《机械工程中的数值分析技术》

作业

学　　生：易弘睿

学　　号：20186103

专业班级：机械一班

作业编号：2021053003

重庆大学-辛辛那提大学联合学院

二〇二一年五月

Catalog

Lec.5 Nonlinear Eq.

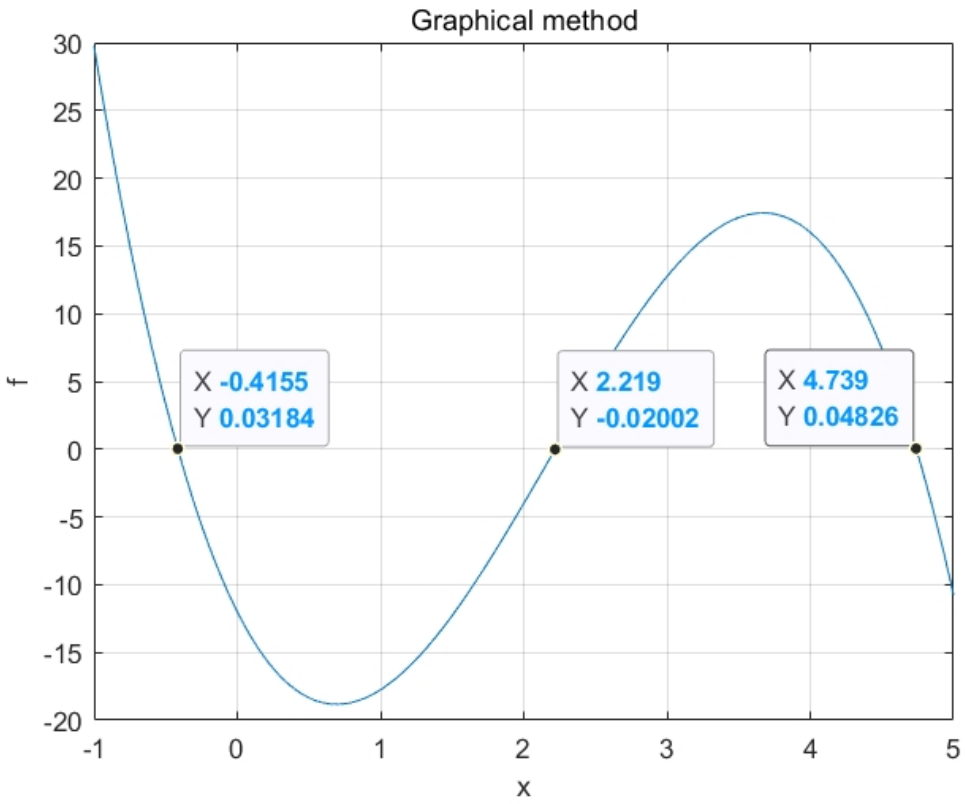
1.1 Question 5.5

5.5 **(a)** Determine the roots of $f(x) = -12 - 21x + 18x^2 - 2.75x^3$ graphically. In addition, determine the first root of the function with **(b)** bisection and **(c)** false position. For **(b)** and **(c)** use initial guesses of $x_l = -1$ and $x_u = 0$ and a stopping criterion of 1%.

a) For the graphical method, the Matlab code is below:

```
clear;close all;clc
x = linspace(-1,5,10000);
f = -12 - 21*x + 18*x.^2 - 2.75*x.^3;
plot(x,f)
title('Graphical method')
xlabel('x')
ylabel('f')
grid on
```

The output is below:



From the graph above, we can see the roots are x=-0.4155, 2.219, 4,739.

b) For the bisection method, the Matlab code is below:

```
clear;close all;clc
f = @(x) -12 - 21*x + 18*x.^2 - 2.75*x.^3;
xl = -1;
xu = 0;
xr = xl;
E = 0.01;
while 1
    xrold = xr;
    xr = (xl + xu)/2;
    if xr ~= 0
       e  = abs((xr - xrold)/xr);
    end
    result = f(xr) * f(xl);
    if result < 0
        xu = xr;
    elseif result > 0
        xl = xr;
    elseif result == 0
```

```
        e = 0;
    end
    if e <= E
        break
    end
end
fprintf('The root is %.5f.',xr)
```

The output is below:
The root is -0.41797.

c) For the false method, the Matlab code is below:

```
clear;close all;clc
xu = 0;
xl = -1;
xr = xl;
E = 0.01;
f = @(x) -12 - 21*x + 18*x.^2 - 2.75*x.^3;
while 1
    xrold = xr;
    f_xl = f(xl);
    f_xu = f(xu);
    xr = xu - (f_xu*(xl - xu))/(f_xl - f_xu);
    if xr ~= 0
        e = abs((xr - xrold)/xr);
    end
    test = f_xu*f(xr);
    if test == 0
        e = 0;
    elseif test > 0
        xu = xr;
    elseif test < 0
        xl = xr;
    end
    if e <= E
        break
    end
end
fprintf('The root is %.5f.',xr)
```

The output is below:
The root is -0.41402.

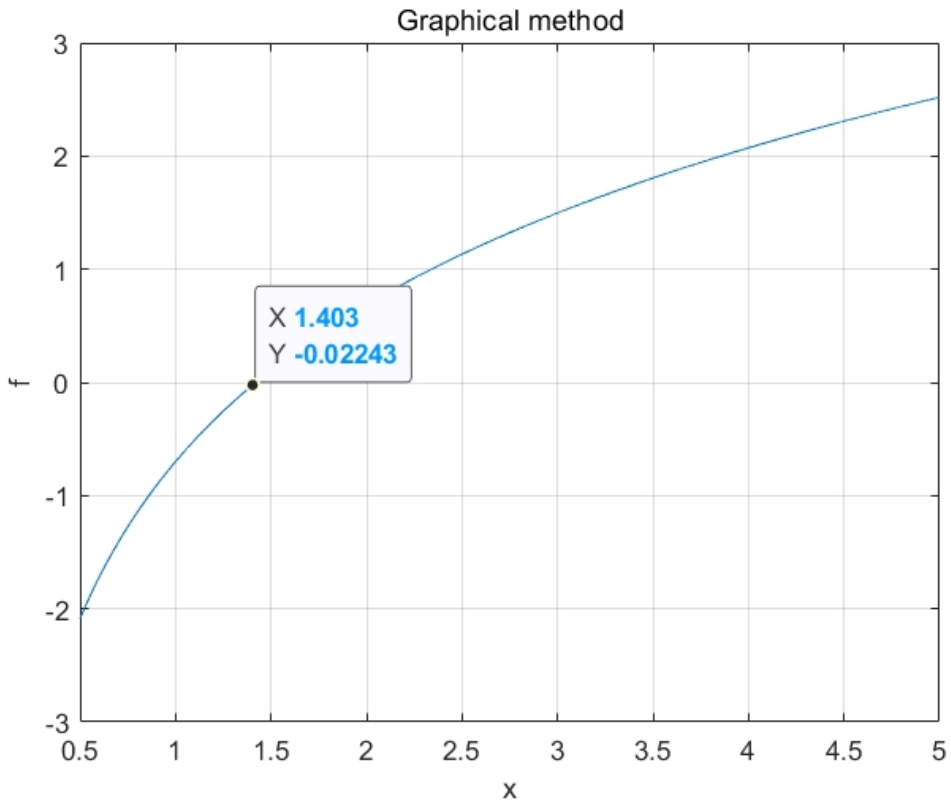
1.2 Question 5.7

5.7 Determine the positive real root of $\ln(x^2) = 0.7$ **(a)** graphically, **(b)** using three iterations of the bisection method, with initial guesses of $x_l = 0.5$ and $x_u = 2$, and **(c)** using three iterations of the false-position method, with the same initial guesses as in **(b)**.

a) For the graphical method, the Matlab code is below:

```
clear;close all;clc
x = linspace(0.5,5,10000);
f = log(x.^2) - 0.7;
plot(x,f)
title('Graphical method')
xlabel('x')
ylabel('f')
grid on
```

The output is below:



b) For the bisection method, using three iterations, the Matlab code

is below:

```
clear;close all;clc
f = @(x) log(x.^2) - 0.7;
xl = 0.5;
xu = 2;
xr = xl;
iter = 0;
while 1
    xr = (xl + xu)/2;
    iter = iter + 1;
    result = f(xr) * f(xl);
    if result < 0
        xu = xr;
    elseif result > 0
        xl = xr;
    elseif result == 0
        break
    end
    if iter == 3
        break
    end
end
fprintf('The root is %.5f.',xr)
```

The output is below:
The root is 1.43750.

c) For the false method, using three iterations, the Matlab code is below:

```
clear;clc;close all
xu = 2;
xl = 0.5;
xr = xl;
f = @(x) log(x.^2) - 0.7;
iter = 0;
while 1
    f_xl = f(xl);
    f_xu = f(xu);
    xr = xu - (f_xu*(xl - xu))/(f_xl - f_xu);
    iter = iter + 1;
    test = f_xu*f(xr);
```

```
    if test == 0
        break
    elseif test > 0
        xu = xr;
    elseif test < 0
        xl = xr;
    end
    if iter == 3
        break
    end
end
fprintf('The root is %.5f.',xr)
```

The output is below:
The root is 1.44840.

1.3 Question 5.8

5.8 The saturation concentration of dissolved oxygen in freshwater can be calculated with the equation

$$\ln o_{sf} = -139.34411 + \frac{1.575701 \times 10^5}{T_a} - \frac{6.642308 \times 10^7}{T_a^2} + \frac{1.243800 \times 10^{10}}{T_a^3} - \frac{8.621949 \times 10^{11}}{T_a^4}$$

where o_{sf} = the saturation concentration of dissolved oxygen in freshwater at 1 atm (mg L^{-1}); and T_a = absolute temperature (K). Remember that $T_a = T + 273.15$, where T = temperature (°C). According to this equation, saturation decreases with increasing temperature. For typical natural waters in temperate climates, the equation can be used to determine that oxygen concentration ranges from 14.621 mg/L at 0 °C to 6.949 mg/L at 35 °C. Given a value of oxygen concentration, this formula and the bisection method can be used to solve for temperature in °C.

(a) If the initial guesses are set as 0 and 35 °C, how many bisection iterations would be required to determine temperature to an absolute error of 0.05 °C?

(b) Based on **(a)**, develop and test a bisection M-file function to determine T as a function of a given oxygen concentration. Test your function for o_{sf} = 8, 10 and 14 mg/L. Check your results.

a) $n = \log_2(\frac{\Delta x_0}{E_{a,d}}) = \log_2(\frac{35}{0.05}) = 9.451$

It can be rounded up to 10 iterations.

b) The Matlab code is below:

```
function root = HW5_8b(osf)
tl = 0 + 273.15;
tu = 35 + 273.15;
tr = tl;
iter = 0;
```

```
f = @(t) log(osf) + 139.34411 - (1.575701*10^5)/t +
(6.642308*10^7)/(t^2) - (1.2438*10^10)/(t^3) +
(8.621949*10^11)/(t^4);
while 1
    tr = (tl + tu)/2;
    iter = iter + 1;
    result = f(tr)*f(tl);
    if result == 0
        break
    elseif result > 0
        tl = tr;
    elseif result < 0
        tu = tr;
    end
    if iter == 10
        break
    end
end
root = tr - 273.15;
fprintf('The temperature is %.4f degrees',root)
end
```

The output is below:

For osf = 8:

The temperature is 26.7627 degrees

root =

26.7627

For osf = 10:

The temperature is 15.4150 degrees

root =

15.4150

For osf = 14:

The temperature is 1.5381 degrees

root =

1.5381

1.4 Question 5.15

5.15 Many fields of engineering require accurate population estimates. For example, transportation engineers might find it necessary to determine separately the population growth trends of a city and adjacent suburb. The population of the urban area is declining with time according to

$$P_u(t) = P_{u,\max} e^{-k_u t} + P_{u,\min}$$

while the suburban population is growing, as in

$$P_s(t) = \frac{P_{s,\max}}{1 + [P_{s,\max}/P_0 - 1]e^{-k_s t}}$$

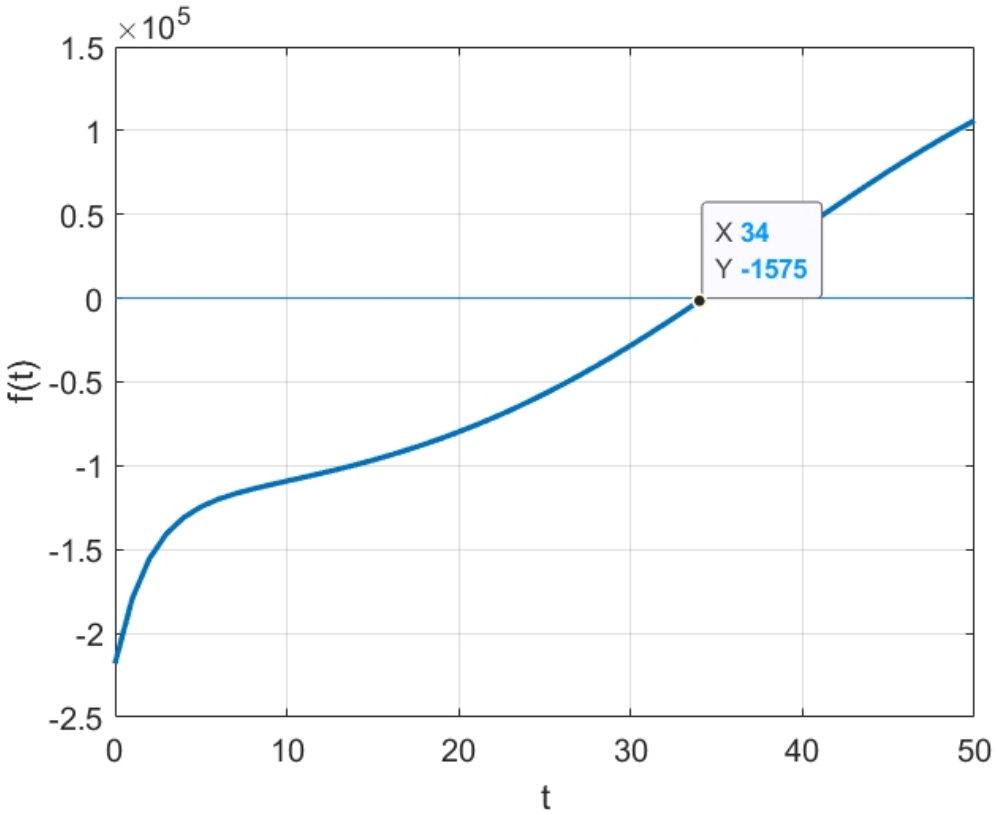
where $P_{u,\max}$, k_u, $P_{s,\max}$, P_0, and k_s = empirically derived parameters. Determine the time and corresponding values of $P_u(t)$ and $P_s(t)$ when the suburbs are 20% larger than the city. The parameter values are $P_{u,\max} = 80{,}000$, $k_u = 0.05$/yr, $P_{u,\min} = 110{,}000$ people, $P_{s,\max} = 320{,}000$ people, $P_0 = 10{,}000$ people, and $k_s = 0.09$/yr. To obtain your solutions, use **(a)** graphical, and **(b)** false-position methods.

a) The Matlab code is below:

```
clear;close all;clc;
HW_5_15a();
function HW_5_15a()
    Pu = @(t) 80000*exp(-0.5*t)+110000;
    Ps = @(t) 320000./(1+31*exp(-0.09*t));
    Time = 0:1:50;
    f = zeros(1,length(Time));
    for i=1:1:length(Time)
        t=Time(i);
        f(i)=Ps(t)-1.2*Pu(t);
    end
    figure()
    plot(Time,f,'linewidth',2);
    hold on
    line([Time(1) Time(end)],[0,0])
    grid on;
```

```
    xlabel('t');
    ylabel('f(t)');
    set(gca,'FontSize',12);
end
```

The output is below:



The time is 34, the corresponding Pu is 1.1000e+05 and Ps is 1.3042e+05.

b) The Matlab code is below:

```
clear;close all;clc;
HW_5_15b();
function HW_5_15b()
    Pu = @(t) 80000*exp(-0.5*t)+110000;
    Ps = @(t) 320000./(1+31*exp(-0.09*t));
    f = @(t) Ps(t) - 1.2*Pu(t);
    a=30;
    b=40;
    tol=1e-5;
    i=1;
    root = zeros(1,1);
    root(1) = 0;
    fprintf('i a b x');
    fprintf('f(a) f(b) f(c)\n')
```

```
    while(1)
        i=i+1;
        c=a-((f(a)*(b-a))/(f(b)-f(a)));
        if(f(c)*f(a)>0)
            b=c;
            g=f(b);
            root(i)=b;
        else
            a=c;
            g=f(a);
            root(i)=a;
        end
        if abs(root(i)-root(i-1))<=tol
            break
        end
        if i == 10,break,end
            fprintf('%3.0f%10.5f%10.5f%10.5f%15.5f%15.5f%15.5f\n',...
            i,a,b,c,f(a),f(b),f(c));
    end
    fprintf('\nThe Root of the equation = %10.5f\n',root(end));
end
```

The output is below:

```
i a b xf(a) f(b) f(c)
  2   30.00000   34.06194   34.06194   -28217.50988     -1144.13811     -
1144.13811
  3   34.23360   34.06194   34.23360        52.46700     -1144.13811
52.46700
  4   34.22607   34.06194   34.22607        -0.06873     -1144.13811
-0.06873

The Root of the equation =   34.22608
```

The corresponding Pu is 1.1000e+05 and Ps is 1.3200e+05

Lec.6 Open Method of Nonlinear Eq.

2.1 Question 6.3

6.3 Determine the highest real root of $f(x) = x^3 - 6x^2 + 11x - 6.1$:

(a) Graphically.

(b) Using the Newton-Raphson method (three iterations, $x_i = 3.5$).

(c) Using the secant method (three iterations, $x_{i-1} = 2.5$ and $x_i = 3.5$).

(d) Using the modified secant method (three iterations, $x_i = 3.5$, $\delta = 0.01$).

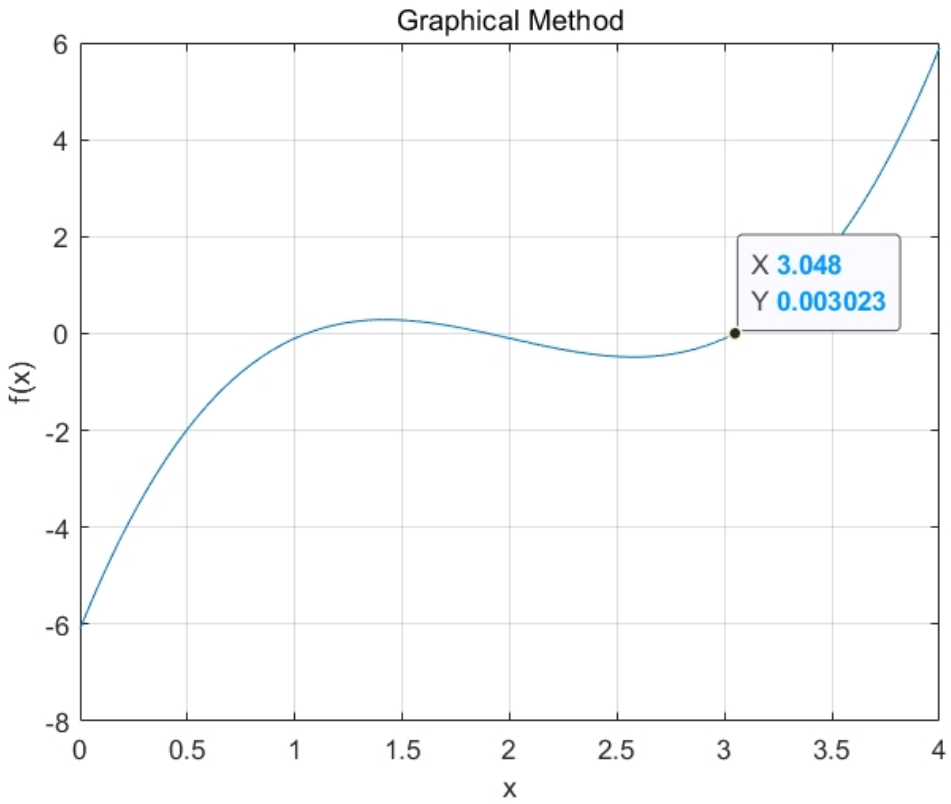
(e) Determine all the roots with MATLAB.

a) The Matlab code for the graphical method is below:

```
clear;close all;clc;
x = 0:0.001:4;
f = @(x) x.^3 - 6*x.^2 + 11*x - 6.1;
plot(x,f(x))
title('Graphical Method')
xlabel('x')
ylabel('f(x)')

grid on
```

The output is below:



The highest real root is 3.048.

b) The Matlab code for the Newton-Raphson method is below:

```
clear;close all;clc;
x = 0:0.001:4;
f = @(x) x.^3 - 6*x.^2 + 11*x - 6.1;
plot(x,f(x))
title('Graphical Method')
xlabel('x')
ylabel('f(x)')
grid on
% (b)
clear;clc;close all
f = @(x) x.^3 - 6*x.^2 + 11*x - 6.1;
n = 0;
x0 = 3.5;
fd = @(x) 3*x.^2 - 12*x + 11;
while 1
    x1 = x0 - f(x0)/fd(x0);
    n = n + 1;
    x0 = x1;
    if n == 3
        fprintf('The highest real root is %.4f.',x0)
```

```
        break
    end
end
```

The output is below:
The highest real root is 3.0473.

c) The Matlab code for the secant method is below:

```
clear;clc;close all;
f = @(x) x.^3 - 6*x.^2 + 11*x - 6.1;
x0 = 2.5;
x1 = 3.5 ;
n = 0;
while 1
    x2 = x1 - (f(x1)*(x1 - x0))/(f(x1) - f(x0));
    n = n + 1;
    x0 = x1;
    x1 = x2;
    if n == 3
        fprintf('The highest real root is %.4f.',x2)
        break
    end
end
```

The output is below:
The highest real root is 3.2219.

d) The Matlab code for the modified secant method is below:

```
clear;clc;close all;
f = @(x) x.^3 - 6*x.^2 + 11*x - 6.1;
x0 = 3.5;
n = 0;
delta = 0.01;
while 1
    x1 = x0 - (delta*x0*f(x0))/(f(x0+x0*delta)-f(x0));
    n = n + 1;
    x0 = x1;
    if n == 3
        fprintf('The highest real root is %.4f.',x1)
        break
    end
```

```
end
```

The output is below:
The highest real root is 3.0488.

e) The Matlab code is below:

```
clear;clc;close all;
A = [1,-6,11,-6.1];
roots(A)
```

The output is below:
ans =
3.0467
1.8990
1.0544

2.2 Question 6.22

6.22 You are designing a spherical tank (Fig. P6.22) to hold water for a small village in a developing country. The volume of liquid it can hold can be computed as

$$V = \pi h^2 \frac{[3R - h]}{3}$$

where V = volume [m^3], h = depth of water in tank [m], and R = the tank radius [m].

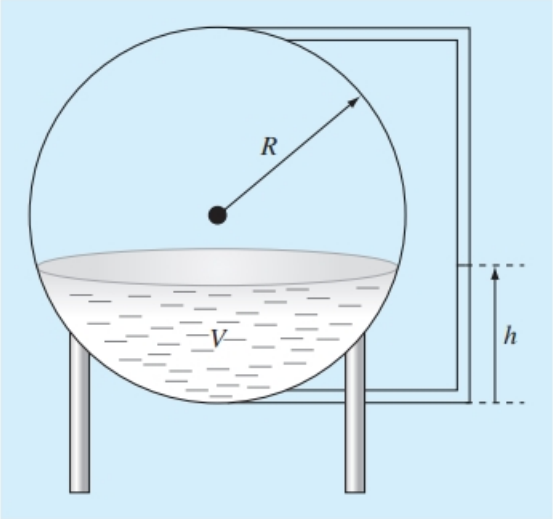


FIGURE P6.22

If $R = 3$ m, what depth must the tank be filled to so that it holds 30 m^3? Use three iterations of the most efficient numerical method possible to determine your answer. Determine the approximate relative error after each iteration. Also, provide justification for your choice of method. Extra information: **(a)** For bracketing methods, initial guesses of 0 and R will bracket a single root for this example. **(b)** For open methods, an initial guess of R will always converge.

a) The Matlab code for the Newton-Raphson method is below:

```
clear;close all;clc;
f = @(h) pi*h^3 - 9*pi*h^2 + 90;
fd = @(h) 3*pi*h^2 - 18*pi*h;
h0 = 3;
n = 0;
fprintf('Iter Root       Relative Error\n')
while 1
    h1 = h0 - f(h0)/fd(h0);
```

```
    n = n + 1;
    error = abs((h1 - h0)/h0)*100;
    h0 = h1;
    fprintf('%d     %.4f      %.4f%%\n',n,h1,error)
    if n == 3
        fprintf('The depth is %.4fm with a relative
error %.4f%%.',h1, error)
        break
    end

end
```

Justification:

Since the Newton-Raphson method can converge much faster, it's the most efficient method possible to find the root in this condition. (Convergence is not an issue)

The output is below:

Iter	Root	Relative Error
1	2.0610	31.2989%
2	2.0270	1.6492%
3	2.0269	0.0067%

The depth is 2.0269m with a relative error 0.0067%.

b) The Matlab code for the Bracketing method is below:

```
clear;close all;clc;
f = @(h) pi*h^3 - 9*pi*h^2 + 90;
xl = 0;
xu = 3;
xr = xl;
iter = 0;
fprintf('Iter Root        Relative Error\n')
while 1
    xrold = xr;
    xr = (xl + xu)/2;
    iter = iter + 1;
    result = f(xr) * f(xl);
    error = abs((xr - xrold)/xr)*100;
    fprintf('%d     %.4f      %.4f%%\n',iter,xr,error)
    if result < 0
        xu = xr;
    elseif result > 0
        xl = xr;
```

```
    elseif result == 0
        break
    end
    if iter == 3
        fprintf('The depth is %.4fm with a relative
error %.4f%%.',xr, error)
        break
    end

end
```

The output is below:

Iter Root Relative Error
1 1.5000 100.0000%
2 2.2500 33.3333%
3 1.8750 20.0000%
The depth is 1.8750m with a relative error 20.0000%.

Conclusion:

From the above method, we can see that the **Newton-Raphson Method is** the most efficient possible method to determine the height since its has the least relative error after three iterations (error = 0.0067%).

2.3 Question 6.27

6.27 Use the Newton-Raphson method to find the root of

$$f(x) = e^{-0.5x}(4 - x) - 2$$

Employ initial guesses of **(a)** 2, **(b)** 6, and **(c)** 8. Explain your results.

The Matlab code is below:

```
function HW6_27(x0)
f = @(x) exp(-0.5*x)*(4-x)-2;
fd = @(x) -3*exp(-0.5*x) + 0.5*x*exp(-0.5*x);
delta = 10^(-5);
while 1
    if fd(x0) == 0
        fprintf('The Newton-Raphson method does not apply here since
the derivative at %d is zero.', x0)
        break
    end
```

```
    x1 = x0 - (f(x0))/(fd(x0));
    error = abs((x1 - x0)/x0);
    x0 = x1;
    if error < delta
        fprintf('The root is %.6f.',x1)
        break
    end
end
end
```

a) The output is below:

The root is 0.885709.

b) The output is below:

The Newton-Raphson method does not apply here since the derivative at 6 is zero.

c) The output is below:

The Newton-Raphson method does not apply here since the derivative at 7.212131e+24 is zero.

2.4 Question 6.28

6.28 Given

$$f(x) = -2x^6 - 1.5x^4 + 10x + 2$$

Use a root-location technique to determine the maximum of this function. Perform iterations until the approximate relative error falls below 5%. If you use a bracketing method, use initial guesses of $x_l = 0$ and $x_u = 1$. If you use the Newton-Raphson or the modified secant method, use an initial guess of $x_i = 1$. If you use the secant method, use initial guesses of $x_{i-1} = 0$ and $x_i = 1$. Assuming that convergence is not an issue, choose the technique that is best suited to this problem. Justify your choice.

a) For the Newton-Raphson Method:

The Matlab code is below:

```
clear;close all;clc
fo = @(x) -2*x^6 - 1.5*x^4 + 10*x + 2;
fod = @(x) -12*x^5 - 6*x^3 + 10;
```

```
fodd = @(x) -60*x^4 - 18*x^2;
delta = 0.05;
x0 = 1;
iter = 0;
while 1
    x1 = x0 - (fod(x0))/(fodd(x0));
    error = abs((x1 - x0)/x0);
    x0 = x1;
    iter = iter + 1;
    if error < delta
        fprintf('The root for the derivative equation is %.5f.\n',x1)
        break
    end
end
f_max = fo(x1);
fprintf('The maximum value is %.5f.\n',f_max)
fprintf('The iter is %d.',iter)
```

Justification: Since the Newton-Raphson method can converge much faster, it's the most efficient method possible to find the root in this condition. (Convergence is not an issue)

The output is below:

The root for the derivative equation is 0.87268.
The maximum value is 8.97341.
The iter is 2

b) For the Bracketing Method:

The Matlab code is below:

```
clear;close all;clc
fo = @(x) -2*x^6 - 1.5*x^4 + 10*x + 2;
fod = @(x) -12*x^5 - 6*x^3 + 10;
fodd = @(x) -60*x^4 - 18*x^2;
delta = 0.05;
x0 = 1;
iter = 0;
while 1
    x1 = x0 - (fod(x0))/(fodd(x0));
    error = abs((x1 - x0)/x0);
    x0 = x1;
    iter = iter + 1;
    if error < delta
        fprintf('The root for the derivative equation is %.5f.\n',x1)
```

```
        break
    end
end
f_max = fo(x1);
fprintf('The maximum value is %.5f.\n',f_max)
fprintf('The iter is %d.',iter)
% Bracketing Method
clear;close all;clc
f = @(x) -2*x^6 - 1.5*x^4 + 10*x + 2;
fd = @(x) -12*x^5 - 6*x^3 + 10;
xl = 0;
xu = 1;
xr = xl;
iter = 0;
delta = 0.05;
while 1
    xrold = xr;
    xr = (xl + xu)/2;
    iter = iter + 1;
    result = fd(xr) * fd(xl);
    error = abs((xr - xrold)/xr);
    if result < 0
        xu = xr;
    elseif result > 0
        xl = xr;
    elseif result == 0
        break
    end
    if error < delta
        break
    end
end
fprintf('The root is %.5f.\n',xr)
f_max = f(xr);
fprintf('The maximum value is %.5f.\n',f_max)
fprintf('The iter is %d.',iter)
```

The output is below:

The root is 0.84375.

The maximum value is 8.95564.

The iter is 5.

c) For the Secant Method:

The Matlab code is below:

```
clear;clc;close all
f = @(x) -2*x^6 - 1.5*x^4 + 10*x + 2;
fd = @(x) -12*x^5 - 6*x^3 + 10;
x0 = 0;
x1 = 1 ;
n = 0;
delta = 0.05;
while 1
    x2 = x1 - (fd(x1)*(x1 - x0))/(fd(x1) - fd(x0));
    error = abs((x2 - x1)/x1);
    n = n + 1;
    x0 = x1;
    x1 = x2;
    if error < delta
        fprintf('The root is %.4f.\n',x2)
        break
    end
end
f_max = f(x2);
fprintf('The maximum value is %.5f.\n',f_max)
fprintf('The iter is %d.',n)
```

The output is below:

The highest real root is 0.8691.
The maximum value is 8.97332.
The iter is 5.

d) For the Modified-Secant Method:

The Matlab code is below:

```
clear;clc;close all
f = @(x) -2*x^6 - 1.5*x^4 + 10*x + 2;
fd = @(x) -12*x^5 - 6*x^3 + 10;
x0 = 1;
n = 0;
delta = 0.01;
threshold = 0.05;
while 1
    x1 = x0 - (delta*x0*fd(x0))/(fd(x0+x0*delta)-fd(x0));
    error = abs((x1 - x0)/x0);
    n = n + 1;
    x0 = x1;
```

```
    if error < threshold
        fprintf('The root is %.4f.\n',x1)
        break
    end
end
f_max = f(x1);

fprintf('The maximum value is %.5f.\n',f_max)

fprintf('The iter is %d.',n)
```

The output is below:
The root is 0.8733.
The maximum value is 8.97336.
The iter is 2.

Conclusion:
From the above method, we can see that the **Newton-Raphson Method and the Modified-Secant Method** are the most efficient possible methods to determine the maximum of the function since they both have the least iteration number(iter = 2).

Lec.7 Optimization

3.1 Question 7.12

7.12 The head of a groundwater aquifer is described in Cartesian coordinates by

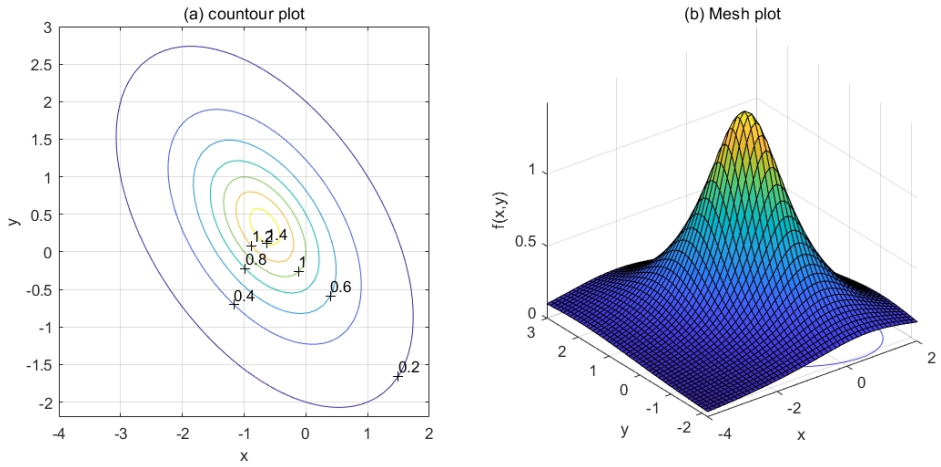
$$h(x, y) = \frac{1}{1 + x^2 + y^2 + x + xy}$$

Develop a single script to **(a)** generate contour and mesh subplots of the function in a similar fashion to Example 7.4, and **(b)** determine the maximum with `fminsearch`.

a) The Matlab code is below:

```
clear;close all;clc
x = linspace(-4,2,50);
y = linspace(-2.2,3,50);
[X,Y] = meshgrid(x,y);
Z = (1./(1 + X.^2 + Y.^2 + X + X.*Y));
subplot(1,2,1);
cs = contour(X,Y,Z);
clabel(cs);
xlabel('x');
ylabel('y');
title('(a) countour plot');
grid on
subplot(1,2,2);
cs = surfc(X,Y,Z);
zmin = floor(min(Z));
zmax = ceil(max(Z));
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('(b) Mesh plot');
```

The output is below:



b) The Matlab code is below:

```
clear;clc;close all
h = @(x) -(1/(1 + x(1)^2 + x(2)^2 + x(1) + x(1)*x(2)));
[x, fval] = fminsearch(h, [-4,2]);
h = (1/(1 + x(1)^2 + x(2)^2 + x(1) + x(1)*x(2)));
```

The output is below:

```
h =

    1.5000
```

3.2 Question 7.25

7.25 Given the following function:

$$f(x, y) = -8x + x^2 + 12y + 4y^2 - 2xy$$

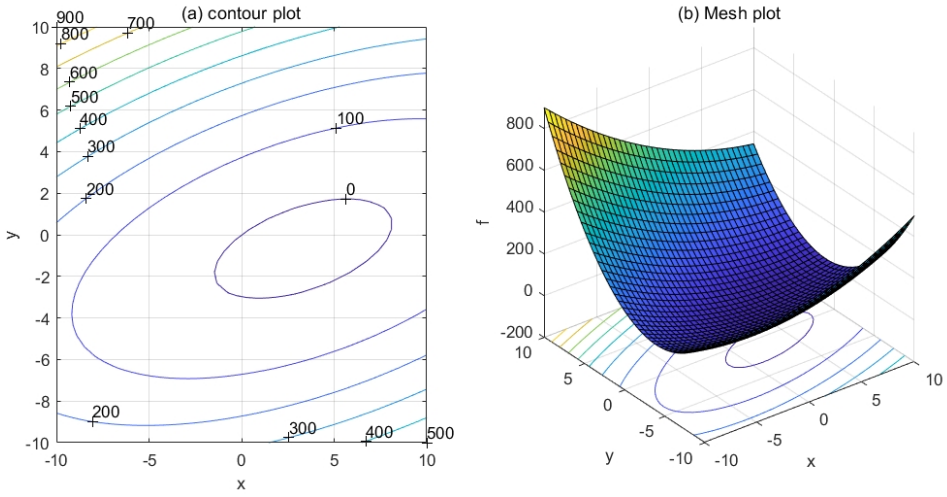
Determine the minimum **(a)** graphically, **(b)** numerically with the `fminsearch` function, and **(c)** substitute the result of **(b)** back into the function to determine the minimum $f(x, y)$.

a) The Matlab code is below:

```
clear;close all;clc
```

```
x = linspace(10,-10,40);
y = linspace(10,-10,40);
[X,Y] = meshgrid(x,y);
Z = -8*X + X.^2 + 12*Y + 4*Y.^2 - 2*X.*Y;
subplot(1,2,1)
cs = contour(X,Y,Z);
clabel(cs);
title('(a) contour plot')
xlabel('x')
ylabel('y')
grid on
subplot(1,2,2)
cs = surfc(X,Y,Z);
zmin = floor(min(Z));
zmax = ceil(max(Z));
title('(b) Mesh plot')
xlabel('x')
ylabel('y')
zlabel('f')
```

The output is below:



From the graph, we can see that the minimum value occurs at x = 3.333 y = -0.7692, and the minimum value is -17.29.

b) The Matlab code is below:

```
clear;clc;close all
f = @(x) -8*x(1) + x(1)^2 + 12*x(2) + 4*x(2)^2 - 2*x(1)*x(2);
[x, fval] = fminsearch(f, [4,-1]);
```

The output is below:

```
x =

    3.3334   -0.6666


fval =

  -17.3333
```

c) The Matlab code is below:

```
clear;clc;close all
x = [3.3334, -0.6666];
f =  -8*x(1) + x(1)^2 + 12*x(2) + 4*x(2)^2 - 2*x(1)*x(2)
```

The output is below:

```
f =

  -17.3333
```

3.3 Question 7.35

7.35 In a similar fashion to the case study described in Sec. 7.4, develop the potential energy function for the system depicted in Fig. P7.35. Develop contour and surface

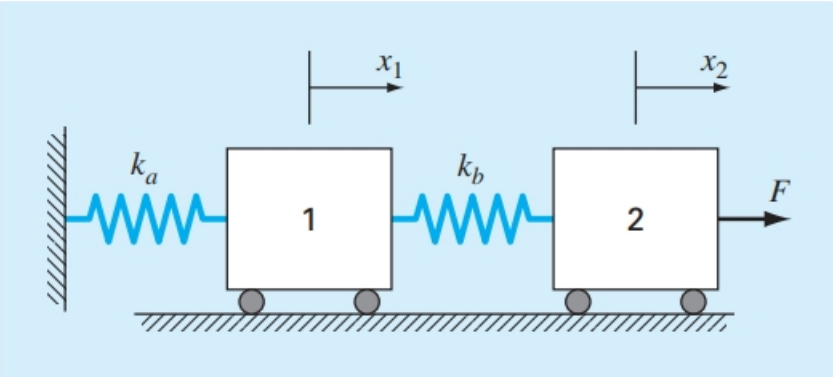


FIGURE P7.35
Two frictionless masses connected to a wall by a pair of linear elastic springs.

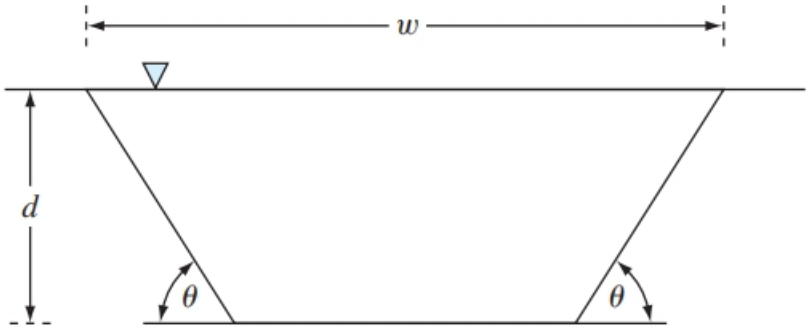


FIGURE P7.36

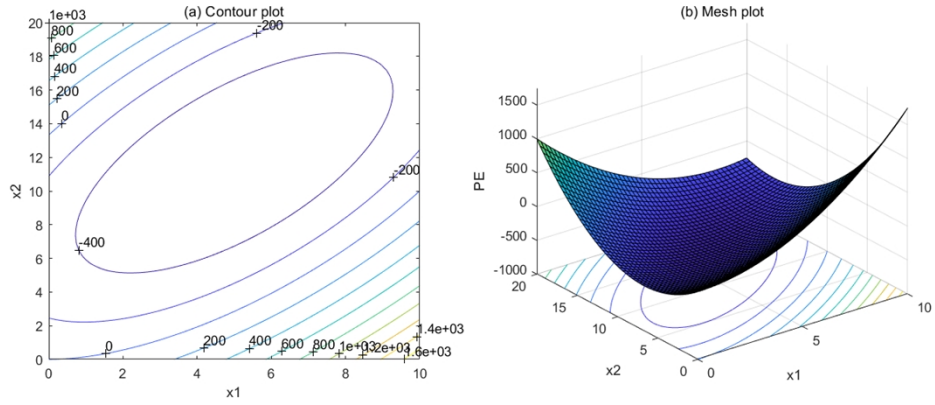
plots in MATLAB. Minimize the potential energy function to determine the equilibrium displacements x_1 and x_2 given the forcing function $F = 100$ N and the parameters $k_a = 20$ and $k_b = 15$ N/m.

The Matlab code for contour and surface plots is below:

```
clear;close all;clc
F = 100;
ka = 20;
```

```
kb = 15;
x1 = linspace(0,10,50);
x2 = linspace(0,20,50);
[X1,X2] = meshgrid(x1,x2);
Z = 0.5*ka*X1.^2 + 0.5*kb*(X2 - X1).^2 - F*X2;
subplot(1,2,1)
cs = contour(X1,X2,Z);
clabel(cs)
xlabel('x1')
ylabel('x2')
title('(a) Contour plot')
subplot(1,2,2)
cs = surfc(X1,X2,Z);
xlabel('x1')
ylabel('x2')
zlabel('PE')
title('(b) Mesh plot')
```

The output is below:



The Matlab code to find the min value is below:

```
clear;close all;clc
F = 100;
ka = 20;
kb = 15;
Z = @(X) 0.5*ka*X(1)^2 + 0.5*kb*(X(2) - X(1))^2 - F*X(2);
[x, fval] = fminsearch(Z, [5,13])
```

The output is below:

```
x =

    5.0000   11.6666
```

```
fval =

  -583.3333
```