Санкт-Петербургский государственный университет
Математико-механический факультет

Отчёт по практическому заданию №6

Решение полной и частичной проблемы собственных значений.

Выполнил:
студент 451 гр.
Челабов А.Г.

Санкт – Петербург
2024

1.Поставновка задачи

Дана краевая задача $-(a(x)u')' + c(x)u = f(x) \qquad u(0) = u(1) = 0$

В процессе её решения МКЭ возникает симметричная вещественная матрица $\boldsymbol{K}$ размера $N \times N$.

$$\boldsymbol{Ku = f}$$

Необходимо найти собственные числа и соответствующие им собственные векторы матрицы $\boldsymbol{K}$, т.е. такие числа λ_i и вектора $\boldsymbol{e_i}$, удовлетворяющие:

$$\boldsymbol{Ke_i} = \lambda_i \boldsymbol{e_i}$$

2.Степенной метод

Пусть у матрицы $\boldsymbol{K}$ $\exists N$ линейно независимых собственных векторов и её собственные числа упорядочены: $|\lambda_1|>|\lambda_2|\geq|\lambda_3|\geq \cdots \geq|\lambda_N|$ Тогда для нахождения λ_1 строится следующий итерационный процесс:

1. Произвольно выбираем ненулевой вектор $\boldsymbol{y^0}$, который можно разложить по базису из собственных векторов: $\boldsymbol{y^0} = c_1\boldsymbol{e_1} + c_2\boldsymbol{e_2} + \cdots$

2. Умножаем $\boldsymbol{y^0}$ на начальную матрицу $\boldsymbol{K}$:

$$\boldsymbol{y^1} = \boldsymbol{Ky^0} = \textstyle\sum_{i=1}^{N} c_i \boldsymbol{Ke_i} = \sum_{i=1}^{N} c_i\lambda_i \boldsymbol{e_i}$$

…

$$\boldsymbol{y^{k+1}} = \boldsymbol{Ky^k} = \sum_{i=1}^{N} c_i\lambda_i^k \boldsymbol{Ke_i} = \sum_{i=1}^{N} c_i\lambda_i^{k+1} \boldsymbol{e_i}$$

3. Посчитаем отношение j-ых компонент векторов $\boldsymbol{y^{k+1}}$ и $\boldsymbol{y^k}$:

$$\frac{y_j^{k+1}}{y_j^k} = \frac{\sum_{i=1}^{N} c_i\lambda_i^{k+1} e_{i_j}}{\sum_{i=1}^{N} c_i\lambda_i^{k} e_{i_j}} = \frac{c_1\lambda_1^{k+1} e_{1_j}}{c_1\lambda_i^{k} e_{1_j}} \frac{1 + \sum\left(\frac{c_i e_{i_j}}{c_1 e_{1_j}}\right)\left(\frac{\lambda_i}{\lambda_1}\right)^{k+1}}{1 + \sum\left(\frac{c_i e_{i_j}}{c_1 e_{1_j}}\right)\left(\frac{\lambda_i}{\lambda_1}\right)^{k}}$$

$$= \lambda_1 \frac{1 + \sum\left(\frac{c_i e_{i_j}}{c_1 e_{1_j}}\right)\left(\frac{\lambda_i}{\lambda_1}\right)^{k+1}}{1 + \sum\left(\frac{c_i e_{i_j}}{c_1 e_{1_j}}\right)\left(\frac{\lambda_i}{\lambda_1}\right)^{k}} = \lambda_1^{(k)}$$

$$\lim_{k\to\infty} \lambda_1^{(k)} = \lambda_1$$

т.к. $\lim\limits_{(k\to\infty)_{i\neq 1}} \left(\frac{\lambda_i}{\lambda_1}\right)^k = 0$

При этом $\lim\limits_{k\to\infty} \boldsymbol{y}^{\boldsymbol{k}} = \boldsymbol{e_1}$

В конце каждой итерации нормеруем вектор $\mathbf{y}^{\boldsymbol{k+1}}$.

Для нахождения минимального собственного числа $\lambda_{\min}^{\boldsymbol{K}}$ и соответствующего данному собственному числу собственного вектора матрицы $\boldsymbol{K}$ применяют этот же метод для матрицы $\boldsymbol{K} - \lambda_1\boldsymbol{E}$. После этого к полученному числу $\lambda_{\max}^{\boldsymbol{K}-\lambda_1\boldsymbol{E}}$ прибавляют λ_1 и получают $\lambda_{\min}^{\boldsymbol{K}}$ **.** Собственные векторы максимального собственного числа новой матрицы и минимального собственного числа первоначальной матрицы совпадают.

3.Метод Якоби

Любую вещественную симметричную матрицу $\boldsymbol{K}$ можно привести к диагональном виду таким образом:

$$\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{V} = \boldsymbol{\Lambda}$$

где

$\boldsymbol{V}$ – ортогональная матрица, $\boldsymbol{\Lambda} = diag\{\lambda_1, \lambda_2, \dots, \lambda_N\}$

Построим последовательность $\boldsymbol{K}^{(\mathbf{0})} = \boldsymbol{K}, \boldsymbol{K}^{(1)}, \boldsymbol{K}^{(2)}, \dots, \boldsymbol{K}^{(k)}, \dots$ которые будут сходится к $\boldsymbol{\Lambda}$ с ростом k.

$$\boldsymbol{K}^{(k)} = \boldsymbol{V}_{i_k j_k}^{\boldsymbol{T}}\boldsymbol{K}^{(k-1)}\boldsymbol{V}_{i_k j_k}$$

$$\boldsymbol{V}_{i_k j_k} = v_{ij}, i_k < j_k$$

$$v_{ii} = 1, i \neq i_k, i \neq j_k$$

$$v_{i_k i_k} = v_{j_k j_k} = \cos(\phi)$$

$$v_{i_k j_k} = -v_{j_k i_k} = -\sin(\phi)$$

$$v_{ij} = 0, i \neq i_k, i \neq j_k, j \neq i_k, j \neq j_k$$

$$\boldsymbol{V}_{i_k j_k} = \begin{pmatrix} 1 & 0 & . & . & . & . & 0 \\ 0 & 1 & . & . & . & . & 0 \\ . & . & \cos\varphi & . & . & -\sin\varphi & . \\ . & . & . & 1 & & . & . \\ . & . & . & & 1 & . & . \\ 0 & . & \sin\varphi & . & . & \cos\varphi & 0 \\ 0 & . & . & . & . & 0 & 1 \end{pmatrix}$$

Матрица $\boldsymbol{K}^{(k+1)}$ строится так, чтобы

$$\sum_{i,j=1, i\neq j} (K_{ij}^{(k+1)})^2 \le \sum_{i,j=1, i\neq j} (K_{ij}^{(k)})^2$$

Поэтому i_k, j_k выбираются как индексы максимального наддиагонального элемента:

$$\left|K_{i_k, j_k}^{(k)}\right| = \max_{i,j, i\neq j} |K_{ij}^{(k)}|$$

Угол ϕ выбирается так, чтобы:

$$K_{i_k j_k}^{(k+1)} = \frac{K_{j_k j_k}^{(k)} - K_{i_k i_k}^{(k)}}{2} \sin(2\phi) + K_{i_k j_k}^{(k)} \cos(2\phi) = 0$$

$$tg(2\phi) = \frac{2K_{i_k j_k}^{(k)}}{K_{j_k j_k}^{(k)} - K_{i_k i_k}^{(k)}}, |\phi| < \frac{\pi}{4}$$

если $K_{j_k j_k}^{(k)} - K_{i_k i_k}^{(k)} = 0 => |\phi| = \frac{\pi}{4}$

В итоге собственные числа $\lambda_i^{(k)}$ будут находиться на главной диагонали $\boldsymbol{K}^{(k)}$, а собственные вектора будут столбцами матрицы $\boldsymbol{X} = \boldsymbol{V}_{i_0 j_0} \boldsymbol{V}_{i_1 j_1} \boldsymbol{V}_{i_2 j_2} \dots$

4.Оценка погрешности

Если $\tilde{\lambda}$ – приближенное собственное число $\boldsymbol{K}, \tilde{\boldsymbol{e}}$ – приближенный собственный вектор, то справедлива оценка погрешности:

$$|\lambda - \tilde{\lambda}| \le \frac{\left|\left|\boldsymbol{K}\tilde{\boldsymbol{e}} - \tilde{\lambda}\tilde{\boldsymbol{e}}\right|\right|_2}{||\tilde{\boldsymbol{e}}||_2}$$

Результаты

Степенной метод и метод обратных итераций для N = 10^5

	μ_{max}	$\lVert Kx_j - \mu_{max}x_j \rVert$	μ_{min}	$\lVert Kx_j - \mu_{min}x_j \rVert$
$\varepsilon = 10^3$	3.9719e+06	0.0099	1.0865e-05	0.0039
$\varepsilon = 10^4$	3.9972e+06	9.9865e-04	1.0865e-05	6.9563e-07
$\varepsilon = 10^5$	3.9997e+06	9.9986e-05	1.0865e-05	6.9249e-07

метод Якоби и сравнение с точным значением для N = 10

$\varepsilon = 10^5$	λ_{ex} * e6	λ_j * e6	$\lVert Kx_j - \lambda_j x_j \rVert$
1	0.0810	0.0810	0.000284798
2	0.3175	0.3175	0.000184825
3	0.6903	0.6903	0.000989491
4	1.1692	1.1692	0.000124841
5	1.7154	1.7154	0.000658265
6	2.2846	2.2846	0.000367452
7	2.8308	2.8308	0.000375912
8	3.3097	3.3097	0.000825982
9	3.6825	3.6825	0.000924150
10	3.9190	3.9190	0.000112509

Реализация

Степенной метод

```
def itermax(A, B, C, EPS):
    n = len(B)
    x = np.random.rand(n, 1)
    lambda_ = 1
    Ax = np.zeros((n, 1))
    err = 1
    while err > EPS:
        Ax[0] = B[0]*x[0] + A[1]*x[1]
        Ax[n-1] = B[n-1]*x[n-1] + C[n-2]*x[n-2]
        for i in range(1, n-1):
            Ax[i] = A[i]*x[i-1] + B[i]*x[i] + C[i]*x[i+1]
        lambda_ = np.dot(Ax.T, x) / np.dot(x.T, x)
        err = np.linalg.norm(Ax - lambda_*x) / np.linalg.norm(Ax)
        x = Ax / np.linalg.norm(Ax)
    return lambda_, x
```

Метод обратных итераций

```
def progonka(A, B, C, x):
    n = len(B)
    alpha = np.zeros((n-1,))
    beta = np.zeros((n,))
    alpha[0] = -C[0] / B[0]
    beta[0] = x[0] / B[0]
    for i in range(1, n-1):
        alpha[i] = -C[i] / (A[i]*alpha[i-1] + B[i])
    for i in range(1, n):
        beta[i] = (x[i] - A[i]*beta[i-1]) / (A[i]*alpha[i-1] + B[i])
    y = np.zeros((n,))
    y[n-1] = beta[n-1]
    for i in range(n-2, -1, -1):
        y[i] = alpha[i]*y[i+1] + beta[i]
    return y

def iterinv(A, B, C, EPS):
    n = len(B)
    mu = 0
    x = np.random.rand(n, 1)
    x = x / np.linalg.norm(x)
    Ax = progonka(A, B-mu, C, x)
    lambda_ = np.dot(Ax.T, x)
    mu = mu + 1/lambda_
    err = np.linalg.norm(Ax - lambda_*x) / np.linalg.norm(Ax)
    while err > EPS:
        x = Ax / np.linalg.norm(Ax)
        Ax = progonka(A, B-mu, C, x)
        lambda_ = np.dot(Ax.T, x)
        mu = mu + 1/lambda_
        err = np.linalg.norm(Ax - lambda_*x) / np.linalg.norm(Ax)
    return mu, x, err
```

Метод Якоби

```
def jacobi(A, EPS):
    n = A.shape[0]
    it = 0
    X = np.eye(n)
    M = 1
    while M > EPS:
        M = abs(A[0][1])
        ik = 0
        jk = 1
        for i in range(n):
            for j in range(i+1, n):
                if abs(A[i][j]) > M:
                    M = abs(A[i][j])
                    ik = i
                    jk = j
        it += 1
        d = np.sqrt((A[ik][ik]-A[jk][jk])2 + 4*A[ik][jk]2)
        c = np.sqrt(1/2*(1+abs(A[ik][ik]-A[jk][jk])/d))
        s = np.sign(A[ik][jk]*(A[ik][ik]-A[jk][jk])) * np.sqrt(1/2*(1-abs(A[ik][ik]-A[jk][jk])/d))
        a_ii = A[ik][ik]
        a_ij = A[ik][jk]
        a_jj = A[jk][jk]
        A[ik][ik] = c2*a_ii + 2*c*s*a_ij + s2*a_jj
        A[jk][jk] = s2*a_ii - 2*c*s*a_ij + c2*a_jj
        A[ik][jk] = (c2-s2)*a_ij + c*s*(a_jj-a_ii)
        A[jk][ik] = A[ik][jk]
        for i in range(n):
            a_iik = A[i][ik]
            if i != ik and i != jk:
                A[i][ik] = c*A[i][ik] + s*A[i][jk]
                A[ik][i] = A[i][ik]
                A[i][jk] = -s*a_iik + c*A[i][jk]
                A[jk][i] = A[i][jk]
        A[ik][jk] = 0
        A[jk][ik] = 0
        for i in range(n):
            a_ii = X[i][ik]
            X[i][ik] = c*a_ii + s*X[i][jk]
            X[i][jk] = -s*a_ii + c*X[i][jk]
    return A, X
```

```
N = 10
h = 1/N
A = np.zeros(N)
B = np.zeros(N)
C = np.zeros(N)

for i in range(N):
    A[i] = -1/h + h*2*h*(i+1/2)/6
    B[i] = 2/h + h/3*(2*(h*(i-1/2))+2*(h*(i+1/2)))
    C[i] = -1/h + h*2*h*(i-1/2)/6

A[0] = 0
C[N-1] = 0

ABC = np.diag(B) + np.diag(A[1:], k=1) + np.diag(C[:-1], k=-1)

alleighvalues = np.linalg.eigvals(ABC)
itermax(A, B, C, 10e-5)
iterinv(A, B, C, 10e-5)
jacobi(ABC, 10e-5)
```