# Практикум по вычислительным методам: методы решения систем уравнений

Рогачёв Юрий Витальевич, 208 группа

26 апреля, 2019

# 1 Решение системы нелинейных уравнений методом Ньютона

Решение систем нелинейных уравнений методом ньютона состоит в построении следующей итерационной последовательности
$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \left[\frac{\partial f(\vec{x}^{k})}{\partial \vec{x}}\right]^{-1} * f(\vec{x}^{(k)})$

```
In [2]: def f(x):
            return [np.tan(x[0] * x[1] + 0.2) - x[0] ** 2, 0.6 * x[0] ** 2 + 2 * x[1] ** 2 - 1]
```

```
In [3]: def g(x):
            return [[x[1] / np.cos(x[0] * x[1] + 0.2) ** 2 - 2 * x[0], 1 / np.cos(x[0] * x[1] +
                    [1.6 * x[0], 4 * x[1]]]
```

```
In [4]: # n - количество уравнений
        # x - начальное приближение
        # f - f(x)
        # g - g(x) вычисляет матрицу производных

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

```
In [5]: x = [1, 1]
        g(x)
```

```
Out[5]: [[5.615963967207052, 7.615963967207052], [1.6, 4]]
```

```
In [6]: ans = Newts(x, f, g, 1e-6, )
        ans
```

```
Out[6]: (array([0.87646187, 0.51917663]), 10)
```

```
In [7]: f(ans[0])
```

```
Out[7]: [-4.940548370413467e-08, -1.3592064251888303e-08]
```

# 2 Решение систем линейных уравнений методом Гаусса

В данном методе мы дополняем матрицу $A$ вектором $b$ и имеем $(A|b)$. После чего мы осуществляем прямой и обратный ход метода Гаусса

```
In [8]: A = np.array([[2.16, 1.96, 1.56],
            [3.55, 3.23, 2.78],
            [4.85, 4.47, 3.97]])
```

```
In [9]: b = np.array([[13.16], [21.73], [29.75]])
```

```
In [10]: def gaus(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
```

```
In [11]: ans = gaus(A.copy(), b.copy())
         ans
```

```
Out[11]: array([[ 6.06990622],
                [-0.35959079],
                [ 0.48320546]])
```

```
In [12]: A @ ans - b
```

```
Out[12]: array([[1.77635684e-15],
                [0.00000000e+00],
                [0.00000000e+00]])
```

# 3 Решение систем уравнений методом простых итераций

Систему уравнений

$Cx = d$

Преобразуем к виду

$x = b + Ax$

И вычисляем решение как предел последовательности

$x^{(k+1)} = b + Ax^{(k)}$

```
In [13]: C = np.array([[13.4, 0.581, 0.702, 0.0822],
                       [0.0408, 12.5, 0.65, 0.77],
                       [0.0356, 0.0477, 11.6, 0.718],
                       [0.0304, 0.0425, 0.0546, 10.7]])
```

```
In [14]: d = np.array([[17.7828], [19.0599], [19.9744], [20.5261]])
```

```
In [15]: def simple_iter(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_
```

```
In [16]: x = np.array([1, 1, 1, 1])
```

```
In [17]: ans = simple_iter(C, d, x, 1e-15)
         ans[0] = ans[0].reshape(-1, 1)
         ans
```

```
Out[17]: [array([[1.17457001],
                 [1.32086943],
                 [1.59519211],
                 [1.90160361]]), 13]
```

```
In [18]: C @ ans[0] - d
```

```
Out[18]: array([[0.],
                [0.],
                [0.],
                [0.]])
```

# 4 Обращение симметрично положительно определенной матрицы методом квадратного корня

В данном методе мы сначала должны представить матрицу в виде $A = L * L^T$ (треугольное разложение Холецкого), тогда $A^{-1} = (L^T)^{-1} * L^{-1}$. Вычисление элементов матриц $L$ и $L^T$ производим по формулам:

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

```
In [90]: A = np.array([[0.1399, 0.5513, 0.9627, 1.3741],
                       [0.5513, 10.8634, 15.7276, 20.5918],
                       [0.9627, 15.7276, 53.5043, 66.8627],
                       [1.3741, 20.5918, 66.8627, 149.3963]])
```

```
In [85]: def L(A):
             L = np.ndarray(A.shape)
             n = A.shape[0]
             for i in range(n):
```

```
            for j in range(n):
                if i == j:
                    buf = A[i][i]
                    for k in range(i-1):
                        buf -= L[i][k] ** 2
                    L[i][i] = buf ** 0.5
                elif j < i:dd
                    buf = A[i][j] / L[j][j]
                    for k in range(j-1):
                        buf -= L[i][k] * L[j][k] / L[j][j]
                    L[i][j] = buf
                else:
                    L[i][j] = 0
        return L
```

```
In [86]: def inverse_matrix(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):
                     buf = 0
                     for k in range(j):
                         buf += l[j,k] * P[k][i]
                     P[i][j] = - buf / l[j,j]

             return P.T @ P
```

```
In [87]: ans = inverse_matrix(A.copy())
```

```
In [92]: A @ ans
```

```
Out[92]: array([[ 1.00000000e+00, -2.77555756e-17, -3.46944695e-17,
                  2.77555756e-17],
                [ 0.00000000e+00,  1.00000000e+00,  1.11022302e-16,
                 -1.11022302e-16],
                [-8.88178420e-16,  0.00000000e+00,  1.00000000e+00,
                  0.00000000e+00],
                [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
                  1.00000000e+00]])
```