Numerical Methods for Astrophysics:

SOLUTION OF NON-LINEAR EQUATIONS

Michela Mapelli

Non-linear equations. Concept

- Many equations / systems of equations in physics and astrophysics are NON-LINEAR (e.g. chemical networks in molecular clouds)

- Non-linear equations are generally harder to solve than linear eq.

- A non-linear eq. for 1 single variable x can be written as

$$f(x) = 0$$

Thus, solving the equation is equivalent to FINDING THE ZEROES or FINDING THE ROOTS of f(x)

→ Several methods to solve non-linear equations are called ROOT FINDING METHODS

During this course: relaxation method (not a root finding method)
bisection method (root finding method)
Newton-Raphson method (root finding method)

Non-linear equations. Relaxation method

VERY SIMPLE IDEA:

solve the equation by ITERATING the equation
(analogous to idea of Gauss – Seidel for linear systems)

STEPS:

1. write the equation in form *x = f(x)*
2. Let's start with a GUESS for the value of x
3. then ITERATE

Example x = 2 – exp(–x)

```
import numpy as np
tol=1e-6
x=1.0
xold=10.0
while(abs(x-xold)>tol):
    xold=x
    x=2.-np.exp(-x)
    print(x)
print("Converged to x=", x)
```

Non-linear equations. Relaxation method

VERY SIMPLE METHOD: easy to code

PROBLEM 1: not always possible/easy to write eq. in form x= f(x)

Example: $\log x + x^2 - 1 = 0$

Try to re-arrange: $x = \exp\left(1 - x^2\right)$

Plunge this into previous code with guess x = 0.5
→ the result does not converge

PROBLEM 2: does not always converge

Before giving up, try re-arrange again $x = \sqrt{1 - \log x}$

The same script converges

Non-linear equations. Relaxation method

Why it does not converge sometimes?

Assume we have an equation in the form $x = f(x)$ with solution $x = x^*$

Let's derive the Taylor expansion around x^*

If we name $\tilde{x}$ the result of $x = f(x)$ after the first iteration:

$$\tilde{x} = f(x) = f(x^*) + (x - x^*)\, f'(x^*) + O((x - x^*)^2)$$

Since x^* is the exact solution, $f(x^*) = x^*$

$$\tilde{x} = x^* + (x - x^*)\, f'(x^*)$$

$$\tilde{x} - x^* = (x - x^*)\, f'(x^*)$$

if f'(x)>1 the value of ($\tilde{x}$ – x*) increases at 2nd iteration → divergence*

if f' (x*)<1** the value of ($\tilde{x}$ – x*) decreases at 2nd iteration →* ***convergence

Non-linear equations. Relaxation method

WHEN DO I STOP ITERATING?

Reasonable approach:

1. **decide the maximum error you can tolerate ε**
   (based on your specific needs)
2. **exit the loop when | x – xold | < ε**
   where epsilon ε measures the maximum error

SUMMARY of PROS and CONS

PROS:

- **trivial to implement**
- **fast**
- **might be used to solve systems of non-linear equations**

CONS:

- **frequent problems of convergence**
- **might require re-arranging the equation manually**

Non-linear equations. Overrelaxation

We can try to speed up convergence with overrelaxation

Rewrite $\tilde{x} = f(x)$ in the form $\tilde{x} = x + \Delta x$ where

$$\Delta x = \tilde{x} - x = f(x) - x$$

Let us now add a $(1 + \omega)$ term in the original equation:

$$\tilde{x} = x + (1 + \omega)\Delta x$$

where ω is a free parameter.

Then the equation to iterate becomes

$$\tilde{x} = x + (1 + \omega)[f(x) - x] = (1 + \omega)\, f(x) - \omega\, x$$

Values of $\omega > 0$ imply that we take a larger step wrt previous iteration

This might help speeding up convergence but also cause divergence

$\omega < 1$ recommended No analytic way to choose the best ω

Non-linear equations. Bisection method

- ROOT FINDING ALGORITHM:

First rearrange our non-linear equation so that *f(x) = 0*
(move all the non-zero terms to the left)

- Choose an interval x_1, x_2

Suppose we want to find a root of f(x) between x_1 and x_2 (if any)

- Estimate f(x_1) and f(x_2)

if f(x_1)>0 and f(x_2)<0
or f(x_1)<0 and f(x_2)>0
there is at least one zero between x_1 and x_2

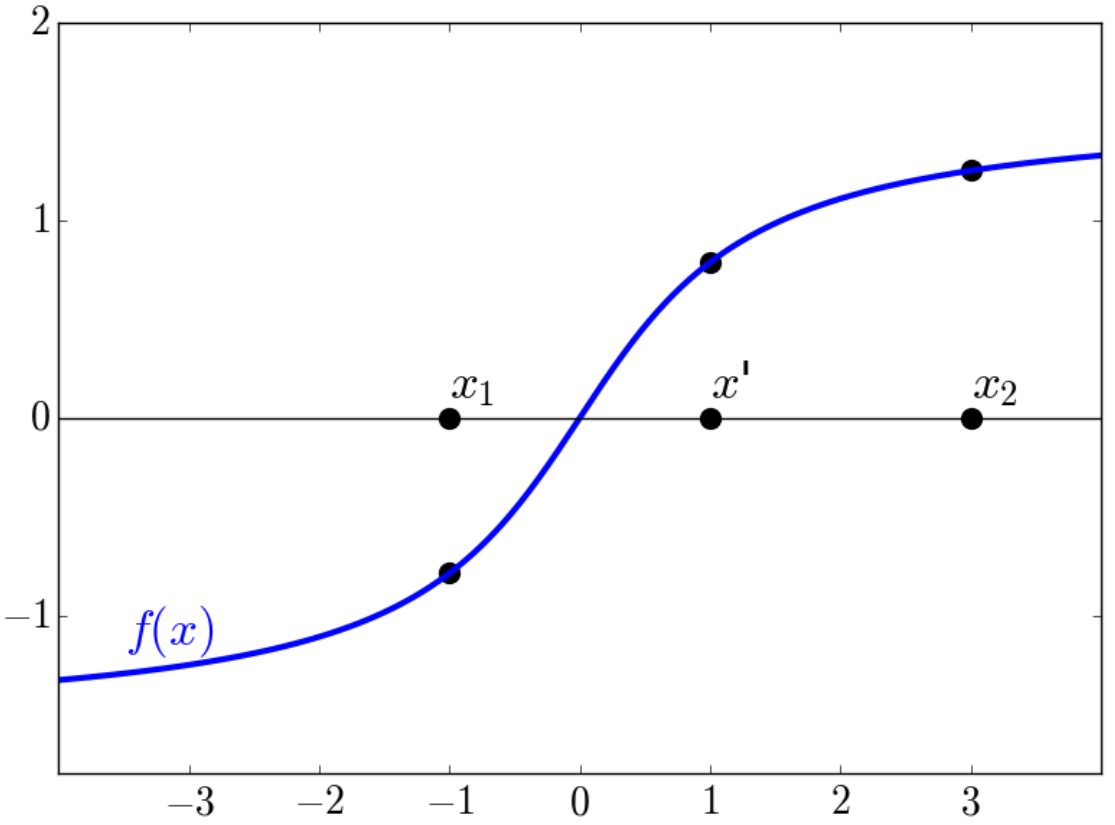
- Calculate the mid-point

x' = 0.5 * (x_1 + x_2)

- Estimate f(x')

if f(x_1)<0 and f(x')>0
restrict the search interval between x_1 and x'
(vice versa take x', x_2)

- REPEAT with smaller and smaller intervals



Non-linear equations. Bisection method

SUMMARY OF THE ALGORITHM:

1. **Choose an initial interval x_1 , x_2 .**
   Choose the minimum accuracy ε you want

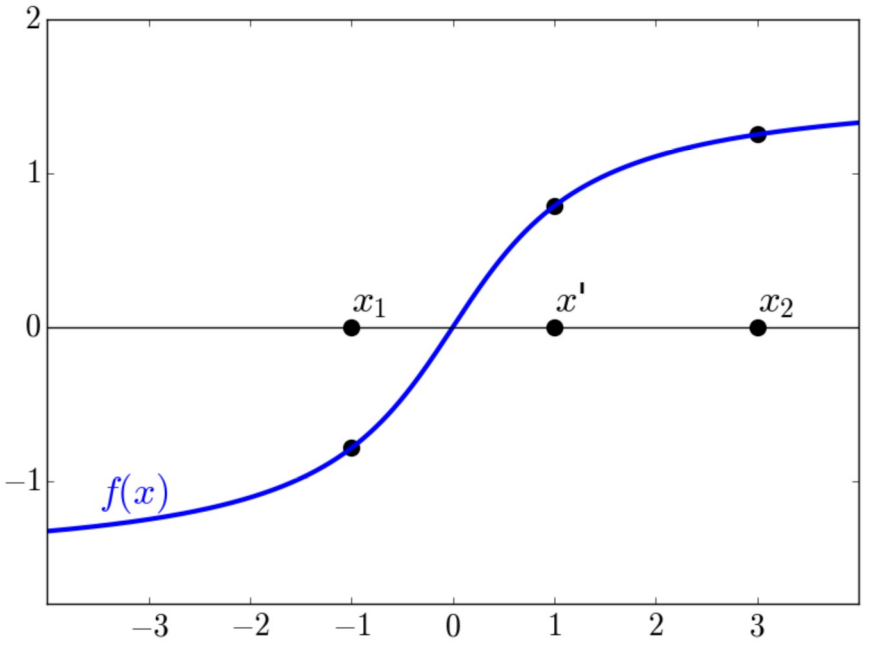
2. **Check that f (x_1) and f (x_2) have opposite signs.** If they don't, you need to choose another interval or another method.

3. **Calculate the midpoint x' = 0.5 $*$ (x_1 + x_2)**

4. **If f (x') has the same sign as f (x_1), then define the new interval as x' , x_2 . Otherwise, define the new interval as x_1, x'**

5. **If $|x_1 - x_2| > \varepsilon$,**
   repeat from step 3.

   Otherwise, calculate
   0.5 $*$ (x_1 + x_2)
   once more and this is the final estimate of the root



Non-linear equations. Bisection method

WHEN DO I STOP ITERATING?

If we want to find the root with an accuracy of 10^{-6} ,
we simply stop when the interval is ≤ 10^{-6} .

At every step, we know the root with a factor of 2 better accuracy

Since the error decreases by a factor of 2 on each time-step,
the accuracy improves exponentially.

Number N of steps we need to take to reach the desired accuracy:

$$N = \log_2 \Delta / \epsilon$$

where Δ = initial interval
ε = maximum error tolerated

Non-linear equations. Bisection method

SUMMARY of PROS and CONS

PROS:

- robust method: if there is a zero in the interval, it converges

- fast

CONS:

- it might be that $f(x_1)$ and $f(x_2)$ have the same sign,
because there is an even number of zeros in the x_1, x_2 interval.
In this case, the bisection does not work
→ try to reduce interval

- it cannot find even-order polynomial roots of functions,
e.g, function $(1 - x^2)$.
Functions with such roots only touch the horizontal axis at the position of the root but do not cross it

- it does not allow to solve systems of non-linear equations.

Non-linear equations. Newton-Raphson method

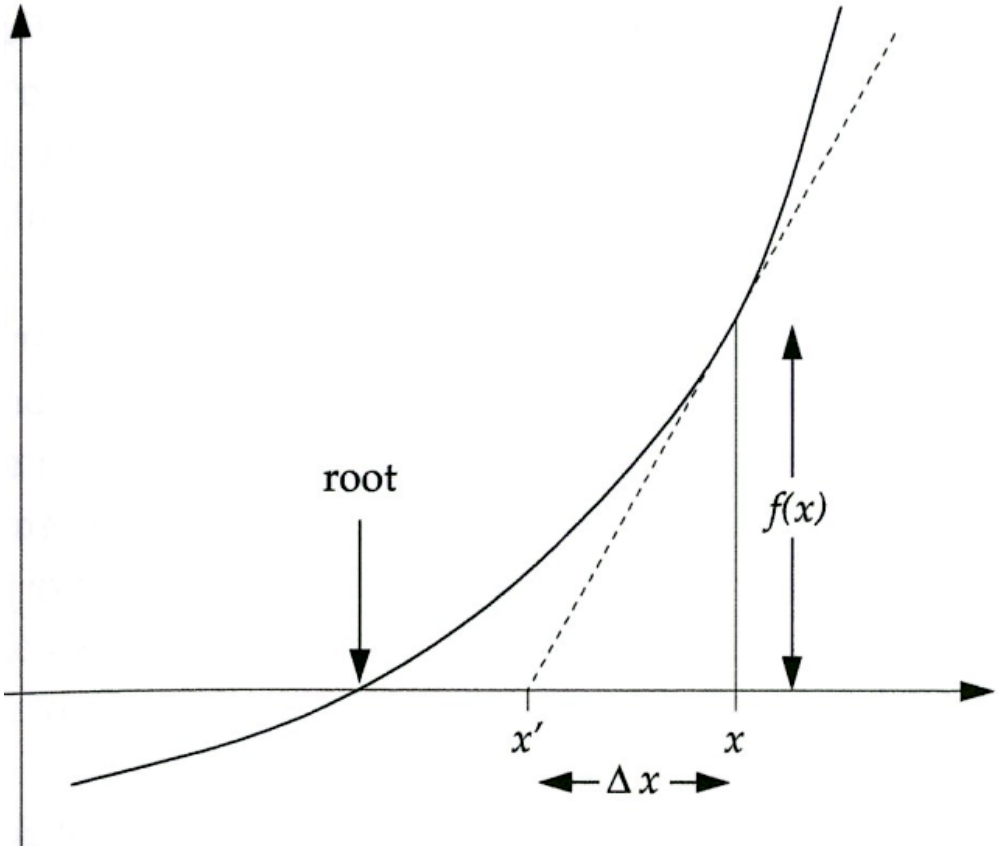
- **ROOT FINDING ALGORITHM:**
  First rearrange equation
  so that ***f(x) = 0***

- **start with a guess x for the solution**

- **then calculate the slope**
  of the function at x
  to refine our initial guess

$$f'(x) = \frac{f(x)}{\Delta x}$$

SLOPE is the first derivative
of *f(x)* in *x*



- **the new guess *x'* for the root is then**

$$x' = x - \Delta x = x - \frac{f(x)}{f'(x)}$$

Non-linear equations. Newton-Raphson method

- requires to know derivative *f'(x)*

- For functions with more than one root,
this method typically finds a root close to the starting value of *x*.
Thus, to find different roots we can start from different guesses.

- Error on our next estimate *x'* is given by $$\epsilon' = \left[\frac{f''(x)}{2\,f'(x)}\right]\,\epsilon^2$$

where ε is the error on the previous estimate *x :*

$x^* = x + \varepsilon$ with x^* = exact solution

and ε' is the error on the next estimate *x'* :

$x^* = x' + \varepsilon'$

→**converges quickly:** the error on the next step is of the size of the square of the error on the previous step

Use Taylor expansions to demonstrate above relation

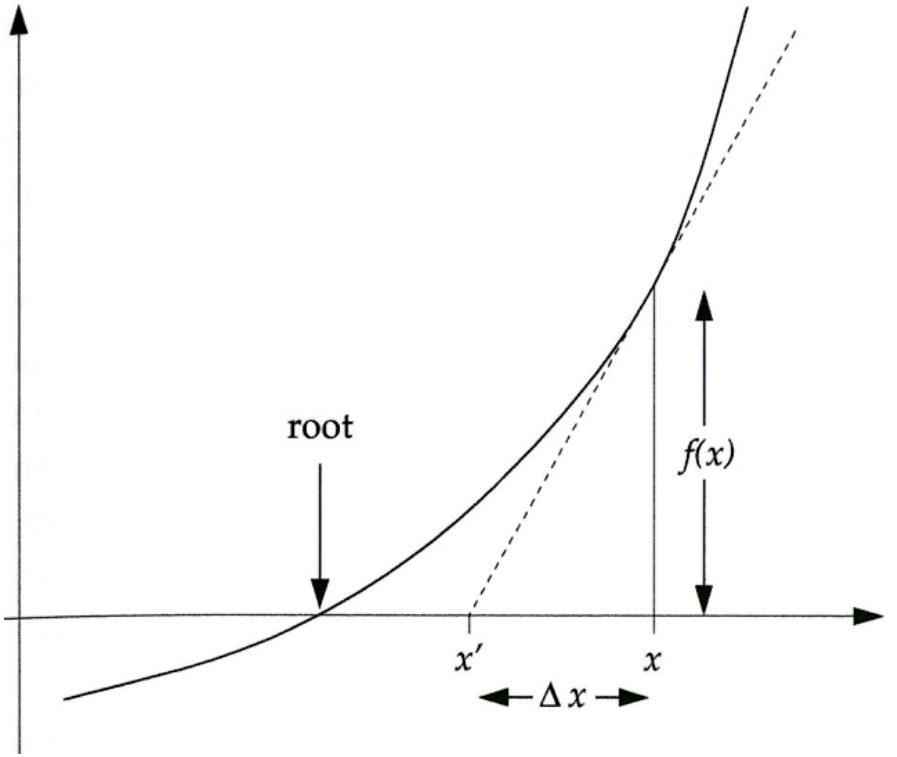
Non-linear equations. Newton-Raphson method

SUMMARY OF THE ALGORITHM:

1. Choose a starting guess *x* and the accuracy ε you want

2. Calculate a new guess

$$x' = x - \Delta x = x - \frac{f(x)}{f'(x)}$$



3. If |*x*' – *x* | > ε , repeat from point 2.
If |*x*' – *x* | < = ε , *x*' is the result.

Or (alternative choice to stop iterations)

If *f*(*x*') > ε_1, repeat from point 2
If *f*(*x*') < = ε_1, *x*' is the result

Non-linear equations. Newton-Raphson method

SUMMARY of PROS and CONS

PROS:

- very fast

- easy to implement

- can be used to solve systems of non linear equations

CONS:

- sometimes, it does not converge (less serious problem than for the relaxation algorithm)

- we need to know the derivative of *f(x)*; if we do not, we can try to use numerical derivatives

Non-linear equations. Comparison

SUMMARY of PROS and CONS

relaxation, bisection and Newton-Raphson:

all of them have pros and cons.

In some cases, one will perform better than the other.

Thus, we must be flexible and, when one algorithm fails to solve a problem, try another one.

Following Mark Newman's book:

``*One of the keys to doing good computational (astro)physics is to have a range of methods at your disposal, so that if one works poorly for a particular problem you have others up your sleeve.*"

Non-linear equations. Newton-Raphson method for systems of equations

1. write the system in the form

$$f_1(x_1, x_2, ..., x_N) = 0,$$
$$f_2(x_1, x_2, ..., x_N) = 0,$$
$$\vdots$$
$$f_N(x_1, x_2, ..., x_N) = 0.$$

2. Suppose these equations have a root ($x_1^*, x_2^*, \ldots, x_N^*$)

Then, we can write the Taylor expansion around the x_i as

$$f_i(x_1^*, x_2^*, ..., x_N^*) = f_i(x_1, x_2, ..., x_N) + \sum_j (x_j^* - x_j) \frac{\partial f_i}{\partial x_j} + ...$$

In vector notation

$$\mathbf{f}(\mathbf{x}^*) = \mathbf{f}(\mathbf{x}) + \mathbf{J} \cdot (\mathbf{x}^* - \mathbf{x}) + ...$$

where **J** = Jacobian matrix = N x N matrix of elements $J_{ij} = \frac{\partial f_i}{\partial x_j}$

Non-linear equations. Newton-Raphson method for systems of equations

3. Since x* is a root, f(x*) = 0

$$\mathbf{f}(\mathbf{x}^*) = \mathbf{f}(\mathbf{x}) + \mathbf{J} \cdot (\mathbf{x}^* - \mathbf{x}) + \ldots$$

0

Neglecting higher order terms and setting **$\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$**

$$\mathbf{J} \cdot \Delta\mathbf{x} = \mathbf{f}(\mathbf{x})$$

This is a set of linear equations of the form $A\,x = b$
Let us solve it using Gaussian elimination or Gauss-Seidel method.

4. After we solve for **$\Delta\mathbf{x}$**, the new estimate of the root is

$$\mathbf{x}' = \mathbf{x} - \Delta\mathbf{x}$$

Non-linear equations. Exercise: Eccentric anomaly

EXERCISE:

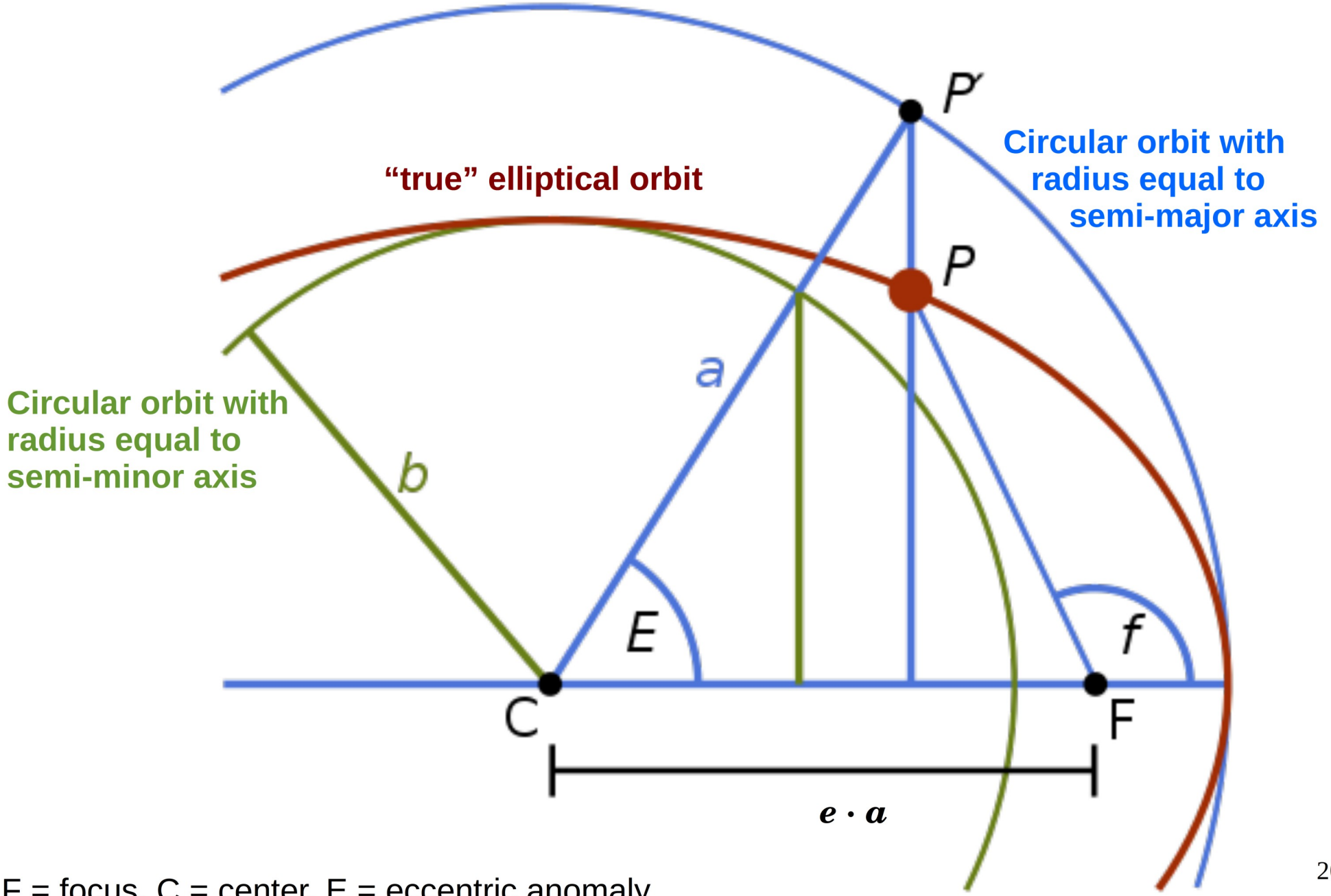
Write a script to implement relaxation method, bisection method and Newton-Rapshon. Use them to find the zeros of the following function of the eccentric anomaly E:

$$\mathcal{F} = \mathrm{E} - ecc * np.sin(\mathrm{E}) \tag{62}$$

In the above equation, ecc is the eccentricity of a Keplerian binary system: it can take any value in the range $[0, 1)$. Please, try to solve the equation for three different values of $ecc = 0.1, 0.7, 0.9$.
The term $\mathcal{F} = 2\,\pi\,t_p/\mathrm{T}$ (where T is the orbital period of the binary and t_p is the time elapsed from pericentre passage) is the mean anomaly and can take any values between 0 and $2\,\pi$. Please calculate the result for $\mathcal{F} = \pi$ and $\pi/3$.
Finally, E is the eccentric anomaly of a Keplerian binary system and is the unknown you should try to find with this exercise. For more information on the eccentric anomaly, see `https://en.wikipedia.org/wiki/Eccentric_anomaly` and the figure below.

Solutions of the exercise (with tolerance 10^{-6}):
$\mathrm{E}(ecc = 0.1, \mathcal{F} = \pi/3) \sim 1.137976$
$\mathrm{E}(ecc = 0.7, \mathcal{F} = \pi/3) \sim 1.737494$
$\mathrm{E}(ecc = 0.9, \mathcal{F} = \pi/3) \sim 1.899123$
$\mathrm{E}(ecc = *, \mathcal{F} = \pi) = \pi$

Non-linear equations. Exercise: Eccentric anomaly



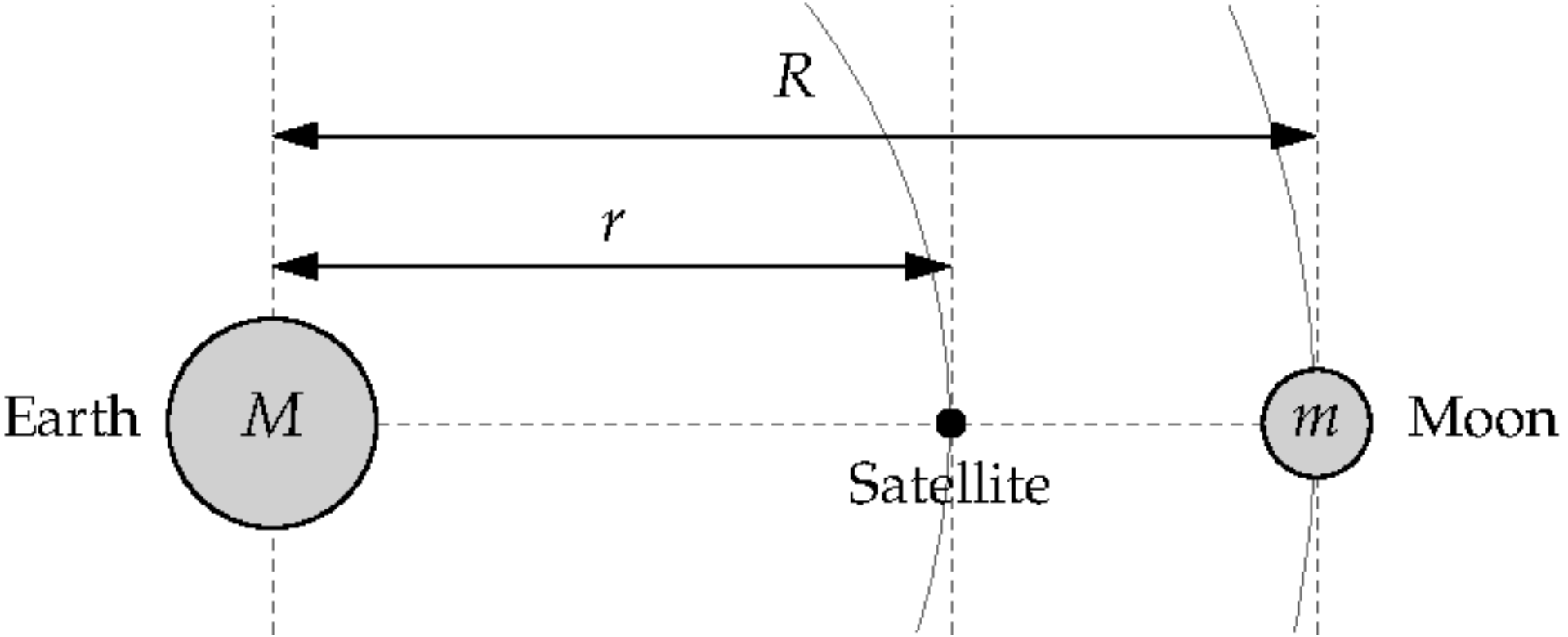
F = focus, C = center, E = eccentric anomaly

Non-linear equations. Exercise: Lagrangian point

The L1 Lagrangian point between the Earth and the Moon is the point at which the inward pull of the Earth and the outward pull of the Moon balance

to give the centripetal force that keeps a satellite in a synchronous orbit with the Moon

→ a satellite in L1 orbits the Earth in perfect synchrony with the Moon



Non-linear equations. Exercise: Lagrangian point

- Assuming circular orbit (eccentricity = 0) and
- assuming $m_{\text{Satellite}} << m_{\text{Moon}} << M_{\text{Earth}}$

(i.e. the mass of the satellite is negligible and
the mass of the Moon is negligible with respect to that of the Earth),

you can easily find that the distance *r* from the center of the Earth to the L1 point satisfies

$$\frac{G\,M_{\text{Earth}}}{r^2} - \frac{G\,m_{\text{Moon}}}{(R-r)^2} = \omega^2\,r$$

where *G* is the gravity constant (6.674e-8 cm^3/g/s^2)

M_{Earth} = 5.974e27 g

m_{Moon} = 7.348e25 g

R = 3.844e10 cm (distance between Earth and Moon)

ω = 2.662e-6 s^{-1} (angular frequency of both the Moon and the satellite)

You could have derived ω as $$\omega = \left(\frac{G\,M_{\text{Earth}}}{R^3}\right)^{1/2}$$

Solve this equation with the methods you know and estimate *r*