Теплопроводность, детерминированное горение

Этап №3

Махорин И., Шаповалова Д., Егорова Ю., Лебединец Т., Павлова В., Великоднева Е.

24 февраля 2024

Российский университет дружбы народов, Москва, Россия

Вводная часть

Цели проекта

- Описать программную реализацию проекта

Основная часть

Выбор языка программирования

Для реализации алгоритмов, описанных на предыдущем этапе проекта, был выбран язык программирования Julia и Python.

Структура программного кода

Программа будет структурирована в виде набора модулей, каждый из которых реализует определенные части алгоритмов. Основные модули будут включать в себя реализацию уравнений теплопроводности и детерминированного горения, а также модули для визуализации результатов.

Реализация алгоритмов

Модуль уравнений теплопроводности и горения:

Реализация численных методов для решения размерной системы уравнений, описывающих процессы теплопроводности и горения. Использование явных или неявных методов конечных разностей для численного интегрирования уравнений.

Модуль визуализации результатов:

Визуализация процессов теплопроводности и горения с использованием библиотеки Matplotlib. Создание графиков распределения температуры и концентрации продуктов горения в пространстве и времени..

Ввод и вывод данных

Входные данные будут представлены в виде файлов, содержащих начальные условия, параметры системы и другие необходимые данные. Результаты будут сохранены в файлы формата CSV или изображения графиков для последующего анализа и визуализации.

Тестирование и отладка

Проведение систематического тестирования программы на различных наборах входных данных.

Использование отладчика Python для выявления и устранения ошибок в программном коде.

Реализация на языке Julia

```
using Printf

function calculate_temperature(N, t_end, L, lamda, ro, c, kapa, Te, q, k0, E, T0)
    mf = 1000
    eps = 1e-5
    R = 8.31

    # Определяем расчетный шаг сетки по пространственной координате
    h = L / (N - 1)
    # Определяем расчетный шаг сетки по времени
    tau = t_end / 1000.0

    # Инициализируем массивы для температур и прогоночных коэффициентов
    T = fill(T0, N)
    alfa = fill(0.0, mf)
    beta = fill(0.0, mf)
    Tn = fill(0.0, mf)
    Ts = fill(0.0, mf)

    # Проводим интегрирование нестационарного уравнения теплопроводности
    time = 0
    while time < t_end
        time += tau

        # Определяем начальные прогоночные коэффициенты на основе левого граничного условия
        alfa[2] = 2.0 * tau * lamda / (2.0 * tau * (lamda + kapa * h) + ro * c * (h ^ 2))
        beta[2] = (ro * c * (h ^ 2) * T[1] + 2.0 * tau * kapa * h * Te) / (2.0 * tau * (lamda + kapa * h) + ro * c * (h ^ 2))

        # Запоминаем поле температуры на предыдущем временном слое
        Tn .= T

        # Цикл с постусловием, позволяющий итерационно вычислять поле температуры
        while true
            # Запоминаем поле температуры на предыдущей итерации
            Ts .= T

            # Цикл с параметром для определения прогоночных коэффициентов
            for i in 2:N-1
                ai = lamda / (h ^ 2)
                bi = 2.0 * lamda / (h ^ 2) + ro * c / tau
                ci = lamda / (h ^ 2)
                fi = -ro * c * Tn[i] / tau - q * k0 * ro * exp(-E / (R * Tn[i]))

                alfa[i] = ai / (bi - ci * alfa[i - 1])
                beta[i] = (ci * beta[i - 1] - fi) / (bi - ci * alfa[i - 1])
            end

            # Определяем значение температуры на правой границе на основе правого граничного условия
            T[N] = (ro * c * (h ^ 2) * Tn[N] + 2.0 * tau * (lamda * beta[N - 1] + kapa * h * Te)) /
                   (ro * c * (h ^ 2) + 2.0 * tau * (lamda * (1 - alfa[N - 1]) + kapa * h))

            # Используя соотношение, определяем неизвестное поле температуры
            for i in N-1:-1:2
                T[i] = alfa[i] * T[i + 1] + beta[i]
            end

            # Определяем максимум модуля разности температур на данной и предыдущей итерации
            max_diff = maximum(abs.(T - Ts))
            if max_diff <= eps
                break  # Выходим из цикла, если достигнута необходимая точность
            end
```

Реализация на языке Julia

```
    end
    return T
end

# Вводим параметры
kapa = 40
Te = 243
q = 1000
k0 = 30000
E = 36500
N = 1000
t_end = 1800
L = 0.2
lamda = 0.7
ro = 1500
c = 750
T0 = 298

# Рассчитываем температурное поле
temperature_field = calculate_temperature(N, t_end, L, lamda, ro, c, kapa, Te, q, k0, E, T0)

# Выводим результаты в файл
open("res.txt", "w") do f
    @printf(f, "Толщина пластины L = %.4f\n", L)
    @printf(f, "Число узлов по координате N = %d\n", N)
    @printf(f, "Коэффициент теплопроводности материала пластины lamda = %.4f\n", lamda)
    @printf(f, "Плотность материала пластины ro = %.4f\n", ro)
    @printf(f, "Теплоемкость материала пластины c = %.4f\n", c)
    @printf(f, "Коэффициент теплообмена kapa = %.4f\n", kapa)
    @printf(f, "Температура внешней среды Te = %.4f\n", Te)
    @printf(f, "Тепловой эффект химической реакции q = %.4f\n", q)
    @printf(f, "Предэкспонент k0 = %.4f\n", k0)
    @printf(f, "Энергия активации химической реакции E = %.4f\n", E)
    @printf(f, "Температурное поле в момент времени t = %.4f\n", t_end)
end

# Визуализация результатов
using Plots
t = collect(0:1.8:t_end)
plot(t, temperature_field, xlabel="Time", ylabel="Temperature", title="Temperature Field", legend=false)
```

Реализация на Python

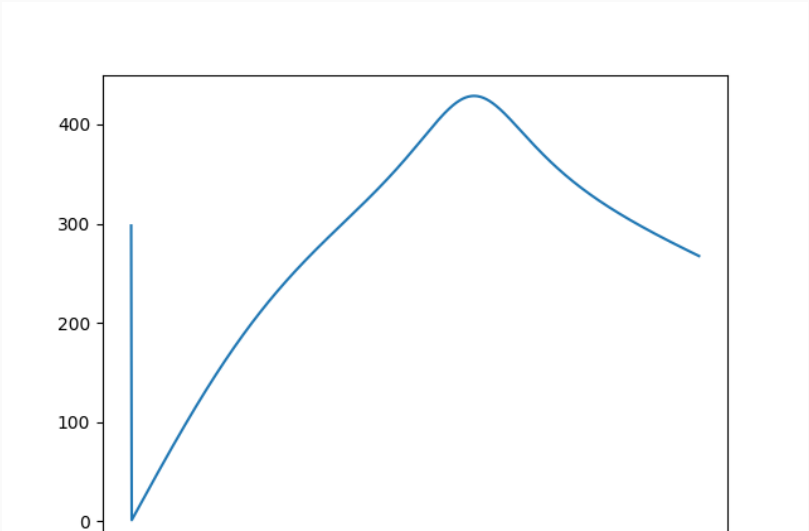
```
import math
import matplotlib.pyplot as plt
import numpy as np

# Функция для расчета поля температур методом конечных разностей
def calculate_temperature(N, t_end, L, lamda, ro, c, kapa, Te, q, k0, E, T0):
    mf = 1000
    eps = 1e-5
    R = 8.31

    # Определяем расчетный шаг сетки по пространственной координате
    h = L / (N - 1)
    # Определяем расчетный шаг сетки по времени
    tau = t_end / 1000.0

    # Инициализируем массивы для температур и прогоночных коэффициентов
    T = [T0] * N
    alfa = [0] * mf
    beta = [0] * mf
    Tn = [0] * mf
    Ts = [0] * mf

    # Проводим интегрирование нестационарного уравнения теплопроводности
    time = 0
    while time < t_end:
        time += tau

        # Определяем начальные прогоночные коэффициенты на основе левого граничного условия
        alfa[1] = 2.0 * tau * lamda / (2.0 * tau * (lamda + kapa * h) + ro * c * (h ** 2))
        beta[1] = (ro * c * (h ** 2) * T[0] + 2.0 * tau * kapa * h * Te) / (2.0 * tau * (lamda + kapa * h) + ro * c * (h ** 2))

        # Запоминаем поле температуры на предыдущем временном слое
        Tn = T.copy()

        # Цикл с постусловием, позволяющий итерационно вычислять поле температуры
        while True:
            # Запоминаем поле температуры на предыдущей итерации
            Ts = T.copy()

            # Цикл с параметром для определения прогоночных коэффициентов
            for i in range(1, N - 1):
                ai = lamda / (h ** 2)
                bi = 2.0 * lamda / (h ** 2) + ro * c / tau
                ci = lamda / (h ** 2)
                fi = -ro * c * Tn[i] / tau - q * k0 * ro * math.exp(-E / (R * Tn[i]))

                alfa[i] = ai / (bi - ci * alfa[i - 1])
                beta[i] = (ci * beta[i - 1] - fi) / (bi - ci * alfa[i - 1])

            # Определяем значение температуры на правой границе на основе правого граничного условия
            T[N - 1] = (ro * c * (h ** 2) * Tn[N - 1] + 2.0 * tau * (lamda * beta[N - 2] + kapa * h * Te)) / \
                       (ro * c * (h ** 2) + 2.0 * tau * (lamda * (1 - alfa[N - 2]) + kapa * h))

            # Используя соотношение, определяем неизвестное поле температуры
            for i in range(N - 2, 0, -1):
                T[i] = alfa[i] * T[i + 1] + beta[i]

            # Определяем максимум модуля разности температур на данной и предыдущей итерации
            max_diff = max([abs(T[i] - Ts[i]) for i in range(N)])
            if max_diff <= eps:
                break  # Выходим из цикла, если достигнута необходимая точность
```

Реализация на Python

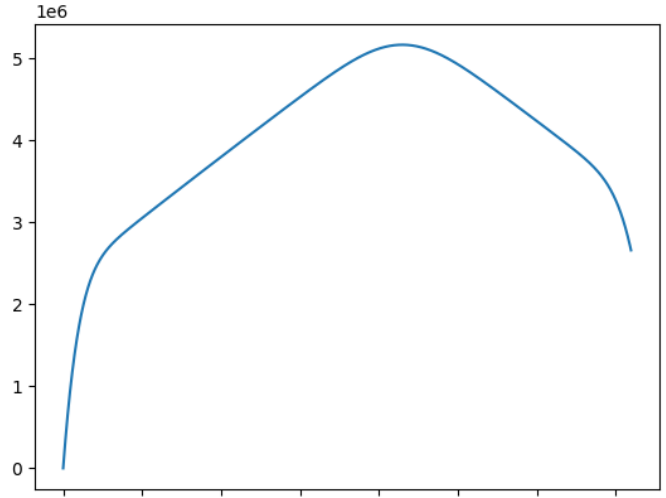
```
    return T

# Вводим параметры с клавиатуры
kapa = 40
Te = 243
q = 1000
k0 = 30000
E = 10
N = 1000
t_end = 1800
L = 0.2
lamda = 0.7
ro = 1500
c = 750
T0 = 298
Tl = 300
Tr = 100
# Рассчитываем температурное поле
temperature_field = calculate_temperature(N, t_end, L, lamda, ro, c, kapa, Te, q, k0, E, T0)

# Выводим результаты в файл
with open('res.txt', 'w') as f:
    f.write(f'Толщина пластины L = {L:.4f}\n')
    f.write(f'Число узлов по координате N = {N}\n')
    f.write(f'Коэффициент теплопроводности материала пластины lamda = {lamda:.4f}\n')
    f.write(f'Плотность материала пластины ro = {ro:.4f}\n')
    f.write(f'Теплоемкость материала пластины c = {c:.4f}\n')
    f.write(f'Коэффициент теплообмена kapa = {kapa:.4f}\n')
    f.write(f'Температура внешней среды Te = {Te:.4f}\n')
    f.write(f'Тепловой эффект химической реакции q = {q:.4f}\n')
    f.write(f'Предэкспонент k0 = {k0:.4f}\n')
    f.write(f'Энергия активации химической реакции E = {E:.4f}\n')
    # f.write(f'Результат получен с шагом по координате h = {h:.4f}\n')
    # f.write(f'Результат получен с шагом по времени tau = {tau:.4f}\n')
    f.write(f'Температурное поле в момент времени t = {t_end:.4f}\n')

# with open('tempr.txt', 'w') as g:
#     for i, temp in enumerate(temperature_field):
#         g.write(f'{i * h:.8f} {(temp - 273):.5f}\n')
t = np.arange(0.0, 1800.0, 1.8)
plt.plot(t, temperature_field)
print(len(temperature_field))
```

Графики

36600 - это критическое значение энергии



Графики



Заключительная часть

Результаты

Для реализации проекты был выпран язык программирования julia и python, структурируем данные в виде набора модулей, каждый из которых реализует определенные части алгоритмов. Входные данные представим в виде файлов с необходимыми данными, вывод - файлы формата CSV и/или графиков

Источники

- Моделирование физических процессов и явлений на ПК / Д. А. Медведев, А. Л. Куперштох, Э. Р. Прууэл [и др.]. – Новосибирск : Новосиб. гос. ун-т, 2010. – 101 с. – ISBN 978-5-94356-933-3.