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

```
# Simple iteration (Brute Force) single root
#For single root.
def b_force(f,h):
  epochs =50
  x_roots = []
  for epoch in range(epochs):
    x_guess = f(h)
    print(x_guess)
    if x_guess == 0:
      x_roots.append(h)
      break
    else:
         h+=1
  return print(f"The root is: {x_roots}, found at epoch {epoch}")
```

```
# Finding n number of roots.
def brute_nforce(f,h,epochs = 10): #default
  n_roots = 3
  x_roots = []
  end_epoch = 0
  for epoch in range(epochs):
    print(f(h))
    if np.allclose(0,f(h)):
      x_roots.append(h)
      end_epoch = epoch
      if len(x_roots)==n_roots:
        break
    h+=1
  return print(f"The root is: {x_roots}, found at epoch {end_epoch+1}")
```

```
#Newton- Rhapson Method single root
## Single Root
def newt_R(f,f_prime,epochs):
  x = 0
  root = 0
  for epoch in range(epochs):
    x_prime = x - (f(x)/f_prime(x))
    if np.allclose(x, x_prime):
      root = x
      break
    x = x_prime
  return print(f"The root is: {root}, found at epoch {epoch}")
```

```
# Findng n number of  roots
def num_newt(f,f_prime,epochs):
```

```
def num_newt(f,f_prime,epochs):
  x_inits = np.arange(0,5)
  roots = []
  for x_init in x_inits:
    x = x_init
    for epoch in range(epochs):
      x_prime = x - (f(x)/f_prime(x))
      if np.allclose(x, x_prime):
        roots.append(x)
        break
      x = x_prime
  np_roots = np.round(roots,3)
  print("np_roots before round: ",roots)
  np_roots = np.round(roots,3)
  print("np_roots after round: ", np_roots)
  np_roots = np.unique(np_roots)
  print("np_roots after sorting to unique: ", np_roots)
  return np_roots
```

```
def bisect_n(func, iv1, iv2, nm_roots, epochs, tol):
    roots = []
    y1, y2 = func(iv1), func(iv2)
    end_bisect = 0
    if np.sign(y1) == np.sign(y2):
          print("Root cannot be found in the given interval")
    else:
          for bisect in range(epochs):
              midp = np.mean([iv1,iv2])
              y_mid = func(midp)
              y1 = func(iv1)
              if np.allclose(0, y1,tol):
                  roots.append(iv1)
                  end_bisect = bisect
                  if len(roots) == nm_roots: # getting the number of roots.
                    break
              if np.sign(y1) != np.sign(y_mid): #root for first-half interval
                  iv2 = midp
              else: #root for second-half interval
                  iv1 = midp

    return print(" the roots are" ,roots, " found at" ,end_bisect)
```

```
## Regula Falsi Method
## Finding multiple roots
def rfalsi_n(f,a,b,tol):
  x_inits = np.arange(0,10)
  arr_len = len(x_inits) - 1
  y1, y2 = f(a), f(b)
  root = None
  n_roots = []
  pos = 0
  pos = 0
  if np.allclose(0,y1):
    root = a
    n_roots.append(a)

  elif np.allclose(0,y2):
    root = b
    n_roots.append(b)

  elif np.sign(y1) == np.sign(y2):
    print("No root here")
  else:
    for iter in range(arr_len):
      a = x_inits[iter]
      b = x_inits[iter+1]
      for pos in range(0,100):
        c = b - (f(b)*(b-a))/(f(b)-f(a)) ##false root
        if np.allclose(0,f(c),tol):
          root = c
          n_roots.append(c)
        if np.sign(f(a)) != np.sign(f(c)):
         b,y2 = c,f(c)
        else:
          a,y1 = c,f(c)

  roots = n_roots
  n_roots = np.unique(np.round(n_roots,3))
  return roots, n_roots, pos
```

```
def sec_n(f,a,b,epochs):
  x_inits = np.arange(0,10)
  arr_len = len(x_inits) - 1
  root = None
  n_roots = []
  end_epoch = 0
  for iter in range(arr_len):
    a = x_inits[iter]
    b = x_inits[iter+1]
    for epoch in range(epochs):
      c = b - (f(b)*(b-a))/(f(b)-f(a))
      if np.allclose(b,c):
        root = c
        n_roots.append(root)
        end_epoch = epoch
        break
      else:
        a,b = b,c
  roots = n_roots
  n_roots = np.unique(np.round(n_roots,3))
  return roots, n_roots, end_epoch
```