MO_03	Romaniak Hubert	Informatyka niestacjonarna II rok	Semestr letni 2023/24

Zadanie 1

Metody różnic skończonych w języku Python

```python
def progresive_diff_derivative(f, x, h):
    return (f(x+h) - f(x)) / h

def regresive_diff_derivative(f, x, h):
    return (f(x) - f(x-h)) / h

def central_diff_derivative(f, x, h):
    return (f(x+h) - f(x-h)) / (2*h)

def central_diff_derivative_2(f, x, h):
    return (f(x-h) - 2*f(x) + f(x+h)) / h**2
```

Dane

$$f(x) = \cos x$$

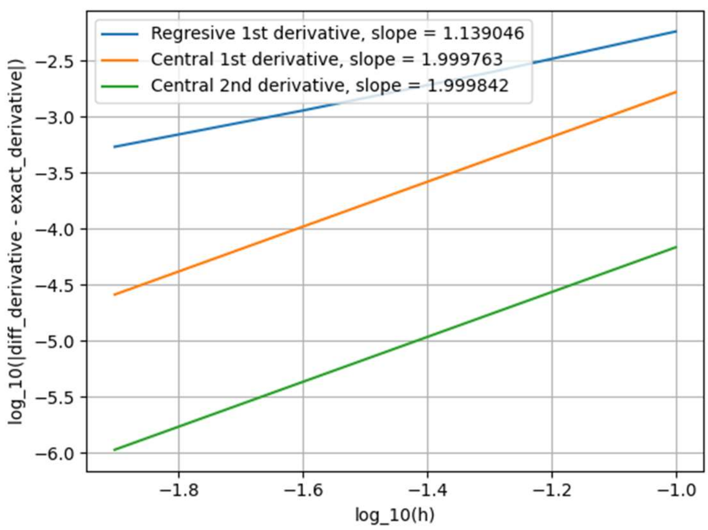
$$x = 4{,}63$$

$$h \in \{0{,}1;\ 0{,}05;\ 0{,}025;\ 0{,}0125\}$$

Rozwiązanie za pomocą różnic skończonych i błąd w stosunku do wartości dokładnej

pochodna i rodzaj różnicy	**h**	**wartość**	**błąd**
1. pochodna za pomocą różnic wstecznych	1.000000e-01	9.908364e-01	5.771545e-03
	5.000000e-02	9.941358e-01	2.472168e-03
	2.500000e-02	9.954755e-01	1.132454e-03
	1.250000e-02	9.960677e-01	5.402952e-04
1. pochodna za pomocą różnic centralnych	1.000000e-01	9.949478e-01	1.660183e-03
	5.000000e-02	9.961927e-01	4.152014e-04
	2.500000e-02	9.965041e-01	1.038101e-04
	1.250000e-02	9.965820e-01	2.595313e-05
2. pochodna za pomocą różnic centralnych	1.000000e-01	8.222725e-02	6.855698e-05
	5.000000e-02	8.227866e-02	1.714353e-05
	2.500000e-02	8.229152e-02	4.286150e-06
	1.250000e-02	8.229473e-02	1.071554e-06

Wykresy zależności $\log_{10} \mathit{błędu\ obliczenia}$ od $\log_{10} h$



Współczynnik nachylenia wykresu dla 1. pochodnej za pomocą różnic wstecznych wynosi około 1,14, a dla 1. i 2. pochodnej za pomocą różnic centralnych wynosi około 2,00.

Współczynniki te korelują z rzędami dokładności (wykładnikami przy wartości h) dla tych metod, gdzie błąd obcięcia dla 1. pochodnej za pomocą różnic wstecznych wynosi $O(h^1)$, a dla 1. i 2. pochodnej za pomocą różnic centralnych - $O(h^2)$.

Zadanie 2

Metoda pośrednia Eulera w języku Python, używając biblioteki *numpy*

```python
from numpy import sin, cos, exp, square, pi, arange, abs

def y_next_getter_getter(a, dt):
    def y_next_getter(y_previous, t):
        return (y_previous + dt * sin(pi * t)) / (1 + a * dt)
    return y_next_getter

def exact_solution_getter(a):
    def exact_solution(t):
        numerator = pi * exp(-a*t) - pi * cos(pi*t) + a * sin(pi*t)
        denominator = square(pi) + square(a)
        return numerator / denominator
    return exact_solution

def implicit_euler_getter(a, t_min, t_max, y0):
    def implicit_euler(N):
        dt = (t_max - t_min) / N
        exact_solution = exact_solution_getter(a)
        y_next_getter = y_next_getter_getter(a, dt)
        current_t_getter = lambda i: t_min + dt * i
        ts = [current_t_getter(0)]
        ys = [y0]
        errors = [abs(exact_solution(ts[0]) - ys[0])]
        for i in arange(1, N + 1):
            ts.append(current_t_getter(i))
            ys.append(y_next_getter(ys[i - 1], ts[i]))
            errors.append(abs(exact_solution(ts[i]) - ys[i]))
        return array(ts), array(ys), array(errors), dt
    return implicit_euler
```

Problem początkowy

$$\begin{cases} y'(t) = \sin \pi t - a y(t) & t \in (0,2) \\ y(0) = 0 \end{cases}$$

Dane i ilość iteracji N

$$a = 1.05$$

$$N \in \{28;\ 56;\ 112\}$$

Ścisłe rozwiązanie

$$y(t) = \frac{\pi e^{-at} - \pi \cos(\pi t) + a \sin(\pi t)}{\pi^2 + a^2}$$

$$y'(t) = \frac{-a\pi e^{-at} + a\pi \cos(\pi t) + \pi^2 \sin(\pi t)}{\pi^2 + a^2}$$

Sprawdzenie rozwiązania dla $t = 0$

$$y(0) = \frac{\pi e^{-a \cdot 0} - \pi \cos(\pi \cdot 0) + a \sin(\pi \cdot 0)}{\pi^2 + a^2} = \frac{\pi e^0 - \pi \cos 0 + a \sin 0}{\pi^2 + a^2} = \frac{\pi \cdot 1 - \pi \cdot 1 + a \cdot 0}{\pi^2 + a^2}$$

$$= \frac{\pi - \pi + 0}{\pi^2 + a^2} = \frac{0 + 0}{\pi^2 + a^2} = \frac{0}{\pi^2 + a^2} = 0$$

Sprawdzenie rozwiązania w ogólnym przypadku

$$\frac{-a\pi e^{-at} + a\pi\cos(\pi t) + \pi^2\sin(\pi t)}{\pi^2 + a^2} = \sin(\pi t) - a\frac{\pi e^{-at} - \pi\cos(\pi t) + a\sin(\pi t)}{\pi^2 + a^2} \qquad // \cdot (\pi^2 + a^2)$$

$$-a\pi e^{-at} + a\pi\cos(\pi t) + \pi^2\sin(\pi t) = \sin(\pi t)\,(\pi^2 + a^2) - a\pi e^{-at} + a\pi\cos(\pi t) - a^2\sin(\pi t) \qquad // +a\pi e^{-at}$$

$$a\pi\cos(\pi t) + \pi^2\sin(\pi t) = \pi^2\sin(\pi t) + a^2\sin(\pi t) + a\pi\cos(\pi t) - a^2\sin(\pi t) \qquad // -a\pi\cos(\pi t)$$

$$\pi^2\sin(\pi t) = \pi^2\sin(\pi t) \qquad // -\pi^2\sin(\pi t)$$

$$0 = 0$$

Kolejne kroki iteracyjne w metodzie pośredniej Eulera

$N = 28$

n	t	y	error
0	0.000000e+00	0.000000e+00	0.000000e+00
1	7.142857e-02	1.478544e-02	7.000909e-03
2	1.428571e-01	4.258339e-02	1.258963e-02
3	2.142857e-01	8.104035e-02	1.659683e-02
4	2.857143e-01	1.273353e-01	1.892209e-02
5	3.571429e-01	1.783165e-01	1.953981e-02
6	4.285714e-01	2.306551e-01	1.850141e-02
7	5.000000e-01	2.810080e-01	1.593345e-02
8	5.714286e-01	3.261821e-01	1.203204e-02
9	6.428571e-01	3.632902e-01	7.053620e-03
10	7.142857e-01	3.898933e-01	1.302678e-03
11	7.857143e-01	4.041194e-01	4.882946e-03
12	8.571429e-01	4.047545e-01	1.114854e-02
13	9.285714e-01	3.913012e-01	1.713983e-02
14	1.000000e+00	3.640012e-01	2.252037e-02
15	1.071429e+00	3.238203e-01	2.698801e-02
16	1.142857e+00	2.723987e-01	3.028972e-02
17	1.214286e+00	2.119662e-01	3.223396e-02
18	1.285714e+00	1.452290e-01	3.270001e-02
19	1.357143e+00	7.523166e-02	3.164376e-02
20	1.428571e+00	5.203677e-03	2.909967e-02
21	1.500000e+00	-6.160455e-02	2.517885e-02
22	1.571429e+00	-1.220858e-01	2.006329e-02
23	1.642857e+00	-1.734332e-01	1.399655e-02
24	1.714286e+00	-2.132822e-01	7.271367e-03
25	1.785714e+00	-2.398299e-01	2.148680e-04
26	1.857143e+00	-2.519271e-01	6.828000e-03
27	1.928571e+00	-2.491362e-01	1.351188e-02
28	2.000000e+00	-2.317546e-01	1.950842e-02

$N = 56$

n	t	y	error
0	0.000000e+00	0.000000e+00	0.000000e+00
1	3.571429e-02	3.854199e-03	1.877529e-03
2	7.142857e-02	1.137482e-02	3.590284e-03
3	1.071429e-01	2.233301e-02	5.124377e-03
4	1.428571e-01	3.646155e-02	6.467797e-03
5	1.785714e-01	5.345802e-02	7.610577e-03
6	2.142857e-01	7.298845e-02	8.544936e-03
7	2.500000e-01	9.469134e-02	9.265391e-03
8	2.857143e-01	1.181821e-01	9.768843e-03
9	3.214286e-01	1.430576e-01	1.005463e-02
10	3.571429e-01	1.689012e-01	1.012454e-02
11	3.928571e-01	1.952880e-01	9.982818e-03
12	4.285714e-01	2.217898e-01	9.636114e-03
13	4.642857e-01	2.479802e-01	9.093409e-03
14	5.000000e-01	2.734405e-01	8.365921e-03
15	5.357143e-01	2.977641e-01	7.466968e-03
16	5.714286e-01	3.205619e-01	6.411815e-03
17	6.071429e-01	3.414670e-01	5.217493e-03
18	6.428571e-01	3.601392e-01	3.902587e-03
19	6.785714e-01	3.762693e-01	2.487019e-03
20	7.142857e-01	3.895825e-01	9.918030e-04
21	7.500000e-01	3.998422e-01	5.612118e-04
22	7.857143e-01	4.068527e-01	2.149607e-03
23	8.214286e-01	4.104616e-01	3.750670e-03
24	8.571429e-01	4.105613e-01	5.341672e-03
25	8.928571e-01	4.070911e-01	6.900146e-03
26	9.285714e-01	4.000369e-01	8.404166e-03
27	9.642857e-01	3.894319e-01	9.832614e-03
28	1.000000e+00	3.753561e-01	1.116544e-02
29	1.035714e+00	3.579348e-01	1.238391e-02
30	1.071429e+00	3.373375e-01	1.347082e-02
31	1.107143e+00	3.137752e-01	1.441076e-02
32	1.142857e+00	2.874982e-01	1.519022e-02
33	1.178571e+00	2.587923e-01	1.579783e-02
34	1.214286e+00	2.279757e-01	1.622445e-02
35	1.250000e+00	1.953946e-01	1.646332e-02
36	1.285714e+00	1.614189e-01	1.651010e-02
37	1.321429e+00	1.264373e-01	1.636295e-02
38	1.357143e+00	9.085288e-02	1.602254e-02
39	1.392857e+00	5.507736e-02	1.549204e-02
40	1.428571e+00	1.952627e-02	1.477708e-02
41	1.464286e+00	-1.538646e-02	1.388564e-02
42	1.500000e+00	-4.925373e-02	1.282803e-02

43	1.535714e+00	-8.168043e-02	1.161666e-02
44	1.571429e+00	-1.122885e-01	1.026594e-02
45	1.607143e+00	-1.407215e-01	8.792095e-03
46	1.642857e+00	-1.666496e-01	7.212930e-03
47	1.678571e+00	-1.897733e-01	5.547626e-03
48	1.714286e+00	-2.098273e-01	3.816488e-03
49	1.750000e+00	-2.265842e-01	2.040690e-03
50	1.785714e+00	-2.398571e-01	2.420062e-04
51	1.821429e+00	-2.495019e-01	1.557466e-03
52	1.857143e+00	-2.554195e-01	3.335585e-03
53	1.892857e+00	-2.575568e-01	5.070444e-03
54	1.928571e+00	-2.559075e-01	6.740650e-03
55	1.964286e+00	-2.505120e-01	8.325595e-03
56	2.000000e+00	-2.414573e-01	9.805713e-03

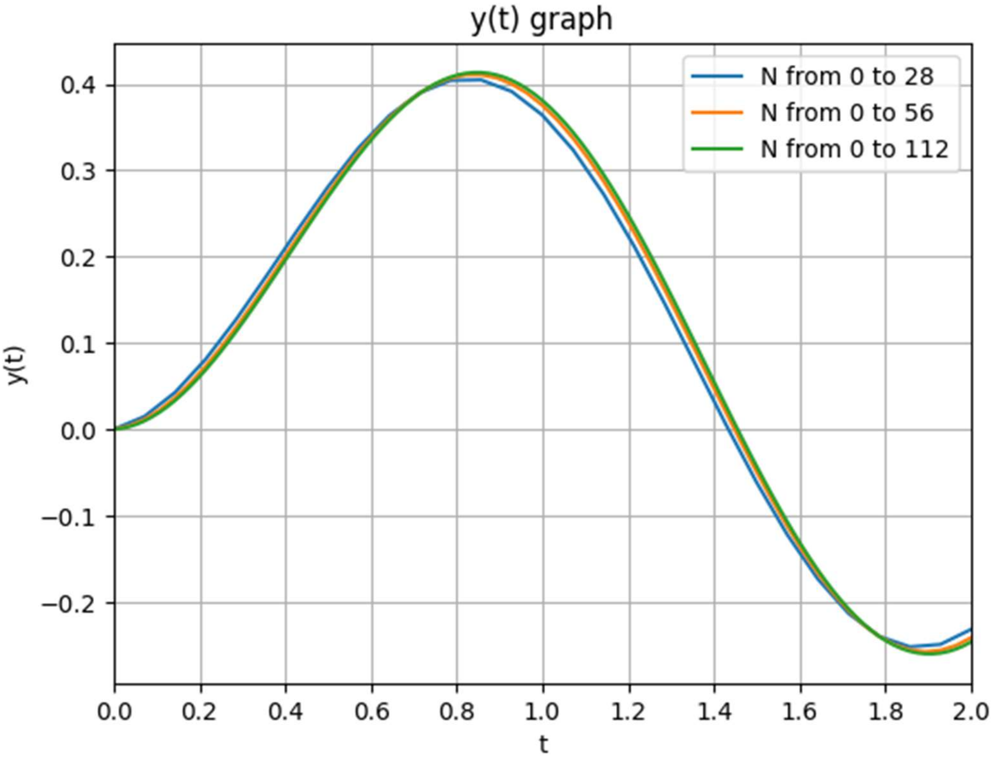
$N = 112$

n	**t**	**y**	**error**
0	0.000000e+00	0.000000e+00	0.000000e+00
1	1.785714e-02	9.828299e-04	4.851851e-04
2	3.571429e-02	2.927309e-03	9.506387e-04
3	5.357143e-02	5.809562e-03	1.395399e-03
4	7.142857e-02	9.603092e-03	1.818557e-03
5	8.928571e-02	1.427885e-02	2.219258e-03
6	1.071429e-01	1.980534e-02	2.596710e-03
7	1.250000e-01	2.614869e-02	2.950178e-03
8	1.428571e-01	3.327275e-02	3.278995e-03
9	1.607143e-01	4.113923e-02	3.582558e-03
10	1.785714e-01	4.970778e-02	3.860334e-03
11	1.964286e-01	5.893614e-02	4.111860e-03
12	2.142857e-01	6.878026e-02	4.336747e-03
13	2.321429e-01	7.919441e-02	4.534676e-03
14	2.500000e-01	9.013135e-02	4.705408e-03
15	2.678571e-01	1.015425e-01	4.848776e-03
16	2.857143e-01	1.133779e-01	4.964692e-03
17	3.035714e-01	1.255867e-01	5.053145e-03
18	3.214286e-01	1.381171e-01	5.114200e-03
19	3.392857e-01	1.509164e-01	5.148001e-03
20	3.571429e-01	1.639314e-01	5.154769e-03
21	3.750000e-01	1.771085e-01	5.134801e-03
22	3.928571e-01	1.903937e-01	5.088472e-03
23	4.107143e-01	2.037329e-01	5.016230e-03
24	4.285714e-01	2.170723e-01	4.918598e-03
25	4.464286e-01	2.303579e-01	4.796170e-03
26	4.642857e-01	2.435364e-01	4.649612e-03
27	4.821429e-01	2.565551e-01	4.479659e-03
28	5.000000e-01	2.693617e-01	4.287110e-03

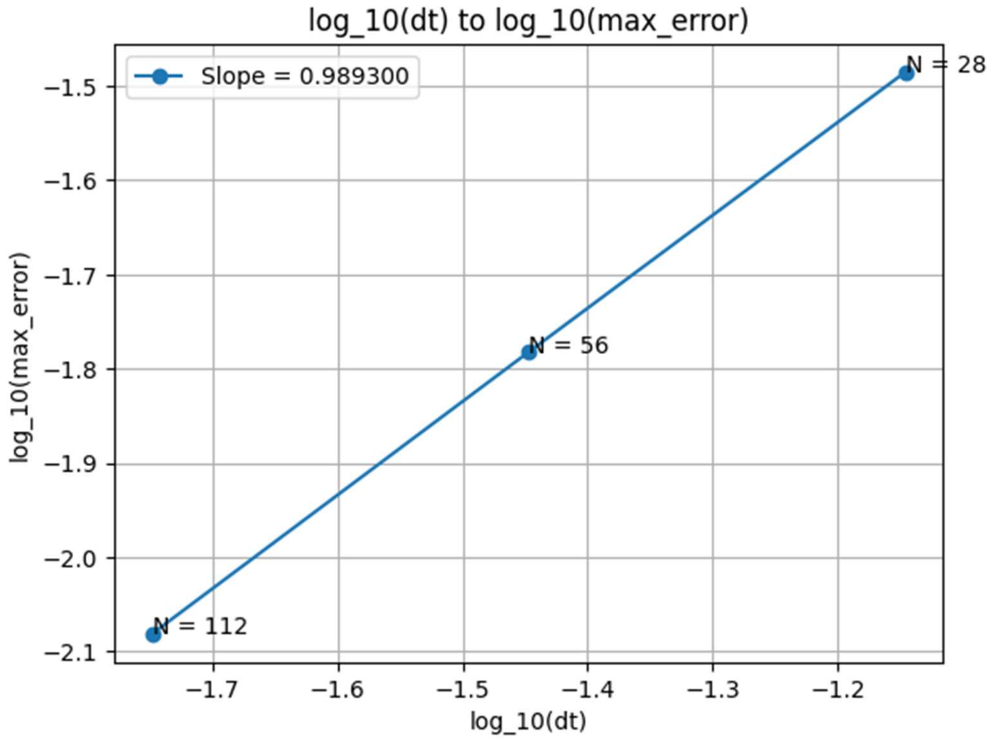
29	5.178571e-01	2.819050e-01	4.072832e-03
30	5.357143e-01	2.941349e-01	3.837752e-03
31	5.535714e-01	3.060022e-01	3.582854e-03
32	5.714286e-01	3.174592e-01	3.309181e-03
33	5.892857e-01	3.284598e-01	3.017827e-03
34	6.071429e-01	3.389594e-01	2.709937e-03
35	6.250000e-01	3.489151e-01	2.386700e-03
36	6.428571e-01	3.582860e-01	2.049350e-03
37	6.607143e-01	3.670331e-01	1.699158e-03
38	6.785714e-01	3.751197e-01	1.337430e-03
39	6.964286e-01	3.825112e-01	9.655039e-04
40	7.142857e-01	3.891754e-01	5.847437e-04
41	7.321429e-01	3.950826e-01	1.965362e-04
42	7.500000e-01	4.002057e-01	1.977133e-04
43	7.678571e-01	4.045200e-01	5.965854e-04
44	7.857143e-01	4.080037e-01	9.986510e-04
45	8.035714e-01	4.106376e-01	1.402475e-03
46	8.214286e-01	4.124056e-01	1.806623e-03
47	8.392857e-01	4.132942e-01	2.209662e-03
48	8.571429e-01	4.132929e-01	2.610167e-03
49	8.750000e-01	4.123941e-01	3.006728e-03
50	8.928571e-01	4.105933e-01	3.397947e-03
51	9.107143e-01	4.078889e-01	3.782450e-03
52	9.285714e-01	4.042822e-01	4.158887e-03
53	9.464286e-01	3.997775e-01	4.525939e-03
54	9.642857e-01	3.943822e-01	4.882316e-03
55	9.821429e-01	3.881065e-01	5.226770e-03
56	1.000000e+00	3.809634e-01	5.558091e-03
57	1.017857e+00	3.729690e-01	5.875115e-03
58	1.035714e+00	3.641420e-01	6.176726e-03
59	1.053571e+00	3.545038e-01	6.461860e-03
60	1.071429e+00	3.440788e-01	6.729507e-03
61	1.089286e+00	3.328935e-01	6.978718e-03
62	1.107143e+00	3.209774e-01	7.208601e-03
63	1.125000e+00	3.083620e-01	7.418332e-03
64	1.142857e+00	2.950813e-01	7.607149e-03
65	1.160714e+00	2.811715e-01	7.774362e-03
66	1.178571e+00	2.666708e-01	7.919351e-03
67	1.196429e+00	2.516195e-01	8.041568e-03
68	1.214286e+00	2.360597e-01	8.140538e-03
69	1.232143e+00	2.200350e-01	8.215865e-03
70	1.250000e+00	2.035907e-01	8.267228e-03
71	1.267857e+00	1.867737e-01	8.294383e-03
72	1.285714e+00	1.696318e-01	8.297165e-03
73	1.303571e+00	1.522142e-01	8.275490e-03

74	1.321429e+00	1.345709e-01	8.229350e-03
75	1.339286e+00	1.167528e-01	8.158819e-03
76	1.357143e+00	9.881137e-02	8.064048e-03
77	1.375000e+00	8.079855e-02	7.945267e-03
78	1.392857e+00	6.276662e-02	7.802785e-03
79	1.410714e+00	4.476798e-02	7.636984e-03
80	1.428571e+00	2.685502e-02	7.448325e-03
81	1.446429e+00	9.079933e-03	7.237340e-03
82	1.464286e+00	-8.505451e-03	7.004635e-03
83	1.482143e+00	-2.584982e-02	6.750886e-03
84	1.500000e+00	-4.290254e-02	6.476836e-03
85	1.517857e+00	-5.961383e-02	6.183294e-03
86	1.535714e+00	-7.593491e-02	5.871131e-03
87	1.553571e+00	-9.181816e-02	5.541281e-03
88	1.571429e+00	-1.072173e-01	5.194731e-03
89	1.589286e+00	-1.220874e-01	4.832527e-03
90	1.607143e+00	-1.363852e-01	4.455761e-03
91	1.625000e+00	-1.500692e-01	4.065576e-03
92	1.642857e+00	-1.630999e-01	3.663156e-03
93	1.660714e+00	-1.754394e-01	3.249726e-03
94	1.678571e+00	-1.870522e-01	2.826547e-03
95	1.696429e+00	-1.979051e-01	2.394913e-03
96	1.714286e+00	-2.079670e-01	1.956143e-03
97	1.732143e+00	-2.172093e-01	1.511583e-03
98	1.750000e+00	-2.256061e-01	1.062596e-03
99	1.767857e+00	-2.331339e-01	6.105606e-04
100	1.785714e+00	-2.397719e-01	1.568670e-04
101	1.803571e+00	-2.455022e-01	2.970889e-04
102	1.821429e+00	-2.503095e-01	7.499093e-04
103	1.839286e+00	-2.541814e-01	1.200199e-03
104	1.857143e+00	-2.571085e-01	1.646569e-03
105	1.875000e+00	-2.590843e-01	2.087643e-03
106	1.892857e+00	-2.601052e-01	2.522059e-03
107	1.910714e+00	-2.601705e-01	2.948476e-03
108	1.928571e+00	-2.592825e-01	3.365576e-03
109	1.946429e+00	-2.574466e-01	3.772072e-03
110	1.964286e+00	-2.546709e-01	4.166707e-03
111	1.982143e+00	-2.509665e-01	4.548261e-03
112	2.000000e+00	-2.463475e-01	4.915554e-03

Wykres funkcji



Wykres zależności $\log_{10} błędu\ obliczenia$ od $\log_{10} \partial t$



Współczynnik nachylenia wykresu wynosi 0,989300 i jest to rząd dokładności metody pośredniej Eulera. Współczynnik ten koreluje z rzędem dokładności (wykładnikami przy wartości ∂t) dla tej metody, gdzie lokalny błąd wynosi $O(\partial t^1)$.

Zadanie 3

Metoda pośrednia Eulera w języku Python, używając biblioteki *numpy*

```
def exact_solution_getter(q: float, r: float, s: float) -> ExactSolutionType:
    def exact_solution(x: float) -> float:
        negative_half_q: float = -q / 2
        square_root: float = sqrt(square(q) - 4 * r)
        lambda_1: float = negative_half_q - square_root / 2
        lambda_2: float = negative_half_q + square_root / 2
        A: float = (exp(5 * lambda_2) - 1) / (exp(5 * lambda_1) - exp(5 * lambda_2))
        B: float = (exp(5 * lambda_1) - 1) / (exp(5 * lambda_2) - exp(5 * lambda_1))
        return s / r * (A * exp(lambda_1 * x) + B * exp(lambda_2 * x) + 1)
    return exact_solution

def lower_upper_decomposition(
        A: FloatArray2D,
) -> tuple[FloatArray2D, FloatArray2D]:
    n: int = A.shape[0]
    a: FloatArray2D = A.copy()
    for k in range(n - 1):
        akk: float = a[k][k]
        for i in range(k + 1, n):
            aux: float = a[i][k] / akk if akk else 0
            for j in range(k + 1, n):
                a[i][j] -= a[k][j] * aux
            a[i][k] = aux
    U: FloatArray2D = triu(a)
    L: FloatArray2D = a - U + eye(n)
    return L, U

def eliminate_forward(L: FloatArray2D, B: FloatArray) -> FloatArray:
    n: int = L.shape[0]
    b: FloatArray = B.copy()
    for k in range(n - 1):
        for i in range(k + 1, n):
            b[i] -= b[k] * L[i][k]
    return b

def substitute_backward(U: FloatArray2D, Y: FloatArray) -> FloatArray:
    n: int = U.shape[0]
    y: FloatArray = Y.copy()
    y[n-1] /= U[n-1][n-1]
    for i in range(n-2, -1, -1):
        s: float = 0.0
        for j in range(i+1, n):
            s += U[i][j] * y[j]
        y[i] -= s
        y[i] /= U[i][i]
    return y

def solve_2_degree_differential_equation_getter(
    q: float, r: float, s: float, x_min: float, x_max: float,
    y_for_x_min: float, y_for_x_max: float, exact_solution: ExactSolutionType,
) -> Solve2DegreeDifferentialEquationType:
    def solve_2_degree_differential_equation(
            N: int,
    ) -> Solve2DegreeDifferentialEquationReturnType:
        h: float = (x_max - x_min) / N

        A: FloatArray2D = zeros((N + 1, N + 1))
        A[0][0] = 1
        A[N][N] = 1

        B: FloatArray = repeat(s, N + 1)
        B[0] = y_for_x_min
        B[N] = y_for_x_max

        for i in arange(1, N):
            A[i][i - 1] = 1 / square(h) - q / 2 / h
```

```
            A[i][i] = r - 2 / square(h)
            A[i][i + 1] = 1 / square(h) + q / 2 / h

        L, U = lower_upper_decomposition(A)
        eliminated: FloatArray = eliminate_forward(L, B)
        ys: FloatArray = substitute_backward(U, eliminated)

        xs: list[float] = []
        errors = []
        for i in arange(0, N + 1):
            xs.append(x_min + h * i)
            errors.append(abs(exact_solution(xs[i]) - ys[i]))

        return array(xs), ys, array(errors), h
    return solve_2_degree_differential_equation
```

Problem początkowy

$$\begin{cases} y''(x) + q \cdot y'(x) + r \cdot y(x) = s \quad x \in (0;\ 5) \\ y(0) = 0 \\ y(5) = 0 \end{cases}$$

Dane i ilość iteracji N

$$q = 0{,}47$$

$$r = -0{,}84$$

$$s = 2{,}37$$

$$N \in \{28;\ 56;\ 112\}$$

Ścisłe rozwiązanie

$$y(\mathrm{x}) = \frac{\mathrm{s}}{\mathrm{r}}\left(\mathrm{A}e^{\lambda_1 \mathrm{x}} + \mathrm{B}e^{\lambda_2 \mathrm{x}} + 1\right)$$

gdzie:

$$\lambda_1 = \frac{-\mathrm{q}-\sqrt{\mathrm{q}^2-4}}{2} \qquad \lambda_2 = \frac{-\mathrm{q}+\sqrt{\mathrm{q}^2-4\mathrm{r}}}{2} \qquad \mathrm{A} = \frac{e^{5\lambda_2}-1}{e^{5\lambda_1}-e^{5\lambda_2}} \qquad \mathrm{B} = \frac{e^{5\lambda_1}-1}{e^{5\lambda_2}-e^{5\lambda_1}}$$

Sprawdzenie rozwiązania dla warunków brzegowych

$$\lambda_1 = \frac{-0{,}47 - \sqrt{0{,}47^2 - 4 \cdot (\text{-}0{,}84)}}{2} = \frac{-0{,}47 - \sqrt{0{,}2209 + 3{,}36}}{2} = \frac{-0{,}47 - \sqrt{3{,}5809}}{2}$$
$$\approx \frac{-0{,}47 - 1{,}892327}{2} = \frac{-2{,}362327}{2} \approx -1{,}181163$$

$$\lambda_2 = \frac{-0{,}47 + \sqrt{0{,}47^2 - 4 \cdot (\text{-}0{,}84)}}{2} = \frac{-0{,}47 + \sqrt{0{,}2209 + 3{,}36}}{2} = \frac{-0{,}47 + \sqrt{3{,}5809}}{2}$$
$$\approx \frac{-0{,}47 + 1{,}892327}{2} = \frac{1{,}422327}{2} \approx 0{,}711163$$

$$\mathrm{A} \approx \frac{e^{5 \cdot 0{,}711163} - 1}{e^{5 \cdot (-1{,}181163)} - e^{5 \cdot 0{,}711163}} = \frac{e^{3{,}555815} - 1}{e^{-5{,}905815} - e^{3{,}555815}} \approx \frac{35{,}016347 - 1}{0{,}002724 - 35{,}016347} = \frac{34{,}016347}{-35{,}013623}$$
$$\approx -0{,}971517$$

$$\mathrm{B} \approx \frac{e^{5 \cdot (-1{,}181163)} - 1}{e^{5 \cdot 0{,}711163} - e^{5 \cdot (-1{,}181163)}} = \frac{e^{-5{,}905815} - 1}{e^{3{,}555815} - e^{-5{,}905815}} \approx \frac{0{,}002724 - 1}{35{,}016347 - 0{,}002724} = \frac{-1{,}002724}{35{,}013623}$$
$$\approx -0{,}028638$$

$$y(0)=\frac{s}{r}\left(Ae^{\lambda_1\cdot 0}+Be^{\lambda_2\cdot 0}+1\right)=\frac{s}{r}(Ae^0+Be^0+1)=\frac{s}{r}(A\cdot 1+B\cdot 1+1)=\frac{s}{r}(A+B+1)$$
$$\approx\frac{2{,}37}{-0{,}84}(-0{,}971517-0{,}028638+1)\approx-2{,}821429\cdot(-0{,}000155)\approx 0{,}000437$$

$$y(5)=\frac{s}{r}\left(Ae^{\lambda_1\cdot 5}+Be^{\lambda_2\cdot 5}+1\right)\approx\frac{2{,}37}{-0{,}84}\left(-0{,}971517\cdot e^{5\cdot(-1{,}181163)}-0{,}028638\cdot e^{5\cdot 0{,}711163}+1\right)$$
$$\approx-2{,}821429\cdot(-0{,}971517\cdot e^{-5{,}905815}-0{,}028638\cdot e^{3{,}555815}+1)$$
$$\approx-2{,}821429\cdot(-0{,}971517\cdot 0{,}002724-0{,}028638\cdot 35{,}016347+1)$$
$$\approx-2{,}821429\cdot(-0{,}002646-1{,}002798+1)=-2{,}821429\cdot(-0{,}005444)$$
$$\approx-0{,}015360$$

Ścisłe rozwiązanie spełnia zadane równanie różniczkowe dla warunków brzegowych. Obliczone $y(0)$ i $y(5)$ nie są dokładnie równe 0 ze względu na błąd zaokrąglenia.

Kolejne wartości obliczone za pomocą dekompozycji LU

$N = 28$

n	x	y	error
0	0.000000e+00	0.000000e+00	0.000000e+00
1	1.785714e-01	-5.100519e-01	3.153510e-04
2	3.571429e-01	-9.196016e-01	5.373573e-04
3	5.357143e-01	-1.247273e+00	6.922712e-04
4	7.142857e-01	-1.508083e+00	7.998338e-04
5	8.928571e-01	-1.714124e+00	8.747893e-04
6	1.071429e+00	-1.875103e+00	9.280526e-04
7	1.250000e+00	-1.998789e+00	9.676077e-04
8	1.428571e+00	-2.091364e+00	9.991941e-04
9	1.607143e+00	-2.157715e+00	1.026829e-03
10	1.785714e+00	-2.201659e+00	1.053202e-03
11	1.964286e+00	-2.226131e+00	1.079963e-03
12	2.142857e+00	-2.233329e+00	1.107937e-03
13	2.321429e+00	-2.224828e+00	1.137273e-03
14	2.500000e+00	-2.201676e+00	1.167539e-03
15	2.678571e+00	-2.164456e+00	1.197782e-03
16	2.857143e+00	-2.113346e+00	1.226547e-03
17	3.035714e+00	-2.048150e+00	1.251871e-03
18	3.214286e+00	-1.968327e+00	1.271245e-03
19	3.392857e+00	-1.873003e+00	1.281557e-03
20	3.571429e+00	-1.760976e+00	1.279009e-03
21	3.750000e+00	-1.630711e+00	1.259004e-03
22	3.928571e+00	-1.480329e+00	1.216015e-03
23	4.107143e+00	-1.307585e+00	1.143417e-03
24	4.285714e+00	-1.109839e+00	1.033297e-03
25	4.464286e+00	-8.840211e-01	8.762183e-04
26	4.642857e+00	-6.265878e-01	6.609511e-04
27	4.821429e+00	-3.334676e-01	3.741537e-04
28	5.000000e+00	0.000000e+00	3.132415e-16

$N = 56$

n	t	y	error
0	0.000000e+00	0.000000e+00	0.000000e+00
1	8.928571e-02	-2.690435e-01	4.294541e-05
2	1.785714e-01	-5.102882e-01	7.906035e-05
3	2.678571e-01	-7.264590e-01	1.093539e-04
4	3.571429e-01	-9.200042e-01	1.347035e-04
5	4.464286e-01	-1.093123e+00	1.558709e-04
6	5.357143e-01	-1.247791e+00	1.735168e-04
7	6.250000e-01	-1.385781e+00	1.882130e-04
8	7.142857e-01	-1.508683e+00	2.004537e-04
9	8.035714e-01	-1.617923e+00	2.106649e-04
10	8.928571e-01	-1.714780e+00	2.192133e-04
11	9.821429e-01	-1.800397e+00	2.264134e-04
12	1.071429e+00	-1.875799e+00	2.325340e-04
13	1.160714e+00	-1.941899e+00	2.378044e-04
14	1.250000e+00	-1.999514e+00	2.424188e-04
15	1.339286e+00	-2.049370e+00	2.465407e-04
16	1.428571e+00	-2.092113e+00	2.503072e-04
17	1.517857e+00	-2.128315e+00	2.538318e-04
18	1.607143e+00	-2.158484e+00	2.572072e-04
19	1.696429e+00	-2.183065e+00	2.605081e-04
20	1.785714e+00	-2.202448e+00	2.637931e-04
21	1.875000e+00	-2.216975e+00	2.671062e-04
22	1.964286e+00	-2.226940e+00	2.704789e-04
23	2.053571e+00	-2.232597e+00	2.739309e-04
24	2.142857e+00	-2.234159e+00	2.774717e-04
25	2.232143e+00	-2.231805e+00	2.811009e-04
26	2.321429e+00	-2.225681e+00	2.848090e-04
27	2.410714e+00	-2.215901e+00	2.885783e-04
28	2.500000e+00	-2.202551e+00	2.923828e-04
29	2.589286e+00	-2.185691e+00	2.961887e-04
30	2.678571e+00	-2.165354e+00	2.999543e-04
31	2.767857e+00	-2.141550e+00	3.036302e-04
32	2.857143e+00	-2.114265e+00	3.071590e-04
33	2.946429e+00	-2.083464e+00	3.104753e-04
34	3.035714e+00	-2.049088e+00	3.135049e-04
35	3.125000e+00	-2.011060e+00	3.161651e-04
36	3.214286e+00	-1.969280e+00	3.183636e-04
37	3.303571e+00	-1.923628e+00	3.199981e-04
38	3.392857e+00	-1.873963e+00	3.209554e-04
39	3.482143e+00	-1.820126e+00	3.211111e-04
40	3.571429e+00	-1.761934e+00	3.203282e-04
41	3.660714e+00	-1.699185e+00	3.184563e-04
42	3.750000e+00	-1.631655e+00	3.153302e-04

43	3.839286e+00	-1.559096e+00	3.107694e-04
44	3.928571e+00	-1.481241e+00	3.045760e-04
45	4.017857e+00	-1.397795e+00	2.965335e-04
46	4.107143e+00	-1.308442e+00	2.864054e-04
47	4.196429e+00	-1.212838e+00	2.739332e-04
48	4.285714e+00	-1.110613e+00	2.588346e-04
49	4.375000e+00	-1.001369e+00	2.408017e-04
50	4.464286e+00	-8.846779e-01	2.194982e-04
51	4.553571e+00	-7.600799e-01	1.945578e-04
52	4.642857e+00	-6.270832e-01	1.655808e-04
53	4.732143e+00	-4.851605e-01	1.321321e-04
54	4.821429e+00	-3.337481e-01	9.373747e-05
55	4.910714e+00	-1.722427e-01	4.988071e-05
56	5.000000e+00	0.000000e+00	3.132415e-16

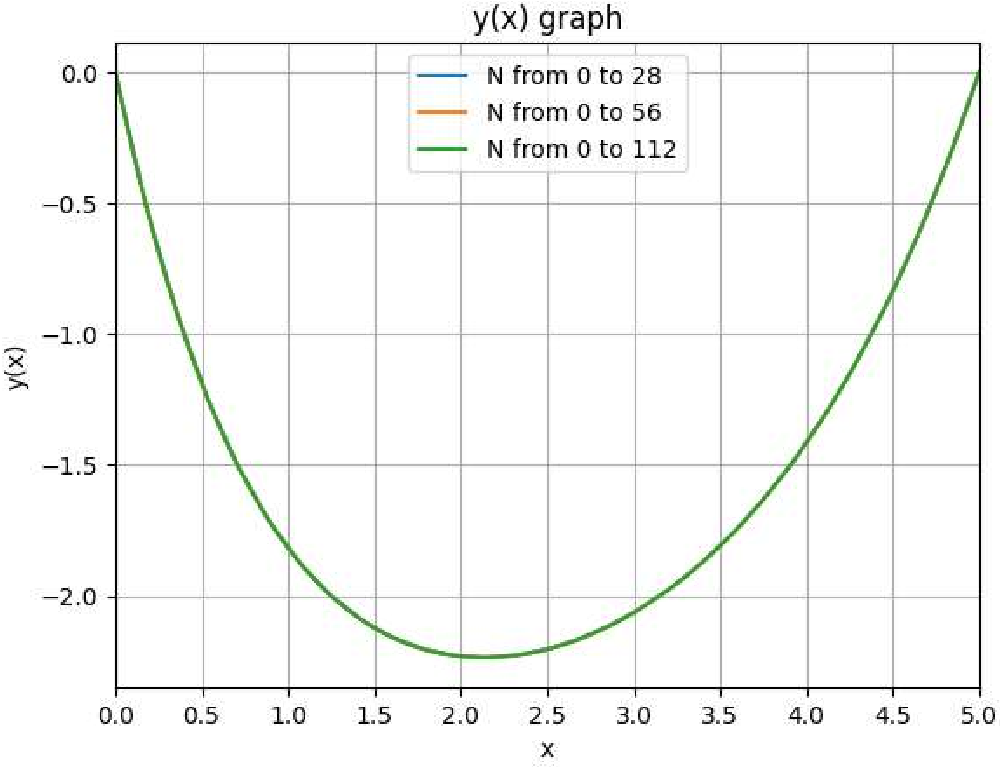
$N = 112$

n	t	y	error
0	0.000000e+00	0.000000e+00	0.000000e+00
1	4.464286e-02	-1.381953e-01	5.602870e-06
2	8.928571e-02	-2.690757e-01	1.074408e-05
3	1.339286e-01	-3.930099e-01	1.545859e-05
4	1.785714e-01	-5.103475e-01	1.977904e-05
5	2.232143e-01	-6.214198e-01	2.373585e-05
6	2.678571e-01	-7.265410e-01	2.735739e-05
7	3.125000e-01	-8.260087e-01	3.067013e-05
8	3.571429e-01	-9.201052e-01	3.369869e-05
9	4.017857e-01	-1.009098e+00	3.646605e-05
10	4.464286e-01	-1.093240e+00	3.899358e-05
11	4.910714e-01	-1.172772e+00	4.130117e-05
12	5.357143e-01	-1.247921e+00	4.340734e-05
13	5.803571e-01	-1.318903e+00	4.532931e-05
14	6.250000e-01	-1.385922e+00	4.708308e-05
15	6.696429e-01	-1.449171e+00	4.868353e-05
16	7.142857e-01	-1.508833e+00	5.014445e-05
17	7.589286e-01	-1.565082e+00	5.147867e-05
18	8.035714e-01	-1.618081e+00	5.269807e-05
19	8.482143e-01	-1.667986e+00	5.381366e-05
20	8.928571e-01	-1.714944e+00	5.483566e-05
21	9.375000e-01	-1.759094e+00	5.577350e-05
22	9.821429e-01	-1.800567e+00	5.663592e-05
23	1.026786e+00	-1.839488e+00	5.743099e-05
24	1.071429e+00	-1.875973e+00	5.816615e-05
25	1.116071e+00	-1.910135e+00	5.884827e-05
26	1.160714e+00	-1.942077e+00	5.948367e-05
27	1.205357e+00	-1.971900e+00	6.007816e-05
28	1.250000e+00	-1.999696e+00	6.063708e-05

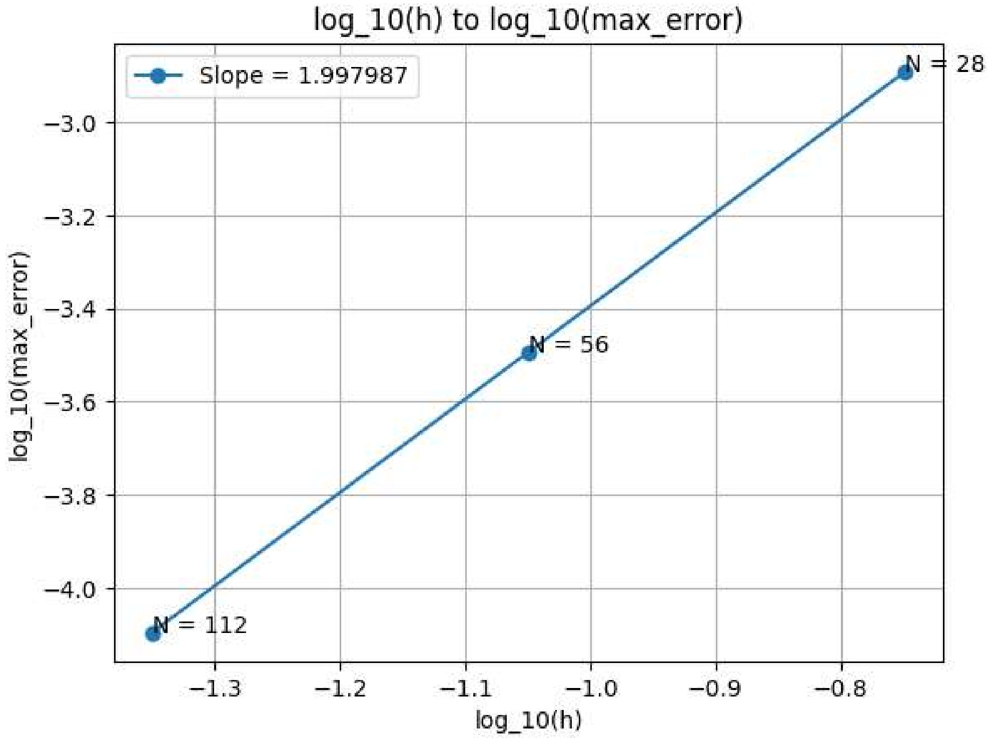
29	1.294643e+00	-2.025553e+00	6.116531e-05
30	1.339286e+00	-2.049555e+00	6.166733e-05
31	1.383929e+00	-2.071779e+00	6.214722e-05
32	1.428571e+00	-2.092300e+00	6.260869e-05
33	1.473214e+00	-2.111188e+00	6.305511e-05
34	1.517857e+00	-2.128506e+00	6.348954e-05
35	1.562500e+00	-2.144316e+00	6.391473e-05
36	1.607143e+00	-2.158677e+00	6.433313e-05
37	1.651786e+00	-2.171641e+00	6.474694e-05
38	1.696429e+00	-2.183260e+00	6.515811e-05
39	1.741071e+00	-2.193580e+00	6.556834e-05
40	1.785714e+00	-2.202646e+00	6.597913e-05
41	1.830357e+00	-2.210498e+00	6.639174e-05
42	1.875000e+00	-2.217175e+00	6.680724e-05
43	1.919643e+00	-2.222713e+00	6.722653e-05
44	1.964286e+00	-2.227143e+00	6.765031e-05
45	2.008929e+00	-2.230497e+00	6.807910e-05
46	2.053571e+00	-2.232802e+00	6.851327e-05
47	2.098214e+00	-2.234085e+00	6.895305e-05
48	2.142857e+00	-2.234367e+00	6.939848e-05
49	2.187500e+00	-2.233671e+00	6.984948e-05
50	2.232143e+00	-2.232016e+00	7.030583e-05
51	2.276786e+00	-2.229419e+00	7.076717e-05
52	2.321429e+00	-2.225894e+00	7.123300e-05
53	2.366071e+00	-2.221457e+00	7.170270e-05
54	2.410714e+00	-2.216117e+00	7.217552e-05
55	2.455357e+00	-2.209886e+00	7.265059e-05
56	2.500000e+00	-2.202770e+00	7.312691e-05
57	2.544643e+00	-2.194778e+00	7.360336e-05
58	2.589286e+00	-2.185913e+00	7.407869e-05
59	2.633929e+00	-2.176180e+00	7.455155e-05
60	2.678571e+00	-2.165579e+00	7.502046e-05
61	2.723214e+00	-2.154112e+00	7.548379e-05
62	2.767857e+00	-2.141778e+00	7.593984e-05
63	2.812500e+00	-2.128574e+00	7.638672e-05
64	2.857143e+00	-2.114496e+00	7.682248e-05
65	2.901786e+00	-2.099539e+00	7.724499e-05
66	2.946429e+00	-2.083697e+00	7.765200e-05
67	2.991071e+00	-2.066961e+00	7.804115e-05
68	3.035714e+00	-2.049324e+00	7.840990e-05
69	3.080357e+00	-2.030773e+00	7.875560e-05
70	3.125000e+00	-2.011297e+00	7.907544e-05
71	3.169643e+00	-1.990884e+00	7.936645e-05
72	3.214286e+00	-1.969518e+00	7.962553e-05
73	3.258929e+00	-1.947185e+00	7.984939e-05

74	3.303571e+00	-1.923868e+00	8.003459e-05
75	3.348214e+00	-1.899547e+00	8.017751e-05
76	3.392857e+00	-1.874204e+00	8.027434e-05
77	3.437500e+00	-1.847818e+00	8.032111e-05
78	3.482143e+00	-1.820367e+00	8.031362e-05
79	3.526786e+00	-1.791827e+00	8.024750e-05
80	3.571429e+00	-1.762174e+00	8.011816e-05
81	3.616071e+00	-1.731382e+00	7.992079e-05
82	3.660714e+00	-1.699424e+00	7.965034e-05
83	3.705357e+00	-1.666270e+00	7.930154e-05
84	3.750000e+00	-1.631891e+00	7.886887e-05
85	3.794643e+00	-1.596255e+00	7.834656e-05
86	3.839286e+00	-1.559329e+00	7.772855e-05
87	3.883929e+00	-1.521079e+00	7.700853e-05
88	3.928571e+00	-1.481469e+00	7.617989e-05
89	3.973214e+00	-1.440462e+00	7.523570e-05
90	4.017857e+00	-1.398018e+00	7.416873e-05
91	4.062500e+00	-1.354097e+00	7.297144e-05
92	4.107143e+00	-1.308657e+00	7.163591e-05
93	4.151786e+00	-1.261654e+00	7.015389e-05
94	4.196429e+00	-1.213044e+00	6.851676e-05
95	4.241071e+00	-1.162778e+00	6.671548e-05
96	4.285714e+00	-1.110807e+00	6.474066e-05
97	4.330357e+00	-1.057082e+00	6.258244e-05
98	4.375000e+00	-1.001550e+00	6.023056e-05
99	4.419643e+00	-9.441553e-01	5.767427e-05
100	4.464286e+00	-8.848425e-01	5.490237e-05
101	4.508929e+00	-8.235528e-01	5.190315e-05
102	4.553571e+00	-7.602258e-01	4.866440e-05
103	4.598214e+00	-6.947989e-01	4.517336e-05
104	4.642857e+00	-6.272073e-01	4.141672e-05
105	4.687500e+00	-5.573840e-01	3.738057e-05
106	4.732143e+00	-4.852596e-01	3.305041e-05
107	4.776786e+00	-4.107625e-01	2.841110e-05
108	4.821429e+00	-3.338184e-01	2.344685e-05
109	4.866071e+00	-2.543507e-01	1.814118e-05
110	4.910714e+00	-1.722801e-01	1.247690e-05
111	4.955357e+00	-8.752477e-02	6.436077e-06
112	5.000000e+00	0.000000e+00	3.132415e-16

Wykres funkcji



Wykres zależności $\log_{10}$ *błędu obliczenia* od $\log_{10} \mathrm{h}$



Współczynnik nachylenia wykresu wynosi 1,997987 i jest to rząd dokładności rozwiązania równania różniczkowego 2-go rzędu. Współczynnik ten koreluje z rzędem dokładności (wykładnikami przy wartości h) obliczania 1. i 2. pochodnej za pomocą różnicy centralnej – zostały one użyte do wypełnienia macierzy A, gdzie lokalny błąd wynosi $O(\mathrm{h}^2)$.

Appendix

ex_1.py

```
from matplotlib.pyplot import figure, Axes, Figure
from numpy import float64, cos, sin, abs, log10, array
from numpy.typing import NDArray
from pandas import DataFrame, set_option, reset_option
from typing import TypeAlias, Callable

FloatArray: TypeAlias = NDArray[float64]
Function: TypeAlias = Callable[[float], float]
DerivativeMethod: TypeAlias = Callable[[Function, float, float], float]

set_option('display.float_format', lambda x: '%.6e' % x)

def progresive_diff_derivative(f: Function, x: float, h: float) -> float:
    return (f(x + h) - f(x)) / h

def regresive_diff_derivative(f: Function, x: float, h: float) -> float:
    return (f(x) - f(x - h)) / h

def central_diff_derivative(f: Function, x: float, h: float) -> float:
    return (f(x + h) - f(x - h)) / (2 * h)

def central_diff_derivative_2(f: Function, x: float, h: float) -> float:
    return (f(x - h) - 2 * f(x) + f(x + h)) / h ** 2

def calculate_slope(xs: FloatArray, ys: FloatArray) -> float:
    return (ys[-1] - ys[0]) / (xs[-1] - xs[0])

def process(diff_derivative_name: str,
            diff_derivative_method: DerivativeMethod,
            f: Function,
            x: float,
            hs: FloatArray,
            exact_value: float,
) -> FloatArray:
    diff_derivatives: list[float] = []
    diff_derivative_errors: list[float] = []
    degree: str = '2' if diff_derivative_name.endswith('2') else ''
    print(f'f_derivative{degree} = {exact_value:.6e}')
    print(diff_derivative_name)
    for h in hs:
        diff_derivative: float = diff_derivative_method(f, x, h)
        diff_derivatives.append(diff_derivative)
        diff_derivative_error: float = abs(exact_value - diff_derivative)
        diff_derivative_errors.append(diff_derivative_error)
    data: dict[str, list[float]] = {
        'values': diff_derivatives,
        'errors': diff_derivative_errors
    }
    data_frame = DataFrame(data, index = hs).rename_axis('h', axis=1)
    print(data_frame, '\n')
    return log10(diff_derivative_errors)

if __name__ == '__main__':
    x: float = 4.63
    hs: FloatArray = array([0.1, 0.05, 0.025, 0.0125])

    f: Function = lambda x: cos(x)
    f_p: Function = lambda x: -sin(x)
    f_pp: Function = lambda x: -cos(x)

    f_derivative: float = f_p(x)
    f_derivative2: float = f_pp(x)

    log10_regr_diff_errors: FloatArray = process(
        'regresive_diff_derivative',
        regresive_diff_derivative,
```

```
        f, x, hs, f_derivative,
    )
    log10_cent_diff_errors: FloatArray = process(
        'central_diff_derivative',
        central_diff_derivative,
        f, x, hs, f_derivative,
    )
    log10_cent_diff2_errors: FloatArray = process(
        'central_diff_derivative_2',
        central_diff_derivative_2,
        f, x, hs, f_derivative2,
    )

    log10_hs: FloatArray = log10(hs)
    regr_diff_slope: float = calculate_slope(log10_hs, log10_regr_diff_errors)
    cent_diff_slope: float = calculate_slope(log10_hs, log10_cent_diff_errors)
    cent_diff2_slope: float = calculate_slope(log10_hs, log10_cent_diff2_errors)
    regr_diff_slope_text: str = f'slope = {regr_diff_slope:.6f}'
    cent_diff_slope_text: str = f'slope = {cent_diff_slope:.6f}'
    cent_diff2_slope_text: str = f'slope = {cent_diff2_slope:.6f}'
    regr_diff_label = f'Regresive 1st derivative, ' + regr_diff_slope_text
    cent_diff_label = f'Central 1st derivative, ' + cent_diff_slope_text
    cent_diff2_label = f'Central 2nd derivative, ' + cent_diff2_slope_text

    axes: Axes = figure().add_subplot()
    axes.set_xlabel('log_10(h)')
    axes.set_ylabel('log_10(|diff_derivative - exact_derivative|)')
    axes.plot(log10_hs, log10_regr_diff_errors, label = regr_diff_label)
    axes.plot(log10_hs, log10_cent_diff_errors, label = cent_diff_label)
    axes.plot(log10_hs, log10_cent_diff2_errors, label = cent_diff2_label)
    axes.grid()
    axes.legend()
    if isinstance(fig := axes.get_figure(), Figure):
        fig.show()

    reset_option('display.float_format')
```

ex_2.py

```
from matplotlib.pyplot import figure, Axes, Figure
from numpy import (sin, cos, exp, square, pi, int_, float64, arange, abs, log10,
                   array, append)
from numpy.typing import NDArray
from pandas import DataFrame, set_option, reset_option
from typing import TypeAlias, Callable, Annotated, Literal

FloatArray: TypeAlias = Annotated[NDArray[float64], Literal[1]]
FloatArray2D: TypeAlias = Annotated[NDArray[float64], Literal[2]]
TCurrentGetterType: TypeAlias = Callable[[int], float]
YNextGetterType: TypeAlias = Callable[[float, float], float]
ExactSolutionType: TypeAlias = Callable[[float], float]
ImplicitEulerReturnType: TypeAlias = tuple[FloatArray, FloatArray, FloatArray, float]
ImplicitEulerType: TypeAlias = Callable[[int], ImplicitEulerReturnType]

set_option('display.float_format', lambda x: '%.6e' % x)
set_option('display.max_columns', 1000)

def y_next_getter_getter(a: float, dt: float) -> YNextGetterType:
    def y_next_getter(y_previous: float, t: float) -> float:
        return (y_previous + dt * sin(pi * t)) / (1 + a * dt)
    return y_next_getter

def exact_solution_getter(a: float) -> ExactSolutionType:
    def exact_solution(t: float) -> float:
        numerator: float = pi * exp(-a*t) - pi * cos(pi*t) + a * sin(pi*t)
        denominator: float = square(pi) + square(a)
        return numerator / denominator
    return exact_solution

def implicit_euler_getter(
```

```
        a: float, t_min: float, t_max: float,
        y0: float, exact_solution: ExactSolutionType,
) -> ImplicitEulerType:
    def implicit_euler(N: int) -> ImplicitEulerReturnType:
        dt: float = (t_max - t_min) / N
        y_next_getter: YNextGetterType = y_next_getter_getter(a, dt)
        current_t_getter: TCurrentGetterType = lambda i: t_min + dt * i
        ts: list[float] = [current_t_getter(0)]
        ys: list[float] = [y0]
        errors: list[float] = [abs(exact_solution(ts[0]) - ys[0])]
        for i in arange(1, N + 1):
            ts.append(current_t_getter(i))
            ys.append(y_next_getter(ys[i - 1], ts[i]))
            errors.append(abs(exact_solution(ts[i]) - ys[i]))
        return array(ts), array(ys), array(errors), dt
    return implicit_euler

def get_axes(
        title: str, x_limits: list[float] | None,
        x_ticks: FloatArray | None, x_label: str, y_label: str,
) -> Axes:
    axes: Axes = figure().add_subplot()
    axes.set_title(title)
    if x_limits is not None:
        axes.set_xlim(*x_limits)
    if x_ticks is not None:
        axes.set_xticks(x_ticks)
    axes.set_xlabel(x_label)
    axes.set_ylabel(y_label)
    axes.grid()
    return axes

def process(
        implicit_euler: ImplicitEulerType, n: int, solution_plot: Axes,
) -> FloatArray2D:
    ts, ys, errors, dt = implicit_euler(n)
    data: dict[str, FloatArray] = {'t': ts, 'y': ys, 'error': errors}
    data_frame: DataFrame = DataFrame(data).rename_axis('N', axis=0)
    max_error = errors.max()
    n_description: str = f'N from 0 to {n}'
    print(f'{n_description}\nMax error: {max_error:.6e}\n')
    print(f'{data_frame.T}\n\n' + '-' * 80 + '\n')
    solution_plot.plot(ts, ys, label = n_description)
    return array([n, log10(dt), log10(max_error)])

def calculate_slope(xs: FloatArray, ys: FloatArray) -> float:
    return (ys[-1] - ys[0]) / (xs[-1] - xs[0])

if __name__ == '__main__':
    # Data
    a: float = 1.05
    N: int = 28

    # Constants
    t_min: float = 0.0
    t_max: float = 2.0
    y0: float = 0.0

    t_gaps_count: int = 10

    # Fuctions and plot preparation
    t_tick_size: float = (t_max - t_min) / t_gaps_count
    t_ticks: FloatArray = arange(t_min, t_max + t_tick_size, t_tick_size)
    solution_plot: Axes = get_axes(
        'y(t) graph', [t_min, t_max], t_ticks, 't', 'y(t)',
    )

    exact_solution: ExactSolutionType = exact_solution_getter(a)
    implicit_euler: ImplicitEulerType = implicit_euler_getter(
        a, t_min, t_max, y0, exact_solution,
```

```python
    )

    # Executing implicit Euler
    error_plot_data: list[FloatArray] = []
    for n in [N, 2*N, 4*N]:
        error_plot_data.append(process(implicit_euler, n, solution_plot))

    # Drawing solution
    solution_plot.legend()
    if isinstance(fig := solution_plot.get_figure(), Figure):
        fig.show()

    # Calculating slope
    _, log10_dts, log10_max_errors = array(error_plot_data).T
    error_plot_slope: float = calculate_slope(log10_dts, log10_max_errors)
    error_plot_slope_text: str = f'Slope = {error_plot_slope:.6f}'

    # Drawing log_10(dt) to log_10(max_error) plot
    errors_plot: Axes = get_axes(
        'log_10(dt) to log_10(max_error)', None, None,
        'log_10(dt)', 'log_10(max_error)',
    )
    errors_plot.plot(
        log10_dts, log10_max_errors, marker = 'o',
        label = error_plot_slope_text,
    )
    for n, log10_dt, log10_max_error in error_plot_data:
        errors_plot.annotate(f'N = {int(n)}', (log10_dt, log10_max_error))

    errors_plot.legend()
    if isinstance(fig := errors_plot.get_figure(), Figure):
        fig.show()

    reset_option('display.float_format')
    reset_option('display.max_columns')
```

ex_3.py

```python
from matplotlib.pyplot import figure, Axes, Figure
from numpy import (square, sqrt, exp, float64, arange, abs, log10, array, zeros,
                   repeat, triu, eye)
from numpy.typing import NDArray
from pandas import DataFrame, set_option, reset_option
from typing import TypeAlias, Callable, Annotated, Literal

FloatArray: TypeAlias = Annotated[NDArray[float64], Literal[1]]
FloatArray2D: TypeAlias = Annotated[NDArray[float64], Literal[2]]
ExactSolutionType: TypeAlias = Callable[[float], float]
Solve2DegreeDifferentialEquationReturnType: TypeAlias = tuple[FloatArray, FloatArray, FloatArray, float]
Solve2DegreeDifferentialEquationType: TypeAlias = Callable[[int], Solve2DegreeDifferentialEquationReturnType]

set_option('display.float_format', lambda x: '%.6e' % x)
set_option('display.max_columns', 1000)

def exact_solution_getter(q: float, r: float, s: float) -> ExactSolutionType:
    def exact_solution(x: float) -> float:
        negative_half_q: float = -q / 2
        square_root: float = sqrt(square(q) - 4 * r)
        lambda_1: float = negative_half_q - square_root / 2
        lambda_2: float = negative_half_q + square_root / 2
        A: float = (exp(5 * lambda_2) - 1) / (exp(5 * lambda_1) - exp(5 * lambda_2))
        B: float = (exp(5 * lambda_1) - 1) / (exp(5 * lambda_2) - exp(5 * lambda_1))
        return s / r * (A * exp(lambda_1 * x) + B * exp(lambda_2 * x) + 1)
    return exact_solution

def lower_upper_decomposition(
        A: FloatArray2D,
) -> tuple[FloatArray2D, FloatArray2D]:
```

```
    n: int = A.shape[0]
    a: FloatArray2D = A.copy()
    for k in range(n - 1):
        akk: float = a[k][k]
        for i in range(k + 1, n):
            aux: float = a[i][k] / akk if akk else 0
            for j in range(k + 1, n):
                a[i][j] -= a[k][j] * aux
            a[i][k] = aux
    U: FloatArray2D = triu(a)
    L: FloatArray2D = a - U + eye(n)
    return L, U

def eliminate_forward(L: FloatArray2D, B: FloatArray) -> FloatArray:
    n: int = L.shape[0]
    b: FloatArray = B.copy()
    for k in range(n - 1):
        for i in range(k + 1, n):
            b[i] -= b[k] * L[i][k]
    return b

def substitute_backward(U: FloatArray2D, Y: FloatArray) -> FloatArray:
    n: int = U.shape[0]
    y: FloatArray = Y.copy()
    y[n-1] /= U[n-1][n-1]
    for i in range(n-2, -1, -1):
        s: float = 0.0
        for j in range(i+1, n):
            s += U[i][j] * y[j]
        y[i] -= s
        y[i] /= U[i][i]
    return y

def solve_2_degree_differential_equation_getter(
    q: float, r: float, s: float, x_min: float, x_max: float,
    y_for_x_min: float, y_for_x_max: float, exact_solution: ExactSolutionType,
) -> Solve2DegreeDifferentialEquationType:
    def solve_2_degree_differential_equation(
            N: int,
    ) -> Solve2DegreeDifferentialEquationReturnType:
        h: float = (x_max - x_min) / N

        A: FloatArray2D = zeros((N + 1, N + 1))
        A[0][0] = 1
        A[N][N] = 1

        B: FloatArray = repeat(s, N + 1)
        B[0] = y_for_x_min
        B[N] = y_for_x_max

        for i in arange(1, N):
            A[i][i - 1] = 1 / square(h) - q / 2 / h
            A[i][i] = r - 2 / square(h)
            A[i][i + 1] = 1 / square(h) + q / 2 / h

        L, U = lower_upper_decomposition(A)
        eliminated: FloatArray = eliminate_forward(L, B)
        ys: FloatArray = substitute_backward(U, eliminated)

        xs: list[float] = []
        errors = []
        for i in arange(0, N + 1):
            xs.append(x_min + h * i)
            errors.append(abs(exact_solution(xs[i]) - ys[i]))

        return array(xs), ys, array(errors), h
    return solve_2_degree_differential_equation

def get_axes(
    title: str, x_limits: list[float] | None,
```

```python
    x_ticks: FloatArray | None, x_label: str, y_label: str,
) -> Axes:
    axes: Axes = figure().add_subplot()
    axes.set_title(title)
    if x_limits is not None:
        axes.set_xlim(*x_limits)
    if x_ticks is not None:
        axes.set_xticks(x_ticks)
    axes.set_xlabel(x_label)
    axes.set_ylabel(y_label)
    axes.grid()
    return axes

def process(
    solve_2_degree_differential_equation: Solve2DegreeDifferentialEquationType,
    n: int, solution_plot: Axes,
) -> FloatArray2D:
    xs, ys, errors, h = solve_2_degree_differential_equation(n)
    data: dict[str, FloatArray] = {'x': xs, 'y': ys, 'error': errors}
    data_frame: DataFrame = DataFrame(data).rename_axis('N', axis=0)
    max_error = errors.max()
    n_description: str = f'N from 0 to {n}'
    print(f'{n_description}\nMax error: {max_error:.6e}\n')
    print(f'{data_frame.T}\n\n' + '-' * 80 + '\n')
    solution_plot.plot(xs, ys, label = n_description)
    return array([n, log10(h), log10(max_error)])

def calculate_slope(xs: FloatArray, ys: FloatArray) -> float:
    return (ys[-1] - ys[0]) / (xs[-1] - xs[0])

if __name__ == '__main__':
    # Data
    q: float = 0.47
    r: float = -0.84
    s: float = 2.37
    N: int = 28

    # Constants
    x_min: float = 0.0
    x_max: float = 5.0
    y_for_x_min: float = 0.0
    y_for_x_max: float = 0.0

    x_gaps_count: int = 10

    # Fuctions and plot preparation
    x_range: float = x_max - x_min
    x_tick_size: float = x_range / x_gaps_count
    x_ticks: FloatArray = arange(x_min, x_max + x_tick_size, x_tick_size)
    solution_plot: Axes = get_axes(
        'y(x) graph', [x_min, x_max], x_ticks, 'x', 'y(x)',
    )

    exact_solution: ExactSolutionType = exact_solution_getter(q, r, s)
    solve_2_degree_differential_equation: Solve2DegreeDifferentialEquationType = (
        solve_2_degree_differential_equation_getter(
            q, r, s, x_min, x_max, y_for_x_min, y_for_x_max, exact_solution,
        )
    )

    # Executing solve 2nd degree differential equation
    error_plot_data: list[FloatArray] = []
    for n in [N, 2*N, 4*N]:
        error_plot_data.append(
            process(solve_2_degree_differential_equation, n, solution_plot)
        )

    # Drawing solution
    solution_plot.legend()
    if isinstance(fig := solution_plot.get_figure(), Figure):
```

```python
        fig.show()

    # Calculating slope
    _, log10_hs, log10_max_errors = array(error_plot_data).T
    error_plot_slope: float = calculate_slope(log10_hs, log10_max_errors)
    error_plot_slope_text: str = f'Slope = {error_plot_slope:.6f}'

    # Drawing log_10(dt) to log_10(max_error) plot
    errors_plot: Axes = get_axes(
        'log_10(h) to log_10(max_error)', None, None,
        'log_10(h)', 'log_10(max_error)',
    )
    errors_plot.plot(
        log10_hs, log10_max_errors, marker = 'o',
        label = error_plot_slope_text,
    )
    for n, log10_h, log10_max_error in error_plot_data:
        errors_plot.annotate(f'N = {int(n)}', (log10_h, log10_max_error))

    errors_plot.legend()
    if isinstance(fig := errors_plot.get_figure(), Figure):
        fig.show()

    reset_option('display.float_format')
    reset_option('display.max_columns')
```