UNIVERSITY OF
LEICESTER

Mini-Project

For Scientific Computing MA3012/MA7012

Gleb Vorobchuk

Leicester

January 2018

UNIVERSITY OF LEICESTER

Broyden’s Method.

Introduction:

Newton’s Method is using for solving nonlinear equations $F(x) = 0$ is computational expense since partial derivatives and the resulting Jacobian matrix needed to be calculated on each iteration of the algorithm. And computing of this require n^2 scalar functions evaluations for a dense problem(every function *f* depends on every component of *x*). Also solving the linear system using LU factorization costs $O(n^3)$ arithmetic operations in cases when Jacobian matrix is dense.

One of the ways to reduce calculation cost is to reuse Jacobian Matrix for several iterations avoiding recomputing and refactoring it at every iteration of the algorithm.

Many algorithms were designed based on Newton’s Method and now they known as quasi-Newton methods. One of the most efficient variantions on Newton’s method is the Secant Updating methods and its extension - Broyden’s Method.[1]

[1] Heath.M.T. (2002)“Scientific Computing. An Introductory Survey. Second Edition.”(Chapter 5. Nonlinear Equations.): 240. ISBN 0-07-239910-ISBN 0-0711229-X

Description:

In numerical analysis, Broyden's method is an extension of Secant Method, a part of quasi-Newton methods for finding roots in k variables. It was originally described by C. G. Broyden in 1965.[2]. As well as Newtons method Broyden’s method uses the Jacobian Matrix at every iteration. But instead of computing it at every iteration Broyden’s Method computes it only at first iteration and update Jacobian Matrix for every next iteration. Which leads to more efficient calculations.

We begin with initial Jacobian approximation B_0. It can be taken as true Jacobian matrix at x_0, or it can be initialised as identity matrix I, in order to avoid computing derivatives.

Algorithm[3]:

```
$x_0$ = initial guess
$B_0$ = initial Jacobian approximation
for k = 0,1,2,...
    Solve $B_k s_k = -f(x_k)$ for $s_k$
    $x_{k+1} = x_k + s_k$
    $y_k = f(x_{k+1}) - f(x_k)$
    $B_{k+1} = B_k + ((y_k - B_k s_k) s_k^T)/(s_k^T s_k)$
end
```

In following section you can find implementation of Newton’s method, Secant method and Broyden’s method. The point is to show difference in algorithms and calculate the errors and elapsed time.

[2] Broyden, C. G. (October 1965). "A Class of Methods for Solving Nonlinear Simultaneous Equations". *Mathematics of Computation*. American Mathematical Society. **19** (92): 577–593. doi:10.1090/S0025-5718-1965-0198670-6. JSTOR 2003941.

[3] Heath.M.T. (2002)“Scientific Computing. An Introductory Survey. Second Edition.”(Chapter 5. Nonlinear Equations.): 241. ISBN 0-07-239910-ISBN 0-0711229-X

Example[4]:

$$f(x) = \begin{bmatrix} x_1 + 2 * x_2 - 2 \\ x_1^2 + 4 * x^2 - 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Let $x_0 = [1 \quad 2]^T$, so $f(x_0) = [3 \quad 13]^T$, and we let

$$B_0 = J_f(x_0) = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix}$$

Solving the system:

$$B_0 s_0 = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} s_0 = \begin{bmatrix} -3 \\ -13 \end{bmatrix} = -f(x_0)$$

Gives $s_0 = [-1.83 \quad -0.58]^T$, and hence

$$x_1 = x_0 + s_0 = \begin{bmatrix} -0.83 \\ 1.42 \end{bmatrix}, f(x_1) = \begin{bmatrix} 0 \\ 4.72 \end{bmatrix}, y_0 = \begin{bmatrix} -3 \\ -8.28 \end{bmatrix}$$

From updating formula, we therefore have:

$$B_1 = \begin{bmatrix} 1 & 2 \\ 2 & 16 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ -2.34 & -0.74 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix}$$

Solving the system:

$$B_1 s_1 = \begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix} s_1 = \begin{bmatrix} 0 \\ -4.72 \end{bmatrix}$$

Gives $s_0 = [0.58 \quad -0.30]^T$, and hence

$$x_2 = x_1 + s_1 \begin{bmatrix} -0.24 \\ 1.120 \end{bmatrix}, f(x_2) = \begin{bmatrix} 0 \\ 1.08 \end{bmatrix}, y_1 = \begin{bmatrix} 0 \\ -3.64 \end{bmatrix}$$

From updating formula, we therefore have:

$$B_2 = \begin{bmatrix} 1 & 2 \\ -0.34 & 15.3 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1.46 & -0.73 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1.12 & 14.5 \end{bmatrix}$$

Iteration continues until convergence to the solution $x* = [0 \quad 1]^T$.

[4] Heath.M.T. (2002)"Scientific Computing. An Introductory Survey. Second Edition."(Chapter 5. Nonlinear Equations.): 241. ISBN 0-07-239910-ISBN 0-0711229-X

Matlab implementation.[5].

Page 1

```
%%%Evaluate the function
%%%Solve for x such that f(x)=0

x1=-2:0.1:2;
x2=x1;
[fx2,fx1]= meshgrid(x2,x1);
zz_1=0*fx2;
zz_2=0*fx2;

for i_dx = 1:length(x1)
    for j_dx = 1:length(x2)
        xij=[x1(i_dx);x2(j_dx)];
        out = f(xij);
        zz_1(i_dx,j_dx)= out(1);
        zz_2(i_dx,j_dx)=out(2);
    end
end
figure()
mesh(fx1,fx2,zz_1)
hold on
surf(fx1,fx2,zz_2)
x =[1 2]' %initial guess

%%%Newton's Method
e_k=[];
while err(x) > 1e-10
   S = -inv(J(x))*f(x);
   x = x + S;
   e_k=[e_k,err(x)];
end
disp("-----------------------------")
disp("Newtons Solution")
disp(x)
disp("Newtons Error")
disp(err(x))
disp("-----------------------------")
figure()
loglog(e_k(1:end-1),e_k(2:end),'o-');
xlabel('log |e_k|'); ylabel('log |
e_{k+1}|');
title("Newtons.")
```

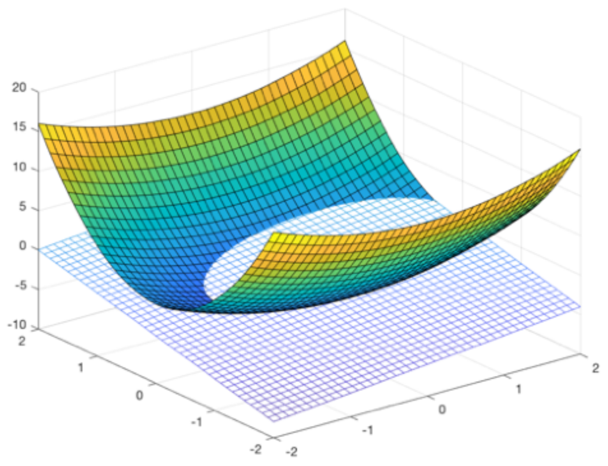
Page 2

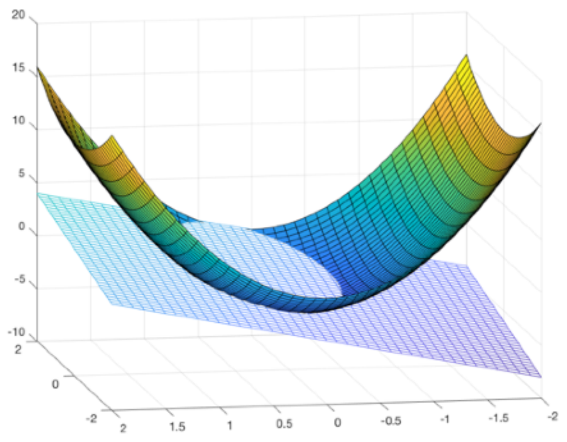
```
%%%Broyden's method
%%%Solve for x such that f(x)=0
x =[1 2]'
e_k1=[];
B = [1 0; 0 1]; %%%Take B as identity matrix
while err(x) > 1e-10
   S = -inv(B)*f(x);
   x1 = x + S;
   y = f(x1) - f(x);
   x = x1;
   e_k1=[e_k1,err(x)];%somehow this formula
is not working properly for this code.
   %%%Update B
   if abs(S'*S) > 1e-2
       B = B + ((y-B*S)*S')/(S'*S);
   end
end
disp("-----------------------------")
disp("Broydens Solution")
disp(x)
disp("Broydens Error")
disp(err(x))
disp("-----------------------------")
figure()
loglog(e_k1(1:end-1),e_k1(2:end),'o-');
xlabel('log |e_k|'); ylabel('log |
e_{k+1}|');
title("Broydens.")
figure()
loglog(e_k(1:end),e_k1(1:length(e_k)),'o-');
title("Newtons/Broydens")

%%%Calculate the error
function outerr = err(x)
out =f(x); %evaluate f(x)
outerr=out(1)^2 + out(2)^2;
end
%%%Calculate f(x)
function out = f(x)
x1=x(1);
x2=x(2);
out =[x1+2*x2-2;x1^2+4*x2^2-4];
end
%%%Calculate exact Jacobian
function out = J(x)
x1=x(1);
x2=x(2);
out=[1 2; 2*x1 8*x2];
end
```

[5] Dr. Carlos Montalvo. MATLAB Help - Broydens Method

https://www.youtube.com/watch?v=aoBq95JQkqM

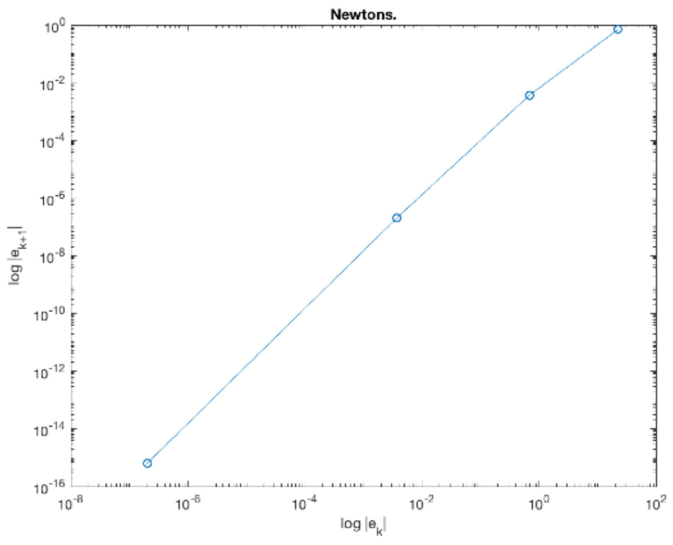
Results:

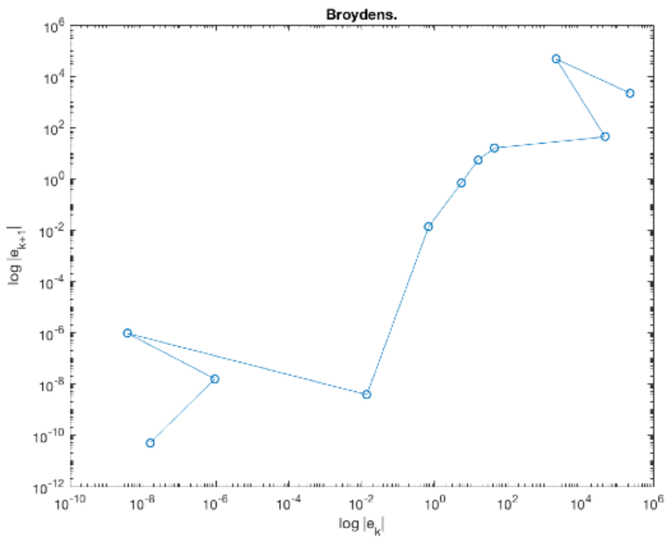




```
----------------------------
Newtons Solution
   -0.0000
    1.0000

Newtons Error
   6.2930e-16

----------------------------
```

```
----------------------------
Broydens Solution
    0.0000
    1.0000

Broydens Error
   5.0267e-11

----------------------------
```





In Newton's method was used exact Jacobian matrix that was recomputed at each iteration.

In Secant Method was used approximation of Jacobian matrix that was updated on each iteration.

In Broydens Method was used Identity matrix that was updated on each iteration.

Newtons method reached demanded accuracy in 5 iterations.

Broyden's method reached demanded accuracy in 12 iterations what coincide with theory since it's convergence rate is superlinear vs quadratic for Newton's method.

Bibliography:

1.Heath.M.T. (2002)“Scientific Computing. An Introductory Survey. Second Edition.”(Chapter 5. Nonlinear Equations.): 238-242. ISBN 0-07-239910-ISBN 0-0711229-X

2. Broyden, C. G. (October 1965). "A Class of Methods for Solving Nonlinear Simultaneous Equations". *Mathematics of Computation*. American Mathematical Society. **19** (92): 577–593. doi:10.1090/S0025-5718-1965-0198670-6. JSTOR 2003941.

3. Dr. Carlos Montalvo. MATLAB Help - Broydens Method. https://www.youtube.com/watch?v=aoBq95JQkqM