# **Numerical Methods** | Implementations in C++

Thanasis Mattas
Aristotle University of Thessaloniki, Greece | 2019
atmattas@physics.auth.gr

## Fixed point iteration schemes

1. **Picard method** $[x_{n+1} = g(x_n)]$

Example: $f(x) = e^{2x} - 3x - 1 \Rightarrow x = \frac{e^{2x}-1}{3}$ with $g(x) = \frac{e^{2x}-1}{3} \Rightarrow g'(x) = \frac{2}{3}e^{2x}$

Regression formula:

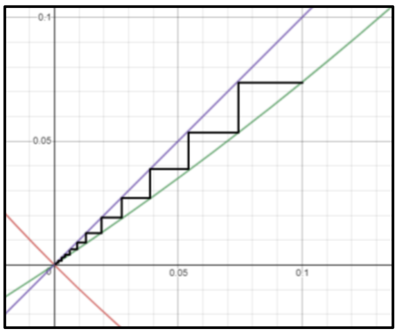
$$\boxed{x_{n+1} = \frac{e^{2x_n} - 1}{3}}$$

Convergence rate of the error:

$$\boxed{e_{n+1} = g'(r)e_n}$$

Convergence condition:

$$\boxed{|g'(x)| < 1 \Rightarrow x < 0.20273}$$



*desmos.com*

For the solution you can refer to **Picard_method.cpp** at the src directory ($x_0 = 0.1$).

2. **Newton-Raphson method** $\left[x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\right]$

In order to compare the convergence speed against the Picard method, the same example is used:

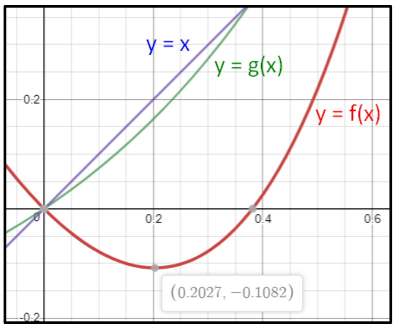
$$f(x) = e^{2x} - 3x - 1$$

Regression formula:

$$\boxed{x_{n+1} = x_n - \frac{e^{2x_n} - 3x - 1}{2e^{2x_n} - 3}}$$

Convergence rate of the error:

$$\boxed{e_{n+1} = -\frac{f''(r)}{2f'(r)}e_n^2}$$



*desmos.com*

2.b Selecting $x_0 = 0.203$

$x = 0.20273$ is the total extremum (min) of the equation $f(x)$. So, starting points that lie before the extremum lead to the root $x = 0$, and fixed points that lie after the extremum lead to the other root of the equation, $x = 0.3813$.

For the solution you can refer to **Newton-Raphson_method.cpp** at the src directory.

**3. <u>Newton method with simultaneous equations</u>** $\begin{bmatrix} x_{n+1} = x_n - \frac{fg_x - gf_y}{f_x g_y - g_x f_y} \\ y_{n+1} = y_n - \frac{gf_x - fg_y}{f_x g_y - g_x f_y} \end{bmatrix}$ $\left(f_i = \frac{\partial f}{\partial i}\right)$

Example: $\left.\begin{matrix} x^2 + y^2 = 3 \to f(x) = x^2 + y^2 - 3 \\ 3x^2 - 9y^2 = 6 \to g(x) = 3x^2 - 9y^2 - 6 \end{matrix}\right\} \Rightarrow \begin{matrix} f_x = 2x,\ f_y = 2y \\ g_x = 6x,\ g_y = -18y \end{matrix}$

Regression formulas:

$$\boxed{\begin{matrix} x_{n+1} = x_n - \frac{4x^2 - 1}{8x} \\ y_{n+1} = y_n - \frac{4y^2 - 1}{8y} \end{matrix}}$$

For the solution you can refer to **Newton_method-Simultaneous_Equations.cpp** at the src directory.

4. **<u>Picard method with simultaneous equations</u>** $\begin{bmatrix} x_{n+1} = f_1(x_n, y_n) \\ y_{n+1} = f_2(x_n, y_n) \end{bmatrix}$

Using the same example as above:

$$\boxed{x_{n+1} = \sqrt{3 - y_n^2}} \qquad \boxed{y_{n+1} = \frac{\sqrt{x_n^2 - 2}}{3}}$$

Convergence conditions:

$$\boxed{\left|\frac{\partial f_1(y)}{\partial y}\right| < 1 \Rightarrow y_n > \sqrt{\frac{3}{2}}} \qquad \boxed{\left|\frac{\partial f_2(x)}{\partial x}\right| < 1 \Rightarrow x > \sqrt{3}}$$

The solutions estimated with Newton's method are out of the limits of these conditions and, consequently, no starting point can lead to convergence. Note that the Picard method needs a good approximation of the solution (possibly acquired with another method), in order to meet convergence.

For the implementation you can refer to **Picard_method-Simultaneous_Equations.cpp** at the src directory.

5. **<u>Gauss-Seidel method</u>** $\left[x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{N} a_{ij} x_j^{(k)}\right)\right]$

Example: $\left.\begin{matrix} 20x + y - 2z = 17 \\ 3x + 20y + z = -18 \\ 2x - 3y + 20z = 25 \end{matrix}\right\}$ $or$ $\begin{bmatrix} 20 & 1 & -2 \\ 3 & 20 & 1 \\ 2 & -3 & 20 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 17 \\ -18 \\ 25 \end{bmatrix}$

Regression formulas:

$$\boxed{\begin{matrix} x^{(k+1)} = \frac{17 - y^{(k)} + 2z^{(k)}}{20} \\ y^{(k+1)} = -\frac{18 + 3x^{(k+1)} + z^{(k)}}{20} \\ z^{(k+1)} = \frac{25 - 2x^{(k+1)} + 3y^{(k+1)}}{20} \end{matrix}}$$

Obvious solution: (1, -1, 1)  Initial values: $x^{(0)} = 2,\ y^{(0)} = 0,\ z^{(0)} = 2$

Convergence condition:

$$\boxed{|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{N} |a_{ij}| \quad , i = 1, 2, \dots, N}$$

Vectorized: $X^{(k+1)} = D^{-1}\left(B - LX^{(k+1)} - UX^{(k)}\right)$
$\quad with\ L{:}\ lower, D{:}\ diagonal, U{:}\ upper\ and\ A = L + D + U$

For the solution you can refer to **Gauss-Seidel_method.cpp** at the src directory.

## 6. Power method

Example: $\begin{bmatrix} 6 & 0 & 2 & 3 & 2 & 4 \\ 4 & 1 & 8 & 0 & 3 & 5 \\ 7 & 3 & 3 & 2 & 9 & 0 \\ 4 & 0 & 0 & 2 & 6 & 1 \\ 1 & 6 & 3 & 4 & 5 & 6 \\ 2 & 8 & 4 & 3 & 9 & 0 \end{bmatrix}$

Greatest eigenvalue of a square matrix:

$$\boxed{\frac{\vec{x}^{(k+1)}}{\vec{x}^{(k)}} = \frac{A^{k+1}\vec{x}}{A^{k}\vec{x}} \approx \frac{\lambda_1^{k+1} a_1 \vec{u}^{(1)}}{\lambda_1^{k} a_1 \vec{u}^{(1)}} \to \lambda_1}$$

Where,

A: square matrix
$\vec{x}$: arbitrary vector (example: $\vec{x} = (1, 1, 1, 1, 1, 1)^T$ )
$\vec{x}^{(k)} = A^k \vec{x}$
$\lambda_1$: greatest eigenvalue
$\vec{u}^{(1)}$: eigenvector that corresponds to $\lambda_1$

Corresponding Eigenvector (normalized):

$$\boxed{\vec{u}^{(1)} \approx \frac{\vec{x}^{(k+1)}}{greatest\ element}}$$

For the implementation you can refer to **Power_method.cpp** at the src directory.

## 7. Simpson method - Numerical integration $\left[I = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n)\right]$

Example: $\int_0^{4\pi} e^{x-10} \sin(10x)\, dx$

Range: 4π

Discretization step: $h = \frac{range}{points-1}$

➢ 3-decimal digit precision after 56 iterations, dividing the range with 113 points

For the implementation you can refer to **Simpson_method.cpp** at the src directory.

## 8. Runge-Kutta method - 2nd & 4th order (ODE evaluation)

Example: $y'' + \omega^2 y = 0$ (harmonic oscillator)

$\omega = 4$
Initial values: $y(0) = 1,\ y'(0) = 0$
Step: $h = 0.1$
Boundaries: $[0,6]$

- Analytical solution (true value):

$$\text{harmonic oscillator:} \quad \left.\begin{array}{r} y(x) = a\sin(\omega x + y) \\ y(0) = 1 \\ y'(0) = 0 \end{array}\right\} \Rightarrow \begin{array}{c} \varphi = \frac{\pi}{2} \\ a = 1 \\ y(x) = \sin(4x + \frac{\pi}{2}) \end{array} \Rightarrow y(0.4) = -0.0292$$

- Numerical solution (2nd order DE mutates into a system of 2 1st order DE):

$$\boxed{\left.\begin{array}{r} y' = \dfrac{dy}{dx} = z = f(x,y,z) \\ z' = \dfrac{dz}{dz} = -16y = g(x,y,z) \end{array}\right\}}$$

Runge-Kutta 2nd order:

$$\left.\begin{array}{r} y_{n+1} = y_n + \dfrac{1}{2}(k_1 + k_2) \\ z_{n+1} = z_n + \dfrac{1}{2}(l_1 + l_2) \end{array}\right\}$$

$k_1 = hf(x_n, y_n, z_n) = hz = hy'_n$

$l_1 = hg(x_n, y_n, z_n) = -16hy_n$

$k_2 = hf(x_n + h, y_n + k_1, z_n + l_1) = h(z_n + l_1) = h(y'_n - 16hy_n)$

$l_2 = hg(x_n + h, y_n + k_1, z_n + l_1) = -16h(y_n + hy'_n)$

Regression formulas:

$$\boxed{\left.\begin{array}{l} y_{n+1} = 0.923077y_n + 0.096154y'_n \\ y'_{n+1} = 0.923077y'_n - 1.538462y_n \end{array}\right\}}$$

Runge-Kutta 4th order:

$$\left.\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ z_{n+1} &= z_n + \frac{1}{6}(l_1 + 2l_2 + 2l_3 + l_4) \end{aligned}\right\}$$

$k_1 = hf(x_n, y_n, z_n) = hz = 0.1y_n'$

$l_1 = hg(x_n, y_n, z_n) = -1.6y_n$

$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1\right) = h\left(z_n + \frac{1}{2}l_1\right) = 0.1y_n' - 0.08y_n$

$l_2 = hg\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1, z_n + \frac{1}{2}l_1\right) = -16h\left(y_n + \frac{1}{2}hy_n'\right) = -1.6y_n - 0.08y_n'$

$k_3 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2, z_n + \frac{1}{2}l_2\right) = h\left(z_n + \frac{1}{2}l_2\right) = 0.0996y_n' - 0.008y_n$

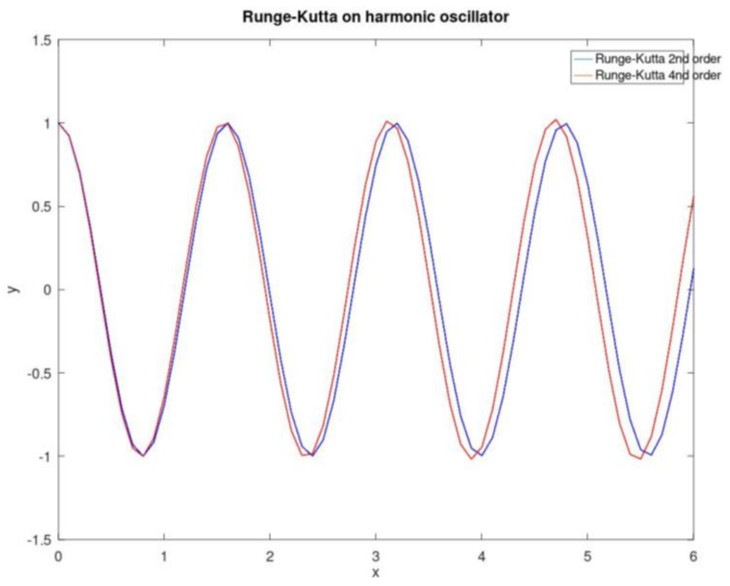
$l_3 = hg\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2, z_n + \frac{1}{2}l_2\right) = -16h\left(y_n + \frac{1}{2}k_2\right) = -1.536y_n - 0.08y_n'$

$k_4 = hf(x_n + h, y_n + k_3, z_n + l_3) = h(z_n + l_3) = 0.092y_n' - 0.1536y_n$

$l_4 = hg(x_n + h, y_n + k_3, z_n + l_3) = -16h(y_n + k_3) = -1.472y_n - 0.15936y_n'$

Regression formulas:

$$\boxed{\left.\begin{aligned} y_{n+1} &= 0.921067y_n + 0.098533y_n' \\ y_{n+1}' &= -1.557333y_n + 0.920107y_n' \end{aligned}\right\}}$$



- 2nd order RK error: $O(h^3)$ – same as Euler-Heum
- 4th order RK error: $O(h^4)$

The graphs of the 2 methods are diverging, while iterations advance, due to the step-wise propagation of different errors (different errors are summing up). 2nd order has bigger error $\rightarrow$ precedes 4th order.

For the implementation you can refer to **Runge-Kutta_method.cpp** at the src directory.

## 9. Runge-Kutta 2nd order - Stability

Example: $y' = -10\frac{y^2}{x}$

Step values: $0.005, 0.01, 0.015, 0.02$
Range: $0.1 \leq x \leq 1.1$
Initial value: $y(0.1) = 1$
Error: $\varepsilon_n = y_n - Y_n$
y: evaluated with RK 2nd order
Y: true value

$$\begin{aligned}
\varepsilon_{n+1} &= y_{n+1} - Y_{n+1} \\
&= y_n + \frac{h}{2}f(x_n, y_n) + \frac{h}{2}f\big(x_n + h, y_n + hf(x_n, y_n)\big) - Y_n - \frac{h}{2}f(x_n, Y_n) - \frac{h}{2}f\big(x_n + h, Y_n + hf(x_n, Y_n)\big) \\
&= \varepsilon_n + \frac{h}{2}\frac{f(x_n, y_n) - f(x_n, Y_n)}{x_n - Y_n}\varepsilon_n + \frac{h}{2}\frac{f\big(x_n + h, y_n + hf(x_n, y_n)\big) - f\big(x_n + h, Y_n + hf(x_n, Y_n)\big)}{x_n - Y_n}\varepsilon_n \\
&= \varepsilon_n[1 + \frac{h}{2}\Big(f_y(x_n, y_n) + f_y\big(x_n + h, y_n + hf(x_n, y_n)\big)\Big)] \\
&= \varepsilon_n[1 + \frac{h}{2}(m_1 + m_2)]
\end{aligned}$$

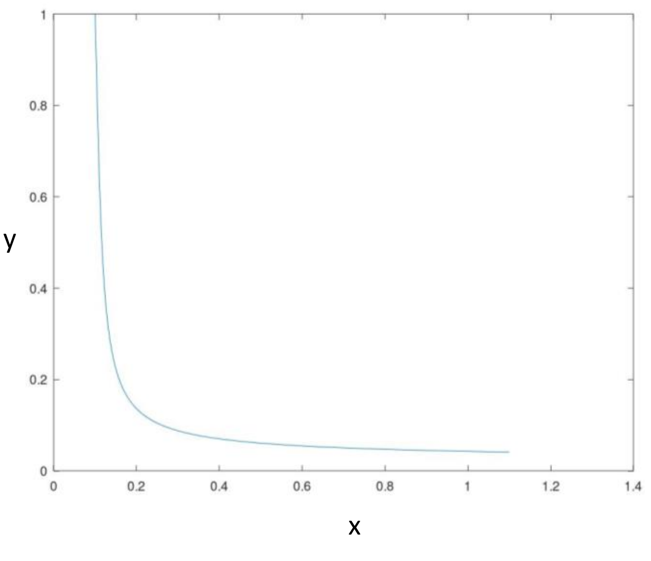
In order to be completely stable:

$$\left.\begin{array}{c}
\left|1 + \frac{h}{2}(m_1 + m_2)\right| < 1 \Rightarrow -4 < h(m_1 + m_2) < 0 \\
y' = -10\frac{y^2}{x} \leq 0 \;\; in\ [0.1, 1.1] \Rightarrow \; y\ is\ decreasing \; \Rightarrow y_{\max} = y(0.1) = 1 \\
m_1 = -20\frac{y}{x} \\
m_2 = \frac{-20y + 600h\frac{y^2}{x} - 4000h\frac{y^3}{x^2}}{x + h}
\end{array}\right\} \Rightarrow$$

$$\Rightarrow \boxed{0 < \frac{0.2h - 29h^2 + 2000h^3}{0.1 + h} < 0.005}$$

Out of the 4 step values, only the 1st fulfills the above condition and, thus, leads on to a stable solution.

Runge-Kutta 2nd order
$h = 0.005$



Implementation: **Runge-Kutta-Stability_method.cpp**