НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Етап №8

із розрахунково-графічної роботи

із дисципліни «Алгоритми і системи комп’ютерної математики»

на тему

«Апроксимація періодичних сигналів рядами Фур’є»

|  |  |
| --- | --- |
| Виконав: | Перевірив: |
| студент групи КМ-51 | канд. техн. наук |
| Галета М.С. | Олефір О.С. |

Київ 2018

ЗМІСТ

1 ЛІСТИНГ ПРОГРАМИ 3

# 1 ЛІСТИНГ ПРОГРАМИ

import numpy as np

import sys

import os

import matplotlib.pyplot as plt

from PyQt5.QtWidgets import QWidget

from PyQt5.QtWidgets import QLabel

from PyQt5.QtWidgets import QApplication

from PyQt5.QtWidgets import QPushButton

from PyQt5.QtWidgets import QLineEdit

from PyQt5.QtWidgets import QMessageBox

from PyQt5.QtWidgets import QInputDialog

from PyQt5.QtWidgets import QFileDialog

from PyQt5.QtWidgets import QCheckBox

from PyQt5.QtGui import QIcon, QFont

from PyQt5 import QtCore, QtGui

class MainWindow(QWidget):

def \_\_init\_\_(self, parent=None):

super().\_\_init\_\_(parent)

self.initUI()

self.save\_results = False

self.route = None

def initUI(self):

reg\_int = QtCore.QRegExp("[0-9]{1,8}")

validator\_int = QtGui.QRegExpValidator(reg\_int)

reg = QtCore.QRegExp("^[-]?[0-9]{1,4}(\.[0-9]{1,5})?")

validator = QtGui.QRegExpValidator(reg)

self.t\_0\_label = QLabel(self)

self.t\_0\_label.move(10, 12)

self.t\_0\_label.setFont(QFont("Arial", 11))

self.t\_0\_label.setText("Введіть початок інтервалу дискретизації "+

"t\_0: ")

self.t\_0 = QLineEdit(self)

self.t\_0.move(337, 10)

self.t\_0.resize(80, 20)

self.t\_0.setValidator(validator)

self.t\_n\_label = QLabel(self)

self.t\_n\_label.move(10, 40)

self.t\_n\_label.setFont(QFont("Arial", 11))

self.t\_n\_label.setText("Введіть кінець інтервалу дискретизації "+

"t\_n: ")

self.t\_n = QLineEdit(self)

self.t\_n.move(337, 38)

self.t\_n.resize(80, 20)

self.t\_n.setValidator(validator)

self.N\_label = QLabel(self)

self.N\_label.move(10, 68)

self.N\_label.setFont(QFont("Arial", 11))

self.N\_label.setText("Введіть кількість розбиттів N: ")

self.N = QLineEdit(self)

self.N.move(337, 66)

self.N.resize(80, 20)

self.N.setValidator(validator\_int)

self.K\_label = QLabel(self)

self.K\_label.move(10, 96)

self.K\_label.setFont(QFont("Arial", 11))

self.K\_label.setText("Введіть кількість гармонік K: ")

self.K = QLineEdit(self)

self.K.move(337, 94)

self.K.resize(80, 20)

self.K.setValidator(validator\_int)

self.cb = QCheckBox('Зберегти результати у файли', self)

self.cb.move(10, 135)

self.cb.setFont(QFont("Arial", 11))

self.cb.stateChanged.connect(self.save)

but1 = QPushButton("Ввести значення\nв точках розбиття", self)

but1.move(150, 180)

but1.resize(130, 40)

but1.clicked.connect(self.manual\_values)

but2 = QPushButton("Згенерувати випадкові\nзначення", self)

but2.move(300, 180)

but2.resize(130, 40)

but2.clicked.connect(self.random\_values)

self.text\_label = QLabel(self)

self.text\_label.move(450, 20)

self.text\_label.setFont(QFont("Arial", 11))

self.text\_label.setText("Довідка:")

self.text\_label = QLabel(self)

self.text\_label.move(450, 50)

self.text\_label.setFont(QFont("Arial", 11))

self.text\_label.setText("1) Період T = t\_n - t\_0")

self.text\_label = QLabel(self)

self.text\_label.move(450, 80)

self.text\_label.setFont(QFont("Arial", 11))

self.text\_label.setText("2) Для більш точного результату")

self.text\_label = QLabel(self)

self.text\_label.move(466, 100)

self.text\_label.setFont(QFont("Arial", 11))

self.text\_label.setText("варто обрати N = 2K+1")

self.setGeometry(150, 150, 680, 250)

self.setWindowTitle("Апроксимація періодичного сигналу рядами Фур\'є")

self.setWindowIcon(QIcon('icon.jpg'))

self.show()

def save(self, state):

if state == QtCore.Qt.Checked:

self.save\_results = True

if self.route is None:

self.route = str(QFileDialog.getExistingDirectory(self,

"Виберіть робочу директорію для збереження",

"/", QFileDialog.ShowDirsOnly))

if self.route != "":

self.route = self.route+"/"

else:

self.save\_results = False

def manual\_values(self):

if (not self.isEmpty() or not self.t\_0\_less\_than\_t\_n() or

not self.N\_more\_than\_2K\_1() or not self.positive\_N() or

not self.positive\_K()):

QMessageBox.warning(self, 'Warning',

'1) Всі поля мають бути заповнені\n'+

'2) Значення t\_0 має бути меншим, ніж значення t\_n\n'+

'3) Значення N має бути більшим, ніж 2\n'+

'4) Значення N має бути не меншим, ніж 2K+1\n'+

'5) Значення K має бути більшим, ніж 0')

else:

N = int(self.N.text())

t\_0 = float(self.t\_0.text())

t\_n = float(self.t\_n.text())

K = int(self.K.text())

T = t\_n - t\_0

L = T/2

t = np.arange(t\_0, t\_n+T/(2\*N), T/N)

y = []

for i in range(N):

value, ok = QInputDialog.getDouble(self, 'Input Dialog',

'Введіть значення в точці t['+str(i)+'] = '+

str(round(t[i],3)), decimals=5)

if ok:

y.append(value)

else:

break

y = np.array(y)

if y.shape[0] == N:

a, b = self.fourier\_coeffs(y, t[:N], K, L)

t\_f = np.arange(t\_0, t\_n+0.0005, 0.001)

f\_approx = self.fourier\_approximation(a, b, t\_f, K, L)

y = np.append(y, y[0])

if self.save\_results == True:

dir\_name = 'manual/'

self.save\_results\_function(dir\_name, a, b, t\_f, f\_approx)

self.plot\_res(t, y, t\_f, f\_approx)

def random\_values(self):

if (not self.isEmpty() or not self.t\_0\_less\_than\_t\_n() or

not self.N\_more\_than\_2K\_1() or not self.positive\_N() or

not self.positive\_K()):

QMessageBox.warning(self, 'Warning',

'1) Всі поля мають бути заповнені\n'+

'2) Значення t\_0 має бути меншим, ніж значення t\_n\n'+

'3) Значення N має бути більшим, ніж 2\n'+

'4) Значення N має бути не меншим, ніж 2K+1\n'+

'5) Значення K має бути більшим, ніж 0')

else:

N = int(self.N.text())

t\_0 = float(self.t\_0.text())

t\_n = float(self.t\_n.text())

K = int(self.K.text())

T = t\_n - t\_0

L = T/2

t = np.arange(t\_0, t\_n+T/(2\*N), T/N)

y = np.random.randn(N)

a, b = self.fourier\_coeffs(y, t[:N], K, L)

t\_f = np.arange(t\_0, t\_n+0.0005, 0.001)

f\_approx = self.fourier\_approximation(a, b, t\_f, K, L)

y = np.append(y, y[0])

if self.save\_results:

dir\_name = 'random/'

self.save\_results\_function(dir\_name, a, b, t\_f, f\_approx)

self.plot\_res(t, y, t\_f, f\_approx)

def save\_results\_function(self, dir\_name, a, b, t\_f, f\_approx):

i = 1

while os.path.exists(self.route+dir\_name+'model\_'+str(i)+'/') == True:

i += 1

dir\_name = self.route+dir\_name+'model\_'+str(i)+'/'

os.makedirs(os.path.dirname(dir\_name))

with open(dir\_name+'result.txt', 'w') as file:

file.write('Точки розбиття Значення функції\n\n')

for k in range(t\_f.shape[0]-1):

file.write(str(round(t\_f[k], 3))+': '+

str(f\_approx[k])+'\n')

file.write(str(round(t\_f[t\_f.shape[0]-1], 3))+

': '+str(f\_approx[t\_f.shape[0]-1]))

with open(dir\_name+'fourier\_coeffs.txt', 'w') as file:

file.write('Коефіцієнти ряду Фур\'є\n\n')

file.write('a[0] = '+str(a[0])+'\n\n')

for k in range(1, a.shape[0]-1):

file.write('a['+str(k)+'] = '+str(a[k])+'\n')

file.write('a['+str(a.shape[0]-1)+'] = '+

str(a[a.shape[0]-1])+'\n\n')

for k in range(b.shape[0]-1):

file.write('b['+str(k+1)+'] = '+str(b[k])+'\n')

file.write('b['+str(b.shape[0])+'] = '+str(b[b.shape[0]-1]))

def isEmpty(self):

if (self.N.text() == '' or self.t\_0.text() == '' or

self.t\_n.text() == '' or self.K.text() == ''):

return False

else:

return True

def t\_0\_less\_than\_t\_n(self):

if float(self.t\_0.text()) < float(self.t\_n.text()):

return True

else:

return False

def positive\_N(self):

if int(self.N.text()) >= 3:

return True

else:

return False

def positive\_K(self):

if int(self.K.text()) > 0:

return True

else:

return False

def N\_more\_than\_2K\_1(self):

if int(self.N.text()) >= 2\*int(self.K.text())+1:

return True

else:

return False

def fourier\_coeffs(self, y, t, K, L):

a = [2\*np.mean(y)]

b = []

for i in range(1, K+1):

a.append(2\*np.mean(y\*np.cos(i\*t\*np.pi/L)))

b.append(2\*np.mean(y\*np.sin(i\*t\*np.pi/L)))

return np.array(a), np.array(b)

def fourier\_approximation(self, a, b, t, K, L):

f = a[0]/2

for i in range(1, K+1):

f += a[i]\*np.cos(i\*t\*np.pi/L)

f += b[i-1]\*np.sin(i\*t\*np.pi/L)

return f

def plot\_res(self, t, y, t\_f, f\_approx):

plt.figure()

plt.plot(t\_f, f\_approx)

plt.scatter(t, y, c='r')

plt.grid(True)

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

w = MainWindow()

sys.exit(app.exec\_())