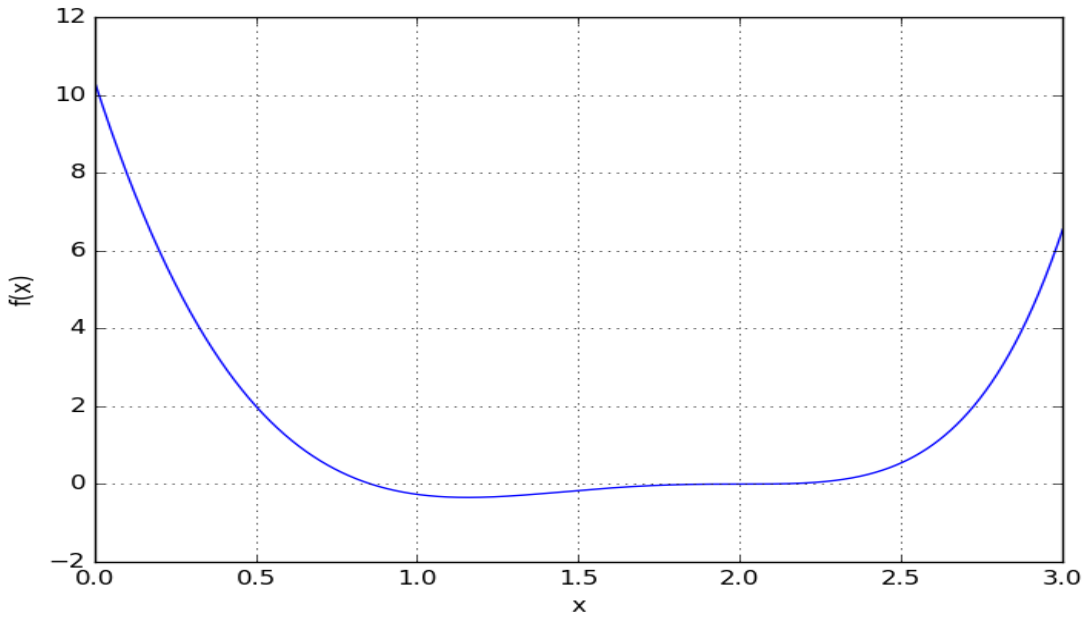
Στην αρχή έχω τα απαραίτητα imports για `numpy`, `scipy` και `matplotlib`:

```
from numpy import *
import numpy

from scipy import *
import scipy

import matplotlib.pyplot as plt
```

Η συνάρτηση και οι 2 πρώτες παράγωγοι ορίζονται και γίνεται το plot της $f(x)$:

```
t = arange(0,3.001,step=.001)

def f(t):
  return (14*t)*(e**(t-2)) - 12*(e**(t-2)) - 7*(t**3) + 20*(t**2) - 26*t + 12

def f_der(t):
  return (14*t)*(e**(t-2)) + 2*(e**(t-2)) - 21*(t**2) + 40*t - 26

def f_der_2(t):
  return (14*t)*(e**(t-2)) - 16*(e**(t-2)) - 42*t + 40

plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(t, f(t))
plt.grid(True)
plt.show()
```



Καθώς η άσκηση δεν το απαιτεί δεν υπάρχουν έλεγχοι για σφάλματα ούτε συνθήκες τερματισμού σε περίπτωση που δεν υπάρχουν ρίζες.

Ακολουθεί η συνάρτηση για διχοτόμηση:

```
def bisect(a, b):
  a_1, b_1 = a, b
  root = (a + b) / 2

  # number of times to repeat to achieve 6 points accuracy
  N = int(ceil((log(b - a) - log(0.0000005)) / log(2)))

  for i in range(0, N):
    if (f(a) < 0 and f(root) > 0) or (f(a) > 0 and f(root) < 0):
      b = root
    elif (f(b) < 0 and f(root) > 0) or (f(b) > 0 and f(root) < 0):
      a = root
    root = (a + b) / 2

  return root, N, a_1, b_1
```

Απλά γίνετε η εγαρμογή του θεωρήματος για $N = \frac{\ln b - a - \ln error}{\ln 2}$ φορές

Στην συνέχεια έχω τη συνάρτηση για την μέθοδο Newton-Raphson:

```
def new_raph(start):
  temp_l = [start, start - (f(start)/f_der(start))]

  N = 1
  while 0.0000005*abs(temp_l[N]) < abs(temp_l[N-1] - temp_l[N]):
    temp =  temp_l[N] - (f(temp_l[N])/f_der(temp_l[N]))
    temp_l.append(temp)
    N = N + 1

  root = temp_l[N]

  #N-1 because the N=N+1 happens at the very end of the while loop
  return root, N - 1, start
```

Εδώ αρχικοποιώ τον πίνακα `temp_l` με την τιμή εισόδου της συνάρτησης και το αποτέλεσμα μιας πρώτης εφαρμογής της αναδρομικής συνάρτησης

$$f(x_n) = f(x_{n-1}) - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Στην συνέχεια με τον έλεγχο στο while η συνάρτηση θα τρέχει μέχρι να επιτευχθεί η επιθυμητή ακρίβεια (6 δεκαδικά ψηδία)

Επιστρέφω την ρίζα, τον αριθμό επαναλήψεων και το σημείο εκκίνησης. Ο αριθμός επαναλήψεων είναι N-1 γιατί η αύξηση του δείκτη γίνεται στο τέλος του `while` ενώ ξεκινάει με τον δείκτη στο 1 αντι του 0 για να μπορεί να γίνει ο έλεγχος

Ακολουθέι η συνάρτηση της μεθόδου της τέμνουσας:

```
def intersection(a, b):
  temp_l = [a, b, b - ((f(b)*(b - a))/f(b) - f(a))]

  N = 1
  while 0.0000005*abs(temp_l[N]) < abs(temp_l[N-1] - temp_l[N]):
    temp =  temp_l[N] - (f(temp_l[N])*
                         (temp_l[N] - temp_l[N-1]))/(f(temp_l[N]) - f(temp_l[N
   -1]))
    temp_l.append(temp)
    N = N + 1

  root = temp_l[-1]

  #N-1 for the same reason as the N-R method
  return root, N - 1, a, b
```

Η συνάρτηση είναι αντίστοιχη με αυτή της Newton-Raphson. Χρειάζεται δύο αρχικές τιμές, και για να γίνει ο έλεγχος του `while`, υπολογίζω και την τρίτη σύμφωνα με την αναδρομική συνάρτηση

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

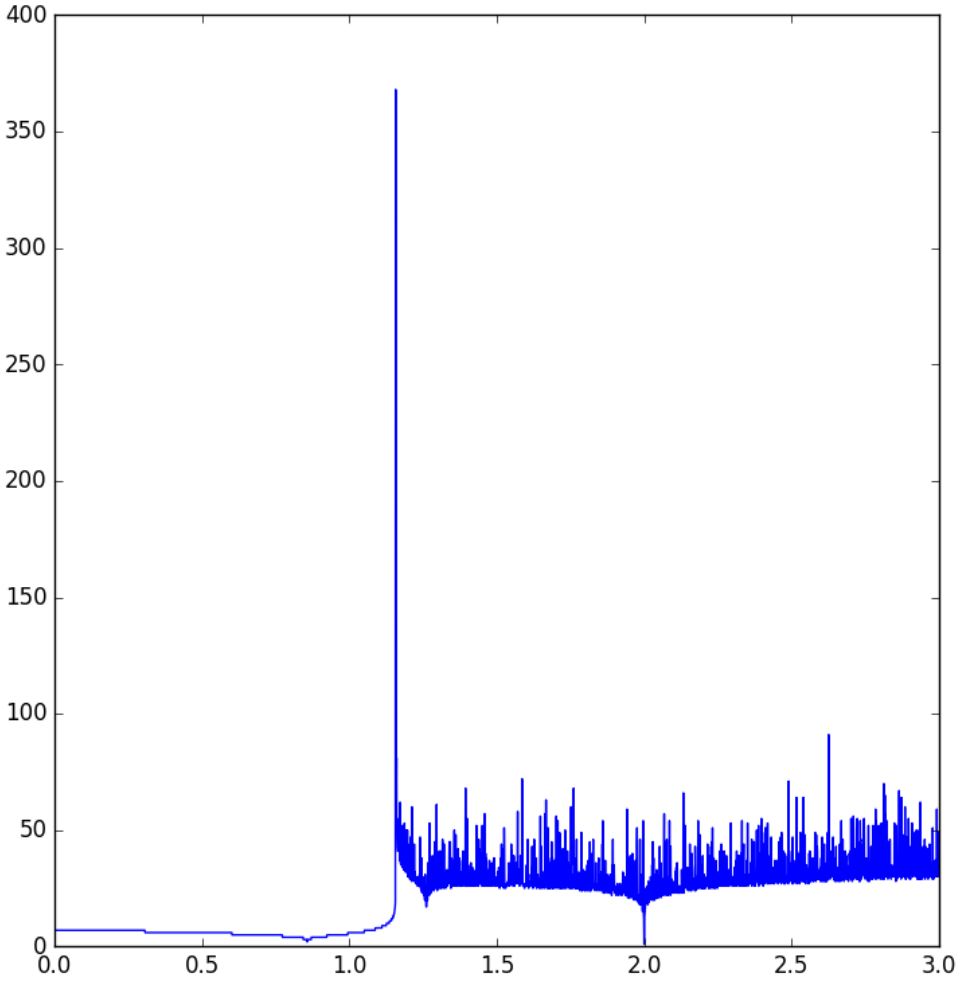
Ο έλεγχος είναι ο ίδιος με την Newton-Raphson για να πετύχω τα 6 δεκαδικά ψηφία

Επιστρέφω την ρίζα, τον αριθμό επαναλήψεων καθως και τις 2 αρχικές τιμές.

Ακολουθεί η έξοδος του προγράμματος όταν το τρέχω με τις κατάλληλες αρχικές τιμές (βασισμένες στο διάγραμμα την συνάρτησης):

```
$ python ex1.py
++++ Bisection ++++

Root in [0.00,1.50] after 22 loops: f(0.857143) = -0.000000

Root in [1.50,3.00] after 22 loops: f(2.000004) = 0.000000

++++ Newton - Raphson ++++

Starting at 1.00:
after 5 iterations the root is: f(0.857143) = -0.000000

Starting at 3.00:
after 29 iterations the root is: f(2.000011) = 0.000000

++++ Interpolation ++++

Root found in [0.70,0.90] after 11 iterations:
f(0.857143) = -0.000000

Root found in [1.70,2.10] after 52 iterations:
f(2.000023) = 0.000000
```

Στο διάγραμμα που ακολουθεί βλέπουμε τις επαναλήψεις που χρειάζετε η μέθοδος Newton-Raphson σε σχέση με την αρχική τιμή που δίνεται



Παρατηρώ οτι για την ρίζα στο 0.857143 η μέθοδος συγκλίνει τετραγωνικά ενώ για την ρίζα στο 2.000004 θέλει εμφανώς περισσότερες επαναλήψεις.

Συγκρίνοντας αυτό το διάγραμμα με το plot της συνάρτησης στην 1η σελίδα παρατηρώ οτι η ταχύτητα σύγκλισης στην πρώτη ρίζα είναι μεγάλη γιατι η συνάρτηση διατηρεί την ίδια κυρτώτητα ενώ στην δεύτερη η συνάρτηση αλλάζει.

Καθώς η μέθοδος N-R χρησιμοποιεί την κλίση της παραγόγου για να βρεί το επόμενο x στην αναδρομή, όταν η κυρτώτητα αλλάζει γύρω απο την ρίζα ο αλγόριθμος ταλαντώνετε και αργεί να συγκλίνει σε έναν αριθμό που να ικανοποιεί το σφάλμα που θέσαμε.