Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Численные методы»

Студент:	А. А. Литвина
Преподаватель:	И. Э. Иванов
Группа:	М8О-306Б
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Вариант №13

1 Многочлены Лагранжа и Ньютона

1 Постановка задачи

Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, Y_i\}$. Вычислить значение погрешности интерполяции в точке X^*.

Функция: $y = \cos(x) + x$.

a) $X_i = 0, \frac{\Pi}{6}, \frac{2\Pi}{6}, \frac{3\Pi}{6}$; b) $X_i = 0, \frac{\Pi}{6}, \frac{\Pi}{4}, \frac{\Pi}{2}$; $X^* = 1.0$.

2 Описание

2.1 Многочлен Лагранжа

Функция задана в 4 точках, следовательно, искомым является многочлен Лагранжа 3 степени:

$$L_3(x) = \sum_{i=0}^{3} f_i \frac{\omega_4(x)}{(x - x_i)\omega_4'(x_i)},$$

где $\omega_4(x) = (x - x_0)(x - x_1)(x - x_2)(x - x_3)$,
$\omega_4'(x_0) = (x_0 - x_1)(x_0 - x_2)(x_0 - x_3)$,
$\omega_4'(x_1) = (x_1 - x_0)(x_1 - x_2)(x_1 - x_3)$,
$\omega_4'(x_2) = (x_2 - x_0)(x_2 - x_1)(x_2 - x_3)$,
$\omega_4'(x_3) = (x_3 - x_0)(x_3 - x_1)(x_3 - x_2)$,

Заполним таблицу для случая a):

i	x_i	f_i	$w_4'(x_i)$	$f_i/w_4'(x_i)$
0	0	1	-0.861285	-1.16106
1	П/6	1.38962	0.287095	4.84029
2	П/3	1.5472	-0.287095	-5.38915
3	П/2	1.5708	0.861285	1.82378

Запишем многочлен Лагранжа:

$L_3(x) = -1.16106(x - \Pi/6)(x - \Pi/3)(x - \Pi/2) + 4.84029x(x - \Pi/3)(x - \Pi/2) -$

$-5.38915x(x-\Pi/6)(x-\Pi/2)+1.82378x(x-\Pi/6)(x-\Pi/3)$

Вычислим значение интерполяционного многочлена и точное значение функции в точке $X^*=1.0$:

$L_3(1)=1.53995, \qquad y(1)=\cos(1)+1=1.5403.$

Абсолютная погрешность интерполяции составляет: $\Delta(L_3(1))=0.000353069$.

2.2 Многочлен Ньютона

Функция задана в четырех точках, следовательно, искомым является многочлен Ньютона третьей степени:

$P_3(x)=f(x_0)+(x-x_0)f(x_1,x_0)+(x-x_0)(x-x_1)f(x_0,x_1,x_2)+$
$+(x-x_0)(x-x_1)(x-x_2)f(x_0,x_1,x_2,x_3)$, где

$$f(x_i,x_j)=\frac{f_i-f_j}{x_i-x_j},$$
$$f(x_i,x_j,x_k)=\frac{f(x_i,x_j)-f(x_j,x_k)}{x_i-x_k}.$$

Заполним таблицу конечных разностей для случая а), где на главной диагонали будут стоять значения f_i, а остальные значения будут рассчитываться по формуле: $A[i][j]=f(A[i+1][j],A[i][j-1])$.

	0	1	2	3
0	1	0.744127	-0.42321	0.113872
1	-	1.38962	0.300943	-0.24434
2	-	-	1.5472	0.0450703
3	-	-	-	1.5708

Коэффициентами многочлена Ньютона будут значения в верхней строке таблицы. Запишем искомый многочлен:

$P_3(x)=1x+0.744127x-0.42321x(x-\Pi/6)+0.113872x(x-\Pi/6)(x-\Pi/3)$

Вычислим значение интерполяционного многочлена в точке $X^*=1.0$:

$P_3(1)=1.53995, \qquad y(1)=\cos(1)+1=1.5403.$

Абсолютная погрешность интерполяции составляет: $\Delta(P_3(1)) = 0.000353069$.

3 Исходный код

```

#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
#include <vector>

using namespace std;

double f(double x) {
  return cos(x)+x;
}

void lag(vector <double> x, vector <int> k, double X) {
  double w,L,P,f_i;
  L=0;
  cout << "L_3(x)=";
  for (int i=0; i<x.size(); i++) {
    w=1;
    for (int j=0; j<x.size(); j++) {
      if (i==j)
        continue;
      w=w*(x[i]-x[j]);
    }
    f_i=f(x[i]);

    if ((i!=0)&&(f_i/w>0))
      cout << "+";
    cout << f_i/w;

    P=f_i/w;
    for (int j=0; j<x.size(); j++) {
      if (i==j)
        continue;

      if (j!=0)
        cout << "(x-P/" << k[j] << ")";
      else
        cout << "x";

      P=P*(X-x[j]);
    }
    L=L+P;
  }
  cout << endl;
  cout << "L_3(" << X << ")=" << L << endl;
  cout << "cos(" << X << ")+" << X << "=" << f(X) << endl;
  cout << "delta=" << abs(f(X)-L) << endl;
```

```
}

void newt(vector <double> x, vector <int> k, double X) {
  int N = x.size();
  vector <vector <double>> mat (N, vector <double> (N,0));
  int i, j, m;
  double P, S;

  for (i=0; i<N; i++) {
    mat[i][i]=f(x[i]);
  }

  i=0;
  j=1;
  m=1;

  while (j<N) {
    while (j<N) {
      mat[i][j]=(mat[i+1][j]-mat[i][j-1])/(x[j]-x[i]);
      i++;
      j=i+m;
    }
    m++;
    i=0;
    j=m;
  }

  cout << "P_3(x)=";
  P=0;

  for (i=0; i<N; i++) {
    if ((mat[0][i]>0)&&(i!=0))
      cout << "+";
    cout << mat[0][i] << "x";
    S=mat[0][i]*X;

    for (j=1; j<i; j++) {
      cout << "(x-P/" << k[j] << ")";
      S=S*(X-x[j]);
    }
    P=P+S;
  }

  cout << endl;
  cout << "P_3(" << X << ")=" << P << endl;
  cout << "cos(" << X << ")+" << X << "=" << f(X) << endl;
  cout << "delta=" << abs(f(X)-P) << endl;
}

```

```
int main() {
  vector <double> x1 = {0, M_PI/6, M_PI/3, M_PI/2};
  vector <int> k1 = {0, 6, 3, 2};
  vector <double> x2 = {0, M_PI/6, M_PI/4, M_PI/2};
  vector <int> k2 = {0, 6, 4, 2};
  double X=1;

  cout << "LAGRANGE:\na)\n";
  lag(x1,k1,X);
  cout << "\nb)\n";
  lag(x2,k2,X);

  cout << "\nNEWTON:\na)\n";
  newt(x1, k1, X);
  cout << "\nb)\n";
  newt(x2, k2, X);
}
```

4 Консоль

```
LAGRANGE:
a)
L_3(x)=-1.16106(x-P/6)(x-P/3)(x-P/2)+4.84029x(x-P/3)(x-P/2)-
-5.38915x(x-P/6)(x-P/2)+1.82378x(x-P/6)(x-P/3)
L_3(1)=1.53995
cos(1)+1=1.5403
delta=0.000353069

b)
L_3(x)=-1.54807(x-P/6)(x-P/4)(x-P/2)+9.68058x(x-P/4)(x-P/2)-
-9.24203x(x-P/6)(x-P/2)+1.21585x(x-P/6)(x-P/4)
L_3(1)=1.542
cos(1)+1=1.5403
delta=0.00169701

NEWTON:
a)
P_3(x)=1x+0.744127x-0.42321x(x-P/6)+0.113872x(x-P/6)(x-P/3)
P_3(1)=1.53995
cos(1)+1=1.5403
delta=0.000353069

b)
P_3(x)=1x+0.744127x-0.4471x(x-P/6)+0.106333x(x-P/6)(x-P/4)
P_3(1)=1.542
cos(1)+1=1.5403
delta=0.00169701
```

2 Сплайн-интерполяция

1 Постановка задачи

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

$X^* = 1.5$

i	0	1	2	3	4
x_i	0	1	2	3	4
f_i	1	1.5403	1.5839	2.01	3.3464

2 Описание

Обозначим $h_i = x_i - x_{i-1}$ и составим систему из n-1 уравнений с трехдиагональной матрицей:

$$2(h_1 + h_2)c_2 + h_2 c_3 = 3(\frac{f_2 - f_1}{h_2} - \frac{f_1 - f_0}{h_1})$$

$$h_{i-1}c_{i-1} + 2(h_{i-1} + h_i)c_i + h_i c_{i+1} = 3(\frac{f_i - f_{i-1}}{h_i} - \frac{f_{i-1} - f_{i-2}}{h_{i-1}}), \quad i = 3, .., n-1$$

$$h_{n-1}c_{n-1} + 2(h_{n-1} + h_n)c_n = 3(\frac{f_n - f_{n-1}}{h_n} - \frac{f_{n-1} - f_{n-2}}{h_{n-1}}).$$

Для данной задачи:

$4c_2 + c_3 = -1.4901$
$c_2 + 4c_3 + c_4 = 1.1475$
$c_3 + 4c_4 = 2.7309$

Решаем данную систему, находим C_2, C_3, C_4; C_1 положим равным 0. Найдем остальные коэффициенты A,B и D для каждого C:

$$A_i = f_{i-1}, \quad i = 1, .., n$$

$$B_i = (f_i - f_{i-1})/h_i - \frac{1}{3}h_i(C_{i+1} + 2C_i),$$

$$D_i = \frac{C_{i+1} - C_i}{3h_i}, \quad i = 1, .., n-1$$

$$B_n = (f_n - f_{n-1})/h_n - \frac{2}{3}h_n C_n$$

$$D_n = -\frac{C_n}{3h_n}$$

Запишем функцию через кубический сплайн:

$$f(x) = A_i + B_i(x - x_{i-1}) + C_i(x - x_{i-1})^2 + D_i(x - x_{i-1})^3, \quad x_{i-1} \le x \le x_i$$

Далее определим, в каком промежутке $[x_{i-1}, x_i]$ лежит X^*. В нашем случае это промежуток [1,2] для i=2.

Запишем функцию на этом промежутке:

$$f(x) = 1.5403 + 0.252079(x - 1) - 0.432332(x - 1)^2 + 0.223854(x - 1)^3, \quad 1 \le x \le 2,$$

и найдем значение $f(X^*) = f(1.5) = 1.58624$.

3 Исходный код

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

int main() {
  vector <double> x = {0, 1, 2, 3, 4};
  vector <double> f = {1, 1.5403, 1.5839, 2.01, 3.3464};
  int N=4;
  vector <double> h (N+1,0);
  vector <vector <double >> c (N-1, vector <double> (N-1,0));
  vector <vector <double>> L (N-1, vector <double> (N-1,0));
  vector <vector <double>> U (N-1, vector <double> (N-1,0));
  vector <double> b (N-1,0);
  vector <double> C (N+1,0);
  vector <double> y (N-1,0);
  double X = 1.5;
  vector <double> A (N+1,0);
  vector <double> B (N+1,0);
  vector <double> D (N+1,0);

  for (int i=1; i<N+1; i++) {
    h[i]=x[i]-x[i-1];
  }

  c[0][0]=2*(h[1]+h[2]);
  c[0][1]=h[2];

  c[1][0]=h[2];
  c[1][1]=2*(h[2]+h[3]);
  c[1][2]=h[4];

  c[2][1]=h[3];
  c[2][2]=2*(h[3]+h[4]);

  b[0]=3*((f[2]-f[1])/h[2]-(f[1]-f[0])/h[1]);
  b[1]=3*((f[3]-f[2])/h[3]-(f[2]-f[1])/h[2]);
  b[2]=3*((f[4]-f[3])/h[4]-(f[3]-f[2])/h[3]);

  int K=N-1;

  U=c;

  for (int k=0; k<K-1; k++) {

    for (int i=k; i<K; i++) {
```

```
      for (int j=i; j<K; j++) {
        L[j][i]=U[j][i]/U[i][i];
      }
    }

    for (int i=k+1; i<K; i++) {
      for (int j=k; j<K; j++) {
        U[i][j]=U[i][j]-L[i][k]*U[k][j];
      }
    }
  }

  for (int i=0; i<K; i++) {
    double S=0;
    for (int j=0; j<i; j++) {
        S+=L[i][j]*y[j];
    }
    y[i]=b[i]-S;
  }

  for (int i=K-1; i>=0; i--) {
    double S=0;
    for (int j=K-1; j>i; j--) {
        S+=U[i][j]*C[j];
    }
    C[i]=(y[i]-S)/U[i][i];
  }

  for (int i=K-1; i>=0; i--) {
    C[i+2]=C[i];
  }
  C[0]=0;
  C[1]=0;

  int r=ceil(X);

  for (int i=1; i<N+1; i++) {
    A[i]=f[i-1];
    if (i==N) {
      B[i]=(f[i]-f[i-1])/h[i]-2*h[i]*C[i]/3;
      D[i]=-C[i]/(3*h[i]);
    }
    else {
      B[i]=(f[i]-f[i-1])/h[i]-1*h[i]*(C[i+1]+2*C[i])/3;
      D[i]=(C[i+1]-C[i])/(3*h[i]);
    }
  }

  for (int i=1; i<N+1; i++) {
```

```
    cout << "f(x)=" << A[i];
    if (B[i]>=0)
      cout << "+";
    cout << B[i] << "(x-" << x[i-1] << ")";
    if (C[i]>=0)
      cout << "+";
    cout << C[i] << "(x-" << x[i-1] << ")^2";
    if (D[i]>=0)
      cout << "+";
    cout << D[i] << "(x-" << x[i-1] << ")^3, ";
    cout << x[i-1] << "<x<" << x[i] << "\n\n";
  }

  cout << "f(" << X << ")=" << A[r]+B[r]*(X-x[r-1])+C[r]*pow((X-x[r-1]),2)+D[r]*pow((X-x[r-1]),3) << endl;
}
```

4 Консоль

```
f(x)=1+0.684411(x-0)+0(x-0)^2-0.144111(x-0)^3, 0<x<1

f(x)=1.5403+0.252079(x-1)-0.432332(x-1)^2+0.223854(x-1)^3, 1<x<2

f(x)=1.5839+0.058975(x-2)+0.239229(x-2)^2+0.127896(x-2)^3, 2<x<3

f(x)=2.01+0.921121(x-3)+0.622918(x-3)^2-0.207639(x-3)^3, 3<x<4

f(1.5)=1.58624
```

3 Метод наименьших квадратов

1 Постановка задачи

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

i	0	1	2	3	4	5
x_i	-1	0	1	2	3	4
y_i	-0.4597	1	1.5403	1.5839	2.01	3.3464

2 Описание

Найдем приближающий многочлен первой степени: $F_1(x) = a_0+a_1x$. Для нахождения a_0 и a_1 запишем нормальную систему МНК:

$$a_0(N+1) + a_1\sum_{j=0}^{N} x_j = \sum_{j=0}^{N} y_j$$

$$a_0\sum_{j=0}^{N} x_j + a_1\sum_{j=0}^{N} x_j^2 = \sum_{j=0}^{N} x_j y_j$$

Для данной задачи:

$6a_0 + 9a_1 = 9.0209$
$9a_0 + 31a_1 = 24.5834$

Решив систему, находим a_0 и a_1. Запишем приближающий многочлен 1-ой степени: $F_1(x) = 0.556165 + 0.631546x$. Сумма квадратов ошибок вычисляется по формуле:

$$\Phi = \sum_{j=0}^{5}(F_1(x_j) - y_j)^2 = 0.788433$$

Найдем приближающий многочлен второй степени: $F_2(x) = a_0 + a_1x + a_2x^2$. Для нахождения a_0, a_1 и a_2 запишем нормальную систему МНК:

$$a_0(N+1) + a_1\sum_{j=0}^{N} x_j + a_2\sum_{j=0}^{N} x_j^2 = \sum_{j=0}^{N} y_j$$

$$a_0 \sum_{j=0}^{N} x_j + a_1 \sum_{j=0}^{N} x_j^2 + a_2 \sum_{j=0}^{N} x_j^3 = \sum_{j=0}^{N} x_j y_j$$

$$a_0 \sum_{j=0}^{N} x_j^2 + a_1 \sum_{j=0}^{N} x_j^3 + a_2 \sum_{j=0}^{N} x_j^4 = \sum_{j=0}^{N} x_j^2 y_j$$

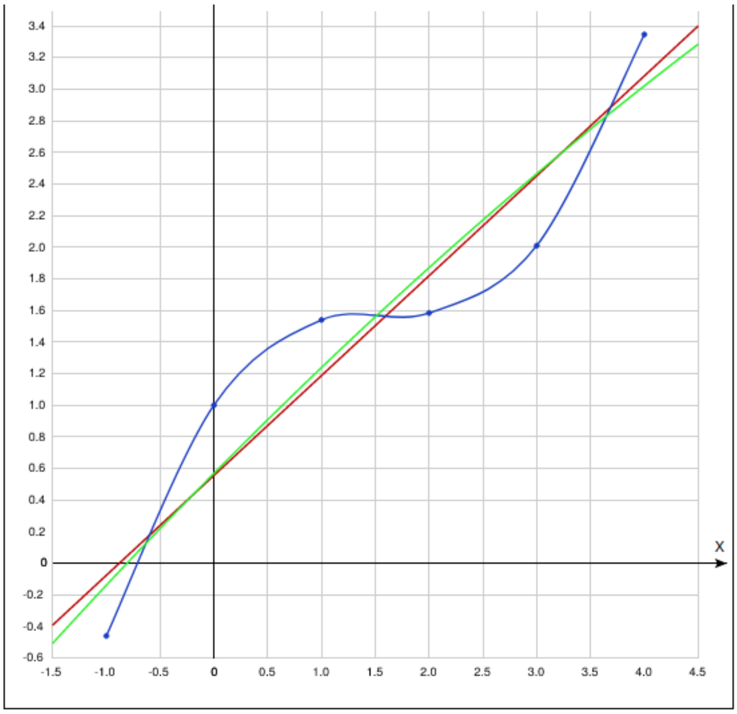
Для данной задачи получим:

$6a_0 + 9a_1 + 31a_2 = 9.0209$
$9a_0 + 31a_1 + 99a_2 = 24.5834$
$31a_0 + 99a_1 + 355a_2 = 79.0486$

Решив систему, находим a_0, a_1 и a_2. Запишем приближающий многочлен 2-ой степени: $F_2(x) = 0.568942 + 0.689044x - 0.0191661x^2$. Сумма квадратов ошибок вычисляется по формуле:

$$\Phi = \sum_{j=0}^{5} (F_2(x_j) - y_j)^2 = 0.774719$$

Построим графики приближающей функции и приближающих многочленов:



$y(x) = 0.556 + 0.632x$ Показать таблицу точек
$y(x) = 0.569 + 0.689x - 0.019x^2$ Показать таблицу точек

Синим цветом обозначена функция, построенная по точкам, красным - многочлен 1-ой степени, зеленым - многочлен 2-ой степени.

3 Исходный код

3.1 Многочлен 1-ой степени

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

int main() {

  int K=2;
  int N=6;
  double S_x=0;
  double S_x2=0;
  double S_y=0;
  double S_xy=0;
  double Q=0;
  vector <double> x = {-1, 0, 1, 2, 3, 4};
  vector <double> y = {-0.4597, 1, 1.5403, 1.5839, 2.010, 3.3464};
  vector <double> f (N,0);
  vector <vector <double>> A (K, vector <double> (K,0));
  vector <vector <double>> L (K, vector <double> (K,0));
  vector <vector <double>> U (K, vector <double> (K,0));
  vector <double> b (K,0);
  vector <double> z (K,0);
  vector <double> a (K,0);


  for (int i=0; i<N; i++) {
    S_x+=x[i];
    S_x2+=pow(x[i],2);
    S_y+=y[i];
    S_xy+=x[i]*y[i];
  }

  A[0][0]=N;
  A[0][1]=S_x;
  A[1][0]=S_x;
  A[1][1]=S_x2;

  b[0]=S_y;
  b[1]=S_xy;

  U=A;

  for (int k=0; k<K-1; k++) {

```

```
    for (int i=k; i<K; i++) {
      for (int j=i; j<K; j++) {
        L[j][i]=U[j][i]/U[i][i];
      }
    }

    for (int i=k+1; i<K; i++) {
      for (int j=k; j<K; j++) {
        U[i][j]=U[i][j]-L[i][k]*U[k][j];
      }
    }
  }

  for (int i=0; i<K; i++) {
    double S=0;
    for (int j=0; j<i; j++) {
        S+=L[i][j]*z[j];
    }
    z[i]=b[i]-S;
  }

  for (int i=K-1; i>=0; i--) {
    double S=0;
    for (int j=K-1; j>i; j--) {
        S+=U[i][j]*a[j];
    }
    a[i]=(z[i]-S)/U[i][i];
  }

  for (int i=0; i<N; i++) {
    f[i]=a[0]+a[1]*x[i];
    Q+=pow(f[i]-y[i],2);
  }

  cout << "F_1(x)=" << a[0] << "+" << a[1] << "x\n\n";

  cout << "x\t";
  for (int i=0; i<N; i++) {
    cout << x[i] << "\t";
  }
  cout << endl;

  cout << "F_1(x)\t";
  for (int i=0; i<N; i++) {
    cout << f[i] << " ";
  }
  cout << endl << endl;

  cout << "Q=" << Q << endl;
```

```
}
```

3.2 Многочлен 2-ой степени

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

int main() {

  int K=3;
  int N=6;
  double S_x=0;
  double S_x2=0;
  double S_x3=0;
  double S_x4=0;
  double S_y=0;
  double S_xy=0;
  double S_x2y=0;
  double Q=0;
  vector <double> x = {-1, 0, 1, 2, 3, 4};
  vector <double> y = {-0.4597, 1, 1.5403, 1.5839, 2.010, 3.3464};
  vector <double> f (N,0);
  vector <vector <double>> A (K, vector <double> (K,0));
  vector <vector <double>> L (K, vector <double> (K,0));
  vector <vector <double>> U (K, vector <double> (K,0));
  vector <double> b (K,0);
  vector <double> z (K,0);
  vector <double> a (K,0);


  for (int i=0; i<N; i++) {
    S_x+=x[i];
    S_x2+=pow(x[i],2);
    S_x3+=pow(x[i],3);
    S_x4+=pow(x[i],4);
    S_y+=y[i];
    S_xy+=x[i]*y[i];
    S_x2y+=pow(x[i],2)*y[i];
  }

  A[0][0]=N;
  A[0][1]=S_x;
  A[0][2]=S_x2;

  A[1][0]=S_x;
  A[1][1]=S_x2;
  A[1][2]=S_x3;
```

```
  A[2][0]=S_x2;
  A[2][1]=S_x3;
  A[2][2]=S_x4;

  b[0]=S_y;
  b[1]=S_xy;
  b[2]=S_x2y;

  U=A;

  for (int k=0; k<K-1; k++) {

    for (int i=k; i<K; i++) {
      for (int j=i; j<K; j++) {
        L[j][i]=U[j][i]/U[i][i];
      }
    }

    for (int i=k+1; i<K; i++) {
      for (int j=k; j<K; j++) {
        U[i][j]=U[i][j]-L[i][k]*U[k][j];
      }
    }
  }

  for (int i=0; i<K; i++) {
    double S=0;
    for (int j=0; j<i; j++) {
        S+=L[i][j]*z[j];
    }
    z[i]=b[i]-S;
  }

  for (int i=K-1; i>=0; i--) {
    double S=0;
    for (int j=K-1; j>i; j--) {
        S+=U[i][j]*a[j];
    }
    a[i]=(z[i]-S)/U[i][i];
  }

  for (int i=0; i<N; i++) {
    f[i]=a[0]+a[1]*x[i]+a[2]*pow(x[i],2);
    Q+=pow(f[i]-y[i],2);
  }

  cout << "F_2(x)=" << a[0] << "+" << a[1] << "x" << a[2] << "x^2\n\n";
```

```
  cout << "x\t";
  for (int i=0; i<N; i++) {
    cout << x[i] << "\t";
  }
  cout << endl;

  cout << "F_2(x)\t";
  for (int i=0; i<N; i++) {
    cout << f[i] << " ";
  }
  cout << endl << endl;

  cout << "Q=" << Q << endl;
}
```

4 Консоль

4.1 Многочлен 1-ой степени

```
F_1(x)=0.556165+0.631546x

x -1 0 1 2 3 4
F_1(x) -0.075381 0.556165 1.18771 1.81926 2.4508 3.08235

Q=0.788433
```

4.2 Многочлен 2-ой степени

```
F_2(x)=0.568942+0.689044x-0.0191661x^2

x -1 0 1 2 3 4
F_2(x) -0.139268 0.568942 1.23882 1.87037 2.46358 3.01846

Q=0.774719
```

4 Численное дифференцирование

1 Постановка задачи

Вычислить первую и вторую производную от таблично заданной функции $y_i = f(x_i)$ в точке $x = X^*$.

$X^* = 0.8$

i	0	1	2	3	4
x_i	0.2	0.5	0.8	1.1	1.4
y_i	12.906	5.5273	3.8777	3.2692	3.0319

2 Описание

Вычислим первую производную с первым порядком точности по формуле:

$$y'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad x \in [x_i, x_{i+1}].$$

Найдем левостороннюю производную на отрезке $[x_1, x_2]$, т.к. правая граница отрезка совпадает с X^*:

$$y'(0.8) = \frac{y_2 - y_1}{x_2 - x_1}.$$

Аналогично найдем правостороннюю производную на отрезке $[x_2, x_3]$:

$$y'(0.8) = \frac{y_3 - y_2}{x_3 - x_2}.$$

Далее вычислим первую производную со вторым порядком точности по формуле:

$$y'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}(2x - x_i - x_{i+1}), \quad x \in [x_i, x_{i+1}].$$

Для данной задачи:

$$y'(0.8) = \frac{y_2 - y_1}{x_2 - x_1} + \frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}(2 * 0.8 - x_1 - x_2).$$

И наконец, вычислим вторую производную по формуле:

$$y''(x) = 2\frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}, \quad x \in [x_i, x_{i+1}].$$

В точке $x = 0.8$ она будет равна:

$$y''(0.8) = 2\frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}.$$

3 Исходный код

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

int main() {

  double X=0.8;
  int border=2;
  double y_pl, y_pr, y_p, y_pp;

  vector <double> x = {0.2, 0.5, 0.8, 1.1, 1.4};
  vector <double> y = {12.906, 5.5273, 3.8777, 3.2692, 3.0319};
  y_pl=(y[border]-y[border-1])/(x[border]-x[border-1]);
  y_pr=(y[border+1]-y[border])/(x[border+1]-x[border]);
  y_p=y_pl+(y_pr-y_pl)/(x[border+1]-x[border-1])*(2*X-x[border]-x[border-1]);
  y_pp=2*(y_pr-y_pl)/(x[border+1]-x[border-1]);
  cout << "Left-side y'(" << X << ")=" << y_pl << endl;
  cout << "Right-side y'(" << X << ")=" << y_pr << endl;
  cout << "y'(" << X << ")=" << y_p << endl;
  cout << "y''(" << X << ")=" << y_pp << endl;
}
```

4 Консоль

```
Left-side y'(0.8)=-5.49867
Right-side y'(0.8)=-2.02833
y'(0.8)=-3.7635
y''(0.8)=11.5678
```

5 Численное интегрирование

1 Постановка задачи

Вычислить определенный интеграл $F = \int_{X_0}^{X_k} y dx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга.

$y = \frac{x^2}{x^3-27}$, $X_0 = -2$, $X_k = 2$, $h_1 = 1$, $h_2 = 0.5$.

2 Описание

Формула интегрирования методом прямоугольников с постоянным шагом:

$$F = h(y(\frac{x_0 + x_1}{2}) + y(\frac{x_1 + x_2}{2}) + ... + y(\frac{x_{N-1} + x_N}{2})).$$

Методом трапеций:

$$F = h(\frac{y_0}{2} + y_1 + y_2 + ... + y_{N-1} + \frac{y_N}{2}).$$

Методом Симпсона:

$$F = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + ... + 2y_{N-2} + 4y_{N-1} + y_N).$$

Метод Рунге-Ромберга-Ричардсона для вычисления более точного значения:

$$F = F_h + \frac{F_h - F_{kh}}{k^p - 1} + O(h^{p+1}).$$

Для нашей задачи $k = \frac{h_2}{h_1}$, $p = 2$.

Погрешность для каждого метода вычисляется как модуль разности точного значения и значения, высчитанного по методу Рунге-Ромберга-Ричардсона.

3 Исходный код

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

double f(double x) {
  return pow(x,2)/(pow(x,3)-27);
}

double rectangle(double X0, double Xk, double h) {
  int N=(Xk-X0)/h;
  double S=0;
  double temp;
  for (int i=0; i<N; i++) {
    temp=X0+i*h;
    S+=f((2*temp+h)/2);
  }
  return h*S;
}

double trapeze(double X0, double Xk, double h) {
  int N=(Xk-X0)/h;
  double S=f(X0)/2;
  double temp;
  for (int i=1; i<N; i++) {
    temp=X0+i*h;
    S+=f(temp);
  }
  S+=f(temp+h)/2;
  return h*S;
}

double simpson(double X0, double Xk, double h) {
  int N=(Xk-X0)/h;
  double S=f(X0);
  double temp;
  for (int i=1; i<N; i++) {
    temp=X0+i*h;
    S+=2*f(temp)*(i%2+1);
  }
  S+=f(temp+h);
  return h*S/3;
}

double rectangle_RRR(double X0, double Xk, double h1, double h2) {
  double F_h=rectangle(X0, Xk, h1);
```

```
  double F_hk=rectangle(X0, Xk, h2);
  double k=h2/h1;
  int p=2;
  return F_h + (F_h-F_hk)/(pow(k,p)-1);
}

double trapeze_RRR(double X0, double Xk, double h1, double h2) {
  double F_h=trapeze(X0, Xk, h1);
  double F_hk=trapeze(X0, Xk, h2);
  double k=h2/h1;
  int p=2;
  return F_h + (F_h-F_hk)/(pow(k,p)-1);
}

double simpson_RRR(double X0, double Xk, double h1, double h2) {
  double F_h=simpson(X0, Xk, h1);
  double F_hk=simpson(X0, Xk, h2);
  double k=h2/h1;
  int p=2;
  return F_h + (F_h-F_hk)/(pow(k,p)-1);
}

int main() {

  double h1=1;
  double h2=0.5;
  double X0=-2;
  double Xk=2;
  double sol=-0.203636;

  cout << "h=" << h1 << endl;
  cout << "rectangle\ttrapeze\t\tsimpson\n";
  cout << rectangle(X0, Xk, h1) << "\t" << trapeze(X0, Xk, h1) << "\t" << simpson(X0, Xk, h1) << endl;

  cout << endl;

  cout << "h=" << h2 << endl;
  cout << "rectangle\ttrapeze\t\tsimpson\n";
  cout << rectangle(X0, Xk, h2) << "\t" << trapeze(X0, Xk, h2) << "\t" << simpson(X0, Xk, h2) << endl;

  cout << endl;

  cout << "exact solution=" << sol << endl;

  cout << endl;

  cout << "method R-R-R\n";
```

```
    cout << "rectangle\ttrapeze\t\tsimpson\n";
    cout << rectangle_RRR(X0, Xk, h1, h2) << "\t"<< trapeze_RRR(X0, Xk, h1, h2) << "\t"
        << simpson_RRR(X0, Xk, h1, h2) << endl;

    cout << endl;

    cout << "error\n";
    cout << "rectangle\ttrapeze\t\tsimpson\n";
    cout << abs(sol-rectangle_RRR(X0, Xk, h1, h2)) << "\t" << abs(sol-trapeze_RRR(X0, Xk
        , h1, h2)) << "\t" << abs(sol-simpson_RRR(X0, Xk, h1, h2)) << endl;
}
```

4 Консоль

```
h=1
rectangle trapeze simpson
-0.187831 -0.236582 -0.207172

h=0.5
rectangle trapeze simpson
-0.199406 -0.212206 -0.204081

exact solution=-0.203636

method R-R-R
rectangle trapeze simpson
-0.203265 -0.204081 -0.203051

error
rectangle trapeze simpson
0.000371321 0.000445336 0.00058481
```

Выводы

В этой лабораторной работе я познакомилась с задачами приближения функций с помощью многочленов Лагранжа и Ньютона, узнала что такое сплайн-интерполяция и метод наименьших квадратов. Также я познакомилась с методами численного дифференцирования и методами численного интегрирования, такими как метод прямоугольников, трапеций, Симпсона, метод Рунге-Ромберга для вычисления более точного значения.