

Lecture 5

DIY numerical algorithms

Mathematica built-in function to find roots

- Syntax: **FindRoot**[]. **Nsolve**[].
- **Nsolve**[] find multiple solutions automatically, but may fail in certain types of equations. Best used for algebraic equations and polynomials.
- **FindRoot**[] finds only one root at a time, and needs an initial guess value. More robust than **Nsolve**[].
- Use **FindRoot**[] and **Nsolve**[] to verify your codes developed in previous exercises.

Exercise: Brute-force 'auto-detection' of roots

Partition the interval $[a, b] = [2, 10]$ of the given function

$$f(x) = 10 + x^3 - \sin x \sinh x$$

into $N=25$ sub intervals of equal length,

$$\Delta_i = [x_i, x_{i+1}], i = 0, 1, \dots, N - 1,$$

where $x_0 = a, x_N = b$. Use the command **FindRoot**[] to scan all subintervals Δ_i for the roots of the function, which obey the condition $f(x) = 0$, using an initial value in each Δ_i . Your code should then automatically print out all distinct roots lying in the interval $[a, b]$. You may have to use the command **Union**[] to render repeated roots into distinct ones.

Root finding 1: Bisectioning method

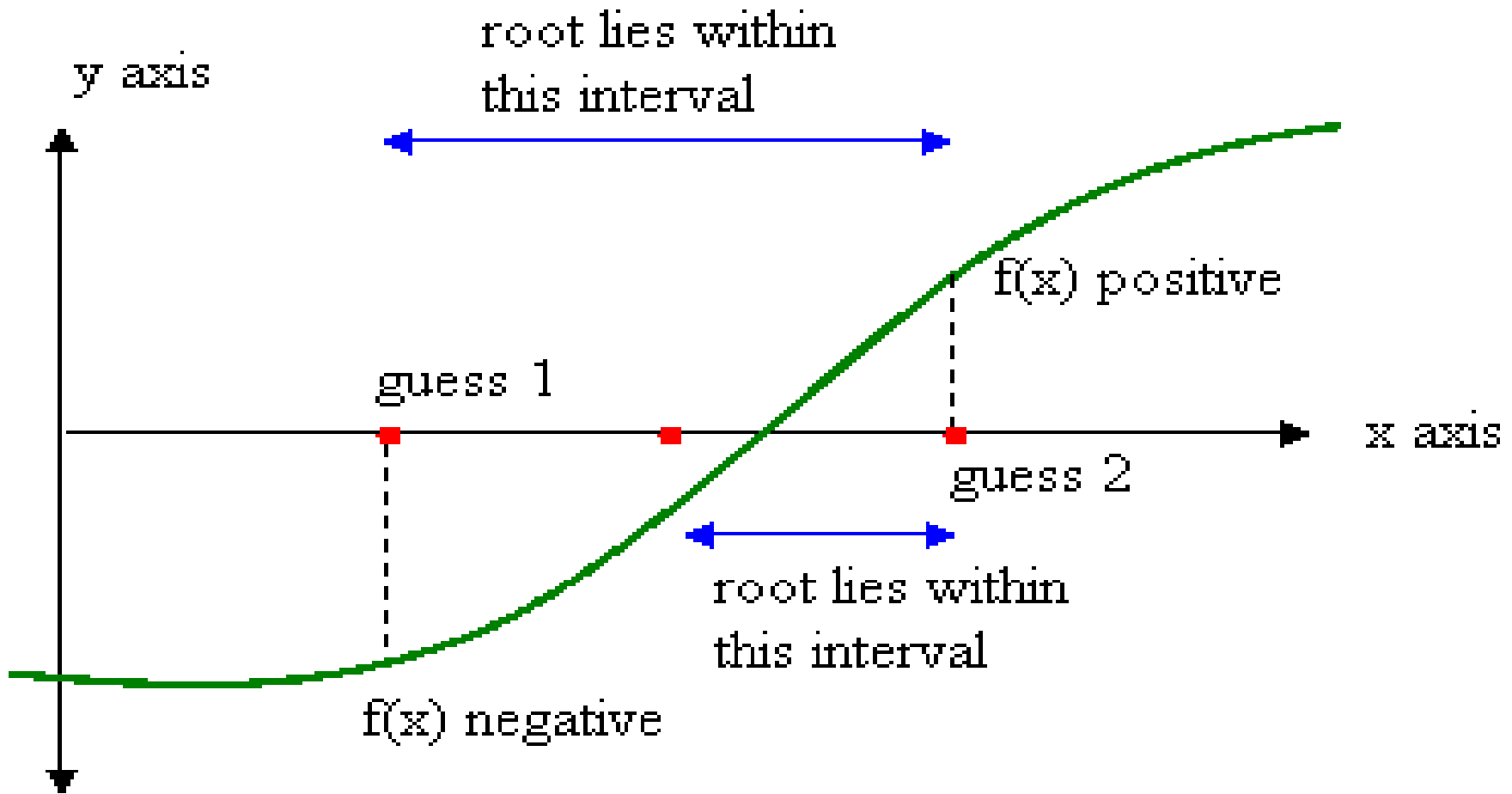
Root of a continuous function

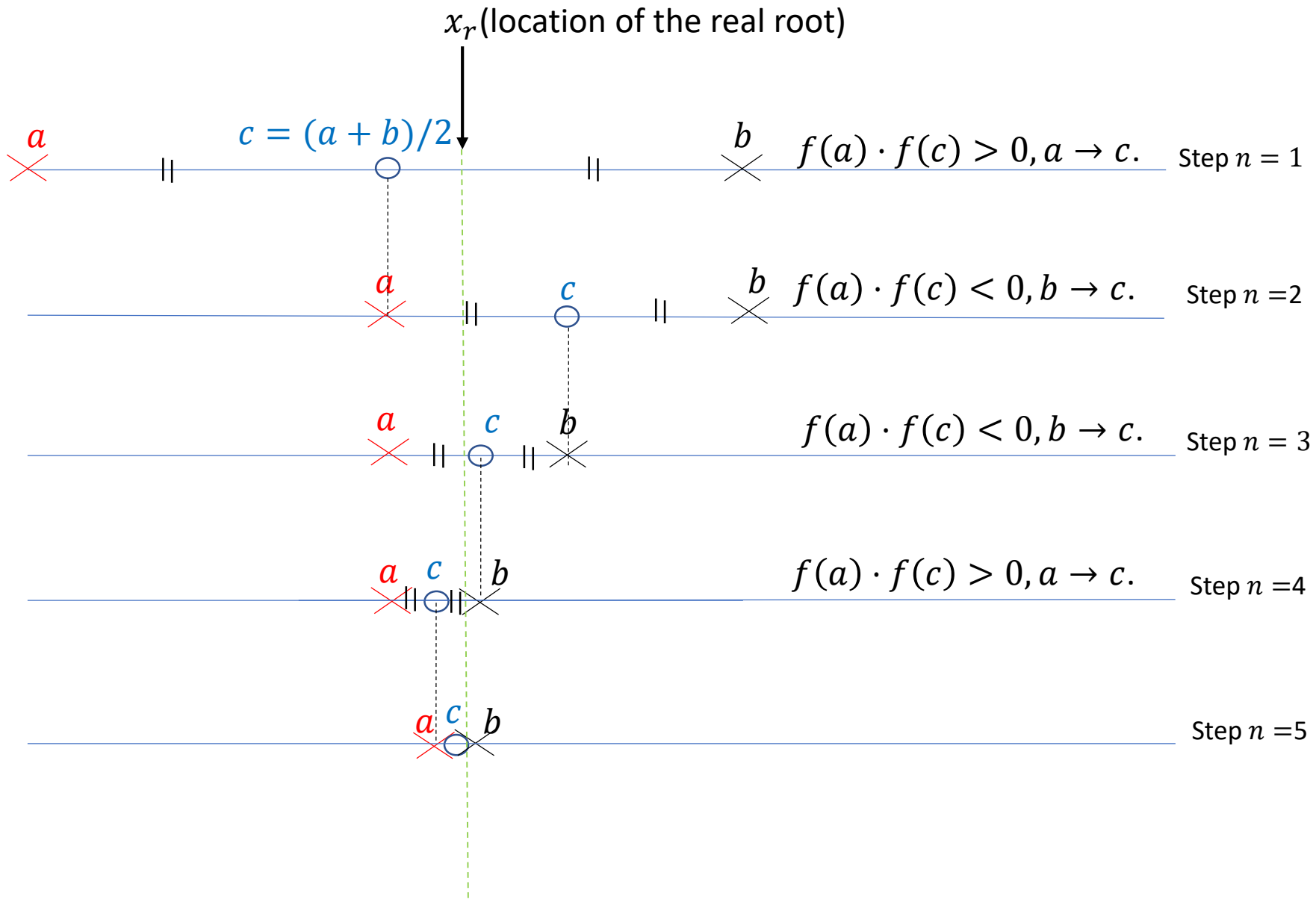
- The roots of a function $f(x)$ are defined as the values for which the value of the function becomes equal to zero.
- Finding the roots of $f(x)$ means solving the equation $f(x) = 0$.
- The value of $x = r$ such that $f(r) = 0$ is the root for the function f .
- Given a continuous function in an interval $[a, b]$, how do we find its roots?

Bisection method

- We shall refer to the lecture note by Dr. Dana Mackey, Dublin Institute of Technology:
<http://www.maths.dit.ie/~dmackey/lectures/Roots.pdf>

Bisection method



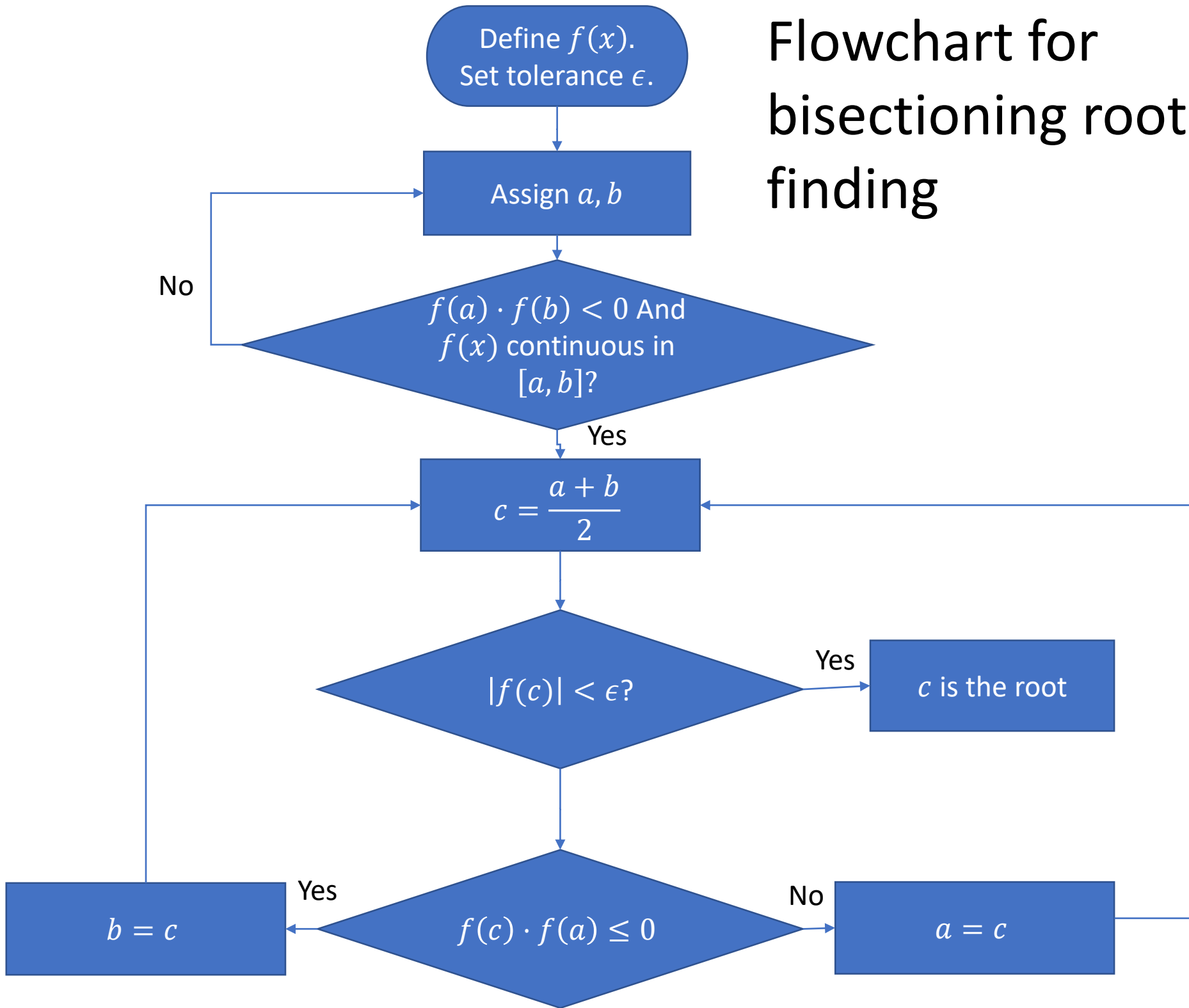


The algorithm of bisection method

Suppose we wish to find the root for $f(x)$, and we have an error tolerance of ε (the absolute error in calculating the root must be less than ε).

- Step 1: Find two numbers a and b at which f has different signs.
- Step 2: Define $c = (a + b)/2$.
- Step 3: If $|f(c)| \leq \varepsilon$ then accept c as the root and stop.
- Step 4: If $f(a) \cdot f(c) \leq 0$ then set c as the new b . Otherwise, set c as the new a . Return to step 1.

Flowchart for bisectioning root-finding



Note

- The bisectioning method only works if $f(x)$ is continuous between $[a, b]$.
- The condition " \leq " as appear in $f(a) \cdot f(c) \leq 0$ is crucial. Replacing it by " $<$ " may upset the robustness of the algorithm

Exercise

Search for the roots in the interval $[2,10]$ for the following equation:

$$10 + x^3 - \sin x \sinh x = 0$$

up to an accuracy of $\epsilon = 10^{-6}$

- The function contains two roots
- Use bisectioning method to find both roots manually.
- Implement your code in the form of a **Module[]**

Exercise: Brute-force 'auto-detection' of roots via bisectioning Module

Write a code that can automatically obtain all roots of a continuous function $f(x)$ in any given interval $[a, b]$ by making use of the DIY bisectioning Module for root-finding that you have developed in previous exercise. Test it out for the following case:

$$f(x) = 10 + x^3 - \sin x \sinh x$$

$$[a, b] = [2, 10].$$

Exercises

Obtain all roots of a continuous function $f(x)$ for the interval $[a, b]$ by making use of the DIY bisectioning Module for root-finding. Note that for the case of *i.*, *ii.*, below, you have to manually determine x_0, x_N first.

i. $f(x) = x - \text{Tanh } x$, for all x .

ii. $f(x) = x^3 + 2x^2 - 3x - 1$, for all x

iii. $f(x) = (1/x) \sin x$, for $-3\pi \leq x \leq 3\pi$.

iv. $f(x) = \tan(\pi x) - x - 6$, for $-3\pi \leq x \leq 3\pi$.

Root finding 2: Newton-Raphson (NR) method

Newton-Raphson Method

Recall that the equation of a straight line is given by the equation

$$y = mx + n \quad (1)$$

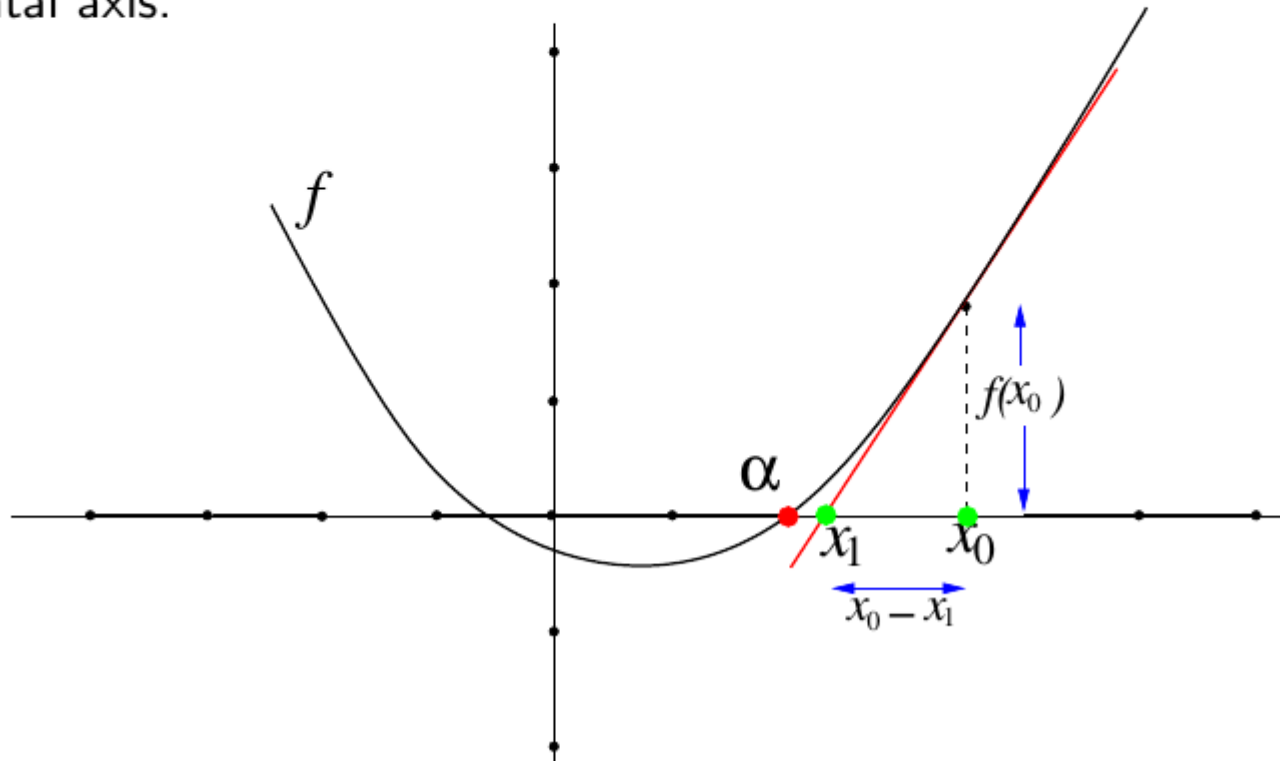
where m is called the *slope* of the line. (This means that all points (x, y) on the line satisfy the equation above.)

If we know the slope m and one point (x_0, y_0) on the line, equation (1) becomes

$$y - y_0 = m(x - x_0) \quad (2)$$

Idea behind Newton's method

Assume we need to find a root of the equation $f(x) = 0$. Consider the graph of the function $f(x)$ and an initial estimate of the root, x_0 . To improve this estimate, take the tangent to the graph of $f(x)$ through the point $(x_0, f(x_0))$ and let x_1 be the point where this line crosses the horizontal axis.



According to eq. (2) above, this point is given by

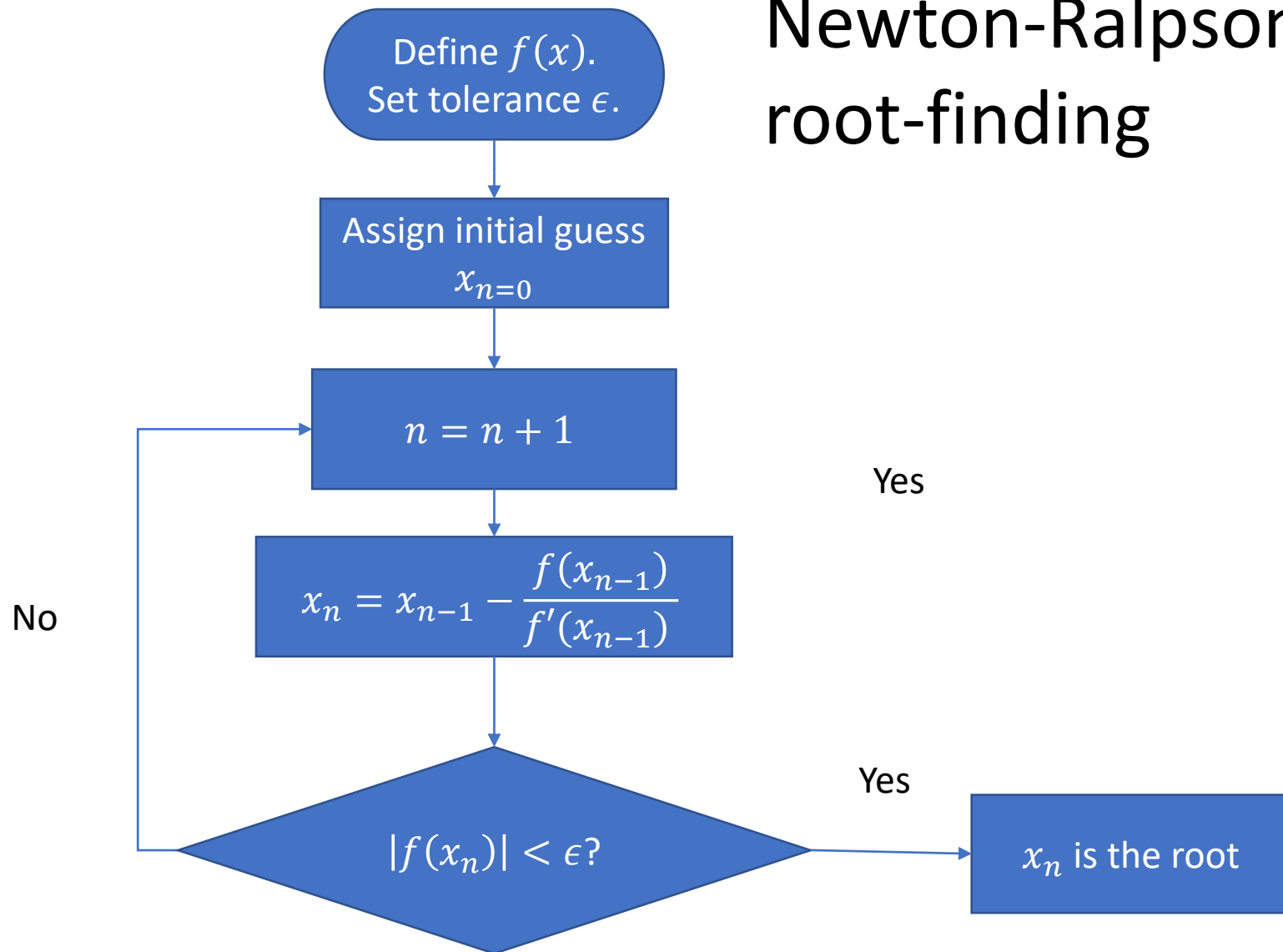
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

where $f'(x_0)$ is the derivative of f at x_0 . Then take x_1 as the next approximation and continue the procedure. The general iteration will be given by

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

and so on.

Flowchart for Newton-Ralpson root-finding



Exercise

Search for the roots in the interval $[2,10]$ for the following equation:

$$10 + x^3 - \sin x \sinh x = 0$$

up to an accuracy of $\epsilon = 10^{-6}$

- The function contains two roots
- Use NR method to find both roots manually.
- Implement your code in the form of a **Module[]**

Exercise: Brute-force 'auto-detection' of roots via NR Module

Write a code that can automatically obtain all roots of a continuous function $f(x)$ in any given interval $[a, b]$ by making use of the DIY NR Module for root-finding that you have developed in previous exercise. Test it out for the following case:

$$f(x) = 10 + x^3 - \sin x \sinh x$$

$$[a, b] = [2, 10].$$

Exercises

Obtain all roots of a continuous function $f(x)$ for the interval $[a, b]$ by making use of the DIY NR Module for root-finding. Note that for the case of *i.*, *ii.*, below, you have to manually determine x_0, x_N first.

i. $f(x) = x - \text{Tanh } x$, for all x .

ii. $f(x) = x^3 + 2x^2 - 3x - 1$, for all x

iii. $f(x) = (1/x) \sin x$, for $-3\pi \leq x \leq 3\pi$.

iv. $f(x) = \tan(\pi x) - x - 6$, for $-3\pi \leq x \leq 3\pi$.

You are now a powerful root finder

After all these exercises, now you should be confident to proclaim to the whole world that:

Give me any single variable continuous function, and I'll find you their roots at a click. =)

Assignment 14

Q1. Develop a code that could auto detect all the roots for the following function in the given interval by using built-in root finding command **FindRoot[]**:

$$f(x) = P_{n=4}(x)$$

- $P_{n=4}(x)$ is the Legendre polynomial with order $n = 4$.
- Interval = $[-1,1]$.
- You can use the command **Legendre[n, x]** to generate the function.

Q2. Repeat Q1, but use instead the DIY Module package you have coded yourself using the bisectioning method.

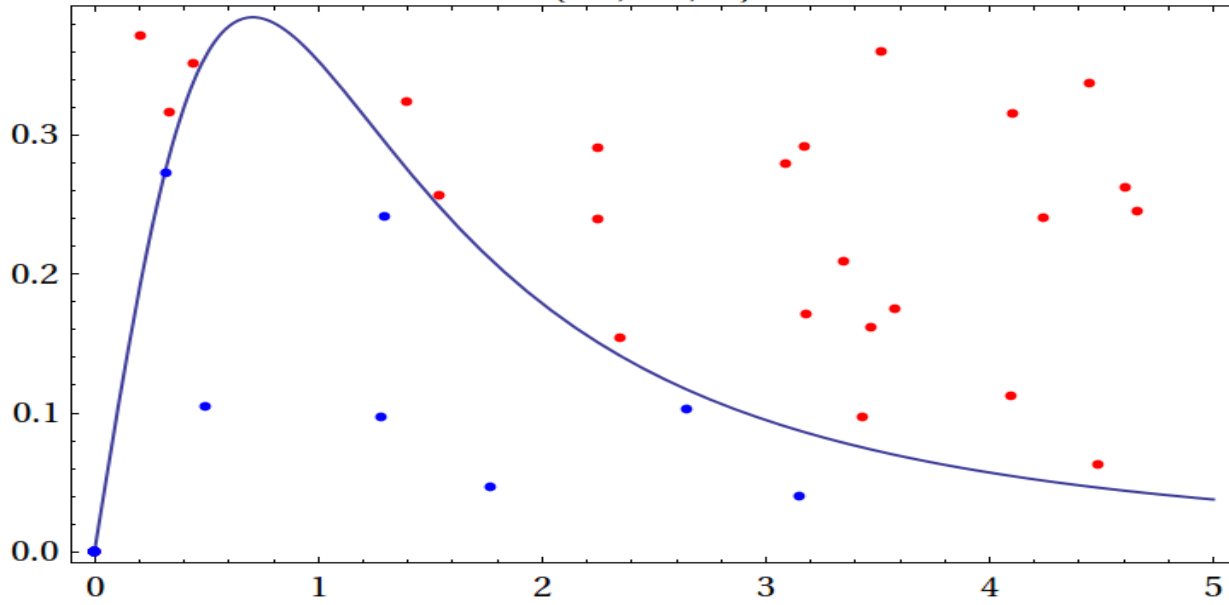
Q3. Repeat Q1, but use instead the DIY Module package you have coded yourself using the Newton-Raphson method.

Numerical integration: Stochastic method

Numerical integration with stochastic method

- Assume $f(x) \geq 0$ for $x_{\text{init}} \leq x \leq x_{\text{last}}$.
- Set totalcountmax .
- Set $\text{inboxcount} = 0$, $\text{totalcount} = 0$.
- Find $\text{Max } f(x)$, and call it f_{max} .
- Define an area $A = f_{\text{max}} * L$, where $L = x_{\text{last}} - x_{\text{init}}$.
- 1. Generate a pair of random numbers $(x_{\text{rand}}, y_{\text{rand}})$, with the condition
 - $x_{\text{init}} \leq x_{\text{rand}} \leq x_{\text{last}}$, $0 \leq y_{\text{rand}} \leq f_{\text{max}}$.
- 2. $\text{totalcount} = \text{totalcount} + 1$.
- 3. If $0 \leq y_{\text{rand}} \leq f(x_{\text{rand}})$, $\text{inboxcount} = \text{inboxcount} + 1$
- 4. Stop if $\text{totalcount} = \text{totalcountmax}$.
- 5. Repeat step 1.
- The area of the curve $f(x)$ in the interval for $[x_{\text{init}}, x_{\text{last}}]$ is given by
- $\text{Area} = A * \text{inboxcount} / \text{totalcountmax}$.

{30, 23, 7}



Z=1

A=L*fmax=1.9245

L=5-0=5.0

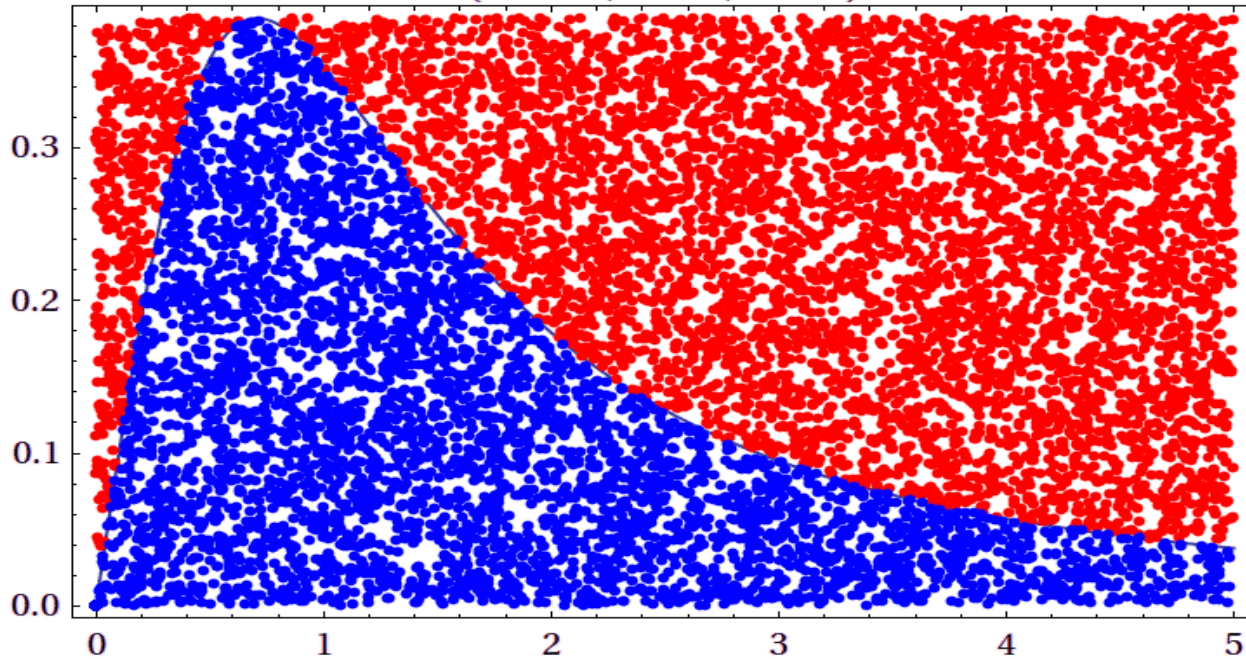
fmax=0.3849

at x=0.707107

$$f(x) = \frac{x}{(z^2 + x^2)^{3/2}}$$

$$\int_{x_0}^{x_1} f(x) dx = ?$$

{10 000, 5668, 4332}



Exercise

- Write a code to evaluate the following integral using stochastic method. z is a constant set to 1.
- Let the integration limits be from $x_0=0$ to $x_1=+5.0$.

$$f(x) = \frac{x}{(z^2 + x^2)^{3/2}}$$
$$\int_{x_0}^{x_1} f(x) dx = ?$$

Trapezoid rule for integration

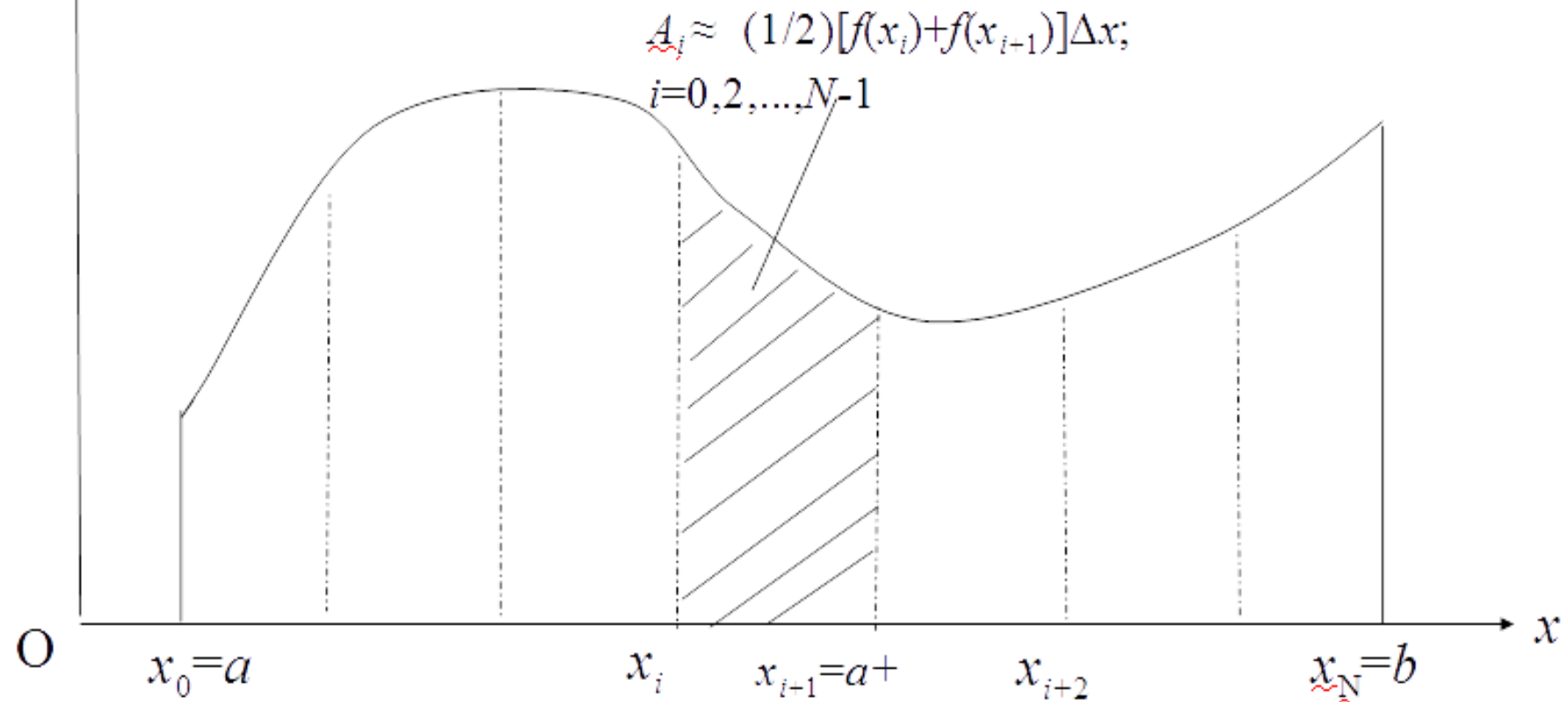
$$\int_{x_0}^{x_1} f(x) dx$$

Many methods can be used to numerically evaluate the integral

Basically the integral is the area represented between the curve and the vertical axis.

Trapezoid rule for integration

Basically the integral is the area represented between the curve and the x-axis.



There is a total of N subintervals, $[x_0, x_1], [x_1, x_2], [x_2, x_3], \dots, [x_{N-2}, x_{N-1}], [x_{N-1}, x_N]$.
 A_i represents the area under the curve in subinterval i , $\Delta_i = [x_i, x_{i+1}]$

Trapezoidal rule

$$\begin{aligned} \int_a^b f(x) dx &\approx \sum_{i=0}^{i=N-1} A_i = \sum_{i=0}^{i=N-1} \frac{1}{2} \Delta x [f(x_{i+1}) + f(x_i)] \\ &= \Delta x \left\{ \frac{1}{2} [f(x_0) + f(x_1)] + \frac{1}{2} [f(x_1) + f(x_2)] + \dots + \frac{1}{2} [f(x_{N-1}) + f(x_N)] \right\} \\ &= \frac{\Delta x}{2} [f(x_0) + f(x_N)] + \Delta x [f(x_1) + f(x_2) + \dots + f(x_{N-2}) + f(x_{N-1})] \\ &= \frac{\Delta x}{2} [f(x_0) + f(x_N)] + \Delta x \sum_{i=1}^{i=N-1} f(x_i) \end{aligned}$$

The error, is of the order $O(\Delta x)^2$

Simpson's rule for integration

- The numerical integration can be improved by treating the curve connecting the points $\{x_{i+1}, f(x_{i+1})\}, \{x_i, f(x_i)\}$ as a section of a parabola instead of a straight line (as was assumed in trapezoid rule).
- This results in $A_i + A_{i+1} = (\Delta x/3)[f(x_i) + 4f(x_{i+1}) + f(x_{i+2})]$.
- For details of the derivation, see the lecture notes by Gilles Cazalais of Camosun College, Cadana,
- <http://pages.pacificcoast.net/~cazelais/187/simpson.pdf>

Simpson's rule for integration (cont.)

$$\begin{aligned}\int_a^b f(x) dx &= \sum_{i=0,2,4,6,\dots,N-2} A_i + A_{i+1} = \frac{\Delta x}{3} \sum_{i=0,2,4,6,\dots,N-2} f(x_i) + 4f(x_{i+1}) + f(x_{i+2}) \\ &= \frac{\Delta x}{3} [\{f(x_0) + 4f(x_{0+1}) + f(x_{0+2})\} + \{f(x_2) + 4f(x_{2+1}) + f(x_{2+2})\} \\ &\quad + \{f(x_4) + 4f(x_{4+1}) + f(x_{4+2})\} + \dots + \{f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)\}] \\ &= \frac{\Delta x}{3} [\{f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) \\ &\quad + 4f(x_5) + 2f(x_6) + \dots + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)\}] \end{aligned}$$

Assume N a large even number.

Simpson's rule for integration

$$\int_a^b f(x) dx = \frac{\Delta x}{3} [f(x_0) + f(x_N)] + \frac{4\Delta x}{3} [f(x_1) + f(x_3) + f(x_5) + \dots + f(x_{N-1})] \\ + \frac{2\Delta x}{3} [f(x_2) + f(x_4) + f(x_6) + \dots + f(x_{N-2})]$$

The number of interval, N , matters: if it is too small, large error occurs. The error in Simpson's rule is of the order $O(\Delta x)^4$

Exercise

Write a code to evaluate the following integral using both Trapezoid and Simpson's rule. z is a constant set to 1.

Let the integration limits be from $x_0 = -1.5$ to $x_1 = +5.0$.

$$f(x) = \frac{x}{(z^2 + x^2)^{3/2}}$$

$$\int_{x_0}^{x_1} f(x) dx = ?$$

Numerical solutions for first order differential equation: Euler's method

Refer to http://www.cse.salford.ac.uk/physics/gsmcdonald/pp/PPLATOResources/h-flap/M6_2t.pdf

Examples of first order differential equation commonly encountered in physics

$$\frac{dv_y}{dt} = -g;$$

$$\frac{dy}{dt} = v_y$$

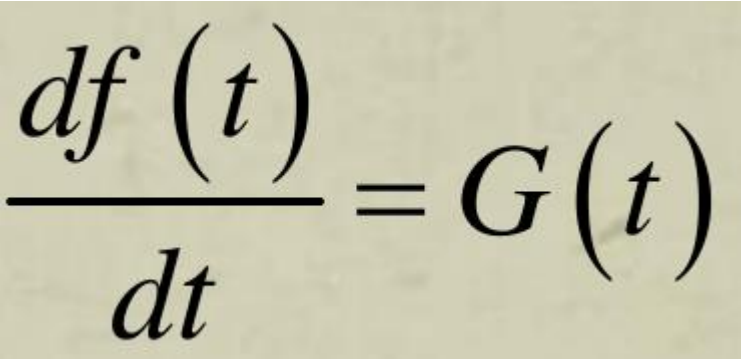
$$m \frac{dv}{dt} = -mg - \eta v$$

$$\frac{dN}{dx} = -\lambda N;$$

$$\frac{dN}{dt} = -\frac{N}{\tau}$$

$$m \frac{dv}{dx} = -kx$$

General form of first order differential equations


$$\frac{df(t)}{dt} = G(t)$$

Now, how would you write your own algorithm to solve first order differential equations numerically?

$$\frac{df(t)}{dt} = G(t)$$

Euler's method for discreteising a first order differential equation into a difference equation

Discretising the differential equation

In Euler method, where the differentiation of a function at time t is approximated as

$$\frac{df(t)}{dt} \simeq \frac{f(t+\Delta t) - f(t)}{\Delta t} = \frac{f(t_{i+1}) - f(t_i)}{\Delta t}, \quad t_i = i\Delta t, \quad t_{i+1} - t_i = \Delta t,$$
$$\frac{df(t_i)}{dt} \simeq \frac{f(t_{i+1}) - f(t_i)}{\Delta t}; t_i = i\Delta t \Rightarrow f(t_{i+1}) \simeq f(t_i) + \frac{df(t_i)}{dt} \Delta t = f(t_i) + G(t_i) \Delta t$$

Essentially, Euler's method says

Differential equation $\frac{df(t)}{dt} = G(t)$

Discretise

$$f(t_{i+1}) \approx f(t_i) + G(t_i) \Delta t$$

Difference equation

Boundary condition: The numerical value at $f(t_0)$ has to be supplied.

Discretising the differential equation

$$\frac{dN(t)}{dt} = -\frac{N(t)}{\tau} \quad \longleftrightarrow \quad \frac{df(t)}{dt} = G(t)$$

c.f

• $\frac{dN(t)}{dt} = -\frac{N(t)}{\tau}$ is discretised into a difference equation,

•
$$N(t + \Delta t) \approx N(t) - \frac{N(t)}{\tau} \Delta t$$

$$\Rightarrow N(t_{i+1}) \approx N(t_i) - \frac{N(t_i)}{\tau} \Delta t$$

c.f

$$f(t_{i+1}) \approx f(t_i) + G(t_i) \Delta t$$

$$f(t_i) + G(t_i)$$

which is suitable for numerical manipulation using computer.

- There are many different way to discretise a differential equation.
- Euler's method is just among the easiest of them.

The code's structure

- Initialisation:
- Assign (i) $N(t=0)$, τ , (ii) Number of steps, Nstep, (iii) Time when to stop, say $t_{\text{final}} = 10\tau$.
- The global error is of the order $O \sim \Delta t$
- $\Delta t = t_{\text{final}}/N_{\text{step}}$
- In principle, the finer the time interval Δt is, the numerical solution becomes more accurate.
- $t_i = t_0 + i\Delta t$; $i=0,1,2,\dots,t_{\text{final}}$
- Calculate $N(t_1) = N(t_0) - \Delta t N(t_0)/\tau$.
- Then calculate $N(t_2) = N(t_1) - \Delta t N(t_1)/\tau$
- $N(t_3) = N(t_2) - \Delta t N(t_2)/\tau, \dots$
- Stop when $t = t_{\text{final}}$.
- Plot the output: $N(t)$ as function of t .

Exercise

- A freely falling object through a fluid medium can alternatively be modeled such that the drag force is proportional to its speed. The first order differential equation for such an object is given by

$$\frac{dv}{dt} = -g - \frac{k}{m}v$$

- Let $k=0.01$, $m=1$; $g=9.81 \text{ m/s}^2$; and the boundary condition is $v(0) = 0 \text{ m/s}$.
- Use **NDSolve**[] to obtain and plot the numerical solution for $v(t)$ until it enters a terminal velocity.
- Develop a code that implements Euler's method to numerically solve the equation.
- Overlap your numerical solution on top of the analytically obtained plot. Both should agree to each other.

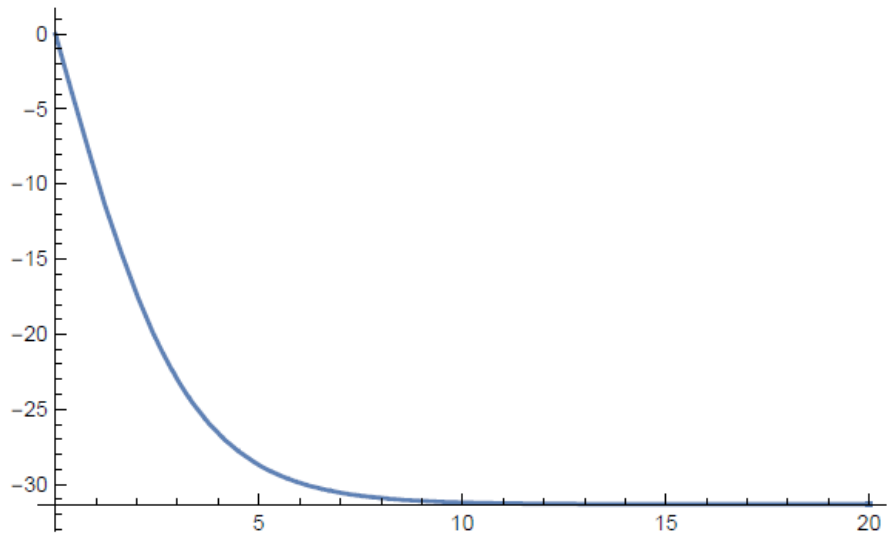
Exercise

- A freely falling object through a fluid medium can alternatively be modeled such that the drag force is proportional to the square of its speed. The first order differential equation for such an object is given by

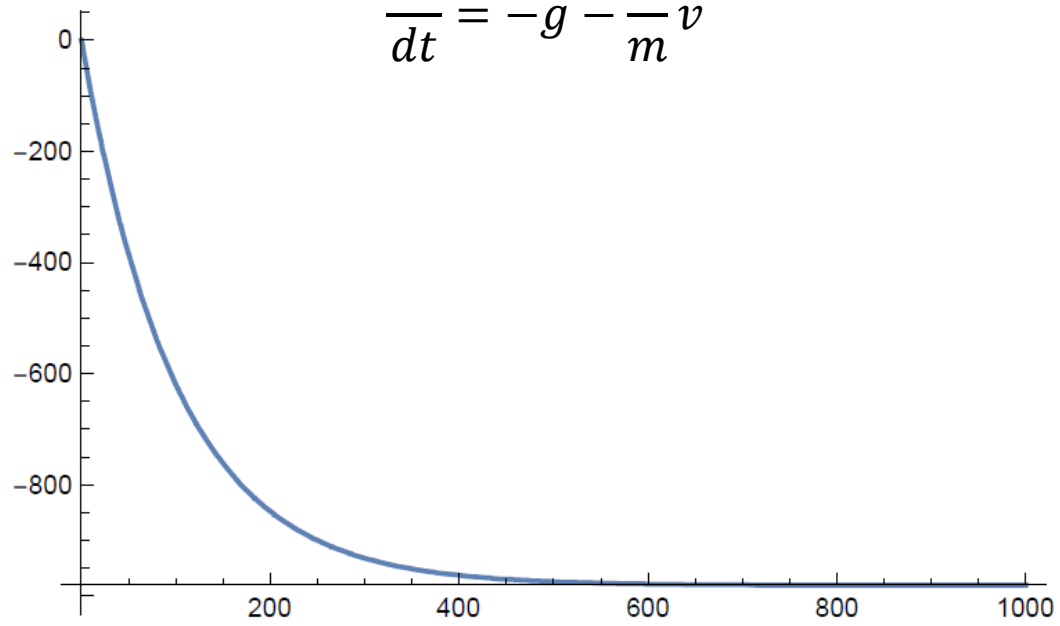
$$\frac{dv}{dt} = -g + \frac{k}{m}v^2$$

- Let $k=0.01$, $m=1$; $g=9.81$ m/s²; and the boundary condition is $v(0) = 0$ m/s.
- Use **NDSolve**[] to obtain and plot the numerical solution for $v(t)$ until it enters a terminal velocity.
- Develop a code that implements Euler's method to numerically solve the equation.
- Overlap your numerical solution on top of the analytically obtained plot. Both should agree to each other.

$$\frac{dv}{dt} = -g + \frac{k}{m}v^2$$



$$\frac{dv}{dt} = -g - \frac{k}{m}v$$



Exercise

1D pendulum is governed by the equation

$$m \frac{dv}{dx} = -kx$$

- Use Euler method to plot the solution for this 1D first order differential equation.
- Set $m = k = 1$.

Numerical solutions for second order differential equation: Second order Runge-Kutta method (RK2)

EXAMPLES

$$\frac{d^2\theta(t)}{dt^2} = -\frac{g\theta}{l}$$

$$\frac{d^2y(t)}{dt^2} = -g - \frac{k}{m} \frac{dy}{dt}$$

$$\frac{d^2\theta(t)}{dt^2} = -\frac{g\theta}{l} - q \frac{d\theta}{dt}$$

$$\frac{d^2y(t)}{dt^2} = -g + \frac{k}{m} \left(\frac{dy}{dt}\right)^2$$

Second order Runge-Kutta (RK2) method

Consider a generic second order differential equation.

$$\frac{d^2 u(x)}{dx^2} = G(u)$$

It can be numerically solved using second order Runge-Kutta method.

Split the second order DE into two first order parts:

$$v(x) = \frac{du(x)}{dx} \qquad \frac{dv(x)}{dx} = G(u)$$

Algorithm

Set boundary conditions: $u(x=x_0)=u_0$, $u'(x=x_0)=v(x=x_0)=v_0$.

calculate

$$\tilde{u} = u_i + \frac{1}{2} v_i \Delta x$$

calculate

$$\tilde{v} = v_i + \frac{1}{2} G(\tilde{u}) \Delta x$$

calculate

$$u_{i+1} = u_i + \tilde{v} \Delta x$$

calculate

$$v_{i+1} = v_i + G(\tilde{u}) \Delta x$$

Translating the SK2 algorithm into the case of simple pendulum

$$\frac{d^2 u(x)}{dx^2} = G(u)$$

$$v(x) = \frac{du(x)}{dx}$$

$$\frac{dv(x)}{dx} = G(u)$$

Set boundary conditions:

$$u(x=x_0)=u_0, u'(x=x_0)=v(x=x_0)=v_0$$

$$\tilde{u} = u_i + \frac{1}{2} v_i \Delta x$$

$$\tilde{v} = v_i + \frac{1}{2} G(\tilde{u}) \Delta x$$

$$u_{i+1} = u_i + \tilde{v} \Delta x$$

$$v_{i+1} = v_i + G(\tilde{u}) \Delta x$$

$$G(u) \equiv -\frac{g}{l} \theta(t)$$

$$\frac{d^2 \theta(t)}{dt^2} = -\frac{g\theta}{l}$$

$$\omega(t) = \frac{d\theta(t)}{dt}$$

$$\frac{d\omega(t)}{dt} = -\frac{g\theta(t)}{l}$$

Set boundary conditions:

$$\theta(t=t_0) = \theta_0, \theta'(t=t_0) = \omega(t=t_0) = \omega_0$$

$$\tilde{\theta} = \theta_i + \frac{1}{2} \omega_i \Delta t$$

$$\tilde{\omega} = \omega_i + \frac{1}{2} \left(-\frac{g\tilde{\theta}}{l} \right) \Delta t$$

$$\theta_{i+1} = \theta_i + \tilde{\omega} \Delta t$$

$$\omega_{i+1} = \omega_i + \left(-\frac{g\tilde{\theta}}{l} \right) \Delta t$$

Exercise: Develop a code to implement SK2 for the case of the simple pendulum.

Boundary conditions: $\omega(0) = \sqrt{\frac{g}{l}}; \theta(0) = 0$

Exercise:

Develop a code to implement SK2 for the case of a pendulum experiencing a drag force, with damping coefficient $q = 0.1 * (4\Omega)$, $\Omega = \sqrt{g/l}$, $l = 1.0$ m.
Boundary conditions: $\theta(0) = 0.2$; $\omega(t = 0) = 0$;

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta - q\frac{d\theta}{dt}$$

Translating the SK2 algorithm into the case of damped pendulum

$$\frac{d^2 u(x)}{dx^2} = G(u)$$

$$v(x) = \frac{du(x)}{dx}$$

$$\frac{dv(x)}{dx} = G(u)$$

$$G(u) \equiv -\frac{g}{l}\theta(t) - q\omega(t)$$

$$\frac{d^2 \theta(t)}{dt^2} = -\frac{g\theta}{l} - q \frac{d\theta}{dt}$$

$$\omega(t) = \frac{d\theta(t)}{dt}$$

$$\frac{d\omega(t)}{dt} = -\frac{g\theta}{l} - q\omega(t)$$

Set boundary conditions:

$$u(x=x_0)=u_0, u'(x=x_0)=v(x=x_0)=v_0$$

$$\tilde{u} = u_i + \frac{1}{2} v_i \Delta x$$

$$\tilde{v} = v_i + \frac{1}{2} G(\tilde{u}) \Delta x$$

$$u_{i+1} = u_i + \tilde{v} \Delta x$$

$$v_{i+1} = v_i + G(\tilde{u}) \Delta x$$

Set boundary conditions:

$$\theta(t=t_0) = \theta_0, \theta'(t=t_0) = \omega(t=t_0) = \omega_0$$

$$\tilde{\theta} = \theta_i + \frac{1}{2} \omega_i \Delta t$$

$$\tilde{\omega} = \omega_i + \frac{1}{2} \left(-\frac{g}{l} \tilde{\theta}(t) - q \tilde{\omega} \right) \Delta t \Rightarrow \tilde{\omega} = \frac{\left(\omega_i - \frac{g}{2l} \tilde{\theta}(t) \Delta t \right)}{\left(1 + \frac{1}{2} q \Delta t \right)}$$

$$\theta_{i+1} = \theta_i + \tilde{\omega} \Delta t$$

$$\omega_{i+1} = \omega_i + \left(-\frac{g\tilde{\theta}}{l} - q\tilde{\omega} \right) \Delta t$$

Exercise: Stability of the total energy a SHO in RK2.

$\omega = \frac{d\theta}{dt}$, angular velocity. $m=1$ kg; $l=1$ m.

The total energy of the SHO in can be calculated as

$$\begin{aligned} E_{i+1} &= K_{i+1} + U_{i+1} = \frac{1}{2}m(l\omega_{i+1})^2 + mgl(1 - \cos \theta_{i+1}) \\ &\approx \frac{1}{2}ml^2\omega_{i+1}^2 + mgl \left[1 - \left(1 - \frac{\theta_{i+1}^2}{2} \right) \right] \\ &= \frac{1}{2}ml^2\omega_{i+1}^2 + \frac{1}{2}mgl\theta_{i+1}^2 \end{aligned}$$

User your RK2 code to track the total energy for t running from $t=0$ till $t=25T$; $T=2\pi\sqrt{l/g}$. Boundary conditions: $\omega(0) = \sqrt{\frac{g}{l}}$; $\theta(0) = 0$
 E_i should remain constant throughout all t_i .