Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Кафедра прикладной математики

Лабораторная работа №2

по дисциплине «Уравнения математической физики»

Факультет: ПМИ

Группа: ПМ-01

Студент: Александров М.Е.

Преподаватели: Персова М.Г.

Задорожный А.Г.

Вариант: 3

Новосибирск, 2013

**Цель работы**

Разработать программу решения нелинейной одномерной краевой задачи методом конечных элементов. Сравнить метод простой итерации и метод Ньютона для решения данной задачи.

**Задание**

Уравнение:



Базисные функции – линейные.

Краевые условия всех типов.

**Анализ задачи**

*Метод простой итерации*







Получаем, что .



Следовательно:





Матрица жесткости



Матрица масс



Вектор правой части



Проводим сборку глобальной матрицы и вектора правой части конечно элементной СЛАУ. В результате получится три диагональная матрица.

*Метод Ньютона*

Основа метода состоит в том, что каждый нелинейный член уравнения представляется в виде разложения в ряд Тейлора в окрестности вектора весов:





Получается новая СЛАУ: , для которой













*Остановка итерационного процесса*

Для выхода из итерационного процесса воспользуемся формулой 

Где числитель – норма абсолютной невязки нелинейной системы, а знаменатель – норма ее правой части.

*Использование релаксации*

Для ускорения процесса сходимости нелинейной задачи, ищем следующие приближение по формуле:



**Учет краевых условий**

Учет первых краевых осуществляется так: соответствующий диагональный элемент заменяют единицей, а соответствующую координату вектора правой части значением , где  – есть первое краевое условие в узле.



Вторые краевые условия учитываются добавкой значения θ к глобальному вектору.



Третьи краевые условия учитываются добавкой значения β к диагональному элементу глобальной матрицы и значения  к глобальному вектору.

**Текст программы**

/\*Файл main.cpp\*/

#include<stdio.h>

#include<conio.h>

#include<math.h>

#include"structs.h"

#include"GenerateSetkaMKE.h"

#include"MethodProgonki.h"

#include"Multiply\_TridiaganalMatrixOnVector.h"

#define key 0 //0 - генератор не включается, любое число не ноль - генератор сетки

#define NumberFunction 2

//номер функции

const int MaxIter = 1000;

const double eps = 1.0e-7;

int N; //кол-во КЭ

Matrix A;

FinalElement \*Mass;//массив КЭ

double \*f,\*q,\*q1;//вектор правой части,вектор весов новый,вектор весов на рпошлой итерации + вспомгательный

double \*temp;

double LocalMatrix[2][2];//локальная матрица

double \*MassivGamma;//массив значений гамма на каждом КЭ

double \*MassivBetta;//массив значений бетта на каждом КЭ

double U(double x)//аналитическая функция

{

switch(NumberFunction)

{

case 0: return 4.675;

case 1: return x;

case 2: return x\*x;

case 3: return pow(x,3);

case 4: return sin(x);

default:

printf("\n\n\tFatal error!!! Incorrect number function");

\_getch();

return 0.0;

}

}

double dLambda(double t\_aproks, double x)

{

return 1.0;

}

double lambda(double t\_aproks, double x)//функция лямбда

{

switch(NumberFunction)

{

case 0: return t\_aproks + 1;//U'+1

case 1: return t\_aproks + 1;

case 2: return t\_aproks + 1;

case 3: return t\_aproks + 1;

case 4: return t\_aproks + 1;

default:

printf("\n\n\tFatal error!!! Incorrect number function");

\_getch();

return 0.0;

}

}

double func(double x, double gamma)//функция правой части

{

switch(NumberFunction)

{

case 0: return gamma\*U(x);

case 1: return gamma\*U(x);

case 2: return -8.0\*x - 2.0 + gamma\*U(x);

case 3: return -36.0\*pow(x,3)-6.0\*x+gamma\*U(x);

case 4: return sin(x)-sin(2\*x)+gamma\*U(x);

default:

printf("\n\n\tFatal error!!! Incorrect number function");

\_getch();

return 0.0;

}

}

double Derivative(double x)//производная

{

return (U(x+0.1)-U(x-0.1))/0.2;

}

int input()

{

FILE \*in = fopen("Setka.txt","r");

if(in){

fscanf(in,"%d",&N);

Mass = new FinalElement[N];

for(int i=0; i<N; i++)//сформировали массив КЭ

fscanf(in,"%lf %lf %lf",&Mass[i].x1,&Mass[i].x2,&Mass[i].h);

fclose(in);

//выделяем память под матрицу,под векторы

A.b = new double[N+1];

A.c = new double[N+1];

A.d = new double[N+1];

q = new double[N+1];

q1 = new double[N+1];

f = new double[N+1];

temp = new double[N+1];

MassivGamma = new double[N];//под массив значений гамма

MassivBetta = new double[N];//под массив значений гамма

in = fopen("MassivGamma.txt","r");

for(int i=0; i<N; i++)//сформировали массив гамма

fscanf(in,"%lf",&MassivGamma[i]);

fclose(in);

in = fopen("MassivBetta.txt","r");

for(int i=0; i<N; i++)//сформировали массив бетта

fscanf(in,"%lf",&MassivBetta[i]);

fclose(in);

in = fopen("InitialApproach.txt","r");

//начальное приближение

for(int i=0; i<=N; i++)

fscanf(in,"%lf",&q[i]);

fclose(in);

return 1;

}

else{

fclose(in);

return 0;

}

}

double EuclidNorma(double \*t)

{

double norma = 0.0;

for(int i=0; i<=N; i++)

norma += pow(t[i],2);

return sqrt(norma);

}

int Continue(int k)

{

double chisl,znam;

//по невязке

Multiply\_TridiaganalMatrixOnVector B;

B.Multiply(temp,A.b,A.c,A.d,q,N+1);

for(int i=0; i<=N; i++)

temp[i] = f[i] - temp[i];

chisl = EuclidNorma(temp);

znam = EuclidNorma(f);

printf("Iter = %d\tNormaNevjazka = %le\n",k,sqrt(chisl/znam));

if(sqrt(chisl/znam) < eps)return 0;

else return 1;

}

double NewtonsAdd(double Xk,double Xk1,double h,double q1,double q2,double arg, int num)

{

double dLdQ1\_left,dLdQ1\_right,dLdQ2\_left,dLdQ2\_right;

dLdQ1\_left = -dLambda(arg,Xk)/h;

dLdQ1\_right = -dLambda(arg,Xk1)/h;

dLdQ2\_left = dLambda(arg,Xk)/h;

dLdQ2\_right = dLambda(arg,Xk1)/h;

double dA11dQ1,dA11dQ2,dA12dQ1,dA12dQ2,dA21dQ1,dA21dQ2,dA22dQ1,dA22dQ2;

dA11dQ1 = dA22dQ1 = (dLdQ1\_left + dLdQ1\_right)/(2.0\*h);

dA11dQ2 = dA22dQ2 = (dLdQ2\_left + dLdQ2\_right)/(2.0\*h);

dA12dQ1 = dA21dQ1 = -(dLdQ1\_left + dLdQ1\_right)/(2.0\*h);

dA12dQ2 = dA21dQ2 = -(dLdQ2\_left + dLdQ2\_right)/(2.0\*h);

switch(num){

case 1:

return (dA11dQ1\*q1 + dA12dQ1\*q2);

case 2:

return (dA11dQ2\*q1 + dA12dQ2\*q2);

case 3:

return (dA21dQ1\*q1 + dA22dQ1\*q2);

case 4:

return (dA21dQ2\*q1 + dA22dQ2\*q2);

case 5:

return (q1\*(dA11dQ1\*q1 + dA11dQ2\*q2) + q2\*(dA12dQ1\*q1 + dA12dQ2\*q2));

case 6:

return (q1\*(dA21dQ1\*q1 + dA21dQ2\*q2) + q2\*(dA22dQ1\*q1 + dA22dQ2\*q2));

}

}

void CreateLocalMatrixAndRightPart(int j)

{

double Xk,Xk1,h,gamma,arg,q1,q2;//локальные границы КЭ (Хк и Хк+1) и размер КЭ

//формируем все локальные переменные

Xk = Mass[j].x1;

Xk1 = Mass[j].x2;

h = Mass[j].h;

gamma = MassivGamma[j];

q1 = q[j];

q2 = q[j+1];

arg = (q2-q1)/h;

//считаем матрицу G - жесткости

LocalMatrix[0][0] = (lambda(arg,Xk) + lambda(arg,Xk1))/(2.0\*h) + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,1);

LocalMatrix[0][1] = -(lambda(arg,Xk) + lambda(arg,Xk1))/(2.0\*h) + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,2);

LocalMatrix[1][0] = -(lambda(arg,Xk) + lambda(arg,Xk1))/(2.0\*h) + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,3);

LocalMatrix[1][1] = (lambda(arg,Xk) + lambda(arg,Xk1))/(2.0\*h) + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,4);

//считаем матрицу М - массы и плюсуем ее к матрице G

LocalMatrix[0][0] += gamma\*h/3.0;

LocalMatrix[0][1] += gamma\*h/6.0;

LocalMatrix[1][0] += gamma\*h/6.0;

LocalMatrix[1][1] += gamma\*h/3.0;

//считаем вектор правой части и сразу же плюсуем его в глобальный вектор

f[j] += h\*(2.0\*func(Xk,gamma)+func(Xk1,gamma))/6.0 + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,5);

f[j+1] += h\*(func(Xk,gamma)+2.0\*func(Xk1,gamma))/6.0 + NewtonsAdd(Xk,Xk1,h,q1,q2,arg,6);

}

Void MethodSimpleIteration()

{

int i;

double W = 1.00;

MethodProgonki Calc;

SaveInitialApproach();

GenereteGlobalMatrix();

for(i=0; i<MaxIter && Continue(i); i++){//глобальный цикл

SaveInitialApproach(); //Qi-1 = Qi

Calc.CalcProgonka(A.b,A.c,A.d,f,q,N+1); //метод Прогонки

ParameterRelaxation(W); //Параметр релаксации

GenereteGlobalMatrix();

}

Output(i);

}

void AddLocalInGlobal(int j)

{

A.c[j] += LocalMatrix[0][0];

A.d[j] += LocalMatrix[0][1];

A.b[j+1] += LocalMatrix[1][0];

A.c[j+1] += LocalMatrix[1][1];

}

void ParameterRelaxation(double W) //получение нового приближения с учетом параметра релаксации

{

for(int i=0; i<=N; i++)

q[i] = W\*q[i] + (1.0 - W)\*q1[i];

}

void NullMatrixAndVector()

{

for(int i=0; i<=N; i++)

f[i] = A.b[i] = A.c[i] = A.d[i] = 0.0;

}

double Tetta(int num)

{

double q1,q2,h,x,ret; //возвращаемое значение

double dx; //значение производной

if(!num){

q1 = q[0];

q2 = q[1];

h = Mass[0].h;

x = Mass[0].x1;

dx = Derivative(Mass[0].x1);

}

else{

q1 = q[N-1];

q2 = q[N];

h = Mass[N-1].h;

x = Mass[N-1].x2;

dx = Derivative(Mass[N-1].x2);

}

ret = dx\*lambda((q2-q1)/h,x); //ret = lambda\*(dU/dx)

if(!num)//если левая граница

return -ret;

else return ret;

}

double Ubetta(int num)

{

double q1,q2,h,x,ret,betta; //возвращаемое значение

double dx; //значение производной

//Ubetta = (lambda\* dU/dn + U|x=x0)/betta

if(!num){

q1 = q[0];

q2 = q[1];

h = Mass[0].h;

x = Mass[0].x1;

betta = MassivBetta[0];

dx = -Derivative(Mass[0].x1);

}

else{

q1 = q[N-1];

q2 = q[N];

h = Mass[N-1].h;

x = Mass[N-1].x2;

betta = MassivBetta[N-1];

dx = Derivative(Mass[N-1].x2);

}

ret = 1.0/betta \*(lambda((q2-q1)/h,x)\*dx + betta\*U(x));

return ret;

}

void FirstKray(int i)

{

if(!i){//если левая граница отрезка

A.c[0] = 1.0;

A.d[0] = 0.0;

f[0] = U(Mass[0].x1);

}

else{//если правая граница отрезка

A.c[N] = 1.0;

A.b[N] = 0.0;

f[N] = U(Mass[N-1].x2);

}

}

void SecondKray(int i)

{

switch(i){

case 0://если левая граница

f[0] += Tetta(0);

break;

case 1://если правая граница

f[N] += Tetta(1);

break;

}

}

void ThirdKray(int i)

{

if(!i){//если левая граница

A.c[0] += MassivBetta[0];

f[0] += MassivBetta[0]\*Ubetta(i);

}

else{//если правая граница

A.c[N] += MassivBetta[N-1];

f[N] += MassivBetta[N-1]\*Ubetta(i);

}

}

void AddKray(Matrix \*A)

{

int num; //номер краевого

FILE \*kray = fopen("Kray.txt","r");

for(int i=0; !feof(kray) && i<2; i++){

fscanf(kray,"%d",&num);

switch(num){

case 1:

FirstKray(i);

break;

case 2:

SecondKray(i);

break;

case 3:

ThirdKray(i);

break;

default:

printf("\tFatal Error! Net takih kraevyh!\n");

break;

}

}

fclose(kray);

}

void Output(int iter)

{

int t;

double x = Mass[0].x1;

double norma = 0.0;

FILE \*res = fopen("result.txt","w");

FILE \*kr = fopen("Kray.txt","r");

fscanf(kr,"%d",&t);

fprintf(res,"Kray:\t%d",t);

fscanf(kr,"%d",&t);

fprintf(res," and %d\n\n",t);

printf("\n\tIter = %d\n",iter);

for(int i=0; i<=N; i++,x+=Mass[i-1].h){

norma += pow(U(x)-q[i],2);

printf("%.15lf\t%.15lf\t%le\n",q[i],U(x),fabs(U(x)-q[i]));

fprintf(res,"%.15lf\t\t%.15lf\t\t%le\n",q[i],U(x),fabs(U(x)-q[i]));

}

printf("\n\tNorma = %le",sqrt(norma));

fprintf(res,"\n\tNorma = %le",sqrt(norma));

fclose(res);

fclose(kr);

}

void SaveInitialApproach()

{

//вектор Qk-1 получает значения вектора Qk

for(int j=0; j<=N; j++)

q1[j] = q[j];

}

void GenereteGlobalMatrix()

{

NullMatrixAndVector(); //обнуляем матрицу и вектор правой части

for(int j=0; j<N; j++){//цикл генерации глоабльной матрицы на i-й итерации метода простой итерации

CreateLocalMatrixAndRightPart(j); //сгенерировали локальную матрицу и вектор правой части

AddLocalInGlobal(j); //добавляем локальное в глобальное

}

AddKray(&A);//учет краевых

}

void MethodNewton()

{

int i;

double W = 1.00;

MethodProgonki Calc;

SaveInitialApproach();

GenereteGlobalMatrix();

for(i=0; i<MaxIter && Continue(i); i++){//глобальный цикл

SaveInitialApproach(); //Qi-1 = Qi

Calc.CalcProgonka(A.b,A.c,A.d,f,q,N+1); //метод Прогонки

ParameterRelaxation(W); //Параметр релаксации

GenereteGlobalMatrix();

}

Output(i);

}

int main()

{

GenerateSetkaMKE A;

if(key)A.Generate();

if(input()){

MethodNewton();

printf("\n\n\tComplete!");

\_getch();

return 0;

}

else{

printf("\n\n\tError in input data!!!");

return 1;

}

}

/\*Файл structs.h\*/

struct FinalElement{

double x1,x2; //коордианты начала и конца КЭ

double h; //длина КЭ

};

struct Matrix{

double \*b,\*c,\*d;//диагонали матрицы

};

/\*Файл GenerateSetkaMKE.h\*/

class GenerateSetkaMKE{

private:

double beg,end,h,len,k;//начало отрезка,конец, шаг, длина, коэффициент разрядки

int N;//кол-во отрезков

void input();

public:

void Generate();

};

/\*Файл GenerateSetkaMKE.cpp\*/

#include "GenerateSetkaMKE.h"

#include <stdio.h>

#include <math.h>

void GenerateSetkaMKE::input()

{

int P=1;

while(P){//ввод числа КЭ

printf("\n\tEnter N = ");

scanf("%d",&N);

if(N>0)P=0;

else printf("\nError in input data! Please return!\n");

}

P=1;

while(P){//ввод границ интервала

printf("\n\tEnter beg = ");

scanf("%lf",&beg);

printf("\n\tEnter end = ");

scanf("%lf",&end);

if(end>beg)P=0;

else printf("\nError in input data! Please return!\n");

}

len = fabs(end - beg);

P=1;

while(P){//ввод коэф. разрядки

printf("\n\tEnter k = ");

scanf("%lf",&k);

if(k>0.0)P=0;

else printf("\nError in input data! Please return!\n");

}

if(k == 1.0)

h = len/N;

else

h = (1.0 - k)\*len/(1.0 - pow(k,N));

}

void GenerateSetkaMKE::Generate()

{

double x;//текущая позиция

input();

FILE \*setka = fopen("Setka.txt","w");

fprintf(setka,"%d\n",N);//вывели кол-во КЭ

printf("%d\n",N);

int i=0;

x = beg+h;

while(x < end + h/10.0 && i<=N){

fprintf(setka,"%.16lf\n%.16lf\n%.16lf\n\n",x-h,x,h);

printf("%le\t%le\t%le\n",x-h,x,h);

h \*= k;

x += h;

i++;

}

fclose(setka);

}

/\*Файл MethodProgonki.h\*/

class MethodProgonki{

public:

void CalcProgonka(double \*,double \*,double \*,double \*,double \*,int);

//три диагонали + вектор правой части + вектор куда результат положим + размерность МАТРИЦЫ а не КЭ

};

/\*Файл MethodProgonki.cpp\*/

#include"MethodProgonki.h"

void MethodProgonki::CalcProgonka(double \*b, double \*c, double \*d, double \*r, double \*X, int N)//здесь N должно быть N+1,т.к. кол-во узлов,а не КЭ

{

int i;

double \*P,\*T;//коэффициенты лямбда и дельта

P = new double[N];

T = new double[N];

P[0] = (-1.0)\*d[0]/c[0];

T[0] = r[0]/c[0];

for(i=1; i<N; i++){//посчитали бэтта и лямбда

P[i] = (-1.0)\*d[i]/(c[i] + b[i]\*P[i-1]);

T[i] = (r[i] - b[i]\*T[i-1])/(c[i] + b[i]\*P[i-1]);

}

X[N-1] = T[N-1];

for(i=N-2; i>-1; i--){

X[i] = P[i]\*X[i+1] + T[i];

}

}

/\*Файл Multiply\_TridiaganalMatrixOnVector.h\*/

class Multiply\_TridiaganalMatrixOnVector{

private:

public:

void Multiply(double \*,double \*,double \*,double \*,double \*,int);//вектор куда результат; 3 диагонали; вектор на который умножать + размерность МАТРИЦЫ

};

/\*Файл Multiply\_TridiaganalMatrixOnVector.cpp\*/

#include"Multiply\_TridiaganalMatrixOnVector.h"

void Multiply\_TridiaganalMatrixOnVector::Multiply(double \*res, double \*b, double \*c, double \*d, double \*q, int N)

{

int i,j;

for(i=0; i<N; i++){

res[i] = b[i]\*q[i-1];

res[i] += c[i]\*q[i];

res[i] += d[i]\*q[i+1];

}

}

**Тесты и исследования**

1)Функция: *U(x)=x*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: простой итерации

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 2

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.10000000000000 | 0.10000000000000 | 4.16333600000000E-17 |
| 0.20000000000000 | 0.20000000000000 | 8.32667300000000E-17 |
| 0.30000000000000 | 0.30000000000000 | 1.66533500000000E-16 |
| 0.40000000000000 | 0.40000000000000 | 2.77555800000000E-16 |
| 0.50000000000000 | 0.50000000000000 | 3.33066900000000E-16 |
| 0.60000000000000 | 0.60000000000000 | 3.33066900000000E-16 |
| 0.70000000000000 | 0.70000000000000 | 3.33066900000000E-16 |
| 0.80000000000000 | 0.80000000000000 | 1.11022300000000E-16 |
| 0.90000000000000 | 0.90000000000000 | 1.11022300000000E-16 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=6.862142e-016

Невязка = 3.841401e-002

2)Функция: *U(x)=*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: простой итерации

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 60

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.01000000000001 | 0.01000000000000 | 1.04950800000000E-14 |
| 0.04000000000002 | 0.04000000000000 | 2.25514100000000E-14 |
| 0.09000000000004 | 0.09000000000000 | 3.68038900000000E-14 |
| 0.16000000000005 | 0.16000000000000 | 5.42899100000000E-14 |
| 0.25000000000008 | 0.25000000000000 | 7.68274300000000E-14 |
| 0.36000000000011 | 0.36000000000000 | 1.07969200000000E-13 |
| 0.49000000000016 | 0.49000000000000 | 1.56597000000000E-13 |
| 0.64000000000025 | 0.64000000000000 | 2.52242700000000E-13 |
| 0.81000000000046 | 0.81000000000000 | 4.55524500000000E-13 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=5.640319e-013

Невязка = 9.514638e-007

3)Функция: *U(x)=*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: простой итерации

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 83

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| -0.00099670617898 | 0.00100000000000 | 1.99670600000000E-03 |
| 0.00442346685394 | 0.00800000000000 | 3.57653300000000E-03 |
| 0.02245228785784 | 0.02700000000000 | 4.54771200000000E-03 |
| 0.05907486998990 | 0.06400000000000 | 4.92513000000000E-03 |
| 0.12019025478873 | 0.12500000000000 | 4.80974500000000E-03 |
| 0.21168895753933 | 0.21600000000000 | 4.31104200000000E-03 |
| 0.33947990372298 | 0.34300000000000 | 3.52009600000000E-03 |
| 0.50949385240731 | 0.51200000000000 | 2.50614800000000E-03 |
| 0.72767972101591 | 0.72900000000000 | 1.32027900000000E-03 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=1.112882e-002

Невязка = 8.560801e-007

4)Функция: *U(x)=* sin*(x)*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: простой итерации

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 32

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.10067214038762 | 0.12428432961673 | 2.36121900000000E-02 |
| 0.19683483303830 | 0.24059881829741 | 4.37639900000000E-02 |
| 0.28906975432519 | 0.34800303079858 | 5.89332800000000E-02 |
| 0.37775847162365 | 0.44598518265947 | 6.82267100000000E-02 |
| 0.46311992550614 | 0.53437443286936 | 7.12545100000000E-02 |
| 0.54525407548226 | 0.61326349319912 | 6.80094200000000E-02 |
| 0.62418324374710 | 0.68294219753334 | 5.87589500000000E-02 |
| 0.69988678632352 | 0.74384181741842 | 4.39550300000000E-02 |
| 0.77232763369905 | 0.79648938344187 | 2.41617500000000E-02 |
| 0.84147098480790 | 0.84147098480790 | 0.00000000000000E+00 |

||U\*-U||=1.620857e-001

Невязка = 7.477823e-007

5)Сходимость от параметра релаксации

Функция *U(x)=*

λ(*U’)=U’+1*

= 1

Расчетная область: [0;5]

Число конечных элементов: 5

Неравномерная сетка: k = 1.15

Краевые условия: 1;1

Начальное приближение: нулевой вектор

|  |  |
| --- | --- |
| w | Количество итераций |
| 0.1 | 246 |
| 0.2 | 119 |
| 0.3 | 76 |
| 0.4 | 54 |
| 0.5 | 41 |
| 0.6 | 31 |
| **0.7** | **26** |
| 0.8 | 37 |
| 0.9 | 64 |
| 1.0 | 155 |

6)Функция: *U(x)=x*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: Ньютона

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 1

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.10000000000000 | 0.10000000000000 | 1.94289000000000E-16 |
| 0.20000000000000 | 0.20000000000000 | 3.33066900000000E-16 |
| 0.30000000000000 | 0.30000000000000 | 4.99600400000000E-16 |
| 0.40000000000000 | 0.40000000000000 | 6.66133800000000E-16 |
| 0.50000000000000 | 0.50000000000000 | 7.21645000000000E-16 |
| 0.60000000000000 | 0.60000000000000 | 7.77156100000000E-16 |
| 0.70000000000000 | 0.70000000000000 | 6.66133800000000E-16 |
| 0.80000000000000 | 0.80000000000000 | 5.55111500000000E-16 |
| 0.90000000000000 | 0.90000000000000 | 3.33066900000000E-16 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=1.682134e-015

Невязка = 9.910054e-008

7)Функция: *U(x)=*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: Ньютона

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 6

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.01000000000000 | 0.01000000000000 | 4.16333600000000E-17 |
| 0.04000000000000 | 0.04000000000000 | 6.93889400000000E-17 |
| 0.09000000000000 | 0.09000000000000 | 8.32667300000000E-17 |
| 0.16000000000000 | 0.16000000000000 | 8.32667300000000E-17 |
| 0.25000000000000 | 0.25000000000000 | 0.00000000000000E+00 |
| 0.36000000000000 | 0.36000000000000 | 1.11022300000000E-16 |
| 0.49000000000000 | 0.49000000000000 | 1.66533500000000E-16 |
| 0.64000000000000 | 0.64000000000000 | 1.11022300000000E-16 |
| 0.81000000000000 | 0.81000000000000 | 1.11022300000000E-16 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=2.917638e-016

Невязка = 7.843229e-008

8)Функция: *U(x)=*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: Ньютона

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 6

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| -0.00099670617897 | 0.00100000000000 | 1.99670600000000E-03 |
| 0.00442346685397 | 0.00800000000000 | 3.57653300000000E-03 |
| 0.02245228785788 | 0.02700000000000 | 4.54771200000000E-03 |
| 0.05907486998996 | 0.06400000000000 | 4.92513000000000E-03 |
| 0.12019025478881 | 0.12500000000000 | 4.80974500000000E-03 |
| 0.21168895753944 | 0.21600000000000 | 4.31104200000000E-03 |
| 0.33947990372312 | 0.34300000000000 | 3.52009600000000E-03 |
| 0.50949385240750 | 0.51200000000000 | 2.50614800000000E-03 |
| 0.72767972101624 | 0.72900000000000 | 1.32027900000000E-03 |
| 1.00000000000000 | 1.00000000000000 | 0.00000000000000E+00 |

||U\*-U||=1.112882e-002

Невязка = 3.187864e-008

9)Функция: *U(x)=* sin*(x)*

λ(*U’)=U’+1*

*f(x)=*

= 1

Метод: Ньютона

Сетка: равномерная

Область: [0;1]

Конечных элементов: 10

Краевые условия: 1,1

Число итераций: 4

|  |  |  |
| --- | --- | --- |
| U(x) | U\*(x) | |U\*-U| |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000E+00 |
| 0.08076194757883 | 0.12428432961673 | 1.90714700000000E-02 |
| 0.16182496297707 | 0.24059881829741 | 3.68443700000000E-02 |
| 0.24346686105203 | 0.34800303079858 | 5.20533500000000E-02 |
| 0.32592072498971 | 0.44598518265947 | 6.34976200000000E-02 |
| 0.40935670370212 | 0.53437443286936 | 7.00688300000000E-02 |
| 0.49386812287547 | 0.61326349319912 | 7.07743500000000E-02 |
| 0.57946241587182 | 0.68294219753334 | 6.47552700000000E-02 |
| 0.66605690174738 | 0.74384181741842 | 5.12991900000000E-02 |
| 0.75347908845260 | 0.79648938344187 | 2.98478200000000E-02 |
| 0.84147098480790 | 0.84147098480790 | 0.00000000000000E+00 |

||U\*-U||=1.615460e-001

Невязка = 8.942118e-008

**Выводы**

В результате исследований, метод Ньютона в среднем решает нелинейную систему быстрее по количеству итераций чем метод простой итерации. На рассмотренных задачах точность методов примерно одинаковая. С точки зрения построения матрицы и получения формул, метода Ньютона значительно сложнее.