Numerical solution of nonlinear equations

Anne Kværnø

Oct 23, 2018

The Python codes for this note are given in `nonlinearequations.py`.

1 Introduction

We will start by considering some numerical techniques for solving nonlinear scalar equations (one equation, one unknown), such as, for example

$$x^3 + x^2 - 3x = 3.$$

or systems of equations, such as, for example

$$xe^y = 1,$$
$$-x^2 + y = 1.$$

NB!

- Refer to section 3.1 in *Preliminaries* for some general comments on convergence.

- There are no examples of numerical calculations done by hand in this note. If you would like some, you can easily generate them yourself. Take one of the computer exercises, do your calculations by hand, if needed modify the code so that you get the output you want, run the code, and compare with the results you got by your pencil and paper calculation.

2 Scalar equations

In this section we discuss the solution of scalar equations. The techniques we will use are known from previous courses. When they are repeated here, it is because the techniques used to develope and analyse these methods can, at least in principle be extended to systems of equations. We will also emphasise the error analysis of the methods.

A scalar equation is given by

$$f(x) = 0, \tag{1}$$

where f is a continuous function defined on some interval $[a, b]$. A solution r of the equation is called a *zero* or a *root* of f. A nonlinear equation may have one, more than one or none roots.

Example 1: Given the function $f(x) = x^3 + x^2 - 3x - 3$ on the interval $[-2, 2]$. Plot the function on the interval, and locate possible roots.

See `example1` in `nonlinearequations.py`.

According to the plot, the function f has three real roots in the interval $[-2, 2]$.

The function can be rewritten as

$$f(x) = x^3 + x^2 - 3x - 3 = (x+1)(x^2 - 3).$$

Thus the roots of f are -1 and $\pm\sqrt{3}$. We will use this example as a test problem later on.

2.1 Existence and uniqueness of solutions

The following theorem is well known:

Theorem 1: Existence and uniqueness of a solution.

- If $f \in C[a, b]$ with $f(a)$ and $f(b)$ of opposite sign, there exist at least one $r \in (a, b)$ such that $f(r) = 0$.

- The solution is unique if $f \in C^1[a, b]$ and $f'(x) > 0$ or $f'(x) < 0$ for all $x \in (a, b)$.

The first condition guarantees that the graph of f will pass the x-axis at some point r, the second guarantees that the function is either strictly increasing or strictly decreasing.

2.2 Bisection method

The first part of the theorem can be used to construct the first, quite intuitive algorithm for solving scalar equations. Given an interval, check if f has a root in the interval, divide it in two, check in which of the two halfs the root is, and continue until a root is located with sufficient accuracy.

Bisection method.

- Given the function f and the interval $[a, b]$, such that $f(a) \cdot f(b) < 0$.

- Set $a_0 = a$, $b_0 = b$.

- For $k = 0, 1, 2, 3, \ldots$

$$c_k = \frac{a_k + b_k}{2}$$

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, c_k] & \text{if } f(a_k) \cdot f(c_k) \leq 0 \\ [c_k, b_k] & \text{if } f(b_k) \cdot f(c_k) \leq 0 \end{cases}$$

Use c_k as the approximation to the root. Since c_k is the midpoint of the interval $[a_k, b_k]$, the error satisfies $|c_k - r| \leq (b_k - a_k)/2$. The loop is terminated when $(b_k - a_k)/2$ is smaller than some user specified tolerance. We will of course also terminate the loop if $f(c_k)$ is very close to 0.

Implementation. The algorithm is implemented in the function `bisection()`. See the function `bisection()` in `nonlinearequations.py`.

Example 2: Use the code above to find the root of $f(x) = x^3 + x^2 - 3x - 3$ in the interval $[1.5, 2]$. Use 10^{-6} as the tolerance.

See the function `example2` in `nonlinearequations.py`.

Control that the numerical result is within the error tolerance:

Exercises:

1. Choose some appropriate intervals and find the two other roots of f.

2. Compute the solution(s) of $x^2 + \sin(x) - 0.5 = 0$ by the bisection method.

3. Given a root in the interval $[1.5, 2]$. How many iterations are required to guarantee that the error is less than 10^{-4}.

2.3 Fixed point iterations

The bisection method is very robust, but not particular fast. We will now discuss a major class of iteration schemes, e.g. the so-called fixed point iterations. The idea is:

- Given a scalar equation $f(x) = 0$ with a root r.

- Rewrite the equation in the *fixed point form* $x = g(x)$ such that the root r of f is a *fixed point* of g, that is, r satisfies $r = g(r)$.

The fixed point iterations are then given by

Fixed point iterations.

- Given g and a starting value x_0.

- For $k = 0, 1, 2, 3, \ldots$

$$x_{k+1} = g(x_k)$$

Implementation. The fixed point scheme is implemented in the function `fixpoint`. The iterations are terminated when either the error estimate $|x_{k+1} - x_k|$ is less than a given tolerance `tol`, or the number of iterations reaches some maximum number `max_iter`.

See the function `fixpoint` in `nonlinearequations.py`.

Example 3: The equation

$$x^3 + x^2 - 3x - 3 = 0 \qquad \text{can be rewritten as} \qquad x = \frac{x^3 + x^2 - 3}{3}.$$

The fixed points are the intersections of the two graphs $y = x$ and $y = \frac{x^3 + x^2 - 3}{3}$, as can be demonstrated by the following script:

See the function `example3` in `nonlinearequations.py`. We observe that the fixed points of g are the same as the zeros of f.

Apply fixed point iterations on $g(x)$. Aim for the fixed point between 1 and 2, so choose $x_0 = 1.5$. Do the iterations converge to the root $r = \sqrt{3}$? See the function `example3_iter` in `nonlinearequations.py`.

Exercises: Repeat the experiment with the following reformulations of $f(x) = 0$:

$$x = g_2(x) = \frac{-x^2 + 3x + 3}{x^2},$$

$$x = g_3(x) = \sqrt[3]{3 + 3x - x^2},$$

$$x = g_4(x) = \sqrt{\frac{3 + 3x - x^2}{x}}$$

Use $x_0 = 1.5$ in your experiments, you may well experiment with other values as well.

2.4 Theory

Let us first state some existence and uniqueness results. Apply Theorem 1 in this note on the equation $f(x) = x - g(x) = 0$. The following is then given (the details are left to the reader):

- If $g \in C[a, b]$ and $a < g(x) < b$ for all $x \in [a, b]$ then g has at least one fixed point $r \in (a, b)$.

- If in addition $g \in C^1[a, b]$ and $|g'(x)| < 1$ for all $x \in [a, b]$ then the fixed point is unique.

In the following, we will write the assumption $a < g(x) < b$ for all $x \in [a, b]$ as $g([a, b]) \subset (a, b)$.

In this section we will discuss the convergence properties of the fixed point iterations, under the conditions for existence and uniqueness given above.

The error after k iterations are given by $e_k = r - x_k$. The iterations converge when $e_k \to 0$ as $k \to \infty$. Under which conditions is this the case?

This is the trick: For some arbitrary k we have

$$x_{k+1} = g(x_k), \qquad \qquad \text{the iterations}$$
$$r = g(r). \qquad \qquad \text{the fixed point}$$

Take the difference between those and use the mean value theorem (Result 3 in section 5 in *Preliminaries*), and finally take the absolute value of each expression in the sequence of equalities:

$$|e_{k+1}| = |r - x_{k+1}| = |g(r) - g(x_k)| = |g'(\xi_k)| \cdot |r - x_k| = |g'(\xi_k)| \cdot |e_k|. \tag{2}$$

Here ξ_k is some unknown value between x_k (known) and r (unknown). We can now use the assumptions from the existence and uniqueness result.

- The condition $g([a, b]) \subset (a, b)$ guarantees that if $x_0 \in [a, b]$ then $x_k \in (a, b)$ for $k = 1, 2, 3, \ldots$.

- The condition $g'(x) \leq L < 1$ guarantees convergence towards the unique fixed point r, since

$$|e_{k+1}| \leq L\,|e_k| \qquad \Rightarrow \qquad |e_k| \leq L^k\,|e_0| \to 0 \quad \text{as } k \to \infty,$$

and $L^k \to 0$ as $k \to \infty$ when $L < 1$.

In summary we have

The fixed point theorem.

If there is an interval $[a, b]$ such that $g \in C^1[a, b]$, $g([a, b]) \subset (a, b)$ and there exist a positive constant $L < 1$ such that $|g'(x)| \leq L < 1$ for all $x \in [a, b]$ then

- g has a unique fixed point r in (a, b).

- The fixed point iterations $x_{k+1} = g(x_k)$ converges towards r for all starting values $x_0 \in [a, b]$.

From the discussion above, we can draw some conclusions:

- The smaller the constant L, the faster the convergence.

- If $|g'(r)| < 1$ then there will always be a neighbourhood around r, $(r - \delta, r + \delta)$ for some δ on which all the conditions are satisfied. Meaning that the iterations will always converge if x_0 is sufficiently close to r.

- If $|g'(r)| > 1$ the fixed point iterations will never converge towards r.

Example 3 revisited: Use the theory above to analyse the iteration scheme from Example 3, where

$$g(x) = \frac{x^3 + x^2 - 3}{3}, \qquad g'(x) = \frac{3x^2 + 2x}{3}$$

It is clear that g is differentiable. We already know that g has three fixed points, $r = \pm\sqrt{3}$ and $r = -1$. For the first two, we have that $g'(\sqrt{3}) = 3 + \frac{2}{3}\sqrt{3} = 4.15$ and $g'(-\sqrt{3}) = 3 - \frac{2}{3}\sqrt{3} = 1.85$, so the fixed point iterations will never converges towards those roots. But $g'(-1) = 1/3$, so we get convergence towards this root, given sufficiently good starting values. The figure below demonstrates that the assumptions of the theorem are satisfied in some region around $x = -1$, for example $[-1.3, -0.7]$.

See `example3_revisited` in `nonlinearequations.py`. The plot to the left demonstrates the assumption $g([a, b]) \subset (a, b)$, as the graph $y = g(x)$ enters at the left boundary and exits at the right and does not leave the region $[a, b]$ anywhere in between. The plot to the right shows that $|g'(x)| <= g'(a) = L < 1$ is satisfied in the interval.

Exercises:

1. See how far the interval $[a, b]$ can be stretched, still with convergence guaranteed.

2. Do a similar analysis on the three other iteration schemes suggested above, and confirm the results numerically. The schemes are:

$$x = g_2(x) = \frac{-x^2 + 3x + 3}{x^2},$$
$$x = g_3(x) = \sqrt[3]{3 + 3x - x^2},$$
$$x = g_4(x) = \sqrt{\frac{3 + 3x - x^2}{x}}$$

```
# Insert your code here.
```

2.5   Newton's method

Based on the previous discussion, it is clear that fast convergence can be achieved if $g'(r)$ is as small as possible, preferable $g'(r) = 0$. Can this be achieved for a general problem?

We have that $x_k = r - e_k$ where e_k is the error in iteration k. Do a Taylor expansion (*Preliminaries*, section 4) of $g(x_k)$ around r:

$$e_{k+1} = r - x_{k+1} = g(r) - g(x_k) = g(r) - g(r - e_k) = -g'(r)e_k + \frac{1}{2}g''(\xi_k)e_k^2 \qquad (3)$$

If $g'(r) = 0$ and there is a constant M such that $|g''(x)|/2 \leq M$, then

$$|e_{k+1}| \leq M|e_k|^2$$

and *quadratic convergence* has been achieved, see *Preliminaries*, section 3.1 on "Convergence of an iterative process".

Question: Given the equation $f(x) = 0$ with an unknown solution r. Can we find a g with r as a fixed point, satisfiying $g'(r) = 0$?

Idea: Let

$$g(x) = x - h(x)f(x).$$

for some function $h(x)$. Independent of the choice of $h(x)$, r will be a fixed point of g. Choose $h(x)$ such that $g'(r) = 0$, that is

$$g'(x) = 1 - h'(x)f(x) - h(x)f'(x) \qquad \Rightarrow \qquad g'(r) = 1 - h(r)f'(r)$$

By choosing $h(x) = 1/f'(x)$ the aim $g'(r) = 0$ is achieved. The result is

---

Newton's method.

- Given f and a starting value x_0.

- For $k = 0, 1, 2, 3, \ldots$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Implementation. The method is implemented in the function `newton()`. The iterations are terminated when $|f(x_k)|$ is less than a given tolerance.

See `newton()` in `nonlinearequations.py`.

Example 4: Solve $f(x) = x^3 + x^2 - 3x - 3 = 0$ by Newton's method. Choose $x_0 = 1.5$. The derivative of f is $f'(x) = 3x^2 + 2x - 3$.

See `example4` in `nonlinearequations.py`.

Error analysis. The method was constructed to give quadratic convergence, that is

$$|e_{k+1}| \le M |e_k|^2,$$

where $e_k = r - x_k$ is the error after k iterations. But under which conditions is this true, and can we say something about the size of the constant M?

By a Taylor expansion of $f(r)$ around x_k, we get:

$$0 = f(r) = f(x_k) + f'(x_k)(r - x_k) + \frac{1}{2}f''(\xi_k)(r - x_k)^2 \qquad \text{(Taylor series)}$$
$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) \qquad \text{(Newton's method)}$$

where ξ_k is between r and x_k.

Subtract the two equations to get

$$f'(x_k)(r - x_{k+1}) + \frac{1}{2}f''(\xi_k)(r - x_k)^2 = 0 \qquad \Rightarrow \qquad e_{k+1} = -\frac{1}{2}\frac{f''(\xi_k)}{f'(x_k)}e_k^2$$

So we obtain quadratic convergence if f is twice differentiable around r, $f'(x_k) \ne 0$, and x_0 is chosen sufficiently close to r. More precise:

> **Theorem: Convergence of Newton iterations.**
>
> Assume that the function f has a root r, and let $I_\delta = [r - \delta, r + \delta]$ for some $\delta > 0$. Assume further that
>
> - $f \in C^2(I_\delta)$.
>
> - There is a positive constant M such that
>
> $$\left| \frac{f''(y)}{f'(x)} \right| \leq 2M, \qquad \text{for all } x, y \in I_\delta.$$
>
> In this case, Newton's iterations converge quadratically,
>
> $$|e_{k+1}| \leq M|e_k|^2$$
>
> for all starting values satisfying $|x_0 - r| \leq \min\{1/M, \delta\}$.

Exercises:

1. Repeat Example 4 using different starting values x_0. Find the two other roots.

2. Verify quadratic convergence numerically. How to do so is explained in *Preliminaries*, section 3.1.

3. Solve the equation $x(1 - \cos(x)) = 0$, both by the bisection method and by Newton's method with $x_0 = 1$. Comment on the result.

3 System of nonlinear equations

In this section we will discuss how to solve systems of non-linear equations, given by

$$f_1(x_1, x_2, \ldots, x_n) = 0$$
$$f_2(x_1, x_2, \ldots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, \ldots, x_n) = 0$$

or short by

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$.

Example 5: Given the two equations

$$x_1^3 - x_2 + \frac{1}{4} = 0$$
$$x_1^2 + x_2^2 - 1 = 0$$

This can be illustrated by `example5_graphs` in `nonlinearequations.py`.

The solutions of the two equations are the intersections of the two graphs. So there are two solutions, one in the first and one in the third quadrant.

We will use this as a test example in the sequel.

3.1 Newton's method for systems of equations

The idea of fixed point iterations can be extended to systems of equations. But it is in general hard to find convergent schemes. So we will concentrate on the extension of Newton's method to systems of equations. And for the sake of illustration, we only discuss systems of two equations and two unknowns written as

$$f(x, y) = 0$$
$$g(x, y) = 0$$

to avoid getting completely lost in indices.

Let $\mathbf{r} = [r_x, r_y]^T$ be a solution to these equations and some $\hat{\mathbf{x}} = [\hat{x}, \hat{y}]^T$ a known approximation to $\mathbf{r}$. We search for a better approximation. This can be done by replacing the nonlinear equation $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ by its linear approximation. Which can be found by a multidimensional Taylor expansion around $\hat{\mathbf{x}}$.

$$f(x, y) = f(\hat{x}, \hat{y}) + \frac{\partial f}{\partial x}(\hat{x}, \hat{y})(x - \hat{x}) + \frac{\partial f}{\partial y}(\hat{x}, \hat{y})(y - \hat{y}) + \ldots$$

$$g(x, y) = g(\hat{x}, \hat{y}) + \frac{\partial g}{\partial x}(\hat{x}, \hat{y})(x - \hat{x}) + \frac{\partial g}{\partial y}(\hat{x}, \hat{y})(y - \hat{y}) + \ldots$$

where the ... represent higher order terms, which are small if $\hat{\mathbf{x}} \approx \mathbf{x}$. By ignoring these terms we get a *linear approximation* to $\mathbf{f}(\mathbf{x})$, and rather than solving the nonlinear original system, we can solve its linear approximation:

$$f(\hat{x}, \hat{y}) + \frac{\partial f}{\partial x}(\hat{x}, \hat{y})(x - \hat{x}) + \frac{\partial f}{\partial y}(\hat{x}, \hat{y})(y - \hat{y}) = 0$$

$$g(\hat{x}, \hat{y}) + \frac{\partial g}{\partial x}(\hat{x}, \hat{y})(x - \hat{x}) + \frac{\partial g}{\partial y}(\hat{x}, \hat{y})(y - \hat{y}) = 0$$

or more compact

$$\mathbf{f}(\hat{\mathbf{x}}) + J(\hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}}) = 0,$$

where the Jacobian $J(\mathbf{x})$ is given by

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x}(x, y) & \frac{\partial f}{\partial y}(x, y) \\ \frac{\partial g}{\partial x}(x, y) & \frac{\partial g}{\partial y}(x, y) \end{pmatrix}$$

It is to be hoped that the solution of the linear equation $\mathbf{x}$ provides a better approximation to $\mathbf{r}$ than our initial guess $\hat{\mathbf{x}}$, so the process can be repeated, resulting in

---

Newton's method for system of equations.

- Given a function $\mathbf{f}(\mathbf{x})$, its Jacobian $J(\mathbf{x})$ and a starting value $\mathbf{x}_0$.

- For $k = 0, 1, 2, 3, \ldots$

  Solve the system $J(\mathbf{x}_k)\Delta_k = -\mathbf{f}(\mathbf{x}_k)$.

  Let $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta_k$.

The strategy can be generalized to systems of n equations in n unknowns, in which case the Jacobian is given by:

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \frac{\partial f_n}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{pmatrix}$$

Implementation.   Newton's method for system of equations is implemented in the function `newton_system`. The numerical solution is accepted when all components of $\mathbf{f}(\mathbf{x}_k)$ are smaller than a tolerance in absolute value, that means when $\|\mathbf{f}(\mathbf{x}_k)\|_\infty < \texttt{tol}$. See *Preliminaries*, section 1 for a description of norms.

See the function `newton_sys` in `nonlinearequations.py`.

Example 6:   Solve the equations from Example 5 by Newton's method. The vector valued function $\mathbf{f}$ and the Jacobian J are in this case

$$\mathbf{f}(\mathbf{x}) = \left(\begin{array}{c} x_1^3 - x_2 + \frac{1}{4} \\ x_1^2 + x_2^2 - 1 \end{array} \right) \qquad \text{and} \qquad J(\mathbf{x}) = \left(\begin{array}{cc} 3x_1^2 & -1 \\ 2x_1 & 2x_2 \end{array} \right)$$

We already know that the system has 2 solutions, one in the first and one in the third quadrant. To find the first one, choose $\mathbf{x}_0 = [1,1]^T$ as starting value.

See the function `example6` in `nonlinearequations.py`.

Exercises:

1. Search for the solution of Example 5 in the third quadrant by changing the initial values.

2. Apply Newton's method to the system

$$\begin{array}{rl} xe^y &= 1 \\ -x^2 + y &= 1 \end{array}, \qquad \text{using } x_0 = y_0 = 0.$$

Final remarks.   A complete error and convergence analysis of Newton's method for systems is far from trivial, and outside the scope of this course. But in summary: If $\mathbf{f}$ is sufficiently differentiable, and there is a solution $\mathbf{r}$ of the system $\mathbf{f}(\mathbf{x}) = 0$ and with $J(\mathbf{r})$ nonsingular, then the Newton iterations will converge quadratically towards $\mathbf{r}$ for all $\mathbf{x}_0$ sufficiently close to $\mathbf{r}$.

Finding solutions of nonlinear equations is difficult. Even if the Newton iterations in principle will converge, it can be very hard to find sufficient good starting values. Nonlinear equations can have none or many solutions. Even when there are solutions, there is no guarantee that the solution you find is the one you want.

If n is large, each iteration is computationally expensive since the Jacobian is evaluated in each iteration. In practice, some modified and more efficient version of Newton's method will be used, maybe together with more robust but slow algorithms for finding sufficiently good starting values.