Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Численные методы»

Студент:	Ветренко П. С.
Преподаватель:	Иванов И. Э.
Группа:	М8О-306Б
Вариант:	2
Дата:	
Оценка:	
Подпись:	

Москва, 2020

Лабораторная работа №2.1

Задача: Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

$ln(x+2) - x^2 = 0$

1 Описание метода решения

Численное решение нелинейных (алгебраических или трансцендентных) уравнений вида

$$f(x) = 0 \qquad (2.1)$$

заключается в нахождении значений x, удовлетворяющих (с заданной точностью) данному уравнению и состоит из следующих основных этапов:

1. Отделение (изоляция, локализация) корней уравнения.
2. Уточнение с помощью некоторого вычислительного алгоритма конкретного выделенного корня с заданной точностью.

Для уточнения корня с требуемой точностью обычно применяется какой-либо итерационный метод, заключающийся в построении числовой последовательности $x^{(k)}$ $(k = 0, 1, 2, ...)$, сходящейся к искомому корню $x^{(*)}$ уравнения (2.1).

Метод Ньютона (метод касательных).
При нахождении корня уравнения (2.1) методом Ньютона, итерационный процесс определяется формулой

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad k = 0, 1, 2, ... \qquad (2.2)$$

Для начала вычислений требуется задание начального приближения $x^{(0)}$.
Условия сходимости метода определяются следующей теоремой [2.1]:

Теорема2.1. Пусть на отрезке $[a, b]$ функция $f(x)$ имеет первую и вторую производные постоянного знака и пусть $f(a)f(b) < 0$.
Тогда если точка $x^{(0)}$ выбрана на $[a, b]$ так, что

$$f(x^{(0)})f''(x^{(0)}) > 0, \qquad (2.3)$$

то начатая с нее последовательность $x^{(k)}(k = 0, 1, 2, ...)$, определяемая методом Ньютона (2.2), монотонно сходится к корню $x^{(*)} \in (a, b)$ уравнения (2.1).

В качестве условия окончания итераций в практических вычислениях часто используется правило $|x^{(k+1)} - x^{(k)} < \epsilon| \Rightarrow x^{(*)} \approx x^{(k+1)}$.

Метод простой итерации.
При использовании метода простой итерации уравнение (2.1) заменяется эквивалентным уравнением с выделенным линейным членом

$$x = \phi(x) \qquad (2.4)$$

Решение ищется путем построения последовательности

$$x^{(k+1)} = (x^{(k)})k = 0, 1, 2, ... \qquad (2.5)$$

начиная с некоторого заданного значения $x^{(0)}$. Если $\phi(x)$ - непрерывная функция, а $x^{(k)}(k = 0, 1, 2, ...)$ - сходящаяся последовательность, то значение $x^{(*)} = \lim_{x\to\infty} x^{(k)}$ является решением уравнения (2.4).
Условия сходимости метода и оценка его погрешности определяются теоремой [2.1]:

Теорема 2.2. Пусть функция $\phi(x)$ определена и дифференцируема на отрезке $[a, b]$. Тогда если выполняются условия:

1) $\phi(x) \in [a, b]\forall x \in [a, b]$,
2) $\exists q : \phi^{'}(x) <= q < 1\forall x \in (a, b)$,

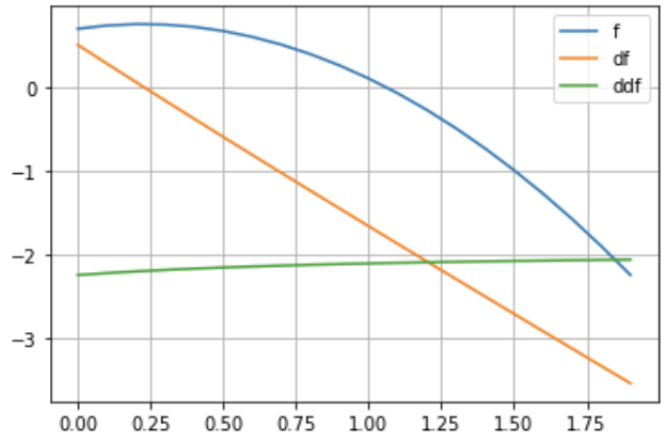
то уравнение (2.4) имеет и притом единственный на $[a, b]$ корень $x^{(*)}$;
к этому корню сходится определяемая методом простой итерации последовательность $x^{(k)}(k = 0, 1, 2, ...)$,начинающаяся с любого $x^{(0)} \in [a, b]$. При этом справедливы оценки погрешности $(\forall k \in N)$:

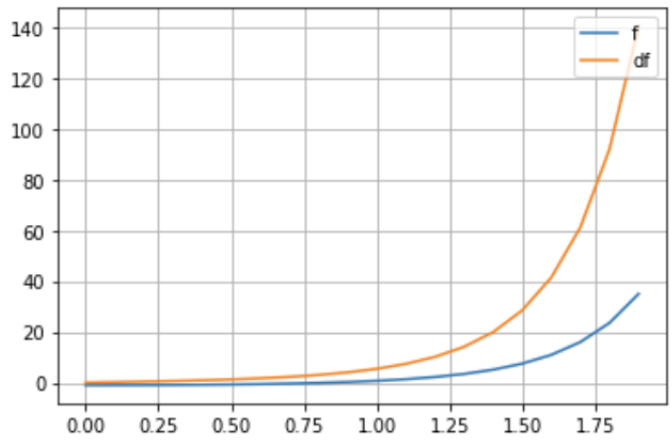
$$|x^{(*)} - x^{(k+1)}| <= \frac{q}{1-q}|x^{(k+1)} - x^{(k)}| \qquad (2.6)$$

$$|x^{(*)} - x^{(k+1)}| <= \frac{q^{k+1}}{1-q}|x^{(1)} - x^{(0)}|$$

2 Входные и выходные данные

```
Лабораторная работа 2.1
Вариант 2

Моя функция: ln(x+2) - x^2

Метод простых итераций решения нелинейного уравнения:
x: 7.487735836358526, k: 1, q/(1-q)*|x_cur - x|: -6.0152792570301905

Метод Ньютона решения нелинейного уравнения:
x: 1.058135145295947, k: 1, |x_cur - x|: 0.04186485470405299
x: 1.0571041765600742, k: 2, |x_cur - x|: 0.0010309687358729391
x: 1.0571035499949695, k: 3, |x_cur - x|: 6.265651046888365e-07
```





3 Программный код

```
import math
import numpy as np
import matplotlib.pyplot as plt

def f(x):
return math.log(x+2) -x**2

def df(x):
return 1/(x+2) -2*x

def ddf(x):
return -1/(x+2)**2 -2

def phi(x):
return math.exp(x**2)-2

def dphi(x):
return 2*x*math.exp(x**2)

def simpleIteration(pi,dphi,a,b,eps=0.001):
q = max(abs(dphi(a)),abs(dphi(b)))
x = (a + b) / 2
k = 0
go = True

while go:
k += 1
x_cur = phi(x)

print(f'x: {x_cur},k: {k},q/(1-q)*|x_cur -x|: {q * abs(x_cur -x) / (1 -q)}')
if (q * abs(x_cur -x) / (1 -q)) <= eps:
go = False

x = x_cur
if k == 10:
break

def newton(f,df,x0,eps=0.001):
x = x0
```

```
k = 0
go = True
while go:
k += 1
x_cur = x -f(x) / df(x)
print(f'x: {x_cur},k: {k},|x_cur -x|: {abs(x_cur -x)}')
if abs(x_cur -x) <= eps:
go = False

x = x_cur

def show(f,df,x,file = None,step = 0.5,ddf = None):
X = np.arange(x[0],x[-1],step)
Y = [f(i) for i in X]
dY = [df(i) for i in X]

if ddf:
ddY = [ddf(i) for i in X]

fig,axis = plt.subplots()
axis.plot(X,Y,label='f')
axis.plot(X,dY,label='df')

if ddf:
axis.plot(X,ddY,label='ddf')

axis.legend(loc='upper right')
axis.grid()

if file:
fig.savefig(file)
print(f'File {file} was saved correctly')
plt.close(fig)

plt.show()

if __name__ == '__main__':
print('Лабораторная работа 2.1\nВариант 2')
print('\nМоя функция: ln(x+2) -x^2')
print("\nМетод простых итераций решения неоднородного линейного уравнения:")
simpleIteration(phi,dphi,1,2)
```

```
print("\nМетод Ньютона решения неоднородного линейного уравнения:")
newton(f,df,1.1)
show(f,df,[0,2],step=0.1,ddf=ddf)
show(phi,dphi,[0,2],step=0.1)
```

Лабораторная работа №2.2

Задача: Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

$a = 3$

$$\begin{cases} (x_1^2 + a^2)x_2 - a^3 = 0 \\ (x_1 - \frac{a}{2})^2 + (x_2 - \frac{a}{2})^2 - a^2 = 0 \end{cases}$$

1 Описание метода решения

Систему нелинейных уравнений с n неизвестными можно записать в виде

$$\begin{cases} f_1(x_1, x_2, ..., x_n) = 0 \\ f_2(x_1, x_2, ..., x_n) = 0 \\ ... \\ f_n(x_1, x_2, ..., x_n) = 0 \end{cases} \qquad (2.7)$$

или, более коротко, в векторной форме

$$f(x) = 0, \qquad (2.8)$$

где x - вектор неизвестных величин, f - вектор-функция

$$X = \begin{pmatrix} {}_1 \\ x_2 \\ ... \\ x_n \end{pmatrix}, \qquad f = \begin{pmatrix} {}_1(x) \\ f_2(x) \\ ... \\ f_n(x) \end{pmatrix}, \qquad 0 = \begin{pmatrix} 0 \\ ... \\ 0 \end{pmatrix}.$$

Метод Ньютона.
Если определено начальное приближение $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})^T$, итерационный процесс нахождения решения системы (2.7) методом Ньютона можно представить в виде

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} + \Delta x_1^{(k)} \\ x_2^{(k+1)} = x_2^{(k)} + \Delta x_2^{(k)} \qquad k = 0, 1, 2, ... \\ ... \\ x_n^{(k+1)} = x_n^{(k)} + \Delta x_n^{(k)} \end{cases} \qquad (2.9)$$

где значения приращений $\Delta x_1^{(k)}, \Delta x_2^{(k)}, .., \Delta x_n^{(k)}$ определяются из решения системы линейных алгебраических уравнений, все коэффициенты которой выражаются через известное предыдущее приближение $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})$

$$\begin{cases} f_1(x^{(k+1)}) + \frac{df_1(x^{(k)})}{dx_1}\Delta x_1^{(k)} + \frac{df_1(x^{(k)})}{dx_2}\Delta x_2^{(k)} + ... + \frac{df_1(x^{(k)})}{dx_n}\Delta x_n^{(k)} = 0 \\ f_2(x^{(k+1)}) + \frac{df_2(x^{(k)})}{dx_1}\Delta x_1^{(k)} + \frac{df_2(x^{(k)})}{dx_2}\Delta x_2^{(k)} + ... + \frac{df_2(x^{(k)})}{dx_n}\Delta x_n^{(k)} = 0 \\ ... \\ f_n(x^{(k+1)}) + \frac{df_n(x^{(k)})}{dx_1}\Delta x_1^{(k)} + \frac{df_n(x^{(k)})}{dx_2}\Delta x_2^{(k)} + ... + \frac{df_n(x^{(k)})}{dx_n}\Delta x_n^{(k)} = 0 \end{cases} \qquad (2.10)$$

В векторно-матричной форме расчетные формулы имеют вид

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)} \qquad k = 0, 1, 2, ... \qquad (2.11)$$

где вектор приращений $X = \begin{pmatrix} x_1^{(k)} \\ \Delta x_2^{(k)} \\ ... \\ \Delta x_n^{(k)} \end{pmatrix}$ находится из решения уравнения

$$f(x^{(k)}) + J(x^{(k)})\Delta x^{(k)} = 0 \qquad (2.12)$$

Здесь $\begin{bmatrix} \frac{df_1(x^{(k)})}{dx_2} \frac{df_1(x^{(k)})}{dx_2} ... \frac{df_1(x^{(k)})}{dx_n} \\ \frac{df_2(x^{(k)})}{dx_1} \frac{df_2(x^{(k)})}{dx_2} ... \frac{df_2(x^{(k)})}{dx_n} \\ ... \\ \frac{df_n(x^{(k)})}{dx_1} \frac{df_n(x^{(k)})}{dx_2} \frac{df_n(x^{(k)})}{dx_n} \end{bmatrix}$ - матрица Якоби первых производных вектор-функции $f(x)$.

Выражая из (2.12) вектор приращений $\Delta x^{(k)}$ и подставляя его в (2.11), итерационный процесс нахождения решения можно записать в виде

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})f(x^{(k)}) \qquad k = 0, 1, 2, ... \qquad (2.13)$$

где $J^{-1}(x)$ - матрица, обратная матрице Якоби. Формула (2.13) есть обобщение формулы (2.2) на случай систем нелинейных уравнений.

В практических вычислениях в качестве условия окончания итераций обычно используется критерий

$$||x^{(k+1)} - x^{(k)}|| <= \epsilon, \qquad (2.14)$$

где ϵ - заданная точность.

Метод простой итерации.

При использовании метода простой итерации система уравнений (2.7) приводится к эквивалентной системе специального вида

$$\begin{cases} x_1 = \phi_1(x_1, x_2, ..., x_n) \\ x_2 = \phi_1(x_1, x_2, ..., x_n) \\ ... \\ x_n = \phi_1(x_1, x_2, ..., x_n) \end{cases} \qquad (2.15)$$

или, в векторной форме

$$x = \phi(x), \qquad \phi(x) = \begin{pmatrix} {}_1(x) \\ \phi_2(x) \\ ... \\ \phi_n(x) \end{pmatrix} \qquad (2.16)$$

где функции $\phi_1(x), \phi_2(x), ..., \phi_n(x)$ - определены и непрерывны в некоторой окрестности искомого изолированного решения $x^{(*)} = (x_1^{(*)}, x_2^{(*)}, ..., x_n^{(*)})^T$

Если выбрано некоторое начальное приближение $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})^T$ последующие приближения в методе простой итерации находятся по формулам

$$\begin{cases} x_1^{(k+1)} = \phi_1(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \\ x_2^{(k+1)} = \phi_1(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \qquad k = 0, 1, 2, ... \qquad (2.17) \\ ... \\ x_n^{(k+1)} = \phi_1(x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)}) \end{cases}$$

или, в векторной форме

$$x^{(k+1)} = \phi(x^{(k)}), \qquad k = 0, 1, 2, ... \qquad (2.18)$$

Если последовательность векторов $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})^T$ сходится, то она сходится к решению $x^{(*)} = (x_1^{(*)}, x_2^{(*)}, ..., x_n^{(*)})^T$.

Достаточное условие сходимости итерационного процесса (2.17) формулируется следующим образом:
Теорема 2.3.Пусть вектор-функция $\phi(x)$ непрерывна, вместе со своей производной

$$\phi^{'}(x) = \begin{bmatrix} \frac{df_1(x)}{dx_2} \frac{df_1(x)}{dx_2} ... \frac{df_1(x)}{dx_n} \\ \frac{df_2(x)}{dx_1} \frac{df_2(x)}{dx_2} ... \frac{df_2(x)}{dx_n} \\ ... \\ \frac{df_n(x)}{dx_1} \frac{df_n(x)}{dx_2} \frac{df_n(x)}{dx_n} \end{bmatrix}$$

в ограниченной выпуклой замкнутой области G и

$$\max_{x \in G} ||\phi^{'}(x)|| <= q < 1 \qquad (2.19)$$

где q - постоянная. Если $x^{(0)} \in G$ и все последовательные приближения

$$x^{(k+1)} = \phi(x^{(k)}), \qquad k = 0, 1, 2, ...$$

также содержатся в G , то процесс итерации (2.23) сходится к единственному решению уравнения

$$x = \phi(x)$$

и справедливы оценки погрешности $(\forall k \in N)$:

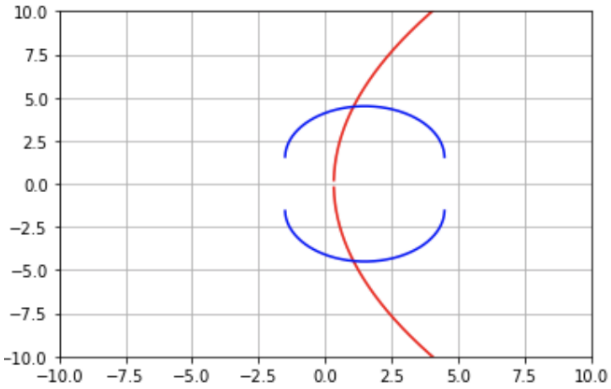
$$||x^{(*)} - x^{(k+1)}|| <= \frac{q^{(k+1)}}{1-q}||x^{(1)} - x^{(0)}||$$

$$||x^{(*)} - x^{(k+1)}|| <= \frac{q}{1-q}||x^{(k+1)} - x^{(k)}|| \qquad (2.20)$$

2 Входные и выходные данные

```
Лабораторная работа 2.2
Вариант 2

Моя система:
(x1^2+9)/x2 - 27 = 0
(x1-1.5)^2 + (x2-1.5)^2 - 9 = 0

Введите точность вычислений:
0.01

Метод простых итераций

lambda =  [[ 0.08333333 -0.83333333]
 [-0.08333333 -0.16666667]]

q =  0.9659863945578231
1 итерация
x =  [-3.91666667 -1.58333333]
0.9659863945578231
epsk =  133.24984886378735

2 итерация
x =  [24.48708577 -0.13986355]
0.9659863945578231
epsk =  807.7075623841786

3 итерация
x =  [ 824.44215307 -277.99966619]
0.9659863945578231
epsk =  24050.192180475784

4 итерация
x =  [630484.48580656 125406.80738108]
0.9659863945578231
epsk =  18235109.925961044

5 итерация
x =  [3.44363105e+11 6.88729373e+10]
0.9659863945578231
epsk =  9973576119171.123

6 итерация
x =  [1.02774525e+23 2.05549049e+22]
0.9659863945578231
epsk =  2.9766000634286307e+24

7 итерация
x =  [9.15425587e+45 1.83085117e+45]
0.9659863945578231
epsk =  2.651295025085743e+47

8 итерация
x =  [7.26270138e+91 1.45254028e+91]
0.9659863945578231
epsk =  2.103454864089711e+93

9 итерация
x =  [4.57139205e+183 9.14278410e+182]
0.9659863945578231
epsk =  inf

10 итерация
x =  [nan inf]
0.9659863945578231
epsk =  nan

11 итерация
x =  [nan nan]
0.9659863945578231
epsk =  nan


Метод Ньютона
1 итерация
x =  [69.09250026 15.96777315]
epsk =  69.05839956771548

2 итерация
x =  [34.15477027 14.37784277]
epsk =  34.97388819556771

3 итерация
x =  [17.21093931  9.85150077]
epsk =  17.537992451093626

4 итерация
x =  [10.63367322  3.81000444]
epsk =  8.930851419637623

5 итерация
x =  [6.76522052 1.84166597]
epsk =  4.340424254490401

6 итерация
x =  [5.01680583 1.21575499]
epsk =  1.857072553474772

7 итерация
x =  [4.51589436 1.08470913]
epsk =  0.5177695579429815
```

```
8 итерация
x =  [4.46987635 1.07328869]
epsk =  0.047413961792559176

9 итерация
x =  [4.46948455 1.073196  ]
epsk =  0.0004026117621692048
```



3 Программный код

```
from sympy import diff,symbols
import matplotlib.pyplot as plt
import numpy as np
import math
import cmath
from itertools import product
from numpy.linalg import norm,solve,det

def g1(x):
return (27*x-9)**(0.5)
def g2(x):
return (9-(x-1.5)**2)**(0.5) + 1.5


def f1(x1,x2):
return (x1**2+9)/x2-27
def f2(x1,x2):
return (x1-1.5)**2+(x2-1.5)**2 -9
def f11(x1,x2):
return 2*x1/x2
def f12(x1,x2):
return -(x1**2+9)/x2**2
def f21(x1,x2):
return 2*x1-3
def f22(x1,x2):
return 2*x2-3
def jf(x1,x2):
return np.array([[f11(x1,x2),f12(x1,x2)],[f21(x1,x2),f22(x1,x2)]])

def phi1(x1,x2):
lmbd = calc_lambda()
return x1 -(f1(x1,x2) * lmbd[0,0] + f2(x1,x2) * lmbd[0,1])
def phi2(x1,x2):
lmbd = calc_lambda()
return x2 -(f1(x1,x2) * lmbd[1,0] + f2(x1,x2) * lmbd[1,1])
def phi11(x1,x2):
lmbd = calc_lambda()
return 1 -(f11(x1,x2) * lmbd[0,0] + f21(x1,x2) * lmbd[0,1])
def phi12(x1,x2):
```

```
lmbd = calc_lambda()
return -(f12(x1,x2) * lmbd[0,0] + f22(x1,x2) * lmbd[0,1])
def phi21(x1,x2):
lmbd = calc_lambda()
return -(f11(x1,x2) * lmbd[1,0] + f21(x1,x2) * lmbd[1,1])
def phi22(x1,x2):
lmbd = calc_lambda()
return 1 -(f12(x1,x2) * lmbd[1,0] + f22(x1,x2) * lmbd[1,1])
def jphi(x):
return np.array([[phi11(*x),phi12(*x)],[phi21(*x),phi22(*x)]])

def a1(x1,x2):
return np.array([[f1(x1,x2),f12(x1,x2)],[f2(x1,x2),f22(x1,x2)]])
def a2(x1,x2):
return np.array([[f11(x1,x2),f1(x1,x2)],[f21(x1,x2),f2(x1,x2)]])

def plots():
plt.figure()
x = np.linspace(-10,10,10000)
y11=[g1(i) for i in x]
y12=[-g1(i) for i in x]
y21=[g2(i) for i in x]
y22=[-g2(i) for i in x]
plt.plot(x,y11,color='red')
plt.plot(x,y12,color='red')
plt.plot(x,y21,color='blue')
plt.plot(x,y22,color='blue')
plt.xlim(-10.,10.)
plt.ylim(-10.,10.)
plt.grid()
plt.show()
return

def newton(eps):
x0=[1,g2(1)]
k=0
x_old=x0
while True:
x_new=np.zeros(2)
A11 = a1(x_old[0],x_old[1])
A22 = a2(x_old[0],x_old[1])
```

```
J11 = jf(x_old[0],x_old[1])
deta1=det(A11)
deta2=det(A22)
detj=det(J11)
x_new[0]=x_old[0] -(deta1/detj)
x_new[1]=x_old[1] -(deta2/detj)
k+=1
print(k,'итерация')
print('x = ',x_new)
epsk=norm(x_new-x_old)
print('epsk = ',epsk,'\n')
x_old=x_new
if epsk<eps:
break
return x_new

def calc_lambda():
shape = 2
current_j = jf(1.,1.)
inv_j = np.array([solve(current_j,i) for i in np.eye(shape)])
return np.transpose(inv_j)

def calc_q():
x1 = np.linspace(1.,1.4,100)
x2 = np.linspace(1.,1.4,100)
points = list(product(x1,x2))
vals = [norm(jphi(point),np.inf) for point in points]
q = np.max(vals)
return q

def mpi(eps):
x0=[0,1]
k=0
x_old=x0
q=calc_q()
while True:
x_new=np.array([phi1(*x_old),phi2(*x_old)])
k+=1
print(k,'итерация')
print('x = ',x_new)
print(q)
```

```
epsk = norm(x_new -x_old) * q / math.fabs(1-q)
print('epsk = ',epsk,'\n')
x_old=x_new
if k>10:
break
if epsk<eps:
break
return x_new

def main():
print('Лабораторная работа 2.2\nВариант 2')
print('\nМоя система:\n(x1^2+9)/x2 -27 = 0\n(x1-1.5)^2 + (x2-1.5)^2 -9 = 0')
print('\nВведите точность вычислений:')
eps = float(input())
print('\nМетод простых итераций')
print('\nlambda = ',calc_lambda())
print('\nq = ',calc_q())
mpi(eps)
print('\nМетод Ньютона')
newton(eps)
plots()
if __name__ == "__main__":
main()
```

1 Выводы

При выполнении лабораторной работы я изучила методы простых итераций и Ньютона решения нелинейных уравненийю Эти методы легко и удобно использовать при решении нелинейных уравнений и систем, но у них также есть недостатки. К примеру, недостатком метода Ньютона является необходимость вычисления производных на каждом шаге.