating pi via perimeters of regular
% polygons inscribed in a circle.

  format long, format loose
  k = 2.^(1:5)';                                % geometrically varying k
  a = k.*sin(pi./k);                            % data
  N = length(a);                                % number of data points
  A = NaN*ones(N); A(:,1) = a;                  % put data in matrix A
  for j = 1:N-1
    r = 4^(-j); f = (r/(1-r));                  % extrapolation factors
    a = a(2:end) + f*(a(2:end)-a(1:end-1));     % extrapolate
    A(1:N-j,j+1) = a;                           % store results in A
  end
  A
  error = pi-A
```

```
% bernoulli.m  Use FFT to compute Bernoulli numbers
%              L. N. Trefethen 2/05
%
% The nth Bernoulli number is n! times the nth Taylor coefficient of
% z/(exp(z)-1).  This code, a variant of taylor_coeffs.m, computes
% B_0,...,B_20 by the FFT.

  f = inline('z./(exp(z)-1)');           % generating function
  N = 128;                               % no. of sample points
  r = 4;                                 % radius of sample circle
  zz = r*exp(2i*pi*(0:N-1)/N);           % sample points
  c = real(fft(f(zz)))/N;                % Taylor coeffs. via FFT
  c = c.*gamma(1:N)./r.^(0:N-1);         % rescale

  exact = [1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66 0 -691/2730 ...
           0 7/6 0 -3617/510 0 43867/798 0 -174611/330];
  disp('  n         computed           exact         error')
  for n = 1:length(exact)
    fprintf('%3d %17.10f %17.10f %11.1e\n',...
            n-1,c(n),exact(n),c(n)-exact(n)')
  end
```

```
% gamma_talbot.m  Compute complex gamma function via Talbot contour
%                 L. N. Trefethen 6/05
%
% Hankel's integral formula for Gamma(z) is
%
% 1/Gamma(z) = (2*pi*i)^(-1) int_C e^t t^{-z} dt
%
% where the contour C passes from -infty-0i around 0 to -infty+0i.
% We evaluate the integral by the trapezoid rule transplanted to a
% cotangent contour due to Talbot (1979) with parameters optimized
% by Weideman (2005).  Adapted from p31.m of Trefethen, "Spectral
% Methods in MATLAB".

  N = 40;                                          % no. of quadrature pts
  th = (-N/2+.5:N/2-.5)*pi/(N/2);                  % trapezoid pts in [-pi,pi]
  a = -.2407; b = .2387; c = .7409; d = .1349i;    % Weideman's parameters
  z = N*(a + b*th.*cot(c*th) + d*th);              % Talbot pts in z-plane
  zp = b*cot(c*th) - b*c*th./sin(c*th).^2 + d;     % N^(-1) times derivative
  x = -3.5:.1:4; y = -2.5:.1:2.5;
  [xx,yy] = meshgrid(x,y);
  zz = xx + 1i*yy;                                 % plotting grid
  gaminv = zeros(size(zz));
  Npts = length(th)                                % no. of pts on contour
  for k = 1:Npts                                   % Loop goes over contour
    t = z(k);                                      %   points, not grid pts
    gaminv = gaminv + exp(t)*t.^(-zz)*zp(k);       %   (which are vectorized)
  end
  gam = 1i./gaminv;                                % Gamma(z) on the grid
  figure, mesh(xx,yy,abs(gam))                     % plot |Gamma(z)|
  axis([-3.5 4 -2.5 2.5 0 6])
  xlabel Re(z), ylabel Im(z)
  text(4,-1.4,5.5,'|\Gamma(z)|','fontsize',20)
  colormap([0 0 0])                                % make the image black
  format long, disp(gam(26,46:10:76).')            % approxs. to Gamma(1:4)
```

```
% euler_const.m  Compute Euler's constant by contour integral
%                Nick Trefethen 6/05
%
% It is known that euler = -Gamma'(1), where Gamma is the gamma
% function.  By Hankel's contour integral for Gamma this implies
%
% euler = (2*pi*i)^(-1) int_C e^t log(t)/t dt
%
% where the contour C passes from -infty-0i around 0 to -infty+0i.
% This code, adapted from gamma_talbot.m, computes the integral to
% 15-digit precision by adding up values from 15 sample points
% along a Talbot/Weideman contour.

  format long
  N = 30;                                          % twice the no. of pts
  th = pi*(1:2:N-1)/N;                             % angles in [0,pi]
  a = -.2407; b = .2387; c = .7409; d = .1349i;  % Weideman's parameters
  z = N*(a + b*th.*cot(c*th) + d*th);              % Talbot pts in z-plane
  zp = b*cot(c*th) - b*c*th./sin(c*th).^2 + d;   % N^(-1) times derivative
  euler = sum(zp.*exp(z).*log(z)./z);              % add values at sample pts
  euler = -2*imag(euler)                           % exploit symmetry
  exact = 0.57721566490153286060651209           % actual Euler constant
```

```
% gamma_cmv.m  Compute complex gamma function via ratl approx of exp(z)
%              T. Schmelzer and L. N. Trefethen 6/05
%
% Hankel's integral formula for Gamma(z) is
%
% 1/Gamma(z) = (2*pi*i)^(-1) int_C e^t t^{-z} dt
%
% where the contour C passes from -infty-0i around 0 to -infty+0i.
% To evaluate this integral approximately we replace e^t by a
% type (16,16) rational approximant over (-inf,0] of the form
% K + SUM R(k)/(z-P(k)), following Cody, Meinardus & Varga (1969).
% Values of z in the left half-plane are handled by reflection.
% (The details of all this are not obvious!)

  n = 16;                                                 % order
  K=0.21248537104952237488e-15;                           % constant
  P = [   -10.843917078696988026+19.277446167181652284i     % poles
          -5.2649713434426468895+16.220221473167927305i
          -1.4139284624888862114+13.497725698892745390i
           1.4193758971856659786+10.925363484496722585i
           3.5091036084149180974+8.4361989858843750826i
           4.9931747377179963991+5.9968817136039422260i
           5.9481522689511774808+3.5874573620183222829i
           6.4161776990994341923+1.1941223933701386874i ];
  R = [-.50901521865224915650e-6-.24220017652852287970e-4i  % residues
        .00021151742182466030907+.0043892969647380673918i
          .041023136835410021273-.15743466173455468191i
          -1.4793007113557999718+1.7686588323782937906i
           15.059585270023467528-5.7514052776421819979i
          -62.518392463207918892-11.190391094283228480i
           113.39775178483930527+101.94721704215856450i
          -64.500878025539646595-224.59440762652096056i ];

  x = -3.5:.1:4; y = -2.5:.1:2.5;
  [xx,yy] = meshgrid(x,y); z = xx + 1i*yy;     % plotting grid
  ii = (xx<0); z(ii) = 1 - z(ii);              % reflect left half-plane
  gi = zeros(size(z));                         % reciprocal of Gamma(z)
  for k = 1:n/2
    r = R(k); p = P(k);
    gi = gi - r*(p.^(-z)) - conj(r)*conj(p).^(-z);
  end
  gam = 1./gi;
  gam(ii) = pi./(gam(ii).*sin(pi*(1-z(ii))));  % reflect back
  figure, mesh(xx,yy,abs(gam))                 % plot |Gamma(z)|
  axis([-3.5 4 -2.5 2.5 0 6])
  xlabel Re(z), ylabel Im(z)
  text(4,-1.4,5.5,'|\Gamma(z)|','fontsize',20)
  colormap([0 0 1])                            % make the image blue
  format long, disp(gam(26,46:10:76).')        % approxs. to Gamma(1:4)
```

```
% taylor_coeffs.m  Use FFT to compute Taylor coefficients of f(z)=exp(z)
%                  L. N. Trefethen 8/03
%
% If f(z) is analytic in the closed disk about z=0 of radius R>1,
% its jth Taylor coefficient is given by
%
% c_j  =  (2*pi*i)^(-1) INT z^-(j+1) f(z) dz
%
% where the integral is over the unit circle.  This integral can be
% evaluated with error O(R^(-N)) by the N-point trapezoid rule.  The
% Fast Fourier Transform does this for many values of j at once, giving
% an extraordinarily fast method for computing Taylor coefficients.
% This method was introduced by Lyness and Moler (1967) and Lyness and
% Sandee (1971) and surveyed by Henrici in his SIAM Review article
% "Fast Fourier methods in complex analysis" (1979) and in v. 3 of
% Applied and Computational Complex Analysis (Wiley, 1986).  This
% short code demonstrates the method for f(z)=exp(z).

  f = inline('exp(z)');           % exp(z) has recognizable coeffs
  N = 18;                         % often one would use, say, 64 points
  zz = exp(2i*pi*(0:N-1)'/N);     % roots of unity
  c = fft(f(zz))/N;               % FFT of f evaluated at roots of unity
  c = real(c);                    % c must be real by symmetry
  disp(['       computed    '...  % table headings
        '         exact     '])
  disp([c 1./gamma(1:N)'])        % results, approximate and exact
```

```
% cheb_coeffs.m  Use FFT to compute Chebyshev coeffs of f(x)=exp(x)
%               L. N. Trefethen 1/05
%
% A smooth function f(x) on [-1,1] has a Chebyshev expansion
%
% f(x) = SUM_{k=0}^infty a_k T_k(x),
%
% a_k = (2/pi) INT_{-1}^1 (1-x^2)^(-1/2) f(x) T_k(x) dx,
%
% where T_k is the kth Chebyshev polynomial (except for k=0 the factor
% is 1/pi, not 2/pi); see e.g. Mason & Handscomb, Chebyshev Polynomials,
% 2003.  Like Taylor coeffs, Chebyshev coeffs can be computed by the FFT
% (Geddes, SIAM J. Numer. Anal. 1978).  If f is analytic in the closed
% ellipse with foci 1 and -1 whose semiaxis lengths add to R>1, then a_k =
% O(R^{-k}) and the error in its approx. by an N-point FFT is O(R^{-N}).

  format long
  f = inline('exp(x)');                          % e^x just for simplicity
  N = 14;                                        % usually one uses bigger N
  x = cos(pi*(0:2*N-1)'/N);                      % Chebyshev points (twice)
  g = real(fft(f(x)))/(2*N);                     % FFT of f evaluated at pts
  a = [g(1); g(2:N)+g(2*N:-1:N+2)];              % Chebyshev coeffs
  exact = [besseli(0,1) 2*besseli(1:N-1,1)]';  % analytic solution
  error = a - exact;                             % error
  disp(['       computed    '...                 % table headings
        '         exact     '...
        '         error     '])
  disp([a exact error])                          % print results
```

```
% barycentric.m  Barycentric interpolation in Chebyshev points
%                L. N. Trefethen 12/04
%
% A fast, stable, and exponentially accurate way to represent a smooth
% function f on [-1,1] is polynomial interpolation in Chebyshev points
% evaluated by the barycentric formula.  This method was proposed by
% Salzer (Computer J. 1972), publicized by Berrut & Trefethen (SIAM
% Review 2004), proved stable by N. Higham (IMA J. Numer. Anal. 2004).
% It is the basis of Battles' "chebfun" software system, and theorems
% concerning convergence are given by Battles and Trefethen (SIAM J.
% Sci. Comp. 2004).  This code, adapted from Berrut & Trefethen,
% illustrates the robustness of the method by applying it to a nonsmooth
% function f.  The interpolant is computed to machine accuracy, though
% it mtaches f to only 2-3 digits since f is not smooth.

  fun = inline('(x<=0).*(.5-abs(x+.5)) + (x>0).*(.1*sin(10*pi*x))');

  N = 100;                        % number of interpolation pts
  x = cos(pi*(0:N)'/N);           % Chebyshev points of 2nd kind
  f = fun(x);                     % function values to be interpolated
  xx = linspace(-1,1,2000)';      % interpolant is evaluated at 2000 pts
  numer = zeros(size(xx));
  denom = numer; exact = numer;
  c = [1/2
       ones(N-1,1)                % Salzer's coefficients
       1/2].*(-1).^((0:N)');
  warning off                     % turn off harmless divide-by-0 warnings
  for j = 1:N+1                   % loop over interpolation points
    xdiff = xx-x(j);
    temp = c(j)./xdiff;
    numer = numer + temp*f(j);
    denom = denom + temp;
    exact(xdiff==0) = j;          % check for sampling at interpolation pt
  end
  ff = numer./denom;              % the barycentric formula
  jj = find(exact);
  ff(jj) = f(exact(jj));          % use exact values at interpolation pts
  warning on

  % Plot the result:
  hold off, plot(xx,ff,'-b','linewidth',1.4)
  hold on, plot(x,f,'.k','markersize',15)
  ylim([-.15 .55]), grid on
  title(['barycentric interpolant in ' int2str(N) ' Chebyshev points'])
```

```
% pade.m  Compute Pade coefficients of a function f
%         L. N. Trefethen 3/05
%
% Let f(z) be analytic in the closed disk of radius R>0 about z=0.
% This program uses the most straightforward of methods to compute
% the coefficients of the (m,n) Pade approximant of f, i.e., the
% rational function of type (m,n) whose Taylor series matches that
% of f as far as possible.  We assume the generic case where
% f - r = O(z^(m+n+1)).  Notation: r = p/q with
% f(z) = a(1) + a(2)z + a(3)z^2 + ...
% p(z) = b(1) + b(2)z + ... + b(m+1)z^m
% q(z) =   1  + c(1)z + ... + c( n )z^n

  f = inline('exp(z)');                   % function to approximate
  N = 64;                                 % no. of pts. in FFT
  R = 1;                                  % radius of circle
  zz = R*exp(2i*pi*(0:N-1)'/N);
  a = fft(f(zz))/N;
  a = real(a);                            % delete if f is complex
  a = a./R.^(0:length(a)-1)';             % Taylor coeffs. of f
  m = 5; n = 5;                           % degrees of numer & denom

% Find denominator coefficients:
  if n==0, b = a(1:m+1), break, end       % n=0 is a special case
  firstcol = [zeros(n-m-1,1)
              a(max([1,m-n+2]):m+1)];
  lastrow = a(m+1:m+n);
  H = hankel(firstcol,lastrow);
  rhs = -a(m+2:m+n+1);
  c = H\rhs;
  c = [1; flipud(c)];
  disp('denominator coeffs:'), disp(c)
  z = roots(flipud(c));
  disp('poles:'), format long, disp(z)

% Find numerator coefficients:
  if m <= n
    firstcol = [zeros(m,1); a(1)];
    lastrow = a(1:m+1);
    H = hankel(firstcol,lastrow);
    b = H*flipud(c(1:m+1));
  else
    firstcol = [zeros(n,1); a(1:m-n+1)];
    lastrow = a(m-n+1:m+1);
    H = hankel(firstcol,lastrow);
    b = H*flipud(c);
  end
  disp('numerator coeffs:'), disp(b)

% Evaluate at z = 1:
  z = 1;
  fprintf('                f at z = %3.1f: %15.11f\n', z, f(z))
  value = polyval(flipud(b),z)/polyval(flipud(c),z);
  fprintf('(%1d,%1d) Pade approx at z = %3.1f: %15.11f\n', m, n, z, value)
```

```
% real_cf_approx  Caratheodory-Fejer approximation on [-1,1]
%                 L. N. Trefethen 1/05
%
% Best rational approximations to f(x) on [-1,1] equioscillate and
% can be computed by the Remes algorithm.  If f(x) is smooth, the
% simpler Caratheodory-Fejer method produces results that come
% extraordinarily close to equioscillation and optimality; see
% T. and Gutknecht, SIAM J. Numer. Anal. 1983.  This code is adapted
% from Trefethen, "Matlab codes for CF approximation", in Approximation
% Theory V, 1986 (one of the first papers ever to publish a MATLAB
% code).  The number s that it prints matches the best approximation
% error for exp(x) on [-1,1] to 10 digits.  For less smooth f, the
% approximations are less good and larger K is needed.

  f = inline('exp(x)');                     % function to approximate
  K = 16;                                   % no. of Cheby. coeffs
  m = 3; n = 3;                             % numer. and denom. degrees
  nfft = 256;                               % no. of points for FFT
  dim = K+n-m;                              % dimension of Hankel matrix
  np = n+1; nfft2 = nfft/2;                 % convenient abbreviations
  z = exp(2*pi*1i*(0:nfft-1)/nfft);         % roots of unity
  x = real(z);                              % Chebyshev points (twice)
  fx = f(x); c = real(fft(fx))/nfft2;
  H = hankel(c(1+mod((1:dim)+m-n,nfft)));  % Hankel matrix
  [U,S,V] = svd(H);                         % SVD of H
  s = S(np,np); u = U(dim:-1:1,np)';        % the needed singular value
  v = V(:,np)';                             %     and vector
  zr = roots(v); qout = zr(abs(zr)>1);      % roots outside the disk
  qc = real(poly(qout)); qc = qc/qc(np);
  q = polyval(qc,z);
  Q = q.*conj(q); Qc = real(fft(Q))/nfft2; % denominator
  Qc(1) = Qc(1)/2; Q = Q/Qc(1);
  Qc = Qc(1:np)/Qc(1);
  b = fft([u zeros(1,nfft-dim)])./fft([v zeros(1,nfft-dim)]);
  Rt = fx-real(s*z.^K.*b);
  Rtc = real(fft(Rt))/nfft2;
  gam = real(fft((1)./Q))/nfft2;
  gam = toeplitz(gam(1:2*m+1));
  if m==0 Pc = 2*Rtc(1)/gam;
     else Pc = 2*[Rtc(m+1:-1:2) Rtc(1:m+1)]/gam; end
  Pc = Pc(m+1:2*m+1); Pc(1) = Pc(1)/2;
  P = real(polyval(Pc(m+1:-1:1),z));        % numerator
  R = P./Q;                                 % rational CF approximant
  plot(x,fx-R,'-',x,[s;0;-s]*ones(1,nfft),'--')
  text(-.5,1.06*s,sprintf('s = %0.10g',s),'fontsize',18)
  s, error = norm(fx-R,'inf'), Pc, Qc
```

```
% integral1D.m  Evaluate a 1D integral by Gauss quadrature
%               L. N. Trefethen 6/03
%
% The integrand is just chosen to look interesting.

  f = inline('sin(x+cos(10*exp(x))/3)');
  hold off, fplot(f,[-1,1]), grid on            % plot the function
  disp('     N        integral')                % table headings
  for N = 10:10:60                              % loop over various N
    [x,w] = gauss(N);                           % compute nodes and weights
    fprintf('%6d%20.14f\n',N,w*f(x))            % results for this N value
  end
  text(-.8,.7,sprintf('integral = %13.11f'...   % label the plot
       ,w*f(x)),'fontsize',18)
  disp('press <return> to see nodes'), pause    % wait for user input
  hold on, plot(x,f(x),'.k','markersize',20)    % plot the Gauss nodes
```

```
% integral2D.m  Evaluate a 2D integral by tensor product Gauss quadrature
%               L. N. Trefethen 11/03
%
% As with integral1D.m, here the integrand is just chosen to
% be interesting.

% Define the function and plot it:
  f = inline('(y+1).*exp(x).*sin(16*y-4*(x+1).^2)');
  x = linspace(-1,1,50); y = x;
  [xx,yy] = meshgrid(x,y);
  ff = f(xx,yy);
  surf(x,y,ff), view(-40,50), zlim([-2 8])

% Compute its double integral using a 22x22 Gauss grid:
  N = 22;
  [x,w] = gauss(N); y = x;
  [xx,yy] = meshgrid(x,y);
  ff = f(xx,yy);
  format long
  I = w*ff*w'
  exact = 0.01795155832370
```

```
% cheb_vs_gauss.m  Compare Chebyshev and Gauss quadrature in [-1,1]
%                  L. N. Trefethen 1/05
%
% Chebyshev quadrature here means integration of the polynomial
% interpolant to f(x) sampled at Chebyshev points of the 2nd kind --
% i.e., Clenshaw-Curtis quadrature, implemented here with the FFT.
% Gauss quadrature means, as usual, integration of the interpolant
% in Legendre points.  Change the "%" characters to vary f.  The
% polynomial degree is N; the number of points is N+1.

  s = 'exp(-x.^2)';   exact = sqrt(pi)*erf(1); % entire
% s = '1./(1+x.^2)';  exact = pi/2;            % analytic
% s = 'abs(x.^3)';    exact = 1/2;             % C^2
% s = 'cos(10*x)';    exact = sin(10)/5;       % two-phase
  f = inline(s);

  cc = []; gg = []; NN = 1:40;                 % vectors of results
  for N = NN                                   % loop over various N

% Chebyshev quadrature:
    x = cos(pi*(0:2*N-1)'/N);                  % Chebyshev points (twice)
    g = real(fft(f(x)))/(2*N);                 % FFT of f evaluated at pts
    a = [g(1); g(2:N)+g(2*N:-1:N+2); g(N+1)];  % Chebyshev coeffs
    w = zeros(1,N+1);
    w(1:2:end) = 2./(1-(0:2:N).^2);            % weight vector
    I = w*a;                                   % integral as inner product
    cc = [cc; I];                              % store this result

% Gauss quadrature:
    beta = .5./sqrt(1-(2*(1:N)).^(-2));        % 3-term recurrence coeffs
    T = diag(beta,1) + diag(beta,-1);          % Jacobi matrix
    [V,D] = eig(T);                            % eigenvalue decomposition
    x = diag(D); [x,i] = sort(x);              % Legendre points
    w = 2*V(1,i).^2;                           % Gauss quadrature weights
    I = w*f(x);                                % integral as inner product
    gg = [gg; I];                              % store this result

  end
  hold off, semilogy(NN,abs(cc-exact),'.-r')   % plot Chebyshev error (red)
  hold on, semilogy(NN,abs(gg-exact),'b')      % plot Gauss error (blue)
  grid on, legend('Chebyshev','Gauss')
  xlabel('degree N'), ylabel('error'), title(s)
```

```
% pseudospectra.m  Compute pseudospectra of 60x60 Grcar matrix
%                  L. N. Trefethen 10/03
%
% This program computes pseudospectra by a method much faster than
% the obvious one of simply computing the SVD at many points.  The
% algorithm is due to Lui (SIAM J. Sci. Comp. 1997) and Trefethen
% (Acta Numerica 1999), and this code is adapted from psa.m in the
% latter paper.  The same algorithm is used by EigTool (Wright 2002).

% Define matrix and set up grid for contour plot:
  N = 60; A = gallery('grcar',N);                     % Grcar matrix
  npts = 40;                                          % grid resolution
  x1 = -2.4; x2 = 4.4; y1 = -3.4; y2 = 3.4;           % grid limits
  x = x1:(x2-x1)/(npts-1):x2;
  y = y1:(y2-y1)/(npts-1):y2;
  [xx,yy] = meshgrid(x,y); zz = xx + 1i*yy;           % grid

% Compute Schur form and plot eigenvalues:
  [U,T] = schur(A);                                   % triangularize
  if isreal(A), [U,T] = rsf2csf(U,T); end
  T = triu(T); eigA = diag(T); hold off               % eigenvalues
  plot(real(eigA),imag(eigA),'.','markersize',10)
  hold on, axis([x1 x2 y1 y2]), axis square, drawnow

% Compute resolvent norms and plot contours:
  sigmin = Inf*ones(length(y),length(x)); I = eye(N);
  for i = 1:length(y)
    if isreal(A) & (y2==-y1) & (i>length(y)/2)        % exploit symmetry
      sigmin(i,:) = sigmin(length(y)+1-i,:);
    else
      for j = 1:length(x)
        z = zz(i,j); T1 = z*I-T; T2 = T1';            % shifted matrix
        sigold = 0; qold = zeros(N,1);
        beta = 0; H = [];
        q = randn(N,1)+1i*randn(N,1); q = q/norm(q); % initial vector
        for k = 1:99                                  % crude Lanczos loop
          v = T1\(T2\q) - beta*qold;                  % inverse Lanczos step
          alpha = real(q'*v); v = v - alpha*q;
          beta = norm(v); qold = q; q = v/beta;
          H(k+1,k) = beta; H(k,k+1) = beta;           % tridiagonal matrix
          H(k,k) = alpha;
          sig = max(eig(H(1:k,1:k)));
          if (abs(sigold/sig-1)<.001)|(sig<3&k>2)...
                break, end
          sigold = sig;
        end
        sigmin(i,j) = 1/sqrt(sig);                    % min. singular value
      end
    end
  end
  contour(x,y,log10(sigmin+1e-20),-8:-1)              % levels 1e-1 ... 1e-8
  colormap([0 0 0])                                   % make it monochrome
```

```
% cuberootA.m  Compute the cube root of a matrix by a Cauchy integral
%              L. N. Trefethen 8/03
%
% If f is an analytic function and A is a matrix, f(A) can be computed
% by the Cauchy integral
%
% f(A)  =  (2*pi*i)^(-1) INT_G (zI-A)^(-1) f(z) dz
%
% where G is a closed contour enclosing the eigenvalues of A.  Often
% exponential accuracy can be achieved by taking G to be a circle and
% approximating the integral by the trapezoid rule.  Here we demonstrate
% the method for f(A) = A^(1/3).  Other functions such as e^A can be
% handled similarly.  For many functions other methods such as Newton's
% method may be superior, but this approach is certainly effective.  For
% larger matrices it would speeds things up to do a preliminary Schur or
% Hessenberg reduction (see Golub & Van Loan, 3rd ed., Problem P7.4.3).

  format long, format loose
  N = 4;                                % dimension of A
  A = randn(N)/sqrt(N) + 3*eye(N)       % eigs approx in disk rad 1, ctr 3
  disp('    np       ||B^3 - A||')     % table headings
  for np = 2.^(1:6)                     % number of pts in trapezoid rule
    circle = exp(2i*pi*(1:np)/np);      % np pts on the unit circle
    z0 = 3; radius = 2;                 % ctr, radius of circular contour
    z = z0 + radius*circle;             % nppoints on the contour
    I = eye(N); B = zeros(N);
    for i = 1:np
      R = inv(z(i)*I-A);                % resolvent matrix at point z(i)
      B = B + R*(z(i)-z0)*z(i)^(1/3);   % add up contributions to integral
    end
    B = real(B)/np;                     % clean up rounding errors
    resid = norm(B^3-A);                % compute the residual
    fprintf('%6d%20.14f\n',np,resid)    % display this result
  end
  disp(' ')
  disp('press <return> to see B')       % wait for user input
  pause, B                              % print final matrix
```

```
% operator_norm.m  Norm of infinite matrix from "100 Digit-Challenge"
%                  C. Ortner and L. N. Trefethen 2/05
%
% Let A be the infinite matrix with a_11=1, a_12=1/2, a_21=1/3,
% a_13=1/4, a_22=1/5, etc.  Problem 3 of the 100-Digit Challenge is
% to find the 2-norm of this operator; see Chap. 3 of Bornemann, et
% al., The SIAM 100-Digit Challenge, 2004.  This code gets the answer
% to about 12 digits by computing norms of matrices of dimensions
% 1,2,4,...,1024 and using epsilon extrapolation.

% Compute norms of finite matrices:
  M = 10; N = 2^M;                                % max matrix dimension
  A = zeros(N); b = cumsum(1:2*N)';
  for j = 1:N, A(:,j) = b(j+(1:N)-1)+1-j; end
  A = 1./A;                                       % matrix of that dim.
  data = [];
  for N = 2.^(0:M)
    data = [data; normest(A(1:N,1:N),5e-16)];   % norms of submatrices
  end
  format long
  disp('Raw data:'), disp(data)                   % print the data

% Extrapolate the data by epsilon algorithm:
  nd = length(data);
  E = zeros(nd,nd+1);
  E(:,2) = data;
  for j = 2:nd
    k = j:nd;                                     % epsilon extrap.
    E(k,j+1) = E(k-1,j-1)-1./(E(k,j)-E(k-1,j));
  end
  Y = diag(E,1); Y = Y(1:2:end);
  disp('Extrapolated values:'), disp(Y)
  disp('Exact solution:')
  disp('   1.27422415282122818821234 0...')
```

```
% planets.m  Orbits of three planets going off to infinity
%            L. N. Trefethen 1/05
%
% Here we use one of MATLAB's adaptive ODE solvers to track the orbits
% of three planets attracting each other with pairwise 1/r^2 forces
% initially at rest at the corners of a 3-4-5 right triangle.  The
% system is transiently chaotic: at around t=86, it "self-ionizes".
% Complex arithmetic is used for brevity.

% Set the mood by plotting some "stars":
  function planets()
  figure, set(gcf,'doublebuffer','on')
  fill(20*[-1 1 1 -1 -1],20*[-1 -1 1 1 -1],'k')
  hold on, grid on, axis([1.27*[-6.3 4.7] -3.5 7.5])
  x = 15*rand(250,1)-8; y = 11*rand(250,1)-3.5;
  plot(x,y,'.w','markersize',4)

% Initial conditions:
  x = 0; y = 3; z = 4i; u0 = [x y z 0 0 0].';
  xh = plot(real(x),imag(x),'.r','markersize',60);
  yh = plot(real(y),imag(y),'.y','markersize',60);
  zh = plot(real(z),imag(z),'.g','markersize',60);
  tol = 3e-14;
  opts = odeset('reltol',tol,'abstol',tol);
  axis off, title('t = 0','fontsize',18)

% Time-stepping:
  dt = .4; tmax = 100;
  for tnext = dt:dt:tmax;
    tspan = [tnext-dt tnext];
    [t,u] = ode113(@planetsfun,tspan,u0,opts);
    u0 = u(end,:);
    x = u(end,1); y = u(end,2); z = u(end,3);
    set(xh,'xdata',real(x),'ydata',imag(x))
    set(yh,'xdata',real(y),'ydata',imag(y))
    set(zh,'xdata',real(z),'ydata',imag(z))
    errest = tol*10^(1+tnext/12);
    title(sprintf('t = %3.0f        estimated error = %5.0e',...
      tnext,errest),'fontsize',18), drawnow
  end



  function v = planetsfun(t,u)
  x = u(1); y = u(2); z = u(3);
  yx = (y-x)/abs(y-x)^3;
  zx = (z-x)/abs(z-x)^3;
  zy = (z-y)/abs(z-y)^3;
  v = [u(4:6); yx+zx; -yx+zy; -zx-zy];
```

```
% two_disks.m  Compute Green's function exterior to two disks
%              L. N. Trefethen 8/03
%
% Suppose we seek the real function g(z) that is harmonic (i.e.,
% satisfies Laplace's equation) exterior to |z-2|=1 and |z+2|=1 with
% g(z)~log|z| as |z| -> infty.  This function can be approximated
% with an error decreasing exponentially as N->inf by series
%
% g(z) = .5*log|z-2| + .5*log|z+2|
%
%        + Re {SUM_{k=0}^N c(k)*[(2-z)^(-k)+(z+2)^(-k)]}.
%
% Given N, this code chooses good values for the coefficients c(k)
% by minimizing the sum of squares of the values of g(z) on equally-
% spaced points along the circles -- a linear least-squares problem.
% Symmetry is exploited to reduce the two circles to half of just
% one of them.  This method was used by Finn et al. in "Toplogical
% chaos in inviscid and viscous mixers", J. Fluid Mech. (2003).

  format long
  N = 20;                                 % no. of terms in expansion
  npts = 30;                              % no. of sample points
  z = 2 + exp((.5:npts-.5)'*1i*pi/npts);  % sample points on semicircle
  rhs = -real(log(2-z)+log(z+2))/2;       % right-hand side
  A = [];
  for k = 0:N                             % set up least-squares matrix
    A(:,k+1) = (2-z).^(-k)+(z+2).^(-k);
  end
  A = real(A);                            % symmetry is imposed here
  c = A\rhs                               % solve least-squares problem
  value_at_0 = log(2) + 2.^(1:-1:1-N)*c   % print value at z=0

% Contour plot of the solution:
  x = linspace(-5,5,95); y = linspace(-4,4,75);
  [xx,yy] = meshgrid(x,y); zz = xx+1i*yy;
  gg = real(log(2-zz)+log(zz+2))/2;
  for k = 0:N
    gg = gg + c(k+1)*real((2-zz).^(-k)+(zz+2).^(-k));
  end
  gg(abs(2-zz)<1 | abs(zz+2)<1) = 0;
  levels = 1.1:.1:4;
  z = exp(pi*1i*(-50:50)'/50); z = [z+2 z-2];
  hold off, fill(real(z),imag(z),[.7 .7 1]), hold on
  plot(z,'-b','linewidth',2)
  contour(xx,yy,gg,log(levels))
  axis equal, axis([-5 5 -4 4]), colormap([0 0 0])
  set(gca,'xtick',-4:2:4,'ytick',-4:2:4)
```

```
% many_disks.m  Compute Green's function exterior to several disks
%               L. N. Trefethen 8/03
%
% Suppose we seek the real function g(z) that is harmonic exterior to
% disks with centers c(j) and radii r(j) in the z-plane with g(z)~log|z|
% as |z|->infty.   This code is the analogue of two_disks.m for this
% more general problem.  The Green's function is approximated by series
%
% g(z) = e + SUM_{j=1}^J
%
%     d(j)*log|z-c(j)| + Re[sum_{k=1}^N (a(j,k)-i*b(j,k))*(z-z(j))^-k]
%
% with SUM d(j) = 1; all these coeffs are collected in the vector X.
% The unknowns determined by linear least-squares are e, d(2),...,d(J),
% {a(j,k)}, {b(j,k)}.  This method was used by Finn et al. in "Topo-
% logical chaos in inviscid and viscous mixers", J. Fluid Mech. 2003.

% Solve the problem:
  J = 3; c = [-2 2+1i 1-2i]; r = [1 .5 .7];  % define the geometry
  N = 10;                                    % no. terms in expansions
  npts = 30;                                 % no. sample pts on circles
  circ = exp((1:npts)'*2i*pi/npts);          % roots of unity
  z = []; for j = 1:J
     z = [z; c(j)+r(j)*circ]; end            % collocation points
  A = ones(size(z));                         % the constant term
  for j = 1:J
    A = [A log(abs(z-c(j)))];                % the logarithmic terms
    for k = 1:N
      zck = (z-c(j)).^(-k);
      A = [A real(zck) imag(zck)];           % the algebraic terms
    end
  end
  X = -A(:,[1 3:end])\A(:,2);                % solve least-squares prob.
  X = [X(1); 1; X(2:end)]/...
         (1+sum(X(2*N+2:2*N+1:end)));
  e = X(1); X(1) =[];
  d = X(1:2*N+1:end); X(1:2*N+1:end) = [];
  a = X(1:2:end); b = X(2:2:end);

% Plot the result:
  x = linspace(-5,5,95); y = linspace(-4,4,75);
  [xx,yy] = meshgrid(x,y); zz = xx+1i*yy; gg = e*ones(size(zz));
  for j = 1:J
    gg = gg+d(j)*log(abs(zz-c(j)));
    for k = 1:N, zck = (zz-c(j)).^(-k); kk = k+(j-1)*N;
      gg = gg+a(kk)*real(zck)+b(kk)*imag(zck); end
  end
  for j = 1:J, gg(abs(zz-c(j))<r(j)) = 0; end
  levels = 1.1:.1:4; z = exp(pi*1i*(-50:50)'/50); hold off
  for j = 1:J, disk = c(j)+r(j)*z; fill(real(disk),imag(disk),[1 .7 .7])
    hold on, plot(disk,'-r','linewidth',2), end
  contour(xx,yy,gg,log(levels)), axis equal, axis([-5 5 -4 4])
  colormap([0 0 0]), set(gca,'xtick',-4:2:4,'ytick',-4:2:4)
```

```
% rayleigh_disks.m  Potential exterior to a rectangular array of disks
%                   L. N. Trefethen 1/05
%
% Lord Rayleigh (Phil. Mag. 1892) considered the problem of the
% potential in the region exterior to an infinite rectangular array
% of circular disks.  He observed that the solution can be expressed
% by a rapidly convergent series.  This code computes that solution
% and plots it in a fundamental region.  We take the disks to have
% radius 1 and the rectangles to have half-length L and half-width W.
% The solutions are the real or imaginary parts of
%       w = SUM_{k=1}^N c(k)(z^s+z^(-s)) (s = 2*k-1).
% We find the coefficients c(k) by solving a least-squares problem, a
% discretization of the conditions Im w'=0 on the right boundary of T,
% Im w=1 on the top boundary.  Compare trapmap.m.

% Compute the coefficients:
  function rayleigh_disks()
  L = 2; W = 1.5;                           % dimensions of rectangle
  for N = 2:2:24                            % try various values of N
    np = 2*N;                               % no. of points on bndries
    z1 = L+linspace(0,1i*W,np).';           % points on right boundary
    z2 = 1i*W+linspace(0,L,np).';           % points on top boundary
    A1 = []; A2 = [];
    for k = 1:N
      s = 2*k-1;
      A1 = [A1 s*z1.^(s-1)-s*z1.^(-s-1)];   % w' on right boundary
      A2 = [A2 z2.^s+z2.^(-s)];             % w on top boundary
    end
    A = [imag(A1); imag(A2)];               % matrix of sampled fncts
    scl = (W^2+L^2).^(-1:-1:-N)';           % rescaling vector
    A = A*diag(sparse(scl));                % rescale the columns of A
    rhs = [zeros(np,1); ones(np,1)];        % right-hand side
    c = scl.*(A\rhs);                       % find coefficient vector
    s = 1:2:2*N; mu = (L.^s+L.^(-s))*c;     % conformal modulus
    fprintf('N = %2d  mu = %11.9f\n',N,mu)  % print modulus for this N
  end
  format short e, c                         % print coefficient vector

% Plot the map:
  f = figure;
  set(f,'defaultlinelinewidth',3)
  set(f,'defaultlinemarkersize',4)
  d = 0.08; [x,y] = meshgrid(-L:d:L,0:d:W);
  z = x+1i*y; z = z(abs(z)>1);
  draw(z,c,'.k')                            % points in domain
  draw(L+1i*linspace(-W,W),c,'r')           % left and right
  draw(1i*W+linspace(-L,L),c,'b')           % top and bottom
  draw(exp(1i*linspace(0,pi,101)),c,'g')    % boundary of disk
  title(sprintf('modulus = %11.9f',mu))
  for i=1:2, subplot(2,1,i), axis equal
     axis(1.05*axis), axis off, end

  function draw(z,c,color)   % draw points/lines in domain and range
  w = 0*z;
  for k = 1:length(c), s = 2*k-1; w = w+c(k)*(z.^s+z.^(-s)); end
  subplot(2,1,1), plot(z,color), hold on, plot(-z,color)
  subplot(2,1,2), plot(w,color), hold on, plot(-w,color)
```

```
% jaisson.m  Solve a Laplace problem posed by Denis Jaisson
%            L. N. Trefethen 12/04
%
% Let I = [-1,1] and let a>1 and b>0 be constants.  We seek the function
% u satisfying Laplace's equation in the region of the complex plane
% exterior to the four slits given by I plus the constants c2 = -a+ib,
% c1 = a+ib, d2 = -a-ib, d1 = a-ib, with u=1 on the c1 and d1 slits,
% u=-1 on the c2 and d2 slits, and u(inf) = 0.  We represent u by a sum
%
% u(z) =  Re { c(0)*[  log(jk(z-c1)) - log(jk(c2-z))
%                    + log(jk(z-d1)) - log(jk(d2-z)) ]
%
%      + SUM_{k=1}^N c(k)*[  jk(z-c1)^(-k) - jk(c2-z)^(-k) ]
%                          + jk(z-d1)^(-k) - jk(d2-z)^(-k) ] }
%
% where jk(z) = z+sqrt(z^2-1) maps exterior(I) to exterior(unit disk).
% The coefficients c(k) are found by linear least-squares.

  function jaisson()
  a = 2; b = 1;                                % define the geometry
  c1 = a+1i*b; c2 = -a+1i*b;                   % centers of slits in upper...
  d1 = a-1i*b; d2 = -a-1i*b;;                  %    ...and lower half-planes
  N = 18;                                      % no. of terms in expansion
  npts = 3*N;                                  % no. of sample points
  circle = exp((1:npts)'*2i*pi/npts);          % unit circle
  circle = (1+1e-14)*circle;                   % expand it slightly
  I = .5*(circle+1./circle);                   % unit interval
  z = c1 + I;                                  % sample points on slit
  A =  log(jk(z-c1))-log(jk(c2-z))...
     + log(jk(z-d1))-log(jk(d2-z));
  for k = 1:N                                  % set up least-squares matrix
    A = [A (jk(z-c1).^(-k)-jk(c2-z).^(-k)...
            +jk(z-d1).^(-k)-jk(d2-z).^(-k))];
  end
  A = real(A);                                 % symmetry is imposed here
  c = A\ones(size(z))                          % solve least-squares problem

% Contour plot of the solution:
  x = linspace(-5,5,85); y = linspace(-4,4,65);
  [xx,yy] = meshgrid(x,y); zz = xx+1i*yy;
  gg = c(1)*(log(jk(zz-c1))-log(jk(c2-zz))...
             +log(jk(zz-d1))-log(jk(d2-zz)));
  for k = 1:N
    gg = gg + c(k+1)*(jk(zz-c1).^(-k)-jk(c2-zz).^(-k)...
                    + jk(zz-d1).^(-k)-jk(d2-zz).^(-k));
  end
  gg = real(gg); levels = -1:.1:1; contour(xx,yy,gg,levels)
  hold on, axis equal, axis([-5 5 -4 4]), colormap([0 0 0])
  set(gca,'xtick',-4:2:4,'ytick',-4:2:4)
  plot(c1+I,'-b'), plot(c2+I,'-b'), plot(d1+I,'-b'), plot(d2+I,'-b')

  function jk = jk(z)       % Joukowski map ext(interval) -> ext(disk)
  main_branch = (real(z)>0)|((real(z)==0)&(imag(z)>0));
  sgn = 2*main_branch - 1; jk = z + sgn.*sqrt(z.^2-1);
```

```
% circular_drum.m - First 20 eigenmodes of Laplacian on unit disk
%                   L. N. Trefethen 8/03
%
% The problem of computing eigenmodes of a circular drum with zero
% boundary condition is classical; see e.g. Sec. 206 of Rayleigh's
% Theory of Sound.  The solutions can be expressed in terms of Bessel
% functions.  Here they are computed by a Fourier-Chebyshev spectral
% method in polar coordinates following Fornberg (SIAM J. Sci. Comp.
% 1965); see also Chap. 18 of Boyd, Chebyshev and Fourier Spectral
% Methods, 2001.  This code is adapted from p28.m of Trefethen,
% Spectral Methods in MATLAB, and calls the function cheb.m from that
% book.  All numbers are accurate to the digits printed but the plots
% of nodal lines nevertheless look rough because the grid is coarse.

% r coordinate, ranging from -1 to 1 (N must be odd):
  N = 27; [D,r] = cheb(N); N2 = (N-1)/2; D2 = D^2;
  D1 = D2(2:N2+1,2:N2+1); D2 = D2(2:N2+1,N:-1:N2+2);
  E1 =  D(2:N2+1,2:N2+1); E2 =  D(2:N2+1,N:-1:N2+2);

% t = theta coordinate, ranging from 0 to 2*pi (M must be even):
  M = 18; dt = 2*pi/M; t = dt*(1:M)'; M2 = M/2;
  D2t = toeplitz([-pi^2/(3*dt^2)-1/6 ...
              .5*(-1).^(2:M)./sin(dt*(1:M-1)/2).^2]);

% Laplacian in polar coords (chap. 11, Spectral Methods in MATLAB):
  R = diag(1./r(2:N2+1)); Z = zeros(M2); I = eye(M2);
  L = kron(D1+R*E1,eye(M))+ kron(D2+R*E2,[Z I;I Z])+ kron(R^2,D2t);

% Compute 20 eigenmodes:
  index = 1:20;
  [V,Lam] = eig(-L); Lam = diag(Lam);
  [Lam,ii] = sort(real(Lam)); ii = ii(index); V = real(V(:,ii));
  Lam = sqrt(Lam(index)/Lam(1));

% Plot them:
  [rr,tt] = meshgrid(r(1:N2+1),[0;t]); [xx,yy] = pol2cart(tt,rr);
  z = exp(1i*pi*(-100:100)/100);
  [ay,ax] = meshgrid(.72:-.24:0,0:.20:.80);
  for i = index
    u = reshape(real(V(:,i)),M,N2);
    u = [zeros(M+1,1) u([M 1:M],:)]; u = u/norm(u(:),inf);
    subplot('position',[ax(i) ay(i) .20 .24]), plot(z)
    axis(1.25*[-1 1 -1 1 -1 1]), axis off, hold on
    view(0,90), colormap([0 0 0]), axis square
    contour3(xx,yy,u-1,[-1 -1]), plot3(real(z),imag(z),-abs(z))
    text(-.7,1.18,sprintf('%13.10f',Lam(i)),'fontsize',8)
  end
```

```
% Ldrum.m  Compute eigenvalues of Laplacian on L-shaped region
%          T. Betcke and L. N. Trefethen 9/03
%
% The first three eigenvalues are computed by the method of
% particular solutions (Betcke & Trefethen, SIAM Review 2005).

% Compute subspace angles for various values of lambda:
  N = 36; k = 1:N;                          % orders in Bessel expansion
  np = 2*N;                                 % no. of bndry & interior pts
  t1 = 1.5*pi*(.5:np-.5)'/np;               % angles of bndry pts
  r1 = 1./max(abs(sin(t1)),abs(cos(t1)));   % radii of bndry pts
  t2 = 1.5*pi*rand(np,1);                   % angles of interior pts
  r2 = rand(np,1)./max(...
           abs(sin(t2)),abs(cos(t2)));      % radii of interior pts
  t = [t1;t2]; r = [r1;r2];                 % bndry and interior combined
  lamvec = .2:.2:25; S = [];                % trial values of lam
  for lam = lamvec
    A = sin(2*t*k/3).*...
          besselj(2*k/3,sqrt(lam)*r);
    [Q,R] = qr(A,0);
    s = min(svd(Q(1:np,:))); S = [S s];     % subspace angle for this lam
  end

% Convert to signed subspace angles:
  I = 1:length(lamvec);                     % all lam points
  J = I(2:end-1);                           % interior points
  J = J( S(J)<S(J-1) & S(J)<S(J+1) );       % local minima
  J = J + (S(J-1)>S(J+1));                  % points where sign changes
  K = 0*I; K(J) = 1;
  S = S.*(-1).^cumsum(K);                   % introduce sign flips
  subplot(3,1,1)
  hold off, plot(lamvec,S), hold on         % plot signed angle function
  plot([0 max(lamvec)],[0 0],'-k')          % plot lam axis

% Find eigenvalues via 9th-order interpolation:
  for j = length(J):-1:1
    I = J(j)-5:J(j)+4;
    lam = polyval(polyfit(S(I)/norm(S(I)),lamvec(I),9),0);
    plot(lam*[1 1],[-1 1],'r')
    text(lam,.6,sprintf('%13.9f',lam),'color','r')
  end

% Plot the first eigenfunction:
  [X,Y] = meshgrid(-1:.05:1,-1:.05:1); Z = X(:)+i*Y(:);
  p = [0 1i -1+1i -1-1i 1-1i 1];
  [in on] = inpolygon(real(Z),imag(Z),real(p),imag(p));
  zB = Z(on); zI = Z(in&~on); z = [zB;zI]; t = mod(angle(z/i),2*pi);
  A = besselj(2*k/3,sqrt(lam)*abs(z)).*sin(2*t*k/3);
  [Q,R] = qr(A,0); [U,S,V] = svd(Q(1:length(zB),:));
  V = V(:,end); Q = Q*V; [t,I] = max(abs(Q)); Q = Q/Q(I);
  F = NaN*zeros(size(Z));
  F(in&~on) = Q(length(zB)+1:end); F(on) = Q(1:length(zB),:);
  F = reshape(F,length(X),length(Y)); subplot(3,1,2:3)
  surf(X,Y,F), view(-150,40), axis off, zlim([0 .7])
```

```
% isodrum.m - Third eigenmode of Laplacian on isospectral octagon
%             L. N. Trefethen 4/05
%
% The eigenmode is computed to about 8 digits by the method of particular
% solutions.  This image first computed by Driscoll (SIAM Review 1999).

% Define geometry, fix boundary and interior pts:
  function isodrum()
  v = [0 2 2-2i 4 4+2i 2i 4i 2i-2];                  % vertices
  N = 16;                                            % max Bessel order
  s = -cos((1:N)'*pi/(N+1)); s = .5*(s+1);           % N pts in [0,1]
  bd = v(8) + s*(v(1)-v(8));
  for j = 1:7, bd = [bd;v(j)+s*(v(j+1)-v(j))]; end   % boundary pts
  x = -2+6*rand(50*N,1); y = -2+6*rand(50*N,1);
  ii = inpolygon(x,y,real(v),imag(v));
  int = x(ii) + 1i*y(ii); int = int(1:10*N);         % interior points
  z = [bd; int];                                     % all pts
  Nbd = length(bd);                                  % no. of boundary pts

% Find eigenvalue and coeffs of eigenmode:
  opt = optimset('tolx',1e-8);
  lam = fminbnd(@isofun,5,6,opt,N,Nbd,z,v);          % find eig in [5,6]
  N = 12;                                            % small N for plotting
  [f,A,Q,R] = isofun(lam,N,Nbd,z,v);
  [U,S,V] = svd(Q(1:Nbd,:)); c = R\V(:,end);         % coeffs of eigenmode

% Plot eigenmode:
  [X,Y] = meshgrid(-2:.075:4,-2:.075:4);             % plotting grid
  z = X(:) + 1i*Y(:);
  [f,A] = isofun(lam,N,Nbd,z,v); F = A*c;            % construct eigenmode
  in = inpolygon(real(z),imag(z),real(v),imag(v));
  F(~in) = NaN; F = F/max(abs(F));                   % make exterior blank
  F = reshape(F,size(X)); hold off
  surf(X,Y,F,'Linestyle','none'), caxis([-1 1])      % color plot
  view(2), axis off, hold on
  contour3(X,Y,F,-.8:.2:.8), axis equal              % level curves
  set(get(gca,'children'),'edgecolor','k')           % make curves black
  plot(v([1:8 1]),'-k')                              % plot polygon
  text(.5,2.4,sprintf('%11.7f',lam),'fontsize',22)   % print eigenvalue


  function [f,A,Q,R] = isofun(lam,N,Nbd,z,v)
  w1 = z-v(1); t1 = angle(w1);      r1 = abs(w1);    % angles and radii
  w2 = z-v(2); t2 = angle(w2)+pi/2; r2 = abs(w2);    %   of sample points
  w4 = z-v(4); t4 = angle(w4/1i);   r4 = abs(w4);    %   with respect to
  w6 = z-v(6); t6 = angle(w6/1i);   r6 = abs(w6);    %   singular corners
  t6(t6<0) = t6(t6<0)+2*pi;                          % fix up t6 branch
  k = 1:N;                                           % Bessel orders
  a1 = 4*k/3; a2 = 2*k/3; a4 = 4*k/3; a6 = 2*k/3;    % scaled orders
  A = [sin(t1*a1).*besselj(a1,sqrt(lam)*r1)...       % matrix of
       sin(t2*a2).*besselj(a2,sqrt(lam)*r2)...       %   particular solns
       sin(t4*a4).*besselj(a4,sqrt(lam)*r4)...
       sin(t6*a6).*besselj(a6,sqrt(lam)*r6)];
  [Q,R] = qr(A,0); f = min(svd(Q(1:Nbd,:)));         % subspace angle
```

```
% trapmap.m  Conformal map of a trapezoid to a rectangle
%            L. N. Trefethen 7/03
%
% Conformal maps of polygons can be represented by the Schwarz-
% Christoffel formula, which must then be implemented numerically.
% Another approach is to seek an efficient numerical representation
% directly.  This code follows that route to compute a conformal map
% w = f(z) of a trapezoid T to a rectangle R; it plots the map and
% prints the conformal module of T, i.e., the aspect ratio of R.
% f is analytic except at the lower-left corner of T, and the map
% is represented as a series of fractional powers at that point:
%
% w = f(z) = SUM_{k=1}^N c(k) z^((2/3)(2*k-1))
%
% We find the coefficients c(k) by solving a least-squares problem,
% a discretization of the conditions Im w' = 0 on the right boundary
% of T, Im w = 1 on the top boundary.

% Compute the coefficients:
  format long
  N = 30;                               % order of expansion
  np = 2*N;                             % no. of points on boundaries
  z1 = linspace(1,1+1i,np).';           % points on right boundary
  z2 = linspace(-1+1i,1+1i,np).';       % points on top boundary
  A1 = []; A2 = [];
  for k = 1:N
    s = (2/3)*(2*k-1);
    A1 = [A1 s*z1.^(s-1)];              % f' on right boundary
    A2 = [A2 z2.^s];                    % f on top boundary
  end
  A = [imag(A1); imag(A2)];             % matrix of sampled functions
  rhs = [zeros(np,1); ones(np,1)];      % right-hand side
  c = A\rhs                             % coefficients
  mu = sum(c)                           % conformal modulus

% Plot the map:
  subplot(2,1,1)
  plot([0 1 1+1i -1+1i 0],'r')
  hold on, axis equal, axis off
  subplot(2,1,2)
  plot([0 mu mu+1i 1i 0],'b')
  hold on, axis equal, axis off
  for y = .1:.1:.9
    z = linspace(y*(-1+1i),1+(1i*y),50).';
    subplot(2,1,1), plot(z,'r'), A = [];
    for k = 1:N, A = [A z.^((2/3)*(2*k-1))]; end
    subplot(2,1,2), plot(A*c,'b')
  end
  for x = -.9:.1:.9
    z = linspace(x-1i*min(x,0),x+1i,50).';
    subplot(2,1,1), plot(z,'r'), A = [];
    for k = 1:N, A = [A z.^((2/3)*(2*k-1))]; end
    subplot(2,1,2), plot(A*c,'b')
  end
```

```
% schwarz_christoffel.m  Schwarz-Christoffel map of a polygon
%                        L. N. Trefethen 2/05
%
% Let P be a polygonal region in the complex plane with N vertices
% w(k) and interior angles pi*a(k).  Any conformal map f of the unit
% disk to P can be represented by the S-C integral
%       f(z) = A + C INT_0^z PROD (1-z/z(k))^(a(k)-1) dz
% for some constants A and C and prevertices z(k) on the unit circle.
% This code constructs such a map following Sec. 3.1 of Driscoll & T.,
% Schwarz-Christoffel Mapping", 2002, evaluating integrals by a
% stripped-down version of Laurie's double exponential quadrature code.
% For polygons with "crowding" one would need other quadrature methods.
% The map is made unique by z(N-2)=-1, z(N-1)=-i, z(N)=1.

% Define geometry & solve parameter problem:
  function schwarz_christoffel()
  p = [-1+1i -1-1i 1-1i 1 0 1i];              % vertices of P
  N = length(p);                              % number of vertices
  sides = angle(diff(p([N 1:N 1])))/pi;       % orientations of sides
  a = mod(1-diff(sides),2), b = a-1;          % angle parameters
  c = pi/2; h = 1/8;                          % compute quadrature data
  m1 = 6.6135/c; m2 = 3.6045/c;               %   -- these lines taken
  n1 = floor(m1/h); n2 = floor(m2/h);         %   from Laurie's q_de.m
  x = h*(-n1:n2); w = h*ones(1,length(x));
  w = c*w.*cosh(c*x); x = sinh(c*x);
  e2 = exp(x); e1 = 1./e2; s = e1+e2;
  x = (e2./s)'; w = 2*w./s./s;                % nodes and weights
  op = optimset('tolfun',1e-10);
  phi = fsolve(@scfun,zeros(1,N-3),op,p,b,N,x,w);
  t = cumsum(exp(cumsum([0 phi])));
  z = exp(pi*1i*[t/t(end) -1/2 0]).'          % prevertices
  C = (p(2)-p(1))/scquad(z(1),z(2),z,b,N,x,w) % the constant C
  A = p(1) - C*scquad(0,z(1),z,b,N,x,w);      % the constant A

% Plot the map:
  figure, subplot(1,2,1), plot(exp(linspace(0,2i*pi)),'k')
  hold on, plot(z,'.'), axis(1.05*axis,'equal','off')
  for r = .1:.1:.9
    npts = round(20/(1-r)); zz = r*exp(linspace(0,2i*pi,npts));
    subplot(1,2,1), plot(zz,'b'), ww = 0*zz;
    for k = 1:npts, ww(k) = A + C*scquad(0,zz(k),z,b,N,x,w); end
    subplot(1,2,2), plot(ww,'r'), hold on
  end
  subplot(1,2,2), plot(p([1:N 1]),'k'), axis(1.05*axis,'equal','off')

  function I = scquad(z1,z2,z,b,N,x,w)        % evaluate SC integral
  zz = z1+x*(z2-z1); s = ones(size(zz));
  for k = 1:N, s = s.*(1-(1-1e-14)*zz/z(k)).^b(k); end
  I = (z2-z1)*w*s;

  function f = scfun(phi,p,b,N,x,w)           % S-C parameter problem
  t = cumsum(exp(cumsum([0 phi])));           % change of variables to
  z = exp(pi*1i*[t/t(end) -1/2 0]).';         %   eliminate constraints
  C = (p(2)-p(1))/scquad(z(1),z(2),z,b,N,x,w);
  f = zeros(N-3,1);
  for k = 1:N-3
    s = C*scquad(z(k+1),z(k+2),z,b,N,x,w);
    f(k) = abs(s) - abs(p(k+2)-p(k+1));
  end
```

```
% mathieu.m  Mathieu eigenvalues following Abramowitz & Stegun
%           L. N. Trefethen 6/03
%
% The Mathieu operator is L: u -> -u_xx + 2qcos(2x)u with periodic
% boundary conditions on [-pi,pi], where q is a real parameter.
% A plot of eigenvalues of L as a function of q is given on p. 724
% of Abramowitz & Stegun, Handbook of Mathematical Functions, 1964.
% This program produces almost the same image based on numbers
% computed to close to machine precision by a Fourier spectral
% collocation method (small but interesting differences can be seen
% on close inspection).  The idea comes from Weideman and Reddy, ACM
% Trans. Math. Softw. 2001, and this particular code is adapted from
% p21.m of Trefethen, Spectral Methods in MATLAB.

  N = 42;                                          % no. of grid points
  h = 2*pi/N; x = h*(1:N);                         % grid
  D2 = toeplitz([-pi^2/(3*h^2)-1/6 ...             % differentiation matrix
      -.5*(-1).^(1:N-1)./sin(h*(1:N-1)/2).^2]);
  qq = 0:.2:15;                                    % vector of values of q
  data = [];
  for q = qq;                                      % for each value of q
    e = sort(eig(-D2 + 2*q*diag(cos(2*x))))';      %    compute eigenvalues
    data = [data; e(1:11)];                        %    and store them
  end
  figure, subplot(1,2,1)
  set(gca,'colororder',[0 0 1],...
          'linestyleorder','-|--')
  hold on, plot(qq,data)                           % draw the plot
  xlabel q, ylabel \lambda                         %    in panel 1
  axis([0 15 -24 32])
  set(gca,'ytick',-24:4:32)
  subplot(1,2,2)                                   % print a message
  c = 'color'; f = 'fontsize'; r = [.4 0 .7];      %    in panel 2
  text(0,.70,'This plot duplicates',c,r,f,15)
  text(0,.64,'Fig. 20.1 of Abramowitz',c,r,f,15)
  text(0,.58,'& Stegun (1964)',c,r,f,15)
  axis([0 1 0 1]), axis off
```

```
% orr_sommerfeld.m  Eigenvalues of Orr-Sommerfeld operator
%                   L. N. Trefethen 8/03
%
% In classical hydrodynamic stability theory, stability of plane
% Poiseuille flow (between two infinite flat plates) is determined by
% looking for eigenvalues in the right half-plane of the Navier-Stokes
% operator linearized about the laminar flow solution (parabolic
% velocity profile).  Orszag showed that at Reynolds number R~=5772
% and Fourier parameter alpha~=1.02, an eigenvalue crosses into the
% right half-plane (J. Fluid Mech. 1971).  This code uses spectral
% methods to show these eigenvalues for four successively better
% resolutions.  The method is derived from Schmid & Henningson (1993)
% and Weideman & Reddy (ACM Trans. Math. Softw. 2001) and the code
% comes from p40.m of Trefethen, "Spectral Methods in MATLAB", 2000.

  R = 5772.2217;                                % Reynolds number
  a = 1.02055;                                  % Fourier param. alpha
  [ay,ax] = meshgrid([.54 .04],[.1 .5]);        % plotting grid
  for N = 40:20:100                             % various resolutions

% 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*a^2*D2+a^4*I)/R ...
         - 2i*a*I - 1i*a*diag(1-x(2:N).^2)*(D2-a^2*I);
    B = D2-a^2*I;
    e = eig(A,B);
    i = N/20-1; subplot('position',[ax(i) ay(i) .37 .37])
    plot(e,'.','markersize',12)
    grid on, axis([-.8 .2 -1 0]), axis square
    set(gca,'fontsize',7)
    title(['N = ' int2str(N) '    \lambda_{max} = ' ...
      num2str(max(real(e)),'%14.10f')],'fontsize',10), drawnow
  end
```

```
% OScritical.m  Find critical R for Orr-Sommerfeld problem
%               L. N. Trefethen 1/05
%
% The context here is as in orr_sommerfeld.m: plane Poiseuille flow.
% This code finds the critical parameters R (Reynolds number) and
% alpha (streamwise Fourier parameter) for eigenvalue instability:
% the smallest value of R for which, for some alpha, the linearized
% Navier-Stokes operator has an imaginary eigenvalue.  It gets R
% correct to about 8 digits.

  function OScritical()
  R0 = 6000;
  opts = optimset('tolx',1e-7);
  R = fzero(@fun1,R0,opts);         % critical Reynolds number



% fun1.m  Given R, optimizes over alpha to maximize the
%         largest real part of an Orr-Sommerfeld eigenvalue
%         (actually, to minimize the negative of that)

  function s = fun1(R)
  opts = optimset('tolx',1e-6);
  [alpha,s] = fminbnd(@fun2,0,2,opts,R);
  disp(sprintf('R = %10.5f    alpha = %8.6f    s = %14.11f',R,alpha,s))



% fun2.m  Given alpha and R, finds maximum real part of an
%         Orr-Sommerfeld eigenvalue (actually its negative)

  function s = fun2(alpha,R)

% 2nd- and 4th-order differentiation matrices:
  N = 48;
  [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*alpha^2*D2+alpha^4*I)/R - 2i*alpha*I ...
      - 1i*alpha*diag(1-x(2:N).^2)*(D2-alpha^2*I);
  B = D2-alpha^2*I;
  s = -max(real(eig(A,B)));
```

```
% kuramoto_siv.m  Solve Kuramoto-Sivashinsky eq. by ETDRK4 scheme
%                 A.-K. Kassam and L. N. Trefethen 4/03
%
% We consider the time-dependent PDE u_t = -u*u_x - u_xx - u_xxxx
% with periodic BCs on [0,32*pi].  The solutions are chaotic, with
% small perturbations growing by a factor of 10^8 over the range
% 0<t<150.  This code achieves 10-digit accuracy for small t and
% graphical accuracy for larger t.  The x discretization is Fourier
% spectral (see e.g. Trefethen, "Spectral Methods in MATLAB") and
% the t discretization is ETDRK4 4th-order (see Cox & Matthews, J.
% Comp. Phys. 2002 and Kassam & Trefethen, SIAM J. Sci. Comp. 2004).

% Spatial grid and initial condition:
  N = 128; x = 32*pi*(0:N-1)'/N;
  u = cos(x/16).*(1+sin(x/16)); v = fft(u);

% Precompute constants by contour integrals (Kassam-Trefethen):
  h = 1/4;                          % time step
  k = [0:N/2-1 0 -N/2+1:-1]'/16;    % wave numbers
  L = k.^2 - k.^4;                  % Fourier multipliers
  E = exp(h*L); E2 = exp(h*L/2);
  M = 16;                           % no. of pts for complex means
  r = exp(1i*pi*((1:M)-.5)/M);      % roots of unity
  LR = h*L(:,ones(M,1)) + r(ones(N,1),:);
  Q  = h*real(mean(           (exp(LR/2)-1)./LR             ,2));
  f1 = h*real(mean(  (-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3  ,2));
  f2 = h*real(mean(    (4+2*LR+exp(LR).*(-4+2*LR))./LR.^3    ,2));
  f3 = h*real(mean(  (-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3  ,2));

% Time-stepping by ETDRK4 formula (Cox-Matthews):
  uu = u; tt = 0;
  tmax = 150; nmax = round(tmax/h); nplt = floor((tmax/100)/h);
  g = -0.5i*k;
  for n = 1:nmax
    t = n*h;
    Nv = g.*fft(real(ifft(v)).^2);
    a = E2.*v + Q.*Nv;
    Na = g.*fft(real(ifft(a)).^2);
    b = E2.*v + Q.*Na;
    Nb = g.*fft(real(ifft(b)).^2);
    c = E2.*a + Q.*(2*Nb-Nv);
    Nc = g.*fft(real(ifft(c)).^2);
    v = E.*v + Nv.*f1 + (Na+Nb).*f2 + Nc.*f3;
    if mod(n,nplt)==0
      u = real(ifft(v)); uu = [uu u]; tt = [tt t];
    end
  end

% Plot the solution:
  figure, surf(x,tt,uu'), shading interp, lighting phong, axis tight
  view([0 90]), colormap(autumn), zlim([-5 50])
  light('color',[1 1 0],'position',[1 2 2])
  material([.3 .6 .6 40 1])
```

```
% kdv.m - Solve KdV equation by Fourier spectral/ETDRK4 scheme
%         A.-K. Kassam and L. N. Trefethen 4/03
%
% This code solves the Korteweg-de Vries eq. u_t+uu_x+u_xxx=0
% with periodic BCs on [-pi,pi] and initial condition given by
% a pair of solitons.  The curve evolves up to t=0.005 and at
% the end u(x=0) is printed to 6-digit accuracy.  Changing N
% to 384 and h to 2.5e-7 improves this to 10 digits but takes
% four times longer.

% Set up grid and two-soliton initial data:
  N = 256; x = (2*pi/N)*(-N/2:N/2-1)';
  A = 25; B = 16;
  u = 3*A^2*sech(.5*(A*(x+2))).^2+3*B^2*sech(.5*(B*(x+1))).^2;
  p = plot(x,u,'linewidth',3);
  axis([-pi pi -200 2200]), grid on

% Precompute ETDRK4 scalar quantities (Kassam-Trefethen):
  h = 1e-6;                                  % time step
  k = [0:N/2-1 0 -N/2+1:-1]';                % wave numbers
  L = 1i*k.^3;                               % Fourier multipliers
  E = exp(h*L); E2 = exp(h*L/2);
  M = 64;                                    % no. pts for complex means
  r = exp(2i*pi*((1:M)-0.5)/M);              % roots of unity
  LR = h*L(:,ones(M,1))+r(ones(N,1),:);
  Q  = h*mean(                 (exp(LR/2)-1)./LR   ,2);
  f1 = h*mean((-4-LR+exp(LR).*(4-3*LR+LR.^2))./LR.^3,2);
  f2 = h*mean(    (4+2*LR+exp(LR).*(-4+2*LR))./LR.^3,2);
  f3 = h*mean((-4-3*LR-LR.^2+exp(LR).*(4-LR))./LR.^3,2);
  g = -.5i*k;

% Time-stepping by ETDRK4 formula (Cox-Matthews):
  set(gcf,'doublebuffer','on')
  disp('press <return> to begin'), pause  % wait for user input
  t = 0; step = 0; v = fft(u);
  while t+h/2 < .005
    step = step+1;
    t = t+h;
    Nv = g.*fft(real(ifft(v)).^2);
    a = E2.*v+Q.*Nv;        Na = g.*fft(real(ifft(a)).^2);
    b = E2.*v+Q.*Na;        Nb = g.*fft(real(ifft(b)).^2);
    c = E2.*a+Q.*(2*Nb-Nv); Nc = g.*fft(real(ifft(c)).^2);
    v = E.*v+(Nv.*f1+(Na+Nb).*f2+Nc.*f3);
    if mod(step,50)==0
      u = real(ifft(v)); set(p,'ydata',u)
      title(sprintf('t = %7.5f',t),'fontsize',18), drawnow
    end
  end
  text(-2.4,900,sprintf('u(0) = %11.7f',u(N/2+1)),...
               'fontsize',18,'color','r')
```

TWO ITEMS OF "SOFTWARE" USED BY SEVERAL OF THE CODES:

```
% cheb.m - compute D = differentiation matrix, x = Chebyshev grid
%
% This function serves as the basis for any Chebyshev collocation
% spectral method.  From Trefethen, "Spectral Methods in MATLAB".

  function [D,x] = cheb(N)
  if N==0, D=0; x=1; return, end
  x = cos(pi*(0:N)/N)';
  c = [2; ones(N-1,1); 2].*(-1).^(0:N)';
  X = repmat(x,1,N+1);
  dX = X-X';
  D  = (c*(1./c)')./(dX+(eye(N+1)));      % off-diagonal entries
  D  = D - diag(sum(D,2));                % diagonal entries
```

```
% gauss.m  Gauss quadrature nodes and weights on [-1,1]
%          L. N. Trefethen 6/03

% This function computes nodes (column vector x) and
% weights (row vector w) for the N-point Gauss-Legendre
% quadrature formula:
%
% INT_{-1}^1 f(s) ds ~= w*f(x)
%
% x and w are obtained from a tridiagonal eigenvalue problem as
% proposed by Golub & Welsch (Math. Comp. 1969), an idea previously
% considered by by Goertzel 1954, Wilf 1960, and Gordon 1968; see
% Gautschi, Orthogonal Polynomials: Computation and Approximation,
% Oxford 2004.  This code comes from Trefethen, Spectral Methods
% in MATLAB, 2000.  It is regrettably slow since MATLAB does not
% take advantage of tridiagonal structure for eigenvalue problems.

  function [x,w] = gauss(N)
  beta = .5./sqrt(1-(2*(1:N-1)).^(-2));   % 3-term recurrence coeffs.
  T = diag(beta,1) + diag(beta,-1);       % set up Jacobi matrix
  [V,D] = eig(T);                         % the eigenvalue problem
  x = diag(D); [x,i] = sort(x);           % nodes
  w = 2*V(1,i).^2;                        % weights
```