Министерство образования и науки Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Санкт-Петербургский государственный архитектурно-строительный университет»**

|  |  |
| --- | --- |
| Факультет: | Инженерной экологии и городского хозяйства |
| Кафедра: | Информационных систем и  технологий |

|  |
| --- |
| **ОТЧЕТ** |

**Естественная и загрязненная атмосфера. Уравнение атмосферной диффузии (УАД)**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | Мельниченко Дмитрий Сергеевич | | |
| Направление подготовки | | 01.03.02 – Прикладная математика и информатика | |
|  | |  | |
| Группа: | ПМИб-4 | | |
|  | |  | |
| Руководитель СПбГАСУ:    Мовсесова Л.В. | | |  |
|  | | |  |

Санкт-Петербург

2023 г

**Аналитическое решение**

import math  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.ticker import MaxNLocator  
from matplotlib import rc  
  
Stranght = 1.4  
high = 100.0  
  
coef\_turb\_z = 5.0  
k0 = 0.5  
u = 5.0  
coef\_turb\_y = k0 \* u  
  
x, y = np.mgrid[0.1:10:100j, -5:5:100j]  
z = 100  
  
result = (Stranght / (4 \* math.pi \* x \* math.sqrt(coef\_turb\_y \* coef\_turb\_z))  
 \* np.exp(-u \* y \*\* 2 / (4 \* coef\_turb\_y \* x))  
 \* (np.exp(-u \* (z + high) \*\* 2 / (4 \* coef\_turb\_z \* x)) + np.exp(-u \* (z - high) \*\* 2 / (4 \* coef\_turb\_z \* x)))  
 )  
  
# расчет для y = 0; вертикальная плоскость  
x1, z1 = np.mgrid[0.1:10:100j, 90:110:100j]  
y1 = 0.  
result\_y0 = (Stranght / (4 \* math.pi \* x1 \* math.sqrt(coef\_turb\_y \* coef\_turb\_z))  
 \* np.exp(-u \* y1 \*\* 2 / (4 \* coef\_turb\_y \* x1))  
 \* (np.exp(-u \* (z1 + high) \*\* 2 / (4 \* coef\_turb\_z \* x1)) + np.exp(-u \* (z1 - high) \*\* 2 / (4 \* coef\_turb\_z \* x1)))  
 )  
  
# определение линий уровня и цветов  
Line\_highs= [0.002, 0.004, 0.006, 0.008, 0.010, 0.012, 0.016, 0.020, 0.030, 0.040, 0.300]  
Color\_lines= ['cyan', 'turquoise', 'teal', 'steelblue', 'cornflowerblue', 'mediumslateblue',  
 'indigo', 'darkmagenta', 'orchid', 'lightpink', 'hotpink']  
  
# выводим два графика рядом  
  
plt.contourf(x, y, result, levels = Line\_highs, colors=Color\_lines)  
plt.xlabel('$x$, м')  
plt.ylabel('$y$, м')  
plt.title('$а$) $z$ = 100 м', loc='left')  
plt.show()  
  
  
plt.contourf(x1, z1, result\_y0, levels = Line\_highs, colors=Color\_lines)  
plt.xlabel('$x$, м')  
plt.ylabel('$z$, м')  
plt.title('$б$) $y$ = 0', loc='left')  
  
plt.show()

Результаты

Изображение выглядит как снимок экрана, текст, Красочность, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

**Решение несколькими методами:**

**Изображение выглядит как текст, снимок экрана, дисплей, число

Автоматически созданное описание**

**Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание**

Код программы

import math  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.ticker import MaxNLocator  
from matplotlib import rc  
  
  
# источник мощностью M на высоте H  
Stranght = 1.4  
high = 100.0  
  
# рассматриваем задачу с постоянными коэффициентами  
coef\_turb\_z = 5.0 #Турбулентность по z (поток)  
k0 = 0.5  
u = 5.0  
coef\_turb\_y = k0 \* u #Турбулентность по х (поток)  
y = 0.  
  
uH = u  
# Сетка значений  
z\_max = 300.  
x\_max = 500.  
  
step\_x = 0.25  
step\_z = 1  
  
x = np.arange(0.001, x\_max + step\_x, step\_x)  
z = np.arange(0, z\_max + step\_z, step\_z)  
  
n = len(z) - 1  
m = len(x) - 1  
  
  
Wind\_speed = np.ones(n + 1)  
coef\_turb = np.ones(n + 1)  
  
Wind\_speed = Wind\_speed \* u #скорость ветра - в примере константа  
coef\_turb = coef\_turb \* coef\_turb\_z #коэф турбулентности - в примере константа  
  
# прогонка  
def progon(A, C, B, F, Y):  
 alfa = np.zeros(n + 1)  
 beta = np.zeros(n + 1)  
 # прямой ход метода прогонки  
 # вычисление коэффициентов прогонки  
 alfa[0] = B[0] / C[0]  
 beta[0] = F[0] / C[0]  
 for i in range(1, n + 1):  
 alfa[i] = B[i] / (C[i] - alfa[i - 1] \* A[i])  
 beta[i] = (A[i] \* beta[i - 1] + F[i]) / (C[i] - alfa[i - 1] \* A[i])  
 # обратный ход метода прогонки  
 Y[i] = beta[n]  
 for i in range(n - 1, -1, -1):  
 Y[i] = alfa[i] \* Y[i+1] + beta[i]  
  
#Массивчики для прогонки  
A = np.zeros(n+1)  
C = np.zeros(n+1)  
B = np.zeros(n+1)  
F = np.zeros(n+1)  
  
#Массивчики для результатов  
results = np.zeros((n + 1, m + 1))  
results\_y0 = np.zeros((n + 1, m + 1))  
result\_not\_explicit = np.zeros((n + 1, m + 1))  
result\_Krank\_Nikolson = np.zeros((n + 1, m + 1))  
result\_analyt = np.zeros((n + 1, m + 1))  
  
index\_first\_high =0  
for index in range(0,len(z)):  
 if z[index] == 100:  
 index\_first\_high = index  
  
# Начальные условия для некоторых методов  
results[index\_first\_high, 0] = Stranght / (uH \* step\_z)  
result\_not\_explicit[index\_first\_high, 0] = Stranght / (uH \* step\_z)  
result\_Krank\_Nikolson[index\_first\_high, 0] = Stranght / (uH \* step\_z)  
  
  
print("Start excplicit result")  
for j in range(m):  
 for i in range(1, n-1):  
 sigma = step\_x / (Wind\_speed[i] \* step\_z \*\* 2)  
 k\_minus = 0.5 \* (coef\_turb[i] + coef\_turb[i - 1])  
 k\_plus = 0.5 \* (coef\_turb[i] + coef\_turb[i + 1])  
 #c[i, j+1] = (sigma \* k\_plus \* c[i+1, j] + (1- sigma \* (k\_minus + k\_plus)) \* c[i, j]  
 # + sigma \* k\_minus \* c[i-1, j])  
 results[i, j + 1] = (sigma \* (k\_plus \* results[i + 1, j] + k\_minus \* results[i - 1, j]) +  
 (1 - sigma \* (k\_minus + k\_plus)) \* results[i, j])  
  
 results[0, j + 1] = results[1, j + 1]  
 results[n, j] = 0  
for j in range(1, m):  
 for i in range(n):  
 results\_y0[i,j] = results[i,j] / (2 \* math.sqrt(math.pi \* k0 \* x[j]))  
  
print("end Result get: ",results\_y0)  
  
print("Start not - excplicit result")  
for j in range(1, m+1):  
 # вычисление коэффциентов системы уравнений  
 C[0] = 1.; B[0] = 1.; F[0] = 0.  
 C[n] = 1.; A[n] = 0.; F[n] = 0.  
 for i in range(1,n):  
 sigma = Wind\_speed[i] \* step\_z \*\* 2 / step\_x  
 k\_minus = 0.5 \* (coef\_turb[i] + coef\_turb[i - 1])  
 k\_plus = 0.5 \* (coef\_turb[i] + coef\_turb[i + 1])  
 A[i] = k\_minus  
 C[i] = k\_plus + k\_minus + sigma  
 B[i] = k\_plus  
 F[i] = sigma \* result\_not\_explicit[i, j - 1] #\*0.983  
 progon(A, C, B, F, result\_not\_explicit[:, j])  
  
print("end Result get: ",result\_not\_explicit)  
  
print("Start Krank-Nikolson result")  
for j in range(1, m+1):  
 # вычисление коэффциентов системы уравнений  
 C[0] = 1.; B[0] = 1.; F[0] = 0.  
 C[n] = 1.; A[n] = 0.; F[n] = 0.  
 for i in range(1,n):  
 sigma = Wind\_speed[i] \* step\_z \*\* 2 / step\_x  
 k\_minus = 0.5 \* (coef\_turb[i] + coef\_turb[i - 1])  
 k\_plus = 0.5 \* (coef\_turb[i] + coef\_turb[i + 1])  
 A[i] = k\_minus  
 C[i] = k\_plus + k\_minus + 2 \* sigma  
 B[i] = k\_plus  
 F[i] = (2 \* sigma \* result\_Krank\_Nikolson[i, j - 1] + k\_plus \* (result\_Krank\_Nikolson[i + 1, j - 1] - result\_Krank\_Nikolson[i, j - 1])  
 - k\_minus \* (result\_Krank\_Nikolson[i, j - 1] - result\_Krank\_Nikolson[i - 1, j - 1]))  
 progon(A, C, B, F, result\_Krank\_Nikolson[:, j])  
  
print("end Result get: ", result\_Krank\_Nikolson)  
  
# сетка для графика  
x\_array\_grid, z\_array\_grid = np.meshgrid(x, z)  
  
# определение линий уровня и цветов  
Line\_highs = [0.002, 0.004, 0.006, 0.008, 0.010, 0.012, 0.016, 0.020, 0.030, 0.040, 0.300]  
Color\_lines = ['cyan', 'turquoise', 'teal', 'steelblue', 'cornflowerblue', 'mediumslateblue',  
 'indigo', 'darkmagenta', 'orchid', 'lightpink', 'hotpink']  
  
temp = np.zeros(m+1)  
temp1 = np.zeros(m+1)  
temp2 = np.zeros(m+1)  
temp3 = np.zeros(m+1)  
temp4 = np.zeros(m+1)  
temp5 = np.zeros(m+1)  
temp6 = np.zeros(m+1)  
index\_first\_high, = np.where(z == 90)  
  
temp[:] = results[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
temp1[:] = result\_not\_explicit[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
temp2[:] = (Stranght / (4 \* math.pi \* x[:] \* math.sqrt(coef\_turb\_y \* coef\_turb\_z))  
 \* np.exp(-u \* y \*\* 2 / (4 \* coef\_turb\_y \* x[:]))  
 \* (np.exp(-u \* (z[index\_first\_high] + high) \*\* 2 / (4 \* coef\_turb\_z \* x[:])) + np.exp(-u \* (z[index\_first\_high] - high) \*\* 2 / (4 \* coef\_turb\_z \* x[:])))  
 )  
temp3[:] = result\_Krank\_Nikolson[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
  
index\_first\_high, = np.where(z == 70)  
temp4[:] = result\_Krank\_Nikolson[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
index\_first\_high, = np.where(z == 80)  
temp5[:] = result\_Krank\_Nikolson[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
  
index\_first\_high, = np.where(z == 100)  
temp6[:] = result\_Krank\_Nikolson[index\_first\_high, :] / (2 \* np.sqrt(math.pi \* k0 \* x[:])) \* np.exp(-y \*\* 2 / (4 \* k0 \* x[:]))  
  
print('явная неявная аналитическое')  
for i in range(0, 20):  
 print(temp[i], temp1[i], temp2[i], temp3[i])  
  
  
plt.plot(x, temp, c = "r", label = "явная")  
plt.plot(x, temp1, c = "g", label = "неявная")  
plt.plot(x, temp2, c = "b", label = "аналитическое")  
plt.plot(x, temp3, c = "m", label = "схема Кранка-Николсона")  
plt.plot(x, temp4, c = "c", label = "К-Н высота 70")  
plt.plot(x, temp5, c = "r", label = "К-Н высота 110")  
plt.plot(x, temp6, c = "g", label = "К-Н высота 100")  
  
plt.ylim([0.0, 0.001])  
plt.xlim([0.05, 500.])  
plt.xlabel('$x$, м')  
plt.ylabel('$z$, м')  
plt.title('Все графики', loc='left')  
plt.legend()  
plt.show()