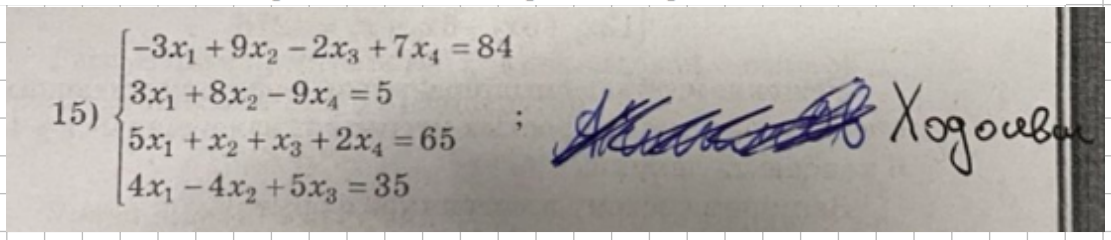
№16 (вариант 15)

Ходосевич Данила
3 МО, 2 группа

16. **Дана система линейных алгебраических уравнений с четырьмя неизвестными.**
   A. **Преобразовать систему к виду, пригодному для применения принципа сжимающих операторов.**
   B. **Методом простых итераций найти приближенные решения с точностью 10^{-2} и с точностью 10^{-4}, используя априорную и апостериорную оценки числа итераций.**
   C. **Найти точное решение системы и сравнить с приближенными.**



$$A = \begin{pmatrix} -3 & 9 & -2 & 7 \\ 3 & 8 & 0 & -9 \\ 5 & 1 & 1 & 2 \\ 4 & -4 & 5 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 84 \\ 5 \\ 65 \\ 35 \end{pmatrix}$$

$\det A = 2833$, $\det A \neq 0 \Rightarrow$ матрица не вырождена имеет единств. решение

$$x_1 = \frac{\Delta_1}{\Delta} = \frac{19831}{2833} = 7$$

$$x_2 = \frac{\Delta_2}{\Delta} = \frac{19831}{2833} = 7$$

$$\vec{x} = \begin{pmatrix} 7 \\ 7 \\ 7 \\ 8 \end{pmatrix}$$

$$x_3 = \frac{\Delta_3}{\Delta} = \frac{19831}{2833} = 7$$

$$x_4 = \frac{\Delta_4}{\Delta} = \frac{22664}{2833} = 8$$

A) Необходимо преобразовать СЛАУ к виду $F(x) = X = CX + D$, где $F(x)$ - сжим. отобр. Сделаем A симметрической:

$A^T | AX = B \Rightarrow A^T AX = A^T B$

Все собств. значения $A^T A$ положе.

$$A^T = \begin{pmatrix} -3 & 3 & 5 & 4 \\ 9 & 8 & 1 & -4 \\ -2 & 0 & 1 & 5 \\ 7 & -9 & 2 & 0 \end{pmatrix}$$

$$A^T A = \begin{pmatrix} 59 & -14 & 31 & -38 \\ -14 & 162 & -37 & -7 \\ 31 & -37 & 30 & -12 \\ -38 & -7 & -12 & 134 \end{pmatrix}$$

```python
import numpy as np
B = np.array([[-3, 9, -2, 7],
              [3, 8, 0, -9],
              [5, 1, 1, 2],
              [4, -4, 5, 0]])

C = np.array([[-3, 3, 5, 4],
              [9, 8, 1, -4],
              [-2, 0, 1, 5],
              [7, -9, 2, 0]])
symmetric = np.matmul(C,B)
print(symmetric)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[[ 59 -14  31 -38]
 [-14 162 -37  -7]
 [ 31 -37  30 -12]
 [-38  -7 -12 134]]
```

$Jet(A^TA - \lambda E) = 0$

$\lambda_1 = 176.282$ $\lambda_2 = 152.561$

$\lambda_3 = 50.214$ $\lambda_4 = 5.943$

```python
import numpy as np

A = np.array([[59, -14, 31, -38],
              [-14, 162, -37, -7],
              [31, -37, 30, -12],
              [-38, -7, -12, 134]])

eigenvalues = np.linalg.eigvals(A)
print(eigenvalues)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
[  5.94318885  50.2136514  152.56080713 176.28235263]
```

Для сжатия необходимо, чтобы все собств. значения $\lambda < 1$. Разделим уравнение $A^TAx = A^TB$ на $\lambda_{max} = \lambda_1$

$\frac{A^TAx}{\lambda_1} = \frac{A^TB}{\lambda_1}$, преобразуем его к виду: $X = CX + D$

$$\frac{A^TAx}{\lambda_1} + x - x = \frac{A^TB}{\lambda_1} \Rightarrow x = x - \frac{A^TAx}{\lambda_1} + \frac{A^TB}{\lambda_1} \Rightarrow x = \underbrace{\left(E - \frac{A^TA}{\lambda_1}\right)}_{C} x + \underbrace{\frac{A^TB}{\lambda_1}}_{D}$$

$$C = \begin{bmatrix} 0.66530966 & 0.07941805 & -0.17585424 & 0.21556327 \\ 0.07941805 & 0.08101975 & 0.20989055 & 0.03970902 \\ -0.17585424 & 0.20989055 & 0.82981847 & 0.06807261 \\ 0.21556327 & 0.03970902 & 0.06807261 & 0.23985584 \end{bmatrix}$$

$$D = \begin{bmatrix} 1.29337961 \\ 4.09002937 \\ 0.40843567 \\ 3.81773893 \end{bmatrix}$$

```python
import numpy as np

Y = np.array([[84],
              [5],
              [65],
              [35]])
Q = np.array([[-3, 3, 5, 4],
              [9, 8, 1, -4],
              [-2, 0, 1, 5],
              [7, -9, 2, 0]])

D = np.matmul(Q,Y) / (176.28235263)
print(D)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[[1.29337961]
 [4.09002937]
 [0.40843567]
 [3.81773893]]
```

```python
import numpy as np
B = np.array([[-3, 9, -2, 7],
              [3, 8, 0, -9],
              [5, 1, 1, 2],
              [4, -4, 5, 0]])
Q = np.array([[-3, 3, 5, 4],
              [9, 8, 1, -4],
              [-2, 0, 1, 5],
              [7, -9, 2, 0]])

symmetric = np.matmul(Q,B)

C = np.identity(4) - symmetric/(176.28235263)
print(C)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

[[ 0.66530966  0.07941805 -0.17585424  0.21556327]
 [ 0.07941805  0.08101975  0.20989055  0.03970902]
 [-0.17585424  0.20989055  0.82981847  0.06807261]
 [ 0.21556327  0.03970902  0.06807261  0.23985584]]
```

Собств. матрицы C: $1 - \frac{\lambda_1}{\lambda_1}$; $1 - \frac{\lambda_2}{\lambda_1}$; $1 - \frac{\lambda_3}{\lambda_1}$; $1 - \frac{\lambda_4}{\lambda_1}$

Наибольшее значение λ: $\lambda_1(C) = 0,9663$

$\lambda < 1$, λ - коэфф. сжатия

=> Найдем решение методом итераций

```python
C = np.identity(4) - symmetric/(176.28235263)

c_eigvals = np.linalg.eigvals(C)
print(max(c_eigvals))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

0.9662859681812652
```

B) С помощью метода простых итераций последовательно найдем решение, приближающееся к точному, исп. рекуррентную формулу: $x_n = Cx_n + d$, $n = 1, 2, 3, \dots$

Найти приближенное решение – довести итерации до $x_n: \rho(x_n, x) \le \varepsilon$, где x – точное решение, ε – заданная точность

Точное решение неизвестно => используем оценку точности:

$\rho(x_n, x) \le \frac{\alpha^n}{1-\alpha} \cdot \rho(x_0, x_1)$ – априорная оценка

$\rho(x_n, x) \le \frac{\alpha}{1-\alpha} \cdot \rho(x_n, x_{n-1})$ – апостериорная оценка

Найдем решение с помощью априорной оценки:

$$N_{апр} = \left[\log_\alpha \frac{\varepsilon(1-\alpha)}{\rho(x_0, x_1)} \right] + 1$$

Возьмем $x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$ Тогда $x_1 = Cx_0 + D = \begin{pmatrix} 1,293 \\ 4,09 \\ 0,408 \\ 3,817 \end{pmatrix}$

$\rho(x_0, x_1) = \sqrt{\sum_{i=1}^{n} (x_{i0} - x_{i1})^2}$ – евклидова метрика в пр-ве $\mathbb{R}^4$

```
6   x0 = [0, 0, 0, 0]
7   x1 = [1.293, 4.09, 0.408, 3.817]
8
9   ro = calculate_ro(x0, x1)
10  print(f"ρ(x₀, x₁) = {ro:.4f}")
11  5.757160932265139
```

$\rho(x_0, x_1) = 5,76$

Для $\varepsilon = 10^{-2}$ $N_{apr} = 284$

```
31  alpha = 0.9663
32  eps_2 = 0.01
33  p = 5.757160932265139
34  n_apr_2 = np.emath.logn(alpha, (1-alpha) * eps_2/p)
35  print(n_apr_2)
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
284.2942404567581
```

Для $\varepsilon = 10^{-4}$ $N_{apr} = 419$

```
31  alpha = 0.9663
32  eps_2 = 0.0001
33  p = 5.757160932265139
34  n_apr_2 = np.emath.logn(alpha, (1-alpha) * eps_2/p)
35  print(n_apr_2)
36
37
38
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
418.63043412915704
```

Вычислим апостериарную оценку:

Для $\varepsilon = 10^{-2}$:

```
Eps:  0.01
Start dist from (0,0,0,0):  5.757006279064861 5
Num steps(from aprior estimate):  284
posterior estimate:  188
result
x1 = 7.0023377293720745
dist (result- > analitic):  0.0046823246653099225
x2 = 6.999278117305979
dist (result- > analitic):  0.0046823246653099225
x3 = 6.996017649935161
dist (result- > analitic):  0.0046823246653099225
x4 = 8.00028099642606
dist (result- > analitic):  0.0046823246653099225
```

$N_{apost} = 188$

Для $\varepsilon = 10^{-4}$:

```
Eps:  0.0001
Start dist from (0,0,0,0):  5.757006279064861 5
Num steps(from aprior estimate):  419
posterior estimate:  323
result
x1 = 7.00002321769608
dist (result- > analitic):  4.6312669799584096e-05
x2 = 6.99999286086076
dist (result- > analitic):  4.6312669799584096e-05
x3 = 6.999960664227168
dist (result- > analitic):  4.6312669799584096e-05
x4 = 8.000002744383231
dist (result- > analitic):  4.6312669799584096e-05
```

$N_{apost} = 323$

```
D=[1.29337961, 4.09002937, 0.40843567, 3.81773893]

C=[[ 0.66530966, 0.07941805, -0.17585424, 0.21556327],
[0.07941805, 0.08101975, 0.20989055 , 0.03970902],
 [-0.17585424 , 0.20989055 , 0.82981847 , 0.06807261],
 [ 0.21556327, 0.03970902, 0.06807261, 0.23985584]]
x_analitic = [7, 7, 7, 8]
x_cur_iter = [0, 0, 0, 0]
x_past_iter = [0, 0, 0, 0]
alpha = 0.9838420715662617
eps = 10**(-4)
print("Eps: ", eps)
num_iter = 419



def dist(a,b):
    dist_ab = 0
    for i in range(4):
        dist_ab += (a[i] - b[i])**2
    return dist_ab**(0.5)
print("Start dist from (0,0,0,0): ", dist(x_past_iter,D))

print("Num steps(from aprior estimate): ", num_iter)
for i in range(num_iter):
    x_past_iter = x_cur_iter
    x_cur_iter = [0, 0, 0, 0]
    flag = True

    for j in range (4):
        x_cur_iter[j] += D[j]
        for k in range(4):
            x_cur_iter[j] += C[j][k] * x_past_iter[k]

    for j in range(4):
        posteriori = alpha*dist(x_cur_iter,x_past_iter)/(1-alpha)
        if posteriori > eps:
            flag = False

    if flag:
        print("posterior estimate: ", i)
        break

print("result")
for i in range(4):
    print(f"x{i+1} = {x_cur_iter[i]}")
    print("dist (result- > analitic): ", dist(x_cur_iter,x_analitic))
```