Chapter 5 – Problem 5.18

The saturation concentration of dissolved oxygen in fresh-water can be calculated with the equation (APHA, 1992)

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

Where o_{sf} = the saturation concentration of dissolved oxygen in freshwater at 1 atm (mg/L) 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 6.413 mg/L at 40°C. Given a value of oxygen concentration, this formula and the bisection method can be used to solve for temperature in °C.

Develop and test a bisection program to determine T as a function of a given oxygen concentration to a prespecified absolute error. Given initial guesses of 0 and 40°C, test your program for an absolute error = 0.05°C and the following cases: o_{sf} = 8, 10 and 12 mg/L. Check your results.

Your code should consist of a main program (say, *MainBisect*), and a function (*Bisect*):

The main program should do the following:

(1) Input the following values:
- *xl*, the lowest value in a bracket for the bisection method.
- *xu*, the upper value in a bracket for the bisection method.
- *Ead*, the desired absolute error.
- *imax*, the maximum allowed number of iterations before declaring a divergent process.
- *osf*, the variable o_{sf} given for problem 5.18(b) – see more details below.

(2) Check if *xl* and *xu* are suitable for the bisection method, i.e., check that $f(xl) \times f(xu) < 0$. If not, the program should report this situation as an error and stop.

(3) Call function or subroutine *Bisect* (described below) to return a solution.

(4) Outputs the solution returned by *Bisect*, the number of iterations required to solve the equation, the absolute error at the point that the solution was found, and the value of the function $f(x)$ at the solution point (i.e., the values *xr*, *iter*, *Ead*, and $|f(xr)|$ from *Bisect*).

Function or subroutine *Bisect* implements the bisection method according to the pseudocode of **Figure 5.11** (or, if you prefer, that of **Figure 5.10**). However, the pseudocode of Figures 5.10 or 5.11, use the relative percent error, *ea*, as a criterion to stop the program (i.e., the line: IF *ea* < *es* OR *iter* > *imax* EXIT), while Problem 5.18 requires you to use an absolute error, *Ead* = 0.05°C as the stopping criteria. Thus, the IF line in the pseudocode should be replaced by: IF |*xr*-*xrold*| < *Ead* OR *iter* > *imax* EXIT. NOTE: The criteria |*xr*-*xrnew* – *xr*-*xrold*| < *Ead* requires that *Ead* be a number close to zero, such as 0.05°C, in Problem 5.18.

A function $f(x)$ will be necessary in order to define the equation $f(x) = 0$ to be solved. The implementation of this function will depend on the equation being solved. For testing purposes, use $f(x) = x^2 - 5x + 6$, whose solutions, for $f(x) = 0$, are $x = 2$ and $x = 3$. Test the performance of your program using starting values $x_1 = 1.0$, $x_u = 2.5$ for the first solution, and $x_1 = 2.5$, $x_u = 4.0$ for the second solution, using an absolute error of 0.05 for both. After testing the program for this $f(x)$, replace $f(x)$ for the function required to solve problem 5.18, as detailed next.

The equation to solve in problem 5.18 is shown at the top of the right column in page 140 in the textbook. This equation involves the variables T_a and o_{sf}, however, the solution sought requires us to find a value of T, from $T_a = T + 273.15$, for different values of o_{sf}. Thus, the equation for problem 5.18 needs to be rewritten in the form $f(T, o_{sf}) = 0$. The function $f(T, o_{sf})$ should then replace the $f(x)$ function in your code.

Figure 5.11

```
FUNCTION BiSect(xl, xu, es, imax, xr, iter, ea)
   iter = 0
   fl = f(xl)
   DO
      xrold = xr
      xr = (xl + xu) / 2
      fr = f(xr)
      iter = iter + 1
      IF xr ≠ 0 THEN
         ea = ABS((xr − xrold) / xr) * 100
      END IF
      test = fl * fr
      IF test < 0 THEN
         xu = xr
      ELSE IF test > 0 THEN
         xl = xr
         fl = fr
      ELSE
         ea = 0
      END IF
      IF ea < es OR iter ≥ imax EXIT
   END DO
   Bisect = xr
END Bisect
```

Figure 5.10

```
FUNCTION BiSect(xl, xu, es, imax, xr, iter, ea)
   iter = 0
   fl = f(xl)
   DO
      xrold = xr
      xr = (xl + xu) / 2
      iter = iter + 1
      IF xr ≠ 0 THEN
         ea = ABS((xr − xrold) / xr) * 100
      END IF
      test = f(xl) * f(xr)
      IF test < 0 THEN
         xu = xr
      ELSE IF test > 0 THEN
         xl = xr
      ELSE
         ea = 0
      END IF
      IF ea < es OR iter ≥ imax EXIT
   END DO
   Bisect = xr
END Bisect
```

```
// 02/05/2014 - ENGR 2450 - Meine, Joel
// Chapter 5 - Problem 5.18

#include <iostream>
#include <math.h>
using namespace std;

// Oxygen Saturation in Freshwater, osf(mg/L)
const double Tl = 0; // Temperature_lower, Tl(C)
const double Tu = 40; // Temperature_upper, Tu(C)

double Osf(double T,double osf)
{
	double Ta = T + 273.15; // Absolute Temperature, Ta(K); Temperature, T(C)
	double O = -(log(osf)) - 139.34411 + ((1.575701*pow(10,5)) / (pow(Ta,1))) -
((6.642308*pow(10,7)) / (pow(Ta,2))) + ((1.243800*pow(10,10)) / (pow(Ta,3))) - ((8.621949*pow(10, 11))
/ (pow(Ta,4)));
	return O;
}

// Problem Function
double mainF(double xm,double ym)
{
	double Y = Osf(xm,ym);
	return Y;
}

void mainR(double i0,double i1,double i2,double i3,double i4,double i5,int i6)
{
	printf("%2.0f  %2.0f   %2.0f  %2.4f  %2.4f  %2.4f  %2i \n",i0,i1,i2,i3,i4,i5,i6);
}

// Test Function
double testF(double xt)
{
	double G = pow(xt,2) - 5*xt + 6;
	return G;
}

void testR(double j0,double j1,double j2,double j3,double j4,int j5)
{
	printf("%2.1f  %2.1f  %2.4f  %2.4f  %2.4f  %2i \n",j0,j1,j2,j3,j4,j5);
}

// Bisection Method
void BiSect(int F,double C,double xl,double xu,double Ead,int Imax)
{
	double xli = 0; // Bracket_lower (initial)
	double xui = 0; // Bracket_upper (initial)
	int iter = 0; // Iteration
	double ea = Ead + 1; // Absolute Error, actual (initial)
	double xr = 0; // Root of Function (initial)
	double xrold = 0; // Root of Function (previous)

	double fl = 0; // Function Value at Bracket_lower (initial)
	if (F == 0) fl = mainF(xl,C);  // Function Value at Bracket_lower (actual function)
	else if (F == 1) fl = testF(xl);  // Function Value at Bracket_lower (test function)
```

```
	double fu = 0; // Function Value at Bracket_upper (initial)
	if (F == 0) fu = mainF(xu,C);  // Function Value at Bracket_upper (actual function)
	else if (F == 1) fu = testF(xu);  // Function Value at Bracket_upper (test function)
	double fr = 0; // Function Value at Root (initial)

	do
	{
		if (iter == 0)
		{
			xli = xl;
			xui = xu;
		}
		xrold = xr;
		xr = (xl + xu) / 2;
		ea = abs(xr-xrold);
		if (F == 0) fr = mainF(xr,C);
		else if (F == 1) fr = testF(xr);
		iter++;
		double test = fl * fr;
		if (test < 0) xu = xr;
		else if (test > 0)
		{
			xl = xr;
			fl = fr;
		}
		else ea = 0;
	} while (ea >= Ead && iter < Imax && fl*fu < 0);
	if (ea < Ead)
	{
		if (F == 0)
		{
			fr = mainF(xr,C);
			mainR(C,xli,xui,xr,fr,ea,iter);
		}
		else if (F == 1)
		{
			fr = testF(xr);
			testR(xli,xui,xr,fr,ea,iter);
		}
	}
	else if (iter >= Imax) std::cout << "No Solution due to Divergence" << std::endl;
	else if (fl*fu >= 0) std::cout << "Invalid Values for xl and xu" << std::endl;
}

int main()
{
	const double ead = 0.05; // Absolute Error
	const int imax = 100; // Maximum Number of Iterations

	std::cout << "Chapter 5 - Problem 5.18" << std::endl;
	std::cout << "============================================" << std::endl;
	std::cout << "Bracket_lower, xl" << std::endl;
	std::cout << "Bracket_upper, xu" << std::endl;
	std::cout << "Root of Function, xr" << std::endl;
	std::cout << "Function Value at Root, f(xr)" << std::endl;
	std::cout << "Absolute Error, ead = " << ead << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
```

```
Chapter 5 - Problem 5.18
============================================
Bracket_lower, xl
Bracket_upper, xu
Root of Function, xr
Function Value at Root, f(xr)
Absolute Error, ead = 0.05
Actual Error, ea
Maximum Number of Iterations, imax = 100
Iterations, iter
******************************************
Test Function
------------------------------------------
xl   xu   xr      f(xr)   ea      iter
------------------------------------------
1.0  2.5  1.9844  0.0159  0.0469   5
2.5  4.0  3.0156  0.0159  0.0469   5
******************************************
Problem Function
------------------------------------------
Oxygen Saturation in Freshwater, osf(mg/L)
Temperature_lower, Tl(C)
Temperature_upper, Tu(C)
Temperature_root, Tr(C)
Function Value at Temperature_root, f(Tr)
------------------------------------------
osf  Tl   Tu  Tr       f(Tr)   ea      iter
------------------------------------------
 8   0    40  26.7578  0.0004  0.0391  10
10   0    40  15.3516  0.0008  0.0391  10
12   0    40  7.4609  0.0001  0.0391  10
******************************************

Press any key to continue . . .
```

```
    std::cout << "Maximum Number of Iterations, imax = " << imax << std::endl;
    std::cout << "Iterations, iter" << std::endl;
    std::cout << "*********************************************" << std::endl;

    std::cout << "Test Function" << std::endl;
    std::cout << "---------------------------------------------" << std::endl;
    std::cout << "xl   xu   xr      f(xr)   ea      iter" << std::endl;
    std::cout << "---------------------------------------------" << std::endl;
    BiSect(1,0,1.0,2.5,ead,imax);
    BiSect(1,0,2.5,4.0,ead,imax);
    std::cout << "*********************************************" << std::endl;

    std::cout << "Problem Function" << std::endl;
    std::cout << "---------------------------------------------" << std::endl;
    std::cout << "Oxygen Saturation in Freshwater, osf(mg/L)" << std::endl;
    std::cout << "Temperature_lower, Tl(C)" << std::endl;
    std::cout << "Temperature_upper, Tu(C)" << std::endl;
    std::cout << "Temperature_root, Tr(C)" << std::endl;
    std::cout << "Function Value at Temperature_root, f(Tr)" << std::endl;
    std::cout << "---------------------------------------------" << std::endl;
    std::cout << "osf  Tl  Tu  Tr       f(Tr)   ea     iter" << std::endl;
    std::cout << "---------------------------------------------" << std::endl;
    BiSect(0,8,Tl,Tu,ead,imax); // osf = 8
    BiSect(0,10,Tl,Tu,ead,imax); // osf = 10
    BiSect(0,12,Tl,Tu,ead,imax); // osf = 12
    std::cout << "******************************************** \n" << std::endl;

    system("pause");
    return 0;
}
```

Chapter 5 – Problem 5.22

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

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

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 $P_{u,\max}$ = 75,000, k_u = 0.045/yr, $P_{u,\min}$ = 100,000 people, $P_{s,\max}$ = 300,000 people, P_0 = 10,000 people, k_s = 0.08/yr. To obtain your solution, use the false-position method.

Your code should consist of a main program (say *MainFalsePosition*), and a function (*ModFalsePos*):

The main program should do the following:

(1) Input the following values:
- *xl*, the lowest value in a bracket for the modified false position method.
- *xu*, the upper value in a bracket for the modified false position method.
- *es*, the percent error tolerance for convergence; *es* = 0.05%.
- *imax*, the maximum allowed number of iterations before declaring a divergent process.
- Other parameters in the problem ($P_{u,\max}$, k_u, $P_{u,\min}$, $P_{s,\max}$, k_s, and P_0) can be defined in code, e.g., *Pumax* = 75000, etc.

(2) Check if *xl* and *xu* are suitable for the modified false position method, i.e., check that $f(xl)$*f(xu) < 0. If not, the program should report this situation as an error and stop.

(3) Call subroutine or function *ModFalsePos* (described below) to return a solution.

(4) Outputs the solution returned by *ModFalsePos*, the number of iterations required to solve the equation, the percent relative error of the approximation at the point that the solution was found, and the value of the function $f(x)$ at the solution point (i.e., the values *xr*, *iter*, *ea* from *ModFalsePos*, and $f(xr)$).

Subroutine or function *ModFalsePos* implements the modified false position method according to the pseudocode of **Figure 5.15**.

A third function $f(x)$ will be necessary in order to define the equation $f(x) = 0$ to be solved. For testing purposes, use $f(x) = x^2 - 5x + 6$, whose solutions are $x = 2$ and $x = 3$. Test the performance of your program using starting values $x_1 = 1.0$, $x_u = 2.5$ for the first solution, and $x_1 = 2.5$, $x_u = 4.0$ for the second solution, using a relative percent approximation error, *ea* = 0.05% for both. After testing the program for this $f(x)$, replace $f(x)$ for the function required to solve problem 5.22 as detailed below.

Problem 5.22 gives functions $P_u(t)$ and $P_s(t)$ representing population of urban and suburban areas in a city (page 141 in the textbook), and requests that you find the value of t "when the suburbs are 20% larger than the city", i.e., when $P_u(t) = 1.2\ P_s(t)$. To solve for t using the modified false position methods, you need to produce a function $f(t) = 0$ out of the last equation. This $f(t)$ will replace the $f(x)$ used above to verify your program.

Figure 5.15

```
FUNCTION ModFalsePos(xl, xu, es, imax, xr, iter, ea)
  iter = 0
  fl = f(xl)
  fu = f(xu)
  DO
    xrold = xr
    xr = xu - fu * (xl − xu) / (fl − fu)
    fr = f(xr)
    iter = iter + 1
    IF xr ≠ 0 THEN
      ea = ABS((xr − xrold) / xr) * 100
    END IF
    test = fl * fr
    IF test < 0 THEN
      xu = xr
      fu = f(xu)
      iu = 0
      il = il + 1
      IF il ≥ 2 THEN fl = fl / 2
    ELSE IF test > 0 THEN
      xl = xr
      fl = f(xl)
      il = 0
      iu = iu + 1
      IF iu ≥ 2 THEN fu = fu / 2
    ELSE
      ea = 0
    END IF
    IF ea < es OR iter ≥ imax EXIT
  END DO
  ModFalsePos = xr
END ModFalsePos
```

```
// 02/05/2014 - ENGR 2450 - Meine, Joel
// Chapter 5 - Problem 5.22

#include <iostream>
#include <math.h>
using namespace std;

// Population of Urban and Suburban Areas
const double Pumax = 75000; // Population, Urban (maximum)
const double ku = 0.045; // Population Rate of Change, Urban
const double Pumin = 100000; // Population, Urban (minimum)
const double Psmax = 300000; // Population, Suburban (maximum)
const double ks = 0.08; // Population Rate of Change, Suburban
const double P0 = 10000; // Population (initial)

const double Tl = 0; // Time_lower, Tl(yr)
const double Tu = 100; // Time_upper, TU(yr)

double P(double T,double Pc)
{
	double Pu = Pumax * exp(-ku*T) + Pumin; // Population, Urban
	double Ps = Psmax / (1 + (Psmax/(P0-1)) * exp(-ks*T)); // Population, Suburban
	double Pt = Pc - (Ps/Pu); // Suburb-to-Urban Population Comparison
	return Pt;
}

// Problem Function
double mainF(double xm,double ym)
{
	double Y = P(xm,ym);
	return Y;
}

void mainR(double i0,double i1,double i2,double i3,double i4,double i5,int i6)
{
	printf("%2.1f  %2.0f   %2.0f  %2.4f  %2.4f  %2.4f  %2i \n",i0,i1,i2,i3,i4,i5,i6);
}

// Test Function
double testF(double xt)
{
	double G = pow(xt,2) - 5*xt + 6;
	return G;
}

void testR(double j0,double j1,double j2,double j3,double j4,int j5)
{
	printf("%2.1f  %2.1f  %2.4f  %2.4f  %2.4f  %2i \n",j0,j1,j2,j3,j4,j5);
}

// Modified False-Position Method
void ModFalsePos(int F,double C,double xl,double xu,double Es,int Imax)
{
	double xli = 0; // Bracket_lower (initial)
	double xui = 0; // Bracket_upper (initial)
	int iter = 0; // Iteration
	int il = 0; // Iteration_lower
```

```
	int iu = 0; // Iteration_upper
	double ea = Es + 1; // Absolute Error, actual (initial)
	double xr = 0; // Root of Function (initial)
	double xrold = 0; // Root of Function (previous)

	double fl = 0; // Function Value at Bracket_lower (initial)
	if (F == 0) fl = mainF(xl,C);  // Function Value at Bracket_lower (actual function)
	else if (F == 1) fl = testF(xl);  // Function Value at Bracket_lower (test function)
	double fu = 0; // Function Value at Bracket_upper (initial)
	if (F == 0) fu = mainF(xu,C);  // Function Value at Bracket_upper (actual function)
	else if (F == 1) fu = testF(xu);  // Function Value at Bracket_upper (test function)
	double fr = 0; // Function Value at Root (initial)

	do
	{
		if (iter == 0)
		{
			xli = xl;
			xui = xu;
		}
		xrold = xr;
		xr = xu - fu * (xl-xu)/(fl-fu);
		if (F == 0) fr = mainF(xr,C);
		else if (F == 1) fr = testF(xr);
		iter++;
		if (xr != 0) ea = abs((xr-
xrold)/xr)*100;
		double test = fl * fr;
		if (test < 0)
		{
			xu = xr;
			if (F == 0) fu = mainF(xu,C);
			else if (F == 1) fu = testF(xu);
			iu = 0;
			il++;
			if (il >= 2) fl = fl/2;
		}
		else if (test > 0)
		{
			xl = xr;
			if (F == 0) fl = mainF(xl,C);
			else if (F == 1) fl = testF(xl);
			il = 0;
			iu++;
			if (iu >= 2) fu = fu/2;
		}
		else ea = 0;
	} while (ea >= Es && iter < Imax && fl*fu <
0);
	if (ea < Es)
	{
		if (F == 0)
		{
			fr = mainF(xr,C);
			mainR(C,xli,xui,xr,fr,ea,iter);
		}
		else if (F == 1)
```

```
Chapter 5 - Problem 5.22
============================================
Bracket_lower, xl
Bracket_upper, xu
Root of Function, xr
Function Value at Root, f(xr)
Error Criteria, es = 0.0005
Actual Error, ea
Maximum Number of Iterations, imax = 100
Iterations, iter
********************************************
Test Function
--------------------------------------------
xl   xu   xr      f(xr)    ea      iter
--------------------------------------------
1.0  2.5  2.0000  -0.0000  0.0000  9
2.5  4.0  3.0000  -0.0000  0.0000  9
********************************************
Problem Function
--------------------------------------------
Population, Urban (maximum), Pumax = 75000
Population Rate of Change (Urban), ku = 0.045
Population, Urban (minimum), Pumin = 100000
Population, Suburban (maximum), Psmax = 300000
Population Rate of Change (Suburban), ks = 0.08
Population (initial), P0 = 10000
Suburb-to-Urban Population Comparison, Pc
Time_lower, Tl(yr)
Time_upper, Tu(yr)
Time_root, Tr(yr)
Function Value at Time_root, f(Tr)
--------------------------------------------
Pc    Tl   Tu   Tr        f(Tr)    ea      iter
--------------------------------------------
1.2   0    100  39.9884   -0.0000  0.0000  5
********************************************

Press any key to continue . . .
```

```
			{
				fr = testF(xr);
				testR(xli,xui,xr,fr,ea,iter);
			}
		}
	else if (iter >= Imax) std::cout << "No Solution due to Divergence" << std::endl;
	else if (fl*fu >= 0) std::cout << "Invalid Values for xl and xu" << std::endl;
}

int main()
{
	const double es = 0.0005; // Error Criteria
	const int imax = 100; // Maximum Number of Iterations

	std::cout << "Chapter 5 - Problem 5.22" << std::endl;
	std::cout << "==============================================" << std::endl;
	std::cout << "Bracket_lower, xl" << std::endl;
	std::cout << "Bracket_upper, xu" << std::endl;
	std::cout << "Root of Function, xr" << std::endl;
	std::cout << "Function Value at Root, f(xr)" << std::endl;
	std::cout << "Error Criteria, es = " << es << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
	std::cout << "Maximum Number of Iterations, imax = " << imax << std::endl;
	std::cout << "Iterations, iter" << std::endl;
	std::cout << "**********************************************" << std::endl;

	std::cout << "Test Function" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	std::cout << "xl    xu    xr      f(xr)    ea      iter" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	ModFalsePos(1,0,1.0,2.5,es,imax);
	ModFalsePos(1,0,2.5,4.0,es,imax);
	std::cout << "**********************************************" << std::endl;

	std::cout << "Problem Function" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	std::cout << "Population, Urban (maximum), Pumax = " << Pumax << std::endl;
	std::cout << "Population Rate of Change (Urban), ku = " << ku << std::endl;
	std::cout << "Population, Urban (minimum), Pumin = " << Pumin << std::endl;
	std::cout << "Population, Suburban (maximum), Psmax = " << Psmax << std::endl;
	std::cout << "Population Rate of Change (Suburban), ks = " << ks << std::endl;
	std::cout << "Population (initial), P0 = " << P0 << std::endl;
	std::cout << "Suburb-to-Urban Population Comparison, Pc" << std::endl;
	std::cout << "Time_lower, Tl(yr)" << std::endl;
	std::cout << "Time_upper, Tu(yr)" << std::endl;
	std::cout << "Time_root, Tr(yr)" << std::endl;
	std::cout << "Function Value at Time_root, f(Tr)" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	std::cout << "Pc     Tl   Tu   Tr        f(Tr)    ea     iter" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	ModFalsePos(0,1.2,Tl,Tu,es,imax); // Pc = 1.2; Pc > 1.% (% larger than) && Pc < 1.% (% smaller
than)
	std::cout << "********************************************* \n" << std::endl;

	system("pause");
	return 0;
}
```

Chapter 6 – Problem 6.30

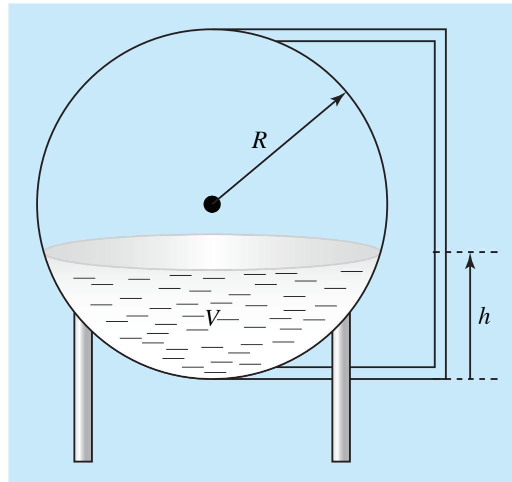
You are designing a spherical tank (Fig. P6.30) to hold water for a small village in a developing country. The volume of liquid it can hold can be computed as

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

where V = volume (m³), h = depth of water in tank (m), and R = the tank radius (m).

If R = 3 m, what depth must the tank be filled to so that it holds 30 m³? Use three iterations of the Newton-Raphson method to determine your answer. Determine the approximate relative error after each iteration. Note that an initial guess R will always converge.

Figure P6.30



Your code should consists of a main program (say *MainNewtonRaphson*), and a function (*NewtonRaphson*):

The main program should do the following:
(1) Input the following values:
- $x0$, initial guess for the Newton-Raphson method.
- *es*, the percent error tolerance for convergence; *es* = 0.05%.
- *imax*, the maximum allowed number of iterations before declaring a divergent process.

(2) Call function *NewtonRaphson* to return a solution.
(3) Outputs the solution returned by *NewtonRaphson*, the number of iterations required to solve the equation, the percent relative error of the approximation at the point that the solution was found, and the value of the function $f(x)$ at the solution point (i.e., the values *xr*, *iter*, *ea* from *NewtonRaphson*, and $f(xr)$).

Function *NewtonRaphson* implements the Newton-Raphson method according to the algorithm described in **section 6.2.3**.

Two more functions will be necessary in order to implement the Newton-Raphson method: function $f(x)$ that defines the equation $f(x) = 0$ to be solved, and function $fp(x) = f'(x)$, its derivative. For testing purposes, use $f(x) = x^2 - 5x + 6$, whose solutions are $x = 2$ and $x = 3$. For this case, $fp(x) = 2x - 5$. After testing the program for this $f(x)$ and its derivative $fp(x)$, replace $f(x)$ and $fp(x)$ with the function (and its derivative) required to solve problem 6.30.

```
// 02/05/2014 - ENGR 2450 - Meine, Joel
// Chapter 6 - Problem 6.30

#include <iostream>
#include <math.h>
using namespace std;

// Spherical Tank Volume
const double R = 3;
const double PI = 3.141592653589793;

double V(double h,double v)
{
	double Vr = PI * pow(h,2) * ((3*R-h)/3) - v;
	return Vr;
}

double dV(double dh)
{
	double dv = PI*2*R*dh - PI*pow(dh,2);
	return dv;
}

// Problem Function
double mainF(double xm,double ym)
{
	double Y = V(xm,ym);
	return Y;
}

double maindF(double dxm)
{
	double dY = dV(dxm);
	return dY;
}

void mainR(double i0,double i1,double i2,double
i3,double i4,int i5)
{
	printf("%2.0f %2.0f  %2.3f %2.4f  %2.4f  %2i \n",i0,i1,i2,i3,i4,i5);
}

// Test Function
double testF(double xt)
{
	double G = pow(xt,2) - 5*xt + 6;
	return G;
}

double testdF(double dxt)
{
	double dG = 2*dxt - 5;
	return dG;
}

void testR(double j0,double j1,double j2,double j3,int j4)
{
```

```
Chapter 6 - Problem 6.30
============================================
Guess_initial, x0
Root of Function, xr
Function Value at Root, f(xr)
Error Criteria, es = 0.0005
Actual Error, ea
Maximum Number of Iterations, imax = 100
Iterations, iter
********************************************
Test Function
--------------------------------------------
x0   xr     f(xr)  ea     iter
--------------------------------------------
5.0  3.0000 0.0000 0.0001  6
1.0  2.0000 0.0000 0.0000  6
********************************************
Problem Function
--------------------------------------------
Volume of Spherical Tank, V(m^3)
Spherical Tank Radius, R(m) = 3
Depth of Liquid in Tank, h(m)
Depth_initial, h0(m)
Depth_root, hr(m)
Function Value at Depth_root, f(hr)
Iteration, iter.
--------------------------------------------
iter.  V     h0     hr      f(hr)  ea
--------------------------------------------
  1    30.0  10.00  8.928  -23.98  12.01
  2    30.0  10.00  8.636  -1.56   3.38
  3    30.0  10.00  8.614  -0.01   0.25
********************************************
V  h0  hr     f(hr)   ea      iter
--------------------------------------------
30 10  8.614 -0.0000  0.0000   5
********************************************

Press any key to continue . . .
```

```
	printf("%2.1f  %2.4f %2.4f %2.4f %2i \n",j0,j1,j2,j3,j4);
}

// Newton-Raphson Method
void NewtonRaphson(int F,double C,double X0,double Es,int Imax,int Ilim)
{
	int iter = 0; // Iteration
	double ea = Es + 1; // Absolute Error, actual (initial)
	double xr = X0; // Root of Function (initial)
	double xrold = 0; // Root of Function (previous)
	double fr = 0; // Function Value at Root (initial)

	do
	{
		xrold = xr;
		if (F == 0) xr = xrold - (mainF(xrold,C)/maindF(xrold));
		else if (F == 1) xr = xrold - (testF(xrold)/testdF(xrold));
		iter++;
		if (xr != 0) ea = abs((xr-xrold)/xr)*100;
		if (F == 0 && iter <= Ilim)
		{
			fr = mainF(xr,C);
			printf(" %2i    %2.1f  %2.2f  %2.3f  %2.2f  %2.2f \n",iter,C,X0,xr,fr,ea);
		}
	} while (ea >= Es && iter < Imax);
	if (ea < Es)
	{
		if (F == 0 && Ilim <= -1)
		{
			fr = mainF(xr,C);
			mainR(C,X0,xr,fr,ea,iter);
		}
		else if (F == 1)
		{
			fr = testF(xr);
			testR(X0,xr,fr,ea,iter);
		}
	}
	else if (iter >= Imax) std::cout << "No Solution due to Divergence" << std::endl;
}

int main()
{
	const double es = 0.0005; // Error Criteria;
	const int imax = 100; // Maximum Number of Iterations

	std::cout << "Chapter 6 - Problem 6.30" << std::endl;
	std::cout << "=============================================" << std::endl;
	std::cout << "Guess_initial, x0" << std::endl;
	std::cout << "Root of Function, xr" << std::endl;
	std::cout << "Function Value at Root, f(xr)" << std::endl;
	std::cout << "Error Criteria, es = " << es << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
	std::cout << "Maximum Number of Iterations, imax = " << imax << std::endl;
	std::cout << "Iterations, iter" << std::endl;
	std::cout << "*********************************************" << std::endl;
```

```
        std::cout << "Test Function" << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        std::cout << "x0   xr     f(xr)  ea     iter" << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        NewtonRaphson(1,0,5.0,es,imax,0);
        NewtonRaphson(1,0,1.0,es,imax,0);
        std::cout << "*********************************************" << std::endl;

        std::cout << "Problem Function" << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        std::cout << "Volume of Spherical Tank, V(m^3)" << std::endl;
        std::cout << "Spherical Tank Radius, R(m) = " << R << std::endl;
        std::cout << "Depth of Liquid in Tank, h(m)" << std::endl;
        std::cout << "Depth_initial, h0(m)" << std::endl;
        std::cout << "Depth_root, hr(m)" << std::endl;
        std::cout << "Function Value at Depth_root, f(hr)" << std::endl;
        std::cout << "Iteration, iter." << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        std::cout << "iter.  V     h0     hr      f(hr)  ea" << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        NewtonRaphson(0,30,10,es,imax,3);
        std::cout << "*********************************************" << std::endl;
        std::cout << "V  h0  hr     f(hr)   ea      iter" << std::endl;
        std::cout << "---------------------------------------------" << std::endl;
        NewtonRaphson(0,30,10,es,imax,-1);
        std::cout << "********************************************* \n" << std::endl;

        system("pause");
        return 0;
}
```

Chapter 6 – Problem 6.18

A mass balance for a pollutant in a well-mixed lake can be written as

$$V\frac{dc}{dt} = W - Qc - kV\sqrt{c}$$

Given the parameter values $V = 1 \times 10^6 \text{m}^3$, $Q = 1 \times 10^5\ \text{m}^3/\text{yr}$, $W = 1 \times 10^6$ g/yr, and $k = 0.25\ \text{m}^{0.5}/\text{g}^{0.5}/\text{yr}$, use the modified secant method to solve for the steady-state concentration. Employ an initial guess of $c = 4\ \text{g/m}^3$ and $\delta = 0.5$. Perform three iterations and determine the percent relative error after the third iteration.

Your code should consists of a main program (say *MainModSecant*), and a function (*ModSecant*):

The main program should do the following:

(1) Input the following values:
- $x0$, initial guess for the modified secant method.
- δ, increment for the modified secant method; $\delta = 0.001$.
- *es*, the percent error tolerance for convergence; *es* = 0.05%.
- *imax*, the maximum allowed number of iterations before declaring a divergent process.

(2) Call function *ModSecant* to return a solution.

(3) Outputs the solution returned by *ModSecant*, the number of iterations required to solve the equation, the percent relative error of the approximation at the point that the solution was found, and the value of the function $f(x)$ at the solution point (i.e., the values *xr*, *iter*, *ea* from *ModSecant*, and $f(xr)$).

Function *ModSecant* implements the modified secant method (see **section 6.3.3**).

A third function $f(x)$ will be necessary in order to define the equation $f(x) = 0$ to be solved. The implementation of this function will depend on the equation being solved. For testing purposes, use $f(x) = x^2 - 5x + 6$, whose solutions are $x = 2$ and $x = 3$. After testing the program for this $f(x)$, replace $f(x)$ for the function required to solve problem 6.18.

```
// 02/05/2014 - ENGR 2450 - Meine, Joel
// Chapter 6 - Problem 6.18

#include <iostream>
#include <math.h>
using namespace std;

const double V = 1 * pow(10,6); // Volume (m^3)
const double Q = 1 * pow(10,5); // Volume per Year
(m^3/yr)
const double W = 1 * pow(10,6); // Mass per Year
(g/yr)
const double k = 0.25; // Volume per Mass per Year
(m^0.5/g^0.5/yr)

// Steady-State Concentration of Pollutant
double pSS(double c,double t)
{
	double p = -c + t*(W/V) - t*((Q*c)/V) -
pow(c,0.5)*k*t;
	return p;
}

// Problem Function
double mainF(double xm,double ym)
{
	double Y = pSS(xm,ym);
	return Y;
}

void mainR(double i0,double i1,double i2,double
i3,double i4,double i5,int i6)
{
	printf("%2.0f  %2.3f  %2.2f  %2.3f  %2.3f
%2.3f   %2i \n",i0,i1,i2,i3,i4,i5,i6);
}

// Test Function
double testF(double xt)
{
	double G = pow(xt,2) - 5*xt + 6;
	return G;
}

void testR(double j0,double j1,double j2,double
j3,double j4,int j5)
{
	printf("%2.1f %2.3f %2.3f %2.3f %2.3f %2i \n",j0,j1,j2,j3,j4,j5);
}

// Modified Secant Method
void ModSecant(int F,double C,double X0,double D,double Es,int Imax,int Ilim)
{
	int iter = 0; // Iteration
	double ea = Es + 1; // Absolute Error, actual (initial)
	double xr = X0; // Root of Function (initial)
	double xrold = 0; // Root of Function (previous)
```

```
Chapter 6 - Problem 6.18
============================================
Guess_initial, x0
Increment, d
Root of Function, xr
Function Value at Root, f(xr)
Error Criteria, es = 0.0005
Actual Error, ea
Maximum Number of Iterations, imax = 100
Iterations, iter
********************************************
Test Function
--------------------------------------------
x0  d      xr     f(xr)  ea     iter
--------------------------------------------
5.0 0.001 3.000 0.000 0.000  6
1.0 0.001 2.000 -0.000 0.000  5
********************************************
Problem Function
--------------------------------------------
Volume, V(m^3) = 1e+006
Volume per Year, Q(m^3/yr) = 100000
Mass per Year, W(g/yr) = 1e+006
Volume per Mass per Year, k(m^0.5/g^0.5/yr) = 0.25
Time, t(yr)
cVariable_initial, c0(g/m^3)
cVaribale_root, cr(g/m^3)
Function Value at cVaribale_root, f(cr)
Iteration, iter.
--------------------------------------------
iter. t     d     c0    cr     f(cr)  ea
--------------------------------------------
  1   10.0  0.500 4.00 2.829  0.137  41.394
  2   10.0  0.500 4.00 2.880  -0.004  1.785
  3   10.0  0.500 4.00 2.879  0.000  0.048
********************************************
t   d      c0    cr     f(cr)  ea     iter
--------------------------------------------
 5  0.500  4.00  2.120  0.000  0.000    5
10  0.500  4.00  2.879  0.000  0.000    5
15  0.500  4.00  3.282  0.000  0.000    5
20  0.500  4.00  3.534  0.000  0.000    5
25  0.500  4.00  3.705  -0.000  0.000    4
30  0.500  4.00  3.830  -0.000  0.000    4
********************************************

Press any key to continue . . .
```

```
	double fr = 0; // Function Value at Root (initial)

	do
	{
		xrold = xr;
		if (F == 0) xr = xrold - ((D*xrold*mainF(xrold,C))/(mainF(xrold+D*xrold,C)-
mainF(xrold,C)));
		else if (F == 1) xr = xrold - ((D*xrold*testF(xrold))/(testF(xrold+D*xrold)-
testF(xrold)));
		iter++;
		if (xr != 0) ea = abs((xr-xrold)/xr)*100;
		if (F == 0 && iter <= Ilim)
		{
			fr = mainF(xr,C);
			printf(" %2i   %2.1f  %2.3f %2.2f %2.3f  %2.3f  %2.3f \n",iter,C,D,X0,xr,fr,ea);
		}
	} while (ea >= Es && iter < Imax);
	if (ea < Es)
	{
		if (F == 0 && Ilim <= -1)
		{
			fr = mainF(xr,C);
			mainR(C,D,X0,xr,fr,ea,iter);
		}
		else if (F == 1)
		{
			fr = testF(xr);
			testR(X0,D,xr,fr,ea,iter);
		}
	}
	else if (iter >= Imax) std::cout << "No Solution due to Divergence" << std::endl;
}

int main()
{
	const double dT = 0.001; // Increment, Test
	const double dP = 0.5; // Increment, Problem
	const double es = 0.0005; // Error Criteria;
	const int imax = 100; // Maximum Number of Iterations

	std::cout << "Chapter 6 - Problem 6.18" << std::endl;
	std::cout << "==============================================" << std::endl;
	std::cout << "Guess_initial, x0" << std::endl;
	std::cout << "Increment, d" << std::endl;
	std::cout << "Root of Function, xr" << std::endl;
	std::cout << "Function Value at Root, f(xr)" << std::endl;
	std::cout << "Error Criteria, es = " << es << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
	std::cout << "Maximum Number of Iterations, imax = " << imax << std::endl;
	std::cout << "Iterations, iter" << std::endl;
	std::cout << "**********************************************" << std::endl;

	std::cout << "Test Function" << std::endl;
	std::cout << "----------------------------------------------" << std::endl;
	std::cout << "x0  d     xr     f(xr)  ea    iter" << std::endl;
	std::cout << "----------------------------------------------" << std::endl;
	ModSecant(1,0,5,dT,es,imax,0);
```

```
    ModSecant(1,0,1,dT,es,imax,0);
    std::cout << "********************************************" << std::endl;

    std::cout << "Problem Function" << std::endl;
    std::cout << "--------------------------------------------" << std::endl;
    std::cout << "Volume, V(m^3) = " << V << std::endl;
    std::cout << "Volume per Year, Q(m^3/yr) = " << Q << std::endl;
    std::cout << "Mass per Year, W(g/yr) = " << W << std::endl;
    std::cout << "Volume per Mass per Year, k(m^0.5/g^0.5/yr) = " << k << std::endl;
    std::cout << "Time, t(yr)" << std::endl;
    std::cout << "cVariable_initial, c0(g/m^3)" << std::endl;
    std::cout << "cVaribale_root, cr(g/m^3)" << std::endl;
    std::cout << "Function Value at cVaribale_root, f(cr)" << std::endl;
    std::cout << "Iteration, iter." << std::endl;
    std::cout << "--------------------------------------------" << std::endl;
    std::cout << "iter. t     d     c0   cr     f(cr)  ea" << std::endl;
    std::cout << "--------------------------------------------" << std::endl;
    ModSecant(0,10,4,dP,es,imax,3);
    std::cout << "********************************************" << std::endl;
    std::cout << "t   d      c0    cr     f(cr)  ea     iter" << std::endl;
    std::cout << "--------------------------------------------" << std::endl;
    ModSecant(0,5,4,dP,es,imax,-1);
    ModSecant(0,10,4,dP,es,imax,-1);
    ModSecant(0,15,4,dP,es,imax,-1);
    ModSecant(0,20,4,dP,es,imax,-1);
    ModSecant(0,25,4,dP,es,imax,-1);
    ModSecant(0,30,4,dP,es,imax,-1);
    std::cout << "******************************************** \n" << std::endl;

    system("pause");
    return 0;
}
```

Chapter 7 – Problem 7.19

In control systems analysis, transfer functions are developed that mathematically relate the dynamics of a system's input to its output. A transfer function for a robotic positioning system is given by

$$G(s)=\frac{C(s)}{N(s)}=\frac{s^3+12.5s^2+50.5s+66}{s^4+19s^3+122s^2+296s+192}$$

where $G(s)$ = system gain, $C(s)$ = system output, $N(s)$ = system input, and s = Laplace transform complex frequency. Use a numerical technique to find the roots of the numerator and denominator and factor these into the form

$$G(s)=\frac{(s+a_1)(s+a_2)(s+a_3)}{(s+b_1)(s+b_2)(s+b_3)(s+b_4)}$$

where a_i and b_i = the roots of the numerator and denominator respectively.

Solve the problem using the function *roots* in either *Matlab* or *Scilab*.

```
-->s = poly(0,'s');

-->C = s^3 + 12.5*s^2 + 50.5*s + 66;

-->N = s^4 + 19*s^3 + 122*s^2 + 296*s + 192;

-->roots(C)
 ans  =

  - 5.5
  - 4.
  - 3.

-->roots(N)
 ans  =

  - 8.
  - 6.
  - 4.
  - 1.

-->polfact(C)
 ans  =

    1     5.5 + s     4 + s     3 + s

-->polfact(N)
 ans  =

    1     8 + s     6 + s     4 + s     1 + s
```

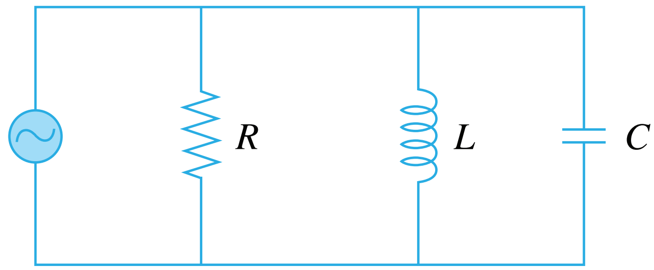
Chapter 8 – Problem 8.32

Figure P8.32 shows a circuit with a resistor, an inductor, and a capacitor in parallel. Kirchhoff's rules can be used to express the impedance of the system as

$$\frac{1}{Z} = \sqrt{\frac{1}{R^2} + \left(\omega C - \frac{1}{\omega L}\right)^2}$$

where Z = impedance (Ω) and ω = the angular frequency. Find the ω that results in an impedance of 75 (Ω) using the following parameters: $R = 225$ Ω, $C = 0.6 \times 10^{-6}$ F, and $L = 0.5$ H.

Figure P8.32



Solve the problem using the function *fzero* in *Matlab* or *fsolve* in *Scilab*.

```
-->Z = 75; R = 225; C = 0.6D-6; L = 0.5;

-->function [g] = f(w)
-->g1 = 1/(R^2); g2 = w*C; g3 = 1/(w*L);
-->g = sqrt(g1+(g2-g3)^2)-(1/Z);
-->endfunction

-->wr1 = fsolve(-23D+3, f)
 wr1  =

  - 21109.221

-->wr2 = fsolve(-160, f)
 wr2  =

  - 157.90888

-->wr3 = fsolve(160, f)
 wr3  =

    157.90888

-->wr4 = fsolve(23D+3, f)
 wr4  =

    21109.221
```