Technical University of Moldova

Homework nr.3

on Numerical Analysis

executed in Python programming language

by the student from FAF – 213 academic group

Bajenov Sevastian

Chisinau - 2022

Problem 3.1:

Solution and output:

Newton and Splines.py × Gamma function.py × Minimax and Equal Spaces.py ×

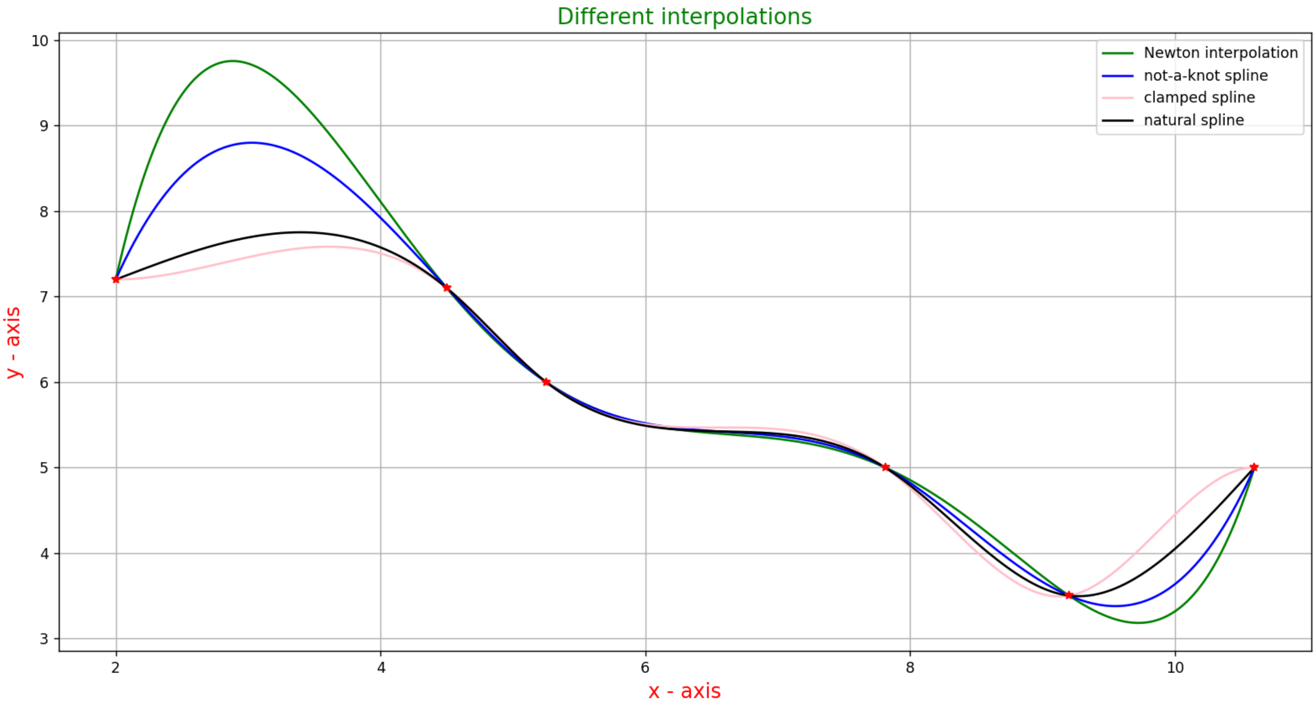
```python
# in this problem we plot two polynomial functions
# which were obtained using Newton's divided differences
# and spline interpolation methods

import matplotlib.pyplot as plot
from scipy.interpolate import CubicSpline
import numpy as np


# first we introduce Newton's divided differences
def Newton_divided_differences(x1, y1, m):
    for i1 in range(1, m):
        for j1 in range(m - i1):
            y1[j1][i1] = (y1[j1][i1 - 1] - y1[j1 + 1][i1 - 1]) / (x1[j1] - x1[i1 + j1])
    return y1


# function for printing the differences
def print_differences(y1, m):
    for i1 in range(0, m):
        for j1 in range(0, m - i1):
            print(y1[i1][j1], "\t", end=" ")
        print()


# function for the obtained interpolating polynomial of fifth degree
def Newton_polynomial(x1, a, b, c, d, e, f):
    return a * pow(x1, 5) + b * pow(x1, 4) + c * pow(x1, 3) + d * pow(x1, 2) + e * pow(x1, 1) + f
```

Newton and Splines.py × Gamma function.py × Minimax and Equal Spaces.py ×

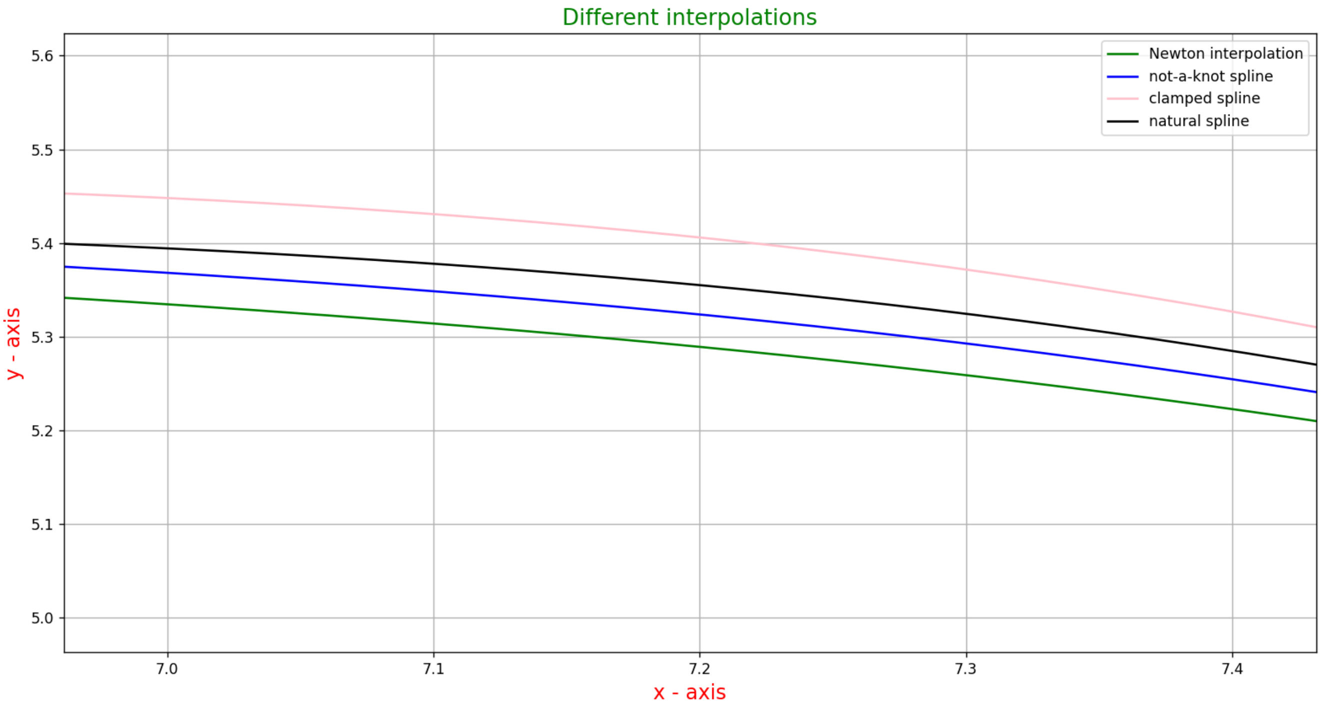
```python
# then we define the sets of points for interpolation
x = [2, 4.5, 5.25, 7.81, 9.2, 10.6]
y = [7.2, 7.1, 6.0, 5.0, 3.5, 5.0]


# and the range of the Newton's polynomial
# the number of points is six
n = 6

Newton_matrix = [list(range(0, 1 + n * (i + 1))) for i in range(n)]
for k in range(0, len(y)):
    Newton_matrix[k][0] = y[k]

table = Newton_divided_differences(x, Newton_matrix, n)


# now we calculate the data sets for plotting the Newton's polynomial
Newton_x = np.linspace(2, 10.6, 1000)
```

```python
# introducing polynomial coefficients (as the result of the previous operations)
a1 = 0.0092327
b1 = - 0.298554952
c1 = 3.68429308515
d1 = -21.442441277
e1 = 57.196254815
f1 = -46.415656367
Newton_y = list()
for i in range(0, len(Newton_x)):
    Newton_y.append(Newton_polynomial(Newton_x[i], a1, b1, c1, d1, e1, f1))


# and finally we create the corresponding cubic splines
spline = CubicSpline(x, y)
Spline_x = np.linspace(2, 10.6, 1000)
Spline_y = spline(Spline_x)

spline = CubicSpline(x, y, bc_type='clamped')
Spline_x1 = np.linspace(2, 10.6, 1000)
Spline_y1 = spline(Spline_x1)

spline = CubicSpline(x, y, bc_type='natural')
Spline_x2 = np.linspace(2, 10.6, 1000)
Spline_y2 = spline(Spline_x2)

# periodic spline is not available for our data
```

```python
# plotting the results
plot.figure()

plot.plot(Newton_x, Newton_y, color='green', label='Newton interpolation')

plot.plot(Spline_x, Spline_y, color='blue', label='not-a-knot spline')
plot.plot(Spline_x1, Spline_y1, color='pink', label='clamped spline')
plot.plot(Spline_x2, Spline_y2, color='black', label='natural spline')

plot.legend(loc='upper right')

plot.plot(x, y, 'r*')

plot.title("Different interpolations", color='green', fontsize=15)
plot.xlabel("x - axis", fontsize=14, color='red')
plot.ylabel("y - axis", fontsize=14, color='red')
plot.grid()
plot.show()

# analyzing the results we can state that the natural spline
# is the one with the shortest path
```



Different interpolations
Newton interpolation
not-a-knot spline
clamped spline
natural spline
y - axis
x - axis



Different interpolations
Newton interpolation
not-a-knot spline
clamped spline
natural spline
y - axis
x - axis

Problem 3.2:

Solution and output:

Gamma function.py   Minimax and Equal Spaces.py

```python
# in this exercise we will be working with Gamma function

import matplotlib.pyplot as plot
from scipy.special import gamma
from scipy.interpolate import CubicSpline
import numpy as np
import math

# first of all we introduce the data set
n = [1, 2, 3, 4, 5]
data = [1, 1, 2, 6, 24]
log_data = [0, 0, math.log(2, math.e), math.log(6, math.e), math.log(24, math.e)]


# the we will create Newton's interpolating polynomial
# using Newton's divided differences
def Newton_divided_differences(x1, y1, m):
    for i1 in range(1, m):
        for j1 in range(m - i1):
            y1[j1][i1] = (y1[j1][i1 - 1] - y1[j1 + 1][i1 - 1]) / (x1[j1] - x1[i1 + j1])
    return y1


# function for printing the differences
def print_differences(y1, m):
    for i1 in range(0, m):
        for j1 in range(0, m - i1):
            print(y1[i1][j1], "\t", end=" ")
        print()
```

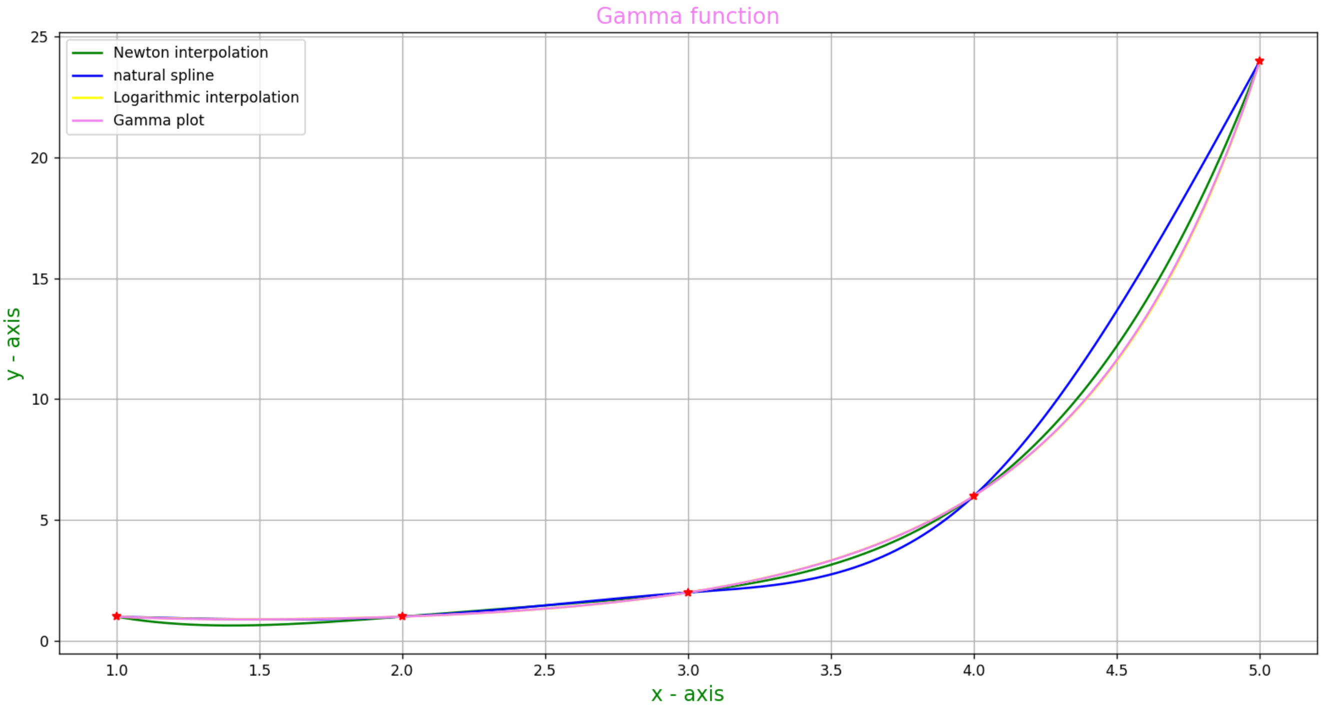
```python
# function for the obtained interpolating polynomial
def Newton_polynomial(x1, a, b, c, d, e):
    return a * pow(x1, 4) + b * pow(x1, 3) + c * pow(x1, 2) + d * pow(x1, 1) + e


Newton_matrix = [list(range(0, 1 + 5 * (i + 1))) for i in range(5)]
for k in range(0, len(data)):
    Newton_matrix[k][0] = data[k]

table = Newton_divided_differences(n, Newton_matrix, 5)


# now we calculate the data sets for plotting the Newton's polynomial
Newton_x = np.linspace(1, 5, 1000)
```

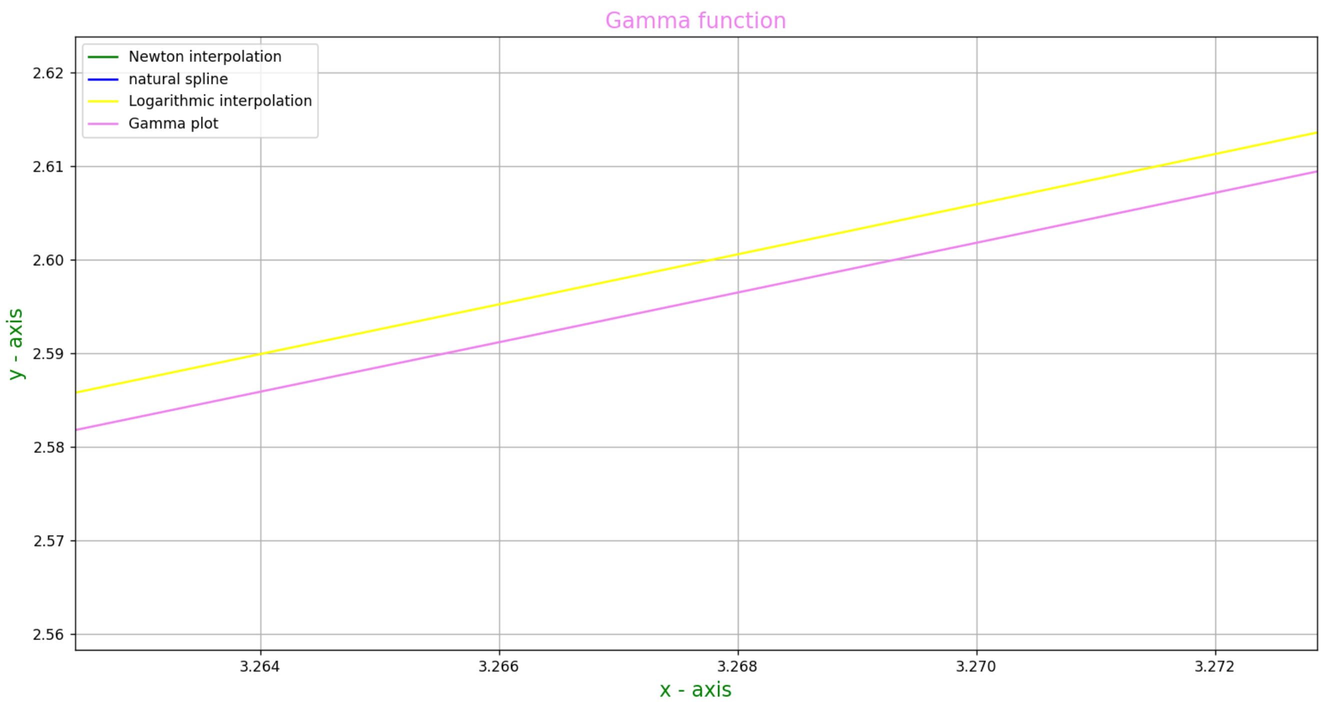
```python
# introducing polynomial coefficients (as the result of the previous operations)
a1 = 0.375
b1 = -3.417
c1 = 11.627
d1 = -16.587
e1 = 9.002
Newton_y = list()
for i in range(0, len(Newton_x)):
    Newton_y.append(Newton_polynomial(Newton_x[i], a1, b1, c1, d1, e1))


# similarly we create logarithmic polynomial
# function for the logarithmic interpolating polynomial
def Logarithmic_polynomial(x1, a, b, c, d, e):
    return pow(math.e, a * pow(x1, 4) + b * pow(x1, 3) + c * pow(x1, 2) + d * pow(x1, 1) + e)


logarithmic_matrix = [list(range(0, 1 + 5 * (i + 1))) for i in range(5)]
for k in range(0, len(data)):
    logarithmic_matrix[k][0] = log_data[k]

table = Newton_divided_differences(n, logarithmic_matrix, 5)

# now we calculate the data sets for plotting the logarithmic polynomial
log_x = np.linspace(1, 5, 1000)
```

```python
# introducing logarithmic coefficients
a1 = 0.007079126
b1 = -0.118738272
c1 = 0.882025072
d1 = -1.921094202
e1 = 1.150728276
log_y = list()
for i in range(0, len(log_x)):
    log_y.append(Logarithmic_polynomial(log_x[i], a1, b1, c1, d1, e1))


# then we create the corresponding cubic spline for our function
spline = CubicSpline(n, data, bc_type='natural')
Spline_x = np.linspace(1, 5, 1000)
Spline_y = spline(Spline_x)


# finally we create data sets for plotting gamma function itself
gamma_x = np.linspace(1, 5, 1000)
gamma_y = gamma(gamma_x)
```

```python
# plotting the results
plot.figure()

plot.plot(Newton_x, Newton_y, color='green', label='Newton interpolation')

plot.plot(Spline_x, Spline_y, color='blue', label='natural spline')

plot.plot(log_x, log_y, color='yellow', label='Logarithmic interpolation')

plot.plot(gamma_x, gamma_y, color='violet', label='Gamma plot')

plot.legend(loc='upper left')

plot.plot(n, data, 'r*')

plot.title("Gamma function", color='violet', fontsize=15)
plot.xlabel("x - axis", fontsize=14, color='green')
plot.ylabel("y - axis", fontsize=14, color='green')

plot.grid()
plot.show()
```

```python
# our final task is to compute the accuracy of all three approximations
def max_error(start, end, array, test_array):
    max = abs(array[0] - test_array[0])
    index = start + 1
    while index < end:
        if abs(array[index] - test_array[index]) > max:
            max = abs(array[index] - test_array[index])
        index += 1

    return max


max_Newton = max_error(0, len(gamma_y), gamma_y, Newton_y)
max_spline = max_error(0, len(gamma_y), gamma_y, Spline_y)
max_logarithmic = max_error(0, len(gamma_y), gamma_y, log_y)

print("Maximum error of each method:")
print(f"Newton's method: {max_Newton}")
print(f"Cubic spline: {max_spline}")
print(f"Logarithmic approximation: {max_logarithmic}")

# so finally we can conclude that using logarithmic method to
# approximate Gamma function is the most accurate way
```





Run: Gamma function

```
C:\Users\User\PycharmProjects\Laboratories\venv
Maximum error of each method:
Newton's method: 0.6788651825693393
Cubic spline: 2.2441335028010005
Logarithmic approximation: 0.06640997844430885

Process finished with exit code 0
```

Problem 3.3:

Solution and output:

Minimax and Equal Spaces.py

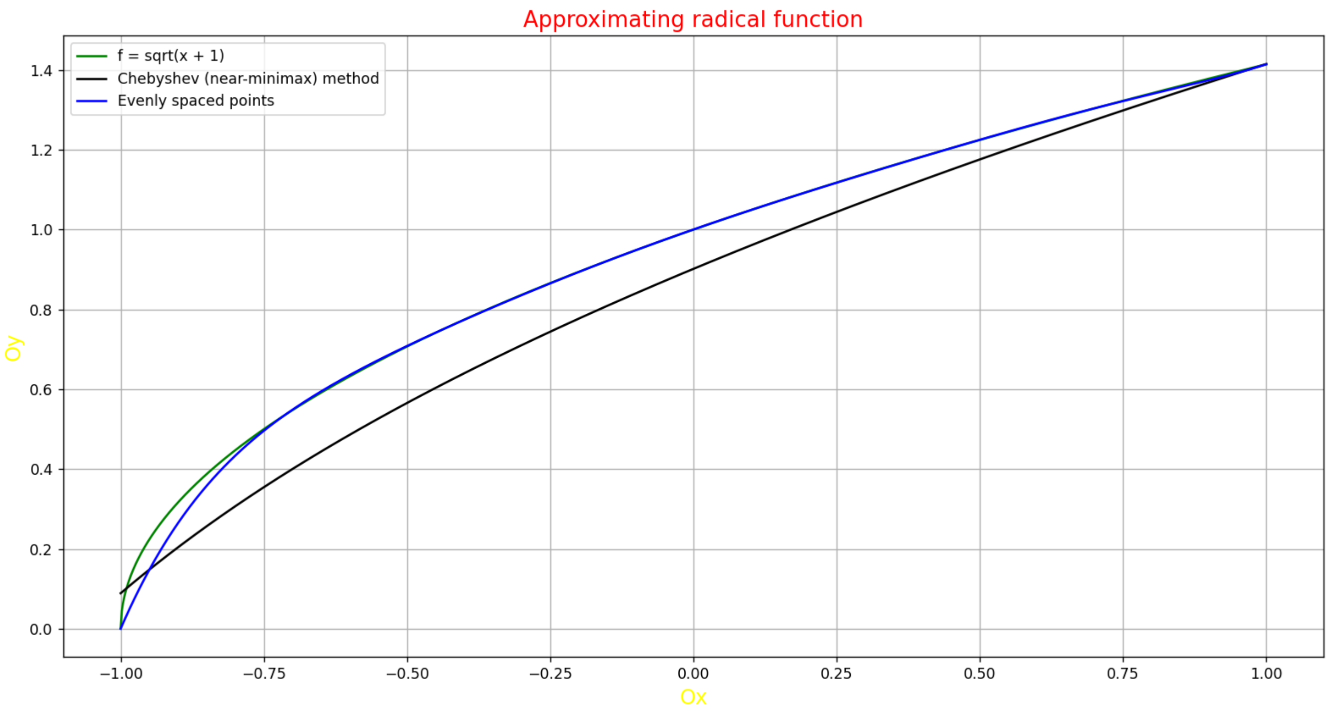
```python
# in this exercise we are going to work with the radical function
# and its approximations
import matplotlib.pyplot as plot
import numpy as np


def function(x):
    return np.sqrt(x + 1)


# first of all let us define its domain and range
x_data = np.linspace(-1, 1, 1000)
y_data = list()
for i in range(0, len(x_data)):
    y_data.append(function(x_data[i]))


# for the minimax approximation we will use Chebyshev polynomials
# the degree should be seven and the most accurate points
# will be chosen automatically
cheb_coefficients = np.polynomial.chebyshev.chebinterpolate(lambda x: np.sqrt(x + 1), 7)


# function for Chebyshev interpolating polynomial
def chebyshev_polynomial(x1, degree, coefs):
    result = 0
    for j in range(0, degree):
        result += coefs[j] * pow(x1, j)

    return result
```

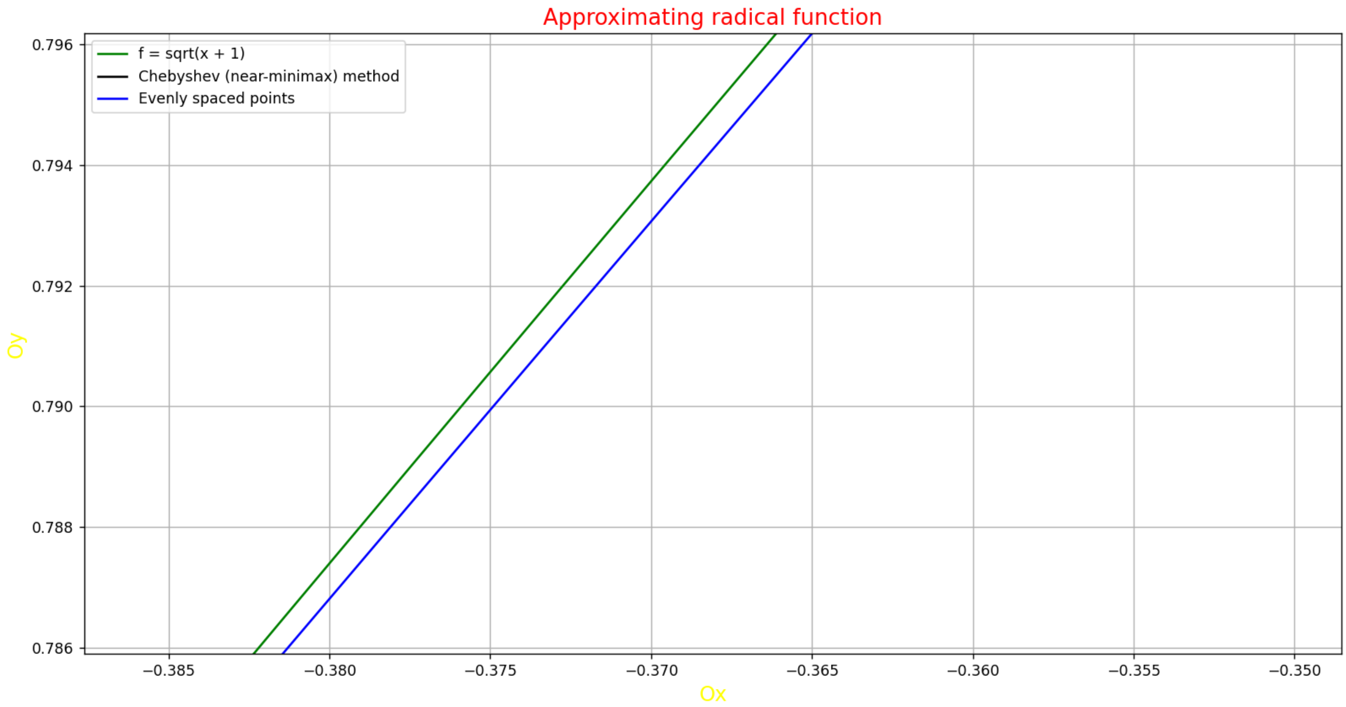
```python
cheb_data = list()
for i in range(0, len(x_data)):
    cheb_data.append(chebyshev_polynomial(x_data[i], len(cheb_coefficients), cheb_coefficients))


# then we will try to interpolate the function
# using the method of evenly spaced points
# the task requires the polynomial of degree seven
# therefore we will have to obtain eight points
# the ends of the interval plus 6 more points
even_data = np.linspace(-1, 1, 8)
even_values = list()
for i in range(0, len(even_data)):
    even_values.append(function(even_data[i]))
```

```python
def Newton_divided_differences(x1, y1, m):
    for i1 in range(1, m):
        for j1 in range(m - i1):
            y1[j1][i1] = (y1[j1][i1 - 1] - y1[j1 + 1][i1 - 1]) / (x1[j1] - x1[i1 + j1])
    return y1


# function for printing the differences
def print_differences(y1, m):
    for i1 in range(0, m):
        for j1 in range(0, m - i1):
            print(y1[i1][j1], "\t", end=" ")
        print()


# function for the obtained interpolating polynomial of seventh degree
def Newton_polynomial(x1, a, b, c, d, e, f, g, h):
    return a * pow(x1, 7) + b * pow(x1, 6) + c * pow(x1, 5) + d * pow(x1, 4) + e * pow(x1, 3) \
        + f * pow(x1, 2) + g * x1 + h
```

```python
# and the range of the Newton's polynomial
# the number of points is six
Newton_matrix = [list(range(0, 1 + 8 * (i + 1))) for i in range(8)]
for k in range(0, len(even_values)):
    Newton_matrix[k][0] = even_values[k]

table = Newton_divided_differences(even_data, Newton_matrix, 8)

Newton_x = np.linspace(-1, 1, 1000)

# introducing polynomial coefficients (as the result of the previous operations)
a1 = 0.25937
b1 = -0.27937
c1 = -0.1392
d1 = 0.1373
e1 = 0.08738
f1 = -0.15129
g1 = 0.49956
h1 = 1.00046
Newton_y = list()
for i in range(0, len(Newton_x)):
    Newton_y.append(Newton_polynomial(Newton_x[i], a1, b1, c1, d1, e1, f1, g1, h1))
```

```python
# plotting the results
plot.figure()

plot.plot(x_data, y_data, color='green', label='f = sqrt(x + 1)')

plot.plot(x_data, cheb_data, color='black', label='Chebyshev (near-minimax) method')

plot.plot(Newton_x, Newton_y, color='blue', label='Evenly spaced points')

plot.legend(loc='upper left')

plot.title("Approximating radical function", color='red', fontsize=15)
plot.xlabel("Ox", fontsize=14, color='yellow')
plot.ylabel("Oy", fontsize=14, color='yellow')

plot.grid()
plot.show()
```





Final remark: all the python files with the code will be uploaded together with this document;