Лабораторная работа 4 по МВМ
Моисеев Дмитрий А–14–21

Рассматривается задача конвекции - диффузии:

$$\begin{cases} - \dfrac{\partial}{\partial x}\left(k_1(x,y,u)\dfrac{\partial u}{\partial x}\right) - \dfrac{\partial}{\partial y}\left(k_2(x,y,u)\dfrac{\partial u}{\partial y}\right) + v_1\dfrac{\partial u}{\partial x} + v_2\dfrac{\partial u}{\partial y} = f \\ \mathrm{u}\big|_{\partial\Omega} = g \end{cases}$$

в прямоугольной области $\Omega = \{(x,y) \in \mathbb{R}^2 | 0 < x < X, 0 < y < Y\}$
Здесь $v = (v_1(x,y,u), v_2(x,y,u))^T$ - заданный вектор скорости.
Заменим область Ω конечномерной сеткой, а производные их разностными аппроксимациями. В результате получим систему уравнений:
$-\Big(L_x(i,j,u) + L_y(i,j,u)\Big) + D_x(i,j,u) + D_y(i,j,u) = f(i \cdot h_x, j \cdot h_y, u)$ где операторы определены следующим образом:

$$L_x(i,j,u) = \frac{a_x(i+1,j,u)\cdot(u[i+1,j] - u[i,j]) - a_x(i,j,u)\cdot(u[i,j] - u[i-1,j])}{h_x^2},$$

$$L_y(i,j,u) = \frac{a_y(i,j+1,u)\cdot(u[i,j+1] - u[i,j]) - a_y(i,j,u)\cdot(u[i,j] - u[i,j-1])}{h_y^2},$$

$$a_x(i,j,u) = \frac{k_1(i\cdot h_x, j\cdot h_y, u[i,j]) + k_1((i-1)\cdot h_x, j\cdot h_y, u[i-1,j])}{2},$$

$$a_y(i,j,u) = \frac{k_2(i\cdot h_x, j\cdot h_y, u[i,j]) + k_2(i\cdot h_x, (j-1)\cdot h_y, u[i,j-1])}{2}.$$

для узлов внутри области
Используем разности против потока:

$$D_x u_{ij} = \frac{1}{h_x}\begin{cases} \mathrm{u}_{ij} - u_{i-1,j}, \text{ если } \mathrm{v}_{1ij} > 0, \\ \mathrm{u}_{i+1,j} - u_{i,j}, \text{ если } \mathrm{v}_{1ij} < 0, \end{cases}$$

$$D_y u_{ij} = \frac{1}{h_y}\begin{cases} \mathrm{u}_{ij} - u_{i,j-1}, \text{ если } \mathrm{v}_{2ij} > 0, \\ \mathrm{u}_{i,j+1} - u_{i,j}, \text{ если } \mathrm{v}_{2ij} < 0, \end{cases}$$

1. Постановка задачи

Рассматриваем задачу, решаемую методом Ньютона-Крылова (JFNK), где требуется решение системы нелинейных уравнений вида:222ее

$$F(p) = 0$$

где p — это вектор неизвестных, а $F(p)$ — это нелинейная функция, зависящая от p. Метод Ньютона-Крылова используется для приближенного решения данной системы, где на каждом шаге аппроксимируем нелинейную задачу линейной системой.

2. Линеаризация задачи

Метод Ньютона для решения нелинейной задачи предполагает линейную аппроксимацию функции в текущей точке p_{solv}:

$$F(p) \approx F(p_{\text{solv}}) + J(p_{\text{solv}}) \cdot (p - p_{\text{solv}})$$

где $J(p_{\text{solv}})$ — это якобиан функции $F(p)$ в точке p_{solv}, а p — это новый вектор, который мы ищем.

Предположим, что $F(p_{\text{solv}})$ уже вычислено, и нам нужно решить систему линейных уравнений для обновления вектора p:

$$J(p_{\text{solv}}) \cdot \Delta p = -F(p_{\text{solv}})$$

где Δp — это шаг для обновления решения.

3. Упрощение задачи с использованием метода Ньютона-Крылова

Вместо явного вычисления якобиана и решения линейной системы с использованием полной матрицы $J(p_{\text{solv}})$, в методах Ньютона-Крылова используется так называемая "свободная форма Якоби"(Jacobian-free), что означает, что матрица якобиана не вычисляется напрямую. Вместо этого задача сводится к вычислению оператора $J(p_{\text{solv}}) \cdot v$ для некоторого вектора v через умножение на аппроксимированную матрицу оператора.

Таким образом, задача на каждом шаге сводится к решению следующей линейной системы:

$$A(p_{\text{solv}}) \cdot \Delta p = -F(p_{\text{solv}})$$

где $A(p_{\text{solv}})$ представляет собой аппроксимированную матрицу системы, зависящую от текущего решения p_{solv}, а $F(p_{\text{solv}})$ — это вектор невязки.

4. Применение метода BiCGStab

Теперь задача сводится к решению линейной системы с использованием метода BiCGStab. Он решает систему уравнений вида:

$$A \cdot x = b$$

где A — это матрица системы (в нашем случае, оператор $A(p_{\text{solv}})$, зависящий от текущего приближения p_{solv}), а b — вектор правой части (в данном случае, это $-F(p_{\text{solv}})$).

Находим решение задачи с помощью метода Ньютона-Крылова.

Примеры:

1) искомое решение $u(x, y) = x + 100y + 10000$
$v_1(x, y) = 0, v_2(x, y) = 0$
$k_1(x, y, u) = 1, k_2(x, y, u) = 1$
$f(x, y) = 0$
$X = 0.1, Y = 0.1$ - размеры области

N_x, N_y	Погрешность	Время
10	0.0146	0.0420
20	0.0312	0.5607
40	0.0642	5.9445
80	0.1302	59.6068

Num_1.png

2) искомое решение $u(x, y) = \pi * \cos(x) * \sin(x * y)$
$f(x, y) = \pi \left(\cos(y) \sin(xy) \left(1 + xy^2\right) + x \left(2 \sin(y) \cos(xy) + \cos(y) x \sin(xy)\right) - \sin(y) \sin(xy)\right.$
$v_1(x, y) = 0, v_2(x, y) = 0$
$k_1(x, y, u) = x, k_2(x, y, u) = 1$
$X = 2, Y = 2$

N_x, N_y	Погрешность	Время
10	0.0921	0.1806
20	0.0438	1.9751
40	0.0214	20.5081
80	0.0105	264.7923

Num_2.png

3) искомое решение $u(x, y) = x^2 + y^2$
$f(x, y) = -2 * (1 + (x^2 + y^2) * (5 * x^2 + y^2) - y^2)$
$v_1(x, y) = 0, v_2(x, y) = y$
$k_1(x, y, u) = u^2, k_2(x, y, u) = 1$
$X = 0.5, Y = 0.5$

N_x, N_y	Погрешность	Время
10	1.2054e-09	0.1484
20	1.1837e-09	1.6593
40	2.6071e-09	14.6416
80	0.3232	589.6798

Num_3.png

код функции метода Ньютона-Крылова:

```python
def solve_jfnk(p_solv, tol=1e-5, max_iter=100):
    delta = solve_bicgstab_jfnk(-Au(p_solv), p_solv)
    p_solv += delta
    for k in range(max_iter):
        if np.linalg.norm(delta) < tol:
            break
        #print(p_solv)
        delta = solve_bicgstab_jfnk(-Au(p_solv), p_solv, tol)
        p_solv += delta
        print(np.linalg.norm(p_solv))

    return p_solv

def solve_bicgstab_jfnk(b, p_solv, tol=1e-5, max_iter=1000):
    x = np.zeros_like(b)
    #print(b, apply_jacobian_free(p_solv, x), sep="\n\n")
    r = b - Ad(p_solv, x)

    r0_hat = r.copy()
    rho_old = alpha = omega = 1.0
    v = p = np.zeros_like(b)

    for iteration in range(max_iter):
        rho_new = sc_mult(r0_hat, r)
        if abs(rho_new) < 1e-10:
            print(f"Interrupt: rho_new is too small.")
            break
        if iteration == 0:
            p = r.copy()
        else:
            #print(f"rho_old = {rho_old}, omega = {omega}")
            beta = (rho_new / rho_old) * (alpha / omega)
            p = r + beta * (p - omega * v)

        v = Ad(p_solv, p)
        alpha = rho_new / sc_mult(r0_hat, v)
        s = r - alpha * v

        if np.linalg.norm(s) < tol:
            x += alpha * p
            print(f"It came together for {iteration + 1} iteration.")
            break

        t = Ad(p_solv, s)
        omega = sc_mult(t, s) / sc_mult(t, t)

        x += alpha * p + omega * s
        r = s - omega * t

        if np.linalg.norm(r) < tol:
            break

        rho_old = rho_new
```

```
54
55      return x
56
57  def Ad(x, w, h=1e-8):
58      norm_x = np.linalg.norm(x)
59      norm_w = np.linalg.norm(w)
60
61      if norm_x != 0 and norm_w != 0:
62          delta = h * norm_x / norm_w
63          return (Au(x + delta * w) - Au(x)) / delta
64      elif norm_x == 0 and norm_w != 0:
65          delta = h / norm_w
66          return (Au(delta * w) - Au(np.zeros_like(x))) / delta
67      elif norm_x == 0 and norm_w == 0:
68          return np.zeros_like(w)
69      else:
70          return np.zeros_like(w)
```

5. Вывод

5.1. **Нелинейность уравнений.** Эта задача включает решение нелинейной системы уравнений, где зависимость от переменных и параметров имеет нелинейный характер. Метод Ньютона-Крылова эффективен для таких задач, так как он позволяет на каждом шаге линейно аппроксимировать нелинейные уравнения с использованием итерационного процесса. Это позволяет избегать явного вычисления полной матрицы Якоби и позволяет работать с операторами, зависящими от текущего приближения решения, что делает метод особенно удобным в контексте нелинейных задач.

5.2. **Скорость сходимости.** Метод Ньютона-Крылова обладает хорошей сходимостью, особенно для задач, где начальные приближения близки к точному решению. В моих примерах можно заметить, что при увеличении числа узлов (N_x, N_y) погрешность уменьшается, но время решения растет пропорционально размеру задачи. Однако благодаря использованию метода BiCGStab (который является частью алгоритма Ньютона-Крылова) можно эффективно решать возникающие линейные системы на каждом шаге, что значительно улучшает общую эффективность решения.

5.3. **Экономия памяти и вычислительных ресурсов.** Одним из преимуществ метода Ньютона-Крылова является возможность работы с так называемой "свободной формой Якоби"(Jacobian-free), что позволяет избегать хранения и вычисления полного Якобиана системы. Это критично для больших и сложных задач, где размерность системы может быть очень большой, и хранение полной матрицы Якоби может быть невозможно или неэффективно.