In [1]:

```python
import matplotlib.pyplot as plt
import numpy as np
```

Metoda Newtona - (zwana również metodą Newtona-Raphsona lub metodą stycznych) - iteracyjny algorytm wyznaczania przybliżonej wartości pierwiastka funkcji.

Zadaniem metody jest znalezienie pierwiastka równania zadanej funkcji ciągłej f:

$$\Re \supset |a,b| \ni x \mapsto f(x) \in \Re \tag{1.1}$$

w przedziale $[a,b]$. A zatem znalezienia takiego $x^* \in [a,b]$, które spełnia następujące równanie

$$f(x^*) = 0 \tag{1.2}$$

In [2]:

```python
a, b = 2.2, 5
```

In [3]:

```python
def func(x):
    return np.arctan(np.log(x**3 + 1) - 3)


def dfunc(x):
    return (3*x**2)/((x**3+1)*((3-np.log(x**3+1))**2+1))


def ddfunc(x):
    return 3*x*(8*x**3-(x**3-2)*np.log(x**3+1)**2-12*np.log(x**3+1)+20)/(
    (x**3+1)**2*(np.log(x**3+1)**2-6*np.log(x**3+1)+10)**2
    )
```

Algorytm

Dla funkcji $f(x)$ przyjmujemy jeden punkty startowy x_0 i przedział $< a, b >$ poszukiwań pierwiastka, do którego należy punkt x_0. W przedziale poszukiwań pierwiastka funkcja musi spełniać następujące warunki:

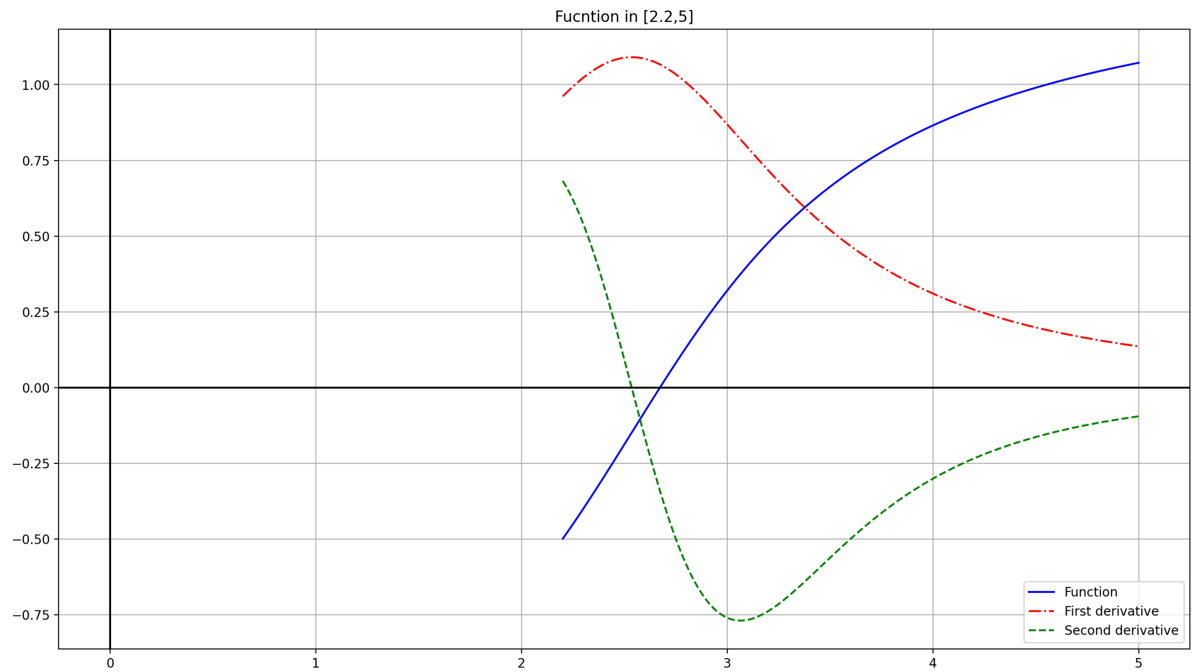
1. Funkcja $f(x)$ musi być określona,
2. Funkcja $f(x)$ musi być ciągła,
3. Funkcja $f(x)$ na krańcach przedziału $< a, b >$ przyjmuje różne znaki,
4. W przedziale $< a, b >$ pierwsza pochodna $f'(x)$ jest różna od zera,
5. Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.

In [4]:
```
x = np.linspace(a, b, 10 ** 4)
y = func(x)
dy = dfunc(x)
ddy = ddfunc(x)

fig, ax = plt.subplots()
ax.plot(x, y, label="Function", color="blue")
ax.plot(x, dy, label="First derivative", color="red", linestyle="-.")
ax.plot(x, ddy, label="Second derivative", color="green", linestyle="--")

# ax.set_aspect('equal')
ax.grid(True, which='both')
fig.set_size_inches(16, 9)
fig.set_dpi(200)
ax.axhline(y=0, color='k')
ax.axvline(x=0, color='k')
ax.set_title(f"Fucntion in [{a},{b}]")
# labelLines(ax.get_lines(), xvals=(0.1, 1), zorder=2.5)
ax.legend()
```

Out[4]:
```
<matplotlib.legend.Legend at 0x7f65a6838be0>
```



Funckja na krańcach <a,b> ma różne znaki

In [5]:
```
assert np.sign(func(a)) != np.sign(func(b))
```

W przedziale <a,b> pierwsza pochodna f'(x) jest różna od zera

In [6]:
```
truth_array = dfunc(x) != 0  # to trochę słabo działa bo sampujemy punkty które mog
assert truth_array.all()
```

Algorytm

W pierwszym kroku metody wybierany jest punkt startowy x_1 (zazwyczaj jest to wartość a lub b, z którego następnie wyprowadzana jest styczna w $f(x1)$. Odcięta punktu przecięcia stycznej z osią OX jest pierwszym przybliżeniem rozwiązania (oznaczona jako x_2). Kolejne przybliżenie dane jest wzorem rekurencyjnym

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1.3}$$

Błąd k-tego przybliżenia można oszacować poprzez nierówności

$$|x^* - x_k| \leqslant \frac{f(x_k)}{m} \qquad \text{lub} \qquad |x^* - x_k| \leqslant \frac{M}{2\,m}(x^* - x_{k-1})^2 \tag{1.4}$$

$$\text{gdzie} \qquad m = \min_{x\in[a,b]} |f'(x)| \qquad \text{oraz} \qquad M = \max_{x\in[a,b]} |f''(x)| \tag{1.5}$$

(x^* to dokładna wartość pierwiastka)

In [7]:

```
x = a
max_iter = 1000
epsilon = 10 ** (-710)

for i in range(max_iter):
    if np.abs(func(x)) <= epsilon:
        print("np.abs(func(x)) <= epsilon")
        break
    new_x = x - func(x) / dfunc(x)

    if np.abs(x - new_x) <= epsilon:
        print("np.abs(x - new_x) <= epsilon")
        break
    x = new_x
    print(f"Iteration {i:<4} | x={x:<20} | f(x)={func(x)} ")
```

```
Iteration 0    | x=2.719028380153164    | f(x)=0.049332048940445086
Iteration 1    | x=2.6719779880284964   | f(x)=-0.0004501734063324539
Iteration 2    | x=2.672399957327933    | f(x)=-3.017194671883771e-08
Iteration 3    | x=2.6723999856133505   | f(x)=-4.440892098500626e-16
Iteration 4    | x=2.672399985613351    | f(x)=0.0
np.abs(func(x)) <= epsilon
```

In [ ]: