Solutions Manual for

SCIENTIFIC COMPUTING

An Introductory Survey

Second Edition

Michael T. Heath
University of Illinois
at Urbana-Champaign

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York
San Francisco St. Louis Bangkok Bogotá Caracas Kuala Lumpur
Lisbon London Madrid Mexico City Milan Montreal New Delhi
Santiago Seoul Singapore Sydney Taipei Toronto

McGraw-Hill Higher Education

A Division of The McGraw-Hill Companies

Solutions Manual for
SCIENTIFIC COMPUTING: AN INTRODUCTORY SURVEY, SECOND EDITION

www.mhhe.com

Contents

Preface

This document provides solutions for the Exercises and Computer Problems in my book *Scientific Computing: An Introductory Survey, Second Edition*. To keep this manual within reasonable size, the solutions provided are quite concise, with no special attempt to make them pedagogically palatable. This should not be a problem, as the intended audience is instructors using the book in a course, who should need solutions only as a time saver or memory jogger. These solutions are **not** intended for direct consumption by students, and this document is not supposed to be obtainable by students anyway. Instructors should think of the solutions provided here as "condensed" versions — you may need to "add water and stir" to make them useful for instructional purposes.

Perhaps some explanation is in order concerning the Computer Problems. For reasons discussed in Section 1.4.4 of the book, I use `MATLAB` in my own classes, and consequently the solutions provided consist of `MATLAB` m-files. `MATLAB`, with its powerful, integrated computational and graphical capabilities and concise programming syntax, enables relatively brief solutions for most of the problems. A conventional programming language would have required much longer programs and would have required invoking a large number routines from external sources, the enormous range of choices for which would have made any particular solution hardly representative. In my experience, even though I allow my students to use any language and software they choose, the students quickly realize the advantages of `MATLAB` and use it for their homework. For reasons of space, the `MATLAB` programs provided in this manual are complete but compactly formatted and only lightly commented; they are not intended as models of good programming style. Again, this should not be a problem, as these programs are not intended for direct consumption by students, but only to provide the instructor with a starting point toward a pedagogically effective solution. Space limitations and production constraints also ruled out including the text or graphics output produced by the solution programs, but these can be easily reproduced simply by running the programs in `MATLAB`.

Answers are not given here for the many Review Questions in the book, as this would have defeated much of their purpose, which is to provide students with a self-test of their comprehension of the basic concepts covered in each chapter and to encourage them to review in the text any concepts of which they are uncertain. Unlike the Exercises and Computer Problems, the Review Questions are not suitable for homework assignments. The Review Questions are not difficult, and most of the answers can be found explicitly in the text simply by flipping back through the chapter, which is their main point. It is difficult to see how a separate listing of the answers to these questions could be used short of simply giving it wholesale to students, in which case they tend just to read the answers rather than honestly test their knowledge and review the text where needed. To avoid this temptation, I have refrained from listing those answers here.

Assembling and polishing solutions for well over 400 problems required a substantial effort over an extended period of time. I would like to acknowledge the contributions of a number of current or former students and teaching assistants, including David Alber, William Cochran, Michael Ham, Rebecca Hartman-Baker, Vanessa Lopez, Ryan Szypowski, Hanna VanderZee, and Wei Wang. I would also like to acknowledge the patience of my wife, Mona, during the months I was preoccupied with this enormous "homework assignment." I welcome any corrections or suggestions for improving the solutions given here, which can be communicated by email to `heath@cs.uiuc.edu`.

Michael T. Heath

Book website: `www.cse.uiuc.edu/heath/scicomp`

Chapter 1

Scientific Computing

Exercises

1.1. (a) Rel. err. $= 0.05/98.6 \approx 0.0005$, or about 0.05%. (b) Rel. err. $= 0.5/37 \approx 0.0135$, or about 1.35%.

1.2. (a) Abs. err. $\approx -1.416 \times 10^{-1}$, rel. err. $\approx -4.507 \times 10^{-2}$. (b) Abs. err. $\approx -1.593 \times 10^{-3}$, rel. err. $\approx -5.070 \times 10^{-4}$. (c) Abs. err. $\approx 1.264 \times 10^{-3}$, rel. err. $\approx 4.025 \times 10^{-4}$.

1.3. $r = (a-t)/t \Rightarrow rt = a - t \Rightarrow a = t + rt = t(1+r)$.

1.4. (a) Abs. err. $\approx \cos(x)h$. (b) Rel. err. $\approx \cot(x)h$. (c) Cond. no. $\approx |x\cot(x)|$. (d) This problem is highly sensitive for x near $k\pi$, where k is any integer.

1.5.

$$\text{Cond} = \frac{(|(x+\Delta x) - (y+\Delta y)) - (x-y)|/|x-y|}{|(|x+\Delta x| + |y+\Delta y|) - (|x|+|y|)|/(|x|+|y|)} \geq \frac{|\Delta x - \Delta y|/\epsilon}{(|\Delta x| + |\Delta y|)/1} \geq \frac{1}{\epsilon}\,.$$

Thus, subtraction is sensitive when ϵ is small, that is, when $x \approx y$.

1.6. (a) For $x = 0.1$, for. err. $\approx 1.6658 \times 10^{-4}$, back err. $\approx 1.6742 \times 10^{-4}$. For $x = 0.5$, for. err. $\approx 2.0574 \times 10^{-2}$, back. err. $\approx 2.3599 \times 10^{-2}$. For $x = 1.0$, for. err. $\approx 1.5853 \times 10^{-1}$, back. err. $\approx 5.7080 \times 10^{-1}$. (b) For $x = 0.1$, for. err. $\approx -8.3313 \times 10^{-8}$, back. err. $\approx -8.3732 \times 10^{-8}$. For $x = 0.5$, for. err. $\approx -2.5887 \times 10^{-4}$, back. err. $\approx -2.9496 \times 10^{-4}$. For $x = 1.0$, for. err. $\approx -8.1377 \times 10^{-3}$, back. err. $\approx -1.4889 \times 10^{-2}$.

1.7. (a) $2365.27 = 2.36527 \times 10^3$, and $0.0000512 = 5.12 \times 10^{-5}$, so $p = 6$, $U = 3$, and $L = -5$. (b) $2365.27 = 2.36527 \times 10^3$, and $0.0000512 = 0.00512 \times 10^{-2}$, so $p = 6$, $U = 3$, and $L = -2$.

1.8. (a) $y-x = 0.00123$, which contains 3 significant digits. (b) $x = 1.23456\times10^0$, $y = 1.23579\times10^0$, and $y-x = 1.23\times10^{-3}$, so $L=-3$ and $U=0$. (c) Yes, because the difference contains fewer significant digits than either x or y, so it can be represented as a subnormal number with exponent 0 without loss of precision.

1.9. (a) 5.101×10^8 km^2. (b) New surface area is 5.102×10^8, so change is 1.0×10^5. (c) With approximate formula, change in surface area is 1.601×10^5. (d) With six-digit arithmetic, original surface area is 5.00906×10^8, new surface area is 5.10065×10^8, difference between these is 1.59×10^5, and change in surface area from approximate formula is 1.60095×10^5, so approximate formula is more nearly correct than "exact" formula. (e) Computing the difference in surface areas incurs cancellation, from which the approximate formula does not suffer. The truncation error of the approximate formula is less than the rounding error of the exact formula using four-digit arithmetic, and hence the approximate formula yields a more accurate result.

1.10. (a) For x near 0. (b) $2x/(1-x^2)$.

1.11. $\log(x/y)$ is unlikely to give a better result because if $x\approx y$, then $(x/y)\approx 1$, and log function is sensitive for arguments near 1.

1.12. (a) $(x-y)(x+y)$ can be evaluated more accurately because x^2-y^2 suffers greater cancellation. (b) When $x\approx\pm y$, i.e., $|x-y|$ or $|x+y|$ is relatively small.

1.13. Divide each entry of $\boldsymbol{x}$ by the absolute value of its entry of largest magnitude, compute the norm of the rescaled vector, and then multiply the resulting norm by the scale factor. This will avoid both overflow (if all entries are large) and harmful underflow (if all entries are small) by normalizing the largest entry to one. Underflow can still occur, but only for entries that are very small relative to the others, which is not harmful because zero is a reasonable approximation in such cases.

1.14. Assuming a and b have the same sign, formula (b) is better because the result is guaranteed to lie in the interval $[a,b]$, and it also avoids the possibility of overflow. With 2-digit arithmetic, for example, formula (a) gives a "midpoint" of 0.7 for the interval $[0.67, 0.69]$, whereas formula (b) gives a midpoint of 0.68.

1.15. (a) Floating-point addition in toy system is in fact associative because $\epsilon_{\text{mach}} < \text{UFL}$. (b) For IEEE single precision $\epsilon_{\text{mach}} = 2^{-24}$, so $(1+2^{-25})+2^{-25} = 1$, whereas $1+(2^{-25}+2^{-25}) = 1+2^{-24}$, so addition is not associative.

1.16. For the toy floating-point system of Example 1.9, $\frac{1}{2}\beta^{1-p}$ gives $\epsilon_{\text{mach}} = 0.125$, the smallest number that perturbs 1 gives $\epsilon_{\text{mach}} = 0.25$, and the distance from 1 to the next larger floating-point number gives $\epsilon_{\text{mach}} = 0.25$.

1.17. (a) Minimum spacing is β^{L-p+1}. (b) Maximum spacing is β^{U-p+1}.

1.18. (a) $2(\beta-1)\beta^{p-1}(U-L+1)+1$. (b) $2(\beta^{p-1}-1)$.

1.19. (a) OFL $= 2^{128}(1-2^{24})$, UFL $= 2^{-126}$. (b) OFL unchanged, UFL $= 2^{-149}$.

1.20. (a) 0.00011001100110011001100. (b) 0.00011001100110011001101.

1.21. (a) Not necessarily (see part b). For practical floating-point systems, however, ϵ_{mach} is a machine number. (b) The toy system of Example 1.9 is an example, as $\epsilon_{\text{mach}} = 0.25$ and UFL $= 0.5$.

1.22. (a) $b^2 - 4ac = (3.34)^2 - 4 \times 1.22 \times 2.28 = 11.2 - 11.1 = 0.1$. (b) 0.0292. (c) $(0.1 - 0.0292)/0.0292 \approx 2.425$.

1.23. (a) UFL $= \beta^L = 10^{-98}$. (b) $x - y = 6.87 \times 10^{-97} - 6.81 \times 10^{-97} = 0.06 \times 10^{-97} = 6.00 \times 10^{-99} \approx 0$. (c) With gradual underflow, result is 0.60×10^{-98}.

1.24. First consider $\beta = 2$. The hypothesis implies that the exponents of x and y differ by at most 1. If the exponents agree, and the mantissas have p digits, then $x - y$ has at most p digits. If the exponents differ by 1, then $x - y$ has the form $\pm(d_0 d_1.d_2 \cdots d_p)_\beta \times \beta^e$, where e is the exponent of the smaller of x and y. However, because $x - y$ is no larger than the smaller of x and y, the leading digit d_0 must be zero. To obtain a counterexample for $\beta > 2$, choose $x = 2 + \beta^{1-p}$ and $y = 2\beta^{-1} + \beta^{-p}$. Then $x - y = 1 + (\beta - 2)\beta^{-1} + (\beta - 1)\beta^{-p}$ requires $p + 1$ digits to represent it exactly.

1.25. Compute `s=a*b` in normal single precision. Then compute `t=mpy_add(a,b,-s)`. Variable `s` now contains the most significant half, and variable `t` the least significant half, of the double precision value for the product `a*b`.

1.26. If $x = 2c/(-b \mp \sqrt{b^2 - 4ac})$, then

$$\begin{aligned}
ax^2 + bx + c &= \frac{a4c^2}{(-b \mp \sqrt{b^2 - 4ac})^2} + \frac{b2c}{-b \mp \sqrt{b^2 - 4ac}} + c \\
&= \frac{4ac^2 + 2bc(-b \mp \sqrt{b^2 - 4ac}) + c(-b \mp \sqrt{b^2 - 4ac})^2}{(-b \mp \sqrt{b^2 - 4ac})^2} \\
&= 4ac^2 + 2bc(-b \mp \sqrt{b^2 - 4ac}) + c(-b \mp \sqrt{b^2 - 4ac})^2 \\
&= 4ac^2 - 2b^2c \mp 2bc\sqrt{b^2 - 4ac} + b^2c \pm 2cb\sqrt{b^2 - 4ac} + c(b^2 - 4ac) \\
&= 4ac^2 - 2b^2c + b^2c + b^2c - 4ac^2 = 0.
\end{aligned}$$

1.27. See page 28.

Computer Problems

1.1.
```
function cp01_1 % error in Stirling's approximation to factorial
n = [1:10]'; trueValue = gamma(n+1);            % n! = gamma(n+1)
approxValue = sqrt(2*pi*n) .* (n/exp(1)).^n;    % Stirling's formula
absError = approxValue-trueValue; relError = absError./trueValue;
disp('       n       True value  Approx. value    Abs. error     Rel. error')
fprintf('      %2d    %13.6g %13.6g   %13.6e  %13.6e\n',...
  [n'; trueValue'; approxValue'; absError'; relError']); disp(' ');
disp('Absolute error increases, relative error decreases.')
```

1.2. function cp01_02 % determine properties of floating-point system

```
n = 0;                  % determine unit roundoff
while 1+2^n > 1
  n = n-1;
end; n = n+1;
disp('                  Computed value      Hexadecimal representation')
fprintf('Unit roundoff     %13.6g          %bx\n', 2^n, 2^n);
n = 0;                  % determine underflow level
while 0+2^n > 0
  n = n-1;
end; n = n+1; fprintf('Underflow level   %13.6g          %bx\n', 2^n, 2^n);
n = 0;                  % determine overflow level
while isfinite(2^(n+1))
  n = n+1;
end; ofl = 2^n;
while 1
  n = n-1; temp = 2^n;
  if isinf(ofl+temp), break;
  else ofl = ofl+temp; end
end; fprintf('Overflow level    %13.6g          %bx\n', ofl, ofl);
```

1.3. function cp01_03 % simple estimator for unit roundoff

```
disp('(a) Trick works because 4/3 is not exactly representable in binary.')
fprintf('(b) abs(3 * (4/3 - 1) - 1) = %g\n', abs(3 * (4/3 - 1) - 1));
fprintf('    Matlab value for eps   = %g\n', eps);
disp('(c) Trick would not work for beta=3 because 4/3 would be exactly')
disp('    representable.')
```

1.4. function cp01_04 % compute e from limit definition

```
k = [1:20]'; n = 10.^k;
trueValue = exp(1)*ones(size(k)); approxValue = (1+1./n).^n;
absError = approxValue-trueValue; relError = absError./trueValue;
disp('   k    True value      Approx. value     Abs. error     Rel. error')
fprintf('  %2d  %14.12f   %14.12f   %13.6e  %13.6e\n', ...
  [k'; trueValue'; approxValue'; absError'; relError']); disp(' ');
disp('Overall error initially decreases but later increases, as decreasing')
disp('truncation error is offset by increasing rounding error.')
```

1.5. function cp01_05 % accuracy of function evaluation

```
k = [1:15]'; x = 10.^(-k);
disp('                    f(x) = (exp(x)-1)/x as x -> 0'); disp(' ');
trueValue = ones(size(k)); approxValue = (exp(x)-1)./x;
absError = approxValue-trueValue; relError = absError./trueValue;
disp('   k    True value      Approx. value     Abs. error    Rel. error')
fprintf('  %2d  %14.12f   %14.12f   %13.6e %13.6e\n', ...
  [k'; trueValue'; approxValue'; absError'; relError']); disp(' ');
disp('                f(x) = (exp(x)-1)/log(exp(x)) as x -> 0'); disp(' ')
approxValue = (exp(x)-1)./log(exp(x));
absError = approxValue-trueValue; relError = absError./trueValue;
disp('   k    True value      Approx. value     Abs. error    Rel. error')
fprintf('  %2d  %14.12f   %14.12f   %13.6e %13.6e\n', ...
```

```
  [k'; trueValue'; approxValue'; absError'; relError']); disp(' ');
disp('First approach suffers severe cancellation in computing numerator.')
disp('In second approach, errors in numerator and denominator cancel each')
disp('other in the division.')
```

1.6.
```
function cp01_06 % generating equally-spaced mesh points
disp('Method 2 is better; Method 1 accumulates rounding error.')
a = 0; b = 1; n = 333333; h = (b-a)/n;
fprintf('For a = %d, b = %d, n = %d, x(%d) should equal 1.\n', a, b, n, n);
x = zeros(n+1,1); x2= zeros(n+1,1); x(1) = a;
for k=2:n+1
  x(k) = x(k-1) + h;
end
disp(sprintf('Method 1, x(%d) = %23.16e',n,x(n+1,1)))
x2(1:n+1) = a+h*[0:n]; fprintf('Method 2, x(%d) = %23.16e\n', n, x2(n+1,1));
```

1.7.
```
function cp01_07 % finite difference approximation to derivative
true_value = sec(1.0)^2; x = 1.0; k = [0:16]; h = 1.0./(10.^k);
forward_diff = abs((tan(x+h)-tan(x))./h);
error_a = abs(forward_diff-true_value);
centered_diff = abs((tan(x+h)-tan(x-h))./(2*h));
error_b = abs(centered_diff-true_value);
loglog(h,error_a,'b-',h,error_b,'r--');
title('Computer Problem 1.7 - Finite Difference Approximations');
xlabel('stepsize h'); ylabel('magnitude of error');
legend('forward difference','centered difference');
```

1.8.
```
function cp01_08(nmax) % "convergence" of divergent series
sum = 0; n = 1; term = 1/n;
fprintf('Summing 1/n for n from 1 to %d\n\n', nmax);
while sum+term > sum & n < nmax
  sum = sum+term; n = n+1; term = 1/n;
  if mod(n,nmax/10) == 0
    fprintf('Partial sum: n = %d, sum = %14.12f\n', n, sum);
  end
end; fprintf('Final sum:   n = %d, sum = %14.12f\n', nmax, sum);
```

1.9.
```
function cp01_09 % compute exponential function from series definition
disp('(b) Terminate summation when next term does not perturb sum.'); disp(' ');
x = [-20; -15; -10; -5; -1; 1; 5; 10; 15; 20];
approxValue  = zeros(size(x)); approxValue2 = zeros(size(x));
approxValue3 = zeros(size(x)); trueValue = exp(x);
for i=1:length(x)
  approxValue(i) = approx_exp(x(i));
  approxValue2(i) = approx_exp2(x(i));
  approxValue3(i) = approx_exp3(x(i));
end; absError = approxValue-trueValue; relError = absError./trueValue;
disp('(c) Approximate exp(x) by 1 + x + x^2/2 + ....'); disp(' ')
disp(sprintf('   x         True value         Approx. value %s', ...
  '      Abs. error     Rel. error'))
fprintf(' %5.1f   %14.12e   %14.12e  %13.6e  %13.6e\n', ...
```

```
  [x'; trueValue'; approxValue'; absError'; relError']); disp(' ');
absError2 = approxValue2-trueValue; relError2 = absError2./trueValue;
disp('(d) Use exp(-x) = 1/exp(x) for x < 0.'); disp(' ')
fprintf('   x         True value         Approx. value %s\n', ...
  '      Abs. error     Rel. error');
fprintf(' %5.1f   %14.12e   %14.12e  %13.6e  %13.6e\n', ...
  [x'; trueValue'; approxValue2'; absError2'; relError2']); disp(' ');
absError3 = approxValue3-trueValue; relError3 = absError3./trueValue;
disp('(e) Group same-sign terms to compute exp(x) for x < 0.'); disp(' ')
disp(sprintf('   x         True value         Approx. value %s', ...
  '      Abs. error     Rel. error'))
fprintf(' %5.1f   %14.12e   %14.12e  %13.6e  %13.6e\n', ...
  [x'; trueValue'; approxValue3'; absError3'; relError3']); disp(' ');
disp('This approach merely postpones cancellation until the end, so it does not')
disp('produce any improvement over straightforward summation.')

function [approx] = approx_exp(x)
approx = 1; n = 1; nfact = 1; xn = x; term = xn/nfact;
while approx+term ~= approx
  approx = approx+term; n = n+1; nfact = nfact*n; xn = xn*x; term = xn/nfact;
end

function [approx] = approx_exp2(x)
if x < 0, approx = 1/approx_exp(-x);
else approx = approx_exp(x); end

function [approx] = approx_exp3(x)
approx = 1; n = 1; nfact = 1; xn = x; term = xn/nfact;
sum_pos = 1; sum_neg = 0;
while approx+term ~= approx
  if term < 0 sum_neg = sum_neg+term;
  else sum_pos = sum_pos+term; end
  approx = sum_pos+sum_neg;
  n = n+1; nfact = nfact*n; xn = xn*x; term = xn/nfact;
end
```

1.10. `function cp01_10 % robust quadratic equation solver`

```
a = [ 6  6*10^154 0  1     1         10^(-155)];
b = [ 5  5*10^154 1 -10^5 -4        -10^155];
c = [-4 -4*10^154 1  1     3.999999  10^155];
root1 = zeros(size(a)); root2 = zeros(size(a));
for i=1:length(a)
  [root1(i),root2(i)] = QuadSolve(a(i),b(i),c(i));
end
disp('      a             b             c            Root1         Root2')
fprintf('%10.7g    %10.7g    %10.7g    %10.6g    %10.7g\n',...
  [a; b; c; root1; root2]);

function [root1, root2] = QuadSolve(unscaled_a,unscaled_b,unscaled_c)
% Returns Nan for roots that don't exist or can't be computed.
```

```
root1 = NaN; root2 = NaN; % initialize to NaN
factor = max(abs([unscaled_a unscaled_b unscaled_c]));
a = unscaled_a/factor; b = unscaled_b/factor; c = unscaled_c/factor;
disc = b*b-4*a*c;         % compute discriminant
% handle degenerate cases first
if abs(a) < eps           % linear case (a is zero)
  if abs(b) >= eps, root1 = -c/b; end
elseif disc < 0           % imaginary roots
  fprintf('imaginary roots for %13.6g,%13.6g,%13.6g\n',...
    unscaled_a,unscaled_b,unscaled_c);
elseif sqrt(disc) < eps   % repeated root
  root1 = -b/(2*a); root2 = root1;
else
  temp1 = sqrt(disc);     % choose formula that avoids cancellation
  if(b > 0), root1 = (2*c)/(-b-temp1); root2 = (-b-temp1)/(2*a);
  else root1 = (-b+temp1)/(2*a); root2 = (2*c)/(-b+temp1); end
end
```

1.11.

```
function cp01_11 % cubic equation solver
a = [ 0  1  6  6*10^154 0  1     1          10^(-154)];
b = [ 6 -3  5  5*10^154 1 -10^5 -4         -10^154];
c = [ 5  3 -4 -4*10^154 1  1     3.9999999  10^154];
d = [-4 -1  0  0        0  0     0          0]; x = zeros(size(a));
for i=1:length(a)
  x(i) = CubicSolve(a(i),b(i),c(i),d(i));
end
disp('                    Cubic Equation                                Real Root')
fprintf('%8.1g x^3 + %8.1g x^2 + %8.1g x + %8.1g = 0   %13.6g\n', [a; b; c; d; x]);

function [root] = CubicSolve(a,b,c,d)
% CubicSolve -- Solver for cubic equations; returns one real root
root = NaN;                   % initialize to NaN
factor = max(abs([a b c d]));  % scale coefficients
a = a/factor; b = b/factor; c = c/factor; d = d/factor;
% handle degenerate cases first
if(abs(a) < eps)              % quadratic case (a is zero)
  if(abs(b) < eps)            % linear case (a and b are zero)
    if(abs(c) >= eps), root = -d/c; end
  else
    [root,ignore] = QuadSolve(b,c,d);
  end
else
  p = c/a/3-b*b/a/a/9; q = (2*b*b*b/a/a/a-9*b*c/a/a+27*d/a)/27;
  d = p*p*p+q*q/4;            % compute discriminant
  if d < 0, y = 2*sqrt(abs(p))*cos(acos(-q/2/sqrt(p*p*p))/3);
  else
    temp1 = -q/2+sqrt(d); temp2 = -q/2-sqrt(d);
    u = abs(temp1)^(1/3); v = abs(temp2)^(1/3);
    if temp1 < 0, u = -u; end
    if temp2 < 0, v = -v; end
```

```
    y = u+v;
  end
  root = y-b/a/3;
end

function [root1, root2] = QuadSolve(unscaled_a,unscaled_b,unscaled_c)
% QuadSolve -- Robust solver for quadratic equations with real roots
% Returns NaN for roots that don't exist (or can't be computed).
root1 = NaN; root2 = NaN;      % initialize to NaN
factor = max(abs([unscaled_a unscaled_b unscaled_c])); % scale input values
a = unscaled_a/factor; b = unscaled_b/factor; c = unscaled_c/factor;
disc = b*b-4*a*c;
% handle degenerate cases first
if(abs(a) < eps)               % linear case (a is zero)
  if(abs(b) >= eps), root1 = -c/b; end
elseif(disc < 0)               % imaginary roots
  fprintf('imaginary roots for %13.6g, %13.6g, %13.6g\n',...
    unscaled_a, unscaled_b, unscaled_c);
elseif(sqrt(disc) < eps)       % repeated root
  root1 = -b/(2*a); root2 = root1;
else                           % choose formula that avoids cancellation
  if(b > 0), root1 = (2*c)/(-b-sqrt(disc)); root2 = (-b-sqrt(disc))/(2*a);
  else root1 = (-b+sqrt(disc))/(2*a); root2 = (2*c)/(-b+sqrt(disc)); end
end
```

1.12.

```
function cp10_12 % one- and two-pass formulas for standard deviation
x = [10000000.2 10000000.1 10000000.3 10000000.1 10000000.3 10000000.1 ...
  10000000.3 10000000.1 10000000.3 10000000.1 10000000.3]';
n = length(x); avg = mean(x); devs = x-avg;
two_pass = sqrt(sum((devs.*devs))/(n-1));
one_pass = sqrt((sum((x.*x))-n*avg*avg)/(n-1));
fprintf('Mean:                                %15.8e\n', avg);
fprintf('Std. dev. from two-pass formula:    %g\n', two_pass);
fprintf('Std. dev. from one-pass formula:    %g\n', one_pass);
fprintf('Std. dev. from MATLAB std function: %g\n', std(x));
```

1.13.

```
function cp01_13 % compound interest
a = 100; r = 0.05; n = [1 4 12 365]; f1 = zeros(size(n)); f2 = zeros(size(n));
for k=1:length(n)
  f1(k) = a;
  for j=1:n(k)
    f1(k) = f1(k)*(1+r/n(k));
  end
  f2(k) = a*exp(n(k)*log(1+r/n(k)));
end
disp(sprintf('Initial investment = %g, Interest rate = %g\n', a, r))
disp('Compounds                    Final Balance')
disp(' per Year           Formula 1             Formula 2')
fprintf(' %8d  %22.15f  %22.15f\n', [n;f1;f2]);
```

1.14.

```
function cp01_14 % evaluation of polynomial
```

```
x = [0.995:0.0001:1.005]; poly1 = (x-1.0).^6;
poly2 = x.^6-6*x.^5+15*x.^4 -20*x.^3+15*x.^2-6*x+1;
plot(x,poly1,'b--',x,poly2,'r-');
title('Computer Problem 1.14 - Computing (x-1)^6');
xlabel('x'); ylabel('p(x)'); legend('compact formula','expanded formula');
```

1.15.
```
function cp01_15 % robust computation of Euclidean norm
x = 10^200*ones(5,1); norm1 = NaiveNorm(x); norm2 = RobustNorm(x);
fprintf('Naive implementation:   norm = %13.6g\n', norm1);
fprintf('Robust implementation:  norm = %13.6g\n', norm2);
fprintf('MATLAB built-in 2-norm: norm = %13.6g\n', norm(x,2));

function [value] = NaiveNorm(x)
% NaiveNorm -- straightforward computation of 2-norm
n = length(x); x = x(:); value = sqrt(sum(x.*x));

function [value] = RobustNorm(x)
% RobustNorm -- robust computation of 2-norm
n = length(x); x = x(:); scale = 0; ssq = 1;
for i=1:n
  if x(i) ~= 0
    absxi = abs(x(i));
    if scale < absxi, ssq = 1+ssq*(scale/absxi)^2; scale = absxi;
    else ssq = ssq+(absxi/scale)^2; end
  end
end; value = scale*sqrt(ssq);
```

1.16.
```
function cp01_16(n) % summation algorithms
x = rand(n,1);
% This problem requires single precision, which is not available in MATLAB.
% This solution uses simulated 6-digit decimal arithmetic where single
% precision is called for.
% (a) - Sum in order, keep sum in double-precision variable
sum1 = 0;
for i=1:n
  sum1 = sum1+KeepSignificantDigits(x(i),6);
end
% (b) - Sum in order, keep sum in single-precision variable
sum2 = 0;
for i=1:n
  sum2 = FinPrecSum(sum2,x(i),6);
end
% (c) - Kahan's method
sum3 = KeepSignificantDigits(x(1),6); c = 0;
for i=2:n
  y = FinPrecDiff(x(i),c,6); t = FinPrecSum(sum3,y,6);
  c = FinPrecDiff(FinPrecDiff(t,sum3,6),y,6); sum3 = t;
end
% (d) - Sum in order of increasing magnitude
x4 = sort(x); sum4 = 0;
for i=1:n
```

```
  sum4 = FinPrecSum(sum4,x4(i),6);
end
% (e) - Sum in order of decreasing magnitude
x5 = x4(n:-1:1); sum5 = 0;
for i=1:n
  sum5 = FinPrecSum(sum5,x5(i),6);
end
disp('                                    Method')
disp(sprintf(['  n          a            b            c' ...
  '            d            e']))
fprintf('%d  %11.6g  %11.6g  %11.6g  %11.6g  %11.6g\n',...
  n, sum1, sum2, sum3, sum4, sum5);

function RoundedValue = KeepSignificantDigits(value,digits)
% KeepSignificantDigits -- Round value to specified number of digits.
if value == 0, RoundedValue = 0;
else
  logval = log10(value);
  if isinf(logval), RoundedValue = 0;
  else
    mag = fix(logval);
    RoundedValue = 10^(mag-digits+1)*round(value/10^(mag-digits+1));
  end
end

function [diff, ResultString] = FinPrecDiff(minuend,subtrahend,digits)
% FinPrecDiff -- Compute difference in finite-precision arithmetic.
%    With two output arguments, generates strings showing details.
rm = KeepSignificantDigits(minuend,digits);
rs = KeepSignificantDigits(subtrahend,digits);
diff = KeepSignificantDigits(minuend-subtrahend,digits);
if nargout > 1
  str1 = strcat('   %',int2str(digits+8),'.',int2str(digits),'g');
  str2 = strcat('   -%',int2str(digits+7),'.',int2str(digits),'g'); str3 = '_';
  for i=1:(digits+7)
    str3 = strcat(str3,'_');
  end
  ResultString = {sprintf(str1, minuend); sprintf(str2, subtrahend); ...
    sprintf('   %s',str3); sprintf(str1, diff)};
end

function [sum, ResultString] = FinPrecSum(addend1,addend2,digits)
% FinPrecSum -- Compute sum in finite-precision arithmetic.
%    With two output arguments, generates strings showing details.
a1 = KeepSignificantDigits(addend1,digits);
a2 = KeepSignificantDigits(addend2,digits);
sum = KeepSignificantDigits(addend1+addend2,digits);
if nargout > 1
  str1 = strcat('   %',int2str(digits+8),'.',int2str(digits),'g');
  str2 = strcat('   -%',int2str(digits+7),'.',int2str(digits),'g'); str3 = '_';
```

```
  for i=1:(digits+7)
    str3 = strcat(str3,'_');
  end
  ResultString = {sprintf(str1, addend1); sprintf(str2, addend2); ...
    sprintf('   %s',str3); sprintf(str1, sum)};
end
```

1.17.
```
function cp01_17 % solution of difference equation
n = 60; x = zeros(n,1); xtrue = zeros(n,1); x(1) = 1/3; x(2) = 1/12;
for k=3:n
  x(k) = 2.25*x(k-1)-0.5*x(k-2);
end;
xtrue(1:n,1) = (4.^(1-[1:n]'))/3;
semilogy([1:n]',xtrue,'r-',[1:n]',x,'b--')
title('Computer Problem 1.17 - Difference Equation')
xlabel('k'); ylabel('x(k)'); legend('computed solution','exact solution');
disp('Rounding error causes other solution component to dominate eventually.');
```

1.18.
```
function cp01_18 % solution of difference equation
n = [1:20]; x = zeros(size(n)); x(1) = 11/2; x(2) = 61/11;
for k=2:19
  x(k+1) = 111-(1130-3000/x(k-1))/x(k);
end;
disp(' k           x(k)'); fprintf('%2d   %13.6g\n', [n;x]);
plot(n,x,'b-'); title('Computer Problem 1.18 - Recurrence');
xlabel('k'); ylabel('x(k)'); disp(' ');
disp('Rounding error causes other solution component to dominate eventually.');
```

Chapter 2

Systems of Linear Equations

Exercises

2.1. (1) $\Rightarrow$ (2): Suppose $\boldsymbol{A}^{-1}$ exists, so that $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}$. Recall that the determinant of a product is the product of the determinants, so we have $\det(\boldsymbol{A})\det(\boldsymbol{A}^{-1}) = \det(\boldsymbol{I}) = 1$, and hence $\det(\boldsymbol{A}) \neq 0$. (2) $\Rightarrow$ (3): Suppose that $\det(\boldsymbol{A}) \neq 0$. Recall that adding a scalar multiple of one column of $\boldsymbol{A}$ to another column of $\boldsymbol{A}$ does not change the value of its determinant. Thus, no nontrivial linear combination of columns of $\boldsymbol{A}$ can produce the zero vector, since the determinant would then be zero. Thus $\operatorname{rank}(\boldsymbol{A}) = n$. (3) $\Rightarrow$ (4): Suppose that $\operatorname{rank}(\boldsymbol{A}) = n$. Then there is no vector $\boldsymbol{z}$ such that $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{o}$, since otherwise the columns of $\boldsymbol{A}$ would be linearly dependent, and we would then have $\operatorname{rank}(\boldsymbol{A}) < n$. (4) $\Rightarrow$ (1): Suppose that $\boldsymbol{A}$ annihilates no nonzero vector. Then $\operatorname{span}(\boldsymbol{A})$ is of dimension n, and hence $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ must have a solution for any vector $\boldsymbol{b}$. Thus, the columns of $\boldsymbol{A}^{-1}$ are given by solving this system for $\boldsymbol{b} = \boldsymbol{e}_j$, $j = 1, \ldots, n$.

2.2. If each row sum of $\boldsymbol{A}$ is zero, then we have $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{o}$, where $\boldsymbol{z} = \boldsymbol{e}$, the vector whose entries are all 1. Thus, by property 4 in the definition, $\boldsymbol{A}$ is singular.

2.3. If $\boldsymbol{A}$ is singular, then by property 4 in the definition, there is a nonzero vector $\boldsymbol{z}$ such that $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{o}$. Thus, if $\boldsymbol{x}$ is a solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, then we have $\boldsymbol{A}(\boldsymbol{x}+\gamma\boldsymbol{z}) = \boldsymbol{b}$ for any scalar γ, of which there are infinitely many choices.

2.4. (a) We have

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

so by property 4 of the definition, the matrix is singular. (b) If $\boldsymbol{b} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$, then $\boldsymbol{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + \gamma \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1-\gamma \\ 1+\gamma \\ 1-\gamma \end{bmatrix}$ is a solution for any value of γ.

2.5. Since

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{A}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} = \boldsymbol{A},$$

i.e., $\boldsymbol{A}$ is its own inverse.

2.6. $\boldsymbol{A}^2 = \boldsymbol{O}$ implies that $\boldsymbol{A}$ annihilates each of its own columns. If any of those columns is nonzero, the by taking that column as $\boldsymbol{z}$ in property 4 of the definition, we see that $\boldsymbol{A}$ must be singular. Otherwise, $\boldsymbol{A} = \boldsymbol{O}$, which again is singular.

2.7. (a) $\det(\boldsymbol{A}) = 1 - (1+\epsilon)(1-\epsilon) = \epsilon^2$. ($b$) In floating-point arithmetic, the computed value of $\det(\boldsymbol{A})$ is zero when $\epsilon < \sqrt{\epsilon_{\text{mach}}}$. ($c$)

$$\boldsymbol{A} = \begin{bmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1-\epsilon & 1 \end{bmatrix} \begin{bmatrix} 1 & 1+\epsilon \\ 0 & \epsilon^2 \end{bmatrix} = \boldsymbol{LU}.$$

(d) In floating-point arithmetic, the computed $\boldsymbol{U}$ is singular when $\epsilon < \sqrt{\epsilon_{\text{mach}}}$.

2.8. (a) Let $\boldsymbol{C} = \boldsymbol{AB}$. Then we have

$$c_{ji} = \sum_{k=1}^{n} a_{jk} b_{ki} = \sum_{k=1}^{n} b_{ki} a_{jk},$$

which is the (i,j) entry of $\boldsymbol{B}^T\boldsymbol{A}^T$. Thus, $(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$. ($b$) Assuming $\boldsymbol{A}$ and $\boldsymbol{B}$ are nonsingular, we have $(\boldsymbol{AB})(\boldsymbol{B}^{-1}\boldsymbol{A}^{-1}) = \boldsymbol{A}(\boldsymbol{BB}^{-1})\boldsymbol{A}^{-1} = \boldsymbol{AA}^{-1} = \boldsymbol{I}$, so $(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$.

2.9. By Exercise 2.8, we have $\boldsymbol{A}^T(\boldsymbol{A}^{-1})^T = (\boldsymbol{A}^{-1}\boldsymbol{A})^T = \boldsymbol{I}^T = \boldsymbol{I}$, so $(\boldsymbol{A}^{-1})^T = (\boldsymbol{A}^T)^{-1}$.

2.10. (a) Let $\boldsymbol{p}_i$ denote the ith column of $\boldsymbol{P}$. By definition, $\boldsymbol{p}_i$ contains exactly one 1 and zeros elsewhere, so we have $\boldsymbol{p}_i^T\boldsymbol{p}_i = 1$. Moreover, no two distinct columns of $\boldsymbol{P}$ contain a 1 in the same row, so $\boldsymbol{p}_i^T\boldsymbol{p}_j = 0$ for $i \neq j$. Thus, $\boldsymbol{P}^T\boldsymbol{P} = \boldsymbol{I}$, and hence $\boldsymbol{P}^T = \boldsymbol{P}^{-1}$. ($b$) Suppose the first row of $\boldsymbol{P}$ has a 1 in column j_1. Then let $\boldsymbol{P}_1$ be the permutation matrix that interchanges 1 and j_1. Note that $\boldsymbol{PP}_1^T$ has a 1 in the $(1,1)$ entry and zeros elsewhere in row and column one. Now suppose that the second row of $\boldsymbol{PP}_1^T$ has a 1 in column j_2. Then let $\boldsymbol{P}_2$ be the permutation matrix that interchanges 2 and j_2. Continue in this manner up to $\boldsymbol{P}_n$. We will then have $\boldsymbol{PP}_1^T \cdots \boldsymbol{P}_n^T = \boldsymbol{I}$, or $\boldsymbol{P} = \boldsymbol{P}_n \cdots \boldsymbol{P}_1$, which is the required product of interchanges.

2.11. See Algorithm 2.1 on page 65.

2.12. Since each lower triangular entry in $\boldsymbol{L}$ enters into exactly one multiplication (and one addition), the number of multiplications (and additions) is simply the number of lower triangular entries, which is

$$\sum_{k=1}^{n} k = n(n+1)/2 = n^2/2 + n/2,$$

whose dominant term is $n^2/2$.

2.13. First solve the lower triangular system $\boldsymbol{L}_1\boldsymbol{x} = \boldsymbol{b}$ for $\boldsymbol{x}$ by forward-substitution, then solve the lower triangular system $\boldsymbol{L}_2\boldsymbol{y} = \boldsymbol{c} - \boldsymbol{B}\boldsymbol{x}$ for $\boldsymbol{y}$ by forward-substitution.

2.14. (1) By construction, $\boldsymbol{M}_k$ is a triangular matrix with unit main diagonal. Thus, $\det(\boldsymbol{M}_k) = 1$, and hence $\boldsymbol{M}_k$ is nonsingular. (2) By definition of the outer product, $\boldsymbol{m}_k\boldsymbol{e}_k^T$ is an $n \times n$ matrix with all entries zero except for having $m_{k+1}, \ldots, m_n$ below the diagonal in column k. Thus, $\boldsymbol{M}_k = \boldsymbol{I} - \boldsymbol{m}_k\boldsymbol{e}_k^T$. (3) $\boldsymbol{M}_k(\boldsymbol{I}+\boldsymbol{m}_k\boldsymbol{e}_k^T) = (\boldsymbol{I}-\boldsymbol{m}_k\boldsymbol{e}_k^T)(\boldsymbol{I}+\boldsymbol{m}_k\boldsymbol{e}_k^T) = \boldsymbol{I}-\boldsymbol{m}_k\boldsymbol{e}_k^T+\boldsymbol{m}_k\boldsymbol{e}_k^T-\boldsymbol{m}_k\boldsymbol{e}_k^T\boldsymbol{m}_k\boldsymbol{e}_k^T = \boldsymbol{I}$, so $\boldsymbol{I} + \boldsymbol{m}_k\boldsymbol{e}_k^T = \boldsymbol{M}_k^{-1}$. (4) $\boldsymbol{M}_k\boldsymbol{M}_j = \boldsymbol{I} - \boldsymbol{m}_k\boldsymbol{e}_k^T - \boldsymbol{m}_j\boldsymbol{e}_j^T + \boldsymbol{m}_k\boldsymbol{e}_k^T\boldsymbol{m}_j\boldsymbol{e}_j^T = \boldsymbol{I} - \boldsymbol{m}_k\boldsymbol{e}_k^T - \boldsymbol{m}_j\boldsymbol{e}_j^T$, since $\boldsymbol{e}_k^T\boldsymbol{m}_k = 0$. Thus, the product is essentially the "union."

2.15. (a) Let $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{M}$, with $\boldsymbol{L}$ and $\boldsymbol{M}$ lower triangular. Then

$$a_{ij} = \sum_{k=1}^{n} \ell_{ik}m_{kj} = 0$$

for $i < j$, and hence $\boldsymbol{A}$ is lower triangular. (b) Column k of $\boldsymbol{L}^{-1}$ is given by the solution to the linear system $\boldsymbol{L}\boldsymbol{x} = \boldsymbol{e}_k$. If $\boldsymbol{L}$ is lower triangular, then from the forward substitution process, we see that the first $k-1$ entries of $\boldsymbol{x}$ are zero, since the first $k-1$ entries of $\boldsymbol{e}_k$ are zero, and thus $\boldsymbol{L}^{-1}$ is lower triangular.

2.16. (a) $\boldsymbol{A} = \begin{bmatrix} 1 & a \\ c & b \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ c & 1 \end{bmatrix} \begin{bmatrix} 1 & a \\ 0 & b-ac \end{bmatrix} = \boldsymbol{L}\boldsymbol{U}$. ($b$) $\boldsymbol{A}$ is singular when $b - ac = 0$.

2.17. $\boldsymbol{A} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} = \boldsymbol{L}\boldsymbol{U}.$

2.18. Without loss of generality, we can assume that $\boldsymbol{L}$ is scaled to have unit diagonal, so that the LU factorization has the form

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \ell & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix} = \boldsymbol{L}\boldsymbol{U}.$$

From the first row of $\boldsymbol{A}$, we see that we must have $u_{11} = 0$ and $u_{12} = 1$. But then from the second row of $\boldsymbol{A}$ we see that we must have $\ell u_{11} = 1$, which is impossible since $u_{11} = 0$. Therefore, no such LU factorization exists.

2.19. (a) No, the algorithm is not numerically stable. (b) The algorithm is unstable because having the largest entry in each column lie on the diagonal initially does not guarantee that this will continue to be the case after each subsequent step of the factorization process. In fact, the algorithm will fail outright if a diagonal entry becomes zero. For example, one step of elimination with the following matrix yields

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix},$$

at which point the algorithm fails.

2.20. If $\boldsymbol{A}$ is diagonally dominant by columns, then the magnitude of each diagonal entry exceeds the sum of the magnitudes of all the off-diagonal entries in the same column. This remains true after each step of elimination because each multiplier is of magnitude at most 1. Thus, no interchanges will occur because the diagonal entry is already the largest entry in the pivot column at each stage.

2.21. Solve $\boldsymbol{Cy} = \boldsymbol{b}$ for $\boldsymbol{y}$. Compute $\boldsymbol{z} = \boldsymbol{Ab}$. Compute $\boldsymbol{u} = \boldsymbol{y} + \boldsymbol{z}$. Compute $\boldsymbol{v} = 2\boldsymbol{Au} + \boldsymbol{u}$. Solve $\boldsymbol{Bx} = \boldsymbol{v}$ for $\boldsymbol{x}$.

2.22. At step k of LU factorization by Gaussian elimination, about $(n-k+1)^2$ multiplications (and a simliar number of additions) are required. Thus, the total number of multiplications required is given by

$$\sum_{k=1}^{n}(n-k+1)^2 = \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6},$$

so the dominant term is $n^3/3$ multiplications (and a simliar number of additions).

2.23. Computing the inverse of a matrix requires an LU factorization, which requires about $n^3/3$ multiplications, and then solving $2n$ triangular systems (one forward-subsititution and one back-substitution with each column of the identity matrix as right-hand side), which would normally require n^3 multiplications, for a total of $4n^3/3$ multiplications. In this context, however, we can take advantage of the sparsity of the right-hand sides to reduce the cost of the forward-substitutions with the columns of the identity matrix. Specifically, with $\boldsymbol{e}_j$ as right-hand side, the number of multiplications in the forward-substitution is given by

$$\sum_{k=j+2}^{n}(n-k+1) = \sum_{k=1}^{n-j-1} k = \frac{j^2-(2n-1)j+n^2-n}{2},$$

so that summing over $j = 1, 2, \ldots, n-2$ we obtain a total of $(n^3-3n^2+2n)/6$ multiplications for all of the forward-substitutions. The n back-substitutions still cost a total of about $n^3/2$ multiplications, so the total cost of computing the inverse is about $n^3/3 + n^3/6 + n^3/2 = n^3$.

2.24. At step k of Gauss-Jordan elimination, about $n(n-k+1)$ multiplications (and a simliar number of additions) are required. Thus, the total number of multiplications required is given by

$$\sum_{k=1}^{n} n(n-k+1) = n\sum_{k=1}^{n} k^2 = n\frac{n(n+1)}{2} = \frac{n^3+n^2}{2},$$

so the dominant term is $n^3/2$ multiplications (and a simliar number of additions).

2.25. (a) Let $\boldsymbol{A} = \boldsymbol{u}\boldsymbol{v}^T$. Then $\text{rank}(\boldsymbol{A}) \leq 1$ because the rank of a product cannot exceed the rank of either factor. To see that $\text{rank}(\boldsymbol{A}) = 1$, note that column j of $\boldsymbol{A}$ is given by $v_j\boldsymbol{u}$. By assumption, $\boldsymbol{u} \neq \boldsymbol{o}$, and at least one of the entries v_j of $\boldsymbol{v}$ is nonzero, so $\boldsymbol{A}$ has at least one nonzero column. (b) If $\text{rank}(\boldsymbol{A}) = 1$, then $\boldsymbol{A}$ has only one linearly independent column, call it $\boldsymbol{u}$, and hence any other column, say column j must be a multiple of it, say $v_j\boldsymbol{u}$. The v_j form the entries of a vector $\boldsymbol{v}$ such that $\boldsymbol{A} = \boldsymbol{u}\boldsymbol{v}^T$.

2.26. (a) First note that $(\boldsymbol{I} - \boldsymbol{u}\boldsymbol{v}^T)(\boldsymbol{I} - \sigma\boldsymbol{u}\boldsymbol{v}^T) = \boldsymbol{I} - \boldsymbol{u}\boldsymbol{v}^T - \sigma\boldsymbol{u}\boldsymbol{v}^T + \sigma\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{u}\boldsymbol{v}^T$, so the inverse exists if $-\boldsymbol{u}\boldsymbol{v}^T - \sigma\boldsymbol{u}\boldsymbol{v}^T + \sigma\boldsymbol{u}(\boldsymbol{v}^T\boldsymbol{u})\boldsymbol{v}^T = (-1-\sigma+\sigma\boldsymbol{v}^T\boldsymbol{u})\boldsymbol{u}\boldsymbol{v}^T = 0$, which holds if $-1-\sigma+\sigma\boldsymbol{v}^T\boldsymbol{u} = 0$, which holds if $\sigma = 1/(\boldsymbol{v}^T\boldsymbol{u} - 1)$. Thus, the condition $\boldsymbol{v}^T\boldsymbol{u} \neq 1$ ensures that $\boldsymbol{I} - \boldsymbol{u}\boldsymbol{v}^T$ is nonsingular. (b) From the derivation in part (a), we have $\sigma = 1/(\boldsymbol{v}^T\boldsymbol{u} - 1)$. (c) For an elementary elimination matrix we have $\boldsymbol{u} = \boldsymbol{m} = [\,0 \quad \ldots \quad 0 \quad m_{k+1} \quad \ldots \quad m_n\,]^T$, $\boldsymbol{v} = \boldsymbol{e}_k$, and $\sigma = 1/(\boldsymbol{v}^T\boldsymbol{u} - 1) = (1/(\boldsymbol{e}_k^T\boldsymbol{m} - 1) = 1/-1 = -1$.

2.27. $(\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)(\boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{u}(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}) =$

$$\boldsymbol{I} + \boldsymbol{u}(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1} - \boldsymbol{u}\boldsymbol{v}^T\boldsymbol{A}^{-1} - \boldsymbol{u}\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1}$$

$$= \boldsymbol{I} + \boldsymbol{u}\left((1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1} - 1 - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\right)\boldsymbol{v}^T\boldsymbol{A}^{-1}$$

$$= \boldsymbol{I} + \boldsymbol{u}\left(\frac{1-(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}) - \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}}{1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}}\right)\boldsymbol{v}^T\boldsymbol{A}^{-1} = \boldsymbol{I},$$

so $\boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{u}(1-\boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})^{-1}\boldsymbol{v}^T\boldsymbol{A}^{-1} = (\boldsymbol{A} - \boldsymbol{u}\boldsymbol{v}^T)^{-1}$.

2.28. $(\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^T)(\boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1}) =$

$$\boldsymbol{I}+\boldsymbol{U}(\boldsymbol{I}-\boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1}-\boldsymbol{U}\boldsymbol{V}^T\boldsymbol{A}^{-1}-\boldsymbol{U}\boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I}-\boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1}$$

$$= \boldsymbol{I} + \boldsymbol{U}\left((\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1} - \boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\right)\boldsymbol{V}^T\boldsymbol{A}^{-1}$$

$$= \boldsymbol{I} + \boldsymbol{U}\left(\boldsymbol{I} - (\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U}) - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U}\right)(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1} = \boldsymbol{I},$$

so $\boldsymbol{A}^{-1} + \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{I} - \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}^T\boldsymbol{A}^{-1} = (\boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^T)^{-1}$.

2.29. Property 1:

$$\begin{aligned}
\|\boldsymbol{x}\|_1 &= \sum_{i=1}^{n} |x_i| > 0 \quad \text{if} \quad \boldsymbol{x} \neq \boldsymbol{o}. \\
\|\boldsymbol{x}\|_2 &= \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} > 0 \quad \text{if} \quad \boldsymbol{x} \neq \boldsymbol{o}. \\
\|\boldsymbol{x}\|_\infty &= \max_{1 \leq i \leq n} |x_i| > 0 \quad \text{if} \quad \boldsymbol{x} \neq \boldsymbol{o}.
\end{aligned}$$

Property 2:

$$\begin{aligned}
\|\gamma \boldsymbol{x}\|_1 &= \sum_{i=1}^{n} |\gamma x_i| = \sum_{i=1}^{n} |\gamma| \cdot |x_i| = |\gamma| \sum_{i=1}^{n} |x_i| = |\gamma| \cdot \|\boldsymbol{x}\|_1. \\
\|\gamma \boldsymbol{x}\|_2 &= \left(\sum_{i=1}^{n} |\gamma x_i|^2 \right)^{1/2} = \left(\sum_{i=1}^{n} |\gamma|^2 |x_i|^2 \right)^{1/2} = \left(|\gamma|^2 \sum_{i=1}^{n} |x_i|^2 \right)^{1/2} \\
&= |\gamma| \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} = |\gamma| \cdot \|\boldsymbol{x}\|_2. \\
\|\gamma \boldsymbol{x}\|_\infty &= \max_{1 \leq i \leq n} |\gamma x_i| = \max_{1 \leq i \leq n} (|\gamma| \cdot |x_i|) = |\gamma| \max_{1 \leq i \leq n} |x_i| = |\gamma| \cdot \|\boldsymbol{x}\|_\infty.
\end{aligned}$$

Property 3:

$$\begin{aligned}
\|\boldsymbol{x} + \boldsymbol{y}\|_1 &= \sum_{i=1}^{n} |x_i + y_i| \leq \sum_{i=1}^{n} (|x_i| + |y_i|) = \sum_{i=1}^{n} |x_i| + \sum_{i=1}^{n} |y_i| = \|\boldsymbol{x}\|_1 + \|\boldsymbol{y}\|_1. \\
\|\boldsymbol{x} + \boldsymbol{y}\|_2^2 &= \sum_{i=1}^{n} |x_i + y_i|^2 = \sum_{i=1}^{n} |x_i|^2 + 2 \sum_{i=1}^{n} |x_i| \cdot |y_i| + \sum_{i=1}^{n} |y_i|^2,
\end{aligned}$$

so by the Cauchy-Schwarz inequality,

$$\begin{aligned}
\|\boldsymbol{x} + \boldsymbol{y}\|_2^2 &\leq \sum_{i=1}^{n} |x_i|^2 + 2 \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} \left(\sum_{i=1}^{n} |y_i|^2 \right)^{1/2} + \sum_{i=1}^{n} |y_i|^2 \\
&= \left(\left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} + \left(\sum_{i=1}^{n} |y_i|^2 \right)^{1/2} \right)^2 = \left(\|\boldsymbol{x}\|_2 + \|\boldsymbol{y}\|_2 \right)^2. \\
\|\boldsymbol{x} + \boldsymbol{y}\|_\infty &= \max_{1 \leq i \leq n} |x_i + y_i| \leq \max_{1 \leq i \leq n} (|x_i| + |y_i|) = \max_{1 \leq i \leq n} |x_i| + \max_{1 \leq i \leq n} |y_i| \\
&= \|\boldsymbol{x}\|_\infty + \|\boldsymbol{y}\|_\infty.
\end{aligned}$$

2.30. There are two approaches: one can either use the definition of a matrix norm induced by a vector norm to obtain a proof for any value of p, or one can use the maximum column sum and maximum row sum formulas to obtain proofs specifically for $p = 1$ and $p = \infty$, respectively. Using the first approach, the first

three properties of a matrix norm follow directly from the corresponding properties for vector norms. Using the second approach, proofs of the first three properties are analogous to the proofs for the corresponding vector norms. We therefore focus on the consistency conditions, properties 4 and 5. Using the first approach, property 5 follows trivially from the definition of a matrix norm induced by a vector norm. From property 5, we then have

$$\|\boldsymbol{AB}\| = \max_{\boldsymbol{x}\neq\boldsymbol{o}} \frac{\|\boldsymbol{ABx}\|}{\|\boldsymbol{x}\|} \leq \max_{\boldsymbol{x}\neq\boldsymbol{o}} \frac{\|\boldsymbol{A}\|\cdot\|\boldsymbol{Bx}\|}{\|\boldsymbol{x}\|} \leq \max_{\boldsymbol{x}\neq\boldsymbol{o}} \frac{\|\boldsymbol{A}\|\cdot\|\boldsymbol{B}\|\cdot\|\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \|\boldsymbol{A}\|\cdot\|\boldsymbol{B}\|.$$

Using the second approach, to prove property 5 we have

$$\begin{aligned}\|\boldsymbol{Ax}\|_1 &= \sum_{i=1}^{n}\Big|\sum_{j=1}^{n} a_{ij}x_j\Big| \leq \sum_{i=1}^{n}\sum_{j=1}^{n}|a_{ij}|\cdot|x_j| = \sum_{j=1}^{n}\sum_{i=1}^{n}|a_{ij}|\cdot|x_j| \\ &= \sum_{j=1}^{n}|x_j|\cdot\sum_{i=1}^{n}|a_{ij}| \leq \sum_{j=1}^{n}|x_j|\cdot\max_{1\leq j\leq n}\sum_{i=1}^{n}|a_{ij}| = \|\boldsymbol{A}\|_1\cdot\|\boldsymbol{x}\|_1,\end{aligned}$$

and

$$\begin{aligned}\|\boldsymbol{Ax}\|_\infty &= \max_{1\leq i\leq n}\Big|\sum_{j=1}^{n} a_{ij}x_j\Big| \leq \max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}|\cdot|x_j| \leq \max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}|\cdot\max_{1\leq j\leq n}|x_j| \\ &= \max_{1\leq j\leq n}|x_j|\cdot\max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}| = \|\boldsymbol{A}\|_\infty\cdot\|\boldsymbol{x}\|_\infty.\end{aligned}$$

Property 4 is proved similarly for both norms.

2.31. Property 1: Follows directly from positive definiteness of $\boldsymbol{A}$. Property 2: $\|\gamma\boldsymbol{x}\|_A = ((\gamma\boldsymbol{x})^T\boldsymbol{A}(\gamma\boldsymbol{x}))^{1/2} = (\gamma^2\boldsymbol{x}^T\boldsymbol{Ax})^{1/2} = |\gamma|(\boldsymbol{x}^T\boldsymbol{Ax})^{1/2} = |\gamma|\cdot\|\boldsymbol{x}\|_A$. Property 3: Since $\boldsymbol{A}$ is positive definite, it has a Cholesky factorization $\boldsymbol{A} = \boldsymbol{LL}^T$. Thus, for any vector $\boldsymbol{x}$, we have $\|\boldsymbol{x}\|_A^2 = \boldsymbol{x}^T\boldsymbol{Ax} = \boldsymbol{x}^T\boldsymbol{LL}^T\boldsymbol{x} = (\boldsymbol{Lx})^T(\boldsymbol{Lx}) = \|\boldsymbol{Lx}\|_2^2$. Thus, the triangle inequality for the $\boldsymbol{A}$-norm follows from the same result for the 2-norm.

2.32. (a) The proof is completely analogous to the corresponding proof for the vector ∞-norm given in Exercise 2.29. (b) The proof is completely analogous to the corresponding proof for the vector 2-norm given in Exercise 2.29.

2.33. The claim is false. For a counterexample, take $\boldsymbol{A} = \mathrm{diag}(2, 1)$. Then for any subordinate matrix norm, $\|\boldsymbol{A}\| = 2$, so $\|\boldsymbol{A}\|^{-1} = 1/2$, whereas $\|\boldsymbol{A}^{-1}\| = 1$.

2.34. (a) If $\boldsymbol{A}$ were singular, then there would be some vector $\boldsymbol{x} \neq \boldsymbol{o}$ such that $\boldsymbol{Ax} = \boldsymbol{o}$. But then $\boldsymbol{x}^T\boldsymbol{Ax} = 0$, contradicting the assumption that $\boldsymbol{A}$ is positive definite. Thus, $\boldsymbol{A}$ must be nonsingular. (b) If $\boldsymbol{A}^{-1}$ were not positive definite, then there would be some vector $\boldsymbol{x} \neq \boldsymbol{o}$ such that $\boldsymbol{x}^T\boldsymbol{A}^{-1}\boldsymbol{x} = 0$. Let $\boldsymbol{z} = \boldsymbol{A}^{-1}\boldsymbol{x}$. Then $\boldsymbol{z}^T\boldsymbol{Az} = \boldsymbol{z}^T\boldsymbol{A}^T\boldsymbol{z} = \boldsymbol{z}^T\boldsymbol{A}^T\boldsymbol{A}^{-1}\boldsymbol{Az} = (\boldsymbol{Az})^T\boldsymbol{A}^{-1}(\boldsymbol{Az}) = \boldsymbol{x}^T\boldsymbol{A}^{-1}\boldsymbol{x} = 0$, contradicting the assumption that $\boldsymbol{A}$ is positive definite. Thus, $\boldsymbol{A}^{-1}$ must be positive definite.

2.35. $\boldsymbol{A}$ is symmetric because $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^T = (\boldsymbol{B}\boldsymbol{B}^T)^T = \boldsymbol{A}^T$. Let $\boldsymbol{x}$ be any nonzero vector. Then $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{B}\boldsymbol{B}^T\boldsymbol{x} = (\boldsymbol{B}^T\boldsymbol{x})^T(\boldsymbol{B}^T\boldsymbol{x}) > 0$ unless $\boldsymbol{B}^T\boldsymbol{x} = 0$, which would contradict the assumption that $\boldsymbol{B}$ is nonsingular. Thus, $\boldsymbol{A}$ must be positive definite.

2.36. See Algorithm 2.7.

2.37. (a) We have

$$\boldsymbol{e}_1^T\boldsymbol{B}\boldsymbol{e}_1 = \begin{bmatrix} 1 \\ \boldsymbol{o} \end{bmatrix}^T \begin{bmatrix} \alpha & \boldsymbol{a}^T \\ \boldsymbol{a} & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} 1 \\ \boldsymbol{o} \end{bmatrix} = \alpha,$$

and hence $\alpha > 0$, since $\boldsymbol{B}$ is positive definite. Let $\boldsymbol{x}$ be any nonzero n-vector. Then we similarly have

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix}^T \begin{bmatrix} \alpha & \boldsymbol{a}^T \\ \boldsymbol{a} & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} > 0,$$

so that $\boldsymbol{A}$ must be positive definite. (b) Let $\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{A} - \boldsymbol{a}\boldsymbol{a}^T/\sqrt{\alpha}$. Then

$$\begin{bmatrix} \sqrt{\alpha} & \boldsymbol{O} \\ \boldsymbol{a}/\sqrt{\alpha} & \boldsymbol{L} \end{bmatrix} \begin{bmatrix} \sqrt{\alpha} & \boldsymbol{a}^T/\sqrt{\alpha} \\ \boldsymbol{O} & \boldsymbol{L}^T \end{bmatrix} = \begin{bmatrix} \alpha & \boldsymbol{a}^T \\ \boldsymbol{a} & \boldsymbol{A} \end{bmatrix}$$

gives the required Cholesky factorization.

2.38. (a) We have

$$\boldsymbol{e}_{n+1}^T\boldsymbol{B}\boldsymbol{e}_{n+1} = \begin{bmatrix} \boldsymbol{o} \\ 1 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \\ \boldsymbol{a}^T & \alpha \end{bmatrix} \begin{bmatrix} \boldsymbol{o} \\ 1 \end{bmatrix} = \alpha,$$

and hence $\alpha > 0$, since $\boldsymbol{B}$ is positive definite. Let $\boldsymbol{x}$ be any nonzero n-vector. Then we similarly have

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \\ \boldsymbol{a}^T & \alpha \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix} > 0,$$

so that $\boldsymbol{A}$ must be positive definite. (b) Let $\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{A}$ and let $\boldsymbol{\ell} = \boldsymbol{L}^{-1}\boldsymbol{a}$. Then

$$\begin{bmatrix} \boldsymbol{L} & \boldsymbol{O} \\ \boldsymbol{\ell}^T & \sqrt{\alpha - \boldsymbol{\ell}^T\boldsymbol{\ell}} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}^T & \boldsymbol{\ell} \\ \boldsymbol{O} & \sqrt{\alpha - \boldsymbol{\ell}^T\boldsymbol{\ell}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{a} \\ \boldsymbol{a}^T & \alpha \end{bmatrix}$$

gives the required Cholesky factorization.

2.39. Including only the highest-order terms, the number of multiplications (and a similar number of additions) required for Cholesky factorization is given by

$$\sum_{k=1}^{n} \sum_{j=k+1}^{n} (n-j) \approx \sum_{k=1}^{n} \sum_{j=1}^{k} j \approx \sum_{k=1}^{n} \frac{k^2}{2} \approx \frac{n^3}{6}.$$

2.40. At stage k of Gaussian elimination with partial pivoting for a matrix of bandwidth β, at most the first β entries below the diagonal are nonzero. With partial pivoting, the pivot chosen will therefore be at most β rows below the diagonal. Thus, in the worst case, when the pivot chosen is in row $k+\beta$, interchanging row k with row $k+\beta$ will place nonzeros in columns k through $k+2\beta$ of (the new) row k. Thus, the expansion of the (upper) bandwidth is by at most a factor of two. Note that the lower bandwidth does not grow, because the first $k-1$ columns are already zero below the diagonal anyway.

2.41. (a) Let $\boldsymbol{T}$ be a nonsingular tridiagonal matrix, with nonzero entries on its main diagonal and first sub- and super-diagonals. One way to see that $\boldsymbol{T}^{-1}$ is dense is to note that all the cofactors of $\boldsymbol{T}$ (i.e., determinants of minors of $\boldsymbol{T}$) are in general nonzero, so that its adjoint matrix, $\text{adj}(\boldsymbol{T})$, is dense, and hence $\boldsymbol{T}^{-1} = \text{adj}(\boldsymbol{T})/\det(\boldsymbol{T})$ is dense. Another way to see that $\boldsymbol{T}^{-1}$ is dense is to consider the LU factorization $\boldsymbol{T} = \boldsymbol{L}\boldsymbol{U}$, where $\boldsymbol{L}$ is lower bidiagonal and $\boldsymbol{U}$ is upper bidiagonal. Then we can compute $\boldsymbol{T}^{-1}$ as the solution to the successive linear systems $\boldsymbol{L}\boldsymbol{Y} = \boldsymbol{I}$ and $\boldsymbol{U}\boldsymbol{X} = \boldsymbol{Y}$. Note that because of the special right-hand side, $\boldsymbol{Y}$ will be lower triangular, but then $\boldsymbol{X} = \boldsymbol{T}^{-1}$ fills in completely, in general. (b) Solving a tridiagonal system using Gaussian elimination and back substitution requires $\mathcal{O}(n)$ work and $\mathcal{O}(n)$ storage. Solving a tridiagonal system by explicit inversion and matrix multiplication requires $\mathcal{O}(n^3)$ work and $\mathcal{O}(n^2)$ storage.

2.42. (a) Given an upper triangular matrix $\boldsymbol{U}$, the following algorithm overwrites $\boldsymbol{U}$ with $\boldsymbol{U}^{-1}$, accessing only the upper triangular portion of the array. An analogous algorithm works for a lower triangular matrix.

```
for k = n to 1
    u_kk = 1/u_kk
    for i = k − 1 to 1
        t = 0
        for j = i + 1 to k
            t = t + u_ij u_jk
        end
    u_ik = −t/u_ii
    end
end
```

(b) Given a matrix $\boldsymbol{A}$, one could compute its LU factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ in place, overwriting the upper triangle of $\boldsymbol{A}$ with $\boldsymbol{U}$ and the strict lower triangle of $\boldsymbol{A}$ with the strict lower triangle of $\boldsymbol{L}$, not storing the unit diagonal of $\boldsymbol{L}$. The triangular matrices $\boldsymbol{U}$ and $\boldsymbol{L}$ could then be inverted in place using the algorithms from part (a). This would effectively give one a *representation* of $\boldsymbol{A}^{-1}$, but to obtain $\boldsymbol{A}^{-1}$ explicitly, one would have to compute $\boldsymbol{A}^{-1} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}$ in place, and this is impossible because there is no order in which to compute the product that does not overwrite some entries that will still be needed subsequently.

2.43. If

$$\begin{bmatrix} \boldsymbol{A} & -\boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{A} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{c} \end{bmatrix},$$

then we have $\boldsymbol{Ax} - \boldsymbol{By} = \boldsymbol{b}$ and $\boldsymbol{Bx} + \boldsymbol{Ay} = \boldsymbol{c}$, so that

$$\boldsymbol{Cz} = (\boldsymbol{A} + i\boldsymbol{B})(\boldsymbol{x} + i\boldsymbol{y}) = \boldsymbol{Ax} - \boldsymbol{By} + i(\boldsymbol{Bx} + \boldsymbol{Ay}) = \boldsymbol{b} + i\boldsymbol{c} = \boldsymbol{d}.$$

This is not a particularly good method for solving this problem because it requires twice as much storage and more than twice as much work as solving the original system directly using complex arithmetic.

Computer Problems

2.1.
```
function cp02_01 % solution of singular or nearly singular system
A = [0.1 0.2 0.3; 0.4 0.5 0.6; 0.7 0.8 0.9]; b = [0.1; 0.3; 0.5];
disp('(a) A is singular because A*z=0, where z=[1, -2, 1]^T.  Solution set')
disp('    is all vectors of the form y + gamma z, where y is any particular')
disp('    solution to A*y=b, such as y = [1/3,1/3,0]^T.')
disp('(b) In exact arithmetic, LU factorization would still work but solution')
disp('    process would fail in back-substitution due to zero on diagonal of')
disp('    triangular factor U.'); disp(' ')
disp('(c) Solution computed by Gaussian elimination is'); x = A\b
disp('    which lies in the solution set described earlier, since for')
gamma = x(3), z = [1; -2; 1]; y = [1/3; 1/3; 0]; y_plus_gamma_z = y+gamma*z
disp('In double precision arithmetic, we would expect no digits of accuracy in')
disp('the solution because'); CondA = cond(A)
```

2.2.
```
function cp02_02 % Sherman-Morrison updating of linear system solution
A = [2 4 -2; 4 9 -3; -2 -1 7 ]; b = [2; 8; 10]; c = [4; 8; -6]; [L, U] = lu(A);
disp('(a) Solution to original linear system:'); x = U\(L\b)
disp('(b) Solution to system with new right-hand side:'); y = U\(L\c)
u = [2; 0; 0]; v = [0; 1; 0]; z = U\(L\u); y = x;
disp('(c) Solution to modified system:'); x = y+((v'* y)/(1-v'*z))*z
```

2.3.
```
function cp02_03 % linear system for plane truss in equilibrium
% chapter2, machine problem 3.
A = zeros(13,13); b = zeros(13,1); alpha = 1/sqrt(2);
A(1,2) = 1; A(1,6) = -1; A(2,3) = 1; b(2) = 10; A(3,1) = alpha;
A(3,4) = -1; A(3,5) = -alpha; A(4,1) = alpha; A(4,3) = 1; A(4,5) = alpha;
A(5,4) = 1; A(5,8) = -1; A(6,7) = 1; A(7,5) = alpha; A(7,6) = 1;
A(7,9) = -alpha; A(7,10) = -1; A(8,5) = alpha; A(8,7) = 1; A(8,9) = alpha;
b(8) = 15; A(9,10) = 1; A(9,13) = -1; A(10,11) = 1; b(10) = 20; A(11,8) = 1;
A(11,9) = alpha; A(11,12) = -alpha; A(12,9) = alpha; A(12,11) = 1;
A(12,12) = alpha; A(13,13) = 1; A(13,12) = alpha;
disp('Solution vector of member forces:'); f = A\b
```

2.4.
```
function cp02_04 % matrix condition number estimation
A1 = [10 -7 0; -3 2 6; 5 -1 5]; A2 = [-73 78 24; 92 66 25; -80 37 10];
disp('Matrix 1, 1-norm'); estcond(A1, 1); disp(' ');
disp('Matrix 1, infinity-norm'); estcond(A1, inf); disp(' ');
```

```
disp('Matrix 2, 1-norm'); estcond(A2, 1); disp(' ')
disp('Matrix 2, infinity-norm'); estcond(A2, inf); disp(' ');

function [c] = estcond(A, p)
norm_A = norm(A, p); [L,U,P] = lu(A); n = size(A,1);
% part (a), transposed triangular solves with special rhs
v = zeros(n,1); v(1) = 1/U(1,1);
for i=2:n
  tot = 0;
  for j=1:i-1;
    tot = tot-U(j,i)*v(j);
  end
  if tot > 0, tot = tot+1;
  else tot = tot-1; end
  v(i) = tot/U(i,i);
end
for i=n:-1:1
  tot = v(i);
  for j=i+1:n
    tot = tot-L(j,i)*v(j);
  end
  v(i) = tot;
end
y = P*v; z = U\(L\(P*y));
fprintf('Condition estimate, method a: %e\n', norm_A*norm(z,p)/norm(y,p));
% part (b), several random choices for y
maxratio = 0;
for k=1:5
  y = rand(n,1); z = A\y; t = norm(z,p)/norm(y,p);
  if t > maxratio, maxratio = t;
  end
end
fprintf('Condition estimate, method b: %e\n', norm_A*maxratio);
fprintf('MATLAB condest function:      %e\n', condest(A));
fprintf('MATLAB cond function:         %e\n', cond(A,p));
fprintf('Actual condition number:      %e\n', norm_A*norm(inv(A),p));
```

2.5.
```
function cp02_05 % iterative improvement of linear system solution
A = [21 67 88 73; 76 63  7 20; 0 85 56 54; 19.3 43 30.2 29.4];
b = [141; 109; 218; 93.7]; format long; [L,U,P] = lu(A);
disp('(a) Initial solution'); x = U\(L\(P*b))
disp('(b) Initial residual'); r = b-A*x, disp('(c)')
for i = 1:3
  z =  U\(L\(P*r)); disp('Improved solution'); x = x+z
  disp('New residual'); r = b-A*x
end
```

2.6.
```
function cp02_06 % solve linear system with Hilbert matrix
n = 1; condH = 1; disp(' n     resid_norm      error_norm   digits      cond')
while condH < 1e+16
  n = n+1; H = hilb(n); condH = cond(H); x_true = ones(n,1);
```

```
  b = H*x_true; x = H\b; r_norm = norm(b-H*x,inf);
  e_norm = norm(x-x_true,inf); digits = -log10(e_norm);
  fprintf('%2d   %13.6e   %13.6e   %4.1f   %13.6e\n', n, r_norm,...
    e_norm, digits, condH)
end
disp(' '); disp('Condition number is exponential in n.')
disp('Number of correct digits is about 16 minus log_10 of condition number.')
```

2.7. `function cp02_07 % growth factor in Gaussian elimination with partial pivoting`

```
disp('(a) With partial pivoting, entries of U grow exponentially.')
disp('With complete pivoting, entries of U do not grow.')
n = 1; condU = 1; disp('(b)')
disp(' n     resid_norm      error_norm        condA           condU')
while condU < 1e+16
  n = n+1; A = eye(n,n)-tril(ones(n,n),-1); A(1:n,n) = ones(n,1); condA = cond(A);
  x_true = ones(n,1); b = A*x_true; [L, U] = lu(A); condU = cond(U); x = U\(L\b);
  r_norm = norm(b-A*x,inf); e_norm = norm(x-x_true,inf);
  fprintf('%2d   %13.6e   %13.6e   %13.6e   %13.6e\n', n, r_norm,...
    e_norm, condA, condU)
end
```

2.8. `function cp02_08 % effect of scaling linear system`

```
disp(' n     resid_norm      error_norm       r_norm_s       e_norm_s')
for n = 2:25
  A = rand(n,n); x_true = ones(n,1); b = A*x_true; x = A\b;
  r_norm = norm(b-A*x,inf); e_norm = norm(x-x_true,inf); d = zeros(n,1);
  for i = 1:n
    d(i) = 2^(2*(i-n/2));
  end
  D = diag(d); A_s = D*A; b_s = D*b; x_s = A_s\(b_s);
  r_norm_s = norm(b_s-A_s*x_s,inf); e_norm_s = norm(x_s-x_true,inf);
  fprintf('%2d   %13.6e   %13.6e   %13.6e   %13.6e\n', n, r_norm,...
    e_norm, r_norm_s, e_norm_s)
end
```

2.9. `function cp02_09 % effect of pivoting on accuracy of Gaussian elimination`

```
A = zeros(2,2); b = zeros(2,1); x = zeros(2,1); z = zeros(2,1); x_true = [1; 1];
disp('(a) Gaussian elimination without pivoting')
disp(' k        x(1)            x(2)          res_norm        err_norm')
for k = 1:10
  epsilon = 10^(-2*k); A = [epsilon 1; 1   1]; b = [1+epsilon; 2];
  a11 = epsilon; a12 = 1; a22 = 1-1/epsilon; b1 = 1+epsilon;
  b2 = 2-(1+epsilon)/epsilon; x(2) = b2/a22; x(1) = (b1-a12*x(2))/epsilon;
  r = b-A*x; r_norm = norm(r,inf); e_norm = norm(x-x_true,inf);
  fprintf('%2d   %13.6e   %13.6e   %13.6e   %13.6e\n', k, x(1), x(2),...
    r_norm, e_norm)
end
disp(' '); disp('(b) GE without pivoting -- with iterative refinement')
disp(' k        x(1)            x(2)          res_norm        err_norm')
for k = 1:10
  epsilon = 10^(-2*k); A = [epsilon 1; 1 1]; b = [1+epsilon; 2];
```

```
  a11 = epsilon; a12 = 1; a22 = 1-1/epsilon; b1 = 1+epsilon;
  b2 = 2-(1+epsilon)/epsilon; x(2) = b2/a22; x(1) = (b1-a12*x(2))/epsilon;
  r = b-A*x; r(2) = r(2)-r(1)/epsilon; z(2) = r(2)/a22;
  z(1) = (r(1)-a12*z(2))/epsilon; x = x+z; r = b-A*x;
  r_norm = norm(r,inf); e_norm = norm(x-x_true,inf);
  fprintf('%2d   %13.6e   %13.6e   %13.6e   %13.6e\n', k, x(1), x(2),...
    r_norm, e_norm)
end
```

2.10.
```
function cp02_10 % componentwise conditioning of linear system solution
disp('   epsilon          cond          rel_err_x(1)     rel_err_x(2)')
for k = 1:8
  epsilon = 10^(-k); A = [1, 1+epsilon; 1-epsilon, 1];
  b = [1+(1+epsilon)*epsilon; 1]; x_true = [1; epsilon]; x = A\b;
  rel_err_1 = (x(1)-x_true(1))/x_true(1);
  rel_err_2 = (x(2)-x_true(2))/x_true(2);
  fprintf('%13.6e   %13.6e    %13.6e   %13.6e\n', epsilon, cond(A), ...
    rel_err_1, rel_err_2)
end; disp(' ')
disp('Relative error in largest component satisfies bound given by condition')
disp('number, but relative error in smaller component can be much larger.')
```

2.11.
```
function cp02_11(n) % pivoting strategies for Gaussian elimination
A = rand(n,n); x_true = ones(n,1); b = A*x_true;
x_np = GaussElim_np(A, b);
e_norm_np = norm(x_np-x_true,inf); r_norm_np = norm(b-A*x_np,inf);
x_pp = GaussElim_pp(A,b);
e_norm_pp = norm(x_pp-x_true,inf); r_norm_pp = norm(b-A*x_pp,inf);
x_cp = GaussElim_cp(A,b);
e_norm_cp = norm(x_cp-x_true,inf); r_norm_cp = norm(b-A*x_cp,inf);
disp('e_norm_np    r_norm_np    e_norm_pp    r_norm_pp    e_norm_cp    r_norm_cp')
fprintf('%9.4e   %9.4e   %9.4e   %9.4e   %9.4e   %9.4e\n', ...
  e_norm_np, r_norm_np, e_norm_pp, r_norm_pp, e_norm_cp, r_norm_cp)

function [x] = GaussElim_np(A, b) % Gaussian elimination without pivoting
n = size(A,1); x = zeros(n,1);
for i=1:n
  p = A(i,i);
  for j=(i+1):n
    A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
    b(j) = b(j)-b(i)*(A(j,i)/p);   % forward substitution
  end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1                   % back-subsitution
  x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
end

function [x] = GaussElim_pp(A,b) % Gaussian elimination with partial pivoting
n = size(A,1); x = zeros(n,1);
for i = 1:n
  [p, maxk] = max(abs(A(i:n,i)));  % pivot search
```

```
  maxk = maxk+i-1; p = A(maxk,i);
  if i ~= maxk                      % row interchange
    tmp = A(i,i:n); A(i,i:n) = A(maxk, i:n); A(maxk, i:n) = tmp;
    z = b(i); b(i) = b(maxk); b(maxk) = z;
  end
  for j = (i+1):n                   % elimination
    A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
    b(j) = b(j)-b(i)*(A(j,i)/p);    % forward substitution
  end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1                    % back-subsitution
  x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
end

function [x] = GaussElim_cp(A,b) % Gaussian elimination with complete pivoting
n = size(A,1); x = zeros(n,1); index = 1:n;
for i = 1:n
  [maxA, maxK] = max(A(i:n,i:n));  % pivot search
  [p, maxl] = max(maxA); maxk = maxK(maxl)+i-1;
  maxl = maxl+i-1; p = A(maxk,maxl);
  if i ~= maxk                      % row interchange
    tmp = A(i,:); A(i,:) = A(maxk,:); A(maxk,:) = tmp;
    z = b(i); b(i) = b(maxk); b(maxk) = z;
  end
  if i ~= maxl                      % column interchange
    tmp = A(:,i); A(:,i) = A(:,maxl); A(:,maxl) = tmp;
    ii = index(i); index(i) = index(maxl); index(maxl) = ii;
  end;
  for j = (i+1):n                   % elimination
    A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
    b(j) = b(j)-b(i)*(A(j,i)/p);    % forward substitution
  end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1                    % back-subsitution
  x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
end; y = x;
for i = 1:n                         % reverse permutation
  x(i) = y(index(i));
end;
```

2.12. `function cp02_12 % tridiagonal linear system solver`

```
function [x] = TridiagonalSolve(a,b,c,rhs)   % Using notation in text, p. 88
n = length(b); d = zeros(n,1); m = zeros(n,1); y = zeros(n,1); x = zeros(n,1);
d(1) = b(1);
for i=2:n          % LU factorization
  m(i) = a(i)/d(i-1); d(i) = b(i)-m(i)*c(i-1);
end
y(1) = rhs(1);
for i=2:n          % forward substitution
```

```
  y(i) = rhs(i)-m(i)*y(i-1);
end
x(n) = y(n)/d(n);
for i=n-1:-1:1     % back substitution
  x(i) = (y(i)-c(i)*x(i+1))/d(i);
end

function [x] = TridiagonalSolve(A,rhs)   % Using MATLAB sparse matrix format
n = size(A,1); m=zeros(n,1); d=zeros(n,1); c=zeros(n,1); d(1) = A(1,1);
for i=2:n
  m(i-1) = A(i,i-1)/d(i-1); c(i) = A(i-1,i); d(i) = A(i,i)-m(i-1)*c(i);
end
L = spdiags([m ones(n,1)], -1:0, n, n); U = spdiags([d c], 0:1, n, n);
x = U\(L\rhs);
```

2.13.
```
function cp02_13(n) % compute determinant of matrix using LU factorization
A = rand(n,n); [L, U, P] = lu(A); ipvt = P*[1:n]'; sp = 1;
for i = 1:n    % determine sign sp of det(P)
  while ipvt(i) ~= i
    j = ipvt(i); ipvt(i) = ipvt(j); ipvt(j) = j; sp = -sp;
  end;
end;
logdetA = 0.0; % compute det(U) using log to avoid overflow or underflow
su = 1;
for i = 1:n
  if U(i,i) == 0, su = 0; break; end;
  if U(i,i) > 0, logdetA = logdetA+log(U(i,i));
  else logdetA = logdetA+log(-U(i,i)); su = -su;
  end;
end;
if (su == 0), detA = 0; % matrix singular
elseif sp*su > 0 detA = exp(logdetA);
else detA = -exp(logdetA); end;
fprintf('Computed determinant = %17.14e\n', detA)
fprintf('MATLAB det function  = %17.14e\n', det(A))
```

2.14.
```
function cp02_14(n) % performance of algorithms for matrix multiplication
A = rand(n,n); B = rand(n,n); C = zeros(n,n); m = n; k = n; tic
for i=1:m
  for j=1:k
    C(i,j) = A(i,:)*B(:,j);
  end
end
fprintf('Time for matrix multiply using sdot: %f\n',toc); tic
for j=1:k
  C(:,j) = A*B(:,j);
end
fprintf('Time for matrix multiply using saxpy: %f\n',toc);
```

2.15.
```
function cp02_15(n) % performance of algorithms for Gaussian elimination
a = rand(n,n); time = zeros(6,1); tic;
```

```
for k = 1:n-1                    % kij algorithm
  for i = k+1:n
    a(i,k) = a(i,k)/a(k,k); a(i,k+1:n) = a(i,k+1:n)-a(i,k)*a(k,k+1:n);
  end
end; time(1) = toc; tic;
for k = 1:n-1                    % kji algorithm
  a(k+1:n,k) = a(k+1:n,k)/a(k,k);
  for j = k+1:n
    a(k+1:n,j) = a(k+1:n,j)-a(k+1:n,k)*a(k,j);
  end
end; time(2) = toc; tic;
for i = 2:n                      % ikj algorithm
  for k = 1:i-1
    a(i,k) = a(i,k)/a(k,k); a(i,k+1:n) = a(i,k+1:n)-a(i,k)*a(k,k+1:n);
  end
end; time(3) = toc; tic;
for j = 2:n                      % jki algorithm
  a(j:n,j-1) = a(j:n,j-1)/a(j-1,j-1);
  for k = 1:j-1
    a(k+1:n,j) = a(k+1:n,j)-a(k+1:n,k)*a(k,j);
  end
end; time(4) = toc; tic;
for i = 2:n                      % ijk algorithm
  for j = 2:i
    a(i,j-1) = a(i,j-1)/a(j-1,j-1); a(i,j) = a(i,j)-a(i,1:j-1)*a(1:j-1,j);
  end
  for j = i+1:n
    a(i,j) = a(i,j)-a(i,1:i-1)*a(1:i-1,j);
  end
end; time(5) = toc; tic;
for j = 2:n                      % jik algorithm
  a(j:n,j-1) = a(j:n,j-1)/a(j-1,j-1);
  for i = 2:j
    a(i,j) = a(i,j)-a(i,1:i-1)*a(1:i-1,j);
  end
  for i = j+1:n
    a(i,j) = a(i,j)-a(i,1:j-1)*a(1:j-1,j);
  end
end; time(6) = toc;
fprintf(' n     kij     kji     ikj     jki     ijk     jik\n');
fprintf('%g  %6.3f  %6.3f  %6.3f  %6.3f  %6.3f  %6.3f\n', n, time);
```

2.16.
```
function cp02_16(n) % performance of algorithms for triangular solution
A = rand(n,n); b = rand(n,1); [Q,U] = qr(A); L = U';
tic; row_forward_sub(L, b); time = toc;
fprintf('Time using row-wise forward substitution: %f\n',time);
tic; col_forward_sub(L, b); time = toc;
fprintf('Time using column-wise forward substitution: %f\n',time);
tic; row_back_sub(L, b); time = toc;
fprintf('Time using row-wise back substitution: %f\n',time);
```

```
tic; col_back_sub(L, b); time = toc;
fprintf('Time using column-wise back substitution: %f\n',time);

function [x] = row_forward_sub(L, b)
n = length(b); x = zeros(n,1);
for i = 1:n
  tot = 0;
  for j = 1:i-1
    tot = L(i,j)*x(j);
  end
  if L(i,i) == 0, error('Matrix is singular'); end
  x(i) = (b(i)-tot)/L(i,i);
end

function [x] = row_back_sub(U, b)
n = length(b); x = zeros(n,1);
for i = n:-1:1
  tot = 0;
  for j = i+1:n
    tot = U(i,j)*x(j);
  end
  if U(i,i) == 0, error('Matrix is singular'); end
  x(i) = (b(i)-tot)/U(i,i);
end

function [x] = col_back_sub(U, b)
n = length(b); x = zeros(n,1);
for j = n:-1:1
  if U(j,j) == 0, error('Matrix is singular'); end
  x(j) = b(j)/U(j,j);
  for i = 1:j-1
    b(i) = b(i)-U(i,j)*x(j);
  end
end

function [x] = col_forward_sub(L, b)
n = length(b); x = zeros(n,1);
for j = 1:n
  if L(j,j) == 0, error('Matrix is singular'); end
  x(j) = b(j)/L(j,j);
  for i = j+1:n
    b(i) = b(i)-L(i,j)*x(j);
  end
end
```

2.17.
```
function cp02_17 % linear system for cantilevered beam
disp('(a)'); n = 100; b = ones(n,1);
A = spdiags([b -4*b 6*b -4*b b], -2:2, n, n);
A(1,1) = 9; A(n-1,n-1)=5; A(n,n) = 1; A(n,n-1)=-2; A(n-1,n)=-2;
tic; x_sparse = A\b; tsparse = toc;
```

```
tic; x_dense = full(A)\b; tdense = toc;
fprintf('Time for dense solution = %g\n', tdense);
fprintf('Time for sparse solution = %g\n', tsparse);
fprintf('norm(x_sparse-x_dense) = %g\n', norm(x_sparse-x_dense));

disp('(b)'); n = 1000; b = ones(n,1);
A = spdiags([b -4*b 6*b -4*b b], -2:2, n, n);
A(1,1) = 9; A(n-1,n-1)=5; A(n,n) = 1; A(n,n-1)=-2; A(n-1,n)=-2;
R = spdiags([b -2*b b], 0:2, n, n); R(1,1) = 2;
fprintf('norm(R R^T - A) = %g\n', norm(R*R'-A,inf));
fprintf('cond(A) = %g\n', condest(A))
fprintf('cond(R) = %g\n', condest(R))
x_sparse = A\b; r = b-A*x_sparse;
fprintf('residual norm for x_sparse = %g\n', norm(r));
x_rr = R'\(R\b); r_rr = b-A*x_rr;
fprintf('residual norm for x_rr = %g\n', norm(r_rr));
fprintf('norm(x_sparse-x_rr) = %g\n', norm(x_sparse-x_rr));
disp('Iterative refinement:')
normnew = norm(r); normold = normnew*2.0;
while normnew < normold
  normold = normnew; s = A\r; x_sparse = x_sparse+s;
  r = b-A*x_sparse; normnew = norm(r);
  fprintf('residual norm for x_sparse = %g\n', normnew);
  fprintf('norm(x_sparse-x_rr) = %g\n', norm(x_sparse-x_rr));
end;
```

Chapter 3

Linear Least Squares

Exercises

3.1. (a)

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & 10 \\ 1 & 15 \\ 1 & 20 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 11.60 \\ 11.85 \\ 12.25 \end{bmatrix} = \boldsymbol{b}.$$

(b) This system is inconsistent. Using the first two equations, $\boldsymbol{x} = [11.1, 0.05]^T$. Using the first and third equations, $\boldsymbol{x} = [10.95, 0.065]^T$. Using the second and third equations, $\boldsymbol{x} = [10.65, 0.08]^T$. There is no rigorous reason to prefer any one of these solutions over the others, although the second solution might seem "safest" in that it is closest to the average over all the solutions. (c) The system of normal equations is

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 3 & 45 \\ 45 & 725 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 35.7 \\ 538.75 \end{bmatrix} = \boldsymbol{A}^T\boldsymbol{b},$$

which has solution $\boldsymbol{x} = [10.925, 0.065]^T$. This answer is closest to the second solution obtained in (b), using the first and third equations.

3.2. (a) The overdetermined linear system is

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \boldsymbol{b}.$$

(b) The system of normal equations is

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 3 & 4 \\ 4 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 11 \end{bmatrix} = \boldsymbol{A}^T\boldsymbol{b}.$$

(c) The Cholesky factor of $\boldsymbol{A}^T\boldsymbol{A}$ is

$$\boldsymbol{L} = \begin{bmatrix} 1.7321 & 0 \\ 2.3094 & 2.1603 \end{bmatrix}.$$

Solving the system $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{A}^T\boldsymbol{b}$ by forward-substitution gives $\boldsymbol{y} = \begin{bmatrix} 3.464 & 1.388 \end{bmatrix}^T$. Solving the system $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$ by back-substitution then gives the least squares solution $\boldsymbol{x} = \begin{bmatrix} 1.1427 & 0.6429 \end{bmatrix}^T$.

3.3.

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & e \\ 2 & e^2 \\ 3 & e^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cong \begin{bmatrix} 2 \\ 3 \\ 5 \end{bmatrix} = \boldsymbol{b}.$$

3.4. The least squares system is

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \cong \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \boldsymbol{b},$$

so the least squares solution is unique because the matrix has full column rank, i.e., $\text{rank}(\boldsymbol{A}) = 2$.

3.5. The residual vector for the least squares solution must be orthogonal to columns of $\boldsymbol{A}$. We can therefore rule out (a) because it is not orthogonal to either column of $\boldsymbol{A}$. We can rule out (b) because it is not orthogonal to the second column of $\boldsymbol{A}$. Since (c) is orthogonal to both columns of $\boldsymbol{A}$, it could be a possible residual vector for the least squares solution.

3.6. (a) Because the system is triangular, the residual is given by the norm of the bottom portion of the right-hand-side vector $\boldsymbol{b}$, which in this case gives $\|\boldsymbol{r}\|_2 = 1$. (b) Again, because the system is triangular, the least squares solution is given by solving the square triangular system at the top, which in this case gives $\boldsymbol{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

3.7. (a) The function $\phi(\boldsymbol{y}) = \|\boldsymbol{b} - \boldsymbol{y}\|_2$ is continuous and coercive, and therefore has a minimum on the closed, unbounded set $\text{span}(\boldsymbol{A})$. Thus, by definition of $\text{span}(\boldsymbol{A})$, there is some $\boldsymbol{x}$ that minimizes $\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2$, which is therefore a solution to the least squares problem. (b) Suppose that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are two solutions, and let $\boldsymbol{z} = \boldsymbol{x}_2 - \boldsymbol{x}_1$. Then, since $\boldsymbol{A}\boldsymbol{x}_1 = \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_2$, we have $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{o}$. Now if $\boldsymbol{z} \neq \boldsymbol{o}$, i.e., $\boldsymbol{x}_1 \neq \boldsymbol{x}_2$, then $\text{rank}(\boldsymbol{A}) < n$. For the other direction, if $\text{rank}(\boldsymbol{A}) < n$, then there is a nonzero $\boldsymbol{z}$ such that $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{o}$, and hence if $\boldsymbol{x}$ is a least squares solution, then $\boldsymbol{x} + \boldsymbol{z}$ is also a solution, since $\boldsymbol{A}(\boldsymbol{x} + \boldsymbol{z}) = \boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$. We conclude that the least squares solution is unique if, and only if, $\text{rank}(\boldsymbol{A}) = n$.

3.8. Let $\boldsymbol{x}$ be any nonzero vector. Then $\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = (\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{A}\boldsymbol{x}) > 0$ unless $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{o}$. But the latter case would contradict the assumption that $\text{rank}(\boldsymbol{A}) = n$. Thus, $\boldsymbol{A}^T\boldsymbol{A}$ must be positive definite.

3.9. Let $\boldsymbol{x}$ be any nonzero n-vector. Then

$$\begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{x} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{I} & \boldsymbol{A} \\ \boldsymbol{A}^T & \boldsymbol{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{x} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{A}\boldsymbol{x} \\ \boldsymbol{o} \end{bmatrix} = 0,$$

which shows that the augmented matrix cannot be positive definite.

3.10. (a) Suppose $\boldsymbol{B}$ is upper triangular (the opposite case is similar), and denote the ith column of $\boldsymbol{B}$ by $\boldsymbol{b}_i$. Since $\boldsymbol{B}$ is orthogonal, we have $\boldsymbol{b}_1^T\boldsymbol{b}_1 = 1$, which implies that $b_{11} = \pm 1$, since b_{11} is the only nonzero in $\boldsymbol{b}_1$. Orthogonality requires that $\boldsymbol{b}_1^T\boldsymbol{b}_j = 0$ for $j \neq 1$, which in turn implies that $b_{1j} = 0$ for $j \neq i$ (i.e., the off-diagonal entries in the first row of $\boldsymbol{B}$ are all zero). Similar reasoning for $\boldsymbol{b}_2$ shows that the same holds for row two, and so on. Thus, $\boldsymbol{B}$ must be diagonal. (b) $b_{ii} = \pm 1$, $i = 1, \ldots, n$. (c) Suppose $\boldsymbol{A}$ is $n \times n$ and nonsingular, and let $\boldsymbol{A} = \boldsymbol{Q}_1\boldsymbol{R}_1 = \boldsymbol{Q}_2\boldsymbol{R}_2$ be two QR factorizations of $\boldsymbol{A}$, with $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ necessarily nonsingular (because $\boldsymbol{A}$ is nonsingular). If we let $\boldsymbol{D} = \boldsymbol{R}_2\boldsymbol{R}_1^{-1} = \boldsymbol{Q}_2^T\boldsymbol{Q}_1$, then we see that $\boldsymbol{D}$ is both orthogonal and triangular, and hence by part (a), $\boldsymbol{D}$ is diagonal with diagonal entries ± 1. Thus, $\boldsymbol{R}_2 = \boldsymbol{D}\boldsymbol{R}_1$, which says that $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are the same up to signs of the diagonal entries, and similarly for $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$. In particular, if the signs of the diagonal entries of $\boldsymbol{R}$ are all required to be positive, then the QR factorization is unique.

3.11. Since the matrix is orthogonal, we must have

$$\begin{bmatrix} \boldsymbol{A}^T & \boldsymbol{O} \\ \boldsymbol{B}^T & \boldsymbol{C}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ \boldsymbol{O} & \boldsymbol{C} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}^T\boldsymbol{A} & \boldsymbol{A}^T\boldsymbol{B} \\ \boldsymbol{B}^T\boldsymbol{A} & \boldsymbol{B}^T\boldsymbol{B} + \boldsymbol{C}^T\boldsymbol{C} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{I} \end{bmatrix}.$$

So $\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{I}$, and hence $\boldsymbol{A}$ is orthogonal. Also, $\boldsymbol{A}^T\boldsymbol{B} = \boldsymbol{O}$, so $\boldsymbol{B} = \boldsymbol{O}$, since $\boldsymbol{A}$ is orthogonal. But we then have $\boldsymbol{C}\boldsymbol{C}^T = \boldsymbol{I}$, and hence $\boldsymbol{C}$ is orthogonal.

3.12. (a) (1) and (2) $\Rightarrow$ (3): $\boldsymbol{A}^2 = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$. (1) and (3) $\Rightarrow$ (2): $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}^2 = \boldsymbol{I}$. (2) and (3) $\Rightarrow$ (1): $\boldsymbol{A}^T = \boldsymbol{A}^T\boldsymbol{A}^2 = \boldsymbol{A}$. ($b$)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

(c) Householder transformations.

3.13. If $\boldsymbol{A}$ is an orthogonal projector, then $\boldsymbol{A}^2 = \boldsymbol{A}$ and $\boldsymbol{A}^T = \boldsymbol{A}$. If $\boldsymbol{A}$ is also an orthogonal matrix, then $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$, and hence we have $\boldsymbol{A} = \boldsymbol{A}^2 = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$.

3.14.

$$\boldsymbol{H}^T = \boldsymbol{I}^T - 2\,\frac{(\boldsymbol{v}\boldsymbol{v}^T)^T}{(\boldsymbol{v}^T\boldsymbol{v})^T} = \boldsymbol{I} - 2\,\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} = \boldsymbol{H},$$

so $\boldsymbol{H}$ is symmetric.

$$\boldsymbol{H}^T\boldsymbol{H} = \boldsymbol{H}^2 = \left(\boldsymbol{I} - 2\,\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)^2 = \boldsymbol{I} - 4\,\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} + 4\,\frac{\boldsymbol{v}(\boldsymbol{v}^T\boldsymbol{v})\boldsymbol{v}^T}{(\boldsymbol{v}^T\boldsymbol{v})^2} = \boldsymbol{I},$$

so $\boldsymbol{H}$ is orthogonal.

3.15.

$$\begin{aligned} \boldsymbol{H}\boldsymbol{a} &= \boldsymbol{a} - 2\,\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\,\boldsymbol{a} = \boldsymbol{a} - 2\,\frac{\boldsymbol{v}^T\boldsymbol{a}}{\boldsymbol{v}^T\boldsymbol{v}}\,\boldsymbol{v} = \boldsymbol{a} - 2\,\frac{(\boldsymbol{a}-\alpha\,\boldsymbol{e}_1)^T\boldsymbol{a}}{(\boldsymbol{a}-\alpha\,\boldsymbol{e}_1)^T(\boldsymbol{a}-\alpha\,\boldsymbol{e}_1)}\,\boldsymbol{v} \\ &= \boldsymbol{a} - 2\,\frac{\alpha^2 - \alpha\,\boldsymbol{e}_1^T\boldsymbol{a}}{2\,\alpha^2 - 2\,\boldsymbol{e}_1^T\boldsymbol{a}}\,\boldsymbol{v} = \boldsymbol{a} - \boldsymbol{v} = \alpha\,\boldsymbol{e}_1 \end{aligned}$$

3.16. (*a*) As is easily seen by inspection (or from the first step of Gram-Schmidt), $\boldsymbol{Q} = \boldsymbol{a}/\|\boldsymbol{a}\|_2$, $\boldsymbol{R} = \|\boldsymbol{a}\|_2$. (*b*) The least squares solution $\boldsymbol{x}$ is given by the linear system $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{Q}^T\boldsymbol{b}$, so for this special case we solve the 1×1 system $\|\boldsymbol{a}\|_2 x = (\boldsymbol{a}/\|\boldsymbol{a}\|_2)^T\boldsymbol{b}$, which gives the least squares solution $x = \boldsymbol{a}^T\boldsymbol{b}/\|\boldsymbol{a}\|_2^2 = \boldsymbol{a}^T\boldsymbol{b}/\boldsymbol{a}^T\boldsymbol{a}$, in agreement with the solution given by the normal equations.

3.17. Choose $\alpha = -\text{sign}(a_1)\|\boldsymbol{a}\|_2 = -2$. Then, $\boldsymbol{v} = \boldsymbol{a} - \alpha\,\boldsymbol{e}_1 = [\,3 \quad 1 \quad 1 \quad 1\,]^T$.

3.18. (*a*) Three, because the matrix has three columns. (*b*) $[\,-2 \quad 0 \quad 0 \quad 0\,]^T$. (*c*) $[\,-2 \quad 0 \quad 0 \quad 0\,]^T$, i.e., the first column is unaffected by the second Householder transformation. (*d*) Six, because the matrix has six nonzero entries below the diagonal that must be annihilated.

3.19. (*a*)

$$\boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4/3 & 1 \end{bmatrix} \quad \text{or} \quad \boldsymbol{M} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & 0 & 1 \end{bmatrix}.$$

(*b*) Taking $\alpha = -5$ and $\boldsymbol{v} = [\,0 \quad 8 \quad 4\,]^T$,

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.6 & -0.8 \\ 0 & -0.8 & 0.6 \end{bmatrix}.$$

(*c*)

$$\boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.6 & 0.8 \\ 0 & -0.8 & 0.6 \end{bmatrix}.$$

(*d*) No, because a Householder transformation matrix is always both symmetric and orthogonal, whereas an elementary elimination matrix can never be symmetric or orthogonal except for the identity matrix, which cannot introduce a zero in place of a nonzero. (*e*) No, because a Householder transformation matrix is always symmetric, whereas a Givens rotation can never be symmetric except for the identity matrix, which is a rotation angle of zero and cannot introduce a zero in place of a nonzero.

3.20. (*a*) Yes, by a rotation of 90 degrees (i.e., $c = 0$, $s = 1$):

$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a_2 \end{bmatrix} = \begin{bmatrix} a_2 \\ 0 \end{bmatrix}.$$

(b) No, because it would require dividing by a zero pivot, so if $a_2 \neq 0$, then there is no value of the multiplier m such that

$$\begin{bmatrix} 1 & 0 \\ m & 1 \end{bmatrix} \begin{bmatrix} 0 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Another way to see this is that the first entry, a_1, which is already zero, is unaffected by the elementary elimination matrix, so if the second entry were also annihilated, then the elementary elimination matrix would be singular, which is impossible.

3.21. The most obvious method would be to store only the angle of rotation, from which the sine and cosine could be recovered as needed. Another method would be to store only the tangent (or cotangent) of the angle of rotation. To recover the sine and cosine, if $t \leq 1$ then $c = 1/\sqrt{1+t^2}$, $s = ct$, and if $t > 1$, then $s = 1/\sqrt{1+(1/t)^2}$, $c = s/t$.

3.22. (a)

$$\boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{A}^T\boldsymbol{A} = [\, \boldsymbol{R}^T \quad \boldsymbol{O} \,]\, \boldsymbol{Q}^T\boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix} = \boldsymbol{R}^T\boldsymbol{R}.$$

(b) No, because the entries of $\boldsymbol{R}$ and $\boldsymbol{L}^T$ could still differ in sign (i.e., $\boldsymbol{R}$ can be scaled by a diagonal matrix with ± 1 as diagonal entries).

3.23. We compute $\boldsymbol{R}$ in floating-point arithmetic using Givens rotations. To zero the (2,1) entry, $t = \epsilon$, $c = 1$, and $s = \epsilon$, which gives

$$\boldsymbol{G}_1\boldsymbol{A} = \begin{bmatrix} 1 & \epsilon & 0 \\ -\epsilon & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & -\epsilon \\ 0 & \epsilon \end{bmatrix}.$$

To annihilate the (3,2) entry, $t = -1$, $c = 1/\sqrt{2}$, and $s = -1/\sqrt{2}$, which gives

$$\boldsymbol{G}_2\boldsymbol{G}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & -\epsilon \\ 0 & \epsilon \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & -\epsilon\sqrt{2} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix},$$

whose upper 2×2 submatrix $\boldsymbol{R}$ is nonsingular, since $\epsilon \neq 0$.

3.24. The cross-product matrix $\boldsymbol{A}^T\boldsymbol{A}$ in the normal equations has n^2 entries, each of which requires m multiplications (and a similar number of additions) to compute. The matrix is symmetric, however, so only $n(n+1)/2$ of its entries need to be computed. From Exercise 2.39, computing the Cholesky factorization of $\boldsymbol{A}^T\boldsymbol{A}$ requires about $n^3/6$ multiplications (and a similar number of additions). Forming the right-hand side $\boldsymbol{A}^T\boldsymbol{b}$ and performing forward- and back-subsitutions are lower-order terms. Thus, the total cost for computing the least squares solution is about $mn^2/2 + n^3/6$ multiplications (and a similar number of additions).

3.25. The algorithm requires

$$\begin{aligned}\sum_{k=1}^{n}\Big(2(m-k)+\sum_{j=k}^{n}2(m-k)\Big) &= \sum_{k=1}^{n}\left(2(m-k)+2(m-k)(n-k)\right)\\ &= \sum_{k=1}^{n}\left(2mn+2m-2(m+n+1)k+2k^2\right)\\ &= 2(mn+m)n-(m+n+1)n(n+1)\\ &\quad +n(n+1)(2n+1)/3\end{aligned}$$

multiply-add operations. Retaining only the highest order terms, this simplifies to $mn^2 - n^3/3$ multiply-add operations.

3.26. (a) $\boldsymbol{G}$ rotates any nonzero 2-vector clockwise by an angle of θ. (b) Equating

$$\begin{bmatrix} -c & s \\ s & c \end{bmatrix} = \boldsymbol{H} = \boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^T = \begin{bmatrix} 1-2v_1v_1 & -2v_1v_2 \\ -2v_1v_2 & 1-2v_1v_1 \end{bmatrix},$$

we see that $\boldsymbol{H}$ reflects any nonzero 2-vector through the line defined by the vector $\boldsymbol{v}^\perp = \begin{bmatrix} \sqrt{(1-c)/2} & \sqrt{(1+c)/2} \end{bmatrix}^T$.

3.27. (a) $(\boldsymbol{Q}\boldsymbol{Q}^T)^T = \boldsymbol{Q}\boldsymbol{Q}^T$, and $(\boldsymbol{Q}\boldsymbol{Q}^T)^2 = \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{Q})\boldsymbol{Q}^T = \boldsymbol{Q}\boldsymbol{Q}^T$, so $\boldsymbol{Q}\boldsymbol{Q}^T$ is symmetric and idempotent. By assumption, the columns of $\boldsymbol{Q}$ form a basis for the subspace $\mathcal{S}$, so for any vector $\boldsymbol{v}$, $(\boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{v} = \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{v})$ lies in $\mathcal{S}$. Thus $\boldsymbol{Q}\boldsymbol{Q}^T$ is a projector onto $\mathcal{S}$. (b) $(\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T)^T = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-T}\boldsymbol{A}^T = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$, and $(\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T)^2 = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}(\boldsymbol{A}^T\boldsymbol{A})(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$, so $\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ is symmetric and idempotent. For any vector $\boldsymbol{v}$, $(\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T)\boldsymbol{v} = \boldsymbol{A}((\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{v})$ lies in the column space of $\boldsymbol{A}$. Thus $\boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ is a projector onto the column space of $\boldsymbol{A}$. For the linear least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$, the solution to the normal equations is given by $\boldsymbol{x} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b}$, and $\boldsymbol{A}\boldsymbol{x}$ is the projection of $\boldsymbol{b}$ onto the column space of $\boldsymbol{A}$. (c) If $\boldsymbol{P}$ is a projector, then $\boldsymbol{P}^2 = \boldsymbol{P} = \boldsymbol{P}^T$. To show that $(\boldsymbol{I}-\boldsymbol{P})$ is a projector, we note that it is symmetric, since $(\boldsymbol{I}-\boldsymbol{P})^T = (\boldsymbol{I}^T - \boldsymbol{P}^T) = \boldsymbol{I}-\boldsymbol{P}$, and idempotent, since $(\boldsymbol{I}-\boldsymbol{P})^2 = \boldsymbol{I} - 2\boldsymbol{P} + \boldsymbol{P}^2 = \boldsymbol{I}-\boldsymbol{P}$. For any vector $\boldsymbol{v}$, $(\boldsymbol{I}-\boldsymbol{P})\boldsymbol{v} = \boldsymbol{v} - \boldsymbol{P}\boldsymbol{v}$ lies in the orthogonal complement of $\mathcal{S}$, since the component in $\mathcal{S}$, $\boldsymbol{P}\boldsymbol{v}$, has been removed. (d) If $\boldsymbol{v}$ is any nonzero vector, let $\boldsymbol{P} = (\boldsymbol{v}\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v})$. Then $\boldsymbol{P}$ is clearly symmetric, and $\boldsymbol{P}$ is also idempotent, since $\boldsymbol{P}^2 = (\boldsymbol{v}\boldsymbol{v}^T)(\boldsymbol{v}\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v})^2 = (\boldsymbol{v}(\boldsymbol{v}^T\boldsymbol{v})\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v})^2 = (\boldsymbol{v}\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v})$. Moreover, if $\boldsymbol{u}$ is any vector, then $\boldsymbol{P}\boldsymbol{u} = ((\boldsymbol{v}\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v}))\boldsymbol{u} = \boldsymbol{v}(\boldsymbol{v}^T\boldsymbol{u})/(\boldsymbol{v}^T\boldsymbol{v})$, which is a scalar multiple of $\boldsymbol{v}$. Thus $\boldsymbol{P}$ is the projector onto the space spanned by $\boldsymbol{v}$.

3.28. (a) Proof by induction on k. For $k = 1$, the claim holds trivially. Suppose that it holds for up to $k-1$ terms. Then we have

$$\begin{aligned}&(\boldsymbol{I}-\boldsymbol{P}_k)(\boldsymbol{I}-\boldsymbol{P}_{k-1})\cdots(\boldsymbol{I}-\boldsymbol{P}_1)\\ &= (\boldsymbol{I}-\boldsymbol{P}_k)(\boldsymbol{I}-\boldsymbol{P}_{k-1}-\cdots-\boldsymbol{P}_1)\\ &= (\boldsymbol{I}-\boldsymbol{P}_{k-1}-\cdots-\boldsymbol{P}_1) - \boldsymbol{P}_k(\boldsymbol{I}-\boldsymbol{P}_{k-1}-\cdots-\boldsymbol{P}_1)\\ &= \boldsymbol{I}-\boldsymbol{P}_k-\boldsymbol{P}_{k-1}-\cdots-\boldsymbol{P}_1,\end{aligned}$$

since for $i \neq j$, $\boldsymbol{P}_i\boldsymbol{P}_j = (\boldsymbol{q}_i\boldsymbol{q}_i^T)(\boldsymbol{q}_j\boldsymbol{q}_j^T) = \boldsymbol{q}_i(\boldsymbol{q}_i^T\boldsymbol{q}_j)\boldsymbol{q}_j^T = \boldsymbol{O}$ due to the orthogonality of $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$. Thus, the claim holds for all k. (b) $\boldsymbol{q}_k = (\boldsymbol{I} - (\boldsymbol{P}_1 + \cdots + \boldsymbol{P}_{k-1}))\boldsymbol{a}_k = \boldsymbol{a}_k - \boldsymbol{P}_1\boldsymbol{a}_k - \cdots - \boldsymbol{P}_{k-1}\boldsymbol{a}_k = \boldsymbol{a}_k - \boldsymbol{q}_1\boldsymbol{q}_1^T\boldsymbol{a}_k - \cdots - \boldsymbol{q}_{k-1}\boldsymbol{q}_{k-1}^T\boldsymbol{a}_k$, which is precisely the classical Gram-Schmidt algorithm. (c) Similarly, the second equation reflects the fact that in the modified Gram-Schmidt algorithm the complementary projectors $\boldsymbol{I} - \boldsymbol{P}_j$ are applied successively to each initial vector, so $\boldsymbol{q}_k$ is given by their product rather than the sum of the projectors $\boldsymbol{P}_j$. (d) The first two are mathematically equivalent by part (a), and the third is mathematically equivalent to the first because projectors are idempotent.

3.29. (a) If $\boldsymbol{x}$ is a scalar multiple of $\boldsymbol{v}$, then $\boldsymbol{P}\boldsymbol{x} = \boldsymbol{o}$, so $\boldsymbol{R}\boldsymbol{x} = -\boldsymbol{x} = 2\boldsymbol{P}\boldsymbol{x} - \boldsymbol{x}$. If $\boldsymbol{x}$ is orthogonal to $\boldsymbol{v}$, then $\boldsymbol{P}\boldsymbol{x} = \boldsymbol{x}$, so $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{x} = 2\boldsymbol{P}\boldsymbol{x} - \boldsymbol{x}$. This shows that $\boldsymbol{R} = 2\boldsymbol{P} - \boldsymbol{I}$. (b) $\boldsymbol{R}$ is symmetric, since $\boldsymbol{R}^T = (2\boldsymbol{P} - \boldsymbol{I})^T = 2\boldsymbol{P}^T - \boldsymbol{I}^T = 2\boldsymbol{P} - \boldsymbol{I} = \boldsymbol{R}$. $\boldsymbol{R}$ is orthogonal, since $\boldsymbol{R}^T\boldsymbol{R} = 4\boldsymbol{P}^2 - 4\boldsymbol{P} + \boldsymbol{I} = 4\boldsymbol{P} - 4\boldsymbol{P} + \boldsymbol{I} = \boldsymbol{I}$. (c) If $\boldsymbol{x} = \alpha\boldsymbol{v}$ for some nonzero α, then $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{x} - 2(\boldsymbol{v}^T\boldsymbol{v})^{-1}\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{x} = \boldsymbol{x} - 2\boldsymbol{x} = -\boldsymbol{x}$. If $\boldsymbol{x}$ is orthogonal to $\boldsymbol{v}$, then $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{x} - 2(\boldsymbol{v}^T\boldsymbol{v})^{-1}\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{x} = \boldsymbol{x}$. (d) Let $\boldsymbol{v} = \boldsymbol{s} - \boldsymbol{t}$ and let $\boldsymbol{H}$ be the Householder transformation given by $\boldsymbol{v}$. Then

$$\begin{aligned} \boldsymbol{H}\boldsymbol{s} &= \boldsymbol{s} - 2\,\frac{(\boldsymbol{s}-\boldsymbol{t})(\boldsymbol{s}-\boldsymbol{t})^T}{(\boldsymbol{s}-\boldsymbol{t})^T(\boldsymbol{s}-\boldsymbol{t})}\,\boldsymbol{s} = \boldsymbol{s} - 2\,\frac{(\boldsymbol{s}^T\boldsymbol{s} - \boldsymbol{t}^T\boldsymbol{s})}{\boldsymbol{s}^T\boldsymbol{s} - \boldsymbol{t}^T\boldsymbol{s} - \boldsymbol{s}^T\boldsymbol{t} + \boldsymbol{t}^T\boldsymbol{t}}\,(\boldsymbol{s}-\boldsymbol{t}) \\ &= \boldsymbol{s} - (\boldsymbol{s}-\boldsymbol{t}) = \boldsymbol{t} \end{aligned}$$

(e) Compute the QR factorization of $\boldsymbol{Q}$ using a sequence of Householder reflections, obtaining $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_{n-1}\boldsymbol{R}$. In this instance, $\boldsymbol{R}$ is both triangular and orthogonal, and hence diagonal, with each diagonal entry either 1 or -1 (see Exercise 3.10), and so is itself a reflector. Thus, $\boldsymbol{Q}$ can be written a product of reflectors. (f)

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} c & -s \\ -s & -c \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

3.30. (a) $\boldsymbol{U} = \boldsymbol{a}/\|\boldsymbol{a}\|_2$, $\boldsymbol{\Sigma} = \|\boldsymbol{a}\|_2$, $\boldsymbol{V} = 1$. (b) $\boldsymbol{U} = 1$, $\boldsymbol{\Sigma} = \|\boldsymbol{a}\|_2$, $\boldsymbol{V} = \boldsymbol{a}/\|\boldsymbol{a}\|_2$.

3.31. Consider the $m \times n$ least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$, with $\operatorname{rank}(\boldsymbol{A}) = k$, and let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ be the SVD of $\boldsymbol{A}$. Define

$$\boldsymbol{z} = \boldsymbol{V}^T\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}, \qquad \boldsymbol{c} = \boldsymbol{U}^T\boldsymbol{b} = \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2, \end{bmatrix},$$

where $\boldsymbol{x}_1$ and $\boldsymbol{c}_1$ are k-vectors, and let $\boldsymbol{\Sigma}_1$ denote the leading $k \times k$ submatrix of $\boldsymbol{\Sigma}$. By assumption, $\boldsymbol{\Sigma}_1$ is nonsingular. Now consider the residual norm

$$\begin{aligned} \|\boldsymbol{r}\|_2 &= \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2 = \|\boldsymbol{U}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T\boldsymbol{x})\|_2 \\ &= \left\| \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{\Sigma}_1 & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} \boldsymbol{c}_1 - \boldsymbol{\Sigma}_1\boldsymbol{z}_1 \\ \boldsymbol{c}_2 \end{bmatrix} \right\|_2. \end{aligned}$$

The residual norm is minimized when $\boldsymbol{z_1} = \boldsymbol{\Sigma}_1^{-1}\boldsymbol{c}_1$ and $\boldsymbol{z}_2$ is arbitrary. The solution norm $\|\boldsymbol{x}\|_2 = \|\boldsymbol{z}\|_2$ is minimized when $\boldsymbol{z}_2 = \boldsymbol{o}$. Thus, the least squares solution of

minimum norm is given by

$$\boldsymbol{x} = \boldsymbol{V} \begin{bmatrix} \boldsymbol{\Sigma}_1^{-1} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} \boldsymbol{U}^T \boldsymbol{b} = \sum_{\sigma_i \neq 0} \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i.$$

3.32. (*a*) $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)(\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T) = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{A}$. (*b*) $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = (\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)(\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T) = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{A}^+$. (*c*) $(\boldsymbol{A}\boldsymbol{A}^+)^T = ((\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)(\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T))^T = \boldsymbol{U}\boldsymbol{\Sigma}^+\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{A}\boldsymbol{A}^+$. (*d*) $(\boldsymbol{A}^+\boldsymbol{A})^T = ((\boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T))^T = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}^+\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^+\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \boldsymbol{A}^+\boldsymbol{A}$.

3.33. (*a*) If $\boldsymbol{A}$ is nonsingular, then $\boldsymbol{\Sigma}$ is nonsingular and $\boldsymbol{\Sigma}^+ = \boldsymbol{\Sigma}^{-1}$ by construction, and

$$\boldsymbol{A}^{-1} = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{A}^+.$$

(*b*) If $\boldsymbol{A}$ has full column rank, then $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}$ are both nonsingular, and

$$\begin{aligned} (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T &= (\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T)^{-1}\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T = \boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{-1}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T \\ &= \boldsymbol{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{-1}\boldsymbol{\Sigma}^T\boldsymbol{U}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{A}^+. \end{aligned}$$

(*c*) If $\boldsymbol{A}$ has full row rank, then $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T$ are both nonsingular, and

$$\begin{aligned} \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1} &= \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T)^{-1} = \boldsymbol{V}\boldsymbol{\Sigma}^T\boldsymbol{U}^T\boldsymbol{U}(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{U}^T \\ &= \boldsymbol{V}\boldsymbol{\Sigma}^T(\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{U}^T = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}^T = \boldsymbol{A}^+. \end{aligned}$$

3.34. (*a*) $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. (*b*) $\begin{bmatrix} 1 & 0 \\ 0 & 1/\epsilon \end{bmatrix}$. (*c*) This example shows that the pseudoinverse is not necessarily a continuous function of the matrix entries, and hence computing the pseudoinverse can be a highly ill-conditioned problem.

Computer Problems

3.1.
```
function cp03_01 % polynomial fitting
t = [0; 1; 2; 3; 4; 5]; y = [1; 2.7; 5.8; 6.6; 7.5; 9.9]; plot(t,y,'ko');
hold on; m = size(t,1); A = ones(m,1); p = 51; ts = linspace(0,5,p)';
for n = 1:m
  x = A(:,1:n)\y; ys(1:p,n) = x(n);
  for k = 2:n
    ys(:,n) = ys(:,n).*ts+x(n-k+1);
  end
  A(:,n+1) = A(:,n).*t;
end; plot(ts, ys); legend('data points','deg = 0','deg = 1','deg = 2',...
  'deg = 3', 'deg = 4', 'deg = 5',4)
```

3.2.
```
function cp03_02 % least squares fit to surveying data
b = [2.95; 1.74; -1.45; 1.32; 1.23; 4.45; 1.61; 3.21; 0.45; -2.75];
A = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1; 1 -1 0 0; 1 0 -1 0; 1 0 0 -1; ...
  0 1 -1 0; 0 1 0 -1; 0 0 1 -1]; x = A\b
```

3.3.
```
function cp03_03 % least squares fit to performance data for LU
maxk = 10; n = zeros(maxk,1); t = zeros(maxk,1); runs = 1;
for k = 1:maxk
  n(k) = 100+100*(k-1); A = rand(n(k),n(k)); tic;
  for j = 1:runs
    [L, U] = lu(A);
  end
  t(k) = toc/runs;
end; A = ones(length(n),4);
for j = 2:4
  A(:,j) = A(:,j-1).*n;
end
disp('Coefficients of cubic polynomial are'); x = A\t
ns = 100:10:1000; ts = x(1)+ns.*(x(2)+ns.*(x(3)+ns*x(4)));
plot(n, t, 'bx', ns, ts, 'k-');
t_10000 = x(1)+10000*(x(2)+10000*(x(3)+10000*x(4)));
fprintf('Predicted time for n = 10000 is %g\n', t_10000);
```

3.4.
```
function cp03_04 % nearly rank deficient least squares problem
A=[0.16 0.10; 0.17 0.11; 2.02 1.29];
disp('(a)'); b_a=[0.26; 0.28; 3.31]; x_a = A\b_a
disp('(b)'); b_b=[0.27; 0.25; 3.33]; x_b = A\b_b
disp('(c)'); delx_over_x = norm(x_b-x_a)/norm(x_a), condA = cond(A)
cos_theta = norm(A*x_a)/norm(b_a), delb_over_b = norm(b_b-b_a)/norm(b_a)
bound = condA*cos_theta*delb_over_b
```

3.5.
```
function cp03_05 % least squares fit to planetary orbit data
x = [1.02; 0.95; 0.87; 0.77; 0.67; 0.56; 0.44; 0.30; 0.16; 0.01];
y = [0.39; 0.32; 0.27; 0.22; 0.18; 0.15; 0.13; 0.12; 0.13; 0.15];
A = [y.^2 x.*y x y ones(size(x))]; b = x.^2; disp('(a)'); figure(1); hold on;
title('Computer Problem 3.5(a) - Elliptical Orbit'); alpha = A\b
[xs, ys] = meshgrid(-1:0.1:2, -1:0.1:2);
contour(-1:0.1:2, -1:0.1:2, alpha(1)*ys.^2+alpha(2)*xs.*ys+ ...
alpha(3)*xs+alpha(4)*ys+alpha(5)-xs.^2, [0, 0], 'k-');
plot(x, y, 'bx'); disp('(b)'); figure(2); hold on;
title('Computer Problem 3.5(b) - Perturbed Orbit');
x2 = x+(rand(size(x))*0.01-0.005); y2 = y+(rand(size(y))*0.01-0.005);
A2 = [y2.^2 x2.*y2 x2 y2 ones(size(x2))]; b2 = x2.^2; alpha2 = A2\b2
[xs, ys] = meshgrid(-1:0.1:2, -1:0.1:2);
contour(-1:0.1:2, -1:0.1:2, alpha2(1)*ys.^2+alpha2(2)*xs.*ys+ ...
alpha2(3)*xs+alpha2(4)*ys+alpha2(5)-xs.^2, [0, 0], 'r-');
contour(-1:0.1:2, -1:0.1:2, alpha(1)*ys.^2+alpha(2)*xs.*ys+ ...
alpha(3)*xs+alpha(4)*ys+alpha(5)-xs.^2, [0, 0], 'k-');
plot(x, y, 'bx'); disp('(c) See parts (d)-(f) below.');
disp('(d)'); [U, S, V] = svd(A), disp('(e)'); figure(3); hold on;
title('Computer Problem 3.5(e) - Orbits Using SVD'); alpha = zeros(5,1);
for k = 1:5
  alpha = alpha+U(:,k)'*b*V(:,k)/S(k,k)
  [xs, ys] = meshgrid(-1:0.1:2, -1:0.1:2);
  contour(-1:0.1:2, -1:0.1:2, alpha(1)*ys.^2+alpha(2)*xs.*ys+ ...
    alpha(3)*xs+alpha(4)*ys+alpha(5)-xs.^2, [0, 0], 'k-');
```

```
end; plot(x, y, 'bx'); disp('(f)'); figure(4); hold on;
title('Computer Problem 3.5(f) - Perturbed Orbits Using SVD');
[U2, S2, V2] = svd(A2); alpha2 = zeros(5,1);
for k = 1:5
  alpha2 = alpha2+U2(:,k)'*b2*V2(:,k)/S2(k,k)
  [xs, ys] = meshgrid(-1:0.1:2, -1:0.1:2);
  contour(-1:0.1:2, -1:0.1:2, alpha2(1)*ys.^2+alpha2(2)*xs.*ys+ ...
    alpha2(3)*xs+alpha2(4)*ys+alpha2(5)-xs.^2, [0, 0], 'k-');
end; plot(x, y, 'bx'); disp('(g)'); figure(5); hold on;
title('Computer Problem 3.5(g) - Orbit Using Total Least Squares');
[U, S, V] = svd([A b]); alpha3 = -1/V(6, 6)*V(1:5, 6)
[xs, ys] = meshgrid(-1:0.1:2, -1:0.1:2);
contour(-1:0.1:2, -1:0.1:2, alpha3(1)*ys.^2+alpha3(2)*xs.*ys+ ...
alpha3(3)*xs+alpha3(4)*ys+alpha3(5)-xs.^2, [0, 0], 'r-');
contour(-1:0.1:2, -1:0.1:2, alpha(1)*ys.^2+alpha(2)*xs.*ys+ ...
alpha(3)*xs+alpha(4)*ys+alpha(5)-xs.^2, [0, 0], 'k-'); plot(x, y, 'bx');
```

3.6. `function cp03_06 % compute pseudoinverse of matrix using SVD`

```
fprintf('Example with nonsingular matrix:');
A = [1 1 1; 1 2 4; 1 3 9], pseudoinv(A)
fprintf('Example with singular matrix:');
A = [ 1  2  3; 4  5  6; 7  8  9], pseudoinv(A)
fprintf('Example with Hilbert matrix:'); A = hilb(4), pseudoinv(A)

function [B] = pseudoinv(A)
tol = max(size(A))*norm(A)*eps; [U, S, V] = svd(A);
for k=1:min(size(A))
  if S(k,k) > tol, S(k,k) = 1/S(k,k);
  else S(k,k) = 0; end
end; B = V*S'*U';
```

3.7. `function cp03_07 % least squares solution of rank deficient system`

```
A = [0.16 0.10; 0.17 0.11; 2.02 1.29];
b1 = [0.26; 0.28; 3.31]; b2 = [0.27; 0.25; 3.33];
disp('Least squares solution to nearly rank deficient system using tolerance:');
x = lls_rd(A, b1, 1e-3)
disp('Least squares solution without tolerance:'); x = A\b1
disp('Least squares solution to perturbed system using tolerance:');
x = lls_rd(A, b2, 1e-3)
disp('Least squares solution to perturbed system without tolerance:'); x = A\b2
disp('Note that use of tolerance makes solution much less sensitive.');

function [x] = lls_rd(A, b, tol) % rank deficient least squares using SVD
[m, n] = size(A); [U, S, V] = svd(A); x = zeros(n,1);
for k = 1:min(m,n)
  if S(k,k) < S(1,1)*tol, break;
  else x = x+((U(:,k)'*b)/S(k,k))*V(:,k); end
end
```

3.8. `function cp03_08 % compare accuracy of QR and normal equations`

```
m = 21; n = 12; epsilon = 1e-10; t = linspace(0,1,m)'; A = ones(m,n);
```

```
fprintf('True solution before perturbation:'); x = ones(n,1)
for i = 2:n
  A(:,i) = A(:,i-1).*t;
end
y = A*x+epsilon*(2*rand(m,1)-ones(m,1));
fprintf('Solution using Cholesky factorization:');
R = chol(A'*A); x_chol = R\(R'\(A'*y))
fprintf('Solution using QR factorization:'); [Q, R] = qr(A); x_qr = R\(Q'*y)
```

3.9.
```
function cp03_09 % least squares solution using augmented system
m = 21; n = 12; epsilon = 1e-10;
x = ones(n,1); t = linspace(0,1,m)'; A = ones(m,n);
for i = 2:n
  A(:,i) = A(:,i-1).*t;
end
y = A*x+epsilon*(2*rand(m,1)-ones(m,1)); alpha = max(max(abs(A)))/1000;
x_aug = [alpha*eye(m,m) A; A' zeros(n,n)]\[y; zeros(n,1)];
fprintf('Solution using augmented system:'); x = x_aug(m+1:m+n)
```

3.10.
```
function cp03_10 % compute covariance matrix
fprintf('For matrix:'); A = [1 1 1; 1 2 4; 1 3 9; 1 4 16], [Q, R] = qr(A);
fprintf('Covariance matrix computed using R:'); A_covR = inv(R'*R)
fprintf('Covariance matrix computed using A:'); A_covA = inv(A'*A)
fprintf('For matrix:'); A = [1 0 0; 0 1 0; 0 0 1; -1 1 0; -1 0 1; 0 -1  1]
[Q, R] = qr(A);
fprintf('Covariance matrix computed using R:'); A_covR = inv(R'*R)
fprintf('Covariance matrix computed using A:'); A_covA = inv(A'*A)
```

3.11. This problem is rather pointless in MATLAB, and it is therefore omitted.

3.12.
```
function cp03_12 % methods for computing orthogonal basis
m = 12; x = [2:m]'; y = zeros(m-1,6);
for n = 2:m
  A = hilb(n);
% Classical Gram-Schmidt
  Q = zeros(n,n); R = zeros(n,n);
  for k = 1:n
    Q(:,k) = A(:,k);
    for j = 1:k-1
      R(j,k) = Q(:,j)'*A(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
    end; R(k,k) = norm(Q(:,k));
    if R(k,k) == 0, break; end;
    Q(:,k) = Q(:,k)/R(k,k);
  end; y(n-1,1) = -log10(norm(eye(size(Q))-Q'*Q));
% Modified Gram-Schmidt
  Q = A; R = zeros(n,n);
  for k = 1:n
    R(k,k) = norm(Q(:,k));
    if R(k,k) == 0, break; end;
    Q(:,k) = Q(:,k)/R(k,k);
    for j = k+1:n
```

```
      R(k,j) = Q(:,k)'*Q(:,j); Q(:,j) = Q(:,j)-R(k,j)*Q(:,k);
    end
  end; y(n-1,2) = -log10(norm(eye(size(Q))-Q'*Q));
% Classical Gram-Schmidt with reorthogonalization
  Q = zeros(n,n); R = zeros(n,n);
  for k = 1:n
    Q(:,k) = A(:,k);
    for j = 1:k-1
      R(j,k) = Q(:,j)'*A(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
    end; R(k,k) = norm(Q(:,k));
    if R(k,k) == 0, break; end;
    Q(:,k) = Q(:,k)/R(k,k);
  end; B = Q; Q = zeros(n,n); R = zeros(n,n);
  for k = 1:n
    Q(:,k) = B(:,k);
    for j = 1:k-1
      R(j,k) = Q(:,j)'*B(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
    end; R(k,k) = norm(Q(:,k));
    if R(k,k) == 0, break; end;
    Q(:,k) = Q(:,k)/R(k,k);
  end; y(n-1,3) = -log10(norm(eye(size(Q))-Q'*Q));
% Householder QR
  [Q, R] = qr(A); y(n-1,4) = -log10(norm(eye(size(Q))-Q'*Q));
% SVD
  [Q, S, V] = svd(A); y(n-1,5) = -log10(norm(eye(size(Q))-Q'*Q));
% Normal Equations
  if min(eig((A'*A))) <= 0, y(n-1,6) = 0;
  else
    R = chol(A'*A); Q = A*inv(R); y(n-1,6) = -log10(norm(eye(size(Q))-Q'*Q));
  end
end; plot(x, y);
title('Computer Problem 3.12 - Accuracy of Orthogonalization Methods');
xlabel('Order of Hilbert Matrix');
ylabel('Digits of Accuracy in Orthogonal Basis');
legend('CGS','MGS','CGS-R','HQR','SVD','Chol',3);
```

3.13.

```
function cp03_13 % compare accuracy of least squares methods
eps1 = 2*sqrt(eps); eps2 = sqrt(eps); eps3 = 2*eps; eps4 = eps;
A1 = [1 1 1; eps1 0 0; 0 eps1 0; 0 0 eps1];
A2 = [1 1 1; eps2 0 0; 0 eps2 0; 0 0 eps2];
A3 = [1 1 1; eps3 0 0; 0 eps3 0; 0 0 eps3];
A4 = [1 1 1; eps4 0 0; 0 eps4 0; 0 0 eps4]; b = [1; 0; 0; 0];
x1 = (1/(3+4*eps))*ones(3,1); x2 = (1/(3+eps))*ones(3,1);
x3 = (1/(3+4*eps^2))*ones(3,1); x4 = (1/(3+eps^2))*ones(3,1);
e(1,1) = norm(x1-lls_ne(A1, b)); e(1,2) = norm(x2-lls_ne(A2, b));
e(1,3) = norm(x3-lls_ne(A3, b)); e(1,4) = norm(x4-lls_ne(A4, b));
e(2,1) = norm(x1-lls_as(A1, b)); e(2,2) = norm(x2-lls_as(A2, b));
e(2,3) = norm(x3-lls_as(A3, b)); e(2,4) = norm(x4-lls_as(A4, b));
e(3,1) = norm(x1-lls_hqr(A1, b)); e(3,2) = norm(x2-lls_hqr(A2, b));
e(3,3) = norm(x3-lls_hqr(A3, b)); e(3,4) = norm(x4-lls_hqr(A4, b));
```

```
e(4,1) = norm(x1-lls_gqr(A1, b)); e(4,2) = norm(x2-lls_gqr(A2, b));
e(4,3) = norm(x3-lls_gqr(A3, b)); e(4,4) = norm(x4-lls_gqr(A4, b));
e(5,1) = norm(x1-lls_cgs(A1, b)); e(5,2) = norm(x3-lls_cgs(A2, b));
e(5,3) = norm(x3-lls_cgs(A3, b)); e(5,4) = norm(x4-lls_cgs(A4, b));
e(6,1) = norm(x1-lls_mgs(A1, b)); e(6,2) = norm(x2-lls_mgs(A2, b));
e(6,3) = norm(x3-lls_mgs(A3, b)); e(6,4) = norm(x4-lls_mgs(A4, b));
e(7,1) = norm(x1-lls_cgsr(A1, b)); e(7,2) = norm(x2-lls_cgsr(A2, b));
e(7,3) = norm(x3-lls_cgsr(A3, b)); e(7,4) = norm(x4-lls_cgsr(A4, b));
e(8,1) = norm(x1-lls_svd(A1, b)); e(8,2) = norm(x2-lls_svd(A2, b));
e(8,3) = norm(x3-lls_svd(A3, b)); e(8,4) = norm(x4-lls_svd(A4, b));
disp(' '); disp('Exact solution is (1/(3+epsilon^2)*ones(3,1)).');
disp(' '); disp('Norm of error for various values of epsilon:');
fprintf('\nepsilon          2sqrt(emach)  sqrt(emach)   2 emach       emach\n')
fprintf('                  %10.4e   %10.4e   %10.4e   %10.4e\n\n', ...
  eps1, eps2, eps3, eps4);
fprintf('Normal Equations   %10.4e   %10.4e   %10.4e   %10.4e\n', e(1,1:4));
fprintf('Augmented System   %10.4e   %10.4e   %10.4e   %10.4e\n', e(2,1:4));
fprintf('Householder QR     %10.4e   %10.4e   %10.4e   %10.4e\n', e(3,1:4));
fprintf('Givens QR          %10.4e   %10.4e   %10.4e   %10.4e\n', e(4,1:4));
fprintf('Classical G-S      %10.4e   %10.4e   %10.4e   %10.4e\n', e(5,1:4));
fprintf('Modified G-S       %10.4e   %10.4e   %10.4e   %10.4e\n', e(6,1:4));
fprintf('CGS w. Reorthog.   %10.4e   %10.4e   %10.4e   %10.4e\n', e(7,1:4));
fprintf('SVD                %10.4e   %10.4e   %10.4e   %10.4e\n', e(8,1:4));

function [x] = lls_ne(A, b) % least squares using normal equations
x = (A'*A)\(A'*b);

function [x] = lls_as(A, b) % least squares using augmented system
[m, n] = size(A); alpha = max(max(abs(A)))/1000;
x_aug = [alpha*eye(m,m) A; A' zeros(n,n)]\[b; zeros(n,1)];
x = x_aug(m+1:m+n);

function [x] = lls_hqr(A, b) % least squares using Householder QR
[m, n] = size(A);
for k = 1:n
  alpha = -sign(A(k,k))*norm(A(k:m,k));
  v = [zeros(k-1,1); A(k,k)-alpha; A(k+1:m,k)]; beta = v'*v;
  if beta == 0, continue; end
  for j = k:n
    gamma = v'*A(:,j); A(:,j) = A(:,j)-(2*gamma/beta)*v;
  end; gamma = v'*b; b = b-(2*gamma/beta)*v;
end; x = A(1:n,1:n)\b(1:n);

function [x] = lls_gqr(A, b) % least squares using Givens QR
[m, n] = size(A);
for k = 1:n
  for j = k+1:m
    if A(j,k) ~= 0
      a1 = A(k,k); a2 = A(j,k);
      if abs(a1) > abs(a2), t = a2/a1; c = 1/sqrt(1+t^2); s = c*t;
```

```
      else t = a1/a2; s = 1/sqrt(1+t^2); c = s*t; end
      for i = k:n
        temp = c*A(k,i)+s*A(j,i); A(j,i) = c*A(j,i)-s*A(k,i); A(k,i) = temp;
      end
      temp = c*b(k)+s*b(j); b(j) = c*b(j)-s*b(k); b(k) = temp;
    end
  end
end; x = A(1:n,1:n)\b(1:n);

function [x] = lls_cgs(A, b) % least squares using classical Gram-Schmidt
A = [A b]; [m, n] = size(A); Q = zeros(m,n); R = zeros(n,n);
for k = 1:n
  Q(:,k) = A(:,k);
  for j = 1:k-1
    R(j,k) = Q(:,j)'*A(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
  end; R(k,k) = norm(Q(:,k));
  if R(k,k) == 0, break; end
  Q(:,k) = Q(:,k)/R(k,k);
end; x = R(1:n-1,1:n-1)\R(1:n-1,n);

function [x] = lls_mgs(A, b) % least squares using modified Gram-Schmidt
A = [A b]; [m, n] = size(A); R = zeros(n,n);
for k = 1:n
  R(k,k) = norm(A(:,k));
  if R(k,k) == 0, break; end
  A(:,k) = A(:,k)/R(k,k);
  for j = k+1:n
    R(k,j) = A(:,k)'*A(:,j); A(:,j) = A(:,j)-R(k,j)*A(:,k);
  end
end; x = R(1:n-1,1:n-1)\R(1:n-1,n);

function [x] = lls_cgsr(A, b) % least squares using CGS with reorthogonalization
A = [A b]; [m, n] = size(A); Q = zeros(m,n); R = zeros(n,n);
for k = 1:n
  Q(:,k) = A(:,k);
  for j = 1:k-1
    R(j,k) = Q(:,j)'*A(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
  end; R(k,k) = norm(Q(:,k));
  if R(k,k) == 0, break; end
  Q(:,k) = Q(:,k)/R(k,k);
end; Rold = R; A = Q; Q = zeros(m,n); R = zeros(n,n);
for k = 1:n
  Q(:,k) = A(:,k);
  for j = 1:k-1
    R(j,k) = Q(:,j)'*A(:,k); Q(:,k) = Q(:,k)-R(j,k)*Q(:,j);
  end; R(k,k) = norm(Q(:,k));
  if R(k,k) == 0, break; end
  Q(:,k) = Q(:,k)/R(k,k);
end; R = R*Rold; x = R(1:n-1,1:n-1)\R(1:n-1,n);
```

```
function [x] = lls_svd(A, b) % least squares using SVD
[m, n] = size(A); [U, S, V] = svd(A); x = zeros(n,1);
for k = 1:min(m,n)
  if S(k,k) ~= 0, x = x+((U(:,k)'*b)/S(k,k))*V(:,k); end
end
```

Chapter 4

Eigenvalue Problems

Exercises

4.1. (a) It is easy to see that the matrix

$$\boldsymbol{A} - 5\,\boldsymbol{I} = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 0 & 2 & 4 & 5 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

is singular, for example by observing that its last two rows are linearly dependent or that its determinant is 0, so 5 is an eigenvalue of $\boldsymbol{A}$. (b) Any nonzero multiple of $[\,-3 \quad 2 \quad -1 \quad 0\,]^T$ is an eigenvector of $\boldsymbol{A}$ corresponding to the eigenvalue 5.

4.2. Since the matrix is triangular, its eigenvalues are its diagonal entries, 1, 2, 3. Any nonzero multiple of $[\,1 \quad 0 \quad 0\,]^T$ is an eigenvector corresponding to the eigenvalue 1. Any nonzero multiple of $[\,2 \quad 1 \quad 0\,]^T$ is an eigenvector corresponding to the eigenvalue 2. Any nonzero multiple of $[\,1 \quad 1 \quad 1\,]^T$ is an eigenvector corresponding to the eigenvalue 3.

4.3. (a) $\det(\boldsymbol{A}-\lambda\boldsymbol{I}) = \lambda^2 - 2\lambda - 3$. (b) $\lambda_1 = 3$, $\lambda_2 = -1$. (c) $\lambda_1 = 3$, $\lambda_2 = -1$. (d) Any nonzero multiple of $[\,1 \quad 0.5\,]^T$ is an eigenvector corresponding to the eigenvalue 3. Any nonzero multiple of $[\,1 \quad -0.5\,]^T$ is an eigenvector corresponding to the eigenvalue -1. (e) $\boldsymbol{y}_1 = \boldsymbol{A}\boldsymbol{x}_0 = [\,5 \quad 2\,]^T$, $\boldsymbol{x}_1 = \boldsymbol{y}_1/\|\boldsymbol{y}_1\|_\infty = [\,1 \quad 0.4\,]^T$. (f) It will converge to a multiple of the eigenvector corresponding to the dominant eigenvalue, namely $[\,1 \quad 0.5\,]^T$. (g) 3.5 (h) It will converge to a multiple of the eigenvector corresponding to the eigenvalue of smallest magnitude, namely $[\,1 \quad -0.5\,]^T$. (i) 3, since 3 is closer to 2 than -1 is to 2. (j) It will converge to a triangular matrix, since $\boldsymbol{A}$ is not symmetric.

4.4. $\boldsymbol{A} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$, $\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

4.5. (a) If $\boldsymbol{x} = [\,1 \quad 1 \quad \cdots \quad 1\,]^T$, then $\boldsymbol{Ax} = \alpha\boldsymbol{x}$, so that α is an eigenvalue of $\boldsymbol{A}$. (b) Any nonzero multiple of $[\,1 \quad 1 \quad \cdots \quad 1\,]^T$ is a corresponding eigenvector.

4.6. $\boldsymbol{A}$ is singular if, and only if, $\boldsymbol{Az} = \boldsymbol{o} = 0\boldsymbol{z}$ for some nonzero vector $\boldsymbol{z}$, i.e., zero is an eigenvalue with corresponding eigenvector $\boldsymbol{z}$.

4.7. (a) $\boldsymbol{A}^T - \lambda\boldsymbol{I} = (\boldsymbol{A} - \lambda\boldsymbol{I})^T$ is singular if, and only if, $\boldsymbol{A} - \lambda\boldsymbol{I}$ is singular, so $\boldsymbol{A}$ and $\boldsymbol{A}^T$ have the same eigenvalues. (b) $\boldsymbol{A}^H - \bar{\lambda}\boldsymbol{I} = (\boldsymbol{A} - \lambda\boldsymbol{I})^H$ is singular if, and only if, $\boldsymbol{A} - \lambda\boldsymbol{I}$ is singular, so the eigenvalues of $\boldsymbol{A}^H$ are complex conjugates of the eigenvalues of $\boldsymbol{A}$.

4.8. By definition, $\boldsymbol{A}$ is diagonalizable means that $\boldsymbol{T}^{-1}\boldsymbol{AT} = \boldsymbol{D}$ for some nonsingular matrix $\boldsymbol{T}$ and diagonal matrix $\boldsymbol{D}$, which is equivalent to $\boldsymbol{AT} = \boldsymbol{DT}$, with all n columns of $\boldsymbol{T}$ linearly independent. The latter equation says that the columns of $\boldsymbol{T}$ are eigenvectors corresponding to the eigenvalues given by the diagonal entries of $\boldsymbol{D}$. Thus, $\boldsymbol{A}$ is diagonalizable if, and only if, it has a complete set of n linearly independent eigenvectors.

4.9. (a) First, we show that $p(\bar{z}) = \overline{p(z)}$ for all complex z. Since the coefficients are real, we have

$$\begin{aligned} p(\bar{z}) &= c_0 + c_1\bar{z} + \cdots + c_n\bar{z}^n = c_0 + c_1\bar{z} + \cdots + c_n\overline{z^n} \\ &= c_0 + \overline{c_1 z} + \cdots + \overline{c_n z^n} = \overline{c_0 + c_1 z + \cdots + c_n z^n} = \overline{p(z)}. \end{aligned}$$

Now, if z is a root of $p(\lambda) = 0$, then $p(\bar{z}) = \overline{p(z)} = \bar{0} = 0$, so $\bar{z}$ is also a root of $p(\lambda) = 0$. (b) We have $\boldsymbol{Ax} = \lambda\boldsymbol{x}$, so $\overline{\boldsymbol{Ax}} = \overline{\lambda\boldsymbol{x}}$, which is equivalent to $\bar{\boldsymbol{A}}\bar{\boldsymbol{x}} = \bar{\lambda}\bar{\boldsymbol{x}}$, and since $\boldsymbol{A}$ is real, this says $\boldsymbol{A}\bar{\boldsymbol{x}} = \bar{\lambda}\bar{\boldsymbol{x}}$. Hence, $\bar{\lambda}$ is an eigenvalue of $\boldsymbol{A}$ corresponding to the eigenvector $\bar{\boldsymbol{x}}$.

4.10. (a) Let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then $(\boldsymbol{x}^H\boldsymbol{Ax})^H = \boldsymbol{x}^H\boldsymbol{A}^H\boldsymbol{x} = \boldsymbol{x}^H\boldsymbol{Ax}$ is real, and therefore $\lambda = \boldsymbol{x}^H\boldsymbol{Ax}/\boldsymbol{x}^H\boldsymbol{x}$ is real. (b) If $\boldsymbol{A}$ is real and symmetric, then $\boldsymbol{A}^H = \boldsymbol{A}$, and so by applying part (a) we see that all its eigenvalues are real.

4.11. $\boldsymbol{A} = \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix}$, whose eigenvalues are $1 \pm i$.

4.12. Let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then we have $\boldsymbol{x}^H\boldsymbol{Ax} = \boldsymbol{x}^H(\lambda\boldsymbol{x}) = \lambda\boldsymbol{x}^H\boldsymbol{x}$, or $\lambda = \boldsymbol{x}^H\boldsymbol{Ax}/\boldsymbol{x}^H\boldsymbol{x}$. But $\boldsymbol{x}^H\boldsymbol{x}$ is positive, and $\boldsymbol{x}^H\boldsymbol{Ax} > 0$ since $\boldsymbol{A}$ is positive definite, so we must have $\lambda > 0$.

4.13. Let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ implies that $|\lambda| \cdot \|\boldsymbol{x}\| \leq \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|$, which implies that $|\lambda| \leq \|\boldsymbol{A}\|$. Since this inequality holds for any eigenvalue, in particular it must hold for the eigenvalue of largest modulus, so that $\rho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|$.

4.14. (a) Yes, $\alpha = 0$ yields all real eigenvalues, namely the diagonal entries. (b) No, because complex eigenvalues of a real matrix must occur in conjugate pairs, so a real matrix of odd order must have at least one real eigenvalue.

4.15. If $\boldsymbol{A}$ is nonsingular, then $\boldsymbol{A}^{-1}(\boldsymbol{AB})\boldsymbol{A} = \boldsymbol{BA}$, so $\boldsymbol{AB}$ and $\boldsymbol{BA}$ are similar.

4.16. Let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ implies that $\boldsymbol{A}^{-1}\boldsymbol{x} = (1/\lambda)\boldsymbol{x}$. We have thus shown that (a) the eigenvalues of $\boldsymbol{A}^{-1}$ are the reciprocals of the eigenvalues of $\boldsymbol{A}$, and (b) the eigenvectors of $\boldsymbol{A}^{-1}$ are the same as the eigenvectors of $\boldsymbol{A}$.

4.17. Let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ implies that $\boldsymbol{A}^2\boldsymbol{x} = \lambda\boldsymbol{Ax} = \lambda^2\boldsymbol{x}$, so λ^2 is an eigenvalue of $\boldsymbol{A}^2$.

4.18. (a) Suppose $\boldsymbol{A}^k = \boldsymbol{O}$, and let λ be an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{x}$. Then $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ implies that $\lambda^k\boldsymbol{x} = \boldsymbol{A}^k\boldsymbol{x} = \boldsymbol{o}$. Since $\boldsymbol{x} \neq \boldsymbol{o}$, we must therefore have $\lambda = 0$. Thus, all of the eigenvalues of $\boldsymbol{A}$ are zero. (b) If $\boldsymbol{A}$ is normal, then it is unitarily diagonalizable, so we have $\boldsymbol{A} = \boldsymbol{U}^H\boldsymbol{DU}$ for some unitary matrix $\boldsymbol{U}$ and diagonal matrix $\boldsymbol{D}$. But by part (a) we have $\boldsymbol{D} = \boldsymbol{O}$, and hence $\boldsymbol{A} = \boldsymbol{O}$.

4.19. If $\boldsymbol{A}^2 = \boldsymbol{A}$, then the eigenvalues must be such that $\lambda^2 = \lambda$. The only possible values which satisfy this are 0 and 1.

4.20. (a) $\lambda\boldsymbol{y}^H\boldsymbol{x} = \boldsymbol{y}^H(\lambda\boldsymbol{x}) = \boldsymbol{y}^H(\boldsymbol{Ax}) = \boldsymbol{y}^H(\boldsymbol{A}^H\boldsymbol{x}) = (\boldsymbol{Ay})^H\boldsymbol{x} = \mu\boldsymbol{y}^H\boldsymbol{x}$, which implies $(\lambda - \mu)\boldsymbol{y}^H\boldsymbol{x} = 0$, and hence $\boldsymbol{y}^H\boldsymbol{x} = 0$, since $\lambda \neq \mu$. (b) $\lambda\boldsymbol{y}^H\boldsymbol{x} = \boldsymbol{y}^H(\lambda\boldsymbol{x}) = \boldsymbol{y}^H\boldsymbol{Ax} = \mu\boldsymbol{y}^H\boldsymbol{x}$, which implies $(\lambda - \mu)\boldsymbol{y}^H\boldsymbol{x} = 0$, and hence $\boldsymbol{y}^H\boldsymbol{x} = 0$, since $\lambda \neq \mu$. (c) We just saw in part (b) that $\boldsymbol{y}$ is orthogonal to any eigenvector corresponding to an eigenvalue distinct from λ, which by assumption is simple. Thus, if $\boldsymbol{y}$ were also orthogonal to $\boldsymbol{x}$, then it would be orthogonal to the invariant subspace generated by *all* of the eigenvectors of $\boldsymbol{A}$ (the whole n-dimensional space if $\boldsymbol{A}$ is nondefective) or else $\boldsymbol{y}$ is the zero vector, neither of which is possible. Thus, $\boldsymbol{y}^H\boldsymbol{x} \neq 0$.

4.21. (a) $\mathcal{S}_\lambda$ is a subspace if it is closed under addition and scalar multiplication. If $\boldsymbol{Ax} = \lambda\boldsymbol{x}$ and $\boldsymbol{Ay} = \lambda\boldsymbol{y}$, then $\boldsymbol{A}(\boldsymbol{x} + \boldsymbol{y}) = \boldsymbol{Ax} + \boldsymbol{Ay} = \lambda\boldsymbol{x} + \lambda\boldsymbol{y} = \lambda(\boldsymbol{x} + \boldsymbol{y})$. Also, for any scalar α, if $\boldsymbol{Ax} = \lambda\boldsymbol{x}$, then $\boldsymbol{A}(\alpha\boldsymbol{x}) = \alpha\boldsymbol{Ax} = \alpha\lambda\boldsymbol{x} = \lambda(\alpha\boldsymbol{x})$. Hence $\mathcal{S}_\lambda$ is a subspace. (b) λ is an eigenvalue of $\boldsymbol{A} \Leftrightarrow$ there is some $\boldsymbol{x} \neq \boldsymbol{o}$ such that $\boldsymbol{Ax} = \lambda\boldsymbol{x} \Leftrightarrow$ there is some $\boldsymbol{x} \neq \boldsymbol{o}$ such that $\boldsymbol{x} \in \mathcal{S}_\lambda \Leftrightarrow \mathcal{S}_\lambda \neq \{\boldsymbol{o}\}$.

4.22. (a) Let $\boldsymbol{v}$ be the $(n-k)$-vector of zeros. Then

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{O} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{u} + \boldsymbol{A}_{12}\boldsymbol{v} \\ \boldsymbol{A}_{22}\boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \lambda\boldsymbol{u} \\ \boldsymbol{o} \end{bmatrix} = \lambda \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix}$$

(b) Let $\boldsymbol{u} = -\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12}\boldsymbol{v}$. Then

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{O} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11}\boldsymbol{u} + \boldsymbol{A}_{12}\boldsymbol{v} \\ \boldsymbol{A}_{22}\boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{o} \\ \lambda\boldsymbol{v} \end{bmatrix} = \lambda \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix}$$

(c) If λ is an eigenvalue of $\boldsymbol{A}$, then $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$. Hence

$$\det\left(\begin{bmatrix} \boldsymbol{A}_{11} - \lambda\boldsymbol{I} & \boldsymbol{A}_{12} \\ \boldsymbol{O} & \boldsymbol{A}_{22} - \lambda\boldsymbol{I} \end{bmatrix}\right) = \det(\boldsymbol{A}_{11} - \lambda\boldsymbol{I})\det(\boldsymbol{A}_{22} - \lambda\boldsymbol{I}) = 0.$$

The only way for this to occur is if either $\det(\boldsymbol{A}_{11} - \lambda \boldsymbol{I}) = 0$, or $\det(\boldsymbol{A}_{22} - \lambda \boldsymbol{I}) = 0$. Hence λ must be either an eigenvalue of $\boldsymbol{A}_{11}$ or $\boldsymbol{A}_{22}$. (d) From the above arguments, we have that if λ is an eigenvalue of $\boldsymbol{A}$, then it must either be an eigenvalue of $\boldsymbol{A}_{11}$ or $\boldsymbol{A}_{22}$. We also have that if λ is an eigenvalue of $\boldsymbol{A}_{11}$ or $\boldsymbol{A}_{22}$, then λ is an eigenvalue of $\boldsymbol{A}$.

4.23. (a) The characteristic polynomial of a matrix $\boldsymbol{A}$ can be written as

$$p(\lambda) = (-1)^n(\lambda - \lambda_1)(\lambda - \lambda_2)\cdots(\lambda - \lambda_n).$$

In this form, we can see that $p(0) = \prod_{i=1}^n \lambda_i$. Also, by definition, $p(0) = \det(\boldsymbol{A})$, showing that $\det(\boldsymbol{A}) = \prod_{i=1}^n \lambda_i$. (b) Once again, express the characteristic polynomial of $\boldsymbol{A}$ as

$$p(\lambda) = (-1)^n(\lambda - \lambda_1)(\lambda - \lambda_2)\cdots(\lambda - \lambda_n).$$

Expanding this to see the coefficient for λ^{n-1} we get $(-1)^{n+1}(\lambda_1 + \lambda_2 + \cdots + \lambda_n)$. To see that this is equal to the trace of $\boldsymbol{A}$, we observe that the coefficient of λ^{n-1} can also be calculated from the definition of the characteristic polynomial, $p(\lambda) = \det(\boldsymbol{A} - \lambda \boldsymbol{I})$. In this form, while expanding out the determinant, λ^{n-1} shows up only when multiplied by each of the a_{ii} components, and so the final coefficient derives from the sum of these (times $(-1)^{n+1}$). This yields $\text{trace}(\boldsymbol{A}) = \sum_{i=1}^n \lambda_i$.

4.24. (a) If $\boldsymbol{A}$ is rank one, then it has at least one nonzero column, and we let $\boldsymbol{u}$ be that vector. Each column of $\boldsymbol{A}$ must be a scalar, say v_i, times $\boldsymbol{u}$, and we let $\boldsymbol{v}$ be the vector whose components are the v_i. Then we have $\boldsymbol{A} = \boldsymbol{u}\boldsymbol{v}^T$. (b) $\boldsymbol{A}\boldsymbol{u} = (\boldsymbol{u}\boldsymbol{v}^T)\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{v}^T\boldsymbol{u}) = (\boldsymbol{u}^T\boldsymbol{v})\boldsymbol{u}$, so $\boldsymbol{u}^T\boldsymbol{v}$ is an eigenvalue of $\boldsymbol{A}$ with corresponding eigenvector $\boldsymbol{u}$. (c) The other eigenvalues of $\boldsymbol{A}$ are all 0. (d) Power iteration converges in a single iteration provided the start vector has some component in the direction of $\boldsymbol{u}$.

4.25. From Exercise 4.23, we know that $\det(\boldsymbol{A}) = \prod_{i=1}^n \lambda_i$. Now the eigenvalues of $\boldsymbol{I} + \boldsymbol{A}$ are given by $1 + \lambda_i$, where λ_i is an eigenvalue of $\boldsymbol{A}$. Hence, using the result from Exercise 4.24, the eigenvalues of $\boldsymbol{I} + \boldsymbol{u}\boldsymbol{v}^T$ are $1 + \boldsymbol{u}^T\boldsymbol{v}$ and 1. Thus, $\det(\boldsymbol{I} + \boldsymbol{u}\boldsymbol{v}^T) = 1 + \boldsymbol{u}^T\boldsymbol{v}$.

4.26. (a) Without loss of generality, assume that $\boldsymbol{T}$ is upper triangular. Consider the matrix $\boldsymbol{A} = \boldsymbol{T}\boldsymbol{T}^H = \boldsymbol{T}^H\boldsymbol{T}$. From $\boldsymbol{T}^H\boldsymbol{T}$, we see that $a_{11} = t_{11}\bar{t}_{11} = |t_{11}|^2$. From $\boldsymbol{T}\boldsymbol{T}^H$, on the other hand, we see that $a_{11} = \sum_{j=1}^n t_{1j}\bar{t}_{1j} = \sum_{j=1}^n |t_{1j}|^2$. Therefore, since all the terms in the sum are nonnegative, we must have $t_{1j} = 0$ for $j = 2, \ldots, n$. Similar reasoning for a_{22} shows that $t_{2j} = 0$ for $j = 3, \ldots, n$, and so on up to a_{nn}. Thus, $\boldsymbol{T}$ must be diagonal.

(b) First assume that $\boldsymbol{A}$ is normal. Let $\boldsymbol{U}^H\boldsymbol{A}\boldsymbol{U} = \boldsymbol{B}$ be the Schur decomposition of $\boldsymbol{A}$, where $\boldsymbol{U}$ is unitary and $\boldsymbol{B}$ is triangular. Now, $\boldsymbol{B}^H\boldsymbol{B} = (\boldsymbol{U}^H\boldsymbol{A}^H\boldsymbol{U})(\boldsymbol{U}^H\boldsymbol{A}\boldsymbol{U}) = \boldsymbol{U}^H\boldsymbol{A}^H\boldsymbol{A}\boldsymbol{U} = \boldsymbol{U}^H\boldsymbol{A}\boldsymbol{A}^H\boldsymbol{U} = (\boldsymbol{U}^H\boldsymbol{A}\boldsymbol{U})(\boldsymbol{U}^H\boldsymbol{A}^H\boldsymbol{U}) = \boldsymbol{B}\boldsymbol{B}^H$, so $\boldsymbol{B}$ is also normal. Using the result from part (a), $\boldsymbol{B}$ is diagonal, and thus $\boldsymbol{U}$ unitarily diagonalizes $\boldsymbol{A}$. Now, assume that $\boldsymbol{A}$ is unitarily diagonalizable, $\boldsymbol{U}^H\boldsymbol{A}\boldsymbol{U} = \boldsymbol{D}$, where $\boldsymbol{U}$ is unitary and $\boldsymbol{D}$ is diagonal. Then $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^H$, and therefore $\boldsymbol{A}^H\boldsymbol{A} = (\boldsymbol{U}^H\boldsymbol{D}^H\boldsymbol{U})(\boldsymbol{U}^H\boldsymbol{D}\boldsymbol{U}) = \boldsymbol{U}^H\boldsymbol{D}^H\boldsymbol{D}\boldsymbol{U} = (\boldsymbol{U}^H\boldsymbol{D}\boldsymbol{U})(\boldsymbol{U}^H\boldsymbol{D}^H\boldsymbol{U}) = \boldsymbol{A}\boldsymbol{A}^H$, and thus $\boldsymbol{A}$ is normal.

4.27. (a) The eigenvalues of $\boldsymbol{I} - \boldsymbol{A}$ are given by $1 - \lambda_i$ where λ_i is an eigenvalue of $\boldsymbol{A}$. Since $\rho(A) < 1$, the eigenvalues of $\boldsymbol{I} - \boldsymbol{A}$ therefore must all lie in the interior of a disk in the complex plane of radius 1 centered at -1. Thus, none of the eigenvalues of can be 0, and hence the matrix is nonsingular. (b) To see that $(\boldsymbol{I} - \boldsymbol{A})^{-1} = \sum_{k=0}^{\infty} \boldsymbol{A}^k$, we simply multiply it by $\boldsymbol{I} - \boldsymbol{A}$ and see that this yields $\boldsymbol{I}$: $(\boldsymbol{I} - \boldsymbol{A}) \sum_{k=0}^{\infty} \boldsymbol{A}^k = \sum_{k=0}^{\infty} \boldsymbol{A}^k - \sum_{k=1}^{\infty} \boldsymbol{A}^k = \boldsymbol{A}^0 = \boldsymbol{I}$.

4.28. (a) Power iteration with shift can converge only to either λ_1 or λ_n, since no matter what shift is chosen, these are the only two that can be farthest from the shift value. (b) For power iteration to converge to λ_1 with the best convergence rate, the required shift is $(\lambda_2 + \lambda_n)/2$. For it to converge to λ_n with the best convergence rate, the required shift is $(\lambda_1 + \lambda_{n-1})/2$. ($c$) Inverse iteration can converge to any of the eigenvalues. To converge to λ_i with the best convergence rate, λ_i should be chosen as the shift.

4.29. (a) Since $\boldsymbol{C} = \boldsymbol{A} + i\boldsymbol{B}$ is Hermitian, and since $\boldsymbol{C}^H = \boldsymbol{A}^T - i\boldsymbol{B}^T$, we must have $\boldsymbol{A} = \boldsymbol{A}^T$ and $\boldsymbol{B} = -\boldsymbol{B}^T$. This is exactly what is required for $\tilde{\boldsymbol{C}}$ to be symmetric. (b) Since $\boldsymbol{x} + i\boldsymbol{y}$ is an eigenvalue of $\boldsymbol{C}$, we must have $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y} + i(\boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{x}) = \lambda\boldsymbol{x} + i\lambda\boldsymbol{y}$. Equating real and imaginary parts gives $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y} = \lambda\boldsymbol{x}$ and $\boldsymbol{A}\boldsymbol{y} + \boldsymbol{B}\boldsymbol{x} = \lambda\boldsymbol{y}$. Now,

$$\tilde{\boldsymbol{C}} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}\boldsymbol{x} - \boldsymbol{B}\boldsymbol{y} \\ \boldsymbol{B}\boldsymbol{x} + \boldsymbol{A}\boldsymbol{y} \end{bmatrix} = \lambda \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$$

and

$$\tilde{\boldsymbol{C}} \begin{bmatrix} -\boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{A}\boldsymbol{y} - \boldsymbol{B}\boldsymbol{x} \\ -\boldsymbol{B}\boldsymbol{y} + \boldsymbol{A}\boldsymbol{x} \end{bmatrix} = \lambda \begin{bmatrix} -\boldsymbol{y} \\ \boldsymbol{x} \end{bmatrix},$$

showing that these are eigenvalue-eigenvector pairs for $\tilde{\boldsymbol{C}}$. (c) This is not really a good approach because it requires twice as much storage and at least twice as much work as treating the complex Hermitian eigenvalue problem directly, and it also turns simple eigenvalues into multiple eigenvalues.

4.30. (a) i is an eigenvalue of multiplicity two. (b) There is only one linearly independent eigenvector, $[\, i \quad 1 \,]^T$. (c) In the real symmetric or complex Hermitian cases, we are guaranteed to have n linearly independent eigenvectors even if there are repeated eigenvalues.

4.31. (a) Let λ be an eigenvalue of $\boldsymbol{Q}$ with corresponding eigenvector $\boldsymbol{x}$. Then $\boldsymbol{Q}\boldsymbol{x} = \lambda\boldsymbol{x}$ implies that $\|\boldsymbol{Q}\boldsymbol{x}\|_2 = \|\lambda\boldsymbol{x}\|_2$. Since an orthogonal matrix preserves the 2-norm, we have $\|\boldsymbol{x}\|_2 = |\lambda| \cdot \|\boldsymbol{x}\|_2$, which implies that $|\lambda| = 1$. (b) The singular values of $\boldsymbol{Q}$ are the nonnegative square roots of the eigenvalues of $\boldsymbol{Q}^T\boldsymbol{Q}$. But since $\boldsymbol{Q}$ is orthogonal, $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$, and hence the singular values of $\boldsymbol{Q}$ are all equal to 1.

4.32. (a) The eigenvalues of $\boldsymbol{I} + \boldsymbol{A}$ are given by $1 + \lambda_i$ where λ_i is an eigenvalue of $\boldsymbol{A}$. Thus, since the eigenvalues of $-2(\boldsymbol{v}\boldsymbol{v}^T)/(\boldsymbol{v}^T\boldsymbol{v})$ are -2 and 0, the eigenvalues of a Householder transformation are -1 and 1. (b) The eigenvalues of a plane rotation are $\lambda = c \pm is$.

4.33. For any λ, Let $\boldsymbol{B}_\lambda$ denote the matrix $\boldsymbol{A}-\lambda\boldsymbol{I}$. We first show that $\text{rank}(\boldsymbol{B}_\lambda) \geq n-1$ for any λ, i.e., that the number of linearly independent columns of $\boldsymbol{B}_\lambda$ is at least $n-1$. Since $\boldsymbol{A}$ is symmetric, tridiagonal, and has no zero subdiagonal entries, $\boldsymbol{B}_\lambda$ has these same properties. Denote column k of $\boldsymbol{B}_\lambda$ by $\{\boldsymbol{B}_\lambda\}_k$. Because of the structure of this matrix, the interior columns (columns k with $2 \leq k \leq n-1$) can be written in the form $\{\boldsymbol{B}_\lambda\}_k = \beta_{k-1}\boldsymbol{e}_{k-1} + \alpha_k\boldsymbol{e}_k + \beta_k\boldsymbol{e}_{k+1}$ and the first and last columns can be written as $\{\boldsymbol{B}_\lambda\}_1 = \alpha_1\boldsymbol{e}_1+\beta_1\boldsymbol{e}_2$, $\{\boldsymbol{B}_\lambda\}_n = \beta_{n-1}\boldsymbol{e}_{n-1}+\alpha_n\boldsymbol{e}_n$ where all $\beta_i \neq 0$. We thus see that for any linear combination with nonzero coefficients of the first $n-1$ columns of $\boldsymbol{B}_\lambda$, the only column with a component in the $\boldsymbol{e}_n$ direction is column $n-1$. Hence, this linear combination can never be zero, which shows that the first $n-1$ columns form a linearly independent set. We have thus shown that there is only one linearly independent eigenvector corresponding to any eigenvalue, which, since a symmetric matrix has a full set of eigenvectors, implies that each eigenvalue must be of multiplicity 1, i.e., the eigenvalues of $\boldsymbol{A}$ are all distinct.

4.34. In the QR factorization of $\boldsymbol{A}$, the bottom row of $\boldsymbol{R}$ will be all zeros. Thus, when $\boldsymbol{RQ}$ is formed, it will also have a row of all zeros as the bottom row, which therefore yields an exact eigenvalue after one step, namely the eigenvalue 0.

4.35. The Lanczos recurrence is given by

$$\beta_j\boldsymbol{q}_{j+1} = \boldsymbol{A}\boldsymbol{q}_j - \beta_{j-1}\boldsymbol{q}_{j-1} - \alpha_j\boldsymbol{q}_j.$$

Under the assumption that $\boldsymbol{q}_i^H\boldsymbol{q}_j = 0$ for $i,j \leq k$ and $i \neq j$, we must show that $(\boldsymbol{A}\boldsymbol{q}_k)^H\boldsymbol{q}_j = 0$ for all $j \leq k-2$. Using the fact that $\boldsymbol{A}$ is Hermitian, we have

$$\begin{aligned}(\boldsymbol{A}\boldsymbol{q}_k)^H\boldsymbol{q}_j &= \boldsymbol{q}_k^H(\boldsymbol{A}\boldsymbol{q}_j) = \boldsymbol{q}_k^H(\beta_j\boldsymbol{q}_{j+1} + \beta_{j-1}\boldsymbol{q}_{j-1} + \alpha_j\boldsymbol{q}_j)\\ &= \beta_j\boldsymbol{q}_k^H\boldsymbol{q}_{j+1} + \beta_{j-1}\boldsymbol{q}_k^H\boldsymbol{q}_{j-1} + \alpha_j\boldsymbol{q}_k^H\boldsymbol{q}_j = 0.\end{aligned}$$

This tells us that $\boldsymbol{A}\boldsymbol{q}_k$ need only be orthogonalized against $\boldsymbol{q}_k$ and $\boldsymbol{q}_{k-1}$ to form the next Lanczos vector.

4.36. (a) This equation can be rewritten as two equations, $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{u}$ and $\boldsymbol{A}^T\boldsymbol{u} = \lambda\boldsymbol{v}$. Multiplying the first equation by $\boldsymbol{A}^T$ yields $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{A}^T\boldsymbol{u} = \lambda^2\boldsymbol{v}$. Thus, λ^2 is an eigenvalue of $\boldsymbol{A}^T\boldsymbol{A}$, and therefore $|\lambda|$ is a singular value of $\boldsymbol{A}$. The corresponding right singular vector is $\boldsymbol{v}$. By multiplying the second equation by $\boldsymbol{A}$, it can be seen that $\boldsymbol{u}$ is the corresponding left singular vector. (b) No, because this matrix is much larger than the original matrix and requires storing two copies of $\boldsymbol{A}$, so this approach is much more expensive than computing the SVD of $\boldsymbol{A}$ directly.

Computer Problems

4.1.
```
function cp04_01 % conditioning of eigenvalues
A = [1 1000; 0.001 1], [X, Lambda] = eig(A); [Y, Lambda] = eig(A');
Lambda = diag(Lambda), cond_X = cond(X)
cond_lambda1 = norm(Y(:,1))*norm(X(:,1))/abs(Y(:,1)'*X(:,1))
cond_lambda2 = norm(Y(:,2))*norm(X(:,2))/abs(Y(:,2)'*X(:,2))
A2 = [1 1000; 0 1], Lambda = eig(A2)
```

4.2. function cp02_02 % power iteration and deflation

```
A = [2 3 2; 10 3 4; 3 6 1]; x = [0; 0; 1]; tol = eps; delx = 1;
disp('(a) Dominant eigenvalue/vector, computed using power iteration:');
while delx > tol
  y = A*x; y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
end; y = A*x; [xk, k] = max(abs(x)); lambda1 = y(k)/x(k), x
disp('(b) Next eigenvalue, computed using deflation and power iteration:');
e = zeros(size(x)); e(k) = 1; u = A'*e; B = A-x*u'; x = [1; 0; 0]; delx = 1;
while delx > tol
  y = B*x; y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
end; y = B*x; [xk, k] = max(abs(x)); lambda2 = y(k)/x(k)
disp('(c) All eigenvalues/vectors, computed using MATLAB eig function:');
[X, Lambda] = eig(A);
for i = 1:size(X,1)
  X(:,i) = X(:,i)/norm(X(:,i),inf);
end; Lambda = diag(Lambda), X
```

4.3. function cp04_03 % inverse iteration

```
A = [6 2 1; 2 3 1; 1 1 1]; x = rand(size(A,1),1); sigma = 2;
[L, U] = lu(A-sigma*eye(size(A))); tol = eps; delx = 1;
disp('(a) Eigenvalue closest to 2, computed using inverse iteration:');
while delx > tol
  y = U\(L\x); y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
end; y = A*x; [xk, k] = max(abs(x)); lambda = y(k)/x(k), x
disp('(b) All eigenvalues/vectors, computed using MATLAB eig function:');
[X, Lambda] = eig(A);
for i = 1:size(X,1)
  X(:,i) = X(:,i)/norm(X(:,i),inf);
end; Lambda = diag(Lambda), X
```

4.4. function cp04_04 % Rayleigh quotient iteration

```
A = [6 2 1; 2 3 1; 1 1 1]; x = rand(size(A, 1),1); tol = eps; delx = 1;
while delx > tol
  sigma = (x'*A*x)/(x'*x); y = (A-sigma*eye(size(A)))\x;
  y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
end; y = A*x; [xk, k] = max(abs(x)); lambda = y(k)/x(k), x
```

4.5. function cp04_05 % conditioning of eigenvalues

```
A = [9, 4.5, 3; -56, -28, -18; 60, 30, 19]; [X, D] = eig(A);
fprintf('(a) Eigenvalues of original matrix:  %6.3f  %6.3f  %6.3f\n\n',...
  D(1,1), D(2,2), D(3,3))
condx = cond(X); [Y, C] = eig(A'); YHX = Y'*X;
A(3,3) = 18.95; [X2, D2] = eig(A);
fprintf('(b) Eigenvalues of first modified matrix:  %6.3f  %6.3f  %6.3f\n',...
  D2(1,1), D2(2,2), D2(3,3))
fprintf('    Relative change in largest eigenvalue: %6.3f \n\n',...
  abs(max(D2)-max(D))/max(D))
A(3,3) = 19.05; [X3, D3] = eig(A);
fprintf('(c) Eigenvalues of second modified matrix:  %6.3f  %6.3f  %6.3f\n',...
  D3(1,1), D2(2,2), D3(3,3))
fprintf('    Relative change in largest eigenvalue: %6.3f\n\n',...
```

```
  abs(max(D3)-max(D))/max(D))
fprintf('(d) Eigenvalues are sensitive, changing by 40 to 80 percent\n')
fprintf('    Condition number of eigenvector matrix:  %6.3f\n', condx)
fprintf('    Inner products of right and left eigenvectors:  %6.3f  %6.3f  %6.3f\n',...
  YHX(1,1), YHX(2,2), YHX(3,3))
fprintf('    Condition numbers of eigenvalues: %6.3f  %6.3f  %6.3f\n',...
  abs(1/YHX(1,1)), abs(1/YHX(2,2)), abs(1/YHX(3,3)))
fprintf('    Angles between right and left eigenvectors:  %6.3f  %6.3f  %6.3f\n',...
  acos(YHX(1,1))*180/pi, acos(YHX(2,2))*180/pi, acos(YHX(3,3))*180/pi)
```

4.6.
```
function cp04_06 % simple implementation of QR iteration
A = [2 3 2; 10 3 4; 3 6 1], Lambda = QR_iteration(A)
A = [6 2 1; 2 3 1; 1 1 1], Lambda = QR_iteration(A)

function [Lambda] = QR_iteration(A)
n = size(A,1); tol = eps*norm(A);
for k = n:-1:2
  while norm(A(k,1:k-1),inf) > tol
    sigma = A(k,k); [Q, R] = qr(A(1:k,1:k)-sigma*eye(k,k));
    A(1:k,1:k) = R*Q+sigma*eye(k,k);
  end
end; Lambda = diag(A);
```

4.7.
```
function cp04_07(n) % Lanczos iteration
A = rand(n,n); [Q, R] = qr(A); D = diag(1:n); A = Q*D*Q';
q = zeros(n,n+1); u = zeros(n,n+1); alpha = zeros(n+1,1); beta = zeros(n+1,1);
T = zeros(n+1,n+1); x = rand(n,1); q(:,2) = x/norm(x);
figure(1); title('Computer Problem 4.7 -- Lanczos Iteration');
xlabel('Ritz values'); ylabel('iteration number'); axis([0 n 0 n]); hold on;
for k = 2:n+1
  u(:,k) = A*q(:,k); alpha(k) = q(:,k)'*u(:,k);
  u(:,k) = u(:,k)-beta(k-1)*q(:,k-1)-alpha(k)*q(:,k); beta(k) = norm(u(:,k));
  if beta(k) == 0, break; end
  T(k-1,k-1) = alpha(k); T(k,k-1) = beta(k); T(k-1,k) = beta(k);
  plot(eig(T(1:k-1,1:k-1)),(k-1)*ones(k-1,1),'.'); pause(0.5);
  q(:,k+1) = u(:,k)/beta(k);
end
```

4.8.
```
function cp04_08 % compute roots of polynomial using companion matrix
p = [24; -40; 35; -13; 1]; n = length(p); C = diag(ones(n-2,1),-1);
disp('Companion matrix:'); C(:,n-1) = -p(1:n-1)
disp('Eigenvalues of companion matrix:'); Lambda = eig(C)
disp('Output of MATLAB function roots:'); Roots = roots(flipud(p))
```

4.9.
```
function cp04_09 % eigenvalues of Hilbert matrices
disp(' n    smallest eigenvalue     largest eigenvalue       ratio'); disp(' ');
for n = 1:20
  H = hilb(n); Lambda = eig(H);
  fprintf('%2g  %21.13e   %21.13e    %10.4e\n', n, min(Lambda), max(Lambda),...
    max(Lambda)/min(Lambda));
end
```

4.10.
```
function cp04_10 % singular values of special triangular matrix
disp(' n    min singular value      max singular value       ratio'); disp(' ');
for n = 1:20
  A = eye(n,n);
  for i = 1:n-1
    A = A+diag(-1*ones(n-i,1),i);
  end; S = svd(A);
  fprintf('%2g  %21.13e   %21.13e    %10.4e\n', n, S(n), S(1), S(1)/S(n));
end
```

4.11.
```
function cp04_11 % eigenvalues of special symmetric tridiagonal matrix
disp(' n     largest eigenvalue      2nd largest eigenvalue'); disp(' ');
for k = 1:12
  A = diag([k:-1:0 1:k])+diag(ones(2*k,1),1)+diag(ones(2*k,1),-1);
  Lambda = eig(A); n = 2*k+1;
  fprintf('%2g  %23.15e   %23.15e\n', n, Lambda(n), Lambda(n-1));
end
```

4.12.
```
function cp04_12 % Markov chain
A = [0.8 0.2 0.1; 0.1 0.7 0.3; 0.1 0.1 0.6]; x = [1; 0; 0];
disp('(a) Probability distribution vector after three steps:');
for k = 1:3
  x = A*x;
end; x
disp('(b) Long-term value of probability distribution vector:');
tol = eps; del = 1;
while del > tol
  y = A*x; del = norm(abs(y)-abs(x),inf); x = y;
end; x
disp('(c) Long-term value does not depend on starting vector.');
disp('(d) Limiting value of matrix A^k:');
tol = eps; del = 1; Ak = eye(size(A));
while del > tol
  Bk = A*Ak; del = norm(abs(Bk)-abs(Ak),inf); Ak = Bk;
end; Ak, rank_Ak = rank(Ak)
disp('(e) Eigenvalues and eigenvectors of A:');
[X, Lambda] = eig(A); Lambda = diag(Lambda), X = (Ak(1,1)/X(1,1))*X
disp('(f) Yes, 1 must always be an eigenvalue because probabilities, i.e.,');
disp('     columns of A, always sum to 1.');
disp('(g) Stationary vector is eigenvector corresponding to eigenvalue 1.');
disp('(h) Repeated multiplication by A, as in part (b).');
disp('(i) Cyclic behavior can occur if an eigenvalue has nonzero');
disp('    imaginary part or more than one eigenvalue has modulus 1.');
disp('(j) Stationary vector is unique if eigenvalue 1 is simple.  Distinct');
disp('    stationary vectors are possible if eigenvalue 1 has geometric');
disp('    multiplicity greater than 1.');
disp('(k) Power iteration for computing eigenvalues and eigenvectors.');
```

4.13.
```
function cp04_13 % generalized eigenvalue problem for spring-mass system
K = [2 -1 0; -1 2 -1; 0 -1 1]; M = diag([2 3 4]); A = M\K; B = A; tol = eps;
for i = 1:size(A,1)
```

```
  x = rand(size(B,1),1); delx = 1;
  while delx > tol
    y = B*x; y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
  end
  y = B*x; [xk, k] = max(abs(x)); lambda = y(k)/x(k), omega = sqrt(lambda)
  x = x/x(k), e = zeros(size(B,1),1); e(k) = 1; u = B'*e; B = B-x*u';
end; [X, Lambda] = eig(K,M); Lambda = diag(Lambda), Omega = sqrt(Lambda), X
```

4.14.
```
function cp04_14 % compute matrix exponential
A1 = [2 -1; -1 2]; A2 = [-49 24; -64 31];
disp('Method (a), infinite series:')
EA1 = matexp_series(A1), EA2 = matexp_series(A2)
disp('Method (b), eigenvalue/eigenvector decomposition:')
EA1 = matexp_ev(A1), EA2 = matexp_ev(A2)
disp('MATLAB expm function:'); EA1 = expm1(A1), EA2 = expm1(A2)

function [EA] = matexp_series(A)
EA = eye(size(A)); Ak = eye(size(A)); kf = 1; k = 0; delA = 1; tol = eps;
while norm(delA) > tol
  k = k+1; Ak = A*Ak; kf = k*kf; delA = Ak/kf; EA = EA+delA;
end

function [EA] = matexp_ev(A)
[U, Lambda] = eig(A); EA = U*diag(exp(diag(Lambda)))/U;
```

4.15.
```
function cp04_15(n) % compute coefficients of characteristic polynomial
A = rand(n,n);        % also try A = hilb(n) and A = pascal(n)
disp('(a) Krylov method'); x = rand(n,1); K = zeros(n,n); K(:,1) = x;
for k = 2:n
  K(:,k) = A*K(:,k-1);
end; c = -K\(A*K(:,n)); c = [c; 1], Lambda = sort(roots(flipud(c)))
disp('(b) Leverrier method');
c = zeros(n+1,1); c(n+1) = 1; Bk = A; c(n) = -trace(Bk);
for k = n-2:-1:0
  Bk = A*(Bk+c(k+2)*eye(n,n)); c(k+1) = -trace(Bk)/(n-k);
end; c, Lambda = sort(roots(flipud(c)))
disp('(c) Danilevsky method'); Ak = A;
for k = 1:n-1
  [x, j] = max(abs(Ak(k+1:n,k))); j = j+k;
  if j ~= k+1
    temp = Ak(k+1,:); Ak(k+1,:) = Ak(j,:); Ak(j,:) = temp;
    temp = Ak(:,k+1); Ak(:,k+1) = Ak(:,j); Ak(:,j) = temp;
  end; T = eye(n,n); T(:,k+1) = Ak(:,k); Ak = T\Ak*T;
end; c = [-Ak(:,n); 1], Lambda = sort(roots(flipud(c)))
disp('(d) Hessenberg-Danilevsky method'); [Q, H] = hess(A);
for k = 1:n-1
  T = eye(n,n); T(:,k+1) = H(:,k); H = T\H*T;
end; c = [-H(:,n); 1], Lambda = sort(roots(flipud(c)))
disp('(e) Eigenvalue method')
Lambda = sort(eig(A)); c = fliplr(poly(Lambda))', Lambda
```

4.16.
```
function cp04_16 % compute SVD of 2x2 matrix using plane rotations
A = rand(2,2); [J1, B] = symmetrize(A); [J2, D] = diagonalize(B);
U = J1'*J2; S = diag(diag(D)); V = J2; D = eye(2,2);
if S(1,1) < 0, D(1,1) = -1; end % ensure that singular values
if S(2,2) < 0, D(2,2) = -1; end % are nonnegative
U = U*D; S = D*S;
if S(2,2) > S(1,1)  % sort singular values by decreasing magnitude
  U = U*[0 1; 1 0]; S = [0 1; 1 0]*S*[0 1; 1 0]; V = V*[0 1; 1 0];
end; disp('Computed SVD using rotations:'); U, S, V
disp('Results from MATLAB svd function:'); [U S V] = svd(A)

function [J, B] = symmetrize(A)
if abs(A(1,1)+A(2,2)) < abs(A(2,1)-A(1,2))
  t = (A(1,1)+A(2,2))/(A(2,1)-A(1,2)); s = 1/sqrt(1+t^2); c = s*t;
else
  t = (A(2,1)-A(1,2))/(A(1,1)+A(2,2)); c = 1/sqrt(1+t^2); s = c*t;
end; J = [c s; -s c]; B = J*A;

function [J, B] = diagonalize(A)
alpha = (A(1,1)-A(2,2))/A(1,2); t = (alpha+sqrt(alpha^2+4))/2;
c = 1/sqrt(1+t^2); s = c*t; J = [c s; -s c]; B = J'*A*J;
```

Chapter 5

Nonlinear Equations

Exercises

5.1. (a) The derivative of $f(x) = x^2 - 2$ is $f'(x) = 2x$. With $x_0 = 1$, the value of x_1 using Newton's method is $x_1 = x_0 - f(x_0)/f'(x_0) = 1 + 1/2 = 3/2$. ($b$) With $x_0 = 1, x_1 = 2$, the value of x_2 using the secant method is $x_2 = x_1 - f(x_1)(x_1 - x_0)/(f(x_1) - f(x_0)) = 2 - (2(2-1))/(2-(-1)) = 4/3$.

5.2. (a) $x_{k+1} = x_k - (x_k^3 - 2x_k - 5)/(3x_k^2 - 2)$. ($b$) $x_{k+1} = x_k + (e^{-x_k} - x_k)/(e^{-x_k} + 1)$. ($c$) $x_{k+1} = x_k - (x_k \sin(x_k) - 1)/(x_k \cos(x_k) + \sin(x_k))$.

5.3. (a) Since the derivative of $f(x) = x^2 - y$ is $f'(x) = 2x$, the Newton iteration for solving $f(x) = 0$ is $x_{k+1} = x_k - (x_k^2 - y)/2x_k$. ($b$) To attain m-bit accuracy given an initial guess with 4 bits of accuracy, the number of iterations k needed satisfies $4 \times 2^k = m$. To obtain 24-bit accuracy we then need $k = \lceil \log_2(24/4) \rceil = 3$ iterations and for 53-bit accuracy we need $k = \lceil \log_2(53/4) \rceil = 4$ iterations.

5.4. The derivative of $f(x) = x^{-1} - y$ is $f'(x) = -x^{-2}$. The Newton iteration for solving $f(x) = 0$ is thus $x_{k+1} = x_k - (x_k^{-1} - y)/(-x_k^{-2}) = x_k + x_k^2(x_k^{-1} - y) = 2x_k - x_k^2 y$, which does not contain any divisions.

5.5. (a)

$$\begin{aligned} x_{k+1} &= \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} = \frac{x_k f(x_k) - x_k f(x_{k-1}) + x_{k-1}f(x_k) - x_k f(x_k)}{f(x_k) - f(x_{k-1})} \\ &= \frac{x_k(f(x_k) - f(x_{k-1})) - f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}, \end{aligned}$$

which is the secant method iteration. (b) The formula in part (a) gives the new iterate as a quotient of two differences, each of which is between two quantities

that are nearly equal, and hence may suffer substantial cancellation. The standard formula gives the new iterate as a small perturbation to the previous iterate, and only the perturbation computation suffers from potential cancellation.

5.6. (*a*) Since $g_1'(x) = 1 - 2x$ and $|g_1'(\sqrt{3})| = |1 - 2\sqrt{3}| \approx 2.46 > 1$, the iterative scheme is not convergent. (*b*) Since $g_2'(x) = 1 - 2x/y$ and $|g_1'(\sqrt{3})| = |1 - 2\sqrt{3}/3| \approx 0.155 < 1$, the iterative scheme is locally convergent. (*c*) $f'(x) = 2x$, so the fixed-point iteration function given by Newton's method is $g(x) = x - (x^2 - y)/2x$.

5.7. We seek an x such that $f(x) = \Gamma(x) - 1.5 = 0$. Let $a = 0.5$, $b = 1.0$, $c = 1.5$. Then

$$\begin{aligned} f_a &= f(a) = \Gamma(a) - 1.5 = \sqrt{\pi} - 1.5 \approx 0.2725, \\ f_b &= f(b) = \Gamma(b) - 1.5 = 1 - 1.5 = -0.5, \\ f_c &= f(c) = \Gamma(c) - 1.5 = \sqrt{\pi}/2 - 1.5 \approx -0.6138. \end{aligned}$$

(*a*) To approximate x using one step of quadratic interpolation, we fit a quadratic polynomial $p(t) = \alpha + \beta t + \gamma t^2$ to (a, f_a), (b, f_b), and (c, f_c), and use one of its roots as an approximation to x. To determine the coefficients α, β, and γ of $p(t)$, we solve the system of linear equations

$$\begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} f_a \\ f_b \\ f_c \end{bmatrix}$$

and obtain $\alpha \approx 1.7036$, $\beta \approx -3.5210$, $\gamma \approx 1.3174$. Thus, $p(t) = 0$ has roots $t \approx 0.6344$ and $t \approx 2.0383$, one of which is chosen as an approximation to x. (*b*) Using the formulas in Section 5.5.5 for inverse quadratic interpolation,

$$\begin{aligned} u &= f_b/f_c \approx 0.8146, \\ v &= f_b/f_a \approx -1.8352, \\ w &= f_a/f_c \approx -0.4439, \\ p &= v(w(u - w)(c - b) - (1 - u)(b - a)) \approx 0.6827, \\ q &= (w - 1)(u - 1)(v - 1) \approx -0.7588, \\ x &= b + p/q \approx 0.1003. \end{aligned}$$

(*c*) Using the procedure in Section 5.5.6 for linear fractional interpolation,

$$\begin{aligned} x &= c + h = c + \frac{(a - c)(b - c)(f_a - f_b)f_c}{(a - c)(f_c - f_b)f_a - (b - c)(f_c - f_a)f_b} \\ &\approx c - 0.9386 \approx 0.5614. \end{aligned}$$

Note: $\Gamma(0.59533) \approx 1.5$.

5.8. From the first equation of the system of nonlinear equations in Example 5.2 we have $x_2 = x_1^2 + \gamma$. If we substitute this into the second equation we get

$$0 = -x_1 + (x_1^2 + \gamma)^2 + \gamma = -x_1 + x_1^4 + 2\gamma x_1^2 + \gamma^2 + \gamma.$$

Thus, one can investigate the existence and uniqueness of solutions to the system of nonlinear equations by studying the one dimensional problem $g(x) = x^4 + 2\gamma x^2 - x + \gamma^2 + \gamma = 0$. Since $g(x)$ is a polynomial of degree 4 with real coefficients, the number of real solutions to $g(x) = 0$ can be zero, one (a root with multiplicity 2), two, or four. (a) For $\gamma = 0.5$, $g'(x) = 4x^3 + 2x - 1 = 0$ has solutions $x \approx 0.3855, x \approx -0.1927 \pm 0.7819i$. Therefore, there is only one real critical point $c = 0.3855$ of $g(x)$ and, since $g'(x) < 0$ for $x < c$ and $g'(x) > 0$ for $x > c$, c is a minimum of $g(x)$. Also, $g(c) \approx 0.5352$ so $g(x)$ assumes positive values and, therefore, has no real roots. (b) For $\gamma = 0.25$, Example 5.1 gives $x = 0.5$ as a solution to $g(x) = 0$. Since $g'(0.5) = 4 \times (0.5)^3 + 0.5 - 1 = 0$, this is a multiple root. $x = 0.5$ is also a minimum of $g(x)$, so trying to bracket this solution will fail. After deflating $x = 0.5$ from $g(x)$, if there are other real roots they will solve $\tilde{g}(x) = x^2 + x + 5/4 = 0$. The discriminant of $\tilde{g}(x)$ is $1 - 4(1)(5/4) = -4$, so there are no other real roots. (c) When $\gamma = -0.5$, $g(x) = x^4 - x^2 - x - 1/4$. We have $g(-1) = 3/4 > 0$, $g(0) = -1/4 < 0$, and $g(2) = 39/4 > 0$. Therefore, the intervals $[-1, 0]$ and $[0, 2]$ bracket two solutions to $g(x) = 0$. These are given by $x \approx -0.3660$ and $x \approx 1.3660$. Deflating these two roots from $g(x)$ shows that there are no other real solutions. (d) When $\gamma = -1$, $g(x) = x^4 - 2x^2 - x$. We have $g(-1.5) = 2.0625 > 0$, $g(-0.7) = -0.0399 < 0$, $g(-0.5) = 1/16 > 0$, $g(0.5) = -15/16 < 0$, and $g(2) = 6 > 0$. Therefore, the intervals $[-1.5, -0.7], [-0.7, -0.5], [-0.5, 0.5]$, and $[0.5, 2]$ bracket the four real solutions to $g(x) = 0$.

5.9. The Newton iteration is $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$, where $\boldsymbol{s}_k$ is determined by solving the linear system $\boldsymbol{J}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = -\boldsymbol{f}(\boldsymbol{x}_k)$. We give the latter system for each part of this exercise.

(a)

$$\boldsymbol{J}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = \begin{bmatrix} 2x_1 & 2x_2 \\ 2x_1 & -1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} x_1^2 + x_2^2 - 1 \\ x_1^2 - x_2 \end{bmatrix} = -\boldsymbol{f}(\boldsymbol{x}_k).$$

(b)

$$\boldsymbol{J}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = \begin{bmatrix} 2x_1 + x_2^3 & 3x_1x_2^2 \\ 6x_1x_2 & 3x_1^2 - 3x_2^2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} x_1^2 + x_1x_2^3 - 9 \\ 3x_1^2x_2 - x_2^3 - 4 \end{bmatrix} = -\boldsymbol{f}(\boldsymbol{x}_k).$$

(c)

$$\boldsymbol{J}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = \begin{bmatrix} 1 - 2x_2 & 1 - 2x_1 \\ 2x_1 - 2 & 2x_2 + 2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} x_1 + x_2 - 2x_1x_2 \\ x_1^2 + x_2^2 - 2x_1 + 2x_2 + 1 \end{bmatrix} = -\boldsymbol{f}(\boldsymbol{x}_k).$$

(d)

$$\boldsymbol{J}_f(\boldsymbol{x}_k)\boldsymbol{s}_k = \begin{bmatrix} 3x_1^2 & -2x_2 \\ 1 + 2x_1x_2 & x_1^2 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} x_1^3 - x_2^2 \\ x_1 + x_1^2x_2 - 2 \end{bmatrix} = -\boldsymbol{f}(\boldsymbol{x}_k).$$

(e)

$$\begin{bmatrix} 2\cos(x_1) - 5 & -\sin(x_2) \\ -4\sin(x_1) & 2\cos(x_2) - 5 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} 2\sin(x_1) + \cos(x_2) - 5x_1 \\ 4\cos(x_1) + 2\sin(x_2) - 5x_2 \end{bmatrix}.$$

5.10. The first iteration of Newton's method is $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0$, where $\boldsymbol{s}_0$ is determined by solving the linear system $\boldsymbol{J}_f(\boldsymbol{x}_0)\boldsymbol{s}_0 = -\boldsymbol{f}(\boldsymbol{x}_0)$. For this problem, we therefore have

$$\boldsymbol{J}_f(\boldsymbol{x}_0)\boldsymbol{s}_0 = \begin{bmatrix} 2x_1 & -2x_2 \\ 2x_2 & 2x_1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} x_1^2 \\ 2x_1x_2 - 1 \end{bmatrix} = -\boldsymbol{f}(\boldsymbol{x}_0),$$

or, for $\boldsymbol{x}_0 = [\,0 \quad 1\,]^T$,

$$\begin{bmatrix} 0 & -2 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Solving this system, we obtain $\boldsymbol{s}_0 = [\,0.5 \quad -0.5\,]^T$. Therefore, we have

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

5.11. If $x_k = x^*$, then $f(x_k) = f(x^*) = 0$ and

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} = x^* - 0 = x^*.$$

If $x_{k-1} = x^*$, then $f(x_{k-1}) = f(x^*) = 0$ and

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} = x_k - f(x_k)\frac{x_k - x^*}{f(x_k)} = x_k - (x_k - x^*) = x^*.$$

5.12. (a) The iteration scheme $x_{k+1} = x_k - f(x_k)/d$ will be locally convergent if

$$\begin{array}{lrcl} & & |(x^* - f(x^*)/d)'| & < 1 \\ \Rightarrow & & |1 - f'(x^*)/d| & < 1 \\ \Rightarrow & -1 < & 1 - f'(x^*)/d & < 1 \\ \Rightarrow & 0 < & f'(x^*)/d & < 2. \end{array}$$

(b) In general, the convergence rate will be linear with constant $C = |1 - f'(x^*)/d|$.
(c) To yield quadratic convergence one needs $1 - f'(x^*)/d = 0$, or, equivalently, $d = f'(x^*)$.

5.13. The Jacobian matrix of the system of nonlinear equations is

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ x_2 & x_1 \end{bmatrix}.$$

Newton's method will fail for starting points of the form $\boldsymbol{x} = [\,0 \quad \alpha\,]^T$, α arbitrary, since $\boldsymbol{J}(\boldsymbol{x})$ is singular at such points.

5.14. The details of a proof are given at the end of Section 5.5.2.

5.15. Solving the 3×3 system of linear equations from Section 5.5.6 gives

$$\begin{aligned} w &= \frac{(c-a)(bf_b - af_a) - (b-a)(cf_c - af_a)}{(f_a - f_c)(bf_b - af_a) - (f_a - f_b)(cf_c - af_a)}, \\ v &= \frac{b-a}{bf_b - af_a} - w\frac{f_a - f_b}{bf_b - af_a} \\ &= \frac{c(f_a - f_b) + a(f_b - f_c) + b(f_c - f_a)}{af_a(f_b - f_c) + cf_c(f_a - f_b) + bf_b(f_c - f_a)}, \\ u &= a - vaf_a + wf_a \\ &= \frac{bcf_a(f_c - f_b) + acf_b(f_a - f_c) + abf_c(f_b - f_a)}{af_a(f_b - f_c) + cf_c(f_a - f_b) + bf_b(f_c - f_a)}. \end{aligned}$$

Then

$$\begin{aligned} h &= u - c \\ &= \frac{(a-c)(c-b)(f_a - f_b)f_c}{af_a(f_b - f_c) + cf_c(f_a - f_b) + bf_b(f_c - f_a)} \\ &= \frac{-(a-c)(b-c)(f_a - f_b)f_c}{af_a(f_b - f_c) + cf_c(f_a - f_b) + bf_b(f_c - f_a) + cf_af_b - cf_af_b} \\ &= \frac{(a-c)(b-c)(f_a - f_b)f_c}{(a-c)(f_c - f_b)f_a - (b-c)(f_c - f_a)f_b}. \end{aligned}$$

Computer Problems

5.1.
```
function cp05_01 % zeros of nonlinear function
fs = 'sin(10*x)-x'; f = inline(fs,'x'); x = -2:0.01:2;
y = f(x); plot(x,y); xlabel('x'); ylabel('f(x)'); grid on;
title(['Computer Problem 5.1 -- Zeros of f(x) = ' fs]); clear x;
x(1) = fzero(f,-0.85); x(2) = fzero(f,-0.7); x(3) = fzero(f,-0.3);
x(4) = fzero(f,0); x(5) = fzero(f,0.3); x(6) = fzero(f,0.7);
x(7) = fzero(f,0.85); fprintf('f(x) = %s has 7 zeros:\n', fs); x'
```

5.2. (*a*) $|g_1'(x)| = |2x/3| = |4/3| > 1 \Rightarrow$ divergence.

$|g_2'(x)| = |3/(2\sqrt{3x-2})| = |3/4| < 1 \Rightarrow$ linear convergence with constant 0.75.

$|g_3'(x)| = |2/x^2| = |1/2| < 1 \Rightarrow$ linear convergence with constant 0.5.

$|g_4'(x)| = \left|\frac{-2(x^2-2)}{(2x-3)^2} + \frac{2x}{2x-3}\right| = |-4+4| = 0 \Rightarrow$ quadratic convergence.

```
function cp05_02 % fixed-point iteration
fs = {'(x^2+2)/3' 'sqrt(3*x-2)' '3-2/x' '(x^2-2)/(2*x-3)'};
tol = eps; maxits = 10; x_true = 2; disp('(b)');
for i=1:4
  fprintf('g_%g(x) = %s\n', i, fs{i}); g = inline(fs{i},'x');
  disp(['  k              x                      err  ',...
    '                ratio']);
  k = 0; x = 2.5; err = abs(x-x_true);
```

```
  fprintf('%3d    %20.12e    %20.12e\n', k, x, err);
  while err > tol  &  k < maxits
    k = k+1; x = g(x); err_new = abs(x-x_true);
    ratio = err_new/err; err = err_new;
    fprintf('%3d    %20.12e    %20.12e    %20.12e\n', k, x, err, ratio);
  end; disp(' ');
end
```

5.3. function cp05_03 % bisection, Newton, and secant methods

```
fs = {'x^3-2*x-5' 'exp(-x)-x' 'x*sin(x)-1' 'x^3-3*x^2+3*x-1'};
dfs = {'3*x^2-2' '-exp(-x)-1' 'x*cos(x)+sin(x)' '3*x^2-6*x+3'};
a0 = [1.5 0 0.5 0.5]; b0 = [2.5 1 1.5 1.4]; tol = 1e-10; maxits = 10;
part = {'(a)' '(b)' '(c)' '(d)'};
for i=1:4
  fprintf('\n%s f(x) = %s = 0\n', part{i}, fs{i}); f = inline(fs{i},'x');
  disp(' '); disp('Bisection method:');
  disp(['  k         a               f(a)               b ',...
    '              f(b)']);
  k = 0; a = a0(i); fa = f(a); b = b0(i); fb = f(b);
  fprintf('%3d %17.10e %17.10e %17.10e %17.10e\n', k, a, fa, b, fb);
  while b-a > tol  &  k < maxits
    k = k+1; m = a+(b-a)/2; fm = f(m);
    if sign(fa) == sign(fm), a = m; fa = fm;
    else b = m; fb = fm; end
    fprintf('%3d %17.10e %17.10e %17.10e %17.10e\n', k, a, fa, b, fb);
  end
  disp(' '); disp('Newton''s method:');
  disp('  k         x               f(x)'); delx = 1;
  k = 0; x = a0(i); fx = f(x); df = inline(dfs{i},'x');
  fprintf('%3d %17.10e %17.10e\n', k, x, fx);
  while abs(delx) > tol  &  k < maxits
    k = k+1; d = df(x); delx = -fx/d; x = x+delx; fx = f(x);
    fprintf('%3d %17.10e %17.10e\n', k, x, fx);
  end
  disp(' '); disp('Secant method:');
  disp('  k          x               f(x)'); delx = 1;
  k = 0; x0 = a0(i); fx0 = f(x0);
  fprintf('%3d %17.10e %17.10e\n', k, x0, fx0);
  k = 1; x1 = b0(i); fx1 = f(x1);
  fprintf('%3d %17.10e %17.10e\n', k, x1, fx1);
  while abs(delx) > tol  &  k < maxits
    k = k+1; d = (fx1 - fx0)/(x1-x0); delx = -fx1/d; x0 = x1; fx0 = fx1;
    x1 = x0 - fx0/d; fx1 = f(x1);
    fprintf('%3d %17.10e %17.10e\n', k, x1, fx1);
  end
  disp(' '); disp('MATLAB function fzero:');
  options = optimset('TolX', tol, 'MaxIter', maxits);
  [x, fx, exitflg, output] = fzero(f, [a0(i) b0(i)], options);
  disp('  k         x               f(x)');
  fprintf('%3d %17.10e %17.10e\n', output.iterations, x, fx); disp(' ');
```

```
end
```

5.4. function cp05_04 % inverse and linear fractional interpolation

```
fs = {'x^3-2*x-5' 'exp(-x)-x' 'x*sin(x)-1' 'x^3-3*x^2+3*x-1'};
a0 = [1.5 0 0.5 0.5]; b0 = [2 0.5 1 0.8]; c0 = [2.5 1 1.5 1.4];
tol = 1e-10; maxits = 10;
for i=1:4
  fprintf('\nf(x) = %s = 0\n', fs{i}); f = inline(fs{i},'x');
  disp(' '); disp('Inverse quadratic interpolation:');
  disp('  k         x               f(x)              h');
  k = 0; a = a0(i); fa = f(a); b = b0(i); fb = f(b);
  c = c0(i); fc = f(c); h = 1;
  k = 0; fprintf('%3d %17.10e %17.10e\n', k, a, fa);
  k = 1; fprintf('%3d %17.10e %17.10e\n', k, b, fb);
  k = 2; fprintf('%3d %17.10e %17.10e\n', k, c, fc);
  while abs(h) > tol  &  k < maxits
    k = k+1; u = fb/fc; v = fb/fa; w = fa/fc;
    p = v*(w*(u-w)*(c-b)-(1-u)*(b-a)); q = (w-1)*(u-1)*(v-1);
    h = p/q; c = a; fc = fa; a = b; fa = fb; b = b+h; fb = f(b);
    fprintf('%3d %17.10e %17.10e %17.10e\n', k, b, fb, h);
  end
  disp(' '); disp('Linear fractional interpolation:');
  disp('  k         x               f(x)              h');
  k = 0; a = a0(i); fa = f(a); b = b0(i); fb = f(b);
  c = c0(i); fc = f(c); h = 1;
  k = 0; fprintf('%3d %17.10e %17.10e\n', k, a, fa);
  k = 1; fprintf('%3d %17.10e %17.10e\n', k, b, fb);
  k = 2; fprintf('%3d %17.10e %17.10e\n', k, c, fc);
  while abs(h) > tol  &  k < maxits
    k = k+1; h = (a-c)*(b-c)*(fa-fb)*fc/((a-c)*(fc-fb)*fa-(b-c)*(fc-fa)*fb);
    a = b; fa = fb; b = c; fb = fc; c = c+h; fc = f(c);
    fprintf('%3d %17.10e %17.10e %17.10e\n', k, c, fc, h);
  end
  disp(' '); disp('MATLAB function fzero:');
  options = optimset('TolX', tol, 'MaxIter', maxits);
  [x, fx, exitflg, output] = fzero(f, [a0(i) c0(i)], options);
  disp('  k         x               f(x)');
  fprintf('%3d %17.10e %17.10e\n', output.iterations, x, fx); disp(' ');
end
```

5.5. function cp05_05 % effects of floating-point arithmetic

```
fs = '(((x-0.5)+x)-0.5)+x'; f = inline(fs,'x');
disp('Real solution is 1/3, which is not finitely representable in binary.');
disp('Evaluating f(x) for consecutive floating-point numbers near 1/3:');
x = linspace((1-eps)/3,(1+eps)/3,6); fx = f(x);
disp('            x                       f(x)');
fprintf('%24.16e %24.16e\n', x, fx); disp(' ');
disp('Using fzero, solution obtained does not depend on tolerance used:');
clear x fx; tol = [1.0e-4; 1.0e-8; 1.0e-12; 1.0e-16];
disp('  tolerance   iterations            x                       f(x)');
for k=1:4
```

```
  options = optimset('TolX', tol(k));
  [x, fx, exitflg, output] = fzero(f, [0 1], options);
  iter = output.iterations;
  fprintf('%e    %3d      %24.16e  %24.16e\n', tol(k), iter, x, fx);
end
```

5.6. function cp05_06 % convergence rate of Newton's method

```
fs = {'x^2-1' '(x-1)^4'}; dfs = {'2*x' '4*(x-1)^3'};
x_true = 1; x0 = [1000000 10]; tol = 1e-10; maxits = 25;
for i=1:2
  fprintf('\nf(x) = %s = 0\n', fs{i}); f = inline(fs{i},'x');
  disp('  k         x               f(x)              err              ratio');
  k = 0; x = x0(i); fx = f(x); err = abs(x-x_true);
  df = inline(dfs{i},'x'); delx = 1;
  fprintf('%3d %17.10e %17.10e %17.10e\n', k, x, fx, err);
  while abs(delx) > tol  &  k < maxits
    k = k+1; d = df(x); delx = -fx/d; x = x+delx; fx = f(x);
    err_new = abs(x-x_true); ratio = err_new/err; err = err_new;
    fprintf('%3d %17.10e %17.10e %17.10e %17.10e\n', k, x, fx, err, ratio);
  end
end; disp(' ');
disp('Convergence rate for function (a) is initially linear, with constant 0.5,');
disp('but convergence eventually becomes quadratic near root.');
disp('Convergence rate for function (b) is linear with constant 1 - 1/4 = 0.75');
disp('because root is of multiplicity 4.');
```

5.7. function cp05_07 % failure of Newton's method

```
fs = 'x^5-x^3-4*x'; dfs = '5*x^4-3*x^2-4';
x0 = 1; tol = 1e-10; maxits = 10;
disp('(a) Newton iterates alternate between 1 and -1:');
fprintf('\nf(x) = %s = 0\n', fs); f = inline(fs,'x');
disp('  k         x               f(x)');
k = 0; x = x0; fx = f(x); df = inline(dfs,'x'); delx = 1;
fprintf('%3d %17.10e %17.10e\n', k, x, fx);
while abs(delx) > tol  &  k < maxits
  k = k+1; d = df(x); delx = -fx/d; x = x+delx; fx = f(x);
  fprintf('%3d %17.10e %17.10e\n', k, x, fx);
end; disp(' ');
disp(['(b) Roots of ', fs, ' are:']); roots([1 0 -1 0 -4 0]), disp(' ');
disp('Roots are simple, but symmetry of problems causes Newton''s method to fail.');
fplot(f, [-2 2]); hold on; plot([-1 1], [4 -4], 'x');
```

5.8. function cp05_08 % fixed-point iteration

```
fs = {'acos(-1/(1+exp(-2*x)))' '0.5*log(-1/(1+1/cos(x)))' ...
  'x-(cos(x)+1/(1+exp(-2*x)))/(-sin(x)+2*exp(-2*x)/(1+exp(-2*x))^2)'};
tol = 1e-12; maxits = 10; x_true = fzero('cos(x)+1/(1+exp(-2*x))',3);
for i=1:3
  fprintf('g_%g(x) = %s\n', i, fs{i}); g = inline(fs{i},'x');
  disp(['  k              x                      err  ',...
    '                ratio']);
  k = 0; x = 3; err = abs(x-x_true);
```

```
  fprintf('%3d    %20.12e    %20.12e\n', k, x, err);
  while err > tol  &  k < maxits
    k = k+1; x = g(x); err_new = abs(x-x_true);
    ratio = err_new/err; err = err_new;
    fprintf('%3d    %20.12e    %20.12e    %20.12e\n', k, x, err, ratio);
  end; disp(' ');
end
```

5.9.
```
gs = '1+0.5*sin(x)'; fs = 'x-0.5*sin(x)-1'; dfs = '1-0.5*cos(x)';
g = inline(gs,'x'); f = inline(fs,'x'); df = inline(dfs,'x');
tol = 1e-10; maxits = 10; x0 = 1; disp(' ');
disp('Fixed-point iteration:');
disp('  k         x               f(x)'); delx = 1;
k = 0; x = x0; fprintf('%3d %17.10e %17.10e\n', k, x, f(x));
while abs(delx) > tol  &  k < maxits
  k = k+1; gx = g(x); delx = gx-x; x = gx;
  fprintf('%3d %17.10e %17.10e\n', k, x, f(x));
end; disp(' ');
disp('Newton''s method:');
disp('  k         x               f(x)'); delx = 1;
k = 0; x = x0; fx = f(x);
fprintf('%3d %17.10e %17.10e\n', k, x, fx);
while abs(delx) > tol  &  k < maxits
  k = k+1; d = df(x); delx = -fx/d; x = x+delx; fx = f(x);
  fprintf('%3d %17.10e %17.10e\n', k, x, fx);
end; disp(' ');
disp('MATLAB function fzero:');
options = optimset('TolX',tol,'MaxIter',maxits);
[x, fx, exitflg, output] = fzero(f,x0,options);
disp('  k         x               f(x)');
fprintf('%3d %17.10e %17.10e\n', output.iterations, x, fx); disp(' ');
```

5.10.
```
x = fzero(inline('cot(x)-(x^2-1)/(2*x)','x'), [0.01 2])
```

5.11.
```
x = fzero(inline('tan(x)*tanh(x)+1','x'), [2 3])
```

5.12.
```
t = fzero(inline('1000-log(cosh(t*sqrt(9.8065*0.00341)))/0.00341','t'), [1 30])
```

5.13.
```
h = fzero(inline('h^3-3*h^2+1.6','h'), [0 1])
```

5.14.
```
global a b p R T; disp('pressure   volume(ideal)   volume(van der Waals)');
a = 3.592; b = 0.04267; R = 0.082054; T = 300; P = [1 10 100];
for k=1:3
  p = P(k); v_ideal = R*T/p; v = fzero(@f,v_ideal);
  fprintf('  %3d     %13.6e        %13.6e\n', p, v_ideal, v);
end
```

```
function [fv] = f(v)
global a b p R T; fv = (p+a/v^2)*(v-b)-R*T;
```

5.15.
```
function cp05_15 % compound interest
disp('(a)'); n = fzero(@fa, [10 30])
disp('(b)'); r = fzero(@fb, [0.05 0.1])
disp('(c)'); p = fzero(@fc, [5000 10000])

function [y] = fa(n)
a = 100000; p = 10000; r = 0.06; y = (a*r-p)*(1+r)^n+p;

function [y] = fb(r)
a = 100000; p = 10000; n = 20; y = (a*r-p)*(1+r)^n+p;

function [y] = fc(p)
a = 100000; r = 0.06; n = 20; y = (a*r-p)*(1+r)^n+p;
```

5.16.
```
function cp05_16 % compute nth root using Newton and Muller methods
n = 3; y = 3; tol = 1e-12; maxits = 50;
disp('(a) Newton''s method:'); k = 0; delx = 1; x = -1+i;
while abs(delx) > tol  &  k < maxits
  k = k+1; delx = -(x^n-y)/(n*x^(n-1)); x = x+delx;
end; root = x
disp('(b) Muller''s method:'); k = 3; delx = 1;
x = [-1+i; -0.5+i; -1+1.5i]; fx = x.^n-y;
while abs(fx(k)) > tol  &  k < maxits
  k = k+1; r = roots(polyfit(x(k-3:k-1),fx(k-3:k-1),2)); % fit quadratic
  if abs(r(1)-x(k-1)) < abs(r(2)-x(k-1)), x(k) = r(1);   % choose closer root
  else x(k) = r(2); end;
  fx(k) = x(k)^n-y;
end; root = x(k)
```

5.17.
```
function cp05_17 % solving cubic equation using Vieta's Theorem
global p; p = [1; 0; -2; -5]; x0 = [1; i; -i];     % cubic from CP5.3(a)
if p(1) ~= 0, p = p/p(1);
else error('Leading coefficient of p must be nonzero.'); end
disp('Solution to Vieta system by Newton''s method:');
tol = 1e-15; maxits = 50; k = 0; x = x0; s = ones(size(x));
while norm(s) > tol  &  k < maxits
  k = k+1; s = Df(x)\f(x); x = x-s;
end; x
disp('MATLAB function roots:'); r = roots(p)

function [y] = f(x)         % nonlinear system given by Vieta's Theorem
global p; y = [x(1)+x(2)+x(3)+p(2); x(1)*x(2)+x(2)*x(3)+x(1)*x(3)-p(3); ...
  x(1)*x(2)*x(3)+p(4)];

function [J] = Df(x)
J = [1 1 1; x(2)+x(3) x(1)+x(3) x(1)+x(2); x(2)*x(3) x(1)*x(3) x(1)*x(2)];
```

5.18.
```
function cp05_18 % Newton and Broyden methods for nonlinear system
x0 = [-0.5; 1.4]; x_true = [0; 1]; tol = eps; maxits = 20;
disp('Newton''s method:');
disp('  k        x(1)              x(2)            err_norm');
k = 0; x = x0; s = ones(size(x)); err = norm(x-x_true);
fprintf('%3d %17.10e %17.10e %17.10e\n', k, x(1), x(2), err);
while norm(s) > tol  &  k < maxits
  k = k+1; s = -(Df(x)\f(x)); x = x+s; err = norm(x-x_true);
  fprintf('%3d %17.10e %17.10e %17.10e\n', k, x(1), x(2), err);
end; disp(' ');
disp('Broyden''s method:');
disp('  k        x(1)              x(2)            err_norm');
k = 0; x = x0; fx = f(x); B = Df(x); s = ones(size(x)); err = norm(x-x_true);
fprintf('%3d %17.10e %17.10e %17.10e\n', k, x(1), x(2), err);
while norm(s) > tol  &  k < maxits
  k = k+1; s = -(B\fx); x = x+s; y = fx; fx = f(x); y = fx-y;
  B = B+((y-B*s)*(s'))/(s'*s); err = norm(x-x_true);
  fprintf('%3d %17.10e %17.10e %17.10e\n', k, x(1), x(2), err);
end;

function [y] = f(x)
y = [(x(1)+3)*(x(2)^3-7)+18; sin(x(2)*exp(x(1))-1)];

function [J] = Df(x)
J = [x(2)^3-7, 3*x(2)^2*(x(1)+3); x(2)*exp(x(1))*cos(x(2)*exp(x(1))-1), ...
  exp(x(1))*cos(x(2)*exp(x(1))-1)];
```

5.19.
```
function cp05_19 % nonlinear system modeling bioremediation
global gamma delta; gamma = 5; delta = 1; tol = eps; maxits = 10;
% Exact solutions are given by y = 1/(gamma-1), x = (delta-y)*(1+y)/y,
% [0; -1], and [0; delta].
x0 = [1 0.5; 0.3 0.5];                          % starting guesses
for i = 1:2
  k = 0; x = x0(:,i); s = ones(size(x));
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -(Df(x)\f(x)); x = x+s;
  end; x
end

function [y] = f(x); global gamma delta;
y = [gamma*x(1)*x(2)-x(1)*(1+x(2)); -x(1)*x(2)+(delta-x(2))*(1+x(2))];

function [J] = Df(x); global gamma delta;
J = [(gamma-1)*x(2)-1, (gamma-1)*x(1); -x(2), -x(1)+delta-1-2*x(2)];
```

5.20.
```
function cp05_20 % ground and excited states in spherical well
global s; s = 3.5;
disp('(a)'); x = fsolve(@fa, [1; 2], optimset('Display','off'))
disp('(b)'); x = fsolve(@fb, [2; 1], optimset('Display','off'))
```

```
function [y] = fa(x); global s;
y = [x(1)/tan(x(1))+x(2); x(1)^2+x(2)^2-s^2];

function [y] = fb(x); global s;
y = [1/(x(1)*tan(x(1)))-1/(x(1)^2)-1/x(2)-1/(x(2)^2); x(1)^2+x(2)^2-s^2];
```

5.21.
```
function cp05_21 % Lorenz model of buoyant convection
global sigma r b; sigma = 10; r = 28; b = 8/3; tol = 1e-14; maxits = 10;
% Exact solutions are [0; 0; 0], [sqrt(b*(r-1)); sqrt(b*(r-1)); r-1], and
% [-sqrt(b*(r-1)); -sqrt(b*(r-1)); r-1].
x0 = [-0.5 9 -9; 1 12 -12; -1 30 30];                  % starting guesses
for i = 1:3
  k = 0; x = x0(:,i); s = ones(size(x));
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -(Df(x)\f(x)); x = x+s;
  end; x
end

function [y] = f(x); global sigma r b;
y = [sigma*(x(2)-x(1)); r*x(1)-x(2)-x(1)*x(3); x(1)*x(2)-b*x(3)];

function [J] = Df(x); global sigma r b;
J = [-sigma sigma 0; r-x(3) -1 -x(1); x(2) x(1) -b];
```

5.22.
```
function cp05_22 % fixed-point and Newton methods for nonliner system
tol = eps; maxits = 20; x_true = [0; 1/3; 0]; x0 = [0.5; 0.5; 0.5];
disp('(a) Fixed-point iteration:');
disp('  k         err              ratio');
k = 0; x = x0; gx = g(x); err = norm(x-x_true);
fprintf('%3d %17.10e\n', k, err);
while err > tol  &  k < maxits
  k = k+1; x = g(x);
  err_new = norm(x-x_true); ratio = err_new/err; err = err_new;
  fprintf('%3d %17.10e %17.10e\n', k, err, ratio);
end; x
disp('(b) Spectral radius:'); rho = max(abs(eig(Dg(x_true))))
disp('(c) Newton''s method:');
disp('  k         err              ratio');
k = 0; x = x0; s = ones(size(x)); err = norm(x-x_true);
fprintf('%3d %17.10e\n', k, err);
while err > tol  &  k < maxits
  k = k+1; s = -(Df(x)\f(x)); x = x+s;
  err_new = norm(x-x_true); ratio = err_new/err; err = err_new;
  fprintf('%3d %17.10e %17.10e\n', k, err, ratio);
end; x

function [y] = f(x)
y = g(x)-[x(1); x(2); x(3)];

function [J] = Df(x)
J = Dg(x)-eye(3);
```

```
function [y] = g(x)                                % fixed-point function
y = [-cos(x(1))/81+(x(2)*x(2))/9+sin(x(3))/3; sin(x(1))/3+cos(x(3))/3; ...
  -cos(x(1))/9+x(2)/3+sin(x(3))/6];

function [J] = Dg(x)
J = [sin(x(1))/81, 2*x(2)/9, cos(x(3))/3; cos(x(1))/3, 0, -sin(x(3))/3; ...
  sin(x(1))/9, 1/3, cos(x(3))/6];
```

5.23.
```
function cp05_23 % Newton's method for nonlinear system
tol = eps; maxits = 10; x0 = [1; 1; 1]; disp('Newton''s method:');
k = 0; x = x0; s = ones(size(x));
while norm(s) > tol  &  k < maxits
  k = k+1; s = -(Df(x)\f(x)); x = x+s;
end; x, iterations = k, disp('MATLAB function fsolve:');
[x, fx, exitflg, output] = fsolve(@f, x0, optimset('TolX', tol, 'TolFun', ...
  tol, 'Display','off')); x, iterations = output.iterations

function [y] = f(x)
y = [16*x(1)^4+16*x(2)^4+x(3)^4-16; x(1)^2+x(2)^2+x(3)^2-3; x(1)^3-x(2)];

function [J] = Df(x)
J = [64*x(1)^3, 64*x(2)^3, 4*x(3)^3; 2*x(1), 2*x(2), 2*x(3); 3*x(1)^2, -1, 0];
```

5.24.
```
function cp05_24 % nodes and weights for Gaussian quadrature rule
% Exact solutions are x = [-1/sqrt(3); 1/sqrt(3)], w = [1; 1] and
% x = [1/sqrt(3); -1/sqrt(3)], w = [1; 1].
x = fsolve(@f, [1; -1; 2; 2], optimset('Display','off')); x
x = fsolve(@f, [-1; 1; 2; 2], optimset('Display','off')); x

function [y] = f(x); w = [x(3); x(4)];
y = [w(1)+w(2)-2; w(1)*x(1)+w(2)*x(2); w(1)*x(1)^2+w(2)*x(2)^2-2/3; ...
  w(1)*x(1)^3+w(2)*x(2)^3];
```

5.25.
```
function cp05_25 % system of nonlinear equations
x = fsolve(@f, [1; 1; 1], optimset('Display','off')); x
x = fsolve(@f, [1; -1; 1], optimset('Display','off')); x
x = fsolve(@f, [-1; 1; 1], optimset('Display','off')); x
x = fsolve(@f, [-1; -1; 1], optimset('Display','off')); x

function [y] = f(x)
y = [sin(x(1))+x(2)^2+log(x(3))-3; 3*x(1)+2^x(2)-x(3)^3; x(1)^2+x(2)^2+x(3)^2-6];
```

5.26.
```
function cp05_26 % propane combustion model
R = 4.056734; x1 = 3; x2 = 0; x3 = 2*R; x4 = 0; x5 = 0; x7 = 0; x6 = 8-x7;
x9 = 0; x10 = 15+3*R-x7; x8 = R+4-x7-x9-2*x10;   % x7 and x9 are arbitrary
x0 = [x1; x2; x3; x4; x5; x6; x7; x8; x9; x10];
x = fsolve(@f, x0, optimset('Display','off')), norm_f = norm(f(x))

function [y] = f(x); R = 4.056734; S = sum(x);
y = [x(1)+x(4)-3; 2*x(1)+x(2)+x(4)+x(7)+x(8)+x(9)+2*x(10)-R-10; ...
```

```
  2*x(2)+2*x(5)+x(6)+x(7)-8; 2*x(3)+x(5)-4*R; x(1)*x(5)-0.193*x(2)*x(4); ...
  x(6)*sqrt(abs(x(2)))-0.002597*sqrt(abs(x(2)*x(4)*S)); ...
  x(7)*sqrt(abs(x(4)))-0.003448*sqrt(abs(x(1)*x(4)*S)); ...
  x(4)*x(8)-0.00001799*x(2)*S; x(4)*x(9)-0.0002155*x(1)*sqrt(abs(x(3)*S)); ...
  x(4)^2*(x(10)-0.00003846*S)];
```

5.27.
```
function cp05_27 % several difficult nonlinear systems
tol = eps; maxits = 100; part = {'(a)' '(b)' '(c)' '(d)' '(e)'};
funs = [@fa @fb @fc @fd @fe]; derivs = [@Dfa @Dfb @Dfc @Dfd @Dfe];
x0 = {[15; -2] [(1+sqrt(3))/2; (1-sqrt(3))/2; sqrt(3)] [1; 2; 1; 1] [1.8; 0] [0; 1]};
for i = 1:5
  disp(part{i}); disp('Newton''s method:');
  disp('  k       norm(f)          norm(s)');
  k = 0; x = x0{i}; fx = feval(funs(i),x); s = ones(size(x));
  fprintf('%3d %17.10e\n', k, norm(fx));
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -(feval(derivs(i),x)\fx); x = x+s; fx = feval(funs(i),x);
    fprintf('%3d %17.10e %17.10e\n', k, norm(fx), norm(s));
  end; x
  disp('MATLAB function fsolve:');
  [x, fx, exitflg, output] = fsolve(funs(i), x0{i}, optimset('TolX', tol, ...
    'TolFun', tol, 'Display','off'));
  x, norm_f = norm(feval(funs(i),x)), iterations = output.iterations
end

function [fx] = fa(x)
fx = [x(1)+x(2)*(x(2)*(5-x(2))-2)-13; x(1)+x(2)*(x(2)*(1+x(2))-14)-29];

function [J] = Dfa(x)
J = [1, -3*x(2)^2+10*x(2)-2; 1, 3*x(2)^2+2*x(2)-14];

function [fx] = fb(x)
fx = [x(1)^2+x(2)^2+x(3)^2-5; x(1)+x(2)-1; x(1)+x(3)-3];

function [J] = Dfb(x)
J = [2*x(1), 2*x(2) 2*x(3); 1, 1, 0; 1, 0, 1];

function [fx] = fc(x)
fx = [x(1)+10*x(2); sqrt(5)*(x(3)-x(4)); (x(2)-x(3))^2; sqrt(10)*(x(1)-x(4))^2];

function [J] = Dfc(x)
J = [1, 10, 0, 0; 0, 0, sqrt(5), -sqrt(5); 0, 2*(x(2)-x(3)), 2*(x(3)-x(2)), 0;...
  2*sqrt(10)*(x(1)-x(4)), 0, 0, 2*sqrt(10)*(x(4)-x(1))];

function [fx] = fd(x)
fx = [x(1); 10*x(1)/(x(1)+0.1)+2*x(2)^2];

function [J] = Dfd(x)
J = [1, 0; 10*x(1)/(x(1)+0.1)^2+10/(x(1)+0.1), 4*x(2)];
```

```
function [fx] = fe(x)
fx = [10000*x(1)*x(2)-1; exp(-x(1))+exp(-x(2))-1.0001];

function [J] = Dfe(x)
J = [10000*x(2), 10000*x(1); -exp(-x(1)), -exp(-x(2))];
```

5.28.
```
function cp05_28(A) % compute matrix inverse using Newton's method
tol = 1e-14; maxits = 50; I = eye(size(A));
disp('Newton''s method:');
k = 0; X = A'/(norm(A,1)*norm(A,inf)); R = I-A*X; S = ones(size(A));
while norm(S,inf) > tol  &  k < maxits
  k = k+1; S = X*R; X = X+S; R = I-A*X;
end; X
disp('MATLAB function inv:'); A_inv = inv(A)
```

5.29.
```
function cp05_29(A) % compute eigenvalues using Newton's method
tol = eps; maxits = 50; n = size(A,1); I = eye(n); disp('Newton''s method:');
k = 0; x = rand(n,1); x = x/norm(x); lambda = x'*A*x; s = ones(n+1,1);
while norm(s) > tol  &  k < maxits
  k = k+1; s = -([A-lambda*I, -x; 2*x', 0]\[A*x-lambda*x; x'*x-1]);
  x = x+s(1:n); lambda = lambda+s(n+1);
end; lambda, x
disp('Power iteration:'); k = 0; x = rand(n,1); delx = 1;
while delx > tol  &  k < maxits
  k = k+1; y = A*x; y = y/norm(y,inf); delx = norm(abs(y)-abs(x),inf); x = y;
end; y = A*x; [xk, k] = max(abs(x)); lambda = y(k)/x(k), x = x/norm(x)
disp('MATLAB function eig:'); [X, Lambda] = eig(A); Lambda = diag(Lambda), X
```

Chapter 6

Optimization

Exercises

6.1. (a) Not coercive: $\lim_{x\to-\infty} f(x,0) = -\infty$. (b) Coercive. (c) Not coercive: $\lim_{\|x\|\to\infty, x=y} f(x,y) = 0$. (d) Coercive: $\lim_{\|x\|\to\infty} x^4 + y^4 > xy$.

6.2. (a) Strictly convex. (b) Nonconvex. (c) Strictly convex. (d) Convex.

6.3. (a) $f'(x) = 2x$ and $f''(x) = 2$, so $f'(0) = 0$ and $f''(0) = 2 > 0$, and hence 0 is a minimum. (b) $f'(x) = 3x^2$ and $f''(x) = 6x$, so $f'(0) = 0$ and $f''(0) = 0$, and hence 0 is a critical point, but the second-order optimality condition is inconclusive. In fact, 0 is easily seen to be an inflection point. (c) $f'(x) = 4x^3$ and $f''(x) = 12x^2$, so $f'(0) = 0$ and $f''(0) = 0$, and hence 0 a critical point, but the second-order optimality condition is inconclusive. In fact, 0 is easily seen to be a minimum. (d) $f'(x) = -4x^3$ and $f''(x) = -12x^2$, so $f'(0) = 0$ and $f''(0) = 0$, and hence 0 a critical point, but the second-order optimality condition is inconclusive. In fact, 0 is easily seen to be a maximum.

6.4. (a) Roots of $f'(x) = 3x^2 + 12x - 15 = 0$ give critical points $x_1 = 1$, $x_2 = -5$. $f''(x) = 6x + 12$, so $f''(x_1) = 18$, and hence x_1 is a local minimum, and $f''(x_2) = -18$, so x_2 is a local maximum. There is no global minimum or maximum. (b) Roots of $f'(x) = 6x^2 - 50x - 12 = 0$ give critical points $x_1 = (25 + \sqrt{697})/6$, $x_2 = (25 - \sqrt{697})/6$. $f''(x) = 12x - 50$, so $f''(x_1) = 2\sqrt{697}$, and hence x_1 is a local minimum, and $f''(x_2) = -2\sqrt{697}$, so x_2 is a local maximum. There is no global minimum or maximum. (c) Roots of $f'(x) = 9x^2 + 14x - 15 = 0$ give critical points $x_1 = (-7 + \sqrt{184})/9$, $x_2 = (-7 - \sqrt{184})/9$. $f''(x) = 18x + 14$, so $f''(x_1) = 2\sqrt{184}$, and hence x_1 is a local minimum, and $f''(x_2) = -2\sqrt{184}$, so x_2 is a local maximum. There is no global minimum or maximum. (d) Roots of $f'(x) = 2xe^x + x^2e^x = e^x(x^2 + 2x) = 0$ give critical points $x_1 = 0$, $x_2 = -2$.

$f''(x) = e^x(x^2+2x) + e^x(2x+2) = e^x(x^2+4x+2)$, so $f''(x_1) = 2$, and hence x_1 is a local minimum, and $f''(x_2) = -2e^{-2}$, so x_2 is a local maximum. There is no global maximum, but $x_1 = 0$ is a global minimum.

6.5. (a)

$$\nabla f(x,y) = \begin{bmatrix} 2x - 4y \\ 2y - 4x \end{bmatrix}, \quad \boldsymbol{H}_f(x,y) = \begin{bmatrix} 2 & -4 \\ -4 & 2 \end{bmatrix}.$$

Critical point $(0,0)$ is a saddle point because $\boldsymbol{H}_f(0,0)$ is indefinite. There is no global minimum or maximum.

(b)

$$\nabla f(x,y) = \begin{bmatrix} 4x^3 - 4y \\ -4x + 4y^3 \end{bmatrix}, \quad \boldsymbol{H}_f(x,y) = \begin{bmatrix} 12x^2 & -4 \\ -4 & 12y^2 \end{bmatrix}.$$

Critical point $(0,0)$ is a saddle point because $\boldsymbol{H}_f(0,0)$ is indefinite. Critical point $(1,1)$ is a local minimum because $\boldsymbol{H}_f(1,1)$ is positive definite. Critical point $(-1,-1)$ is a local minimum because $\boldsymbol{H}_f(-1,-1)$ is positive definite. Both local minima are global minima, but there is no global maximum.

(c)

$$\nabla f(x,y) = \begin{bmatrix} 6x^2 - 12xy + 6y^2 - 6x + 6y \\ -6x^2 + 12xy + 6x \end{bmatrix},$$

$$\boldsymbol{H}_f(x,y) = \begin{bmatrix} 12x - 12y - 6 & -12x + 12y + 6 \\ -12x + 12y + 6 & 12x \end{bmatrix}.$$

Critical point $(0,0)$ is a saddle point because $\boldsymbol{H}_f(0,0)$ is indefinite. Critical point $(1,0)$ is a local minimum because $\boldsymbol{H}_f(1,0)$ is positive definite. Critical point $(0,-1)$ is a saddle point because $\boldsymbol{H}_f(0,-1)$ is indefinite. There is no global minimum or maximum.

(d)

$$\nabla f(x,y) = \begin{bmatrix} 4(x-y)^3 + 2x - 2 \\ -4(x-y)^3 - 2y + 2 \end{bmatrix},$$

$$\boldsymbol{H}_f(x,y) = \begin{bmatrix} 12(x-y)^2 + 2 & -12(x-y)^2 \\ -12(x-y)^2 & 12(x-y)^2 - 2 \end{bmatrix}.$$

Critical point $(1,1)$ is saddle point because $\boldsymbol{H}_f(1,1)$ is indefinite. There is no global minimum or maximum.

6.6. (a)

$$\nabla f(x,y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}, \quad \boldsymbol{J}_g(x,y) = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \nabla \mathcal{L}(x,y,\lambda) = \begin{bmatrix} 2x + \lambda \\ 2y + \lambda \\ x + y - 1 \end{bmatrix},$$

$$\boldsymbol{B}(x,y,\lambda) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \quad \boldsymbol{z}(x,y) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Critical point $(0.5, 0.5, -1)$ is a constrained minimum because $\boldsymbol{z}^T \boldsymbol{B} \boldsymbol{z} = 4 > 0$.

(*b*)

$$\nabla f(x,y) = \begin{bmatrix} 3x^2 \\ 3y^2 \end{bmatrix}, \quad \boldsymbol{J}_g(x,y) = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad \nabla \mathcal{L}(x,y,\lambda) = \begin{bmatrix} 3x^2+\lambda \\ 3y^2+\lambda \\ x+y-1 \end{bmatrix},$$

$$\boldsymbol{B}(x,y,\lambda) = \begin{bmatrix} 6x & 0 \\ 0 & 6y \end{bmatrix}, \quad \boldsymbol{z}(x,y) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

Critical point $(0.5, 0.5, -0.75)$ is a constrained minimum because $\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z} = 4 > 0$.

(*c*)

$$\nabla f(x,y) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{J}_g(x,y) = \begin{bmatrix} 2x & 2y \end{bmatrix}, \quad \nabla \mathcal{L}(x,y,\lambda) = \begin{bmatrix} 2+2\lambda x \\ 1+2\lambda y \\ x^2+y^2-1 \end{bmatrix},$$

$$\boldsymbol{B}(x,y,\lambda) = \begin{bmatrix} 2\lambda & 0 \\ 0 & 2\lambda \end{bmatrix}, \quad \boldsymbol{z}(x,y) = \begin{bmatrix} y \\ -x \end{bmatrix}.$$

Critical point $(-0.894, -0.447, 1.12)$ is a constrained minimum because $\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z} = 2.24 > 0$. Critical point $(0.894, 0.447, -1.12)$ is a constrained maximum because $\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z} = -2.24 < 0$.

(*d*)

$$\nabla f(x,y) = \begin{bmatrix} 2x \\ 2y \end{bmatrix}, \quad \boldsymbol{J}_g(x,y) = \begin{bmatrix} y^2 & 2xy \end{bmatrix}, \quad \nabla \mathcal{L}(x,y,\lambda) = \begin{bmatrix} 2x+\lambda y^2 \\ 2y+2\lambda xy \\ xy^2-1 \end{bmatrix},$$

$$\boldsymbol{B}(x,y,\lambda) = \begin{bmatrix} 2 & 2\lambda y \\ 2\lambda y & 2+2\lambda x \end{bmatrix}, \quad \boldsymbol{z}(x,y) = \begin{bmatrix} 2x \\ -y \end{bmatrix}.$$

Critical point $(0.794, 1.12, -1.26)$ is a constrained minimum because $\boldsymbol{z}^T\boldsymbol{B}\boldsymbol{z} = 15.1 > 0$.

6.7.

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1 - 2 \\ 2x_2 \\ -2x_3 + 4 \end{bmatrix}, \quad \boldsymbol{J}_g(\boldsymbol{x}) = \begin{bmatrix} 1 & -1 & 2 \end{bmatrix}, \quad \nabla \mathcal{L}(\boldsymbol{x},\lambda) = \begin{bmatrix} 2x_1 - 2 + \lambda \\ 2x_2 - \lambda \\ -2x_3 + 4 + 2\lambda \\ x_1 - x_2 + 2x_3 - 2 \end{bmatrix},$$

$$\boldsymbol{B}(\boldsymbol{x},\lambda) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -2 \end{bmatrix}, \quad \boldsymbol{Z}(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z} = \begin{bmatrix} 4 & 4 \\ 4 & 6 \end{bmatrix}.$$

Critical point $(2.5, -1.5, -1, -3)$ is a constrained minimum because $\boldsymbol{Z}^T\boldsymbol{B}\boldsymbol{Z}$ is positive definite.

6.8. (a)

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1^3 - 2x_1x_2 + x_1 - 1 \\ -x_1^2 + x_2 \end{bmatrix}, \quad \boldsymbol{H}_f(\boldsymbol{x}) = \begin{bmatrix} 6x_1 - 2x_2 + 1 & -2x_1 \\ -2x_1 & 1 \end{bmatrix}.$$

Critical point $\boldsymbol{x}^* = [\,1 \quad 1\,]$ is a local minimum because $\boldsymbol{H}_f(\boldsymbol{x}^*)$ is positive definite. (b) The linear system for the Newton step $\boldsymbol{s}_0$ from starting point $\boldsymbol{x}_0 = [\,2 \quad 2\,]^T$ is

$$\boldsymbol{H}_f(\boldsymbol{x}_0)\boldsymbol{s}_0 = \begin{bmatrix} 9 & -4 \\ -4 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} -9 \\ 2 \end{bmatrix} = -\nabla f(\boldsymbol{x}_0).$$

Solving this system, we obtain $\boldsymbol{s}_0 = [\,0.143 \quad 2.57\,]^T$, so the new iterate is $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0 = [\,2.14 \quad 4.57\,]^T$. (c) This is a good step in that the value of the objective function has decreased from $f(\boldsymbol{x}_0) = 2.5$ to $f(\boldsymbol{x}_1) = 0.653$. (d) This is a bad step in that $\boldsymbol{x}_1$ is farther from the true solution $\boldsymbol{x}^*$ than the starting point $\boldsymbol{x}_0$ was.

6.9. (a) The solution $\boldsymbol{x}^*$ to this problem clearly occurs when $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$. Since $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ and $\boldsymbol{H}_f(\boldsymbol{x}) = \boldsymbol{A}$, the linear system for the Newton step $\boldsymbol{s}_0$ from any starting point $\boldsymbol{x}_0$ is

$$\boldsymbol{H}_f(\boldsymbol{x}_0)\boldsymbol{s}_0 = \boldsymbol{A}\boldsymbol{s}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0 = -\nabla f(\boldsymbol{x}_0),$$

so that $\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0$. We then have $\boldsymbol{A}\boldsymbol{x}_1 = \boldsymbol{A}(\boldsymbol{x}_0 + \boldsymbol{s}_0) = \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{A}\boldsymbol{s}_0 = (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0) + \boldsymbol{A}\boldsymbol{s}_0 = \boldsymbol{b}$, which implies that $\boldsymbol{x}_1 = \boldsymbol{x}^*$, i.e., Newton's method converges to the solution in only one iteration from any starting point. (b) Using the steepest descent method, we want to compute $\min_\alpha f(\boldsymbol{x}_0 + \alpha\boldsymbol{s}_0)$, where $\boldsymbol{s}_0 = -\nabla f(\boldsymbol{x}) = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$. It is easy to show (see page 472 of textbook) that the minimizing value for α is given by $\alpha = \boldsymbol{s}_k^T\boldsymbol{s}_k / \boldsymbol{s}_k^T\boldsymbol{A}\boldsymbol{s}_k$. Now if $\boldsymbol{x}_0 - \boldsymbol{x}^*$ is an eigenvector of $\boldsymbol{A}$, then we have $\lambda(\boldsymbol{x}_0 - \boldsymbol{x}^*) = \boldsymbol{A}(\boldsymbol{x}_0 - \boldsymbol{x}^*) = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}$. Thus, we have $\boldsymbol{s}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0 = \lambda(\boldsymbol{x}^* - \boldsymbol{x}_0)$, and hence $\boldsymbol{A}\boldsymbol{s}_0 = \lambda\boldsymbol{A}(\boldsymbol{x}^* - \boldsymbol{x}_0) = \lambda(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{A}\boldsymbol{x}_0) = \lambda(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0) = \lambda\boldsymbol{s}_0$, which shows that $\boldsymbol{s}_0$ is also an eigenvector of $\boldsymbol{A}$. This implies that $\lambda = \boldsymbol{s}_0^T\boldsymbol{A}\boldsymbol{s}_0 / \boldsymbol{s}_0^T\boldsymbol{s}_0$. Thus, by our earlier result, we have $\alpha = 1/\lambda$. We therefore have $\boldsymbol{x}_1 = \boldsymbol{x}_0 - \alpha\nabla f(\boldsymbol{x}_0) = \boldsymbol{x}_0 + (1/\lambda)(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0) = \boldsymbol{x}_0 + (1/\lambda)(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{A}\boldsymbol{x}_0) = \boldsymbol{x}_0 + (1/\lambda)(\boldsymbol{A}(\boldsymbol{x}^* - \boldsymbol{x}_0)) = \boldsymbol{x}_0 + (1/\lambda)(\lambda(\boldsymbol{x}^* - \boldsymbol{x}_0)) = \boldsymbol{x}^*$. Thus, the steepest descent method converges to the solution in only one iteration from this starting point.

6.10. (a) Since f is coercive on $\mathbb{R}^n$, there is an $r > 0$ such that $f(\boldsymbol{x}) > f(\boldsymbol{o})$ for all $\boldsymbol{x} \in \mathbb{R}^n$ such that $\|\boldsymbol{x}\| > r$. Let $B_r = \{\boldsymbol{x} \in \mathbb{R}^n : \|\boldsymbol{x}\| \le r\}$ denote the closed ball of radius r. Then the continuous function f must have a minimum $\boldsymbol{x}^*$ in the closed and bounded set B_r. Moreover, since $\boldsymbol{o} \in B_r$, we have $f(\boldsymbol{x}^*) \le f(\boldsymbol{o}) < f(\boldsymbol{x})$ for any $\boldsymbol{x} \notin B_r$. Thus, $\boldsymbol{x}^*$ is a minimum of f on $\mathbb{R}^n$. (b) Let $\boldsymbol{x}_0$ be any point in S. Since f is coercive, there is an $r > \|\boldsymbol{x}_0\|$ such that $f(\boldsymbol{x}) > f(\boldsymbol{x}_0)$ for all $\boldsymbol{x} \in S$ such that $\|\boldsymbol{x}\| > r$. Then the continuous function f must have a minimum $\boldsymbol{x}^*$ in the closed and bounded set $B_r \cap S$. Moreover, since $\boldsymbol{x}_0 \in B_r \cap S$, we have $f(\boldsymbol{x}^*) \le f(\boldsymbol{x}_0) < f(\boldsymbol{x})$ for any $\boldsymbol{x} \in S \setminus B_r$. Thus, $\boldsymbol{x}^*$ is a minimum of f on S.

6.11. Suppose that f has a sublevel set $L_\gamma = \{\boldsymbol{x} \in S : f(\boldsymbol{x}) \le \gamma\}$ that is closed and bounded. Then since f is continuous, it must have a minimum $\boldsymbol{x}^*$ in L_γ. By definition, we then have $f(\boldsymbol{x}^*) \le \gamma < f(\boldsymbol{x})$ for any $\boldsymbol{x} \in S \setminus L_\gamma$. Thus, $\boldsymbol{x}^*$ is a minimum of f on S.

6.12. (a) Suppose that $\boldsymbol{x}$ is a local minimum of a convex function f on a convex set $S \in \mathbb{R}^n$, but not a global minimum. Then there exists a point $\boldsymbol{y} \in S$ such that $f(\boldsymbol{y}) < f(\boldsymbol{x})$. Consider the line segment between $\boldsymbol{x}$ and $\boldsymbol{y}$, and recall that the graph of a convex function along any line segment in S must lie on or below the chord connecting the function values at the endpoints of the segment. But this contradicts the assumption that $\boldsymbol{x}$ is a local minimum. Thus, any local minimum of f must be a global minimum. (b) Suppose that $\boldsymbol{x}$ and $\boldsymbol{y}$ are distinct local minima of a strictly convex function f on a convex set $S \in \mathbb{R}^n$. Consider the line segment between $\boldsymbol{x}$ and $\boldsymbol{y}$, and recall that the graph of a strictly convex function along any line segment in S must lie strictly below the chord connecting the function values at the endpoints of the segment. But this contradicts the assumption that $\boldsymbol{x}$ and $\boldsymbol{y}$ are local minima. Thus, any local minimum of f must be the unique global minimum.

6.13. First, suppose that f is strictly quasiconvex. If $x_1 < x_2 \leq x^*$, then $f(x_2) < \max\{f(x_1), f(x^*)\} = f(x_1)$. If, on the other hand, $x^* \leq x_1 < x_2$, then $f(x_1) < \max\{f(x^*), f(x_2)\} = f(x_2)$. Thus, f is unimodal. Now suppose that f is unimodal. For $x_1, x_2 \in [a, b]$, $x_1 < x_2$, let $x_\alpha = \alpha x_1 + (1-\alpha)x_2$, $\alpha \in (0, 1)$. If $x_1 < x_\alpha \leq x^*$, then $f(x_\alpha) < f(x_1) \leq \max\{f(x_1), f(x_2)\}$. If, on the other hand, $x^* \leq x_\alpha < x_2$, then $f(x_\alpha) < f(x_2) \leq \max\{f(x_1), f(x_2)\}$. Thus, f is quasiconvex.

6.14. Let $\boldsymbol{y}$ be any nonzero m-vector. Then we have

$$\begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{y} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{B} & \boldsymbol{J}^T \\ \boldsymbol{J} & \boldsymbol{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \boldsymbol{o} \\ \boldsymbol{y} \end{bmatrix}^T \begin{bmatrix} \boldsymbol{J}^T\boldsymbol{y} \\ \boldsymbol{o} \end{bmatrix} = 0.$$

Thus, the matrix cannot be positive definite.

6.15. Using the penalty function method, we minimize the unconstrained function

$$\phi_\rho(x, y) = x^2 + y^2 + \tfrac{1}{2}\rho(x + y - 1)^2,$$

whose gradient is given by

$$\nabla\phi_\rho(x, y) = \begin{bmatrix} 2x + \rho(x + y - 1) \\ 2y + \rho(x + y - 1) \end{bmatrix}.$$

The critical point is therefore given in this case by the linear system

$$\begin{bmatrix} 2 + \rho & \rho \\ \rho & 2 + \rho \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \rho \\ \rho \end{bmatrix}.$$

Solving this system by Gaussian elimination, we obtain

$$x = y = \frac{\rho}{2\rho + 2} = \frac{1}{2 + 2/\rho} \rightarrow \frac{1}{2} \quad \text{as} \quad \rho \rightarrow \infty.$$

The point $(1/2, 1/2)$ is easily confirmed as the true solution using, for example, Lagrange multipliers.

6.16. (*a*) The Jacobian $\boldsymbol{J}_g(x, y)$ vanishes (i.e., the constraint set has a cusp) at the solution $(1, 0)$, and thus the Lagrange multipliers do not exist. (*b*) Using the penalty method, we minimize the unconstrained function

$$\phi_\rho(x, y) = x^2 + y^2 + \tfrac{1}{2}\rho(y^2 - (x-1)^3)^2,$$

whose gradient is

$$\nabla\phi_\rho(x, y) = \begin{bmatrix} 2x + \rho(y^2 - (x-1)^3)(-3(x-1)^2) \\ 2y + \rho(y^2 - (x-1)^3)(2y) \end{bmatrix}.$$

Thus, at a critical point of this function, we have

$$y = 0, \quad x = 1 + (-2x/3\rho)^{1/5},$$

so that $(x, y) \to (1, 0)$ as $\rho \to \infty$. The penalty method in effect smooths out the cusp, and the solutions to the unconstrained subproblems converge to the constrained minimum in the limit as $\rho \to \infty$.

6.17. (*a*) There are five vertices: $(0, 0)$, $(0, 1.5)$, $(0.857, 0.857)$, $(1.09, 0.545)$, and $(1.2, 0)$. (*b*) The corresponding values of the objective function are 0, -3, -4.29, -4.37, and -3.6, respectively, so the minimum occurs at $(1.09, 0.545)$.

6.18.

$$\min_{\boldsymbol{x}} \boldsymbol{c}^T\boldsymbol{x} = \begin{bmatrix} -8 \\ -11 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

subject to

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 5 & 4 & 1 & 0 \\ -1 & 3 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 40 \\ 12 \end{bmatrix} = \boldsymbol{b} \quad \text{and} \quad \boldsymbol{x} \geq \boldsymbol{o},$$

where x_3 and x_4 are *slack variables*.

Computer Problems

6.1.
```
function cp06_01 % effects of floating-point arithmetic
disp('(a)          x                      f(x)');
x = linspace(1-3*sqrt(eps), 1+3*sqrt(eps), 30);
f = inline('x.^2-2*x+2','x'); y = f(x); fprintf('%23.15f %23.15f\n', [x; y]);
disp('(b)          x                      f(x)');
x = linspace(sqrt(2)/2-4*sqrt(eps), sqrt(2)/2+4*sqrt(eps), 30);
f = inline('0.5-x.*exp(-x.^2)'); y = f(x); fprintf('%23.15f %23.15f\n', [x; y]);
```

6.2.
```
function cp06_02 % accuracy of computing minimum
a=-2*pi; b=2*pi; disp('(c) Original formulation of function:');
disp('    tol      computed maximum');
```

```
for i=2:15,
  tol = 10^(-i); xmin = fminbnd(@f, a, b, optimset('TolX', 10^(-i)));
  fprintf('%9.1e  %20.14e\n', tol, xmin);
end; disp(' ');
x=[-0.001:0.00001:0.001]'; y=f(x); plot(x,y); xlabel('x'); ylabel('-f(x)');
title('Computer Problem 6.2 -- Maximization of Function');
disp('(e) Reformulation of function:');
disp('    tol      computed maximum');
for i=2:15,
  tol = 10^(-i); xmin = fminbnd(@g, a, b, optimset('TolX', 10^(-i)));
  fprintf('%9.1e  %20.14e\n', tol, xmin);
end

function [fx] = f(x) % original formulation of objective function
if x == 0, fx = -0.5;
else fx = (cos(x)-1)/(x.*x); end

function [fx] = g(x) % reformulation using identity 1-cos(x) = 2 sin^2(x/2).
if x == 0, fx = -0.5;
else fx = -(2*sin(x/2)^2)/(x.*x); end
```

6.3.
```
function cp06_03 % minimize nonlinear functions of one variable
fs = {'x.^4-14*x.^3+60*x.^2-70*x' '0.5*x.^2-sin(x)' 'x.^2+4*cos(x)' 'gamma(x)'};
part = {'(a)' '(b)' '(c)' '(d)'};
for i=1:4,
  f = inline(fs{i},'x'); disp([part{i} ' f(x) = ' fs{i}]); min = fminbnd(f,0,3)
  figure(i); x = linspace(0,3,100); y = f(x); plot(x,y); xlabel('x'); ylabel('f(x)');
  title(['Computer Problem 6.3' part{i} ' -- Minimize f(x) = ' fs{i}]);
end
```

6.4.
```
function cp06_04 % minimize non-unimodal function
a = 0; b = 2*pi;
fs = {'cos(2*x)+(1-x*(2*pi-x))' 'sin(4*x)+(1-x*(2*pi-x))'};
for i = 1:2
  f = inline(fs{i},'x'); figure(i); fplot(f,[a b]); hold on;
  xlabel('x'); ylabel('f(x)'); title(['Computer Problem 6.4 -- f(x) = ' fs{i}]);
  min = fminbnd(f,a,b), plot(min,f(min),'x');
end
```

6.5.
```
function cp06_05 % optimization of water hose problem
f = inline('-max(roots([9.8065/(2*20^2*cos(alpha)^2) -tan(alpha) 13.5]))');
alpha = fminbnd(f,0.9,2.0), dist = -f(alpha)
```

6.6.
```
function cp06_06 % general line search routine
x0 = [2; 3];, s = [-5; -7]; [x, alpha] = LineSearch(@test, x0, s)

function [x_new, alpha] = LineSearch(f, x, s) % min_alpha f(x+alpha*s)
k = 0; maxits = 10; f0 = phi(0,f,x,s); alpha = 1; f1 = phi(alpha,f,x,s);
while f1 > f0  &  k < maxits
  k = k+1; alpha = alpha/2; f1 = phi(alpha,f,x,s);
end
```

```
options = optimset('TolX', 1e-6, 'MaxIter', 50);
alpha = fminbnd(@phi,0,2*alpha,options,f,x,s); x_new = x+alpha*s;

function [val] = phi(alpha,fun,x,s)
val = feval(fun,x+alpha*s);

function [y] = test(x)
y = 5*x(1)^2+7*x(2)^2-3*x(1)*x(2);
```

6.7.
```
function cp06_07 % classify critical points of nonlinear function
disp('(a) Critical points are (-1,-1), (1,0), (0,-1), and (0,0).'); disp(' ');
disp('(b) (-1,-1) is a maximum, (1,0) and (0,-1) are saddle points, and');
disp('    (0,0) is a degenerate saddle point (Hessian is singular).');
[x, y] = meshgrid(-1.5:0.01:1.5, -1.5:0.01:0.5);
contour(-1.5:0.01:1.5,-1.5:0.01:0.5,2*x.^3-3*x.^2-6*x.*y.*(x-y-1),100);
hold on; plot([0; 1; 0; -1],[0; 0; -1; -1],'x'); xlabel('x'); ylabel('y');
title('Computer Problem 6.7(c) -- Contour plot of f');
disp('(d)'); disp('Seeking minimum:');
options = optimset('GradObj', 'on', 'Hessian', 'on', 'Display', 'off');
[x, fval, exitflat, output, g, H] = fminunc(@f,[0;-0.5],options);
x, grad = g, eigenvalues = eig(H), disp('Seeking maximum:');
[x, fval, exitflat, output, g, H] = fminunc(@minus_f,[-1.1;-1.1],options);
x, grad = g, eigenvalues = eig(H)

function [y, g, H] = f(x)
y = 2*x(1)^3-3*x(1)^2-6*x(1)*x(2)*(x(1)-x(2)-1);
g = 6*[x(1)^2-x(1)-x(1)*x(2)-x(2)*(x(1)-x(2)-1); x(1)*x(2)-x(1)*(x(1)-x(2)-1)];
H = [12*(x(1)-x(2))-6, -12*(x(1)-x(2)); -12*(x(1)-x(2)), 12*x(1)];

function [y, g, H] = minus_f(x)
[y, g, H] = f(x); y = -y; g = -g; H = -H;
```

6.8.
```
function cp06_08 % find and classify critical points of nonlinear function
f = inline('2*x(1)^2-1.05*x(1)^4+x(1)^6/6+x(1)*x(2)+x(2)^2','x');
gradf = inline('[4*x(1)-4.2*x(1)^3+x(1)^5+x(2); x(1)+2*x(2)]','x');
hessf = inline('[4-12.6*x(1)^2+5*x(1)^4, 1; 1, 2]','x');
[x, y] = meshgrid(-2.5:0.01:2.5, -2.5:0.01:2.5); figure; hold on;
contour(-2.5:0.01:2.5,-2.5:0.01:2.5,2*x.^2-1.05*x.^4+x.^6/6+x.*y+y.^2,50);
xlabel('x'); ylabel('y'); title('Computer Problem 6.8(c) -- Contour plot of f');
x0 = [-1.7 0 1.7 -1 1; 1 0 -1 0.5 -0.5];
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
for i = 1:5
  disp('Critical point');
  x = fsolve(gradf, x0(:,i), options), plot(x(1),x(2),'x'); lambda = eig(hessf(x));
  if min(lambda) > 0, disp('is a minimum.');
  elseif max(lambda) < 0, disp('is a maximum.');
  else disp('is a saddle point.'); end; disp(' ');
end
disp('Global minimum value of function is 0 at (0,0).');
```

6.9.
```
function cp06_09 % minimize Rosenbrock's function
```

```
f = inline('100*(x(2)-x(1)^2)^2+(1-x(1))^2','x');
gradf = inline('[400*x(1)*(x(1)^2-x(2))+2*(x(1)-1); 200*(x(2)-x(1)^2)]','x');
hessf = inline('[1200*x(1)^2-400*x(2)+2, -400*x(1); -400*x(1), 200]','x');
tol = 1e-6; maxits = 50; x0 = [-1 0 2; 1 1 1];
[x, y] = meshgrid(-1.0:0.01:2.0, 0.0:0.01:2.0); figure; hold on;
contour(-1.0:0.01:2.0,0.0:0.01:2.0,100*(y-x.^2).^2+(1-x).^2,50); plot(1,1,'x');
xlabel('x'); ylabel('y'); title('Computer Problem 6.9 -- Contour plot of f');
for i = 1:3
  k = 0; x = x0(:,i); s = ones(size(x));
  fprintf('Starting point:\n'); x
  disp('Steepest descent:');
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -gradf(x); [x_new, alpha] = LineSearch(f, x, s);
    s = x_new-x; x = x_new;
  end; x, k
  disp('Newton''s method:');
  k = 0; x = x0(:,i); s = ones(size(x));
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -(hessf(x)\gradf(x)); x = x+s;
  end; x, k
  disp('Damped Newton method:');
  k = 0; x = x0(:,i); s = ones(size(x));
  while norm(s) > tol  &  k < maxits
    k = k+1; s = -(hessf(x)\gradf(x)); [x_new, alpha] = LineSearch(f, x, s);
    s = x_new-x; x = x_new;
  end; x, k
end

function [x_new, alpha] = LineSearch(f, x, s)      % min_alpha f(x+alpha*s)
k = 0; maxits = 10; f0 = phi(0,f,x,s); alpha = 1; f1 = phi(alpha,f,x,s);
while f1 > f0  &  k < maxits
  k = k+1; alpha = alpha/2; f1 = phi(alpha,f,x,s);
end
options = optimset('TolX', 1e-6, 'MaxIter', 50);
alpha = fminbnd(@phi,0,2*alpha,options,f,x,s); x_new = x+alpha*s;

function [val] = phi(alpha,fun,x,s)
val = feval(fun,x+alpha*s);
```

6.10.
```
function cp06_10 % compute eigenvalues of matrix using optimization
A = [6 2 1; 2 3 1; 1 1 1];
f = inline('(transpose(x)*[6 2 1; 2 3 1; 1 1 1]*x)/(transpose(x)*x)','x');
mf = inline('-(transpose(x)*[6 2 1; 2 3 1; 1 1 1]*x)/(transpose(x)*x)','x');
cf = inline('(transpose(x)*[6 2 1; 2 3 1; 1 1 1]*x)','x');
mcf = inline('-(transpose(x)*[6 2 1; 2 3 1; 1 1 1]*x)','x');
disp('(a) MATLAB function fminunc:');
options = optimset('LargeScale', 'off', 'Display', 'off');
x = fminunc(f,rand(3,1),options); min_eigenvalue = f(x), eigenvector = x/norm(x)
x = fminunc(mf,rand(3,1),options); max_eigenvalue = f(x), eigenvector = x/norm(x)
disp('    MATLAB function eig:');
```

```
[X, Lambda] = eig(A); eigenvalues = diag(Lambda), eigenvectors = X
disp('(b) MATLAB function fmincon:');
[x, fval, exitflag, output, LAMBDA] = fmincon(cf,rand(3,1),[],[],[],[],[],...
  [],@cons,options); min_eigenvalue = f(x), eigenvector = x
Lagrange_multiplier = LAMBDA.eqnonlin
[x, fval, exitflag, output, LAMBDA] = fmincon(mcf,rand(3,1),[],[],[],[],[],...
  [],@cons,options); max_eigenvalue = f(x), eigenvector = x
Lagrange_multiplier = LAMBDA.eqnonlin

function [c, ceq] = cons(x)
c = []; ceq = x'*x-1;
```

6.11.

```
function cp06_11 % implement BFGS method for unconstrained optimization
f = inline('transpose(x)*diag([10 5 1])*x','x');
gradf = inline('diag([10 5 1])*x','x'); hessf = inline('diag([10 5 1])','x');
x0 = [3; 1; -5]; tol = 1e-6; maxits = 20; disp('BFGS method:');
k = 0; x = x0; g = gradf(x); s = ones(size(x)); B = eye(length(x)); % B = hessf(x);
while norm(s) > tol  &  k < maxits
  k = k+1; s = -(B\g); x = x+s; y = g; g = gradf(x); y = g-y; z = B*s;
  B = B+((y*y')/(y'*s))-((z*z')/(z'*s));
end; x,k, disp(' '); disp('Newton''s method:');
k = 0; x = x0; s = ones(size(x));
while norm(s) > tol  &  k < maxits
  k = k+1; s = -(hessf(x)\gradf(x)); x = x+s;
end; x, k, disp('Steepest descent:');
k = 0; x = x0; s = ones(size(x));
while norm(s) > tol  &  k < maxits
  k = k+1; s = -gradf(x); [x_new, alpha] = LineSearch(f, x, s);
  s = x_new-x; x = x_new;
end; x, k

function [x_new, alpha] = LineSearch(f, x, s)      % min_alpha f(x+alpha*s)
k = 0; maxits = 10; f0 = phi(0,f,x,s); alpha = 1; f1 = phi(alpha,f,x,s);
while f1 > f0  &  k < maxits
  k = k+1; alpha = alpha/2; f1 = phi(alpha,f,x,s);
end
options = optimset('TolX', 1e-6, 'MaxIter', 50);
alpha = fminbnd(@phi,0,2*alpha,options,f,x,s); x_new = x+alpha*s;

function [val] = phi(alpha,fun,x,s)
val = feval(fun,x+alpha*s);
```

6.12.

```
function cp06_12 % implement conjugate gradient method for optimization
f = inline('transpose(x)*diag([10 5 1])*x','x');
gradf = inline('diag([10 5 1])*x','x'); hessf = inline('diag([10 5 1])','x');
ms = {'Fletcher-Reeves' 'Polak-Ribiere'};
x0 = [3; 1; -5]; tol = 1e-6; maxits = 30;
for i = 1:2
  disp(['Congugate gradient method using ' ms{i} ' formula for beta:']);
  k = 0; x = x0; g = gradf(x); s = -g;
  while norm(s) > tol  &  k < maxits
```

```
    k = k+1; [x, alpha] = LineSearch(f, x, s); g_new = gradf(x);
    if i == 1, beta = (g_new'*g_new)/(g'*g);
    else beta = ((g_new-g)'*g_new)/(g'*g)'; end
    s = -g_new+beta*s; g = g_new;
  end; x, k
end

function [x_new, alpha] = LineSearch(f, x, s)      % min_alpha f(x+alpha*s)
k = 0; maxits = 10; f0 = phi(0,f,x,s); alpha = 1; f1 = phi(alpha,f,x,s);
while f1 > f0  &  k < maxits
  k = k+1; alpha = alpha/2; f1 = phi(alpha,f,x,s);
end
options = optimset('TolX', 1e-6, 'MaxIter', 50);
alpha = fminbnd(@phi,0,2*alpha,options,f,x,s); x_new = x+alpha*s;

function [val] = phi(alpha,fun,x,s)
val = feval(fun,x+alpha*s);
```

6.13.
```
function cp06_13 % nonlinear least squares
fa = inline('[x(1)^2+x(2)^2-2; (x(1)-2)^2+x(2)^2-2; (x(1)-1)^2+x(2)^2-9]','x');
fb = inline('[x(1)^2+x(2)^2+x(1)*x(2); (sin(x(1)))^2; (cos(x(2)))^2]','x');
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
disp('(a)'); x = lsqnonlin(fa,rand(2,1),[],[],options)
disp('(b)'); x = lsqnonlin(fb,rand(2,1),[],[],options)
```

6.14.
```
function cp06_14 % least squares fit to drug concentration data
f = inline('x(1).*exp(x(2).*t)','x', 't');
t = [0.5:0.5:4]'; y = [6.8; 3; 1.5; 0.75; 0.48; 0.25; 0.2; 0.15];
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
x = [ones(length(t),1), t]\log(y); x0 = [exp(x(1)); x(2)];
disp('(a) Nonlinear fit:'); x = lsqcurvefit(f,x0,t,y,[],[],options)
disp('(b) Linear fit:'); x = x0
```

6.15.
```
function cp06_15 % least squares fit to bacterial population data
f = inline('x(1).*x(2).^k','x', 'k');      % x(1) = P_0, x(2) = r
k = [1:8]'; P = [0.19; 0.36; 0.69; 1.3; 2.5; 4.7; 8.5; 14];
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
x = [ones(length(k),1), k]\log(P); x0 = [exp(x(1)); exp(x(2))];
disp('(a) Nonlinear fit:'); x = lsqcurvefit(f,x0,k,P,[],[],options)
disp('(b) Linear fit:'); x = x0
```

6.16.
```
function cp06_16 % Michaelis-Menten equation for enzyme reaction kinetics
f = inline('x(1)./(1+x(2)./S)','x', 'S');          % x(1) = V, x(2) = K_m
S = [2.5; 5; 10; 15; 20]; v0 = [0.024; 0.036; 0.053; 0.060; 0.064];
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
e = ones(length(S),1); x = [e, e./S]\(e./v0); x0 = [1/x(1); x(2)/x(1)];
disp('(a) Nonlinear fit:'); x = lsqcurvefit(f,x0,S,v0,[],[],options)
disp('(b) Lineweaver-Burk linear fit:'); x = x0
disp('Dixon linear fit:'), x = [e, S]\(S./v0); x = [1/x(2); x(1)/x(2)]
disp('Eadie-Hofstee linear fit:'); x = [e, v0./S]\v0; x = [x(1); -x(2)]
```

6.17.
```
function cp06_17 % nonlinear least squares fit
```

```
global t y; t = [0; 0.25; 0.5; 0.75; 1; 1.25; 1.5; 1.75; 2];
y = [20; 51.58; 68.73; 75.46; 74.36; 67.09; 54.73; 37.98; 17.28];
x0 = [100; 5; -20; -100; -2]; % starting guesses correct to 1 digit
options = optimset('LargeScale', 'off', 'Display', 'off');
disp('(a) MATLAB function fminunc:'); x = fminunc(@phi,x0,options)
disp('(b) MATLAB function fsolve:'); x = fsolve(@grad_phi,x0,options)
disp('(c) MATLAB function fminbnd:'); x5 = fminbnd(@f1d,x0(5)-1,x0(5)+1,options);
A = [ones(length(t),1), t, t.^2,exp(x5.*t)]; x = A\y, x5
disp('(d) MATLAB function fzero:'); x5 = fzero(@df1d,x0(5),options);
A = [ones(length(t),1), t, t.^2, exp(x5.*t)]; x = A\y, x5
disp('(e) MATLAB function lsqcurvefit:'); x = lsqcurvefit(@fun,x0,t,y,[],[],options)

function [f] = phi(x); global t y;
r = y-x(1)-x(2).*t-x(3).*t.^2-x(4).*exp(x(5).*t); f = 0.5*r'*r;

function [g] = grad_phi(x); global t y;
r = y-x(1)-x(2).*t-x(3).*t.^2-x(4).*exp(x(5).*t);
J = -[ones(length(t),1), t, t.^2, exp(x(5).*t), x(4).*t.*exp(x(5).*t)];
g = J'*r;

function [f] = f1d(x5); global t y;
A = [ones(length(t),1), t, t.^2, exp(x5.*t)]; x = A\y; f = phi([x; x5]);

function [d] = df1d(x5); global t y;
A = [ones(length(t),1), t, t.^2, exp(x5.*t)]; x = A\y;
r = y-x(1)-x(2).*t-x(3).*t.^2-x(4).*exp(x5.*t);
d = -(x(4).*t.*exp(x5.*t))'*r;

function [f] = fun(x, t);
f = x(1)+x(2).*t+x(3).*t.^2+x(4).*exp(x(5).*t);
```

6.18.
```
function cp06_18 % constrained optimization of chemical plant
f = inline('-5*x(1)*x(2)*x(4)+4*x(1)*x(2)^1.4+0.75*x(3)^0.6','x');
options = optimset('LargeScale', 'off', 'Display', 'off');
[x, fval, exitflag, output] = fmincon(f,[1;1;1;1],[],[],[],[],...
[0;0;0;0],[Inf;Inf;Inf;1],@nonlcon,options); x, fval = -fval

function [c, ceq] = nonlcon(x)
c = []; ceq = x(1)*x(4)-8.4*x(2)*x(3)*(1-x(4))^2;
```

6.19.
```
function cp06_19 % Fermat's Principle of Least Time
disp('(a) min_x norm(a-[x;0])/refrac_index + norm(b-[x;0]).'); disp(' ');
g = inline('norm([-1;1]-[x;0])/1.5+norm([1;-1]-[x;0])','x');
disp('(b) For glass:'); x = fminbnd(g,-1,1), disp('(c) Snell''s Law:');
theta_i = atan(x+1), theta_r = atan(1-x), ratio = sin(theta_i)/sin(theta_r)
w = inline('norm([-1;1]-[x;0])/1.33+norm([1;-1]-[x;0])','x');
disp('(d) For water:'); x = fminbnd(w,-1,1), disp('Snell''s Law:');
theta_i = atan(x+1), theta_r = atan(1-x), ratio = sin(theta_i)/sin(theta_r)
```

6.20.
```
function cp06_20 % optimizing lifeguard's path
f = inline('norm([-40;30]-[x;0])/5+norm([40;-40]-[x;0])','x');
```

```
[x, t] = fminbnd(f,-40,40);
fprintf('Lifeguard should enter water %g meters from center point along\n', x);
fprintf('shore between lifeguard and swimmer.\n')
fprintf('Lifeguard requires %g seconds to reach swimmer by optimal path.\n', t);
```

6.21.
```
function cp06_21 % linear programming
f = [2; 4; 1; 1]; A = [1 3 0 1; 2 1 0 0; 0 1 4 1]; b = [4; 3; 3];
options = optimset('Display', 'off');
x = linprog(-f,A,b,[],[],[0;0;0;0],[],[],options)
```

6.22.
```
function cp06_22 % constrained optimization using Lagrange multipliers
part = {'(a)' '(b)' '(c)'}; n = [5 3 5]; m = [3 2 3];
options = optimset('TolX', 1e-6, 'MaxIter', 50, 'Display', 'off');
fs = {'(4*x(1)-x(2))^2+(x(2)+x(3)-2)^2+(x(4)-1)^2+(x(5)-1)^2 ' ...
  '4*x(1)*x(1)+2*x(2)*x(2)+2*x(3)*x(3)-33*x(1)+16*x(2)-24*x(3) ' ...
  '(x(1)-1)^2+(x(1)-x(2))^2+(x(2)-x(3))^2+(x(3)-x(4))^4+(x(4)-x(5))^4'};
gs = {['8*(4*x(1)-x(2))+x(6); -2*(4*x(1)-x(2))+2*(x(2)+x(3)-2)+3*x(6)+x(8); ' ...
  '2*(x(2)+x(3)-2)+x(7); 2*(x(4)-1)+x(7); 2*(x(5)-1)-2*x(7)+x(8); ' ...
  'x(1)+3*x(2); x(3)+x(4)-2*x(5); x(2)-x(5)]'], ...
['8*x(1)-33+3*x(4)+4*x(5); 4*x(2)+16-4*x(2)*x(4); 4*x(3)-24-2*x(3)*x(5); ' ...
  '3*x(1)-2*x(2)*x(2)-7; 4*x(1)-2*x(3)*x(3)-11]'], ...
['2*(x(1)-1)+2*(x(1)-x(2))+x(6)+x(5)*x(8); ' ...
  '-2*(x(1)-x(2))+2*(x(2)-x(3))+2*x(2)*x(6)+x(7); ' ...
  '-2*(x(2)-x(3))+4*(x(3)-x(4))^3+3*x(3)*x(3)*x(6)-2*x(3)*x(7); ' ...
  '-4*(x(3)-x(4))^3+4*(x(4)-x(5))^3+x(7); -4*(x(4)-x(5))^3+x(1)*x(8); ' ...
  'x(1)+x(2)*x(2)+x(3)*x(3)*x(3)-3*sqrt(2)-2; ' ...
  'x(2)-x(3)*x(3)+x(4)-2*sqrt(2)-2; x(1)*x(5)-2]']};
for i = 1:3
  disp(part{i}); f = inline(fs{i}); gradL = inline(gs{i});
  x = fsolve(gradL, ones(n(i)+m(i),1), options);
  xmin = x(1:n(i)), lambda_min = x(n(i)+1:n(i)+m(i)), fmin = f(xmin)
end
```

Chapter 7

Interpolation

Exercises

7.1. (a) The linear system using the monomial basis is

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Solving this system by Gaussian elimination, we obtain $\boldsymbol{x} = [\,0 \quad 0 \quad 1\,]^T$, so the interpolating polynomial is $p(t) = t^2$. (b) The form of a polynomial of degree two using the Lagrange basis is

$$p(t) = y_1 \frac{(t-t_2)(t-t_3)}{(t_1-t_2)(t_1-t_3)} + y_2 \frac{(t-t_1)(t-t_3)}{(t_2-t_1)(t_2-t_3)} + y_3 \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)}.$$

Substituting the data for this problem, the polynomial becomes

$$\begin{aligned} p(t) &= 1 \frac{(t-0)(t-1)}{(-1-0)(-1-1)} + 0 \frac{(t-(-1))(t-1)}{(0-(-1))(0-1)} + 1 \frac{(t-(-1))(t-0)}{(1-(-1))(1-0)} \\ &= \frac{t(t-1)}{2} + \frac{t(t+1)}{2}, \end{aligned}$$

which is equivalent to the polynomial in part (a). (c) The linear system using the Newton basis is

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0-(-1) & 0 \\ 1 & 1-(-1) & (1-(-1))(1-0) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Solving this system by forward substitution, we obtain $\boldsymbol{x} = [\,1 \quad -1 \quad 1\,]^T$, so the interpolating polynomial is $p(t) = 1 - (t+1) + (t+1)t$, which again is equivalent to the polynomial in part (a).

7.2. $p(t) = 5t^3 - 3t^2 + 7t - 2 = -2 + t(7 + t(-3 + 5t))$.

7.3. (a) Evaluate polynomial $p(t) = x_1 + x_2 t + \cdots + x_n t^{n-1}$ of degree $n-1$ using Horner's method.

$p = x_n$
for $k = n-1$ **to** 1
  $p = p \cdot t$
  $p = p + x_k$
end

(b) Evaluate polynomial $p(t) = x_1 + x_2(t-t_1) + x_3(t-t_1)(t-t_2) + \cdots + x_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$ of degree $n-1$ using Horner's method.

$p = x_n$
for $k = n-1$ **to** 1
  $p = p \cdot (t - t_k)$
  $p = p + x_k$
end

7.4. (a) Without Horner's method, $p_{n-1}(t) = x_n t^n + x_{n-1} t^{n-1} + \cdots + x_1$. If t^k is computed from scratch at each step, then it takes $(n-1)(n-2)/2$ multiplications to compute t^k and $n-1$ multiplications by the x_k, for a total of $n^2/2$ multiplications. If t^k is computed inductively using t^{k-1} from previous step, then it takes $n-1$ multiplications to compute t^n and $n-1$ multiplications by the x_k, for a total of $(2n-2)$ multiplications. With Horner's method, $p_{n-1}(t) = x_1 + t(x_2 + t(\cdots(x_{n-1} + x_n t)\cdots))$, which requires $n-1$ multiplications.

(b)

$$p_{n-1}(t) = y_1 \frac{(t-t_2)(t-t_3)\cdots(t-t_n)}{(t_1-t_2)(t_1-t_3)\cdots(t_1-t_n)} + \cdots + y_n \frac{(t-t_1)(t-t_2)\cdots(t-t_{n-1})}{(t_n-t_1)(t_n-t_2)\cdots(t_n-t_{n-1})}$$

For a given polynomial, $\boldsymbol{y}$ and the denominators are constants, so we have

$$p_{n-1}(t) = c_1(t-t_2)(t-t_3)\cdots(t-t_n) + \cdots + c_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$$

Each basis function requires n multiplications, so for n basis functions the total is n^2 total multiplications.

(c) Without Horner's method, $p_{n-1}(t) = x_1 + x_2(t-t_1) + \cdots + x_n(t-t_1)(t-t_2)\cdots(t-t_{n-1})$, which requires $n(n-1)/2$ multiplications. With Horner's method, $p_{n-1}(t) = x_1 + (t-t_1)(x_2 + (t-t_2)(\cdots(x_{n-1} + x_n(t-t_{n-1}))\cdots))$, which requires $n-1$ multiplications.

7.5. (a) The linear system using the monomial basis is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 11 \\ 29 \\ 65 \\ 125 \end{bmatrix}.$$

Solving this system by Gaussian elimination, we obtain $\boldsymbol{x} = [\,5 \quad 2 \quad 3 \quad 1\,]^T$, so the interpolating polynomial is $p(t) = 5 + 2t + 3t^2 + t^3$.

(b) The Lagrange form of the interpolating polynomial for these data is

$$\begin{aligned} p(t) &= 11\,\frac{(t-2)(t-3)(t-4)}{(1-2)(1-3)(1-4)} + 29\,\frac{(t-1)(t-3)(t-4)}{(2-1)(2-3)(2-4)} \\ &\quad + 65\,\frac{(t-1)(t-2)(t-4)}{(3-1)(3-2)(3-4)} + 125\,\frac{(t-1)(t-2)(t-3)}{(4-1)(4-2)(4-3)} \\ &= -11\,\frac{t^3-9t^2+26t-24}{6} + 29\,\frac{t^3-8t^2+19t-12}{12} \\ &\quad - 65\,\frac{t^3-7t^2+14t-8}{2} + 125\,\frac{t^3-6t^2+11t-6}{6}. \end{aligned}$$

Further reduction shows that this polynomial is equivalent to the polynomial from part (a).

(c) Triangular matrix: The linear system using the Newton basis is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2-1 & 0 & 0 \\ 1 & 3-1 & (3-1)(3-2) & 0 \\ 1 & 4-1 & (4-1)(4-2) & (4-1)(4-1)(4-3) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 11 \\ 29 \\ 65 \\ 125 \end{bmatrix},$$

or

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 2 & 0 \\ 1 & 3 & 6 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 11 \\ 29 \\ 65 \\ 125 \end{bmatrix}.$$

Solving this system by forward substitution, we obtain $\boldsymbol{x} = [\,11 \quad 18 \quad 9 \quad 1\,]^T$, so the interpolating polynomial is $p(t) = 11+18(t-1)+9(t-1)(t-2)+(t-1)(t-2)(t-3)$.

Incremental interpolation:

$$\begin{aligned} p_0(t) &= y_1 = 11 \\ p_1(t) &= p_0(t) + x_2\pi_1(t) = 11 + \frac{29-11}{2-1}(t-1) = 11 + 18(t-1) \\ p_2(t) &= p_1(t) + x_3\pi_2(t) = 11 + 18(t-1) + \frac{65 - p_1(t_3)}{(3-1)(3-2)}(t-1)(t-2) \\ &= 11 + 18(t-1) + 9(t-1)(t-2) \\ p_3(t) &= p_2(t) + x_4\pi_3(t) \\ &= 11 + 18(t-1) + 9(t-1)(t-2) + \\ &\quad \frac{125 - p_2(t_4)}{(4-1)(4-2)(4-3)}(t-1)(t-2)(t-3) \\ &= 11 + 18(t-1) + 9(t-1)(t-2) + (t-1)(t-2)(t-3) \end{aligned}$$

Divided differences:

$$f[t_1] = 11, \quad f[t_2] = 29, \quad f[t_3] = 65, \quad f[t_4] = 125, \quad f[t_1, t_2] = \frac{f[t_2] - f[t_1]}{t_2 - t_1} = 18,$$

$$f[t_2,t_3]=\frac{f[t_3]-f[t_2]}{t_3-t_2}=36,\quad f[t_3,t_4]=\frac{f[t_4]-f[t_3]}{t_4-t_3}=60,$$
$$f[t_1,t_2,t_3]=\frac{f[t_2,t_3]-f[t_1,t_2]}{t_3-t_1}=9,\quad f[t_2,t_3,t_4]=\frac{f[t_3,t_4]-f[t_2,t_3]}{t_4-t_2}=12,$$
$$f[t_1,t_2,t_3,t_4]=\frac{f[t_2,t_3,t_4]-f[t_1,t_2,t_3]}{t_4-t_1}=1.$$

Using these divided differences, the interpolant is

$$\begin{aligned}p_3(t) &= f[t_1]\pi_1(t)+f[t_1,t_2]\pi_2(t)+f[t_1,t_2,t_3]\pi_3(t)+f[t_1,t_2,t_3,t_4]\pi_4(t)\\ &= 11+18(t-1)+9(t-1)(t-2)+(t-1)(t-2)(t-3)\end{aligned}$$

Once again, further reduction shows that the polynomial found by each of the three different methods in this part is equivalent to the polynomial from part (a).

7.6. We need to plug M, h, and n into the bound

$$\max_{t\in[t_1,t_n]}|f(t)-p_{n-1}(t)|\le\frac{Mh^n}{4n},$$

where $|f^{(n)}(t)|\le M$. For this problem $n=5$, so we need the fifth derivative of $f(t)$, $f^{(5)}(t)=\cos(t)$. The maximum of $\cos(t)$ on $[0,\pi/2]$ occurs at $t=0$, where $f^{(5)}(0)=1$, so we can take $M=1$ and the bound becomes

$$\max_{t\in[t,t_n]}|f(t)-p_{n-1}(t)|\le\frac{1\cdot(\pi/8)^5}{4\cdot5}\le 4.6695\times10^{-4}.$$

The interpolant is $p_4(t)=0.2871t^4-0.20359t^3+0.1995t^2+0.99632t$. Using the interpolant, $|f(t)-p_{n-1}(t)|$ may be found for any t. To achieve an error bound of 10^{-10},

$$\begin{aligned}\frac{Mh^n}{4n} &= 10^{-10}\\ \frac{1\cdot(\pi/(2(n-1)))^n}{4n} &= 10^{-10}\\ \frac{\pi^n}{n(2(n-1))^n} &= 4\times10^{-10}\\ n &= 10.63.\end{aligned}$$

Therefore, at least 11 points are required to achieve the bound.

7.7. The behavior of Runge's function can be understood by considering the successive derivatives

$$f(t)=\frac{1}{1+25t^2},\qquad f'(t)=-\frac{50t}{(1+25t^2)^2},$$
$$f''(t)=-\left(\frac{50}{(1+25t^2)^2}-\frac{50t(100t+2500t^3)}{(1+25t^2)^4}\right),\ldots.$$

With each higher derivative, the value of M in the error bound becomes much larger, which explains the poor behavior of interpolants to Runge's function.

7.8. If the Vandermonde matrix is computed inductively, then $n-1$ multiplications are required to compute t_i^{n-1} (and all of the lower powers of t_i), for a total cost for n rows of $n(n-1)$ multiplications. Using explicit exponentiation, the cost for one row is

$$\sum_{j=1}^{n-2} j = \frac{(n-1)(n-2)}{2},$$

for a total cost for n rows of $n(n-1)(n-2)/2$ multiplications.

7.9. The Lagrange interpolant for points (f_a, a), (f_b, b), and (f_c, c) is

$$p(t) = a\,\frac{(t-f_b)(t-f_c)}{(f_a-f_b)(f_a-f_c)} + b\,\frac{(t-f_a)(t-f_c)}{(f_b-f_a)(f_b-f_c)} + c\,\frac{(t-f_a)(t-f_b)}{(f_c-f_a)(f_c-f_b)}.$$

Next we evaluate the interpolant at $t=0$ and rearrange to obtain the form given in Section 5.5.5. Let u, v, w, q, and p be defined as in Section 5.5.5. Then

$$\begin{aligned}
p(0) &= a\,\frac{f_bf_c}{(f_a-f_b)(f_a-f_c)} + b\,\frac{f_af_c}{(f_b-f_a)(f_b-f_c)} + c\,\frac{f_af_b}{(f_c-f_a)(f_c-f_b)} \\
&= \frac{bf_af_c(f_a-f_c) - af_bf_c(f_b-f_c) + cf_af_b(f_b-f_a)}{(f_c-f_a)(f_c-f_b)(f_b-f_a)} \\
&= \frac{bf_a^2f_c - bf_af_c^2 - (af_b^2f_c - af_bf_c^2) + cf_af_b^2 - cf_a^2f_b}{f_c(w-1)f_c(u-1)f_a(v-1)} \\
&= \frac{bw - b - af_b^2/(f_af_c) + av + cu^2 - cf_af_b/(f_c^2)}{(w-1)(u-1)(v-1)} \\
&= \frac{bw - b - avu + av + cu^2 - cwu}{q} \\
&= b + \frac{bw - b - avu + av + cu^2 - cwu - b(w-1)(u-1)(v-1)}{q} \\
&= b + \frac{av - avu + cu^2 - cwu - buvw + buw + buv - bu + bvw - bv}{q}.
\end{aligned}$$

Making the substitutions $u^2 = uvw$, $wu = w^2v$, and $uv - u + vw = vw^2$, so that all terms contain a v, we then have

$$\begin{aligned}
p(0) &= b + \frac{v(a - au + cuw - cw^2 - buw + bw^2 + bu - b)}{q} \\
&= b + \frac{v(cuw - buw - cw^2 + bw^2 - b + a + bu - au)}{q} \\
&= b + \frac{v(w(cu - bu - cw + bw) - (b - a - bu + au))}{q} \\
&= b + \frac{v(w(u-w)(c-b) - (1-u)(b-a))}{q} \\
&= b + \frac{p}{q}.
\end{aligned}$$

7.10. (a) From the definition

$$\pi(t) = (t-t_1)(t-t_2)\cdots(t-t_{j-1})(t-t_j)(t-t_{j+1})\cdots(t-t_n),$$

we have

$$\begin{aligned}
\pi'(t) &= (t-t_1)'(t-t_2)\cdots(t-t_j)\cdots(t-t_n) \\
&\quad + (t-t_1)(t-t_2)'\cdots(t-t_j)\cdots(t-t_n)+\cdots \\
&\quad + (t-t_1)(t-t_2)\cdots(t-t_j)'\cdots(t-t_n)+\cdots \\
&\quad + (t-t_1)(t-t_2)\cdots(t-t_j)\cdots(t-t_n)' \\
&= (t-t_2)\cdots(t-t_j)\cdots(t-t_n)+(t-t_1)\cdots(t-t_j)\cdots(t-t_n) \\
&\quad +\cdots+(t-t_1)(t-t_2)\cdots(t-t_{j-1})(t-t_{j+1})\cdots(t-t_n) \\
&\quad +\cdots+(t-t_1)(t-t_2)\cdots(t-t_j)\cdots(t-t_{n-1}).
\end{aligned}$$

Evaluating at t_j, we obtain

$$\pi'(t_j) = (t_j-t_1)\cdots(t_j-t_{j-1})(t_j-t_{j+1})\cdots(t_j-t_n),$$

since only one term in the derivative does not contain $(t-t_j)$.

(b) The jth Lagrange basis function is given by

$$\ell_j(t) = \frac{\prod_{k=1,k\neq j}^{n}(t-t_k)}{\prod_{k=1,k\neq j}^{n}(t_j-t_k)} = \frac{(t-t_1)\cdots(t-t_{j-1})(t-t_{j+1})\cdots(t-t_n)}{(t_j-t_1)\cdots(t_j-t_{j-1})(t_j-t_{j+1})\cdots(t_j-t_n)},$$

and multiplying both its numerator and denominator by $(t-t_j)$ gives

$$\ell_j(t) = \frac{(t-t_1)\cdots(t-t_{j-1})(t-t_j)(t-t_{j+1})\cdots(t-t_n)}{(t-t_j)(t_j-t_1)\cdots(t_j-t_{j-1})(t_j-t_{j+1})\cdots(t_j-t_n)} = \frac{\pi(t)}{(t-t_j)\pi'(t_j)}.$$

7.11. The proof is inductive on j. The assertion is trivially true for $j=1$, since $x_1 = y_1 = f[t_1]$. We make the inductive hypothesis that the assertion is true for $j-1$, and prove that it must be true or j. Let $p(t)$ be the polynomial of degree at most $j-1$ interpolating $(t_1,y_1),\ldots,(t_j,y_j)$, $q(t)$ be the polynomial of degree at most $j-2$ interpolating $(t_1,y_1),\ldots,(t_{j-1},y_{j-1})$, and $r(t)$ be the polynomial of degree at most $j-2$ interpolating $(t_2,y_2),\ldots,(t_j,y_j)$. Then we have

$$p(t) = q(t) + \frac{t-t_1}{t_j-t_1}(r(t)-q(t)),$$

since the polynomials on either side are of degree at most $j-1$ and agree for $t = t_1, t_2, \ldots, t_j$, and thus the coefficients of their terms of degree $j-1$ must be the same. From the inductive hypothesis, the coefficients of the $j-1$st basis function (i.e., the term of degree $j-2$) for $q(t)$ and $r(t)$ are $f[t_1,\ldots,t_{j-1}]$ and $f[t_2,\ldots,t_j]$, respectively. Thus, from the equality of the two polynomials above, the coefficient of the term of degree $j-1$ for $p(t)$ is given by

$$x_j = \frac{f[t_2,\ldots,t_j]-f[t_1,\ldots,t_{j-1}]}{t_j-t_1} = f[t_1,t_2,\ldots,t_j].$$

7.12. (a)

$$\int_{-1}^{1} 1 \cdot t\, dt = \left(\frac{t^2}{2}\right)\Big|_{-1}^{1} = 0, \qquad \int_{-1}^{1} 1 \cdot \frac{3t^2-1}{2}\, dt = \left(\frac{t^3-t}{2}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} 1 \cdot \frac{5t^3-3t}{2}\, dt = \left(\frac{5t^4}{8} - \frac{3t^2}{4}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} 1 \cdot \frac{35t^4-30t^2+3}{8}\, dt = \left(\frac{7t^5-10t^3+3t}{8}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} 1 \cdot \frac{63t^5-70t^3+15t}{8}\, dt = \left(\frac{63t^6}{48} - \frac{70t^4}{32} + \frac{15t^2}{16}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} t \cdot \frac{3t^2-1}{2}\, dt = \left(\frac{3t^4}{8} - \frac{t^2}{4}\right)\Big|_{-1}^{1} = 0, \quad \int_{-1}^{1} t \cdot \frac{5t^3-3t}{2}\, dt = \left(\frac{5t^5}{10} - \frac{3t^3}{6}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} t \cdot \frac{35t^4-30t^2+3}{8}\, dt = \left(\frac{35t^6}{48} - \frac{30t^4}{32} + \frac{3t^2}{16}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} t \cdot \frac{63t^5-70t^3+15t}{8}\, dt = \left(\frac{9t^7}{8} - \frac{14t^5}{8} + \frac{5t^3}{8}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} \frac{3t^2-1}{2} \cdot \frac{5t^3-3t}{2}\, dt = \left(\frac{15t^6}{24} - \frac{14t^4}{16} + \frac{3t^2}{8}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} \frac{3t^2-1}{2} \cdot \frac{35t^4-30t^2+3}{8}\, dt = \left(\frac{15t^7-25t^5+13t^3-3t}{16}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} \frac{5t^3-3t}{2} \cdot \frac{35t^4-30t^2+3}{8}\, dt = \left(\frac{175t^8}{128} - \frac{225t^6}{96} + \frac{105t^4}{64} - \frac{9t^2}{32}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} \frac{5t^3-3t}{2} \cdot \frac{63t^5-70t^3+15t}{8}\, dt = \left(\frac{35t^9-77t^7+57t^5-15t^3}{16}\right)\Big|_{-1}^{1} = 0,$$

$$\int_{-1}^{1} \frac{35t^4-30t^2+3}{8} \cdot \frac{63t^5-70t^3+15t}{8}\, dt =$$

$$\left(\frac{2205t^{10}}{640} - \frac{4340t^8}{512} + \frac{2814t^6}{384} - \frac{660t^4}{256} + \frac{45t^2}{128}\right)\Big|_{-1}^{1} = 0.$$

(b) The general three-term recurrence for the Legendre polynomials is

$$P_{k+1}(t) = \frac{(2k+1)tP_k(t) - kP_{k-1}(t)}{k+1}.$$

Using this recurrence, the Legendre polynomials can be generated sequentially, starting with $P_0(t) = 1$ and $P_1(t) = t$:

$$\begin{aligned}
P_2(t) &= \frac{(2\cdot 1+1)t\cdot t - 1\cdot 1}{1+1} = \frac{3t^2-1}{2},\\
P_3(t) &= \frac{(2\cdot 2+1)t\cdot (3t^2-1)/2 - 2\cdot t}{2+1} = \frac{5t^3-3t}{2},\\
P_4(t) &= \frac{(2\cdot 3+1)t\cdot (5t^3-3t)/2 - 3\cdot (3t^2-1)/2}{3+1} = \frac{35t^4-30t^2+3}{8},\\
P_5(t) &= \frac{(2\cdot 4+1)t\cdot (35t^4-30t^2+3)/8 - 4\cdot (5t^3-3t)/2}{4+1} = \frac{63t^5-70t^3+15t}{8}.
\end{aligned}$$

(*c*) One way to solve this problem is to set up and solve a linear system. For example, the linear combination of Legendre polynomials that is equal to t^4 can be determined by solving the system

$$\begin{bmatrix}
1 & 0 & -1/2 & 0 & 3/8 & 0\\
0 & 1 & 0 & -3/2 & 0 & 15/8\\
0 & 0 & 3/2 & 0 & -30/8 & 0\\
0 & 0 & 0 & 5/2 & 0 & -70/8\\
0 & 0 & 0 & 0 & 35/8 & 0\\
0 & 0 & 0 & 0 & 0 & 63/8
\end{bmatrix}
\begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5 \end{bmatrix}
=
\begin{bmatrix} 0\\ 0\\ 0\\ 0\\ 1\\ 0 \end{bmatrix}.$$

For this example, $\boldsymbol{x} = [\,0.2 \quad 0 \quad 0.5714 \quad 0 \quad 0.2286\,]^T$. To solve for another monomial, one need change only the right-hand side and solve again. The linear combinations of Legendre polynomials for the first six monomials are

$$\begin{aligned}
1 &= P_0(t),\\
t &= P_1(t),\\
t^2 &= \frac{2}{3}P_2(t) + \frac{1}{3}P_0(t),\\
t^3 &= \frac{2}{5}P_3(t) + \frac{3}{5}P_1(t),\\
t^4 &= \frac{8}{35}P_4(t) + \frac{4}{7}P_2(t) + \frac{1}{5}P_0(t),\\
t^5 &= \frac{8}{63}P_5(t) + \frac{4}{9}P_3(t) + \frac{3}{7}P_1(t).
\end{aligned}$$

7.13. (*a*) From the multiple angle formulas for cosines and sines, we see that

$$\begin{aligned}
T_k(t) &= \cos(k\arccos(t))\\
&= \cos((n-1)\arccos(t) + \arccos(t))\\
&= \cos((n-1)\arccos(t))\cos(\arccos(t)) - \sin((n-1)\arccos(t))\sin(\arccos(t))\\
&= t\,T_{k-1}(t) - [\cos((n-2)\arccos(t)) - \cos((n-1)\arccos(t))\cos(\arccos(t))]\\
&= t\,T_{k-1}(t) - T_{k-2}(t) + t\,T_{k-1}(t)\\
&= 2\,t\,T_{k-1}(t) - T_{k-2}(t).
\end{aligned}$$

(b) Applying the three-term recurrence for Chebyshev polynomials, starting with $T_0(t) = 1$ and $T_1(t) = t$, we have

$$\begin{aligned} T_2(t) &= 2\,t \cdot t - 1 = 2\,t^2 - 1, \\ T_3(t) &= 2\,t \cdot (2\,t^2 - 1) - t = 4t^3 - 3t, \\ T_4(t) &= 2\,t \cdot (4t^3 - 3t) - (2\,t^2 - 1) = 8t^4 - 8t^2 + 1, \\ T_5(t) &= 2\,t \cdot (8t^4 - 8t^2 + 1) - (4t^3 - 3t) = 16t^5 - 20t^3 + 5t. \end{aligned}$$

(c) Substituting the expression for the roots into the definition $T_k(t) = \cos(k \arccos(t))$, we obtain

$$\begin{aligned} T_k(t_i) &= \cos(k \arccos(\cos((2i-1)\pi/(2k)))) \\ &= \cos(k((2i-1)\pi)/(2k)) = \cos((2i\pi - \pi)/2) = 0, \quad i = 1, \ldots, k. \end{aligned}$$

The extrema of $T_k(t)$ correspond to the zeros of its derivative,

$$T_k'(t) = \frac{k \sin(k \arccos(t))}{\sqrt{1-t^2}}.$$

Substituting the expression for the extrema into the derivative, we obtain

$$\begin{aligned} T_k'(t_i) &= \frac{k \sin(k \arccos(\cos(i\pi/k)))}{\sqrt{1-(\cos(i\pi/k))^2}} = \frac{k \sin(ki\pi/k)}{\sqrt{1-(\cos(i\pi/k))^2}} \\ &= \frac{k \sin(i\pi)}{\sqrt{1-(\cos(i\pi/k))^2}} = 0, \quad i = 0, \ldots, k. \end{aligned}$$

7.14. To transform the Chebyshev points from $[-1, 1]$ to an arbitrary interval $[a, b]$, we scale by the relative width of the new interval, $(b-a)/2$, and translate by the center of the new interval, $(a+b)/2$, so the transformed Chebyshev points are given by

$$\tilde{t}_i = t_i \, \frac{b-a}{2} + \frac{a+b}{2}, \quad i = 1, \ldots, k.$$

7.15. (a) Yes. A piecewise quadratic has $3n-3$ parameters to be determined. Interpolating the given data points gives $2n-2$ equations, and continuity of the first derivative gives $n-2$ additional equations, for a total of $3n-4$ equations. Thus, there are more unknowns than equations, leaving one free parameter, so such an interpolant is always possible.

(b) No, in general. Continuity of the second derivative would give an additional $n-2$ equations beyond those for part (a), for a total of $4n-6$ equations, which exceeds the number of parameters available whenever $n > 3$. Thus, such an interpolant is possible only for $n \leq 3$.

7.16. In keeping with the recursive definition of the B-spline functions B_i^k, all proofs proceed by induction on k.

Property 1:

Base case: By definition, $B_i^0(t) = 0$ for $t < t_i$ or $t > t_{i+1}$.

Inductive hypothesis: $B_i^{k-1}(t) = 0$ for $t < t_i$ or $t > t_{i+k}$.

From the inductive hypothesis, we have

$$\begin{aligned} B_i^{k-1}(t) &= 0 \text{ for } t < t_i \text{ or } t > t_{i+k}, \\ B_{i+1}^{k-1}(t) &= 0 \text{ for } t < t_{i+1} \text{ or } t > t_{i+k+1}, \end{aligned}$$

and hence

$$B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t) = 0 \text{ for } t < t_i \text{ or } t > t_{i+k+1}.$$

Property 2:

Base case: By definition, $B_i^0(t) = 1 > 0$ for $t_i < t < t_{i+1}$.

Inductive hypothesis: $B_i^{k-1}(t) > 0$ for $t_i < t < t_{i+k}$.

From the inductive hypothesis, we have

$$\begin{aligned} v_i^k(t) > 0 \text{ for } t > t_i &\Rightarrow v_i^k(t)B_i^{k-1}(t) > 0 \text{ for } t_i < t < t_{i+k}, \\ (1 - v_{i+1}^k(t)) > 0 \text{ for } t < t_{i+k} &\Rightarrow (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t) > 0 \text{ for } t_{i+1} < t < t_{i+k+1}, \end{aligned}$$

and hence

$$B_i^k(t) = v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t) > 0 \text{ for } t_i < t < t_{i+k+1}.$$

Property 3:

Base case: By definition, $\sum_{i=-\infty}^{\infty} B_i^0(t) = 1$.

Inductive hypothesis: $\sum_{i=-\infty}^{\infty} B_i^{k-1}(t) = 1$.

From the inductive hypothesis, we have

$$\begin{aligned} \sum_{i=-\infty}^{\infty} B_i^k(t) &= \sum_{i=-\infty}^{\infty} \left(v_i^k(t)B_i^{k-1}(t) + (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t)\right) \\ &= \sum_{i=-\infty}^{\infty} v_i^k(t)B_i^{k-1}(t) + \sum_{i=-\infty}^{\infty} (1 - v_{i+1}^k(t))B_{i+1}^{k-1}(t) \\ &= \sum_{i=-\infty}^{\infty} v_i^k(t)B_i^{k-1}(t) + \sum_{i=-\infty}^{\infty} B_{i+1}^{k-1}(t) - \sum_{i=-\infty}^{\infty} v_{i+1}^k(t)B_{i+1}^{k-1}(t) \\ &= \sum_{i=-\infty}^{\infty} v_i^k(t)B_i^{k-1}(t) \; + \; 1 \; - \sum_{i=-\infty}^{\infty} v_i^k(t)B_i^{k-1}(t) \; = \; 1\,. \end{aligned}$$

Property 4:

Base case: From its definition, B_i^1 is continuous.

Inductive hypothesis: B_i^{k-1} is $k-2$ times continuously differentiable.

As is easily established by a separate induction, for $k \geq 2$ we have

$$\frac{d}{dx} B_i^k(t) = \frac{k}{t_{i+k} - t_i} B_i^{k-1}(t) - \frac{k}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(t).$$

The derivative of B_i^k is therefore a linear combination of B_i^{k-1} and B_{i+1}^{k-1}, which by the inductive hypothesis are both continuously differentiable $k-2$ times. Thus, B_i^k is continuously differentiable $k-1$ times.

Property 5:

Base case: B_i^1 is obviously linearly independent.

Inductive hypothesis: $B_{1-k}^{k-1}, \ldots, B_{n-1}^{k-1}$ are linearly independent.

If f is a linear combination of $B_{1-k}^k, \ldots, B_{n-1}^k$ such that $f = 0$, then we would also have $f' = 0$. But we saw in the proof of Property 4 that the derivative of each B_i^k is a linear combination of B_i^{k-1} and B_{i+1}^{k-1}. Since $B_{1-k}^{k-1}, \ldots, B_{n-1}^{k-1}$ are linearly independent by the inductive hypothesis, this implies that there can be no such f that is a nontrivial linear combination of $B_{1-k}^k, \ldots, B_{n-1}^k$, and hence the latter are linearly independent.

Property 6:

This follows from the linear independence of $B_{1-k}^k, \ldots, B_{n-1}^k$ and the dimensionality of the spaces involved.

Computer Problems

7.1.
```
function cp07_01 % evaluate polynomial using Horner's method
x = [4 3 2]; t = [1; 2], a = 1; b = 2;
[p, d, I] = horner_eval(x, t, a, b)

function [p, d, I] = horner_eval(x, t, a, b)
[m,n] = size(t); nc = length(x); p = x(nc)*ones(m,n); d = zeros(m,n);
for i=nc-1:-1:1 % compute polynomial value p and derivative value d
  d = p+t.*d; p = x(i)+t.*p;
end
if nargin > 2 % compute integral value I if requested
  [m,n] = size(a); Ia = (x(nc)/nc)*ones(m,n); Ib = (x(nc)/nc)*ones(m,n);
  for i=nc-1:-1:1
    Ia = (x(i)/i)+a.*Ia; Ib = (x(i)/i)+b.*Ib;
  end
  Ia = a.*Ia; Ib = b.*Ib; I = Ib-Ia;
end
```

7.2.
```
function cp07_02 % Newton polynomial interpolation
t = [-2; 0; 1]; y = [-27; -1; 0];
disp('(a) Coefficients of Newton interpolant:'); x = Newton_interp(t, y)
disp('Value of Newton interpolant at t = 2:'); f = Newton_eval(x, t, 2)
disp('(b) Adding second data point:'); x = Newton_addpt(-27, -2, -27, 0, -1)
```

```
disp('(c) Recursive Newton interpolant:'); x = recursive_Newton_interp(t, y)

function [x] = Newton_interp(t, y) % compute Newton interpolant
n = length(t); A = zeros(n,n); A(:,1) = ones(n,1);
for i = 2:n
  A(i:n,i) = (t(i:n)-t(i-1)).*A(i:n,i-1);
end; x = A\y;

function [f] = Newton_eval(x, t, t_out) % evaluate Newton interpolant
n = length(x); f = x(n);
for i = n-1:-1:1
  f = x(i)+f.*(t_out-t(i));
end

function [x, t, y] = Newton_addpt(x, t, y, t_new, y_new) % add new point to
if length(x) == 0                                        % interpolant
  x = y_new; t = t_new; y = y_new;
else
  p = Newton_eval(x, t, t_new); Pi = 1;
  for k = 1:length(t)
    Pi = Pi*(t_new-t(k));
  end
  x = [x; (y_new-p)/Pi]; t = [t; t_new]; y = [y; y_new];
end

function [x] = recursive_Newton_interp(t, y) % compute interpolant recursively
n = length(t);
if n > 1
  x = recursive_Newton_interp(t(1:n-1), y(1:n-1));
  [x, t, y] = Newton_addpt(x, t(1:n-1), y(1:n-1), t(n), y(n));
else
  x = y;
end
```

7.3.
```
function cp07_03 % natural spline interpolant in Example 7.6
t = [-2; 0; 1]; y = [-27; -1; 0];
A = [1 t(1) t(1)^2 t(1)^3 0 0 0 0; 1 t(2) t(2)^2 t(2)^3 0 0 0 0;
  0 0 0 0 1 t(2) t(2)^2 t(2)^3; 0 0 0 0 1 t(3) t(3)^2 t(3)^3;
  0 1 2*t(2) 3*t(2)^2 0 -1 -2*t(2) -3*t(2)^2; 0 0 2 6*t(2) 0 0 -2 -6*t(2);
  0 0 2 6*t(1) 0 0 0 0; 0 0 0 0 0 0 2 6*t(3)];
x = A\[y(1); y(2); y(2); y(3); zeros(4,1)]; alpha = x(1:4), beta = x(5:8)
ts = linspace(t(1),t(2),20)'; d2 = 2*alpha(3)+ts.*(6*alpha(4));
ds = alpha(2)+ts.*(2*alpha(3)+ts.*(3*alpha(4)));
ys = alpha(1)+ts.*(alpha(2)+ts.*(alpha(3)+ts.*alpha(4)));
plot(t, y, 'ko', ts, ys, 'b-', ts, ds, 'r--', ts, d2, 'm-.'); hold on;
ts = linspace(t(2),t(3),20)'; d2 = 2*beta(3)+ts.*(6*beta(4));
ds = beta(2)+ts.*(2*beta(3)+ts.*(3*beta(4)));
ys = beta(1)+ts.*(beta(2)+ts.*(beta(3)+ts.*beta(4)));
plot(ts, ys, 'b-', ts, ds, 'r--', ts, d2, 'm-.'); xlabel('t'); ylabel('y');
legend('data points', 'spline', '1st deriv.', '2nd deriv.');
```

```
title('Computer Problem 7.3 -- Spline Function and Derivatives');
```

7.4.
```
function cp07_04 % interpolation of Runge's function
runge = inline('1./(1+25*t.^2)');
for n = 11:10:21
  t = linspace(-1, 1, n)'; y = runge(t); p = polyfit(t, y, n-1);
  pp = spline(t, y); ts = linspace(-1, 1, 100); yr = runge(ts);
  yp = polyval(p, ts); ys = ppval(pp, ts); figure; hold on;
  plot(t, y, 'ko', ts, yr, 'b-', ts, yp, 'r--', ts, ys, 'm:');
  text(-0.5, n/10, sprintf('n = %d', n)); xlabel('t'); ylabel('y');
  title('Computer Problem 7.4 -- Interpolants to Runge''s Function');
  legend('data points', 'Runge''s function', 'polynomial', 'spline', 0);
end; axis([-1, 1, -5, 5]);
```

7.5.
```
function cp07_05 % interpolation of experimental data
t = [0; 0.5; 1; 6; 7; 9]; y = [0; 1.6; 2; 2; 1.5; 0];
p = polyfit(t, y, 5); pp = spline(t, y); ts = linspace(0, 9, 50);
yp = polyval(p, ts); ys = ppval(pp, ts);
plot(t, y, 'ko', ts, yp, 'b-', ts, ys, 'r-.', t, y, 'k:');
xlabel('t'); ylabel('y'); axis([0, 9, -1, 3]);
title('Computer Problem 7.5 -- Interpolants to Experimental Data');
legend('data points', 'polynomial', 'spline', 'piecewise linear', 0);
disp('(c) Polynomial gives unreasonable values between data points because');
disp('    it necessarily oscillates.  Spline gives somewhat more reasonable');
disp('    values between data points, but it overshoots to achieve smoothness.');
disp('(d) Piecewise linear interpolation may be better in this case because');
disp('    it yields more reasonable values between data points, although it');
disp('    is less smooth.');
```

7.6.
```
function cp07_06 % interpolation of square root function
t = [0; 1; 4; 9; 16; 25; 36; 49; 64]; y = [0:8]';
p = polyfit(t,y,8); pp = spline(t,y);
ts = linspace(0,64,100)'; yr = sqrt(ts); yp = polyval(p,ts); ys = ppval(pp,ts);
figure; hold on; plot(t,y,'ko',ts,yr,'k-',ts,yp,'b--',ts,ys,'r-.');
title('Computer Problem 7.6(a)-(c) -- Interpolants to Square Root');
xlabel('t'); ylabel('y'); axis([0,64,0,8]);
legend('data points','square root','polynomial','spline',0);
ts = linspace(0,1,100)'; yr = sqrt(ts); yp = polyval(p,ts); ys = ppval(pp,ts);
figure; plot(t(1:2),y(1:2),'ko',ts,yr,'k-',ts,yp,'b--',ts,ys,'r-.');
title('Computer Problem 7.6(d) -- Interpolants to Square Root');
xlabel('t'); ylabel('y'); axis([0,1,0,1]);
legend('data points','square root','polynomial','spline',0);
```

7.7.
```
function cp07_06 % interpolation of gamma function
t = [1:5]'; y = [1; 1; 2; 6; 24]; p = polyfit(t,y,8); pp = spline(t,y);
ts = linspace(0,5,100)'; yr = gamma(ts); yp = polyval(p,ts); ys = ppval(pp,ts);
figure; hold on; plot(t,y,'ko',ts,yr,'k-',ts,yp,'b--',ts,ys,'r-.');
title('Computer Problem 7.7(a)-(c) -- Interpolants to Gamma Function'); xlabel('t');
ylabel('y'); legend('data points','gamma function','polynomial','spline',2);
ts = linspace(1,2,100)'; yr = gamma(ts); yp = polyval(p,ts); ys = ppval(pp,ts);
figure; plot(t(1:2),y(1:2),'ko',ts,yr,'k-',ts,yp,'b--',ts,ys,'r-.');
```

```
title('Computer Problem 7.7(d) -- Interpolants to Gamma Function'); xlabel('t');
ylabel('y'); legend('data points','gamma function','polynomial','spline',4);
```

7.8.
```
function cp07_08 % interpolation of population data
t = [1900:10:1980]'; disp('(a) Conditioning of basis matrices:');
cond_A1 = cond(vander(t)), cond_A2 = cond(vander(t-1900))
cond_A3 = cond(vander(t-1940)), cond_A4 = cond(vander((t-1940)/40))
y = [76212168; 92228496; 106021537; 123202624; 132164569; 151325798; ...
179323175; 203302031; 226542199]; xp = fliplr(vander((t-1940)/40))\y;
y2 = [76000000; 92000000; 106000000; 123000000; 132000000; 151000000; ...
  179000000; 203000000; 227000000]; xp2 = fliplr(vander((t-1940)/40))\y2;
ts = linspace(1900,1980,81)'; tic; yp = heval(xp,ts); time_p = toc;
xh = pchip(t,y); tic; yh = ppval(xh,ts); time_h = toc;
xs = spline(t,y); tic; ys = ppval(xs,ts); time_s = toc; figure;
plot(t,y,'ko',ts,yp,'k-',ts,yh,'b--',ts,ys,'r-.'); xlabel('t'); ylabel('y');
legend('data points','polynomial','Hermite cubic','cubic spline',2);
title('Computer Problem 7.8(b)-(d) -- Interpolants to Population Data');
ts = linspace(1980,1990,11)'; yp = heval(xp,ts);
yh = pchip(t,y,ts); ys = spline(t,y,ts); figure;
plot([t(9);1990],[y(9);248709873],'ko',ts,yp,'k-',ts,yh,'b--',ts,ys,'r-.');
xlabel('t'); ylabel('y');
legend('data points','polynomial','Hermite cubic','cubic spline',3);
title('Computer Problem 7.8(e) -- Extrapolation of Population Data');
ts = linspace(1900,1980,81)'; tic; yl = leval(t,y,ts); time_l = toc; figure;
plot(t,y,'ko',ts,yl,'k-'); xlabel('t'); ylabel('y');
legend('data points','Lagrange polynomial',2);
title('Computer Problem 7.8(f) -- Lagrange Interpolant to Pop. Data');
xn = Newton_interp(t,y); tic; yn = Newton_eval(xn,t,ts); time_n = toc;
ts = linspace(1900,1990,91)'; yn = Newton_eval(xn,t,ts);
[xn2, t, y] = Newton_addpt(xn,t,y,1990,248709873); yn2 = Newton_eval(xn2,t,ts);
figure; plot(t,y,'ko',ts,yn,'b--',ts,yn2,'r-.'); xlabel('t'); ylabel('y');
legend('data points','deg. 8 polynomial','deg. 9 polynomial',2);
title('Computer Problem 7.8(g) -- Newton Interpolants to Pop. Data');
disp('(f) Time to evaluate interpolating polynomial at yearly intervals:');
fprintf('Monomial basis: %g\n', time_p); fprintf('Hermite cubic: %g\n', time_h);
fprintf('Cubic spline: %g\n', time_s); fprintf('Lagrange basis: %g\n', time_l);
fprintf('Newton basis: %g\n', time_n); disp(' ');
disp('(h) Polynomial coefficients for original data:'); xp
disp('Polynomial coefficients for rounded data:'); xp2

function [y] = heval(x, t) % evaluate polynomial using Horner's method
n = length(x); y = zeros(size(t));
for i=n:-1:1
  y = x(i)+((t-1940)./40).*y;
end;

function [yy] = leval(t, y, tt) % evaluate Lagrange interpolant
n = length(t); yy = zeros(size(tt));
for i = 1:n
  z = ones(size(tt));
```

```
  for j = 1:n
    if i ~= j, z = z.*(tt-t(j))/(t(i)-t(j)); end
  end
  yy = yy+z*y(i);
end

function [x] = Newton_interp(t, y) % compute Newton interpolant
n = length(t); A = zeros(n,n); A(:,1) = ones(n,1);
for i = 2:n
  A(i:n,i) = (t(i:n)-t(i-1)).*A(i:n,i-1);
end; x = A\y;

function [f] = Newton_eval(x, t, t_out) % evaluate Newton interpolant
n = length(x); f = x(n);
for i = n-1:-1:1
  f = x(i)+f.*(t_out-t(i));
end

function [x, t, y] = Newton_addpt(x, t, y, t_new, y_new) % add point to
if length(x) == 0                                        % interpolant
  x = y_new; t = t_new; y = y_new;
else
  p = Newton_eval(x, t, t_new); Pi = 1;
  for k = 1:length(t)
    Pi = Pi*(t_new-t(k));
  end
  x = [x; (y_new-p)/Pi]; t = [t; t_new]; y = [y; y_new];
end
```

Chapter 8

Numerical Integration and Differentiation

Exercises

8.1. (a)

$$M_f = (1-0)f(1/2) = 1(1/2)^3 = 1/8,$$
$$T_f = (1-0)(f(0)+f(1))/2 = (0^3+1^3)/2 = 1/2.$$

(b)

$$E \approx (T-M)/3 = (1/2 - 1/8)/3 = 1/8,$$

so

$$E_M \approx 1/8 \quad \text{and} \quad E_T \approx -1/4.$$

(c)

$$S = 2M/3 + T/3 = 2(1/8)/3 + (1/2)/3 = 1/12 + 1/6 = 1/4.$$

(d) Yes, because Simpson's rule integrates all polynomials of degree ≤ 3 exactly.

8.2. (a) For $h = 1$,

$$M_1(f) = 1(0.5)^3 = 0.125.$$

For $h = 0.5$,

$$M_2(f) = 0.5\left((0.25)^3 + (0.75)^3\right) = 0.21875.$$

(b)

$$F(0) = F(h) + \frac{F(h) - F(h/2)}{2^{-2} - 1} = 0.125 + \frac{0.125 - 0.21875}{0.25 - 1} = 0.25$$

(c) Romberg integration gives the exact answer since it increases the degree of the error term, thereby attaining third-order accuracy in this instance, which implies exact results for cubic polynomials.

8.3. Yes, because an n-point interpolatory quadrature rule integrates all polynomials of degree $\leq n-1$ exactly, so in particular, it integrates the constant function $f(x) = 1$ exactly. Thus, we have

$$1 = \int_0^1 1\,dx = \sum_{i=1}^{n} w_i \cdot 1 = \sum_{i=1}^{n} w_i.$$

8.4. For $m = (a+b)/2$,

$$\int_a^b f(x)\,dx = \int_a^b \sum_{i=0}^{\infty} \frac{f^{(i)}(m)}{i!}\,(x-m)^i\,dx = \sum_{i=0}^{\infty} \frac{f^{(i)}(m)}{(i+1)!}\,(x-m)^{i+1}\Big|_{x=a}^{x=b}.$$

Substituting for m and simplifying, we have

$$\int_a^b f(x)\,dx = \frac{1}{2}\left(\sum_{i=0}^{\infty} \frac{f^{(i)}(m)}{(i+1)!}\,(a-b)^{i+1} - \sum_{i=0}^{\infty} \frac{f^{(i)}(m)}{(i+1)!}\,(b-a)^{i+1}\right).$$

Since $(b-a)^{i+1} = (a-b)^{i+1}$ for odd i, and $(b-a)^{i+1} = -(a-b)^{i+1}$ for even i, we have

$$\int_a^b f(x)\,dx = \sum_{i=0}^{\infty} \frac{f^{(2i)}(m)}{(2i+2)!}\,(b-a)^{2i+1}.$$

8.5. (a) In one subinterval $[a,b]$, we have

$$\begin{aligned} M(f) &= I(f) - \frac{f''(m)}{24}(b-a)^3 + \mathcal{O}((b-a)^5), \\ T(f) &= I(f) + \frac{f''(m)}{8}(b-a)^3 + \mathcal{O}((b-a)^5). \end{aligned}$$

Since $f''(x) \geq 0$, this implies that in each subinterval we have

$$M(f) \leq I(f) \leq T(f).$$

Summing over all the subintervals shows that the composite midpoint and trapezoid rules satisfy the bracketing property

$$M_k(f) \leq \int_a^b f(x)\,dx \leq T_k(f).$$

(b) If f is convex on an interval $[a,b]$, then by definition

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y)$$

for any $\alpha \in (0,1)$. Taking $\alpha = 1/2$, we thus have

$$f((x+y)/2) \leq (f(x) + f(y))/2,$$

which shows that $M(f) \leq T(f)$ on $[a,b]$. More generally, taking arbitrary $\alpha \in (0,1)$ shows that the graph of f lies on or below the line segment connecting $f(a)$ and

$f(b)$, which shows that $I(f) \le T(f)$. Similarly, the value of f between any two points is always less than or equal to the weighted average value of the function at those two points, which shows that $M(f) \le I(f)$. Thus, the bracketing property $M(f) \le I(f) \le T(f)$ is satisfied.

8.6. Given a set of nodes x_i, $i = 1, \ldots, n$, the Lagrange polynomial interpolating the corresponding values of the integrand function f is

$$p(x) = \sum_{i=1}^{n} f(x_i)\, \ell_i(x).$$

We therefore have

$$\int_a^b p(x)\, dx = \int_a^b \sum_{i=1}^{n} f(x_i)\, \ell_i(x)\, dx = \sum_{i=1}^{n} f(x_i) \int_a^b \ell_i(x)\, dx,$$

and thus, by definition, the weights w_i are given by

$$w_i = \int_a^b \ell_i(x)\, dx.$$

8.7. The nodes in a 2-point, open Newton-Cotes rule are $x_1 = a + (b-a)/3 = (2a+b)/3$ and $x_2 = a + 2(b-a)/3 = (a+2b)/3$. The first two moment equations are

$$\begin{aligned} w_1 \cdot 1 + w_2 \cdot 1 = w_1 + w_2 &= b - a \\ w_1 \cdot x_1 + w_2 \cdot x_2 = w_1 \cdot (2a+b)/3 + w_2 \cdot (a+2b)/3 &= (b^2 - a^2)/2. \end{aligned}$$

Solving this system, we obtain the weights $w_1 = w_2 = (b-a)/2$. By construction, this rule has degree one, i.e., it integrates all polynomials of degree ≤ 1 exactly.

8.8. (a) $n = 10$. (b) $n = 9$. (c) $n = 4$. (d) $n = 3$.

8.9. (a) Consider the polynomial $q_k(x) = (x - x_1)(x - x_2) \cdots (x - x_k)$ of degree $k \le n$, where x_i, $i = 1, \ldots, k$, are the roots of p in (a, b). By its definition, q_k has a sign change in (a, b) whenever p does, and therefore we have

$$\int_a^b p(x) q_k(x)\, dx \neq 0.$$

But by assumption, p is orthogonal to any polynomial of degree less than n. Hence, we must have $k = n$, i.e., all the roots of p are real and lie in (a, b). Now suppose that some root, say x_j, is not simple. Then $r(x) = p(x)/(x - x_j)^2$ is a polynomial of degree $n - 2$, and $p(x)r(x) = (p(x)/(x - x_j))^2$, so that

$$\int_a^b p(x) r(x)\, dx > 0.$$

But this is impossible, since p is orthogonal to any polynomial of degree less than n. Thus, the roots of p are all simple. (b) Let f be any polynomial of degree $2n-1$ or less. Then f can be written in the form

$$f(x) = p(x)q(x) + r(x),$$

where q and r are the quotient and remainder polynomials, respectively, both of degree $n-1$ or less. Then we have

$$\int_a^b f(x)\,dx = \int_a^b p(x)q(x)\,dx + \int_a^b r(x)\,dx.$$

The first of the two integrals on the right is zero because p is orthogonal to any polynomial of degree less than n, and the second integral is computed exactly by the interpolatory quadrature rule. Thus, the quadrature rule is exact for f, showing that the quadrature rule is of degree at least $2n-1$. The degree does not exceed $2n-1$, however, because p^2 is a polynomial of degree $2n$ that is not integrated exactly by the quadrature rule, since its integral is necessarily positive, while the quadrature rule gives a result of zero, since p^2 is zero at all of its nodes.

8.10. (a) The first four moment equations are

$$\begin{aligned}
w(1+1+1) &= \int_{-1}^{1} 1\,dx &&= 2,\\
w(x_1+x_2+x_3) &= \int_{-1}^{1} x\,dx &&= 0,\\
w(x_1^2+x_2^2+x_3^2) &= \int_{-1}^{1} x^2\,dx &&= \frac{2}{3},\\
w(x_1^3+x_2^3+x_3^3) &= \int_{-1}^{1} x^3\,dx &&= 0.
\end{aligned}$$

Solving this nonlinear system, we obtain

$$w = 2/3, \quad x_1 = -1/\sqrt{2}, \quad x_2 = 0, \quad x_3 = 1/\sqrt{2}.$$

(b) By construction, the degree of this quadrature rule is three.

8.11. (a) The error for the trapezoid rule on an interval $[a,b]$ is given by

$$E_T \approx 2\,\frac{f''(m)}{24}(b-a)^3 = \frac{f''(m)}{12}(b-a)^3,$$

where $m = (a+b)/2$, and the error for Simpson's rule is given by

$$E_S \approx \frac{2}{3}\,\frac{f^{(4)}(m)}{1920}(b-a)^5 = \frac{f^{(4)}(m)}{2880}(b-a)^5.$$

Thus, provided $f''(m)$ and $f^{(4)}(m)$ do not differ substantially in magnitude and $b-a<1$, Simpson's rule is the more accurate. Halving the interval width for the

trapezoid rule reduces the error by a factor of 1/8 in each subinterval, for an overall reduction in the error by a factor of 1/4. This gain in accuracy is not enough to offset the superior accuracy of Simpson's rule on the original interval. (*b*) A similar error estimate for the five-point closed Newton-Cotes quadrature rule is given by

$$E \approx \frac{8f^{(6)}(m)}{945}(b-a)^7.$$

Thus, provided $f^{(4)}(m)$ and $f^{(6)}(m)$ do not differ substantially in magnitude and $b - a < 1$, the five-point Newton-Cotes rule is the more accurate. Halving the interval width for the trapezoid rule reduces the error by a factor of 1/32 in each subinterval, for an overall reduction in the error by a factor of 1/16. This gain in accuracy is not enough to offset the superior accuracy of the five-point Newton-Cotes rule on the original interval.

(*c*) For an n-point interpolatory quadrature rule Q_n, we have the general error bound

$$E \leq \frac{1}{4}\, h^{n+1}\, \|f^{(n)}\|_\infty.$$

The comparable bound for the rule Q_{2n-1} on an interval of twice the length is

$$E \leq \frac{1}{4}\, (2h)^{2n}\, \|f^{(2n-1)}\|_\infty.$$

Again, provided the relevant derivatives of f are well behaved, we see that the higher-degree rule is more accurate than the lower-degree rule applied twice on intervals of half the size.

8.12. Averaging the first-order accurate forward and backward difference formulas, we obtain

$$\frac{1}{2}\left(\frac{f(x+h)-f(x)}{h} + \frac{f(x)-f(x-h)}{h}\right) = \frac{f(x+h)-f(x-h)}{2h},$$

which is the centered difference formula we already know to be second-order accurate from the analysis in Section 8.6.1.

8.13. In the Taylor series expansion

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(x)}{6}h^3 + \cdots,$$

we solve for $f'(x)$ to obtain

$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{f''(x)}{2}h - \frac{f'''(x)}{6}h^2 + \cdots.$$

Similarly, in the Taylor series expansion

$$\begin{aligned} f(x+2h) &= f(x) + f'(x)(2h) + \frac{f''(x)}{2}(2h)^2 + \frac{f'''(x)}{6}(2h)^3 + \cdots \\ &= f(x) + 2f'(x)h + 2f''(x)h^2 + \frac{4f'''(x)}{3}h^3 + \cdots, \end{aligned}$$

we solve for $f'(x)$ to obtain

$$f'(x) = \frac{f(x+2h) - f(x)}{2h} - f''(x)h - \frac{2f'''(x)}{3}h^2 + \cdots.$$

If we now subtract the second of these series for $f'(x)$ from twice the first, we obtain

$$f'(x) = \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h} + \frac{f'''(x)}{3}h^2 + \cdots,$$

so that

$$f'(x) \approx \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$$

is a second-order accurate, one-sided difference approximation to $f'(x)$.

8.14.

$$F(0) = F(h) + \frac{F(h) - F(h/2)}{2^{-1} - 1} = -0.8333 + \frac{-0.8333 + 0.9091}{0.5 - 1} = -0.9849.$$

8.15. (a) The power series expansion for the sine function is

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,$$

so

$$\begin{aligned} p_n &= n\left(\frac{\pi}{n} - \frac{\pi^3}{n^3 3!} + \frac{\pi^5}{n^5 5!} - \frac{\pi^7}{n^7 7!} + \cdots\right) = \pi - \frac{\pi^3}{n^2 3!} + \frac{\pi^5}{n^4 5!} - \frac{\pi^7}{n^6 7!} + \cdots \\ &= \pi - \frac{\pi^3}{3!}h^2 + \frac{\pi^5}{5!}h^4 - \frac{\pi^7}{7!}h^6 + \cdots, \end{aligned}$$

where $h = 1/n$, and hence we have $a_0 = \pi$. Similarly, the power series expansion for the tangent function is

$$\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^7}{315} + \cdots,$$

so

$$\begin{aligned} q_n &= n\left(\frac{\pi}{n} + \frac{\pi^3}{n^3 3} + \frac{2\pi^5}{n^5 15} + \frac{17\pi^7}{n^7 315} + \cdots\right) = \pi + \frac{\pi^3}{n^2 3} + \frac{2\pi^5}{n^4 15} + \frac{17\pi^7}{n^6 315} + \cdots \\ &= \pi + \frac{\pi^3}{3}h^2 + \frac{2\pi^5}{15}n^4 + \frac{17\pi^7}{315}n^6 + \cdots, \end{aligned}$$

and hence we have $b_0 = \pi$. (b) Using Richardson extrapolation with $p = 2$ and $q = 2$, we have

$$a_0 = F(1/6) + \frac{F(1/6) - F(1/12)}{2^{-2} - 1} = 3.0000 + \frac{3.0000 - 3.1058}{0.25 - 1} = 3.1411.$$

$$b_0 = F(1/6) + \frac{F(1/6) - F(1/12)}{2^{-2} - 1} = 3.4641 + \frac{3.4641 - 3.2154}{0.25 - 1} = 3.1325.$$

Computer Problems

8.1. function cp08_01 % numerical integration using several methods

```
f = inline('4./(1+x.^2)', 'x'); a = 0; b = 1; h_min = 1e-6;
k = 0; h = b-a; I_true = pi;
disp('(a),(b) Error in quadrature rules for given stepsize h:');
disp('    h      Midpoint  Trapezoid   Simpson    Romberg');
while h >= h_min, k = k+1;
  err_m = abs(Midpoint(f,a,b,h)-I_true); err_t = abs(Trapezoid(f,a,b,h)-I_true);
  err_s = abs(Simpson(f,a,b,h)-I_true); err_r = abs(Romberg(f,a,b,h)-I_true);
  fprintf('%8.1e %10.3e %10.3e %10.3e %10.3e\n', h, err_m, err_t, err_s, err_r);
  h = h/10;
end; disp(' ');
disp('(c) Error and function evaluations for given tolerance tol:');
disp('                 quad            quadl');
disp('   tol       err     evals    err     evals');
for k = 1:9
  tol = 10^(-k); [Q, fcnt] = quad(f,a,b,tol); err_q = abs(Q-I_true);
  [Ql, fcntl] = quadl(f,a,b,tol); err_ql = abs(Ql-I_true);
  fprintf('%8.1e %10.3e %5d %10.3e %5d\n', tol, err_q, fcnt, err_ql, fcntl);
end; disp(' ');
disp('(d) Error in Monte Carlo integration using n random evaluation points:');
disp('    n        err     1/(sqrt(n))'); n = 1;
for k = 1:7
  err_mc = abs(MonteCarlo(f,a,b,n)-I_true);
  fprintf('%8d %10.3e %10.3e\n', n, err_mc, 1/sqrt(n)); n = n*10;
end; disp(' ');

function [I] = Midpoint(f, a, b, h) % composite midpoint quadrature
x = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,x));

function [I] = Trapezoid(f, a, b, h) % composite trapezoid quadrature
x = [a:h:b]'; I = h*(sum(feval(f,x))-(feval(f,a)+feval(f,b))/2);

function [I] = Simpson(f, a, b, h) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h)+Trapezoid(f,a,b,h))/3;

function [I] = Romberg(f, a, b, h_min) % Romberg integration
k = 1; h = b-a; T(k,k) = Trapezoid(f,a,b,h);
while h > h_min
  k = k+1; h = h/2; T(k,1) = Trapezoid(f,a,b,h);
  for j = 2:k
    T(k,j) = (4^(j-1)*T(k,j-1)-T(k-1,j-1))/(4^(j-1)-1);
  end
end; I = T(k,k);

function [I] = MonteCarlo(f, a, b, n) % Monte Carlo integration
x = a+(b-a)*rand(n,1); I = (b-a)*sum(feval(f,x))/n;
```

8.2. function cp08_02 % numerical integration using several methods

```
f = inline('sqrt(x).*log(x)', 'x'); a = 1e-10; b = 1; h_min = 1e-7;
```

```
k = 0; h = b-a; I_true = -4/9;
disp('(a),(b) Error in quadrature rules for given stepsize h:');
disp('    h      Midpoint  Trapezoid   Simpson    Romberg');
while h >= h_min, k = k+1;
  err_m = abs(Midpoint(f,a,b,h)-I_true); err_t = abs(Trapezoid(f,a,b,h)-I_true);
  err_s = abs(Simpson(f,a,b,h)-I_true); err_r = abs(Romberg(f,a,b,h)-I_true);
  fprintf('%8.1e %10.3e %10.3e %10.3e %10.3e\n', h, err_m, err_t, err_s, err_r);
  h = h/10;
end; disp(' ');
disp('(c) Error and function evaluations for given tolerance tol:');
disp('                  quad            quadl');
disp('   tol       err     evals    err     evals');
for k = 1:9
  tol = 10^(-k);
  [Q, fcnt] = quad(f,a,b,tol); err_q = abs(Q-I_true);
  [Ql, fcntl] = quadl(f,a,b,tol); err_ql = abs(Ql-I_true);
  fprintf('%8.1e %10.3e %5d %10.3e %5d\n', tol, err_q, fcnt, err_ql, fcntl);
end; disp(' ');
disp('(d) Error in Monte Carlo integration using n random evaluation points:');
disp('    n        err     1/(sqrt(n))'); n = 1;
for k = 1:7
  err_mc = abs(MonteCarlo(f,a,b,n)-I_true);
  fprintf('%8d %10.3e %10.3e\n', n, err_mc, 1/sqrt(n));
  n = n*10;
end; disp(' ');

function [I] = Midpoint(f, a, b, h) % composite midpoint quadrature
x = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,x));

function [I] = Trapezoid(f, a, b, h) % composite trapezoid quadrature
x = [a:h:b]'; I = h*(sum(feval(f,x))-(feval(f,a)+feval(f,b))/2);

function [I] = Simpson(f, a, b, h) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h)+Trapezoid(f,a,b,h))/3;

function [I] = Romberg(f, a, b, h_min) % Romberg integration
k = 1; h = b-a; T(k,k) = Trapezoid(f,a,b,h);
while h > h_min
  k = k+1; h = h/2; T(k,1) = Trapezoid(f,a,b,h);
  for j = 2:k
    T(k,j) = (4^(j-1)*T(k,j-1)-T(k-1,j-1))/(4^(j-1)-1);
  end
end; I = T(k,k);

function [I] = MonteCarlo(f, a, b, n) % Monte Carlo integration
x = a+(b-a)*rand(n,1); I = (b-a)*sum(feval(f,x))/n;
```

8.3. `function cp08_03 % numerical integration using several methods`

```
fs = {'cos(x)' '1./(1+100*x.^2)' 'sqrt(abs(x))'}; part = {'(a)' '(b)' '(c)'};
a = -1; b = 1; h_min = 1e-6;
```

```
for i = 1:3
f = inline(fs{i}, 'x');
disp([part{i} 'Integrand = ' fs{i}]); disp(' ');
k = 0; h = b-a;
disp('Results for given stepsize h:');
disp('    h        Midpoint        Trapezoid       Simpson         Romberg');
while h >= h_min, k = k+1;
  Q_m = Midpoint(f,a,b,h); Q_t = Trapezoid(f,a,b,h);
  Q_s = Simpson(f,a,b,h); Q_r = Romberg(f,a,b,h);
  fprintf('%8.1e %15.8e %15.8e %15.8e %15.8e\n', h, Q_m, Q_t, Q_s, Q_r); h = h/10;
end; disp(' ');
disp('Results and function evaluations for given tolerance tol:');
disp('   tol         quad       evals      quadl      evals');
for k = 1:9
  tol = 10^(-k);
  [Q, fcnt] = quad(f,a,b,tol);
  [Q_l, fcntl] = quadl(f,a,b,tol);
  fprintf('%8.1e %15.8e %5d %15.8e %5d\n', tol, Q, fcnt, Q_l, fcntl);
end; disp(' ');
end

function [I] = Midpoint(f, a, b, h) % composite midpoint quadrature
x = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,x));

function [I] = Trapezoid(f, a, b, h) % composite trapezoid quadrature
x = [a:h:b]'; I = h*(sum(feval(f,x))-(feval(f,a)+feval(f,b))/2);

function [I] = Simpson(f, a, b, h) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h)+Trapezoid(f,a,b,h))/3;

function [I] = Romberg(f, a, b, h_min) % Romberg integration
k = 1; h = b-a; T(k,k) = Trapezoid(f,a,b,h);
while h > h_min
  k = k+1; h = h/2; T(k,1) = Trapezoid(f,a,b,h);
  for j = 2:k
    T(k,j) = (4^(j-1)*T(k,j-1)-T(k-1,j-1))/(4^(j-1)-1);
  end
end; I = T(k,k);
```

8.4.

```
function cp08_04 % numerical integration to test conjectures
fs = {'sqrt(x.^3)' '1./(1+10*x.^2)' ...
'(exp(-9*x.^2)+exp(-1024*(x-0.25).^2))/sqrt(pi)' '50./(pi*(2500*x.^2+1))' ...
'1./sqrt(abs(x))' '25*exp(-25*x)' 'log(x)'}; c = [0.4 0.4 0.2 0.5 26 1 -1];
a = [0 0 0 0 -9 0 1e-10]; b = [1 1 1 10 100 10 1];
disp('Truth or falsity of conjecture for given tolerance using quadl:');
disp('   tol   abcdefg');
for k = 1:7
  tol = 10^(-k);
  for i = 1:7
    f = inline(fs{i},'x');
```

```
    if abs(quadl(f,a(i),b(i),tol)-c(i)) <= tol, torf(i) = 'T';
    else torf(i) = 'F'; end
  end; fprintf('%8.1e %s\n', tol, torf);
end; disp(' ');
```

8.5. function cp08_05 % numerical integration of piecewise functions

```
fs = {'x>=0.3' '(x<(exp(1)-2))./(x+2)' '(x<0).*exp(x)+(x>=0).*exp(1-x)' ...
'(x<0.5).*exp(10*x)+(x>=0.5).*exp(10*(1-x))' ...
'(x<0.5).*sin(pi*x)+(x>=0.5).*sin(pi*x).^2'};
part = {'(a)' '(b)' '(c)' '(d)' '(e)'};
a = [0 0 -1 -1 0]; b = [1 1 2 1.5 1]; c = [0.3 exp(1)-2 0 0.5 0.5];
for i = 1:5
  f = inline(fs{i},'x');
  disp([part{i} ' Integral and function evaluations for given tolerance tol:']);
  disp('                  One call             Two calls');
  disp('   tol       integral     evals    integral     evals');
  for k = 1:7
    tol = 10^(-k);
    toll = tol*(c(i)-a(i))/(b(i)-a(i)); tolr = tol*(b(i)-c(i))/(b(i)-a(i));
    [Ql, fcntl] = quadl(f,a(i),c(i),toll); [Qr, fcntr] = quadl(f,c(i),b(i),tolr);
    Q2 = Ql+Qr; fcnt2 = fcntl+fcntr; [Q1, fcnt1] = quadl(f,a(i),b(i),tol);
    fprintf('%8.1e %15.8e %5d %15.8e %5d\n', tol, Q1, fcnt1, Q2, fcnt2);
  end; disp(' ');
end
```

8.6. function cp08_06 % evaluate integrals and recurrences

```
f = inline('x.^k.*exp(x)','x','k'); tol = 1e-7;
Ia = zeros(21,1); Ib = zeros(21,1); Ic = zeros(22,1);
Ia(1) = exp(-1)*quadl(f,0,1,tol,[],0); Ib(1) = 1-exp(-1); Ic(23) = 0;
disp('  k   (a) adapt. quad.     (b) forw. recurr.    (c) back recurr.');
tic; for k = 2:21
  Ia(k) = exp(-1)*quadl(f,0,1,tol,[],k-1);
end; time_quad = toc;
tic; for k = 2:21
  Ib(k) = 1-k*Ib(k-1);
end; time_forw = toc;
tic; for k = 23:-1:2
  Ic(k-1) = (1-Ic(k))/(k-1);
end; time_back = toc;
for k = 1:21
  fprintf('%3d %20.12e %20.12e %20.12e\n', k-1, Ia(k), Ib(k), Ic(k));
end; disp(' ');
disp('(d) Execution times:');
fprintf('    %20.12e %20.12e %20.12e\n', time_quad, time_forw, time_back);
```

8.7. function cp08_07 % compute surface area of ellipsoid by integration

```
f = inline('sqrt(1-K*x.^2)','x','K'); beta = 100; alpha = (3-2*sqrt(2))/beta;
K = beta*sqrt(1-alpha*beta); theta = acos(sqrt(K/beta));
I_true = pi*sqrt(alpha/K)*(pi+sin(2*theta)-2*theta);
disp('Error and function evaluations for given tolerance tol:');
disp('                 quad            quadl');
```

```
disp('   tol       err     evals    err     evals');
for k = 1:9
  tol = 10^(-k);
  [I, fcnt] = quad(f,0,1/sqrt(beta),tol,[],K);
  err = abs(4*pi*sqrt(alpha)*I-I_true);
  [Il, fcntl] = quadl(f,0,1/sqrt(beta),tol,[],K);
  err_l = abs(4*pi*sqrt(alpha)*Il-I_true);
  fprintf('%8.1e %10.3e %5d %10.3e %5d\n', tol, err, fcnt, err_l, fcntl);
end
```

8.8. `function cp08_08 % compute Fresnel integrals`

```
fc = inline('cos(pi*t.^2/2)','t'); fs = inline('sin(pi*t.^2/2)','t');
n = 200; x = linspace(0,5,n); C = zeros(n,1); S = zeros(n,1);
for i = 2:n
  C(i) = C(i-1)+quad(fc,x(i-1),x(i)); S(i) = S(i-1)+quad(fs,x(i-1),x(i));
end
plot(x,C,'b-',x,S,'r--'); xlabel('x'); ylabel('y'); legend('C(x)', 'S(x)');
title('Computer Problem 8.8 -- Fresnel Integrals');
```

8.9. `function cp08_09 % compute complete elliptic integral`

```
f = inline('1./sqrt(1-x.^2*sin(theta).^2)','theta','x');
n = 50; x = linspace(0,1,n); K = zeros(n,1); tol = 1e-7;
for i = 1:n-1
  K(i) = quad(f,0,pi/2,tol,[],x(i));
end; K(n) = Inf; plot(x,K,'b-'); xlabel('x'); ylabel('K(x)');
title('Computer Problem 8.9 -- Complete Elliptic Integral');
```

8.10. `function cp08_10 % evaluate integral for gamma function`

```
f = inline('t.^(x-1).*exp(-t)','t','x'); a = 0; b = 1e2; h_min = 1e-3;
x = [1 2 5 10];
for i = 1:4
fprintf('For x = %d,\n', x(i));
disp('(a) Error in quadrature rules for given stepsize h:');
disp('    h      Midpoint  Trapezoid   Simpson'); k = 0; h = b-a;
while h >= h_min, k = k+1;
  err_m = abs((Midpoint(f,a,b,h,x(i))-gamma(x(i)))/gamma(x(i)));
  err_t = abs((Trapezoid(f,a,b,h,x(i))-gamma(x(i)))/gamma(x(i)));
  err_s = abs((Simpson(f,a,b,h,x(i))-gamma(x(i)))/gamma(x(i)));
  fprintf('%8.1e %10.3e %10.3e %10.3e\n', h, err_m, err_t, err_s); h = h/10;
end; disp(' ');
disp('(b) Error and function evaluations for given tolerance tol:');
disp('                 quad            quadl');
disp('   tol       err     evals    err     evals');
for k = 1:7
  tol = 10^(-k);
  [Q, fcnt] = quad(f,a,b,tol,[],x(i)); err_q = abs((Q-gamma(x(i)))/gamma(x(i)));
  [Ql, fcntl] = quadl(f,a,b,tol,[],x(i)); err_ql = abs((Ql-gamma(x(i)))/gamma(x(i)));
  fprintf('%8.1e %10.3e %5d %10.3e %5d\n', tol, err_q, fcnt, err_ql, fcntl);
end; disp(' ');
disp('(c) Error in n-point Gauss-Laguerre quadrature:'); disp(' n     err');
for n = 2:5
```

```
  err_g = abs((GaussLaguerre(f,n,x(i))-gamma(x(i)))/gamma(x(i)));
  fprintf('%2d %10.3e\n', n, err_g);
end; disp(' ');
end

function [I] = Midpoint(f, a, b, h, x) % composite midpoint quadrature
t = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,t,x));

function [I] = Trapezoid(f, a, b, h, x) % composite trapezoid quadrature
t = [a:h:b]'; I = h*(sum(feval(f,t,x))-(feval(f,a,x)+feval(f,b,x))/2);

function [I] = Simpson(f, a, b, h, x) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h,x)+Trapezoid(f,a,b,h,x))/3;

function [I] = GaussLaguerre(f, n, x) % Gauss-Laguerre quadrature
switch n
  case 2
    t = [0.585786437627; 3.414213562373];
    w = [8.53553390593e-1; 1.46446609407e-1];
  case 3
    t = [0.415774556783; 2.294280360279; 6.289945082937];
    w = [7.11093009929e-1; 2.78517733569e-1; 1.03892565016e-2];
  case 4
    t = [0.322547689619; 1.745761101158; 4.536620296921; 9.395070912301];
    w = [6.03154104342e-1; 3.57418692438e-1; 3.88879085150e-2; 5.39294705561e-4];
  case 5
    t = [0.263560319718; 1.413403059107; 3.596425771041; 7.085810005859; ...
      12.640800844276]; w = [5.21755610583e-1; 3.98666811083e-1; ...
      7.59424496817e-2; 3.61175867992e-3; 2.33699723858e-5];
end; I = w'*(feval(f,t,x).*exp(t));
```

8.11.

```
function cp08_11 % evaluate Planck's integral for black body radiaton
f = inline('x.^3./(exp(x)-1)','x'); a = 1e-6; b = 5e1; h_min = 1e-3;
disp('Results for given stepsize h:');
disp('    h        Midpoint        Trapezoid       Simpson');
k = 0; h = b-a;
while h >= h_min, k = k+1;
  Q_m = Midpoint(f,a,b,h); Q_t = Trapezoid(f,a,b,h); Q_s = Simpson(f,a,b,h);
  fprintf('%8.1e %15.8e %15.8e %15.8ee\n', h, Q_m, Q_t, Q_s); h = h/10;
end; disp(' ');
disp('Results and function evaluations for given tolerance tol:');
disp('   tol         quad       evals      quadl      evals');
for k = 1:7
  tol = 10^(-k); [Q, fcnt] = quad(f,a,b,tol); [Ql, fcntl] = quadl(f,a,b,tol);
  fprintf('%8.1e %15.8e %5d %15.8e %5d\n', tol, Q, fcnt, Ql, fcntl);
end; disp(' ');
disp('Results for n-point Gauss-Laguerre quadrature:');
disp(' n     Integral');
for n = 2:5
  Q_g = GaussLaguerre(f,n); fprintf('%2d %15.8e\n', n, Q_g);
```

```
end; disp(' ');

function [I] = Midpoint(f, a, b, h) % composite midpoint quadrature
x = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,x));

function [I] = Trapezoid(f, a, b, h) % composite trapezoid quadrature
x = [a:h:b]'; I = h*(sum(feval(f,x))-(feval(f,a)+feval(f,b))/2);

function [I] = Simpson(f, a, b, h) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h)+Trapezoid(f,a,b,h))/3;

function [I] = GaussLaguerre(f, n) % Gauss-Laguerre quadrature
switch n
  case 2
    x = [0.585786437627; 3.414213562373];
    w = [8.53553390593e-1; 1.46446609407e-1];
  case 3
    x = [0.415774556783; 2.294280360279; 6.289945082937];
    w = [7.11093009929e-1; 2.78517733569e-1; 1.03892565016e-2];
  case 4
    x = [0.322547689619; 1.745761101158; 4.536620296921; 9.395070912301];
    w = [6.03154104342e-1; 3.57418692438e-1; 3.88879085150e-2; 5.39294705561e-4];
  case 5
    x = [0.263560319718; 1.413403059107; 3.596425771041; 7.085810005859; ...
      12.640800844276]; w = [5.21755610583e-1; 3.98666811083e-1; ...
      7.59424496817e-2; 3.61175867992e-3; 2.33699723858e-5];
end; I = w'*(feval(f,x).*exp(x));
```

8.12.

```
function cp08_12 % evaluate integral from -Inf to +Inf
f = inline('exp(-x.^2).*cos(x)','x'); a = -5e1; b = 5e1; h_min = 1e-3;
disp('(a) Results for given stepsize h:'); k = 0; h = b-a;
disp('    h        Midpoint        Trapezoid       Simpson');
while h >= h_min, k = k+1;
  Q_m = Midpoint(f,a,b,h); Q_t = Trapezoid(f,a,b,h); Q_s = Simpson(f,a,b,h);
  fprintf('%8.1e %15.8e %15.8e %15.8ee\n', h, Q_m, Q_t, Q_s); h = h/10;
end; disp(' ');
disp('(b) Results and function evaluations for given tolerance tol:');
disp('   tol         quad       evals      quadl      evals');
for k = 1:7
  tol = 10^(-k); [Q, fcnt] = quad(f,a,b,tol); [Ql, fcntl] = quadl(f,a,b,tol);
  fprintf('%8.1e %15.8e %5d %15.8e %5d\n', tol, Q, fcnt, Ql, fcntl);
end; disp(' ');
disp('(c) Results for n-point Gauss-Hermite quadrature:'); disp(' n     Integral');
for n = 2:5
  Q_g = GaussHermite(f,n); fprintf('%2d %15.8e\n', n, Q_g);
end; disp(' ');

function [I] = Midpoint(f, a, b, h) % composite midpoint quadrature
x = [a+h/2:h:b-h/2]'; I = h*sum(feval(f,x));
```

```
function [I] = Trapezoid(f, a, b, h) % composite trapezoid quadrature
x = [a:h:b]'; I = h*(sum(feval(f,x))-(feval(f,a)+feval(f,b))/2);

function [I] = Simpson(f, a, b, h) % composite Simpson quadrature
I = (2*Midpoint(f,a,b,h)+Trapezoid(f,a,b,h))/3;

function [I] = GaussHermite(f, n) % Gauss-Hermite quadrature
switch n
  case 2
    x = [-0.707106781186548; 0.707106781186548];
    w = [8.862269254528e-1; 8.862269254528e-1];
  case 3
    x = [-1.224744871391589; 0; 1.224744871391589];
    w = [2.954089751509e-1; 1.181635900604; 2.954089751509e-1];
  case 4
    x = [-1.650680123885785; -0.524647623275290; 0.524647623275290; ...
      1.650680123885785];
    w = [8.131283544725e-2; 8.049140900055e-1; 8.049140900055e-1; ...
      8.131283544725e-2];
  case 5
    x = [-2.020182870456086; -0.958572464613819; 0; 0.958572464613819; ...
      2.020182870456086];
    w = [1.995324205905e-2; 3.936193231522e-1; 9.453087204829e-1; ...
      3.936193231522e-1; 1.995324205905e-2];
end; I = w'*(feval(f,x).*exp(x.^2));
```

8.13.
```
function cp08_13 % evaluate double integral for electrostatic potential
f = inline('1./sqrt((xhat-x).^2+(yhat-y).^2)','x','y','xhat','yhat');
n = 10; xhat = linspace(2,10,n)'; Phi = zeros(n,n); tol = 1e-2;
for i = 1:n
  for j = 1:n
    Phi(i,j) = dblquad(f,-1,1,-1,1,tol,[],xhat(i),xhat(j));
  end
end; [x, y] = meshgrid(xhat); surf(x,y,Phi); view(135,30);
```

8.14.
```
function cp08_14 % evaluate double integral over two regions
I_a = quad(@innera,0,1), I_b = quad(@innerb,0,1)

function [f] = innera(y), n = length(y); f = zeros(n,1);
for i = 1:n
  f(i) = quad(inline('exp(-x.*y)','x','y'),0,1,[],[],y(i));
end

function [f] = innerb(y), n = length(y); f = zeros(n,1);
for i = 1:n
  f(i) = quad(inline('exp(-x.*y)','x','y'),0,sqrt(1-y(i)^2),[],[],y(i));
end
```

8.15.
```
function cp08_15 % automatic and adaptive quadrature
f = inline('4./(1+x.^2)','x'); % integral from Computer Problem 8.1
disp('(a) Automatic Simpson quadrature:'); I = autoquad(f,0,1,1e-6)
```

```
disp('(b) Adaptive Simpson quadrature:'); I = adaptquad(f,0,1,1e-6)

function [I] = autoquad(f, a, b, tol) % automatic quadrature
h = (b-a)/2; x = (a:h:b)'; y = feval(f,x); Q = h*(y(1)+2*y(2)+y(3))/6;
I = recurquad(f,a,b,y(1),y(3),y(2),Q,h,tol);

function [I] = recurquad(f, a, b, fa, fb, y, Q1, h, tol);
h = h/2; x = (a+h:2*h:b-h)'; z = feval(f,x);
Q2 = h*(fa+2*sum(y)+4*sum(z)+fb)/3; % composite Simpson's rule
if abs(Q2-Q1) < tol, I = Q2;
else I = recurquad(f,a,b,fa,fb,[y;z],Q2,h,tol);
end

function [I] = adaptquad(f, a, b, tol) % adaptive Simpson quadrature
x = [a; a+(b-a)/2; b]; y = feval(f,x); I = recurSimp(f,a,b,y(1),y(3),y(2),tol);

function [I] = recurSimp(f, a, b, fa, fb, fm, tol) % recursive Simpson
m = a+(b-a)/2; x = [a+(m-a)/2; m+(b-m)/2]; y = feval(f,x);
Q1 = (b-a)*(fa+4*fm+fb)/6; Q2 = (b-a)*(fa+4*y(1)+2*fm+4*y(2)+fb)/12;
if abs(Q2-Q1) < tol, I = Q2;
else I = recurSimp(f,a,m,fa,fm,y(1),tol)+recurSimp(f,m,b,fm,fb,y(2),tol);
end
```

8.16.

```
function cp08_16 % devise integrand for which adaptive quadrature is wrong
global points; points = [];
disp('Computed value of integral is'); I = quad(@f,0,1)
disp('for polynomial with coefficients'); p = poly(points)

function [y] = f(x)
global points; if isempty(points), points = x; else points = [points, x]; end
y = zeros(size(x));
```

8.17.

```
function cp08_17 % solve integral equation
K = inline('sqrt(s.^2+t.^2)','s','t'); f = inline('((s.^2+1).^1.5-s.^3)/3','s');
resid = inline('norm(y-A*x)','x','A','y');
err = inline('norm(([A; sqrt(mu)*eye(n)]\[y; zeros(n,1)])-t)','mu','A','y','t','n');
options = optimset('LargeScale', 'off', 'Display', 'off');
disp('       (a)          (b)        (c)         (d)         (e)         (f)');
disp('  n  err_norm    cond(A)   err_norm   err_norm   err_norm   err_norm');
tol = 1e-3; mu = 6.6e-5;
for n = 3:2:15
  h = 1/(n-1); s = linspace(0,1,n)'; t = s; y = f(s);
  w = 2*(2-mod((1:n)',2)); w(1) = 1; w(n) = 1; w = h*w/3;
  for j = 1:n
    A(1:n,j) = w(j)*K(s,t(j));
  end
  x = A\y;
  x_s = lls_rd(A,y,tol);
%  mu = fminbnd(err,0,1,[],A,y,t,n)
%  for k = 1:12
%    mu = 10^(-k);
```

```
%    x_r = [A; sqrt(mu)*eye(n)]\[y; zeros(n,1)];
%    ns(k) = norm(x_r); nr(k) = norm(y-A*x_r);
%  end
%  figure; loglog(ns,nr,'o'); xlabel('norm of soln'); ylabel('norm of resid');
%  mu = 6.6e-5;
  x_r = [A; sqrt(mu)*eye(n)]\[y; zeros(n,1)];
  x_n = fmincon(resid,0.5*ones(n,1),[],[],[],[],zeros(n,1),[],[],options,A,y);
  C = diag(ones(n-1,1),0)-diag(ones(n-2,1),1); C(n-1,n) = -1;
  x_m = fmincon(resid,0.5*ones(n,1),C,zeros(n-1,1),[],[],zeros(n,1),[],[],...
    options,A,y);
  figure; plot(t,t,'g-',t,x_s,'b--',t,x_r,'r-.',t,x_n,'k:',t,x_m,'m-');
  legend('true solution','truncated SVD','regularization','nonnegative',...
    'monotonic',2); xlabel('t'); ylabel('x'); text(0.475,1,['n = ' num2str(n)]);
  title('Computer Problem 8.17 -- Solution of Integral Equation');
  fprintf('%3d %10.3e %10.3e %10.3e %10.3e %10.3e %10.3e\n', n, ...
    norm(x-t), cond(A), norm(x_s-t), norm(x_r-t), norm(x_n-t), norm(x_m-t));
end

function [x] = lls_rd(A, b, tol) % rank deficient least squares using SVD
[m, n] = size(A); [U, S, V] = svd(A); x = zeros(n,1);
%m, diag(S)
for k = 1:min(m,n)
  if S(k,k) < S(1,1)*tol, break;
  else x = x+((U(:,k)'*b)/S(k,k))*V(:,k); end
end
```

8.18.

```
function cp08_18 % numerical differentiation of discrete data
t = [0; 1; 2; 3; 4; 5]; y0 = [1; 2.7; 5.8; 6.6; 7.5; 9.9]; y = y0; pert = 0;
for i = -1:0
fprintf('For random data perturbation of size %6.1e:\n', pert);
disp('(a) Derivative values for least squares polynomial of degree n:');
for n = 0:5
  x = fliplr(polyfit(t,y,n)); [p, d] = horner_eval(x, t);
  fprintf(' %3d    %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n', n, d);
end;
disp('(b) Derivative values for cubic spline interpolant:');
pp = spline(t,y); sd = fnval(fnder(pp),t);
fprintf('        %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n', sd);
disp('(c) Derivative values for smoothing spline with parameter tol:');
for k = -6:1
  tol = 10^k; sp = spaps(t,y,tol); sd = fnval(fnder(sp),t);
  fprintf('%6.1e %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n', tol, sd);
end
disp('(d) Derivative values for monotonic Hermite cubic interpolant:');
pp = pchip(t,y); sd = fnval(fnder(pp),t);
fprintf('        %8.4f %8.4f %8.4f %8.4f %8.4f %8.4f\n', sd);
pert = 10^i; y = y0+pert*2*(rand(6,1)-0.5); disp(' ');
end;

function [p, d] = horner_eval(x, t)
```

```
[m,n] = size(t); nc = length(x); p = x(nc)*ones(m,n); d = zeros(m,n);
for i=nc-1:-1:1 % compute polynomial value p and derivative value d
  d = p+t.*d; p = x(i)+t.*p;
end
```

Chapter 9

Initial Value Problems for Ordinary Differential Equations

Exercises

9.1. (a)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= t + u_1 + u_2. \end{aligned}$$

(b)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= u_3, \\ u_3' &= u_3 + t u_1. \end{aligned}$$

(c)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= u_3, \\ u_3' &= u_3 - 2u_2 + u_1 - t + 1. \end{aligned}$$

9.2. (a)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= u_2(1 - u_1^2) - u_1. \end{aligned}$$

(b)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= u_3, \\ u_3' &= -u_1 u_3. \end{aligned}$$

(c)

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -GMu_1/(u_1^2+u_3^2)^{3/2}, \\ u_3' &= u_4, \\ u_4' &= -GMu_3/(u_1^2+u_3^2)^{3/2}. \end{aligned}$$

9.3. Yes, since this first-order, linear, homogeneous system with constant coefficients can be written

$$\boldsymbol{y}' = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \boldsymbol{A}\boldsymbol{y},$$

and the eigenvalues of $\boldsymbol{A}$, -1 and -2, are both negative.

9.4. (a) Yes, because the eigenvalue, -5, is negative, and hence the solutions decay exponentially. (b) No, since $|1 + h\lambda| = |1 + 0.5\,(-5)| = 1.5 > 1$. ($c$) $y_1 = 1 + 0.5(-5) = -1.5$. (d) Yes, since backward Euler is unconditionally stable. (e) $y_1 = 1 + 0.5\,(-5y_1) \Rightarrow y_1 = 1/3.5 = 0.2857$.

9.5. (a) $y_1 = y_0 + hf(y_0) = 1 + 1 \cdot (-1) = 0$. ($b$) $y_1 = y_0 + hf(y_1) = 1 + 1 \cdot (-y_1)$, so $y_1 = 0.5$.

9.6. The fixed-point iteration procedure is

$$y_{n+1} = 1 - y_n^3/2,$$

with $y_0 = 1$ (or $y_0 = 0.5$ if forward Euler is used to obtain the starting guess). Note that for the fixed-point iteration function $g(y) = 1 - y^3/2$, we have $g'(y) = -3y^2/2$, so that at the solution, $|g'(0.7709)| = 0.8914 < 1$, and hence the fixed-point iteration is locally linearly convergent, but that alone does not show that it converges from either of these particular starting points. To prove the latter, we first use induction to prove that $1/2 \le y_n \le 1$. This holds for $n = 0$ with either starting point. Suppose it holds for $n = k$. Then $0 < y_k \Rightarrow y_{k+1} = 1 - y_k^3/2 < 1$, and $y_k \le 1 \Rightarrow y_{k+1} \ge 1 - 1/2 = 1/2$. Next, we observe that

$$\begin{aligned} y_{n+1} - y_n &= (1 - y_n^3/2) - (1 - y_{n-1}^3/2) \\ &= (y_n - y_{n-1})(y_n^2 + y_n y_{n-1} + y_{n-1}^2)/2, \end{aligned}$$

so

$$\begin{aligned} \frac{|y_{n+1} - y_n|}{|y_n - y_{n-1}|} &= (y_n^2 + y_n y_{n-1} + y_{n-1}^2)/2 \\ &= \left((1 - y_{n-1}^3/2)^2 + (1 - y_{n-1}^3/2) y_{n-1} + y_{n-1}^2\right)/2. \end{aligned}$$

If we prove that the right-hand side is strictly less than 1, then the sequence $\{y_i\}_{i=0}^{\infty}$ is a Cauchy sequence and therefore converges. We show that for $1/2 \le x \le 1$,

$$f(x) = (1 - x^3/2)^2 + (1 - x^3/2)x + x^2 < 2$$

by the following reasoning:

$$\begin{aligned} f(x) - 2 &= -1 + x + x^2 - x^3 - x^4/2 + x^6/4 \\ &< -1 + x + x^2 - x^3 - x^4/2 + x^6/2 \\ &= -(1 - x^2)(1 - x) - x^4(1 - x^2)/2 \\ &= -(1 - x^2)(1 - x + x^4/2) \le 0. \end{aligned}$$

The convergence rate is linear ($r = 1$) as shown in the proof.

9.7. (a) $\boldsymbol{u}' = \begin{bmatrix} u_1' \\ u_2' \end{bmatrix} = \begin{bmatrix} u_2 \\ u_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \boldsymbol{A}\boldsymbol{u}$. (b) $\boldsymbol{u}(0) = \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. (c) No, because the two eigenvalues of $\boldsymbol{A}$ are ± 1, and the positive eigenvalue indicates that the solutions are unstable. (d)

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_1 = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_0 + h \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + 0.5 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 2.5 \end{bmatrix}.$$

(e) No, because $\rho(\boldsymbol{I} + h\boldsymbol{A}) = 3/2 > 1$. (f) No, because the solutions of the ODE are unstable.

9.8. (a) $y_1 = 1 - 0.1y_1^2$, or $0.1y_1^2 + y_1 - 1 = 0$. (b) $x_{k+1} = x_k - f(x_k)/f'(x_k) = x_k - (0.1x_k^2 + x_k - 1)/(0.2x_k + 1)$. (c) $\hat{y}_1 = 1 - 0.1 \cdot 1^2 = 0.9$. (d) $y_1 = 0.9 + 0.019/1.18 = 0.916$.

9.9.

Property / Method	(1)	(2)	(3)
(a) second-order accurate	yes	yes	yes
(b) single-step method	yes	no	yes
(c) implicit method	no	no	yes
(d) self-starting	yes	no	yes
(e) unconditionally stable	no	no	yes
(f) Runge-Kutta type method	yes	no	no
(g) good for solving stiff ODEs	no	no	yes

Note that (1) is Heun's method, (2) is the explicit two-step Adams method, and (3) is the trapezoid method, all of which are discussed in detail in the textbook.

9.10. For the sample problem $y' = f(t, y) = \lambda y$, Heun's method is

$$y_{k+1} = y_k + \frac{h}{2}(f(t_k, y_k) + f(t_k + h, y_k + hf(t_k, y_k))) = y_k(1 + h\lambda + \frac{1}{2}(h\lambda)^2).$$

The exact solution is

$$y_{k+1} = y_k e^{\lambda h} = y_k((1 + h\lambda + \frac{1}{2}(h\lambda)^2 + \mathcal{O}(h^3)),$$

so Heun's method is second-order accurate. The stability requirement is $|1 + h\lambda + \frac{1}{2}(h\lambda)^2| < 1$. In the complex plane, we are to find z such that $|1 + z + \frac{1}{2}z^2| < 1$. For $z = x + iy$, this reduces to

$$\left(1 + x + \frac{x^2 - y^2}{2}\right)^2 + (1 + x)^2 y^2 < 1.$$

Simplifying, we obtain

$$\left(\frac{y^2}{2} - t\right)^2 < t,$$

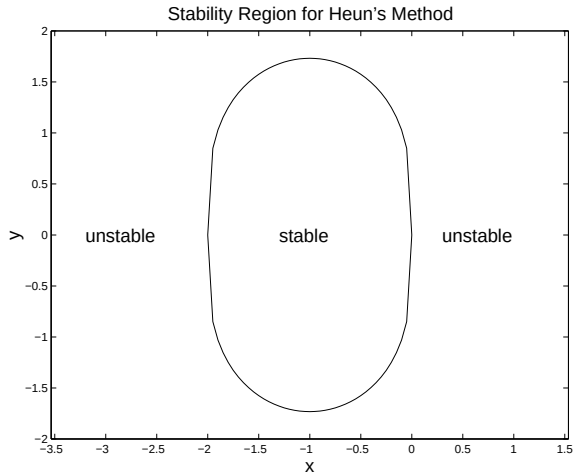
where $t = -(x + x^2/2)$. It is obvious that t should be greater than 0, which imposes the constraint $-2 < x < 0$. From the preceding inequality, we have

$$t - 2\sqrt{t} < y^2 < t + 2\sqrt{t}.$$

Since $-2 < x < 0$, we have $0 < t < 1$, so that $t < \sqrt{t} < 2\sqrt{t}$, and $0 \le |y| < \sqrt{t + 2\sqrt{t}}$. The solution is

$$-\sqrt{-(2x + x^2) + 2\sqrt{-(2x + x^2)}} < y < \sqrt{-(2x + x^2) + 2\sqrt{-(2x + x^2)}}.$$

The stability region, whose boundary is shown graphically below, is bounded and roughly oval shaped.



9.11. To simplify notation, we define $f_k = f(t_k, y_k)$, $f_{k,y} = f_y(t_k, y_k)$ and $f_{k,t} = f_t(t_k, y_k)$. Note that $y''(t_k) = f'(t_k, y_k) = f_{k,t} + f_{k,y} f_k$. A Taylor series expansion at t_k gives

$$\begin{aligned} y_{k+1} - y(t_{k+1}) &= \left(y_k + h\left(f_k + f_{k,t}\frac{h}{2} + f_{k,y}\frac{y_{k+1} - y_k}{2} + \mathcal{O}(h^2)\right)\right) \\ &\quad - \left(y_k + h f_k + \frac{h^2}{2} y_k'' + \mathcal{O}(h^3)\right) \end{aligned}$$

$$\begin{aligned}
&= f_{k,t}\frac{h^2}{2} + f_{k,y}\frac{h}{2}(y_{k+1} - y_k) - \frac{h^2}{2}(f_{k,t} + f_k f_{k,y}) + \mathcal{O}(h^3) \\
&= \frac{h^2}{2} f_{k,y}\left(\frac{y_{k+1} - y_k}{h} - f_k\right) + \mathcal{O}(h^3) \\
&= \frac{h^2}{2} f_{k,y}\left(f(t_k + \frac{h}{2}, \frac{y_{k+1} + y_k}{2}) - f_k\right) + \mathcal{O}(h^3) \\
&= \frac{h^2}{2} f_{k,y}\left(f_{k,t}\frac{h}{2} + f_{k,y}\frac{y_{k+1} - y_k}{2}\right) + \mathcal{O}(h^3) \\
&= \mathcal{O}(h^3).
\end{aligned}$$

The last step assumes that f_k, $f_{k,y}$, $f_{k,t}$ are all finite. This means the midpoint method is at least second-order accurate. For the model problem $y' = f(t, y) = \lambda y$, the midpoint method simplifies to

$$y_{k+1} = \frac{2 + h\lambda}{2 - h\lambda} y_k,$$

where $h = h_k$. The stability requirement gives

$$\left|\frac{2 + h\lambda}{2 - h\lambda}\right| < 1.$$

For $\lambda < 0$, any $h > 0$ satisfies this requirement, so the midpoint method is unconditionally stable. The exact solution is $y(t_{k+1}) = y_k e^{\lambda t}$, so

$$y_{k+1} - y(t_{k+1}) = y_k(\frac{1}{12}(h\lambda)^3 + \mathcal{O}(h^4)),$$

and hence the midpoint method is second-order accurate.

9.12. Assuming $y_k = y(t_k)$ and $y_{k-1} = y(t_{k-1})$, a Taylor expansion at t_k for both y_k and y_{k-1} gives

$$\begin{aligned}
y(t_{k+1}) - y_{k+1} &= y(t_k + h) - (y_{k-1} + 2hf_k) \\
&= \left(y_k + hy'_k + \frac{h^2}{2}y''_k + \frac{h^3}{6}y'''_k + \mathcal{O}(h^4)\right) - \\
&\quad \left(y_k - hy'_k + \frac{h^2}{2}y''_k - \frac{h^3}{6}y'''_k + \mathcal{O}(h^4) + 2hf_k\right) \\
&= \frac{h^3}{3}y'''_k + \mathcal{O}(h^4),
\end{aligned}$$

so the leapfrog method is second-order accurate. To analyze its stability, consider the test ODE $y' = \lambda y$, for which the leapfrog scheme becomes $y_{k+1} = y_{k-1} + 2h\lambda y_k$. Making the substitution $y_k = \alpha^k$, we then have $\alpha^{k+1} = \alpha^{k-1} + 2h\lambda\alpha^k$, or $\alpha = 1/\alpha + 2h\lambda$. Now if α_0 is a root of the latter equation, then $-1/\alpha_0$ is the other root. Thus, if one root lies inside the unit circle, the other lies outside it, and hence for stability, both roots must lie on the unit circle. Moreover, the roots must be

distinct, which rules out $\alpha_0 = \pm i$. Now as α_0 ranges over the unit circle (other than $\pm i$), $\alpha_0 - 1/\alpha_0$ ranges over the open interval $(-2i, 2i)$ along the imaginary axis. Thus, the stability region for the leapfrog method is that $h\lambda$ must lie in the open interval $(-i, i)$ on the imaginary axis.

9.13. $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k \Rightarrow \boldsymbol{x}_k = \boldsymbol{A}^k\boldsymbol{x}_0$, so boundedness of the solution requires that $\rho(\boldsymbol{A}) \leq 1$, i.e., the eigenvalues of $\boldsymbol{A}$ must lie in the unit disk in the complex plane. $\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x} \Rightarrow \boldsymbol{x}(t) = e^{\boldsymbol{A}t}\boldsymbol{x}(0)$, so boundedness of the solution requires that $\mathrm{Re}(\lambda_i) \leq 0$ for every eigenvalue, i.e., the eigenvalues of $\boldsymbol{A}$ must lie in the left half of the complex plane (see Example 9.7).

9.14. The kth order ODE is equivalent to the first-order system

$$\boldsymbol{u}' = \begin{bmatrix} \boldsymbol{u}_1' \\ \boldsymbol{u}_2' \\ \vdots \\ \boldsymbol{u}_{k-1}' \\ \boldsymbol{u}_k' \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 & -c_0 \\ 1 & 0 & \cdots & 0 & -c_1 \\ 0 & 1 & \cdots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -c_{k-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \vdots \\ \boldsymbol{u}_{k-1} \\ \boldsymbol{u}_k \end{bmatrix} = \boldsymbol{A}\boldsymbol{u}.$$

Since $\boldsymbol{A}$ is a companion matrix, its eigenvalues are the roots of the polynomial $p(\lambda) = c_0 + c_1\lambda + \cdots + c_{k-1}\lambda^{k-1} + \lambda^k$. Thus, the criterion for stability is that the roots of $p(\lambda)$ must lie in the left half of the complex plane. Asymptotic stability requires additionally that any nonsimple root must have strictly negative real part.

Computer Problems

9.1.
```
function cp09_01 % predatory-prey models
[t, y] = ode45(@Lotka_Volterra, 0:0.1:25, [100;10]); figure(1); subplot(2,1,1);
plot(t,y(:,1), 'b-', t, y(:,2), 'r--'); legend('y_1','y_2',0);  xlabel('t');
ylabel('y'); title('Computer Problem 9.1(a) -- Lotka-Volterra Model');
subplot(2,1,2); plot(y(:,1), y(:,2)); xlabel('y_1'); ylabel('y_2');
title('Phase Portrait');

[t, y] = ode45(@Leslie_Gower, 0:0.1:25, [100;10]); figure(2); subplot(2,1,1);
plot(t,y(:,1), 'b-', t, y(:,2), 'r--'); legend('y_1','y_2',0); xlabel('t');
ylabel('y'); title('Computer Problem 9.1(b) -- Leslie-Gower Model');
subplot(2,1,2); plot(y(:,1), y(:,2)); xlabel('y_1'); ylabel('y_2');
title('Phase Portrait');

function [yprime] = Lotka_Volterra(t,y)
alpha1 = 1; beta1 = 0.1; alpha2 = 0.5; beta2 = 0.02;
yprime = [y(1)*(alpha1-beta1*y(2)); y(2)*(-alpha2+beta2*y(1))];

function [yprime] = Leslie_Gower(t,y)
alpha1 = 1.0; beta1 = 0.1; alpha2 = 0.5; beta2 = 10;
yprime = [y(1)*(alpha1-beta1*y(2)); y(2)*(alpha2-beta2*y(2)/y(1))];
```

9.2.
```
function cp09_02 % Kermack-McKendrick epidemic model
```

```
[t, y] = ode45(@Kermack_McKendrick, 0:0.01:1, [95;5;0]);
plot(t,y(:,1),'b-',t,y(:,2),'r--',t,y(:,3),'g-.'); legend('y_1','y_2','y_3');
xlabel('t'); ylabel('y'); title('Computer Problem 9.2 -- Kermack-McKendrick Model');

function [yprime] = Kermack_McKendrick(t,y); c = 1; d = 5;
yprime = [-c*y(1)*y(2); c*y(1)*y(2)-d*y(2); d*y(2)];
```

9.3.
```
function cp09_03 % stiff ODE
f = inline('-200*t*y^2','t','y'); fjac = inline('-400*t*y','t','y');
fs = {'ode23' 'ode23s' 'ode23t' 'ode23tb' 'ode45' 'ode113' 'ode15s'};
options = odeset('RelTol', 1e-5, 'AbsTol', 1e-8');
for i = 1:7
  solver = str2func(fs{i});
  [t1,y1] = feval(solver,f,[0,1],1,options); s1 = t1(2:end)-t1(1:end-1);
  [t2,y2] = feval(solver,f,[-3,1],1/901,options); s2 = t2(2:end)-t2(1:end-1);
  figure; plot(t1,y1,'bo-',t2,y2,'ro--'); xlabel('t'); ylabel('y');
  title(['Computer Problem 9.3 -- Solution of ODE Using ' fs{i}]);
  figure; plot(t1(1:end-1),s1,'b-',t2(1:end-1),s2,'r--');
  xlabel('t'); ylabel('step size');
  title(['Computer Problem 9.3 -- Step Size Using ' fs{i}]);
end
```

9.4.
```
function cp09_04 % linear chemical reaction kinetics
fs = {'ode23' 'ode23s' 'ode23t' 'ode23tb' 'ode45' 'ode113' 'ode15s'};
k2 = [10; 100; 1000]; time = zeros(3,7);
for k = 1:3 % compute and plot solution for each value of k2
  [t,y] = ode23tb(@ode,[0,5],[1;1;1],[],k2(k));
  figure; plot(t,y(:,1),'b-',t,y(:,2),'r--',t,y(:,3),'g-.'); xlabel('t');
  ylabel('y'); title('Computer Problem 9.4 -- Solution of ODE');
  legend('y_1','y_2','y_3',0); text(2.25,1.5,['k_2 = ' num2str(k2(k),'%3d')]);
end
for i = 1:7 % time execution for each method
  solver = str2func(fs{i}); figure;
  for k = 1:3
    tic; [t,y] = feval(solver,@ode,[0,5],[1;1;1],[],k2(k)); time(k,i) = toc;
    s = t(2:end)-t(1:end-1); subplot(3,1,k);
    plot(t(1:end-1),s,'b-'); xlabel('t'); ylabel('step size');
    if k == 1, title(['Computer Problem 9.4 -- Step Size Using ' fs{i}]); end
  end
end
disp('Time to compute solution for each method and each value of k2:');
disp(['  k2    ' fs{1} '    ' fs{2} '   ' fs{3} '   ' fs{4} '  ' fs{5} ...
  '    ' fs{6} '   ' fs{7}]);
for k = 1:3
  fprintf('%4d %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n', k2(k), time(k,:));
end

function [f] = ode(t, y, k2)
f = [-y(1); y(1)-k2*y(2); k2*y(2)];
```

9.5.
```
function cp09_05 % nonlinear chemical reaction kinetics
```

```
fs = {'ode23' 'ode23s' 'ode23t' 'ode23tb' 'ode45' 'ode113' 'ode15s'};
time = zeros(3,7); [t,y] = ode23tb(@ode,[0,3],[1;0;0]);
figure; plot(t,y(:,1),'b-',t,y(:,2),'r--',t,y(:,3),'g-.'); xlabel('t');
ylabel('y'); title('Computer Problem 9.5 -- Solution of ODE');
legend('y_1','y_2','y_3',0);
for i = 1:7 % time execution for each method
  solver = str2func(fs{i}); figure;
  for k = 1:3
    tol = 10^(-k); options = odeset('RelTol', tol, 'AbsTol', tol*1e-3);
    tic; [t,y] = feval(solver,@ode,[0,3],[1;0;0],options); time(k,i) = toc;
    s = t(2:end)-t(1:end-1); subplot(3,1,k);
    plot(t(1:end-1),s,'b-'); xlabel('t'); ylabel('step size');
    if k == 1; title(['Computer Problem 9.5 -- Step Size Using ' fs{i}]); end
  end
end
disp('Time to compute solution for each method and each tolerance:');
disp(['  tol    ' fs{1} '    ' fs{2} '   ' fs{3} '   ' fs{4} '  ' fs{5} ...
  '    ' fs{6} '   ' fs{7}]);
for k = 1:3
  tol = 10^(-k);
  fprintf('%4.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f\n', tol, time(k,:));
end

function [yprime] = ode(t,y); alpha = 0.04; beta = 1e4; gamma = 3e7;
yprime = [-alpha*y(1)+beta*y(2)*y(3); ...
  alpha*y(1)-beta*y(2)*y(3)-gamma*y(2)*y(2); gamma*y(2)*y(2)];
```

9.6.

```
function cp09_06 % Lorenz model for atmospheric circulation
n = 2001;
[t,y] = ode45(@lorenz,linspace(0,100,n),[0;1;0]); figure;
plot(t,y(:,1),'b-',t,y(:,2),'r--',t,y(:,3),'g-.');
legend('y_1','y_2','y_3',0); xlabel('t');
figure; plot(y(:,1),y(:,2)); xlabel('y_1(t)'); ylabel('y_2(t)');
figure; plot(y(:,1),y(:,3)); xlabel('y_1(t)'); ylabel('y_3(t)');
figure; plot(y(:,2),y(:,3)); xlabel('y_2(t)'); ylabel('y_3(t)');
disp('Original solution at t = 100:'); y = y(n,:)
[t,y] = ode45(@lorenz,linspace(0,100,n),[eps;1;0]);
disp('Perturbed solution at t = 100:'); y = y(n,:)

function [yprime] = lorenz(t,y); sigma = 10; b = 8/3; r = 28;
yprime = [sigma*(y(2)-y(1)); r*y(1)-y(2)-y(1)*y(3); y(1)*y(2)-b*y(3)];
```

9.7.

```
function cp09_07 % two-body orbit problem
time = zeros(6,3); e = [0 0.5 0.9];
for i = 1:3
  for k = 1:6
    tol = 10^(-k); options = odeset('RelTol', tol, 'AbsTol', tol*1e-3);
    tic; [t,y] = ode45(@orbit,0:0.01:13,[1-e(i);0;0;sqrt((1+e(i))/(1-e(i)))],...
      options); time(k,i) = toc;
    E = (y(:,2).^2+y(:,4).^2)/2-1./sqrt(y(:,1).^2+y(:,3).^2); stdE(k,i) = std(E);
    M = y(:,1).*y(:,4)-y(:,2).*y(:,3); stdM(k,i) = std(M);
```

```
  end
  figure; subplot(2,1,1); plot(t, y(:,1)); xlabel('t'); ylabel('x');
  title('Computer Problem 9.7 -- Solution of ODE for Two-Body Problem');
  subplot(2,1,2); plot(t, y(:,3)); xlabel('t'); ylabel('y');
  figure; plot(y(:,1), y(:,3)); axis equal; xlabel('x'); ylabel('y');
  xmin = min(y(:,1)); xmax = max(y(:,1)); ymin = min(y(:,3)); ymax = max(y(:,3));
  title('Computer Problem 9.7 -- Computed Orbit for Two-Body Problem');
  text((xmin+xmax)/2, (ymin+ymax)/2, ['e = ', num2str(e(i), 2)]);
end
disp('Time to compute solution for each tolerance and eccentricity:');
disp('   tol         0.0          0.5          0.9  ');
for k = 1:6
  tol = 10^(-k); fprintf('%8.1e %12.7f %12.7f %12.7f\n', tol, time(k,:));
end; disp(' ');
disp('Standard deviation of energy for each tolerance and eccentricity:');
disp('   tol         0.0          0.5          0.9  ');
for k = 1:6
  tol = 10^(-k); fprintf('%8.1e %12.3e %12.3e %12.3e\n', tol, stdE(k,:));
end; disp(' ');
disp('Standard deviation of angular momentum for each tolerance and eccentricity:');
disp('   tol         0.0          0.5          0.9  ');
for k = 1:6
  tol = 10^(-k); fprintf('%8.1e %12.3e %12.3e %12.3e\n', tol, stdM(k,:));
end; disp(' ');

function [yprime] = orbit(t,y); % x = y(1), x' = y(2), y = y(3), y' = y(4)
r = sqrt(y(1)*y(1)+y(3)*y(3)); yprime = [y(2); -y(1)/r^3; y(4); -y(3)/r^3];
```

9.8.

```
function cp09_08 % three-body orbit problem
d = 4.669e6; y0=[4.613e8; 0; 0; -1074]; time = zeros(6);
for k = 1:6
  tol = 10^(-k); options = odeset('RelTol', tol, 'AbsTol', tol*1e-3);
  tic; [t,y] = ode45(@apollo,[0,2.4e6], y0, options); time(k) = toc;
end
figure; subplot(2,1,1); plot(t, y(:,1)); xlabel('t'); ylabel('x');
title('Computer Problem 9.8 -- Solution of ODE for Three-Body Problem');
subplot(2,1,2); plot(t, y(:,3)); xlabel('t'); ylabel('y'); figure;
s = t(2:end)-t(1:end-1); plot(t(1:end-1),s); xlabel('t'); ylabel('step size');
title('Computer Problem 9.8 -- Step Size for Three-Body Problem');
figure; plot(y(:,1), y(:,3)); axis equal; xlabel('x'); ylabel('y'); hold on;
title('Computer Problem 9.8 -- Computed Orbit for Three-Body Problem');
plot(-d,0,'o'); plot(3.844e8-d,0,'o')
text(-3d7,-5e7,'earth'); text(3.5e8,-5e7,'moon')
disp('Minimum distance from spacecraft earth''s surface:');
min_dist = sqrt(min((y(:,1)+d).^2+y(:,3).^2))-6.378e6
disp('Time to compute solution for each tolerance:');
disp('   tol        time');
for k = 1:6
  tol = 10^(-k); fprintf('%8.1e %12.7f\n', tol, time(k));
end; disp(' ');
```

```
function [yprime] = apollo(t,y); % y(1) = x, y(2) = x', y(3) = y, y(4) = y'
G = 6.67259e-11; M = 5.974e24; m = 7.348e22; must = M/(m+M); mu = m/(m+M);
D = 3.844e8; d = 4.669e6; omega = 2.661e-6;
r1 = sqrt((y(1)+d)^2+y(3)^2); r2 = sqrt((D-d-y(1))^2+y(3)^2);
yprime = [y(2); -G*(M*(y(1)+mu*D)/r1^3+m*(y(1)-must*D)/r2^3)+omega^2*y(1)+ ...
  2*omega*y(4); y(4); -G*(M*y(3)/r1^3+m*y(3)/r2^3)+omega^2*y(3)-2*omega*y(2)];
```

9.9. function cp09_09 % numerical integration using ODE solver

```
fs = {'4./(1+x.^2)' 'sqrt(x).*log(x)' 'cos(x)' '1./(1+100*x.^2)' 'sqrt(abs(x))'};
prob = {'8.1' '8.2' '8.3a' '8.3b' '8.3c'}; a = [0 1e-10 -1 -1 -1]; b = [1 1 1 1 1];
for i = 1:5
  f = inline(fs{i},'x'); g = inline(fs{i},'x','y');
  disp(['Computed integral and execution time for Computer Problem ' prob{i}]);
  disp('   tol      quad    time     quadl   time     ode23   time     ode45   time');
  for k = 1:9
    tol = 10^(-k); options = odeset('RelTol',tol,'AbsTol',tol);
    tic; I1 = quad(f,a(i),b(i),tol); time1 = toc;
    tic; I2 = quadl(f,a(i),b(i),tol); time2 = toc;
    tic; [t,y] = ode23(g,[a(i),b(i)],0,options); time3 = toc; I3 = y(end);
    tic; [t,y] = ode45(g,[a(i),b(i)],0,options); time4 = toc; I4 = y(end);
    fprintf('%8.1e %9.6f %6.3f %9.6f %6.3f %9.6f %6.3f %9.6f %6.3f\n', tol, I1, ...
      time1, I2, time2, I3, time3, I4, time4);
  end; disp(' ');
end
```

9.10. function cp09_10 % ODE homotopy method for solving nonlinear algebraic equation

```
[t, y] = ode45(@homotopy,[0,10],[1;1;1]); plot(t,y(:,1),t,y(:,2),t,y(:,3));
legend('x','y','z',0); xlabel('t'); ylabel('solution components'); y(end,:)
title('Computer Problem 9.10 -- Solution of Nonlinear System by Homotopy Method');

function [yprime] = homotopy(t,y)
f = [16*y(1)^4+16*y(2)^4+y(3)^4-16; y'*y-3; y(1)^3-y(2)];
J = [64*y(1)^3, 64*y(2)^3, 4*y(3)^3; 2*y(1), 2*y(2), 2*y(3); 3*y(1)^2, -1, 0];
yprime = -(J\f);
```

Chapter 10

Boundary Value Problems for Ordinary Differential Equations

Exercises

10.1. (a) If $u(t) = c_1 + c_2 t + \int_0^t F(s)\,ds$, where $F(s) = \int_0^s f(x)\,dx$, then $u'(t) = c_2 + F(t)$ and $u''(t) = F'(t) = f(t)$, so the ODE is satisfied. (b) Using integration by parts, we have

$$\begin{aligned}\int_0^t F(s)\,ds &= (sF(s)|_{s=t} - sF(s)|_{s=0}) - \int_0^t s\,dF(s) \\ &= t\int_0^t f(x)\,dx - \int_0^t sf(s)\,ds = \int_0^t (t-s)f(s)\,ds.\end{aligned}$$

(c) $u(0) = 0 \Rightarrow c_1 = 0, \qquad u(1) = 0 \Rightarrow c_2 = \int_0^1 (s-1)f(s)\,ds.$

(d)

$$\begin{aligned}u(t) &= \int_0^t (t-s)f(s)\,ds + t\int_0^1 (s-1)f(s)\,ds \\ &= \int_0^t (t-s)f(s)\,ds + t\left(\int_0^t (s-1)f(s)\,ds + \int_t^1 (s-1)f(s)\,ds\right) \\ &= \int_0^t s(t-1)f(s)\,ds + \int_t^1 t(s-1)f(s)\,ds = \int_0^1 G(t,s)f(s)\,ds.\end{aligned}$$

(e) $u(t) = \int_0^1 G(t,s)1\,ds = \int_0^t s(t-1)\,ds + \int_t^1 t(s-1)\,ds = (t^2-t)/2.$

(*f*) $u(t) = \int_0^1 G(t,s)s\,ds = \int_0^t s^2(t-1)\,ds + \int_t^1 ts(s-1)\,ds = (t^3-t)/6$.

(*g*) $u(t) = \int_0^1 G(t,s)s^2\,ds = \int_0^t s^3(t-1)\,ds + \int_t^1 ts^2(s-1)\,ds = (t^4-t)/12$.

10.2. (*a*) Defining $y_1(t) = u(t)$, $y_2(t) = u'(t)$, we have

$$\boldsymbol{y}' = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \boldsymbol{A}\boldsymbol{y},$$

with

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} y_1(b) \\ y_2(b) \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

(*b*)

$$\boldsymbol{Y}'(t) = \begin{bmatrix} \sinh(t) & \cosh(t) \\ \cosh(t) & \sinh(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \cosh(t) & \sinh(t) \\ \sinh(t) & \cosh(t) \end{bmatrix} = \boldsymbol{A}\boldsymbol{Y}(t).$$

(*c*) Solutions to the ODE are unstable, since $\|\boldsymbol{Y}(t)\|$ grows exponentially with t.

(*d*)

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 \\ \cosh(b) & \sinh(b) \end{bmatrix}.$$

(*e*)

$$\Phi(t) = \boldsymbol{Y}(t)\boldsymbol{Q}^{-1} = \frac{1}{\sinh(b)} \begin{bmatrix} \sinh(b-t) & \sinh(t) \\ -\cosh(b-t) & \cosh(t) \end{bmatrix}.$$

(*f*) $\boldsymbol{Q}$ becomes increasingly ill-conditioned as b increases. Because of scaling by $1/\sinh(b)$, $\|\boldsymbol{Q}\| \approx 1$ for any $b > 0$. The Green's function is also bounded, so the solutions to the BVP are stable for any $b > 0$.

10.3. (*a*) Euler's method gives $\beta = \alpha + s(b-a)$, where s is the initial slope. (*b*) Solving for s, we obtain $s = (\beta - \alpha)/(b-a)$.

10.4. (*a*) Defining $y_1(t) = u(t)$ and $y_2(t) = u'(t)$, we have

$$\boldsymbol{y}' = \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \end{bmatrix} = \boldsymbol{A}\boldsymbol{y} + \boldsymbol{b}.$$

1. Using the trapezoid method, $\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h(\boldsymbol{y}_k' + \boldsymbol{y}_{k+1}')/2$, with $h = 1$, we have

$$\begin{bmatrix} 1 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 \\ s_0 \end{bmatrix} + \frac{1}{2} \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ s_0 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ s_1 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \end{bmatrix} \right).$$

2. Solving the above equation gives $s_0 = 2$, $s_1 = -2$.

3. From

$$\begin{bmatrix} y_1(0.5) \\ y_2(0.5) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \frac{0.5}{2} \left(\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1(0.5) \\ y_2(0.5) \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \end{bmatrix} \right),$$

we obtain $u(0.5) \approx y_1(0.5) = 1.5$. (b) Using a finite difference method, we have

$$u''(0.5) \approx \frac{y_0 - 2y_1 + y_2}{h^2} = -4.$$

With $h = 0.5$, $y_0 = u(0) = 1$, and $y_2 = u(1) = 1$, we obtain $u(0.5) \approx y_1 = 1.5$. (c) Assuming $u(t) = x_0 + x_1 t + x_2 t^2$, we have the system of equations

$$\begin{aligned} u(0) = 1 &\quad \Rightarrow \quad x_0 = 1, \\ u(1) = 1 &\quad \Rightarrow \quad x_0 + x_1 + x_2 = 1, \\ u'(0.5) = -4 &\quad \Rightarrow \quad 2x_2 = -4. \end{aligned}$$

Solving this system, we obtain $x_0 = 1$, $x_1 = 2$, and $x_2 = -2$, so that $u(0.5) \approx 1 + 2(0.5) - 2(0.5)^2 = 1.5$.

Computer Problems

10.1.

```
function cp10_01 % two-point boundary value problem
global a b ua ub rhs ivp; a = 0; b = 1; ua = 0; ub = 1;
rhs = inline('10*u.^3+3*u+t.^2','t','u');
ivp = inline('[y(2); 10*y(1)^3+3*y(1)+t^2]','t','y');
disp('(a) Initial slope determined by shooting method:');
slope = fzero(@shooting_fun,0.5)
[t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1),'r-');
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.1(a) -- Shooting Method for Two-Point BVP');
n = [1 3 7 15]; figure;
options = optimset('MaxFunEvals', 5000, 'Display', 'off');
for i = 1:4
  t = linspace(a,b,n(i)+2); y = [ua; fsolve(@fd_fun,t(2:n(i)+1),options)'; ub];
  plot(t,y); hold on;
end
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.1(b) -- Finite Difference Method for Two-Point BVP');
n = [3 4 5 6]; figure;
disp('(c) Coefficients of polynomials determined by collocation:');
for i = 1:4
  x = fsolve(@colloc_fun,ones(n(i),1),options)', t = linspace(a,b,50)';
  plot(t,polyval(x,t)); hold on;
end
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.1(c) -- Collocation Method for Two-Point BVP');

function [g] = shooting_fun(slope); global a b ua ub ivp;
[t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1)); hold on; g = ub-y(end,1);

function [z] = fd_fun(y); global a b ua ub rhs;
n = length(y); h = 1/(n+1); t = linspace(h,1-h,n); f = rhs(t,y);
if n > 1
  z = [(y(2)-2*y(1)+ua)/h^2-f(1); ((y(3:n)-2*y(2:n-1)+y(1:n-2))/h^2-f(2:n-1))'; ...
```

```
    (ub-2*y(n)+y(n-1))/h^2-f(n)];
else
  z = (ub-2*y+ua)/h^2-f;
end

function [z] = colloc_fun(x); % collocation method
global a b ua ub rhs; n = length(x); h = 1/(n-1); t = linspace(a,b,n)';
d = (n-1:-1:1)'.*x(1:n-1); d2 = (n-2:-1:1)'.*(d(1:n-2));
z = [polyval(x,a)-ua; polyval(d2,t(2:n-1))-rhs(t(2:n-1),polyval(x,t(2:n-1)));...
  polyval(x,b)-ub];
```

10.2. `function cp10_02 % two-point boundary value problem`

```
global a b ua ub rhs ivp; a = 0; b = 1; ua = 0; ub = 1;
rhs = inline('-(1+exp(u))','t','u');
ivp = inline('[y(2); -(1+exp(y(1)))]','t','y');
disp('(a) Initial slope determined by shooting method:');
slope = fzero(@shooting_fun,2.5)
[t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1),'r-');
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.2(a) -- Shooting Method for Two-Point BVP');
n = [1 3 7 15]; figure;
options = optimset('MaxFunEvals', 5000, 'Display', 'off');
for i = 1:4
  t = linspace(a,b,n(i)+2); y = [ua; fsolve(@fd_fun,t(2:n(i)+1),options)'; ub];
  plot(t,y); hold on;
end
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.2(b) -- Finite Difference Method for Two-Point BVP');
n = [3 4 5 6]; figure;
disp('(c) Coefficients of polynomials determined by collocation:');
for i = 1:4
  x = fsolve(@colloc_fun,ones(n(i),1),options)', t = linspace(a,b,50)';
  plot(t,polyval(x,t)); hold on;
end
xlabel('t'); ylabel('u'); axis([0 1 0 1.2]);
title('Computer Problem 10.2(c) -- Collocation Method for Two-Point BVP');

function [g] = shooting_fun(slope); global a b ua ub ivp;
[t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1)); hold on; g = ub-y(end,1);

function [z] = fd_fun(y); global a b ua ub rhs;
n = length(y); h = 1/(n+1); t = linspace(h,1-h,n); f = rhs(t,y);
if n > 1
  z = [(y(2)-2*y(1)+ua)/h^2-f(1); ((y(3:n)-2*y(2:n-1)+y(1:n-2))/h^2-f(2:n-1))'; ...
    (ub-2*y(n)+y(n-1))/h^2-f(n)];
else
  z = (ub-2*y+ua)/h^2-f;
end

function [z] = colloc_fun(x); % collocation method
```

```
global a b ua ub rhs; n = length(x); h = 1/(n-1); t = linspace(a,b,n)';
d = (n-1:-1:1)'.*x(1:n-1); d2 = (n-2:-1:1)'.*(d(1:n-2));
z = [polyval(x,a)-ua; polyval(d2,t(2:n-1))-rhs(t(2:n-1),polyval(x,t(2:n-1)));...
  polyval(x,b)-ub];
```

10.3.
```
function cp10_03 % two-point boundary value problem
global a b ua ub ivp; a = 0; b = 4; ua = 0; ub = -2;
ivp = inline('[y(2); -abs(y(1))]','t','y'); guess = [-0.07, 2];
for i = 1:2
  disp('Initial slope determined by shooting method:');
  slope = fzero(@shooting_fun,guess(i)),
  [t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1),'r-');
  xlabel('t'); ylabel('u');
  title('Computer Problem 10.3 -- Shooting Method for Two-Point BVP');
end

function [g] = shooting_fun(slope) % shooting method
global a b ua ub ivp; [t, y] = ode45(ivp,[a,b],[ua;slope]);
plot(t, y(:,1)); hold on; g = ub-y(end,1);
```

10.4.
```
function cp10_04 % two-point boundary value problem
global a b ua ub ivp; a = 0; b = 1; ua = 0; ub = 0;
ivp = inline('[y(2); -exp(y(1)+1)]','t','y'); guess = [2, 7];
for i = 1:2
  disp('Initial slope determined by shooting method:');
  slope = fzero(@shooting_fun,guess(i)),
  [t, y] = ode45(ivp,[a,b],[ua;slope]); plot(t, y(:,1),'r-');
  xlabel('t'); ylabel('u');
  title('Computer Problem 10.4 -- Shooting Method for Two-Point BVP');
end

function [g] = shooting_fun(slope) % shooting method
global a b ua ub ivp; [t, y] = ode45(ivp,[a,b],[ua;slope]);
plot(t, y(:,1)); hold on; g = ub-y(end,1);
```

10.5.
```
function cp10_05 % hanging cable problem
global a b ua ub k n; a = 0; b = 1; ua = [0; 0]; ub = [0.75 0.85; 0 0.5];
n = 15; options = optimset('MaxFunEvals',10000,'Display','off');
for k = 1:2
  guess = [0; 1]; fval = fsolve(@shooting_fun,guess,options);
  [t, y] = ode45(@cable,[a,b],[ua; fval]); plot(y(:,1),y(:,2)); hold on;
end
title('Computer Problem 10.5 -- Shooting Method for Hanging Cable');
xlabel('y_1'); ylabel('y_2'); axis([0 0.9 -0.4 0.5]);
text(0.47,-0.15,'slack cable'); text(0.39,0.25,'taut cable'); figure;
for k = 1:2
  guess = zeros(4,n+2); guess(4,:) = ones(1,n+2); guess(1,n+2) = ub(1,k);
  y = fsolve(@fd_fun,guess,options); plot(y(1,:), y(2,:)); hold on;
end
title('Computer Problem 10.5 -- Finite Difference Method for Hanging Cable');
xlabel('y_1'); ylabel('y_2'); axis([0 0.9 -0.4 0.5]);
```

```
text(0.47,-0.15,'slack cable'); text(0.39,0.25,'taut cable');

function [f] = shooting_fun(guess) % shooting method
global a b ua ub k; [t, y] = ode45(@cable, [a,b], [ua; guess]);
f = ub(:,k)-y(end,1:2)';

function [yprime] = cable(t, y)
yprime = [cos(y(3)); sin(y(3)); (cos(y(3))-sin(y(3))*abs(sin(y(3))))/y(4); ...
  sin(y(3))-cos(y(3))*abs(cos(y(3)))];

function [f] = fd_fun(y)
global ua ub k n; h = 1/(n+1);
f = [y(:,2:end)-y(:,1:end-1)-h*rhs(y(:,1:n+1)), ...
  [y(1,1)-ua(1); y(2,1)-ua(2); y(1,end)-ub(1,k); y(2,end)-ub(2,k)]];

function [f] = rhs(y); c = cos(y(3,:)); s = sin(y(3,:));
f = [c; s; (c-s.*abs(s))./y(4,:); s-c.*abs(c)];
```

10.6.
```
function cp10_06 % eigenvalue problem for horizontal beam
for n = 2:10
  h = 2/(n+1); A = 2*diag(ones(n,1))-diag(ones(n-1,1),1)-diag(ones(n-1,1),-1);
  A = A/h^2; t = linspace(-1,1,n+2); tt = (t(2:n+1).^2)'; B = diag(tt+1);
  Lambda = eig(A,B)
end
```

10.7.
```
function cp10_07 % eigenvalue problem for 1-D Schroedinger equation
disp('Computed and exact eigenvalues for 1-D Schroedinger equation:');
for n = 2:5
  h = 1/(n+1); A = diag(-ones(n-1,1),-1)+diag(2*ones(n,1),0)+diag(-ones(n-1,1),1);
  E = eig(A)/h^2; Exact = ((1:1:n)'*pi).^2;
  disp('  computed    exact'); disp([E'; Exact']');
end
```

Chapter 11

Partial Differential Equations

Exercises

11.1. (*a*) If we let $y(t) = \int_a^t g(s)\,ds$, then we seek $y(b)$. Thus, we take $y'(t) = g(t)$ as the ODE subproblem, with initial condition $y(a) = 0$. There are no special properties of the ODE subproblem unless we know more about the function $g(t)$. (*b*) Defining new unknown functions $u_1(t) = y(t)$ and $u_2(t) = y'(t)$, we obtain an equivalent system of first-order ODEs

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= u_1^2 + t, \end{aligned}$$

to be solved by the ODE solver. Since we have only an initial value solver, we solve the boundary value problem using the shooting method. We already have $u_1(0) = y(0) = 0$, and we try various initial guesses for $u_2(0)$ until the correct value for $u_1(1) = y(1) = 1$ is attained. There are no special properties of the ODE subproblem. (*c*) We can use an ODE solver to solve the PDE using the method of lines. Using a finite difference method, we introduce spatial mesh points $x_i = i\Delta x$, $i = 0, \ldots, n+1$, where $\Delta x = 1/(n+1)$. The system of ODEs for the ODE solver is

$$y_i'(t) = \frac{c}{(\Delta x)^2}\left(y_{i+1}(t) - 2y_i(t) + y_{i-1}(t)\right), \quad i = 1, \ldots, n,$$

where $y_i(t) \approx u(t, x_i)$. The initial conditions are given by $y_i(0) = g(x_i)$, $i = 1, \ldots, n$, and $y_0(t)$ and $y_{n+1}(t)$ are determined by the given boundary conditions. This problem requires a stiff ODE solver because the Jacobian matrix of the ODE system tends to be ill-conditioned.

11.2. The second-order accurate, centered finite difference scheme gives

$$u_{xx} \approx \frac{u_{x+\Delta x,y} - 2u_{x,y} + u_{x-\Delta x,y}}{(\Delta x)^2}, \qquad u_{yy} \approx \frac{u_{x,y+\Delta y} - 2u_{x,y} + u_{x,y-\Delta y}}{(\Delta y)^2}.$$

Substituting these into the PDE gives the approximation

$$\frac{u_{x+\Delta x,y} - 2u_{x,y} + u_{x-\Delta x,y}}{(\Delta x)^2} + \frac{u_{x,y+\Delta y} - 2u_{x,y} + u_{x,y-\Delta y}}{(\Delta y)^2} \approx x + y.$$

Substituting $x = y = 0.5$, the given mesh spacing $\Delta x = \Delta y = 0.5$, and the known boundary values gives

$$\frac{1 - 2u_{x,y} + 0}{(0.5)^2} + \frac{1 - 2u_{x,y} + 0}{(0.5)^2} \approx 0.5 + 0.5 = 1,$$

which we solve for $u_{x,y}$ to obtain

$$u_{x,y} \approx 7/16 = 0.4375.$$

11.3. The finite difference scheme for the heat equation given in Example 11.2 is consistent, but it is unstable if the time step size violates the inequality given at the bottom of page 458, causing the computed solution to grow without bound and obviously fail to converge to the true solution. Similarly, the finite difference scheme for the wave equation given in Example 11.3 is consistent, but it is unstable if the time step size violates the inequality given by the CFL condition in Example 11.4, again causing the computed solution to grow without bound and obviously fail to converge to the true solution. Thus, consistency alone does not guarantee convergence. To show that stability alone does not guarantee convergence, consider the "zero method," which gives zero as the solution at every time step, regardless of the time step size or the specific problem being solved. This method satisfies the definition of stability, but it is obviously not consistent, and the "computed solution" clearly does not converge to the true solution.

11.4. For $d = 1$, the matrix is tridiagonal, the diagonal entries are 2, and the nonzero off-diagonal entries are -1. For $d = 2$, the matrix is block tridiagonal, the diagonal blocks are tridiagonal, the nonzero off-diagonal blocks are diagonal, the diagonal entries are 4, and the nonzero off-diagonal entries are -1. For $d = 3$, the matrix is block block tridiagonal, the diagonal blocks are block tridiagonal, the nonzero off-diagonal blocks are diagonal, the diagonal entries are 6, and the nonzero off-diagonal entries are -1.

11.5. (a) Shown by removing nodes from the graph representation in numerical order. Whenever a node is removed, completely connect all of its neighbors. Added edges represent fill. (b) Verify by inspection following same process as in part (a). For minimum degree, the first four nodes removed are of degree two, the next two are of degree three. Then one of degree two is removed, then one of degree one, and we are left with the last node which must be of degree zero. At every step a

node of minimum degree is removed. The permutation matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

For nested dissection, nodes 7, 8, and 9 split the graph into two equal pieces, as do nodes 3 and 6 for the remaining subgraphs. The permutation matrix is

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

11.6. Let $\boldsymbol{B} = \boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U}) = \boldsymbol{D}^{-1}\boldsymbol{L} + \boldsymbol{D}^{-1}\boldsymbol{U}$. Then $b_{ii} = 0$, $i = 1, \ldots, n$, and $b_{ij} = -a_{ij}/a_{ii}$ for $i \neq j$. The diagonal dominance of $\boldsymbol{A}$ implies that

$$\frac{1}{|a_{ii}|} \sum_{j=1, j\neq i}^{n} |a_{ij}| < 1, \quad i = 1, \ldots, n,$$

and hence

$$\sum_{j=1, j\neq i}^{n} |b_{ij}| = \sum_{j=1, j\neq i}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1, \quad i = 1, \ldots, n.$$

Therefore $\|\boldsymbol{B}\|_\infty < 1$, which implies that $\rho(\boldsymbol{B}) < 1$, and hence Jacobi converges.

11.7. Let $\boldsymbol{B} = (\boldsymbol{D}+\omega\boldsymbol{L})^{-1}((1-\omega)\boldsymbol{D} - \omega\boldsymbol{U})$. Now $(\boldsymbol{D}+\omega\boldsymbol{L})^{-1}$ is lower triangular with diagonal entries $1/a_{ii}$, $i = 1, \ldots, n$, and $((1-\omega)\boldsymbol{D} - \omega\boldsymbol{U})$ is upper triangular with diagonal entries $(1-\omega)a_{ii}$, $i = 1, \ldots, n$. Since the determinant of a product of matrices is the product of their determinants, and the determinant of a triangular matrix is equal to the product of its diagonal entries, it follows that $\det(\boldsymbol{B}) = (1-\omega)^n$. But since the determinant of a matrix is equal to the product of its eigenvalues, this implies that $\rho(\boldsymbol{B}) = |1-\omega|$, and thus SOR converges only if $\omega \in (0, 2)$

11.8. From the CG algorithm, $\boldsymbol{s}_{k+1}$ can be expressed as

$$\boldsymbol{s}_{k+1} = \boldsymbol{r}_{k+1} - \sum_{i=0}^{k} c_{ki}\boldsymbol{s}_i,$$

where

$$c_{ki} = \boldsymbol{r}_{k+1}^T \boldsymbol{A}\boldsymbol{s}_i / \boldsymbol{s}_i^T \boldsymbol{A}\boldsymbol{s}_i.$$

Now $\boldsymbol{r}_{k+1}^T \boldsymbol{r}_i = 0$ for $i = 0, \ldots, k$, but this implies that $\boldsymbol{r}_{k+1}^T \boldsymbol{A}\boldsymbol{s}_i = 0$ for $i = 0, \ldots, k-1$, since $\text{span}([\boldsymbol{r}_0, \ldots, \boldsymbol{r}_k]) = \text{span}([\boldsymbol{s}_0, \ldots, \boldsymbol{s}_k]) = \text{span}([\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0])$ (see Exercise 11.9). Thus, $c_{ki} = 0$, $i = 0, \ldots, k-1$, and hence the claimed three-term recurrence holds.

11.9. We want to show that $\text{span}([\boldsymbol{s}_0, \ldots, \boldsymbol{s}_m]) = \text{span}([\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0]) \equiv \mathcal{K}_m$. We will show that both of these spaces are also equal to $\text{span}([\boldsymbol{r}_0, \ldots, \boldsymbol{r}_m])$. Without loss of generality, assume that $\boldsymbol{x}_0 = \boldsymbol{o}$ (otherwise we can work with the residual equation $\boldsymbol{A}\boldsymbol{e} = \boldsymbol{r}_0 = \boldsymbol{b}_0 - \boldsymbol{A}\boldsymbol{x}_0$). The proof proceeds by induction on m. For $m = 0$, the three spaces are equal, since $\boldsymbol{s}_0 = \boldsymbol{r}_0$. Now assume the three spaces are equal for $m = k$, and we will show that they must be equal for $m = k+1$. By assumption, $\boldsymbol{s}_k \in \mathcal{K}_k = \text{span}([\boldsymbol{r}_0, \ldots, \boldsymbol{r}_k])$, so we have $\boldsymbol{A}\boldsymbol{s}_k \in \mathcal{K}_{k+1}$, and hence $\boldsymbol{r}_{k+1} \in \mathcal{K}_{k+1}$, since $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A}\boldsymbol{s}_k$, which shows that $\text{span}([\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{k+1}]) \subseteq \mathcal{K}_{k+1}$. By similar reasoning using the inductive hypothesis, we have $\text{span}([\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{k+1}]) \subseteq \text{span}([\boldsymbol{r}_0, \ldots, \boldsymbol{r}_{k+1}])$, since $\boldsymbol{s}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{s}_k$. But all three of these spaces are generated by the same number of vectors, namely $k+1$, and those vectors are linearly independent because of their $\boldsymbol{A}$-orthogonality. Thus, the three spaces must be of the same dimension and hence equal.

11.10. See any good textbook on numerical linear algebra.

Computer Problems

11.1.

```
function cp11_01 % solution of heat equation by various methods
dx = 0.05; n = 19; dt = 0.0012; k = 50;
[u, x, t] = explicit_method(n, dx, k, dt); figure; mesh(t,x,u);
view(110,30); xlabel('t'); ylabel('x'); zlabel('u'); axis([0 dt*k 0 1 0 1]);
title(sprintf('Computer Problem 11.1(a) -- Explicit Method, dt = %5.4f', dt));
dt = 0.0013; k = 47;
[u, x, t] = explicit_method(n, dx, k, dt); figure; mesh(t,x,u);
view(110,30); xlabel('t'); ylabel('x'); zlabel('u'); axis([0 dt*k 0 1 0 1]);
title(sprintf('Computer Problem 11.1(b) -- Explicit Method, dt = %5.4f', dt));
dt = 0.005; k = 12;
[u, x, t] = backward_Euler(n, dx, k, dt); figure; mesh(t,x,u);
view(110,30); xlabel('t'); ylabel('x'); zlabel('u'); axis([0 dt*k 0 1 0 1]);
title(sprintf('Computer Problem 11.1(c) -- Backward Euler, dt = %5.4f', dt));
[u, x, t] = Crank_Nicolson(n, dx, k, dt); figure; mesh(t,x,u);
view(110,30); xlabel('t'); ylabel('x'); zlabel('u'); axis([0 dt*k 0 1 0 1]);
title(sprintf('Computer Problem 11.1(d) -- Crank-Nicolson, dt = %5.4f', dt));
[u, x, t] = semidiscrete_method(n, dx, 0.06);
[m,k] = size(u); U = zeros(m,n+2); U(:,2:n+1) = u; figure; mesh(t,x,U');
view(110,30); xlabel('t'); ylabel('x'); zlabel('u'); axis([0 t(length(t)) 0 1 0 1]);
title('Computer Problem 11.1(e) -- Semidiscrete Method');

function [u, x, t] = explicit_method(n, dx, k, dt)
u = zeros(n+2,k+1); x = dx*(0:n+1)'; t = dt*(0:k)';
```

```
u(1:(n+1)/2+1,1) = 2*x(1:(n+1)/2+1); u((n+1)/2+2:n+2,1) = 2-2*x((n+1)/2+2:n+2);
s = dt/dx^2; A = sparse(diag((1-2*s)*ones(n,1))+diag(s*ones(n-1,1),-1)+...
  diag(s*ones(n-1,1), 1)); v = zeros(n,1); k = k+1;
for m = 2:k
  v(1) = s*u(1,m-1); v(n) = s*u(n+2,m-1); u(2:n+1,m) = A*u(2:n+1,m-1)+v;
end

function [u, x, t] = backward_Euler(n, dx, k, dt)
u = zeros(n+2,k+1); x = dx*(0:n+1)'; t = dt*(0:k)';
u(1:(n+1)/2+1,1) = 2*x(1:(n+1)/2+1); u((n+1)/2+2:n+2,1) = 2-2*x((n+1)/2+2:n+2);
s = dt/dx^2; A = diag((1 + 2*s)*ones(n,1)) + diag(-s*ones(n-1,1), -1) + ...
  diag(-s*ones(n-1,1), 1); [L, U] = lu(A); v = zeros(n,1); k = k+1;
for m = 2:k
  v(1) = s*u(1,m-1); v(n) = s*u(n+2,m-1); b = u(2:n+1,m-1)+v; u(2:n+1,m) = U\(L\b);
end

function [u, x, t] = Crank_Nicolson(n, dx, k, dt)
u = zeros(n+2,k+1); x = dx*(0:n+1)'; t = dt*(0:k)';
u(1:(n+1)/2+1,1) = 2*x(1:(n+1)/2+1); u((n+1)/2+2:n+2,1) = 2-2*x((n+1)/2+2:n+2);
s = dt/dx^2; A = diag((1+s)*ones(n,1))+diag(-0.5*s*ones(n-1,1),-1)+...
  diag(-0.5*s*ones(n-1,1),1); [L, U] = lu(A); v = zeros(n,1); clear A;
A = sparse(diag((1-s)*ones(n,1))+diag(0.5*s*ones(n-1,1),-1)+...
  diag(0.5*s*ones(n-1,1), 1)); k = k+1;
for m = 2:k
  v(1) = s*u(1,m-1); v(n) = s*u(n+2,m-1); b = A*u(2:n+1,m-1)+v; u(2:n+1,m) = U\(L\b);
end

function [u, x, t] = semidiscrete_method(n, dx, endtime)
x = dx*(0:n+1)'; u0(1:(n+1)/2) = 2*x(2:(n+1)/2+1);
u0((n+1)/2+1:n) = 2-2*x((n+1)/2+2:n+1); u0 = u0.'; s = 1/dx^2;
A = sparse(diag(-2*s*ones(n,1))+diag(s*ones(n-1,1),-1)+diag(s*ones(n-1,1),1));
[t, u] = ode23s(@ode_fun, [0 endtime], u0, [], A, n);

function [f] = ode_fun(t, u, A, n); f = A*u;
```

11.2.
```
function cp11_02 % method of lines solution of heat equation
for k=1:6
  n = 2^k-1; dx(k) = 1/(n+1); x = dx(k)*(0:n+1)'; u0 = exact_sol_a(n,x,0);
  s = 1/dx(k)^2; A = sparse(diag(-2*s*ones(n,1))+diag(s*ones(n-1,1),-1)+...
    diag(s*ones(n-1,1),1));
  [t, u] = ode23s(@ode_fun_a,[0 0.1],u0(2:n+1),[],A,s,n);
  m = length(t); U = zeros(m, n+2); U(:,2:n+1) = u;
  uexact = exact_sol_a(n,x,0.1); err(k) = max(abs(uexact'-U(m,:)));
end
figure; mesh(t,x,U'); view(110,30); xlabel('t'); ylabel('x'); zlabel('u');
title(sprintf('Computer Problem 11.2(a) -- Numerical Solution, dx = %5.4f',dx(k)));
figure; loglog(dx,err,'o-'); xlabel('dx'); ylabel('error');
title('Computer Problem 11.2(a) -- Maximum Error vs. Spatial Mesh Size');
for k=1:6
  n = 2^k-1; dx(k) = 1/(n+1); x = dx(k)*(0:n+1)'; u0 = exact_sol_b(n,x,0);
```

```
  s = 1/dx(k)^2; A = sparse(diag(-2*s*ones(n,1))+diag(s*ones(n-1,1),-1)+...
    diag(s*ones(n-1,1),1));
  [t, U] = ode23s(@ode_fun_b,[0 0.1],u0,[],A,s,n); m = length(t);
  uexact = exact_sol_b(n,x,0.1); err(k) = max(abs(uexact'-U(m,:)));
end
figure; mesh(t,x,U'); view(110,30); xlabel('t'); ylabel('x'); zlabel('u');
title(sprintf('Computer Problem 11.2(b) -- Numerical Solution, dx = %5.4f',dx(k)));
figure; loglog(dx,err,'o-'); xlabel('dx'); ylabel('error');
title('Computer Problem 11.2(b) -- Maximum Error vs. Spatial Mesh Size');

function [f] = ode_fun_a(t, u, A, s, n); f = A*u;

function [f] = ode_fun_b(t, u, A, s, n)
v = zeros(n,1); v(1) = s*u(1); v(n) = s*u(n+2); f = zeros(n+2,1);
f(2:n+1) = A*u(2:n+1)+v; f(1) = s*(u(2)-u(1)); f(n+2) = s*(u(n+1)-u(n+2));

function [u] = exact_sol_a(n, x, t); u = exp(-pi^2*t)*sin(pi*x);

function [u] = exact_sol_b(n, x, t); u = exp(-pi^2*t)*cos(pi*x);
```

11.3.

```
function cp11_03 % method of lines solution of wave equation
for k=1:6
  n = 2^k-1; dx(k) = 1/(n+1); x = dx(k)*(0:n+1)'; u0 = ic_fun(n,x);
  s = 1/dx(k)^2; [t, u] = ode23s(@ode_fun,[0 2],u0,[],s,n);
  m = length(t); ue = exact_sol_fun(n,x,2); err(k) = max(abs(ue'-u(m,1:n+2)));
end
figure; mesh(t,x,u(:,1:n+2)'); view(135,30); xlabel('t'); ylabel('x'); zlabel('u');
title(sprintf('Computer Problem 11.3 -- Numerical Solution, dx = %5.4f',dx(k)));
figure; loglog(dx,err,'o-'); xlabel('dx'); ylabel('error');
title('Computer Problem 11.3 -- Maximum Error vs. Spatial Mesh Size');

function [f] = ode_fun(t, u, s, n)
f = zeros(2*n+4,1); i1 = 2:n+1; i2 = n+4:2*n+3;
f(i1) = u(i2); f(i2) = s*(u(i1+1)-2*u(i1)+u(i1-1));

function [u] = ic_fun(n, x); u = zeros(2*n+4,1); u(1:n+2) = sin(pi*x);

function [u] = exact_sol_fun(n, x, t); u = cos(pi*t).*sin(pi*x);
```

11.4.

```
function cp11_04 % method of lines solution of advection equation
tfinal = 2; n = 39; dx = 1/(n+1); x = dx*(0:n+1)'; u0 = zeros(n+1,1); s = 1/dx;
[t, u] = ode23s(@ode_fun_1,[0 tfinal],u0,[],s,n);
[m,k] = size(u); U = zeros(m,n+2); U(:,2:n+2) = u; U(:,1) = ones(m,1);
figure; mesh(t,x,U'); view(170,30); xlabel('t'); ylabel('x'); zlabel('u');
title(sprintf('Computer Problem 11.4 -- One-Sided Scheme, dx = %4.3f',dx));
s = stepfun(t,1); figure; plot(t,u(:,k),t,s); xlabel('t'); ylabel('u');
title('Computer Problem 11.4 -- Solution at x=1'); axis([0 2 0 1.4]);
legend('one-sided approximation','exact solution'); s = 1/(2*dx);
[t, u] = ode23s(@ode_fun_2,[0 tfinal],u0,[],s,n); [m,k] = size(u);
U = zeros(m,n+2); U(:,2:n+2) = u; U(:,1) = ones(m,1);
figure; mesh(t,x,U'); view(170,30); xlabel('t'); ylabel('x'); zlabel('u');
```

```
title(sprintf('Computer Problem 11.4 -- Centered Scheme, dx = %4.3f',dx));
s = stepfun(t,1); figure; plot(t,u(:,k),t,s); xlabel('t'); ylabel('u');
title('Computer Problem 11.4 -- Solution at x = 1'); axis([0 2 0 1.4]);
legend('centered approximation','exact solution');

function [f] = ode_fun_1(t, u, s, n); u0 = 1; f = zeros(n+1,1);
i = 2:n+1; f(1) = -s*(u(1)-u0); f(i) = -s*(u(i)-u(i-1));

function [f] = ode_fun_2(t, u, s, n); u0 = 1; f = zeros(n+1,1); i = 2:n;
f(1) = -s*(u(2)-u0); f(i) = -s*(u(i+1)-u(i-1)); f(n+1) = -s*(u(n+1)-u(n));
```

11.5.

```
function cp11_05 % method of lines solution of Burgers' equation
n = 19; dx = 1/(n+1); x = dx*(0:n+1)'; tfinal = 2;
for j=1:3
  v = 10^(1-j); u0 = ic_fun(x(2:n+1),v); s1 = 1/dx; s2 = s1^2;
  [t, u] = ode23s(@ode_fun_1,[0 tfinal],u0,[],v,s1,s2,n);
  [m, k] = size(u); U = zeros(m,n+2); U(:,2:n+1) = u;
  U(:,1) = (bc_fun(t,0,v)); U(:,n+2) = (bc_fun(t,1,v));
  figure; mesh(t,x,U'); view(140,30); xlabel('t'); ylabel('x'); zlabel('u');
  title(sprintf('Computer Problem 11.5 -- One-Sided Scheme, v = %3.2f, dx = %4.3f',...
    v,dx));
  u_one = U(m,:); s1 = 1/(2*dx); s2 = 1/(dx*dx);
  [t, u] = ode23s(@ode_fun_2,[0 tfinal],u0,[],v,s1,s2,n);
  [m, k] = size(u); U = zeros(m,n+2); U(:,2:n+1) = u;
  U(:,1) = (bc_fun(t,0,v)); U(:,n+2) = (bc_fun(t,1,v));
  figure; mesh(t,x,U'); view(140,30); xlabel('t'); ylabel('x'); zlabel('u');
  title(sprintf('Computer Problem 11.5 -- Centered Scheme, v = %3.2f, dx = %4.3f',...
    v,dx));
  figure; plot(x,bc_fun(tfinal,x,v),'>-',x,u_one,'o-',x,U(m,:),'*-');
  xlabel('x'); ylabel('u'); legend('exact','one-sided FD','centered FD',3);
  title(sprintf('Computer Problem 11.5 -- Solution at t = 2; v = %3.2f, dx = %4.3f',...
    v,dx));
end

function [f] = ode_fun_1(t, u, v, s1, s2, n)
f = zeros(n,1); u0 = bc_fun(t,0,v); u1 = bc_fun(t,1,v); i = 2:n-1;
f(1) = -s1*u(1).*(u(1)-u0)+v*s2*(u(2)-2*u(1)+u0);
f(i) = -s1*u(i).*(u(i)-u(i-1))+v*s2*(u(i+1)-2*u(i)+u(i-1));
f(n) = -s1*u(n).*(u(n)-u(n-1))+v*s2*(u1-2*u(n)+u(n-1));

function [f] = ode_fun_2(t, u, v, s1, s2, n)
f = zeros(n,1); u0 = bc_fun(t,0,v); u1 = bc_fun(t,1,v); i = 2:n-1;
f(1) = -s1*u(1).*(u(2)-u0)+v*s2*(u(2)-2*u(1)+u0);
f(i) = -s1*u(i).*(u(i+1)-u(i-1))+v*s2*(u(i+1)-2*u(i)+u(i-1));
f(n) = -s1*u(n).*(u1-u(n-1))+v*s2*(u1-2*u(n)+u(n-1));

function [u] = ic_fun(x, v); u = 1./(1+exp(x./(2*v)));

function [u] = bc_fun(t, x, v); u = 1./(1+exp(x./(2*v)-t./(4*v)));
```

11.6.

```
function cp11_06 % solution of Poisson's equation on L-shaped region
```

```
n = 19; h = 1/(n+1); n2 = n^2; b = h^2*2*ones(n2,1);
A = sparse((diag(4*ones(n2,1))-diag(ones(n2-1,1),1)-diag(ones(n2-1,1),-1)-...
  diag(ones(n2-n,1),n)-diag(ones(n2-n,1),-n)));
for k = 1:n-1 % zero out entries because of L-shaped region
  m = k*n; A(m,m+1) = 0; A(m+1,m) = 0;
end; d = (n-1)/2; el = (n-1)*n+1;
for k = d*n+1:n:el
  for m = 0:d
    j = k+m;
    A(j,j-n) = 0; A(j,j-1) = 0; A(j,j+1) = 0;
    if k ~= el
      A(j,j+n) = 0;
    end
    b(j) = 0;
  end
end
u = A\b; U = zeros(n+2,n+2); U(2:n+1,2:n+1) = reshape(u,n,n); v = 0:h:1;
figure; mesh(v,v,U); xlabel('x'); ylabel('y'); zlabel('u'); view(80,30);
title(sprintf('Computer Problem 11.6 -- Direct Method, h = %4.3f',h));
D = diag(diag(A)); L = tril(A,-1); U = triu(A,1);
options = optimset('TolX',1e-2,'TolFun',1e-2);
omega = fminbnd(@spectral_radius,0,2,options,D,L,U);
bb = omega*((D+omega*L)\b); x = zeros(n2,1); resid = 1; tol = 1e-4;
while resid > tol
  x = (D+omega*L)\(((1-omega)*D-omega*U)*x)+bb; resid = norm(b-A*x,inf);
end
X = zeros(n+2,n+2); X(2:n+1,2:n+1) = reshape(x,n,n); v = 0:h:1;
figure; mesh(v,v,X); xlabel('x'); ylabel('y'); zlabel('u'); view(80,30);
title(sprintf('Computer Problem 11.6 -- SOR Method, h = %4.3f, omega = %4.3f',...
  h,omega));

function [rho] = spectral_radius(omega,D,L,U)
rho = max(abs((eig(full((D+omega*L)\((1-omega)*D-omega*U))))));
```

11.7.
```
function cp11_07 % eigenvalue problem for 2-D Schroedinger equation
for n = 2:5
  h = 1/(n+1); I = speye(n,n); K = sparse(2:n,1:n-1,1,n,n); D = K+K'-2*I;
  A = kron(D,I)+kron(I,D); E = eig(-full(A))/h^2; Exact = zeros(n,n);
  for j=1:n
    k = (1:n)'; Exact(k,j) = (k.^2+j^2);
  end; Exact = sort(reshape(Exact,n^2,1))*pi^2;
  disp('  computed    exact'); disp([E' Exact']');
end
```

11.8.
```
function cp11_08 % eigenvalues of Jacobian of semidiscrete ODE system
disp('    dx          -4/dx^2       lambda_min     lambda_max');
for k = 1:10
  n = 2^k-1; dx = 1/(n+1);
  A = diag(-2*ones(n,1))+diag(ones(n-1,1),1)+diag(ones(n-1,1),-1);
  Lambda = eig(A)/dx^2;
  fprintf('%e %e  %e  %e\n',dx,-4/dx^2,min(Lambda),max(Lambda));
```

```
end
```

11.9.
```
function cp11_09 % stability analysis for discretized advection equation
disp('                  Centered Discretization');
disp('Jacobian matrix is skew symmetric, so its eigenvalues are purely');
disp('imaginary, and hence rho(I+dt*J)>=1 for any dt>0.  Sample values:');
disp('       dx                  dt              rho(I+dt*J)');
for k = 1:6
  n = 2^k-1; dx = 1/(n+1);
  for m = 1:6
    dt = 2^(-m); J = (diag(ones(1,n-1),-1)-diag(ones(1,n-1),1))/(2*dx);
    rho = max(abs(eig(eye(n,n)+dt*J)));
    fprintf('%14.6e      %14.6e      %14.6e\n', dx, dt, rho);
  end
end
disp(' '); disp('                 One-Sided Discretization');
disp('One-sided discretization is stable if dt <= 2*dx.  Sample values:');
disp('       dx                  dt              rho(I+dt*J)');
for k = 1:6
  n = 2^k-1; dx = 1/(n+1);
  for m = 1:6
    dt = 2^(-m); J = (diag(ones(1,n-1),-1)-diag(ones(1,n),0))/dx;
    rho = max(abs(eig(eye(n,n)+dt*J)));
    fprintf('%14.6e      %14.6e      %14.6e\n', dx, dt, rho);
  end
end
```

11.10.
```
function cp11_10 % optimal omega in SOR for Laplace's equation
k = [5 10 20];
disp(' k     omega_theory   omega_approx  rho(G)_theory  rho(G)_approx');
for j = 1:length(k)
  h = 1/(k(j)+1); omega = 0:0.1:2; m = length(omega);
  I = speye(k(j),k(j)); K = sparse(2:k(j),1:k(j)-1,1,k(j),k(j)); T = K+K'-2*I;
  A = (kron(T,I)+kron(I,T))/h^2; D = diag(diag(A)); L = tril(A,-1); U = triu(A,1);
  for n=1:m
    rho(n) = spectral_radius(omega(n),D,L,U);
  end
  min_theory = (1-sin(pi*h))/(1+sin(pi*h)); omega_theory = 2/(1+sin(pi*h));
  options = optimset('TolX',1e-2,'TolFun',1e-2);
  [omega_approx, min_approx] = fminbnd(@spectral_radius,0,2,options,D,L,U);
  figure; plot(omega_theory,min_theory,'ro',omega,rho,'-');
  axis([0 2 min(min_theory-0.05,min_approx-0.05) 1]);
  xlabel('omega'); ylabel('rho(G)'); legend('theoretical minimum',3);
  title(sprintf('Computer Problem 11.10 -- Optimal SOR Parameter, k = %d', k(j)));
  fprintf('%2d %14.6f %14.6f %14.6f %14.6f\n', k(j), omega_theory, omega_approx,...
    min_theory, min_approx);
end

function [rho] = spectral_radius(omega,D,L,U)
G = (D+omega*L)\((1-omega)*D-omega*U); rho = max(abs((eig(full(G)))));
```

11.11. function cp11_11 % convergence rates of stationary iterative methods

```
k = [5 10 20 50 100 200 500];  disp('Theoretical values of rho(G) and R:');
disp(' k      rho_J        R_J        rho_GS       R_GS      rho_SOR      R_SOR');
for j = 1:7
  h = 1/(k(j)+1); rho_J = cos(pi*h); R_J = -log10(rho_J);
  rho_GS = rho_J^2; R_GS = -log10(rho_GS);
  rho_SOR = (1-sin(pi*h))/(1+sin(pi*h)); R_SOR = -log10(rho_SOR);
  fprintf('%3d %11.7f %11.7f %11.7f %11.7f %11.7f %11.7f\n', k(j), rho_J, R_J,...
    rho_GS, R_GS, rho_SOR, R_SOR);
end; disp(' ');
disp('Computed values of rho(G) and R:');
disp(' k      rho_J        R_J        rho_GS       R_GS      rho_SOR      R_SOR');
for j = 1:3
  h = 1/(k(j)+1); I = speye(k(j),k(j)); K = sparse(2:k(j),1:k(j)-1,1,k(j),k(j));
  T = K+K'-2*I; A = (kron(T,I)+kron(I,T))/h^2;
  D = diag(diag(A)); L = tril(A,-1); U = triu(A,1);
  G_J = D\(-L-U); rho_J = max(abs((eig(full(G_J))))); R_J = -log10(rho_J);
  G_GS = (D+L)\(-U); rho_GS = max(abs((eig(full(G_GS))))); R_GS = -log10(rho_GS);
  omega = 2/(1+sin(pi*h)); G_SOR = (D+omega*L)\((1-omega)*D-omega*U);
  rho_SOR = max(abs((eig(full(G_SOR))))); R_SOR = -log10(rho_SOR);
  fprintf('%3d %11.7f %11.7f %11.7f %11.7f %11.7f %11.7f\n', k(j), rho_J, R_J,...
    rho_GS, R_GS, rho_SOR, R_SOR);
end
```

11.12. function cp11_12

```
n = 100; condA = [1 10 100 1000];
b = rand(n,1); x0 = zeros(n,1); tol = 1e-6; maxits = 10000;
disp('cond(A)     SD_iter      CG_iter        SD_time            CG_time');
for i = 1:4
  [Q, R] = qr(rand(n,n)); A = Q'*diag(linspace(1,condA(i),n))*Q;
  tic; j = 0; x = x0; r = b-A*x; d = r'*r; s = r; % steepest descent
  while sqrt(d) > tol  &  j < maxits
    j = j+1; v = A*s; alpha = d/(s'*v); x = x+alpha*s; r = r-alpha*v;
    d = r'*r; s = r;
  end; time_SD = toc;
  tic; k = 0; x = x0; r = b-A*x; d = r'*r; s = r; % conjugate gradient
  while sqrt(d) > tol  &  k < maxits
    k = k+1; v = A*s; alpha = d/(s'*v); x = x+alpha*s; r = r-alpha*v;
    beta = d; d = r'*r; beta = d/beta; s = r+beta*s;
  end; time_CG = toc;
  fprintf(' %4d         %4d        %4d         %f           %f\n', ...
    condA(i), j, k, time_SD, time_CG);
end
```

11.13. function cp11_13 % Gauss-Seidel and Jacobi smoothers

```
n = 50; k = [1 5 10 15 20 25]; b = zeros(n,1);  maxits = 10;
part = {'(a)' '(b)'}; method = {'Gauss-Seidel' 'Jacobi'};
for i = 1:2
  for j=1:7
    if j < 7, x0 = sin((1:n)'*k(j)*pi./(n+1)); str = sprintf('k = %d', k(j));
    else x0 = ones(n,1); str = 'x0 = 1'; end;
```

```
    x = zeros(n+2,1); x(2:n+1) = x0; i1 = 2:n+1; i2 = linspace(0,1,n);
    figure; plot(i2, x(i1)); hold on;
    title(['Computer Problem 11.13' part{i} ' -- ' method{i} ', ' str]);
    for p = 1:maxits
      if i == 1
        for m = 2:n+1
          x(m) = (b(m-1)+x(m-1)+x(m+1))/2; % Gauss-Seidel
        end
      else
        x(i1) = (b+x(i1-1)+x(i1+1))/2; % Jacobi
      end
      pause(0.5); plot(i2, x(i1));
    end
  end
end
```

11.14.

```
function [u] = cp11_14 % two-grid method for 1-D Laplace equation
n = 19; tol = 1e-6; maxits = 50; n_pre = 3; n_post = 2; n_coarse = 5;
if mod(n,2) == 0, n = n+1; end; nn = (n-1)/2;
b = rand(n,1); u = zeros(n,1); r = residual(u,b); k = 0;
while norm(r) > tol  &  k < maxits
  k = k+1; u = smooth(u,b,n_pre); r = residual(u,b); r2 = restrict(r);
  e2 = smooth(zeros(nn,1),r2,n_coarse); u = correct(e2,u);
  u = smooth(u,b,n_post); r = residual(u,b);
end

function [r] = residual(x0, b)
n = length(x0); x = zeros(n+2,1); x(2:n+1) = x0; i1 = 2:n+1;
r = b+x(i1-1)-2*x(i1)+x(i1+1);

function [x] = smooth(x0, b, numits)
n = length(x0); x = zeros(n+2,1); x(2:n+1) = x0; i1 = 2:n+1;
for p = 1:numits
  x(i1) = (b+x(i1-1)+x(i1+1))/2; % Jacobi
end; x = x(i1);

function [r2] = restrict(r) % could also use injection, r2 = r(i2);
n = length(r); i2 = 2:2:n-1; r2 = (r(i2-1)+2*r(i2)+r(i2+1))/4; % full weighting

function [u] = correct(e2, u); n = length(u); nn = length(e2); i1 = 1:nn-1;
e1 = zeros(n,1); e1(1) = e2(1)/2; e1(2:2:n-1) = e2;
e1(3:2:n-2) = (e2(i1)+e2(i1+1))/2; e1(n) = e2(nn)/2; u = u+e1;
```

Chapter 12

Fast Fourier Transform

Exercises

12.1. (a) $4\cos(2\pi kt) = 2(e^{2\pi ikt} + e^{-2\pi ikt})$. (b) $6\sin(2\pi kt) = 3i(e^{-2\pi ikt} - e^{2\pi ikt})$.

12.2. (a) $\sin(4\pi t)$. (b) You will always obtain the same value. (c) To obtain an accurate value for the true frequency, the signal would need to be sampled four times per second.

12.3. (a) $\boldsymbol{x} = [\,0 \quad n/2 \quad \ldots \quad 0 \quad n/2\,]^T$. (b) $\boldsymbol{x} = [\,0 \quad -(n/2)i \quad \ldots \quad 0 \quad (n/2)i\,]^T$. (c) $\boldsymbol{x} = [\,0 \quad n/2 - (n/2)i \quad \ldots \quad 0 \quad n/2 + (n/2)i\,]^T$.

12.4. From the formula for the DFT coefficients, $y_0 = \sum_{k=0}^{n-1} x_k \omega_n^0 = \sum_{k=0}^{n-1} x_k$.

12.5. $\boldsymbol{F}_n$ is Hermitian for $n = 1$ and $n = 2$.

12.6. (a) Since $\boldsymbol{F}_n^{-1} = (1/n)\boldsymbol{F}_n^H$, $((1/\sqrt{n})\boldsymbol{F}_n)^{-1} = \sqrt{n}\boldsymbol{F}_n^{-1} = \sqrt{n}(1/n)\boldsymbol{F}_n^H = (1/\sqrt{n})\boldsymbol{F}_n^H = ((1/\sqrt{n})\boldsymbol{F}_n)^H$. Therefore, $(1/\sqrt{n})\boldsymbol{F}_n$ is unitary. (b) If $\boldsymbol{y} = \boldsymbol{F}_n\boldsymbol{x}$, then $\|\boldsymbol{y}\|_2^2 = \boldsymbol{y}^H\boldsymbol{y} = \boldsymbol{x}^H\boldsymbol{F}_n^H\boldsymbol{F}_n\boldsymbol{x} = n\boldsymbol{x}^H((1/\sqrt{n})\boldsymbol{F}_n)^H((1/\sqrt{n})\boldsymbol{F}_n)\boldsymbol{x} = n\boldsymbol{x}^H\boldsymbol{x} = n\|\boldsymbol{x}\|_2^2$.

12.7. From the definition of the DFT we have

$$y_0 = \sum_{k=0}^{n-1} x_k \omega_n^{0\cdot k} = \sum_{k=0}^{n-1} x_k,$$

which is a sum of real numbers, and therefore y_0 is real. Similarly,

$$y_{n/2} = \sum_{k=0}^{n-1} x_k \omega_n^{(n/2)k} = x_0 - x_1 + x_2 - x_3 + \cdots + x_{n-2} - x_{n-1},$$

which again is a sum of real numbers, and therefore $y_{n/2}$ is real.

12.8. The recurrence relation for the running time of the FFT algorithm is $T(n) = 2T(n/2) + cn$, $T(1) = 0$. This recurrence has exact solutions $T(n) = cn \log_2 n$.

12.9.

$$\begin{aligned}
\frac{1}{n}\overline{\mathrm{DFT}(\bar{\boldsymbol{y}})} &= \frac{1}{n}\overline{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)^2} \end{bmatrix} \begin{bmatrix} \bar{y}_0 \\ \bar{y}_1 \\ \bar{y}_2 \\ \vdots \\ \bar{y}_{n-1} \end{bmatrix}} \\
&= \frac{1}{n}\begin{bmatrix} \overline{\overline{y_0} + \overline{y_1} + \overline{y_2} + \cdots + \overline{y_{n-1}}} \\ \overline{\overline{y_0} + \omega^1\overline{y_1} + \omega^2\overline{y_2} + \cdots + \omega^{n-1}\overline{y_{n-1}}} \\ \overline{\overline{y_0} + \omega^2\overline{y_1} + \omega^4\overline{y_2} + \cdots + \omega^{2(n-1)}\overline{y_{n-1}}} \\ \vdots \\ \overline{\overline{y_0} + \omega^{n-1}\overline{y_1} + \omega^{2(n-1)}\overline{y_2} + \cdots + \omega^{(n-1)^2}\overline{y_{n-1}}} \end{bmatrix} \\
&= \frac{1}{n}\begin{bmatrix} y_0 + y_1 + y_2 + \cdots + y_{n-1} \\ y_0 + \omega^1 y_1 + \omega^2 y_2 + \cdots + \omega^{n-1} y_{n-1} \\ y_0 + \omega^2 y_1 + \omega^4 y_2 + \cdots + \omega^{2(n-1)} y_{n-1} \\ \vdots \\ y_0 + \omega^{n-1} y_1 + \omega^{2(n-1)} y_2 + \cdots + \omega^{(n-1)^2} y_{n-1} \end{bmatrix} \\
&= \frac{1}{n}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \cdots & \omega^{-(n-1)^2} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \frac{1}{n}\boldsymbol{F}_n^H \boldsymbol{y},
\end{aligned}$$

so the stated formula indeed gives the inverse DFT.

12.10. Let column k of $\boldsymbol{F}_n$ be denoted by $\{\boldsymbol{F}_n\}_k$, whose entries are given by ω_n^{mk}. Forming $\boldsymbol{A}\{\boldsymbol{F}_n\}_k$, entry m of the resulting vector will be $-\omega_n^{(m-1)k} + 2\omega_n^{mk} - \omega_n^{(m+1)k} = \omega_n^{mk}(2 - \omega_n^{-k} - \omega_n^k)$. Thus $\{\boldsymbol{F}_n\}_k$ is an eigenvector of $\boldsymbol{A}$ with corresponding eigenvalue $(2 - \omega_n^{-k} - \omega_n^k)$.

Computer Problems

12.1.
```
function cp12_01 % experiments with DFT
disp('(a)'); k = (0:7)';
disp('m   __________x__________  _______y_______');
for m = 1:5
  x = cos(m*k*pi); y = fft(x);
  fprintf('%d  %2d %2d %2d %2d %2d %2d %2d %2d  %d %d %d %d %d %d %d %d\n', ...
    m, x, y);
end; disp(' ');
disp('Each sequence is either constant or oscillates at the Nyquist frequency.');
disp(' '); disp('(b)');
```

```
t = linspace(0,7); x1 = cos(pi*t); x2 = cos(3*t); plot(t,x1,'b',t,x2,'r');
xlabel('t'); title('Computer Problem 12.1(b) -- Plot of cos(\pi t) and cos(3t)');
DFT_cos_pik = fft(cos(pi*k)), DFT_cos_3k = fft(cos(3*k))
disp('cos(pi*k) has period 2*pi, whereas cos(3*pi) does not.');
```

12.2. `function cp12_02 % analysis of asteroid orbit data`

```
theta = [0:30:330]'; x = [408 89 -66 10 338 807 1238 1511 1583 1462 1183 804]';
n = 12; A = zeros(n,n); A(:,1) = ones(n,1); A(:,7) = cos(2*pi*6*theta./360);
for k = 1:5
  A(:,k+1) = cos(2*pi*k*theta./360); A(:,k+7) = sin(2*pi*k*theta./360);
end
disp('(a)'); c = A\x; a = c(1:7), b = c(8:12),
t = linspace(0,360,120); y = a(1)+a(7)*cos(2*pi*6.*t./360);
for k = 1:5
  y = y+a(k+1)*cos(2*pi*k.*t./360)+b(k)*sin(2*pi*k.*t./360);
end
plot(t,y,theta,x,'ro'); axis([0 360 -200 1600]); xlabel('\theta'); ylabel('x');
title('Computer Problem 12.2(b) -- Plot of Interpolant to Asteroid Data');
disp('(c)'); y = fft(x), disp(' ');
disp('(d) y(1) = 12*a_0, real(y(2)),...,real(y(6)) = 6*a_1,...,6*a_5,');
disp('    imag(y(2)),...,imag(y(6)) = -6*b_1,...,-6*b_5, y(7) = 12*a_6,');
disp('    and second half of y is conjugate to first half, as expected for');
disp('    real input.');
```

12.3. `function cp12_03 % DFT and amplitude spectrum of shifted sequences`

```
n = 8; x = rand(n,1);
for k = 1:n
  dft = fft(x), amp_spec = abs(dft), x = [x(n); x(1:n-1)];
end
disp('Amplitude spectrum is invariant under circular shifts.');
```

12.4. `function cp12_04 % DFT of sequences of length not a power of two`

```
x = [1; 1; 1; 1; 1]; x_hat = [x; 0; 0; 0]; y = fft(x), y_hat = fft(x_hat)
```

12.5. `function cp12_05 % complexity of FFT`

```
for n = 1:1024
  x = rand(n,1); tic; fft(x); t(n) = toc;
end
semilogy(1:1024,t,'-'); axis tight; xlabel('n'); ylabel('time');
title('Computer Problem 12.5 -- Timing of FFTs of Varying Size');
```

12.6. `function cp12_06(n) % plot basis functions for DFT`

```
F = fft(eye(n));
for j = 1:n
  plot(0:n-1,real(F(:,j)),'r-',0:n-1,imag(F(:,j)),'b-'); legend('real','imag',0);
  title(['Computer Problem 12.6(a) - DFT Basis Function ' sprintf('%d',j-1)]);
  xlabel('index'); ylabel('real and imaginary parts'); pause
end;
for j = 1:n
  plot3(real(F(:,j)),imag(F(:,j)),0:n-1);
  title(['Computer Problem 12.6(b) - DFT Basis Function ' sprintf('%d',j-1)]);
```

```
  xlabel('real part'); ylabel('imaginary part'); zlabel('index'); pause
end;
```

12.7.
```
function cp12_07 % eigenvalues of DFT matrix
for n = 1:16
  F = fft(eye(n)); Lambda = eig((1/sqrt(n))*F)
end
disp('Eigenvalues are +1 with multiplicity ceil((n+1)/4), -1 with multiplicity');
disp('ceil((n-1)/4), i with multiplicity ceil((n-4)/4), and -i with multiplicity');
disp('ceil((n-2)/4)');
for n = 17:20
  F = fft(eye(n)); Lambda = eig((1/sqrt(n))*F)
end
```

12.8.
```
function cp12_08(n) % diagonalize circulant matrix using DFT matrix
v = rand(1,n); C = toeplitz([v(1) v(n:-1:2)],v), F = fft(eye(n)); D = (1/n)*F*C*F'
```

12.9.
```
function cp12_09 % recursive FFT
for j = 1:11
  n(j) = 2^(j-1); x = rand(n(j),1); tic; y = fft(x); tn(j) = toc;
  omega = exp(-2*pi*i/n(j)); tic; y = recursive_fft(x,omega); tr(j) = toc;
end
semilogy(n,tr,'r-',n,tn,'b-'); axis tight;
legend('recursive','nonrecursive',2); xlabel('n'); ylabel('time');
title('Computer Problem 12.9 -- Timing of Recursive and Nonrecursive FFT');

function [y] = recursive_fft(x, omega)
n = length(x); y = zeros(n,1);
if(n == 1)
  y(1) = x(1);
else
  for k = 1:(n/2)
    p(k) = x(2*k-1); s(k) = x(2*k);
  end
  q = recursive_fft(p,omega^2); t = recursive_fft(s,omega^2);
  for k = 1:n
    y(k) = q(mod(k-1,n/2)+1)+omega^(k-1)*t(mod(k-1,n/2)+1);
  end
end
```

12.10.
```
function cp12_10 % DFT via matrix-vector multiplication
k = 512; n = zeros(k,1); tf = zeros(k,1); tm = zeros(k,1);
for n = 1:k
  x = rand(n,1); tic; y = fft(x); tf(n) = toc;
  F = fft(eye(n)); tic; y = F*x; tm(n) = toc;
end
semilogy(1:k,tm,'r-',1:k,tf,'b-'); axis tight;
legend('mat-vec mult','FFT',2); xlabel('n'); ylabel('time');
title('Computer Problem 12.10 -- Timing of Matrix-Vector Multiply vs. FFT');
```

12.11.
```
function cp12_11 % digital filtering using FFT
```

```
n = 64; x = ones(n,1)+randn(n,1); xf = digital_filter(x);
plot(1:n,x,'b-',1:n,xf,'r-'); axis tight; xlabel('index');
ylabel('sequence value'); legend('original sequence','filtered sequence',0);
title('Computer Problem 12.11 -- FFT Digital Filter');

function [xf] = digital_filter(x); n = length(x); y = fft(x);
for k = 1:floor(3*n/8)
  y(floor(n/2)-k+2) = 0; y(floor(n/2)+k) = 0;
end; xf = real(ifft(y));
```

12.12.
```
function cp12_12(n) % compute convolution of two sequences using FFT
u = rand(n,1), v = rand(n,1), z = convolution(u,v)

function [z] = convolution(u, v); z = ifft(fft(u).*fft(v));
```

12.13.
```
function cp12_13 % fast polynomial multiplication using FFT
p = [0 1], q = [0 0 1], x = poly_mult(p,q)
p = [1 1 1 1], q = [1 1], x = poly_mult(p,q)

function [x] = poly_mult(p, q)
n = length(p)+length(q); yp = fft([p zeros(1,n-length(p))]);
yq = fft([q zeros(1,n-length(q))]); x = ifft(yp.*yq);
```

12.14.
```
function cp12_14 % power spectrum of Gauss-Seidel and Jacobi smoothers
n = 64; b = zeros(n,1); maxits = 10;
part = {'(a)' '(b)'}; method = {'Gauss-Seidel' 'Jacobi'};
for i = 1:2
  x0 = ones(n,1); x = zeros(n+2,1); x(2:n+1) = x0;
  i1 = 2:n+1; i2 = linspace(0,1,n); figure;
  for p = 1:maxits
    if i == 1
      for m = 2:n+1
        x(m) = (b(m-1)+x(m-1)+x(m+1))/2; % Gauss-Seidel
      end
    else
      x(i1) = (b+x(i1-1)+x(i1+1))/2; % Jacobi
    end
    y = fft(x(2:n+1)); p = y.*conj(y); semilogy(2:n/2, p(2:n/2));
    axis([2 32 1e-6 1e2]); hold on; pause(1);
  end
  title(['Computer Problem 12.14' part{i} ' -- Power Spectrum for ' method{i}]);
end
```

Chapter 13

Random Numbers and Stochastic Simulation

Exercises

13.1. (a) Note that $x_k = a^k$ for $k \leq 13$ and $x_k = 0$ for $k > 13$. Therefore, the period is 1 after 14 iterations. (b) For $a = 125$, $x_{2048} = x_0 = 1$. Therefore, the period is 2048. (c) The longest period is at most 8192. However, since $b = 0$ and $M = 8192$, an even value for a will generate a sequence of even numbers and an odd value for a will generate a sequence of odd numbers. For even numbers, they factor to $2n$ for some number n. After at most thirteen iterations, this becomes $2^{13}n = 8192n$ which is 0 modulo 8192. Therefore, the longest sequence must exist in the odd numbers, with a period no more than 4096. An exhaustive search of the entire domain of a yields a longest period of 2048.

13.2. (a) Well-structured sports such as baseball or football break down into a well-defined sequence of discrete steps, each with a fairly limited number of possible outcomes. A baseball game, for example, is composed of nine innings, each of which is composed of three outs for each team. A given inning is composed of a sequence of batters, each of whom faces a sequence of pitches. Thus, a baseball game can be simulated at a pitch-by-pitch level. The random choices might include the type of pitch thrown for each pitch (fastball, curve, etc.), and the resulting outcome could be determined probabilistically from a given pitcher's known effectiveness (balls vs. strikes, walks vs. strikeouts, etc.) and each batter's known effectiveness (batting average, etc.). Similarly, football breaks down into a sequence of possessions by each team, which are further subdivided into downs, and each down offers a somewhat limited number of options (run, pass, punt, etc.). The random choices might include the type of play run on each down, and the resulting outcome could be determined probabilistically from known statistics for the quarterbacks, running backs, etc.

(*b*) Less well-structured sports such as basketball, soccer, and hockey do not break down so readily into a well-defined sequence of discrete steps with options and outcomes that are easily characterized. In basketball, for example, teams alternate possessions after made baskets, but unlike baseball or football, these possessions do not break down into discrete substeps, and play between baskets is relatively unstructured. Such play could still be modeled stochastically, but at a much more gross level.

Computer Problems

13.1.
```
function cp13_01 % chi-square test of random number generators
disp('(a) MATLAB built-in function rand');
for n=10:10:20
  fprintf('n = %d\n',n); disp('  k    chi-sq');
  for k=100:100:1000
    x = rand(k,1); chi_square(x,n,k);
  end; disp(' ');
end
global a b M s; a = 125; b = 0; M = 8192; s = 1;
fprintf('(b) Linear congruential generator with a = %d, b = %d, M = %d\n',a,b,M);
for n=10:10:20
  fprintf('n = %d\n',n); disp('  k    chi-sq');
  for k=100:100:1000
    x = lin_cong(k,1); chi_square(x,n,k);
  end; disp(' ');
end
a = 7^5; b = 0; M = 2^31-1; s = 1;
fprintf('Linear congruential generator with a = %d, b = %d, M = %d\n',a,b,M);
for n=10:10:20
  fprintf('n = %d\n',n); disp('  k    chi-sq');
  for k=100:100:1000
    x = lin_cong(k,1); chi_square(x,n,k);
  end; disp(' ');
end

function chi_square(x, n, k)
count = zeros(n,1);
for i=1:k
  bin = floor(x(i)*n)+1; count(bin) = count(bin)+1;
end
chisq = 0;
for i=1:n
  chisq = chisq+((count(i)-k/n)^2)/(k/n);
end; fprintf('%4d %8.4f\n',k,chisq);

function [x] = lin_cong(m, n);
global a b M s; x = zeros(m,n);
for j = 1:n
  for i = 1:m
    s = mod(a*s+b,M); x(i,j) = s/M;
```

```
  end
end
```

13.2. `function cp13_02 % plot pairs of random numbers`

```
global a b M s; a = 125; b = 0; M = 8192; s = 1;
for n = [100 1000]
  x = lin_cong(2*n,1); y = reshape(x,2,n); figure; plot(y(1,:),y(2,:),'.');
  title(['Computer Problem 13.2(a) -- ' sprintf('%d',n) ' Random Pairs']);
  xlabel([sprintf('a = %d, b = %d, M = %d',a,b,M)]);
end
a = 7^5; b = 0; M = 2^31-1; s = 1;
for n = [100 1000]
  x = lin_cong(2*n,1); y = reshape(x,2,n); figure; plot(y(1,:),y(2,:),'.');
  title(['Computer Problem 13.2(b) -- ' sprintf('%d',n) ' Random Pairs']);
  xlabel([sprintf('a = %d, b = %d, M = %d',a,b,M)]);
end
for n = [100 1000]
  x = rand(2*n,1); y = reshape(x,2,n); figure; plot(y(1,:),y(2,:),'.');
  title(['Computer Problem 13.2(c) -- ' sprintf('%d',n) ' Random Pairs']);
  xlabel('MATLAB built-in function rand');
end

function [x] = lin_cong(m, n);
global a b M s; x = zeros(m,n);
for j = 1:n
  for i = 1:m
    s = mod(a*s+b,M); x(i,j) = s/M;
  end
end
```

13.3. `function cp13_03 % DFT of random sequences`

```
global a b M s; a = 125; b = 0; M = 8192; s = 1; n = 1024;
x = lin_cong(n,1); y = fft(x);
figure; plot(2:n,real(y(2:n)),'b-'); hold on; plot(2:n,imag(y(2:n)),'r-');
axis tight; title('Computer Problem 13.3(a) -- DFT of Random Sequence');
xlabel([sprintf('a = %d, b = %d, M = %d',a,b,M)]); legend('real','imag',0);
a = 7^5; b = 0; M = 2^31-1; s = 1; x = lin_cong(n,1); y = fft(x);
figure; plot(2:n,real(y(2:n)),'b-'); hold on; plot(2:n,imag(y(2:n)),'r-');
axis tight; title('Computer Problem 13.3(b) -- DFT of Random Sequence');
xlabel([sprintf('a = %d, b = %d, M = %d',a,b,M)]); legend('real','imag',0);
x = rand(n,1); y = fft(x);
figure; plot(2:n,real(y(2:n)),'b-'); hold on; plot(2:n,imag(y(2:n)),'r-');
axis tight; title('Computer Problem 13.3(c) -- DFT of Random Sequence');
xlabel('MATLAB built-in function rand'); legend('real','imag',0);

function [x] = lin_cong(m, n);
global a b M s; x = zeros(m,n);
for j = 1:n
  for i = 1:m
    s = mod(a*s+b,M); x(i,j) = s/M;
  end
```

```
end
```

13.4.
```
function cp13_04 % mean and variance of random sequence
global a b M s; a = 125; b = 0; M = 8192; s = 1; n = 1000;
disp('Ideal values'); mu = 1/2, sigmasq = 1/12
fprintf('(a) Linear congruential generator with a = %d, b = %d, M = %d\n',a,b,M);
x = lin_cong(n,1); mu = mean(x), sigmasq = var(x)
a = 7^5; b = 0; M = 2^31-1; s = 1;
fprintf('(b) Linear congruential generator with a = %d, b = %d, M = %d\n',a,b,M);
x = lin_cong(n,1); mu = mean(x), sigmasq = var(x)
disp('(c) MATLAB built-in function rand');
x = rand(n,1); mu = mean(x), sigmasq = var(x)

function [x] = lin_cong(m, n);
global a b M s; x = zeros(m,n);
for j = 1:n
  for i = 1:m
    s = mod(a*s+b,M); x(i,j) = s/M;
  end
end
```

13.5.
```
function cp13_05 % poker test of random sequence
global a b M s; a = 125; b = 0; M = 8192; s = 1;
expected = [0.0001 0.0045 0.0090 0.0720 0.1080 0.5040 0.3024];
name = {'Five of a kind ' 'Four of a kind ' 'Full house     ' ...
  'Three of a kind' 'Two pairs      ' 'One pair       ' 'Bust           '};
disp('(b) MATLAB built-in function rand');
for n = [1000 10000];
  hand = zeros(7,1); x = rand(5,n); fprintf('n = %d\n',n);
  disp('Hand              Expected  Computed');
  for i=1:n
    k = poker_test(x(:,i)); hand(k) = hand(k)+1;
  end; hand = hand/n;
  for k = 1:7
    fprintf('%s %9.4f %9.4f\n', name{k}, expected(k), hand(k));
  end; disp(' ');
end
fprintf('(c) Linear congruential generator with a = %d, b = %d, M = %d\n',a,b,M)
;
for n = [1000 10000];
  hand = zeros(7,1); x = lin_cong(5,n); fprintf('n = %d\n',n);
  disp('Hand              Expected  Computed');
  for i=1:n
    k = poker_test(x(:,i)); hand(k) = hand(k)+1;
  end; hand = hand/n;
  for k = 1:7
    fprintf('%s %9.4f %9.4f\n', name{k}, expected(k), hand(k));
  end; disp(' ');
end

function [k] = poker_test(x)
```

```
x = sort(floor(10*x)); if x(1) == x(5), k = 1; return; end;          % 5 of kind
if x(1) == x(4) | x(2) == x(5), k = 2; return; end;                  % 4 of kind
if x(1) == x(3) & x(4) == x(5), k = 3; return; end;                  % full house
if x(1) == x(2) & x(3) == x(5), k = 3; return; end;                  % full house
if x(1) == x(3) | x(2) == x(4) | x(3) == x(5), k = 4; return; end;   % 3 of kind
if x(1) == x(2) & (x(3) == x(4) | x(4) == x(5)), k = 5; return; end; % 2 pairs
if x(2) == x(3) & x(4) == x(5), k = 5; return; end;                  % 2 pairs
if x(1) == x(2) | x(2) == x(3), k = 6; return; end;                  % 1 pair
if x(3) == x(4) | x(4) == x(5), k = 6; return; end;                  % 1 pair
k = 7;                                                               % bust

function [x] = lin_cong(m, n);
global a b M s; x = zeros(m,n);
for j = 1:n
  for i = 1:m
    s = mod(a*s+b,M); x(i,j) = s/M;
  end
end
```

13.6.
```
function cp13_06 % birthday paradox
ntrials = 1000;
for k = 2:366
  count = 0;
  for i = 1:ntrials
    x = rand(k,1); x = sort(floor(365*x));
    if min(x(2:k)-x(1:k-1)) == 0, count = count+1; end
  end
  if 2*count > ntrials, break; end
end
fprintf('Probability of duplicate birthdays exceeds 0.5 for %d or more people.',k);
```

13.7.
```
function cp13_07 % probability that a quadratic equation has real roots
ntrials = 1000; x = 2*rand(ntrials,3)-1; d = x(:,2).^2-4*x(:,1).*x(:,3);
fprintf('Probability of real roots is %5.4f\n',sum(d >= 0)/ntrials);
```

13.8.
```
function cp13_08 % generate normally distributed random numbers
n = 1000; expected = [2.14 13.59 34.13 34.13 13.59 2.14];
fprintf('Expected     %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\n',expected);
y = method_a(n); h = 100*histc(y,[-inf -3 -2 -1 0 1 2 3 inf])/n;
fprintf('Method a     %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\n',h(2:7));
y = method_b(n); h = 100*histc(y,[-inf -3 -2 -1 0 1 2 3 inf])/n;
fprintf('Method b     %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\n',h(2:7));
y = method_c(n); h = 100*histc(y,[-inf -3 -2 -1 0 1 2 3 inf])/n;
fprintf('Method c     %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\n',h(2:7));
y = randn(n,1); h = 100*histc(y,[-inf -3 -2 -1 0 1 2 3 inf])/n;
fprintf('MATLAB randn %5.2f  %5.2f  %5.2f  %5.2f  %5.2f  %5.2f\n',h(2:7));

function [y] = method_a(n)
y = sum(rand(n,12),2)-6;

function [y] = method_b(n)
```

```
x1 = rand(n/2,1); x2 = rand(n/2,1);
y = [sin(2*pi*x1).*sqrt(-2*log(x2)); cos(2*pi*x1).*sqrt(-2*log(x2))];

function [y] = method_c(n)
y = zeros(n,1);
for i = 1:2:n
  r = 2;
  while r > 1
    x = 2*rand(2,1)-1; r = x(1)^2+x(2)^2;
  end
  y(i) = x(1)*sqrt(-2*log(r)/r); y(i+1) = x(2)*sqrt(-2*log(r)/r);
end
```

13.9.
```
function cp13_09 % Buffon needle problem
disp('  Trials  Approx Pi  Digits');
for k = 1:6
  n = 10^k; theta = rand(n,1)*pi/2; y = rand(n,1); p = 2*n/sum(y <= sin(theta));
  digits = -log10(abs(p-pi)); fprintf('%8d %8.4f   %6.2f\n',n,p,digits);
end
```

13.10.
```
function cp13_10 % Monte Carlo approximation to area and volume
r = 0.5^2; fprintf('(a) True Area = %8.6f\n',pi/4);
disp('  Trials  Approx Area   Error     0.1/sqrt(n)');
for k = 1:6
  n = 10^k; x = rand(n,2)-0.5; a = sum((x(:,1).^2+x(:,2).^2) <= r)/n;
  err = abs(a-pi/4); fprintf('%8d %10.6f   %10.4e  %10.4e\n',n,a,err,0.1/sqrt(n));
end; disp(' ');
fprintf('(b) True Volume = %8.6f\n',pi/6);
disp('  Trials  Approx Vol    Error     0.1/sqrt(n)');
for k = 1:6
  n = 10^k; x = rand(n,3)-0.5; a = sum((x(:,1).^2+x(:,2).^2+x(:,3).^2) <= r)/n;
  err = abs(a-pi/6); fprintf('%8d %10.6f   %10.4e  %10.4e\n',n,a,err,0.1/sqrt(n));
end; disp('(c) Error goes to zero roughly as 1/sqrt(n), where n = number of trials.');
```

13.11.
```
function cp13_11 % simulation of Markov chain
A = [0.8 0.2 0.1; 0.1 0.7 0.3; 0.1 0.1 0.6]; m = 1000; k = 10; count = zeros(3,1);
for i = 1:m
  b = rand(k,1); state = 1;
  for j = 1:k
    new_state = 0;
    while b(j) > 0
      new_state = new_state+1; b(j) = b(j)-A(new_state,state);
    end; state = new_state;
  end; count(state) = count(state)+1;
end; disp('Probability distribution vector:'); x = count/m
```

13.12.
```
function cp13_12 % simulation of radioactive decay
lambda = 0.1; m = 50;
for k = 1:4
  n = 10^k; x = ones(n,1); y = lambda*x; r = zeros(m,1);
  for i = 1:m
```

```
    z = rand(n,1); x = x.*(z >= y); r(i) = sum(x);
  end;
  e = n*exp(-lambda*(1:m)'); figure; plot(1:m,r,'b-',1:m,e,'r-');
  xlabel('time increments'); ylabel('atoms remaining');
  legend('(a) stochastic simulation','(b) exponential model');
  title('Computer Problem 13.12 -- Simulation of Radioactive Decay');
end
```

13.13.
```
function cp13_13 % one-dimensional random walk
ntrials = 20; disp('    n   avg dist');
for k = 1:4
  n = 10^k; d = 0;
  for i = 1:ntrials
    x = rand(n,1)-0.5; y = sign(x); d = d+abs(sum(y));
  end;
  fprintf('%6d   %6.2f\n', n, d/ntrials);
end; disp(' '); disp('Average distance grows roughly as sqrt(n)');
```

13.14.
```
function cp13_14 % two-dimensional random walk for solving Laplace equation
n = 10000; u = zeros(4,1); u(1) = walk(1,1,n); u(2) = walk(2,1,n);
u(3) = walk(1,2,n); u(4) = walk(2,2,n); u

function [u] = walk(x0, y0, n); u = 0;
for i=1:n
  x = x0; y = y0; z = 0;
  while (x == 1 | x == 2) & (y == 1 | y == 2)
    dir = rand; if dir < 0.25, x = x-1; elseif dir < 0.5, x = x+1;
    elseif dir < 0.75, y = y-1; else y = y+1; end; if y == 3, z = 1; end
  end; u = u+z;
end; u = u/n;
```

13.15.
```
function cp13_15 % random simulation of polymer chains
disp('(a) Two-dimensional polymer chain'); disp('    n   distance');
for k = 1:3
  n = 10^k; x = zeros(n,1); y = zeros(n,1);
  for i=2:n
    theta = 2*pi*rand; x(i) = x(i-1)+cos(theta); y(i) = y(i-1)+sin(theta);
  end; d = sqrt(x(n)^2+y(n)^2); fprintf('%6d   %6.2f\n', n, d);
  figure; plot(x,y,'bo-');
  title(['Computer Problem 13.15(a) -- 2-D Polymer for n = ' sprintf('%d',n)]);
end
disp('(b) Three-dimensional polymer chain'); disp('    n   distance');
for k = 1:3
  n = 10^k; x = zeros(n,1); y = zeros(n,1); z = zeros(n,1);
  for i=2:n
    theta = 2*pi*rand; phi = 2*pi*rand; x(i) = x(i-1)+cos(theta)*sin(phi);
    y(i) = y(i-1)+sin(theta)*sin(phi); z(i) = z(i-1)+cos(phi);
  end; d = sqrt(x(n)^2+y(n)^2+z(n)^2);
  fprintf('%6d   %6.2f\n', n, d); figure; plot3(x,y,z,'bo-');
  title(['Computer Problem 13.15(b) -- 3-D Polymer for n = ' sprintf('%d',n)]);
end
```

```
disp('(c) Polyethylene polymer chain'); disp('    n   distance');
theta = 109*pi/180;
for k = 1:3
  n = 10^k; x = zeros(n,1); y = zeros(n,1); z = zeros(n,1);
  for i=2:n
    phi = 2*pi*rand; x(i) = x(i-1)+cos(theta)*sin(phi);
    y(i) = y(i-1)+sin(theta)*sin(phi); z(i) = z(i-1)+cos(phi);
  end; d = sqrt(x(n)^2+y(n)^2+z(n)^2);
  fprintf('%6d   %6.2f\n', n, d); figure; plot3(x,y,z,'bo-');
  title(['Computer Problem 13.15(c) -- Polyethylene for n = ' sprintf('%d',n)]);
end
```

13.16.
```
function cp13_16 % random simulation of neutron shielding
disp('Two-dimensional wall'); disp('     n    percent escaping');
for k = 1:4
  n = 10^k; count = 0;
  for j = 1:n
    i = 1; theta = pi*rand-pi/2; x = cos(theta); y = sin(theta);
    while (i < 8)  &  (x < 4)  &  (x > 0)
      i = i+1; theta = 2*pi*rand; x = x+cos(theta); y = y+sin(theta);
    end; if x > 4, count = count+1; end;
  end; fprintf('%6d   %10.2f\n', n, 100*count/n);
end; disp(' ');
disp('Three-dimensional wall'); disp('     n    percent escaping');
for k = 1:4
  n = 10^k; count = 0;
  for j = 1:n
    i = 1; theta = pi*rand-pi/2; phi = pi*rand; x = cos(theta)*sin(phi);
    y = sin(theta)*sin(phi); z = cos(phi);
    while i < 8  &  x < 4  &  x > 0
      i = i+1; theta = 2*pi*rand; phi = pi*rand; x = x+cos(theta)*sin(phi);
      y = y+sin(theta)*sin(phi); z = z+cos(phi);
    end; if x > 4, count = count+1; end;
  end; fprintf('%6d   %10.2f\n', n, 100*count/n);
end
```

13.17. No quasi-random number generator is conveniently available in MATLAB, so this solution is omitted.

13.18.
```
function cp13_18 % Monte Carlo integration of double integral
disp('Using MATLAB function rand:'); disp('     n      I_a        I_b');
for k = 1:4
  n = 10^k; x = rand(n,1); y = rand(n,1); f = exp(-x.*y); z = x.^2+y.^2 <= 1;
  fprintf('%6d %10.5f %10.5f\n',n,mean(f),(sum(z.*f)/sum(z))*pi/4);
end
```