In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

sns.set_theme(style="darkgrid")
```

1

a

Finding a polynomial of $\deg P(t) \leq 2$ that exactly goes throught the three first points

In [ ]:
```
t = np.arange(4)
y = np.array([1, 2, 7, 5])
```

In [ ]:
```
div_diff = lambda x_i, x_j, y_i, y_j: (y_i - y_j) / (x_i - x_j)

results = np.zeros((3, 4))

results[:, 0] = t[:3]
results[:, 1] = y[:3]

results[0, 2] = div_diff(results[0, 0], results[1, 0], results[0, 1], res
results[1, 2] = div_diff(results[1, 0], results[2, 0], results[1, 1], res
results[0, 3] = div_diff(results[2, 0], results[0, 0], results[1, 2], res
print(results)
```

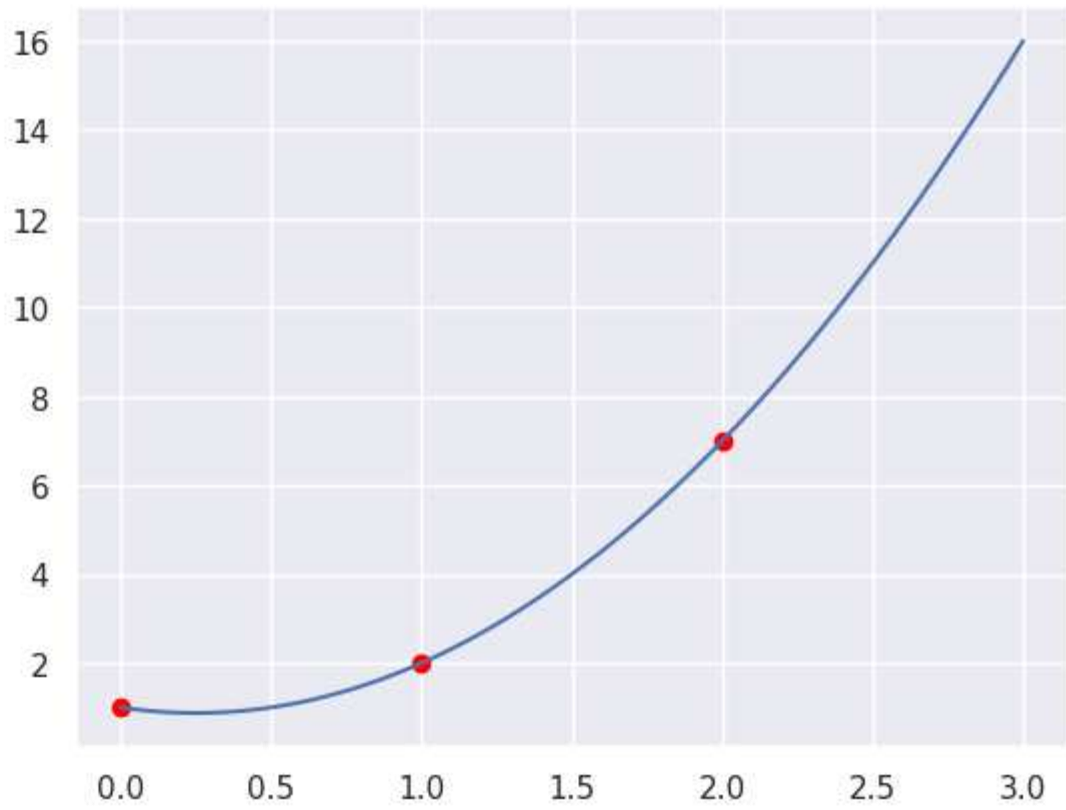
```
[[0. 1. 1. 2.]
 [1. 2. 5. 0.]
 [2. 7. 0. 0.]]
```

From the above calculations the top row from its first element is the coefficients of the polynomial

In [ ]:
```
P_2 = lambda t: results[0, 1] + results[0, 2] * (t - results[0, 0]) + res
```

Therefore the polynomial is $P_2(t) = 2t^2 - 2t + 1$, verifying the result:

In [ ]:
```
plt.scatter(t[:3], y[:3], color="red")
t_arr = np.linspace(0, 3, 100)
plt.plot(t_arr, P_2(t_arr))
plt.show()
```



This line goes through all three points exactly

b

Now we find a polynomial of degree 2 that fit the four points provided. It should minimize the sum $\sum_{i=1}^{4}(S(t_i)-y_i)^2$

Therefore using,

$$\begin{aligned} S(t) &= at^2+bt+c \\ S(0) &= 0+0+c=1 \\ S(1) &= a+b+c=2 \\ S(2) &= 4a+2b+c=7 \\ S(3) &= 9a+3b+c=5 \\ \implies Ax=b &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 7 \\ 5 \end{bmatrix} \end{aligned}$$

Now we use normal equations to find the least squares solution:

$$A^TA\hat{x}=A^Tb$$

In [ ]:
```python
A = np.array([
    [0, 0, 1],
    [1, 1, 1],
    [4, 2, 1],
    [9, 3, 1]
])

b = np.array([1, 2, 7, 5])

lhs = A.T @ A
rhs = A.T @ b

x = np.linalg.solve(lhs, rhs)
print(x)
```

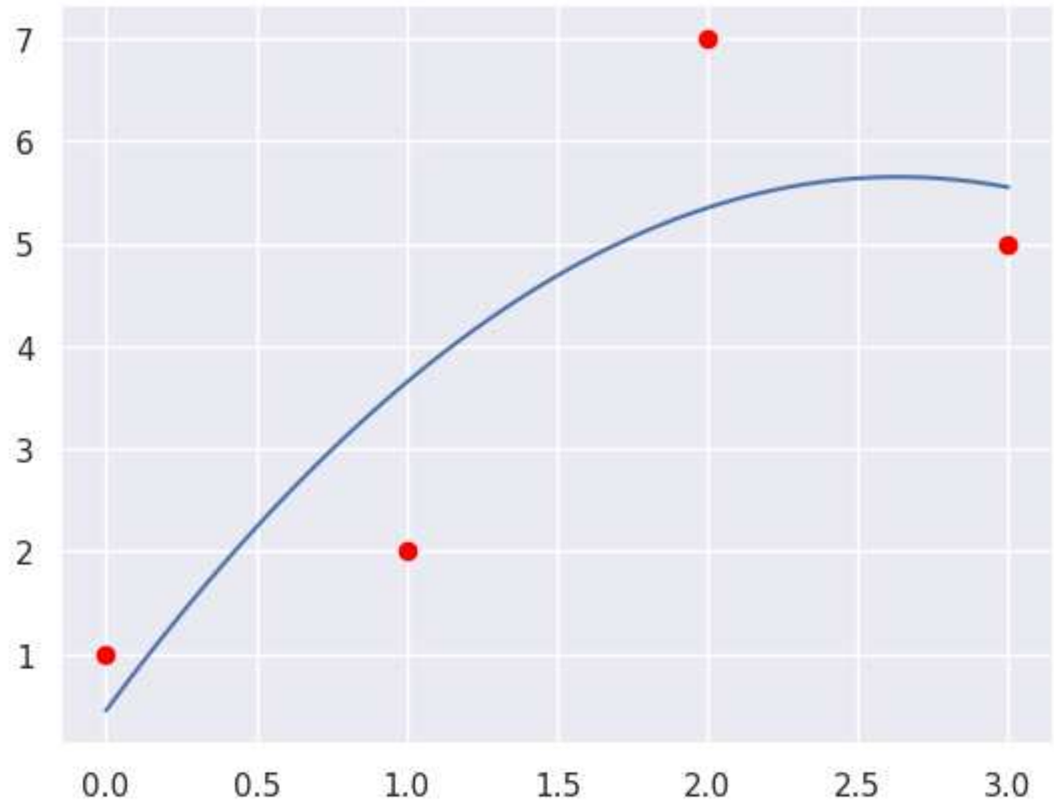
```
[-0.75  3.95  0.45]
```

The solution to this system is $a = -0.75, b = 3.95, c = 0.45$, resulting in,

$$S(t) = -0.75t^2 + 3.95t + 0.45$$

Verifying the results:

In [ ]:
```python
plt.scatter(t, y, color="red")

# Construct function from coefficients
S_t = lambda t: x[0] * t**2 + x[1] * t + x[2]

plt.plot(t_arr, S_t(t_arr))
plt.show()
```



This curve is interpolating the datapoints well.

C

Given the matrix $B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ we are to find the reduced QR factorization of B, that is we express B as $B = QR$, where $Q = \begin{bmatrix} q_1 & q_2 \end{bmatrix}$

In [ ]:
```
B = np.array([[1, 1, 1, 1], [0, 1, 2, 3]]).T

# First column of B
y_1 = A_1 = B[:, 0]
A_2 = B[:, 1]
r_11 = np.linalg.norm(y_1)
q_1 = y_1 / r_11

# Second column of B
y_2 = A_2 - q_1 * (q_1.T @ A_2)
r_22 = np.linalg.norm(y_2)
q_2 = y_2 / r_22

# Results
Q = np.array([q_1, q_2]).T
print(f"Q = \n{Q}")

r_12 = q_1.T @ A_2
R = np.array([[r_11, r_12], [0, r_22]])
print(f"R = \n{R}")
```

```
Q =
[[ 0.5        -0.67082039]
 [ 0.5        -0.2236068 ]
 [ 0.5         0.2236068 ]
 [ 0.5         0.67082039]]
R =
[[2.         3.        ]
 [0.         2.23606798]]
```

The results are

$$B = \begin{bmatrix} 0.5 & -0.67082039 \\ 0.5 & -0.2236068 \\ 0.5 & 0.2236068 \\ 0.5 & 0.67082039 \end{bmatrix}$$

Let us verify that this makes sense by checking $B = QR$

In [ ]:
```
try:
    assert np.allclose(Q @ R, B)
except AssertionError:
    print("Q @ R != B")
else:
    print("Q @ R == B")
```

```
Q @ R == B
```

d

Are going to find a polynomial of $\deg P_1(t) \leq 1$

To do this we can use a property of the orthogonal matrix Q, that is $Q^{-1} = Q^T$, and Q must of course be square for this to be true.

We begin with what we know:

$$\begin{aligned} B &= QR \\ Bx &= b \\ \implies QRx &= b \\ Q^{-1}QRx &= Q^{-1}b \\ Rx &= Q^Tb \\ R^{-1}Rx &= R^{-1}Q^Tb \\ x &= R^{-1}Q^Tb \end{aligned}$$

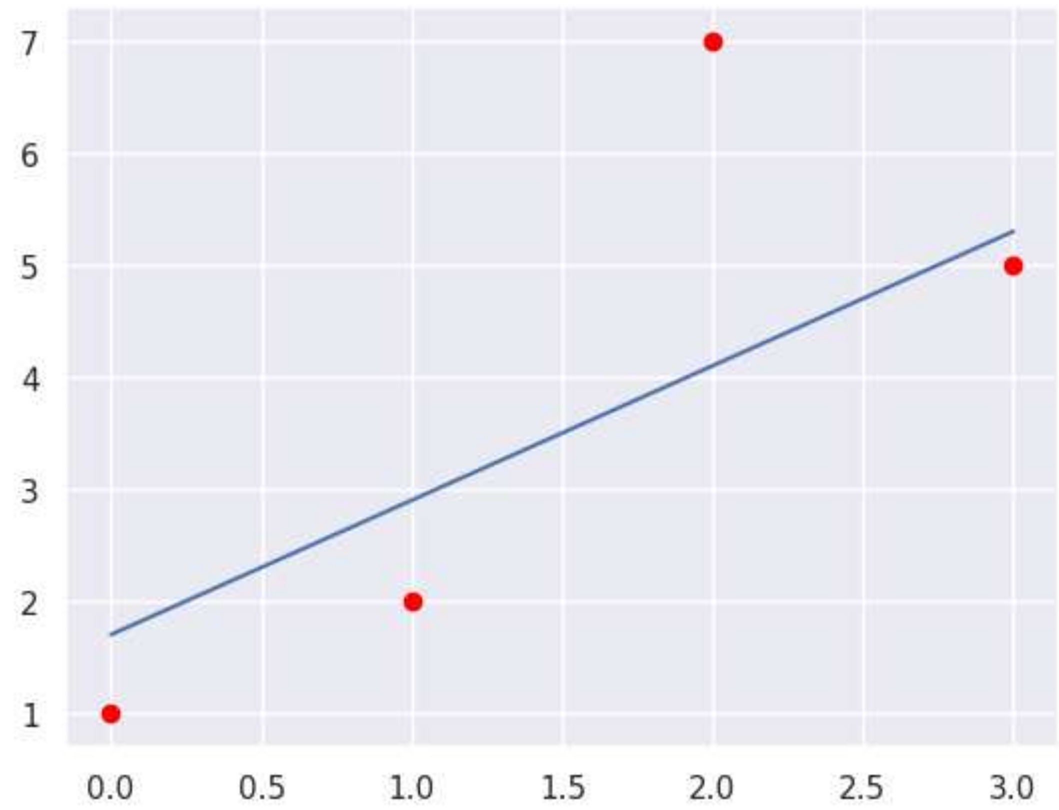
where $x = \begin{bmatrix} b \\ a \end{bmatrix}$ and we have our polynomial $U(t) = ax + b$

In [ ]:
```
x = np.linalg.inv(R) @ Q.T @ b

print(f"coefficients: b = {x[0]}, a = {x[1]}")
```

```
coefficients: b = 1.2000000000000002, a = 1.7
```

Therefore we get the polynomial $U(t) = 1.2x + 1.7$. Verifying our results:

In [ ]:
```
U = lambda t: x[0] * t + x[1]

plt.scatter(t, y, color="red")
plt.plot(t_arr, U(t_arr))
plt.show()
```



This line looks to be the best fit degreee 1 polynomial that fits all datapoints.

Problem 2

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^{\pi} e^{-t^2}$$

a

Finding the first 5 derivatives of $\operatorname{erf}(x)$ evaluated at $x = 0$:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^{x} e^{-t^2}\,dt \implies \operatorname{erf}(0) = 0$$

$$\frac{d}{dx}\int_a^x f(t)\,dt = f(x) \quad \text{(fundamental theorem of calculus)}$$

$$\operatorname{erf}'(x) = \frac{2}{\sqrt{\pi}} e^{-x^2} \implies \operatorname{erf}'(0) = \frac{2}{\sqrt{\pi}}$$

$$\operatorname{erf}^{(2)}(x) = -\frac{4x}{\sqrt{\pi}} e^{-x^2} \implies \operatorname{erf}^{(2)}(0) = 0$$

$$\operatorname{erf}^{(3)}(x) = -\frac{4}{\sqrt{\pi}} e^{-x^2} + \frac{16}{\sqrt{\pi}} x e^{-x^2} \implies \operatorname{erf}^{(3)}(0) = -\frac{4}{\sqrt{\pi}}$$

$$\operatorname{erf}^{(4)}(x) = \frac{8}{\sqrt{\pi}} x e^{-x^2} + \frac{16}{\sqrt{\pi}} x e^{-x^2} - \frac{16}{\sqrt{\pi}} x^3 e^{-x^2} \implies \operatorname{erf}^{(4)}(0) = 0$$

$$\operatorname{erf}^{(5)}(x) = \frac{8}{\sqrt{\pi}} e^{-x^2} - \frac{16}{\sqrt{\pi}} x^2 e^{-x^2} + \frac{16}{\sqrt{\pi}} e^{-x^2} - \frac{32}{\sqrt{\pi}} x^2 e^{-x^2} - \frac{48}{\sqrt{\pi}} x^2 e^{-x^2} +$$

This means we can construct a Taylor-polynomial of degree 5, even an Maclauring polynomial, that is the special case where the factor c in the Taylor polynomial is 0:

$$T_5(x) = \operatorname{erf}(0) + \operatorname{erf}'(0)(x-0) + \frac{\operatorname{erf}^{(2)}(0)}{2!}(x-0)^2 + \frac{\operatorname{erf}^{(3)}(0)}{3!}(x-0)^3 + \frac{\operatorname{erf}^{(4)}(0)}{4!}$$

$$T_5(x) = \frac{2}{\sqrt{\pi}} x - \frac{4}{6\sqrt{\pi}} x^3 + \frac{1}{5\sqrt{\pi}} x^5$$

We see from the derivatives that the multiplicity of erf for the root $x = 0$ is 1.

b

Given points $(x_i, y_i), i = 1, \ldots, 5$, where $x_1 = 0, x_2 = \frac{1}{2}, x_3 = 1, x_4 = \frac{3}{2}, x_5 = 2$, and $y_i = e^{-x_i^2}$:

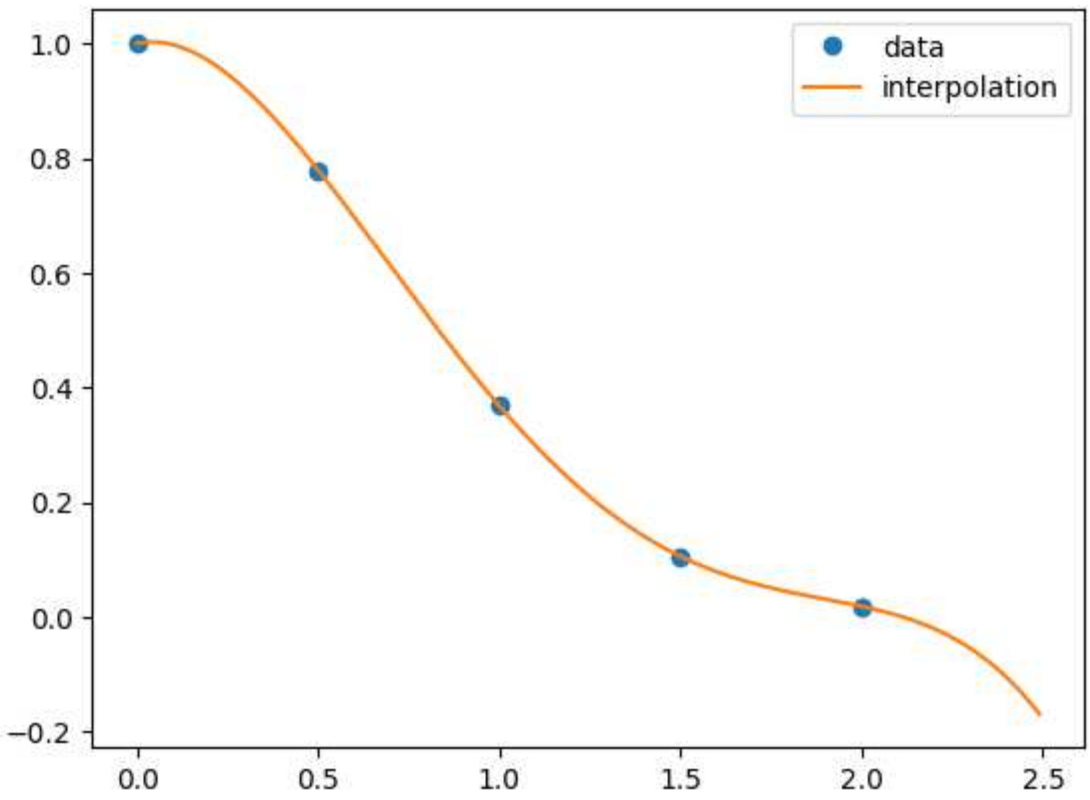
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 2.5, 0.5)
y = np.exp(-x**2)
res = np.hstack((x[:,None], y[:,None]))
print(res)
```

```
[[0.         1.        ]
 [0.5        0.77880078]
 [1.         0.36787944]
 [1.5        0.10539922]
 [2.         0.01831564]]
```

Are now to find a degree 4 interpolating polynomial for the provided points. I am going to use Newtons divided differences

In [ ]:
```
div_diff = lambda x_j, x_i, y_j, y_i: (y_i - y_j) / (x_i - x_j)

results = np.zeros((x.shape[0], x.shape[0] + 1))
results[:, :2] = res

results[0, 2] = div_diff(results[0, 0], results[1, 0], results[0, 1], res
results[1, 2] = div_diff(results[1, 0], results[2, 0], results[1, 1], res
results[2, 2] = div_diff(results[2, 0], results[3, 0], results[2, 1], res
results[3, 2] = div_diff(results[3, 0], results[4, 0], results[3, 1], res

results[0, 3] = div_diff(results[0, 0], results[2, 0], results[0, 2], res
results[1, 3] = div_diff(results[1, 0], results[3, 0], results[1, 2], res
results[2, 3] = div_diff(results[2, 0], results[4, 0], results[2, 2], res

results[0, 4] = div_diff(results[0, 0], results[3, 0], results[0, 3], res
results[1, 4] = div_diff(results[1, 0], results[4, 0], results[1, 3], res

results[0, 5] = div_diff(results[0, 0], results[4, 0], results[0, 4], res

coefficients = results[0, 1:]

print(f"coefficients: {coefficients}")

inter_poly = lambda x: results[0, 1] + results[0, 2] * (x - results[0, 0]

interp_x = np.arange(0, 2.5, 0.01)
interp_y = np.array([inter_poly(int_x) for int_x in interp_x])

plt.plot(x, y, 'o', label='data')
plt.plot(interp_x, interp_y, label='interpolation')
plt.legend()
plt.show()
```

```
coefficients: [ 1.         -0.44239843 -0.37944425  0.45088433 -0.2074718
3]
```



In the above python cell I have computed the coefficients of the polynomial that interpolates the given points. And verified that the function does interpolate the points.

The anti-derivative of $P_4(x)$ we can find as:

$$\begin{aligned}P_4(x) &= c_1 + c_2(x-x_1) + c_3(x-x_1)(x-x_2) + c_4(x-x_1)(x-x_2)(x-x_3)\\&+ c_5(x-x_1)(x-x_2)(x-x_3)(x-x_4), \quad \text{using } x_1=0, x_3=1, c_1=1\\&= 1 + c_2x + c_3(x^2 - x_2x) + c_4(x^3 - (x_3+x_2)x^2 + x_1x_3x)\\&+ c_5(x^4 - (x_3+x_2+x_4)x^3 + (x_2x_3 + x_3x_4 + x_2x_4 - x_2x_3x_4)x^2)\end{aligned}$$

Then we may integrate this:

$$\begin{aligned}P_5(x) &= \int_0^x P_4(t)dt\\&= x + \frac{c_2}{2}x^2 + \frac{c_3}{3}x^3 - x_2\frac{c_3}{2}x^2 + \frac{c_4}{4}x^4 - (x_3+x_2)\frac{c_4}{3}x^3\\&+ x_1x_3\frac{c_4}{2}x^2 + \frac{c_5}{5}x^5 - (x_3+x_2+x_4)\frac{c_5}{4}x^4\\&+ (x_2x_3 + x_3x_4 + x_2x_4 - x_2x_3x_4)\frac{c_5}{3}x^3\end{aligned}$$

In [ ]:
```
x1, x2, x3, x4, x5 = x
c1, c2, c3, c4, c5 = coefficients

x_pow_5 = c5/5
x_pow_4 = (c4/4 - (x4 + x3 + x2)*c5/4)
x_pow_3 = (c3/3 - (x3 + x2)*c4/3 + (x3*x2 + x3*x4 + x2*x4 - x2*x3*x4)*c5/
x_pow_2 = (c2/2 - x2 * c3/2 + x1*x3 * c4/2)
x_pow_1 = 1

print(f"P_5(x) = {x_pow_5:.5f}x^5 + {x_pow_4:.5f}x^4 + {x_pow_3:.5f}x^3 +
```

```
P_5(x) = -0.04149x^5 + 0.26832x^4 + -0.49024x^3 + -0.12634x^2 + 1.00000x
```

$$P_5(x) = -0.04149x^5 + 0.26832x^4 - 0.49024x^3 - 0.12634x^2 + x$$

I do believe this would be a good approximation for $\operatorname{erf}(x)$ on the interval $[x_1, x_5]$, however outside this interval, I would not think so. The reason I think it is a good approximation is that we first find a function f and use a integral to mimic the $\operatorname{erf}(x)$.