**Лабораторная работа №4**

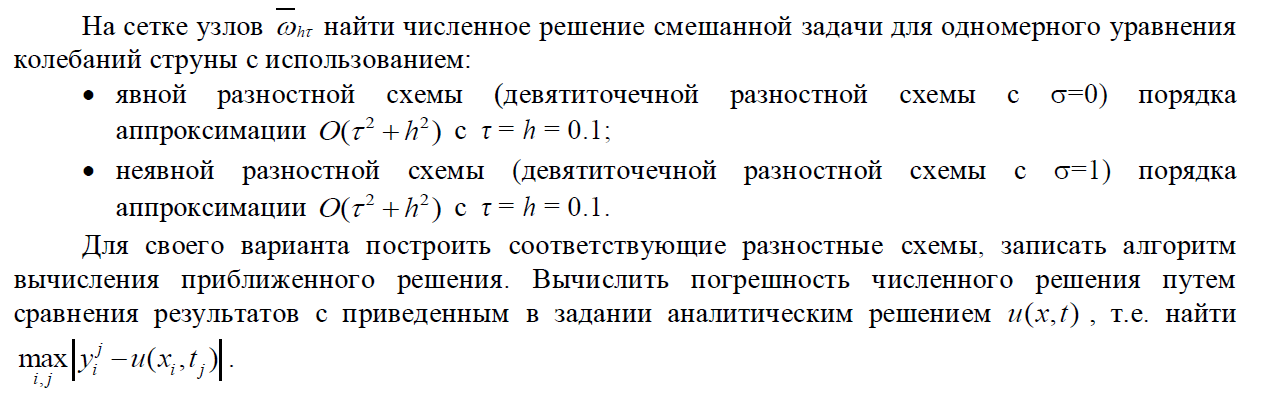
**Численное решение смешанной задачи для уравнения колебания струны**

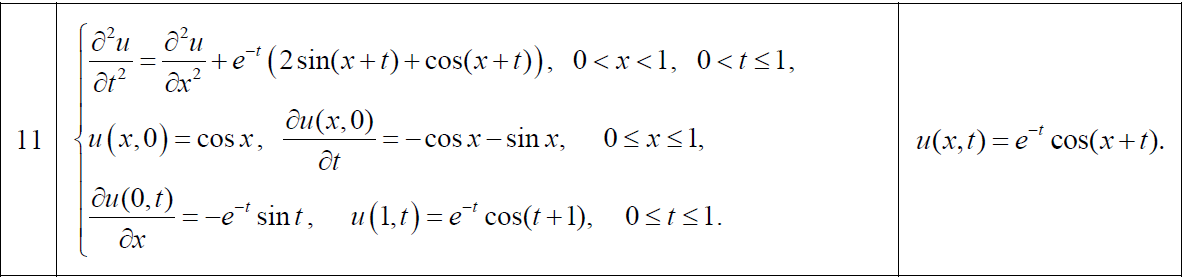
**Вариант 11**

**Чеботаревский Никита**

**3 курс, 8 группа**

**Постановка задачи**

****

****

По результатам необходимо оформить отчёт, содержащий следующую информацию:

* Титульный лист
* Постановка задачи
* Краткие теоретические сведения
* Результаты
* Выводы
* Листинг программы с комментариями

**Теоретические сведения:**

Разностная схема для исходной задачи имеет вид:

Уравнение имеет искомый порядок , а граничные и начальные условия, содержащие производную только первый. Поэтому повысим порядок.

Начальное условие:

Тогда

Граничное условие:

Тогда

Тогда получим:

В индексной форме

В результате разностная схема в индексной в форме примет вид:

**Алгоритм вычисления приближённого решения**

**1) Явная схема ():**

1) Заполняем нулевой слой:

2) Заполняем первый слой:

Из второго условия выразим:

3) Заполняем j+1 слой, где

Из третьего условия выразим :

Из исходного уравнения, учитывая что , выразим

**2) Явная схема ():**

1) Заполняем нулевой слой:

2) Заполняем первый слой:

3) Получаем следующую СЛАУ:

Данную систему будем решать методом прогонки, выпишем коэффициенты:

**Метод прогонки**

Полученную трехдиагональную матрицу будем решать методом прогонки по формулам:

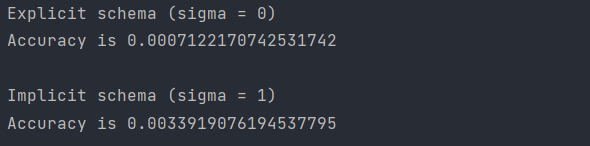
Проверка условий устойчивости метода прогонки:

Все неравенства выполняются строго, поэтому метод прогонки будет устойчив.

**Листинг программы**

*import* numpy *as* np  
  
  
*class* Solver:  
 *def* \_\_init\_\_(*self*, a=0, b=1, h=0.1, t=0.1):  
 *self*.a = a  
 *self*.b = b  
 *self*.h = h  
 *self*.t = t  
  
 *def* create\_grid(*self*):  
 n = int(1 / *self*.h)  
 xi, ti = [], []  
  
 *for* i *in* range(n + 1):  
 xi.append(np.around(*self*.a + i \* *self*.h, 3))  
 ti.append(np.around(*self*.a + i \* *self*.t, 3))  
  
 *return* xi, ti  
  
 @staticmethod  
 *def* u(xi, ti):  
 *return* np.exp(-ti) \* np.cos(xi + ti)  
  
 @staticmethod  
 *def* f(xi, ti):  
 *return* np.exp(-ti) \* (2 \* np.sin(xi + ti) + np.cos(xi + ti))  
  
 @staticmethod  
 *def* u0(xi):  
 *return* np.cos(xi)  
  
 *def* u\_tilde(*self*, xi):  
 *return* -np.cos(xi) - np.sin(xi) + *self*.t / 2 \* (-np.cos(xi) + *self*.f(xi, 0))  
  
 @staticmethod  
 *def* mu\_1(ti):  
 *return* np.exp(-ti) \* np.cos(ti + 1)  
  
 *def* mu\_0(*self*, i, values):  
 xs, ts = *self*.create\_grid()  
 *return* (np.exp(-ts[i]) \* np.sin(ts[i]) + *self*.h / 2 \* *self*.f(0, ts[i]) - *self*.h / (2 \* *self*.t \*\* 2) \*  
 (values[i - 1][0] - 2 \* values[i][0]) + (values[i][1] - values[i][0]) / *self*.h) / \  
 (*self*.h / (2 \* *self*.t \*\* 2))  
  
 *def* solution(*self*):  
 xi, ti = *self*.create\_grid()  
 result = [[0] \* len(xi) *for* \_ *in* range(len(ti))]  
  
 *for* i *in* range(len(ti)):  
 *for* j *in* range(len(xi)):  
 result[i][j] = *self*.u(xi[j], ti[i])  
  
 *return* result  
  
 *def* explicit\_schema(*self*):  
 xi, ti = *self*.create\_grid()  
 y = [[0] \* len(xi) *for* \_ *in* range(len(ti))]  
  
 *for* i *in* range(len(xi)):  
 y[0][i] = *self*.u0(xi[i])  
  
 y[1][-1] = *self*.mu\_1(*self*.t)  
 *for* i *in* range(len(xi) - 1):  
 y[1][i] = y[0][i] + *self*.t \* *self*.u\_tilde(xi[i])  
  
 *for* j *in* range(1, len(ti) - 1):  
 y[j + 1][0] = *self*.mu\_0(j, y)  
 y[j + 1][-1] = *self*.mu\_1(ti[j + 1])  
 *for* i *in* range(1, len(xi) - 1):  
 y[j + 1][i] = 2 \* y[j][i] - y[j - 1][i] + *self*.t \*\* 2 \* \  
 ((y[j][i - 1] - 2 \* y[j][i] + y[j][i + 1]) / *self*.h \*\* 2 + *self*.f(xi[i], ti[j]))  
  
 *return* y  
  
 *def* get\_coeff(*self*, y, index, size, sigma):  
 xi, ti = *self*.create\_grid()  
 a, b, c, fs = [], [], [], []  
  
 *for* i *in* range(size + 1):  
 *if* i == 0:  
 b.append(0)  
 c.append(1)  
 fs.append(*self*.mu\_0(index, y))  
 *elif* i == size:  
 c.append(1)  
 a.append(0)  
 fs.append(*self*.mu\_1(ti[index + 1]))  
 *else*:  
 a.append(sigma / *self*.h \*\* 2)  
 b.append(sigma / *self*.h \*\* 2)  
 c.append((2 \* sigma) / *self*.h \*\* 2 + 1 / *self*.t \*\* 2)  
 fs.append(*self*.f(xi[i], ti[index]) + (sigma / *self*.h \*\* 2) \*  
 (y[index - 1][i - 1] - 2 \* y[index - 1][i] + y[index - 1][i + 1]) -  
 (y[index - 1][i] - 2 \* y[index][i]) / *self*.t \*\* 2 - (2 \* sigma - 1) \*  
 (y[index][i - 1] - 2 \* y[index][i] + y[index][i + 1]) / *self*.h \*\* 2)  
  
 *return* a, c, b, fs  
  
 *def* sweep\_method(*self*, y, index, size, sigma):  
 a, c, b, f = *self*.get\_coeff(y, index, size, sigma)  
 xi, ti = *self*.create\_grid()  
  
 alpha = [b[0] / c[0]]  
 betta = [f[0] / c[0]]  
 y\_res = [0] \* len(xi)  
  
 *for* i *in* range(1, len(xi) - 1):  
 alpha.append(b[i] / (c[i] - a[i - 1] \* alpha[i - 1]))  
  
 *for* i *in* range(1, len(xi)):  
 betta.append((f[i] + a[i - 1] \* betta[i - 1]) / (c[i] - a[i - 1] \* alpha[i - 1]))  
  
 y\_res[len(xi) - 1] = betta[-1]  
 *for* i *in* range(len(xi) - 2, -1, -1):  
 y\_res[i] = alpha[i] \* y\_res[i + 1] + betta[i]  
  
 *return* y\_res  
  
 *def* implicit\_schema(*self*, sigma=1):  
 xi, ti = *self*.create\_grid()  
 y = [[0] \* len(xi) *for* \_ *in* range(len(ti))]  
  
 *for* i *in* range(len(xi)):  
 y[0][i] = *self*.u0(xi[i])  
  
 y[1][-1] = *self*.mu\_1(*self*.t)  
 *for* i *in* range(len(xi) - 1):  
 y[1][i] = y[0][i] + *self*.t \* *self*.u\_tilde(xi[i])  
  
 *for* j *in* range(1, len(ti) - 1):  
 y[j + 1][-1] = *self*.mu\_1(ti[j + 1])  
 tmp = *self*.sweep\_method(y, j, len(xi) - 1, sigma)  
  
 *for* k *in* range(len(xi) - 1):  
 y[j + 1][k] = tmp[k]  
  
 *return* y  
  
 @staticmethod  
 *def* count\_accuracy(u, y):  
 result = -np.inf  
  
 *for* i *in* range(len(u)):  
 *for* j *in* range(len(u[0])):  
 *if* result < np.abs(u[i][j] - y[i][j]):  
 result = np.abs(u[i][j] - y[i][j])  
  
 *return* result  
  
  
sol1 = Solver()  
y1 = sol1.explicit\_schema()  
u1 = sol1.solution()  
  
print("Explicit schema (sigma = 0)")  
print(f"Accuracy is {sol1.count\_accuracy(u1, y1)}", end="\n" \* 2)  
  
y2 = sol1.implicit\_schema()  
  
print("Implicit schema (sigma = 1)")  
print(f"Accuracy is {sol1.count\_accuracy(u1, y2)}")

**Результаты программы:**

****