Problem 1 – Linear Regression

Using your preferred high-level language (VBA, C++, etc.) write a program to perform least square linear regression. Your program should contain a main program that performs the following tasks:

- Reads a vector of data *x* of *n* elements.
- Reads a vector of data *y* of *m* elements.
- Checks if *m* = *n*.
  - If so,
    - Call subroutine *Regress* (Fig. 17.6 – Pg. 463 | Chapra & Canale).
    - Print the following results: *n*, *a1*, *a0*, *syx*, *r2*, *r*.
    - Call subroutine *FitData* (calculates *yf* for a specified data fitting type).
    - Print the values of *x*, *y*, and *yf*.
  - If *m* and *n* are not equal, indicate that no regression is possible.

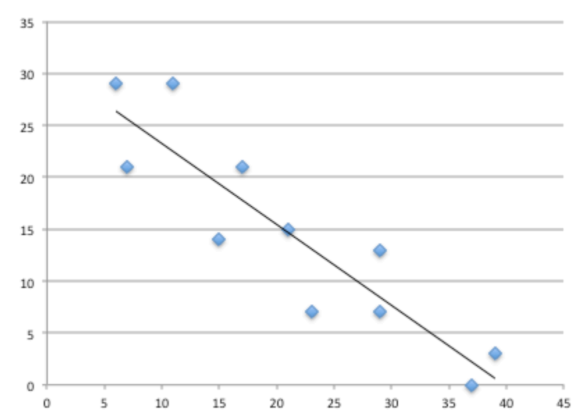
(a) Solve problem 17.4 (Pg. 485 | Chapra & Canale). Your solution should include:

- Table of fitted values *yf*.
- Plot of the original data as points and fitted data as a continuous line.
- Code for the program used.

(b) Solve problem 17.7 parts (a) and (b) (Pg. 485 | Chapra & Canale). Your solution should include:

- The fitted equation.
- The correlation coefficient *r*.

```
// 03/05/2014 - ENGR 2450 - Meine, Joel
// Problems 17.4, 17.7

// Linear Regression

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

void Regress(double x[],double y[],int n,double&
a1,double& a0,double& syx,double& r2,double& r)
{
	double sumx = 0; double sumxy = 0; double st = 0;
	double sumy = 0; double sumx2 = 0; double sr = 0;
	for (int i = 0; i < n; i++)
	{
		sumx = sumx + x[i];
		sumy = sumy + y[i];
		sumxy = sumxy + x[i]*y[i];
		sumx2 = sumx2 + x[i]*x[i];
	}
	double xm = sumx/n;
	double ym = sumy/n;
```



```
        a1 = (n*sumxy - sumx*sumy)/(n*sumx2 - sumx*sumx);
        a0 = ym - a1*xm;
        for (int i = 0; i < n; i++)
        {
                st = st + pow(y[i] - ym,2);
                sr = sr + pow(y[i] - a1*x[i] - a0,2);
        }
        syx = sqrt(sr/(n - 2));
        r2 = (st - sr)/st;
        r = sqrt(r2);
}

void FitData(double x[],int n,double a1,double a0,double yf[],int
F)
{
        double alpha = 0; double beta = 0;
        for (int i = 0; i < n; i++)
        {
                if (F == 1) // Linear Equation
                {
                        yf[i] = a0 + a1*x[i];
                }
                else if (F == 2) // Saturation-Growth-Rate Equation
                {
                        alpha = 1/a0; beta = alpha*a1;
                        yf[i] = alpha*(x[i]/(beta + x[i]));
                }
                else if (F == 3) // Power Equation
                {
                        alpha = pow(10,a0); beta = a1;
                        yf[i] = alpha*pow(x[i],beta);
                }
        }
}

int main()
{
        // Problem 17.4 - Least-Squares Regression
        const int n = 11;
        const int m = 11;
        double x1[n] = {6,7,11,15,17,21,23,29,29,37,39};
        double y1[m] = {29,21,29,14,21,15,7,7,13,0,3};
        double yf1[n];

        // Problem 17.7 - Data Fitting Types
        const int k = 7;
        const int j = 7;
        double x2[n] = {0.75,2,3,4,6,8,8.5};
        double y2[m] = {1.2,1.95,2,2.4,2.4,2.7,2.6};

        double yfA[n];
        double yfB[n];

        if (n == m || k == j)
        {
                // Problem 17.4 - Linear Equation
                double a1 = 0; // Slope
```

```
Chapter 17 - Problem 17.4
==========================================
Number of Data Points, n = 11
Slope, a1 = -0.7805
Intercept, a0 = 31.06
Standard Error of Estimate, syx = 4.476
Coefficient of Determination, r2 = 0.8127
Correlation Coefficient, r = 0.9015
******************************************
Linear Equation
------------------------------------------
 x   y    yf
------------------------------------------
  6  29   26.376
  7  21   25.595
 11  29   22.473
 15  14   19.351
 17  21    17.79
 21  15   14.667
 23   7   13.106
 29   7    8.423
 29  13    8.423
 37   0   2.1787
 39   3  0.61758
++++++++++++++++++++++++++++++++++++++++++

Chapter 17 - Problem 17.7
==========================================
Number of Data Points, n = 7
Slope, a1 = 0.1548
Intercept, a0 = 1.465
Standard Error of Estimate, syx = 0.2504
Coefficient of Determination, r2 = 0.8028
Correlation Coefficient, r = 0.896
******************************************
Saturation-Growth-Rate Equation
------------------------------------------
 x     y    yf
------------------------------------------
0.75   1.2  0.59818
   2  1.95   0.6482
   3     2  0.65923
   4   2.4  0.66488
   6   2.4  0.67063
   8   2.7  0.67355
 8.5   2.6  0.67407
******************************************
Power Equation
------------------------------------------
 x     y     yf
------------------------------------------
0.75   1.2   27.924
   2  1.95   32.503
   3     2   34.609
   4   2.4   36.185
   6   2.4   38.529
   8   2.7   40.284
 8.5   2.6   40.664
++++++++++++++++++++++++++++++++++++++++++

Press any key to continue . . .
```

```cpp
double a0 = 0; // Intercept
double syx = 0; // Standard Error of the Estimate
double r2 = 0; // Coefficient of Determination
double r = 0; // Correlation Coefficient

Regress(x1,y1,n,a1,a0,syx,r2,r);

std::cout << "Chapter 17 - Problem 17.4" << std::endl;
std::cout << "============================================" << std::endl;
std::cout << "Number of Data Points, n = " << n << std::endl;
std::cout << "Slope, a1 = " << setprecision(4) << a1 << std::endl;
std::cout << "Intercept, a0 = " << setprecision(4) << a0 << std::endl;
std::cout << "Standard Error of Estimate, syx = " << setprecision(4) << syx << std::endl;
std::cout << "Coefficient of Determination, r2 = " << setprecision(4) << r2 << std::endl;
std::cout << "Correlation Coefficient, r = " << setprecision(4) << r << std::endl;
std::cout << "********************************************" << std::endl;

FitData(x1,n,a1,a0,yf1,1); // Linear Equation

std::cout << "Linear Equation" << std::endl;
std::cout << "--------------------------------------------" << std::endl;
std::cout << " x   y    yf" << std::endl;
std::cout << "--------------------------------------------" << std::endl;
for (int i = 0; i < n; i++)
{
       cout << setw(3) << x1[i];
       cout << setw(4) << y1[i];
       cout << setw(9) << setprecision(5) << yf1[i] << endl;
}
std::cout << "++++++++++++++++++++++++++++++++++++++++++++" << std::endl;
cout << "\n";

// Problem 17.7 - Saturation-Growth-Rate Equation
a1 = 0; // Slope
a0 = 0; // Intercept
syx = 0; // Standard Error of the Estimate
r2 = 0; // Coefficient of Determination
r = 0; // Correlation Coefficient

Regress(x2,y2,k,a1,a0,syx,r2,r);

std::cout << "Chapter 17 - Problem 17.7" << std::endl;
std::cout << "============================================" << std::endl;
std::cout << "Number of Data Points, n = " << k << std::endl;
std::cout << "Slope, a1 = " << setprecision(4) << a1 << std::endl;
std::cout << "Intercept, a0 = " << setprecision(4) << a0 << std::endl;
std::cout << "Standard Error of Estimate, syx = " << setprecision(4) << syx << std::endl;
std::cout << "Coefficient of Determination, r2 = " << setprecision(4) << r2 << std::endl;
std::cout << "Correlation Coefficient, r = " << setprecision(4) << r << std::endl;
std::cout << "********************************************" << std::endl;

FitData(x2,k,a1,a0,yfA,2); // Saturation-Growth-Rate Equation

std::cout << "Saturation-Growth-Rate Equation" << std::endl;
std::cout << "--------------------------------------------" << std::endl;
std::cout << " x     y    yf" << std::endl;
std::cout << "--------------------------------------------" << std::endl;
```

```
            for (int i = 0; i < k; i++)
            {
                   cout << setw(4) << x2[i];
                   cout << setw(6) << y2[i];
                   cout << setw(9) << setprecision(5) << yfA[i] << endl;
            }
            std::cout << "*********************************************" << std::endl;

            FitData(x2,k,a1,a0,yfB,3); // Power Equation

            std::cout << "Power Equation" << std::endl;
            std::cout << "---------------------------------------------" << std::endl;
            std::cout << " x     y     yf" << std::endl;
            std::cout << "---------------------------------------------" << std::endl;
            for (int i = 0; i < k; i++)
            {
                   cout << setw(4) << x2[i];
                   cout << setw(6) << y2[i];
                   cout << setw(9) << setprecision(5) << yfB[i] << endl;
            }
            std::cout << "+++++++++++++++++++++++++++++++++++++++++++++" << std::endl;
     }
     else std::cout << "Regression is Not Possible" << std::endl;

     cout << "\n";
     system("pause");
     return 0;
}
```

Problem 2 – Polynomial Regression

Using your preferred high-level language (VBA, C++, etc.) write a program to perform polynomial regression. Your program should contain a main program that performs the following tasks:

- Read a vector *x* of size *n*.
- Read a vector *y* of size *n*.
- Read the order *m* of a polynomial for a polynomial fitting.
- Call a new subroutine *BuildZP* that creates a matrix [*Z*] so that the first column is full of 1's, the second corresponds to vector *x*, the third to x^2, and so on.
- Call subroutine *NLRegress* to calculate the vector of coefficients $\{a\}$.
- Calculate the fitted values $yf = [Z]*\{a\}$ using the subroutine *MultiplyMatrixToVector*(*Z*, *a*, *yf*).
- Show the table of values of *x*, *y*, and *yf*.
- Show the values of vector $\{a\}$.

(a) Solve problem 17.20 (Pg. 486 | Chapra & Canale). Your solution should include:

- Table of *x*, *y*, and *yf* data.
- Plot of the original data (*x*,*y*) as points and polynomial fittings as continuous lines.
- Polynomial used for fitting; e.g. $y = a0 + a1*x + a2*x^2$

(b) Solve problem 20.22 parts (a) and (b) (Pg. 573 | Chapra & Canale). Your solution should include:

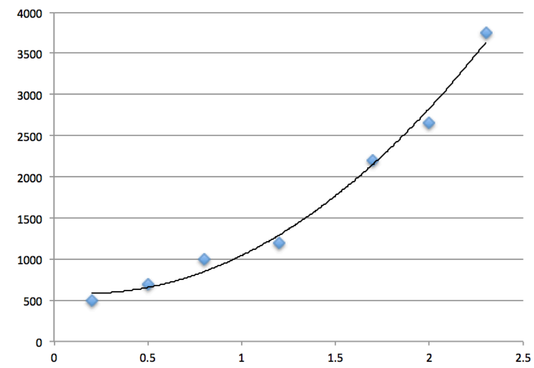
- Table of *T*, *DO*, and *DOf* data.
- Plot of the original data (*T*,*DO*) as points and polynomial fittings as continuous lines.
- Polynomial used for fitting; e.g. $DO = a0 + a1*T + a2*T^2 + a3*T^3$

(c) Show the code for the main program and the subroutines developed in this problem only. You don't need to show the subroutines for matrix operations provided to you.

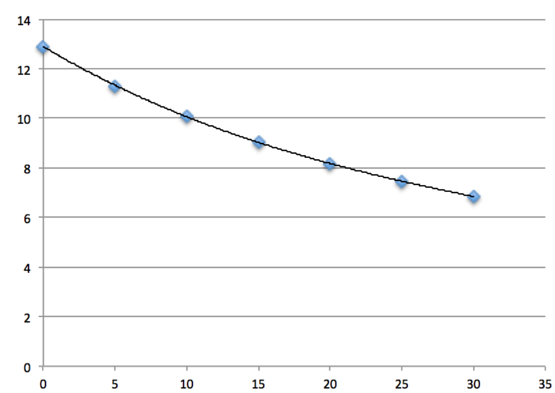
```
// 03/05/2014 - ENGR 2450 - Meine, Joel
// Problems 17.20, 20.22

// Polynomial Regression

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

const int nn = 20;
const int N = 7; // Number of Data Points
const int M2 = 2; // Polynomial, Second Order
const int M3 = 3; // Polynomial, Third Order
const int M2p1 = M2 + 1;  // Polynomial, Second Order plus One
const int M3p1 = M3 + 1;  // Polynomial, Third Order plus One

void NLRegress(double Z[][nn],double Y[],double A[],int n,int
m,int p,double tol,int er)
{
       if (p == 2) // Polynomial, Second Order
       {
              double ZT[M2p1][nn]; double ZTZ[M2p1][nn]; double
ZTZI[M2p1][nn]; double ZTY[M2p1];
              mtranspose(Z,ZT,n,(m+1));
              multiply_matrices(ZT,Z,ZTZ,(m+1),(m+1),n);
              MatrixInverse(ZTZ,ZTZI,(m+1),tol,er);
              multiply_matrix_to_vector(ZT,Y,ZTY,(m+1),n);
              multiply_matrix_to_vector(ZTZI,ZTY,A,(m+1),(m+1));
       }
       else if (p == 3) // Polynomial, Third Order
       {
              double ZT[M3p1][nn]; double ZTZ[M3p1][nn]; double
ZTZI[M3p1][nn]; double ZTY[M3p1];
              mtranspose(Z,ZT,n,(m+1));
              multiply_matrices(ZT,Z,ZTZ,(m+1),(m+1),n);
              MatrixInverse(ZTZ,ZTZI,(m+1),tol,er);
              multiply_matrix_to_vector(ZT,Y,ZTY,(m+1),n);
              multiply_matrix_to_vector(ZTZI,ZTY,A,(m+1),(m+1));
       }
}

void BuildZP(double x[],int n,int m,double Z[][nn])
{
       for (int i = 0; i < n; i++)
```

```
Chapter 17 - Problem 17.20
============================================
Data Fitting: Polynomial, Second Order

yf = a0 + a1*x + a2*x^2

a0 = 604.094
a1 = -233.961
a2 = 674.007
********************************************
 x   y      yf
--------------------------------------------
0.2   500   584.26
0.5   700   655.62
0.8  1000   848.29
1.2  1200   1293.9
1.7  2200   2154.2
  2  2650   2832.2
2.3  3750   3631.5
++++++++++++++++++++++++++++++++++++++++++++

Chapter 20 - Problem 20.22
============================================
Data Fitting: Polynomial, Third Order

DOf = a0 + a1*T + a2*T^2 + a3*T^3

a0 = 12.8879
a1 = -0.341111
a2 = 0.00652381
a3 = -6.22222e-005

Dissolved Oxygen, Fitted (mg/L), DOf
Dissolved Oxygen, Recorded (mg/L), DO
Temperature (C), T
Concentration of Chloride (g/L), c = 10

DOf(T=8) = 10.5446
********************************************
 T   DO      DOf
--------------------------------------------
 0   12.9    12.888
 5   11.3    11.338
10   10.1    10.067
15   9.03     9.029
20   8.17    8.1774
25   7.46    7.4652
30   6.85     6.846
++++++++++++++++++++++++++++++++++++++++++++

Press any key to continue . . .
```

```
        {
                for (int j = 0; j < (m+1); j++)
                        Z[i][j] = pow(x[i],j);
        }
}
```




```
int main()
{
        // Problem 17.20 - Polynomial, Second Order
        double x[N] = {0.2,0.5,0.8,1.2,1.7,2,2.3};
        double y[N] = {500,700,1000,1200,2200,2650,3750};

        double Z2[N][nn];
        BuildZP(x,N,M2,Z2);

        double a2[M2p1];
        NLRegress(Z2,y,a2,N,M2,2,0.0000000001,0);

        double yf[N];
        multiply_matrix_to_vector(Z2,a2,yf,N,M2p1);

        std::cout << "Chapter 17 - Problem 17.20" << std::endl;
        std::cout << "============================================" << std::endl;
        std::cout << "Data Fitting: Polynomial, Second Order" << std::endl;
        cout << "\n";
        std::cout << "yf = a0 + a1*x + a2*x^2" << std::endl;
        cout << "\n";
        std::cout << "a0 = " << setprecision(6) << a2[0] << std::endl;
        std::cout << "a1 = " << setprecision(6) << a2[1] << std::endl;
        std::cout << "a2 = " << setprecision(6) << a2[2] << std::endl;
        std::cout << "********************************************" << std::endl;
        std::cout << " x    y       yf" << std::endl;
        std::cout << "--------------------------------------------" << std::endl;
        for (int i = 0; i < N; i++)
        {
                cout << setw(3) << x[i];
                cout << setw(6) << y[i];
                cout << setw(9) << setprecision(5) << yf[i] << endl;
        }
        std::cout << "++++++++++++++++++++++++++++++++++++++++++++" << std::endl;
        cout << "\n";
```




```
        // Problem 20.22 - Polynomial, Third Order
        double T[N] = {0,5,10,15,20,25,30}; //
Temperature (C)
        double DO[N] =
{12.9,11.3,10.1,9.03,8.17,7.46,6.85}; // Dissolved
Oxygen (mg/L)

        double Z3[N][nn];
        BuildZP(T,N,M3p1,Z3);

        double a3[M3p1];
        NLRegress(Z3,DO,a3,N,M3,3,0.0000000001,0);

        double DOf[N];
        multiply_matrix_to_vector(Z3,a3,DOf,N,M3p1);
```

```
	int t = 8; // Temperature = 8 C
	double DOe = a3[0] +a3[1]*t + a3[2]*pow(t,2) + a3[3]*pow(t,3); // DO(T)

	std::cout << "Chapter 20 - Problem 20.22" << std::endl;
	std::cout << "=============================================" << std::endl;
	std::cout << "Data Fitting: Polynomial, Third Order" << std::endl;
	cout << "\n";
	std::cout << "DOf = a0 + a1*T + a2*T^2 + a3*T^3" << std::endl;
	cout << "\n";
	std::cout << "a0 = " << setprecision(6) << a3[0] << std::endl;
	std::cout << "a1 = " << setprecision(6) << a3[1] << std::endl;
	std::cout << "a2 = " << setprecision(6) << a3[2] << std::endl;
	std::cout << "a3 = " << setprecision(6) << a3[3] << std::endl;
	cout << "\n";
	std::cout << "Dissolved Oxygen, Fitted (mg/L), DOf" << std::endl;
	std::cout << "Dissolved Oxygen, Recorded (mg/L), DO" << std::endl;
	std::cout << "Temperature (C), T" << std::endl;
	std::cout << "Concentration of Chloride (g/L), c = 10" << std::endl;
	cout << "\n";
	std::cout << "DOf(T=" << t << ") = " << DOe << std::endl;
	std::cout << "*********************************************" << std::endl;
	std::cout << " T   DO     DOf" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	for (int i = 0; i < N; i++)
	{
		cout << setw(3) << T[i];
		cout << setw(6) << DO[i];
		cout << setw(9) << setprecision(5) << DOf[i] << endl;
	}
	std::cout << "+++++++++++++++++++++++++++++++++++++++++++++" << std::endl;

	cout << "\n";
	system("pause");
	return 0;
}
```

Problem 3 – Multiple Linear Regression

Using your preferred high-level language (VBA, C++, etc.) write a program to perform multiple linear regression. Your program should contain a main program that performs the following tasks:

- Read a matrix $[X]$ of size n-by-m (n rows and m columns) containing n data points for each of the m explanatory variables $x1, x2, \ldots, xm$.
- Read a vector y of size n.
- Call a new subroutine *BuildZM* that creates a matrix $[Z]$ so that the first column is full of 1's, and the rest of the matrix is the same as matrix $[X]$. Thus matrix $[Z]$ will have n rows and $(m+1)$ columns.
- Call subroutine *NLRegress* to calculate the vector of coefficients $\{a\}$.
- Calculate the fitted values $yf = [Z]*\{a\}$ using the subroutine *MultiplyMatrixToVector*(Z, a, yf).
- Show the table of values of $[X]$, y, and yf.
- Show the values of vector $\{a\}$.

(a) Solve problem 17.18 (Pg. 486 | Chapra & Canale). Your solution should include:

- Table of values *x1*, *x2*, *y*, *yf*, and *a*.

(b) Show the code for the main program and the subroutines developed in this problem only. You don't need to show the subroutines for matrix operations provided to you.

```
// 03/05/2014 - ENGR 2450 - Meine, Joel
// Problem 17.18

// Multiple-Linear Regression

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

const int nn = 20;
const int N = 9; // Number of Data Points
const int M = 2; // Independent Variable Set, M Order
const int Mp1 = M + 1; // Independent Variable Set, M plus One

void NLRegress(double Z[][nn],double Y[],double A[],int n,int
m,double tol,int er)
{
	double ZT[Mp1][nn]; double ZTZ[Mp1][nn]; double
ZTZI[Mp1][nn]; double ZTY[Mp1];
	mtranspose(Z,ZT,n,(m+1));
	multiply_matrices(ZT,Z,ZTZ,(m+1),(m+1),n);
	MatrixInverse(ZTZ,ZTZI,(m+1),tol,er);
	multiply_matrix_to_vector(ZT,Y,ZTY,(m+1),n);
	multiply_matrix_to_vector(ZTZI,ZTY,A,(m+1),(m+1));
}

void BuildZM(double X[][nn],int n,int m,double Z[][nn])
{
	for (int i = 0; i < n; i++)
	{
		Z[i][0] = 1.0;
		for (int j = 1; j < (m+1); j++)
			Z[i][j] = X[i][j-1];
	}
}

int main()
{
	// Problem 17.18 - Multiple Linear Regression
	double x[N][nn] = {{0,0},{1,1},{1,2},{2,1},{2,2},{3,1},{3,2},{4,1},{4,2}};
	double y[N] = {15.1,17.9,12.7,25.6,20.5,35.1,29.7,45.4,40.2};

	double Z[N][nn];
	BuildZM(x,N,M,Z);

	double a[Mp1];
	NLRegress(Z,y,a,N,M,0.0001,0);
```

```
Chapter 17 - Problem 17.18
==========================================
yf = a0 + a1*x1 + a2*x2

Slope for x2, a2 = -5.704
Slope for x1, a1 = 9.025
Intercept, a0 = 14.46
******************************************
 x1  x2   y      yf
------------------------------------------
  0   0   15.1   14.461
  1   1   17.9   17.782
  1   2   12.7   12.077
  2   1   25.6   26.807
  2   2   20.5   21.103
  3   1   35.1   35.832
  3   2   29.7   30.128
  4   1   45.4   44.857
  4   2   40.2   39.153
++++++++++++++++++++++++++++++++++++++++++

Press any key to continue . . .
```

```
	double yf[N];
	multiply_matrix_to_vector(Z,a,yf,N,Mp1);

	std::cout << "Chapter 17 - Problem 17.18" << std::endl;
	std::cout << "============================================" << std::endl;
	std::cout << "yf = a0 + a1*x1 + a2*x2" << std::endl;
	cout << "\n";
	std::cout << "Slope for x2, a2 = " << setprecision(4) << a[2] << std::endl;
	std::cout << "Slope for x1, a1 = " << setprecision(4) << a[1] << std::endl;
	std::cout << "Intercept, a0 = " << setprecision(4) << a[0] << std::endl;
	std::cout << "********************************************" << std::endl;
	std::cout << " x1  x2   y      yf" << std::endl;
	std::cout << "--------------------------------------------" << std::endl;
	for (int i = 0; i < N; i++)
	{
		cout << setw(3) << x[i][0];
		cout << setw(4) << x[i][1];
		cout << setw(7) << y[i];
		cout << setw(9) << setprecision(5) << yf[i] << endl;
	}
	std::cout << "++++++++++++++++++++++++++++++++++++++++++++" << std::endl;

	cout << "\n";
	system("pause");
	return 0;
}
```

Problem 4 – Newton's Polynomial Interpolation

Using your preferred high-level language (VBA, C++, etc.) write a program to perform Newton's polynomial interpolation.

(a) Solve problem 18.6 (Pg. 486 | Chapra & Canale). Show the Newton interpolated values.

(b) Solve problem 20.36 (Pg. 576 | Chapra & Canale). Show the Newton interpolated values.

(c) Show the code for the main program and the subroutines developed in this problem only.

```
// 03/05/2014 - ENGR 2450 - Meine, Joel
// Problems 18.6, 20.36

// Newton's Polynomial Interpolation

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

const int n = 4; // Polynomial Order, max

void NewInt(double x[],double y[],double xi,double yint[],double ea[])
{
```

```
	double yint2 = 0;
	double fdd[n+1][n+1];
	for (int i = 0; i <= n; i++)
		fdd[i][0] = y[i];
	for (int j = 1; j <= n; j++)
	{
		for (int i = 0; i <= n - j; i++)
			fdd[i][j] = (fdd[i+1][j-1] - fdd[i][j-1])/(x[i+j] - x[i]);
	}
	double xterm = 1;
	yint[0] = fdd[0][0];
	for (int order = 1; order <= n; order++)
	{
		xterm = xterm * (xi - x[order-1]);
		yint2 = yint[order-1] + fdd[0][order] * xterm;
		ea[order-1] = yint2 - yint[order-1];
		yint[order] = yint2;
	}
	ea[n] = 0;
}

int main()
{
	// Problem 18.6 - Newton's Polynomial Interpolation, Order
1 to 4
	double x[n+1] = {5,3,7,2,8};
	double y[n+1] = {99,19,291,6,444};
	double xint = 4; // Value of Interest
	double yint[n+1]; // Solution of Interest
	double ea1[n+1];
	NewInt(x,y,xint,yint,ea1);

	std::cout << "Chapter 18 - Problem 18.6" << std::endl;
	std::cout <<
"==============================================" << std::endl;
	std::cout << "Value of Interest, xint = " << xint <<
std::endl;
	std::cout << "Solution of Interest, yint" << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
	std::cout <<
"**********************************************" << std::endl;
	std::cout << " x      y" << std::endl;
	std::cout << "---------------------------------------------
" << std::endl;
	for (int i = 0; i < n + 1; i++)
	{
		cout << setw(2) << x[i];
		cout << setw(6) << y[i] << endl;
	}
	std::cout << "**********************************************" << std::endl;
	std::cout << " yint   ea" << std::endl;
	std::cout << "---------------------------------------------" << std::endl;
	for (int i = 0; i < n + 1; i++)
	{
		cout << setw(5) << setprecision(5) << yint[i];
		cout << setw(5) << setprecision(5) << ea1[i] << endl;
	}
```

```
Chapter 18 - Problem 18.6
=========================================
Value of Interest, xint = 4
Solution of Interest, yint
Actual Error, ea
*****************************************
 x     y
-----------------------------------------
 5    99
 3    19
 7   291
 2     6
 8   444
*****************************************
 yint   ea
-----------------------------------------
   99  -40
   59  -14
   45    3
   48    0
   48    0
+++++++++++++++++++++++++++++++++++++++++

Chapter 20 - Problem 20.36
=========================================
Current (A), I
Voltage (V), V
Value of Interest, Iint = 1.15
Solution of Interest, Vint
Actual Error, ea
*****************************************
 I       V
-----------------------------------------
1.25     0.7
0.75    -0.6
1.5     1.88
0.25   -0.45
 2        6
*****************************************
 Vint       ea
-----------------------------------------
      0.7         -0.26
     0.44     -0.113067
 0.326933  -0.000821333
 0.326112     0.0111744
 0.337286             0
+++++++++++++++++++++++++++++++++++++++++

Press any key to continue . . .
```

```
	std::cout << "++++++++++++++++++++++++++++++++++++++++++++" << std::endl;
	cout << "\n";

	// Problem 20.36 - Newton's Polynomial Interpolation, Order 1 to 4
	double I[n+1] = {1.25,0.75,1.5,0.25,2.0}; // Current (A), I
	double V[n+1] = {0.70,-0.6,1.88,-0.45,6.0}; // Voltage (V), V
	double Iint = 1.15; // Value of Interest
	double Vint[n+1]; // Solution of Interest
	double ea2[n+1];
	NewInt(I,V,Iint,Vint,ea2);

	std::cout << "Chapter 20 - Problem 20.36" << std::endl;
	std::cout << "============================================" << std::endl;
	std::cout << "Current (A), I" << std::endl;
	std::cout << "Voltage (V), V" << std::endl;
	std::cout << "Value of Interest, Iint = " << Iint << std::endl;
	std::cout << "Solution of Interest, Vint" << std::endl;
	std::cout << "Actual Error, ea" << std::endl;
	std::cout << "********************************************" << std::endl;
	std::cout << " I      V" << std::endl;
	std::cout << "--------------------------------------------" << std::endl;
	for (int i = 0; i < n + 1; i++)
	{
		cout << setw(2) << I[i];
		cout << setw(8) << V[i] << endl;
	}
	std::cout << "********************************************" << std::endl;
	std::cout << " Vint       ea" << std::endl;
	std::cout << "--------------------------------------------" << std::endl;
	for (int i = 0; i < n + 1; i++)
	{
		cout << setw(9) << setprecision(6) << Vint[i];
		cout << setw(14) << setprecision(6) << ea2[i] << endl;
	}
	std::cout << "++++++++++++++++++++++++++++++++++++++++++++" << std::endl;

	cout << "\n";
	system("pause");
	return 0;
}
```

Problem 5 – Cubic Splines Interpolation

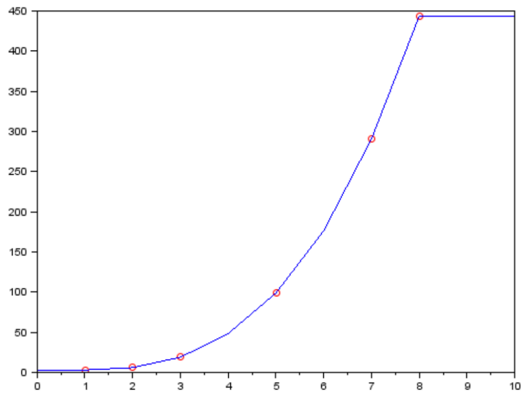
Use SCILAB to interpolate for a specified value using “natural” type cubic splines.

(a) Solve problem 18.6 (Pg. 522 | Chapra & Canale). Plot the original data given as points and the fitted splines as continuous lines.

(b) Solve problem 20.43 (Pg. 576 | Chapra & Canale). Plot the original data given as points and the fitted splines as continuous lines.

Problem 18.6

```
-->x = [1,2,3,5,7,8];

-->y = [3,6,19,99,291,444];

-->d = splin(x,y,"natural");

-->y = interp(4,x,y,d)
 y  =

    48.411572
```



Problem 20.43

```
-->T = [200,250,300,375,425,475,600];

-->e = [7.5,8.6,8.7,10,11.3,12.7,15.3];

-->d = splin(T,e,"natural");

-->e = interp(400,T,e,d)
 e  =

    10.630524
```