Лабораторная работа №2

Содержание

Лабораторная работа 2.1

Решение нелинейных уравнений

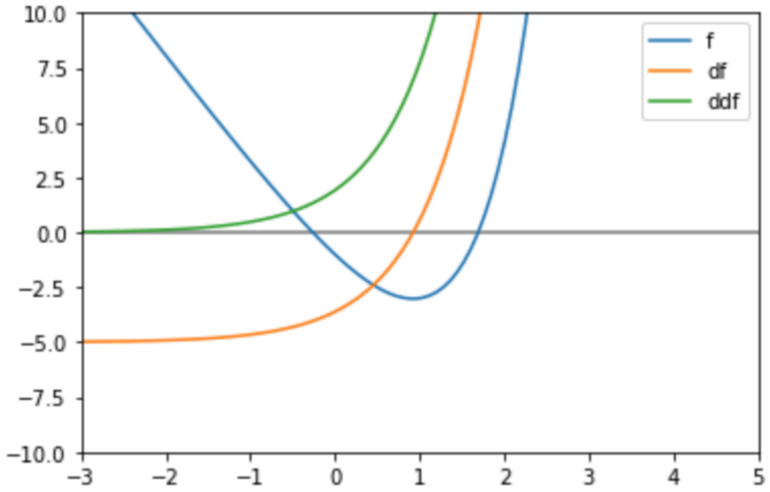
Задача: Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически).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

Уравнение:

$$4^x - 5x - 2 = 0$$

Решение: Метод Ньютона. При нахождении корня уравнения $f(x) = 0$ итерационный процесс определяется формулой

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}, \quad k = 0,1,2,\dots,$$



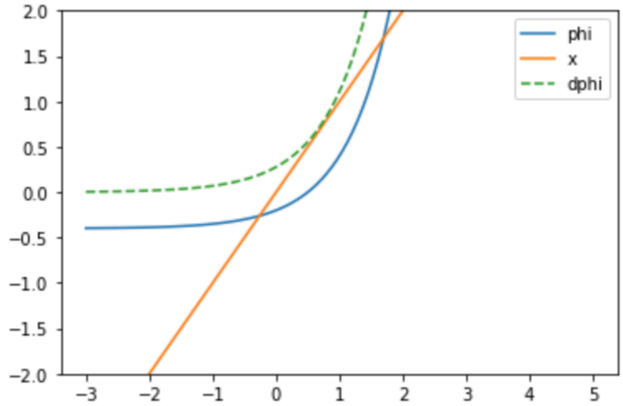
Решение будем искать на отрезке [-1,0], так как выполняются условия наличия корня:

$$f(-1) * f(0) < 0$$

$$x^0 = -0.5 : f(x^0) * f''(x^0) > 0$$

Метод простой итерации. При использовании метода простой итерации исходное уравнение заменяется уравнением с выделенным линейным членом $x = \varphi(x)$. Решение ищется путем построения последовательности

$$x^{(k+1)} = \varphi(x^{(k)}), \quad k = 0,1,2\ldots$$



Будем искать решение на отрезке [-1,0], так как выполнено условие существовании и дифференцируемости phi и:

$$\exists q : |\phi'(x)| \le q < 1$$

Работа программы:

```
Newton method:
x: -0.26781189376004433, k: 1, |x_cur - x|: 0.23218810623995567
x: -0.26065998943748053, k: 2, |x_cur - x|: 0.007151904322563796
x: -0.26065155661440526, k: 3, |x_cur - x|: 8.432823075277263e-06
x: -0.26065155660260353, k: 4, |x_cur - x|: 1.1801726262916645e-11
Simple iteration:
q = 0.2772588722239781 < 1
x: -0.3, k: 1, q/(1-q)*|x_cur - x|: 0.07672425480396929
x: -0.26804920892271056, k: 2, q/(1-q)*|x_cur - x|:
0.012257003179011712
x: -0.26207331722786337, k: 3, q/(1-q)*|x_cur - x|:
0.0022924791853818987
x: -0.26092593893197147, k: 4, q/(1-q)*|x_cur - x|:
0.00044015872365277053
x: -0.2607045511420817, k: 5, q/(1-q)*|x_cur - x|:
8.492906600995625e-05
x: -0.2606617936036601, k: 6, q/(1-q)*|x_cur - x|:
1.6402701363242687e-05
x: -0.26065353415156534, k: 7, q/(1-q)*|x_cur - x|:
3.168501535298095e-06
x: -0.2606519386209567, k: 8, q/(1-q)*|x_cur - x|:
6.120794848216399e-07
x: -0.26065163040011063, k: 9, q/(1-q)*|x_cur - x|:
1.1824007364810977e-07
x: -0.26065157085865476, k: 10, q/(1-q)*|x_cur - x|:
2.2841369160466985e-08

The value of the function at the found point:

5.7510481710210115e-08
```

Листинг программы:

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



def f(x):
    return 4**x-5*x-2

def df(x):
    return np.log(4)*np.float_power(4,x) - 5

def ddf(x):
    return np.log(4)**2*np.float_power(4,x)

def phi(x):
    return (np.float_power(4,x)-2)/5

def dphi(x):
    return np.log(4)*np.float_power(4,x)/5

def simpleIteration(phi, dphi, a, b, eps=0.001):
    q = max(abs(dphi(a)), abs(dphi(b)))
    x = (a + b) / 2
    k = 0
    go = True
    if q < 1:
        print(f'q = {q} < 1')
    else:
        return
    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 == 1000:
            break
    return x_cur

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, 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()

    plt.show()
```

```
"""# Метод простых итераций"""

x = np.linspace(-3,5,100)
plt.plot(x,phi(x))
plt.plot(x,x)
plt.plot(x,dphi(x),linestyle='dashed')
plt.ylim(-2,2)
plt.legend(['phi','x','dphi'])
plt.show()

"""Будем искать решение на отрезке [-1,0], так как
выполнено условие существовании и дифференцируемости
phi и:
$$ \exists q:|\phi'(x)| \leq q < 1$$
"""

print("Simple iteration:")
x_SI = simpleIteration(phi, dphi, -1, 0, eps =
0.0000001)

print("The value of the function at the found
point:")
f(x_SI)

"""# Метод Нютона"""

x = np.linspace(-3,5,100)
plt.plot(x,f(x))
plt.plot(x,df(x))
plt.plot(x,ddf(x))
plt.xlim(-3,5)
plt.ylim(-10,10)
plt.hlines(0,-4,10,color='grey')
plt.legend(['f','df','ddf'])
plt.show()

"""Решение будем искать на отрезке [-1,0], так как
выполняются условия наличия корня:
$$f(-1)*f(0)<0\\x^0=-0.5: f(x^0)*f''(x^0)>0
$$
```

```
"""

print("Newton method:")
x0 = -0.5
newton(f, df, x0, 0.0000001)
```

Лабораторная работа 2.2

Решение нелинейных систем уравнений

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.

Система уравнений:

$$\begin{cases} 3 \cdot x_1 - cos x_2 = 0 \\ 3 \cdot x_2 - e^{x_1} = 0 \end{cases}$$

Решение:

Метод Ньютона. Если определено начальное приближение $\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})^{\mathrm{T}}$, терационный процесс нахождения решения системы (2.11) методом Ньютона можно редставить в виде

$$\begin{cases} x_1^{(k+1)} = x_1^{(k)} + \Delta x_1^{(k)} \\ x_2^{(k+1)} = x_2^{(k)} + \Delta x_2^{(k)} \\ \dots\dots\dots\dots\dots \\ x_n^{(k+1)} = x_n^{(k)} + \Delta x_n^{(k)} \end{cases} \qquad k = 0,1,2,... \qquad (2.13)$$

це значения приращений $\Delta x_1^{(k)}$, $\Delta x_2^{(k)}, \ldots,$ $\Delta x_n^{(k)}$ определяются из решения системы инейных алгебраических уравнений, все коэффициенты которой выражаются через звестное предыдущее приближение $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, ..., x_n^{(k)})$

$$\begin{cases} f_1(\mathbf{x}^{(k)}) + \frac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \frac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \frac{\partial f_1(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ f_2(\mathbf{x}^{(k)}) + \frac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \frac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \frac{\partial f_2(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \\ \dots\dots\dots\dots\dots \\ f_n(\mathbf{x}^{(k)}) + \frac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_1}\Delta x_1^{(k)} + \frac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_2}\Delta x_2^{(k)} + ... + \frac{\partial f_n(\mathbf{x}^{(k)})}{\partial x_n}\Delta x_n^{(k)} = 0 \end{cases} \qquad (2.14)$$

В векторно-матричной форме расчетные формулы имеют вид

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{\Delta x}^{(k)} \qquad k = 0,1,2,... \qquad (2.15)$$

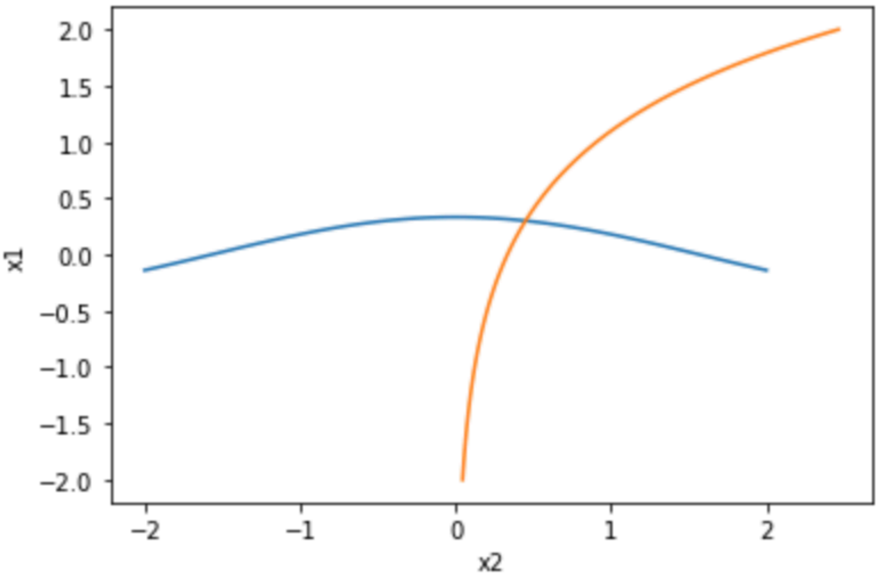
где вектор приращений $\mathbf{\Delta x}^{(k)} = \begin{pmatrix} \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ \dots \\ \Delta x_n^{(k)} \end{pmatrix}$ находится из решения уравнения

$$\mathbf{f}(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)})\mathbf{\Delta x}^{(k)} = \mathbf{0} \tag{2.16}$$

Метод простой итерации. При использовании метода простой итерации исходная системы уравнений приводится в эквивалентной системе специльного вида:

$$\mathbf{x} = \varphi(\mathbf{x}), \varphi(\mathbf{x}) = (\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_n(\mathbf{x}))^T$$

x2



Работа программы:

```
Simple iterations method

Sufficient condition is met : q = 0.9060939428196817 < 1
x1 = 0.2996871823843739, x2 = 0.4588301482684969 4, on 6 iteration
Sufficient condition is met : q = 0.9060939428196817 < 1
x1 = 0.3002303299151301, x2 = 0.45003225019865695, on 9 iteration
Sufficient condition is met : q = 0.9060939428196817 < 1
x1 = 0.30014082940172965, x2 = 0.45001789487707516, on 11
iteration
Sufficient condition is met : q = 0.9060939428196817 < 1
x1 = 0.30014666976107685, x2 = 0.45001883157442907, on 13
iteration
```

```
Newton method

x1 = 0.30199528609907034, x2 = 0.44075556239759067, on 2 iteration
x1 = 0.3001585522523673, x2 = 0.45002352253691436, on 3 iteration
x1 = 0.3001585522523673, x2 = 0.45002352253691436, on 3 iteration
x1 = 0.3001463120293105, x2 = 0.45001877416844893, on 4 iteration
```

Листинг программы:

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



def f_show1(x2):
  return math.cos(x2)/3

def f_show2(x2):
  return math.log(3*x2)

def d_phi1(x2):
    return -math.sin(x2)/3

def d_phi2(x1):
    return math.exp(x1)/3

def phi2(x1):
    return math.exp(x1)/3

def phi1(x2):
    return math.cos(x2)/3

def f1(x1, x2):
    return 3*x1 - math.cos(x2)

def f2(x1, x2):
    return 3*x2 - math.exp(x1)
```

```
def d_1fun_x1(x1, x2):
    return 3

def d_1fun_x2(x1, x2):
    return math.sin(x2)

def d_2fun_x1(x1, x2):
    return - math.exp(x1)

def d_2fun_x2(x1, x2):
    return 3

def det_a1(x1, x2):
    return f1(x1, x2) * d_2fun_x2(x1, x2) - f2(x1,
x2) * d_1fun_x2(x1, x2)

def det_a2(x1, x2):
    return f2(x1, x2) * d_1fun_x1(x1, x2) - f1(x1,
x2) * d_2fun_x1(x1, x2)

# Якобиан
def det_J(x1, x2):
    return d_1fun_x1(x1, x2) * d_2fun_x2(x1, x2) -
d_2fun_x1(x1, x2) * d_1fun_x2(x1, x2)

def minus_vectors(a, b):
    n = len(a)
    res = np.zeros(n)

    for i in range(0, n):
        res[i] = a[i] - b[i]

    return res

def norma_vector(a):
    sum = 0
    n = len(a)
    for i in range(0, n):
        sum += a[i] * a[i]
```

```
    sum = math.sqrt(sum)
    return sum

def SimpleIterationsMethod(epsilon, n):
    xk = np.zeros(n)
    xk_tmp = np.zeros(n)

    xk[0] = 1.8
    xk[1] = 1
    x2a = 1.7
    x2b = 1.8
    x1a = 0.5
    x1b = 1
    #производные на участках мотононные, поэтому 
найдем максимум производных так:
    dphi1_max = max(d_phi1(x2a), d_phi1(x2b))
    dphi2_max = max(d_phi2(x1a), d_phi2(x1b))

    q = max(abs(dphi1_max), abs(dphi2_max))

    #проверка условия наличия решения
    if q > 1:
        print(f"Sufficient condition is not met : q = 
{q} > 1")
        return

    print(f"Sufficient condition is met : q = {q} < 
1")

    counter = 0
    #Вычисляем корень по рекурентной формуле простых 
итераций
    while ((q / (1 - q)) * 
norma_vector(minus_vectors(xk, xk_tmp)) > epsilon):
        if counter != 0:
            xk[0] = xk_tmp[0]
            xk[1] = xk_tmp[1]

        counter = counter + 1
        xk_tmp[0] = phi1(xk[1])
```

```
        xk_tmp[1] = phi2(xk[0])

    print(f"x1 = {xk[0]}, x2 = {xk[1]}, on {counter}
iteration")

def newtonMethod(x0, epsilon, n):

    xk = np.zeros(n)
    xk_tmp = np.zeros(n)
    detk = np.zeros(n)

    xk_tmp = x0

    counter = 0

    #Вычисляем корень по рекурентной формуле Ньютона
    while norma_vector(minus_vectors(xk, xk_tmp)) >
epsilon:
        xk[0] = xk_tmp[0]
        xk[1] = xk_tmp[1]

        detk[0] = det_a1(xk[0], xk[1]) / det_J(xk[0],
xk[1])
        detk[1] = det_a2(xk[0], xk[1]) / det_J(xk[0],
xk[1])

        xk_tmp = minus_vectors(xk, detk)
        counter = counter + 1

    print(f"x1 = {xk[0]}, x2 = {xk[1]}, on {counter}
iteration")

"""# Метод простых итераций"""

#Построим графики
x = np.linspace(-2,2,100)
phi1_ = []
phi2_ = []
for x_ in x:
  phi1_.append(phi1(x_))
```

```python
  phi2_.append(phi2(x_))
plt.plot(x,phi1_)
plt.plot(phi2_,x)
plt.xlabel('x2')
plt.ylabel('x1')
#положительное решение находится в квадрате 0 < x2 <
1, 0 < x1 < 0.5
#за начальное приближение возьмем точку (0.5,0.5)

epsilon = [0.1, 0.001, 0.0001, 0.00001]

# Посчитаем решения для различных точностей методом
простых итераций
print("\n\t\tSimple iterations method\n")
for eps in epsilon:
    SimpleIterationsMethod(eps, 2)

"""# Метод Ньютона"""

#Построим графики
x = np.linspace(0.05,1,100)
f1_ = []
f2_ = []
for x_ in x:
  f1_.append(f_show1(x_))
  f2_.append(f_show2(x_))
plt.plot(x,f1_)
plt.plot(x,f2_)
plt.xlabel('x2')
plt.ylabel('x1')

epsilon = [0.1, 0.001, 0.0001, 0.00001]

# Посчитаем решения для различных точностей методом
Ньютона
print("\n\t\tNewton method\n")
for eps in epsilon:
    newtonMethod([0.5,0.5],eps, 2)
```