In [1625]:

```
from sympy import *
```

In [1626]:

```
from icecream import ic
import pandas as pd
```

In [1627]:

```
x = symbols('x')
y = symbols("y", cls=Function)  # = Function('y')
```

In [1628]:

```
points = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
aEquation = Eq(y(x).diff(x), 2 * y(x) + exp(x) - x)
aInitialCondition = (0, 1/4)  # {y(0): 1 / 4}
bEquation = Eq(y(x).diff(x), x - 1 + (x + 1)*y(x))
bInitialCondition = (0, 0)  # {y(0): 0}
```

In [1629]:

```
def dsolveExactSolution(equation, initialCondition=(0, 0), pointValues=()):
    initialCondition = {y(initialCondition[0]): initialCondition[1]}
    solution = dsolve(equation, y(x), ics=initialCondition)
    solution = solution.simplify()
    values = [round(solution.rhs.subs(x, p).evalf(10), 5) for p in pointValues]
    return solution, values
```

In [1630]:

```
a1Solution, a1Values = dsolveExactSolution(aEquation, aInitialCondition, points)
a1Solution, a1Values
```

Out[1630]:

```
(Eq(y(x), x/2 + exp(2*x) - exp(x) + 1/4),
 [0.25000, 0.62042, 1.18372, 2.04800, 3.37749, 5.42077])
```

In [1631]:

```
b1Solution, b1Values = dsolveExactSolution(bEquation, bInitialCondition, points)
b1Solution, b1Values
```

Out[1631]:

```
(Eq(y(x), (1 + sqrt(2)*sqrt(pi)*exp(1/2)*erf(sqrt(2)/2))*exp(x*(x + 2)/2) - sqrt(2)*sqrt(pi
)*exp(x**2/2 + x + 1/2)*erf(sqrt(2)*(x + 1)/2) - 1),
 [0.0, -0.20283, -0.42446, -0.69114, -1.04407, -1.55268])
```

In [1632]:

```
approximationPrecision = 10
```

In [1633]:

```
def approximateAnalyticalSolution(equation, initialCondition=(0, 0), Nn=5, pointValues=())
:
    if Nn < 1:
        raise Exception("Wymagana minimum precyzja 1 pochodnej")
    func = equation.rhs
    x0 = initialCondition[0]
    y0 = initialCondition[1]
    # ic(func, x0, y0)
    # Wartosc y0
    yValues = [y0]
    # Pierwsza pochodna y(1)
```

```
    yPrim = func.subs(y(x), y0).subs(x, x0)
    yValues.append(yPrim)
    # Iteracyjnie obliczanie kolejnych pochodnych y(n)
    for n in range(2, Nn+1):
        func = diff(func, x)
        # Używanie poprzednich wartości pochodnych jako zamienników
        subsDict = {x: x0, y(x): y0}
        for i in range(1, n):
            subsDict[diff(y(x), x, i)] = yValues[i]
        yn = func.subs(subsDict)
        yValues.append(yn)
    # Podstawienie Y(k)(x0) do wzoru
    taylorSeries = sum(yValues[k] / factorial(k) * ((x - x0) ** k) for k in range(Nn+1))
    # Podstawienie punktow do wzoru
    approximationSolutions = [round(taylorSeries.subs(x, p).evalf(10), 5) for p in pointVa
lues]
    return taylorSeries, approximationSolutions
```

In [1634]:

```
a2Solution, a2Values = approximateAnalyticalSolution(aEquation, aInitialCondition, approxim
ationPrecision, points)
a2Solution, a2Values
```

Out[1634]:

```
(0.000281911375661376*x**10 + 0.00140817901234568*x**9 + 0.00632440476190476*x**8 + 0.02519
84126984127*x**7 + 0.0875*x**6 + 0.258333333333333*x**5 + 0.625*x**4 + 1.16666666666667*x**
3 + 1.5*x**2 + 1.5*x + 0.25,
 [0.25000, 0.62042, 1.18372, 2.04800, 3.37749, 5.42071])
```

In [1635]:

```
b2Solution, b2Values = approximateAnalyticalSolution(bEquation, bInitialCondition, approxim
ationPrecision, points)
b2Solution, b2Values
```

Out[1635]:

```
(-71*x**10/90720 - 43*x**9/18144 - 11*x**8/2016 - x**7/63 - x**6/36 - x**5/12 - x**4/12 - x
**3/3 - x,
 [0, -0.20283, -0.42446, -0.69114, -1.04403, -1.55226])
```

In [1636]:

```
def numericalSolution(equation, initialCondition=(0, 0)):
    func = equation.rhs
    x0 = initialCondition[0]
    y0 = initialCondition[1]
    yValue = y0
    # akurat punkty roznia sie o 0.2 i jest 6 punktow
    h = 0.2
    pCount = 6

    yValues = [y0]
    for k in range(1, pCount):
        xK = x0 + (k-1) * h
        f = func.subs({x: xK, y(x): yValue})
        yValue = yValue + h * f
        yValues.append(yValue)
    return [round(yValue, 5) for yValue in yValues]
```

In [1637]:

```
a3Values = numericalSolution(aEquation, aInitialCondition)
a3Values
```

Out[1637]:

```
[0.25, 0.55000, 0.97428, 1.58236, 2.45972, 3.72872]
```

In [1638]:

```
b3Values = numericalSolution(bEquation, bInitialCondition)
```

```
b3Values
```

Out[1638]:

```
[0, -0.20000, -0.40800, -0.64224, -0.92776, -1.30175]
```

In [1639]:

```
def createTable(tableName, columnHeaders, rowHeaders, tableValues):
    columns = [f"x{i} = {v}" for i, v in enumerate(columnHeaders)]
    table = pd.DataFrame(tableValues, columns=columns, index=rowHeaders)
    table.columns.name = tableName
    return table
```

In [1640]:

```
aTable = createTable("(a)", points, ["RD", f"RA({approximationPrecision})", "RN"], [a1Valu
es, a2Values, a3Values])
aTable
```

Out[1640]:

(a)	x0 = 0.0	x1 = 0.2	x2 = 0.4	x3 = 0.6	x4 = 0.8	x5 = 1.0
RD	0.25000	0.62042	1.18372	2.04800	3.37749	5.42077
RA(10)	0.25000	0.62042	1.18372	2.04800	3.37749	5.42071
RN	0.25	0.55000	0.97428	1.58236	2.45972	3.72872

In [1641]:

```
bTable = createTable("(b)", points, ["RD", f"RA({approximationPrecision})", "RN"], [b1Valu
es, b2Values, b3Values])
bTable
```

Out[1641]:

(b)	x0 = 0.0	x1 = 0.2	x2 = 0.4	x3 = 0.6	x4 = 0.8	x5 = 1.0
RD	0.0	-0.20283	-0.42446	-0.69114	-1.04407	-1.55268
RA(10)	0	-0.20283	-0.42446	-0.69114	-1.04403	-1.55226
RN	0	-0.20000	-0.40800	-0.64224	-0.92776	-1.30175