Lecture 04
Solving Multivariate Unconstraint Nonlinear

Oran Kittithreerapronchai[1]

[1]Department of Industrial Engineering, Chulalongkorn University
Bangkok 10330 THAILAND

last updated: August 7, 2023

Outline

source: General references [NC20, CŻ13, Win22, Pat14]

Idea of Algorithm

1. choose a certain point
2. convert into an approximate function using Taylor's Series
3. move at a certain direction with certain step size
4. check Stop Condition
   - Gradient = **zero** → stuck
   - Objective value **unchanged** → not significant improvement
   - Solution **unchanged** → not significant improvement
   - Too many iterations
5. check all Eigenvalue are positive → local optimal
6. repeat Step 1

Algorithm = how to find direction $(\mathbf{d}_k)$ and stepsize (γ_k)

Concept of Gradient Method

- **Background:** gradient = normal vector = maximum increasing rate of $f(\cdot)$
- **Key result:** $-\nabla f(\mathbf{x}) =$ maximum decreasing rate
- **Idea:** better to move $\mathbf{x}$ in gradient direction (downhill trails)
- **Properties:** zig-zag & always descent (obj reduce value)
- **Questions:**
  - What is a 'good' step size?
  - How much direction and step size really matter?
- **Variation:**

  Steepest descending: find optimal step size, i.e. line search
  Conjugate direction: search direction orthogonal to one other
  Special Cases: fixed /closed-form step size for Quadratic

Gradient Method

$$
\begin{aligned}
(P) \quad & \text{minimize} && f(\mathbf{x}) \\
& \text{subject to} && \\
& && \mathbf{x} \in \mathbb{R}^n.
\end{aligned}
$$

where,

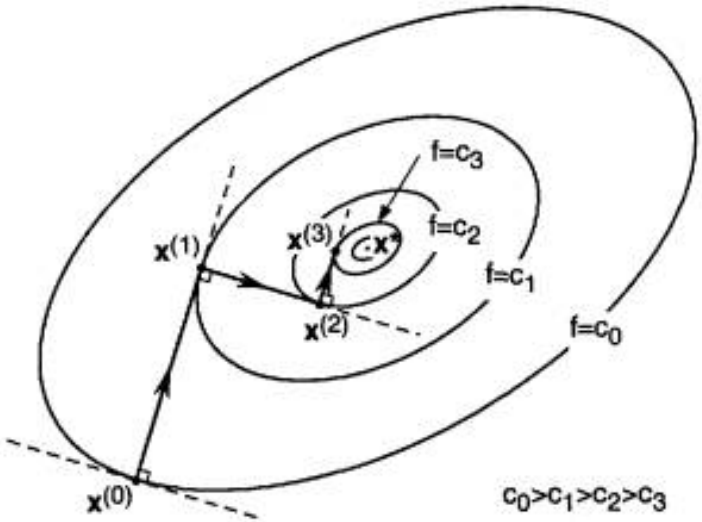
$\mathbf{f}(\cdot) =$ objective function and differentiable $f : \mathbb{R}^n \to \mathbb{R}$

$\mathbf{x} =$ decision variable, $\mathbf{x} \in \mathbb{R}^n$

Solution

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \gamma_k \nabla f(\mathbf{x})$$

Interpretation of Gradient



Source. Chong & Zak. 2001 pp 114-155 [CŻ13]

Idea

- **New point:** $\mathbf{x_{k+1}} = \mathbf{x_k} - \gamma_k \nabla f(\mathbf{x_k})$
- **Scent direction:** $\mathbf{d} = \nabla f(\mathbf{x})^T \mathbf{d} < 0$
- **Steepest direction:** $-\nabla f(\mathbf{x})$
- **Better value:** $f(\mathbf{x_k} - \gamma_k \nabla f(\mathbf{x_k})) = f(\mathbf{x_{k+1}}) < f(\mathbf{x_k})$

Terminating Conditions

- **Iteration:** $k > t_{\max}$
- **Gradient:** $\nabla f(\mathbf{x_{k+1}}) \approx \mathbf{0}$ or $\|\nabla f(\mathbf{x_{k+1}})\| = 0$
- **Solution:**

$$\frac{\|\mathbf{x_{k+1}} - \mathbf{x_k}\|}{\|\mathbf{x_k}\|} < \epsilon_{\mathbf{x}}$$

- **Objective Value:**

$$\frac{|f(\mathbf{x_{k+1}}) - f(\mathbf{x_k})|}{|\mathbf{x_k}|} < \epsilon_{f(\mathbf{x})}$$

Check SNOC:

- all eigenvalues of $\nabla^2 f(\mathbf{x_{k+1}}) > 0$

Example: Gradient Method

Using Steepest Descent Method to find solution of this following function.

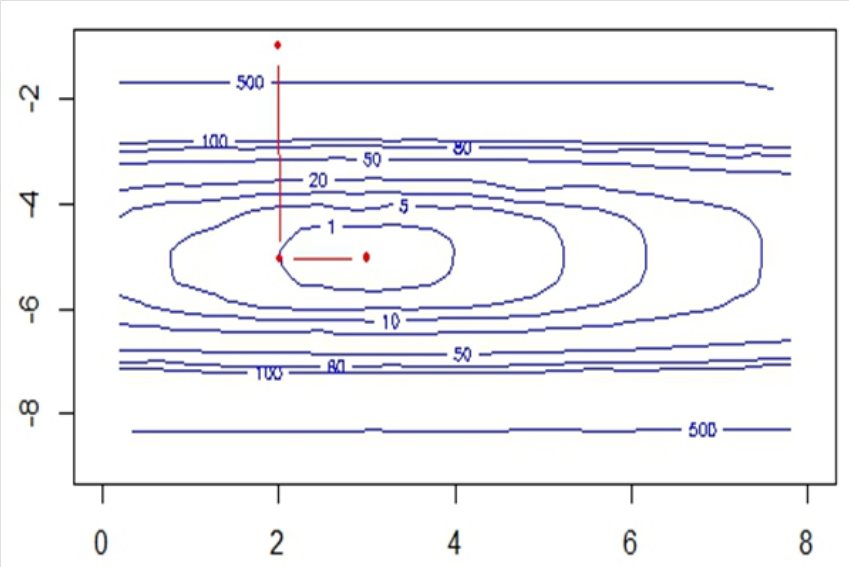
$$f(x_1, x_2, x_3) = (x_1 - 4)^4 + (x_2 - 3)^2 + 4(x_3 + 5)^4$$

The initial points is $\mathbf{x_0} = [4, 2, -1]^T$.

Source. Chong & Zak. 2001 pp 118 [CŻ13]

k	$\mathbf{x_k^T}$	$f(\mathbf{x_k})$	$\nabla f(\mathbf{x_k})^T$	γ_k
0	$[4.0, 2.0, -1.0]$	1024.0	$[0.0, -2.0, 1024.0]$	0.003967
1	$[4.0, 2.008, -5.062]$	0.984	$[0.0, -1.984, -0.004]$	0.5
2	$[4.0, 3.0, -5.06]$	0.000	$[0.0, 0.0, -0.003]$	16.29

Level Set of Example




```
f.expr  <- expression( (x1 - 4)^4 +(x2 - 3)^2 + 4*(x3 + 5)^4 )

exprFn <- function(x1,x2,x3){ }
body(exprFn) <- f.expr
x2Grid <- seq(0.0,8.0,0.1) ; x3Grid <- seq(-9.0,-1.0,0.1)
objGrd <- outer(x2Grid,x3Grid,exprFn,x1=4)

contour(x=x2Grid,y=x3Grid,z=objGrd,col="blue"
        ,levels=c(500,100,80,50,20,5,1))
pt <- data.frame(x2=c(2,2.008,3.0),x3=c(-1,-5.06,-5.06))
points(pt$x2,pt$x3,col="red",pch=16,cex=0.8,type="b")
```

Code of Optimizing γ (Steepest Descending)

- **Input:**

```
f.expr  <- expression( x1 - 4)^4 +(x2 - 3)^2 + 4*(x3 + 5)^4 )
expr.all<- deriv3(f.expr,c("x1","x2","x3"))
```

- **Define:**

```
x1Term   <- quote(pt[1]-grad[1]*step)
x2Term   <- quote(pt[2]-grad[2]*step)
x3Term   <- quote(pt[3]-grad[3]*step)
varList  <- list(x1 = x1Term, x2= x2Term, x3= x3Term)

optStep <- function(pt,grad,step){ }
```

- **Initiate:**

```
curPt <- c(4,2,-1) ## init point

for(i in 1:3){ ## alg for loop
  evalList<- list(x1 = curPt[1],x2=curPt[2],x3 =curPt[3]) ## comb value
  curVal  <- eval(expr.all,evalList) ## eval objFn, grad, hess
  curObj  <- curVal[1]
  curGrad <- as.vector( attr(curVal,"gradient"))
  rndSize <- sqrt( as.numeric(curGrad %*% curGrad))
```

- **Stepsize:**

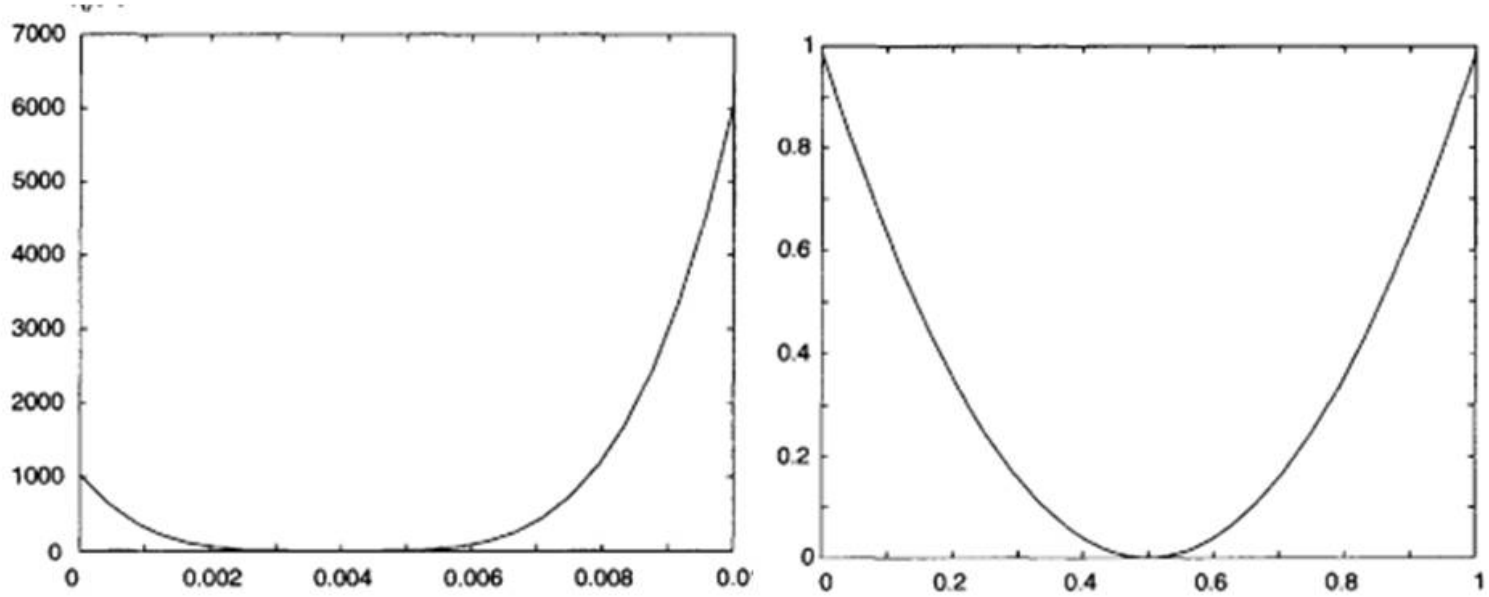
```
  expr.sub   <- eval(substitute(substitute(e, varList), list(e = f.expr[[1]])))
  body(optStep) <- as.expression(expr.sub)
  curSize <- optimize(optStep,interval=c(0.0,rndSize),pt=curPt,grad=curGrad)$minimum
```

- **Update:**

```
  print( c(curObj,curPt,curGrad,curSize) )
  nxtPt  <- curPt - curSize*curGrad

  curPt  <- nxtPt  ## repeat alg
} ## end for i
```

γ in Descent Direction



Source. Chong & Zak. 2001 pp 119 [CŻ13]

Finding the Right step size

- **Observe:** $\gamma_* = \arg\min f(\mathbf{x} - \gamma_1 \nabla f(\mathbf{x}))$ difficult to solve
- **Conclusion:** optimal γ reduce iterations. However, for loop is easy!
- **Idea:** need good γ, not optimal γ

Special Case: Quadratic Function

- **Function:** $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathbf{T}}\,\mathbf{Q}\,\mathbf{x} - \mathbf{b}^{\mathbf{T}}\,\mathbf{x}$
- **Gradient:** $\nabla f(\mathbf{x}) = \mathbf{Q}\,\mathbf{x} - \mathbf{b}$

  where,

  $\mathbf{Q}$ = symmetric positive definite matrix, $\mathbf{Q} \in \mathbb{R}^{n \times n}$
  $\mathbf{b}$ = vector corresponding to linear term, $\mathbf{b} \in \mathbb{R}^{n}$

$$\gamma_k = \frac{\nabla f(\mathbf{x_k})^T \nabla f(\mathbf{x_k})}{\nabla f(\mathbf{x_k})^T \mathbf{Q} \nabla f(\mathbf{x_k})}$$

Strategy for step size

- **In general:** $\nabla f(\mathbf{x_k})$ is easy, but $\gamma_k = \arg\min_{\gamma_k} f(\mathbf{x_k} - \gamma_k \nabla f(\mathbf{x_k}))$ is not
- **Desired properties:** If $\|\mathbf{d}_k\| = 1$,

  Descent Direction $f(\mathbf{x_k}) > f(\mathbf{x_k} + \gamma_k \mathbf{d}_k) \qquad \forall k$

  Sufficient $\sum \gamma_k = \infty$

  Convert $\sum \gamma_k^2 < \infty$
- **Idea:** use easy to compute γ

  Close form: known in Quadratic, $\gamma_k = \frac{\nabla f(\mathbf{x_k})^T \nabla f(\mathbf{x_k})}{\nabla f(\mathbf{x_k})^T \mathbf{Q} \nabla f(\mathbf{x_k})}$

  Fixed step size: known in Quadratic, $0 < \gamma < \frac{2}{\lambda_{\max}(\mathbf{Q})}$

  Pick one in many γ: choose easy and better γ

Check point

- $\|\nabla f(\mathbf{x_k})\| \neq \|\nabla f(\mathbf{x_{k+1}})\|$, so $\gamma_k \overset{?}{=} \gamma_{k+1}$
- easy $\gamma \rightarrow$ fixed or specific function
- better $\gamma \rightarrow$ improve objective value

Example

Resolve and compare $f(x_1, x_2, x_3) = (x_1 - 4)^4 + (x_2 - 3)^2 + 4(x_3 + 5)^4$ using

A)

$$\gamma_k = \frac{2}{\lambda_{\max}(\nabla^2 f(\mathbf{x_k}))}, \text{ where } \lambda_{\max}(\nabla^2 f(\mathbf{x_k})) \text{ is eigenvalue}$$

B)

$$\gamma_k = \begin{cases} 2\,\gamma_{k-1} & \text{if } f(\mathbf{x_k}) < f(\mathbf{x_{k-1}}), \\ \frac{1}{2}\,\gamma_{k-1} & \text{otherwise.} \end{cases}$$

Concept of Newton's Method

- **Background:** Taylor's approximation upto quadratic terms
- **Idea:** move $\mathbf{x}$ along curvature of $\nabla^2 f(\mathbf{x})$
- **Properties:** may not descent or $f(\mathbf{x_{k+1}}) > f(\mathbf{x})$
- **Questions:**
  - compute $\left(\nabla^2 f(\mathbf{x})\right)^{-1}$
  - ensure descent direction (use weight?)

Univariate VS Multivariate Newton

Univariate Newton

- **Approx:** $g(x) = f(x_k) + (x - x_k)f'(x_k) + \frac{1}{2}(x - x_k)^2 f''(x_k)$
- **FOC:** $f'(x_k) = (x - x_k)f''(x_k)$
- **Next point:**

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

- **Terminate condition:** $f'(x_k) = 0$

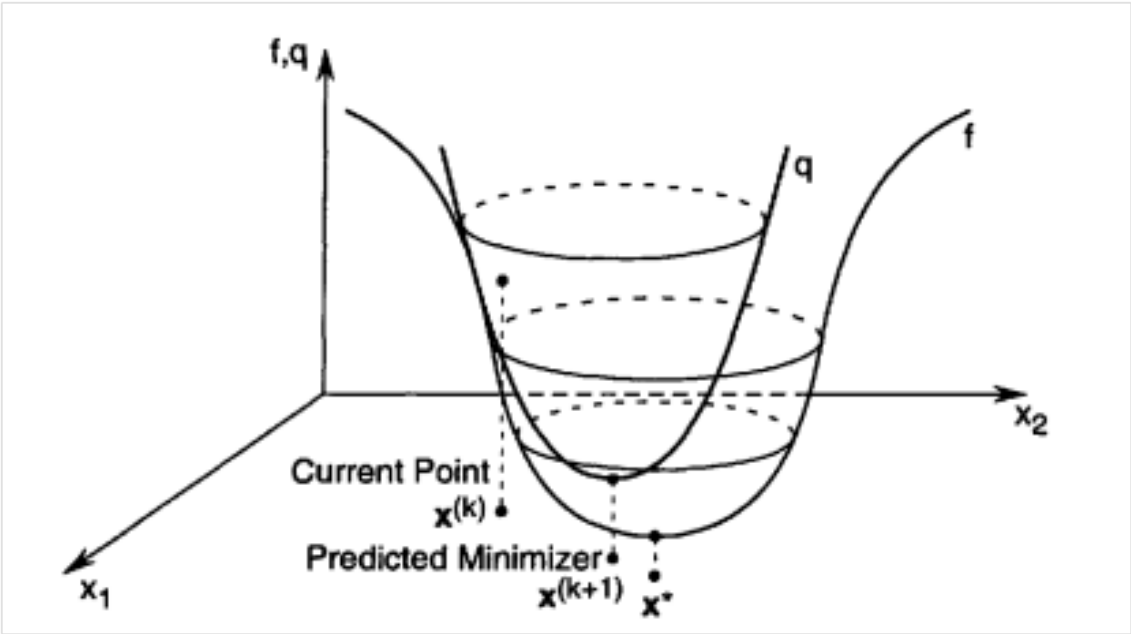
Multivariate Newton

- **Approx:** $g(\mathbf{x_k}) = f(\mathbf{x_k}) + (\mathbf{x} - \mathbf{x_k})^T \nabla f(\mathbf{x_k}) + \frac{1}{2}(\mathbf{x} - \mathbf{x_k})^T \left(\nabla^2 f(\mathbf{x})\right)(\mathbf{x} - \mathbf{x_k})$
- **FOC:** $\nabla f(\mathbf{x_k}) = (\mathbf{x} - \mathbf{x_k})^T \left(\nabla^2 f(\mathbf{x})\right)$
- **Next point:**

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \mu_k \left(\nabla^2 f(\mathbf{x})\right)^{-1} (\nabla f(\mathbf{x_k}))$$

- **Terminate condition:** $\|\nabla f(\mathbf{x_k})\| = 0$

Interpretation of Newton



Source. Chong & Zak. 2001 pp 114-155 [CŻ13]

Idea

- **New point:** $\mathbf{x_{k+1}} = \mathbf{x_k} - \mu_k \left(\nabla^2 f(\mathbf{x})\right)^{-1} \nabla f(\mathbf{x_k})$
- **Direction:** $\mathbf{d} = \left(\nabla^2 f(\mathbf{x})\right)^{-1} \nabla f(\mathbf{x_k})$
- **Scent direction:** $\mathbf{d} = \nabla f(\mathbf{x})^T \mathbf{d} \not< 0$

Example: Newton's Method

Using Newton Method to find solution of this *Powell* function for three iteration.

$$f(\mathbf{x}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

The initial points is $\mathbf{x_0} = [3, -1, 0, 1]^T$.

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 2(x_1 + 10x_2) + 40(x_1 - x_4)^3 \\ 20(x_1 + 10x_2) + 4(x_2 - 2x_3)^3 \\ 10(x_3 - x_4) - 8(x_2 - 2x_3)^3 \\ -10(x_3 - x_4) - 40(x_1 - x_4)^3 \end{bmatrix}$$

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} 2 + 120(x_1 - x_4)^2 & 20 & 0 & -120(x_1 - x_4)^2 \\ 20 & 200x_2 + 12(x_2 - 2x_3)^2 & -24(x_2 - 2x_3)^2 & 0 \\ 0 & -24(x_2 - 2x_3)^2 & 10 + 48(x_2 - 2x_3)^2 & -10 \\ -120(x_1 - x_4)^2 & 0 & -10 & 10 + 120(x_1 - x_4)^2 \end{bmatrix}$$

Example: Newton's Method

$$f(\mathbf{x}) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

The initial points is $\mathbf{x_0} = [3, -1, 0, 1]^T$.

Source. Chong & Zak. 2001 pp 140 [CŻ13]

k	$\mathbf{x_k^T}$	$f(\mathbf{x_k})$	$\nabla f(\mathbf{x_k})$	$\left(\nabla^2 f(\mathbf{x})\right)$	$\left(\nabla^2 f(\mathbf{x})\right)^{-1} \nabla f(\mathbf{x_k})$
0	$\begin{bmatrix} 3 \\ -1 \\ 0 \\ 1 \end{bmatrix}$	215	$\begin{bmatrix} 306.0 \\ -144.0 \\ -2.0 \\ -360.0 \end{bmatrix}$	$\begin{bmatrix} 482.0 & 20.0 & 0.0 & -480.0 \\ 20.0 & 212.0 & -24.0 & 0.0 \\ 0.0 & -24.0 & 58.0 & -10.0 \\ -480.0 & 0.0 & -10.0 & 490.0 \end{bmatrix}$	$\begin{bmatrix} 1.413 \\ -.841 \\ -.254 \\ .746 \end{bmatrix}$
1	$\begin{bmatrix} 1.587 \\ -.159 \\ .254 \\ .254 \end{bmatrix}$	31.8	$\begin{bmatrix} 94.8 \\ -1.18 \\ 2.37 \\ -94.81 \end{bmatrix}$	$\begin{bmatrix} 215.3 & 20.0 & 0.0 & -213.3 \\ 20.0 & 205.3 & -10.7 & 0.0 \\ 0.0 & -10.7 & 31.3 & -10.0 \\ -213.3 & 0.0 & -10.0 & 233.3 \end{bmatrix}$	$\begin{bmatrix} 0.589 \\ -.053 \\ .085 \\ .085 \end{bmatrix}$
2	$\begin{bmatrix} 1.058 \\ -.106 \\ .169 \\ .169 \end{bmatrix}$	6.28	$\begin{bmatrix} 28.09 \\ -0.35 \\ 0.70 \\ -28.08 \end{bmatrix}$	$\begin{bmatrix} 96.8 & 20.0 & 0.0 & -94.8 \\ 20.0 & 202.4 & -4.74 & 0.0 \\ 0.0 & -4.74 & 19.5 & -10.0 \\ -94.8 & 0.0 & -10.0 & 104.8 \end{bmatrix}$	$\begin{bmatrix} -0.354 \\ 0.032 \\ -0.057 \\ -0.058 \end{bmatrix}$

Newton Method in R

- **Input:**

```
expr <- expression( (x1 + 10*x2)^2 +5*(x3 - x4)^2+(x2 - 2*x3)^4+10*(x1 - x4)^4 )
```

- **Define:**

```
allCal <- function(x1,x2,x3,x4){}
body(allCal) <- deriv3(expr,c("x1","x2","x3","x4"))
```

- **Initiate:**

```
curPt <- c(3.,-1.,0.,1.)

for( i in 1:10){
  allVal  <- allCal(curPt[1],curPt[2],curPt[2],curPt[4])
  objtVal <- allVal[1]
  gradVal <- as.vector(attr(allVal,"gradient"))
  hessVal <- matrix(attr(allVal,"hessian"),ncol=4)
```

- **Stepsize:**

```
  stepVal <- 0.5 ; count <- 0
  while(T){ ## for change stepsize
    nextPt  <- curPt - stepVal^count *dircVal
    nextVal <- allCal(nextPt[1],nextPt[2],nextPt[3],nextPt[4])[1]
    if(objtVal >= nextVal)
       break
    count <- count+ 1
  }   ## end while
```

- **Update:**

```
  print( c(i,objtVal,curPt,gradVal,count,nextPt) )
  curPt  <- nextPt

} ## end for loop
```

Pro & Cons of Newton's Method

Benefits

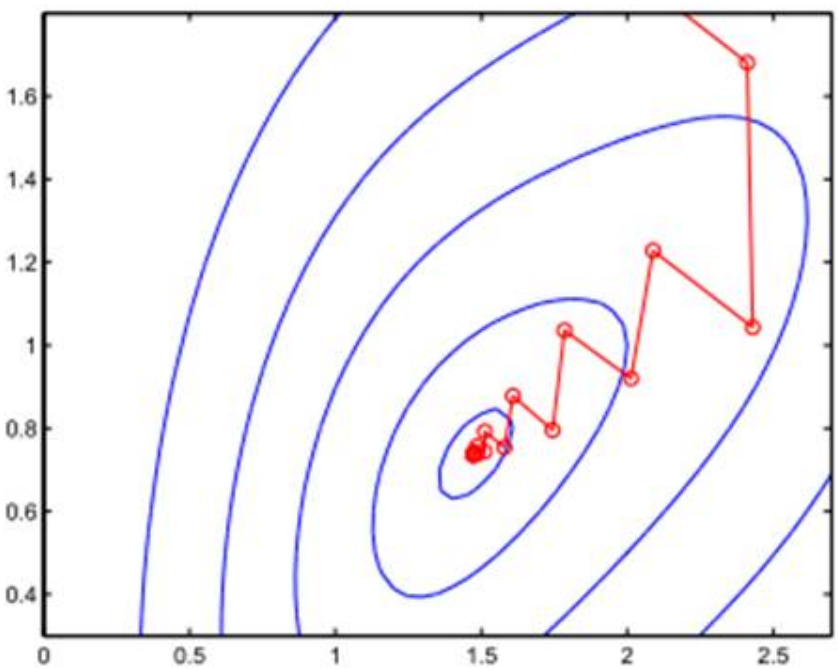
- **Fast Convergence:** quadric convergent rate, near the optimal solution
- **Key Background:** large family of algorithms, including Quadric Programming

Disadvantages

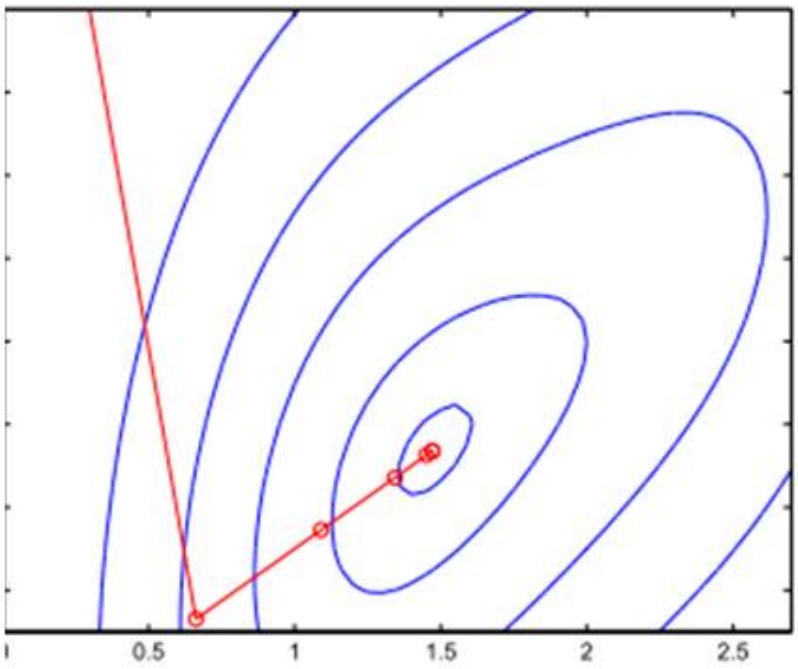
- **Computational Expensive:** inverse of hessian is time consuming
- **Difficulty:** $\mathbf{H_k}$ is not positive definite or close to singular
- **Non-Descent:** next solution may have lower objective value
- **Require modification:**

  Special case Gauss-Newton (non-linear least square)
  Non-Positive Definite Levenberg-Marquardt
  Reduce Computation Quasi-Newton

Converge Rate: Newton VS Gradient



a) Gradient Method



b) Newton's Method

Source. Conrnuejols and Tutuncu. 2006 pp 100 & 104

Gauss-Newton Method

- **What:** special case of Newton's Method for $\sum(y - \hat{y})^2$

$$f(\mathbf{x}) = \sum_{i=1}^{m} r_i(\mathbf{x})^2 = \mathbf{R}(\mathbf{x})^T \mathbf{R}(\mathbf{x})$$

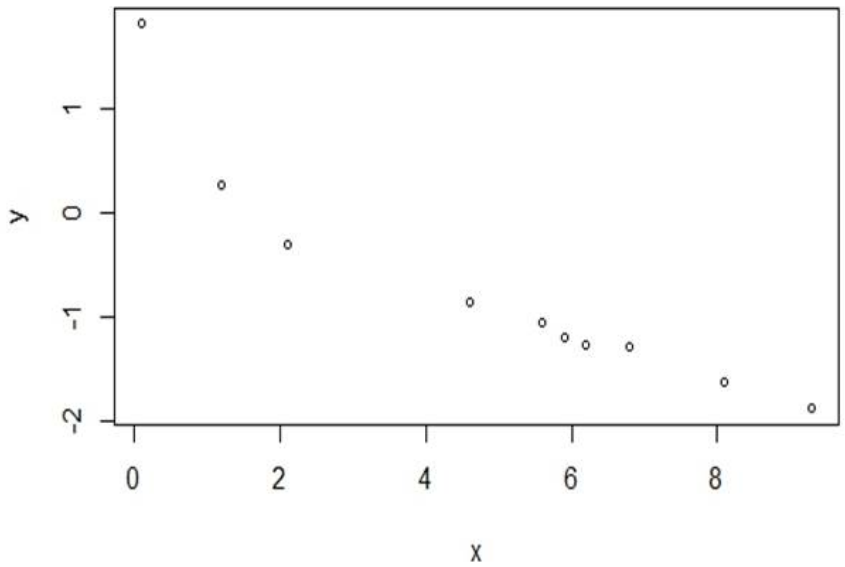
- **Gradient Function:** $\nabla f(\mathbf{x}) = 2\nabla \mathbf{R}(\mathbf{x})^{\mathbf{T}}\, \mathbf{R}(\mathbf{x})$
- **Hessian Function:** $\nabla^2 f(\mathbf{x}) \approx \ 2\nabla \mathbf{R}(\mathbf{x})^{\mathbf{T}}\, \nabla \mathbf{R}(\mathbf{x})$
- **Results:** ignores a part of hessian and easy to inverse
- **Application:** non-linear least square error

Jacobian Matrix:

$$\nabla \mathbf{R}(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial r_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial r_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial r_m}{\partial x_n}(\mathbf{x}) \end{bmatrix}$$

Example: Non-Linear Least Square

	1	2	3	4	5	6	7	8	9	10
t	0.1	1.2	2.1	4.6	5.6	5.9	6.2	6.8	8.1	9.3
y	1.816	0.267	-0.304	-0.849	-1.053	-1.200	-1.260	-1.294	-1.628	-1.881



From the empirical study, the best function to describe the relationship is:

$$\hat{y} \quad = \quad x_1 + x_2\, t + x_3\, e^{-x_4 t}$$

Estimate parameters $(x_1,\ x_2,\ x_3,\ x_4)$ for this data set.

Example: Non-Linear Least Square

$$
\begin{aligned}
\min\ z &= \sum_{i=1}^{10} f(\mathbf{Xa}; t^{(i)}, y^{(i)}) = \sum_{i=1}^{10} \left(y^{(i)} - x_1 - x_2\, t^{(i)} - x_3\, e^{-x_4 t^{(i)}}) \right)^2 \\
&= \mathbf{R}(\mathbf{x})^T \mathbf{R}(\mathbf{x}) \\
&\text{where,} \\
\mathbf{R}(\mathbf{x}) &= [y^{(i)} - x_1 - x_2 t^{(i)} - x_3\, e^{-x_4 t^{(i)}}]_{(i)} \\
\nabla \mathbf{R}(\mathbf{x}) &= \begin{bmatrix} -1 & -t^{(i)} & -e^{-x_4 t^{(i)}} & t^{(i)} x_3 e^{-x_4 t^{(i)}} \end{bmatrix}^T_{(i)} \\
2\nabla \mathbf{R}(\mathbf{x})^T \mathbf{R}(\mathbf{x}) &= \begin{bmatrix} -2\sum_i y^{(i)} - x_1 - x_2 t^{(i)} - x_3\, e^{-x_4 t^{(i)}} \\ -2\sum_i t^{(i)} (y^{(i)} - x_1 - x_2 t^{(i)} - x_3\, e^{-x_4 t^{(i)}}) \\ -2\sum_i e^{-x_4 t^{(i)}} (y^{(i)} - x_1 - x_2 t^{(i)} - x_3\, e^{-x_4 t^{(i)}}) \\ -2\sum_i x_3\, e^{-x_4 t^{(i)}} (y^{(i)} - x_1 - x_2 t^{(i)} - x_3\, e^{-x_4 t^{(i)}}) \end{bmatrix} \\
2\nabla \mathbf{R}(\mathbf{x})^T \nabla \mathbf{R}(\mathbf{x}) &= \begin{bmatrix} 2 & 2\sum_i t^{(i)} & 2\sum_i e^{-x_4 t^{(i)}} & 2x_3 \sum_i t^{(i)} e^{-x_4 t^{(i)}} \\ \cdot & 2\sum_i (t^{(i)})^2 & 2t^{(i)} \sum_i e^{-x_4 t^{(i)}} & 2x_3 \sum_i (t^{(i)})^2 e^{-x_4 t^{(i)}} \\ \cdot & \cdot & 2\sum_i e^{-2x_4 t^{(i)}} & 2x_3 \sum_i t^{(i)} e^{-2x_4 t^{(i)}} \\ \cdot & \cdot & \cdot & 2x_3^2 \sum_i (t^{(i)})^2 e^{-2x_4 t^{(i)}} \end{bmatrix}
\end{aligned}
$$

Summary

- **Overall:** good when initial point near local optimal point
- **Gradient Method:** use $\mathbf{d_{k+1}} = -\gamma_k \nabla f(\mathbf{x_k})$

  Pros descent direction,

  Cons many iterations, impractical to find the best γ_k

- **Newton's Family:** use $\mathbf{d_{k+1}} = -\mathbf{H_k} \nabla f(\mathbf{x_k})$

  Pros not descent direction, popular & practical

  Cons few iterations, computational expensive, problem near optimal

Time of R

- **Non-Linear Least Square:** 'nls '(·) for nonlinear least square

  Find parameters of nonlinear regression model $\hat{y} = x_1 + x_2\, t + x_3\, e^{-x_4 t}$ that minimizes square error (data available at `nlLeastSq.txt`)

```
name <- c("t","y")
nls(y ~ x1+x2*t+x3*exp(-1*x4*t), data=nlLeastSq,start=c(x1=0.1,x2=0.5,x3=1.2,x4=0.7))
```

- **General Optimization:** 'optim '(·) for general nonlinear optimization

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathbf{T}} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{bmatrix} \mathbf{x} - \begin{bmatrix} -8 \\ -9 \\ -8 \end{bmatrix} \mathbf{x}$$

  Find the minimize of $f(\mathbf{x})$ if $\mathbf{x_0} = [0, 0, 0]^T$

```
objFn <- function(x){  allFn(x[1],x[2],x[3])[1]}
grdFn <- function(x){
  grad <- attr(allFn(x[1],x[2],x[3]),"gradient")
  return(as.vector(grad))
  }

optim(c(0,0,0),fn=objFn,gr=grdFn,method="BFGS")
optim(c(0,0,0),fn=objFn)
```

Single Facility Location Problem

Find the location of single warehouse that minimize the total cost from the following location:

No.	customer	location ($\mathbf{p_i}$)	# shipments (w_i)
1	A	(8,2)	9
2	B	(3,10)	7
3	C	(8,15)	2
4	D	(3,4)	6
5	E	(16,8)	7
6	F	(4,5)	5
	X	(x, y)	-

- If the total transportation cost is estimated by

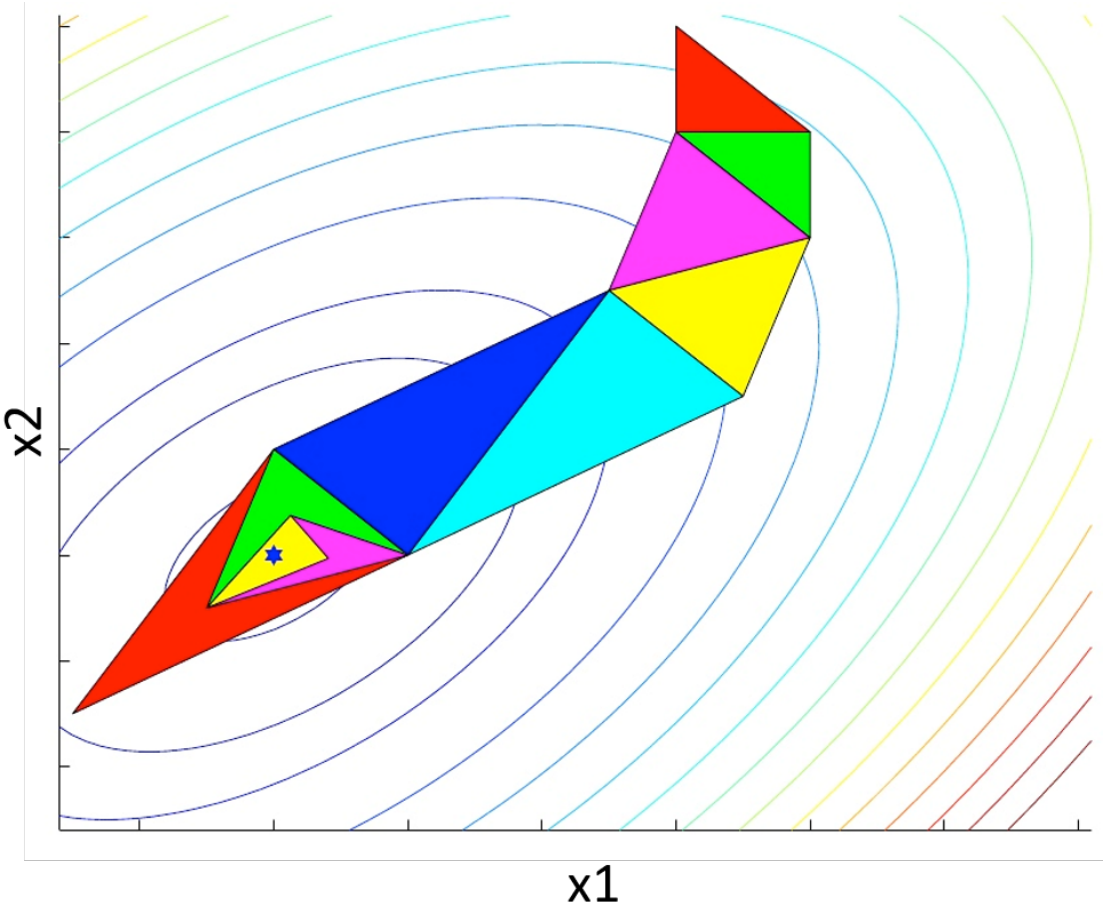
$$2\sum_i w_i \cdot d(p_i, X)$$

- If the total transportation cost that is calculated by

$$\sum_i \left[w_i \cdot d(p_i, X) + \max\left(w_1^{1/2} \cdot d(p_i, X)^{3/2}, w_i^{3/2} \cdot d(p_i, X)^{1/2} \right) \right]$$

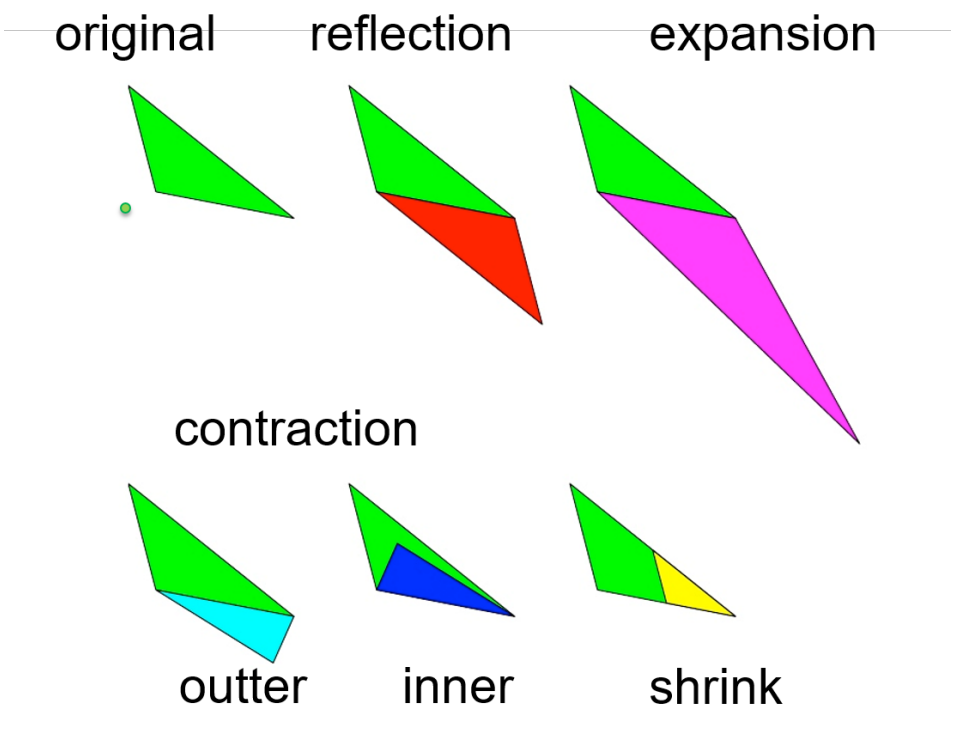
Concept of Nelder-Mead

- **Background:** smart trial-and-error proposed by Nelder, J. & Mead, R. (1965).
- **Idea:** $n+1$ simplex to improve n variable function



Nelder-Mead Algorithm

- **Initial:** random select $n+1$ points & rank by obj.value
- **Find:** $x_m = \frac{1}{n}\sum_{i=1}^{n} x_i$, then do **reflection** accepted if $f_1 < f_r < f_n$
  - If $f_r < f_1$, do **expansion**
  - If $f_r > f_n$, do **contraction**
- **Improve** do **shrink**
- **Re-Rank & Repeat:** until meet condition and **return** x_1



Concept of Newton-Raphson

- **Background:** System of Non-Linear Equitation, $f(\mathbf{x_k}) = \mathbf{0}$. $f : \mathbb{R}^n \to \mathbb{R}^m$
- **Idea:** given a random point, try to find intersection with approximation function
- **Math:**

$$f(\mathbf{x_k}) \approx t(\mathbf{x}|\mathbf{x_0}) = f(\mathbf{x_0}) + (\mathbf{x} - \mathbf{x_0})\nabla f(\mathbf{x}|\mathbf{x_0}) = 0$$

- **Algorithm**
  - approximate the function with a linear tangent $t(\mathbf{x}|\mathbf{x_0}) = f(\mathbf{x})$
  - find the intersect $t(\mathbf{x_i}|\mathbf{x_0}) = \mathbf{0}$ using linear algebra
  - use the intersect as starting point
  - repeated until $f(\mathbf{x}) \approx \mathbf{0}$

Example of Newton-Raphson

Consider the following SNLE

$$f(x,y) \equiv \begin{bmatrix} x^2 + xy - 10 \\ y + 3xy^2 - 57 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

find x and y that satisfies using Newton-Raphson with $[x = \frac{3}{2}, y = \frac{7}{2}]^T$ as initial point.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial}{\partial x} f(x,y) & \frac{\partial}{\partial y} f(x,y) \end{bmatrix} = \begin{bmatrix} 2x + y & x \\ 3y^2 & 1 + 6x \end{bmatrix}$$

```
expr1 <- expression(x^2 + x*y -10)  ;       expr2 <- expression(y + 3*x*y^2 -57)

getVal <- function(x,y){
  val1 <- eval(expr1,list(x=x,y=y)) ;       val2 <- eval(expr2,list(x=x,y=y))
  return(c(val1,val2))
}
getJacob <- function(x,y){
  col1 <- c(eval(D(expr1,"x"),list(x=x,y=y)),eval(D(expr2,"x"),list(x=x,y=y)) )
  col2 <- c(eval(D(expr1,"y"),list(x=x,y=y)),eval(D(expr2,"y"),list(x=x,y=y)) )
  return(cbind(col1,col2))
}
curPt <- c(1.5,3.5)                 ;       result <- NULL
for(i in 1:5){
  fVal  <- getVal(curPt[1],curPt[2])
  nxtPt <- curPt -   solve(getJacob(curPt[1],curPt[2])) %*% fVal
  iterPrint <- cbind(i,t(curPt),t(fVal) )
  curPt <- nxtPt                    ;       result <- rbind(result,iterPrint)
}
result
```

Reference

[CŻ13] E. Chong and S. Żak.
An introduction to optimization.
John Wiley & Sons, 2013.

[NC20] Fred Nwanganga and Mike Chapple.
Practical machine learning in R.
John Wiley & Sons, 2020.

[Pat14] Manas A Pathak.
Beginning data science with R.
Springer, 2014.

[Win22] Wayne L Winston.
Operations research: applications and algorithms.
Cengage Learning, 2022.

Modified Newton: Levenberg-Marquardt

Motivation: Guess wrong $\mathbf{x_0}$

- **Descent direction:** a direction $\mathbf{d_k}$ such that $\mathbf{d_k}^T \nabla f(\mathbf{x_k}) < 0$

  note: for exact Newton $\mathbf{d_k} = [\nabla^2 f(\mathbf{x_k})]^{-1} \nabla f(\mathbf{x_k})$
- **Reach $\nabla^2 f(\mathbf{x_k})$ non positive definite:** how to move out

LM Modification

$$\mathbf{x_{k+1}} = \mathbf{x_k} - \left[\nabla^2 f(\mathbf{x_k}) + \mu_k \mathbf{I}\right]^{-1} \nabla f(\mathbf{x_k})$$

, where $\mu_k \geq 0$ such that $\left[\nabla^2 f(\mathbf{x_k}) + \mu_k \mathbf{I}\right]$ is positive definite

Note: **Pure Newton's Method** when $\mu \to 0$, and **Pure Gradient's Method** when $\mu \to \infty$

Example: Modified Newton with Line Search

Minimize $f(\mathbf{x}) = (1 - x_1)^2 + 10(x_2 - x_1^2)^2$ starting at $\mathbf{x_0} = [0, 0]^T$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 2(1 - x_1) - 40x_1(x_2 - x_1^2) \\ 20(x_2 - x_1^2) \end{bmatrix} \quad \nabla^2 f(\mathbf{x}) = \begin{bmatrix} -2 - 40x_2 + 120x_1^2 & -40x_1 \\ -40x_1 & 20 \end{bmatrix}$$

At $\mathbf{x_0} = [0, 0]^T$ then,

$$\nabla f(\mathbf{x_0}) = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \qquad \nabla^2 f(\mathbf{x_0}) = \begin{bmatrix} -2 & 0 \\ 0 & 20 \end{bmatrix}$$

Consider $\mathbf{G_0} = \left[\nabla^2 f(\mathbf{x_k}) + \mu_k \mathbf{I}\right] = \begin{bmatrix} -2 & 0 \\ 0 & 20 \end{bmatrix} + 3\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ Then, $\mathbf{G_0}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1/23 \end{bmatrix}$

$$\text{Hence, } \mathbf{d_0} = -\mathbf{G_0}^{-1} \nabla f(\mathbf{x_0}) = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$$

Quasi-Newton Method

- **Idea:** compute approximation of hessian inverse $(\mathbf{H_k})$ at lower 'cost' and find direction $\mathbf{d_k}$ that satisfies $\left(\nabla^2 f(\mathbf{x})\right)\mathbf{d_k} = -\nabla f(\mathbf{x})$
- **Quasi-Newton:**

$$\mathbf{x_{k+1}} = \mathbf{x_k} + \gamma_k \mathbf{H_k}\,\mathbf{d_k}$$

- **More:** update approximation of hessian inverse every iteration, $\mathbf{H_{k+1}} = \mathbf{H_k} + \mathbf{H_k^u}$
- **How:** use information from $\nabla f(\mathbf{x_{k+1}})$ and $\mathbf{x_k}$ to find $\mathbf{H_k}$
- **Notation:**

$$\mathbf{x_{k+1}} - \mathbf{x_k} = \mathbf{H_k}\left(\nabla f(\mathbf{x_{k+1}}) - \nabla f(\mathbf{x_k})\right) \text{ or } \mathbf{s_k} = \mathbf{H_k}\,\mathbf{y_k}$$

- **Desired property:**

  Similar to hessian: symmetric and positives definite

  Easy to compute: use $\mathbf{s_k} = \mathbf{x_{k+1}} - \mathbf{x_k}$ and $\mathbf{y_k} = \nabla f(\mathbf{x_{k+1}}) - \nabla f(\mathbf{x_k})$

  Others: fast converge & low memory

Quasi-Newton Algorithm

Issue:	many way to update $\mathbf{H}_k^u$
Example:	SR1, BFGS, l-BFGS, Broyden
Update rule:	
SR1	$\mathbf{H_k} + \frac{(\mathbf{s_k} - \mathbf{H_k y_k})(\mathbf{s_k} - \mathbf{H_k y_k})^\mathbf{T}}{(\mathbf{s_k} - \mathbf{H_k y_k})^\mathbf{T} \mathbf{y_k}}$
BFGS	$\left(\mathbf{I} - \frac{\mathbf{y_k s_k^T}}{\mathbf{y_k^T s_k}}\right)^\mathbf{T} \mathbf{H_k} \left(\mathbf{I} - \frac{\mathbf{y_k s_k^T}}{\mathbf{y_k^T s_k}}\right) + \frac{\mathbf{s_k s_k^T}}{\mathbf{y_k^T s_k}}$

find $\nabla f(\cdot)$ from $f(\cdot)$
$k \leftarrow 0$
$\mathbf{H_k} \leftarrow \mathbf{I}$
compute $\nabla f(\mathbf{x_k})$
while $\|\nabla f(\mathbf{x_k})\| < \epsilon$ **do**
  solve for $\mathbf{d_k} = -\mathbf{H_k} \nabla f(\mathbf{x_k})$
  compute $\gamma_k = \arg\min_\gamma f(\mathbf{x_k} + \gamma\, \mathbf{d_k})$
  $\mathbf{x_{k+1}} = \mathbf{x_k} + \gamma_k\, \mathbf{d_k}$
  compute $\nabla f(\mathbf{x_{k+1}})$
  $\mathbf{s_k} = \mathbf{x_{k+1}} - \mathbf{x_k}$
  $\mathbf{y_k} = \nabla f(\mathbf{x_{k+1}}) - \nabla f(\mathbf{x_k})$
  update $\mathbf{H_{k+1}} = g(\mathbf{H_k}, \mathbf{s_k}, \mathbf{y_k})$
  $k \leftarrow k + 1$
end while
return point $\mathbf{x_k}$ and value $f(\mathbf{x_k})$