Solutions to Selected Problems

Chapter 1

1.1. **(a)** Since some x are negative, the corresponding square roots are imaginary and $\imath = \sqrt{-1}$ is used.
(b) In executing `x./y`, the divide by zero produces the symbol ∞ and a warning.

1.2. **(b)** Note that `t2` is identical to `c` but `t1` is not since the `sqrt` function gives the square root of the individual elements of `c`.

1.4. $x = 2.4545$, $y = 1.4545$, $z = -0.2727$. Note that when using the `/` operator the solution is given by `x=b'/a'`.

1.8. The plot does not truly represent the function $\cos(x^3)$ because there are insufficient plotting points.

1.9. The function `fplot` automatically adjusts to provide a smoother plot. However, changing `x` to `-2:0.01:2` gives a similar quality graph using the function `plot`.

1.12. $x = 1.6180$.

1.14. Using $x_1 = 1, x_2 = 2, \ldots, x_6 = 6$, a suitable script is

```
n = 6; x = 1:n;
for j = 1:n,
    p(j) = 1;
    for i = 1:n
        if i~=j
            p(j) = p(j)*x(i);
        end
    end
end
p
```

1.15. A suitable script is

```
x = 0.82; tol = 0.005; s = x; i = 2; term = x;
while abs(term)>tol
    term = -term*x; s = s+term/i; i = i+1;
end
s, log(1+x)
```

Note: The scripts may have been compressed to save space.

1.17. The form of the function is

```
function [x1,x2] = funct1(a,b,c)
d = b*b-4*a*c;
if d==0
    x1 = -b/(2*a); x2 = x1;
else
    x1 = (-b+sqrt(d))/(2*a); x2 = (-b-sqrt(d))/(2*a);
end
```

1.18. A possible script is

```
function [x1,x2] = funct2(a,b,c)
if a~= 0
    %as in problem 1.17
else
    disp('warning only one root'); x1 = -c/b; x2 = x1;
end
```

1.19. The graph provides an initial approximation of 1.5. Use the function call `fzero('funct3',1.5)` to obtain the root as 1.2512.

1.20. A possible script is

```
x=[ ]; x(1) = 1873;
c = 1; xc = x(1);
while xc>1
    if (x(c)/2)==floor(x(c)/2)
        x(c+1) = (x(c))/2;
    else
        x(c+1) = 3*x(c)+1;
    end
    xc = x(c+1); c = c+1;
    if c>1000
        break
    end
end
plot(x)
```

Try different values for `x(1)`. For example, 1173, 1409, and so on.

1.21. A possible script is

```
x = -4:0.1:4; y = -4:0.1:4;
[x,y] = meshgrid(-4:0.1:4,-4:0.1:4);
p = x.^2+y.^2;
z = (1-x.^2).*exp(-p)-p.*exp(-p)-exp(-(x+1).^2-y.^2);
```

```
subplot(3,1,1)
mesh(x,y,z)
xlabel('x'), ylabel('y'), zlabel('z')
title('mesh')
subplot(3,1,2)
surf(x,y,z)
xlabel('x'), ylabel('y'), zlabel('z')
title('surf')
subplot(3,1,3)
mesh(x,y,z)
xlabel('x'), ylabel('y'), zlabel('z')
title('contour')
```

1.22. A possible script is

```
clf
a = 11; b= 6;
t = -20:0.1:20;
% Cycloid
x = a*(t-sin(t));y=a*(1-cos(t));
subplot(3,1,1), plot(x,y)
xlabel('x-xis'), ylabel('y-xis'), title('Cycloid')
% witch of agnesi
x1 = 2*a*t;y1=2*a./(1+t.^2);
subplot(3,1,2), plot(x1,y1)
xlabel('x-xis'), ylabel('y-xis')
title('witch of agnesi')
% Complex structure
x2 = a*cos(t)-b*cos(a/b*t);
y2 = a*sin(t)-b*sin(a/b*t);
subplot(3,1,3), plot(x2,y2)
xlabel('x-xis'), ylabel('y-xis')
title('Complex structure')
```

1.23. A possible function is

```
function r = zetainf(s,acc)
sum = 0; n = 1; term = 1+acc;
while abs(term)>acc
    term = 1/n.^s;
    sum = sum +term;
    n = n+1;
end
r = sum;
```

1.24. A possible function is

```
function res = sumfac(n)
sum = 0;
for i = 1:n
    sum = sum+i^2/factorial(i);
end
res = sum;
```

1.26. A possible script is

```
rho1 = [zeros(2), eye(2); eye(2), zeros(2)]
rho2 = [zeros(2), i*eye(2); -i*eye(2), zeros(2)]
rho3 = [eye(2), zeros(2); zeros(2), -eye(2)]
q1 = [zeros(4)  rho1;-rho1  zeros(4)]
q1 = [zeros(4)  rho2;-rho2  zeros(4)]
q1 = [zeros(4)  rho3;-rho3  zeros(4)]
```

1.27. A possible script is

```
x = -4:0.001:4;
y = 1./(((x+2.5).^2).*((x-3.5).^2));
plot(x,y)
ylim([0,20])
xlim([-3,-2])
```

1.28. A possible script is

```
y = @(x)x.^2.*cos(1+x.^2);
y1 = @(x) (1+exp(x))./(cos(x)+sin(x));
x = 0:0.1:2;
subplot(1,2,1), plot(x,y(x))
xlabel('x'), ylabel('y')
subplot(1,2,2), plot(x,y1(x))
xlabel('x'), ylabel('y')
```

Chapter 2

2.1. For $n=5$, the norm of $\mathbf{P}-\mathbf{R}$ is large. For $n=6$ the norm of $\mathbf{Q}-\mathbf{R}$ is large. Different versions of MATLAB and possibly diffrent platforms give slightly different results for this problem but the trends in the results and the conclusions that may be drawn are not affected.

Note the large error in the inverse of the square of the Hilbert matrix when $n=6$.

2.2. For $n = 3$, 4, 5, and 6, the answers are 2.7464×10^5, 2.4068×10^8, 2.2715×10^{11}, and 2.2341×10^{14}, respectively. The large errors in Problem 2.1 arise from the fact that the Hilbert matrix is very ill-conditioned, as shown by these results.

2.3. For example, taking $n = 5$, $a = 0.2$, and $b = 0.1, a + 2b < 1$ and maximum error in the matrix coefficients is 1.0412×10^{-5}. Taking $n = 5$, $a = 0.3$, $b = 0.5$, $a + 2b > 1$ and after 10 terms, maximum error in the matrix coefficients is 10.8770. After 20 terms, maximum error is 50.5327, clearly diverging.

2.4. The eigenvalues are $5, 2 + 2\imath$, and $2 - 2\imath$. Thus taking $\lambda = 5$ in the matrix $(\mathbf{A} - \lambda\mathbf{I})$ and finding the RREF gives

$$\mathbf{p} = \begin{bmatrix} 1 & 0 & -1.3529 \\ 0 & 1 & 0.6471 \\ 0 & 0 & 0 \end{bmatrix}$$

Hence $\mathbf{px} = \mathbf{0}$. Solving this gives $x_1 = 1.3529x_3$, $x_2 = -0.6471x_3$, and x_3 is arbitrary.

2.6. $\mathrm{x}^\top = [0.9500\ 0.9811\ 0.9727]$. All methods give the identical solution. Note that if `[q,r] = qr(a)` and `y = q'*b;`, then `x = r(1:3,1:3)\y(1:3)`.

2.7. The solution is $[0\ \ 0\ \ 0\ \ 0\ \ \ldots\ n+1]$.

2.10. For $n = 20$ the condition number is 178.0643; the theoretical condition number is 162.1139. For $n = 50$ the condition number is 1053.5; the theoretical condition number is 1013.2.

2.11. The right vectors are

$$\begin{bmatrix} 0.0484 + 0.4447\imath \\ -0.3962 + 0.4930\imath \\ 0.4930 + 0.3962\imath \end{bmatrix} \quad \begin{bmatrix} 0.0484 - 0.4447\imath \\ -0.3962 - 0.4930\imath \\ 0.4930 - 0.3962\imath \end{bmatrix} \quad \begin{bmatrix} 0.4082 \\ 0.8165 \\ 0.4082 \end{bmatrix}$$

The corresponding eigenvalues are $2 + 4\imath$, $2 - 4\imath$, and 1. The left vectors are obtained by using the function `eig` on the transposed matrix.

2.12. **(a)** The largest eigenvalue is 242.9773.
(b) The eigenvalue nearest 100 is 112.1542.
(c) The smallest eigenvalue is 77.6972.

2.14. For $n = 5$, the largest eigenvalue is 12.3435 and the smallest eigenvalue is 0.2716. For $n = 50$, the largest eigenvalue is 1.0337×10^3 and the smallest eigenvalue is 0.2502.

2.15. Using the function `roots` we compute the eigenvalues 22.9714, -11.9714, $1.0206 \pm 0.0086\imath$, $1.0083 \pm 0.0206\imath$, $0.9914 \pm 0.0202\imath$, and $0.9798 \pm 0.0083\imath$. Using the function `eig` we have 22.9714, -11.9714, 1, 1, 1, 1, 1, 1, 1, and 1. This is a more accurate solution.

2.16. Both `eig` and `roots` give results that only differ by less than 1×10^{-10}. The eigenvalues are 242.9773, 77.6972, 112.1542, 167.4849, and 134.6865.

2.17. The sum of eigenvalues is 55; the product of eigenvalues is 1.

2.18. $c = 0.641n^{1.8863}$.

2.19. A suitable function is

```
function appinv = invapprox(A,k)
ev = eig(A);
evm = max(ev);
if abs(evm)>1
    disp('Method fails')
    appinv = eye(size(A));
else
    appinv = eye(size(A));
    for i = 1:k
        appinv = appinv+A^i;
    end
end
```

2.20. The MATLAB operator gives a much better result. A suitable function is

```
function [res1,res2, nv1,nv2] = udsys(A,b)
newA = A'*A; newb = A'*b;
x1 = inv(newA)*newb;
nv1 = norm(A*x1-b);
x2 = A\b;
nv2 = norm(A*x2-b);
res1 = x1; res2 = x2;
```

2.22. The exact solution is

$$\mathbf{x} = [-12.5\ -24\ -34\ -42\ -47.5\ -50\ -49\ -44\ -34.5\ -20]^\top.$$

The Gauss–Seidel method requires 149 iterations and the Jacobi method requires 283 iterations to give the result to the required accuracy.

Chapter 3

3.2. The solution is 27.8235.

3.3. The solutions are -2 and 1.6344.

3.4. For $c = 5$, with the initial approximation 1.3 or 1.4, the root 1.3735 is obtained after two or three iterations. When $c = 10$, with the initial approximation 1.4, the root 1.4711 is obtained after five iterations. With initial approximation 1.3, convergence is to 130.2764 after 25 iterations. This is a root, but the discontinuity in the function has degraded the performance of the Newton algorithm.

3.5. Schroder's method provides the solution $x = 1.0000$ in one iteration, but Newton's method gives $x = 0.9991$and requires 36 iterations. The solution obtained by Schroder's method is more accurate.

3.6. The equation can be rearranged into the form $x = \exp(x/10)$. Iteration gives $x = 1.1183$. There may be other successful rearrangements.

3.7. The solution is $E = 0.1280$.

3.8. The answers are 2.1602×10^{-16} and -7.1861×10^{-17} for initial values 1 and -1.5, respectively. The exact solution is clearly 0, but this is a difficult problem.

3.9. The three answers are 1.4299, 1.4468, and 1.4458, which are obtained for four, five, and six terms, respectively. These answers are converging to the correct answer.

3.10. Both approaches give identical results, $x = 8.2183$, $y = 2.2747$. The single variable function is $x/5 - \cos x = 2$. Alternatively, the following call can be used:

```
newtonmv([1 1]','p310','p310d',2,1e-4)
```

It requires the functions and derivatives to be defined as follows:

```
function v = p310(x)
v = zeros(2,1);
v(1) = exp(x(1)/10)-x(2);
v(2) = 2*log(x(2))-cos(x(1))-2;

function vd = p310d(x)
vd = zeros(2,2);
vd(1,:) = [exp(x(1)/10)/10 -1];
vd(2,:) = [sin(x(1)) 2/x(2)];
```

3.11. The solution given by `broyden` is $x = 0.1605$, $y = 0.4931$.

3.12. A solution is $x = 0.9397$, $y = 0.3420$. The MATLAB function `newtonmv` requires 7 iterations; `broyden` requires 33.

3.14. The five roots are $1, -\imath, \imath, -\sqrt{2}, \sqrt{2}$.

3.15. The solution is $x = -0.1737 - 0.9848\imath$, $0.9397 + 0.3420\imath$, and $-0.7660 + 0.6428\imath$. This is identical to the exact answer.

3.16. The MATLAB function required is

```
function v = jarrett(f,x1,x2,tol)
gamma = 0.5; d = 1;
while abs(d)>to
    f2 = feval(f,x2);f1=feval(f,x1);
    df = (f2-f1)/(x2-x1); x3 = x2-f2/df; d = x2-x3;
    if f1*f2>0
        x2 = x1; f2 = gamma*f1;
    end
    x2 = x3
end
```

3.17. The third-order method provides the required accuracy after seven iterations. The second-order method requires ten iterations.

3.18. The graphs show that for $c = 2.8$ there is convergence to a single solution, for $c = 3.25$ the iteration oscillates between two values, for $c = 3.5$ the iteration oscillates between four values, and for $c = 3.8$ there is chaotic oscillation between many values.

3.20. Here is an example with p and q chosen to give real roots:

```
>> p=2.5; q = -1; if p^3/q^2>27/4, r = roots([1 0 -p -q]), end

r =
   -1.7523
    1.3200
    0.4323
```

3.21. The commands to solve this problem are

```
>> y1 = roots([1 0 6 -60 36])

y1 =
  -1.8721 + 3.8101i
  -1.8721 - 3.8101i
   3.0999
   0.6444

>> y = y1(3:4)'

y =
    3.0999    0.6444
```

```
>> x = 6./y

x =
    1.9356    9.3110

>> z = 10-x-y

z =
    4.9646    0.0446
```

3.22. A script to solve this problem is

```
c1=(sinh(x)+sin(x))./(2*x);
c3=(sinh(x)-sin(x))./(2*x.^3);
fzero(@ (x) c1^2-x.^4*c3.^2,5)
fzero(@ (x) c1^2-x.^4*c3.^2,30)
```

Chapter 4

4.1. The first derivative is 0.2391, the second derivative is −2.8256. The function `diffgen` gives accurate answers using either $h = 0.1$ or 0.01. The function changes slowly over this range of values.

4.2. When $x = 1$, the computed and exact derivative is −5.0488; when $x = 2$, the computed derivative is −176.6375 (exact = −176.6450), and when $x = 3$, the computed derivative is −194.4680 (exact = −218.6079).

4.3. Using the new formula for Problem 4.1, the first derivative estimate is 0.2267 and 0.2390 for $h = 0.1$ and 0.01, respectively. The second derivative is −2.8249 and −2.8256 for $h = 0.1$ and 0.01, respectively. In Problem 4.2 for $x = 1$, 2, and 3 the first derivative estimates are −5.0489, −175.5798, and −150.1775, respectively. Note that these are less accurate than using `diffgen`.

4.4. The approximate derivatives are −1367.2, −979.4472, −1287.7, and −194.4680. If h is decreased to 0.0001, then the values are the same as the exact derivatives to the given number of decimal places.

4.5. The exact partial derivatives with respect to x and y are 593.652 and 445.2395, respectively. The corresponding approximate values are 593.7071 and 445.2933.

4.6. The integral method estimates 6.3470 primes in the range 1 to 10, 9.633 primes in the range 1 to 17, and 15.1851 primes in the range 1 to 30. The actual numbers are 7, 10, and 15.

4.7. The exact values are 1.5708, 0.5890, and 0.2443 for $r = 0$, 1, and 2, respectively. Approximations provided by integral are 1.5338, 0.5820, and 0.2700.

4.8. The exact values are -0.0811 for $a = 1$ and 0.3052 for $a = 2$. Using `simp1` with 512 points gives agreement to 12 decimal places.

4.9. The exact answer is -0.915965591, and the answer given by `fgauss` is -0.9136. Function `simp1` cannot be used because of the singularity at $x = 0$.

4.10. The exact answer is 0.915965591; `fgauss` gives 0.9159655938. Note that the integrals of Problems 4.9 and 4.10 have the same value apart from the sign.

4.11. **(a)** Using (4.32) with 10 points gives 3.97746326050642; 16-point Gauss gives 3.8145. **(b)** Using (4.33) gives 1.77549968921218; 16-point Gauss gives 1.7758.

4.13. The function `filon` gives

$$2.00000000000098,\ -0.13333333344440,\ \text{and}\ -2.000199980281494 \times 10^{-4}$$

4.14. Using Romberg's method with nine divisions gives $-2.000222004003794 \times 10^{-4}$. Using Simpson's rule with 1024 intervals gives $-1.999899106566088 \times 10^{-4}$.

4.18. The solution for (a) is 48.96321182552904 and for (b) is 9726.56492. These compare well with the exact solution, which can be computed from the formula $4\pi^{(n+1)}/(n+1)^2$ where n is the power of x and y.

4.19. **(a)** To fix limits, substitute $y = (\sqrt{(x/3)} - 1)z + 1$. Answer: -1.71962748468952. **(b)** To fix limits, substitute $y = (2 - x)z$. Answer: 0.22222388780205.

4.20. The answers are **(a)** 9725.75264 and **(b)** 0.22222222200993.

4.21. Values of the integral are given in the following table:

z	Exact	16-point Gauss
0.5	0.493107418	0.49310741784618
1.0	0.946083070	0.94608306999140
2.0	1.605412977	1.60541297617644

4.22. Use `gauss2v` and define the following function:

```
z = @(x,y) 1./(1-x.*y);
```

4.23. The folowing will provide the solution of this problem:

```
% Probability of engine failure
p = [ ];
a = 3.5; b = 8200;
i = 1;
```

```
for T = 200:100:4000
P(i) = quad(@(x) a*b^a./((x+b).^(a+1)),0.001,T);
i = i+1;
end
figure(1)
plot(200:100:4000,P)
xlabel('Time in hours'), ylabel('Probability of failure')
title('plot of probaility of failure against time')
grid
```

4.24. The folowing will provide the solution of this problem; the value of the integral is -0.15415 correct to 5 places.

```
p = 3; q = 4; r = 2;
f = @(x) (x.^p-x.^q).*x.^r ./log(x);
val = quad(f,0,1);
fprintf('\n value of integral = %6.5f\n',val)
check = log((p+r+1)/(q+r+1))
fprintf('\n value of integral = %6.5f\n',check)
```

4.25. The three integrals are approximately equal to 0.91597.

4.26. The integral equals zero to five decimal places.

4.27. A fairly low accuracy result is obtained.

4.28. Accuracy to two decimal places is obtained.

4.29. There is good agreement between values. The script is

```
f = @(x) -log(x).^3 .* exp(-x)
val = quadgk(f,0,Inf);
fprintf('\n value of integral = %6.5f\n',val)
gam = 0.57722;
S3 = gam^3+0.5*gam*pi^2+2*zeta(3)
fprintf('\n Approximate sum of series = %6.5f\n',S3)
```

4.30. The best result is given by `dblquad` and is 2.01131. The script is

```
R = dblquad(@(x,y) (1-cos(50*x).*cos(100*y))./(2-cos(x)-cos(y))...
,0.0001,pi,0.0001,pi);R=R/pi^2;
fprintf('\nValue of integral using dblquad = %6.5f\n',R)
R1 = simp2v(@(x,y) (1-cos(50*x).*cos(100*y))./(2-cos(x)-cos(y))...
    ,.00001,pi,0.00001,pi,64);R1=R1/pi^2;
```

```
gamma = -psi(1);
R = (gamma+3*log(2)/2+log(50^2+100^2)/2)/pi;
fprintf('\nValue of integral using simp2v = %6.5f\n',R1)
fprintf('\n Approximate value check = %6.5f\n',R)
```

4.31. Value of integral = 0.46306.

Chapter 5

5.1. When $t = 10$, the exact value is 30.326533. `feuler`: 29.9368, 30.2885, 30.3227 with $h = 1$, 0.1, and 0.01, respectively. `eulertp`: 30.3281, 30.3266 with $h = 1$ and 0.1, respectively. `rkgen`: 30.3265 with $h = 1$.

5.2. The classical method gives 108.9077, the Butcher method gives 109.1924, and the Merson method gives 109.0706. The exact answer is $2\exp(x^2) = 109.1963$.

5.3. The Adams–Bashforth–Moulton method gives 4.1042, and Hamming's method gives 4.1043. The exact answer is 4.1042499.

5.4. Using `ode23` gives 0.0254; using `ode45` gives 0.0584.

5.5. The solution of Problem 5.1 with $h = 1$ is 30.3265. The solution of Problem 5.2 with $h = 0.2$ is 108.8906. The solution of Problem 5.2 with $h = 0.02$ is 109.1963.

5.6. (a) 7998.6, exact = 8000. (b) 109.1963.

5.9. The method is stable for $h = 0.1$ and 0.2, and unstable for $h = 0.4$.

5.11. Define the right sides using the following function:

```
function = p511(t,x)
v = ones(2,1);
v(1) = x(1)*(1-0.001*x(1)-1.8*x(2));
v(2) = x(2)*(.3-.5*x(2)/x(1));
```

5.12. Define the right sides using the following function:

```
function v = p512(t,x)
v = ones(2,1);
v(1) = -20*x(1); v(2) = x(1);
```

5.13. Define the right sides using the following function:

```
function v = p513(t,x)
v = ones(2,1);
v(1) = -30*x(2);
v(2) = -.01*x(1)*x(2);
```

5.14. Define the right sides using the following function:

```
function v = p514(t,x)
global c
k = 4; m = 1; F = 1;
v = ones(2,1);
v(1) = (F-c*x(1)-k*x(2))/m;
v(2) = x(1);
```

The script to solve this equation is

```
global c
i = 0;
for c = [0,2,1]
    i = i+1;
    c
    [t,x] = ode45('q514',[0 10],[0 0]');
    figure(i)
    plot(t,x(:,2))
end
```

5.16. The function to solve this problem is

```
function prhs = planetrhs(t,x)
% global x0
% NB global is used if initial values x0
% are used to calculate impact probabilities
% rather than x the variable values
for i=1:3
    for j=1:3
        A(i,j)=x(i).*x(j)./(x(i)+x(j))/1000;
    end
end
prhs = zeros(3,1);
prhs(1) = -x(1).*(A(1,2).*x(2)+A(1,3).*x(3));
prhs(2) = 0.5*A(1,1)*x(1).*x(1)-x(2).*(A(2,2).*x(2)+A(2,3).*x(3));
prhs(3) = 0.5*A(1,2)*x(1).*x(2);
```

The script is as follows:

```
% Solution of planetary growth
% The coagulation equation three size model
% Let x(1), x(2) and x(3) represent the
% number of planetesimals of the three sizes
global x0
% Initially
```

```
x0 = [200,25,1];
tspan = [0,2];
[t,x] = ode45('planetrhs', tspan,x0);
fprintf('\n number of smallest planets= %3.0f',x(end,1))
fprintf('\n number of intermediate planets=%3.0f',x(end,2))
fprintf('\nlargest planets=%3.0f\n',x(end,3))
figure(1)
plot(t,x)
xlabel('time'), ylabel('planet numbers')
grid
```

5.17. The script to solve this equation is

```
% Solution of Daisy world problem
span = 10;
[x,t] = ode45('daisyf',span,[0.2, 0.3]);
plot(x,t)
xlabel('Time'), ylabel('black and white daisy areas')
title('daisy world')
grid
```

and the function is

```
function daisyrhs = daisyf(t,x)
daisyrhs = zeros(2,1);
gamma = 0.3;
Tb = 295; Tw = 285;
betab = 1-0.003265*(295.5-Tb)^2;
betaw = 1-0.003265*(295.5-Tw)^2;
barbit = 1-x(1)-x(2);
daisyrhs(1) = x(1).*(barbit.*betab-gamma);
daisyrhs(2) = x(2).*(barbit.*betaw-gamma);
```

Chapter 6

6.1. **(a)** hyperbolic; **(b)** parabolic; **(c)** $f(x,y) > 0$, hyperbolic; $f(x,y) < 0$, elliptic.

6.2. Initial slope $= -1.6714$. The shooting and FD methods give good results.

6.3. This is an example of a stiff equation. (a) The actual slope when $x = 0$ is 1.0158×10^{-24}. Because we cannot determine this slope accurately, the shooting method gives a very inaccurate solution. (b) In this case the shooting method provides a good result because the initial slope is -120. In both cases the FD method requires a large number of divisions to give an accurate result.

6.5. The finite difference method gives $\lambda_1 = 2.4623$. Exact $\lambda_1 = (\pi/L)^2 = 2.4674$.

6.6. At $t = 0.5$ the variation of z is almost linear between the boundaries at 0 and 10.

6.7. The exact and FD approximations are very similar.

6.8. The exact and FD approximations are similar with a maximum error of 0.0479.

6.9. With 9 divisions in x, λ=5.8870 ,14.0418, 19.6215, 27.8876, 29.8780. Even with 36 divisions in x, the eigenvalues have not converged.

6.10. [0.7703 1.0813 1.5548 1.583 1.1943 1.5548 1.583 1.194 1.0813 0.7703].

Chapter 7

7.1. Using the `aitken` function, $E(2^\circ) = 1.5703$, $E(13^\circ) = 1.5507$, and $E(27^\circ) = 1.4864$. These are accurate to the places given.

7.2. The root is 27.8235.

7.3. (a) $p(x) = 0.9814x^2 + 0.1529$, and $p(x) = -1.2083x^4 + 2.1897x^2 + 0.0137$. The fourth-degree polynomial gives a good fit.

7.4. Interpolation gives 0.9284 (linear), 0.9463 (spline), and 0.9380 (cubic polynomial). The MATLAB function `aitken` gives 0.9455. This is the exact value to four decimal places.

7.5. $p(x) = -0.3238x^5 + 3.2x^4 - 6.9905x^3 - 12.8x^2 + 31.1429x$. Note that the polynomial oscillates between data points. The spline does not exhibit this characteristic, suggesting that it better represents any underlying function from which the data might have been taken.

7.6. **(a)** $f(x) = 3.1276 + 1.9811e^x + e^{2x}$.
(b) $f(x) = 685.1 - 2072.2/(1+x) + 1443.8/\,(1+x)^2$.
(c) $f(x) = 47.3747x^3 - 128.3479x^2 + 103.4153x - 5.2803$. Plotting these functions shows that the best fit is given by (a). The polynomial fit is a reasonable one.

7.7. The plot should diplay an airfoil section.

7.8. The product of primes less than P is given by $0.3679 + 1.0182\log_e P$ approximately.

7.9. $a_0 = 1$, $a_1 = -0.5740$, $a_2 = 0.9456$, $a_3 = -0.6865$, $a_4 = 0.4115$, $a_5 = -0.0966$.

7.10. Exact: -78.3323. Interpolation gives -78.3340 (cubic) or -77.9876 (linear).

7.11. The minimum values of E are approximately -14.95 and -6.45 at points 40 and 170. The maximum values of E are 3.68 and 16.47 at points 110 and 252.

7.12. The data is sampled from $y = \sin(2\pi f_1 t) + 2\cos(2\pi f_2 t)$ where $f_1 = 1.25$ Hz and $f_2 = 3.4375$ Hz. At 1.25 Hz, DFT = $-15.9999\imath$ and at 3.4375 Hz, DFT = 32.0001. The

negative complex coefficient is related to the positive size of the coefficient of the sine function, and the positive real component is related to the cosine function. To relate the size of the DFT components to the frequency components in the data we divide the DFT by the number of samples (32) and multiply by 2.

7.13. Algebraically,

$$32\sin^5(30t) = 20\sin(30t) - 10\sin(90t) + 2\sin(150t)$$

and

$$32\sin^6(30t) = 10 - 15\cos(60t) + 6\cos(120t) - \cos(180t)$$

To verify these results from the DFT it is necessary to divide it by n and multiply by 2. The real components are the values of the cosine coefficients. The imaginary components in the DFT are the negative of the values of the sine coefficients. Note also that the coefficient at zero frequency is 20, not 10. This is a consequence of the definition of the DFT; see Section 7.4.

7.14. Components in the spectrum at 30 Hz and 112 Hz. The reason for the large component at 112 Hz is that the component in the data at 400 Hz is above the Nyquist frequency and is folded back to give a spurious component—that is, 400 Hz is 144 Hz above the Nyquist frequency of 256 Hz; 112 Hz is 144 Hz below it.

7.16. With 32 points the frequency increment is 16 Hz and the significant components are at 96 Hz and 112 Hz (the largest amplitude). With 512 points the frequency increment is reduced to 1 Hz and the significant components are at 106, 107, and 108 Hz with the largest amplitude at 107 Hz. With 1024 points the frequency increment is reduced to 0.5 Hz and the component with the largest amplitude is at 107.5 Hz. The original data had a frequency component of 107.5 Hz.

7.17. The estimated production cost in year 6 is \$31.80 using cubic extrapolation and \$20.88 using quadratic extrapolation. Using the revised data the estimated costs are \$24.30 and \$21.57, respectively. These widely varying results, some of which are barely credible, show the dangers of trying to estimate future costs from insufficient data.

7.18. $x = 2.4679$.

7.19. $I = 1.5713$, $\alpha = 8.71406$ degrees

7.20. $f_n = \frac{n}{6}\left(n^2 + 3n + 2\right)$.

7.23. The values are 22.70, 22.42, and 22.42.

7.24. The script for this problem is

```
load sunspot.dat
year = sunspot(:,1);
sunact = sunspot(:,2);
figure(1)
plot(year,sunact)
xlabel('Year'), ylabel('Sunspots')
title('Sunspot activity by year')
Y = fft(sunact);
N = length(Y);
Power = abs(Y(1:N/2)).^2;
freq = (1:N/2)/(N/2)*0.5;
figure(2)
plot(freq,Power)
xlabel('freq'), ylabel('Power')
```

Chapter 8

8.1. The objective is 21.6667. The solution is $x_1 = 3.6667$, $x_3 = 0.3333$; the other variables are zero.

8.2. The objective is 21.6667. The solution is $x_1 = 3.3333$, $x_2 = 1.6667$, $x_4 = 0.3333$; the other variables are zero. Thus this problem and the previous one have objective functions of equal magnitude.

8.3. The objective is 100. The solution is $x_1 = 10$, $x_3 = 20$, $x_5 = 22$; the other variables are zero.

8.4. This is a difficult function for the conjugate gradient method and this is why the accuracy of the line search for the built-in MATLAB function `fminsearch` was changed to produce more accurate results. The solution is [1.0007 1.0014] with gradient [0.33860.5226] $\times 10^{-3}$.

8.5. The exact and computed solutions are both $[-2.9035 \ -2.9035 \ 1 \ 1 \ 1]$.

8.6. The solution is $[-0.4600 \ 0.5400 \ 0.3200 \ 0.8200]^\top$. `norm(b-Ax)`$= 1.3131 \times 10^{-14}$.

8.7. `[xval,maxf] = optga('p807',[0 2],8,12,20,.005,.6)` where `p807` is a MATLAB function defining the problem. A test run gave the following answers:

`xval = 0.9098, maxf = 0.4980.`

8.8. The major modification is to the `fitness` function as follows:

```
function [fit,fitot] = fitness2d(criteria,chrom,a,b)
% calculate fitness of a set of chromosomes for a two variable
% function assuming each variable is defined in the range
% a to b using a two variable function given by criteria
[pop bitl] = size(chrom); vlength = floor(bitl/2);
for k = 1:pop
    v = [ ]; v1 = [ ]; v2 = [ ]; partchrom1 = chrom(k,1:vlength);
    partchrom2 = chrom(k,vlength+1:2*vlength);
    v1 = binvreal(partchrom1,a,b); v2 = binvreal(partchrom2,a,b);
    v = [v1 v2]; fit(k) = feval(criteria,v);
end
fitot = sum(fit);
```

A call of the modified algorithm is `optga2d('f808',[1 2],24,40,100,.005,.6)`, for example, where `f808` defines $z = x^2 + y^2$. This gives the sample results `maxf=7.9795` and `xval=[1.9956 1.9993]`.

8.11. The obtained values are 0.1605 and 0.4931.

8.12. The script to solve this question is

```
clf
[x,y] = meshgrid(-4:0.1:4,-4:0.1:4);
p = x.^2+y.^2;
z = (1-x).^2.*exp(-p)- p.*exp(-p) - exp(-(x+1).^2 - y.^2);
figure(1)
surf(x,y,z)
xlabel('x-axis'), ylabel('y-axis'), zlabel('z-axis')
title('mexhat plot')
figure(2)
contour(x,y,z,20)
xlabel('x-axis'), ylabel('y-axis')
title('contour plot')
optp = ginput(3);
x = optp(:,1); y = optp(:,2);
p = x.^2+y.^2;
z = (1-x).^2.*exp(-p)- p.*exp(-p) - exp(-(x+1).^2 - y.^2)
fprintf('maximum value= %6.2f\n',max(z))
fprintf('minimum value= %6.2f\n',min(z))
x
y
```

```
P=x(1).^2+x(2).^2;
fopt=@ (x)(1-x(1)).^2 .*exp(-(x(1).^2+x(2).^2))...
    - (x(1).^2+x(2).^2).*exp(-(x(1).^2+x(2).^2))...
    - exp(-(x(1)+1).^2 - x(2).^2) ;
[x,fval] = fminsearch(fopt,[-4;4])
fprintf('\nNon global solution= %8.6f\n',fval)
```

8.13. Note that continuous GA gives good agreement with Problem 8.12. Optimum $= -0.3877$

8.14. The minimum is achieved at 63.8157.

8.15. The minimum is achieved at 63.8160, a very similar result to Problem 8.14.

Chapter 9

9.1. Use

```
>>collect((x-1/a-1/b)*(x-1/b-1/c)*(x-1/c-1/a))
```

9.2. Use

```
>>y = x^4+4*x^3-17*x^2+27*x-19; z = x^2+12*x-13;
>>horner(collect(z*y))
```

9.3. Use

```
>>expand(tan(4*x))
>>expand(cos(x+y))
>>expand(cos(3*x))
>>expand(cos(6*x))
```

9.4. Use

```
expand(cos(x+y+z))
```

9.5. Use

```
>>taylor(asin(x),8)
>>taylor(acos(x),8)
>>taylor(atan(x),8)
```

9.6. Use

```
taylor(log(cos(x)),13)
```

9.7. Use

```
>>[solution, how] = simple(symsum((r+3)/(r*(r+1)*(r+2))*(1/3)^r,1,n))
```

9.8. Use

```
symsum(k^10,1,100)
```

9.9. Use
```
symsum(k^(-4),1,inf)
```

9.10. Use
```
a = [1 a a^2;1 b b^2;1 c c^2]; factor(inv(a))
```

9.11. Set
```
a = [a1 a2 a3 a4;1 0 0 0;0 1 0 0;0 0 1 0]
```
and use
```
ev = a-lam*eye(4)
```
and
```
det(ev)
```

9.12. Set
```
trans = [cos(a1) sin(a1);-sin(a1) cos(a1)];
```
and use
```
>>[solution,how] = simple(trans^2)
>>[solution,how] = simple(trans^4)
```

9.13. Set
```
r = solve('x^3+3*h*x+g=0')
```
and use
```
[solution,s] = subexpr(r,'s')
```

9.14. Use
```
>>solve('x^3-9*x+28 = 0')
```

9.15. Use
```
>>p = solve('z^6 = 4*sqrt(2)+i*4*sqrt(2)');
>>res = double(p)
```

9.16. Use
```
>>f5 = log((1-x)*(1+x^3)/(1+x^2)); p = diff(f5);
>>factor(p)
```
Then use `pretty(ans)` to help interpret this result.

9.17. Use
```
>>f = log(x^2+y^2);
>>d2x = diff(f,x,2)
>>d2y = diff(f,y,2)
>>factor(d2x+d2y)
>>f1 = exp(-2*y)*cos(2*x);
>>r = diff(f1,'x',2)+diff(f1,'y',2)
```

9.18. Use
```
>>z = x^3*sin(y);
>>dyx = diff(diff(z,'y'),'x')
```

```
>>dxy = diff(diff(z,'x'),'y')
>>dxy = diff(diff(z,'x',4),'y',6)
>>dxy = diff(diff(z,'y',6),'x',4)
```

9.19. **(a)** Use

```
>>p = int(1/((a+f*x)*(c+g*x)));
>>[solution,how] = simple(p)
>>[solution,how] = simple(diff(solution))
```

(b) Use

```
>>solution = int((1-x^2)/(1+x^2))
>>p = diff(solution); factor(p)
```

9.20. Use

```
>>int(1/(1+cos(x)+sin(x)))
```

and

```
>>int(1/(a^4+x^4))
```

9.21. Use

```
>>int(x^3/(exp(x)-1),0,inf)
```

9.22. Use

```
>>int(1/(1+x^6),0,inf)
```

and

```
>>int(1/(1+x^10),0,inf)
```

9.23. Use

```
>>taylor(exp(-x*x),7)
>>p = int(ans,0,1); vpa(p,10)
>>taylor(exp(-x*x),15)
>>p = int(ans,0,1); vpa(p,10)
```

9.24. Use

```
>>int(sin(x^2)/x,0,inf)
```

9.25. Use

```
>>taylor(log(1+cos(x)),5)
>>int(ans,0,1)
```

9.26. Use

```
>>dint = 1/(1-x*y)
>>int(int(dint,x,0,1),y,0,1)
```

9.27. Use

```
>>[solution,s] = subexpr(dsolve('D2y+(b*p+a*q)*Dy+a*b*(p*q-1)*...
y = c*A ', 'y(0)=0', 'Dy(0)=0','t'),'s')
```

Using the `subs` function

```
>>subs(solution,{p,q,a,b,c,A},{1,2,2,1,1,20})
```

we obtain the solution for the given values as

```
ans =
10-5*s(2)/s(1)^(1/2)*exp(-1/2*s(3)*t)+5*s(3)/s(1)^(1/2)*exp(-1/2*s(2)*t)
```

In addition, since we also require the values of `s(1)`, `s(2)`, and `s(3)`, we again use `subs` as follows:

```
>>s = subs(s,{p,q,a,b,c},{1,2,2,1,1})

s =
[         17]
[ 5+17^(1/2)]
[ 5-17^(1/2)]
```

9.28. Use

```
>>sol = dsolve('2*Dx+4*Dy = cos(t),4*Dx-3*Dy = sin(t)','t')
```

This gives the solution in the form

```
sol =
    x: [1x1 sym]
    y: [1x1 sym]
```

To see the specific elements of the solution, use

```
>>sol.x

ans =
C1+3/22*sin(t)-2/11*cos(t)
```

and

```
>>sol.y

ans =
C2+2/11*sin(t)+1/11*cos(t)
```

9.29. Use

```
>>dsolve('(1-x^2)*D2y-2*x*Dy+2*y = 0','x')
```

9.30. **(a)** Use

```
>>laplace(cos(2*t))
```

and then

```
>>p = solve('s^2*Y+2*s+2*Y = s/(s^2+4)','Y');
>>ilaplace(p)
```

(b) Use

```
>>laplace(t)
```

and then

```
>>p = solve('s*Y-2*Y = 1/s^2','Y');
>>ilaplace(p)
```

(c) Use

```
>>laplace(exp(-2*t))
```

and then

```
>>p = solve('s^2*Y+3*s-3*(s*Y+3)+Y = 1/(s+2)','Y');]
>>ilaplace(p)
```

(d) The Laplace transform of zero is zero. Thus take the Laplace transform of the equation and then use

```
 >>p = solve('(s*Y-V)+Y/c=0','Y');
>>ilaplace(p)
```

9.31. (a) The Z-transform of zero is zero. Thus take the Z-transform of the equation and then use

```
>>p = solve('Y=-2*(Y/z+4)','Y');
>>iztrans(p)
```

(b) Use

```
>>ztrans(n)
```

and then use

```
>>p = solve('Y+(Y/z+10) = z/(z-1)^2','Y');
>>iztrans(p)
```

(c) Use

```
>>ztrans(3*heaviside(n))
```

and then use

```
>>p = solve('Y-2*(Y/z+1)=3*z/(z-1)','Y');
>>iztrans(p)
```

(d) Use

```
>>ztrans(3*4^n)
```

and then use

```
>>p = solve('Y-3*(Y/z-3)+2*(Y/z^2+5-3/z) = 3*z/(z-4)','Y');
>>iztrans(p)
```