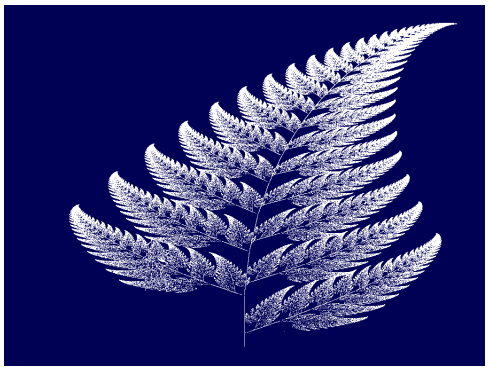


Solutions to Exercises
Numerical Computing with MATLAB

Cleve Moler

The MathWorks, Inc.

February 18, 2004

Please Do Not Copy or Redistribute

Contents

Chapter 1

Introduction to MATLAB

Exercises

1.1. Which familiar rectangle is closest to a golden rectangle.

```
w = [5 11 14 12 16 1024];
h = [3 8.5 8.5 9 9 768];
w./h
abs(w./h-phi)
1.6667    1.2941    1.6471    1.3333    1.7778    1.3333
0.0486    0.3239    0.0290    0.2847    0.1597    0.2847
```

8.5-by-14 inch US legal paper is closest.

1.2. ISO standard A4 paper.
`297/210 = 1.4143` $\approx$ `sqrt(2)`.
Folding A4 paper in half preserves its aspect ratio.
(A4 is A3 folded in half, A5 is A4 folded in half.)
See `a4paper.m`.

1.3. How many terms in the continued fraction?

```
goldfract(22)
err = 5.4457e-010.

for n = 37:39
   goldfract(n)
end
err = [-eps, eps, 0]
```

1.4. Use backslash to compute coefficients.
Numerically:

```
phi = (1 + sqrt(5))/2;
A = [ 1 1; phi 1-phi]
b = [ 1; 1]
c = A\b
= [0.7236; 0.2764]
```

Symbolically:

```
syms phi
A = [ 1 1; phi 1-phi]
b = [ 1; 1]
c = A\b
= 1/(2*phi-1)*[phi; 1-phi]
```

1.5. Slope of the line.
$\log(f_n) \approx \log(\phi) \cdot n$, so the slope is approximately $\log(\phi)$

1.6. Let T_n = execution time of `fibnum(n)`. T_n satisfies $T_n \approx T_{n-1} + T_{n-2}$. Measure T_n for, say, $n = 20$ and $n = 21$. Then use this recursion to compute T_n for $n = 50$.

```
tic, fibnum(20), T(20) = toc;
tic, fibnum(21), T(21) = toc;
for n = 22:50
   T(n) = T(n-1)+T(n-2);
end
T(50)
T(50)/60/60/24
```

On my machine, I get `T(50) = 7.9083e+5` seconds `= 9.1532` days.

1.7. Largest floating point Fibonacci number.
$f_n \approx c\phi^n, c = .7236$

```
n = (log(1/eps)-log(c))/log(phi)
= 75.5741
```

$f_{76} \approx$ `1/eps`. Actually, f_{77} is also computed exactly.

```
n = (log(realmax)-log(c))/log(phi)
= 1475.7
```

f_{1475} does not overflow, but f_{1476} does.

1.8. Repeat `X = A*X`.
$A^n = \begin{pmatrix} f_n & f_{n-1} \\ f_{n-1} & f_{n-2} \end{pmatrix}$
A^{1475} does not overflow, but A^{1476} does.

1.9. See `fernpink.m`. Lines changed from `fern.m`:

```
set(gcf,'color','black','menubar','none', ...
   'numbertitle','off','name','Fractal Fern')
pink = [3/4 1/2 1/2];
stop = uicontrol('style','toggle','string','stop', ...
   'background','black','foreground',pink);
```

1.10. Resizing the fern window causes all memory of the points to be lost. `finitefern.m` saves the points in a MATLAB array instead of in the graphics hardware. This allows MATLAB to plot the points again for printing or refreshing the screen.

1.11. See `fernflip.m`. Lines changed from `fern.m`:

```
h = plot(x(2),x(1),'.');
axis([0 10 -3 3])
set(h,'xdata',x(2),'ydata',x(1));
text(-1.5,-0.5,s,'fontweight','bold');
```

1.12. The length of the base stem of the fern is `A4(2,2)*max(x(2))`, or roughly `.16*10 = 1.6`. Successively shorter copies of this stem appear up the spine of the fern.

1.13. The coordinates of the lower end of the fern's stem are (0,0).

1.14. The coordinates of the upper tip end of the fern can be found by iterating the statement `x = A1*x + b1` a few hundred times, or, preferably, by solving the equation for the fixed point,

$$x = A_1 x + b_1$$

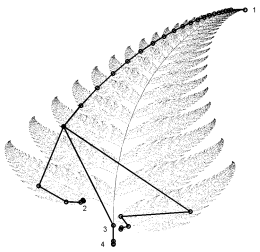
$$(I - A_1)x = b_1$$

```
I = eye(2,2);
x = (I - A1)\b1
```

Either approach gives

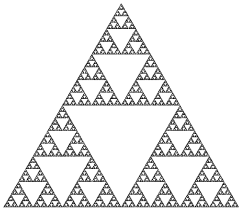
$$x = \begin{pmatrix} 2.6566 \\ 9.9585 \end{pmatrix}.$$

1.15. Fern trajectories.
See `ferntrajs.m`.



1.16. Make your own `fern.png`.

1.17. Sierpinski's triangle.
See `sierpinski.m` and `finitesierpinski.m`.



1.18. `greetings(phi)` needs `phi` to be irrational. Of course, there are no irrational floating point numbers, but `greetings(phi)` works best if `phi` is not the ratio of two small integers. `phi = (1+sqrt(2))/5` is interesting.

1.19. 4-by-4 magic square is singular.

```
A = magic(4)
null(A)
null(A,'r')
null(sym(A))
rref(A)
```

These four statements are four different ways to discover that the 4-by-4 magic square is singular and that the linear combination of its columns

```
A(:,1) + 3*A(:,2) - 3*A(:,3) - A(:,4)
```

is the zero vector.

1.20. Permuting the rows and columns of a magic square preserves the row sums, the column sums, and the matrix rank, but not the diagonal sums. (Symmetric permutations, `A = A(p,p)`, also preserves the diagonal sums.)

1.21. On old teletypes, ASCII character 7 rang a bell. On modern computer terminals, `disp(char(7))` in MATLAB usually produces a dull thud, but the sound can be changed with the operating system.

1.22. The output produced by `display(char([169 174]))` depends upon the font being used in the command window. With most fonts, the command displays the copyright symbol, ©, and the registered symbol, ®.

1.23. Fundamental physical law.

```
crypto(s)
= A rolling stone gathers Momentum
```

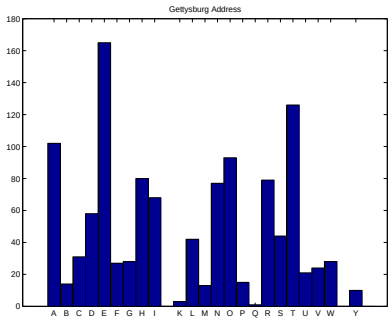
(Another question: why did I choose to capitalize the `M`?)

1.24. Run `crypto` on the Gettysburg address.

```
encrypt gettysburg.txt tempfile.txt
type tempfile.txt
encrypt tempfile.txt
delete tempfile.txt
encrypt encrypt.m
```

1.25. Analyze the text of the Gettysburg address.
See `gettysburg.m`.

```
nchar   = 1463
uniq    = ,-.BFGINTWabcdefghiklmnopqrstuvwy
nuniq   = 35
nblank  = 255
nperiod = 10
ncomma  = 19
ndash   = 7
most    = E
missing = JXZ
```



1.26. Two-character strings that `crypto` does not change. The experiment

```
for i = 0:94
   x = char(32+i);
   for j = 0:94
      y = char(32+j);
      if isequal(crypto([x y]),[x y])
         disp([x y])
      end
   end
end
```

shows that each character `x` has a conspiring character `y` so that `crypto([x y])` is equal to `[x y]`. It turns out that `[1; 62]` is a mod 97 eigenvector of `A`. Consequently, the list of invariant pairs is generated by

```
x = char(32:126)';
y = char(mod(62*(double(x)-32),97)+32);
[x y]
```

Some of the character pairs are: `'1t'`, `'CD'`, `'SZ'`, `'Uu'`, `'Vr'`.

1.27. It's easy to find other matrices with `mod(A*A,97) == I`. Here's a brute force approach

```
A = [];
while ~isequal(mod(A*A,97),eye(2,2))
   A = ceil(100*rand(2,2));
end
A
```

One such matrix is `A = [95  96; 100 99]`.

1.28. `crypto` works with the characters `char((0:96)+32)`. This would include `char(127)` and `char(128)`. The first of these is nonprinting and the second is often nonprinting. So, two other characters from the extended ASCII character set are chosen to replace these two. The remaing extended characters are mapped into characters below ASCII 127.

1.29. `crypto` with just 29 characters.
See `crypto29.m`

1.30. $3n + 1$ sequence for $n = 5, 10, 20, 40, \ldots$
The sequence generated by $n = 2^p \cdot 5$ is

$$n, n/2, n/4, \cdots, 10, 5, 16, 8, 4, 2, 1$$

The plot is always decreasing, except for the one jump from 5 to 16.

1.31. $3n + 1$ sequence for $n = 108, 109$, and 110.
The first nine steps are

```
108   109   110
 54   328    55
 27   164   166
 82    82    83
 41    41   250
124   124   125
 62    62   376
 31    31   188
 94    94    94
```

After that, the sequences are the same. All three sequences have length 114.

1.32. $L(n)$ = length of $3n + 1$ sequence.
See `length3np1.m`

```
L = zeros(1,1000);
for n = 1:1000, L(n) = l3np1(n); end
plot(L)
n = find(L==max(L))
l3np1(n)
threenplus1(n)
```

The longest sequence has $n = 871$, $L(n) = 179$.

1.33. How many `floatgui` numbers?

```
text(.9*xmax,2,num2str((emax-emin+1)*2^t))
```

1.34. Explain output from

```
t = 0.1; n = 1:10; e = n/10 - n*t
```

There is one roundoff error when evaluating `n/10` and two when evaluating `n*(1/10)`. It turns out that

```
4*e/eps =  [0  0  -1  0  0  -2  -2  0  0  0]
```

This shows that the computed value of `n/10` is one bit less than `n*(1/10)` for `n = 3, 6,` and `7`. In these cases, the exact value $n/10$ falls between the two floating point values.

1.35. What does each of these programs do?

```
x = 1; while 1+x > 1, x = x/2, pause(.02), end
```

Exhibits roundoff. The program produces 53 lines of output. The last two values of `x` are `eps` and `eps/2`.

```
x = 1; while x+x > x, x = 2*x, pause(.02), end
```

Exhibits overflow. The program produces 1024 lines of output. The last two values of `x` are `2^1023` $\approx$ `realmax/2` and `Inf`.

```
x = 1; while x+x > x, x = x/2, pause(.02), end
```

Exhibits underflow. The program produces 1075 lines of output. The last two values of `x` are `eps*realmin` and `0`. (On computers without subnormal floating point numbers, this program would produce 1023 lines of output. The last two values would be `realmin` and `0`).

1.36. `format hex`

```
4059000000000000 = 100
3f847ae147ae147b = 1/100
3fe921fb54442d18 = pi/4
```

1.37. $\mathcal{F}$ is the set of all finite, normalized IEEE numbers.

(a) How many elements are there in F?
For each value of sign s and exponent e there are 2^{52} possible fractions. There are two possible values of s and, excluding the denorms and the NaN/Infs, $2^{11} - 2$ possible values of e. So the cardinality of $\mathcal{F}$ is
$2 \cdot (2^{11} - 2) \cdot 2^{52} = 18428729675200069632$.

(b) What fraction are $1 \le x < 2$?
(c) What fraction are $1/64 \le x < 1/32$?
Same fraction between any two consecutive powers of two, namely
$1/(2 \cdot (2^{11} - 2)) = 1/4092$.

(d) What fraction satisfy `x*(1/x) == 1`?

```
k = 0;
for n = 1:2^20
   x = rand;
   if x*(1/x) == 1
      k = k+1;
   end
end
k/2^20
```

Between `0.846` and `0.847`.

1.38. Quadratic formula.
With $b = -10^8$, you get few if any accurate digits out of $-b - \sqrt{b^2 - 4}$ unless you compute the intermediate results to very high precision. In MATLAB there is no trouble with

```
x1 = (10^8 + sqrt(10^6-4))/2 = 1.0000e+008
```

But

```
x2 = (10^8 - sqrt(10^16-4))/2 = 7.4506e-009
```

when it should be `1.0000e-008`. Clearly

```
x2 = 1/x1
```

works well in this situation. Alternatively,

```
roots([1 -10^8 1])
```

gives two good roots, `1.0000e+008` and `1.0000e-008`.

1.39. Power series for computing $\sin x$.
The loop test in `powersin` terminates when `s+t == t`, that is when `t` is so small compared to `s` that the computed value of `s+t` is equal to `s`.
Change the first line of `powersin.m` to

```
function [s,tmax,cnt] = powersin(x)
```

Insert these lines before the start of the while loop.

```
tmax = abs(t);
cnt = 0;
```

Insert these lines in the loop.

```
tmax = max(tmax,abs(t));
cnt = cnt+1;
```

Here is a table of `x`, `sin(x)-powersin(x)`, `tmax`, and `cnt`.

```
    pi/2          11*pi/2        21*pi/2        31*pi/2
 2.2204e-016   -2.1287e-010   -1.3324e-004   -5.8210e+003
 1.5708e+000    3.0665e+006    1.4673e+013    7.9890e+019
    11             37             60             79
```

We see that when the largest term is about 10^p, the computed value looses about p digits. The power series is OK for x less than $\pi/2$. But as x increases, the power series requires more work and yields less accuracy.

1.40. Steganography. See `steganall.m`. There are 16 images hidden in the default `cdata` for the MATLAB `image` function.

1.41. (a)

```
function S = spiral(n)
S = [];
for m = 1:n
   S = rot90(S,2);
   S(m,m) = 0;
   p = m^2-m+1;
   v = (m-1:-1:0);
   S(:,m) = p-v';
   S(m,:) = p+v;
end
if mod(n,2)==1
   S = rot90(S,2);
end
```

(b) Half of the diagonals of `spiral(n)` contain even numbers.
(c) `S = spiral(2*n)`
`j = 1:n-2`, `S(n+1,1:n-2)` is divisible by `n-j`
`j = n+2:2*n`, `S(n-1,n+2:end)` is divisible by `j-n`
(d) For `n = 17` and `n = 41`, `primespiral(n,n)` has `n-1` primes on the main diagonal.
(e) `primespiral(n,6)` has 9 primes on an anti-diagonal.

1.42. See `trinumspiral.m`

1.43. Length of roman numerals. Eg. 88 = 'LXXXVIII', so $f(88) = 8$.

1.44. (a) On which day of the week were you born?

```
birthday = datenum([year,month,day])
datestr(birthday,8)
```

(b) Which week day is the most likely for your birthday?

```
cnt = zeros(1,7);
for y = 2000:2399
   w = weekday(datenum([y,month,day]));
   cnt(w) = cnt(w)+1;
end
cnt
bar(cnt)
set(gca,'xticklabel',{'Su','M','Tu','W','Th','F','Sa'})
```

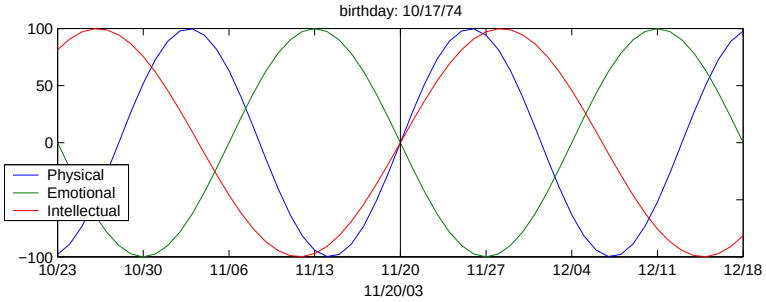
(c) What is the probability of Friday the 13th.

```
cnt = 0;
for y = 1:400
   for m = 1:12
      d = datenum([y,m,13]);
      if weekday(d) == 6
         cnt = cnt+1;
      end
   end
end

cnt = 688
cnt/4800 = 43/300 = .143333
```

1.45. Biorhythms.
See `biorhythm.m` and `biorhythmzero.m`.
The biorhythm period is

```
lcm(lcm(23,28),33) = 21252
```

The first time all of your cycles return to zero simultaneously is halfway through this period, that is 10626 days or 29 years, one month and three days after birth. Here is the biorhythm in November, 2003, for someone born in October, 1974.



Chapter 2

Linear Equations

Exercises

2.1. Buying fruit.

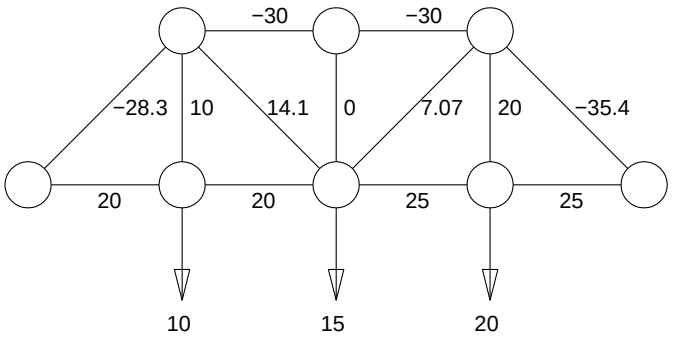
```
A = [3 12 1; 12 0 2; 0 2 3]
b = [2.36; 5.26; 2.77]
format bank
x = A\b

x =
    0.29
    0.05
    0.89
```

2.2. Reduced row echelon form of the magic square of order six.

```
rref(magic(6))

    1     0     0     0     0    -2
    0     1     0     0     0    -2
    0     0     1     0     0     1
    0     0     0     1     0     2
    0     0     0     0     1     2
    0     0     0     0     0     0
```

2.3. See `mytruss.m`.

```
x = [-28.2843, 20.0000, 10.0000, -30.0000,  14.1421, 20.0000,
  0, -30.0000,  7.0711, 25.0000,  20.0000, -35.3553, 25.0000]
```



2.4. A small network of resistors. See `circuit.m`

2.5. (a) Which are positive definite?

`M = magic(n)`	`no`
`H = hilb(n)`	`yes`
`P = pascal(n)`	`yes`
`I = eye(n,n)`	`yes`
`R = randn(n,n)`	`no`
`R = randn(n,n); A = R' * R`	`yes`
`R = randn(n,n); A = R' + R`	`no`

Matrices generated by

```
R = randn(n,n); A = R' + R + n*eye(n,n);
```

may or may not be positive definite, depending upon the specific random matrix generated.
(b) See `mychol.m`.

2.6. A badly conditioned matrix that does not produce small pivots in Gaussian elimination.

```
A = eye(n,n) - triu(ones(n,n),1)
```

Let $X = A^{-1}$.
`sum(abs(A))` is `1:n`, so $\|A\|_1 = n$.
`sum(abs(X))` is `2.^(0:n-1)`, so $\|X\|_1 = 2^{n-1}$.
$\kappa_1(A) = \|A\|_1\|X\|_1 = n2^{n-1}$.
For $n = 48$, $n2^{n-1} >$ `1/eps`.
Near null vector: `z = X(:,n), norm(z,1) == 2^(n-1), norm(A*z,1) == 1`
In Gaussian elimination with partial pivoting, all pivots = 1.

A pivot strategy that will produce smaller pivots than partial pivoting: At the `k`-th stage, choose `A(k,k+1)`. That's just to the right of the highlighted pivot. Produces `U(n,n) = 1/2^(n-2)`.

2.7. Determinant. See `lutxsig.m` and `mydet.m`.

```
function [L,U,p,sig] = lutxsig(A)
....
p = (1:n)';
sig = 1;
for k = 1:n-1
   ....
     % Swap pivot row
       if (m ~= k)
          sig = -sig;
          ....
       end
....
end

function d = mydet(A)
[L,U,p,sig] = lutxsig(A);
d = sig*prod(diag(U))
```

2.8. Explicit `for` loops. See `lutxloops.m`.
On a 1.4 GHz Pentium M laptop running MATLAB 6.5, each of the following reports an elapsed time of about 5 seconds.

```
n = 1920, A = randn(n,n); tic, lu(A); toc

feature accel on, feature jit on
n =  528, A = randn(n,n); tic, lutx(A); toc
n =  525, A = randn(n,n); tic, lutxloops(A); toc

feature accel off, feature jit off
n =  528, A = randn(n,n); tic, lutx(A); toc
n =  130, A = randn(n,n); tic, lutxloops(A); toc
```

2.9. Singular system.

```
A = [1 2 3; 4 5 6; 7 8 9];  b = [1; 3; 5];
```

(a) `p = [1/3; 1/3; 0]` is a particular solution.
`z = null(A,'r') = [1; -2; 1]` is a null vector.
For a free parameter `t`, `x = p + t*z` is the general solution.
(b) Elimination with exact arithmetic would produce $U_{3,3} = 0$ and $c_3 = 0$, so back substitution would start with $x_3 = 0/0$. This arbitary value corresponds to the free parameter t.

(c) `x = bslashtx(A,b) = [13/3; -23/3; 4]`
`dbstop in bslashtx/backsubs` shows back substitution begins with
`x(3) = y(3)/U(3,3) = (2/eps)/(eps/2) = 4`
This is a good solution because the residual, `r = b - A*x`, is small. This is a bad solution because `bslashtx` does not give any warning that the solution is not unique, and that the error, `e = x - inv(A)*b`, is not even defined.
(d) `x = A\b = [17/6; -14/3; 5/2]`
The builtin backslash operator does the arithmetic in a different order, using a column-oriented algorithm where `bslashtx` uses a row oriented algorithm. The function `bslashtx2` from the next exercise uses the same algorithm as the builtin backslash. The back substitution starts with
`x(3) = x(3)/U(3,3) = (5/4*eps)/(eps/2) = 5/2`
The different values of `x(3)` obtained by the different algorithms correspond to different values of the free parameter `t` in the theoretical general solution.

2.10. See `bslashtx2.m`. The `forward` and `backsubs` subfunctions in `bslashtx` use inner product, row-oriented algorithms to solve triangular systems. In `bslash2tx` replaced by the following column-oriented subfunctions.

```
function x = forward(L,x)
[n,n] = size(L);
for k = 1:n
   x(k) = x(k)/L(k,k);
   i = k+1:n;
   x(i) = x(i) - L(i,k)*x(k);
end

function x = backsubs(U,x)
[n,n] = size(U);
for k = n:-1:1
   x(k) = x(k)/U(k,k);
   i = 1:k-1;
   x(i) = x(i) - U(i,k)*x(k);
end
```

2.11. See `myinv.m`

2.12. See `lutx2.m`

```
function [L,U,p] = lutx2(A)
%LUTX2  Triangular factorization, textbook version.
%   With three output arguments, [L,U,p] = LUTX2(A) produces
%   a unit lower triangular matrix L, an upper triangular
%   matrix U, and a permutation vector p, so that L*U = A(p,:)
%
%   With two output arguments, [L,U] = LUTX2(A) produces a
%   "psychologically lower triangular matrix" L (i.e. a product
```

```
%   of lower triangular and permutation matrices), and an
%   upper triangular U so that L*U = A.
%
%   With one output argument, LUTX2(A) returns a single matrix
%   containing L-I+U.  The pivot information is lost.
```

The code for `lutx2.m` is the same as `lutx.m`, except the final section,

```
% Separate result
if nargout == 3
   L = tril(A,-1) + eye(n,n);
elseif nargout == 2
   L(p,:) = tril(A,-1) + eye(n,n);
else
   L = A;
end
U = triu(A);
```

2.13. (a,b) `lugui(magic(8))` is interesting because the rank of the matrix is three. There are only three independent rows. After three elimination steps in exact arithmetic, the remaining matrix should be all zero.
(c) Using `lugui`, pick `A(8,5) = 4` and then `A(4,4) = -8` as the first two pivots. Because the pivots are powers of two, there is no roundoff error. The elements of the remaining matrix are zero and `130` exactly. Any of the `130`'s can be picked as the third pivot.

2.14. See `lupiv.m`.

2.15. (a) See `golubcond.m`.
`condest(golub(n))` appears to grow exponentially, like 64^n.
(b) With diagonal pivoting, the pivots are all equal to one. There is no indication of the bad conditioning.
(c) `det(G) = det(L)*det(U) = 1`

2.16. Pascal matrices.
(a)

```
P = pascal(n);
for j = 1:n
   for k = 1:n
      P(k,j) = nchoosek(k+j-2,max(j,k)-1);
   end
end
```

(b)

```
R = chol(P);
for j = 1:n
```

```
   for k = 1:j
      R(k,j) = P(k,j-k+1);
   end
end
```

(c) $\kappa(P_n) \approx c\rho^n$, where $c = 0.0181, \rho = 14.65$.
(d) `det(P) == det(R')*det(R) = 1*1 = 1.`
(e) `Q = P; Q(n,n) = Q(n,n) - 1;`
`R = chol(Q)` is equal to `chol(P)` except `R(n,n) = 0`
(f) `det(Q) = det(R')*det(R) = 0.`

2.17. Here are the scores obtained in "Pivot Pickin' Golf" by the automatic pivot strategies.

	diagonal	partial	complete
magic	Inf	3284.00	459759.75
testmats	Inf	9228.50	2195.00
rand #1	Inf	31.00	38.00
rand #2	Inf	37.00	28.50
rand #3	83.00	43.00	43.00

Here is a winning strategy for the `testmats` course:

```
1. pick subdiagonals
2. partial pivoting
3. partial pivoting
4. diagonal pivoting
5. pick 1024, then diagonal pivoting
6. diagonal pivoting
7. diagonal pivoting
8. complete pivoting
9. diagonal pivoting
```

2.18. See `myrandncond.m`. `condest(randn(n,n))` grows like `n^(3/2)`.

2.19. Tridiagonal system.

```
n = 100;
e = ones(n,1);
b = (1:n)';

A = 2*diag(e) - diag(e(1:n-1),-1) - diag(e(1:n-1),1);
xa = bslashtx(A,b);

A = spdiags([-e 2*e -e],[-1 0 1],n,n);
xb = A\b;

xc = tridisolve(-e,2*e,-e,b);
```

```
condest(A)
ans =
  5.1000e+003
```

2.20. This problem is open ended. Do something like

```
[U,G] = surfer('http://my.favorite.url',n)
spy(G)
pagerank(U,G)
```

and then comment on the results.

2.21. If `U` is a cell array of URL's and `k` is a single integer,
`U{k}` is a string, the `k`-th URL.
`U(k)` is a 1-by-1 cell array whose only element is `U{k}`.
`G(k,:)` is nonzero for outgoing nodes.
`G(:,k)` is nonzero for incoming nodes.
`U(G(k,:))` is a list of the URLs for outgoing nodes.
`U(G(:,k))` is a list of the URLs for incoming nodes.

2.22. Cliques in the `harvard500` Web connectivity matrix.
`U(168:180)`: Harvard Divinity School
`U(229:248)`: Radcliffe Institute
`U(261:281)`: Dana-Farber Cancer Institute
`U(315:335)`: "Go Crimson", Harvard's athletic program

2.23. (a) For `p >= 8`, `nnz(G^p) = 167985`.
(b) `nnz(G^8)/prod(size(G)) = 0.6719`.
(c) `for p = 1:9, subplot(3,3,p), spy(G^p), end`
(d) The "Go Crimson" athletic program, nodes `46` and `315:335`, has no links to the other pages in the data set.

2.24. Duplicate `hashfuns`.

```
function h = hashfun(url)
% Almost unique numeric hash code for pages already visited.
h = length(url) + 1024*sum(url);
```

Main program. A link starts with `'="http:'` and ends with the next double quote.

```
page = urlread('http://www.mathworks.com');
for f = findstr('="http:',page);
   e = min(findstr('"',page(f+2:end)));
   t = deblank(page(f+2:f+e));
   t(t<' ') = '!';   % Nonprintable characters
   if t(end) == '/', t(end) = []; end
   disp(t)
```

```
   disp(hashfun(t))
end
```

The output includes

```
http://www.mathworks.fr
     2311191
http://www.mathworks.es
     2311191
```

Since `char('f')+char('r') == char('e')+char('s')` and otherwise the two urls are the same, they have the same hash function value.

2.25. Disconnected miniweb.

```
G =
     0     0     0     1     0     0
     1     0     0     0     0     0
     1     1     0     0     0     0
     0     1     1     0     0     0
     0     0     0     0     0     1
     0     0     0     0     1     0

pagerank1(G,.85) =
    0.1981
    0.1092
    0.1556
    0.2037
    0.1667
    0.1667
```

What happens to page rank as $p \to 1$? Two possible answers here. The intuitive answer is that the graph has two disconnected subgraphs and consequently the Markov stationary probabilities are not unique. The direct solution algorithm used in `pagerank` certainly breaks down if `p = 1`. However, a second answer is that `pageranksym`, a symbolic version of `pagerank1`, produces

```
p = sym('p');
pagerank1(G,p) =
   [ 1/3*(p^3+3*p^2+2*p+2)/(p^3+4*p^2+4*p+4)]
   [         1/3*(p^2+p+2)/(p^3+4*p^2+4*p+4)]
   [ 1/6*(p^3+3*p^2+4*p+4)/(p^3+4*p^2+4*p+4)]
   [ 1/6*(p^3+5*p^2+6*p+4)/(p^3+4*p^2+4*p+4)]
   [                                    1/6]
   [                                    1/6]
```

and

```
limit(ans,p,1) =
   [ 8/39]
   [ 4/39]
   [ 6/39]
   [ 8/39]
   [  1/6]
   [  1/6]
```

These values are 2/3 times the limiting values for the 4-by-4 subgraph and 1/3 times the values for the 2-by-2 subgraph.

2.26. Alternative page rank algorithms. See `pagerank1.m` for direct solution of the sparse equations, `pagerank2.m` for inverse iteration, and `pagerank3.m` for the power method.

2.27. Implement the power method page rank algorithm in some other language.

Chapter 3

Interpolation

Exercises

3.1. Reproduce figure with four interpolants. See `interps.m`.

3.2. Tom and Ben. See `twins.m`.

3.3. (a) See `deceptive.m` and figure 3.1.
(b) Values at -0.3. The values from `piecelin` and `pchip` follow the overall trend of the data.

```
plin =  0.4300
poly = -0.9990
spl =  -0.1957
pch =   0.4322
```

(c)

```
V = vander(x);
c = V\y
c = round(c)
p = poly2sym(c')

p =
16*x^5-20*x^3+5*x
```

The data comes from the Chebyshev polynomial $T_5(x)$. In an sense, the value from `polyinterp` is the "correct" result.

3.4. Make a plot of your hand. See `myhand.dat` and `handinterp.m`.

3.5. Use polar coordinates. See `handinterp.m` and figure 3.2. As a function of θ, the distance from the base of the palm to the tips of the fingers varies

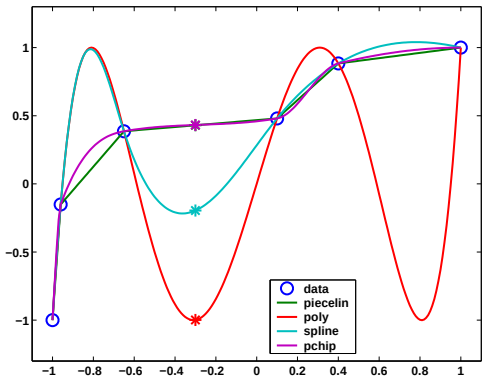


Figure 3.1. *Deceptive data for interpolation*

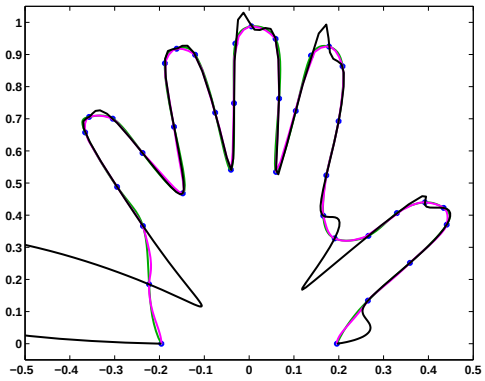


Figure 3.2. *Hand with polar coordinates.*

rapidly. When sampled at only a few points, `pchip` does pretty well with this function, but `spline` has a terrible time.

3.6. (a) `vandal(n)` generates a symbolic Vandermonde matrix and then uses Gaussian elimination to compute its determinant.
(b) If `V = vander(x)`,

$$\det(V) = \prod_{i<j} (x_i - x_j)$$

This shows that the matrix `vander(x)` is nonsingular if and only if the elements of `x` are distinct.

3.7. Here are two proofs that the interpolating polynomial is unique.
(1) From the previous exercise, if the abscissae are distinct, then the Vandermonde matrix is nonsingular and hence the coefficients of the interpolating

polynomial are unique.
(2) If $P(x)$ and $Q(x)$ are both interpolating polynomials, then $P(x)-Q(x)$ is a polynomial of degree less than n that vanishes at n points. Hence $P(x)-Q(x)$ must be identically zero.

3.8. For $T_5(x)$, see `cheby5.m`.

3.9. See `myrungeinterp.m` and follow the bouncing asterisk. Experimentally, it looks like $P_n(x) \to F(x)$ for for $x < \sqrt{2}/2$. Changing to interpolation at the zeros of the Chebyshev polynomial $T_n(x)$ gives convergence over the entire interval $-1 \le x \le 1$.

3.10. To do piecewise quadratic interpolation, you need three conditions to determine one piece. There are two interpolation conditions, but it is not clear what the third condition should be.

3.11. See `myspline.m` and `mypchip.m`.
Change the first line of both `splinetx` and `pchiptx`.

```
function [v,p] = pchiptx(x,y,u)
```

Add this line to the help entries.

```
%  [v,p] = pchip(x,y,u) also returns p(k) = P'(u(k)).
```

Add this line to the main functions.

```
p = d(k) + s.*(2*c(k) + 3*s.*b(k));
```

3.12. See `myspline.m` and `mypchip.m`.

3.13. See `perspline.m` and `perpchip.m`.

3.14. See `splinecond.m`. In `splinetx`, change

```
d = tridisolve(a,b,c,r);#
```

to

```
T = diag(a,-1) + diag(b,0) + diag(c,1);
condest(T)
d = r;
d(:) = T\r(:);
```

Put three data points close together.

```
x = [1 2 2.999 3 3.001 4 5]
y = [16 18 20.99 21 21.01 15 12];
```

The estimated condtion is `6.9947e+003`.

3.15. See `mypchipavg.m`.

3.16. (a) `interpgui(1-x.^2)`. For a second degree polynomial, `spline` and `polyinterp` produce the same curve.
(b) `interpgui(1-x.^4)` None of the plots overlap, although `polyinterp` and `spline` are within 10^{-3}, so they look like they overlap.

3.17. `interpgui(4)`. The spline through four points is a single cubic, so `spline` and `polyinterp` produce the same curve.

3.18. (a) The Vandermonde matrix is very badly conditioned.
`cond(vander(1900:10:2000)) = 3.0562e+48`
(b) What does the check box about centering and scaling do? The check box replaces `x` by `(x - mean(x))/std(x)`.
(c) The function `F(s) = condest(vander((-50:10:50)/s))` is minimized at `s = 42.6` where `F(s) = 1.3e4`.
`sigma = std(-50:10:50) = 33.17` and `F(sigma) = 3.3e4`. So `sigma` is not the optimum scaling, but it's not too bad.

Chapter 4

Zeros and Roots

Exercises

4.1. `fzerogui('x^3-2*x-5',[0,3])`
Easy problem. Converges to `x = 2.09455148154233` in 7 steps.

`fzerogui('sin(x)',[1,4])`
Easy problem. Converges to `x = pi` in 7 steps, all secant.

`fzerogui('x^3-.001',[-1,1])`
Moderately difficult. There is only one real root, but there are two nearby complex roots. Requires 15 steps to converge to `x = 1/10`.

`fzerogui('log(x+2/3)',[0,1])`
Easy problem. Converges to `x = 1/3` in 6 steps.

`fzerogui('sign(x-2)*sqrt(abs(x-2))',[1,4])`
This is the "perverse" example where Newton's method fails. $f'(x)$ is unbounded. `fzero` uses secant for all its steps. Slow convergence, only about half a decimal digit per step. Converges to `x = 2` in 32 steps.

`fzerogui('atan(x)-pi/3',[0,5])`
Easy problem. Converges to `x = sqrt(3)` in 8 steps.

`fzerogui('1/(x-pi)',[0,5])`
Sign change is a pole, not a zero. Take over 50 steps towards `x = pi`. Eventually divides by zero and generates an error in the plot scaling.

4.2. (a)

```
>> syms x
>> f = x^3 - 2*x - 5;
>> z = solve(f)
    <messy symbolic expressions>
```

```
>> z(1)
   ans =
   1/6*(540+12*1929^(1/2))^(1/3)+4/(540+12*1929^(1/2))^(1/3)
>> length(char(z))
   ans =
      340
>> double(z)
   2.09455148154233
  -1.04727574077116 + 1.13593988908893i
  -1.04727574077116 - 1.13593988908893i
```

(b)

```
>> p = [1 0 -2 -5]
p =
     1     0    -2    -5
>> roots(p)
ans =
  2.09455148154233
 -1.04727574077116 + 1.13593988908893i
 -1.04727574077116 - 1.13593988908893i
```

(c)

```
>> F = inline(char(f));
>> fzerotx(F,[2,3])
ans =
   2.09455148154233
```

(d)

```
>> Fp = inline(char(diff(f)));
Fp =
     Inline function:
     Fp(x) = 3.*x.^2-2
>> x = 1i;
>> x = x - F(x)/Fp(x)
x =
 -1.00000000000000 + 0.40000000000000i
>> x = x - F(x)/Fp(x)
x =
 -0.56274873971876 + 1.77192889360573i
```

Use uparrow to iterate .....

```
>> x = x - F(x)/Fp(x)
x =
 -1.04727574077116 + 1.13593988908893i
```

(e) Can bisection be used to find the complex root? No. There is no notion of sign change or positive/negative for complex numbers.

4.3. $p(x) = 816x^3 - 3835x^2 + 6000x - 3125$
(a) What are the exact roots of p?

```
>> p = poly2sym([816 -3835 6000 -3125])
p = 816*x^3-3835*x^2+6000*x-3125

>> factor(p)
ans = (16*x-25)*(17*x-25)*(3*x-5)

>> z = solve(p)
z =
[ 25/15]
[ 25/16]
[ 25/17]
```

(b)

```
>> p = inline(char(p));
>> ezplot(p,1.43,1.71)
>> hold on, plot(double(z),zeros(3,1),'o')
```

(c)

```
>> x = 1.5
>> x = x - (816*x^3-3835*x^2+6000*x-3125)/(2448*x^2-7670*x+6000)
```

Use up arrow to iterate. Converges easily to the nearest root,
`x = 1.47058823529416 = 25/17`

(d) Starting with $x_0 = 1$ and $x_1 = 2$, the secant method converges to $1.6666666666666666 = 25/15$.

(e) Starting with the interval $[1, 2]$, what does bisection do? The first step reduces the interval to $[1, 1.5]$, which contains only one root. Consequently, converges to $x = 1.47... = 25/17$.

(f) What is `fzerotx(p,[1 2])`? Why? The initial secant step happens to be to $x = 1.69...$, which is near the root at $25/15$. `fzerotx` then takes 10 steps, 7 with IQI, to converge to $25/15$. The interval `[a,b]` always includes all three roots.

Note that none of these methods found the "middle" root, $25/16$.

4.4. The convergence test in `fzerotx` is

```
m = 0.5*(a - b);
tol = 2.0*eps*max(abs(b),1.0);
```

```
if (abs(m) <= tol) | (fb == 0.0)
   break
end
```

This says that we have luckily found a `b` for which `f(b)` is exactly zero, or the length of the interval, `abs(b-a)`, is roundoff error in `b` or `1`. Note this is a relative error test if `b` is larger than 1, but an absolute error test if `b` is less than 1.

4.5. In `fzerotx` interpolation is done by the secant method if `c` is equal to `a` or by inverse quadratic interpolation if `a`, `b`, and `c` are distict. Interpolation is acceptable if it leads to a point within the interval `[a,b]` that is not too close (using the quantity `tol`) to the end-points.

4.6. See `iqi.m`. Lagrange formula for inverse quadratic interpolation is:

$$x = \frac{(0-f_b)(0-f_c)}{(f_a-f_b)(f_a-f_c)}a + \frac{(0-f_a)(0-f_c)}{(f_b-f_a)(f_b-f_c)}b + \frac{(0-f_a)(0-f_b)}{(f_c-f_a)(f_c-f_b)}c$$

Cramer's rule for inverse quadratic interpolation is:

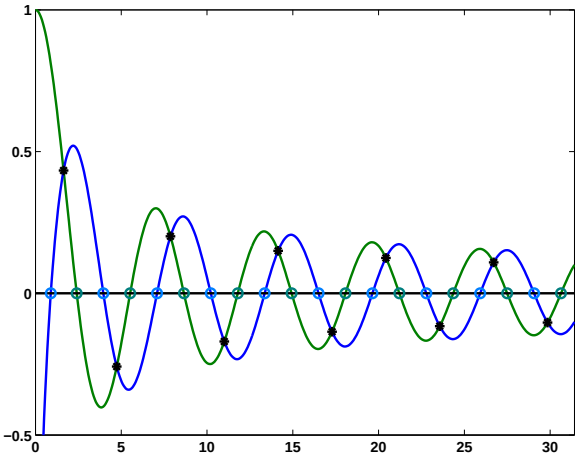
$$x = \frac{\det\begin{pmatrix} f_a^2 & f_a & a \\ f_b^2 & f_b & b \\ f_c^2 & f_c & c \end{pmatrix}}{\det\begin{pmatrix} f_a^2 & f_a & 1 \\ f_b^2 & f_b & 1 \\ f_c^2 & f_c & 1 \end{pmatrix}}$$

The IQI portion of `fzerotx` is:

```
m = (a - b)/2;
s = fb/fc;
q = fc/fa;
r = fb/fa;
p = s*(2*m*q*(q - r) - (b - c)*(r - 1));
q = (q - 1)*(r - 1)*(s - 1);
p = -p;
d = p/q;
x = b + d;
```

See `iqi.m` for code that uses the Symbolic Toolbox to verify that all three x's are equal. The computations in `fzerotx` are arranged to avoid unnecessary underflow and overflow, particularly if IQI is not used.

4.7. `z = fzerotx(@besselj,[0 pi],0)` tries to find a zero of `besselj(nu,x)` considered as a function of its first argument, `nu`, and with its second argument, `x`, set to zero. In other words, it is trying to find a zero of $J_x(0)$, not $J_0(x)$. Since `besselj(pi,0) = 0`, `fzerotx` exits immediately with `b = pi`.



4.8. Secant method on $\text{sign}(x-a)\sqrt{|x-a|}$. This is the "perverse" example where Newton's method fails. $f'(x)$ is unbounded. Converge is roughly linear, with the interval length is reduced by an irregular, sign-changing factor between 0.1 and 0.5 each step. Converges to `x = 2` in 32 steps.

4.9. First ten $x = \tan x$.

```
for k = 1:10
   z(k) = fzerotx('tan(x)-x',[k k+1/2-k*eps]*pi);
end

z
  = 4.4934  7.7253 10.9041 14.0662 ... 29.8116 32.9564

z/pi
  = 1.4303  2.4590  3.4709  4.4774 ...  9.4893 10.4903
```

4.10. See `bessel10.m`.

4.11. (a) What is the largest value of n for which $\Gamma(n+1)$ and $n!$ are exactly represented.

```
vpa('22!') - prod(1:22)
= 0

vpa('23!') - prod(1:23)
= nonzero
```

Thus, n = 22 is the largest integer for which n! is exact in double.

(b) What is the largest value of n for which $\Gamma(n+1)$ and $n!$ do not overflow?

```
gamma(171) = 7.2574e+306
gamma(172) = Inf
```

4.12. (a) What is the relative error in Stirling's approximation and in Gosper's approximation when $x = 2$?

```
x = 2;
s = gammaln(x+1)/(x*log(x) - x + log(2*pi*x)/2)
g = gammaln(x+1)/(x*log(x) - x + log(2*pi*x+pi/3)/2)

s =
    1.0634
g =
    1.0019
```

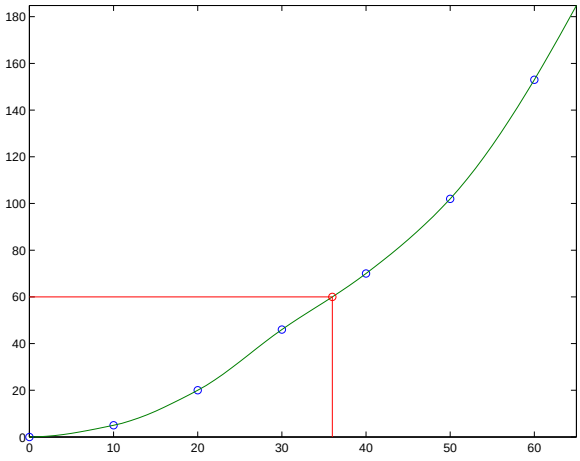
(b) How large must x be for Stirling's approximation and for Gosper's approximation to have a relative error less than 10^{-6}?

```
S = inline( ...
  'gammaln(x+1)./(x.*log(x) - x + log(2*pi*x)/2) - 1.000001')
G = inline( ...
  'gammaln(x+1)./(x.*log(x) - x + log(2*pi*x+pi/3)/2) - 1.000001')

fzerotx(S,[2,200])
ans =
  144.4059

fzerotx(G,[2,100])
ans =
   15.2927
```

4.13. See `gammalninv.m`.

4.14. What is the speed limit for this vehicle?

```
v = (0:10:60)'
d = [0 5 20 46 70 102 153]'
[v d]
eta = 60
x1 = piecelin(d,v,eta)
x2 = fzerotx(inline('pchiptx(xk,yk,x)-y', ...
     'x','y','xk','yk'),[30 40],eta,v,d)
x3 = pchiptx(d,v,eta)
x4 = fzerotx(inline('splinetx(xk,yk,x)-y', ...
     'x','y','xk','yk'),[30 40],eta,v,d)
x5 = splinetx(d,v,eta)
u = (0:65)';
```




```
plot(v,d,'o',u,pchiptx(v,d,u),'-',x2,eta,'ro', ...
   [x2 x2],[0,eta],'r-',[0 x2],[eta eta],'r-')
axis tight

x1 =
  35.83333333333334
x2 =
  36.00066760428985
x3 =
  35.98756534518393
x4 =
  35.86433220451173
x5 =
  36.00342638805324
```

4.15. Kepler's equation for orbit eccentric anomaly.
(a)

```
M = 24.851090;
e = 0.1;
F = inline('E - e*sin(E) - M','E','M','e')
E = fzerotx(F,[0,2*M],M,e)
```

produces

```
E = 24.8204
```

(b)

```
M = 24.851090;
e = 0.1;
```

```
E = M;
m = 1;
Esave = 0;
while E ~= Esave
   Esave = E;
   E = E + 2*besselj(m,m*e)*sin(m*M)/m;
   m = m+1;
end
m
E
```

produces

```
m = 16
E = 24.8204
```

4.16. Freezing water mains.

```
function T = pipetemp(x)
% Temperature of water main at depth of x meters after 60 days.
Ti = 20;
Ts = -15;
alpha = 0.138e-6;
t = 60*24*60*60;  % 60 days * (24*60*60) secs/day
c = 2*sqrt(alpha*t);
T = Ts + (Ti - Ts)*erf(x/c);

ezplot(@pipetemp,[0 2])
fzerotx(@pipetemp,[0 2])

ans =
   0.6770
```

4.17. See `fmintrace.m`.

```
F = inline('-humps(x)');
ezplot(F,-1,2);
axis([-1 2 -115 15])
hold on
fmintrace(F,-1,2,1.e-4)
hold off
```

4.18. The minimizer of

$$\begin{aligned} f(x) &= 9x^2 - 6x + 2 \\ &= 9(x - 1/3)^2 + 1 \end{aligned}$$

is $x = 1/3$. Even with zero tolerance, `fmintx(f,0,1,0)` produces `x = 0.3333333342261373` because all floating point numbers `x` in the interval of length `sqrt(eps)` about `1/3` produce `f(x) = 1`.

4.19. `fmintx(@cos,2,4,eps)` only gets to within about `1.7e-10` of the exact minimizer at π because of the quadratic natural of $\cos(x)$ near $x = \pi$. `fmintx(@cos,0,2*pi)` luckily hits `pi` exactly because the initial interval is symmetric about the minimizer. `fmintx` does two golden section steps that preserve the symmetry, then a symmetric parabolic step.

4.20. Even with `tol = 0`, `fmintx(@F,a,b,tol)` terminates in finite time. The convergence test is `while abs(x-xm) > tol`. With roundoff error, `xm` is eventually exactly equal to `x`.

4.21. Derive the formulas used in `fmintx` for minimization by parabolic interpolation. We can assume without loss of generality that `x = 0` and `fx = 0`. Then the algorithm is

```
r = w*fv;
q = v*fw;
p = w*r-v*q;
s = 2*(q-r);
if s > 0.0, p = -p; end
s = abs(s);
% Is parabolic interpolation acceptable?
para = ( (abs(p)<abs(0.5*s*e)) & (p>s*a) & (p<s*b) );
if para
   e = d;
   d = p/s;
   xnew = d;
end
```

The coefficients a and b of the parabola $P(x) = ax^2 + bx$ that interpolates the three points $(0, 0)$, (v, f_v), and (w, f_w) are the solution to the 2-by-2 Vandermonde system

$$\begin{pmatrix} v^2 & v \\ w^2 & w \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} f_v \\ f_w \end{pmatrix}$$

The next iterate is the solution to $P'(x) = 0$, which $-b/(2a)$. By Cramer's rule this is

$$-\frac{1}{2}\det \begin{pmatrix} v^2 & f_v \\ w^2 & f_w \end{pmatrix} / \det \begin{pmatrix} f_v & v \\ f_w & w \end{pmatrix}$$

This is same as the `d` computed by `fmintx`.
The three inequalities defining the acceptability condition `para` amount to

$$a < x_{new} < b$$

and the length of the next step is less than half of the previous one.

The computations are organized to avoid underflow and overflow, particularly in quantities that will not be used if the parabolic step is not acceptable.

Chapter 5

Least Squares

Exercises

5.1. `[I,J] = ndgrid(1:n)`
`X = min(I,J) + 2*eye(n,n) - 2`
`R = chol(X)`
(a)

$$\kappa_1(R) = n2^{n-1}$$
$$\kappa_1(X) \approx (4/3n^2 - 4n + 1)4^{n-2}$$

(b) `R = chol(X)` and `[L,U] = lu(X)` have `R == L == U'`. The diagonals of `R`, `L` and `U` are all one and so do not reveal the poor conditionings.
`[Q,R] = qr(X)` produces an `R` whose diagonal shows some, but not all, of the poor conditioning.

5.2. See `censusoutlier.m`. The high degree polynomial fits are most affected by the outliner. The spline and exponential fits are also affected, but not as much. The extrapolated portion of the pchip fit is not affected at all.

5.3. See `censusdoomsday.m`. The degree 8 polynomial fit goes to zero on Sept. 21, 2013.

5.4.
```
x =
    9
    2
    6

sigma = norm(x);
u = x;
u(1) = u(1) + sigma;
rho = 1/(sigma*u(1));
```

```
I = eye(3,3);
H = I - rho*u*u';

format rat
H =
    -9/11        -2/11        -6/11
    -2/11        54/55        -3/55
    -6/11        -3/55        46/55

format short
H =
   -0.8182   -0.1818   -0.5455
   -0.1818    0.9818   -0.0545
   -0.5455   -0.0545    0.8364

u =
   20
    2
    6

H*u
  -20
   -2
   -6

tau = rho*u'*x;
v = x - (tau/2)*u
v =
   -1
   -1
    3

H*v =
   -1
   -1
    3
```

5.5. Householder reflections involve

$$\begin{aligned} \sigma &= \operatorname{sign}(x_k)\|x\| \\ u &= x + \sigma e_k \\ \rho &= 1/(\bar{\sigma} u_k) \end{aligned}$$

(a) Note that $\bar{\sigma} x_k$ is real.

$$\begin{aligned} u'u &= (x + \sigma e_k)'(x + \sigma e_k)' \\ &= x'x + \bar{\sigma} x_k + \bar{x}_k \sigma + \bar{\sigma}\sigma \end{aligned}$$

$$\begin{aligned} &= \bar{\sigma}\sigma + 2\bar{\sigma}x_k + \bar{\sigma}\sigma \\ &= 2\bar{\sigma}(x_k + \sigma) \\ &= 2\bar{\sigma}u_k \end{aligned}$$

Hence $\rho = 2/\|u\|^2 = 1/(\bar{\sigma}u_k)$

(b) Note that ρ is real, hence

$$\begin{aligned} H' &= I - (\rho uu')' \\ &= I - \rho uu' \\ &= H \end{aligned}$$

and

$$\begin{aligned} H'H &= (I - \rho uu')'(I - \rho uu') \\ &= I - 2\rho uu' + \rho^2 uu'uu' \\ &= I \end{aligned}$$

(c)

$$\begin{aligned} Hx &= x - \rho uu'x \\ &= x - \rho(x + \sigma e_k)(x' + \bar{\sigma}e_k)x \\ &= x - \rho(x'x + \bar{\sigma}x_k)(x + \sigma e_k) \\ &= x - (x + \sigma e_k) \\ &= -\sigma e_k \end{aligned}$$

(d) For any vector y, let $\tau = \rho u'y$. Then

$$\begin{aligned} Hy &= y - \rho uu'y \\ &= y - \tau u \end{aligned}$$

5.6. `X = reshape(1:15,3,5)'`

```
X =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15
```

(a)

```
rank(X)
     2

Z = pinv(X), B = X\eye(5,5), S = eye(3,3)/X

Z =
   -0.3889   -0.2444   -0.1000    0.0444    0.1889
   -0.0222   -0.0111   -0.0000    0.0111    0.0222
```

```
    0.3444    0.2222    0.1000   -0.0222   -0.1444

B =
   -0.4000   -0.2500   -0.1000    0.0500    0.2000
         0         0         0         0         0
    0.3333    0.2167    0.1000   -0.0167   -0.1333

S =
   -0.6111         0         0         0    0.1111
   -0.0278         0         0         0    0.0278
    0.5556         0         0         0   -0.0556
```

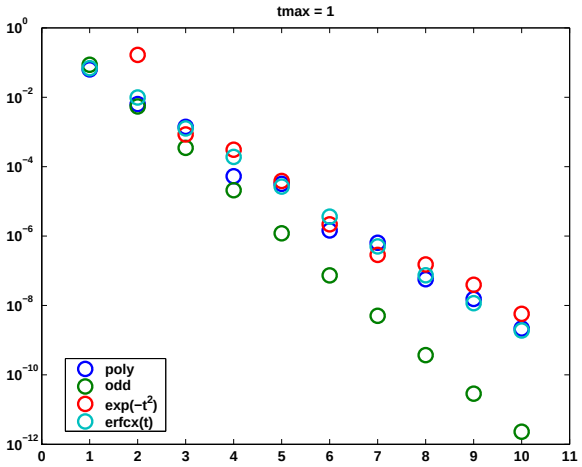
(b)

```
normf = inline('norm(X,''fro'')');
I = eye(5,5); E = eye(3,3);

[normf(Z)     normf(B)     normf(S)
 normf(X*Z-I) normf(X*B-I) normf(X*S-I)
 normf(Z*X-E) normf(B*X-E) normf(S*X-E)]

    0.6777    0.6791    0.8361
    1.7321    1.7321    2.1794
    1.0000    1.2247    1.0000
```

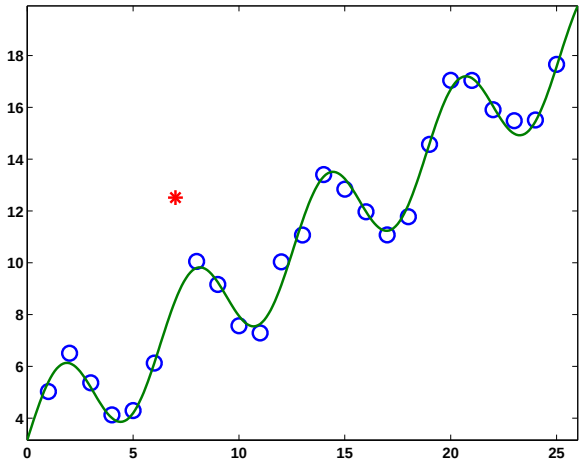
(c)

```
iseq = inline('norm(X-Y,''fro'') <= 200*eps')

[iseq(X*Z*X,X) iseq(Z*X*Z,Z) iseq(X*Z,(X*Z)') iseq(Z*X,(Z*X)')
 iseq(X*B*X,X) iseq(B*X*B,B) iseq(X*B,(X*B)') iseq(B*X,(B*X)')
 iseq(X*S*X,X) iseq(S*X*S,S) iseq(X*S,(X*S)') iseq(S*X,(S*X)')]

     1     1     1     1
     1     1     1     0
     1     1     0     1
```

5.7. Various least squares fits to erf(t)
See `fiterf.m`. I'm not satisfied with this exercise. I will probably eliminate part (c) from the final version of the book. Using the exp($-t^2$) weighting does not help very much. The graph shows the errors from four different fits. The fourth is based on exp(t^2)erfc(t), but that doesn't work very well either.

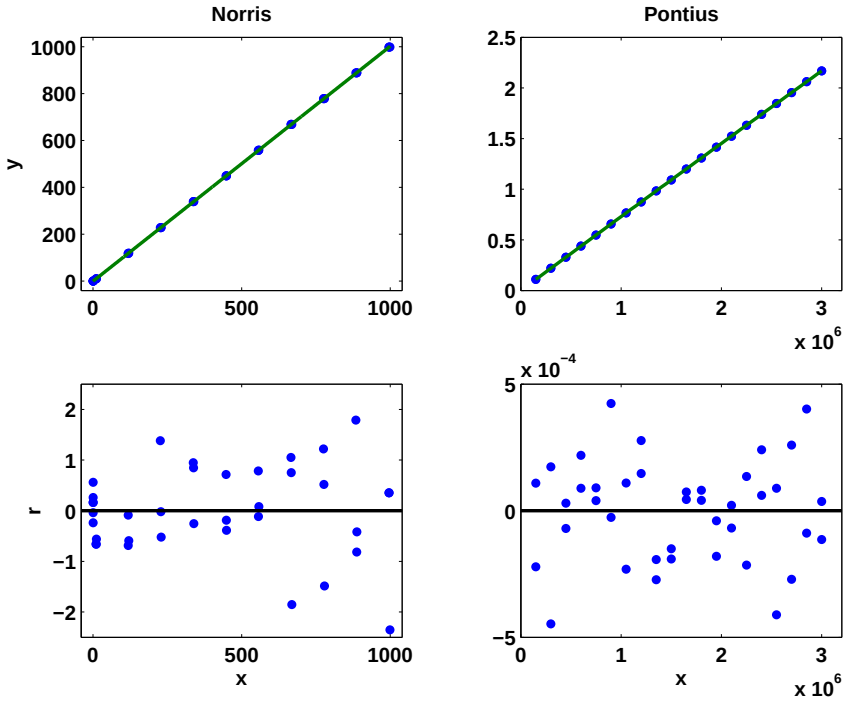


5.8. See `sinusoidalfit.m`.



5.9. Statistical Reference Datasets. See `nist.m`.

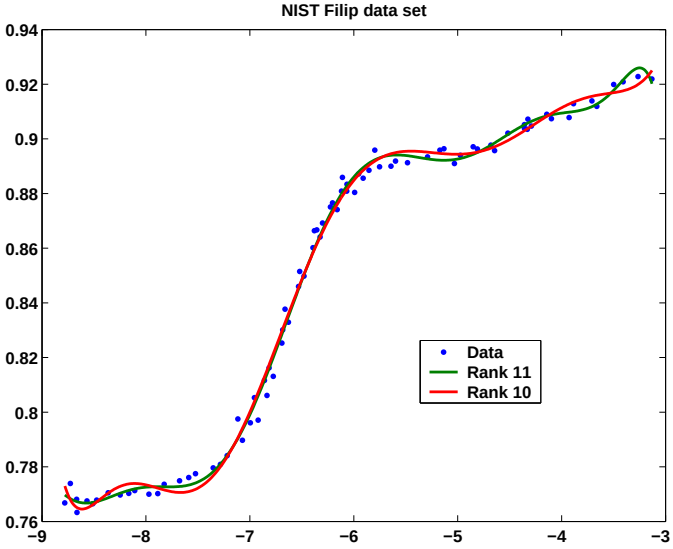
```
Norris

beta =

   -2.623230737740738e-001
    1.002116818020455e+000

Pontius

beta =

    6.735657894733633e-004
    7.320591604010027e-007
   -3.160818713450353e-015
```



5.10. Filip data set. See `xfilip.m`. As it runs, you will notice that there are three warning messages. The first comes while using `polyfit` on the uncentered data set. Polyfit warns that the problem is badly conditioned, but it goes ahead and produces 11 nonzero coefficients. The second warning using backslash on the uncentered data. The third warning comes from trying to invert the normal equations matrix.
Backslash says that the Vandermonde matrix is rank deficient and then produces a "basic" solution with only 10 nonzero coefficients. Pseudoinverse also decides the rank is 10, but produces 11 nonzero coefficients.
The program shows two fits, one labeled rank 11 and one labeled rank 10. The rank 11 fit is the official one, using the NIST coefficients or any MATLAB computation with centered data. The rank 10 curve is actually two different fits that are graphically indistinguishable.
The norm of the residual from the rank 10 fits is slightly larger than from the rank 11 fits, but the coefficients are three orders of magnitude smaller. Now you see why this data set is controversial.

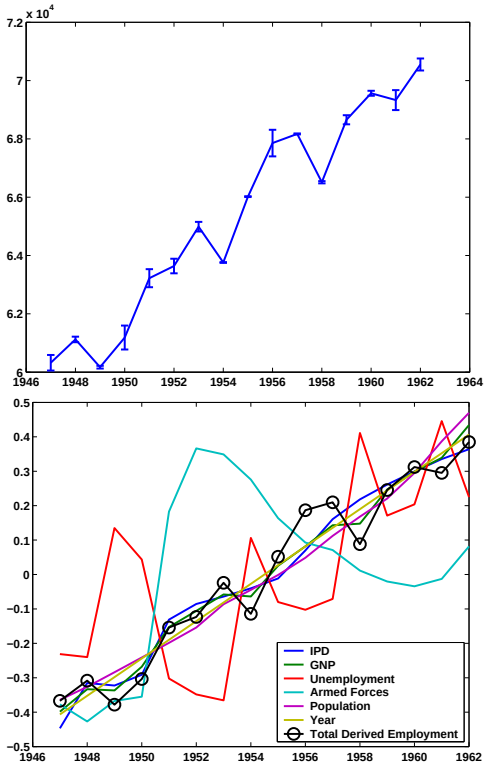
```
Warning: Polynomial is badly conditioned. Remove repeated
  data points or try centering and scaling.
> In c:\moler\ncm\leastsquares\solutions\xfilip.m at line 31
Warning: Rank deficient, rank = 10  tol =   1.3012e-004.
> In c:\moler\ncm\leastsquares\solutions\xfilip.m at line 42
Warning: Matrix is close to singular or badly scaled.
> In c:\moler\ncm\leastsquares\solutions\xfilip.m at line 56

beta =
    NIST                Polyfit             Centered
   -4.0296252508e-05   -4.0296249018e-05   -4.0296252508e-05
   -2.4678107827e-03   -2.4678105736e-03   -2.4678107827e-03
   -6.7019115459e-02   -6.7019109892e-02   -6.7019115459e-02
   -1.0622149858e+00   -1.0622148992e+00   -1.0622149858e+00
   -1.0875318035e+01   -1.0875317163e+01   -1.0875318035e+01
   -7.5124201739e+01   -7.5124195808e+01   -7.5124201739e+01
   -3.5447823370e+02   -3.5447820609e+02   -3.5447823370e+02
   -1.1279739409e+03   -1.1279738541e+03   -1.1279739409e+03
   -2.3163710816e+03   -2.3163709051e+03   -2.3163710816e+03
   -2.7721795919e+03   -2.7721793826e+03   -2.7721795919e+03
   -1.4674896142e+03   -1.4674895042e+03   -1.4674896142e+03

normr =
    2.8210837956e-02    2.8210838089e-02    2.8210838026e-02

beta =
    NIST                Vandermonde         Pseudoinverse
   -4.0296252508e-05    2.8639147336e-06    2.9900011430e-06
   -2.4678107827e-03    1.2468099822e-04    1.3161883533e-04
```

```
   -6.7019115459e-02    2.2361988969e-03    2.4039951090e-03
   -1.0622149858e+00    2.0799334515e-02    2.3140890647e-02
   -1.0875318035e+01    9.8981445320e-02    1.1977164300e-01
   -7.5124201739e+01    1.3574047553e-01    2.5772725697e-01
   -3.5447823370e+02   -8.8115156549e-01   -4.0645879443e-01
   -1.1279739409e+03   -4.5333693227e+00   -3.3419030994e+00
   -2.3163710816e+03   -7.1441784855e+00   -5.3507499126e+00
   -2.7721795919e+03                    0    1.3649967097e+00
   -1.4674896142e+03    8.1337811876e+00    8.4430473114e+00

normr =
    2.8210837956e-002    3.2722405980e-002    3.2726981819e-002
```



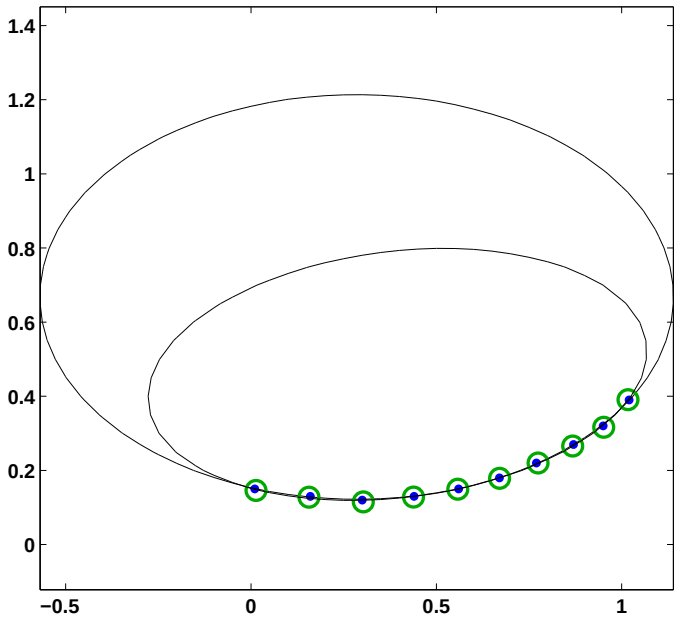
5.11. Longley data set. See `xlongley.m`.

```
                    beta
      MATLAB                  NIST
-3482258.63459545      -3482258.63459582
 15.0618722713330       15.0618722713733
-0.03581917929260      -0.03581917929259
-2.02022980381711      -2.02022980381683
-1.03322686717351      -1.03322686717359
-0.05110410565322      -0.05110410565358
 1829.15146461335       1829.15146461355

corrcoeff =
 1.0000    0.9916    0.6206    0.4647    0.9792    0.9911
 0.9916    1.0000    0.6043    0.4464    0.9911    0.9953
 0.6206    0.6043    1.0000   -0.1774    0.6866    0.6683
 0.4647    0.4464   -0.1774    1.0000    0.3644    0.4172
 0.9792    0.9911    0.6866    0.3644    1.0000    0.9940
 0.9911    0.9953    0.6683    0.4172    0.9940    1.0000
```



5.12. Planetary orbit, Heath97. See `heath.m`. Two orbits, one from original data, one after small random perturbation.

```
c =
   -2.2537948175
   -0.0063247132
   -5.5221834331
    1.2898102053
    7.3773544034

c =
   -2.4423051434
    1.7739414419
   -9.5532799239
    1.1162584602
    8.0629704039
```



Chapter 6

Quadrature

Exercises

6.1. `[Q,cnt] = quadgui(@humps,0,1,1.e-4)`
`Q = 29.85832444437543`
`cnt = 93`
Points concentrated near humps at .3 and .9.

`[Q,cnt] = quadgui(@humps,0,1,1.e-6)`
`Q = 29.85832540194041`
`cnt = 265`
Points concentrated near humps at .3 and .9.

`[Q,cnt] = quadgui(@humps,-1,2,1.e-4)`
`Q = 26.34496347100993`
`cnt = 165`
Some function values negative. Points concentrated near humps.

`[Q,cnt] = quadgui(@sin,0,pi,1.e-8)`
`Q = 1.99999999999795`
`cnt = 121`
Easy problem. Points almost evenly spaced.

`[Q,cnt] = quadgui(@cos,0,9/2*pi,1.e-6)`
`Q = 1.00000000010262`
`cnt = 241`
Slightly wider spacing near odd multiples of pi/2.

`[Q,cnt] = quadgui(@sqrt,0,1,1.e-8)`
`Q = 0.66666666218158`
`cnt = 153`

Points strongly concentrated near $x = 0$ where $f'(x)$ blows up.

```
[Q,cnt] = quadgui('sqrt(x)*log(x)',eps,1,1.e-8)
Q = -0.44444444104962
cnt = 205
```
Points strongly concentrated near $x = 0$ where $f'(x)$ blows up.

```
[Q,cnt] = quadgui('1/(3*x-1)',0,1)
Warning: Divide by zero.
Error in ==> d:\moler\ncm\ncm\quadgui.m (quadguistep)
```
Nonintegrable singularity at $x = 1/3$

```
help beta
 BETA   Beta function.
    Y = BETA(Z,W) computes the beta function for corresponding
    elements of Z and W.  The beta function is defined as
    beta(z,w) = integral from 0 to 1 of t.^(z-1) .* (1-t).^(w-1) dt.
B = beta(11/3,13/3)
[Q,cnt] = quadgui('t^(8/3)*(1-t)^(10/3)',0,1,1.e-8)
B = 0.00737204436004
Q = 0.00737204438345
cnt = 73
```
Points nearly evenly spaced.

```
B = beta(26,3)
[Q,cnt] = quadgui('t^25*(1-t)^2',0,1,1.e-8)
B = 1.017501017501012e-004
Q = 1.017502956491289e-004
cnt = 49
```
Points concentrated under the peak near $t = 1$

6.2.
```
format rat
S1 = [1 0 4 0 1]/6
S2 = [1 4 2 4 1]/12
Q = S2 + (S2 - S1)/15
Q =
     7/90        16/45         2/15        16/45         7/90
syms x
for p = 1:7, I(p,1) = int(x^(p-1),-2,2); end
I =
   [     4]
   [     0]
   [  16/3]
   [     0]
   [  64/5]
   [     0]
   [ 256/7]
```

```
X = [1 1 1 1 1; -2 -1 0 1 2];
for p = 3:7, X(p,:) = X(2,:).*X(p-1,:); end
4*X*Q' =
      4
      0
     16/3
      0
     64/5
      0
    128/3
```

This is Boole's quadrature rule.

$$\int_{-2}^{2} f(x)dx \approx \frac{2}{45}(7f(-2) + 32f(-1) + 12f(0) + 32f(1) + 7f(2))$$

The rule is exact for $f(x) = x^p, p = 0, \ldots, 5$, but not for $f(x) = x^6$.

6.3. See `trapforpi.m`

```
3.134926113810990   6.667e-003  0.666653977880305
3.139925988907159   1.667e-003  0.666665873053596
3.141175986954129   4.167e-004  0.666666617063072
3.141488486923612   1.042e-004  0.666666663562410
3.141566611923134   2.604e-005  0.666666666484161
3.141586143173127   6.510e-006  0.666666666666060
```

The error is proportional to $1/n^2$.

6.4. See `quadtxforpi.m`

```
F = inline('2./(1+x.^2)');
for k = 1:32
   tol = 1/2^k;
   [Q,cnt] = quadtx(F,-1,1,tol);
   err = Q-pi;
   disp(sprintf('%10.2e  %10.2e %5d %7.3f %9.1f', ...
     tol,err,cnt,-log(tol)/cnt,tol/err))
end
```

```
                          -log(tol)/   tol/
  tol        err      cnt    cnt       err
5.00e-01  -2.16e-02     5   0.139     -23.2
2.50e-01  -2.16e-02     5   0.277     -11.6
1.25e-01   5.25e-04     9   0.231     238.1
6.25e-02   5.25e-04     9   0.308     119.0
3.13e-02   5.25e-04     9   0.385      59.5
1.56e-02   5.25e-04     9   0.462      29.8
7.81e-03   5.25e-04     9   0.539      14.9
3.91e-03   1.44e-06    17   0.326    2711.7
```

```
 1.95e-03   1.44e-06    17   0.367    1355.8
 9.77e-04   1.44e-06    17   0.408     677.9
 4.88e-04   1.44e-06    17   0.449     339.0
 2.44e-04   1.44e-06    17   0.489     169.5
 1.22e-04   7.55e-09    33   0.273   16162.3
 6.10e-05   7.55e-09    33   0.294    8081.2
 3.05e-05   7.55e-09    33   0.315    4040.6
 1.53e-05   7.55e-09    33   0.336    2020.3
 7.63e-06   6.11e-08    41   0.287     124.9
 3.81e-06   4.06e-08    49   0.255      93.9
 1.91e-06   1.18e-10    65   0.203   16130.6
 9.54e-07   1.18e-10    65   0.213    8065.3
 4.77e-07   1.18e-10    65   0.224    4032.7
 2.38e-07   9.06e-10    81   0.188     263.2
 1.19e-07   2.21e-10   113   0.141     538.3
 5.96e-08   2.62e-10   121   0.137     227.5
 2.98e-08   2.62e-10   121   0.143     113.8
 1.49e-08   1.85e-12   129   0.140    8064.0
 7.45e-09   1.37e-11   153   0.122     544.6
 3.73e-09   6.06e-12   209   0.093     614.8
 1.86e-09   2.53e-12   249   0.081     736.1
 9.31e-10   2.84e-14   257   0.081   32768.0
 4.66e-10   2.84e-14   257   0.084   16384.0
 2.33e-10   2.13e-13   305   0.073    1092.3
```

The function evaluation count is roughly proportional to `log(tol)`, and accuracy is roughly proportional to `tol`, although the "constants" of proportionality are not really constant.

6.5. $$\int_0^1 \frac{x^4(1-x)^4}{1+x^2}dx$$

```
syms x
f = x^4*(1-x)^4/(1+x^2)
int(f,0,1)
ans =
   22/7-pi
```

A reminder of the famous approximation $\pi \approx \frac{22}{7}$.

```
F = inline(char(f))
quadtx(F,0,1,tol)
```

Presents no difficulties. Default tolerance requires only 25 function evaluations. Tolerance = 10^{-16} requires only 2101 function evaluations.

6.6.
```
f = inline('exp(-x.^2)')
for x = .1:.1:1
```

```
    E = 2/sqrt(pi)*quadtx(f,0,x);
    err = E - erf(x);
    disp([x err]),
end

  1.0000e-001 -6.8660e-012
  2.0000e-001 -1.2790e-011
  3.0000e-001 -1.9464e-010
  4.0000e-001 -4.1409e-010
  5.0000e-001 -7.1429e-011
  6.0000e-001 -1.8632e-010
  7.0000e-001  1.1478e-008
  8.0000e-001  3.3274e-010
  9.0000e-001  7.4209e-010
  1.0000e+000  2.1616e-009
```

Default tolerance leads to error less than 10^{-8} or better.

6.7. $$\beta(z,w) = \int_0^1 t^{z-1}(1-t)^{w-1}dt$$

```
function Q = mybeta(z,w)
F= inline('t^(z-1)*(1-t)^(w-1)','t','z','w');
Q = quadtx(F,eps,1-eps,1.e-8,z,w);
```

The limits of integration avoid the singularities at 0 if $z < 1$ and at 1 if $w < 1$. This function works satisfactorily if you avoid very small or very large arguments.

6.8. $$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt$$

```
function [Q,cnt] = mygamma(x)
F = inline('t^(x-1)*exp(-t)','t','x');
[Q,cnt] = quadtx(F,eps,5*x,1.e-8,x);
```

This is not a very good way to evaluate the gamma function. `quadtx` is not designed to handle infinite integrals. The upper limit of $5x$ is chosen because the integrand is pretty small by then. `mygamma(15)` gets an accurate answer, but takes almost 50,000 function evaluations. I gave up waiting for `mygamma(18)` to finish.

6.9. $$\int_0^{4\pi} \cos^2 x \; dx = 2\pi$$

`quadtx` samples the integrand 5 times, at $k\pi, k = 0, \ldots, 4$. All 5 samples are equal to 1, so `quadtx` thinks the integrand is equal to 1 everyplace and believes the integral is equal to 4π. `quadtx` samples the integral 7 times initially, at `x = [a a+h a+2*h (a+b)/2 b-2*h b-h b]` where `h = 0.13579*(b-a)`. It would take an integrand that "knows" about these 7 points to fool `quad`.

6.10. $\int_0^1 x \sin \frac{1}{x} dx$

```
= 1/2*cos(1)+1/2*sin(1)+1/2*sinint(1)-1/4*pi
= 0.37853001712416
```

`quadtx('x*sin(1/x)',0,1)` divides by zero immediately. The resulting `Inf` leads to potentially infinite recursion.
`[Q,cnt] = quadtx('x*sin(1/x)',realmin,1,1.e-12)` takes 24057 function evaluations and produces a result with an error of `4.8e-9`.

6.11. $\int_0^1 x^x dx$

```
syms x
I = int(x^x,0,1)
Warning: Explicit integral could not be found.
I = int(x^x,x = 0 .. 1)
vpa(I)
ans = .783430510712134407059264386526 98

[Q,cnt] = quadtx('x^x',0,1,1.e-18)
Q = 0.78343051071213
cnt = 141813
```

6.12. $\int_{-1}^1 \log(1+x)\log(1-x)dx = 4 + 2\log(2)^2 - 4\log(2) - \frac{1}{3}\pi^2$

```
= -1.1015508280998312612795 52
```

Integrand is an upside down U. There are integrable singularities at both ends of the interval. `quadtx(f,-1,1)` takes `log(0)` and never recovers. Integrate over slightly smaller interval to avoid singularity.
`quadtx(f,-1+eps,1-eps) = -1.10155178223377`
Justifification: Near $x = 1$, $\log(1+x) \approx log(2)$. The neglected tail is bounded by `log(2)*int(log(1-x),1-eps,1)`, which is less than `26*eps`. Similarly near $x = -1$.

```
for k = 1:12
   tol(k) = 1/10^k;
   [q(k),cnt(k)] = quadtx(f,-1+eps,1-eps,tol(k));
end
err = double(I)-q;
```

The error is very close to the tolerance. This can be seen by
`loglog(tol,err,'o-')`
or

```
              tol                 err
   0.10000000000000   0.09625018963983
   0.01000000000000   0.01116357559052
   0.00100000000000   0.00129450070918
```

```
0.00010000000000   0.00015165834782
0.00001000000000   0.00001732943837
0.00000100000000   0.00000095413394
0.00000010000000   0.00000010616909
0.00000001000000   0.00000001164213
0.00000000100000   0.00000000119678
0.00000000010000   0.00000000012307
0.00000000001000   0.00000000001207
0.00000000000100   0.00000000000104
```

The plot

```
loglog(tol,cnt,'o-')
```

shows that the function count is roughly a power of tol. Some curve fitting shows that `cnt` is roughly inversely proportional to `tol.^(1/6)`.

```
 cnt  round(38./tol.^(1/6))
  49          56
  73          82
  97         120
 129         176
 185         259
 289         380
 441         558
 641         819
1017        1202
1609        1764
2473        2589
3937        3800
```

6.13. $$\int_{-2}^{2} x^{10} - 10x^8 + 33x^6 - 40x^4 + 16x^2 \, dx = 10240/693$$

`quadtx` samples the integrand 5 times, at $-2, -1, 0, 1$, and 2. All 5 samples are equal to 0, so `quadtx` thinks the integrand is equal to 0 everyplace and believes the integral is equal to zero. For any `c` between -2 and 2, `quadtx(f,-2,c) + quadtx(f,c,2)` computes the integral correctly.

6.14. $$\int_{-1}^{2} \frac{1}{\sin(\sqrt{|t|})} dt$$

```
quadtx(f,-1,2,1.e-15)
ans =
   5.31411561028878
```

The singularity at the origin is integrable and, because the interval of integration is not symmetric about the origin, the integrand is never evaluated at exactly 0.

6.15. See `quadtxmod.m`.

```
quadtxmod(@log,0,1)
Warning: Log of zero.
In c:\moler\ncm\quad\solutions\quadtxmod.m at line 32
Warning: Modifying endpoint
In c:\moler\ncm\quad\solutions\quadtxmod.m at line 39
ans =
   -1.0000
```

6.16. See `quadtxmod.m`.

```
quadtxmod(inline('1./x'),-1/2,1)
Warning: Maximum function count exceeded.  Singularity likely.
In c:\moler\ncm\quad\solutions\quadtxmod.m at line 53
```

6.17. Lobatto rule.

$$\int_{-1}^{1} f(x)\ dx \approx w_1 f(-1) + w_2 f(-x_1) + w_2 f(x_1) + w_1 f(1)$$

Taking $f(x) = 1,\ x^2,\ x^4$ leads to

$$\begin{aligned} w_1 + w_2 &= 1 \\ w_1 + w_2 x_1^2 &= 1/3 \\ w_1 + w_2 x_1^4 &= 1/5 \end{aligned}$$

The solution is $x_1 = 1/\sqrt{5},\ w_1 = 1/6,\ w_2 = 5/6$. In `quadl.m`, these parameters are mixed in with the parameters for the higher order Kronrod formulas in the statements

```
s = [ ...  1/sqrt(5) ... ]
```

and

```
Q1 = (h/6)*[1 5 5 1]*y(1:4:13)'
```

6.18.
$$E_k = \int_0^1 x^k e^{x-1}\ dx$$

$$E_0 = \int_0^1 e^{x-1}\ dx = e^{x-1}\Big|_0^1 = 1 - 1/e$$

Integrate by parts

$$E_k = x^k e^{x-1}\Big|_0^1 - \int_0^1 k x^{k-1} e^{x-1}\ dx = 1 - k E_{k-1}$$

Quadrature:

```
f = inline('x^k*exp(x-1)','x','k')
for k = 1:20
   E(k) = quadtx(f,0,1,eps,k)
end
```

Forward recursion:

```
E0 = 1 - 1/exp(1);
E(1) = 1 - E0
for k = 2:20
   E(k) = 1 - k*E(k-1);
end
```

Backward recursion:

```
E(32) = 0;
for k = 32:-1:2
   E(k-1) = (1 - E(k))/k
end
E(21:32) = [];
```

Quadrature is accurate, but slow. Forward recursion is unstable. The initial error is multiplied by `k` at the `k`-th step. The error in `E(20)` is 20! times the initial roundoff error. Backward recursion is stable. Even though `E(32)` is totally wrong, the error is divided by `k` at the `k`-th step.
Here are the results from the three methods, with asterisks in place of incorrect digits.

```
0.36787944117144   0.36787944117144   0.36787944117144
0.26424111765712   0.26424111765712   0.26424111765712
0.20727664702865   0.20727664702865   0.20727664702865
0.17089341188538   0.17089341188539   0.17089341188538
0.14553294057308   0.14553294057307   0.14553294057308
0.12680235656153   0.1268023565615*   0.12680235656153
0.11238350406930   0.112383504069**   0.11238350406930
0.10093196744559   0.10093196744***   0.10093196744559
0.09161229298966   0.0916122929****   0.09161229298966
0.08387707010339   0.083877070*****   0.08387707010339
0.07735222886266   0.07735222******   0.07735222886266
0.07177325364803   0.0717732*******   0.07177325364803
0.06694770257562   0.066947********   0.06694770257562
0.06273216394138   0.06273*********   0.06273216394138
0.05901754087930   0.0589**********   0.05901754087930
0.05571934593124   0.056***********   0.05571934593124
0.05277111916899   0.0*************   0.05277111916899
0.05011985495809   ***************   0.05011985495809
0.04772275579621   ***************   0.04772275579621
0.04554488407582   ***************   0.04554488407582
```

6.19. See `trefethen.m`.

$$T = \lim_{\epsilon \to 0} \int_{\epsilon}^{1} x^{-1} \cos\left(x^{-1} \log x\right)\, dx$$

```
= 0.323367431678
```

6.20.

$$P(s) = \frac{3hs^2 - 2s^3}{h^3} y_{k+1} \quad + \quad \frac{h^3 - 3hs^2 + 2s^3}{h^3} y_k + \frac{s^2(s-h)}{h^2} d_{k+1} \quad + \quad \frac{s(s-h)^2}{h^2} d_k$$

```
syms h s
[int((3*h*s^2-2*s^3)/h^3,0,h)
 int((h^3-3*h*s^2+2*s^3)/h^3,0,h)
 int(s^2*(s-h)/h^2,0,h)
 int(s*(s-h)^2/h^2,0,h)]

 ans =
 [     1/2*h]
 [     1/2*h]
 [ -1/12*h^2]
 [  1/12*h^2]
```

$$\int_0^h P(s) ds = h \frac{y_{k+1} + y_k}{2} - h^2 \frac{d_{k+1} - d_k}{12}$$

6.21. (a) See `splinequad.m` and `pchipquad.m`
(b)

```
x = 1:6
y = [6  8  11  7  5  2]

[pchipquad(x,y)
 splinequad(x,y)
 trapz(x,y)]

ans =
   35.4167
   35.2500
   35.0000
```

(c)

```
x = round(100*[0 sort(rand(1,6)) 1])/100
y = round(400./(1+x.^2))/100
[pchipquad(x,y)
 splinequad(x,y)
 trapz(x,y)]
```

Division by zero is possible if the `x`'s are not distinct. Otherwise, a typical result is

```
3.1325
3.1442
3.1163
```

6.22. Discrete spline quadrature with various end conditions.

```
x = 1:6
y = [6  8  11  7  5  2]
for e = ['c','n','p','s','v']
   disp(e)
   ppval(fnint(csape(x,y,e)),x(end))
end
```

```
complete             35.2778
not-a-knot           35.2500
periodic             35.0000
second derivatives   35.3947
variational          35.3947
```

6.23. How large is your hand? See `handquad.m`.

Chapter 7

Ordinary Differential Equations

Exercises

7.1. (a) Show `ode23tx` is exact for $f(t,y) = 1, t$, and t^2, but not for t^3. Experimentally.

$f(t,y) = 1,\ y = t$

```
[t,y] = ode23tx(inline('t^0','t','y'),[0 10],0);
err = max(abs(y-t))
err = 0
```

$f(t,y) = t,\ y = t^2/2$

```
[t,y] = ode23tx(inline('t^1','t','y'),[0 10],0);
err = max(abs(y-t.^2/2))
err = 0
```

$f(t,y) = t^2,\ y = t^3/3$

```
[t,y] = ode23tx(inline('t^2','t','y'),[0 10],0);
err = max(abs(y-t.^3/3))
err = 1.1369e-013
% This is just roundoff error.
```

$f(t,y) = t^3,\ y = t^4/4$

```
[t,y] = ode23tx(inline('t^3','t','y'),[0 10],0);
err = max(abs(y-t.^4/4))
err = 0.0441
% This is not just roundoff error.
```

Algebraically.

$$f(t,y) = 1,\ y = t$$

$$\begin{aligned} y_n &= t_n \\ s_1 &= 1,\ s_2 = 1,\ s_3 = 1 \\ y_{n+1} &= y_n + h(2s_1 + 3s_2 + s_3)/9 \\ &= t_n + h(2+3+4)/9 \\ &= t_{n+1} \end{aligned}$$

$$f(t,y) = t,\ y = t^2/2$$

$$\begin{aligned} y_n &= t_n^2/2 \\ s_1 &= t_n,\ s_2 = t_n + h/2,\ s_3 = t_n + 3h/4 \\ y_{n+1} &= y_n + h(2s_1 + 3s_2 + s_3)/9 \\ &= t_n^2/2 + h(2t_n + 3(t_n + h/2) + 4(t_n + 3h/4)/9) \\ &= t_n^2/2 + ht_n + h^2/2 \\ &= t_{n+1}^2/2 \end{aligned}$$

$$f(t,y) = t^2,\ y = t^3/3$$

$$\begin{aligned} y_n &= t_n^3/3 \\ s_1 &= t_n^2,\ s_2 = (t_n + h/2)^2,\ s_3 = (t_n + 3h/4)^2 \\ y_{n+1} &= y_n + h(2s_1 + 3s_2 + s_3)/9 \\ &= t_n^3/3 + h(2t_n^2 + 3(t_n + h/)^2 + 4(t_n + 3h/4)^2)/9 \\ &= t_n^3/3 + ht_n^2 + t_n h^2 + h^3/3 \\ &= t_{n+1}^3/3 \end{aligned}$$

$$f(t,y) = t^3,\ y = t^4/4$$

$$\begin{aligned} y_n &= t_n^4/4 \\ s_1 &= t_n^3,\ s_2 = (t_n + h/2)^3,\ s_3 = (t_n + 3h/4)^3 \\ y_{n+1} &= y_n + h(2s_1 + 3s_2 + 4s_3)/9 \\ &= t_n^4/4 + ht_n^3 + 3h^2 t_n^2/2 + t_n h^3 + 11h^4/48 \\ &= (t_n + h)^4/4 - h^4/48 \\ &\neq t_{n+1}^4/4 \end{aligned}$$

(b) When is the error estimator in `ode23tx` exact?
The error estimator is the difference between a second order method and a third order method. (That's why the method is called `ode23`.) So it is exact for $f(t,y) = t^2$, but not for $f(t,y) = t^3$. The error estimator estimates the error in the low order formula, but the function uses the high order formula to advance the solution. The function gets the exact solution, within roundoff error, for $f(t,y) = t^3$, but it doesn't "know" it's getting the exact solution.

7.2.
```
f = inline('2/sqrt(pi)*exp(-x^2)','x','y')
[x,y] = ode23tx(f,[0 2],0)
format long
[x y y-erf(x)]
                   0                   0                   0
   0.00007089815404   0.00007999999987  -0.00000000000000
   0.00042538892422   0.00047999997105  -0.00000000000000
   0.00219784277512   0.00247999600678  -0.00000000000000
   0.01106011202965   0.01247949114036  -0.00000000000183
   0.05537145830229   0.06241619845376  -0.00000000571019
   0.14341156510599   0.16071976604159  -0.00000027267420
   0.26526185575992   0.29243914999818  -0.00000220506665
   0.41928620148511   0.44678270194032  -0.00000953168159
   0.61306570673602   0.61403021088296  -0.00003097764172
   0.81306570673602   0.74974128938601  -0.00005245563314
   1.01306570673602   0.84798787236882  -0.00006603990739
   1.21306570673602   0.91368201872542  -0.00007060930407
   1.41306570673602   0.95425593569379  -0.00006822715409
   1.61306570673602   0.97740226036827  -0.00006227226432
   1.81306570673602   0.98959877083580  -0.00005572624995
   2.00000000000000   0.99527068734127  -0.00005157767768
```

7.3. (a) See `myrk4.m`.

(b) Cutting the step size in half reduces the error by a factor of $2^4 = 16$.

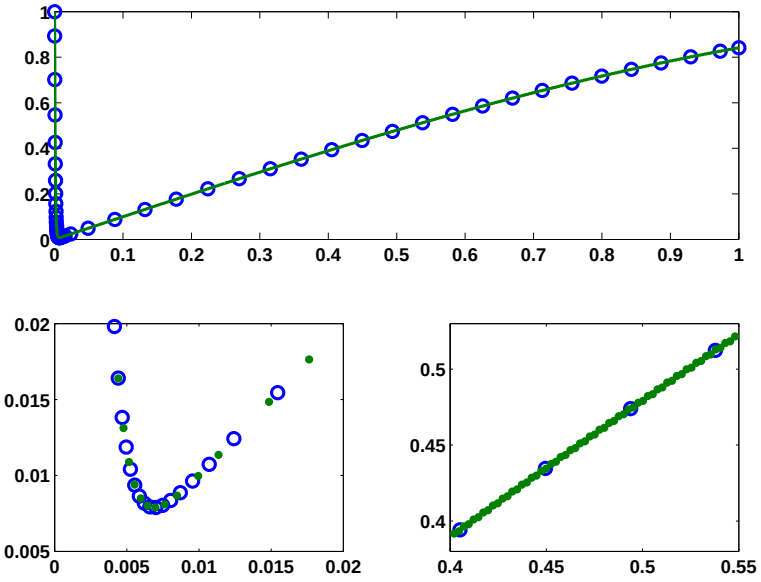
```
[t1,y1] = myrk4(inline('y','x','y'),[0 1],1,.1);
[t2,y2] = myrk4(inline('y','x','y'),[0 1],1,.05);
e = exp(1);
err1 = y1(end)-e, err2 = y2(end)-e, ratio = (y1(end)-e)/(y2(end)-e)
err1 = -2.084323879714134e-006
err2 = -1.358027112985383e-007
ratio = 15.34817574541731
```

(c) Simple harmonic oscillator. $\ddot{y} = -y$. See `oscillator.m`.

```
oscillator = inline('[y(2); -y(1)]','t','y');
opts = odeset('reltol',1.e-6,'abstol',1.e-6,'refine',1);
y0 = [1 0];
tspan = [0 2*pi];
h = pi/50;
for k = 1:4
   switch k
      case 1, [t,y] = ode23(oscillator,tspan,y0,opts);
      case 2, [t,y] = ode45(oscillator,tspan,y0,opts);
      case 3, [t,y] = ode113(oscillator,tspan,y0,opts);
      case 4, [t,y] = myrk4(oscillator,tspan,y0,h);
   end
   err(k) = max(abs(y(end,:)-y0));
   cnt(k) = length(t)-1;
end
fprintf('       ode23        ode45       ode113        myrk4\n')
fprintf('%12.2e %12.2e %12.2e %12.2e\n',err)
fprintf('%12.0f %12.0f %12.0f %12.0f\n',cnt)
```

```
     ode23        ode45       ode113        myrk4
 7.20e-006    7.61e-007    1.24e-006    8.15e-007
       210           30           37          100
```

7.4. See `stiff1d.m`. `ode23tx` requires 416 steps while the stiff solver, `ode23s` requires only 57 steps.



One-dimensional stiff problem.

7.5. $\dot{y} = f_1(t,y) = \cos t,\ y(0) = 0$
$\dot{y} = f_2(t,y) = \sqrt{1-y^2},\ y(0) = 0$
$\ddot{y} = f_3(t,y) = -y,\ y(0) = 0,\ \dot{y}(0) = 1$
$\ddot{y} = f_4(t,y) = -\sin t,\ y(0) = 0,\ \dot{y}(0) = 1$

(a) What is the common solution?

$y(t) = \sin(t),\ 0 \le t \le \pi/2$

(b) Rewrite these problems as first-order systems.

$\dot{y} = \cos t,\ y(0) = 0$
$\dot{y} = \sqrt{1-y^2},\ y(0) = 0$
$\dot{y}_1 = y_2,\ \dot{y}_2 = -y_1, y_1(0) = 0,\ y_2(0) = 1$
$\dot{y}_1 = y_2,\ \dot{y}_2 = -\sin t, y_1(0) = 0,\ y_2(0) = 1$

(c) What is the Jacobian for each problem?

$J_1 = 0$
$J_2 = -2y/\sqrt{1-y^2}$
$J_3 = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$
$J_4 = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$

(d) How much work does `ode45` require to solve each problem? See `fourjacs.m`.

	f_1	f_2	f_3	f_4
steps	13	58	30	19
fevals	79	385	181	115

For the second formulation, the Jacobian $J_2 = -2y/\sqrt{1-y^2}$ becomes infinite as $t \to \pi/2$ and $y \to 1$.

(e) Change the interval to $0 \le t \le \pi$.

	f_1	f_2	f_3	f_4
steps	24	fails	60	37
fevals	145	∞	361	223

Notice that $f_2(t,y)$ is never negative, so the solution cannot decrease. At $t = \pi/2$ the theoretical solution is no longer unique. As t approaches $\pi/2$, y becomes slightly larger than 1, $\sqrt{1-y^2}$ becomes complex and `ode45` has to take impossibly small steps. The other three problems have no difficulties and require about twice as many steps as they did to reach $\pi/2$.

(f) Change the second formulation to $\dot{y} = f_2(t,y) = \sqrt{|1-y^2|},\ y(0) = 0$.
For $t > \pi/2$ and $y > 1$, the equation becomes $\dot{y} = \sqrt{y^2-1},\ y(\pi/2) = 1$, The solution becomes $y = \cosh(t - \pi/2)$.

7.6. The Jacobian for the two body problem. See `orbitjacobian.m`.

7.7. When is the matrix in the Lorenz equations singular and what is its null vector?

$$\begin{pmatrix} -\beta & 0 & \eta \\ 0 & -\sigma & \sigma \\ -\eta & \rho & -1 \end{pmatrix} \begin{pmatrix} \rho - 1 \\ \eta \\ \eta \end{pmatrix} = \begin{pmatrix} \eta^2 - \beta(\rho - 1) \\ 0 \\ 0 \end{pmatrix}$$

7.8. Jacobian for the Lorenz equations.

$$\begin{aligned} J &= \begin{pmatrix} -\beta & y_3 & y_2 \\ 0 & -\sigma & \sigma \\ -y_2 & \rho - y_1 & -1 \end{pmatrix} \\ &= A + \begin{pmatrix} 0 & y_3 & 0 \\ 0 & 0 & 0 \\ 0 & -y_1 & 0 \end{pmatrix} \end{aligned}$$

This function finds the eigenvalues of the Jacobian at the fixed point as a function of ρ with fixed β and σ.

```
function lambda = lorenzeigs(rho)
beta = 8/3;
sigma = 10;
eta = sqrt(beta*(rho-1));
A = [ -beta    0     eta;
           0 -sigma sigma;
        -eta    rho   -1];
y = [rho-1; eta; eta];
J = A + [0 y(3) 0; 0 0 0; 0 -y(1) 0];
lambda = eig(J);
```

Evaluating this function for each of the ρs available in `lorenzgui` shows that the Jacobian has a pair of complex eigenvalues with positive real part.

7.9. For what ρ is the Lorenz stable? Use `fzerotx` to find ρ so that the eigenvalues computed by `lorenzeigs(rho)` from the previous exercise lie on the imaginary axis. See `lorenzstable.m`. Result is $\rho = 24.737$.

7.10. Signature of the Lorenz periodic orbits.

ρ	signature
99.65	`+--+--`
100.50	`++-`
160.00	`++--`
350.00	`+-`

7.11. Period of the Lorenz periodic orbits. See `lorenzperiod.m`.

ρ	period
99.65	2.2033
100.50	1.0962
160.00	1.1529
350.00	0.3885

7.12. See `matlab/demos/orbitode`

```
te =
    0.0000
    3.0953
    6.1933
ye =
    1.2000   -0.0000   -0.0000   -1.0494
   -1.2616   -0.0012   -0.0005    1.0485
    1.1989    0.0000   -0.0047   -1.0480
ie =
     1
     2
     1
```

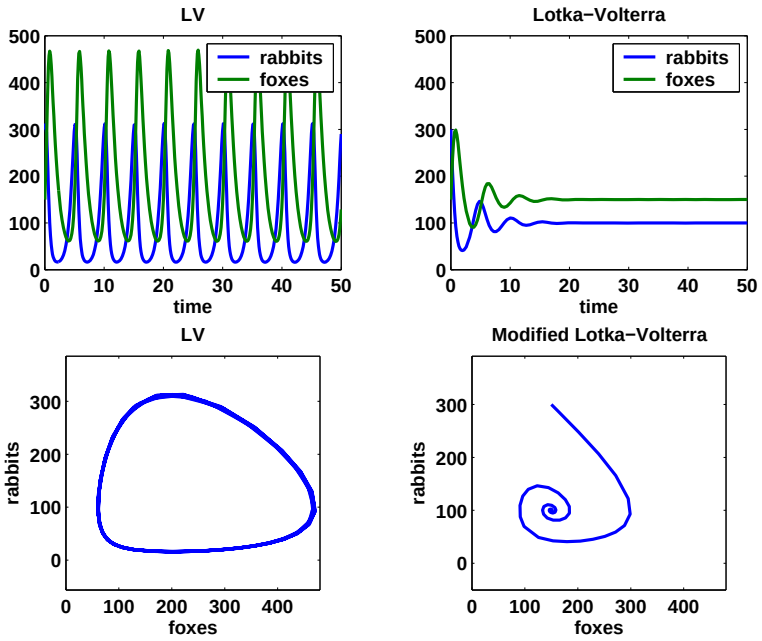
The `events` function in `orbitode` looks for local maxima or minima of the distance from the initial position. At `t = te(1)`, the capsule is at its initial position and velocity, `y = ye(1,:)`. At `t = te(2)`, the capsule is at `ye(2,:)`, which is its maximum distance from the initial position. At `t = te(3)`, the capsule has nearly returned to its initial position, `ye(3,:)` $\approx$ `ye(1,:)`. The time `te(3)` required to return to the initial position is the period. The initial conditions have been chosen to make this periodic orbit possible. The length of `tspan` is anything larger than the period.

7.13. Lotka-Volterra. See `predprey.m`.
(a) `predprey(300,150,5)`
(b) `predprey(15,22,6.62)`
(c) `predprey` returns the difference between the initial and final values. Try `e = predprey(102,198,alpha)` for a few values of `alpha` between 4 and 5. `alpha = 4.443` yields `e = [-0.0055 0.0078]`.
(c) Or, see `predpreyperiod.m`.
(d) $u = r - 1/\alpha, v = f - 2/\alpha$. Ignore terms of $O(uv)$. Resulting linear system

$$\begin{aligned} \dot{u} &= -v \\ \dot{v} &= 2u \end{aligned}$$

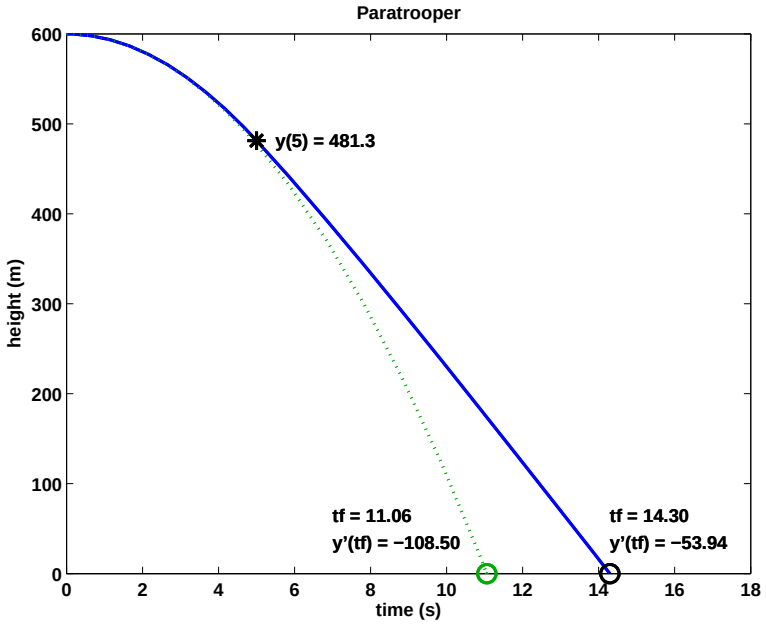
Solutions are combinations of $\cos(\sqrt{2}t)$ and $\sin(\sqrt{2}t)$. Period $= \sqrt{2}\pi = 4.4429$.

7.14. See `predpreymod.m`.



Original and modified Lotka-Volterra predator prey models

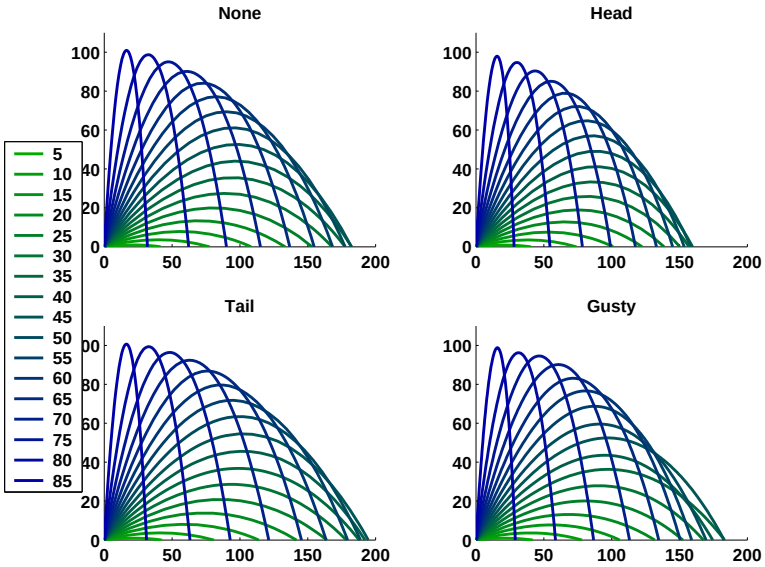
7.15. See `chute.m`.
(a) Free-fall. $t_f = 11.0600, y'(t_f) = -108.4988$.
(b) Paratrooper. $t_f = 14.2965, y'(t_f) = -53.9416$.



Paratrooper, with and without parachute.

7.16. See `cannonball.m`.

wind	theta0	tfinal	range	vfinal	nsteps
None	40	5.981	182.3	36.670	67
Steady head	40	5.786	159.7	32.478	67
Intermittent tail	45	6.669	194.6	39.204	85
Gusty	35	5.444	183.2	37.162	223

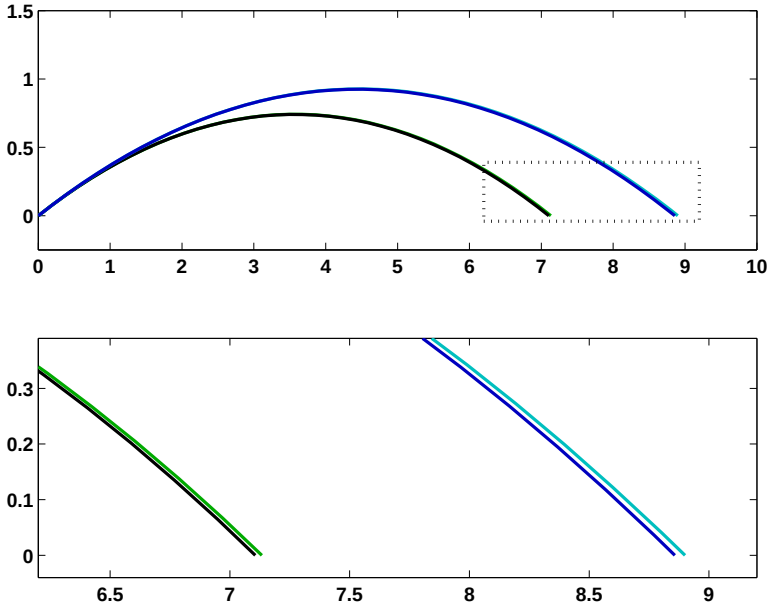


Cannonball trajectory with various wind conditions.

7.17. Bob Beamon long jump. See `beamon.m`.

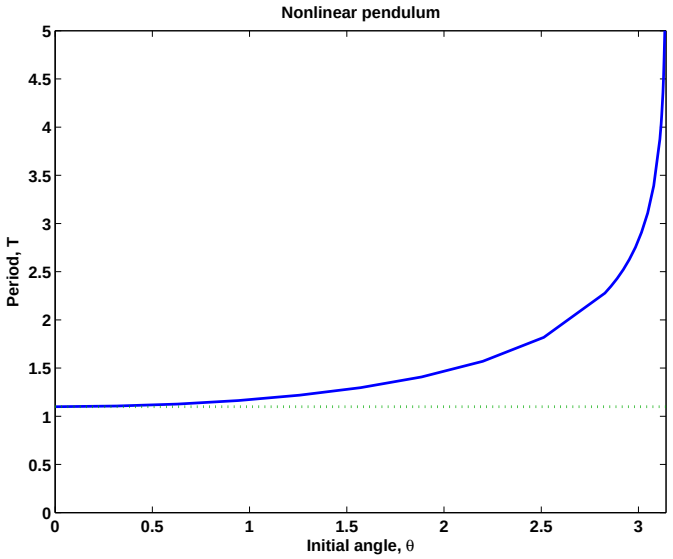
v0	theta0	rho	distance
10.0000	22.5000	0.9400	7.1333
10.0000	22.5000	1.2900	7.1059
11.1844	22.5000	0.9400	8.9000
11.1844	22.5000	1.2900	8.8575

The lower air density can account for at most 5 centimeters of added distance. Beamon's initial velocity was a far more important factor.

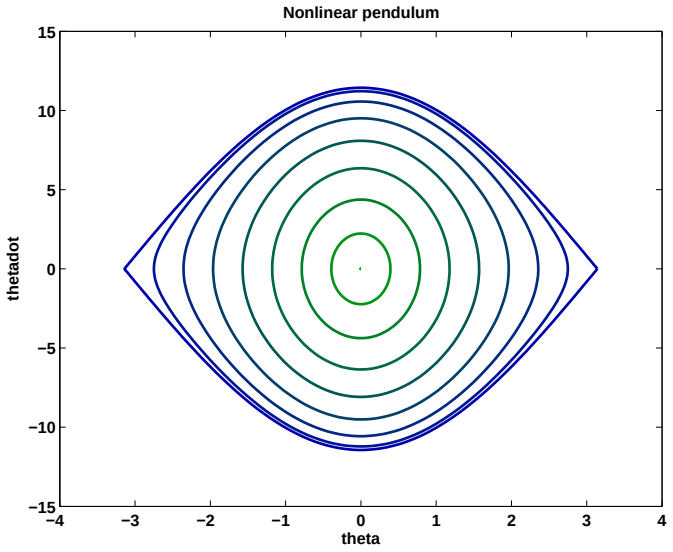


Effect of air density and initial velocity on longjump distance. The two shorter jumps are low and high altitude with nominal initial velocity. The two longer jumps are low and high altitute with a sprinter's initial velocity.

7.18. (a) Linearized period $= 2\pi\sqrt{L/g} = 1.0988$.
(b) See `pendulum.m`.



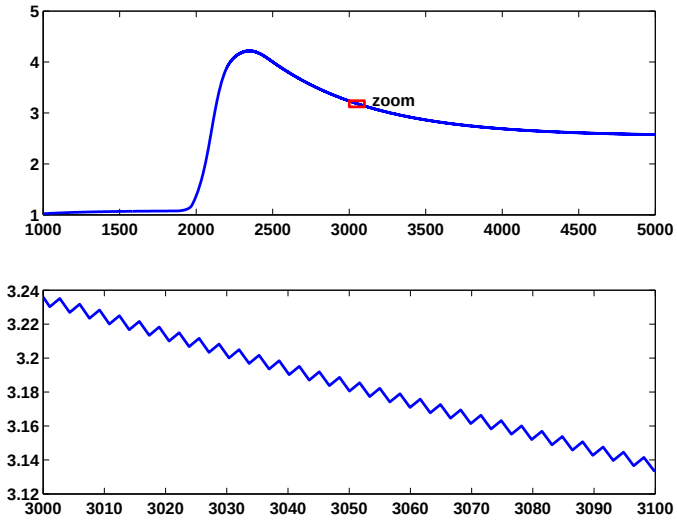
(c) The graph shows that as $\theta_0 \to 0,\ T(\theta_0) \to 1.0988$
(d)



7.19. CO2 in the atmosphere.
(a). See `co2model.m`.
(b)

```
      p        sigs      sigd      alks      alkd
y0 =
    1.0000    2.0100    2.2300    2.2000    2.2600
yfinal =
    2.5746    2.1217    2.3380    2.1989    2.2601
ratio =
    2.5746    1.0556    1.0484
```

Large increase in the atmosphere. Slight increases in the ocean.
(c) At $t = 2347.8$ the CO_2 concentrations reaches its maximum, 4.2171.
(d)



Stiffness with CO_2 model and `ode45`.

(e)

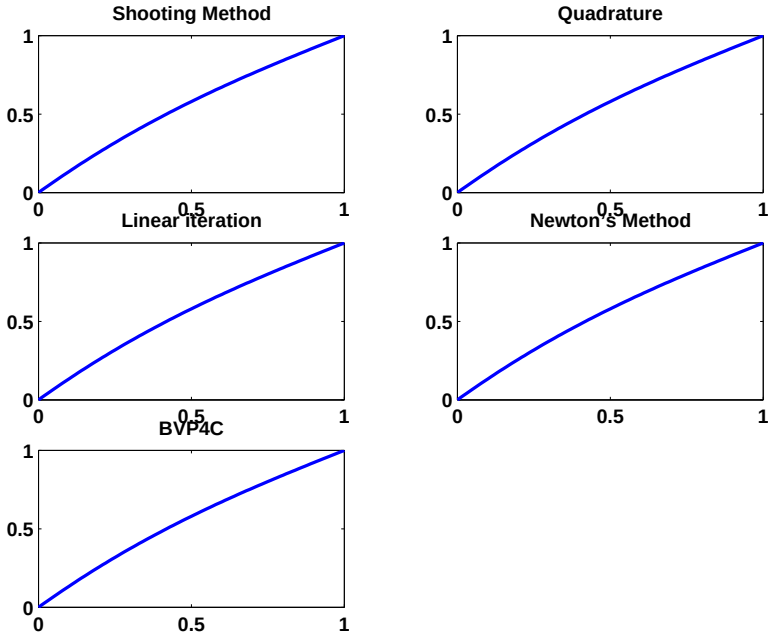
solver	steps	time(sec.)
`ode23tx`	1983	8.4220
`ode23`	1987	7.9420
`ode45`	1491	11.7070
`ode113`	3000	9.9750
`ode23s`	1413	5.2080
`ode15s`	268	0.9710

The variable order stiff solver is clearly the best for this problem.

7.20.

$$\begin{aligned} y'' &= y^2 - 1 \\ y(0) &= 0 \\ y(1) &= 1 \end{aligned}$$

See `bvp.m`.
(a) Shooting Method. `eta = 1.393628`
(b) Quadrature. `kappa = 0.97110034`
(c) Linear Iteration. `n = 99, 14 iterations`
(d) Newton's Method. `n = 99, 4 iterations`
(e) Extra, out of curiousity, use `bvp4c.m`.



7.21. Double pendulum.

(a) With initial radius ≥ 2, initial angles larger than about 1.38 radians lead to chaotic motion.

(b) `swinger(0.862,-0.994)` produces a nearly periodic orbit.

(c) `get(gcf,'userdata') = theta0`, the 2-vector of initial angles.

(d) Changing the sign of `alpha` in `swinginit` causes the other possible initial configuration to be chosen.

(e) Modify `swinger` so that other masses are possible. *Later ...*

(f) Modify `swinger` so that other lengths are possible. *Later ...*

(g) Time scales inversely as $\sqrt{g}$. The solution of the linearized, single pendulum involves $\sin(\sqrt{g}t)$. The effect of gravity on the nonlinear, double pendulum is similar.

(h) Combine `swingmass` and `swingrhs` into one function. and use `ode23tx`. *Later ...*

(i) These equations are not stiff. Small step sizes are required to follow the chaotic motion.

(j) Floating point arithmetic is the force that knocks the inverted pendulum away from its vertical position. We can't balance directly above the center because π is not a floating point number. The closest we can get is `pi`, the IEEE double precision number nearest to π. It turns out that

$$\pi - \texttt{pi} \approx .5515\ \texttt{eps} \approx 1.225 \cdot 10^{-16}$$

The first noticeable movement of the inverted pendulum occurs at about $t = 53$. Before $t = 53$, the state of the pendulum is changing, but it is not apparent on the screen. There are two different regimes, linear behavior for t up to about 36 and then exponential behavior for t between 36 and about 53.

A simplified model is provided by Euler's method for a single pendulum. With $p = \theta$, $q = \dot{\theta}$, and $\epsilon = \pi - \texttt{pi}$, Euler's method is

$$\begin{aligned} p_0 &= \pi + \epsilon \\ q_0 &= 0 \\ p_{n+1} &= p_n + hq_n \\ q_{n+1} &= q_n - h \sin p_n \end{aligned}$$

As long as $h|q_n| < \texttt{eps}$, the addition $p_n + hq_n$ rounds to p_n and p_n remains equal to `pi`. Furthermore, as long as $p_n = \texttt{pi}$, the computed value of $\sin p_n$ is ϵ. Consequently, for the first several hundred steps, Euler's method is effectively

$$\begin{aligned} p_{n+1} &= p_n \\ q_{n+1} &= q_n - h\epsilon \end{aligned}$$

The pendulum appears to remain balanced in its vertical position, but the first derivative is growning linearly with t, that is $q(t) = -\epsilon t$.

The linear behavior changes when $h|q_n|$ reaches `eps`. Then p_n begins to be affected numerically. We have set the maximum step size for `ode23` at

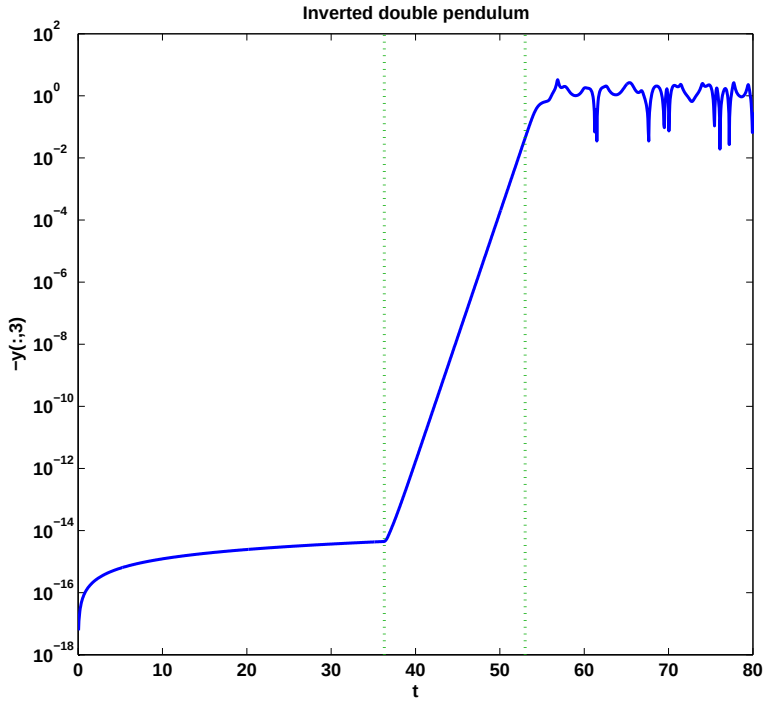
$h = .05$. So q_n begins to change the last few bits of p_n after

$$t = \texttt{eps}/(h\epsilon) \approx 1/(.05 \cdot .5515) \approx 36.3$$

Let's denote this value of t by t_m where $m = t/h = 726$ steps.
For p_n close to π, we have $\sin p_n \approx \pi - p_n$. Let $\tilde{p}_n = p_{n-m} - \pi$ and $\tilde{q}_n = q_{n-m}$. Now the Euler model is

$$\begin{aligned} \tilde{p}_{n+1} &= \tilde{p}_n + h\tilde{q}_n \\ \tilde{q}_{n+1} &= \tilde{q}_n + h\tilde{p}_n \end{aligned}$$

Both $\tilde{p}_n$ and $\tilde{q}_n$ grow like $(1+h)^n$, that is exponentially. This exponential behavior continues until either q_n or $p_n - \pi$ is large enough that our simplifications are no longer reasonable.



The plot shows the initial behavior of $\dot{\theta}_1(t)$ versus t on a logarithmic scale. For $t < 36.3$, $\dot{\theta}_1(t)$ is increasing linearly with t. For $36.3 < t < 53$, $\dot{\theta}_1(t)$ is increasing exponentially, so its logarithm increases linearly. For t greater than about 53, the pendulum begins to move on the screen and the chaotic behavior soon sets in.

Chapter 8

Fourier Transforms

Exercises

8.1. The telephone number in `touchtone` is 1-508-647-7001.

8.2. Modify `touchtone.m` so that it can dial a telephone number. *Later.*

8.3. Modify `touchtone` so that it determines the segments. *Later.*

8.4. Make a recording of a phone number. *Later.*

8.5. Prove that $F^H F = nI$.
Let $\omega = e^{2\pi i/n}$. If $k \neq l$, let $m = l - k$. Then

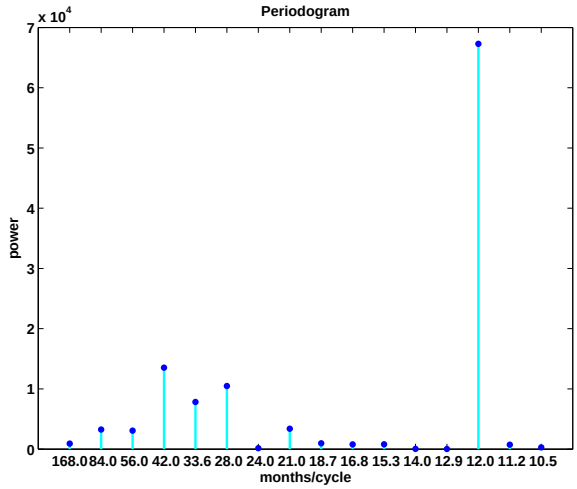
$$\begin{aligned}(F^H F)_{k,l} &= \sum_{j=0}^{n-1} \omega^{-kj}\omega^{lj} \\ &= \sum_{j=0}^{n-1} \omega^{mj} \\ &= (1-\omega^{mn})/(1-\omega^m) \\ &= (1-1)/(1-\omega^m) \\ &= 0\end{aligned}$$

If $k = l$,

$$\begin{aligned}(F^H F)_{k,k} &= \sum_{j=0}^{n-1} \omega^{-kj}\omega^{kj} \\ &= \sum_{j=0}^{n-1} 1 \\ &= n\end{aligned}$$

8.6. `fftmatrix(n,j)` A five-point star results if `n` is divisible by 5 and `j = 2*n/5` or `j = 3*n/5`. A regular pentagon results if `n` is divisible by 5 and `j = n/5` or `j = 4*n/5`.

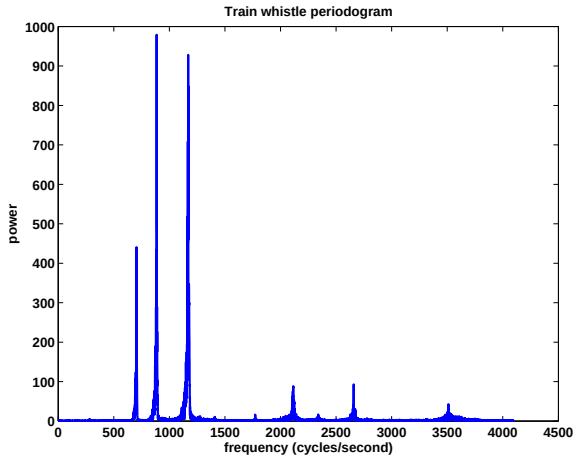
8.7. *el Niño.* See `elnino.m`. The strongest peak is at 12 months per cycle, i.e. yearly. There is another peak spread across three components with 28, 33.6, and 42 months per cycle, i.e. a little less than three years.



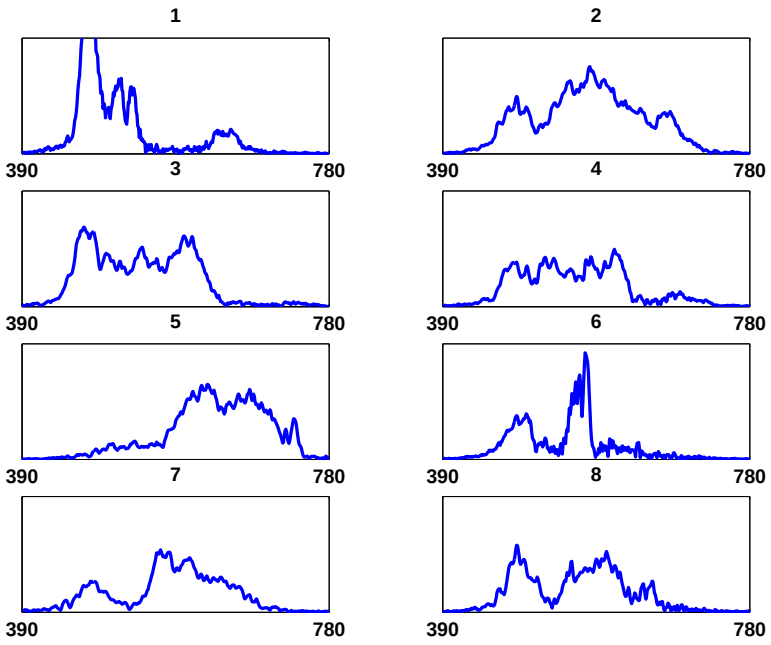
8.8. Train whistle. See `trainwhistle.m` The frequencies (in Hz.) of the peaks, and the ratios to the first peak, are

700	1
875	5/4
1167	5/3
2100	3
2625	15/4
3500	5

The first three peaks are fundamental tones. The fourth and sixth peaks are the first two overtones of the first fundamental tone. The fifth peak is the first overtone of the second fundamental tone.



8.9. Bird chirps. See `chirps.m`. The first chirp has lower frequencies and the fifth chirp has higher frequencies than the others.



Chapter 9

Random Numbers

Exercises

9.1. `rand('state',13)`
`randgui rand`
Computes π to four digits. Pure luck.

9.2. See `randgui2.m`.

9.3. With a transposed random matrix in `randgui randssp`, the autocorrelation of length 3 in `randssp` no longer affects the points in 3-dimensional space.

9.4. (a) See `randphi.m`.
(b) `subplot(2,1,1), hist(randmcg(10000,1),50)`
`subplot(2,1,2), hist(randphi(10000,1),50)`
`randphi` produces a histogram that is "too close" to uniform.
(c) `randgui randphi` does not converge to π, but rather to some value that starts with 3.26... Can anybody tell me what this value is?

9.5. (a) See `randjsr.m`.
(b) `subplot(2,1,1), hist(randjsr(10000,1),50)`
`subplot(2,1,2), hist(randtx(10000,1),50)`
(c) `randgui randjsr` works as expected.

9.6. See `randnpolar.m`.

9.7. (a) See `brownian2.m`.
(b) See `brownian3.m`

9.8. (a) A single deck of cards is represented by the integers `1:52`. Our `blackjack` simulation uses four of these single decks. The combined deck is shuffled with `randperm`, which uses `rand`. Cards are dealt from the bottom of the deck. A counter keeps track of the current length of the deck. Dealing one card

consists of accessing the last integer in the array and decrementing the length of the array.
(b) Our simulations do not reshuffle until the shoe is nearly depleted, but let's find the probability of a blackjack for a freshly shuffled shoe containing four 52-card decks. The probability that the first card is an ace is 16/208. The probability that the second card is worth 10 points is (64/207). The probability of ace followed by 10 is therefore (16/208)(64/207). Similarly, the probability of 10 followed by ace is (64/208)(16/207), the same as ace-10. To qualify as a blackjack, the dealer must not have ace-10 or 10-ace in his first cards. Consequently, the probability of blackjack from a fresh deck is

$$2(16/208)(64/207)(1 - 2(15/206)(63/205)) \approx 0.045437.$$

An extensive simulation with 5 million hands found surprisingly good agreeent. There were 227125 blackjacks, so the observed probability was 0.045425. See `blackjackmod.m`.
(c) `blackjackmod(10000,'bjpays10')`. Player disadvantage about 2%.
(d) `blackjackmod(10000,'pushloses')`. Player disadvantage over 10%.
(e) `blackjackmod(10000,'extraaces')`. Player advantage more than 2%.
(f) `blackjackmod(10000,'nokings')`. Player disadvantage almost 4%.

Chapter 10

Eigenvalues and Singular Values

Exercises

10.1.

`magic(4)`	Singular
`hess(magic(4))`	Hessenberg
`schur(magic(5))`	Schur
`pascal(6)`	Symmetric
`hess(pascal(6))`	Tridiagonal
`schur(pascal(6))`	Diagonal
`orth(gallery(3))`	Orthogonal
`gallery(5)`	Defective
`gallery('frank',12)`	Hessenberg
`[1 1 0; 0 2 1; 0 0 3]`	Schur
`[2 1 0; 0 2 1; 0 0 2]`	Jordan

10.2. M is a magic square of order n. Its largest eigenvalue, and its largest singular value, is the magic sum,

$$\mu_n = n(n^2+1)/2$$

The vector e of all ones is the corresponding eigenvector, and left and right singular vector.

$$Me = \mu_n e$$

and

$$M^T M e = \mu_n^2 e$$

Since M is a positive matrix, the Perron-Froebenium theorem insures that μ_n is the largest eigen- and singular value.

10.3. See `ffteig.m`. The eigenvalues of `fft(eye(n))`, scaled by $\sqrt{n}$ are 1, -1, i, and $-i$, but, surprisingly, not in equal numbers. For example, for $n = 20$, there are six 1's, five -1's, five -i's, and four i's. In general,

```
eigenvalue    multiplicity
      1       floor((n+4)/4)
     -1       floor((n+2)/4)
     -i       floor((n+1)/4)
      i       floor((n-1)/4)
```

(I don't have an elementary explanation of this behavior.)

10.4. `e = 2*sin((-(n-1)/2:(n-1)/2)'*pi/(n+1))`

10.5. See `trackeigs.m`

10.6. (a) `A = gallery(5)` is defective. Its eigenvalue $\lambda = 0$ has multiplicity five, but only one eigenvector. So, `condeig(A)` should be infinite.
(b) The eigenvalues computed with finite preciswion arithmetic are distinct, and each has its own eigenvector.

10.7. (a) Symbolic algebraic expressions are essentially character strings that do not have sortable numeric values until they are evaluated.
(b)

```
factor                    eigenvalue
x                         0
(x-1020)                  1020
(x^2-1020*x+100)          510+100*26^(1/2), 510-100*26^(1/2)
(x^2-1040500)             10*10405^(1/2), -10*10405^(1/2)
(x-1000)^2                1000, 1000
```

(c) What do each of these statements do?
`e = eig(sym(rosser))`, Maple computes eigenvalues symbolically.
`r = eig(rosser)`, MATLAB computes eigenvalues numerically.
`double(e) - r`, MATLAB computes the difference.
`double(e - r)`, Maple computes the difference.
(d) The roundoff errors in the eigenvalues computed by MATLAB using double precision floating point are on the order of `eps*norm(rosser) = 2.2650e-13`.
(e) Change `R(1,1)` from `611` to `612`. The characteristic polynomial of the modified matrix cannot be factored over the rationals.

10.8. Both `P = gallery('pascal',12)` and `F = gallery('frank',12)` have characteristic polynomials that are unchanged if λ is replaced by $1/\lambda$. The Pascal matrix is symmetric, so its eigenvalue problem is perfectly well conditioned. `condeig(P)` is all ones. The computed eigenvalues, `p = eig(P)`, satisfy `p.*flipud(p) == 1` as well as can be expected, given the fact that `p(12)/p(1)` is almost 10^{12}. On the other hand, the Frank matrix has a notoriously poorly conditioned eigenvalue problem. In fact, that's its claim to fame. Some of the values of `condeig(F)` are greater than 10^7. The computed eigenvalues should be sorted, `f = sort(eig(F))`. Then `f.*flipud(f) == 1` to only about seven digits, which is consistent with `condeig(F)`.

10.9. Compare three ways to compute singular values.
`svd(A)` is the recommended method.
`sqrt(eig(A'*A))` is actually somewhat faster, but less accurate because the small singular values are clobbered when you form `A'*A`.
`eig([0 A; A' 0])` could be eight times slower because the matrix is twice as big. It also requires more storage. Its accuracy is comparable with the recommended method.

10.10. I found the number of iterations required by `eigsvdgui` on random symmetric and nonsymmetric matrices to be to be suprisingly consistent. The nonsymmetric eigenvalue algorithm requires about $3.5n$ or $3.6n$, sometimes up to $4.0n$, iterations. The symmetric eigenvalue algorithm and the SVD algorithm both require about $1.7n$ or $1.8n$ iterations.

10.11. `A = diag(ones(n-1,1),-1) + diag(1,n-1)`
`eigsvdgui(A,'eig')` iterates forever. The QR algorithm leaves the matrix unchanged. This is the matrix that Wilkinson had in mind when he introduced the *ad hoc shift*. Our simple implementation does not have this important safety feature.
`eigsvdgui(A,'symm')`. Nothing unusual here.
`eigsvdgui(A,'svd')`. The SVD is computed without any arithmetic or iterations, just permutations.

10.12. (a) If $V\Sigma U' = X'$, then $U\Sigma V' = X$. Consequently the SVD of X and X' just swap U and V. However, in MATLAB 6.5 and earlier, *economy* SVD only works for tall, skinny matrices. (In MATLAB 7.0, we'll have it both ways.) Since we concatenated `R`, `G` and `B` together horizontally, we prefer to compute the economy SVD of the transpose.
(b) `imagesvd.m` does all the work. Enjoy!

10.13. (a) The female swings her pelvis a little more and keeps her arms closer to her body.
(b) See `walkerab.m`.
(c) See `walkerwave.m`.
(d) The five rows of subplots correspond to five components or postures. The three columns are x, y, z coordinates. Each subplot shows male and female components. The subplot in position (4,1) shows the most variation, but its scale is tiny compared to the others. It's hard to see much difference between male and female in these plots, even though we can see it easily with the gui.
(e) See `walkerphase.m`

10.14. `help sparse` says
```
Any elements of s which have duplicate values of i and j
are added together.
```
Consequently, `sparse(i,j,1)` counts duplicate paris.
Figure 10.1 shows `digraph.m` operating on itself. There are only 453 characters, and it's a computer program. But there are lots of comments, so even here we find the vowels.

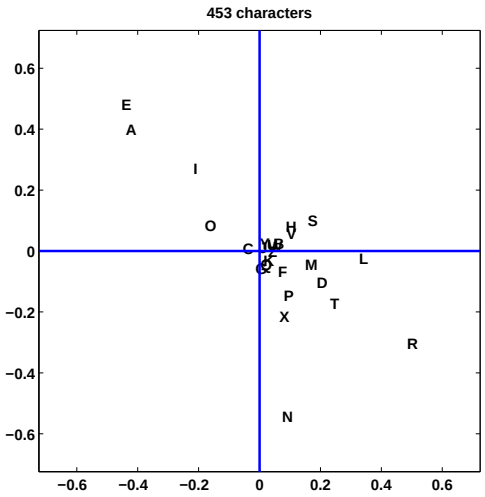


Figure 10.1. `digraph('digraph.m')`

10.15. `circlegen(h)`. See figure 10.2.
$\cos\theta = 1 - h^2/2, \quad \lambda = e^{\pm i\theta}$
If $\theta = 2\pi/p$ for integer p, the orbit is discrete with p points.
$h =$ `default`, $\theta = 2\pi/30$
$h = 1/\phi$, $\theta = 2\pi/10$
$h = \phi$, $\theta = 6\pi/30$
$h = 1.4140$, orbit creeps counterclockwise slowly
$h = \sqrt{2}$, $\theta = 2\pi/4$
$h = 1.4144$, orbit creeps clockwise slowly
$h < 2$, $|\lambda| < 1$, a bounded ellipse.
$h = 2, A^n = \pm \begin{pmatrix} 2n-1 & 2n \\ -2n & -(2n+1) \end{pmatrix}$, linear growth.
$h > 2$, $|\lambda| > 1$, exponential growth.

10.16. Euler's methods. See figure 10.3.
(a) Explicit. $\lambda = 1 \pm ih$, $|\lambda| > 1$, Increasing spiral.
(b) Implicit. $\lambda = 1/(1 \pm ih)$, $|\lambda| < 1$, Decreasing spiral.

10.17. See `circlegenrho.m`. `rho` and `kappa` are essentially equal, except when the orbit is only a discrete set of points, and when the step size gets very close to 2.

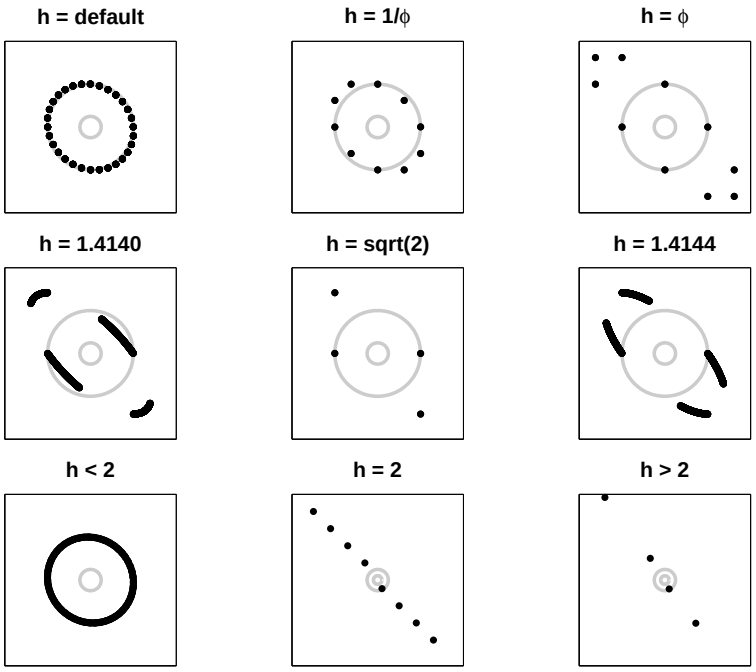


Figure 10.2. *Circle generator with various step sizes*

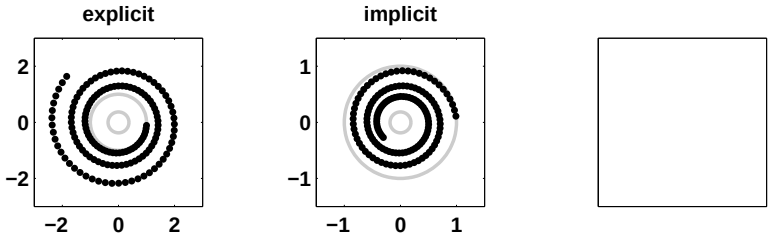


Figure 10.3. *Explicit and implicit circle generators*

Chapter 11

Partial Differential Equations

Exercises

11.1. (a) h should equal $1/(n+1)$.
(b) $(1/h)D$ is a one-sided approximation to the first derivative.
(c) D^TD and DD^T are equal to $-A$, except for the first or last row.
(d) $(1/h^4)A^2$ approximates the fourth derivative operator.
(e) `(1/h^2)*(kron(A,I)+kron(I,A))` is $\triangle_h$ for $[0,1] \times [0,1]$
(f) `plot(inv(full(-A)))` shows that the elements of A^{-1} are piecewise linear functions of the row and column indices, with breaks at the diagonal.

11.2. (a) See `bellshape.m`
(b) $u = (\sqrt{\pi}/2)(t\ \mathrm{erf}(t) - \mathrm{erf}(1)) + (1/2)(e^{-t^2} - e^{-1})$

11.3. See `waveguide.m`.

11.4. See `humpspdes.m`.
(a) The residual is on the order of 10^{-12}
(b) limit $t \to \infty$ $u(x,t) = u(x)$ from part (a).
(c) limit $t \to \infty$ $u(x,t)$ is the linear function interpolating the boundary values, `humps(0) = 5.1765` and `humps(1) = 16`.
(d) $u(x,t)$ is periodic in t with period 2.

11.5. See `peakspdes.m`.
(a) The residual is on the order of 10^{-12}
(b) limit $t \to \infty$ $u(x,y,t) = u(x,y)$ from part (a).
(c) limit $t \to \infty$ $u(x,y,t)$ is the harmonic function interpolating the boundary values, `peaks(x,y)`, for $|x| = 3$ or $|y| = 3$.
(d) Two-dimensional wave equation with `peaks` as the initial value. Solution is not periodic and does not approach a limit as $t \to \infty$.

11.6. Method of lines. *Later.*

11.7. See `ncm/pdegui`.

(a) n = number of grid points $\approx c/h^2$ where

Region	c
Square	4
L	3
H	3
Disc	28/9
Annulus	22/9
Heart	19/9
Drums	14/9

(b) For the heat equation, the maximum stable time step is $(1/4)h^2$. For the wave equation, the maximum stable time step is $(1/\sqrt{2})h$.

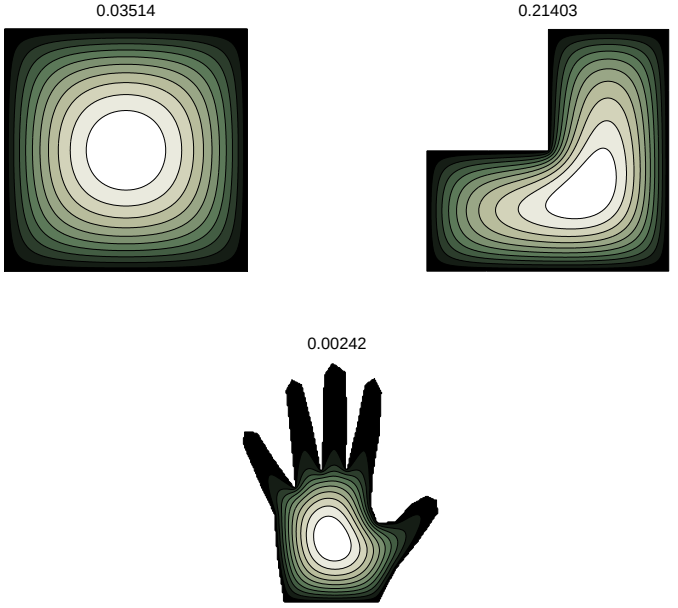
(c) The Poisson problem in `pdegui` is $\triangle_h u = 1$ and the eigenvalue problem with `index = 1` is $\triangle_h u = \lambda u$ with $u \geq 0$. Since both problems have positive source terms, the solutions are similar.

(d) Both `pdegui` and `membranetx` compute the eigenvalues and eigenfunctions of the L-shaped membrane. Even though their methods are very different, the resulting contour plots are identical, except for a 90° rotation.

(e) The regions `Drum1` and `Drum2` have different shapes, but the same eigenvalues. Before 1992, many people believed that this was not possible.

11.8. See `pdeguihand.m`.

11.9. See `capacity.m`.



11.10. See `ncm/pennymelt.m`

(a) What is the limiting behavior of $u(x, y, t)$ as $t \to \infty$? This is a hard question because we haven't been forthcoming about the boundary conditions. The code has

```
% Finite difference indices
[p,q] = size(U);
n = [2:p p];
s = [1 1:p-1];
e = [2:q q];
w = [1 1:q-1];
```

and

```
U = U + sigma*(U(n,:)+U(s,:)+U(:,e)+U(:,w)-4*U);
```

This amounts to Neumann boundary conditions. The discrete Laplacian is singular in this situation. The steady state satisfies

```
U(n,:)+U(s,:)+U(:,e)+U(:,w) = 4*U
```

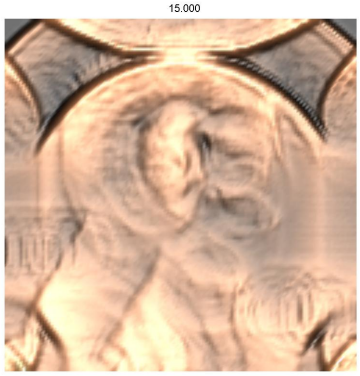
Thus, the steady state is a constant. What constant? With exact arithmetic, `mean(U(:))` would remain constant. The mean of the penny data is `101.8268`, so this is the theoretical steady state value. But with rounding error, we drift away from this value and the actual result depends upon the history of the computation. It takes a very long time to see the limiting value.

(b) The explicit algorithm is stable $\delta \leq 1/4$.

(c) The ADI algorithm is stable for all values of δ.

11.11. See `pennypois.m`.

11.12. See `pennywave.m`.



11.13. See `waves9.m`.

11.14. See `triplets.m`. It finds the high-multiplicity eigenvalues of the square, but does not know where they fit in the numbering of the eigenvalues of the L.
$50 = 1^2 + 7^2 = 7^2 + 1^2 = 5^2 + 5^2$
$65 = 8^2 + 1^2 = 7^2 + 4^2 = 4^2 + 7^2 = 1^2 + 8^2$

11.15. See `membranetx.m`, `membrane.mat`, and `membraneshow.m`.
The function `membranetx` looks for a file named `membrane.mat` that contains precomputed eigenvalues of the L-shaped membrane, together with information about their multiplicities. If the file is not available, `membranetx` can recompute it.
The statement `load membrane` should put two arrays of length 150, `lambdas` and `syms`, in the workspace. `lambdas(k)` is the `k`th eigenvalue. `syms(k) = 1` if the eigenfunction is symmetric about the center line and is not an eigenfunction of the square. `syms(k) = 2` if the eigenfunction is antisymmetric about the center line and is not an eigenfunction of the square. `syms(k) >= 3` for eigenfunctions that come from reflecting eigenfunctions of the square into the three squares that make up the L. `syms(k) = 4, 5, ...` for eigenvalues of multiplicity `2, 3, ...`
(a) `[sum(syms==1) sum(syms==2) sum(syms>=3)]/length(syms)` computes the fraction of eigenvalues for each symmetric class. It turns out that each of the three fractions is about 1/3.
(b) `k = min(find(syms==5)) = 105`. `lambdas(103:105)` are all equal to $50\pi^2$. See `membraneshow(103:105)`.
(c) `k = min(find(syms==6)) = 138`. `lambdas(135:138)` are all equal to $65\pi^2$. See `membraneshow(135:138)`.
(d) `membranetx` assumes that all the multiple eigenvalues of the L are also multiple eigenvalues of the square and hence integer multiples of π^2. It uses this information, instead of tolerances, to determine the rank of $A(\lambda)$ and select the requisite number of columns of V from the SVD for coefficients.
`pdegui` does not use any symmetry information, even though the eigenfunction symmetries of the L are preserved in the finite difference eigenvectors. Instead, it relies the Arnoldi algorithm in `eigs` to determine multiplicities.

11.16. `help cameratoolbar`

11.17. See `ncmlogo3d.m`.