Отчет по численным методам

Сотникова Виктория

2023

Содержание

1 Постановка задачи

Решить с помощью метода прогонки следующие СЛАУ $A_i\mathbf{b} = \mathbf{f}_i$:

1. СЛАУ, получающуюся при построении кубического сплайна для функции $f(x) = x|\cos(5x)|$ на отрезке $x \in [0, 2]$ по значениям в $N = 100$ узлах $x_i = 2i/N$

2.

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & 0 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 6 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

$(\mathbf{f}_2)_l = \cos(x_l) * h^4, \quad x_l = \dfrac{l\pi}{n}, \quad h = \dfrac{\pi}{n}, \quad l = \overline{2, n-2}$

$(\mathbf{f}_2)_0 = (\mathbf{f}_2)_n = 1, \quad (\mathbf{f}_2)_1 = (\mathbf{f}_2)_{n-1} = 0$

$n = 100$

Провести сравнение решений двух задач на основе следующих критериев:
1. Количество операций относительно размерности матриц
2. Норма невязки $\|\mathbf{r}_i\|_2 = \|A_i\mathbf{x} - \mathbf{b}_i\|_2$

2 Обзор методов

2.1 Метод Гаусса

Не ограничивая общности, будем считать, что коэффициент a_{11}, который называют ведущим элементом первого шага, отличен от нуля (в случае $a_{11} = 0$ поменяем местами уравнения с номерами 1 и i, при котором $a_{11} \neq 0$ поскольку система предполагается невырожденной, то такой номер i заведомо найдется) Прямой ход - это приведение матрицы системы $Ax = f$ к треугольному виду. Нормировка первой строки. Поделим первую строку на a_{11}, т.е. выполним следующие преобразования:

$$a_{ij}^{(1)} = \frac{a_{1j}}{a_{11}}, \quad f_1^{(1)} = \frac{f_1}{a_{11}}, \quad j = \overline{1,n}.$$

Обнуление первого столбца. Вычтем из строки i первую строку, домноженную на a_{i1}, т.е.

$$a_{ij}^{(1)} = a_{ij} - a_{i1} \cdot a_{1j}^{(1)}, \quad f_i^{(1)} - a_{i1} \cdot f_1^{(1)}, \quad i = \overline{2,n}, \quad j = \overline{1,n}.$$

Повторяем эти два шага для полученной матрицы $A^{(1)}$, пока не получим треугольную матрицу, из которой можно по цепочке выразить все неизвестные.

2.2 Метод трехточечной прогонки

Имеется система вида:

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & & \ddots & & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{n-1} \\ r_n \end{pmatrix}$$

Главная диагональ состоит из $b_j, j = 1, \ldots, n$. Нижняя побочная диагональ состоит из a_j, где j меняется от 2 до n, а c_j лежат выше главной диагонали, причем $1 \leq j \leq n-1$. Также мы можем переписать СЛАУ как набор уравнений:

$$
\begin{aligned}
&b_1x_1 + c_1x_2 = r_1\\
&a_2x_1 + b_2x_2 + c_2x_3 = r_2\\
&a_3x_2 + b_3x_3 + c_3x_4 = r_3\\
&\dots\\
&a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = r_{n-1}\\
&a_nx_{n-1} + b_nx_n = r_n
\end{aligned}
$$

Можно исключить x_1 из первых двух уравнений. Умножим первое уравнение на a_2/b_1, вычтем из второго. Получим:

$$
\begin{aligned}
&b_1x_1 + c_1x_2 = r_1\\
&\left(b_2 - \frac{a_2}{b_1}c_1\right)x_2 + c_2x_3 = r_2 - \frac{a_2}{b_1}r_1\\
&a_3x_2 + b_3x_3 + c_3x_4 = r_3 \qquad\qquad \dots\\
&a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = r_{n-1}\\
&a_nx_{n-1} + b_nx_n = r_n
\end{aligned}
$$

Введем обозначения:

$$\beta_1 = b_1, \quad \rho_1 = r_1$$

и

$$\beta_2 = b_2 - \frac{a_2}{\beta_1}c_1, \quad \rho_2 = r_2 - \frac{a_2}{\beta_1}\rho_1$$

Таким образом получим

$$
\begin{aligned}
&\beta_1x_1 + c_1x_2 = \rho_1\\
&\beta_2x_2 + c_2x_3 = \rho_2\\
&a_3x_2 + b_3x_3 + c_3x_4 = r_3\\
&\dots\\
&a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = r_{n-1}\\
&a_nx_{n-1} + b_nx_n = r_n
\end{aligned}
$$

Теперь можно исключить x_2, умножая второе уравнение на a_3/β_2 и вычитая его из третьего

$$
\begin{aligned}
&\beta_1x_1 + c_1x_2 = \rho_1\\
&\beta_2x_2 + c_2x_3 = \rho_2\\
&\beta_3x_3 + c_3x_4 = \rho_3\\
&\dots\\
&\beta_{n-1}x_{n-1} + c_{n-1}x_n = \rho_{n-1}\\
&\beta_nx_n = \rho_n
\end{aligned}
$$

где были определены

$$\beta_3 = b_3 - \frac{a_3}{\beta_2}c_2, \quad \rho_3 = r_3 - \frac{a_3}{\beta_2}\rho_2.$$

После $n-1$ таких шагов получим:

$$\begin{aligned}
&\beta_1 x_1 + c_1 x_2 = \rho_1 \\
&\beta_2 x_2 + c_2 x_3 = \rho_2 \\
&\beta_3 x_3 + c_3 x_4 = \rho_3 \\
&\dots \\
&\beta_{n-1} x_{n-1} + c_{n-1} x_n = \rho_{n-1} \\
&\beta_n x_n = \rho_n
\end{aligned}$$

при этом

$$\beta_j = b_j - \frac{a_j}{\beta_{j-1}} c_{j-1} \quad \text{и} \quad \rho_j = r_j - \frac{a_j}{\beta_{j-1}} \rho_{j-1}, \quad j = 2, \dots, n.$$

Из последних уравнений имеем

$$x_n = \rho_n / \beta_n,$$

которое затем можно подставить в предыдущее уравнение, чтобы получить

$$x_{n-1} = \left(\rho_{n-1} - c_{n-1} x_n\right) / \beta_{n-1},$$

и так далее. Поскольку все предыдущие уравнения имеют одинаковый вид, приходим к общему результату

$$x_{n-j} = \left(\rho_{n-j} - c_{n-j} x_{n-j+1}\right) / \beta_{n-j}, \quad j = 1, \dots, n-1,$$

2.3 Интерполяция сплайнами

Определим функцию $p(x)$ в области $x_j \leqslant x \leqslant x_{j+1}$. Она может быть записана в виде

$$p(x) = a_j (x - x_j)^3 + b_j (x - x_j)^2 + c_j (x - x_j) + d_j$$

Апроксимация должна быть верна для точек $x = x_j$:

$$p\left(x_j\right) = f\left(x_j\right) = d_j$$

Апроксимация должна быть верна для точек $x = x_{j+1}$:

$$p_{j+1} = a_j h_j^3 + b_j h_j^2 + c_j h_j + p_j,$$

где $p_j = p\left(x_j\right)$ и $h_j = x_{j+1} - x_j$

Производные кубического сплайна:

$$p'(x) = 3a_j \left(x - x_j\right)^2 + 2b_j \left(x - x_j\right) + c_j$$

$$p''(x) = 6a_j \left(x - x_j\right) + 2b_j.$$

Для второй производной в $x = x_j$ имеем

$$p''\left(x_j\right) = p''_j = 2b_j$$

поэтому

$$b_j = \frac{p''_j}{2}$$

для точек $x = x_{j+1}$ имеем

$$p''_{j+1} = 6a_j h_j + 2b_j$$

поэтому

$$a_j = \frac{1}{6}\frac{p''_{j+1} - p''_j}{h_j}.$$

Из уравнений выше можем найти

$$c_j = \frac{p_{j+1} - p_j}{h_j} - \frac{h_j p''_{j+1} + 2h_j p''_j}{6}.$$

Зная коэффициенты многочлена, по крайней мере, в терминах p''_j, можно написать

$$p(x) = p_j + \left[\frac{p_{j+1} - p_j}{h_j} - \frac{h_j p''_{j+1}}{6} - \frac{h_j p''_j}{3}\right](x - x_j) + \frac{p''_j}{2}(x - x_j)^2 + \frac{p''_{j+1} - p''_j}{6h_j}(x - x_j)^3, \quad x_j \le x \le x_{j+1}$$

и

$$p'(x) = \frac{p_{j+1} - p_j}{h_j} - \frac{h_j p''_{j+1}}{6} - \frac{h_j p''_j}{3} + p''_j(x - x_j) + \frac{p''_{j+1} - p''_j}{2h_j}(x - x_j)^2, \quad x_j \le x \le x_{j+1}.$$

Чтобы определить p''_j, рассмотрим производную на предыдущем интервале. Заменив j на $j - 1$, получим

$$p'(x) = \frac{p_j - p_{j-1}}{h_{j-1}} - \frac{h_{j-1} p''_j}{6} - \frac{h_{j-1} p''_{j-1}}{3} + p''_{j-1}(x - x_{j-1}) + \frac{p''_j - p''_{j-1}}{2h_{j-1}}(x - x_{j-1})^2, \quad x_{j-1} \le x \le x_j.$$

Теперь требуем, чтобы решения давали в точности такие же значения производной при $x = x_j$:

$$\frac{p_j - p_{j-1}}{h_{j-1}} - \frac{h_{j-1} p''_j}{6} - \frac{h_{j-1} p''_{j-1}}{3} + p''_{j-1} h_j + \frac{p''_j - p''_{j-1}}{2h_{j-1}} h_{j-1}^2 = \frac{p_{j+1} - p_j}{h_j} - \frac{h_j p''_{j+1}}{6} - \frac{h_j p''_j}{3}$$

Окончательно преобразовав:

$$h_{j-1} p''_{j-1} + (2h_j + 2h_{j-1}) p''_j + h_j p''_{j+1} = 6\left(\frac{p_{j+1} - p_j}{h_j} - \frac{p_j - p_{j-1}}{h_{j-1}}\right), \quad j = 2, \ldots, n - 1$$

Это дает нам $(n-2)$ уравнений для n неизвестных p''_j — нам нужны еще два уравнения, чтобы найти единственное решение. Эти дополнительные уравнения могут быть получены в результате задания значений производной в граничных точках, т. е. в точках $x = x_1$ и x_n:

$$2h_1 p''_1 + h_1 p''_2 = 6\frac{p_2 - p_1}{h_1} - 6p'_1$$

Также выражаем

$$h_{n-1} p''_{n-1} + 2h_{n-1} p''_n = -6\frac{p_n - p_{n-1}}{h_{n-1}} + p'_n$$

Все эти уравнения могут быть записаны в виде СЛАУ:

$$\begin{pmatrix} 2h_1 & h_1 & & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & & \\ & & & \ddots & & & \\ & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & & & h_{n-1} & 2h_{n-1} \end{pmatrix} \begin{pmatrix} p''_1 \\ p''_2 \\ p''_3 \\ \vdots \\ p''_{n-1} \\ p''_n \end{pmatrix}$$

$$= \begin{pmatrix} 6\frac{p_2-p_1}{h_1} - 6p'_1 \\ 6\frac{p_3-p_2}{h_2} - 6\frac{p_2-p_1}{h_1} \\ 6\frac{p_4-p_3}{h_3} - 6\frac{p_3-p_2}{h_2} \\ \vdots \\ 6\frac{p_n-p_{n-1}}{h_{n-1}} - 6\frac{p_{n-1}-p_{n-2}}{h_{n-1}} \\ -6\frac{p_n-p_{n-1}}{h_{n-1}} + 6p'_n \end{pmatrix}$$

Причем матрица слева является трехдиагональной

3 Анализ методов СЛАУ

3.1 Применимость методов решения СЛАУ

Матрица в первой СЛАУ имеет трёхдиагональную структуру, метод трехточеной прогонки к ней применим. Ко второй же СЛАУ метод трехточечной прогонки неприменим, потому что она не имеет трёхдиагональную структуру. При этом к обеим СЛАУ применим метод Гаусса, так как определители матриц не равны нулю

3.2 Количество операций

Метод Гаусса требует $O(n^3)$ операций относительно размерности матриц. В то время как метод трехточечной прогонки требует $O(n)$ операций относительно размерности матриц

3.3 Норма невязки

Норма вектора невязки позволяет оценить точность приближенного решения. Она рассчитывается по формуле $\|\mathbf{r}_i\|_2 = \|A_i\mathbf{x} - \mathbf{b}_i\|_2$

Для первой СЛАУ это значение равно $2.5783756124564226e-06$
Для второй СЛАУ это значение равно $1.1758938137872912e-15$
Норма вектора невязки для второй СЛАУ меньше

4 Программная реализация

Необходимые импорты:

```
import math
import array
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
```

4.1 Функции для подсчета нормы вектора невязки

Подсчет произведения матрицы на вектор:

```
def prod(A, b):
    result = []
    for row in A:
        row_sum = 0
        for j in range(len(b)):
            row_sum += row[j] * b[j]
        result.append(row_sum)
    return result
```

Подсчет L2 нормы произвольного вектора:

```
def l2_norm(vec):
    sum = 0
    for coord in vec:
        sum += coord ** 2
    return sum ** 0.5
```

Подсчет L2 нормы невязки:

```
def norm_residual(A, B, f):
    res = prod(A, B)
    for i in range(len(res)):
        res[i] -= f[i]
    norm_res = l2_norm(res)
    return norm_res
```

4.2 Метод трехточечной прогонки

```
def tridiagonal_solver(a, b, c, r):
    n = len(r)
    beta = []
    rho = []
    beta.append(b[0])
    rho.append(r[0])

    for j in range(1, n):
        beta.append(b[j] - a[j-1]*c[j-1]/beta[j-1])
        rho.append(r[j] - a[j-1]*rho[j-1]/beta[j-1])

    x = array.array('f', [0]*n)
    x[n-1] = rho[n-1] / beta[n-1]
    for j in range(1, n-1):
        x[n-j-1] = (rho[n-j-1] - c[n-j-1]*x[n-j]) / beta[n-j-1]

    return x
```

4.3 Построение сплайна

```
N = 100
x = [2*i/N for i in range(0, N+1)]
y = [x[i] * abs(math.cos(5*x[i])) for i in range(0, N+1)]
h = [x[i] - x[i-1] for i in range(1, N+1)]

# tridiagonal matrix
diag_1 = [0] + [h[i] for i in range(2, N-1)] + [0]
diag_2 = [1] + [2*(h[i] + h[i+1]) for i in range(1, N-1)] + [1]

f = [0] + [6*((y[i+2] - y[i+1])/h[i+1] - (y[i+1] - y[i])/h[i]) for i in
    range(1, N-1)] + [0]

# solve a system
ssd = tridiagonal_solver(diag_1, diag_2, diag_1, f)
ssd = np.insert(ssd, 0, 0)

a, b, c, d = [], [], [], []
for j in range(0, N):
    d.append(y[j])
    c.append((y[j+1] - y[j])/h[j] - ((h[j]*ssd[j+1] + 2*h[j]*ssd[j])/6)
    )
    b.append(ssd[j]/2)
    a.append((ssd[j+1] - ssd[j])/(6*h[j]))

# draw a plot of function and its interpolant
fig, ax = plt.subplots(figsize=(12, 6))
colors = plt.cm.get_cmap('cool', N)

plt.plot(x, y, label='Original function', color='black')

for i in range(N+1):
```

```
    plt.scatter(x[i], y[i], s=20, c=[colors(i)], edgecolors='none')

for i in range(0, N):
    xs = np.linspace(x[i], x[i+1], N)
    ys = a[i]*(xs - x[i])**3 + b[i]*(xs - x[i])**2 + c[i]*(xs - x[i]) +
    d[i]
    plt.plot(xs, ys, color=colors(i))

plt.xlabel('x')
plt.ylabel('y(x)')
plt.grid(True)
plt.legend()
plt.show()
```

Построенный сплайн:

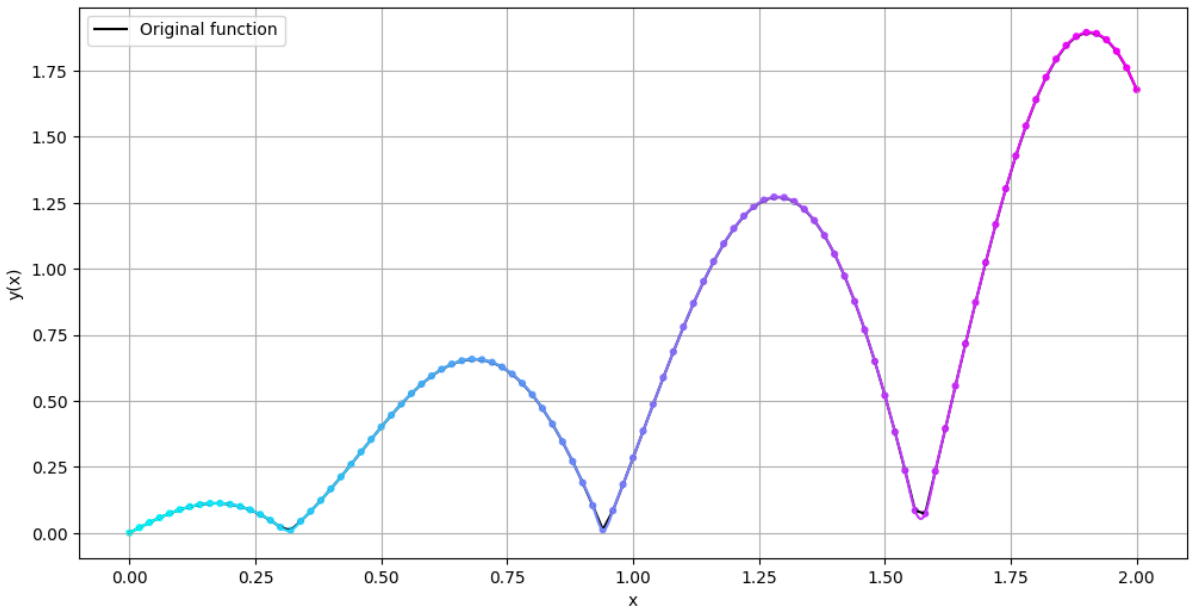

Рис. 4.1: Построенный сплайн

Расчет невязки для СЛАУ ($Ax = f$), получающейся при построении кубического сплайна:

```
A = np.diag(diag_1, -1) + np.diag(diag_2, 0) + np.diag(diag_1, 1)

print(A)
print(f)

# calculate the residual
norm_res = norm_residual(A, ssd[1:], f)
print(norm_res)
```

4.4 Метод Гаусса

```python
def gaussian_elimination(A, b):
    n = len(A)

    for i in range(n):
        for j in range(i+1, n):
            factor = A[j][i] / A[i][i]
            for k in range(i, n):
                A[j][k] -= factor * A[i][k]
            b[j] -= factor * b[i]

    x = [0] * n
    for i in range(n-1, -1, -1):
        x[i] = (b[i] - sum(A[i][j] * x[j] for j in range(i+1, n))) / A[
    i][i]

    return x
```

```python
n = 100

# create matrix A
A = np.zeros((n+1, n+1))
A[0, 0] = 1
A[1, 0] = -1
A[1, 1] = 1
for i in range(2, n-1):
    A[i, i-2 : i+3] = [1, -4, 6, -4, 1]
A[n-1, n-1] = -1
A[n-1, n] = 1
A[n, n] = 1

# create vector f
f = [1, 0]
h = math.pi/n
for l in range(2, n-1):
    f.append(math.cos(l*h) * h**4)
f.extend([0, 1])
f = np.array(f)

# solve a system
print(A)
print(f)
x = gaussian_elimination(A, f)
print(x)

# calculate the residual
norm_res = norm_residual(A, x, f)
print(norm_res)
```

5 Заключение

Метод Гаусса подходит для большего количества СЛАУ, но может быть вычислительно дорогостоящим. В то время как метод трёхточечной прогонки применим к меньшему количеству СЛАУ, однако требует меньшее количество операций.

СЛАУ с такими матрицами возникают при решении дифференциальных уравнений, моделирующих различные физические процессы, такие как теплопроводность, распространение сигналов в электронных схемах и др. Такие системы получаются при численном решении краевых задач для дифференциальных уравнений.

Методы решения таких систем, например, метод прогонки и метод Гаусса, являются эффективными при нахождении решений, что делает их важными в физических моделях и численных методах решения дифференциальных уравнений.

Литература

[1] Костомаров *Вводные лекции по ЧМ*. Москва: Логос, 2004

[2] К.Б. Джакупов. *Вычислительная механика*. Москва: Физматлит, 2005.

[3] А.А. Самарский, Е.С. Николаев. *Методы решения сеточных уравнений*. Москва: Наука, 1983.

[4] Paul L. DeVries, Javier Hasbun. *A First Course in Computational Physics*, Second Edition. John Wiley & Sons, Inc., 2006.