Second Order Optimization

October 18, 2019

In [1]: using ForwardDiff
       using LinearAlgebra

0.1 Jacobian

\[ \nabla f(x) = \left[ \frac{\partial}{\partial x} f(x) \right]^T \in \mathbb{R}^n \]

0.2 Hessian(symmetric matrix)

\[ \nabla^2 f(x) = \begin{bmatrix}
\frac{\partial^2}{\partial x_1 \partial x_1} f(x) & \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\
\frac{\partial^2}{\partial x_2 \partial x_1} f(x) & \frac{\partial^2}{\partial x_2 \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} f(x) \\
\cdots & \cdots & \cdots & \cdots \\
\frac{\partial^2}{\partial x_n \partial x_1} f(x) & \frac{\partial^2}{\partial x_n \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} f(x)
\end{bmatrix} \in \mathbb{R}^{n \times n} \]

1 Newton Method

1.1 By solving this equation

\[ \Delta = \nabla^2 f(x)^{-1} \nabla f(x) \]

1.2 Find the value of \( x \) that minimizes \( f(x) \)

In [2]: function newton_method(initial,f,limitTime)
   x1 = initial
   # stepsize
   = 1 # stepsize
   = 10^-(-10) # damping
   hessian = x -> ForwardDiff.hessian(f, x)
   jacobian = x -> ForwardDiff.gradient(f, x)
   for i in 1:limitTime
       = - *inv(hessian(x1))*jacobian(x1)
   x2 = x1 +
   if f(x2) <= f(x1) # step is accepted
       x1 = x2
   # increase stepsize towards =1
   = ^0.5

1
else # step is rejected  
# decrease stepsize  
end  
end  
return x1  
end

Out[2]: newton_method (generic function with 1 method)

2 Gauss-Newton Method

2.0.1 Residual function

\[ r = f(x) - f(x) \]

2.0.2 Jacobian and Hessian matrix can be simplified to this form:

\[ \nabla f(x) = 2 \nabla r^T r \]

\[ H \approx 2 \nabla r^T \nabla r \]

2.0.3 By solving this equation

\[ \Delta = (\nabla r^T \nabla r)^{-1} \nabla r^T r \]

In [3]: # Solve R1->R1 problems  
# Input: initial->initial point, f->fit function, data->data, limitTime->max loop time, limitDis->min distance  
# Output: the optimized parameters of fit function  
function gaussNewton(initial,f,data, limitTime,limitDis)  
x = data  
# define residual function  
r(a; x) = x[length(x)] - f(x[1: (length(x) - 1)]; a=a)  
# initial a0 for the first iterate  
a0 = initial  
# start iterate  
looptime = 0  
for t in 1:limitTime  
looptime = t  
fs = [r(a0; x = x[1,:])]  
for i in 2:length(x[1,:])  
fs = [fs;r(a0;x = x[i,:])]  
end  
js = Array(ForwardDiff.gradient(a -> r(a;x = x[1,:]), a0)')  
for i in 2:length(x[1,:])  
js = [js;Array(ForwardDiff.gradient(a -> r(a;x=x[i,:]), a0)')]  
end  
a1 = a0 - inv(js'*js)*js'*fs
# if a1 is near a0 jump out the loop <=> minimum for the residual function found
if norm(a0-a1) <= limitDis
    a0 = a1
    break
end
a0 = a1
end
println("Loop time: ", looptime)
return a0
end

Out[3]: gaussNewton (generic function with 1 method)

3 Levenberg-Marquardt method

3.0.1 The residual function, Jacobian and Hessian matrix are same with Gauss-Newton Method

3.0.2 Different from Guass-Newton method, the equation Levenberg-Marquardt method should solve is

\[ \Delta = (\nabla r^T \nabla r + \lambda I)^{-1} \nabla r^T r \]

In [4]: # Solve R1->R1 problems
    # Input: initial->initial point, f->fit function, data->data,limitTime->max loop time, limitDis->min distance
    # Output: the optimized parameters of fit function
    function LevenbergMarquardt(initial, f, data, limitTime, limitDis)
        x = data
        # define residual function
        r(a; x) = x[length(x)] - f(x[1: (length(x) - 1)]; a=a)
        # initial a0 for the first iterate
        a0 = initial
        = 10^-10
        # start iterate
        looptime = 0
        for t in 1:limitTime
            looptime = t
            fs = [r(a0; x = x[1,:])]
            for i in 2:length(x[:,1])
                fs = [fs; r(a0; x = x[i,:])]
            end
            js = Array(ForwardDiff.gradient(a -> r(a;x = x[1,:]), a0)')
            for i in 2:length(x[:,1])
                js = [js; Array(ForwardDiff.gradient(a -> r(a;x=x[i,:]), a0)')]
            end
            # Levenberg-Marquardt method
            a1 = a0 - inv(js'*js+Matrix(I, length(initial), length(initial)))*js'*fs
            # if a1 is near a0 jump out the loop <=> minimum for the residual function found
        end
    end
if norm(a0-a1) <= limitDis
    a0 = a1
    break
end
y1 = []
y2 = []
for i in 1:length(x[:,1])
    y1 = [y1;A(x[i,:];a=a0)]
    y2 = [y2;A(x[i,:];a=a1)]
end
residual1 = norm(y1-x[:,length(x[:,1])])
residual2 = norm(y2-x[:,length(x[:,1])])
if residual2 <= residual1
    a0 = a1
else
    a0 = 10*
end
println("Loop time: ", looptime)
return a0
end

Out[4]: LevenbergMarquardt (generic function with 1 method)

4 Example 1

Define a function
\[ f_1(x) = -x_1^2 + x_2^2 - 3x_1 + 4x_2 - x_1x_3 + 26 \]
to test newton method

Out[5]: f1 (generic function with 1 method)

Set the initial point to be \( x = [1, 2, 3] \)

In [6]: x = newton_method([1;2;3],f1,1000,-100,0.00001)
Out[6]: 3-element Array{Float64,1}:
0.0
-2.0
-3.0

and we find the value of \( x \) that minimizes \( f(x) \) is \([0, -2.0, -3.0]\)
5 Example 2

Define a nonlinear function $B$

$$B(x) = e^{x_1^3 + x_2^2 + x_1x_2}$$

In [7]: B(x) = exp(x[1]^3+x[2]^2 + x[1]*x[2])

Out[7]: B (generic function with 1 method)

Generate a set of data by the above nonlinear function $B$

In [8]: xs1 = 0:0.02:1
xs2 = 0:0.02:1

datar=[0 0 B([0,0])]
for i in 1:length(xs1)
    for j in 1:length(xs2)
        data = [data;xs1[i] xs2[j] B([xs1[i]; xs2[j]])]
    end
end

Define a nonlinear function $A$ to fit the data generated by function $B$

$$A(x;a) = a_1x_1^2 + a_2x_2^2 + a_3x_1x_2 + a_4$$


Out[9]: A (generic function with 1 method)

Now assume the parameter of the function that generate the dataset above is unknow. This is a $\mathbb{R}^1 \rightarrow \mathbb{R}^1$ nonlinear least square problem, that is to find the optimized parameters $a$ in function $A$. In Gauss-Newton Method, set the initial parameters to be $[1, 2, 3, 4]$

In [10]: @time begin
    initial = [1;2;3;4]
    optimize = gaussNewton(initial,A,data,100,10^-5)
end

Loop time: 2
2.736285 seconds (5.03 M allocations: 511.953 MiB, 9.83% gc time)

Out[10]: 4-element Array{Float64,1}:
    2.139104342427509
    2.103249990271506
    6.169006304472237
   -0.053387257275212374
In [11]: ys = data[:,length(data[1,:])]
   xs = data[:,1:(length(data[1,:])-1)]
   yp=[]
   for i in 1:length(ys)
       #println(xs[i,:])
       yp = [yp;A(xs[i,:];a=optimize)]
   end
   residual = norm(yp-ys)
   println("Residual: ",residual)

Residual: 51.05062195039618

In Levenberg-Marquardt method, set the initial parameters to be [1, 2, 3, 4]

In [12]: @time begin
   initial = [1;2;3;4]
   optimize = LevenbergMarquardt(initial,A,data,100,10^-5)
   end

Loop time: 2
   1.679500 seconds (2.88 M allocations: 453.333 MiB, 10.05% gc time)

Out[12]: 4-element Array{Float64,1}:
    2.139104342427509
    2.1032499902715056
    6.169006304472237
   -0.05338725727521243

In [13]: ys = data[:,length(data[1,:])]
   xs = data[:,1:(length(data[1,:])-1)]
   yp=[]
   for i in 1:length(ys)
       #println(xs[i,:])
       yp = [yp;A(xs[i,:];a=optimize)]
   end
   residual = norm(yp-ys)
   println("Residual: ",residual)

Residual: 51.050621950396184

By using Guass-Newton method and Levenberg-Marquardt method, We find the optimized parameters are [2.14, 2.10, 6.17, -0.05], and the residual is 51.05

6 Example 3

To compare these two methods, in this example, we will consider a complicated function
Define a nonlinear function $B_1$

$$B_1(x) = \frac{5x_1^2}{3 + 4 \times 3^{1.5x_2}}$$

In [14]: B1(x) = 5*x[1]^2/(3+4*3^((1.5*x[2])))

Out[14]: B1 (generic function with 1 method)

Generate a set of data by the above nonlinear function $B_1$

In [15]: xs1 = 0:0.02:1
   xs2 = 0:0.02:1
   data1 = [0 0; B1([0,0])]
   for i in 1:length(xs1)
       for j in 1:length(xs2)
           data1 = [data1;xs1[i] xs2[j] B1([xs1[i]; xs2[j]])]
       end
   end

Define a nonlinear function $A_1$ to fit the data generated by function $B_1$

$$A_1(x; a) = \frac{a_1 x_1^2}{a_2 + a_3 \times 3^{a_4 x_2}}$$

In [16]: A1(x; a) = a[1]*x[1]^2/(a[2]+a[3]*3^(a[4]*x[2]))

Out[16]: A1 (generic function with 1 method)

Now assume the parameter of the function that generate the dataset above is unknow. This is a R1->R1 nonlinear least square problem, that is to find the optimized parameters $a$ in function $A_1$. In Gauss-Newton Method, set the initial parameters to be [1, 2, 3, 4]

In [17]: @time begin
       initial = [1;2;3;4]
       optimize = gaussNewton(initial,A1,data1,100,10^-5)
     end

Loop time: 100
9.053897 seconds (13.04 M allocations: 13.283 GiB, 17.17% gc time)

Out[17]: 4-element Array{Float64,1}:
   NaN
   NaN
   NaN
   NaN

7
In [18]:
ys = data1[:,length(data1[1,:])]
xs = data1[:,1:(length(data1[1,:])-1)]
yp=[]
for i in 1:length(ys)
    yp = [yp;A1(xs[i,:];a=optimize)]
end
residual = norm(yp-ys)
println("Residual: ",residual)

Residual: NaN

In Levenberg-Marquardt method, set the initial parameters to be [1, 2, 3, 4]

In [19]: @time begin
    initial = [1;2;3;4]
    optimize = LevenbergMarquardt(initial,A1,data1,100,10^-5)
end
Loop time: 53
6.658476 seconds (15.96 M allocations: 9.996 GiB, 17.02% gc time)

Out[19]: 4-element Array{Float64,1}:
    0.9989291222734306
    5.426350476005932
   -4.156828381469578
   -0.6716430753650134

In [20]:
ys = data1[:,length(data1[1,:])]
xs = data1[:,1:(length(data1[1,:])-1)]
yp=[]
for i in 1:length(ys)
    yp = [yp;A1(xs[i,:];a=optimize)]
end
residual = norm(yp-ys)
println("Residual: ",residual)

Residual: 0.9495149247371497

By using Levenberg-Marquardt method, We find the optimized parameters are [1.00, 5.43, -4.16, -0.67], and the residual is 0.95

Compare Levenberg-Marquardt method with Guass-Newton method, we found that Levenberg-Marquardt method can apply to more situations