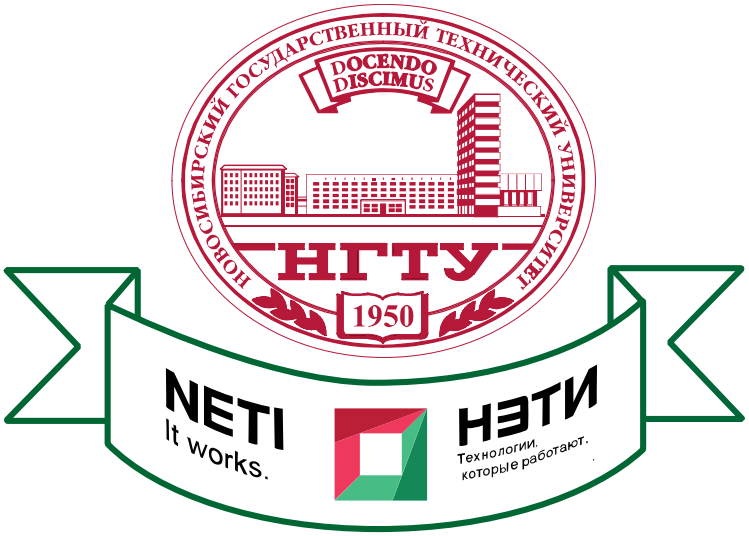
Министерство науки и высшего образования
Российской Федерации

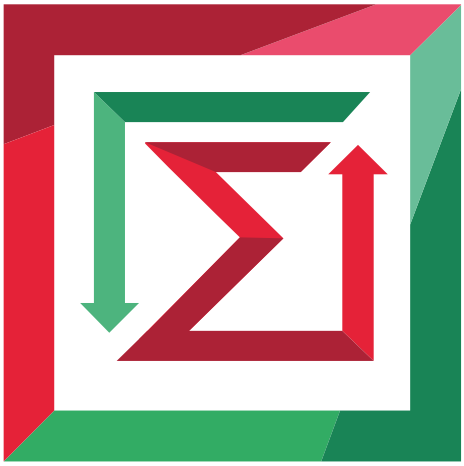
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»




Кафедра прикладной математики

Практическое задание № 3
по дисциплине «Численные методы»

# РЕШЕНИЕ СИСТЕМ НЕЛИНЕЙНЫХ УРАВНЕНИЙ



| | |
|---|---|
| Факультет: | ПМИ |
| Группа: | ПМ-71 |
| Студент: | Востриков Вячеслав |
| Вариант: | 1, 2, 5 |
| Преподаватели: | Патрушев И.И. |
| | Задорожный А.Г. |
| | Персова М.Г. |

Новосибирск
2020

## 1. Цель

Разработать программу решения системы нелинейных уравнений (СНУ) методом Ньютона. Провести исследования метода для нескольких систем размерности от 2 до 10.

## 2. Задание

1. $m \le n$. Для нахождения $\Delta x^k$, являющегося решением системы (4.2), фиксировать как нулевые те ее $(n-m)$ компонентов с номерами $j$, для которых $\max\limits_i \left( abs\left( \dfrac{\partial\left(F_i\left(x^k\right)\right)}{\partial x_j} \right) \right)$ минимальны. Производные при формировании матрицы Якоби вычислять аналитически.

2. $m \ge n$. Для нахождения $\Delta x^k$ из системы (4.2) те ее $(m-n)$ уравнений, для которых абсолютные значения $F_i\left(x^k\right)$ минимальны, исключаются из системы. При вычислении нормы вектора $F^k$ в процессе подбора параметра $\beta^k$ учитывать все уравнения системы. Производные при формировании матрицы Якоби вычислять аналитически.

5. В задании 1 производные при формировании матрицы Якоби вычислять численно.

## 3. Теоретическая часть

Пусть дана СНУ в виде:

$$\begin{aligned} &F_1(x_1, x_2, ..., x_n) = 0; \\ &F_2(x_1, x_2, ..., x_n) = 0; \\ &\quad \dots \\ &F_m(x_1, x_2, ..., x_n) = 0. \end{aligned} \qquad (4.1)$$

Обозначим через $x^k$ решение, полученное на $k$-й итерации процесса Ньютона (для первой итерации $x^0$ – начальное приближение). Запишем исходную систему в виде $F_i\left(x^k + \Delta x\right) = 0$, $i = 1...m$, где $\Delta x^k = \bar{x} - x^k$, $\bar{x}$ – искомое решение. Выполним линеаризацию *i*-го уравнения системы (4.1) с использованием его разложения в ряд Тейлора в окрестности точки $x^k$:

$$F_i(x) \approx F_i\left(x^k\right) + \sum_{j=1}^{n} \left. \frac{\partial\left(F_i(x)\right)}{\partial x_j} \right|_{x=x^k} \Delta x_j^k, \; i = 1...m,$$

или в матричном виде:

$$A^k \Delta x^k = -F^k, \qquad (4.2)$$

где $F^k$ – значение вектор-функции $F$ при $x = x^k$; $A^k$ – матрица Якоби $\left( A_{ij}^k = \left. \frac{\partial (F_i(x))}{\partial x_j} \right|_{x=x^k} \right)$.

Это система уравнений, линейных относительно приращений $\Delta x_j^k$. Решив эту систему, найдем направление $\Delta x^k$ поиска решения.

Для поиска следующего приближения $x^{k+1}$ вдоль направления $\Delta x^k$ организуем итерационный процесс:

$$x_\nu^{k+1} = x^k + \beta_\nu^k \Delta x^k,$$

где $\beta_\nu^k$ – параметр итерационного процесса поиска $x^{k+1}$, $(0 < \beta^k < 1)$, $\nu$ – номер итерации поиска оптимального значения $\beta^k$. Параметр $\beta^k$ будем искать следующим образом: сначала (т. е. после нахождения направления $\Delta x^k$) $\beta^k$ принимается равным 1 и вычисляется значение $F_\nu^k = F(x^k + \beta_\nu^k \Delta x^k)$; далее, пока норма $F_\nu^k$ больше, чем норма $F^{k-1}$, $\beta_\nu^k$ уменьшается вдвое.

## 4. Текст программы

```
#include <stdio.h>
#include "Newton.h"
#include <vector>
#include <math.h>

int       maxiter;
int       m,n;

std::vector<cResult> gr;
xfloat    *x;

int main()
{
   xfloat  *r;
   xfloat  e1,e2,h=0;
   bool   diff;
   FILE   *f=fopen("param.txt","rt");
   if (f==NULL)
   {
          perror("param.txt");
          return 0;
   }
   fscanf(f,"%lf %lf %d",&e1,&e2,&maxiter);
   fclose(f);
   printf("1 to calc ChM\n");
   scanf("%d",&m);
   diff=(m==1);
   if (diff)
   {
          printf("Enter h\n");
```

```
		scanf("%lf",&h);
	}
	printf("Enter m,n\n");
	scanf("%d %d",&m,&n);
	x=new xfloat[n];
	printf("Enter x0(%d):\n",n);
	for (int i=0; i<n; i++)
		scanf("%lf",x+i);
	cSol solution(m,n,diff);
	solution.h=h;
	solution.SetEps(e1,e2);
	solution.SetX(x);
	int i=0;
	cResult	re;
	memcpy(re.x,x,m*sizeof(xfloat));
	re.B=1;
	re.nr=sqrt(solution.norm);
	gr.push_back(re);
	for (int i=0; i<maxiter&&solution.Iteration(); i++)
	{
		r=solution.GetX();
		memcpy(re.x,r,n*sizeof(xfloat));
		re.B=solution.B;
		re.nr=sqrt(solution.norm);
		printf("%d\t",i+1);
		for (int i=0; i<n; i++)
			printf("%lf\t",r[i]);
		printf("\n");
		gr.push_back(re);
	}
	r=solution.GetX();
	memcpy(re.x,r,m*sizeof(xfloat));
	re.B=solution.B;
	re.nr=sqrt(solution.norm);
	for (int i=0; i<n; i++)
		printf("%lf\t",r[i]);
	printf("\n");
	printf("nr:%.3g\n",re.nr);
	gr.push_back(re);
	delete[]x;
}
```

```
#ifndef _func_h_
#define _func_h_

#include <math.h>

double F(int i,double *x);
double A(int i,int j,double *x);


#endif //_func_h_
#include "Func.h"

#define pi 3.14159265358

double F(int i,double *x)
{
	switch(i)
	{
		case 1: return (x[0]-2)*(x[0]-2)+(x[1]-2)*(x[1]-2)-4;
		case 2: return (x[0]-4.5)*(x[0]-4.5)+(x[1]-2)*(x[1]-2)-1;
		case 3: return x[0]-3.85;
	}
```

```
	return 0;
}
double A(int i,int j,double *x)
{
	switch(i)
	{
		case 1:
			switch (j)
			{
				case 1: return 2*(x[0]-2);
				case 2: return 2*(x[1]-2);
				case 3: return 2*x[2];
			}
			return 0;
		case 2:
			switch (j)
			{
				case 1: return 2*(x[0]-4.5);
				case 2: return 2*(x[1]-2);
				case 3: return 2*x[2];
			}
			return 0;
		case 3:
			switch (j)
			{
				case 1: return 1;
				case 2: return 0;
				case 3: return 0;
			}
			return 0;

	}
	return 0;
}


#ifndef _newton_h_
#define _newton_h_

#include "Func.h"
typedef double	xfloat;

class cSol
{
	protected:
		int	m,n;
		xfloat a[10][10];
		xfloat f[10];
		xfloat x[10];
		void Calc1();
		void Calc();
		int	number[10];
		xfloat lmax[10];
		xfloat Ef,Eb;
		xfloat norm0;
		int	fact();
		bool diff;

	public:
		xfloat B;
		xfloat norm;
		xfloat h;
		cSol(int _m,int _n,bool Diff=false);
		void SetX(xfloat *_x);
```

```
		xfloat *GetX();
		void   SetEps(xfloat _ef,xfloat _eb);
		int          Iteration();
		~cSol();
};

class cResult
{
public:
	cResult()
	{
		x[0]=x[1]=x[2]=0;
	}
	xfloat x[10];
	xfloat nr;
	xfloat B;
};

#endif //_newton_h_



#include "Newton.h"

#include <math.h>
#include <stdio.h>
#include <string.h>


xfloat norma2(xfloat *x,int n,int *number)
{
	xfloat S=0;
	for (int i=0; i<n; i++)
		S+=x[number[i]]*x[number[i]];
	return S;
}

cSol::cSol(int _m, int _n,bool Diff)
{
	m=_m;
	n=_n;
	for (int i=0; i<n; i++)
		x[i]=0;
	for (int i=0; i<n; i++)
		f[i]=0;
	diff=Diff;
	h=1E-3;

}



xfloat *cSol::GetX()
{
	return x;
}

void cSol::SetX(xfloat *_x)
{
	memcpy(x,_x,n*sizeof(xfloat));
	for (int i=n; i<10; i++)
		x[i]=0;

	Calc1();
```

```
	norm0=norm=norma2(f,m,number);
	B=1;
}

int cSol::Iteration()
{
	xfloat nr2;
	Calc1();
	nr2=norma2(f,m,number);
	Gauss();
		for (int i=0; i<m; i++)
		x[i]+=B*f[number[i]];
	if (nr2>norm)
		B/=2;
	if (nr2<Ef*norm0)
	{
		printf("|f|/|f0|<%.2g\n",Ef);
		return 0;
	}
	norm=nr2;
	if (B<Eb)
	{
		printf("B<%.2g\n",Eb);
		return 0;
	}
	return 1;
}
void cSol::Calc1()
{
	int i,j,t;
	xfloat s;
	xfloat tx[10];
	if (!diff)
		for (i=0; i<m; i++)
		{
			f[i]=-F(i+1,x);
			for (j=0; j<n; j++)
			{
				a[i][j]=A(i+1,j+1,x);
			}
		}
	if (diff)
	{
		for (i=0; i<m; i++)
		{
			f[i]=-F(i+1,x);
			for (j=0; j<n; j++)
			{
				s=x[j];
				x[j]+=h;
				a[i][j]=F(i+1,x);
				x[j]-=h;
				a[i][j]-=F(i+1,x);
				x[j]=s;
				a[i][j]/=2*h;
			}
		}

	}
	int r;
	for (i=0; i<n; i++)
		number[i]=i;
	if (m<n)
	{
```

```
		for (i=0; i<n; i++)
			lmax[i]=fabs(a[0][i]);
		for (i=1; i<m; i++)
			for (j=0; j<n; j++)
				if (fabs(a[i][j])>lmax[j])
					lmax[j]=fabs(a[i][j]);
		for (i=0; i<n; i++)
			for (j=0; j<i; j++)
			{
				s=a[i][j];
				a[i][j]=a[j][i];
				a[j][i]=s;
			}
		for (i=0; i<n-1; i++)
			for (j=i+1; j<n; j++)
			{
				if (lmax[i]<lmax[j])
				{
					s=lmax[i];
					lmax[i]=lmax[j];
					lmax[j]=s;
					memcpy(tx,a[i],m*sizeof(xfloat));
					memcpy(a[i],a[j],m*sizeof(xfloat));
					memcpy(a[j],tx,m*sizeof(xfloat));

					r=number[i];
					number[i]=number[j];
					number[j]=r;

					s=f[i];
					f[i]=f[j];
					f[j]=s;
				}
			}

		for (i=0; i<n; i++)
			for (j=0; j<i; j++)
			{
				s=a[i][j];
				a[i][j]=a[j][i];
				a[j][i]=s;
			}

	}
}
void cSol::SetEps(xfloat _ef,xfloat _eb)
{
	Ef=_ef*_ef;
	Eb=_eb;
}
//=======================
#include "cSalt.h"

int main()
{
	cSold solution;
	solution.calc();
	return 0;
}
#include <stdio.h>
#include <math.h>
#include <conio.h>

class cSold
```

```
{
protected:
	int maxiter, N, m;
	double nev, normaf, eps, min, *x, *x1, *delta_x, *ff, beta;
	double **A;

	int Gauss();
	void jacobi();
	void iskl();
	void vvod();
	double norma_f(double *x);
	void summa(double *x1,double *x, double *y, double a);
public:
	void calc();
};
#include "cSalt.h"

double F(int i,double *x)
{
	switch(i)
	{
		case 0: return x[0]+x[1]-3;
		case 1: return x[1]-x[0]/2.0-1;
		case 2: return x[1]-x[0]+1;
	}
	return 0;
}

double funk(int i,int j,double *x)
{
	switch(i)
	{
		case 0:
			switch (j)
			{
				case 0: return 1;
				case 1: return 1;
				case 2: return 0;
			}
			return 0;
		case 1:
			switch (j)
			{
				case 0: return -0.5;
				case 1: return 1;
				case 2: return 0;
			}
			return 0;
		case 2:
			switch (j)
			{
				case 0: return -1;
				case 1: return 1;
				case 2: return 0;
			}
			return 0;

	}
	return 0;
}

int cSold::Gauss()
{
	double Max;
```

```
	int Mi;
	for(int nm = 0; nm < m-1; nm++) {
		Max = A[nm][nm];
		Mi = nm;
		for(int i = nm; i < m; i++)
			if(abs(A[i][nm]) > abs(Max)) {
				Max = a[i][nm];
				Mi = i;
			}
		double buf;
		for(int j = nm; j < N; j++) { //переставляем строку
			buf = a[nm][j];
			a[nm][j] = a[Mi][j];
			a[Mi][j] = buf;
		}
		buf = f[nm];
		f[nm] = f[Mi];
		f[Mi] = buf;
		for(int k = nm + 1; k < N; k++) //делим текущую строку
			A[nm][k] /= A[nm][nm];
		f[nm] /= A[nm][nm];
		a[nm][nm] = 1;
		double koef;
		for(int i = nm + 1; i < N; i++) { //отнимаем
			koef = a[i][nm];
			for(int j = nm; j < n; j++)
				a[i][j] -= a[nm][j] * koef;
			f[i] -= f[nm] * koef;
		}
	}
	f[m] /= a[m][m];
	a[m][m] = 1;
	double* X = new double[n];
	double sum;
	for(int i = m; i >= 0 ; i--) { //обратный обход
		sum = 0;
		for(int j = i + 1; j < n; j++) {
			sum += X[j] * a[i][j];

		}
		X[i] = f[i] - sum;
	}
	memcpy(f,X,sizeof(double)*m);
	return 1;
}


void cSold::jacobi()
{
	A = new double*[m];
	for(int i=0; i<m; i++)
		A[i] = new double[N];
	for(int i=0; i<m; i++) {
		for (int j=0; j<N; j++)
			A[i][j] = funk(i,j,x);
	ff[i]=-F(i,x);
	}
}

void cSold::iskl()
{
	int j, minim;
	for (int i=0; i<m-N;i++) {
		for( j=1, minim=0; j<m-i;j++)
```

```
		{
			if(abs(ff[j])<abs(ff[minim]))
				minim=j;
		}
		for(int k=minim; k<m-i-1;k++)
		{
			for (j=0;j<N;j++)
				A[k][j]=A[k+1][j];
			ff[k]=ff[k+1];
		}
	}
}


void cSold::vvod()
{
	FILE *in;
	int k;
	in =fopen("parametrs.txt", "r");
	fscanf(in, "%i%i%i%lf%lf", &N,&m, &maxiter, &eps, &min);
	fclose(in);
	x = new double[N];
	x1 = new double[N];
	ff = new double[N];
	delta_x = new double[N];
	in = fopen("nach.txt", "r");
	for( k=0; k<N; k++)
	fscanf(in, "%lf", &x[k]);
	fclose(in);
}

double cSold::norma_f(double *x)
{
	double n = 0.;
	for(int i=0; i<m; i++)
	n+=F(i,x)*F(i,x);
	n=sqrt(n);
	return n;
}


void cSold::summa(double *x1,double *x, double *y, double a)
{
	for (int i=0; i<N; i++)
		x1[i]=x[i]+a*y[i];
}

void cSold::calc()
{
	FILE *v;
	FILE *out;
	v=fopen("table.txt" ,"w");
	int r;
	int t=0;
	int i=1;
	vvod(); //ввод параметров итерационного процесса, начального приближения, размерности
СНАУ
	normaf=norma_f(x);
	nev=1;
	for (r=0;r<maxiter&&t==0&&nev>eps&&i==1;r++) {
		fprintf(v, "%.16lf\t%.16lf\n",x[0], x[1] );
		jacobi();
		iskl();
		t=Gauss();
```

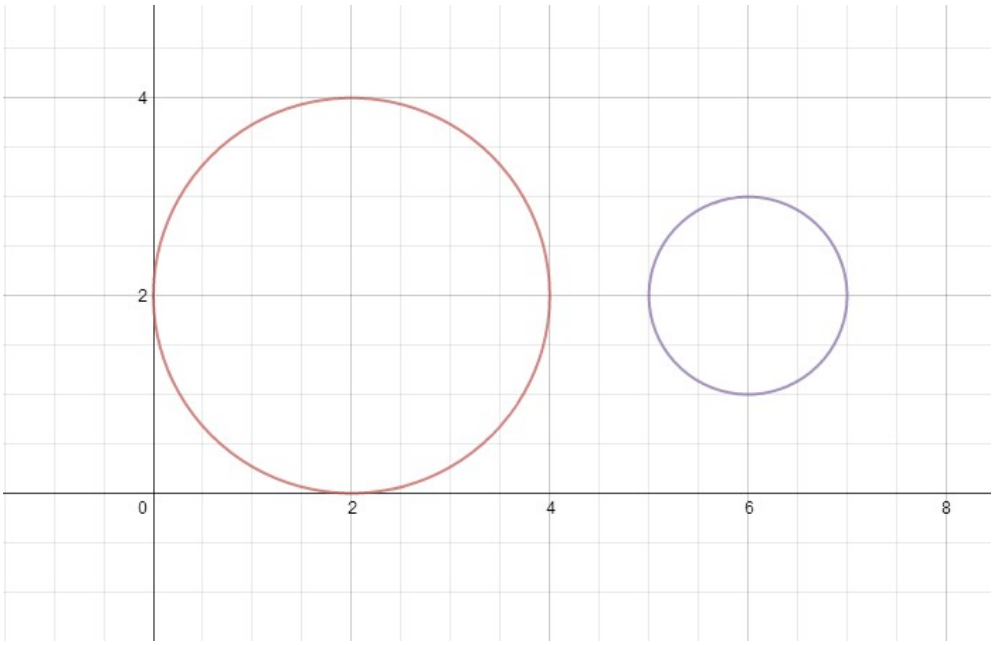
```
		if (t==0) {
			for(beta=1, i=0; beta>min&&i==0;) {
				summa(x1, x, delta_x, beta);
				if (norma_f(x1)>norma_f(x))
					beta=beta/2;
				else
					i=1;
			}
			if(i==1) {
				nev=norma_f(x1)/normaf;
				summa(x, x1, x1, 0);
			}
		}
		else
			printf ("SLAU hasnt result!!!");
	}
	out=fopen("result.txt", "w");
	fprintf(out, "nev = %.1e\n", nev);
	fprintf(out, "beta = %f\n", beta);
	fprintf(out, "kolvo iter = %i", r);
}
```

## 5. Исследования

**Окружности не пересекаются.**

$(x-2)^2+(y-2)^2=4$

$(x-6)^2+(y-2)^2=1$




Приближение (1,0)

**1 и 2**

Результат: 4.3750000000000000 2.0000000122731696

Норма F: 2.49

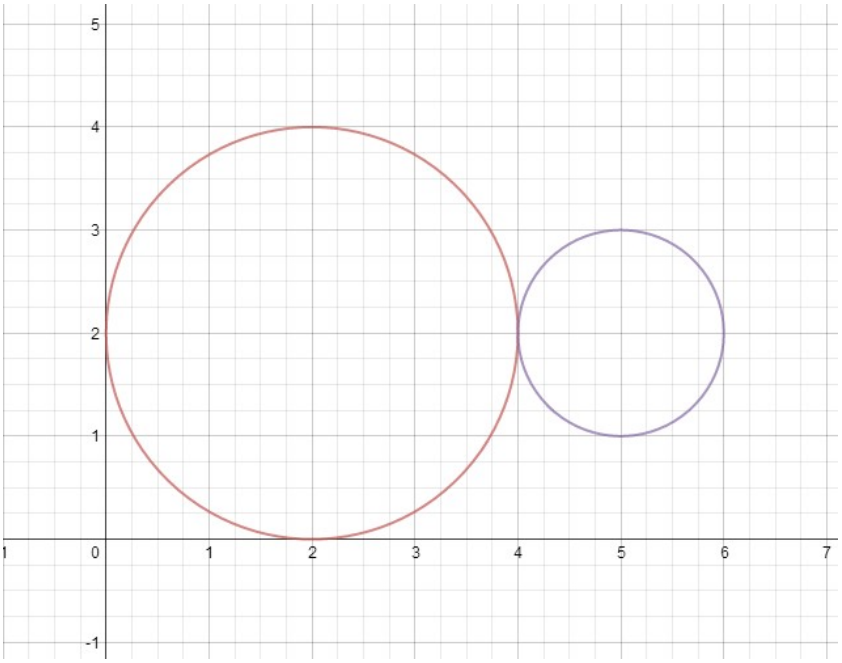
beta = 2.2737367544323210E-013

**5**

Результат: 4.374849 2.069503

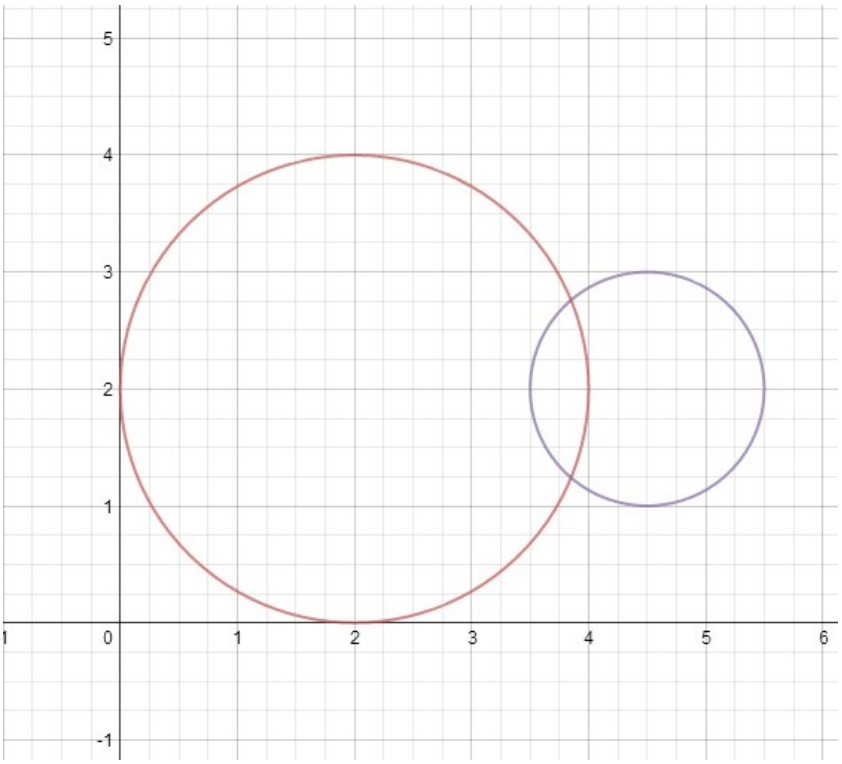
Норма F: 2.42
beta = 1.1641532182693480E-013

**Окружности пересекаются в одной точке.**
$(x-2)^2+(y-2)^2=4$
$(x-5)^2+(y-2)^2=1$




Приближение (-1,1)
**1 и 2**
Результат: 4.000000 1.999884
Норма F: 3.4e-008
beta = 1
**5**
Результат: 4.000000 1.999359
Норма F: 1.32e-006
beta = 1

**Окружности пересекаются в двух точках.**
$(x-2)^2+(y-2)^2=4$
$(x-4.5)^2+(y-2)^2=1$




Приближение (1,1)

**1 и 2**

Результат: 3.850000 1.240066

Норма F: 9.69e-011

beta = 1

**5**

Результат: 3.850000 1.240066

Норма F: 2.42e-010

beta = 1

Приближение (4,4)

**1 и 2**

Результат: 3.850000 2.759934

Норма: 8.88e-016

beta = 1

**5**

Результат: 3.850000 2.759934

Норма: 8.88e-016

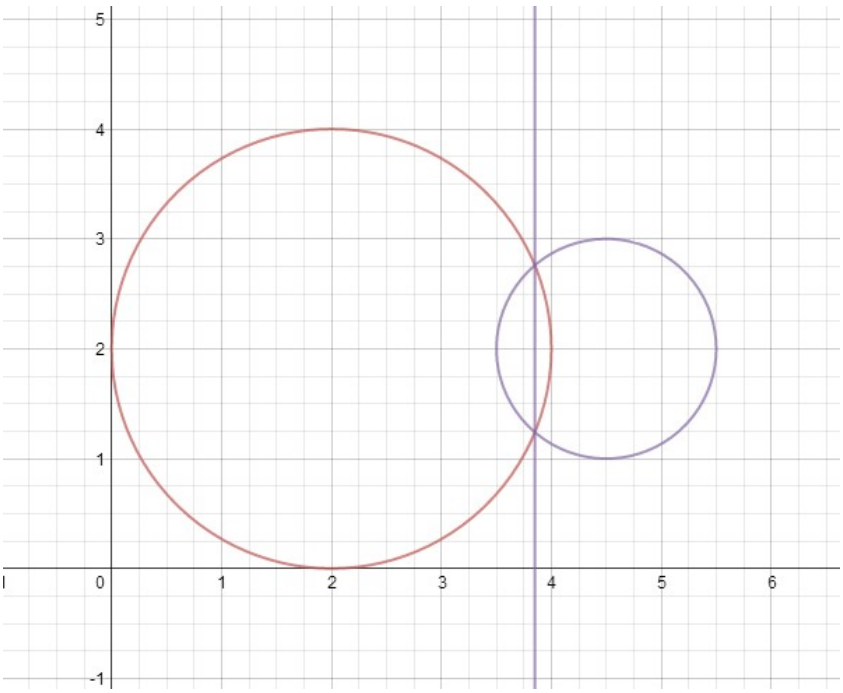
beta = 1

**Добавление уравнения прямой**

$(x-2)^2+(y-2)^2=4$

$(x-4.5)^2+(y-2)^2=1$

x=3.85




Приближение (1,4)

**1 и 2**

Результат: 3.850000 1.240065

Норма: 6.6974315024313530E-013

beta = 1

**5**

Результат: 3.850000 1.240063

Норма: 1.3467691922552350E-013

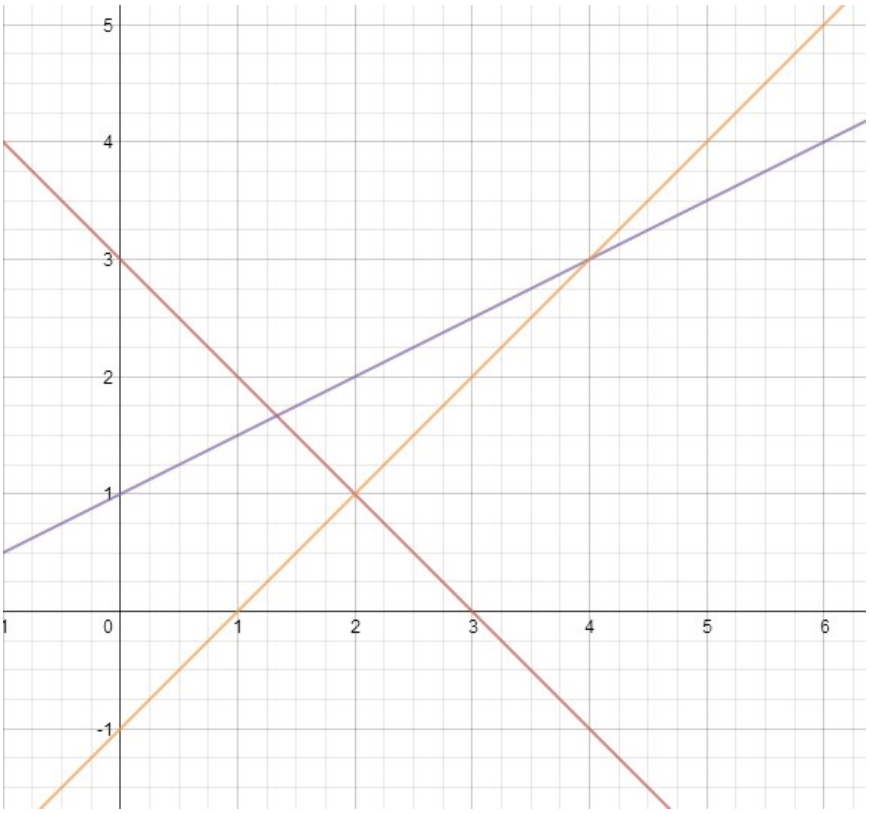
beta = 1

**Три прямые.**

y=3-x

y=x/2+1

y=x-1




**1 и 2**

Приближение (1,1)

Норма:

beta = 0

| x | y |
|---|---|
| 1.0000000000000000 | 1.0000000000000000 |
| 2.0000000000000000 | 1.0000000000000000 |
| 2.1250000000000000 | 1.1250000000000000 |
| 1.7291666666666667 | 1.3958333333333333 |
| 1.7646484375000000 | 1.4208984375000000 |

Приближение (2, 3)

beta = 0

| x | y |
|---|---|
| 2.0000000000000000 | 3.0000000000000000 |
| 2.0000000000000000 | 1.0000000000000000 |
| 2.1250000000000000 | 1.1250000000000000 |
| 1.7291666666666667 | 1.3958333333333333 |
| 1.7646484375000000 | 1.4208984375000000 |

**5**

Приближение (1,1)

Норма: 0.5239017436456296
beta = 2.2737367544323210E-013

| x | y |
|---|---|
| 1.0000000000000000 | 1.0000000000000000 |
| 1.9000000000000000 | 1.1000000000000000 |
| 2.0187500000000000 | 1.2125000000000000 |
| 1.6345703125000000 | 1.2196289062500000 |
| 1.8518406617493430 | 1.4776781433746720 |
| 1.7201755118068040 | 1.1895536230258000 |
| 1.8446152897410870 | 1.3355902786026420 |
| 1.8341780283715920 | 1.3306581310068930 |
| 1.8346855726103750 | 1.3309124623903560 |
| 1.8345752468435650 | 1.3308573344609220 |
| 1.8345713439892470 | 1.3308553830678980 |
| 1.8345723339038080 | 1.3308558780273110 |
| 1.8345721152592500 | 1.3308557687051660 |

## 6. Вывод

В различных условиях метод Ньютона может как сходиться, так и расходиться. Кроме того, при решении может получиться вырожденная СЛАУ, в таком случае может помочь смена начального приближения. Если у системы несколько решений, то будет получено решение, ближайшее к начальному приближению (но не всегда). Добавление к системе прямой, которая проходит через корни системы не ухудшает сходимость метода. Сходимость в системе из трёх прямых, образующих не вырожденный треугольник не зависит от начального приближения и сходится к некоторой точке внутри треугольника.