COURSE 10

4.3. Iterative methods for solving linear systems

Because of round-off errors, direct methods become less efficient than iterative methods for large systems ($>$100 000 variables).

An iterative scheme for linear systems consists of converting the system

$$Ax = b \tag{1}$$

to the form

$$x = \tilde{b} - Bx.$$

After an initial guess for $x^{(0)}$, the sequence of approximations of the solution $x^{(0)}, x^{(1)}, ..., x^{(k)}, ...$is generated by computing

$$x^{(k)} = \tilde{b} - Bx^{(k-1)}, \quad \text{for } k = 1, 2, 3, ...$$

4.3.1. Jacobi iterative method

Consider the $n \times n$ linear system,

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ \qquad\qquad ... \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n, \end{cases}$$

where we assume that the diagonal terms a_{11}, a_{22}, ..., a_{nn} are all nonzero.

We begin our iterative scheme by solving each equation for one of the variables:

$$\begin{cases} x_1 = u_{12}x_2 + ... + u_{1n}x_n + c_1 \\ x_2 = u_{21}x_1 + ... + u_{2n}x_n + c_2 \\ ... \\ x_n = u_{n1}x_1 + ... + u_{nn-1}x_{n-1} + c_n, \end{cases}$$

where $u_{ij} = -\frac{a_{ij}}{a_{ii}}$, $c_i = \frac{b_i}{a_{ii}}$, $i = 1, ..., n$.

Let $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$ be an initial approximation of the solution. The $k+1$-th approximation is:

$$\begin{cases} x_1^{(k+1)} = u_{12}x_2^{(k)} + ... + u_{1n}x_n^{(k)} + c_1 \\ x_2^{(k+1)} = u_{21}x_1^{(k)} + u_{23}x_3^{(k)} + ... + u_{2n}x_n^{(k)} + c_2 \\ ... \\ x_n^{(k+1)} = u_{n1}x_1^{(k)} + ... + u_{nn-1}x_{n-1}^{(k)} + c_n, \end{cases}$$

for $k = 0, 1, 2, ...$

An algorithmic form:

$$x_i^{(k)} = \frac{b_i - \sum\limits_{j=1, j\neq i}^{n} a_{ij}x_j^{(k-1)}}{a_{ii}}, \ i = 1, 2, ..., n, \text{ for } k \geq 1.$$

The iterative process is terminated when a convergence criterion is satisfied.

Stopping criterions: $\left|x^{(k)} - x^{(k-1)}\right| < \varepsilon$ or $\frac{\left|x^{(k)} - x^{(k-1)}\right|}{\left|x^{(k)}\right|} < \varepsilon$, with $\varepsilon > 0$ - a prescribed tolerance.

Example 1 *Solve the following system using the Jacobi iterative method. Use $\varepsilon = 10^{-3}$ and $x^{(0)} = (0\ 0\ 0\ 0)$ as the starting vector.*

$$\begin{cases} 7x_1 - 2x_2 + x_3 = 17 \\ x_1 - 9x_2 + 3x_3 - x_4 = 13 \\ 2x_1 + 10x_3 + x_4 = 15 \\ x_1 - x_2 + x_3 + 6x_4 = 10. \end{cases}$$

These equations can be rearranged to give

$$\begin{aligned} x_1 &= (17 + 2x_2 - x_3)/7 \\ x_2 &= (-13 + x_1 + 3x_3 - x_4)/9 \\ x_3 &= (15 - 2x_1 - x_4)/10 \\ x_4 &= (10 - x_1 + x_2 - x_3)/6 \end{aligned}$$

and, for example,

$$\begin{aligned} x_1^{(1)} &= (17 + 2x_2^{(0)} - x_3^{(0)})/7 \\ x_2^{(1)} &= (-13 + x_1^{(0)} + 3x_3^{(0)} - x_4^{(0)})/9 \\ x_3^{(1)} &= (15 - 2x_1^{(0)} - x_4^{(0)})/10 \\ x_4^{(1)} &= (10 - x_1^{(0)} + x_2^{(0)} - x_3^{(0)})/6. \end{aligned}$$

Substitute $x^{(0)} = (0,0,0,0)$ into the right-hand side of each of these equations to get

$$x_1^{(1)} = (17 + 2 \cdot 0 - 0)/7 = 2.428\ 571\ 429$$
$$x_2^{(1)} = (-13 + 0 + 3 \cdot 0 - 0)/9 = -1.444\ 444\ 444$$
$$x_3^{(1)} = (15 - 2 \cdot 0 - 0)/10 = 1.5$$
$$x_4^{(1)} = (10 - 0 + 0 - 0)/6 = 1.666\ 666\ 667$$

and so $x^{(1)} = (2.428\ 571\ 429, -1.444\ 444\ 444, 1.5, 1.666\ 666\ 667)$.
The Jacobi iterative process:

$$x_1^{(k+1)} = \left(17 + 2x_2^{(k)} - x_3^{(k)}\right)/7$$
$$x_2^{(k+1)} = \left(-13 + x_1^{(k)} + 3x_3^{(k)} - x_4^{(k)}\right)/9$$
$$x_3^{(k+1)} = \left(15 - 2x_1^{(k)} - x_4^{(k)}\right)/10$$
$$x_4^{(k+1)} = \left(10 - x_1^{(k)} + x_2^{(k)} - x_3^{(k)}\right)/6, \qquad k \geq 1.$$

We obtain a sequence that converges to

$$\mathbf{x}^{(9)} = (2.000127203, -1.000100162, 1.000118096, 1.000162172).$$

4.3.2. Gauss-Seidel iterative method

Almost the same as Jacobi method, except that each x-value is improved using the most recent approx. of the other variables.

For a $n \times n$ system, the $k+1$-th approximation is:

$$\begin{cases} x_1^{(k+1)} = u_{12}x_2^{(k)} + ... + u_{1n}x_n^{(k)} + c_1 \\ x_2^{(k+1)} = u_{21}x_1^{(k+1)} + u_{23}x_3^{(k)} + ... + u_{2n}x_n^{(k)} + c_2 \\ ... \\ x_n^{(k+1)} = u_{n1}x_1^{(k+1)} + ... + u_{nn-1}x_{n-1}^{(k+1)} + c_n, \end{cases}$$

with $k = 0, 1, 2, ...$; $u_{ij} = -\frac{a_{ij}}{a_{ii}}$, $c_i = \frac{b_i}{a_{ii}}$, $i = 1, ..., n$ (as in Jacobi method).

Algorithmic form:

$$x_i^{(k)} = \frac{b_i - \sum\limits_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum\limits_{j=i+1}^{n} a_{ij}x_j^{(k-1)}}{a_{ii}}$$

for each $i = 1, 2, ...n,$ and for $k \geq 1$.

Stopping criterions: $\left|x^{(k)} - x^{(k-1)}\right| < \varepsilon$, or $\frac{\left|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right|}{\left|\mathbf{x}^{(k)}\right|} < \varepsilon$, with ε - a prescribed tolerance, $\varepsilon > 0$.

Remark 2 *Because the new values can be immediately stored in the location that held the old values, the storage requirements for* $\mathbf{x}$ *with the Gauss-Seidel method is half than that for Jacobi method and the rate of convergence is faster.*

Example 3 *Solve the following system using the Gauss-Seidel iterative method. Use* $\varepsilon = 10^{-3}$ *and* $\mathbf{x}^{(0)} = (0\ 0\ 0\ 0)$ *as the starting vector.*

$$\begin{cases} 7x_1 - 2x_2 + x_3 = 17 \\ x_1 - 9x_2 + 3x_3 - x_4 = 13 \\ 2x_1 + 10x_3 + x_4 = 15 \\ x_1 - x_2 + x_3 + 6x_4 = 10 \end{cases}$$

We have

$$\begin{aligned}
x_1 &= (17 + 2x_2 - x_3)/7 \\
x_2 &= (-13 + x_1 + 3x_3 - x_4)/9 \\
x_3 &= (15 - 2x_1 - x_4)/10 \\
x_4 &= (10 - x_1 + x_2 - x_3)/6,
\end{aligned}$$

and, for example,

$$\begin{aligned}
x_1^{(1)} &= (17 + 2x_2^{(0)} - x_3^{(0)})/7 \\
x_2^{(1)} &= (-13 + x_1^{(1)} + 3x_3^{(0)} - x_4^{(0)})/9 \\
x_3^{(1)} &= (15 - 2x_1^{(1)} - x_4^{(0)})/10 \\
x_4^{(1)} &= (10 - x_1^{(1)} + x_2^{(1)} - x_3^{(1)})/6,
\end{aligned}$$

which provide the following Gauss-Seidel iterative process:

$$x_1^{(k+1)} = \left(17 + 2x_2^{(k)} - x_3^{(k)}\right)/7$$
$$x_2^{(k+1)} = \left(-13 + x_1^{(k+1)} + 3x_3^{(k)} - x_4^{(k)}\right)/9$$
$$x_3^{(k+1)} = \left(15 - 2x_1^{(k+1)} - x_4^{(k)}\right)/10$$
$$x_4^{(k+1)} = \left(10 - x_1^{(k+1)} + x_2^{(k+1)} - x_3^{(k+1)}\right)/6, \quad \textit{for } k \geq 1.$$

Substitute $\mathbf{x}^{(0)} = (0, 0, 0, 0)$ *into the right-hand side of each of these equations to get*

$$x_1^{(1)} = (17 + 2 \cdot 0 - 0)/7 = 2.428\ 571\ 429$$
$$x_2^{(1)} = (-13 + 2.428\ 571\ 429 + 3 \cdot 0 - 0)/9 = -1.1746031746$$
$$x_3^{(1)} = (15 - 2 \cdot 2.428\ 571\ 429 - 0)/10 = 1.0142857143$$
$$x_4^{(1)} = (10 - 2.428\ 571\ 429 - 1.1746031746 - 1.0142857143)/6$$
$$= 0.8970899472$$

and so

$$\mathbf{x}^{(1)} = (2.428571429 - 1.1746031746, 1.0142857143, 0.8970899472).$$

Similar procedure generates a sequence that converges to

$$\mathbf{x}^{(5)} = (2.000025, -1.000130, 1.000020.0.999971).$$

4.3.3. Relaxation method

In case of convergence, the Gauss-Seidel method is faster than Jacobi method. The convergence can be more improved using **relaxation method (SOR method) (**SOR=Succesive Over Relaxation)

Algorithmic form of the method:

$$x_i^{(k)} = \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) + (1-\omega)x_i^{(k-1)}$$

for each $i = 1, 2, ...n,$ and for $k \geq 1$.

For $0 < \omega < 1$ the procedure is called **under relaxation method,** that can be used to obtain convergence for systems which are not convergent by Gauss-Siedel method.

For $\omega > 1$ the procedure is called **over relaxation method,** that can be used to accelerate the convergence for systems which are convergent by Gauss-Siedel method.

By Kahan's Theorem follows that the method converges for $0 < \omega < 2$.

Remark 4 *For $\omega = 1$, relaxation method is Gauss-Seidel method.*

Example 5 *Solve the following system, using relaxation iterative method. Use $\varepsilon = 10^{-3}$, $\mathbf{x}^{(0)} = (1\ 1\ 1)$ and $\omega = 1.25$,*

$$\begin{array}{rcrcrcr} 4x_1 & + & 3x_2 & & & = & 24 \\ 3x_1 & + & 4x_2 & - & x_3 & = & 30 \\ & - & x_2 & + & 4x_3 & = & -24 \end{array}$$

We have

$$\begin{aligned} x_1^{(k)} &= 7.5 - 0.937x_2^{(k-1)} - 0.25x_1^{(k-1)} \\ x_2^{(k)} &= 9.375 - 9.375x_1^{(k)} + 0.3125x_3^{(k-1)} - 0.25x_2^{(k-1)} \\ x_3^{(k)} &= -7.5 + 0.3125x_2^{(k)} - 0.25x_3^{(k-1)}, \quad \textit{for } k \geq 1. \end{aligned}$$

The solution is $(3, 4, -5)$.

4.3.4 The matriceal formulations of the iterative methods

Split the matrix A into the sum

$$A = D + L + U,$$

where D is the diagonal of A, L the lower triangular part of A, and U the upper triangular part of A. That is,

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}, \qquad L = \begin{bmatrix} 0 & \cdots & & 0 \\ a_{21} \vdots & & & \ddots \\ \vdots & \ddots & \ddots & \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix},$$

$$U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ & & \ddots & a_{n-1,n} \\ 0 & \cdots & & 0 \end{bmatrix}$$

The system $Ax = b$ can be written as

$$(D + L + U)\mathbf{x} = \mathbf{b}.$$

The **Jacobi method** in matriceal form is given by:

$$D\mathbf{x}^{(k)} = -(L+U)\mathbf{x}^{(k-1)} + \mathbf{b}$$

the **Gauss-Seidel method** in matriceal form is given by:

$$(D+L)\mathbf{x}^{(k)} = -U\mathbf{x}^{(k-1)} + \mathbf{b}$$

and **the relaxation method** in matriceal form is given by:

$$(D+\omega L)\mathbf{x}^{(k)} = ((1-\omega)D - \omega U)\mathbf{x}^{(k-1)} + \omega\mathbf{b}$$

Convergence of the iterative methods

Remark 6 *The convergence (or divergence) of the iterative process in the Jacobi and Gauss-Seidel methods does not depend on the initial guess, but depends only on the character of the matrices themselves. However, a good first guess in case of convergence will make for a relatively small number of iterations.*

A sufficient condition for convergence:

Theorem 7 (**Convergence Theorem**) *If A is strictly diagonally dominant, then the Jacobi, Gauss-Seidel and relaxation methods converge for any choice of the starting vector $\mathbf{x}^{(0)}$.*

Example 8 *Consider the system of equations*

$$\begin{bmatrix} 3 & 1 & 1 \\ -2 & 4 & 0 \\ -1 & 2 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}.$$

The coefficient matrix of the system is strictly diagonally dominant since

$$\begin{aligned} |a_{11}| &= |3| = 3 > |1| + |1| = 2 \\ |a_{22}| &= |4| = 4 > |-2| + |0| = 2 \\ |a_{33}| &= |-6| = 6 > |-1| + |2| = 3. \end{aligned}$$

Hence, if the Jacobi or Gauss-Seidel method are used to solve the system of equations, they will converge for any choice of the starting vector $\mathbf{x}^{(0)}$.

5. Numerical methods for solving nonlinear equations in $\mathbb{R}$

The roots of the iterative methods are traced back to Egyptians and Babylonians (cc. 1800 B.C.).

Example of nonlinear equation. Kepler's Equation: consider a two-body problem like a satellite orbiting the earth (or a planet revolving around the sun). Kepler discovered that the orbit is an ellipse and the central body F (earth) is in a focus of the ellipse. The speed of the satellite P is not uniform: near the earth it moves faster than far away. It is used Kepler's law to predict where the satellite will be at a given time. If we want to know the position of the satellite for t = 9 minutes, then we have to solve the equation $f(E) = E - 0.8 \sin E - 2\pi/10 = 0$.

Let $f : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}$. Consider the equation

$$f(x) = 0, \quad x \in \Omega. \tag{2}$$

We attach a mapping $F : D \to D$, $D \subset \Omega^n$ to this equation.

Let $(x_0, ..., x_{n-1}) \in D$. Using F and the numbers $x_0, x_1, ..., x_{n-1}$ we construct iteratively the sequence

$$x_0, x_1, ..., x_{n-1}, x_n, ... \tag{3}$$

with

$$x_i = F\left(x_{i-n}, ..., x_{i-1}\right), \quad i = n, \ldots . \tag{4}$$

The problem consists in choosing F and $x_0, ..., x_{n-1} \in D$ such that the sequence (3) to be convergent to the solution of the equation (2).

Definition 9 *The procedure of approximation the solution of equation (2) by the elements of the sequence (3), computed as in (4), is called* F-**method.**

The numbers $x_0, x_1, ..., x_{n-1}$ *are called* **the starting (initial) points** *and the* k*-th element of the sequence (3) is called an approximation of* k*-th index of the solution.*

If the set of starting points has only one element then the F-method is **an one-step method;** if it has more than one element then the F-method is **a multistep method**.

Definition 10 *If the sequence (3) converges to the solution of the equation (2) then the F-method is convergent, otherwise it is divergent.*

Definition 11 *Let $\alpha \in \Omega$ be a solution of the equation (2) and let $x_0, x_1, ..., x_{n-1}, x_n, ...$ be the sequence generated by a given F-method. The number p having the property*

$$\lim_{x_i \to \alpha} \frac{|\alpha - F(x_{i-n}, ..., x_i)|}{|\alpha - x_i|^p} = C \neq 0, \quad C = constant,$$

is called the order *of the F-method.*

For one-step methods, the above condition reduces to

$$\lim_{x_i \to \alpha} \frac{|\alpha - F(x_i)|}{|\alpha - x_i|^p} = \lim_{x_i \to \alpha} \frac{|\alpha - x_{i+1}|}{|\alpha - x_i|^p} = C \neq 0, \quad C = constant,$$

We construct some classes of F-methods based on the interpolation procedures.

Let $\alpha \in \Omega$ be a solution of the equation (2) and $V(\alpha)$ a neighborhood of α. Assume that f has inverse on $V(\alpha)$ and denote $g := f^{-1}$. Since

$$f(\alpha) = 0$$

it follows that

$$\alpha = g(0).$$

This way, the approximation of the solution α is reduced to the approximation of $g(0)$.

Definition 12 *The approximation of g by means of an interpolating method, and of α by the value of g at the point zero is called* **the inverse interpolation procedure.**

5.1. One-step methods

Let F be a one-step method, i.e., for a given x_i we have $x_{i+1} = F\left(x_i\right)$.

Remark 13 *If $p = 1$, a sufficient convergence condition is $\left|F'\left(x\right)\right| < 1$.*

All information on f are given at a single point, the starting value $\Rightarrow$ we are lead to Taylor interpolation.

Theorem 14 *Let α be a solution of equation (2), $V(\alpha)$ a neighborhood of α, $x, x_i \in V(\alpha)$, f fulfills the necessary continuity conditions. Then we have the following method, denoted by F_m^T, for approximating α:*

$$F_m^T(x_i) = x_i + \sum_{k=1}^{m-1} \frac{(-1)^k}{k!}[f(x_i)]^k g^{(k)}(f(x_i)), \tag{5}$$

where $g = f^{-1}$.

Proof. There exists $g = f^{-1} \in C^m[V(0)]$. Let $y_i = f(x_i)$ and consider Taylor interpolation formula

$$g(y) = (T_{m-1}g)(y) + (R_{m-1}g)(y),$$

with

$$(T_{m-1}g)(y) = \sum_{k=0}^{m-1} \tfrac{1}{k!}(y - y_i)^k g^{(k)}(y_i),$$

and $R_{m-1}g$ is the corresponding remainder.

Since $\alpha = g(0)$ and $g \approx T_{m-1}g$, it follows

$$\alpha \approx (T_{m-1}g)(0) = x_i + \sum_{k=1}^{m-1} \tfrac{(-1)^k}{k!} y_i^k g^{(k)}(y_i).$$

Hence,

$$x_{i+1} := F_m^T(x_i) = x_i + \sum_{k=1}^{m-1} \tfrac{(-1)^k}{k!} [f(x_i)]^k g^{(k)}(f(x_i))$$

is an approximation of α, and F_m^T is an approximation method for the solution α. ■

Concerning the order of the method F_m^T we state:

Theorem 15 *If* $g = f^{-1}$ *satisfies* $g^{(m)}(0) \neq 0$, *then* $\mathrm{ord}(F_m^T) = m$.

Remark 16 *We have an upper bound for the absolute error in approximating* α *by* x_{i+1} :

$$\left|\alpha - F_m^T(x_i)\right| \leq \frac{1}{m!}[f(x_i)]^m M_m g, \quad \textit{with } M_m g = \max_{y \in V(0)} \left|g^{(m)}(y)\right|.$$

Particular cases.

1) Case $m = 2$.

$$F_2^T(x_i) = x_i - \frac{f(x_i)}{f'(x_i)}.$$

This method is called **Newton's method** (**the tangent method**). Its order is 2.

2) Case $m = 3$.

$$F_3^T(x_i) = x_i - \frac{f(x_i)}{f'(x_i)} - \frac{1}{2}\left[\frac{f(x_i)}{f'(x_i)}\right]^2 \frac{f''(x_i)}{f'(x_i)},$$

with $\text{ord}(F_3^T) = 3$. So, this method converges faster than F_2^T.

3) Case $m = 4$.

$$F_4^T(x_i) = x_i - \frac{f(x_i)}{f'(x_i)} - \frac{1}{2}\frac{f''(x_i)f^2(x_i)}{[f'(x_i)]^3} + \frac{\left(f'''(x_i)f'(x_i) - 3[f''(x_i)]^2\right)f^3(x_i)}{3![f'(x_i)]^5}.$$

Remark 17 *The higher the order of a method is, the faster the method converges. Still, this doesn't mean that a higher order method is more efficient (due to computation requirements). By the contrary, the most efficient are the methods of relatively low order, due to their low complexity (methods F_2^T and F_3^T).*

Newton's method (Newton-Raphson method) named after Isaac Newton (1642–1726) and Joseph Raphson (1648–1715), is a root-finding algorithm which produces successively better approximations to the roots of a real-valued function.

This method is so efficient in computing $\sqrt{a}$, that it is a choice even today in modern codes.

Some comments on the history of this method

The traces of this methods can be found in ancient times (Babylon and Egypt, 1800 B.C.), as it appears in the computation of the square root of a number.

Different methods, either algebraically equivalent to Newton's method or of Newton type were known in antiquity to India and then to Arabic culture.

In solving nonlinear problems, Newton (≈1669) and subsequently Raphson (1690) have dealt only with polynomial equations ($x^3-2x-5=0$ is "the classical equation where the Newton method is applied"). Newton has also considered such iterations in solving Kepler's equation $x - e\sin x = M$.

Newton has considered the process of successively computing the corrections, which were added finally altogether to form the final approximation. He didn't compute the successive approximations, but computes a sequence of polynomials, and only at the end arrives at an approximation for the root: let x_0 be a given first estimate of the solution α of $f(x) = 0$. Write $g_0(x) = f(x)$, and suppose $g_0(x) = \sum_{i=0}^{n} a_i x^i$. Writing $e_0 = \alpha - x_0$ we obtain by binomial expansion about the given x_0 a new polynomial equation in the variable e_0:

$$0 = g_0(\alpha) = g_0(x_0 + e_0) = \sum_{i=0}^{n} a_i (x_0 + e_0)^i$$
$$= \sum_{i=0}^{n} a_i \left[\sum_{j=0}^{i} \binom{i}{j} x_0^j {e_0}^{i-j} \right] = g_1(e_0).$$

Neglecting terms involving higher powers of e_0 (linearizing the explicitly computed polynomial g_1) produces

$$0 = g_1(e_0) \approx \sum_{i=0}^{n} a_i (x_0^i + i x_0^{i-1} e_0) = \sum_{i=0}^{n} a_i x_0^i + e_0 \sum_{i=0}^{n} a_i i x_0^{i-1}$$

from which we deduce that

$$e_0 \approx c_0 = \left(- \sum_{i=0}^{n} a_i x_0^i \right) \Big/ \left(\sum_{i=0}^{n} a_i i x_0^{i-1} \right)$$

and set $x_1 = x_0 + c_0$. Formally this correction can be written $c_0 = -g_0(x_0)/g_0'(x_0) = -f(x_0)/f'(x_0)$.

Now repeat the process, but instead of expanding the original polynomial g_0 about x_1 expand the polynomial g_1 about the point c_0, that is considered to be a first estimate of the solution e_0 of the new equation $g_1(x) = 0$. Thus similarly obtain $0 = g_1(e_0) = g_1(c_0 + e_1) = g_2(e_1)$, where the polynomial g_2 is explicitly computed. Linearizing again produces $e_1 \approx c_1 = -g_1(c_0)/g_0'(c_0)$, corresponding $x_2 = x_1 + c_1$.

Raphson has considered the approximations updated at each step (the usual iterations), a process equivalent to what we use nowadays. However, the derivatives of f (which could be calculated with the "fluxions" of that time) do not appear in their formulae. For more than a century, the belief was that these two variants (of Newton and Raphson) represent two different methods.

Simpson (1740) was the first to apply the method to transcendent equations, using the "fluxions" (the fluxions $\dot{x}$ are "essentially" equivalent to dx/dt.) He even extended it to the solving of nonlinear systems of two equations, subsequently generalized to the usual form from today.

The formulation of the method using $f'(x)$ was published by Lagrange in 1798.

Due to the Fourier's influential book *Analyse des Équations Determinées* (1837), where Raphson and Simpson were not mentioned, the name of the method remained "Newton". Some authors use "the Newton-Raphson method".

When there exists a neighborhood of α where the F-method is convergent, choosing x_0 in such a neighborhood allows approximating α by terms of the sequence

$$x_{i+1} = F_2^T\left(x_i\right) = x_i - \frac{f(x_i)}{f\,'(x_i)}, \quad i = 0, 1, ...,$$

with a prescribed error ε.

If α is a solution of equation (2) and $x_{n+1} = F_2^T\left(x_n\right)$, for approximation error, Remark 16 gives

$$\left|\alpha - x_{n+1}\right| \leq \tfrac{1}{2}[f\left(x_n\right)]^2 M_2 g.$$