Assignment 6

1. Find approximations for the two roots of the polynomial $0.0002358x^2 - 5535.0x + 0.00003513$ using the quadratic formula you learned in secondary school. Then, find the same roots, but choosing the appropriate formula for each.

```
a =      0.0002358;
b = -5535.0;
c =      0.00003513;
(-b + sqrt(b^2 - 4*a*c))/(2*a) % best for the larger root in absolute value
        23473282.44274808
(-b - sqrt(b^2 - 4*a*c))/(2*a)
        5.785589705934657e-09
(-2*c)/(b - sqrt(b^2 - 4*a*c)) % best for the smaller root in absolute value
        6.346883468834690e-09
```

2. Find approximations for the two roots of the polynomial $0.0002358x^2 + 5535.0x - 0.00003513$ using the quadratic formula you learned in secondary school. Then, find the same roots, but choosing the appropriate formula for each.

```
a =     0.0002358;
b = 5535.0;
c =    -0.00003513;
(-b + sqrt(b^2 - 4*a*c))/(2*a)
        5.785589705934657e-09
(-b - sqrt(b^2 - 4*a*c))/(2*a) % best for the larger root in absolute value
       -23473282.44274810
(-2*c)/(b + sqrt(b^2 - 4*a*c)) % best for the smaller root in absolute value
        6.346883468834686e-09
```

3. Given the function $f(x) = x^3 - x^2 - x - 1$, approximate the real root using two steps of each of:

a. Newton's method starting with $x_0 = 2.0$,
b. the bisection method starting with [1, 2],
c. the bracketed secant method starting with [1, 2] (optional), and
d. the secant method starting with $x_0 = 2.0$ and $x_1 = 1.9$.

```
% Newton's method
f = @(x)(x^3 - x^2 - x - 1.0);
df = @(x)(3*x^2 - 2*x - 1.0);
x0 = 2.0;
x1 = x0 - f(x0)/df(x0)
        x1 = 1.857142857142857
x2 = x1 - f(x1)/df(x1)
        x2 = 1.839544513457557

% Bisection method
a = 1;
b = 2;
m = (a + b)/2.0;
if sign(f(a)) == sign(f(m)); a = m else b = m end
         a = 1.500000000000000
m = (a + b)/2.0;
if sign(f(a)) == sign(f(m)); a = m else b = m end
         a = 1.750000000000000

% Bracketed secant method (optional)
a = 1;
b = 2;
m = (a*f(b) - b*f(a))/(f(b) - f(a));
if sign(f(a)) == sign(f(m)); a = m else b = m end
         a = 1.666666666666667
m = (a*f(b) - b*f(a))/(f(b) - f(a));
if sign(f(a)) == sign(f(m)); a = m else b = m end
         a = 1.816326530612245

% Secant method
x0 = 2.0;
x1 = 1.9;
x2 = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0))
        x2 = 1.846390168970814
x3 = (x1*f(x2) - x2*f(x1))/(f(x2) - f(x1))
        x3 = 1.839628859068081
```

4. Given the same function as in Question 3, approximate the first positive root using one step of each of:

a. Muller's method starting with $x_0 = 2.0$, $x_1 = 1.9$ and $x_2 = 1.8$ (optional), and
b. inverse quadratic interpolation with the same three points.

```
% Muller's method (optional)
x0 = 2.0;
x1 = 1.9;
x2 = 1.8;
% Find the polynomial passing through
%           (x0 - x2, f(x0)), (x1 - x2, f(x1)), (x2 - x2, f(x2))
p = polyfit( [x0 x1 x2] - x2, [f(x0) f(x1) f(x2)], 2 )
         p = 4.700000000000054   5.099999999999991  -0.208000000000000
delta = (-2*p(3))/(p(2) + sqrt(p(2)^2 - 4*p(1)*p(3)))
     delta = 3.935683999680209e-02
x3 = x2 + delta
        x3 = 1.839356839996802

% Inverse quadratic interpolation
x0 = 2.0;
x1 = 1.9;
x2 = 1.8;
% Find the polynomial passing through (f(x0), x0), (f(x1), x1), (f(x2), x2)
p = polyfit( [f(x0) f(x1) f(x2)], [x0 x1 x2], 2 )
         p = -0.02145975382064491   0.1825590389332351   1.838900714887409
% Get the constant coefficient
x3 = p(3)
        x3 = 1.838900714887409
```

5. Given the function $f(x) = x^2\cos(0.4x)e^{-0.3x}$, approximate the first positive root using two steps of each of:

a. Newton's method starting with $x_0 = 4.0$,
b. the bisection method starting with [3, 4],
c. the bracketed secant method starting with [3, 4] (optional), and
d. the secant method starting with $x_0 = 3.8$ and $x_1 = 3.9$.

```
% Newton's method
f = @(x)(x^2*cos(0.4*x)*exp(-0.3*x));
df = @(x)(2.0*x  *cos(0.4*x)*exp(-0.3*x)
        - 0.4*x^2*sin(0.4*x)*exp(-0.3*x)
        - 0.3*x^2*cos(0.4*x)*exp(-0.3*x));
x0 = 4.0;
x1 = x0 - f(x0)/df(x0)
       x1 = 3.928021373533735
x2 = x1 - f(x1)/df(x1)
       x2 = 3.926991039021064

% Bisection method
a = 3.0;
b = 4.0;
m = (a + b)/2.0;
if sign(f(a)) == sign(f(m)); a = m else b = m end
        a = 3.500000000000000
m = (a + b)/2.0;
if sign(f(a)) == sign(f(m)); a = m else b = m end
        a = 3.750000000000000

% Bracketed secant method (optional)
a = 3.0;
b = 4.0;
m = (a*f(b) - b*f(a))/(f(b) - f(a));
if sign(f(a)) == sign(f(m)); a = m else b = m end
        a = 3.904055035798684
m = (a*f(b) - b*f(a))/(f(b) - f(a));
if sign(f(a)) == sign(f(m)); a = m else b = m end
        a = 3.926649703297110

% Secant method
x0 = 3.8;
x1 = 3.9;
x2 = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0))
       x2 = 3.927770674823227
x3 = (x1*f(x2) - x2*f(x1))/(f(x2) - f(x1))
       x3 = 3.926986348481126
```

6. Given the same function as in Question 5, approximate the first positive root using one step of each of:

a. Muller's method starting with $x_0 = 3.8$, $x_1 = 4.0$ and $x_2 = 3.9$ (optional), and
b. inverse quadratic interpolation with the same three points.

```
% Muller's method (optional)
x0 = 3.8;
x1 = 4.0;
x2 = 3.9;
p = polyfit( [x0 x1 x2] - x2, [f(x0), f(x1), f(x2)], 2 )
         p = -0.4079818564029011  -1.876008417378137   0.05096502657785155
delta = (-2*p(3))/(p(2) - sqrt(p(2)^2 - 4*p(1)*p(3)))
     delta = 2.700810336615210e-02
x3 = x2 + delta
        x3 = 3.927008103366152

x0 = 3.8;
x1 = 4.0;
x2 = 3.9;
p = polyfit( [f(x0) f(x1) f(x2)], [x0 x1 x2], 2 )
         p = -0.06182184308358357  -0.5272495888523211   3.927031867462113
x3 = p(3)
        x3 = 3.927031867462113
```

7. Apply two steps of the Jacobi method or the Gauss-Seidel method (optional) and then two steps of successive over-relaxation applied to these with $\omega = 0.97$ for the Jacobi method and $\omega = 1.03$ for the Gauss-Seidel method to find an approximation of the solution to:

$$\begin{pmatrix} 10 & 2 \\ 2 & 10 \end{pmatrix} \mathbf{u} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}.$$

Answer: You don't have to understand how to code the Matlab code, but you must understand what successive over-relaxation does, and how it may or may not be useful. Observe that while successive over-relaxation does not significantly improve the Jacobi method, it is significantly more beneficial to the Gauss-Seidel method, which is a small modification of the Jacobi method.

```
A = [10 2; 2 10];
b = [3 1]';
x = A \ b         % The correct answer
    x =
       0.2916666666666667
       0.04166666666666667

D = diag( diag(A) );
invD = inv( D );
Aoff = A - D;
x0 = invD*b
    x0 =
       0.3
       0.1
```

```
%%%%%%%%%%
% Jacobi %
%%%%%%%%%%
x1 = invD*(b - Aoff*x0)
    x1 =
       0.28
       0.04

x2 = invD*(b - Aoff*x1)
    x2 =
       0.292
       0.044

norm( x2 - x )
    ans = 2.357022603955160e-03

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Jacobi with over-relaxation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
omega = 0.97;
x1 = invD*(b - Aoff*x0);
x1 = omega*x1 + (1 - omega)*x0
   x1 =
      0.2806
      0.0418

x2 = invD*(b - Aoff*x1);
x2 = omega*x2 + (1 - omega)*x1
    x2 =
       0.2913088
       0.0438176

norm( x2 - x )
    ans = 2.180500574536862e-03
```

```
%%%%%%%%%%%%%%%%
% Gauss-Seidel %   Optional...
%%%%%%%%%%%%%%%%
x1 = x0;
for k = 1:2
    x1(k) = (b(k) - Aoff(k,:)*x1)/D(k, k);
end
x2 = x1;
for k = 1:2
    x2(k) = (b(k) - Aoff(k,:)*x2)/D(k,k);
end

norm( x2 - x )
    ans = 4.759084879353294e-04

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gauss-Seidel with successive over-relaxation %   Optional...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
omega = 1.03;
x1 = x0;
for k = 1:2
    x1(k) = (b(k) - Aoff(k,:)*x1)/D(k,k);
end
x1 = omega*x1 + (1 - omega)*x0
    x1 =
       0.2794
       0.04232

x2 = x1;
for k = 1:2
    x2(k) = (b(k) - Aoff(k,:)*x2)/D(k,k);
end
x2 = omega*x2 + (1 - omega)*x1
    x2 =
       0.2919
       0.041673984

norm( x2 - x )
    ans = 2.335280016291043e-04
```

8. Apply two steps of the Jacobi method or the Gauss-Seidel method (optional) and then two steps of successive over-relaxation applied to these with $\omega = 0.99$ for the Jacobi method and $\omega = 1.08$ for the Gauss-Seidel method to find an approximation of the solution to:

$$\begin{pmatrix} 5 & 2 & 1 \\ 2 & 10 & -3 \\ 1 & -3 & 20 \end{pmatrix} \mathbf{u} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}.$$

Answer: You don't have to understand how to code the Matlab code, but you must understand what successive over-relaxation does, and how it may or may not be useful. Observe that while successive over-relaxation does not significantly improve the Jacobi method, it is significantly more beneficial to the Gauss-Seidel method, which is a small modification of the Jacobi method.

```
A = [5 2 1; 2 10 -3; 1 -3 20];
b = [2 1 1]';
x = A \ b          % The correct answer
    x =
       0.3786635404454864
       0.03516998827667058
       0.03634232121922626

D = diag( diag(A) );
invD = inv( D );
Aoff = A - D;
x0 = invD*b
    x0 =
       0.4
       0.1
       0.05
```

```
%%%%%%%%%%
% Jacobi %
%%%%%%%%%%
x1 = invD*(b - Aoff*x0)
    x1 =
       0.35
       0.035
       0.045

x2 = invD*(b - Aoff*x1)
    x2 =
       0.377
       0.0435
       0.03775

norm( x2 - x )
    ans = 8.610343876664588e-03

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Jacobi with over-relaxation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
omega = 0.99;
x1 = invD*(b - Aoff*x0);
x1 = omega*x1 + (1 - omega)*x0
    x1 =
       0.3505
       0.03565
       0.04505

x2 = invD*(b - Aoff*x1);
x2 = omega*x2 + (1 - omega)*x1
    x2 =
       0.3764677
       0.04333735
       0.037894775

>> norm( x2 - x )
    ans = 8.598699059926416e-03
```

```
%%%%%%%%%%%%%%%%
% Gauss-Seidel %   Optional...
%%%%%%%%%%%%%%%%
x1 = x0;
for k = 1:3
    x1(k) = (b(k) - Aoff(k,:)*x1)/D(k, k);
end
x2 = x1;
for k = 1:3
    x2(k) = (b(k) - Aoff(k,:)*x2)/D(k,k);
end
norm( x2 - x )
    ans = 4.874905937179975e-03

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gauss-Seidel with successive over-relaxation %   Optional...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
omega = 1.08;
x1 = x0;
for k = 1:3
    x1(k) = (b(k) - Aoff(k,:)*x1)/D(k,k);
end
x1 = omega*x1 + (1 - omega)*x0
    x1 =
       0.346
       0.0406
       0.03839

x2 = x1;
for k = 1:3
    x2(k) = (b(k) - Aoff(k,:)*x2)/D(k,k);
end
x2 = omega*x2 + (1 - omega)*x1
    x2 =
       0.37848856
       0.035956648
       0.0365010692

norm( x2 - x )
    ans = 8.213723869300104e-04
```

9. The following system is given with the solution:

$$\begin{pmatrix} 2 & 5 \\ 5 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 6 \end{pmatrix}.$$

If you were to try to apply the Gauss-Seidel method to find the solution, does it seem to converge? Why does this happen?

```
A = [2 5; 5 1];
b = [7 6]';
D = diag( diag(A) );
Aoff = A - D;
x0 = inv(D)*b
     x0 = 3.5
          6.0

x1 = x0;
for k = 1:2
    x1(k) = (b(k) - Aoff(k,:)*x1)/D(k,k);
end
x1
     x1 = -11.5
           63.5

x2 = x1;
for k = 1:2
    x2(k) = (b(k) - Aoff(k,:)*x2)/D(k,k);
end
x2
     x2 = -155.25
           782.25
```

Note that the off-diagonal entries are significantly larger in absolute value, so each time we are calculating $A_{\mathrm{off}}\mathbf{x}_k$, this is magnifying the result, and thus dividing by the diagonal entries does not reduce this value.

10. Could you use the method of successive over-relaxation with a method such as Newton's method? For example, if you found that $x_1 > x_2 > x_3 > x_4$, might it not make sense to try to use $\omega = 1.05$? Similarly, if successive approximations move back and forth, might it not make sense to try to use $\omega = 0.95$?

Yes, but you'd have to be careful. Also, finding the correct value of ω may be difficult if you're only using Newton's method once; however, yes, it would work, for Newton's method, too. Indeed, it would potentially work for any iterative method if a reasonable value of ω is determined.

Note, if Newton's method appears to be converging linearly (that is, O(h)), this may suggest that it is converging to a root with multiplicity greater than one, so you may try using $\omega = 2$ or even higher.