МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РФ

ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Отчет

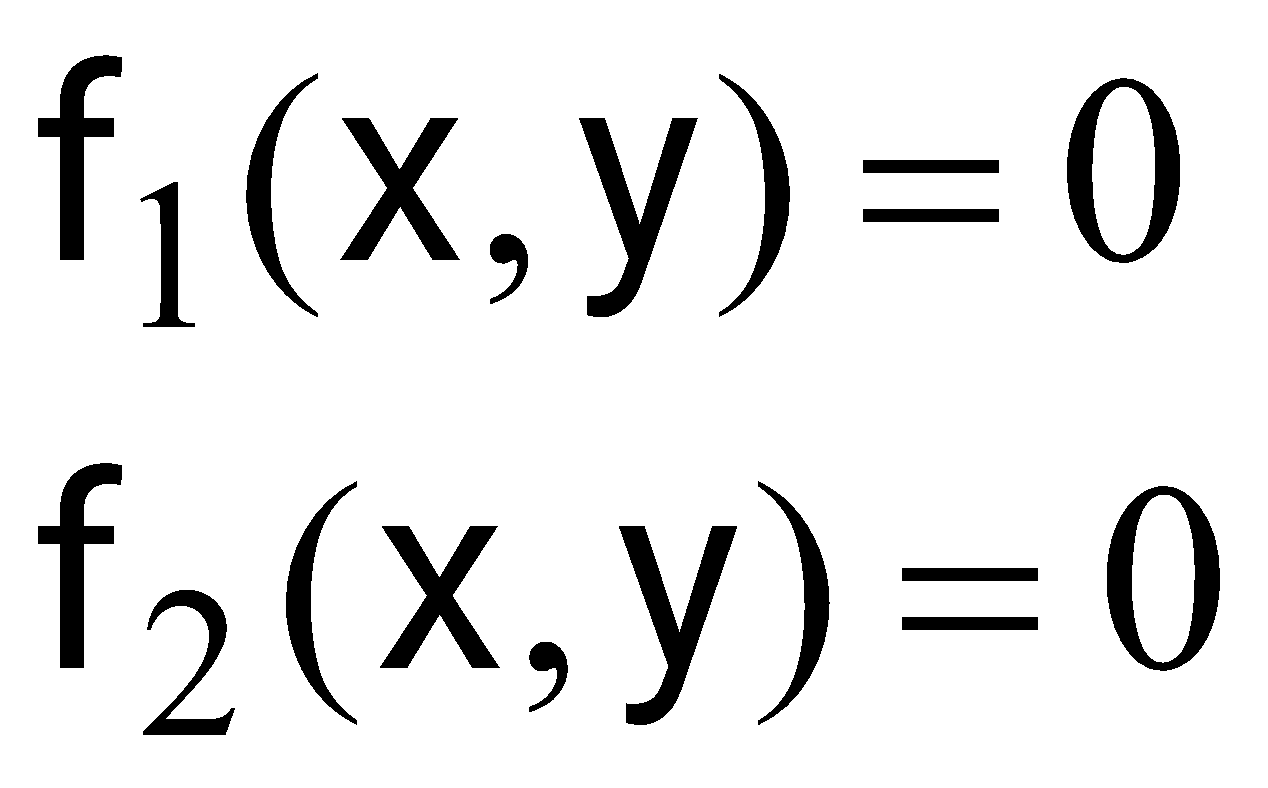
По лабораторной работе №4 «Решение нелинейных уравнений»



Пермь 2022

**Задание**

# Найти корни системы нелинейных уравнений



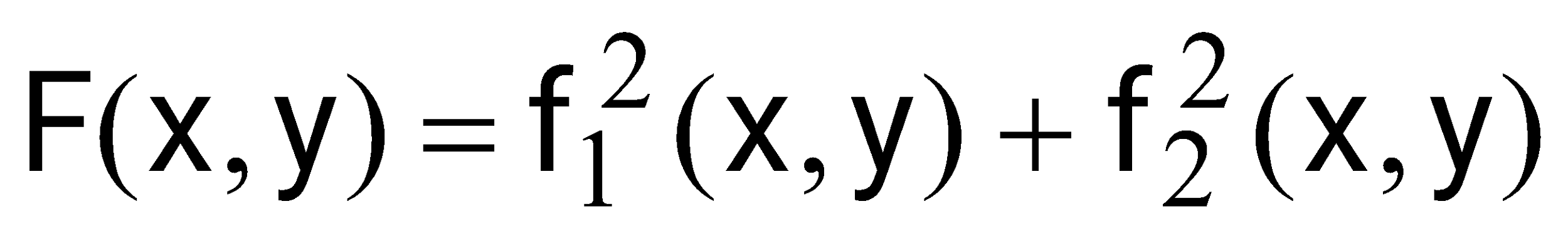
с погрешностью ε=10-4.

1. Приближенно определить корни геометрически.
2. Уточнить корни методом:

- Ньютона; (с погрешностью ε=10-12)

- простой итерации;

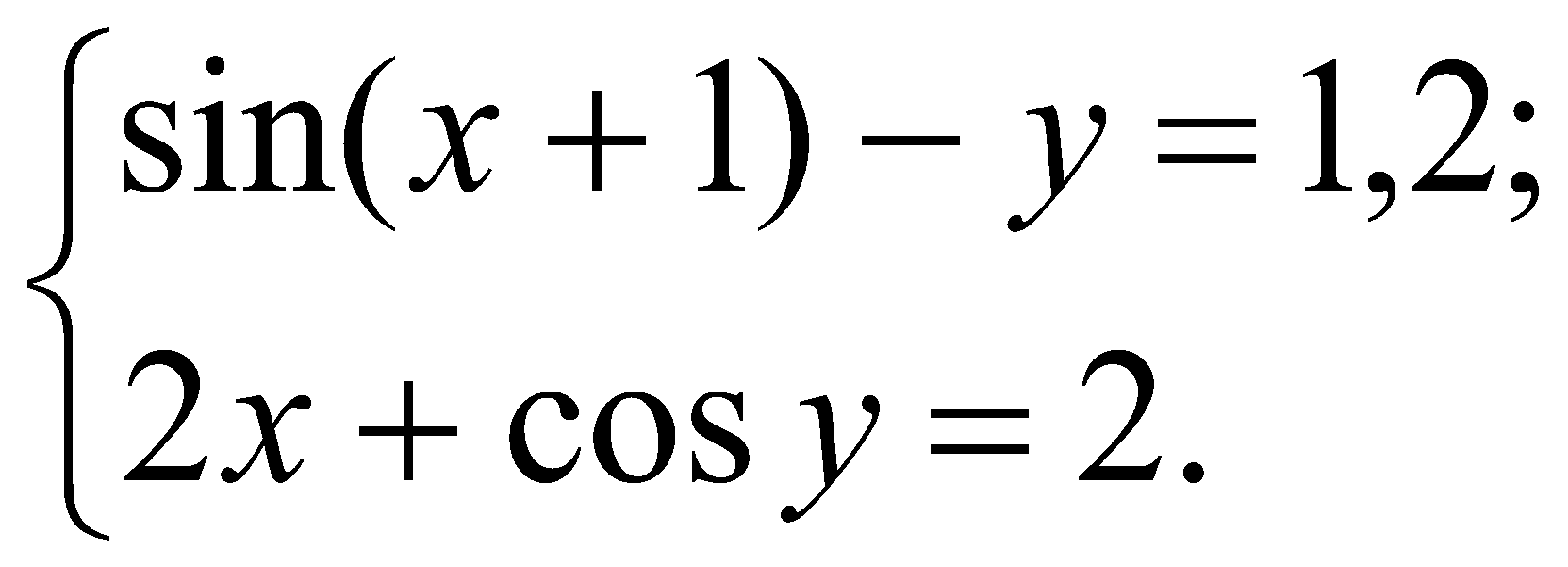
- градиентного спуска, сведя к нахождению минимума функции



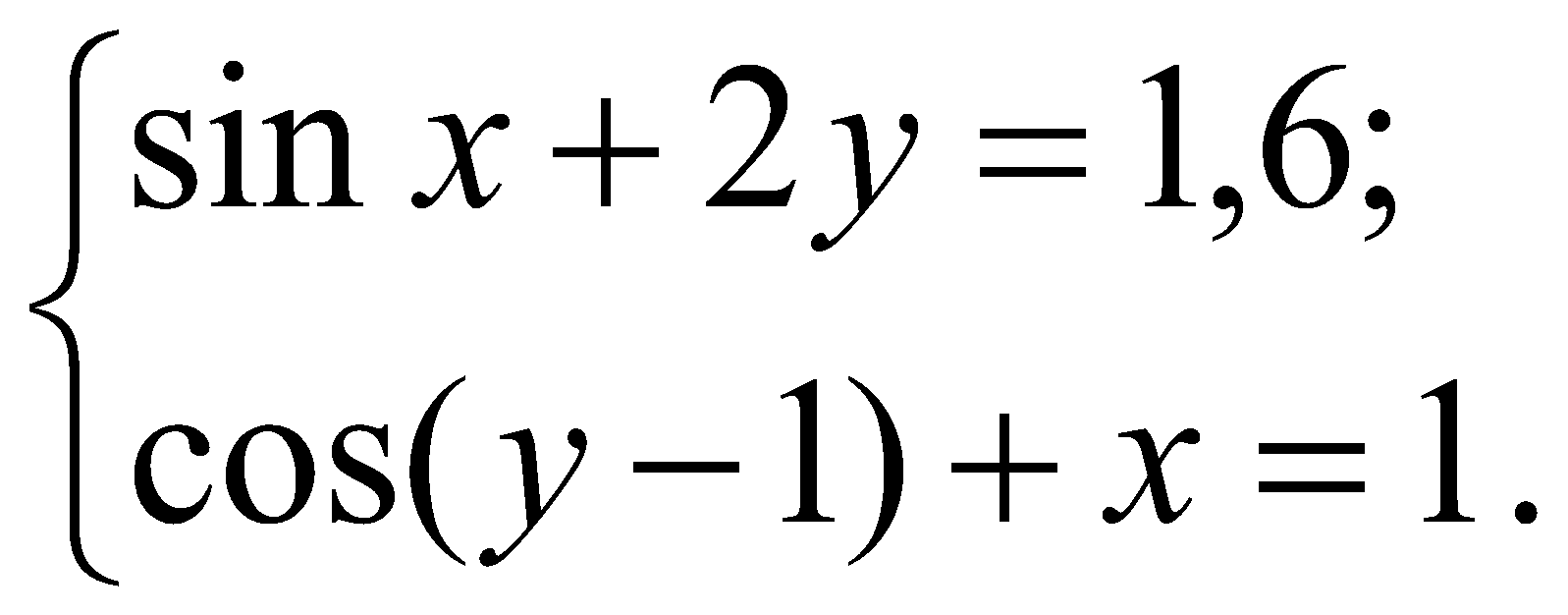
1. Провести анализ скорости сходимости и точности решения рассмотренными методами.

**Исходные данные**

1 вариант:

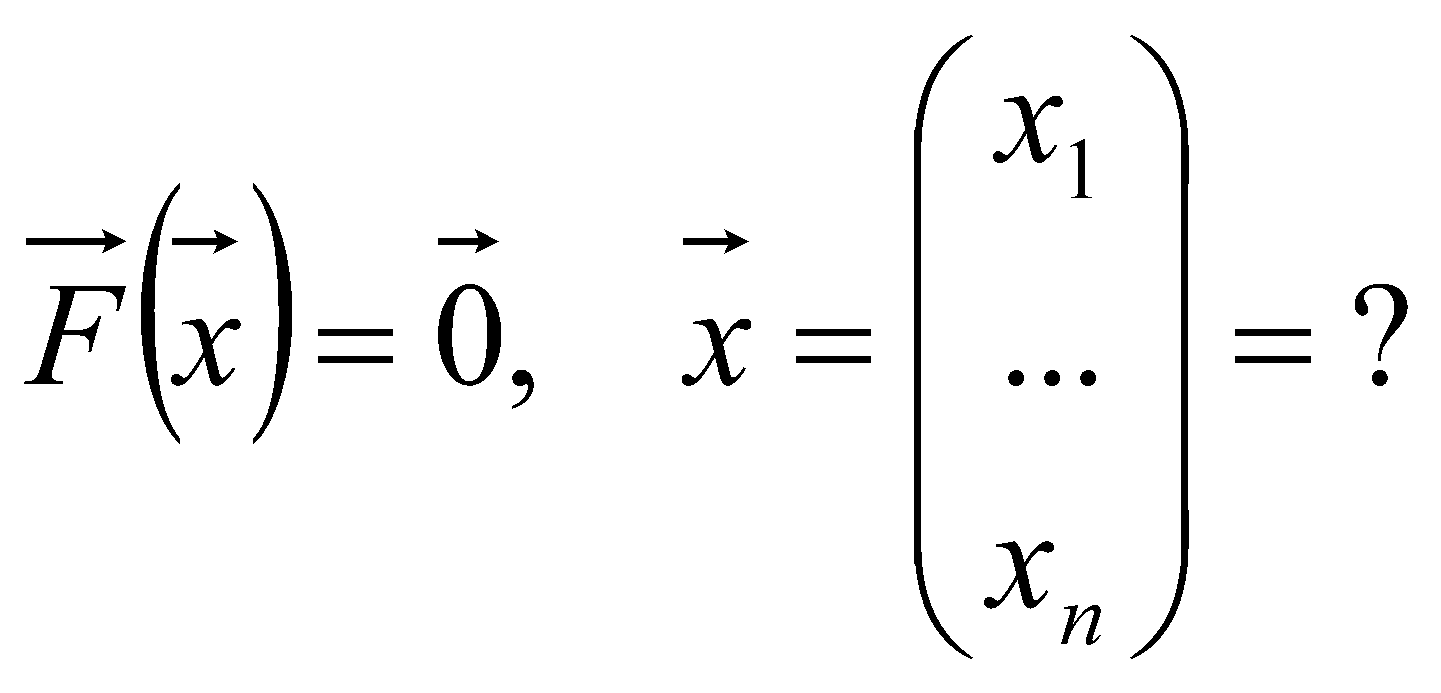


23 вариант:



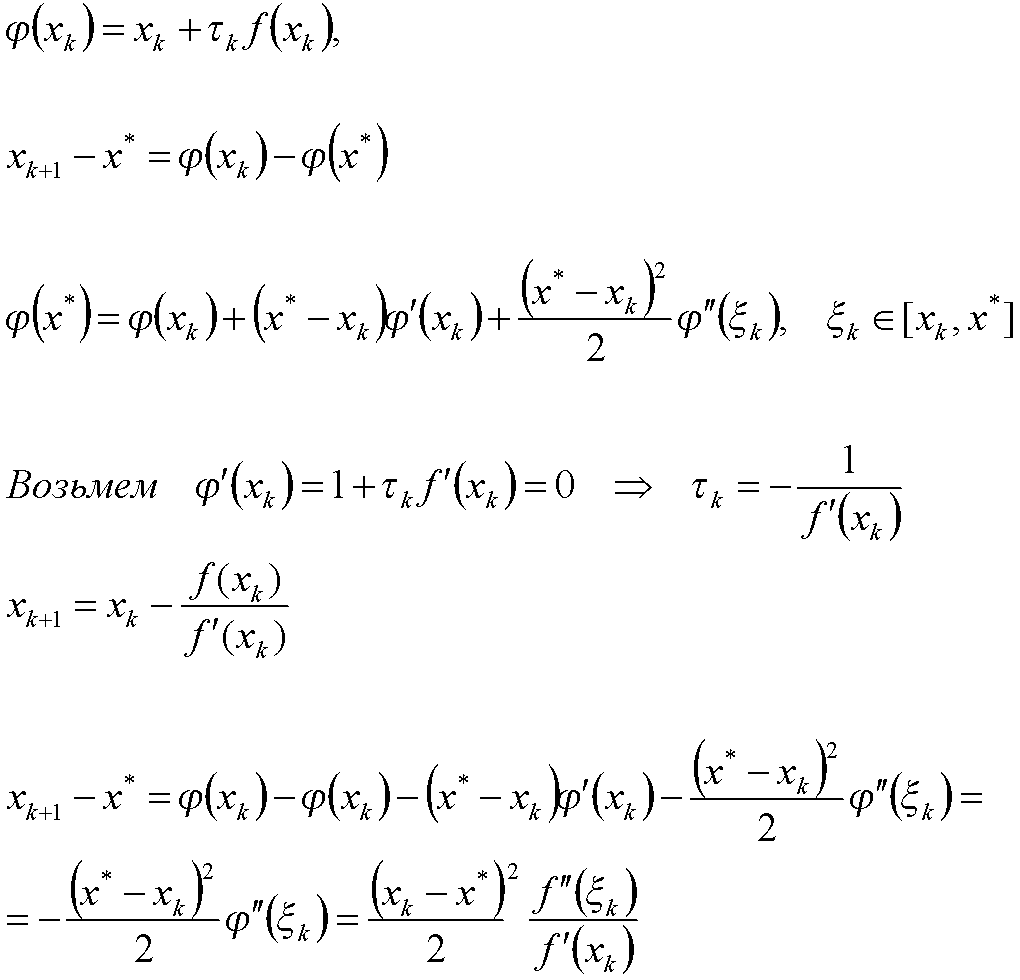
**Теоретическая справка**

*Задача:*

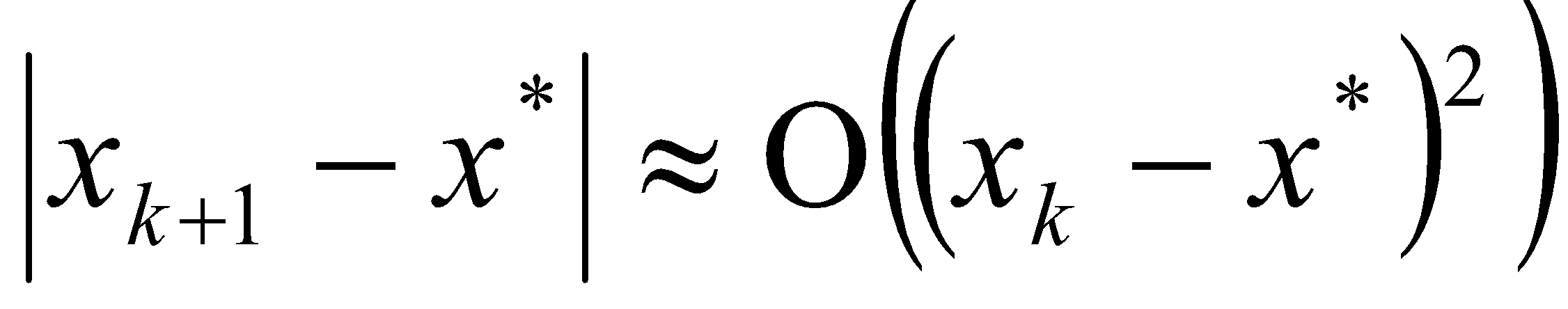
******

при *n=1:*  *f(x)=0, x=?*

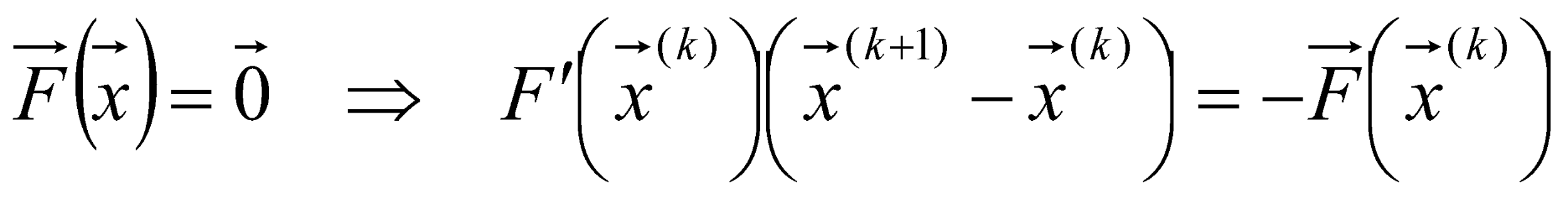
1. ***Метод Ньютона***

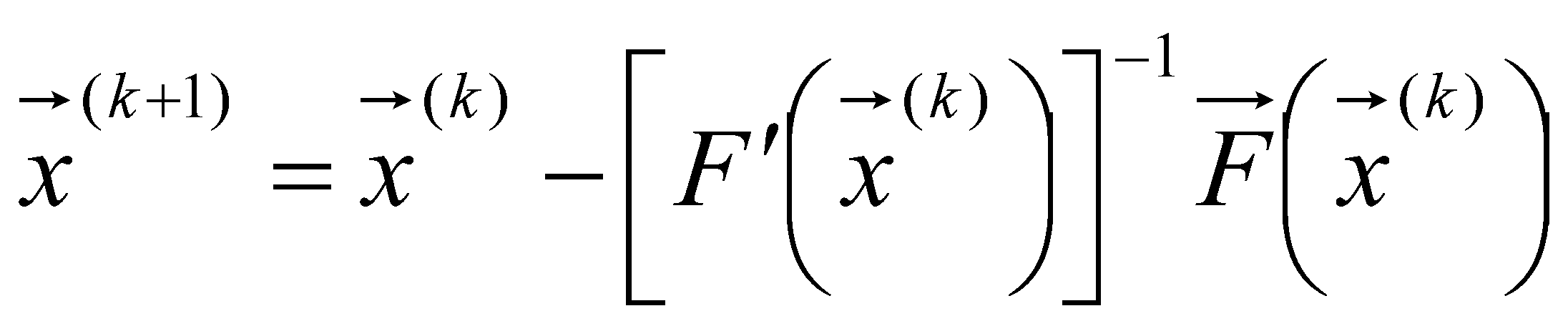
******

*Таким образом, имеет место квадратичная скорость сходимости:*

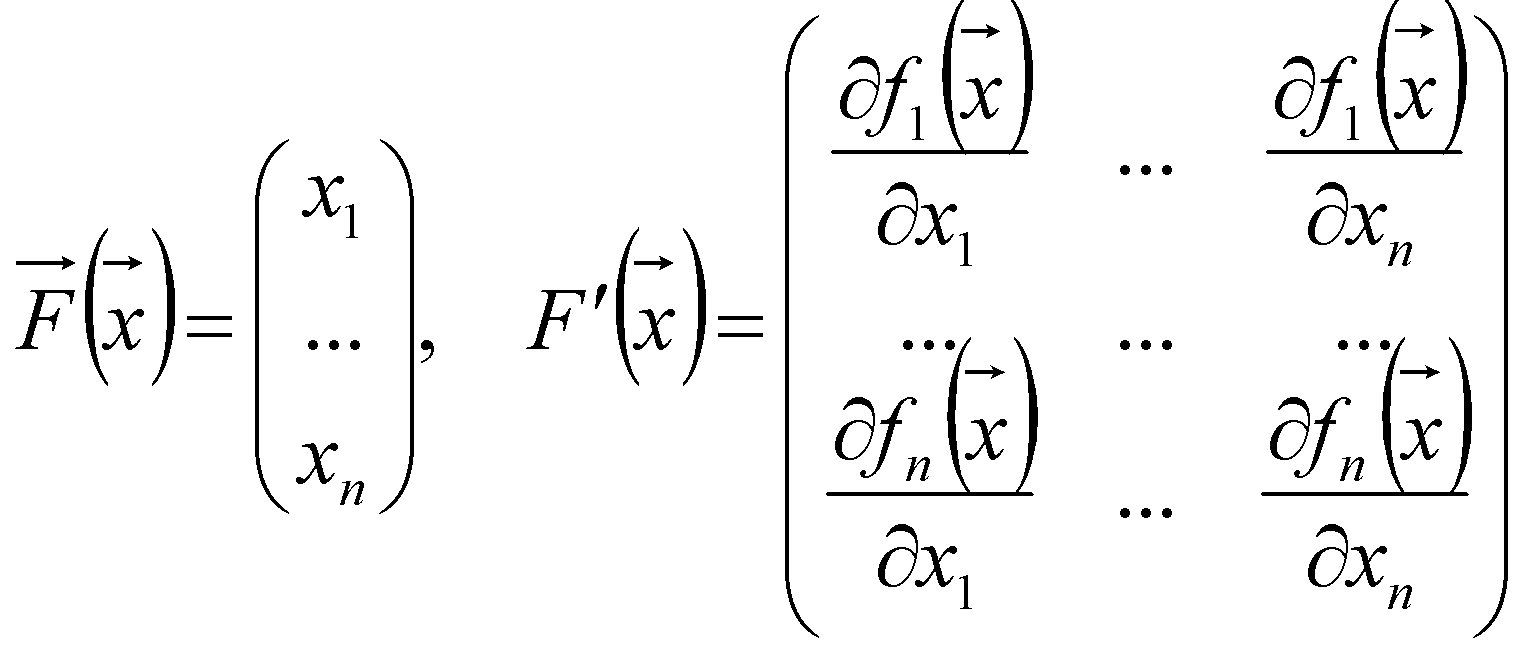
**

**Многомерный случай (*n>1*)**

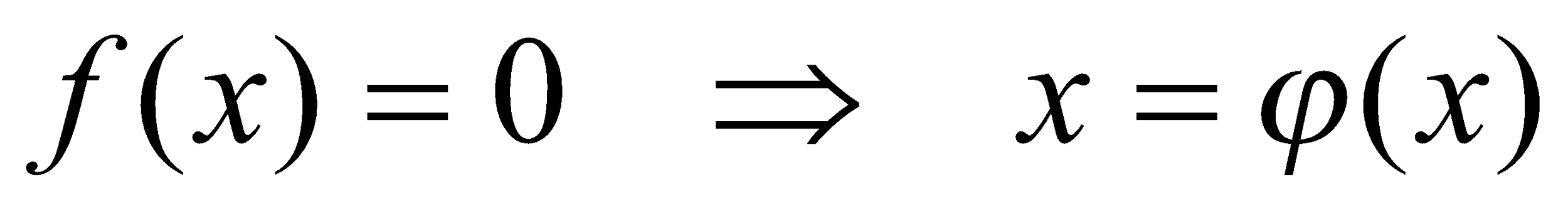
***-*** СЛАУ

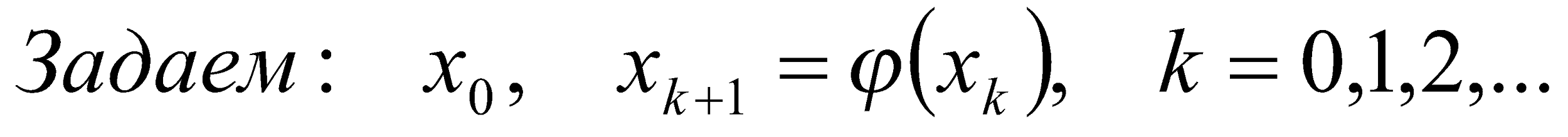
,

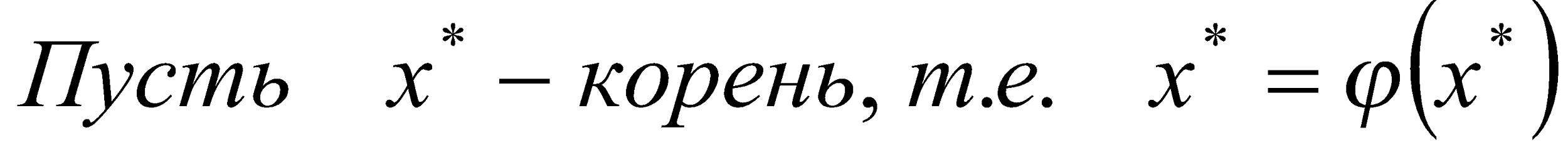
где

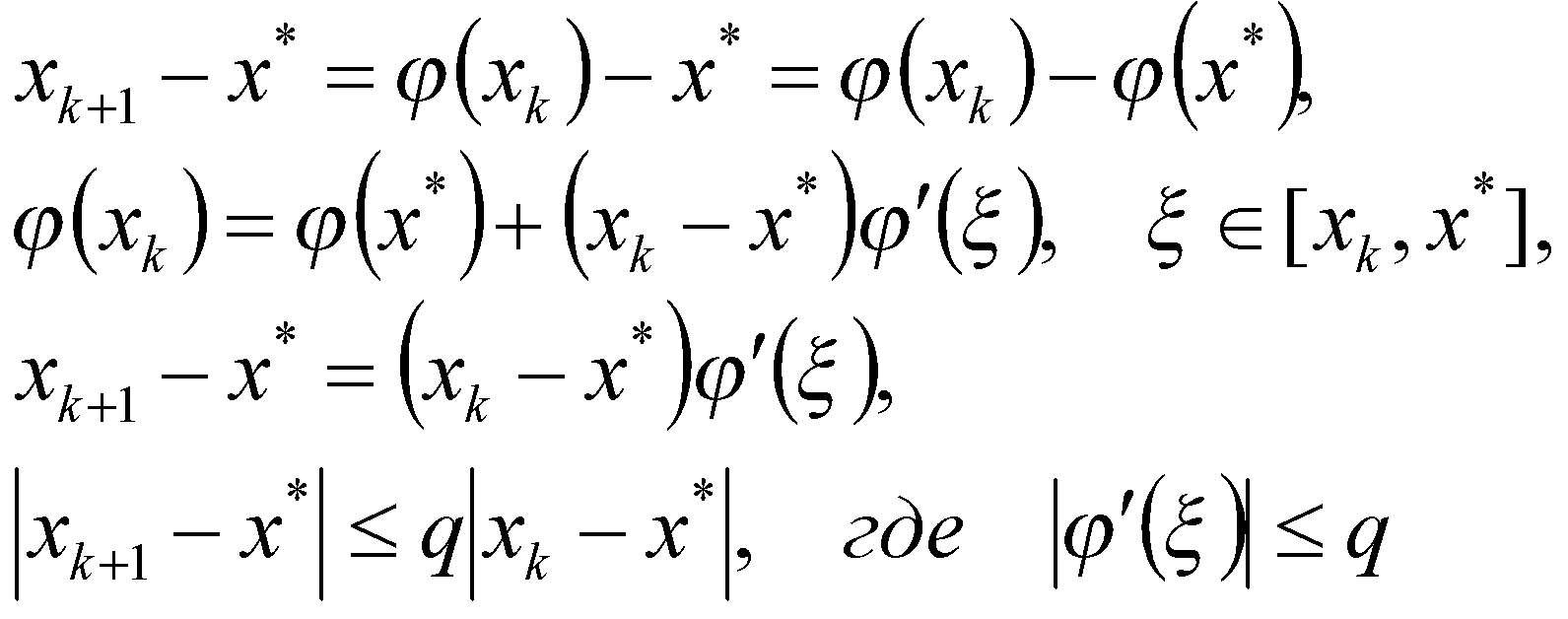


1. ***Метод простой итераци****и*

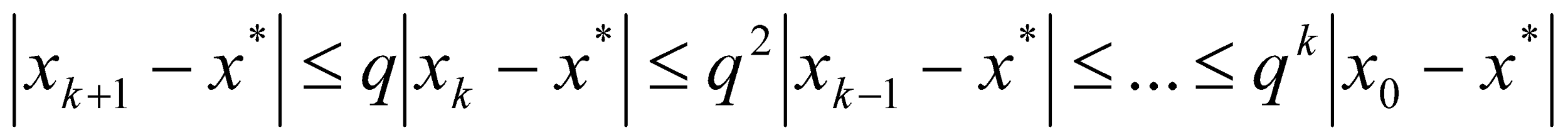
**



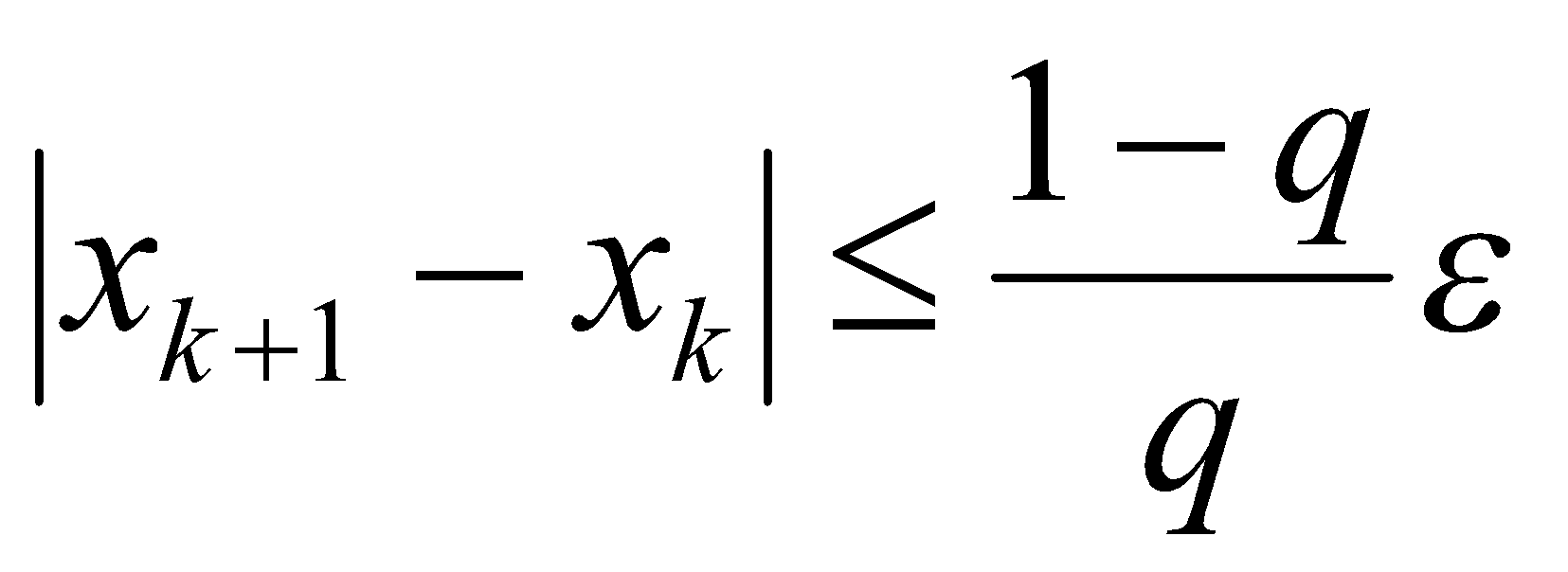
**



*Условие сходимости: q<1.*



*Критерий сходимости при точности ε :*

**

| Рис.2 | *Односторонняя сходимость* |
| --- | --- |

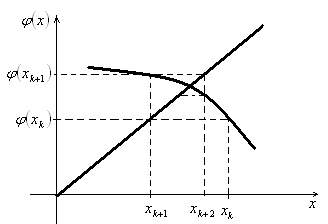
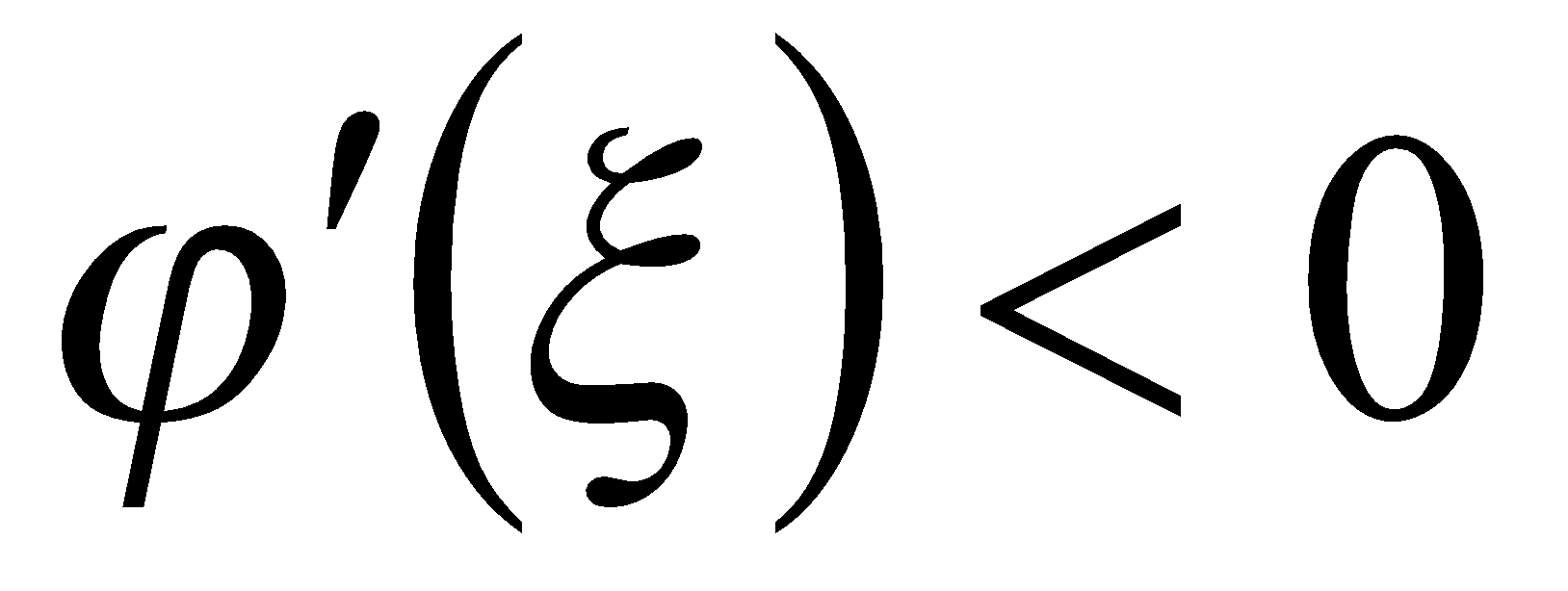
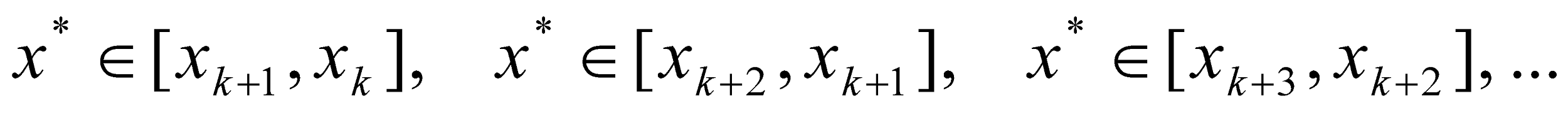
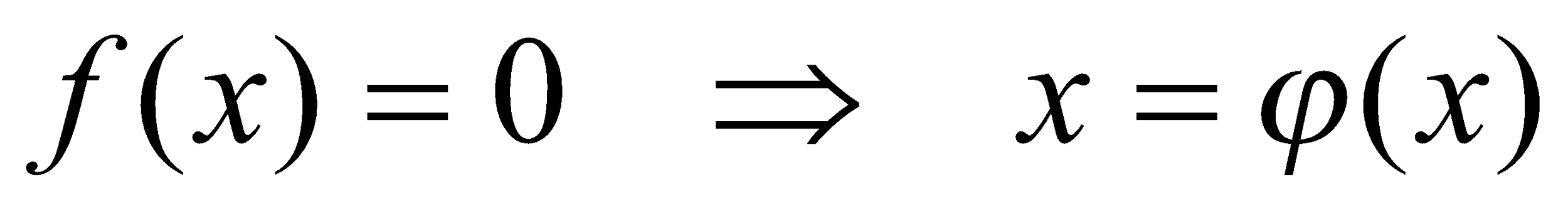


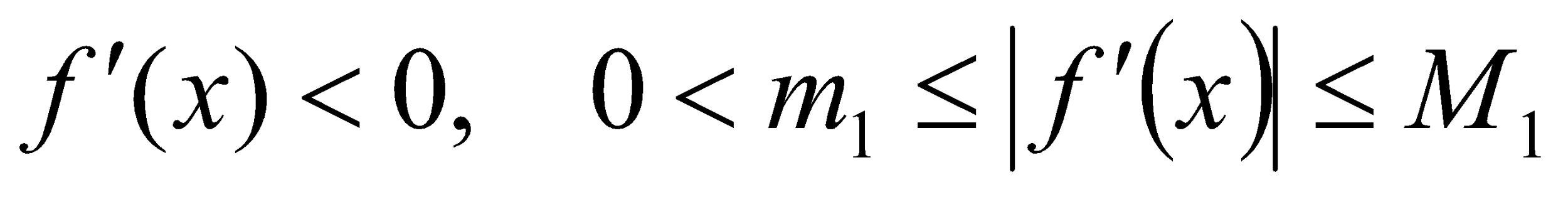
Рис.3

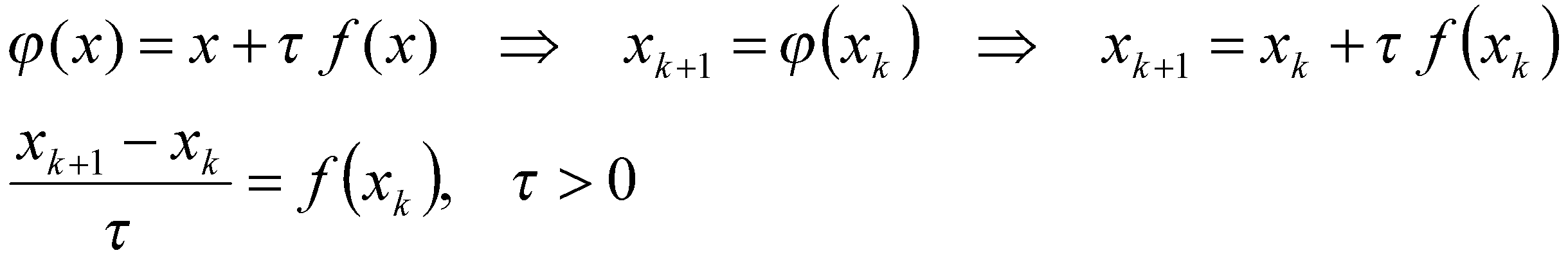
*Двусторонняя сходимость *

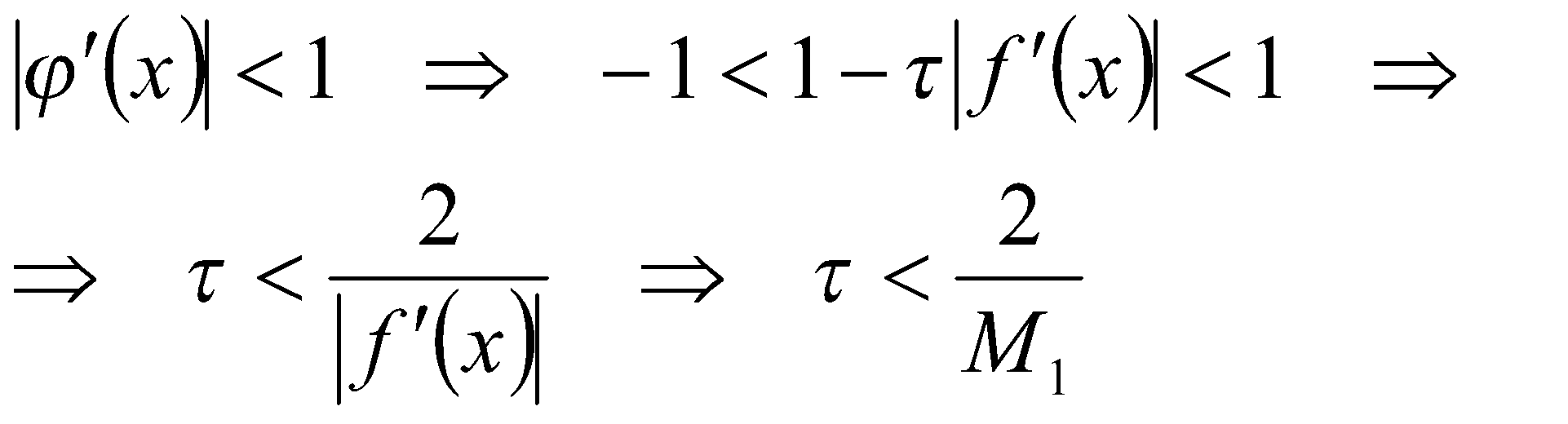
**

***Преобразование уравнения к виду, позволяющему применять метод итераций***

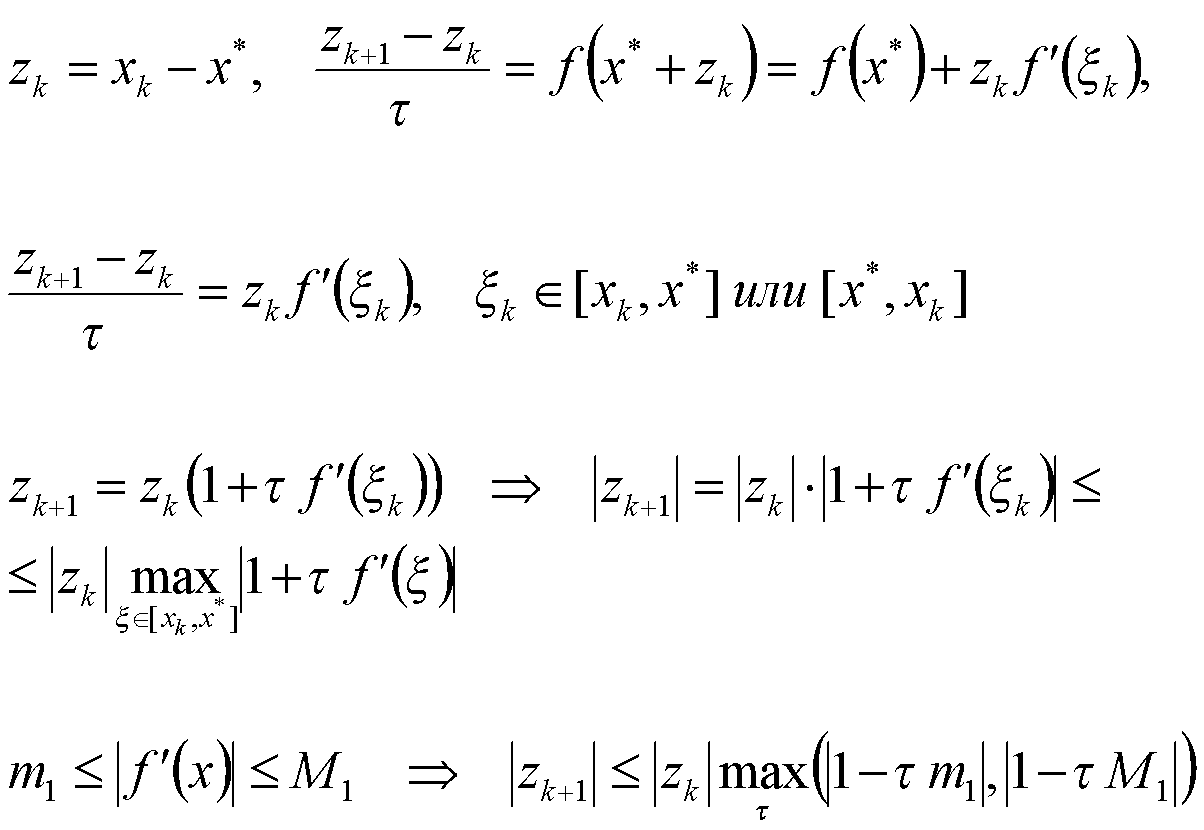
******

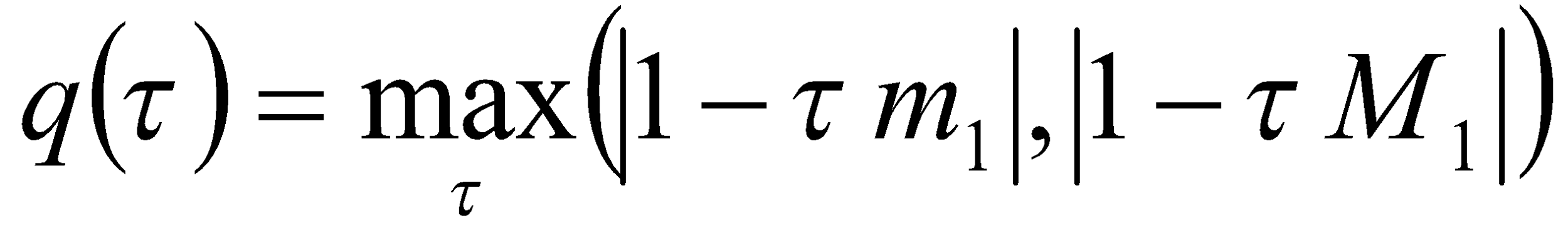
*Пусть *

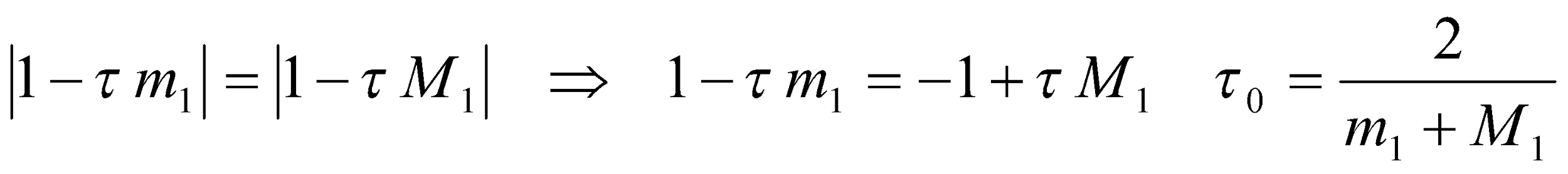
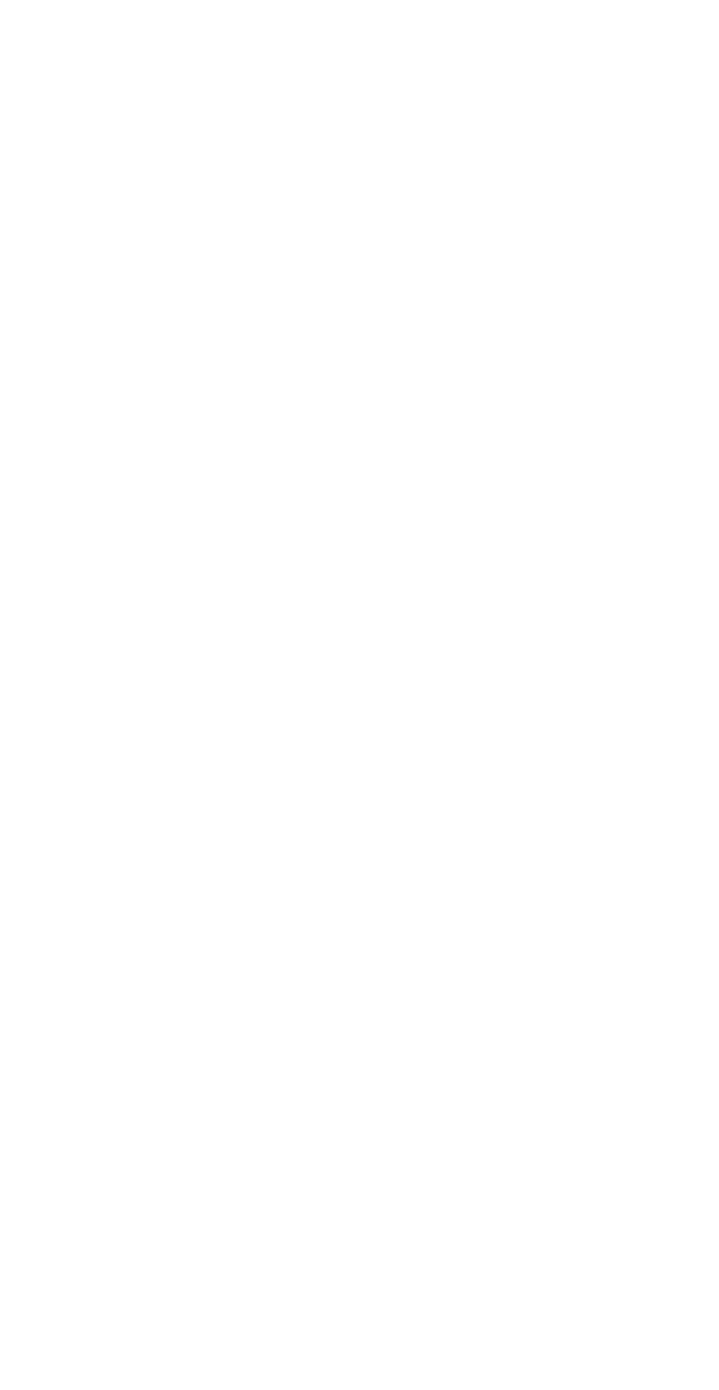
******

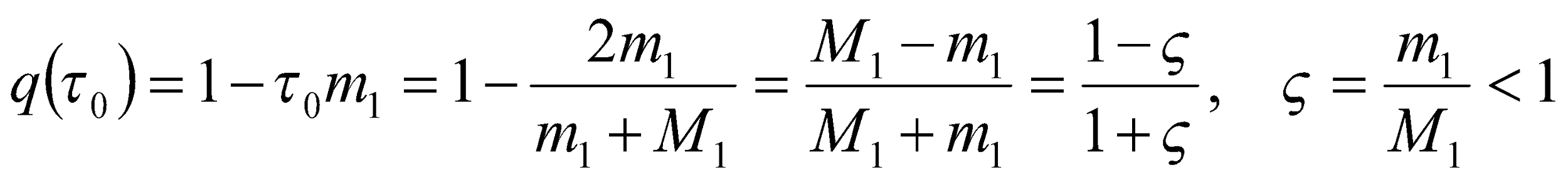
******

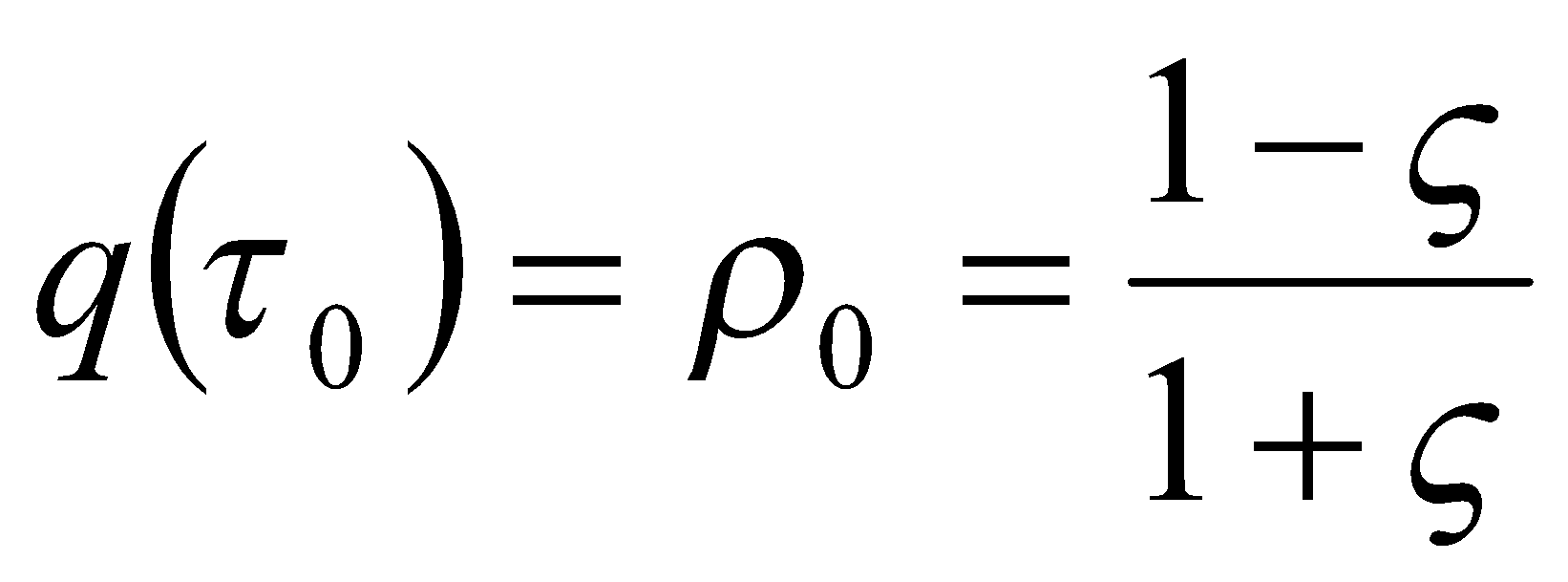
*Выбор оптимального значения τ.*

**

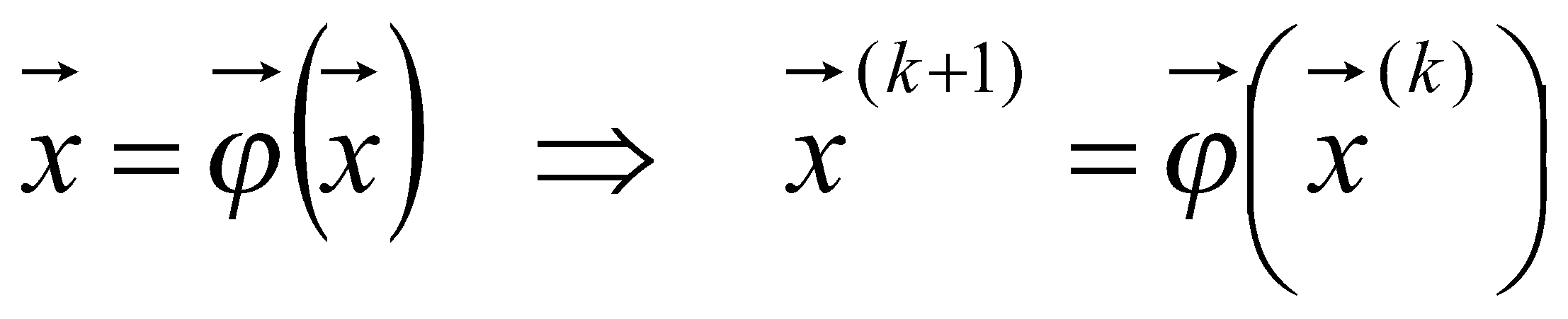
**

**

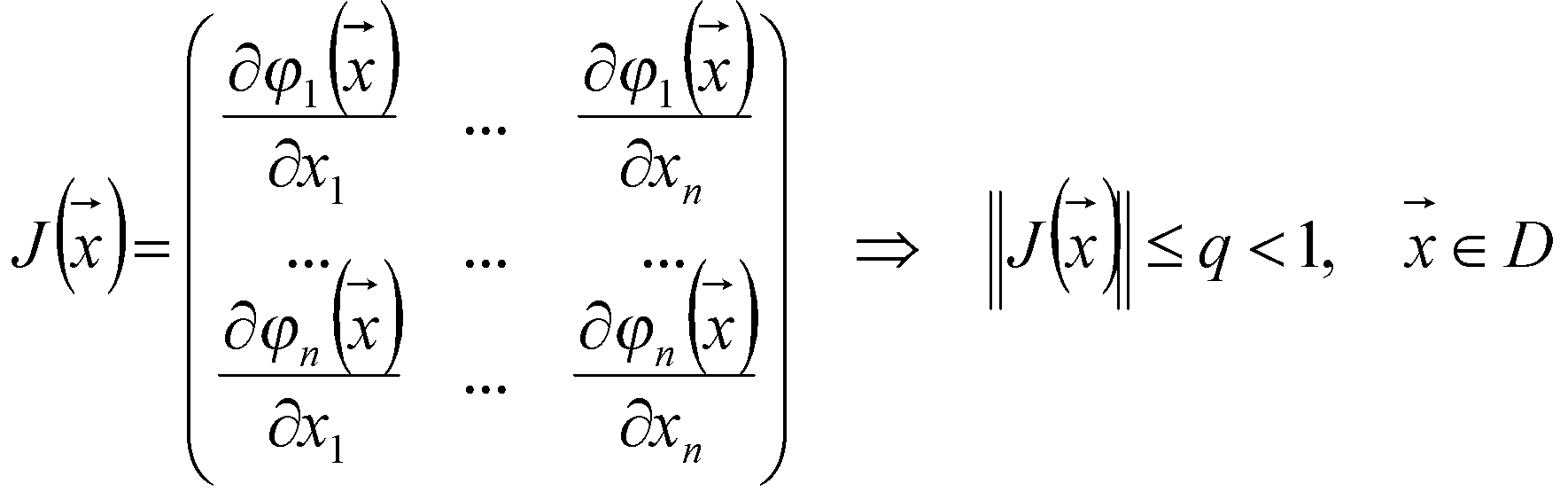
**



**Многомерный случай (*n>1*)**

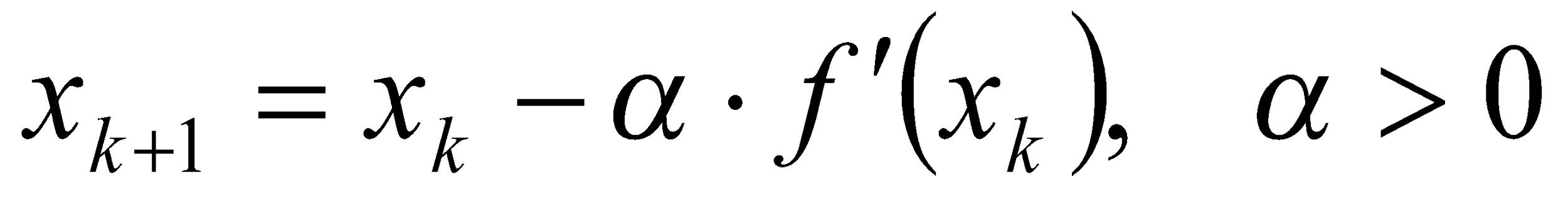
******

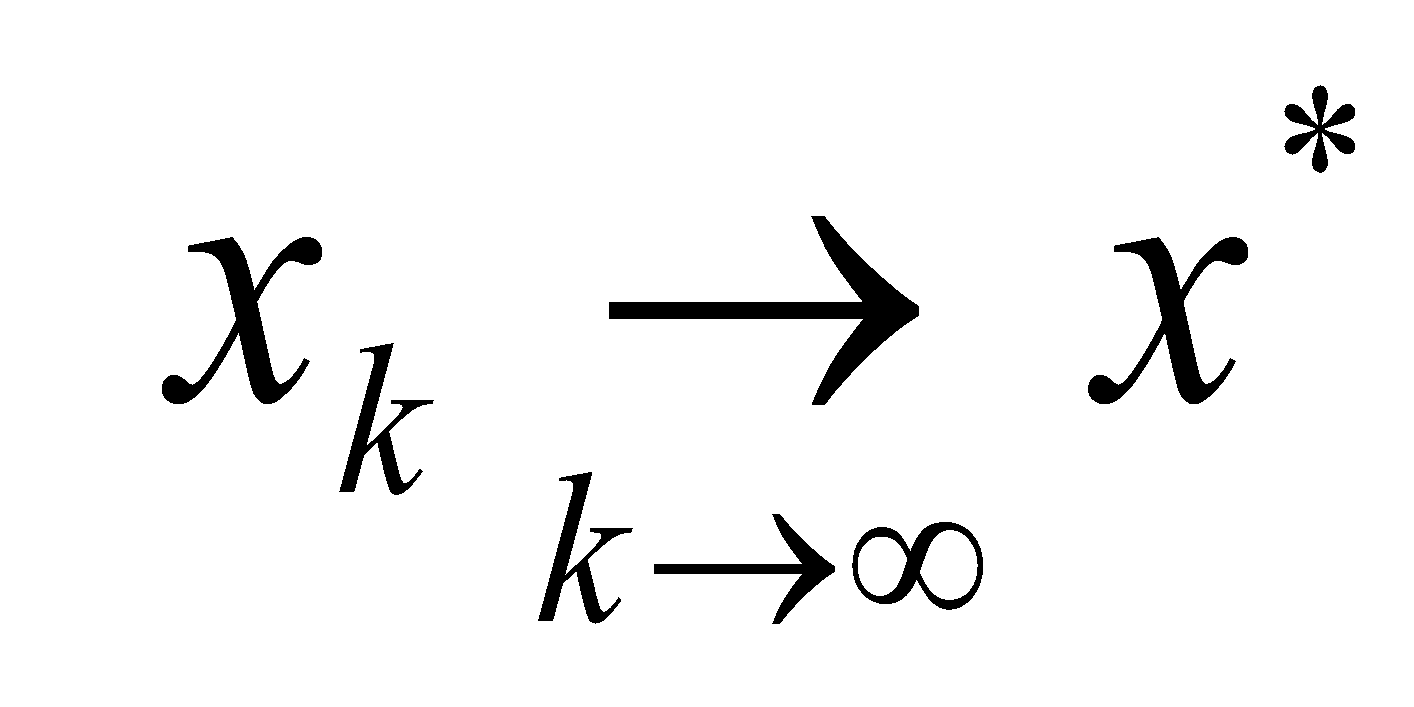
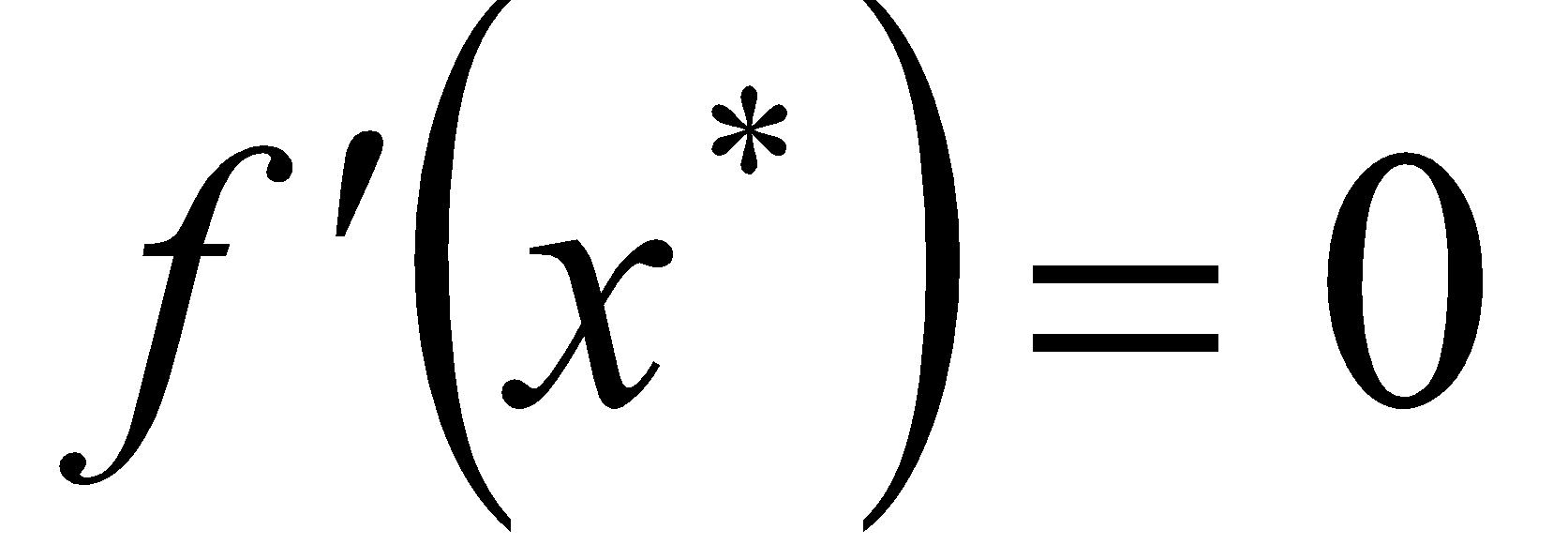
***Условие сходимости***

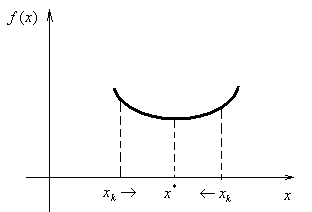
******

*Принцип сжатых отображений.*

1. ***Метод градиентного спуска***

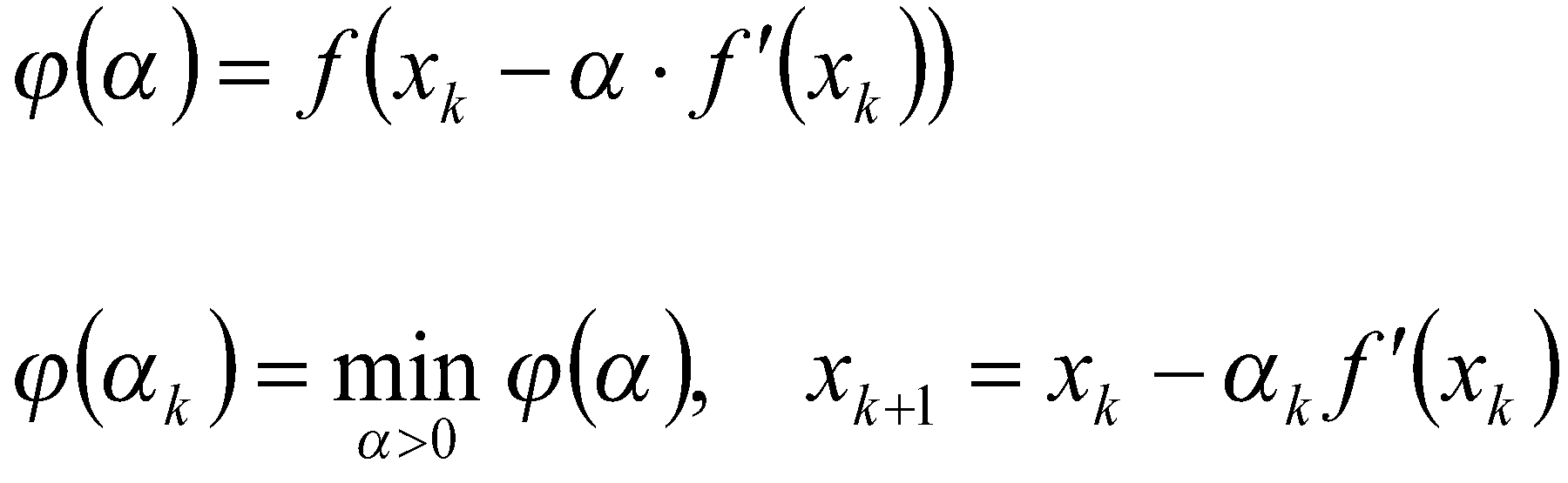
******

При  имеем .

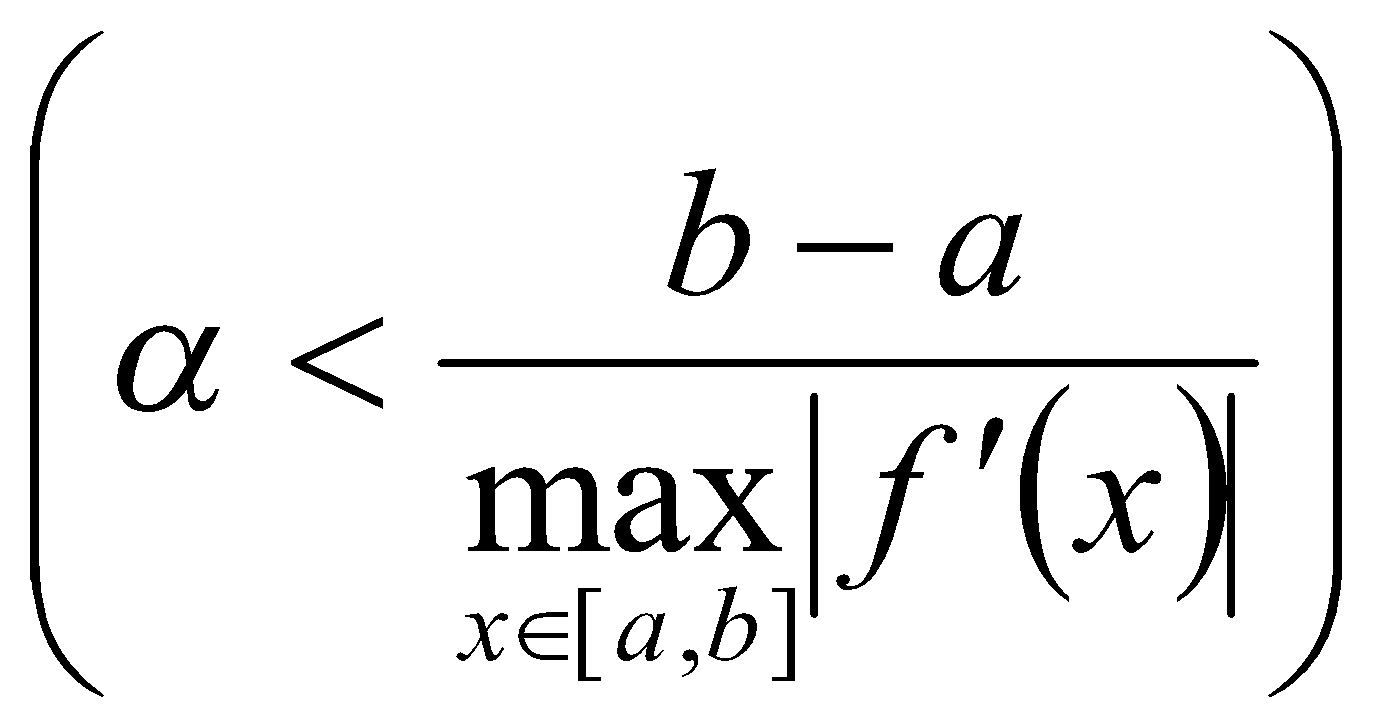


Выбор итерационного параметра *α.*

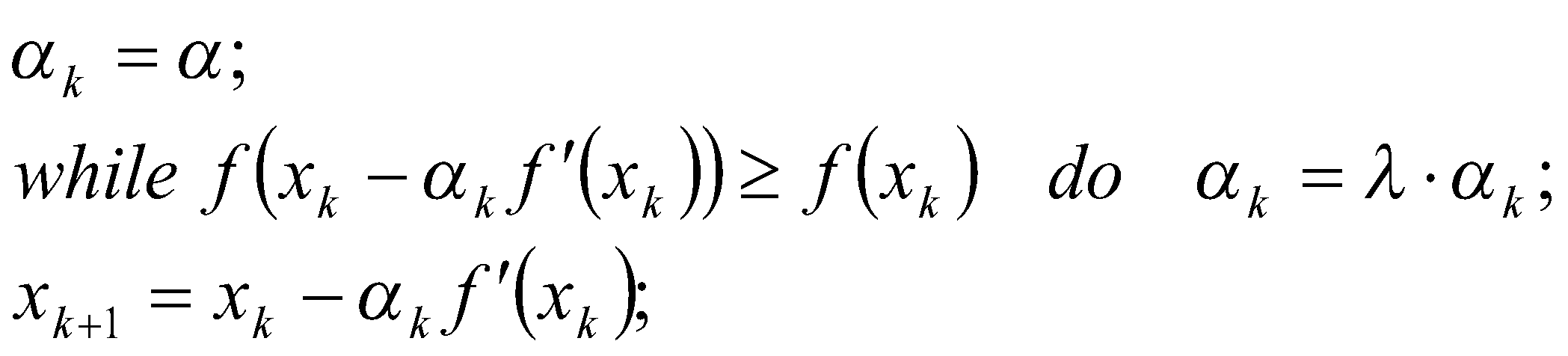
1. *Метод наискорейшего спуска*



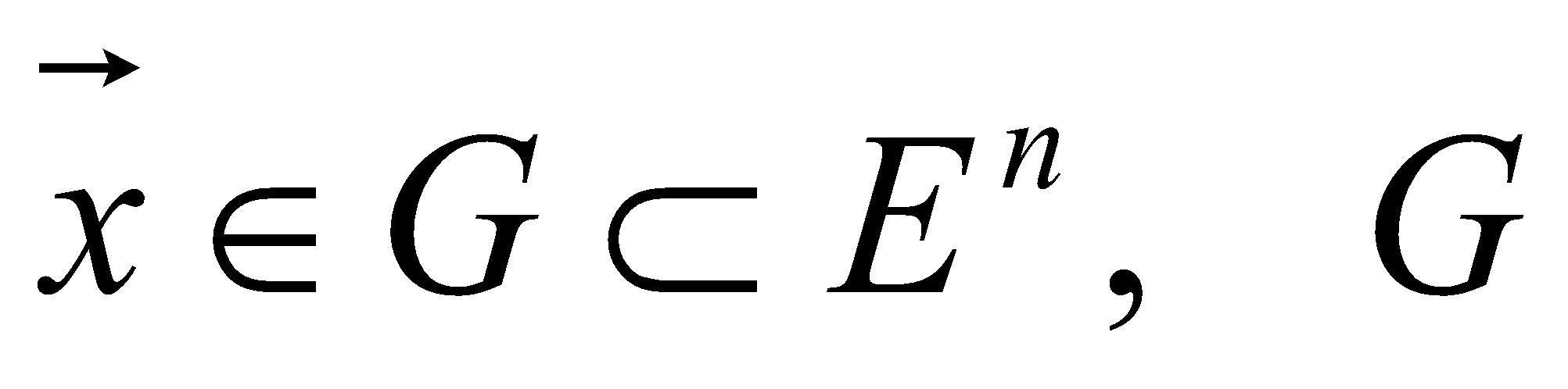
1. «Адаптивный метод»

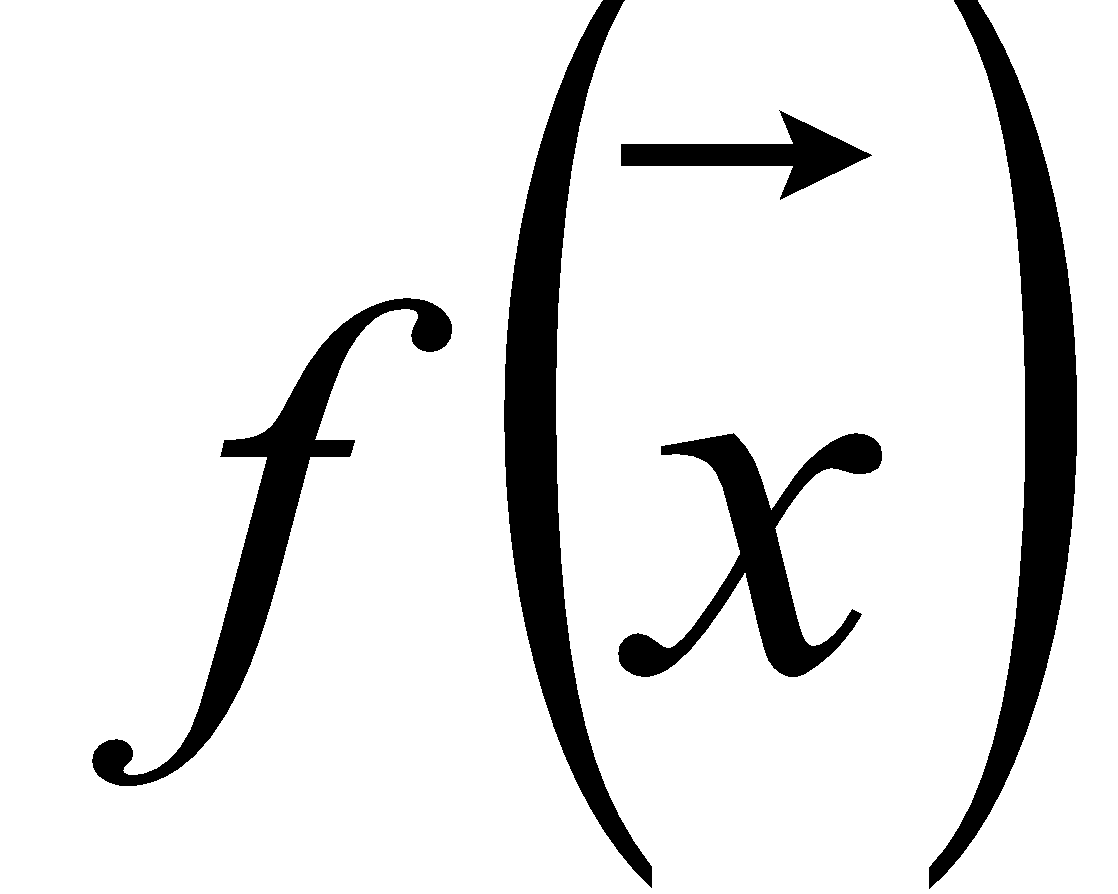
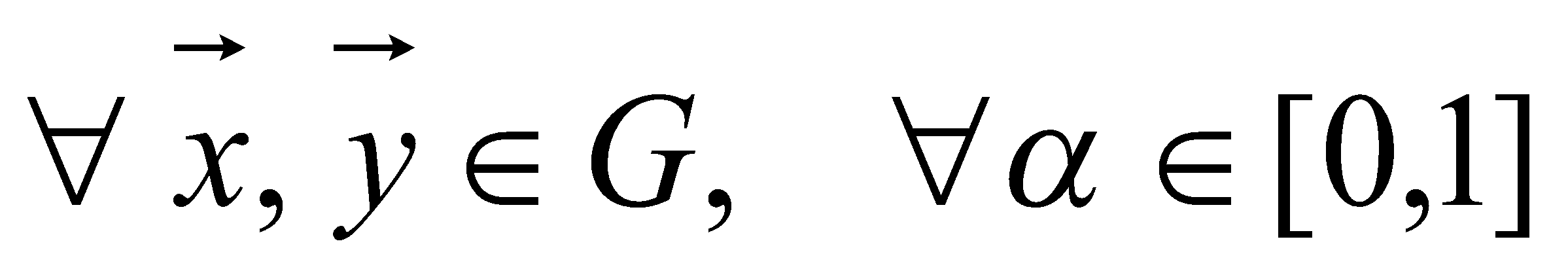
Задаем *α>0, 0<λ<1, *

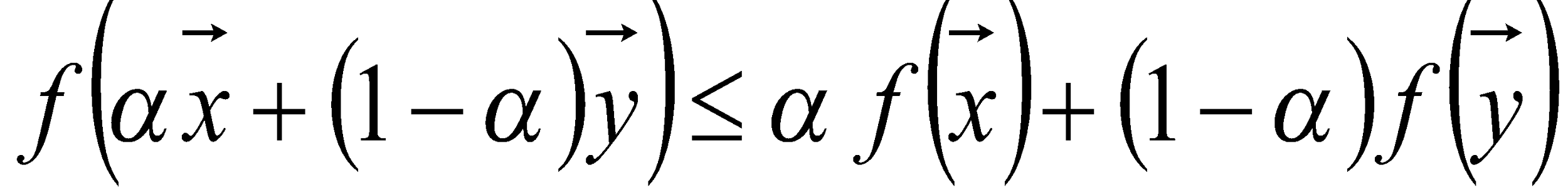
*Алгоритм:*

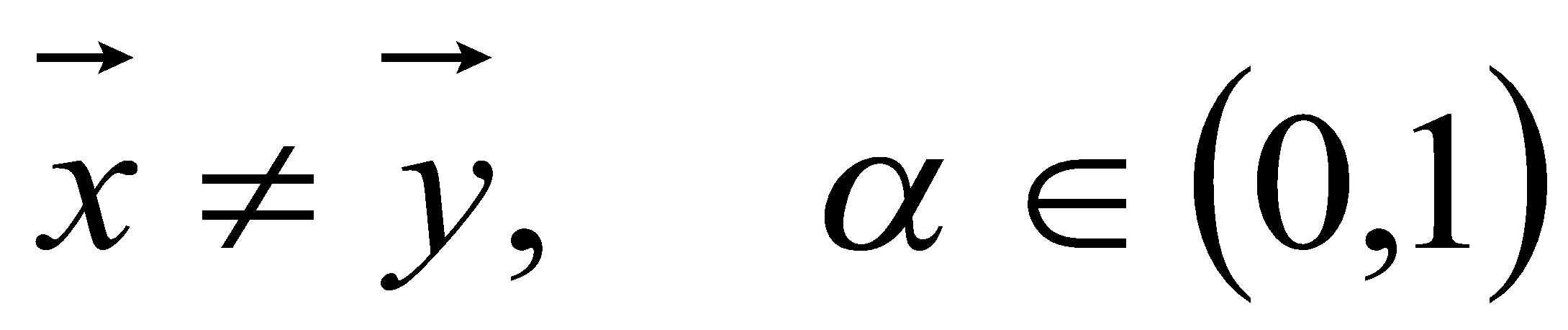


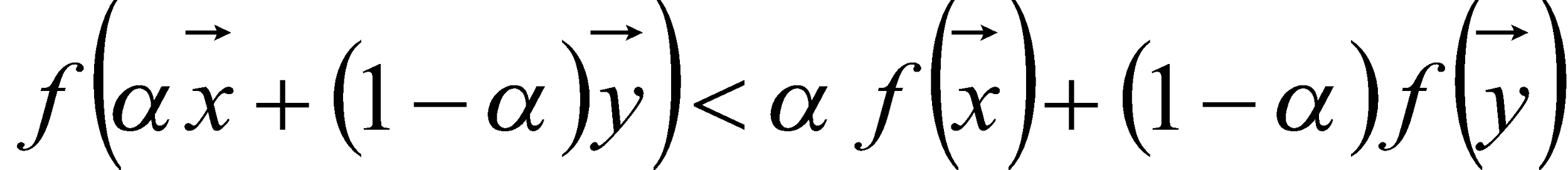
***Многомерный случай ( n>1 )***

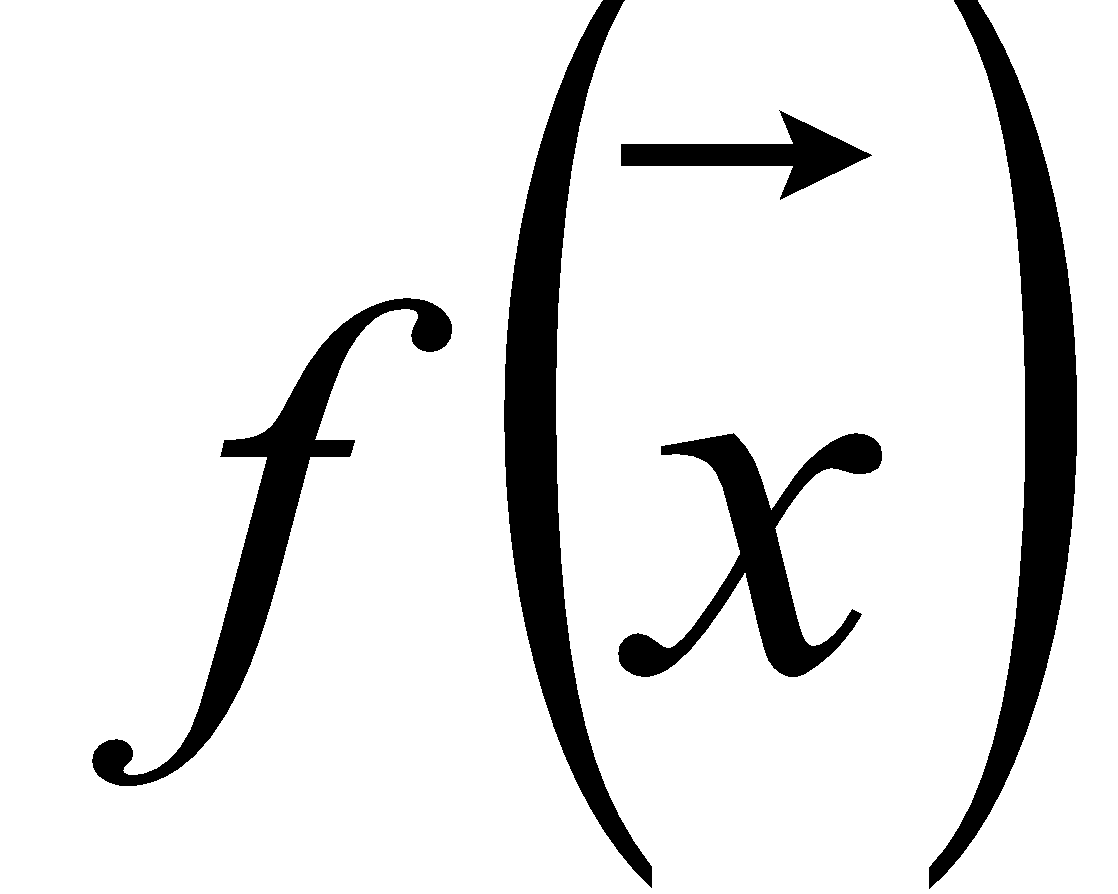
Опр. Пусть  - выпуклое множество.

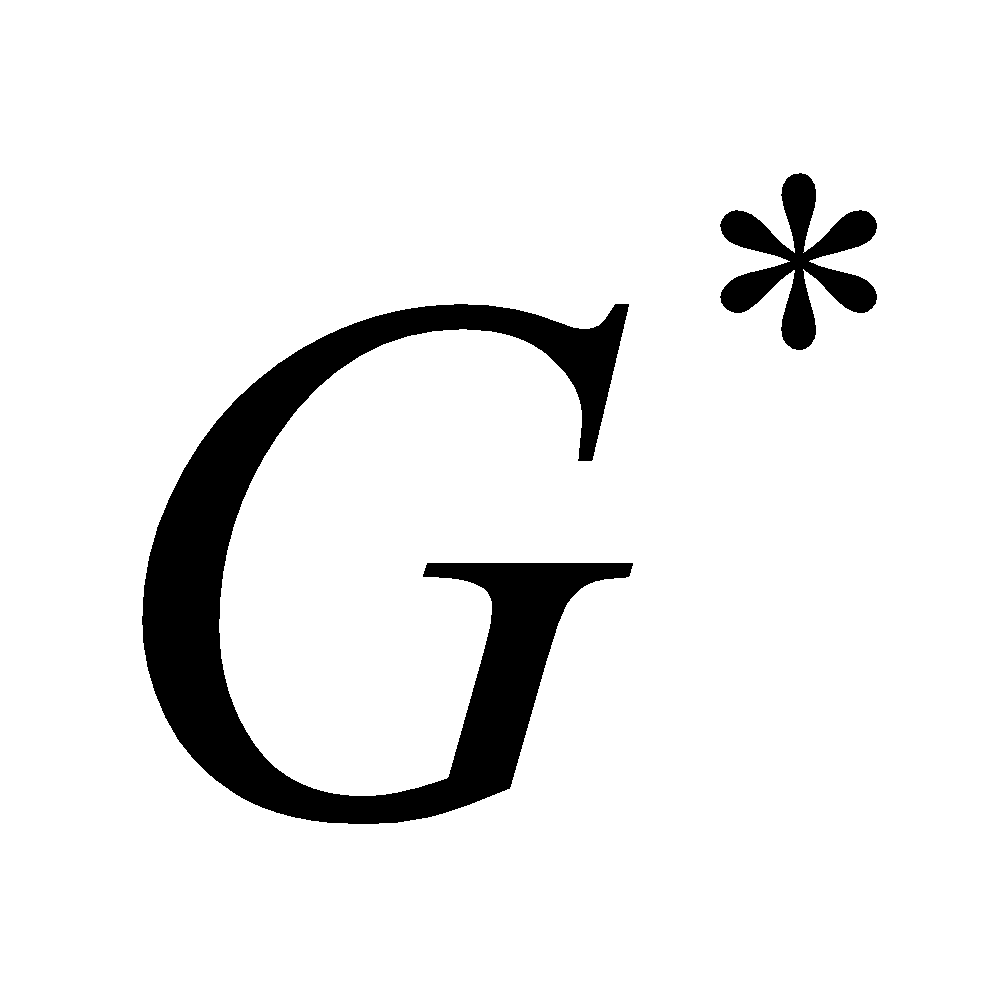
Функция  называется *выпуклой*, если для  справедливо



Если при  и

,

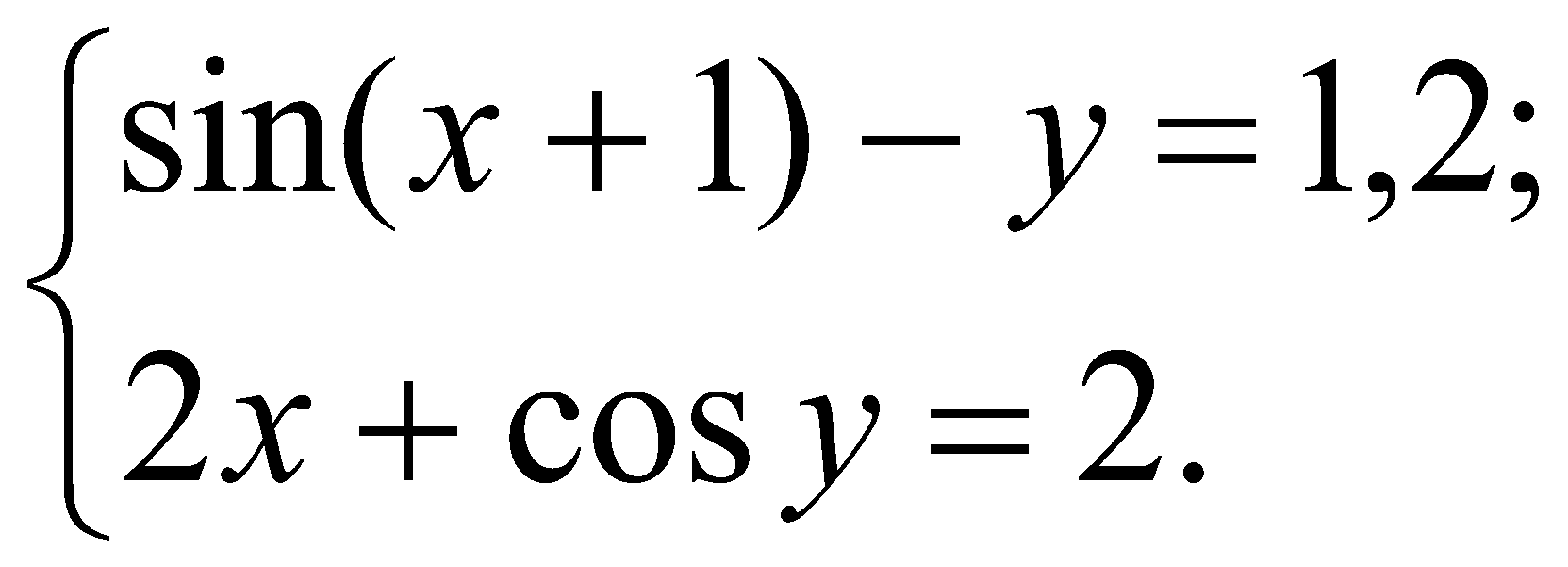
то  называется *строго выпуклой.*

В этом случае множество  состоит из одной точки.

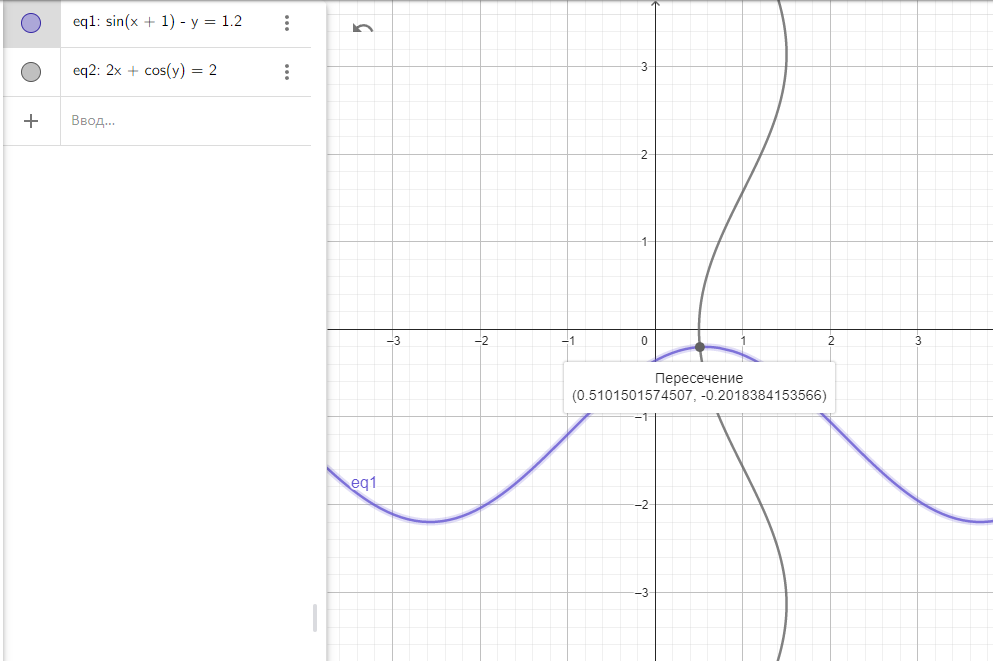
В дальнейшем рассматриваем только случай строго выпуклой функции.

**Решение**

**1 вариант:**

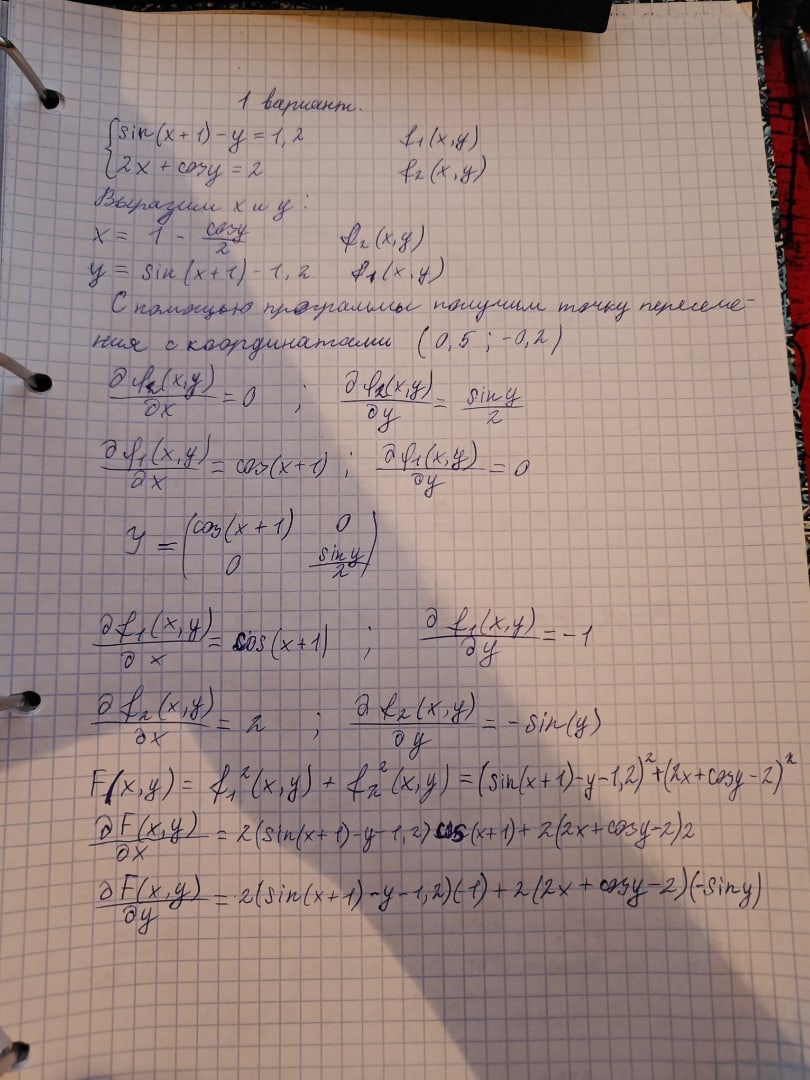


Приближенно определим корни геометрически



Возьмем корни

x0 = 0,5 ; y0 = -0,2



x0 y0

0,500 -0,200

Alfa0 = 1.000 Lambda = 0.500

Метод Ньютона

Матрица производных

cos(x + 1) -1

2 -sin(y)

Itr| x | y | Норма невязки| F1 | F2 |

1| 0,51014427| -0,20178744| 5,135990E-005|-5,1336037445E-005| -1,565432E-006|

2| 0,51015016| -0,20183842| 1,273174E-009|-1,7321921675E-011| -1,273056E-009|

3| 0,51015016| -0,20183842| 2,371437E-016| 0,0000000000E+000| 0,000000E+000|

0,51015015745074 -0,201838415356574

Метод простой итерации

Fi1(x,y) = sin(x + 1) - 1.2

Fi2(x,y) = 1 - 0.5\*cos(y)

Якобиан

cos(x + 1) 0

0 0.5\*sin(y)

Значения

0,0707372016677029 0

0 -0,09933466539753061

Норма матрицы = 0,09933466539753061

Itr| x | y | Норма невязки|Оценка погрешности| Погрешность | Норма якобиана|

1| 0,50996671| -0,20250501| 8,248489E-004| 1,1489881303E-003| 6,913794E-004| 1,005619E-001|

2| 0,51021708| -0,20184955| 1,324923E-004| 7,8170167012E-005| 6,784558E-005| 1,002408E-001|

3| 0,51015127| -0,20183436| 5,016363E-006| 7,5238579985E-006| 4,204904E-006| 1,002334E-001|

Метод градиентного спуска

Itr| x | y | Alfa | Норма невязки | Погрешность | k|

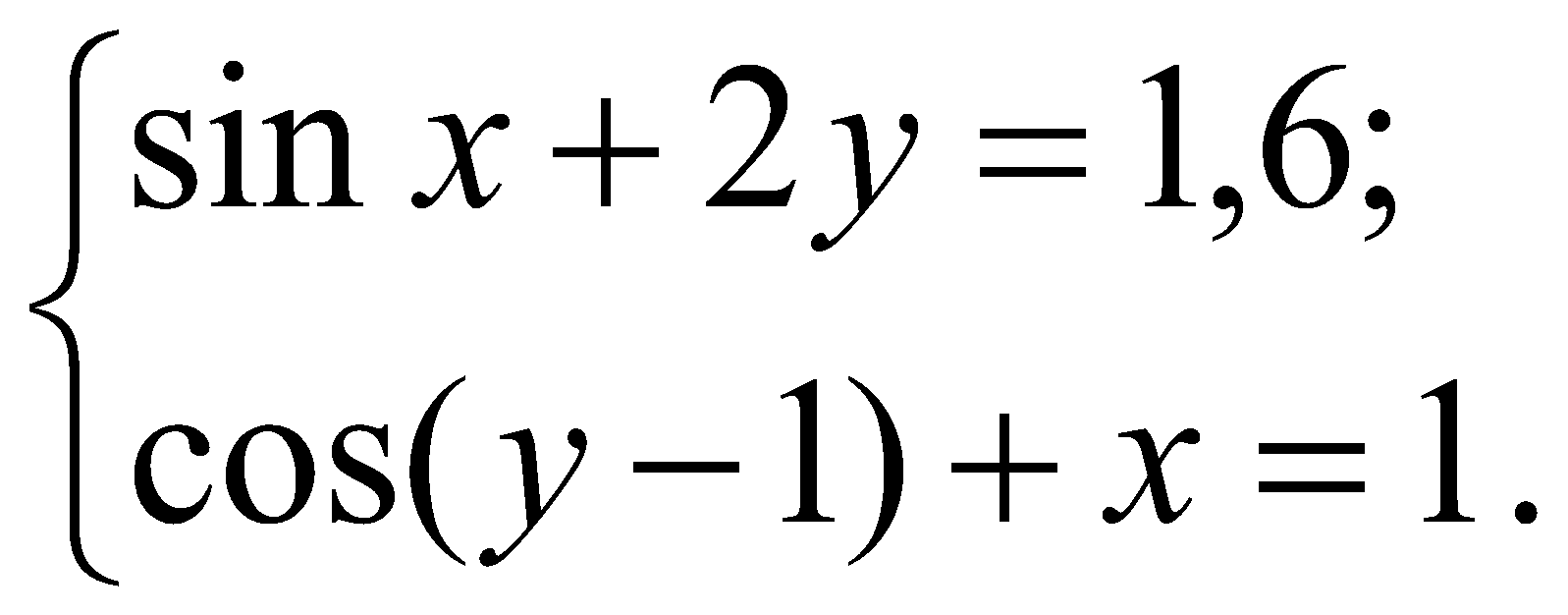
1| 0,51001101| -0,19963621| 0,125| 2,216486E-003| 2,206594E-003| 3|

2| 0,50982369| -0,20187875| 0,5| 6,613427E-004| 3,289514E-004| 1|

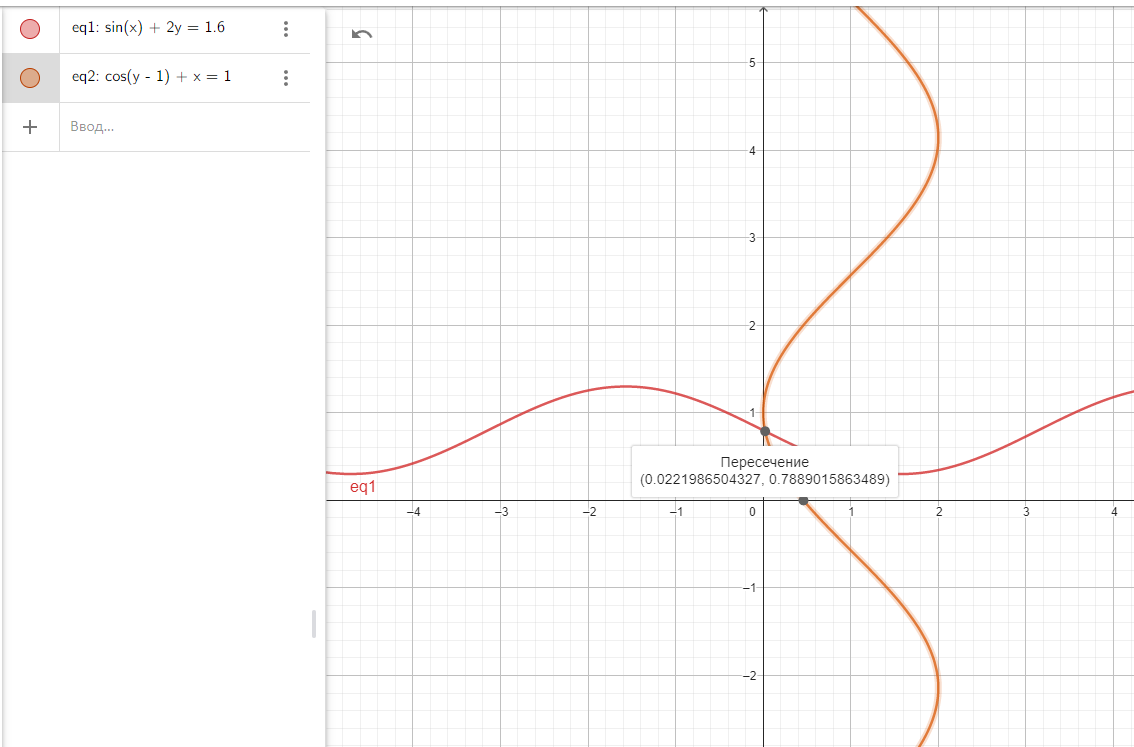
3| 0,51015389| -0,20184049| 0,125| 7,412272E-006| 4,268967E-006| 3|

Общее число итераций = 7

**2 вариант:**

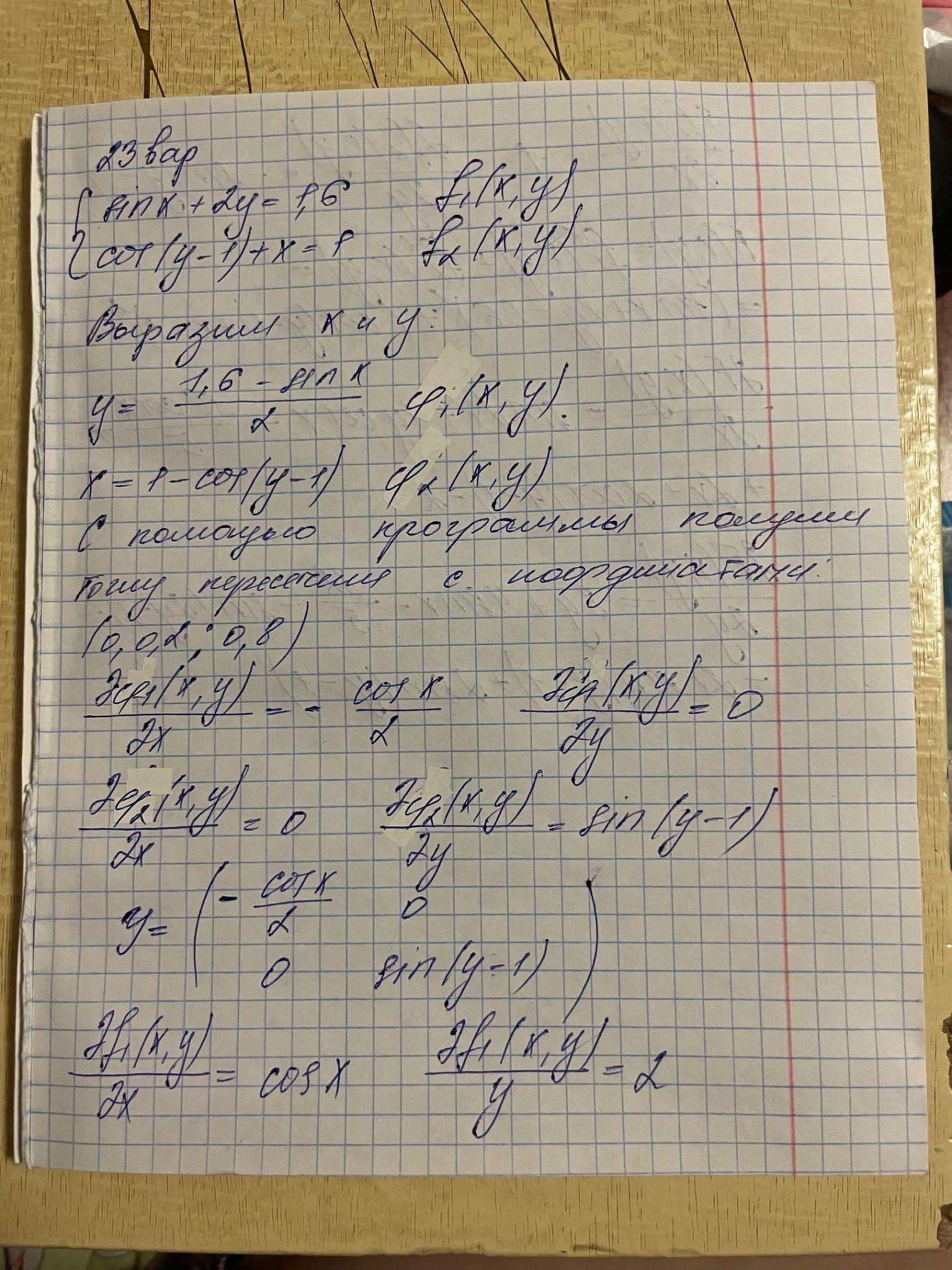


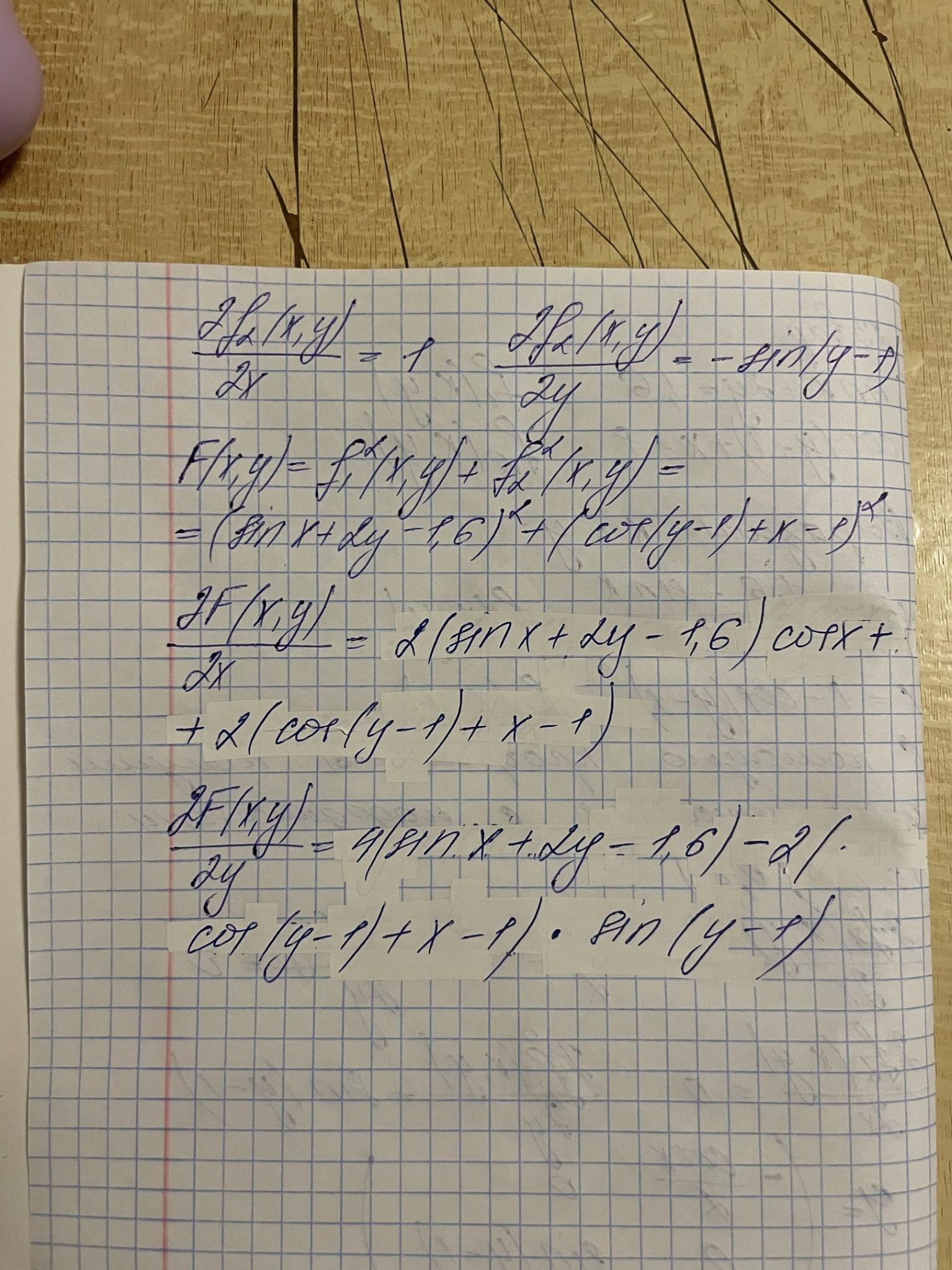
Приближенно определим корни геометрически



Возьмем корни

x0 = 0,02 ; y0 = 0,8





x0 y0

0,020 0,800

Alfa0 = 1.000 Lambda = 0.500

Метод Ньютона

Матрица производных

cos(x) 2

1 -sin(y - 1)

Itr| x | y | Норма невязки| F1 | F2 |

1| 0,02213169| 0,78893503| 5,995103E-005|-4,7052158170E-008| -5,995101E-005|

2| 0,02219865| 0,78890159| 5,492000E-010|-4,9658721579E-011| -5,469503E-010|

3| 0,02219865| 0,78890159| 2,272951E-016| 2,2204460493E-016| 0,000000E+000|

0,0221986504327319 0,7889015863489037

Метод простой итерации

Fi1(x,y) = 0.5\*(1.6 - sin(x))

Fi2(x,y) = 1 - cos(y - 1)

Якобиан

-0.5\*cos(x) 0

0 sin(y - 1)

Значения

-0,4999000033332889 0

0 -0,19866933079506116

Норма матрицы = 0,4999000033332889

Itr| x | y | Норма невязки|Оценка погрешности| Погрешность | Норма якобиана|

1| 0,01993342| 0,79000067| 2,036612E-003| 9,9955826738E-003| 2,517784E-003| 4,999007E-001|

2| 0,02196895| 0,79003395| 2,035089E-003| 2,0348139541E-003| 1,155426E-003| 4,998793E-001|

3| 0,02196201| 0,78901641| 2,127016E-004| 1,0170715181E-003| 2,630281E-004| 4,998794E-001|

4| 0,02217460| 0,78901988| 2,125380E-004| 2,1251224799E-004| 1,207126E-004| 4,998771E-001|

5| 0,02217387| 0,78891361| 2,227191E-005| 1,0621860339E-004| 2,754239E-005| 4,998771E-001|

Метод градиентного спуска

Itr| x | y | Alfa | Норма невязки | Погрешность | k|

1| 0,01498469| 0,78999736| 0,125| 8,602410E-003| 7,296709E-003| 3|

2| 0,01798607| 0,79287196| 0,125| 5,038511E-003| 5,788755E-003| 3|

3| 0,01790106| 0,79118165| 0,125| 3,831449E-003| 4,864978E-003| 3|

4| 0,01968057| 0,79131444| 0,25| 3,064208E-003| 3,487496E-003| 2|

5| 0,01960747| 0,79026473| 0,125| 2,310450E-003| 2,927863E-003| 3|

6| 0,02069287| 0,79036916| 0,25| 1,866130E-003| 2,102646E-003| 2|

7| 0,02063535| 0,78971670| 0,125| 1,394452E-003| 1,763037E-003| 3|

8| 0,02129813| 0,78979478| 0,25| 1,137801E-003| 1,268356E-003| 2|

9| 0,02125510| 0,78938897| 0,125| 8,421293E-004| 1,061993E-003| 3|

10| 0,02166015| 0,78944549| 0,25| 6,944363E-004| 7,653847E-004| 2|

11| 0,02162899| 0,78919296| 0,125| 5,088186E-004| 6,398500E-004| 3|

12| 0,02187671| 0,78923295| 0,25| 4,242378E-004| 4,620081E-004| 2|

13| 0,02185465| 0,78907573| 0,125| 3,075559E-004| 3,855660E-004| 3|

14| 0,02200623| 0,78910355| 0,25| 2,594074E-004| 2,789546E-004| 2|

15| 0,02199088| 0,78900563| 0,125| 1,859715E-004| 2,323619E-004| 3|

16| 0,02208368| 0,78902473| 0,25| 1,587616E-004| 1,684707E-004| 2|

17| 0,02207315| 0,78896372| 0,125| 1,124928E-004| 1,400446E-004| 3|

18| 0,02212999| 0,78897670| 0,25| 9,725202E-005| 1,017701E-004| 2|

Общее число итераций = 46

**Код программы**

using System.Runtime.CompilerServices;

using System.Security.Cryptography.X509Certificates;

internal class Program

{

static double eps = Math.Pow(10, -4); // Критерии остановки

static double epsN = Math.Pow(10, -12);

const int n = 2; // Размерность матриц якобиан и производных

static double x, y;

static double xNewton, yNewton;

static void PrintNewton()

{

Console.WriteLine(" Itr| x | y | Норма невязки| F1 | F2 |");

}

static void PrintSimpleIteration()

{

Console.WriteLine(" Itr| x | y | Норма невязки|Оценка погрешности| Погрешность | Норма якобиана| ");

}

static void PrintGradient()

{

Console.WriteLine(" Itr| x | y | Alfa | Норма невязки | Погрешность | k| ");

}

static void PrintNewton(int itr, double x, double y, double error, double F1, double F2)

{

Console.Write(string.Format("{0,4}", itr) + "|");

Console.Write(string.Format("{0,14:F8}", x) + "|");

Console.Write(string.Format("{0,15:F8}", y) + "|");

Console.Write(string.Format("{0, 15:E}", error) + "|");

Console.Write(string.Format("{0, 18:E10}", F1) + "|");

Console.Write(string.Format("{0, 15:E}", F2) + "|" + "\n");

}

static void PrintSimpleIteration(int itr, double x, double y, double error, double F1, double F2, double normJ)

{

Console.Write(string.Format("{0,4}", itr) + "|");

Console.Write(string.Format("{0,14:F8}", x) + "|");

Console.Write(string.Format("{0,15:F8}", y) + "|");

Console.Write(string.Format("{0, 15:E}", error) + "|");

Console.Write(string.Format("{0, 18:E10}", F1) + "|");

Console.Write(string.Format("{0, 15:E}", F2) + "|");

Console.Write(string.Format("{0, 15:E}", normJ) + "|" + "\n");

}

static void PrintGradient(int itr, double x, double y, double alpha, double error, double pogr, int k)

{

Console.Write(string.Format("{0,4}", itr) + "|");

Console.Write(string.Format("{0,14:F8}", x) + "|");

Console.Write(string.Format("{0,15:F8}", y) + "|");

Console.Write(string.Format("{0, 15}", alpha) + "|");

Console.Write(string.Format("{0, 18:E}", error) + "|");

Console.Write(string.Format("{0, 15:E}", pogr) + "|");

Console.Write(string.Format("{0, 5:D}", k) + "|" + "\n");

}

static void Initialization(ref double x, ref double y, int var)

{

switch (var)

{

case 1:

{

x = 0.5; y = -0.2;

break;

}

case 2:

{

x = 0.02; y = 0.8;

break;

}

}

}

static void Jacobian(double[][] J, double x, double y, int var)

{

switch (var)

{

case 1:

{

J[0][0] = Math.Cos(x + 1);

J[0][1] = 0;

J[1][0] = 0;

J[1][1] = 0.5 \* Math.Sin(y);

break;

}

case 23:

{

J[0][0] = -0.5 \* Math.Cos(x);

J[0][1] = 0;

J[1][0] = 0;

J[1][1] = Math.Sin(y - 1);

break;

}

}

}

static double F1(double x, double y, int var)

{

if (var == 1)

return Math.Sin(x + 1) - y - 1.2;

else

return Math.Sin(x) + 2 \* y - 1.6;

//return Math.Sin(y + 2) - x - 1.5;

}

static double F2(double x, double y, int var)

{

if (var == 1)

return Math.Cos(y) + 2 \* x - 2;

else

return Math.Cos(y - 1) + x - 1;

}

static void PrintMatrix(double[][] A)

{

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

{

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

Console.Write(A[i][j] + " ");

Console.WriteLine();

}

}

static double NormMatrix(double[][] A)

{

double max = 0;

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

{

double sum = 0;

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

sum += Math.Abs(A[i][j]);

if (sum > max)

max = sum;

}

return max;

}

static double NormError(double x, double y, double xNew, double yNew)

{

return (Math.Sqrt(Math.Pow((Math.Abs(x - xNew)), 2) + Math.Pow((Math.Abs(y - yNew)), 2)));

}

static double NormNewyaz(double x, double y, int var)

{

double x1, y1;

if (var == 1)

{

x1 = - 2 + Math.Cos(y) + 2 \* x;

y1 = Math.Sin(x + 1) - 1.2 - y;

return NormError(x1, y1, 0, 0);

}

else

{

x1 = 1 - Math.Cos(y - 1) - x;

y1 = 1.6 - Math.Sin(x) - 2 \* y;

return NormError(x1, y1, 0, 0);

}

}

f1(x,y) и f2(x,y)

static void InitializeDF(double[][] DF, double x, double y, int var)

{

if (var == 1)

{

DF[0][0] = Math.Cos(x + 1);

DF[0][1] = -1;

DF[1][0] = 2;

DF[1][1] = - Math.Sin(y);

}

else if (var == 23)

{

DF[0][0] = Math.Cos(x);

DF[0][1] = 2;

DF[1][0] = 1;

DF[1][1] = -Math.Sin(y - 1);

}

}

static void ReverseDF(double[][] DF)

{

double det = DF[0][0] \* DF[1][1] - DF[0][1] \* DF[1][0];

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

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

DF[i][j] /= (i + j == 1) ? -det : det;

var temp = DF[0][0];

DF[0][0] = DF[1][1];

DF[1][1] = temp;

}

static double FF(double x, double y, int var)

{

return Math.Pow(F1(x, y, var), 2) + Math.Pow(F2(x, y, var), 2);

}

static void InitializeDFF(double[] DFF, double x, double y, int var)

{

if (var == 1)

{

DFF[0] = 2 \* (Math.Sin(x + 1) - y - 1.2) \* Math.Cos(x + 1) + 4 \* (2 \* x + Math.Cos(y) - 2);

DFF[1] = -2 \* (Math.Sin(x + 1) - y - 1.2) + 2 \* (2 \* x + Math.Cos(y) - 2) \* (-Math.Sin(y));

}

else if (var == 23)

{

DFF[0] = 2 \* (Math.Sin(x) + 2 \* y - 1.6) \* Math.Cos(x) + 2 \* (Math.Cos(y - 1) + x - 1);

DFF[1] = 4 \* (Math.Sin(x) + 2 \* y - 1.6) - 2 \* (Math.Cos(y - 1) + x - 1) \* Math.Sin(y - 1);

}

}

static void ShowJacobian(int var)

{

if (var == 1)

{

Console.WriteLine("cos(x + 1) 0\n0 0.5\*sin(y)");

}

else if (var == 23)

{

Console.WriteLine("-0.5\*cos(x) 0\n0 sin(y - 1)");

}

}

static void ShowMatrProizv(int var)

{

if (var == 1)

{

Console.WriteLine("cos(x + 1) -1\n2 -sin(y)\n\n");

}

else if (var == 23)

{

Console.WriteLine("cos(x) 2\n1 -sin(y - 1)\n\n");

}

}

static void ShowFi(int var)

{

//Console.WriteLine("Fi1(x,y) = sin(y + 2) - 1.5");

//Console.WriteLine("Fi2(x,y) = 0.5 - cos(x - 2)");

if (var == 1)

{

Console.WriteLine("Fi1(x,y) = sin(x + 1) - 1.2");

Console.WriteLine("Fi2(x,y) = 1 - 0.5\*cos(y)");

}

else if (var == 23)

{

Console.WriteLine("Fi1(x,y) = 0.5\*(1.6 - sin(x))");

Console.WriteLine("Fi2(x,y) = 1 - cos(y - 1)");

}

}

static void NewtonMethod(double x, double y, int var)

{

double xNew, yNew, error;

int itr = 0;

double[][] DF = new double[n][];

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

DF[i] = new double[n];

Console.WriteLine("Метод Ньютона");

Console.WriteLine("Матрица производных");

ShowMatrProizv(var);

PrintNewton();

do

{

itr++;

InitializeDF(DF, x, y, var);

ReverseDF(DF);

double F1\_ = F1(x, y, var);

double F2\_ = F2(x, y, var);

xNew = x - (DF[0][0] \* F1\_ + DF[0][1] \* F2\_);

yNew = y - (DF[1][0] \* F1\_ + DF[1][1] \* F2\_);

error = NormNewyaz(xNew, yNew, var);

PrintNewton(itr, xNew, yNew, error, F1(xNew, yNew, var), F2(xNew, yNew, var));

x = xNew;

y = yNew;

} while (error > epsN);

Console.WriteLine(x + " " + y);

xNewton = x;

yNewton = y;

}

static void SimpleIterMethod(double x, double y, double[][] J, int var)

{

double xNew, yNew, q, error, err, q1, errN;

int itr = 0;

Console.WriteLine("Метод простой итерации");

ShowFi(var);

Console.WriteLine("Якобиан");

ShowJacobian(var);

Console.WriteLine("Значения");

Jacobian(J, x, y, var);

PrintMatrix(J);

Console.WriteLine("Норма матрицы = " + NormMatrix(J));

PrintSimpleIteration();

do

{

itr++;

if (var == 1)

{

xNew = 1 - 0.5 \* Math.Cos(y);

yNew = Math.Sin(x + 1) - 1.2;

}

else

{

xNew = 1 - Math.Cos(y - 1);

yNew = 0.5 \* (1.6 - Math.Sin(x));

}

Jacobian(J, xNew, yNew, var);

q = NormMatrix(J);

error = NormNewyaz(xNew, yNew, var);

errN = NormError(xNewton, yNewton, xNew, yNew);

err = NormError(x, y, xNew, yNew);

q1 = q / (1 - q) \* err;

PrintSimpleIteration(itr, xNew, yNew, error, q1, errN, q);

x = xNew;

y = yNew;

} while (error > eps);

}

static void GradientDescentMethod(double x, double y, int var)

{

double xNew, yNew, error, err;

int itr = 0;

double[] DFF = new double[n];

int sumk = 0;

Console.WriteLine("Метод градиентного спуска ");

PrintGradient();

do

{

int k = 0;

itr++;

double alpha = 1, lambda = 0.5;

InitializeDFF(DFF, x, y, var);

while (FF(x - alpha \* DFF[0], y - alpha \* DFF[1], var) >= FF(x, y, var))

{

alpha \*= lambda;

k++;

}

xNew = x - alpha \* DFF[0];

yNew = y - alpha \* DFF[1];

error = NormNewyaz(xNew, yNew, var);

err = NormError(xNewton, yNewton, xNew, yNew);

PrintGradient(itr, xNew, yNew, alpha, error, err, k);

sumk += k;

x = xNew;

y = yNew;

} while (error > eps);

Console.WriteLine("Общее число итераций = " + sumk);

}

private static void Main(string[] args)

{

double[][] J = new double[n][];

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

J[i] = new double[n];

Console.WriteLine("Введите номер варианта: 1 или 23");

int var = Convert.ToInt32(Console.ReadLine());

Initialization(ref x, ref y, var);

Console.WriteLine(string.Format("{0,4:f3}", x) + " " + string.Format("{0,4:f3}", y));

Console.WriteLine("Alfa0 = 1.000 Lambda = 0.500");

NewtonMethod(x, y, var);

Console.WriteLine();

SimpleIterMethod(x, y, J, var);

Console.WriteLine();

GradientDescentMethod(x, y, var);

}

}