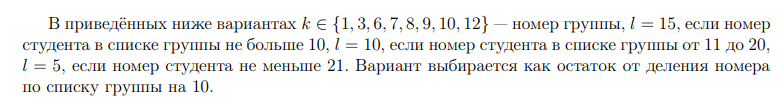
**РГР 2**

**КУЦЕНКО М.Д.**

**М8О-312Б-22**

Задание:



k = 12, l = 10

Вариант 7:



Код:

|  |
| --- |
| import sympy import numpy as np from matplotlib import pyplot as plt, gridspec  # Класс для отрисовки сеткой  class Drawer():      def \_\_init\_\_(self, max\_cols, plot\_cords, initial\_data) -> None:         """Принимает параметры и рисует приближаемую функцию.         """         self.fig = plt.figure()         self.row = 1         self.col = 1         self.max\_cols = max\_cols         self.plot\_cords = plot\_cords                  # Отрисовка начального графика         ax = self.fig.add\_subplot(self.row, self.col, (self.row-1)\*self.max\_cols + self.col)         ax.set\_title('Приближаемая функция')         ax.plot(plot\_cords, initial\_data)      def plot\_new(self, data: list) -> None:         """Добавляет график новой итерации в сетку."""         self.col += 1         if (self.col % (self.max\_cols+1) == 0):             self.col = 1             self.row += 1                  gs = gridspec.GridSpec(self.row, self.max\_cols)          # Перемещение предыдущих графиков         for i, ax in enumerate(self.fig.axes):             ax.set\_position(gs[i].get\_position(self.fig))             ax.set\_subplotspec(gs[i])          # Отрисовка нового графика         ind = (self.row-1)\*self.max\_cols + self.col-1         new\_ax = self.fig.add\_subplot(gs[ind])         new\_ax.set\_title(f'Итерация {ind}')         new\_ax.plot(self.plot\_cords, data)   # Точность приближения  precision = 0.01    # Задание границ отрезка  left\_edge = 0 right\_edge = 0.8 + 12/10  # Задание функции  t = sympy.symbols('t') # переменная для интегрирования и вычислений f = (2\*10/5-t)\*\*2 y = sympy.sin(3\*t) x = [] # содержит ортогональный базис вида t^(n-1)  def scalar(wrapper\_func) -> float:     return sympy.integrate(wrapper\_func, (t, left\_edge, right\_edge))  def norm(wrapper\_func) -> float:     return scalar(wrapper\_func\*\*2)\*\*0.5  # Разбиение отрезка на точки для вычислений  split = 1000 cords = np.linspace(left\_edge, right\_edge, split)  # Задание начального параметра и частичной суммы ряда Фурье  n = 1 fourier\_sum = 0  # Начальная отрисовка  plt.ion() # включает интерактивный режим, чтобы отрисовывать в процессе выполнения drawer = Drawer(3, cords, sympy.lambdify(t, y, 'numpy')(cords))  # Выполняем, пока не будет выполнено условие выхода  while (1):     # Выбор нового элемента базиса      new\_elem = t\*\*(n-1)          # Ортогонализация элемента предыдущим      for old\_elem in x:         new\_elem -= old\_elem \* scalar(new\_elem\*old\_elem)/scalar(old\_elem\*old\_elem)          # Добавление элемента в базис      x.append(new\_elem)          # Изменение частичной суммы ряда Фурье      fourier\_sum += new\_elem \* scalar(y\*new\_elem)/scalar(new\_elem\*new\_elem)          # Рассчёт значений частичной суммы ряда Фурье      data = sympy.lambdify(t, fourier\_sum, 'numpy')(cords)     if type(data) == float:         data = [data]\*len(cords)      # Отрисовка новой итерации      drawer.plot\_new(data)     plt.draw()  # обновляет график     plt.pause(0.5)  # создаёт задержку для обновления графика      # Проверка, что отклонение меньше чем заданная точность      if (norm(fourier\_sum - y) < precision):          break              # Условие выхода не выполнилось => новая итерация      n += 1  plt.ioff() # выключает интерактивный режим, чтобы оставить конечный результат print("\n\nРАБОТА ПРОГРАММЫ ЗАКОНЧЕНА!\n\n") plt.show()  # показывает финальный результат |

Результаты работы кода:

Для точности 0.1 потребовалось 4 итерации.

Для точности 0.01 – 6 итераций.

Для точности 0.001 – 8 итераций.

