Topics on High-Dimensional Data Analytics

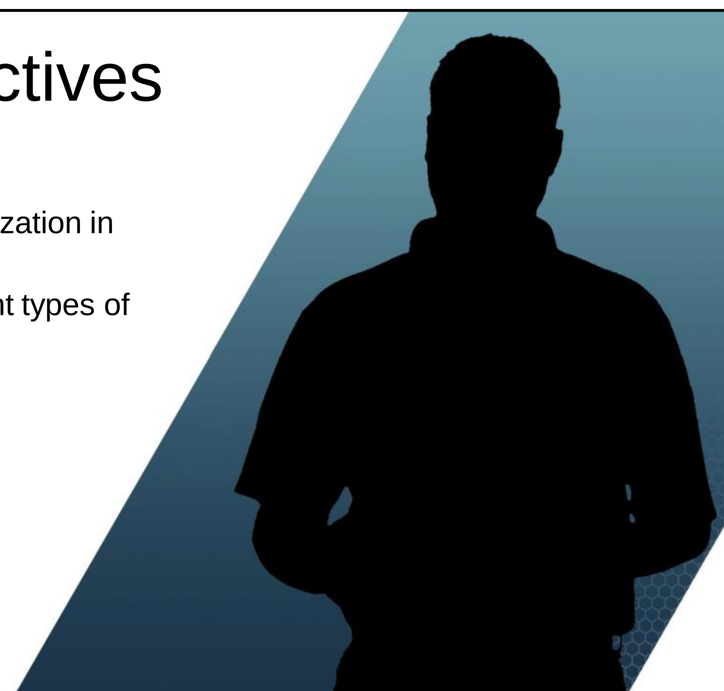
Optimization Methods

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

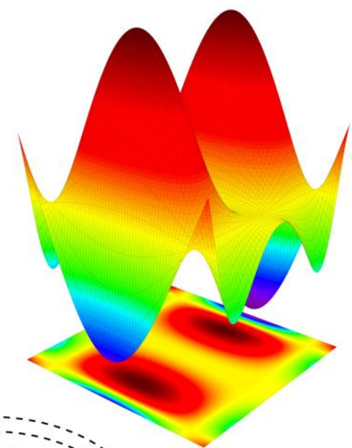
Introduction

Learning Objectives

- Define optimization
- Identify applications of optimization in analytics
- Differentiate between different types of optimization problems
- Define convex optimization



What is Optimization?

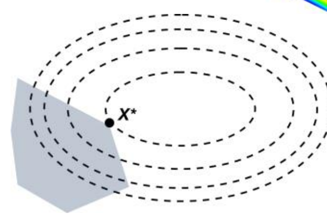


To find the minimum or maximum of an objective function given a set of constraints:

$$\arg\min_{x} f_0(x)$$

$$s.t. f_i(x) \leq 0, i = 1, \cdots, k$$

$$h_j(x) = 0, j = 1, \cdots, l$$



Optimization Applications in Analytics

- Most Statistical and Machine Learning models are (can be presented) in the form of an optimization problem. For example,
  1. Maximum Likelihood Estimation

$$\arg\max_{x} \sum_{i=1}^{n} \log p_x(\xi_i)$$

  2. Regularization/penalization

$$\min f(x) = \frac{1}{n} \sum_{i=1}^{n} l(x; (\xi_i, y_i)) + \lambda p(x)$$

Optimization Applications in Analytics

3. Regularized logistic regression

$$f(x) = -\frac{1}{N}\sum_{i=1}^{N}\log(1+\exp(y_i x^T \xi_i)) + \mu\|x\|_1$$

4. Support Vector Machine

$$\arg\min_x \frac{1}{2}\|x\|^2 + C\sum_{i=1}^{n}\max(1 - y_i(x^T \xi_i), 0)$$

5. Matrix completion

$$\min_{L,R} \sum_{(u,v)\in E} \{(L_{u.}R_{v.}^T - M_{uv})^2 + \mu_u\|L_{u.}\|_F^2 + \mu_v\|L_{v.}\|_F^2$$

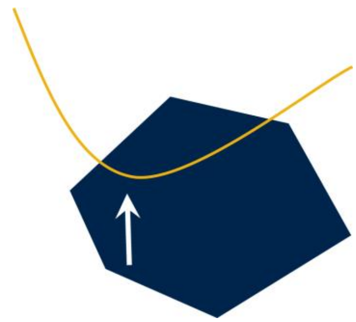
Types of Optimization

- Continuous vs. Discrete
  - **Convex** and non-convex optimization
  - Combinatorial optimization and (mixed) Integer programming
- Deterministic vs. Stochastic
  - All variables and coefficients are deterministic
  - Stochastic programming and Robust optimization
- Static vs. Dynamic
  - It does not consider the time element
  - Dynamic programming, stochastic controls and Markov decision processes

Convex Optimization

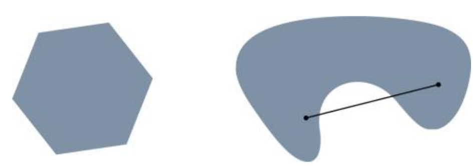
$$\text{minimize}_x f(x) \quad s.t. f_i(x) \leq 0, Ax = b$$

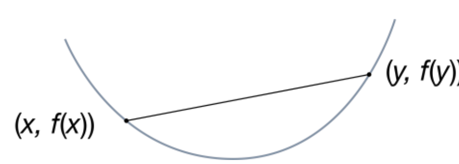
- Objective function and constraints are convex
- Well-developed, reliable and efficient algorithms
- Many of Statistical and Machine Learning models are convex.



Convex Set and Function

- **Convex set:** contains line segments between any two points in the set
  $x_1, x_2 \in C, 0 \leq \theta \leq 1 \Rightarrow \theta x_1 + (1-\theta)x_2 \in C$
- **Convex function:** f is convex if domf is a convex set and
  $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$
  for all $x, y \in \text{dom}f$, $0 \leq \theta \leq 1$
- **Strictly convex :** f is strictly convex if domf is a convex set and
  $f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta)f(y)$
  for all $x, y \in \text{dom}f$, $0 < \theta < 1$

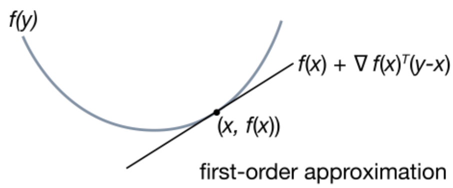




Convex Set and Function

- A differentiable function *f* is convex iff (if and only if) for all feasible *x* and *y* values

$$f(y) \geq f(x) + \nabla f(x)^T(y-x)$$



first-order approximation of *f* is global underestimator

- A twice differentiable function *f* is convex iff

$$\nabla^2 f \succeq 0$$

Optimality in Convex Problems

Proposition

Any locally optimal point of a convex problem is globally optimal

Proposition

x is optimal iff $\nabla f(x)^T(y-x) \geq 0$ for all feasible y

Outline of Optimization I Module

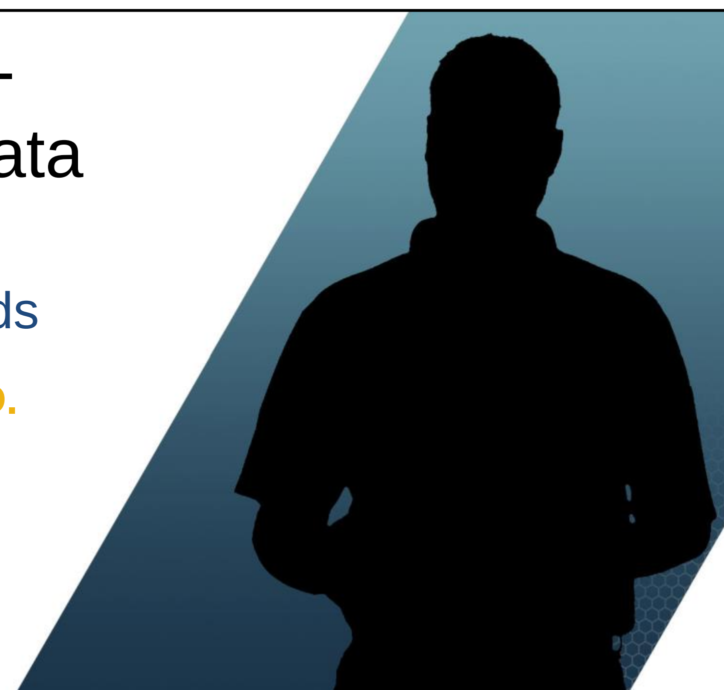
- First Order Methods
  - Gradient Descent
  - Accelerated algorithms
  - Stochastic Gradient Descent
- Second Order Methods
  - Newton method
  - Quasi-Newton method
  - Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm

Topics on High-Dimensional Data Analytics

Optimization Methods

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

First Order Methods



Learning Objectives

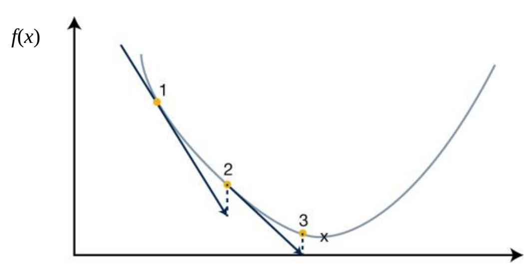
- Understand first order methods
- Use Gradient Descent
- Use Accelerated algorithms
- Use Stochastic Gradient Descent

Gradient Descent

Assume *f* is a continuous and twice differentiable function and we like to solve

$$\text{minimize}_x f(x)$$

- An intuitive approach to solve this problem is to start from an initial point and iteratively move to the direction that decreases the value of *f*.



- A natural choice for the direction is the negative gradient. i.e.,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$$

Gradient Descent

Algorithm 1 Gradient Descent

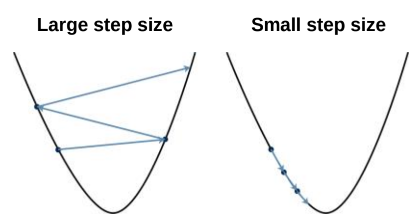
Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
while $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ **do**
  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
  $k \leftarrow k + 1$
end while
return $\mathbf{x}^{(k)}$

Determine the step size t_k:

- Exact line search: In each iteration choose the step size such that it minimizes $f(x^{(k+1)})$.

$$\mathrm{argmin}_{t \geq 0} f(\mathbf{x}^{(k)} - t \nabla f(\mathbf{x}^{(k)}))$$

- Backtracking line search: start with an initial t (e.g. unit step size) and then in iteration k, use $t/2^{(k-1)}$, or in general $t*c$, where $c \in (0,1)$.



Gradient Descent - Exact Line Search

$$\mathrm{argmin}_{t \geq 0} f(\mathbf{x}^{(k)} - t \nabla f(\mathbf{x}^{(k)}))$$

Algorithm 1 Gradient Descent

Guess $\mathbf{x}^{(0)}$, set $k \leftarrow 0$
while $||\nabla f(\mathbf{x}^{(k)})|| \geq \epsilon$ **do**
  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - t_k \nabla f(\mathbf{x}^{(k)})$
  $k \leftarrow k + 1$
end while
return $\mathbf{x}^{(k)}$

- Initialize t, denoted by $t^{(0)}$, and $i = 0$.
- While $f(x^{(k)} + t^{(i)} \nabla f(x^{(k)})) < f(x^{(k)})$

  $t^{(i)} = (t)c$, where $c \in (0,1)$.
- End
- Set $t_k = t^{(i)}$

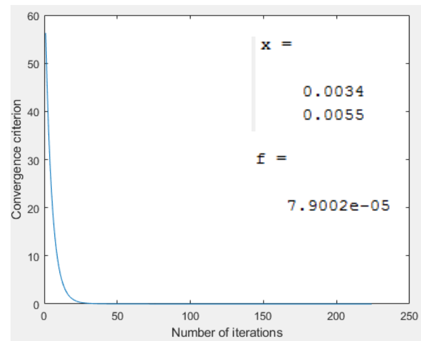
Gradient Descent - Example

Fixed step size

$$\min f(x_1, x_2) = 4x_1^2 + 2x_2^2 - 4x_1\ x_2$$

$$\nabla f = \begin{bmatrix} 8x_1 - 4x_2 \\ 4x_2 - 4x_1 \end{bmatrix}$$

```
x = normrnd(0,1,2,1);
k = 1;
tol = 0.0001;
grad = [8*x(1)-4*x(2); -4*x(1)+4*x(2)];
while grad'*grad > tol
    x = x - 0.01*grad;
    grad = [8*x(1)-4*x(2); -4*x(1)+4*x(2)];
k = k+1;
end
```

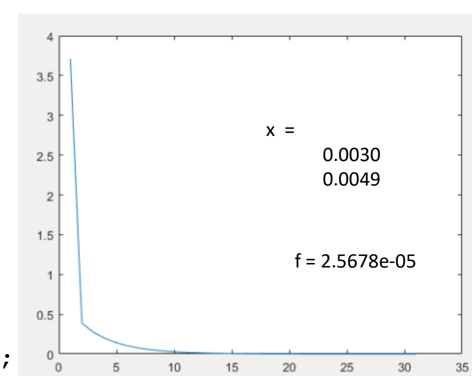


224 iterations

Gradient Descent - Example

Backtracking line search

```
x = normrnd(0,1,2,1);
k = 1;
tol = 0.0001;
grad = [8*x(1)-4*x(2); -4*x(1)+4*x(2)];
f = 4*x(1)^2+2*x(2)^2-4*x(1)*x(2);
t=0.1
while grad'*grad > tol
    x = x - (t)*grad;t=t*0.9999;
    grad = [8*x(1)-4*x(2); -4*x(1)+4*x(2)];
    f = [f; 4*x(1)^2+2*x(2)^2-4*x(1)*x(2)];
    k = k+1;
end
```



31 iterations

Convergence Rate of GD

- Convergence Rate: how quickly the sequence obtained by an algorithm converges to the desired point. For example,

$$f(x^k) - f^* \leq \epsilon_k$$

- For a given precision ϵ, what is the number of iterations required for $\min_{1 \leq t \leq k} \epsilon_t < \epsilon$? e.g., $\frac{1}{\epsilon}$

$$\lim_{k \to \infty} \frac{\epsilon_{k+1}}{\epsilon_k} = \begin{cases} 0 & \text{superlinear rate} & \epsilon_k = e^{-e^k} \\ \in (0,1) & \text{linear rate} & \epsilon_k = e^{-k} \\ 1 & \text{sublinear rate} & \epsilon_k = \frac{1}{k} \end{cases}$$

- The convergence rate for GD is $\frac{1}{k}$, which is sublinear.
- Can the convergence rate be improved?

Accelerated Gradient Descent

- Assume that f is L-Lipschitz* function. A simple version of the Nestrov's Accelerated Gradient Descent algorithm is obtained by iterating the following steps:

$$x^{k+1} = y^k - \frac{1}{L} \nabla f(y^k)$$

$$y^k = x^k + \frac{k-1}{k+2}(x^k - x^{k-1})$$

- The convergence rate for AGD is $\frac{1}{k^2}$, which is optimal.

Gradient descent	$O\left(\frac{1}{\epsilon}\right)$
Nesterov	$O\left(\sqrt{\frac{1}{\epsilon}}\right)$

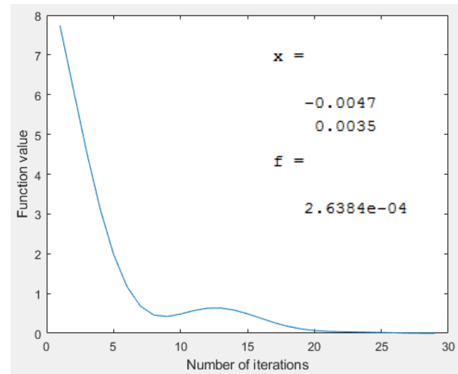
* $\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2$ for any x, y

Accelerated Gradient Descent - Example

$$\min f(x_1, x_2) = 4x_1^2 + 2x_2^2 - 4x_1\, x_2$$

$$\nabla f = \begin{bmatrix} 8x_1 - 4x_2 \\ 4x_2 - 4x_1 \end{bmatrix}$$

```
x = normrnd(0,1,2,1);
y = normrnd(0,1,2,1);
k = 1;
tol = 0.0001;
grad = [8*y(1)-4*y(2); -4*y(1)+4*y(2)];
while grad'*grad > tol
    x = y - 0.01*grad;
    y = x + (k-1)/(k+2)*(x-x0);
    grad = [8*x(1)-4*x(2); -4*x(1)+4*x(2)];
    x0 = x;
    k = k+1;
end
```



30 iterations

$30 \times 7.5 \approx 224!$

Stochastic Gradient Descent

- Consider $F(x) = \sum_{i=1}^{n} f_i(x)$ with convex $f_i(x)$'s. For example, to estimate the mean of population a natural loss function to be minimized is $F(x) = \sum_{i=1}^{n} (y_i - x)^2$.
- The GD algorithm calculates the gradient of all *n* functions, which is quite expensive for large *n*'s. Stochastic Gradient Descent, however, randomly selects one function (observation) to update the estimate of F.
- For example for K iterations:

Algorithm 1 Gradient Descent

Initialize x_1
for $k = 1$ **to** K **do**
  Compute $\nabla F(x_k) = \sum_{i=1}^{n} \nabla f_i(x_k)$
  Update $x_{k+1} \leftarrow x_k - \alpha \nabla F(x_k)$
end for
  Return x_K.

Algorithm 2 Stochastic Gradient Descent

Initialize x_1
for $k = 1$ **to** K **do**
  for $i = 1$ **to** n **do**
    Sample an observation i uniformly at random
    Update $x_{k+1} \leftarrow x_k - \alpha \nabla f_i(x_k)$
  end for
end for
Return x_K.

Stochastic Gradient Descent

- Sometimes running the algorithm over portions of the observations, known as mini-batches, performs better.

Algorithm 3 Stochastic Gradient Descent- Mini Batch

```
Initialize x_1
for k = 1 to K do
    Randomly select a batch of m observations
    for i = 1 to m do
    Sample an observation i uniformly at random
    Update x_{k+1} ← x_k − α∇f_i(x_k)
    end for
end for
Return x_K.
```

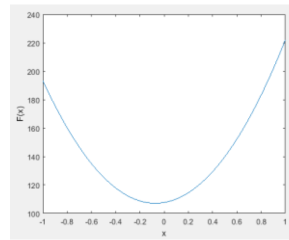
Stochastic Gradient Descent

- Comparison between GD and SGD for strongly convex functions:

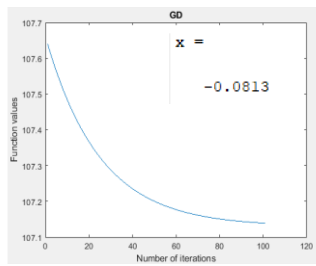
Method	# iterations	cost per iteration	total cost
GD	$\mathcal{O}(\log(1/\epsilon))$	$\mathcal{O}(n)$	$\mathcal{O}(n\log(1/\epsilon))$
SGD	$\mathcal{O}(1/\epsilon)$	$\mathcal{O}(1)$	$\mathcal{O}(1/\epsilon)$

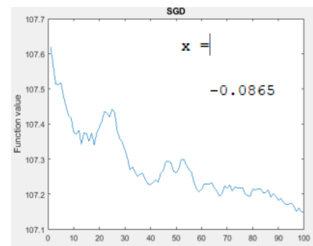
Stochastic Gradient Descent - Example

$$\min F(x) = \sum_{i=1}^{n} (y_i - x)^2.$$




```
% Data generation
K = 100;n = 100; y = normrnd(0,1,n,1);

% SGD
x = normrnd(0,1,1,1);
k = 1; tol = 0.0001;
while k <= K
    for i = 1:n
        s = randsample(n,1);
        x = x - 0.0001*(-2*(y(s)-x));
    end
k = k+1;
end
```





Stochastic Gradient Descent - Example

Fitting Linear regression

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, i = 1, \ldots, n$$

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2$$

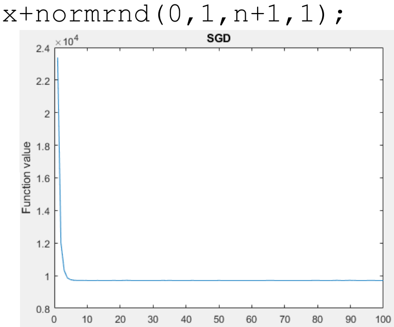
Closed form solution: $\hat{\beta} = (X^T X)^{-1} X^T y$

```
beta =

    4.9990
    2.9756
```

```
%Data generation
n = 10000; beta0 = 5; beta1 = 3;
x = [-1:(2/n):1]';
y =
beta0+beta1*x+normrnd(0,1,n+1,1);
```

Using SGD:

```
b =

    4.9871
    2.9824
```



Stochastic Gradient Descent - Example

Fitting a Logistic regression model with $\boldsymbol{\beta_0 = 10, \beta_1 = 5}$

$$p(y_i = 1) = \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}, i = 1, \dots, n$$

$$\max_{\beta_0,\beta_1} \sum_{i=1}^{n} y_i \log \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}} + (1-y_i)\log(1-\frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}})$$

Using SGD (K=100):

```
b =

    9.4236
    4.8188
```

$$\nabla f_i(\beta_0,\beta_1) = \begin{bmatrix} y_i - \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}} \\ x_i(y_i - \frac{e^{\beta_0+\beta_1 x_i}}{1+e^{\beta_0+\beta_1 x_i}}) \end{bmatrix}$$

Elapsed time:
93s

Logistic Regression Example – Stochastic Gradient Descent

```
%Data generation
n = 10000; K=100; beta0 = 10; beta1 = 5;x = [-1:(2/n):1]';
linear = beta0+beta1*x+normrnd(0,1,n+1,1); y = exp(linear)./(1+exp(linear));
% SGD
X = [ones(n+1,1) x];
b = normrnd(0,1,2,1);
while k <= K
    for i = 1:n
        s = randsample(n,1);
        grad = X(s,:)*(y(s)-exp(X(s,:)*b)/(1+exp(X(s,:)*b)));
        b = b + 1*grad';
    end
    fun = 0;
    for i = 1:n
        s1 = y(i)*log(exp(X(i,:)*b)/(1+exp(X(i,:)*b)));
        s2 = (1-y(i))*log(1-exp(X(i,:)*b)/(1+exp(X(i,:)*b)));
        fun = fun + s1 + s2;
    end
    f = [f; fun];
    k = k+1;
end
```

Topics on High-Dimensional Data Analytics

Optimization Methods

Kamran Paynabar, Ph.D.
Associate Professor
School of Industrial and Systems Engineering

Second Order Methods

Learning Objectives

- Understand second order methods
- Use Newton method
- Use Gauss-Newton Method
- Use Quasi-Newton method–(BFGS) algorithm

Second-Order Methods

- Assume *f* is a continuous and twice differentiable function and we like to solve

$$\text{minimize}_x f(x)$$

- $\nabla f(x) = \left[\frac{\partial f(x)}{\partial x}\right]$ is the gradient vector.

- $\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{bmatrix}$ is the Hessian matrix.

- Taylor expansion $\quad f(x') = f(x) + (x' - x)^\top \nabla f(x) + \frac{1}{2}(x' - x)^\top \nabla^2 f(x)\,(x' - x)$

Newton Method

- To find a root of a function using the Newton method i.e., $f(x) = 0$:

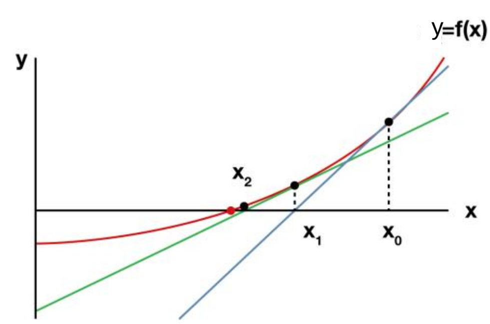
$$x^{(k)} = x^{(k-1)} - \frac{f(x)}{f'(x)}$$

- To find the minimizer of a function i.e., $f'(x) = 0$:

$$x^{(k)} = x^{(k-1)} - \frac{f'(x)}{f''(x)}$$

- For an nD vector-valued functions:

$$x^{(k)} = x^{(k-1)} - \left(\nabla^2 f(x)\right)^{-1} \nabla f(x)$$



Newton Method Algorithm (Adaptive Step Size)

Initialize $x^{(0)}, f(x), \nabla f(x), \nabla^2 f(x)$, step size $\alpha = 1$ and damping factor, $\lambda = 10^{-10}$, and tolerance ϵ.

Repeat until convergence ($\|\Omega\|_\infty < \epsilon$)

Solve $(\nabla^2 f(x) + \lambda I)\Omega = -\nabla f(x)$ for Ω,
k=k+1,

$x^{(k)} = x^{(k-1)} + \alpha\Omega$
While $f\left(x^{(k)}\right) > f\left(x^{(k-1)}\right)$

$\alpha = 0.1\,\alpha$ (*decrease step size*)
$x^{(k)} = x^{(k-1)} + \alpha\Omega$

end
$\alpha = \alpha^{0.5}$ (*increase step size*)
End

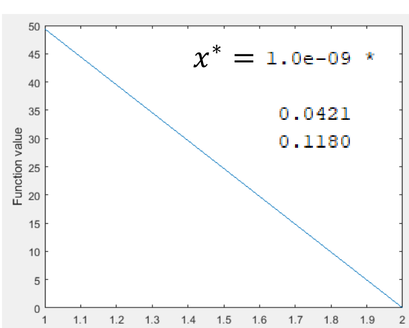
Return $x^{(k)}$

- λ makes the parabola more "steep" around current x

Newton Method - Example

$$\min f(x_1, x_2) = 4x_1^2 + 2x_2^2 - 4x_1\, x_2$$

$$\nabla f = \begin{bmatrix} 8x_1 - 4x_2 \\ 4x_2 - 4x_1 \end{bmatrix} \quad \nabla^2 f = \begin{bmatrix} 8 & -4 \\ -4 & 4 \end{bmatrix}$$



Converges in only 2 iterations!

```
x0 = normrnd(0,1,2,1);
alpha = 1; lambda = 10^(-10); tol = 10^(-10);
f0 = 4*x0(1)^2+2*x0(2)^2-4*x0(1)*x0(2);fun=f0;
g0 = [8*x0(1)-4*x0(2); -4*x0(1)+4*x0(2)];
H = [8 -4; -4 4];
O = -(H+lambda*eye(2))\g0;
k = 1;
while max(O) > tol
    x = x0 + alpha*O;
    f = 8*x(1)^2+2*x(2)^2-4*x(1)*x(2);
    while f > f0
        alpha = 0.1*alpha;
        x = x0 + alpha*O;
        f = 4*x(1)^2+2*x(2)^2-4*x(1)*x(2);
    end
    alpha = alpha^0.5;
    x0 = x;
    f0 = f;
    fun = [fun; f0];
    g0 = [8*x0(1)-4*x0(2); -4*x0(1)+4*x0(2)];
    O = -(H+lambda*eye(2))\g0;
    k = k+1;
end
```

Gauss-Newton Method

- If the function $f(x)$ is in a quadratic form, the Gauss-Newton (GN) method can be used as an approximation to Newton method.
- GN does not require the Hessian matrix computation that might sometimes be difficult to compute.
- Assume $f(x)$ is a continuous function,

$$f(x) = g^T(x)g(x),$$

where $g(x) = [g_1(x) \quad \cdots \quad g_d(x)]^T \in \mathbb{R}^d$.

- Define $\mathbf{J}_{g(x)} = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x_1} & \cdots & \frac{\partial g_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_d(x)}{\partial x_1} & \cdots & \frac{\partial g_d(x)}{\partial x_n} \end{bmatrix}$ as the Jacobian matrix.

Gauss-Newton Method

- The gradient and Hessian of $f(x)$ are

$$\nabla f(x) = 2\mathbf{J}_{g(x)}^T g(x)$$

$$\nabla^2 f(x) = 2\mathbf{J}_{g(x)}^T \mathbf{J}_{g(x)} + 2\mathbf{J}_{g(x)}^T \nabla^2 g(x)$$

- Assuming that $\nabla^2 g(x) \approx 0$, the approximate Hessian becomes

$$\nabla^2 f(x) \approx 2\mathbf{J}_{g(x)}^T \mathbf{J}_{g(x)}.$$

- Therefore, at each step, we have

$$x^{(k)} = x^{(k-1)} - \left(\mathbf{J}_{g(x)}^T \mathbf{J}_{g(x)}\right)^{-1} \mathbf{J}_{g(x)}^T g(x)$$

Gauss-Newton Algorithm
(Adaptive Step Size)

Initialize $x^{(0)}, f(x), \nabla f(x), \nabla^2 f(x)$, step size $\alpha = 1$ and damping factor, $\lambda = 10^{-10}$, and tolerance ϵ.

Repeat until convergence ($\|\Omega\|_\infty < \epsilon$)

    Solve($\mathbf{J}_{g(x)}^T \mathbf{J}_{g(x)} + \lambda I)\Omega = -\mathbf{J}_{g(x)}^T g(x)$ for Ω,
    k=k+1,
    $x^{(k)} = x^{(k-1)} + \alpha\Omega$
    While $f\left(x^{(k)}\right) > f(x^{(k-1)})$
        $\alpha = 0.1\,\alpha$ (*decrease step size*)
        $x^{(k)} = x^{(k-1)} + \alpha\Omega$
    end
    $\alpha = \alpha^{0.5}$ (*increase step size*)
end

Return $x^{(k)}$

- λ makes the parabola more "steep" around current x

Gauss-Newton Method - Example

$$\min f(\alpha, \beta) = \sum_{i=1}^{n} \left(y_i - \alpha e^{\beta x_i}\right)^2$$

$$\boldsymbol{g}(\alpha, \beta) = [y - \alpha e^{\beta x}]_{n\times 1}$$

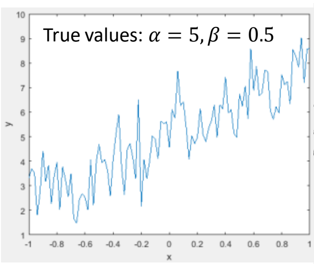
$$\mathbf{J}_{\boldsymbol{g}(\alpha,\beta)} = [-e^{\beta x} \quad -\alpha x e^{\beta x}]_{n\times 2}$$

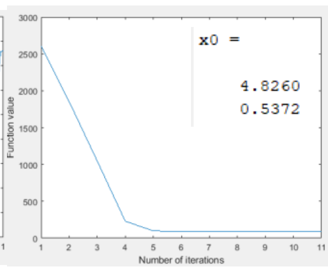
Gauss-Newton Method - Example

$$\min f(\alpha,\beta) = \sum_{i=1}^{n} \left(y_i - \alpha e^{\beta x_i}\right)^2$$

$$\boldsymbol{g}(\alpha,\beta) = [y - \alpha e^{\beta x}]_{n\times 1}$$

$$\mathbf{J}_{\boldsymbol{g}(\alpha,\beta)} = [-e^{\beta x} \quad -\alpha x e^{\beta x}]_{n\times 2}$$






```
%Data generation
n = 100; truealpha = 5; truebeta = 0.5;
x = [-1:(2/n):1]';
y = truealpha*exp(truebeta*x)+normrnd(0,1,n+1,1);
%Gauss-Newton Method
alpha = 1; lambda = 10^(-10); tol = 10^(-10); k = 1;
x0 = normrnd(0,1,2,1);
g0 = y-x0(1)*exp(x0(2)*x);
J0 = [-exp(x0(2)*x) -x0(1)*x.*exp(x0(2)*x)];
f0 = sum((y-x0(1)*exp(x0(2)*x)).^2);
O = -(J0'*J0+lambda*eye(2))\(J0'*g0);
while max(O) > tol
    x1 = x0 + alpha*O;
    f = sum((y-x1(1)*exp(x1(2)*x)).^2);
    while f > f0
        alpha = 0.1*alpha;
        x1 = x0 + alpha*O;
        f = sum((y-x1(1)*exp(x1(2)*x)).^2);
    end
    alpha = alpha^0.5;
    x0 = x1;
    g0 = y-x0(1)*exp(x0(2)*x);
    J0 = [-exp(x0(2)*x) -x0(1)*x.*exp(x0(2)*x)];
    O = -(J0'*J0+lambda*eye(2))\(J0'*g0);
    k = k+1;
end
```

Quasi-Newton Methods

- Quasi – Newton methods where the Hessian matrix cannot analytically be computed. But it can numerically approximated by previous iterations data and gradients,

  i.e.,$\left\{x^{(i)}, \nabla f\left(x^{(i)}\right)\right\}_{i=1}^{k}$.
- For 1D case, using iterations 1 and 2 data, we have

$$\nabla^2 f(x) = \frac{\left(\nabla f(x^{(2)}) - \nabla f(x^{(1)})\right)}{x^{(2)} - x^{(1)}}$$

- For nD case,

$$\nabla^2 f(x) = \frac{\left(\nabla f(x^{(2)}) - \nabla f(x^{(1)})\right)\left(\nabla f(x^{(2)}) - \nabla f(x^{(1)})\right)^T}{\left(\nabla f(x^{(2)}) - \nabla f(x^{(1)})\right)^T \left(x^{(2)} - x^{(1)}\right)}$$

$$\nabla^2 f(x)^{-1} = \frac{\left(x^{(2)} - x^{(1)}\right)\left(x^{(2)} - x^{(1)}\right)^T}{\left(x^{(2)} - x^{(1)}\right)^T \left(\nabla f(x^{(2)}) - \nabla f(x^{(1)})\right)}$$

Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Initialize $x^{(0)}, f(x), \nabla f(x)$, step size $\alpha = 1$, $H^{-1} = I$, and tolerance ϵ.

Repeat until convergence ($\|\Omega\|_\infty < \epsilon$)

Compute $\Omega = -H^{(k)^{-1}} \nabla f\left(x^{(k)}\right)$,
Solve $\min_\alpha f(x + \alpha\Omega)$ using a line search method
$\Omega = \alpha\Omega$,
$x^{(k+1)} = x^{(k)} + \Omega$
$y = \nabla f\left(x^{(k+1)}\right) - \nabla f\left(x^{(k)}\right)$

Update $H^{(k+1)^{-1}} = \left(I - \frac{y\Omega^T}{\Omega^T y}\right)^T H^{(k)^{-1}} \left(\left(I - \frac{y\Omega^T}{\Omega^T y}\right)^T\right) + \frac{\Omega\Omega^T}{\Omega^T y}$

k=k+1
end
Return $x^{(k)}$

Accelerated Gradient Descent - Example

Solve the following optimization problem using BFGS.

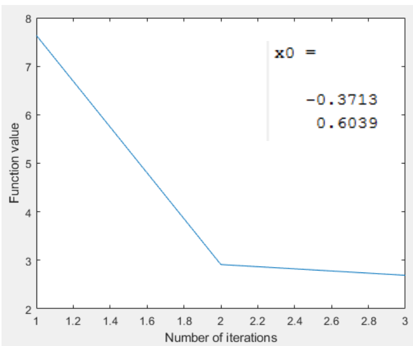
$$\min f(x_1, x_2) = e^{x_1 - 1} + e^{-x_2 + 1} + \left(x_1 - x_2\right)^2$$

$$\nabla f(x_1, x_2) = \begin{bmatrix} e^{x_1 - 1} + 2(x_1 - x_2) \\ -e^{-x_2 + 1} - 2(x_1 - x_2) \end{bmatrix}$$

Accelerated Gradient Descent - Example

$$\min f(x_1, x_2) = e^{x_1-1} + \mathrm{e}^{-x_2+1} + (x_1 - x_2)^2$$

$$\nabla f(x_1, x_2) = \begin{bmatrix} e^{x_1-1} + 2(x_1 - x_2) \\ -e^{-x_2+1} - 2(x_1 - x_2) \end{bmatrix}$$




```
x0 = normrnd(0,1,2,1);
f0 = exp(x0(1)-1)+exp(-x0(2)+1)+(x0(1)-x0(2))^2;f=f0;
g0 = [exp(x0(1)-1)+2*(x0(1)-x0(2)); -exp(x0(2)+1)-2*(x0(1)-x0(2))];
Hinv = eye(2);
O = -Hinv*g0;
tol = 10^(-10);k = 1;
while max(O) > tol
    for i = 0:1000
        alpha = i/1000;
        x = x0 + alpha*O;
        f = [f; exp(x(1)-1)+exp(-x(2)+1)+(x(1)-x(2))^2];
    end
    [fmin ind] = min(f);
    O = (ind/1000)*O;
    x = x0 + O;
    g1 = [exp(x(1)-1)+2*(x(1)-x(2)); -exp(x(2)+1)-2*(x(1)-x(2))];
    y = g1 - g0;
    Hinv = (eye(2)-(y*O')/(O'*y))'*Hinv*(eye(2)-
(y*O')/(O'*y))'+(O*O')/(O'*y);
    x0 = x;
    f0 = exp(x0(1)-1)+exp(-x0(2)+1)+(x0(1)-x0(2))^2;
    g0 = [exp(x0(1)-1)+2*(x0(1)-x0(2)); -exp(x0(2)+1)-2*(x0(1)-
x0(2))];
    O = -Hinv*g0;
    k = k+1;
end
```