Second Order Optimization

By Hrishikesh Patel, Raghav Dhanuka, Siddharth Uniyal & Dicky Ardian Yunindra Wardana

1) Non-linear Equations and Least Squares

Equations **not** having form $f(x) = ax + b$ or $f(x) = constant$ are considered as non-linear equations.

i.e. $f(x) = e^x + x^2$ is a non-liner equation.

1.1) Set of Non-linear equations

Consider a set of m equations in n variables $x_1, \ldots, x_n$.

$f_i(x) = 0; i = 1, \ldots, m$

Here i^{th} equation $f_i(x)$ is also a i^{th} residual. And $x = (x_1, \ldots, x_n)$ is a vector of unknowns.

Collectively $f(x) = (f_1(x), \ldots, f_m(x))$, where $f(x)$ is a vector of residuals.

For example,

Consider a system of two non-linear equations $e^{x_1} + (x_2)^3 = 4$ and $(x_1)^2 + cos(x_2) = 10$.

Here $x = (x_1, x_2)$ is a 2-vector of unknowns,

$f_1(x) = e^{x_1} + (x_2)^3 - 4$ and $f_2(x) = (x_1)^2 + cos(x_2) - 10$ are two residuals and

$f(x) = (e^{x_1} + (x_2)^3 - 4,\ (x_1)^2 + cos(x_2) - 10)$ is a vector of residuals.

1.2) Non-linear Least squares

Here our goal is to find $\widehat{x}$ which minimizes $\|f(x)\|^2\ = f_1(x)^2 + \ldots + f_m(x)^2$.

Optimality condition for any $\widehat{x}$ being a solution is to satisfy $\nabla\|f(x)\|^2\ = 0$.

$2Df(x)^T f(\hat{x}) = 0$

Important: The optimality condition is necessary condition but not sufficient. There may be other values that satisfy the condition but they are not solutions.

1.3) Difficulty in solving Non-linear Least squares problem

Non-linear least squares are difficult to solve in reality as there is no luxury to have QR factorisation like in Linear equations and least squares problem.

However, there are some heuristics algorithms like following are useful for solving the equations.
1) Gauss-Newton algorithm
2) Levenberg–Marquardt algorithm

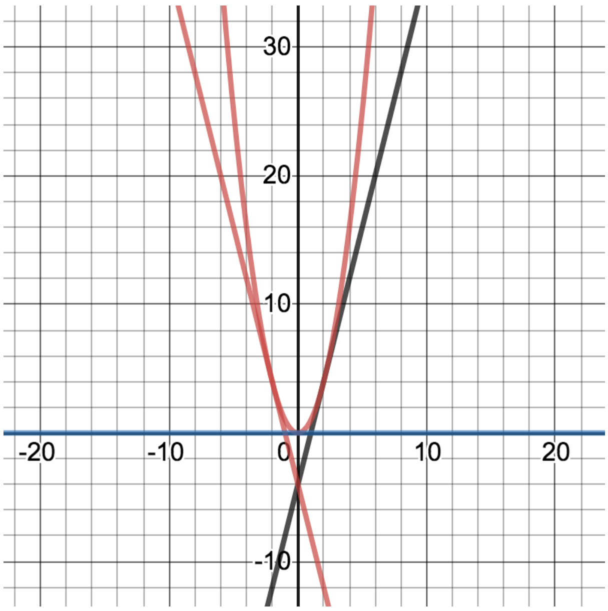
Now, if the function is **convex** then it reduces computation to find the minimum value of the function. Let's understand the concept of Convexity.

Convexity

What is convexity ?

A function is convex if its graph remains above its tangents.

e.g. $f(x) = x^2$
It is clear that the tangents remain below the graph of x^2



Mathematically, function $F(x_1, \ldots, x_n)$ is convex if its Hessian matrix (second derivative matrix) is positive semidefinite at all x.

Why convexity ?

Convexity prevents two local minima and hence if function is convex then it would have usually one minimum value.

Newton Algorithm

$$x^{k+1} = x^k - (\bigtriangledown f(x)^T \bigtriangledown f(x))^{-1} \bigtriangledown f(x)^T * f(x)$$

$$H = \nabla f(x)^T \nabla f(x)$$

In [2]:

```
using LinearAlgebra,Plots

function newton(f, Df, x1; kmax = 20, tol = 1e-6)
    x = x1
    n = length(x)
    fnorms = zeros(0,1)
    for k = 1:kmax
        fk = f(x)
        Dfk = Df(x)
        fnorms = [fnorms; norm(fk)]
        if norm(fk) < tol
            break
        end

        if det(Dfk'Dfk) == 0
            println("(Dfk'Dfk) matrix has dependent Columns, therefore, Quitting")
            break
        end

        x = x - inv(Dfk'Dfk)*Dfk'fk

    end
    return x, fnorms
end
```

Out[2]:

```
newton (generic function with 1 method)
```

In [3]:

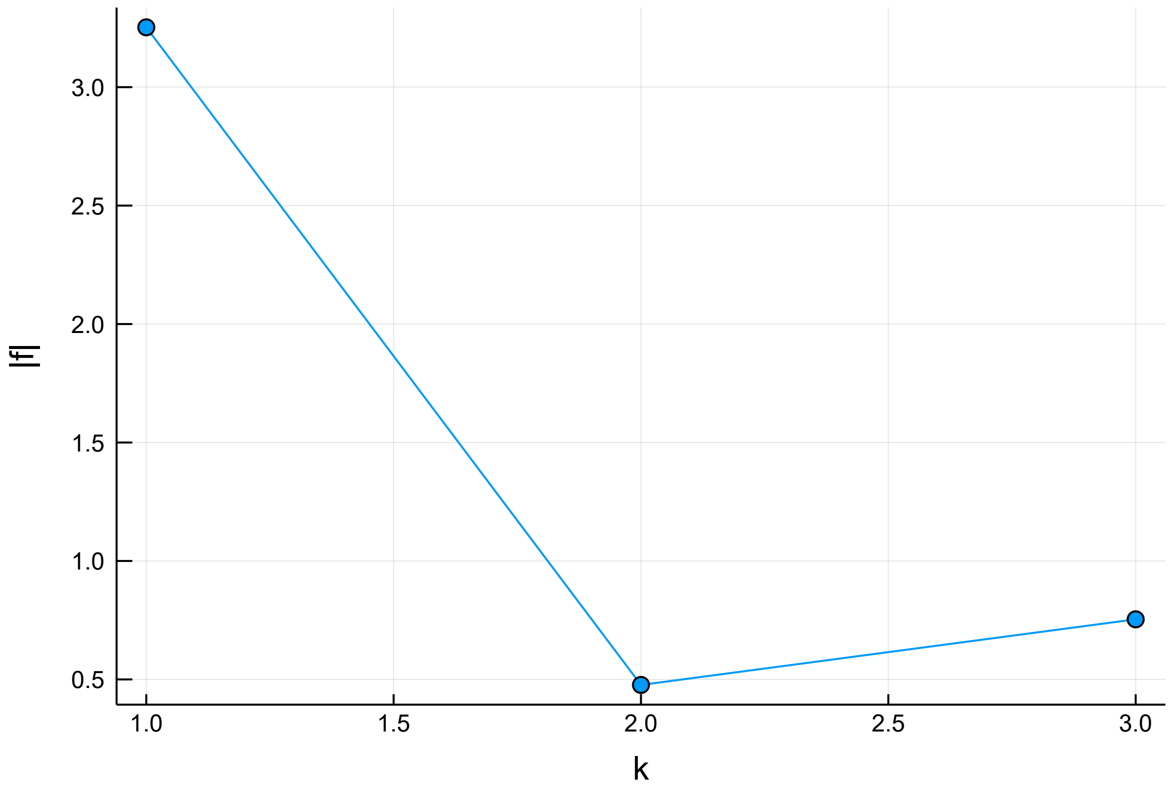
```
f(x) = (x[1]-3)^2/4 + (x[2]-5)^2/9
Df(x) = [(x[1]-3)/2 2*(x[2]-5)/9]

x, gnorms = newton(f, Df, [0.5, 1.1])
println("")
println("Moved ", length(gnorms), " steps before quitting.")
```

```
(Dfk'Dfk) matrix has dependent Columns, therefore, Quitting

Moved 3 steps before quitting.
```

In [4]:

```
plot(gnorms, shape=:circle, legend = false, xlabel = "k", ylabel = "|f|")
```

Out[4]:



levenberg-Marquardt Algorithm

$$x^{k+1} = x^k - (\bigtriangledown f(x)^T \bigtriangledown f(x) + \lambda I)^{-1} \bigtriangledown f(x)^T * f(x)$$

In [5]:

```
eye(n) = 1.0 .* Matrix(I,n,n)
function levenberg_marquardt(f, Df, x1, lambda1; kmax=100, tol=1e-6)
    n = length(x1)
    x = x1
    lambda = lambda1
    objectives = zeros(0,1)
    residuals = zeros(0,1)

    for k = 1:kmax
        fk = f(x)
        Dfk = Df(x)

        objectives = [objectives; norm(fk)^2]
        residuals = [residuals; norm(2*Dfk'*fk)]

        if norm(2*Dfk'*fk) < tol
            break
        end

        xt = x - inv(Dfk'Dfk + lambda*eye(n))*Dfk'fk

        if norm(f(xt)) < norm(fk)
            lambda = 0.8*lambda
            x = xt
        else
            lambda = 2.0*lambda
        end
    end

    return x, Dict([("objectives", objectives),("residuals", residuals)])
end
```

Out[5]:

```
levenberg_marquardt (generic function with 1 method)
```

In [6]:

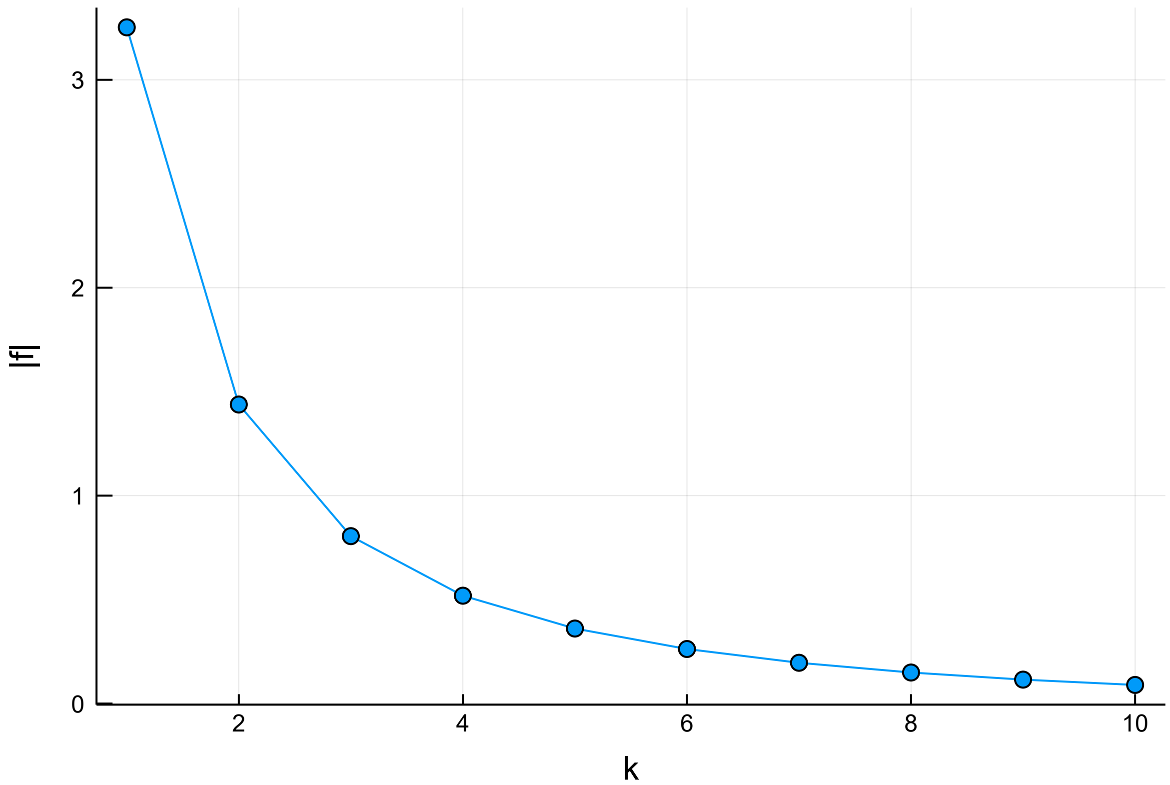
```
f(x) = (x[1]-3)^2/4 + (x[2]-5)^2/9
Df(x) = [(x[1]-3)/2 2*(x[2]-5)/9]

x, history = levenberg_marquardt(f, Df, [0.5, 1.1], 1.0)

println("Moved ", length(history["residuals"][1:10]), " steps before function converges to
```

```
Moved 10 steps before function converges to minimum with final step [2.9999
9; 4.97316]
```

In [7]:

```
plot(sqrt.(history["objectives"][1:10]), shape = :circle,legend = false, xlabel = "k", ylab
```

Out[7]:



Data Fitting

As in linear model fitting, we choose the parameter θ by (approximately) minimizing the sum of the squares of the prediction residuals

$$\sum_{i=1}^{N} \left(\widehat{f} \left(x^{(i)}; \theta\right) - y^{(i)} \right)^2$$

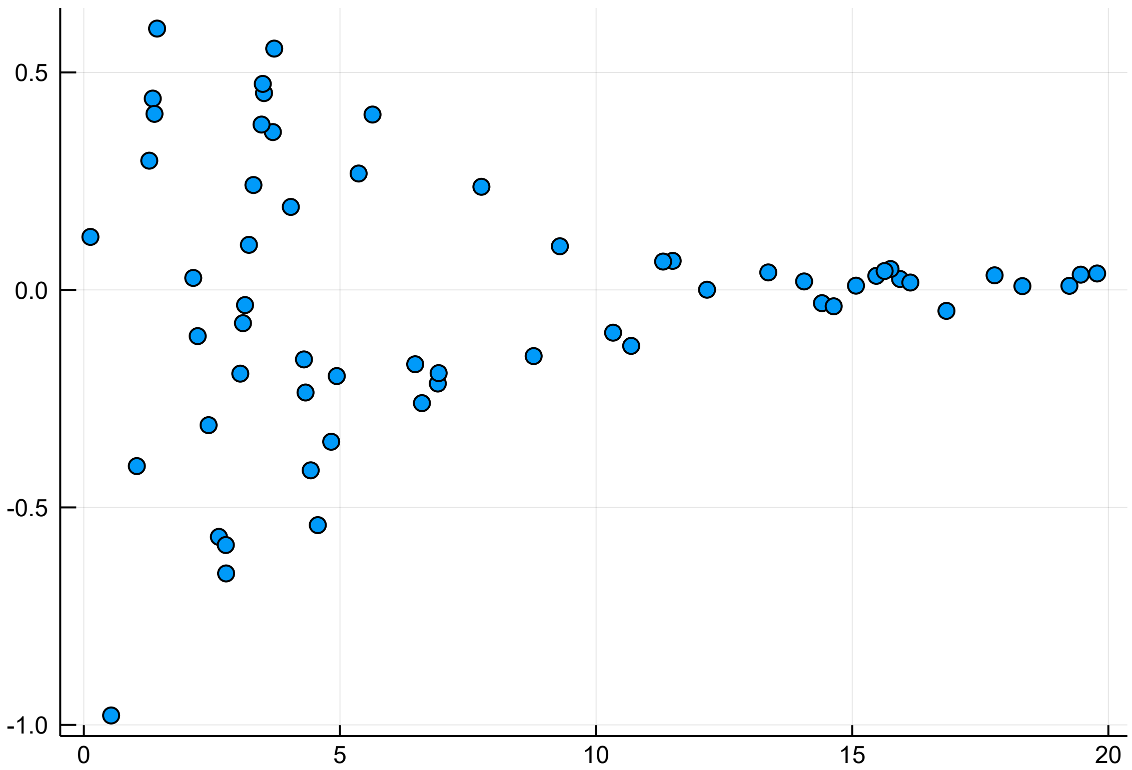
Function we have used here is :

$$f(x; \theta) = \theta_1 e^{\theta_2 x} \cos(\theta_3 x + \theta_4)$$

In [8]:

```
using Random
Random.seed!(45)

theta_ex = [1, -0.2, pi, pi/3]
#Choose 60 points x between 0 and 20.
M = 30
xd = [5*rand(M); 5 .+ 15*rand(M)]

# Evaluate function at these points.
yd = theta_ex[1] * exp.(theta_ex[2]*xd) .*cos.(theta_ex[3] * xd .+ theta_ex[4])

# Create a random perturbation of yd.
N = length(xd)
yd = yd .* (1 .+ 0.2*randn(N)) .+ 0.015 * randn(N)

# Plot data points.
using Plots
scatter(xd, yd, legend=false)
```

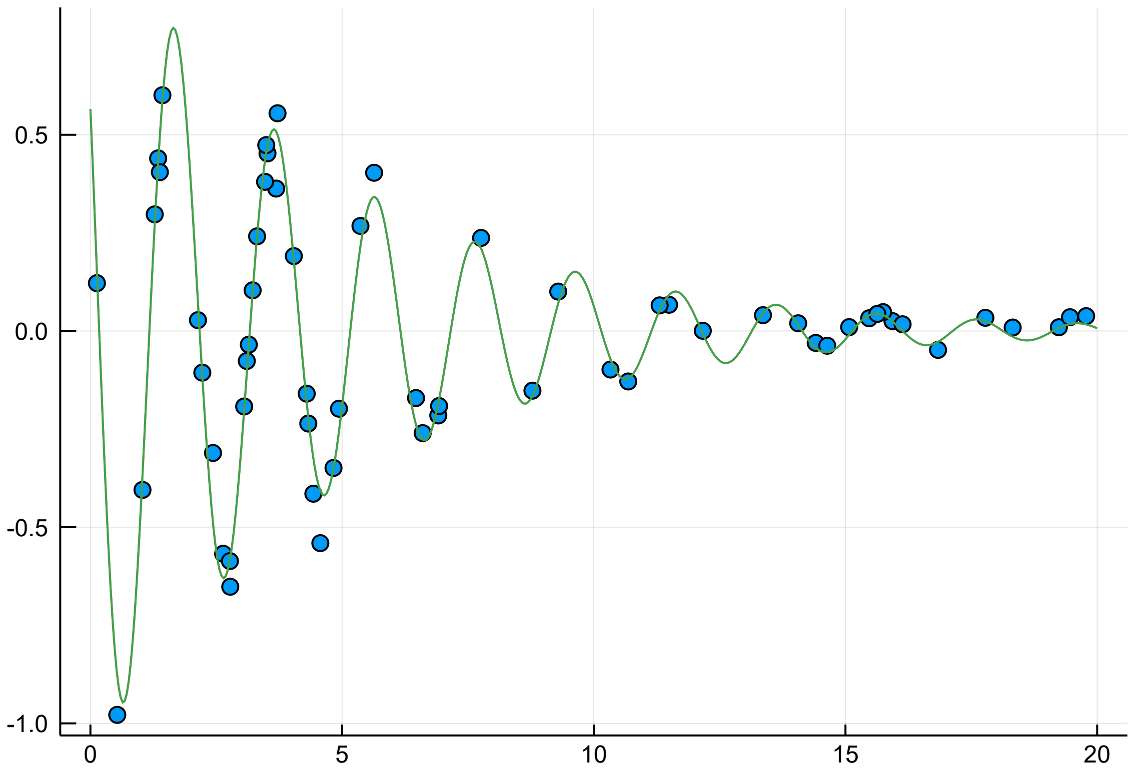
Out[8]:



In [10]:

```
f(theta) = theta[1] * exp.(theta[2]*xd) .*cos.(theta[3] * xd .+ theta[4]) - yd
Df(theta) = hcat(exp.(theta[2]*xd) .* cos.(theta[3] * xd .+ theta[4]),theta[1] * ( xd .* ex
cos.(theta[3] * xd .+ theta[4])),-theta[1] * ( exp.(theta[2]*xd) .* xd .*sin.(theta[3] * xc
-theta[1] * ( exp.(theta[2]*xd) .*sin.(theta[3] * xd .+ theta[4])) )

theta1 = [1, 0, 1, 0]
theta, history = levenberg_marquardt(f, Df, theta1, 1.0)
theta

# Plot the fitted model.
x = range(0, stop= 20, length = 500)
y=theta[1]*exp.(theta[2]*x) .* cos.(theta[3]*x .+ theta[4])
plot!(x, y, legend = false)
```

Out[10]:



In [ ]: