**Лабораторная работа №1**

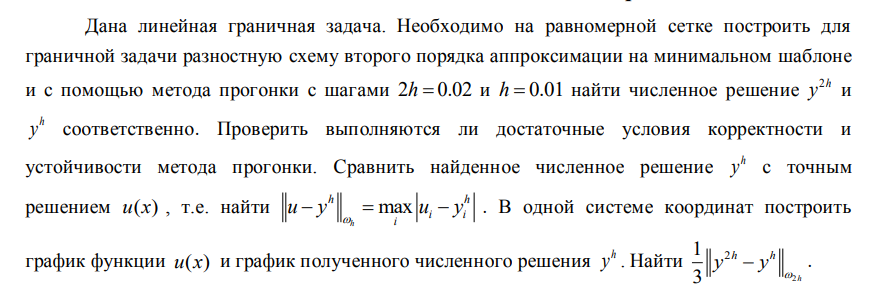
**Метод сеток решения граничной задачи для ОДУ**

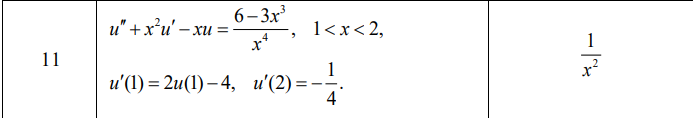
**Вариант 11**

**Чеботаревский Никита**

**3 курс, 8 группа**

**Постановка задачи**





**Построение разностной схемы**

Для исходной задачи необходимо построить разностную схему второго порядка аппроксимации вида на минимальном шаблоне:

Тогда аппроксимируем функции советующими разностными производными:

Рассмотрим погрешность аппроксимации дифференциального уравнения разностным оператором:

Теперь рассмотрим погрешность аппроксимации дополнительного условия разностным оператором на левой границе:

Для того, чтобы иметь 2-ой порядок аппроксимации для дополнительного условия, следует использовать следующий значения :

Аналогично рассмотрим погрешность аппроксимации дополнительного условия разностным оператором на левой границе:

Откуда соответственно получаем:

Где q(x) = x, p(x) = .

Теперь перепишем задачу в индексном виде:

Сгруппируем слагаемые и получим

Таким образом систему можно переписать в виде:

Теперь, учитывая значения p(x) и q(x), получим следующие значения коэффициентов для решения системы.

# **Метод прогонки**

Полученную трехдиагональную матрицу будем решать методом прогонки по формулам:

**Листинг программы**

*import* matplotlib.pyplot *as* plt  
  
  
*class* Solver:  
 *def* \_\_init\_\_(*self*, a, b, h):  
 *self*.a = a  
 *self*.b = b  
 *self*.h = h  
  
 *def* create\_grid(*self*):  
 n = int((*self*.b - *self*.a) / *self*.h)  
 *return* [*self*.a + i \* *self*.h *for* i *in* range(n + 1)]  
  
 *def* ai(*self*, x):  
 *return* 1 / *self*.h \*\* 2 - x \*\* 2 / (2 \* *self*.h)  
  
 *def* c0(*self*):  
 *return* 1 / *self*.h + 2 + *self*.h / 2 \* *self*.a - *self*.h \* *self*.a \*\* 2  
  
 *def* ci(*self*, x):  
 *return* 2 / *self*.h \*\* 2 + x  
  
 *def* cn(*self*):  
 *return* 1 / *self*.h + *self*.h / 2 \* *self*.b  
  
 *def* an\_b0(*self*):  
 *return* 1 / *self*.h  
  
 *def* bi(*self*, x):  
 *return* 1 / *self*.h \*\* 2 + x \*\* 2 / (2 \* *self*.h)  
  
 *def* f0(*self*):  
 *return* 4 + *self*.h / 2 \* *self*.fi(*self*.a) - (*self*.h / 2) \* 4 \* *self*.a \*\* 2  
  
 *def* fn(*self*):  
 *return* -1 / 4 + *self*.h / 2 \* *self*.fi(*self*.b) - (*self*.h / 8) \* *self*.b \*\* 2  
  
 @staticmethod  
 *def* fi(x):  
 *return* (3 \* x \*\* 3 - 6) / x \*\* 4  
  
 @staticmethod  
 *def* u(x):  
 *return* 1 / x \*\* 2  
  
 *def* count\_coefficients(*self*):  
 a, c, b, f = [], [], [], []  
 grid = *self*.create\_grid()  
  
 *for* i *in* range(len(grid)):  
 *if* i == 0:  
 c.append(*self*.c0())  
 b.append(*self*.an\_b0())  
 f.append(*self*.f0())  
 *elif* i == len(grid) - 1:  
 a.append(*self*.an\_b0())  
 c.append(*self*.cn())  
 f.append(*self*.fn())  
 *else*:  
 a.append(*self*.ai(grid[i]))  
 c.append(*self*.ci(grid[i]))  
 b.append(*self*.bi(grid[i]))  
 f.append(*self*.fi(grid[i]))  
  
 *return* a, c, b, f  
  
 *def* sweep\_method(*self*):  
 a, c, b, f = *self*.count\_coefficients()  
 grid = *self*.create\_grid()  
  
 alpha = [b[0] / c[0]]  
 betta = [f[0] / c[0]]  
 y = [0] \* len(grid)  
  
 *for* i *in* range(1, len(grid) - 1):  
 alpha.append(b[i] / (c[i] - a[i - 1] \* alpha[i - 1]))  
  
 *for* i *in* range(1, len(grid)):  
 betta.append((f[i] + a[i - 1] \* betta[i - 1]) / (c[i] - a[i - 1] \* alpha[i - 1]))  
  
 y[len(grid) - 1] = betta[-1]  
 *for* i *in* range(len(grid) - 2, -1, -1):  
 y[i] = alpha[i] \* y[i + 1] + betta[i]  
  
 *return* y  
  
 @staticmethod  
 *def* count\_accuracy(u, y):  
 *return* max([abs(u[i] - y[i]) *for* i *in* range(len(y))])  
  
 @staticmethod  
 *def* count\_value(y2h, yh):  
 *return* 1 / 3 \* max([abs(y2h[i] - yh[2 \* i]) *for* i *in* range(len(y2h))])  
  
  
*def* draw\_plots(x, u, y):  
 plt.plot(x[:2], y[:2])  
 plt.plot(x[:2], u[:2])  
 plt.xlim(1.0, 1.0005)  
 plt.ylim(0.999, 1.0)  
 plt.show()  
  
  
sol1 = Solver(1, 2, 0.01)  
  
y1 = sol1.sweep\_method()  
print(y1)  
  
u1 = [sol1.u(x) *for* x *in* sol1.create\_grid()]  
print(u1)  
  
print(f'Accuracy is {sol1.count\_accuracy(u1, y1)}', end='\n' \* 2)  
draw\_plots(sol1.create\_grid(), u1, y1)  
  
sol2 = Solver(1, 2, 0.02)  
  
y2 = sol2.sweep\_method()  
print(y2)  
  
u2 = [sol2.u(x) *for* x *in* sol2.create\_grid()]  
print(u2)  
print(f'Accuracy is {sol2.count\_accuracy(u2, y2)}', end='\n' \* 2)  
draw\_plots(sol2.create\_grid(), u2, y2)  
  
print(f'Accuracy between y2h and yh is {sol1.count\_value(y2, y1)}')

**Результаты**

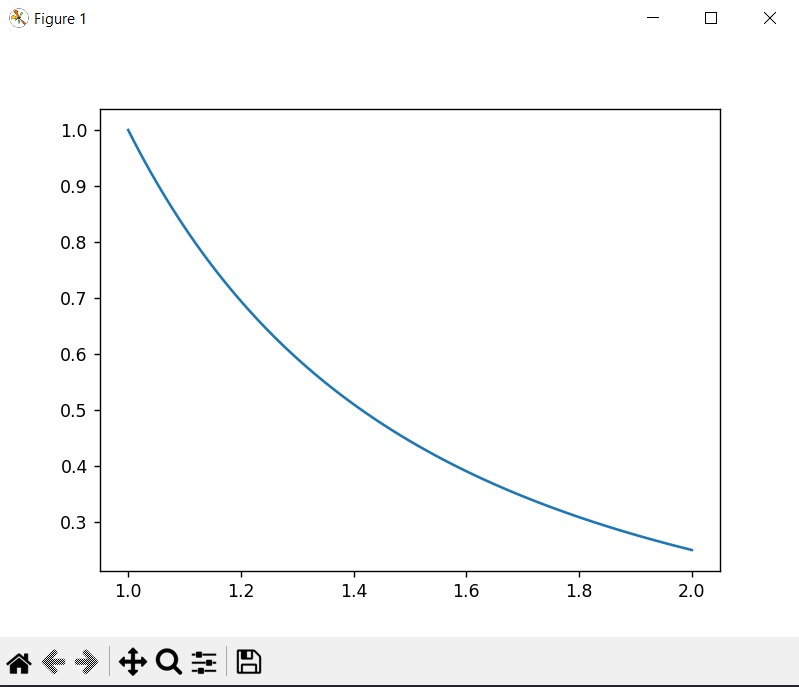
Accuracy is 8.586547046618431e-05 при h = 0.01

Accuracy is 0.0003434262008883415 при h = 0.02

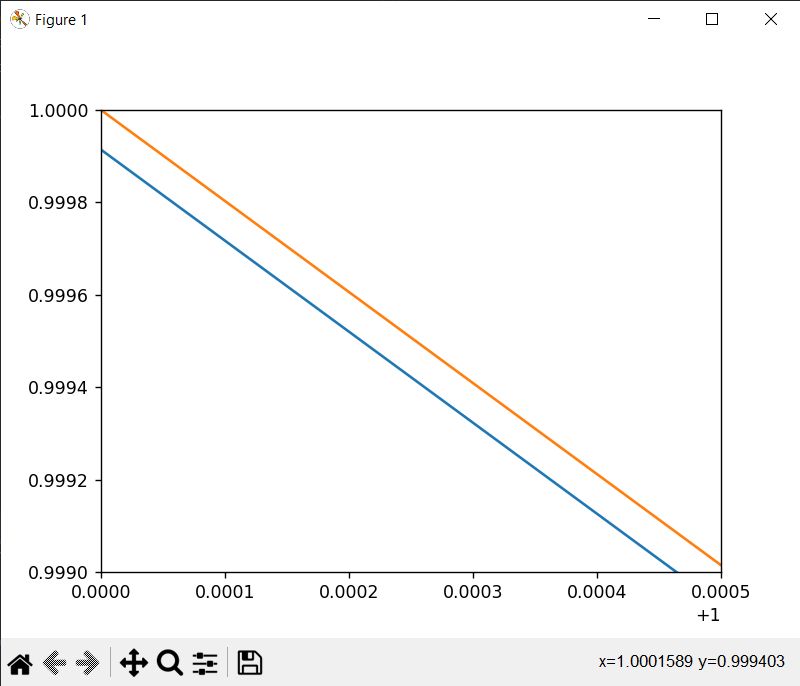
Accuracy between y2h and yh is 8.585357680738573e-05

**Графики**

График функции-решения:



При h = 0.01



При h = 0.02

