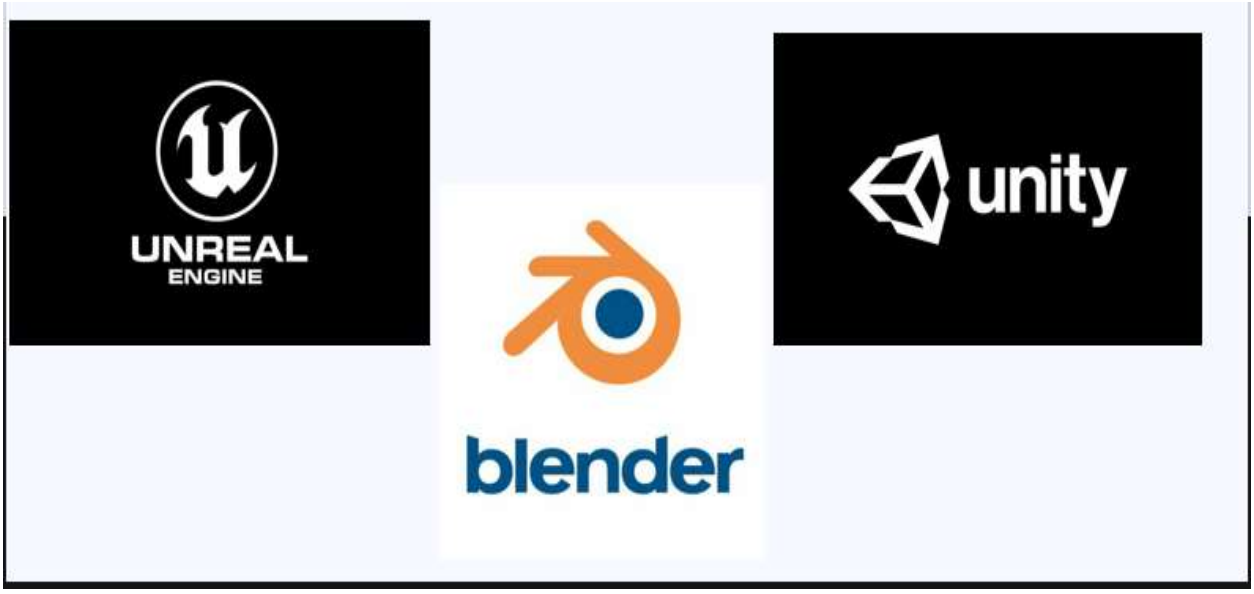
NEWTON-RAPHSON METHOD

In ray tracing, which is a technique used in computer graphics to create realistic images, the Newton-Raphson method helps determine where a light ray intersects with surfaces in a 3D scene. Imagine you have a beam of light traveling through space and hitting different objects, like a glass ball or a shiny floor. To figure out exactly where the light meets these surfaces, we need to solve complex equations that describe the shapes of the objects. The Newton-Raphson method takes our initial guess of where the intersection might be and refines it by using information about how steep the surfaces are (through their derivatives). This process continues until we pinpoint the exact intersection point, allowing us to accurately calculate how light interacts with the surfaces, such as reflections and shadows, resulting in stunningly realistic images in movies and video games.

Newton's method (or the Newton-Raphson method) is one of the most powerful and well-known numerical techniques for solving root-finding problems. There are several ways to introduce Newton's method. If we only want to present the algorithm, we can consider the technique graphically, as is often done in calculus. Alternatively, Newton's method can be derived as a technique to achieve faster convergence compared to other types of functional iterations. A third approach to introducing Newton's method, which will be discussed next, is based on Taylor polynomials. This derivation not only produces the method itself but also provides a bound for the approximation error.

SOFTWARES THAT USE NEWTON RAPHSON METHOD



Unreal Engine, Blender, and Unity utilize the Newton-Raphson method for various computational tasks, particularly in physics simulations and animation. For instance, in Unreal Engine, this method helps calculate realistic collision responses and optimize ray tracing for accurate light interaction with surfaces. In Blender, it is used for solving inverse kinematics problems, allowing for natural character movements and pose adjustments. Similarly, Unity employs the Newton-Raphson method in its physics engine to efficiently solve equations related to object dynamics and collisions, enhancing gameplay realism and responsiveness.

MATHEMATICAL REPRESENTATION

Suppose that $f \in C^2[a, b]$. Let $p_0 \in [a, b]$ be an approximation to p such that $f'(p_0) \neq 0$ and $|p - p_0|$ is "small." Consider the first Taylor polynomial for $f(x)$ expanded about p_0 and evaluated at $x = p$.

$$f(p) = f(p_0) + f'(p_0)(p - p_0) + \frac{f''(\xi(p))}{2!}(p - p_0)^2,$$

Where $\xi(p)$ lies between p and p_0. Since $f(p) = 0$, this equation gives

$$0 = f(p_0) + f'(p_0)(p - p_0) + \frac{f''(\xi(p))}{2!}(p - p_0)^2$$

Newton's method is derived by assuming that since $|p - p_0|$ is small, the term involving $(p - p_0)^2$ is much smaller, so

$$0 \approx f(p_0) + f'(p_0)(p - p_0)$$

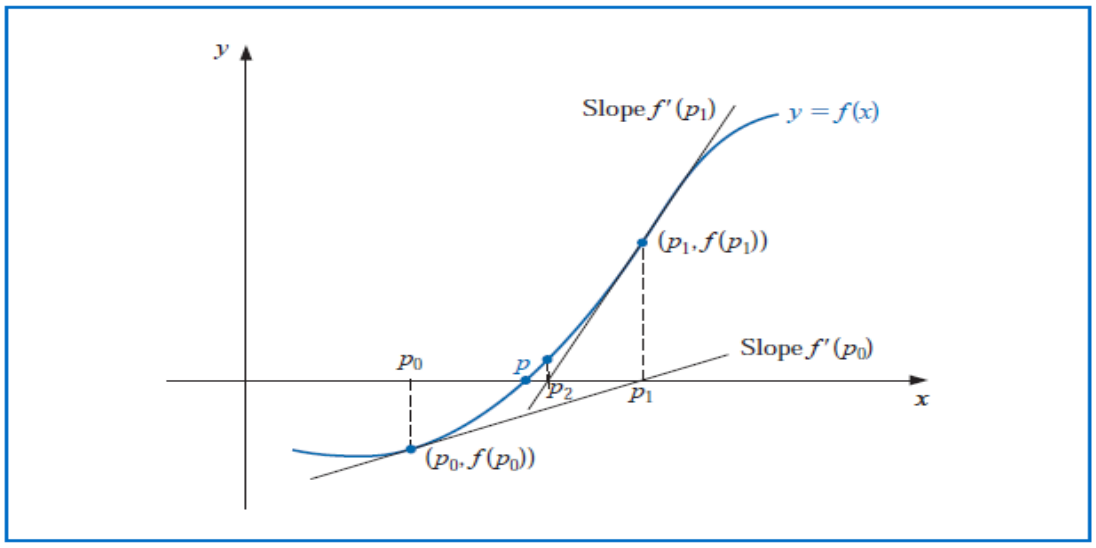
Solving for p gives

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)} \equiv p_1$$

This sets the stage for Newton's method, which starts with an initial approximation p_0 and generates the sequence $\{p_n\}_{n=0}^{\infty}$ by

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad \text{for } n \geq 1.$$

GRAPHICAL REPRESENTATION



Example

Use Newton's method to approximate a root of

$$f(x) = \cos x - x = 0 \text{ with } p_0 = {}^{\pi}/_{4}.$$

Solution

Putting $f(x) = \cos x - x$ and $f'(x) = -\sin x - 1$

in Newton's formula for n =1, we get

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} = 0.739536$$

For n =2, $p_2 = p_1 - \frac{f(p_1)}{f'(p_1)} = 0.739085$

and $p_3 = p_2 - \frac{f(p_2)}{f'(p_2)}$ =0.739085.

Which gives $p_2 = p_3$ upto 7 decimal places, as required solution of $f(x)$.

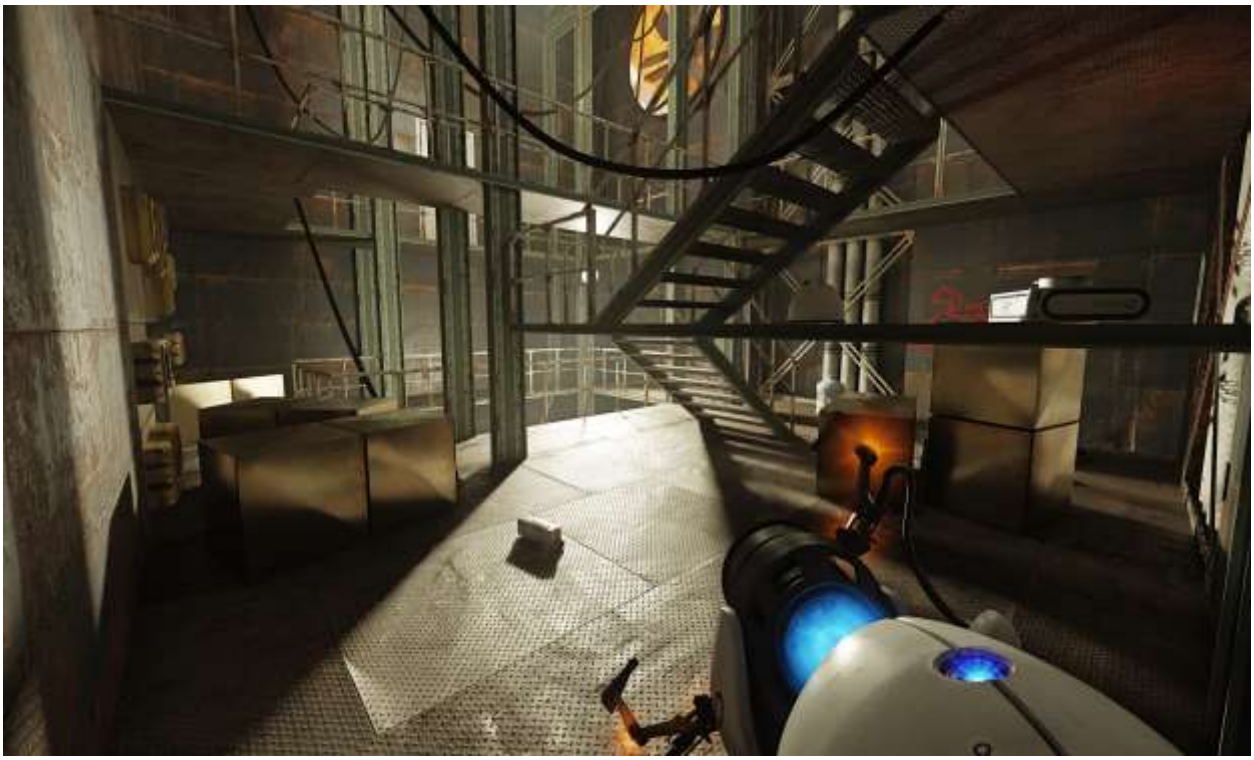
* For p_1 in calculator take $p_0 = x$ and write function as

$$x - \frac{(\cos x - x)}{(-\sin x - 1)}$$

and for p_2 $take$ $x = 0.739536$ in the same function.

For p_3 $take$ $x = 0.739085$ and so on.

PROBLEM STATEMENT



You are working as a graphics programmer on a new 3D game that requires realistic rendering of scenes with curved, reflective objects like mirrors and metallic arcs. To achieve realism, you need to compute where light rays intersect with the curves of these objects to determine how the light reflects and illuminates the scene.

One of the key challenges is solving the non-linear equation that models the curve of one such reflective object. The equation for the curve is given as:

$$f(x) = 2x\,cos(2x) - (x-2)^2 = 0$$

This equation describes the shape of a reflective arc, and solving it will give you the point along the curve where the ray intersects.

Using the **Newton-Raphson method**, find the point of intersection between a light ray and the curve defined by the equation above. Start with an initial guess of $x_0 = 2.5$ (in radians) and perform up to 20 iterations to find the root.

PYTHON CODE, SOLUTION, AND RESULT

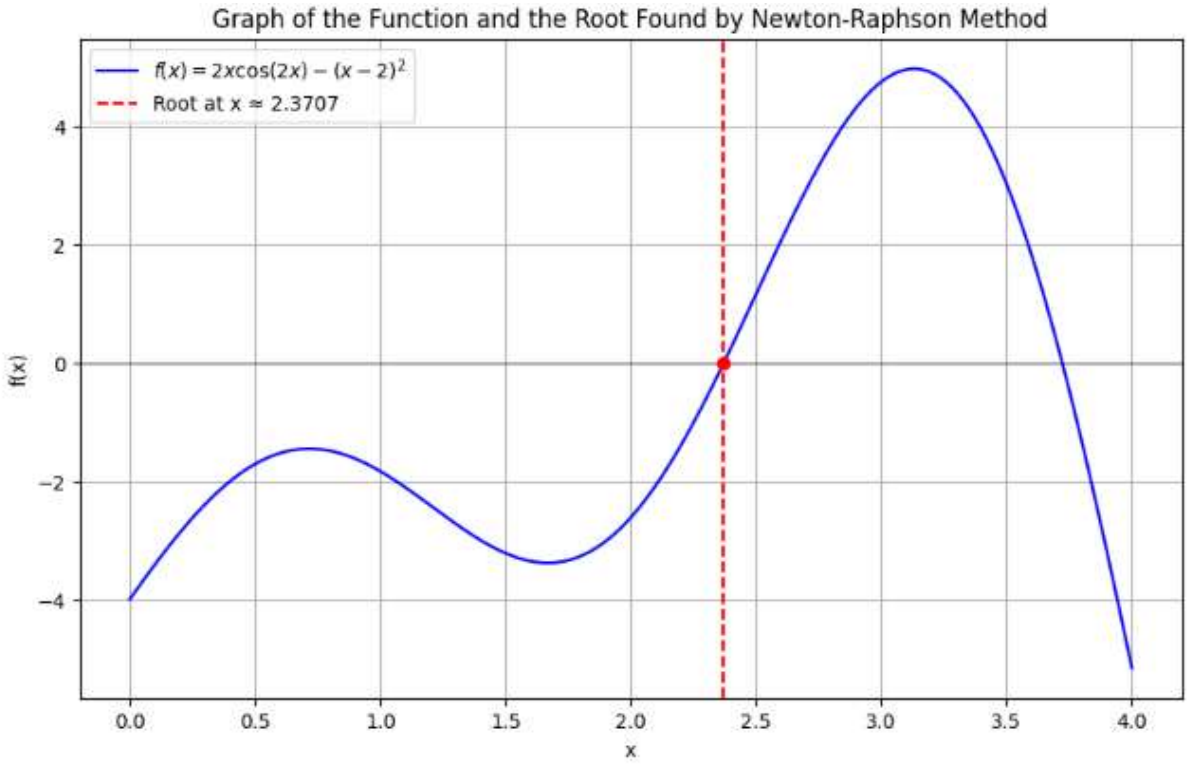
```
import numpy as np
import matplotlib.pyplot as plt

def newton_raphson(f, f_prime, x0, tol=1e-6, max_iter=20):
    """
    Newton-Raphson method for finding roots of the equation f(x) = 0.

    Parameters:
    - f: function representing f(x)
    - f_prime: function representing f'(x)
    - x0: initial guess for the root
    - tol: tolerance for stopping criterion
    - max_iter: maximum number of iterations

    Returns:
    - root: estimated root of f(x) = 0
    - history: list of (iteration, x_value) tuples
    """
```

```
    x = x0
    history = []
    for i in range(max_iter):
        fx = f(x)
        fpx = f_prime(x)

        if fpx == 0:
            raise ValueError("Derivative is zero; Newton-Raphson method fails.")

        x_new = x - fx / fpx
        history.append((i + 1, x_new))

        if abs(x_new - x) < tol:
            return x_new, history

        x = x_new

    raise ValueError("Maximum number of iterations reached without convergence.")
```

```python
# Define the function f(x) and its derivative f'(x)
def f(x):
    return 2*x*np.cos(2*x) - (x - 2)**2

def f_prime(x):
    return 2*(np.cos(2*x) - 2*x*np.sin(2*x)) - 2*(x - 2)

# Initial guess
x0 = 2.5

# Find the root and get iteration history
root, history = newton_raphson(f, f_prime, x0, max_iter=20)

# Print the iteration details
print(f"Converged to root: {root:.6f} after 20 iterations")
print("Iteration | x-value")
print("----------------------")
for iteration, x_value in history:
    print(f"{iteration:9d} | {x_value:.6f}")

# Generate x values for plotting
x_values = np.linspace(0, 4, 400)
f_values = f(x_values)
```

```python
# Plot the function
plt.figure(figsize=(10, 6))
plt.plot(x_values, f_values, label=r'$f(x) = 2x \cos(2x) - (x - 2)^2$', color='blue')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(root, color='red', linestyle='--', label=f'Root at x ≈ {root:.4f}')
plt.scatter([root], [f(root)], color='red', zorder=5)
plt.title('Graph of the Function and the Root Found by Newton-Raphson Method')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Converged to root: 2.370687 after 20 iterations
Iteration | x-value
--------------------
        1 | 2.372407
        2 | 2.370688
        3 | 2.370687
```



ADVANTAGES/DISADVANTAGES

In a practical application, an initial approximation is selected and successive approximations are generated by Newton's method. These will generally either converge quickly to the root, or it will be clear that convergence is unlikely.

PRACTICE

Q1. The accumulated value of a savings account based on regular periodic payments can be determined from the *annuity due equation*,

$$A = \frac{P}{i}[(1+i)^n - 1].$$

In this equation, *A* is the amount in the account, *P* is the amount regularly deposited, and *i* is the rate of interest per period for the *n* deposit periods. An engineer would like to have a savings account valued at $750,000 upon retirement in 20 years and can afford to put $1500 per month toward this goal. What is the minimal interest rate at which this amount can be invested, assuming that the interest is compounded monthly?

Q2. Let $f(x) = -x^3 - \cos x$ and $p_0 = -1$. Use Newton's method to find p_2. Could $p_0 = 0$ be used?

Solution: Putting $f(x) = -x^3 - \cos x$ and $f'(x) = -3x^2 + \sin x$

in Newton's formula $p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}$

for n=1, we get $p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} = -0.88033$

For n=2, $p_2 = p_1 - \frac{f(p_1)}{f'(p_1)} = -0.86568.$

We cannot use $p_0 = 0$ as $f'(0) = 0$ gives p_1 undefined.

Q3. Use Newton's method to find solution accurate to within 10^{-5} for

a. $f(x) = x^3 - 2x^2 - 5$ in [1, 4] and
b. $f(x) = x^3 + 3x^2 - 1$ in [-3, -2].

Q4. Use Newton-Raphson method to find solution accurate within $10^{-3}$for the equation:

$$2x\cos(2x) - (x-2)^2 = 0 \quad for\ 2 \le x \le 3.$$