МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики
Кафедра вычислительной математики

Отчет по лабораторным работам №5-8
по темам: “Метод Данилевского, итерационный метод вращений, метод Крылова, степенной метод”

Вариант 9

Выполнила
Бодина Виктория Александровна
студентка 3 курса 7а группы

Преподаватель
Будник Анатолий Михайлович

Минск 2024

Методом Данилевского построить собственный многочлен $P_n(\lambda_i)$ матрицы A^TA

Введение

Дана матрица размерности n x n следующего вида (n = 5):

```
A = np.array([[0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
        [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
        [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
        [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
        [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])
```

Необходимо методом Данилевского построить собственный многочлен $P_n(\lambda_i)$ матрицы A^TA, вычислить невязки $\phi_i = P_n(\lambda_i)$ и $r_i = A^TAx_i - \lambda x_i$ и оценить их значения. В конце провести сравнительный анализ полученных собственных значений и собственных векторов с точки зрения точности и экономичности используемых методов.

Алгоритм

Идея алгоритма заключается в том, чтобы за (n-1) итерацию получить каноническую форму Фробениуса матрицы A

$$\Phi = \begin{pmatrix} p_1 & p_2 & p_3 & \cdots & p_{n-1} & p_n \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ & & \ldots & \ldots & & \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{pmatrix}.$$

следующими преобразованиями подобия:

$A_{i+1} = M^{-1}_{n-i}\, A_i\, M_{n-i}$, $i = 1 \ldots n$, *где* за A_1 обозначим матрицу A^TA

M^{-1}_{n-1} *и* M_{n-1} выглядит так:

$$M_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ & \dots & \dots & & \\ a_{n1} & a_{n2} & \dots & a_{nn-1} & a_{nn} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}. \qquad M_{n-1} = \begin{pmatrix} 1 & \dots & 0 & 0 & 0 \\ & \dots & \dots & & \\ 0 & \dots & 1 & 0 & 0 \\ -\frac{a_{n1}}{a_{nn-1}} & \dots & -\frac{a_{nn-2}}{a_{nn-1}} & \frac{1}{a_{nn-1}} & -\frac{a_{nn}}{a_{nn-1}} \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}.$$

где a_{ij} – элементы матрицы $A^T A$

после первого преобразования нижняя строка примет такой вид:

$$A_1 = M_{n-1}^{-1} A M_{n-1} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n-1}^{(1)} & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \dots & a_{2n-1}^{(1)} & a_{2n}^{(1)} \\ & \dots & \dots & & \\ a_{n-11}^{(1)} & a_{n-12}^{(1)} & \dots & a_{n-1n-1}^{(1)} & a_{n-1n}^{(1)} \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

после чего проделываем аналогично еще n - 2 итерации со строчками выше, не изменяя нижние

Придя к канонической форме Фробениуса, из первой строки можно будет выписать коэффициенты характеристического многочлена.

Для оценки невязок использую евклидову норму $\|x\| = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

Для подсчета невязки $r_i = A^T A x_i - \lambda x_i$ использую собственные векторы из метода Крылова

Листинг программы

```
import numpy as np

def danilevsky(A):
   n = A.shape[0]

   for k in range(n - 1):
       M = np.eye(n)
       M[n - 2 - k, :] = A[n - 1 - k, :]   # M^-1
       A = M @ A
```

```
        denominator = -1 / A[n - 1 - k, n - 2 - k]

        for i in range(n):
            if i == n - 2 - k:
                M[n - 2 - k, i] = -denominator
            else:
                M[n - 2 - k, i] *= denominator

        A = A @ M

    coefficients = -A[0, :]
    coefficients = np.insert(coefficients, 0, 1)

    return coefficients


A = np.array([
    [0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
    [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
    [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
    [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
    [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])

coefficients = danilevsky(A.T @ A)

print("Коэффициенты многочлена:", coefficients)
eigenvalues = np.roots(coefficients)

phi = [float(np.polyval(coefficients, i)) for i in eigenvalues]
print("Невязка phi:", phi)
phi_norm = np.linalg.norm(phi)
print("Величина невязки phi:", phi_norm)

eigenvectors_krylov = np.array([[0.18499117, 0.02543237, 0.1348814,
-0.06450363, 0.26396545],
```

```
                [-4.48113648e-06, 8.22737837e-04, -9.40479172e-05,
2.42815636e-04, 3.12639590e-05],
                [0.0050194, -0.00589962, -0.0104749, 0.01523646, 0.00612646],
                [-1.04922388e-02, -2.89615555e-03,  2.41899145e-02,
1.89997889e-02, -8.55669132e-05],
                [0.19088684, 0.00225188, 0.05391362, 0.03642805,
-0.15264063]])


for i in range(len(eigenvalues)):
   r = A.T @ A @ eigenvectors_krylov[i] - eigenvalues[i] *
eigenvectors_krylov[i]
   print(f"Собственное значение: {eigenvalues[i]}")
   print(f"Собственный вектор: {eigenvectors_krylov[i]}")
   print(f"Невязка r: {r}")
   print("Величина невязки:", np.linalg.norm(r))
```

Результаты

Коэффициенты многочлена: [1.   -5.10749837 9.69259576 -8.43406569 3.31218997 -0.45915011]

Невязка phi: [7.66053886991358e-15, 4.884981308350689e-15, 1.7763568394002505e-15, 6.106226635438361e-16, 1.1102230246251565e-16]

Величина невязки phi: 9.278336476446299e-15

Собственное значение: 1.8290022904680996

Собственный вектор: [0.18499117 0.02543237 0.1348814 -0.06450363 0.26396545]

Невязка r: [-3.15712256e-09 -2.87747218e-09 2.20410609e-09 5.50769692e-09

 2.70940070e-09]

Величина невязки: 7.796212560665586e-09

Собственное значение: 1.4054951691209334

Собственный вектор: [-4.48113648e-06 8.22737837e-04 -9.40479172e-05 2.42815636e-04

 3.12639590e-05]

Невязка r: [6.10437688e-14 -6.01608607e-14 9.66864934e-14
2.38203283e-13

 3.50630373e-14]

Величина невязки: 2.732474510891709e-13

Собственное значение: 0.954974840436802

Собственный вектор: [0.0050194 -0.00589962 -0.0104749   0.01523646
0.00612646]

Невязка r: [5.06742715e-10 -1.23798883e-09 2.28790626e-09
6.64418832e-11

 2.13924775e-09]

Величина невязки: 3.4065693935949904e-09

Собственное значение: 0.6128297159999979

Собственный вектор: [-1.04922388e-02 -2.89615555e-03 2.41899145e-02
1.89997889e-02

 -8.55669132e-05]

Невязка r: [4.97459712e-12 1.82368096e-12 9.23108985e-12
-8.72112799e-12

 9.46645809e-12]

Величина невязки: 1.670201636647474e-11

Собственное значение: 0.305196353974831

Собственный вектор: [0.19088684 0.00225188 0.05391362 0.03642805
-0.15264063]

Невязка r: [1.49097893e-09 3.44674296e-09 -1.07981533e-10
1.51141546e-09

 2.23797698e-09]

Величина невязки: 4.626838319045755e-09

Итерационным методом вращений с точностью ε = 10^{-5} найти спектр (λ_i, i = 1...n) и систему собственных векторов (x_i, i = 1…n) матрицы A^TA

Введение

Дана матрица размерности n x n следующего вида (n = 5):

```
A = np.array([[0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
        [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
        [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
        [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
        [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])
```

Итерационным методом вращений с точностью ε = 10^{-5} найти спектр (λ_i, i = 1...n) и систему собственных векторов (x_i, i = 1…n) матрицы A^TA, вычислить невязки $\phi_i = P_n(\lambda_i)$ и $r_i = A^TAx_i - \lambda x_i$ и оценить их значения. В конце провести сравнительный анализ полученных собственных значений и собственных векторов с точки зрения точности и экономичности используемых методов.

Алгоритм

Алгоритм основан на последовательном вращении матрицы до достижения заданной точности.

У нас есть две матрицы A^TA и U (единичная) и значение t, которое изначально является суммой квадратов недиагональных элементов матрицы A^TA и после каждой итерации будет вычисляться новая сумма квадратов, после чего сравнивается с 1e-5. Итерация заключается в следующем:

1) находим максимальный недиагональный элемент матрицы A^TA
2) вычисляем тригонометрические параметры для вращения матрицы по формулам:

$$\operatorname{tg} 2\varphi = \frac{2a_{kl}}{a_{kk} - a_{ll}}.$$

где a_{kl} – максимальный недиагональный элемент матрицы

$$\cos\varphi = \pm\sqrt{\frac{1+\cos 2\varphi}{2}}, \quad \sin\varphi = \pm\sqrt{\frac{1-\cos 2\varphi}{2}},$$

а также выражая косинус через тангенс, будем иметь:

$$\cos\varphi = \sqrt{\frac{1}{2}\left(1+\frac{1}{\sqrt{1+\mu^2}}\right)}, \quad \sin\varphi = \operatorname{sign}\mu\sqrt{\frac{1}{2}\left(1-\frac{1}{\sqrt{1+\mu^2}}\right)}, \quad \mu = \operatorname{tg} 2\varphi.$$

3)Далее производится вращение элементов матрицы A и матрицы U с использованием формул поворота.

```
prev_A = A[i, max_l]
prev_U = U[i, max_l]
A[i, max_l] = prev_A * cos - A[i, max_c] * sin
A[i, max_c] = prev_A * sin + A[i, max_c] * cos
U[i, max_l] = prev_U * cos - U[i, max_c] * sin
U[i, max_c] = prev_U * sin + U[i, max_c] * cos
```

```
prev_A = A[max_l, i]
A[max_l, i] = prev_A * cos - A[max_c, i] * sin
A[max_c, i] = prev_A * sin + A[max_c, i] * cos
```

где max_l – индекс строки, где находится максимальный элемент, max_c – индекс столбца.

Матрица $A^T A$ на выходе будет диагональной (диагональные элементы – собственные значения), матрица U (изначально единичная) будет содержать собственные векторы по столбцам.

Для оценки невязок использую евклидову норму $||x|| = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

Для подсчета невязки $\phi_i = P_n(\lambda_i)$ использую коэффициенты многочлена из метода Данилевского

Листинг программы

```python
import numpy as np

def rotation_method(A):

    n = A.shape[0]
    U = np.eye(n)
    counter = 0
    t = 0
    for i in range(n):
        for j in range(i + 1, n):
            t += A[i, j] ** 2
    t = 2 * t

    while t > 1e-5:
        max_val = A[0, 1]
        max_l, max_c = 0, 1
        for l in range(n):
            for c in range(l + 1, n):
                if abs(A[l, c]) > abs(max_val):
                    max_val = A[l, c]
                    max_l, max_c = l, c

        # Вычисляем tgd, cosd, cos и sin для вращения
        tgd = 2 * max_val / (A[max_l, max_l] - A[max_c, max_c])
        cosd = 1 / np.sqrt(1 + tgd ** 2)
        cos = np.sqrt((1 + cosd) / 2)
        sin = -np.sqrt((1 - cosd) / 2) * np.sign(tgd)

        # Выполняем вращение в строках
        for i in range(n):
            prev_A = A[i, max_l]
            prev_U = U[i, max_l]
            A[i, max_l] = prev_A * cos - A[i, max_c] * sin
            A[i, max_c] = prev_A * sin + A[i, max_c] * cos
            U[i, max_l] = prev_U * cos - U[i, max_c] * sin
            U[i, max_c] = prev_U * sin + U[i, max_c] * cos
```

```
        # Выполняем вращение в столбцах
        for i in range(n):
            prev_A = A[max_l, i]
            A[max_l, i] = prev_A * cos - A[max_c, i] * sin
            A[max_c, i] = prev_A * sin + A[max_c, i] * cos

        counter += 1
        t -= 2 * max_val ** 2

    eigenvalues = np.diagonal(A)

    print(f"Количество итераций: {counter}")

    return eigenvalues, U

A = np.array([
    [0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
    [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
    [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
    [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
    [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])

eigenvalues, eigenvectors = rotation_method(A.T @ A)

for i in range(len(eigenvalues)):
    r = A.T @ A @ eigenvectors[:, i] - eigenvalues[i] * eigenvectors[:, i]
    print(f"Собственное значение: {eigenvalues[i]}")
    print(f"Собственный вектор: {eigenvectors[:, i]}")
    print(f"Невязка r: {r}")
    print("Величина невязки:", np.linalg.norm(r))

coefficients_danilevsky = [1, -5.10749837, 9.69259576, -8.43406569,
3.31218997, -0.45915011]
phi = [float(np.polyval(coefficients_danilevsky, i)) for i in eigenvalues]
print("Невязка phi:", phi)
print("Величина невязки phi:", np.linalg.norm(phi))
```

Результаты

Количество итераций: 15

Собственное значение: 0.30519745265592535

Собственный вектор: [0.7540879   0.00873202 0.21455887 0.1451182
-0.60347361]

Невязка r: [-1.76959634e-04 -5.47828047e-05 4.39085400e-04
3.33215674e-04

1.43234771e-05]

Величина невязки: 0.0005816783507179023

Собственное значение: 1.4054899876503288

Собственный вектор: [-0.00436703 0.95178521 -0.11008828 0.28384048
0.03742978]

Невязка r: [-0.00016429 0.00047371 0.00051815 -0.00134647 -0.0003304]

Величина невязки: 0.0015626941237832385

Собственное значение: 0.6128288090598721

Собственный вектор: [-0.32297489 -0.08926081 0.74099197 0.58193507
-0.00148328]

Невязка r: [4.45026671e-04 -3.70470550e-04 1.57075565e-04
-1.07583428e-05

-3.59246659e-04]

Величина невязки: 0.0006993881633844315

Собственное значение: 0.9549798791916294

Собственный вектор: [0.23948647 -0.28462299 -0.49938488 0.72588158
0.2921415]

Невязка r: [-1.53589580e-07 -1.43603843e-03 1.81305752e-04
-4.15579687e-04

-5.64444723e-05]

Величина невязки: 0.0015069739980135052

Собственное значение: 1.8290022414422444

Собственный вектор: [0.51929358 0.07106985 0.37866965 -0.18117061 0.74099242]

Невязка r: [1.49971923e-05 1.36937189e-04 -1.13615163e-05 4.37764486e-05

 -7.13473439e-06]

Величина невязки: 0.000145165682581535

Невязка phi: [3.701699679559134e-07, 8.56445279695528e-07, 9.437526499178617e-08, 4.42034157499549e-07, -5.288101284639524e-08]

Величина невязки phi: 1.038085550832637e-06

Используя алгоритм метода Крылова, построить систему собственных векторов матрицы $A^T A$

Введение

Дана матрица размерности n x n следующего вида (n = 5):

```
A = np.array([[0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
        [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
        [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
        [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
        [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])
```

Используя алгоритм метода Крылова, построить систему собственных векторов матрицы $A^T A$, вычислить невязки $\phi_i = P_n(\lambda_i)$ и $r_i = A^T A x_i - \lambda x_i$ и оценить их значения. В конце провести сравнительный анализ полученных собственных значений и собственных векторов с точки зрения точности и экономичности используемых методов.

Алгоритм

Берем вектор произвольный ненулевой вектор $c^{(0)} = (1;0\ldots0)$, по которому составим последовательность

$$c^{(1)} = Ac^{(0)},\ \ c^{(2)} = Ac^{(1)} = A^2c^{(0)},\ldots$$

где A – симметрическая матрица A^TA

до тех пор, пока не встретим первый вектор, который будет являться линейной комбинацией предыдущих линейно независимых векторов, т.е. пока не станет справедливым равенство

$$q_1c^{(m-1)} + q_2c^{(m-2)} + \ldots + q_mc^{(0)} = c^{(m)},$$

предельно возможная m = n линейная комбинация

$$q_1c^{(n-1)} + q_2c^{(n-2)} + \ldots + q_nc^{(0)} = c^{(n)}$$

в координатном виде:

$$\begin{cases} q_1c_1^{(n-1)} + \ldots + q_nc_1^{(n-1)} = c_1^{(n)}, \\ \qquad \ldots\ldots \\ q_1c_n^{(n-1)} + \ldots + q_nc_n^{(n-1)} = c_n^{(n)}. \end{cases}$$

Решим ее по методу Гаусса (из первой лабораторной работы).
Решением системы являются коэффициента собственного многочлена.
Найдем собственные значения, которые являются корнями многочлена.

Собственный вектор x_i будем искать в виде линейной комбинации векторов $c^{(0)}\ldots c^{(m-1)}$:

$$x^{(i)} = \beta_{i1}c^{(m-1)} + \beta_{i2}c^{(m-2)} + \ldots + \beta_{im}c^{(0)}.$$

где $\beta_{i1} = 1$, β_{ij} , $j = 2\ldots m$, находятся по формуле

$$\beta_{i2} = (\lambda_i - q_1)\beta_{i1},$$
$$\beta_{i3} = (\lambda_i^2 - q_1\lambda_i - q_2)\beta_{i1},$$
$$\ldots\ldots$$
$$\beta_{im} = (\lambda_i^{m-1} - q_1\lambda_i^{m-2} - \ldots - q_{m-1})\beta_{i1},$$

где λ_i – собственное значение, соответствующее собственному вектору,
q_i – i-ый коэффициент многочлена $P(\lambda)$

Для оценки невязок использую евклидову норму $||x|| = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

Листинг программы

```
import numpy as np

def gauss_elimination_with_determinant_inverse_condition_number(A, b):
  n = len(A)
  A_temp = A.astype(float)
  b_temp = b.astype(float).reshape(-1, 1)
  A_ext = np.hstack([A_temp, b_temp])

  for i in range(n):
      if A_ext[i, i] == 0:
          for k in range(i + 1, n):
              if A_ext[k, i] != 0:
                  A_ext[[i, k]] = A_ext[[k, i]]
                  break
          else:
              return None, 0, None, None

      A_ext[i] = A_ext[i] / A_ext[i, i]

      for j in range(i + 1, n):
          A_ext[j] = A_ext[j] - A_ext[j, i] * A_ext[i]

  for i in range(n - 1, -1, -1):
      for j in range(i - 1, -1, -1):
          A_ext[j] = A_ext[j] - A_ext[j, i] * A_ext[i]

  x = A_ext[:, -1]

  return x

def krylov(A):
```

```
    c = []
    n = A.shape[0]
    c.append(np.zeros(n))
    c[0][0] = 1

    for i in range(n):
        c.append(A @ c[i])

    c_n = c[n]
    C = np.array(c[:n])
    p =
list(reversed(gauss_elimination_with_determinant_inverse_condition_number(C
.T, c_n)))

    coefficients = -np.array(p)
    coefficients = np.insert(coefficients, 0, 1)

    eigenvalues = np.roots(coefficients)

    eigenvectors = []
    for k in range(n):
        b = [1 for _ in range(n)]
        for i in range(1, n):
            b[i] = b[i - 1] * eigenvalues[k] - p[i - 1]

        x = np.sum([b[i] * C[n - i - 1] for i in range(n)], axis=0)
        eigenvectors.append(x)

    return coefficients, eigenvalues, eigenvectors

A = np.array([[0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
       [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
       [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
       [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
       [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])
```

```
coefficients, eigenvalues, eigenvectors = krylov(A.T @ A)
print("Коэффициенты собственного многочлена:", coefficients)
phi = [float(np.polyval(coefficients, i)) for i in eigenvalues]
print("Невязка phi:", phi)
phi_norm = np.linalg.norm(phi)
print("Величина невязки phi:", phi_norm)

for i in range(len(eigenvalues)):
   r = A.T @ A @ eigenvectors[i] - eigenvalues[i] * eigenvectors[i]
   print("Собственное значение:", eigenvalues[i])
   print("Собственный вектор:", eigenvectors[i])
   print("Невязка r:", r)
   print("Величина невязки:", np.linalg.norm(r))
```

Результаты

Коэффициенты собственного многочлена: [1.   -5.10749837 9.69259576 -8.43406569 3.31218997 -0.45915011]

Невязка phi: [3.219646771412954e-15, 1.3322676295501878e-15, -3.3306690738754696e-16, 1.6653345369377348e-16, 1.6653345369377348e-16]

Величина невязки phi: 3.508199355312094e-15

Собственное значение: 1.829002290468074

Собственный вектор: [0.18499117 0.02543237 0.1348814 -0.06450363 0.26396545]

Невязка r: [-3.83026943e-15 -2.49800181e-16 -8.60422844e-16 -7.77156117e-16

-7.54951657e-15]

Величина невязки: 8.54826670021255e-15

Собственное значение: 1.4054951691218034

Собственный вектор: [-4.48113648e-06 8.22737837e-04 -9.40479172e-05 2.42815636e-04

3.12639590e-05]

Невязка r: [-3.15143690e-16 -9.08561421e-17 -1.14518854e-16
-1.32603347e-15

-6.13605575e-15]

Величина невязки: 6.287307042511568e-15

Собственное значение: 0.9549748404368605

Собственный вектор: [0.0050194 -0.00589962 -0.0104749   0.01523646
0.00612646]

Невязка r: [4.07660017e-17 -2.35055031e-16 -7.77156117e-16
-9.12464548e-16

-8.35009145e-15]

Величина невязки: 8.439046342294444e-15

Собственное значение: 0.6128297159993766

Собственный вектор: [-1.04922388e-02 -2.89615555e-03 2.41899145e-02
1.89997889e-02

-8.55669132e-05]

Невязка r: [-5.69856662e-16 -1.93855348e-16 -4.31946146e-16
-1.15532584e-15

-7.78151550e-15]

Величина невязки: 7.901623586400839e-15

Собственное значение: 0.305196353974814

Собственный вектор: [0.19088684 0.00225188 0.05391362 0.03642805
-0.15264063]

Невязка r: [-6.66133815e-16 -1.80302821e-16 -2.08166817e-17
-1.30624678e-15

-7.75074449e-15]

Величина невязки: 7.890310355356559e-15

Степенным методом с точностью ε = 10 $^{-5}$ найти максимальное собственное значение матрицы A^TA и соответствующий собственный вектор

Введение

Дана матрица размерности n x n следующего вида (n = 5):

```
A = np.array([[0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
        [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
        [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
        [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
        [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])
```

Степенным методом с точностью ε = 10 $^{-5}$ найти максимальное собственное значение матрицы A^TA и соответствующий собственный вектор, вычислить невязки $\phi_i = P_n(\lambda_i)$ и $r_i = A^TAx_i - \lambda x_i$ и оценить их значения. В конце провести сравнительный анализ полученных собственных значений и собственных векторов с точки зрения точности и экономичности используемых методов.

Алгоритм

Возьмем произвольный вектор $y^{(k)} = (1, 0, \dots, 0)^T$ и построим последовательность векторов по следующему правилу:

$$y^{(k)} = Ay^{(k-1)}, \quad k = 1, 2, \dots .$$

При достаточно больших значениях k будет выполняться приближенное равенство

$$\lambda_1 \approx \frac{y_i^{(k+1)}}{y_i^{(k)}}.$$

где λ_1 – максимальное по модулю собственное значение

В качестве критерия останова итерационного процесса используется условие:

$$|\lambda_1^{k+1} - \lambda_1^k| \leqslant \varepsilon.$$

После окончания итерационного процесса вектор нормируется путем деления его на свою норму, чтобы получить нормализованный собственный вектор. Когда итерационный процесс завершается, финальное значение вектора y_k является приближенным собственным вектором, соответствующим максимальному собственному значению, которое сохраняется в переменной lambda_

Для вычисления невязки $\phi_1 = P_n(\lambda_1)$ использую коэффициенты собственного многочлена из метода Данилевского.

Для оценки невязок использую евклидову норму $\|x\| = \sqrt{\sum_{i=1}^{n} |x_i|^2}$

Листинг программы

```
import numpy as np


A = np.array([
   [0.7277, -0.0958, 0.0192, 0.0383, 0.1341],
   [0.0996, 1.1394, 0.0000, -0.0766, 0.0766],
   [0.0575, 0.0000, 0.9154, -0.2681, 0.1532],
   [-0.1149, 0.2413, 0.0000, 0.9001, -0.0383],
   [0.4788, 0.0000, 0.1724, 0.0192, 1.0724]
])


n = A.shape[0]
y_k = np.ones(n)
y_k_plus_1 = np.zeros(n)
almost_lambda = np.zeros(n)


lambda_ = 0
```

```
norm = 0
flag = True
counter = 0

while flag:

   y_k_plus_1 = A.T @ A @ y_k

   flag = False
   norm = np.linalg.norm(y_k_plus_1)

   for i in range(n):
       lambda_ = y_k_plus_1[i] / y_k[i]

       if not flag and abs(lambda_ - almost_lambda[i]) > 1e-5:
           flag = True

       almost_lambda[i] = lambda_
       y_k[i] = y_k_plus_1[i] / norm

   counter += 1

eigenvector = y_k
eigenvalue = lambda_

r = A.T @ A @ eigenvector - eigenvalue * eigenvector
print("Количество итераций:", counter)
print("Максимальное собственное значение:", eigenvalue)
print("Собственный вектор:", eigenvector)
print("Невязка r:", r)
print("Величина невязки r:", np.linalg.norm(r))

coefficients_danilevsky = [1, -5.10749837, 9.69259576, -8.43406569,
3.31218997, -0.45915011]
phi = float(np.polyval(coefficients_danilevsky, eigenvalue))
print("Невязка phi:", phi)
print("Величина невязки phi:", np.linalg.norm(phi))
```

Результаты

Количество итераций: 46

Максимальное собственное значение: 1.829002179043569

Собственный вектор: [0.51930219 0.07139699 0.37863498 -0.18107167
0.74099684]

Невязка r: [6.69569978e-08 -1.66171034e-06 2.33050242e-07
-5.12946349e-07

1.91180438e-08]

Величина невязки r: 1.756005798370545e-06

Невязка phi: -9.568513781310628e-08

Величина невязки phi: 9.568513781310628e-08

Низкие невязки в точных методах могут свидетельствовать о том, что численные методы, используемые для вычисления коэффициентов многочлена, собственных значений и векторов, достаточно точны. Можно сказать, метод Данилевского немного экономичнее, так как не требует решения системы линейных уравнений, а метод Крылова точнее, так как погрешность получается немного меньше.

В отличие от точных методов, в которых при больших матрицах погрешность сильно возрастает, найти решение дают возможность итерационные методы, которые считают решение с заданной точностью, что часто может пригодиться на практике. Невязки у них больше, чем у точных, что объясняется тем, что итерационный процесс останавливается при достижении определенной точности.

Решение, полученное степенным методом, имеет меньшую невязку, чем решение, полученное итерационным методом вращений. Количество итераций больше у степенного метода. Значит, с данной матрицей итерационный метод вращений имеет большую скорость сходимости, чем степенной метод. Однако если будет нужна лишь часть спектра, то степенной метод будет экономичнее.