2019 год

Практикум по численным методам: методы решения систем уравнений

Мусаева Аида, группа 208

1 Метод Ньютона для решения систем нелинейных уравнений

Данный метод состоит в построении сдедующей итерационной последовательности:
$\vec{x}^{(k+1)} = \vec{x}^{(k)} - [f_{\vec{x}}(\vec{x}^{(k)})]^{-1} * f(\vec{x}^{(k)})$
Задание:

$$\begin{cases} \mathrm{tg}(xy) = x^2 \\ 0,8x^2 + 2y^2 = 1 \end{cases}$$

1.1 Код программмы

```
import numpy as np
def f(x):
    return [np.tan(x[0] * x[1]) - x[0] ** 2, 0.7 * x[0] ** 2 + 2 * x[1] ** 2 - 1]

def g(x):
    return [[x[1] / np.cos(x[0] * x[1]) ** 2 - 2 * x[0], 1 / np.cos(x[0] * x[1]) ** 2],
           [1.4 * x[0], 4 * x[1]]]

def Newton(x, f, g, eps):
    k = 0
    while (True):
        k += 1;
        x_ = x - np.linalg.inv(g(x)) @ f(x)
        if np.linalg.norm(x_ - x) < eps:
            return x_, k
        x = x_

x = [1, 0.5]
res = Newton(x, f, g, 1e-6, )
print(res)
```

1.2 Результат работы программы

$$\mathbf{X} = \begin{pmatrix} 0.63102538 \\ 0.6005268 \end{pmatrix}$$

9 итераций

2 Метод Гаусса для решения систем линейных уравнений

Для осуществления данного метода требуется дополнить матрицу A вектором b и для $(A|b)$ произвести премой и обратный ход метода Гаусса.
Задание:

$$\mathbf{A} = \begin{pmatrix} 3,40 & 3,26 & 2,90 \\ 2,64 & 2,39 & 1,96 \\ 4,64 & 4,32 & 3,85 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 13,05 \\ 10,30 \\ 17,89 \end{pmatrix}$$

2.1 Код программмы

```
import numpy as np
A = np.array([[3.4, 3.26, 2.90],
    [2.64, 2.39, 1.96],
    [4.64, 4.32, 3.85]])
b = np.array([[13.05], [10.30], [17.89]])

def Gauss(A, b):
    n = A.shape[0]
    for i in range(n):
        b[i] /= A[i][i]
        A[i] /= A[i][i]
        for j in range(i + 1, n):
            b[j] -= A[j][i] * b[i]
            A[j] -= A[j][i] * A[i]

    for i in range(n - 1, -1, -1):
        for j in range(i - 1, -1, -1):
            b[j] -= b[i] * A[j][i]
    return b
res = Gauss(A.copy(), b.copy())
print(res)
```

2.2 Результат работы программы

$$\mathbf{X} = \begin{pmatrix} 4.46127093 \\ -0.24673211 \\ -0.45309465 \end{pmatrix}$$

3 Метод простых итераций для решения систем линейных уравнений

Системы $Cx = d$ требуется преобразовать к виду $x = b + Ax$. Затем вычислить решение как предел последовательности:
$x^{(k+1)} = b + Ax^{(k)}$
Задание:

$$\mathbf{A} = \begin{pmatrix} 10,8000 & 0,0475 & 0,0576 & 0,0676 \\ 0,0321 & 9,9000 & 0,0523 & 0,0623 \\ 0,0268 & 0,0369 & 9,0000 & 0,0570 \\ 0,0215 & 0,0316 & 0,0416 & 8,1000 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 12,1430 \\ 13,0897 \\ 13,6744 \\ 13,8972 \end{pmatrix}$$

3.1 Код программмы

```
import numpy as np
C = np.array([[10.8000, 0.0475, 0.0576, 0.0676],
             [0.0321, 9.9000, 0.0523, 0.0623],
             [0.0268, 0.0369, 9.0000, 0.0570],
             [0.0215, 0.0316, 0.0416, 8.1000]])

d = np.array([[12.1430], [13.0897], [13.6744], [13.8972]])
import numpy as np
C = np.array([[10.8000, 0.0475, 0.0576, 0.0676],
             [0.0321, 9.9000, 0.0523, 0.0623],
             [0.0268, 0.0369, 9.0000, 0.0570],
             [0.0215, 0.0316, 0.0416, 8.1000]])

d = np.array([[12.1430], [13.0897], [13.6744], [13.8972]])
def Iterative(C, d, x, eps):
    k = 0
    n = C.shape[0]
    while (True):
        x_ = np.zeros(n)

        for i in range(n):
            x_[i] = d[i] / C[i][i]
            for j in range(n):
                if i != j:
                    x_[i] -= C[i][j] / C[i][i] * x[j]

        k += 1
        if np.linalg.norm(x_ - x) < eps:
            return [x_, k]
        x = x_

x0 = np.array([0, 0, 0, 0])
res = Iterative(C, d, x0, 1e-15)

print(res)
```

3.2 Результат работы программы

$$\mathbf{X} = \begin{pmatrix} 1.09999342 \\ 1.30000297 \\ 1.50000551 \\ 1.70000862 \end{pmatrix}$$

10 итераций

4 Обращение симметрично положительно определенной матрицы методом квадратного корня

Для осуществления данного метода нужно представить матрицу в виде $A = L * L^T$ (треугольное разложение Холецкого). Тогда $A^{-1} = (L^T)^{-1}$ Вычисление элементов матрицы L и $P = L^{-1}$ производится по формулам:

$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$
$l_{ij} = \frac{a_{ii} - \sum_{k=1}^{j-1} l_{ik}^2 l_{kj}^2}{l_{ij}}$
$p_{ii} = \frac{1}{l_{ii}}$
$p_{ij} = -\frac{\sum_{k=1}^{j-1} l_{jk} p_{ki}}{l_{jj}}$

Задание:

$$\mathbf{A} = \begin{pmatrix} 0,0936 & 0,3690 & 0,6444 & 0,9198 \\ 0,3690 & 7,2722 & 10,5284 & 13,7846 \\ 0,6444 & 10,5284 & 35,8169 & 44,7593 \\ 0,9198 & 13,7846 & 44,7593 & 100,0091 \end{pmatrix}$$

4.1 Код программмы

```
import numpy as np
A = np.array([[0.0936, 0.3690, 0.6444, 0.9198],
             [0.3690, 7.2722, 10.5284, 13.7846],
             [0.6444, 10.5284, 35.8169, 44.7593],
             [0.9198, 13.7846, 44.7593, 100.0091]])

def L(A):
    L = np.zeros(A.shape)
    n = A.shape[0]

    for i in range(n):
        for j in range(i + 1):
            tmp = sum(L[i][k] * L[j][k] for k in range(j))

            if (i == j):
                    L[i][j] = (A[i][i] - tmp) ** 0.5
            else:
                    L[i][j] = (1.0 / L[j][j] * (A[i][j] - tmp))
    return L

def Inverse(A):
    l = L(A)
    P = np.zeros(A.shape)
    n = A.shape[0]
    for i in range(n):
        P[i][i] = 1 / l[i, i]
        for j in range(i+1, n):
            tmp = 0
            for k in range(j):
                tmp += l[j, k] * P[k][i]
```

```
            P[i][j] = - tmp / l[j, j]
    return P.T @ P

res = Inverse(A)
print(np.linalg.inv(np.linalg.cholesky(A)).T @ np.linalg.inv(np.linalg.cholesky(A)))
print(res)
```

4.2 Результат работы программы

$$\mathbf{A^{-1}} = \begin{pmatrix} 1.34930399e+01 & -5.76008888e-01 & -3.98818296e-02 & -2.68551880e-02 \\ -5.76008888e-01 & 2.64487664e-01 & -6.45469737e-02 & -2.26945665e-03 \\ -3.98818296e-02 & -6.45469737e-02 & 8.17653156e-02 & -2.73307206e-02 \\ -2.68551880e-02 & -2.26945665e-03 & -2.73307206e-02 & 2.27908148e-02 \end{pmatrix}$$