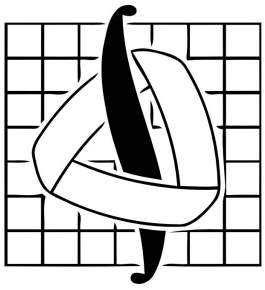
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
имени М.В.Ломоносова
механико-математический факультет



Расчёт конвективного движения жидкости в полости прямоугольного сечения при подогреве, моделирование возникновения конвекнивных валов и ячеек Бенара

Студент:
Мирмоминов Руслан Мэргыязович

Преподаватель:
Корнев Андрей Алексеевич

Москва, 2020 г.

1 Постановка задачи

Требуется рассчитать конвективное движение жидкости в полости квадратного сечения при подогреве. Для этого следует совместно решить систему ДУЧП из уравнений Навье - Стокса в переменных «функция тока – вихрь скорости» и уравнения теплопроводности:

$$\omega = \omega(x, y, t),\ \psi = \psi(x, y, t),\ T = T(x, y, t),$$

$$\frac{\partial \omega}{\partial \tau} = \nu \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right) - \frac{\partial \omega}{\partial x} \cdot \frac{\partial \psi}{\partial y} + \frac{\partial \omega}{\partial y} \cdot \frac{\partial \psi}{\partial x} + G \frac{\partial T}{\partial x},$$

$$\frac{\partial T}{\partial \tau} = k \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) - \frac{\partial \psi}{\partial y} \cdot \frac{\partial T}{\partial x} + \frac{\partial \psi}{\partial x} \cdot \frac{\partial T}{\partial y} + \frac{q(x, y)}{\rho \cdot c},$$

$$\omega = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2},$$

$$(x, y) \in \Omega = [0, 1] \times [0, 1],\ t \geq 0,$$

$$\omega, \psi, T|_{\partial \Omega} = 0$$

где ω – вихрь скорости, ψ – функция тока, T – температура полости; эти функции и необходимо найти. Считаем известными функцию мощности нагревающего источника $q(x, y)$ и константы ρ, c, k, ν, G.

2 Решение

Будем решать сформулированную дифференциальную задачу на множестве $D = [0,1] \times [0,1] \times [0,1]$. Для этого построим разностную схему на сетке D_h, которую положим равной

$$\{ih_x;\ i = \overline{0, N_x},\ N_x h_x = 1\} \times \{jh_y;\ j = \overline{0, N_y},\ N_y h_y = 1\} \times \{k\tau;\ i = \overline{0, M},\ M\tau = 1\},$$

после чего найдём порядок аппроксимации схемы на решении и вычислим решение. Итак, рассмотрим следующую разностную схему:

$$\frac{w_{i,j}^{t+1} - w_{i,j}^{t}}{\tau} = \nu \left(\frac{w_{i+1,j}^{t+1} - 2w_{i,j}^{t+1} + w_{i-1,j}^{t+1}}{h_x^2} + \frac{w_{i,j+1}^{t+1} - 2w_{i,j}^{t+1} + w_{i,j-1}^{t+1}}{h_y^2} \right)$$

$$- \frac{(w_{i+1,j}^{t} - w_{i-1,j}^{t})(\psi_{i,j+1}^{t} - \psi_{i,j-1}^{t})}{4h_x h_y} + \frac{(w_{i,j+1}^{t} - w_{i,j-1}^{t})(\psi_{i+1,j}^{t} - \psi_{i-1,j}^{t})}{4h_x h_y}$$

$$+ G \frac{T_{i+1,j}^{t+1} - T_{i-1,j}^{t+1}}{2h_x}, \quad i = \overline{1, N_x - 1}, \quad j = \overline{1, N_y - 1}, \quad t = \overline{0, M - 1},$$

$$\frac{T_{i,j}^{t+1} - T_{i,j}^{t}}{\tau} = k \left(\frac{T_{i+1,j}^{t+1} - 2T_{i,j}^{t+1} + T_{i-1,j}^{t+1}}{h_x^2} + \frac{T_{i,j+1}^{t+1} - 2T_{i,j}^{t+1} + T_{i,j-1}^{t+1}}{h_y^2} \right)$$

$$- \frac{(\psi_{i,j+1}^{t} - \psi_{i,j-1}^{t})(T_{i+1,j}^{t} - T_{i-1,j}^{t})}{4h_x h_y} + \frac{(\psi_{i+1,j}^{t} - \psi_{i-1,j}^{t})(T_{i,j+1}^{t} - T_{i,j-1}^{t})}{4h_x h_y} + \frac{q_{i,j}}{\rho \cdot c},$$

$$i = \overline{1, N_x - 1}, \quad j = \overline{1, N_y - 1}, \quad t = \overline{0, M - 1},$$

$$w_{i,j}^{t+1} = \frac{\psi_{i+1,j}^{t+1} - 2\psi_{i,j}^{t+1} + \psi_{i-1,j}^{t+1}}{h_x^2} + \frac{\psi_{i,j+1}^{t+1} - 2\psi_{i,j}^{t+1} + \psi_{i,j-1}^{t+1}}{h_y^2}, \quad i = \overline{1, N_x - 1}, \quad j = \overline{1, N_y - 1},$$

$$\omega_{i,0}^{t} = \omega_{0,j}^{t} = \omega_{i,N_x}^{t} = \omega_{N_y,j}^{t} = \psi_{i,0}^{t} = \psi_{0,j}^{t} = \psi_{i,N_x}^{t} = \psi_{N_y,j}^{t} = 0,$$

$$T_{i,0}^{t} = T_{0,j}^{t} = T_{i,N_x}^{t} = T_{N_y,j}^{t} = 0, \quad i = \overline{0, N_x}, \quad j = \overline{0, N_y},$$

где $f_{i,j}^{k} = f(x_i, y_j, t_k)$ для любой функции f.

Порядок аппроксимации на решении для данной разностной схемы очевиден: $O(\tau + h_x^2 + h_y^2),\ N_x,\ N_y,\ M \to +\infty$. Это следует из того, что для любой достаточно гладкой функции $f(x, y, t)$ формула Тейлора даёт

$$f(x + h, y, t) - f(x - h, y, t) = 2h \frac{\partial f}{\partial x}(x, y, t) + O(h^3), \quad h \to 0,$$

$$f(x-h,y,t)-2f(x,y,t)+f(x+h,y,t)=h^2\frac{\partial^2 f}{\partial x^2}(x,y,t)+O(h^4),\quad h\to 0,$$

$$f(x,y,t+\tau)-f(x,y,t)=\tau\frac{\partial f}{\partial t}(x,y,t)+O(\tau^2),\quad \tau\to 0.$$

Для того, чтобы решить задачу, воспользуемся методом Фурье. Заметим, что уравнения на внутренних точках сетки можно записать в операторном виде следующим образом:

$$\left(\frac{1}{\tau}I-\nu\Lambda_{xx}-\nu\Lambda_{yy}\right)\omega^{t+1}=f_1(\omega^t,\psi^t,T^t,T^{t+1}),$$

$$\left(\frac{1}{\tau}I-k\Lambda_{xx}-k\Lambda_{yy}\right)T^{t+1}=f_2(\omega^t,\psi^t,T^t),$$

$$\left(-\Lambda_{xx}-\Lambda_{yy}\right)\psi^{t+1}=\omega^{t+1}.$$

Тогда можно поступить так: на очередном шаге t сначала получать T^{t+1} из второго уравнения, затем ω^{t+1} из первого, и в конце шага – ψ^{t+1} из третьего. Для каждого из операторов известны собственные значения и собственные функции. В частности, для оператора $(-\Lambda_{xx}-\Lambda_{yy})$ собственные значения равны

$$\frac{4}{h_x^2}\sin^2\frac{\pi s h_x}{2}+\frac{4}{h_y^2}\sin^2\frac{\pi k h_y}{2},\quad s=\overline{1,N_x-1},\quad k=\overline{1,N_y-1},$$

а собственные функции –

$$2\sin(\pi i s h_x)\cdot\sin(\pi j k h_y),\quad i=\overline{1,N_x-1},\quad \overline{1,N_y-1}.$$

Итак, пусть решаемое уравнение – $A\omega=f$, где оператор A имеет собственные значения λ_{sk} и собственные функции $\phi_s\cdot\psi_k$, образующие ортонормированный базис относительно скалярного произведения

$$(f,g)_{h_x,h_y}=\sum_{s=1}^{N_x-1}\sum_{k=1}^{N_y-1}f_{sk}g_{sk}h_xh_y.$$

Пусть $\omega_{ij}=\sum_{s=1}^{N_x-1}\sum_{k=1}^{N_y-1}c_{sk}\phi_s^i\psi_k^j$. Покажем, как можно за кубическое время найти коэффициенты c, а затем и значения ω. По определению

$$(A\omega)_{ij}=\sum_{s=1}^{N_x-1}\sum_{k=1}^{N_y-1}\lambda_{sk}c_{sk}\phi_s^i\psi_k^j=\sum_{k=1}^{N_y-1}\psi_k^j\left(\sum_{s=1}^{N_x-1}\lambda_{sk}c_{sk}\phi_s^i\right)=\sum_{k=1}^{N_y-1}\psi_k^jd_k^i$$

$$\Rightarrow d_k^i = \sum_{j=1}^{N_y-1} h_y \psi_k^j f_{ij},$$

значит, подсчёт d_k^i можно организовать за кубическое время. При этом

$$d_k^i = \sum_{s=1}^{N_x-1} \lambda_{sk} c_{sk} \phi_s^i \Rightarrow c_{sk} = \frac{1}{\lambda_{sk}} \sum_{i=1}^{N_x-1} \phi_s^i d_k^i h_x,$$

что тоже можно вычислить за $O((\max(N_x, N_y))^3)$. Итак, коэффициенты удалось найти. Покажем, как теперь посчитать все значения ω. Пусть

$$\hat{d_k^i} = \sum_{s=1}^{N_x-1} c_{sk} \phi_s^i.$$

Тогда

$$\omega_{ij} = \sum_{k=1}^{N_y-1} \psi_k^j \hat{d_k^i}.$$

Обе процедуры можно проделать прямым вычислением за $O((\max(N_x, N_y)^3)$. Таким образом, один шаг решения нашей разностной схемы требует кубического времени. Итоговая асимптотика всей программы - $O(M(\max(N_x, N_y))^3)$.

3 Описание программы

Функция

```
template<class Func>
void Solve(int nx, int ny, int M, Params<Func> params);
```

получает на вход размерности задачи и структуру, содержащую значения констант и функцию мощности нагревателя.

Внутри этой функции запускается цикл, на каждом шаге запускающий решение каждого из трёх уравнений с помощью следующих функций:

```
template<class Func>
std::vector<double> NewT(std::vector<double>& wP,
    std::vector<double>& psiP, std::vector<double>& TP, int nx, int ny,
    int M, Params<Func> params);

template<class Func>
std::vector<double> NewW(std::vector<double>& T,
    std::vector<double>& wP,
    std::vector<double>& psiP, int nx, int ny, int M,
    Params<Func> params);

template<class Func>
std::vector<double> NewPsi(std::vector<double>& w, int nx, int ny,
    int M, Params<Func> params);
```

Внутри этих функций создаётся объект параметризованного класса

```
template<class Func>
class Fourier {
public:
    Fourier(int nx, int ny, Func func) : nx_(nx), ny_(ny),
    eigen_value_(func) {}
    double Basis(int i, int s, double h);
    double FToD(const std::vector<double>& values, int i, int k);
    std::vector<double> FToDMatrix(const std::vector<double>& values);
    double DToC(const std::vector<double>& d, int s, int k);
    std::vector<double> DToCMatrix(const std::vector<double>& d);
    double CToD(const std::vector<double>& c, int i, int k);
    std::vector<double> CToDMatrix(const std::vector<double>& c);
    double DToF(const std::vector<double>& d, int i, int j);
    std::vector<double> DToFMatrix(const std::vector<double>& d);
    std::vector<double> Solve(const std::vector<double>& values);
private:
    Func eigen_value_;
    int nx_;
    int ny_;
```

```
};
```

Его основная функция

```
std::vector<double> Solve(const std::vector<double>& values);
```

получает на вход вектор правой части, который образуют для каждого уравнения функции

```
template<class Func>
std::vector<double> RightPartForT(std::vector<double>& wP,
    std::vector<double>& psiP, std::vector<double>& TP, int nx, int ny,
    int M, Params<Func> params);

template<class Func>
std::vector<double> RightPartForW(std::vector<double>& T,
    std::vector<double>& wP, std::vector<double>& psiP, int nx, int ny,
    int M, Params<Func> params);

template<class Func>
std::vector<double> RightPartForPsi(std::vector<double>& w, int nx,
    int ny, int M, Params<Func> params);
```

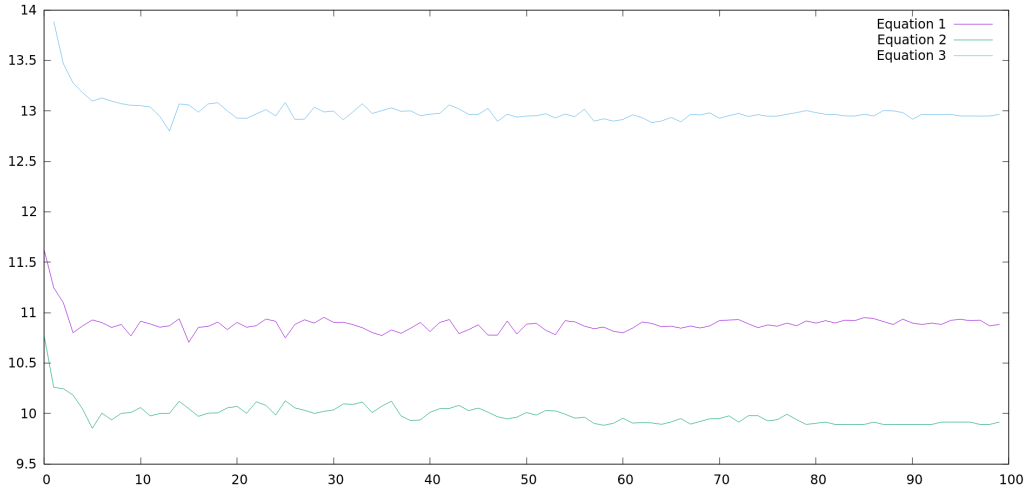
В конце работы программы функции

```
template<class Func>
double CheckFirst(std::vector<std::vector<double>>& ws,
std::vector<std::vector<double>>& psis,
    std::vector<std::vector<double>>& Ts,
    int nx, int ny, int M, Params<Func> params);

template<class Func>
double CheckSecond(std::vector<std::vector<double>>& ws,
    std::vector<std::vector<double>>& psis,
    std::vector<std::vector<double>>& Ts,
    int nx, int ny, int M, Params<Func> params);

template<class Func>
double CheckThird(std::vector<std::vector<double>>& ws,
    std::vector<std::vector<double>>& psis,
    std::vector<std::vector<double>>& Ts,
    int nx, int ny, int M, Params<Func> params);
```

вычисляют погрешности на каждом из трёх уравнений.

4 Результаты вычислений

Запуск при $\nu = k = 3,\ G = 10,\ q = 100,\ \rho = c = 1,\ N_x = M_x = 50,\ M = 100$ дал следующий результат (на графике зависимость отрицательного десятичного логарифма погрешности от номера итерации):



Кроме того, строится gif - анимация, визуализирующая изменение поля скоростей и температуры на каждом шаге: