11 May 2018

Wojciech Ciok
EA1

8.Iteration method, for the equation $x = f(x)$

Project 2

Table of contents

1 Description of the method

Iteration method is a method which produces sequence of improving approximations to find a root of a given function f(x). The function is algebraically converted into equation g(x)=x. The values of arguments for which function g(x) and function y=x cross on a coordinate system are the roots of f(x).
The approximation starts with an initial guess x_0. Then, until we reach a good enough approximated answer or we do the iteration previously stated number of times, we recursively get to another step by setting $x_{k+1} = g(x_k)$.
This method will not always work, we have to ensure that it converges, so the error keeps getting smaller after each iteration. In order for this method to converge the following condition has to be met:
If g(x) and g'(x) are continuous on an interval L about their root s of the equation x = g(x), and if |g'(x)|<1 for all x in the interval L then the fixed point iterative process will converge to the root x = s for any initial approximation x_0 belongs to the interval L .
The method becomes easier to understand when analysing the graphical representation.

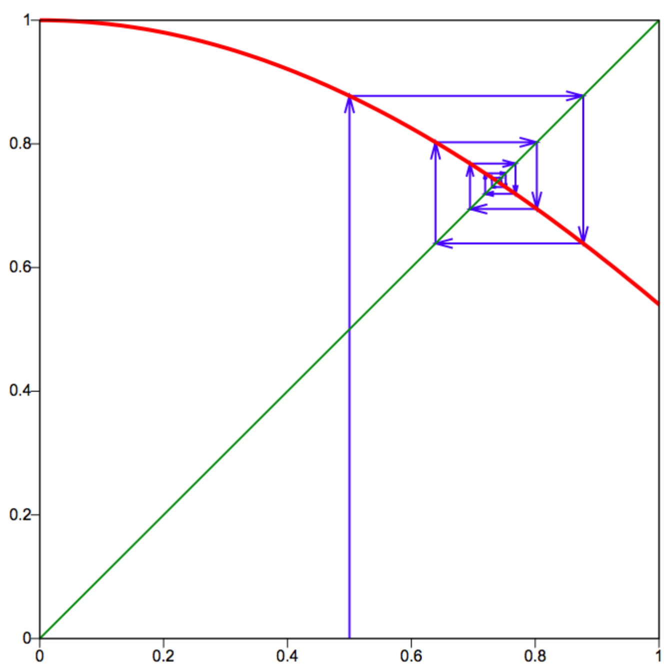

Figure 1: Visualization of convergence

Equation: cos(x)=x
Initial guess: x_0=0.5

2 Description of the program

In my project f(x) is assumed to be in form
$f(x)=p(1)x^n+...+p(n)x+p(n+1)+b*\sin(nx)$
Where p is an array of given coefficients, b is given, n-size of p.

2.1 Fuctions and code

2.1.1 PolyVal function

This function takes an array of coefficients p, an argument x and calculates
$f(x)=p(1)x^n+...+p(n)x+p(n+1)$ using Horner's method.

```
function x = PolyVal(a,z)
n = length(a);
result = a(1);
for j = 2:n
    result = result.*z + a(j);
end
x = result;
```

2.1.2 Iter function

Uses the iteration method to approximate x, f(x)=x.
This function needs following arguments: poly - an array of coefficients, b - a coefficient, tolerance - the desired precision, max_iter - the maximal number of iterations.
This function as well as GraphIter use an internal function f which evaluates f(x).
In order to check the convergence the program tracks error and stop if the error doesn't get smaller with each iteration. Also, the approximations will be saved in arr variable.

```
function [x, arr] = Iter(poly,b,x0,tolerance,max_iter)
%Iter performs iteration method with given function
%f(x)=p(1)x^n+...+p(n)x+p(n+1)+b*sin(nx)on f(x)=x
%   poly - coefficients of polynomial part of the function
%   b - coefficient b*sin(nx)
%   x0 - initial guess
%   tolerance - what is the maximal error
%   max_iter - the max number of iterations
close all;
    function [val] = f(x)
        val = PolyVal(poly,x)+b*sin(n*x);
    end

k=0;
error=tolerance+1;
x=x0;
```

```
n=size(poly,2);

while abs(error)> tolerance && k<= max_iter
    old = x;
    x=f(x);
    olderror=error;
    error = old-x;
    if abs(olderror)<=abs(error)
        error('does not converge');
    end

    %counting iterations:
    k=k+1;
    arr(k)=x;
end

end
```

2.1.3 GraphIter function

Very simmilar to Iter function but this one accepts two more arguments left and right to display the desired part of the coordinate system. It makes a figure with functions y=x, f(x) and shows each iteration as a line connecting the functions (see examples section).

```
function [] = GraphIter(poly,b,left,right,x0,tolerance,
    max_iter)

    function [val] = f(x)
        val = PolyVal(poly,x)+b*sin(n*x);
    end
close all;
k=0;
error=tolerance+1;
x=x0;
n=size(poly,2);

xlin = left:0.001:right;
figure()
ylin = xlin;
plot(xlin, ylin, 'b-', 'LineWidth',2)
hold on
plot(xlin, f(xlin), 'r-', 'LineWidth',2)
xlabel('     x     ')
ylabel(' y    ')
axis('normal')
```

```
legend('y=x','y=f(x)','AutoUpdate','off')

while abs(error)> tolerance && k<= max_iter
    old = x;
    x=f(x);
    olderror=error;
    error = old-x;
    if abs(olderror)<=abs(error)
       close all;
       error('does not converge');
       return;
    end
    plot([old f(old)], [x x])
    plot([x x], [x f(x)])
    %counting iterations:
    k=k+1;
end

end
```

3 Tests and examples

3.1 f(x)=0.5+sin(x)

The consecutive approximations obtain by calling Iter(p,1,0.2,0.000001,100):
0.6987; 1.1432; 1.4100; 1.4871; 1.4965; 1.4972; 1.4973; 1.4973; 1.4973
In this example the answer is well and quickly approximated, the iteration stops after 9 steps.

The graphical representation of approximation GraphIter(p,1,0,2,0.2,0.001,100):

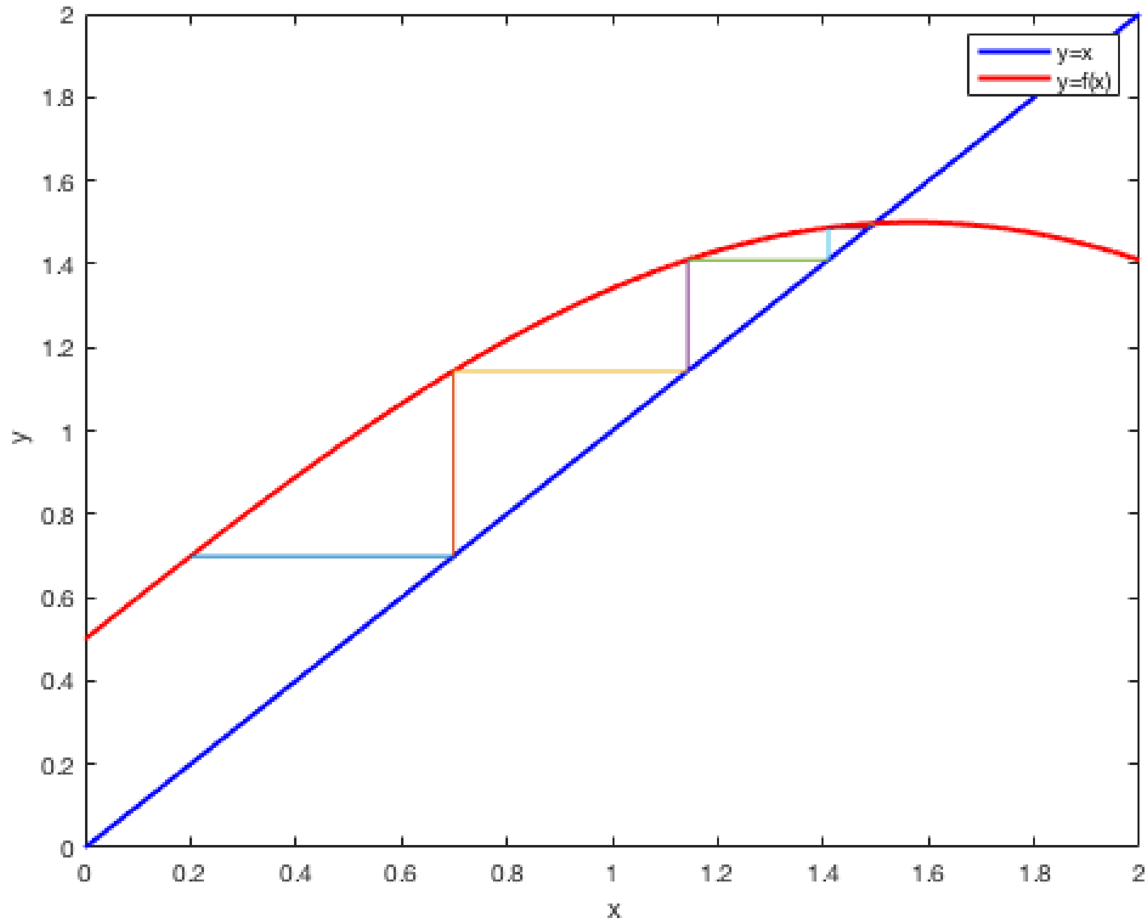

Figure 2: Iteration on f(x)=0.5+sin(x)

3.2 $f(x)=-0.5x^3+1$

The consecutive approximations obtain by calling Iter(p,0,1,0.001,10):
0.5000; 0.9375; 0.5880; 0.8983; 0.6375; 0.8705; 0.6702; 0.8495; 0.6935; 0.8332; 0.7108
In this example it takes time to obtain the desired precision. It is obtained after 53 steps.

The graphical representation of approximation GraphIter(p,0,0.4,1,1,0.001,100):

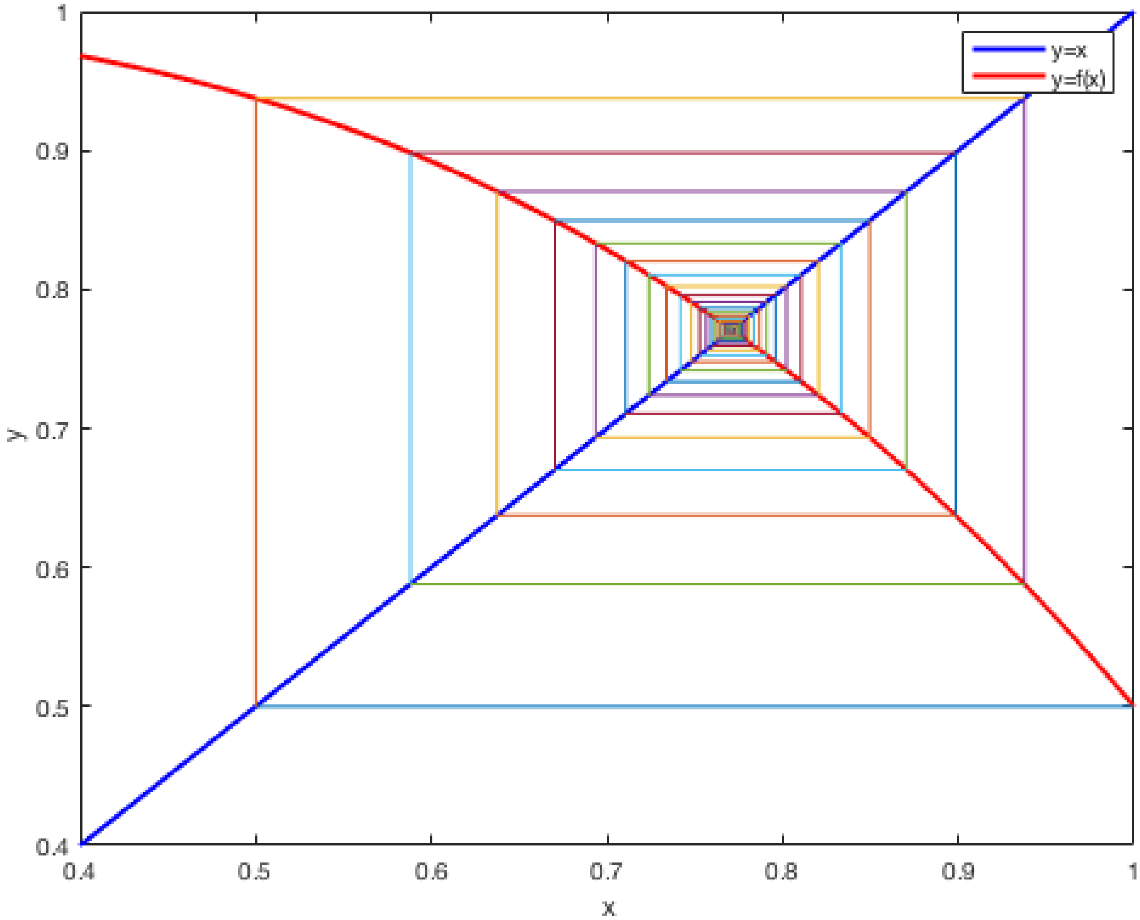

Figure 3: Iteration on f(x)=-0.5x^3+1

3.3 f(x)=0.5x^2-1.5

After calling Iter(p,0,-0.5,0.001,100) we obtain answer=-1.1278:
This answer is far off, the actual answer should be -1.
The method converges for this example but very slowly.

Visualization after 100 steps:

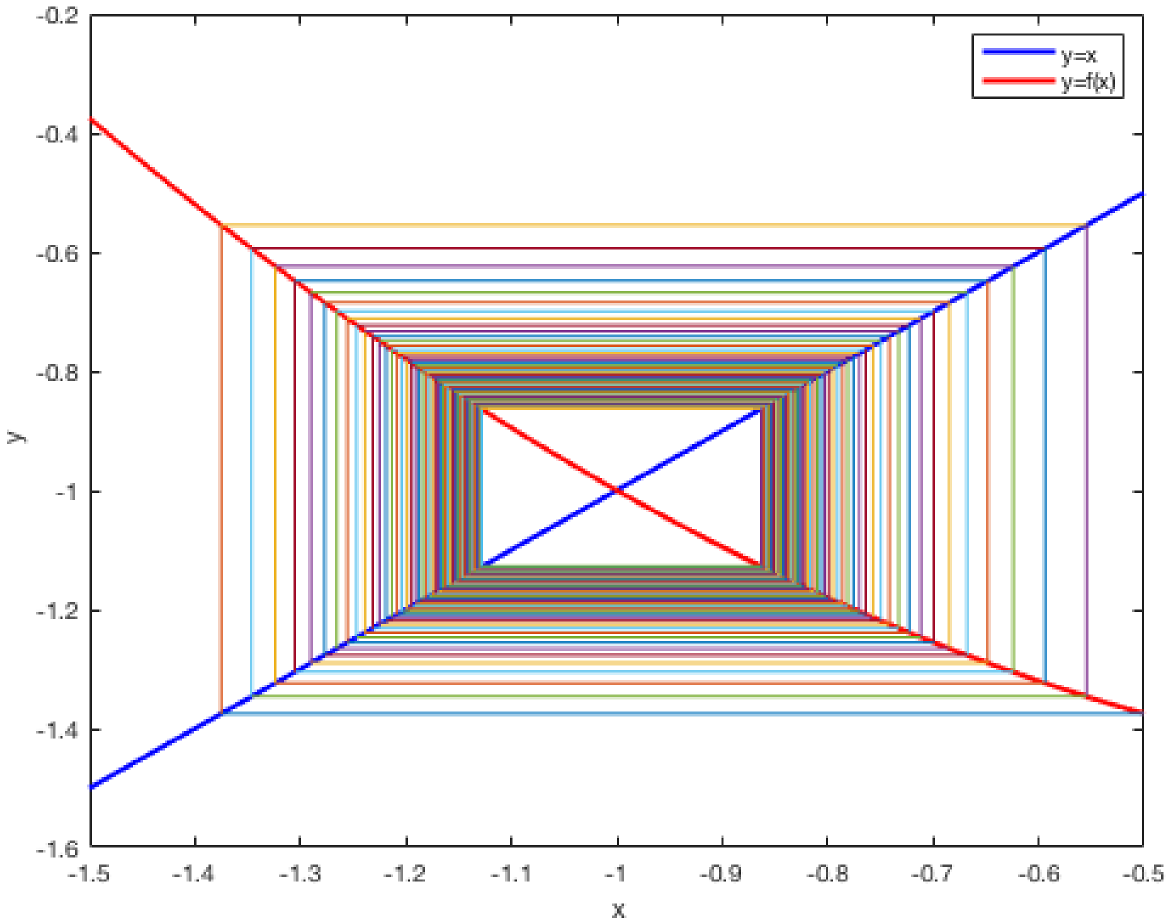

Figure 4: Iteration on f(x)=0.5x^2-1.5

And after 1000 steps:

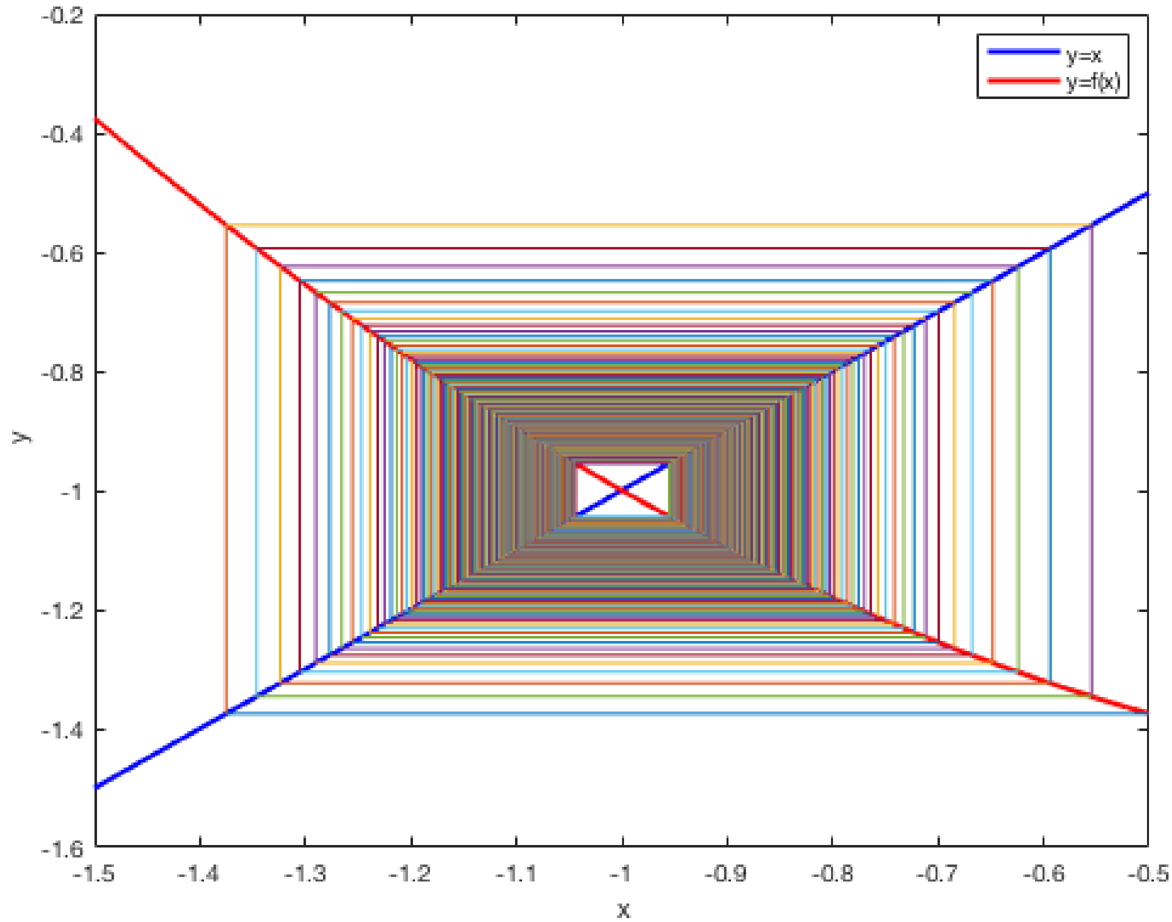

Figure 5: Iteration on f(x)=0.5x^2-1.5

4 Conclusions

The method of iteration is an interesting tool, but there are times when it fails. First of all one has to ensure that it converges and choose appropriate first guess. More than that, as shown in one of examples, the method may fail to deliver an acceptably close approximation even if it converges. As the figures show the convergence may have different shapes like zig-zag in the first example or a spiral.