Министерство образования и науки РФ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа № 3

“**Методы Рунге-Кутты**”

по дисциплине «Численное моделирование динамических систем, описываемых обыкновенными дифференциальными уравнениями»

**Факультет:** ПМИ

**Группа:**  ПМ-92

**Студенты:**  Иванов В., Кутузов И.

**Преподаватель:**  Вагин Д. В.

Новосибирск

2021

**Условие**

Часть I

На трех сетках h = [0.1, 0.05, 0.025] решить задачу

y'=4ty

t=[0,1]

y(0)=1

с помощью классического явного четырехэтапного метода Рунге-Кутты.

Часть II

Решить задачу

y'=-25y+cos(t)+25sin(t)

t=[0,2]

y(0)=1

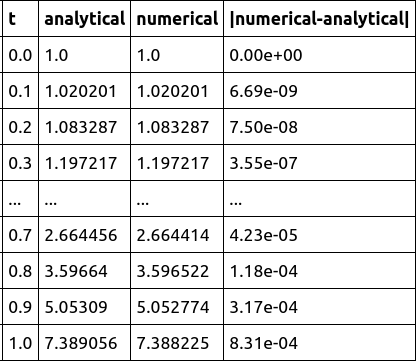
Простые методы Эйлера (явный и неявный), h=[0.1, 0.05, 0.025]

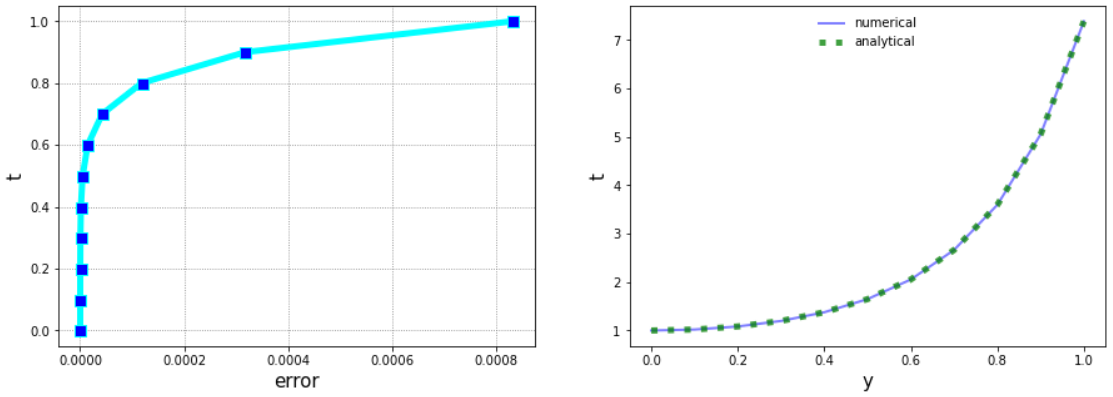
Модифицированный Эйлер и Трапеция, h=[0.2, 0.1, 0.05]

Р-К 4-го порядка, h=[0.5, 0.25, 0.125, 0.1]

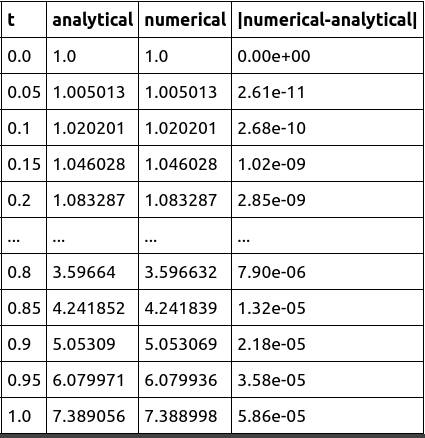
**Часть I (Рунге-Кутта)**

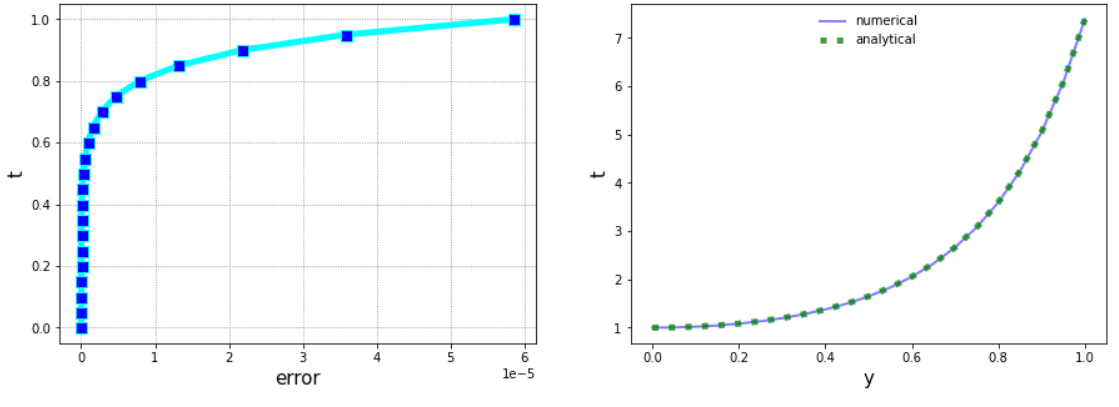
#### h = 0.1

****

****

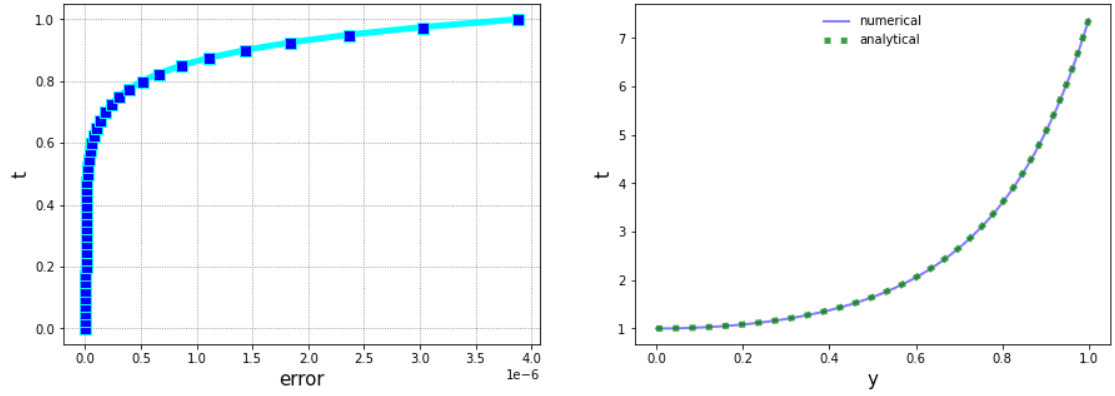
#### h = 0.05





#### h = 0.025

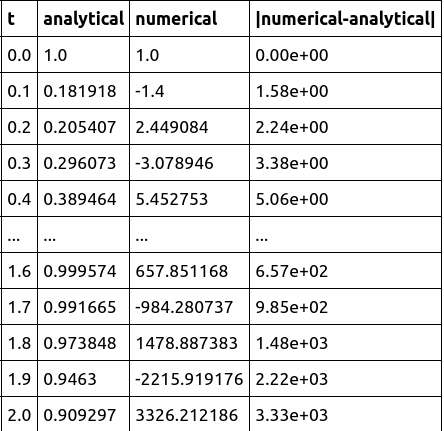
#### 

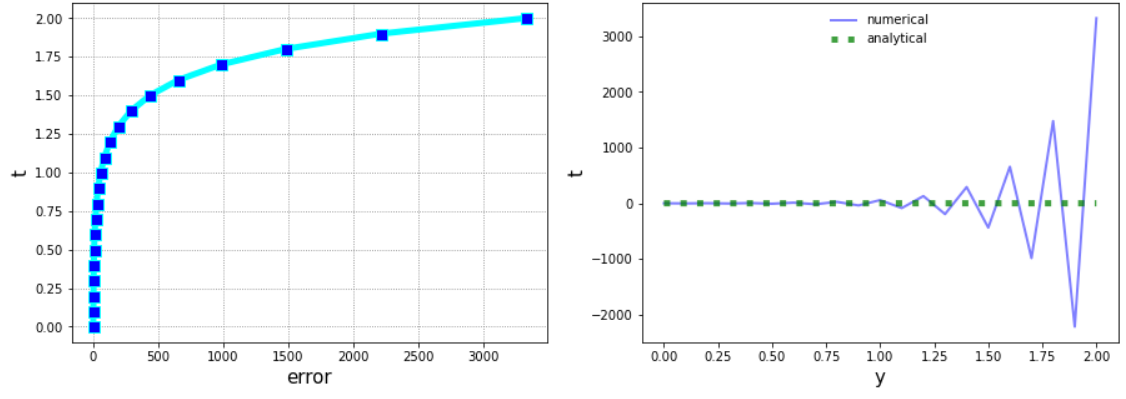


**Часть II**

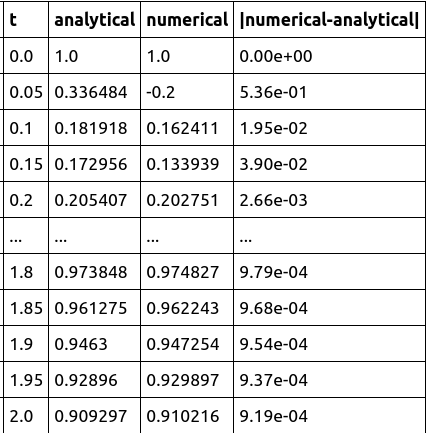
#### Явный Эйлер

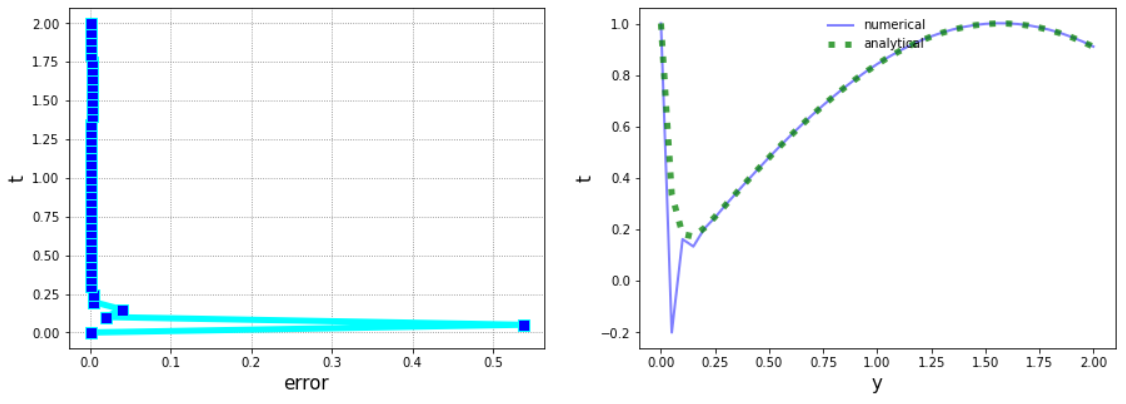
#### h = 0.1

****

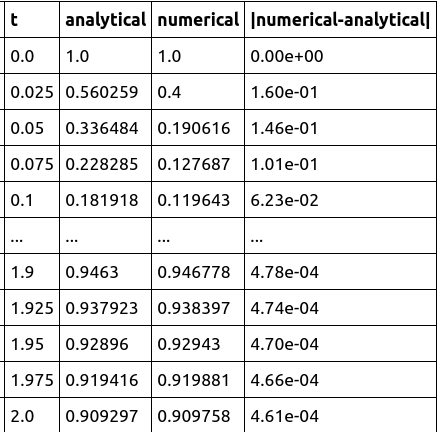
****

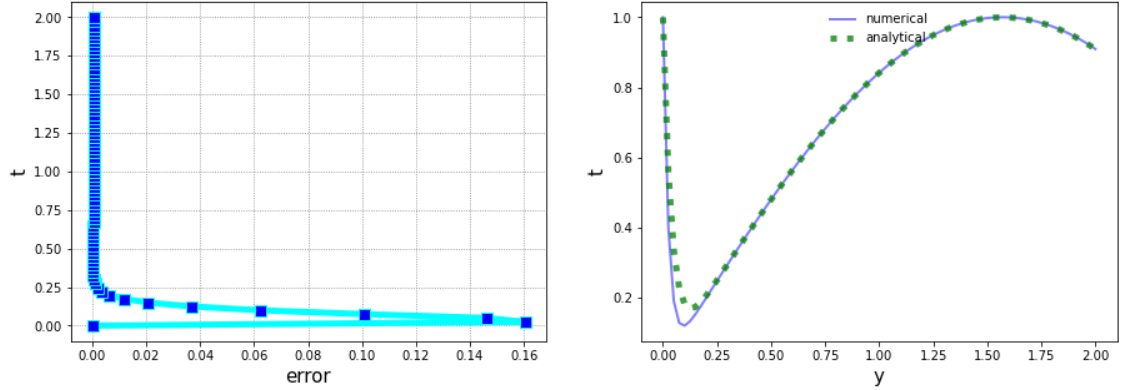
#### h = 0.05





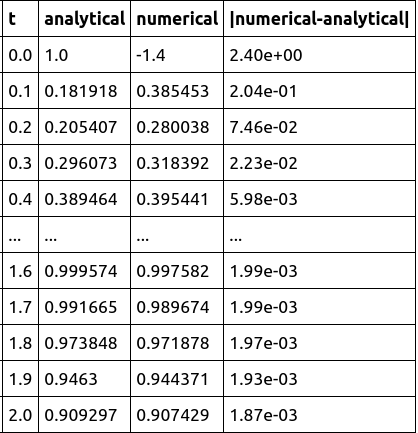
#### h = 0.025

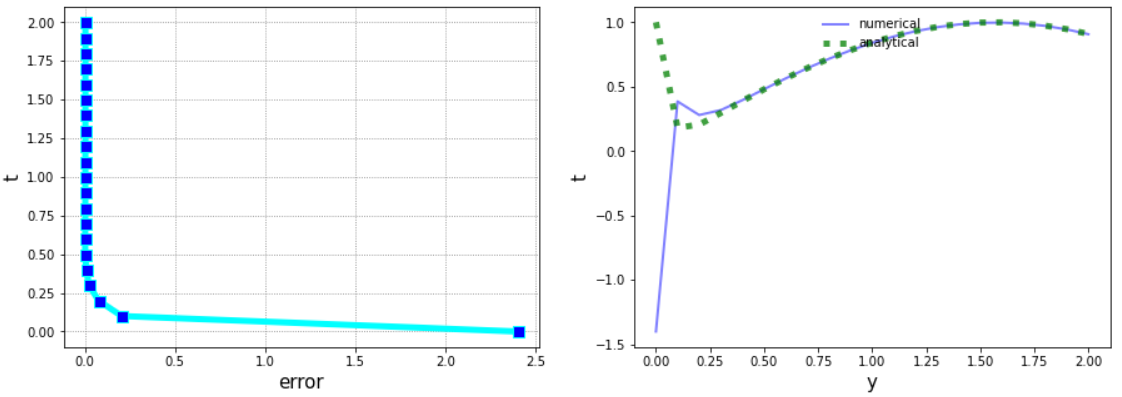




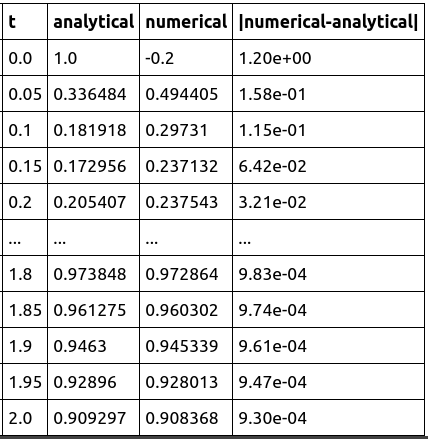
#### Неявный Эйлер (метод Ньютона)

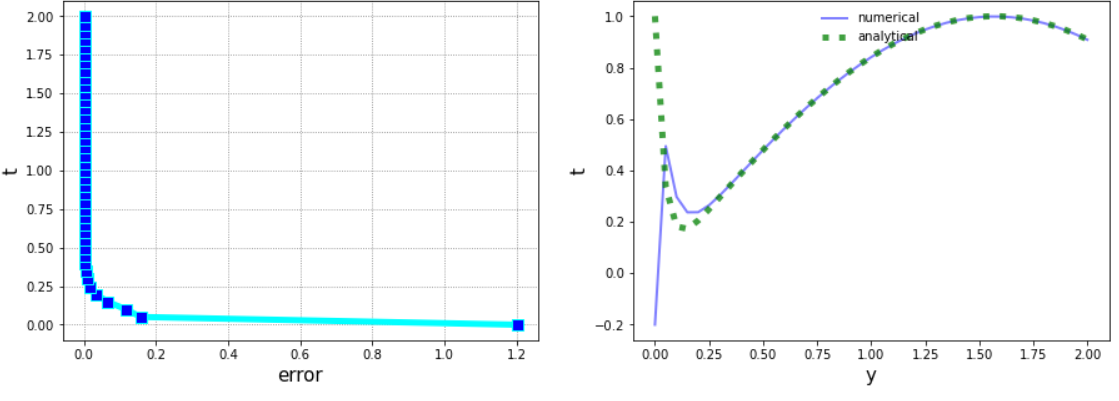
#### h = 0.1

****

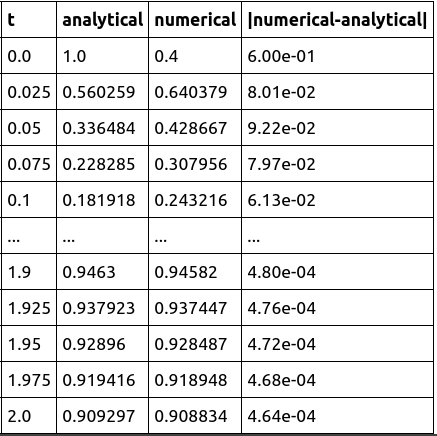
****

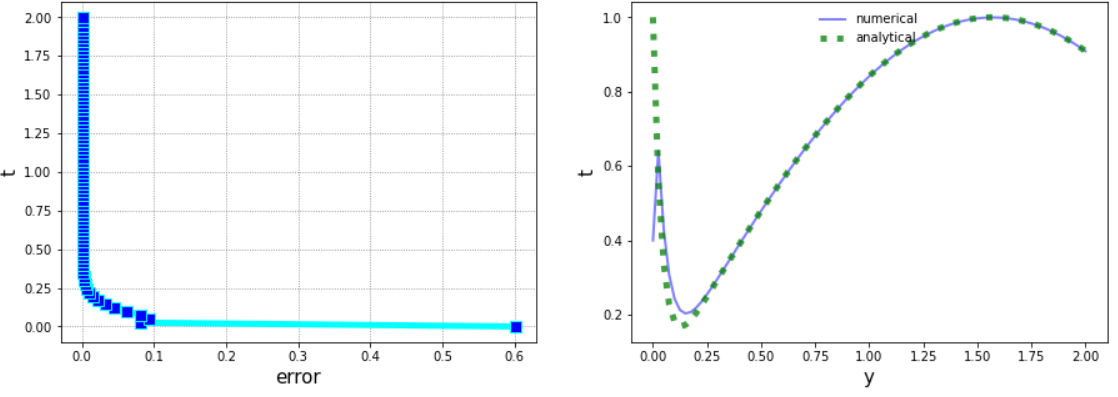
#### h = 0.05

****

****

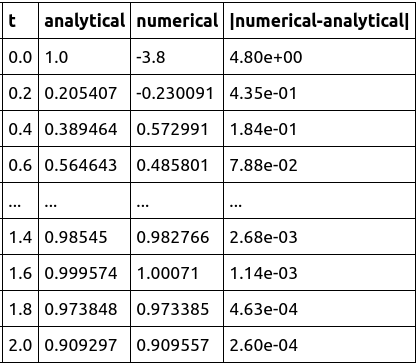
#### h = 0.025

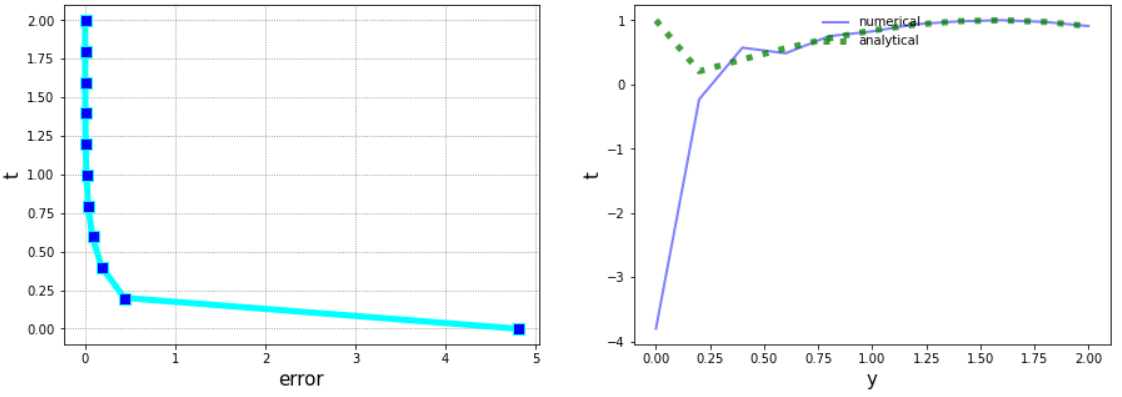
****

****

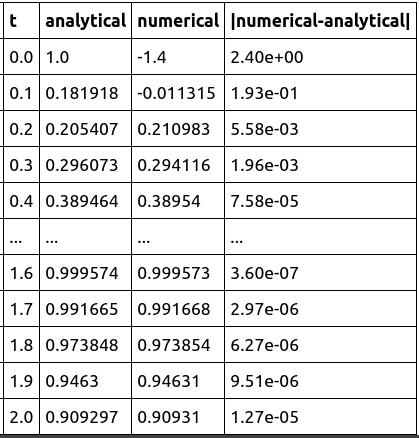
#### Трапеция (метод Ньютона)

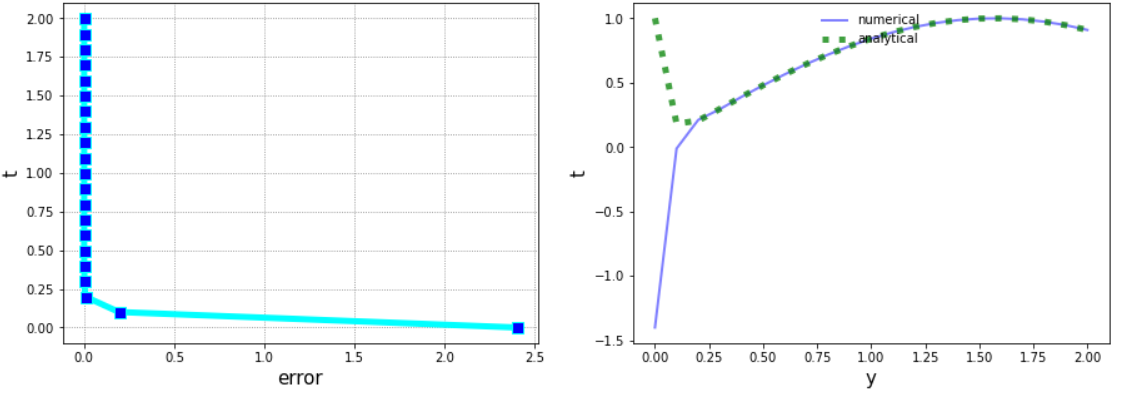
#### h = 0.2

****

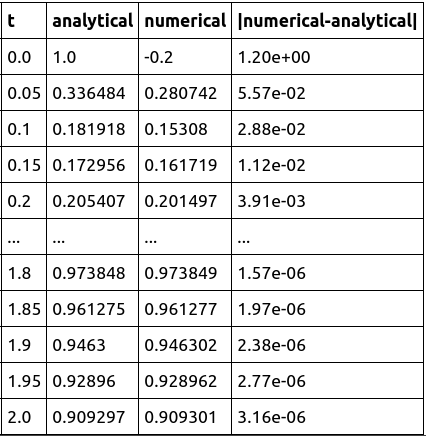
****

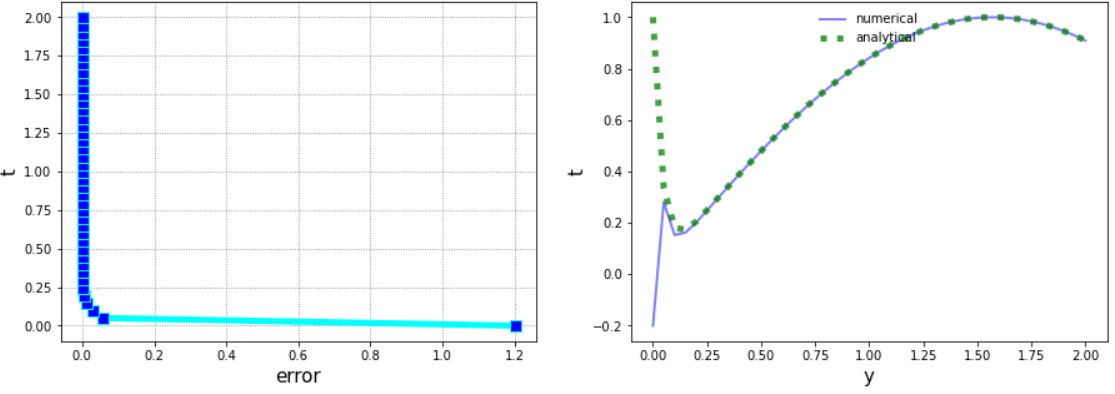
#### h = 0.1





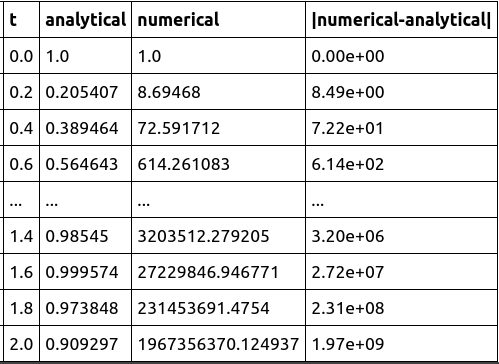
#### h = 0.05

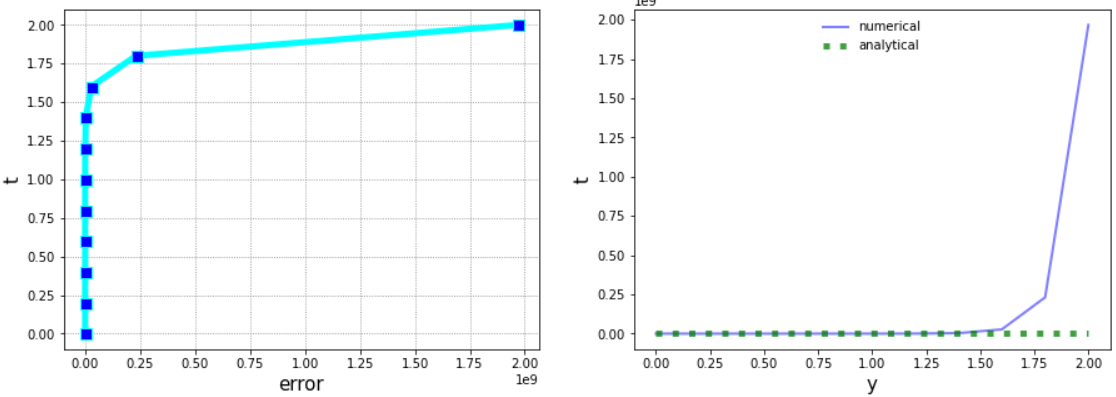




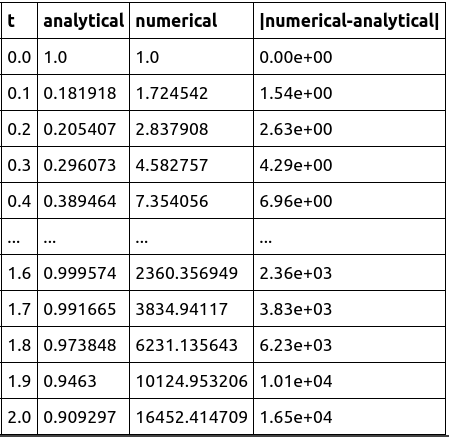
#### Модифицированный Эйлер

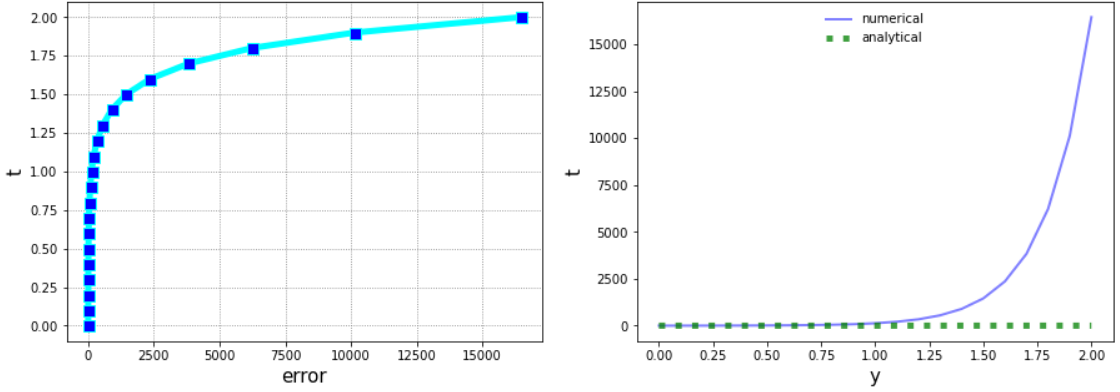
#### h = 0.2



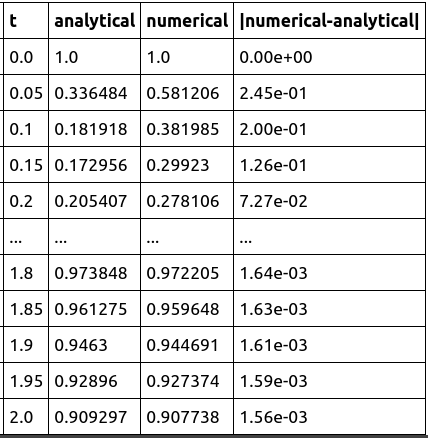


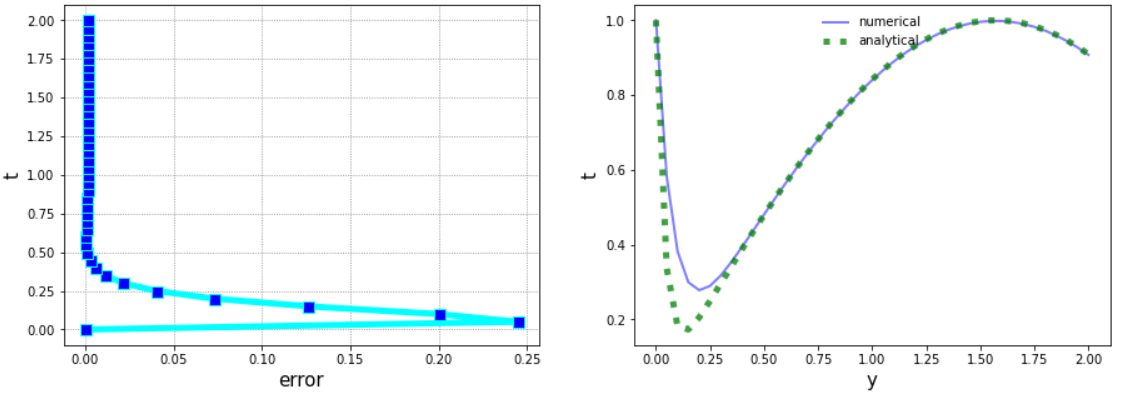
#### h = 0.1





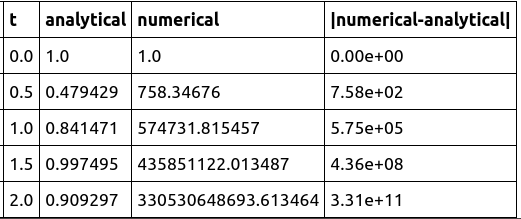
#### h = 0.05

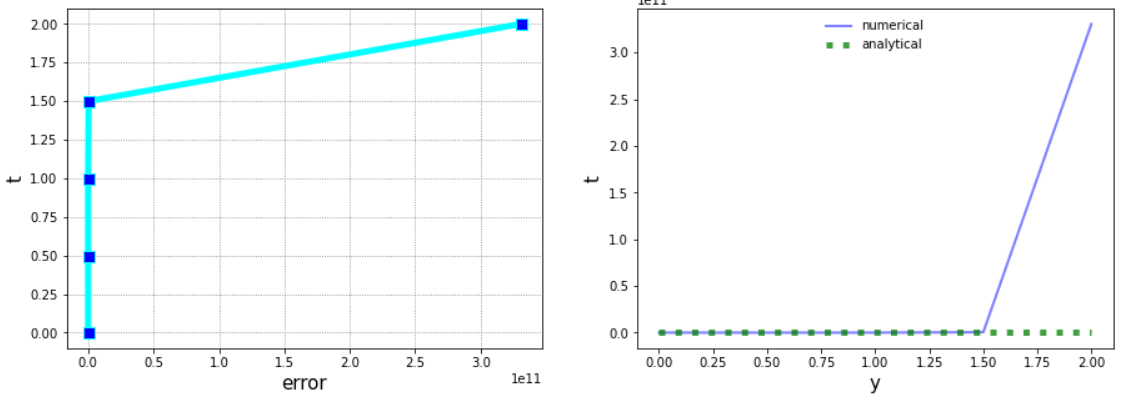




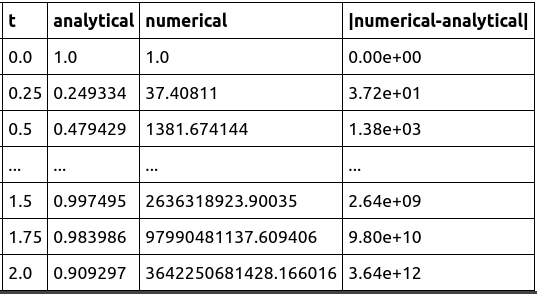
#### Рунге-Кутта

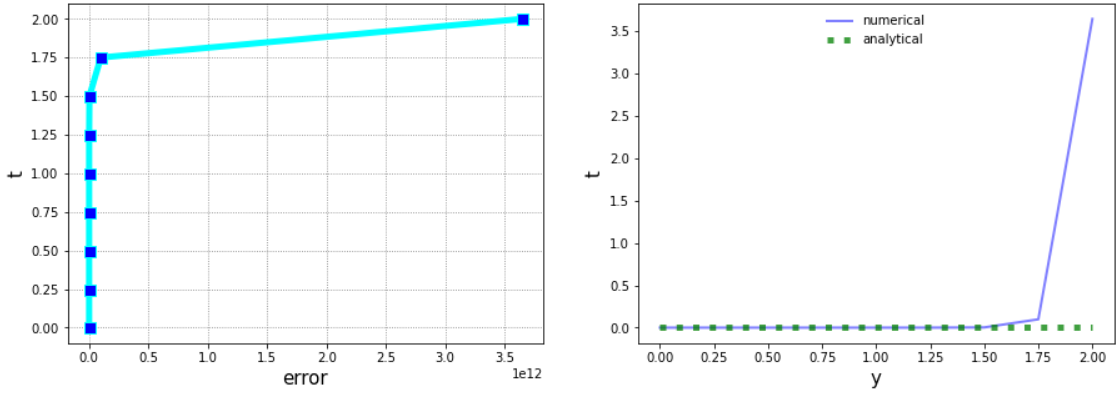
#### h = 0.5



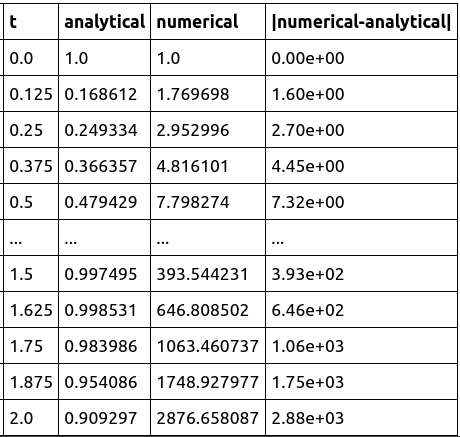


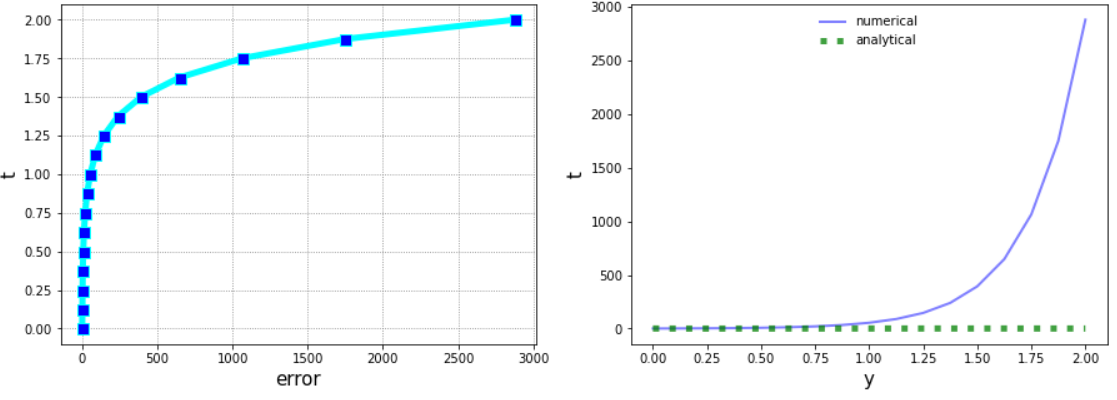
#### h = 0.25



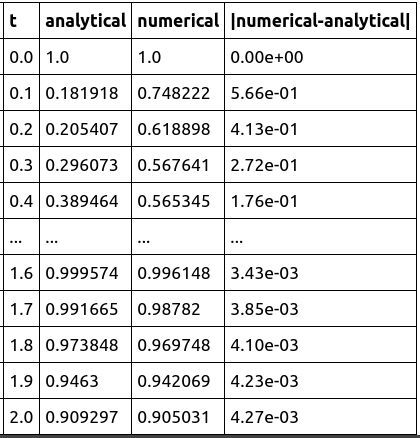


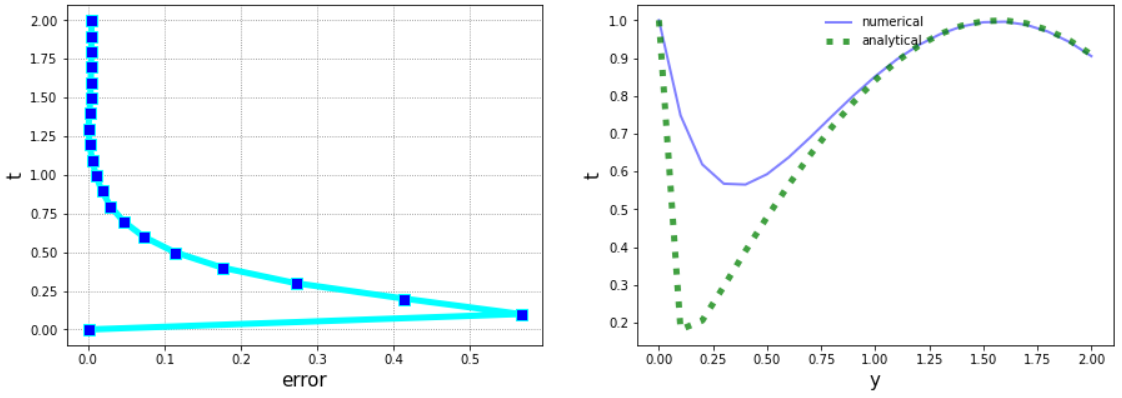
#### h = 0.125





#### h = 0.1





**Код программы**

**In [355]:**

**import numpy as np**

**import pandas as pd**

**from matplotlib import pyplot as plt**

**In [356]:**

***# helper functions***

**def make\_grid(h):**

**return np.arange(interval[0], interval[1], h)**

**def analytical(h):**

**analytical = []**

**grid = make\_grid(h)**

**for tn in grid:**

**yn\_1 = f\_solve(tn)**

**analytical.append(yn\_1)**

**analytical.append(f\_solve(grid[-1]+grid[1]))**

**return analytical**

**def make\_results(grid, analytical, numerical):**

**error = np.absolute(np.array(numerical) - np.array(analytical))**

**error\_scientific = []**

**for x in error:**

**error\_scientific.append('{:.2e}'.format(x))**

**df = pd.DataFrame([grid, analytical, numerical, error\_scientific]).T**

**df\_middle = pd.DataFrame(['...', '...', '...', '...']).T.rename(index={0: '...'})**

**df\_middle.columns = ['t', 'analytical', 'numerical', '|numerical-analytical|']**

**df.columns = ['t', 'analytical', 'numerical', '|numerical-analytical|']**

**df.loc[len(grid), 't'] = grid[-1]+grid[1]**

**make\_plots(grid, analytical, numerical, error)**

**return pd.concat([df.head(), df\_middle, df.tail()])**

**def make\_plots(grid, analytical, numerical, error):**

**fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,5))**

**grid = np.append(grid, grid[-1]+grid[1])**

**ax1.plot(error, grid,**

**marker='s', markersize=9, markerfacecolor='blue',**

**linestyle='-', linewidth=5,**

**color='cyan')**

**ax1.grid(color='grey', linestyle='dotted')**

**ax1.set\_xlabel('error', fontsize=15)**

**ax1.set\_ylabel('t', fontsize=15)**

**ax2.plot(grid, numerical, 'b-', label='numerical', alpha=0.50, linewidth=2)**

**ax2.plot(grid, analytical, 'g:', label='analytical', alpha=0.75, linewidth=5)**

**ax2.legend(loc='upper center', frameon=False)**

**ax2.set\_xlabel('y', fontsize=15)**

**ax2.set\_ylabel('t', fontsize=15)**

### 

# Part I

**In [357]:**

**interval = [0, 1]**

**def f(t, y):**

**return 4\*t\*y**

**def f\_solve(t):**

**return np.exp(2\*t\*t)**

**In [358]:**

**def runge\_kutta(h):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn)**

**for tn in grid:**

**kn1 = f(tn, yn)**

**kn2 = f(tn + h/2, yn + kn1\*h/2)**

**kn3 = f(tn + h/2, yn + kn2\*h/2)**

**kn4 = f(tn + h, yn + kn3\*h)**

**kn = (1/6)\*(kn1 + 2\*kn2 + 2\*kn3 + kn4)**

**yn\_1 = yn + h\*kn**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [359]:**

**runge\_kutta(0.1).drop([4, 6])**

**In [360]:**

**runge\_kutta(0.05)**

**In [361]:**

**runge\_kutta(0.025)**

# Part II

**In [362]:**

**interval = [0, 2]**

**def f(t, y):**

**return -25\*y+np.cos(t)+25\*np.sin(t)**

**def f\_solve(t):**

**return np.exp(-25\*t)\*(1+np.exp(25\*t)\*np.sin(t))**

### Explicit Euler

**In [363]:**

**def explicit\_euler(h):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn)**

**for tn in grid:**

**yn\_1 = yn + h\*f(tn, yn)**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [364]:**

**explicit\_euler(0.1)**

**In [365]:**

**explicit\_euler(0.05)**

**In [366]:**

**explicit\_euler(0.025)**

### Implicit Euler (fixed-point iteration)

**In [367]:**

**def implicit\_euler(h, eps):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn + h\*f(0.0, yn))**

**for tn in grid:**

**yk = yn + h\*f(tn, yn)**

**while True:**

**yk\_1 = yn + h\*f(tn + h, yk)**

**if abs(yk\_1-yk) < eps:**

**break**

**yk = yk\_1**

**yn\_1 = yn + h\*f(tn + h, yk\_1)**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [368]:**

**implicit\_euler(0.1, 1000000000000000000000)**

***# eps < 10^21 gives runtime error due to divergence***

**In [369]:**

**implicit\_euler(0.05, 1.6)**

***# eps < 1.6 gives runtime error due to divergence***

**In [370]:**

**implicit\_euler(0.025, 0.000000001)**

### Modified Euler

**In [371]:**

**def modified\_euler(h):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn)**

**for tn in grid:**

**yn\_1 = yn + (h/2)\*(f(tn, yn)+f(tn + h, yn + h\*f(tn, yn)))**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [372]:**

**modified\_euler(0.2).drop([4, 6])**

**In [373]:**

**modified\_euler(0.1)**

**In [374]:**

**modified\_euler(0.05)**

### Trapezoid (fixed-point iteration)

**In [375]:**

**def trapezoid(h, eps):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn + h\*f(0.0, yn))**

**for tn in grid:**

**yk = yn + h\*f(tn, yn)**

**while True:**

**yk\_1 = yn + h\*f(tn + h, yk)**

**if abs(yk\_1-yk) < eps:**

**break**

**yk = yk\_1**

**yn\_1 = yn + (h/2)\*(f(tn, yn) + f(tn + h, yk\_1))**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [376]:**

**trapezoid(0.2, 100000000000000000).drop([4, 6])**

***# eps < 10^17 gives runtime error due to divergence***

**In [377]:**

**trapezoid(0.1, 10000000000000000)**

***# eps < 10^16 gives runtime error due to divergence***

**In [378]:**

**trapezoid(0.05, 1.6)**

***# eps < 1.6 gives runtime error due to divergence***

### Runge-Kutta

**In [379]:**

**def runge\_kutta(h):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn)**

**for tn in grid:**

**kn1 = f(tn, yn)**

**kn2 = f(tn + h/2, yn + kn1\*h/2)**

**kn3 = f(tn + h/2, yn + kn2\*h/2)**

**kn4 = f(tn + h, yn + kn3\*h)**

**kn = (1/6)\*(kn1 + 2\*kn2 + 2\*kn3 + kn4)**

**yn\_1 = yn + h\*kn**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [380]:**

**runge\_kutta(0.5)[:5]**

**In [382]:**

**runge\_kutta(0.125)**

**In [383]:**

**runge\_kutta(0.1)**

**In [384]:**

**runge\_kutta(0.05)**

### Implicit Euler (Newton's method)

**In [385]:**

**def implicit\_euler(h, eps):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn + h\*f(0.0, yn))**

**for tn in grid:**

**yk = yn + h\*f(tn, yn)**

**while True:**

**yk\_1 = yk - (yk - yn - h\*f(tn + h, yk)) / (1 + 25\*h)**

**if abs(yk\_1-yk) < eps:**

**break**

**yk = yk\_1**

**yn\_1 = yn + h\*f(tn + h, yk\_1)**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [386]:**

**implicit\_euler(0.1, 0.001)**

**In [387]:**

**implicit\_euler(0.05, 0.001)**

**In [388]:**

**implicit\_euler(0.025, 0.0001)**

### Trapezoid (Newton's method)

**In [389]:**

**def trapezoid(h, eps):**

**yn = 1.0**

**numerical = []**

**grid = make\_grid(h)**

**numerical.append(yn + h\*f(0.0, yn))**

**for tn in grid:**

**yk = yn + h\*f(tn, yn)**

**while True:**

**yk\_1 = yk - (yk - yn - (h/2)\*(f(tn, yn) + f(tn + h, yk))) / (1 + 12.5\*h)**

**if abs(yk\_1-yk) < eps:**

**break**

**yk = yk\_1**

**yn\_1 = yn + (h/2)\*(f(tn, yn) + f(tn + h, yk\_1))**

**numerical.append(yn\_1)**

**yn = yn\_1**

**return make\_results(grid, analytical(h), numerical)**

**In [390]:**

**trapezoid(0.2, 0.001).drop([4, 6])**

**In [391]:**

**trapezoid(0.1, 0.001)**

**In [392]:**

**trapezoid(0.05, 0.001)**