Fixed Point Iteration

MATLAB Implementation

Tamas Kis | kis@stanford.edu

TAMAS KIS
https://github.com/tamaskis

LICENSE

Contents

1 Download and Installation

1.1 Download from MATLAB Central's File Exchange

The `fixed_point_iteration` function is available for download on MATLAB® Central's File Exchange at https://www.mathworks.com/matlabcentral/fileexchange/86992-fixed-point-iteration-fixed_point_iteration.

1.2 Download from GitHub

The `fixed_point_iteration` function is available for download on GitHub® at https://github.com/tamaskis/fixed_point_iteration-MATLAB.

1.3 Files Included With Download

There are **five** files included in the downloaded `zip` file:

1. `EXAMPLES.M` – *examples for using the* `fixed_point_iteration` *function*
2. `Fixed-Point Iteration - MATLAB Implementation.pdf` – *this PDF*
3. **`fixed_point_iteration.m`** – ***MATLAB function implementing fixed-point iteration***
4. `LICENSE` – *license for the* `fixed_point_iteration` *function*
5. `README.md` – *markdown file for GitHub documentation*

1.4 Accessing the `fixed_point_iteration` Function in a MATLAB Script

There are **four** options for accessing the `fixed_point_iteration` function in a MATLAB script:

1. Copy the `fixed_point_iteration` function to the *end* of your MATLAB script.
2. Place the `fixed_point_iteration.m` file in the same folder as the MATLAB script.
3. Place the `fixed_point_iteration.m` file into whatever folder you want, and then use the `addpath(folderName)` command[1] where the `folderName` parameter is a string that stores the filepath of the folder that `fixed_point_iteration.m` is in *relative to* the folder that your script is in.
4. Make a toolbox by first opening `fixed_point_iteration.m`, then going to the `HOME` tab in MATLAB, and finally selecting `Package Toolbox` in the drop-down menu under `Add-Ons`. Once you package the `fixed_point_iteration` function as a toolbox, you can use it in any script.

[1] https://www.mathworks.com/help/matlab/ref/addpath.html

2 fixed_point_iteration

Calculates the fixed-point of a univariate function using fixed-point iteration.

Syntax

```
c = fixed_point_iteration(f,x0)
c = fixed_point_iteration(f,x0,TOL)
c = fixed_point_iteration(f,x0,[],imax)
c = fixed_point_iteration(f,x0,TOL,imax)
c = fixed_point_iteration(__,'all')
```

Description

`c = fixed_point_iteration(f,x0)` returns the fixed point of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the fixed point. The default tolerance and maximum number of iterations are `TOL = 1e-12` and `imax = 1e9`, respectively.

`c = fixed_point_iteration(f,x0)` returns the fixed point of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the fixed point and `TOL` is the tolerance. The default maximum number of iterations is `imax = 1e9`.

`c = fixed_point_iteration(f,x0,[],imax)` returns the fixed point of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the fixed point and `imax` is the maximum number of iterations. The default tolerance is `TOL = 1e-12`.

`c = fixed_point_iteration(f,x0,TOL,imax)` returns the fixed point of a function $f(x)$ specified by the function handle `f`, where `x0` is an initial guess of the fixed point, `TOL` is the tolerance, and `imax` is the maximum number of iterations.

`c = fixed_point_iteration(__,'all')` returns a vector, where the first element of this vector is the initial guess, all intermediate elements are the intermediate estimates of the fixed point, and the last element is the converged fixed point. This identifier `'all'` may be appended to any of the syntaxes used above.

Examples

Example 2.1

Find the fixed point $x = c$ of

$$f(x) = \sqrt{x}$$

■ ***SOLUTION***

Just by inspection, we know we should have $c = 1$ (since $\sqrt{1} = 1$). Therefore, for the purpose of demonstrating convergence, we pick an initial guess $x_0 = 10^{10}$.

```
% defines f(x)
f = @(x) sqrt(x);

% finds fixed point x=c of f(x) (i.e. f(c)=c)
c = fixed_point_iteration(f,10^10)
```

This yields the result

```
c =

    1.0000
```

Example 2.2

Consider the function $f(x) = x^2 - 1$, which has roots at $x = \pm 1$. Find the positive root $x = 1$ using fixed-point iteration. Additionally, plot the intermediate root estimates vs. iteration number.

■ *SOLUTION*

Applying fixed-point iteration directly would find the fixed point of $f(x)$, not its root. However, we can define a function $g(x)$ that will allow us to use fixed-point iteration to find the root of $f(x)$ (see Section 3.2 for why this works).

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

Substituting $f(x)$ and its derivative ($f'(x) = 2x$) into $g(x)$,

$$g(x) = x - \frac{x^2 - 1}{2x} = \frac{2x^2 - (x^2 - 1)}{2x} = \frac{x^2 + 1}{2x}$$

Defining $g(x)$ in MATLAB,

```
% defines g(x)=x-f(x)/f'(x)
g = @(x) (x^2+1)/(2*x);
```

Since we want the positive root, we pick an initial guess $x_0 = 10$.

```
% finds root of f(x)=x^2-1 using an initial guess x0=10
x0 = 10;
root = fixed_point_iteration(g,x0)
```

This yields the result

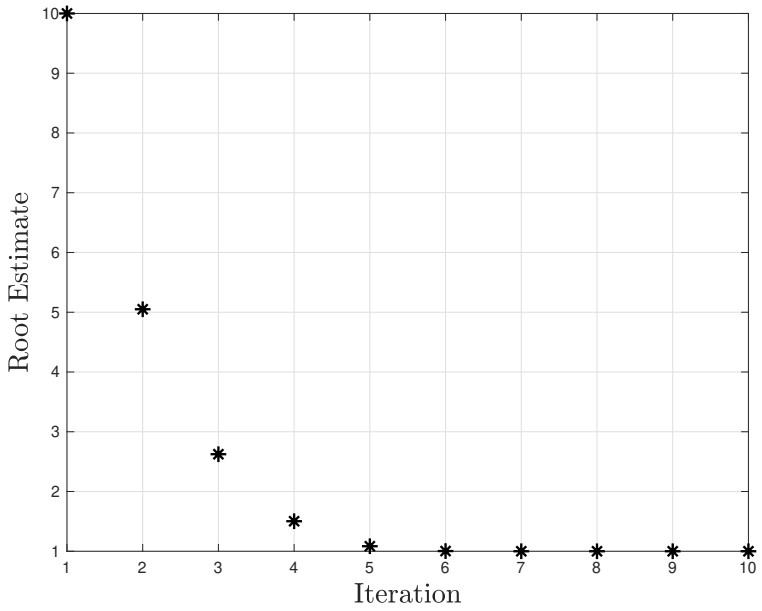
```
root =

     1
```

To plot the intermediate root estimates,

```
% plots the intermediate root estimates
plot(fixed_point_iteration(g,5,[],[],'all'),'k*','markersize',9,...
    'linewidth',1.5);
grid on;
xlabel('Iteration','interpreter','latex','fontsize',18);
ylabel('Root Estimate','interpreter','latex','fontsize',18);
```

This yields the following plot:



Note that this plot is identical to the one found in [2, p. 7].

Example 2.3

Consider a pipe flow problem[a] where the velocity of the fluid, V, is a function of the friction factor, f.

$$V = \sqrt{\frac{3000}{700f + 10}} \text{ ft/s} \tag{1}$$

The Reynolds number, Re, is dependent on the velocity as

$$\mathrm{Re} = 600000V \tag{2}$$

Finally, f is a function of Re (assuming a smooth pipe) [3, p. 431].

$$\frac{1}{\sqrt{f}} = -1.8 \log_{10}\left(\frac{6.9}{\mathrm{Re}}\right) \tag{3}$$

Find the velocity of the flow through the pipe.

■ *SOLUTION*

While we have three equations in three unknowns, we cannot solve this system of equations due to the nonlinearities in the equations for V and $f^{-1/2}$. Let's begin by solving Eq. (3) for f.

$$f = \left[-1.8 \log_{10}\left(\frac{6.9}{\mathrm{Re}}\right)\right]^{-2} \tag{4}$$

Now, we can think of Eqs. (1), (2), and (4) in the following way:

$$V = \phi_1(f)$$

$$\mathrm{Re} = \phi_2(V)$$

$$f = \phi_3(\mathrm{Re})$$

We know that we can use fixed-point iteration to find the fixed point $x = c$ of a function $f(x)$ (i.e. $f(c) = c$). Therefore, our goal is to develop a function where both the input and output are V. Then, we can use fixed-point iteration to solve for V. Since we can get V from f, get f from Re, and get Re from V (i.e. $V \to \mathrm{Re} \to f \to V$), we could determine such a function using function composition:

$$V = \phi_1\left(\phi_3\left(\phi_2(V)\right)\right)$$

Mathematically, this involves substituting Eq. (2) into Eq. (4), and substituting the result into Eq. (1).

$$f = \left[-1.8\log_{10}\left(\frac{6.9}{600000V}\right)\right]^{-2} \quad \to \quad V = \sqrt{\frac{3000}{700\left[-1.8\log_{10}\left(\frac{6.9}{600000V}\right)\right]^{-2} + 10}}$$

Now, let's define the function $g(V)$ as the right hand side of the equation above for V.

$$g(V) = \sqrt{\frac{3000}{700\left[-1.8\log_{10}\left(\frac{6.9}{600000V}\right)\right]^{-2} + 10}}$$

Then the velocity of the flow through the pipe, V, is simply the fixed point of $g(V)$. Defining $g(V)$ in MATLAB,

```
% defines g(V)
g = @(V) sqrt(3000/(700*(-1.8*log10(6.9/(600000*V)))^(-2)+10));
```

Obtaining the flow velocity by finding the fixed point of $g(V)$, using an initial guess of $V_0 = 1$ ft/s,

```
% initial guess for flow velocity through pipe [ft/s]
V0 = 1;

% finds V
V = fixed_point_iteration(g,V0)
```

This yields the result

```
V =

   13.7583
```

Therefore, the flow velocity through the pipe is

$$V = 13.7583 \text{ ft/s}$$

■ *ALTERNATIVE SOLUTION*

In the previous solution, we were able to find a (somewhat) simple form for $g(V)$. However, in many cases, it will not be so easy. Therefore, instead of performing the function composition to get $g(V)$, we can just define a *computational* function that evaluates $g(V)$ in a *sequential* fashion (i.e. first evaluates $\mathrm{Re} = \phi_2(V)$, then $f = \phi_3(\mathrm{Re})$, then $V = \phi_1(f)$).

```
% defines g(V)
function V = g_func(V)
    Re = 600000*V;
    f = (-1.8*log10(6.9/Re))^(-2);
    V = sqrt(3000/(700*f+10));
end
```

We cannot pass `g_func` directly to the `fixed_point_iteration` function. Instead, we first need to assign it a function handle.

```
% assigns function handle for g(V)
g = @(V) g_func(V);
```

Now, we can solve for V by finding the fixed point of $g(V)$.

```
% finds V using an initial guess of V0 = 1 ft/s
V0 = 1;
V = fixed_point_iteration(g,V0)
```

This yields the same result as before:

```
V =

   13.7583
```

[a] This example is adapted from my personal solutions to Problem 8.96 in [3, p. 476]. However, this fluid mechanics text, in general, does not take a computational approach to such problems. Rather, it performs a "trial and error" procedure (including hand calculations and reading values off of a chart), which essentially follows the same process as fixed-point iteration – an example of this can be found in Example 8.7 [3, p. 444].

3 Fixed-Point Iteration

Consider a univariate function $f(x)$. A **fixed point** of $f(x)$, which we denote as c, satisfies

$$f(c) = c$$

Essentially, at the fixed point, the value of the dependent variable is the same as the value of the independent variable. Another way to view fixed points are as the intersections of the curve $y = f(x)$ with the curve $y = x$.

To solve for the fixed point c, we use a technique called **fixed-point iteration.** There are two basic algorithms for implementing fixed-point iteration. The first implementation, shown in Algorithm 1 below, does *not* store the result of each iteration. On the other hand, the second implementation, shown in Algorithm 2, *does* store the result of each iteration. `fixed_point_iteration` implements both of these algorithms [1].

Since Algorithm 2 first needs to preallocate a potentially huge array to store all of the intermediate solutions, Algorithm 1 is significantly faster. Even if $i_{\max}$ (determines size of the preallocated array) is set to be a small number (for example, 10), Algorithm 1 is still faster. The reason we still consider and implement Algorithm 2 is so that convergence studies may be performed.

Algorithm 1: Fixed-point iteration.

Given: $f(x)$, x_0, TOL, $i_{\max}$

// initializes the error so the loop will be entered

$\text{err} = (2)(\text{TOL})$

// sets estimate of fixed point at the first iteration of the fixed-point iteration as the initial guess

$x_{\text{old}} = x_0$

// fixed-point iteration

$i = 2$

while $\varepsilon > \text{TOL}$ ***and*** $i < i_{\max}$ **do**

    // updates estimate of fixed point

    $x_{\text{new}} = f(x_{\text{old}})$

    // calculates error

    $\varepsilon = |x_{\text{new}} - x_{\text{old}}|$

    // stores updated estimate of fixed point for next iteration

    $x_{\text{old}} = x_{\text{new}}$

    // increments loop index

    $i = i + 1$

end

// returns fixed point

$c = x_{\text{new}}$

return c

Algorithm 2: Fixed-point iteration with intermediate estimates of the fixed point.

Given: $f(x)$, x_0, TOL, $i_{\max}$

// initializes the error so the loop will be entered

$\text{err} = (2)(\text{TOL})$

Preallocate an $i_{\max} \times 1$ vector $\mathbf{x}$, where $\mathbf{x}$ is the vector storing the estimates of the fixed point at each iteration.

// inputs 1st and 2nd guesses for the fixed point into $\mathbf{x}$ *vector (note that* x_1 *and* x_2 *are the first and second elements of* $\mathbf{x}$*, while* x_0 *is the input initial guess)*

$x_1 = x_0$

$x_2 = 1.01x_0$

// fixed-point iteration

$i = 2$

while $\varepsilon > \text{TOL}$ ***and*** $i < i_{\max}$ **do**

    // updates estimate of fixed point

    $x_{i+1} = f(x_i)$

    // calculates error

    $\varepsilon = |x_{i+1} - x_i|$

    // increments loop index

    $i = i + 1$

end

// stores intermediate estimates of the fixed point (where last element of $\mathbf{c}$ *is the converged fixed point*

$\mathbf{c} = (x_1, ..., x_i)^T$

// returns fixed point

return $\mathbf{c}$

3.1 Iterative Approaches in Engineering

Fixed-point iteration is especially useful for solving many problems in engineering. Consider the case where we have two unknown quantities, and two highly nonlinear equations that relate them to one another. Mathematically, we have two variables, x and y, and the following two functions:

$$y = f(x) \tag{5}$$

$$x = g(y) \tag{6}$$

In one function, you input x and get y, while in the other function, you input y and get x. Since $f(x)$ and $g(y)$ are nonlinear (as previously mentioned), we cannot obtain closed-form solutions for x and y.

Let's say we're primarily interested in the variable x, where the variable y mainly serves to place a constraint on x. We want to find the value $x = c$ such that both equations are satisfied simultaneously (i.e. $y = f(c)$ and $c = g(y)$). Then we can define a new function $h(x)$ as the function composition $h = g \circ f$ by substituting Eq. (5) into Eq. (6).

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

The solution to our problem, $x = c$, is just the fixed point of $h(x)$. See Example 2.3 for an implementation of this approach to a pipe flow problem.

3.2 Relationship to Newton's Method

Newton's method for finding the root of a univariate function is fundamentally related to fixed-point iteration. The equation defining the iterative procedure in Newton's method is

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Let's define a function $g(x)$.

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

Then Newton's method becomes

$$x_{i+1} = g(x_i)$$

Thus, Newton's method is just a fixed-point iteration of the function $g(x) = x - f(x)/f'(x)$ [1]. See Example 2.2.

References

[1] James Hateley. *Nonlinear Equations*. MATH 3620 Course Reader (Vanderbilt University). 2019.

[2] Tamas Kis. *Newton's Method – MATLAB Implementation*. https://github.com/tamaskis/newtons_method-MATLAB. 2021.

[3] Bruce R. Munson et al. *Fundamentals of Fluid Mechanics*. 7th. Hoboken, NJ: John Wiley & Sons, 2013.