Lecture Notes on Computational Optimization[1]

Rongjie Lai
Department of mathematics
Rensselaer Polytechnic Institute
lair@rpi.edu

[1]The lecture notes are recorded by Zachary Wimer and Alex Shin. Thanks Zachary and Alex for sharing the notes.

Contents

1 Fundamentals of Unconstrained Optmization **3**
- 1.1 Lecture 1 . . . 3
- 1.2 Lecture 2 . . . 4
- 1.3 Lecture 3 . . . 6

2 Gradient descent and line search **8**
- 2.1 Lecture 3 . . . 8
- 2.2 Lecture 4 . . . 9
- 2.3 Lecture 5 . . . 11
- 2.4 Lecture 6 . . . 12

3 Conjugate gradient method **15**
- 3.1 Lecture 7 . . . 15
- 3.2 Lecture 8 . . . 17
- 3.3 Lecture 9 . . . 19

4 Newton's method **21**
- 4.1 Lecture 10 . . . 21
- 4.2 Lecture 11 . . . 23

5 Barzilai-Borwein method **25**
- 5.1 lecture 11 . . . 25
- 5.2 Lecture 12 . . . 26

6 Quasi-Newton Method **29**
- 6.1 Lecture 12 . . . 29
- 6.2 Lecture 13 . . . 30

7 Linear programming **34**
- 7.1 Lecture 14 . . . 34
- 7.2 Lecture 15 . . . 37
- 7.3 Lecture 16 . . . 40
- 7.4 Lecture 17 . . . 43
- 7.5 Lecture 18 . . . 47

8 Fundamentals of non-linear programming **51**
8.1 Lecture 18 51

Chapter 1

Fundamentals of Unconstrained Optmization

1.1 Lecture 1

General form:

$$\min_x F(x) \quad s.t. \quad x \in \Omega \subseteq R^n$$

$\min F(x)$ such that $x \in \Omega \subseteq R^n$
Objective function: $F : R^n \to R^n$
Decision variables: $x = [x_1, xx_2, ...]^T$
Constrained if $\Omega \subseteq R^n$

n-Regression example:

$$\min_{a_0,a_1,a_2,...a_k} \sum_{i=1}^{n} \left(\sum_{j=0}^{k} \left(a_j x_i^j \right) - y_i \right)^2$$

Here the basis is: $\{1, x, x^2, x^3, ...x^k\}$

More generally:

$$\min_{a_0,a_1,a_2,...a_k} \sum_{i=1}^{n} \left(f(x_i) - y_i)^2 \right)$$

where $f(x) = a_1\Phi_1(x) + a_2\Phi_2(x) + ...a_k\Phi_k(x)$ where Φ is some given pattern.

Definitions:

Local minimizer:

- x^* is a local minimizer if there exists a small neighborhood, N, of x^* such that $f(x^*) \leq f(x)\ \forall x \in N$

Isolated local minimizer:

- x^* is an isolated local minimzer if x^* is a local minimizer of N and x^* is unique in N.

Global minimizer:

- $f(x^*) \leq f(x)\ \forall x$

Strict <any of the above>:

- Change $\leq$ to $<$

1.2 Lecture 2

Let F : $\mathbb{R}^n \to \mathbb{R}$
Gradient of F:

$$\nabla F(x) = \left[\frac{\partial F}{\partial x_1},\ \frac{\partial F}{\partial x_2},\ \ldots\ \frac{\partial F}{\partial x_n}\right]^T$$

Hessian (2nd deriv):

$$H_F(x) = \nabla^2 F(x) = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F}{\partial^2 n} \end{bmatrix}$$

If the function is continuous, $H_F(x)$ is symmetric.

General form:

$$F = \frac{1}{2}(x_1, x_2, \ldots x_n)^T A_{n*n}(x_1, x_2, \ldots x_n) - (x_1, x_2, \ldots x_n)^T(b_1, b_2, \ldots b_n)$$

$$F = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

Special case:
Assume symmetric A: $A^T = A$. If $F = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ and A is symmetric, then $\nabla F = Ax - b$ and $\nabla^2(x) = A$

Vector norm: l_p norm: $\|\mathbf{x}\|_p = \Big(|x_1|^p + |x_2|^p + ... + |x_n|^p\Big)^{\frac{1}{p}}$

Matrix Norm:

$$\|A\| = sup_{\|\mathbf{x}\|=1}\|A\mathbf{x}\| = sup_{\mathbf{x}\neq 0}\frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$

$\|A\|_2$ = max of singular value
$\|A\|_\infty$ = max sum of the absolute values of items in a row
$\|A\|_1$ = max sum of the absolute values of items in a column

Single value decomposition: $A = UDV^*$

- Elements of D = singular values
- D = Diagonal matrix: $(m \times n)$
- UU^* = Identity matrix: $(m \times m)$
- VV^* = Identity matrix: $(n \times n)$

You do not need to know how to perform SVD

Taylor expansion:

$$f(x+h) = f(x)+f'(x)h+\frac{1}{2}f''(x)h^2+O(h^3) = f(x)+f'(x)h+\frac{1}{2}f''(\eta)h^2+O(h^3)$$

Multi-variable version: For some $t \in (0,1)$

$$F(\mathbf{x}+\mathbf{p}) = F(\mathbf{x})+\nabla F(\mathbf{x})^T\mathbf{p}+\frac{1}{2}\mathbf{p}^T\nabla^2 F(\mathbf{x})\mathbf{p}+O(\mathbf{p}^3) = F(\mathbf{x})+\nabla F(\mathbf{x})^T\mathbf{p}+\mathbf{p}^T\nabla^2 F(\mathbf{x}+t*\mathbf{p})\mathbf{p}$$

Definitions:

- **Minimal Value**: $F(x^*)$
- **Minimizer**: x^*

Theorem: First Order Necessary Condition

If $F \in C^1$ and x^* is a local minimizer of F, then $\nabla F(x^*) = 0$

Linear Regression:

Given $(x_1, y_1), (x_2, y_2), ...(x_n, y_n)$ find $y = ax + b$ that is the best fit.
Solve

$$\min_{a,b} F(a,b) = \sum_{i=1}^{n}\Big(ax_i + b - y_i\Big)^2$$

We can also see that:

$$\min_{a,b}\sum\Big(ax_i + b - y_i\Big)^2 \iff \min_{a,b}||A(a,b)^T - \mathbf{y}||^2$$

1.3 Lecture 3

First Order Necessary Condition
If $F \in C^1$ and x^* is a local minimizer of F, then $\nabla F(x^*) = 0$

Linear Regression

Given n points, find $F(\mathbf{x}) = a\mathbf{x}+\mathbf{b}$ such that $\min_{a,b} \sum_{i=1}^{n} \left(a\mathbf{x}_i+\mathbf{b}-\mathbf{y}_i \right)^2$

Let $\mathbf{a} = (a, b)^T$, $\mathbf{b} = (y_1, y_2, ...y_n)^T$, and $A = \begin{bmatrix} \mathbf{x}_1 & 1 \\ \mathbf{x}_2 & 1 \\ \vdots & \vdots \\ \mathbf{x}_n & 1 \end{bmatrix}$.
Let $F(a, b) = \|A\mathbf{a} - \mathbf{b}\|_2^2$. We must find the min of this. Because at the minimum value the gradient must equal zero, we can derive the following:

$$\nabla F = \mathbf{0} \quad \rightarrow \quad A^T A\mathbf{a} = A^T\mathbf{b}$$

$$For\ each\ point:\ ax + b \approx y$$
$$a\mathbf{x}^T + b\mathbf{1}^T \approx \mathbf{y}^T$$
$$\begin{bmatrix} \mathbf{x}_1 & 1 \\ \mathbf{x}_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$
$$A\mathbf{a} = \mathbf{y}^T$$

Multiplying by A^T on both sides yields:

$$A^T A(a, b)^T = A^T\mathbf{y}^T$$

The equation above is called the **normal equation**.

Generalization

Let $f = a_0 + a_1x + a_2x^2 + ...a_kx^k$ and the basis $= (1, x, x^2, ...x^k)$. The problem in general is:

$$a_0 + a_1x_1 + a_2x_1 + ...a_kx_1^k \approx y_1$$
$$\vdots$$
$$a_0 + a_1x_n + a_2x_n + ...a_kx_n^k \approx y_n$$
$$\min F(a_0, a_1, ...a_k) = \sum_{i=1}^{n} \left(a_0 + a_1x_1 + ...a_ky_i^k - y_i \right)^2$$

At the minimum, $\nabla F = \mathbf{0}$ thus, expanding $A\mathbf{a} = \mathbf{b}$ we get:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & ... & x_1^k \\ 1 & x_2 & x_2^2 & ... & x_2^k \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & ... & x_n^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$F(\mathbf{a}) = ||A\mathbf{a} - \mathbf{b}||_2^2$$

$\nabla F = 0 \iff A^T A\mathbf{a} = A^T\mathbf{b}$

Positive definite: $H > 0$ (Positive definite) $\iff \neq 0, \mathbf{x}^T H\mathbf{x} > 0, \forall \mathbf{x}$

Semi-positive definite: $H(\mathbf{x}^*) \geq 0 \iff \mathbf{x}^T H(\mathbf{x}^*)\mathbf{x} \geq 0, \forall \mathbf{x}$
Note: a matrix is semi-positive definite if all of it's eigenvalues are non-negative.

Theorem 1.1 *Second Order Necessary Condition: If $F \in C^2$ and $\mathbf{x}^*$ is the minimizer of $F(x)$, then $\nabla F(\mathbf{x}^*) = 0$ and $\nabla^2 F(\mathbf{x}^*) \geq 0$. Here Hessian is semi-positive definite.*

Theorem 1.2 *Second Order Sufficient Condition: If $F \in C^2$, $\nabla F(\mathbf{x}^*) = 0$ and $\nabla^2 F(\mathbf{x}^*) > 0$ then $\mathbf{x}^*$ is a strictly local minimizer. Here the Hessian is positive definite.*

Convex function: F is called convex if for any two points, a line drawn between them is at all times strictly between the two points the line is strictly greater than the line F.

$$F(\alpha x + (1-\alpha)y) \leq \alpha F(x) + (1-\alpha)F(y), \ \forall \alpha \in [0,1], \ \forall x \neq y$$

Theorem 1.3 *If F is a convex function, then any local minimizer is a global minimizer.*
• *Lemma: $F \in C^1$ and convex, any stationary point is a global minimizer.*

Stationary point: Point $(z, F(z))$ is a stationary point if $\nabla F(z) = 0$

Chapter 2

Gradient descent and line search

2.1 Lecture 3

Gradient descent and line search: Assume F is convex. $F(x_1, x_2) = x_1^2 + x_2^2$. For a level curve (cross section) where $C = x_1^2 + x_2^2$.
Facts:

- $\nabla F = (2x_1, 2x_2) \perp$ the tangent vector of $F(x_1, x_2) = C$
- For $F(x_1, x_2) = C$: $\frac{\partial F}{\partial x_1} X_1(t) + \frac{\partial F}{\partial x_2} X_2(t) = 0$
- The inner product of the gradient of F and the tangent of the level curve is 0. In other words $< \nabla F \ , \ (X_1(t), X_2(t)) > = \ 0$

We are using the gradient direction because this will let us guarantee that $F(\mathbf{x} + \alpha\mathbf{p}) < F(\mathbf{x})$.

Un-refined equation: $F(\mathbf{x} + \alpha\mathbf{p}) = F(\mathbf{x}) + \alpha(\nabla F(\mathbf{x}))^T\mathbf{p} + O(\alpha^2))$
To decrease $F(\mathbf{x} + \alpha\mathbf{p})$ as much as possible we need to choose $\mathbf{p}$ such that $(\nabla F(\mathbf{x}))^T\mathbf{p}$ is as small as possible. Thus we choose $\mathbf{p} = -\nabla F(\mathbf{x})$. This suggests that

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$$

This is gradient descent, as long as you know how to choose α_k.

Remark: Here is an informal proof for the above.
Let $\mathbf{p} = -\nabla F(\mathbf{x}_k)$

$$\begin{aligned} F(\mathbf{x}_{k+1}) &= F(x_k - \alpha_k(-\nabla F(\mathbf{x}_k))) = F(\mathbf{x}_k) + \alpha_k(\nabla F(\mathbf{x}_k))^T\mathbf{p} + O(\alpha_k^2) \\ &= F(\mathbf{x}_k) + \alpha_k(-||\nabla F(\mathbf{x}_k)||^2) + O(\alpha_k^2) \end{aligned}$$

2.2 Lecture 4

Gradient decent formula

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$$

If $\alpha_k = \alpha$ we call it a gradient decent algorithm with a fixed step size α.

> Another way to view gradient decent: $\mathbf{x_{k+1}}$ is nothing but find an optimizer of a new objective function which his provided by a linear approximation of F at $\mathbf{x_k}$ added to the proximal term $\frac{1}{2\alpha_k}\|\mathbf{x} - \mathbf{x_k}\|_2^2$

Example: Let $F(\mathbf{x}) = x_1^3 + x_1^2 - x_1x_2 + x_2^2 + 5x_1 + 8x_2 + 4$. Solve $\min_{x_1,x_2} F(x_1, x_2)$

1. Choose α to be a small positive number. In this case, let $\alpha = 0.1$.
2. $\nabla F = \begin{bmatrix} 3x_1^2 + 2x_1 - x_2 + 5 \\ -x_1 + 2x_2 + 8 \end{bmatrix}$
3. Need to create a sequence of vectors. We choose $\mathbf{x_0}$ randomly
4. Let $\mathbf{x_0} = (0, 1)^T$. Then $\nabla F(\mathbf{x_0}) = (10, 7)^T$.
5. From there we use the formula above:
6. $$\mathbf{x_1} = \mathbf{x_0} - \alpha \begin{bmatrix} 10 \\ 7 \end{bmatrix} = \begin{bmatrix} 0 \\ -0.7 \end{bmatrix}$$
7. And repeat until $\mathbf{x_k}$ converges.

Extensions

Projected gradient method

Proximal gradient method

Accelerated gradient method

Stopping criteria

1. Gradient Condition: $\|\nabla F(\mathbf{x_{k+1}})\| < \epsilon$, where ϵ is a chosen tolerance
2. Successive objective cond.: $|F(\mathbf{x_{k+1}} - F(\mathbf{x_k})| < \epsilon$ or $\frac{|F(\mathbf{x_{k+1}} - F(\mathbf{x_k})|}{\max(1, |F(\mathbf{x_k})|)} < \epsilon$
3. Successive point difference: $\|\mathbf{x_{k+1}} - \mathbf{x_k}\| < \epsilon$ or $\frac{\|\mathbf{x_{k+1}} - \mathbf{x_k}\|}{\max(1, \|\mathbf{x_k}\|)} < \epsilon$

• We choose the maximum of 1 and the value to avoid dividing by 0

Step size α

1. α is small

   Pros: Iterations are more likely to converge
   Cons: Need more iterations

2. α is large

Pros: Less iterations
Cons: Iterations are less likely to converge (may over-shoot, may zig-zag and diverge)

How to choose α

1. Line search
2. As a fixed value if ∇F is L-Lipschitz.
3. A method called Barzilai-Borwein step size.

How to choose a fixed alpha (one method): Given $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$, $\alpha \geq 0$. Let $p_k = \nabla F(\mathbf{x}_k)$. Then we have $\mathbf{x}_{k+1} = \mathbf{x_k} - \alpha p_k$. We want that: $\|\mathbf{x_{k+1}} - \mathbf{x}^*\|_2^2 = \|\mathbf{x_k} - \alpha p_k - \mathbf{x}^*\|_2^2$ should be less than $\|\mathbf{x_k} - \mathbf{x}^*\|_2^2$.
From this we can derive that:

$$\frac{\alpha}{2}\|p_k\|^2 \quad \leq \quad < p_k, \mathbf{x_k} - \mathbf{x}^* >$$

L-Lipschitz differentiable: A function F is called L-Lipschitz differentiable if $F \in C^1$ and

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|, \ \forall \mathbf{x}, \mathbf{y}$$

Theorem 2.1 (Baillon-Haddad theorem) *: If $F \in C^1$ and is convex, then F is L-Lipschitz differentiable if and only if*

$$\|\nabla F(x) - F(y)\|^2 \leq L < \nabla F(x) - \nabla F(y), x - y >$$

Theorem 2.2 *For the fixed-step size gradient descent algorithm, where A is positive definite, applying to*

$$F(x) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

For any x_0, the sequence $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$ converges to $\mathbf{x}^*$ is $0 < \alpha \leq \frac{2}{\lambda_{max}(A)}$

Example:

$$F(\mathbf{x}) = \begin{bmatrix}\mathbf{x_1} & \mathbf{x_2}\end{bmatrix}\begin{bmatrix}1 & 0\\ 0 & 2\end{bmatrix}\mathbf{x} - \begin{bmatrix}1 & 3\end{bmatrix}\mathbf{x}$$

Thus we can derive

$$\|\nabla F(x) - \nabla F(y)\| \leq 2\|x - y\|$$

Thus $0 < \alpha \leq 2/L = 2/2 = 1$

Quadratic programming: Let A be symmettric and positive definite $(x^T Ax > 0, \forall x \neq 0)$. Then

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} \;\rightarrow\; \nabla F(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$$

$$\|\nabla F(\mathbf{x}) - \nabla F(\mathbf{y})\| = \|A\mathbf{x} - \mathbf{b} - (A\mathbf{y} - \mathbf{b})\| = \|A(\mathbf{x} - \mathbf{y})\| \leq \|A\| * \|\mathbf{x} - \mathbf{y}\|$$

Note, when A is a symmetric positive definite matrix, $||A||_2 = \max \lambda$

2.3 Lecture 5

Steepest descent approach: An adaptive way to chose α_k (not a constant). Ideally choose $\alpha_k = argmin_{\alpha \geq 0} F(\mathbf{x}_k - \alpha \mathbf{p}_k)$. Note that this itself is an optimization problem, but is only one dimensional. α_k will have a closed-form solution for quadratic programming: $F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x},\ A > 0$.

Theorem 2.3 *If $\{\mathbf{x}_k\}$ is a steepest descent sequence, then $(\mathbf{x}_{k+1} - \mathbf{x}_k) \perp (\mathbf{x}_{k+2} - \mathbf{x}_{k-1})$. Note, a steepest descent sequence is the following:*

$$\{\mathbf{x}_k\} = \begin{cases} \mathbf{x}_0 \\ \mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k) \text{ and } \alpha_k = argmin_{\alpha \geq 0} F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k)) \end{cases}$$

Proof: First let A be the inner product of the following two equations: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$ and $\mathbf{x}_{k+2} = \mathbf{x}_{k+1} - \alpha_k \nabla F(\mathbf{x}_{k+1})$. Then let $\phi_k(\alpha) = F(\mathbf{x} - \alpha \nabla F(\mathbf{x}_k))$. Since we want $\alpha_k = argmin_{\alpha \geq 0} \phi_k(\alpha)$, set the derivative equal to zero. From this we get that the foll=wing equalities $< \nabla F(\mathbf{x}_{k+1}),\ \nabla F(\mathbf{x}_k) >= 0$ and $< \nabla F(\mathbf{x}_{k+1}),\ \nabla F(\mathbf{x}_k) >= A$. Thus $A = 0$.

Theorem 2.4 *Given a steepest descent sequence $\{\mathbf{x}_k\}$: if $\nabla F(\mathbf{x}_{k+1}) \neq 0 \rightarrow F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$. In other words, it each step is always better until you hit the minimizer.*

Proof:

$$\phi_k(\alpha) = F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k))$$

$$\alpha_k = argmin_{\alpha \geq 0} \phi_k(\alpha) \text{ implies that } \forall \alpha \geq 0,\ \phi_k(\alpha_k) \leq \phi_k(\alpha)$$

$$\phi_k'(0) \;=\; < \nabla F(\mathbf{x}_k) - 0,\ -\nabla F(\mathbf{x}_k) >$$

$$\phi_k'(0) \;=\; -||\nabla F(\mathbf{x}_k)||_2^2 \;<\; 0$$

Since $\phi_k'(0) < 0$, and since F is continuous, $\exists \beta \geq 0$ such that $\forall \alpha \in [0, \beta)$, $\phi_k'(\alpha) \leq 0$. It follows that $\phi_k(0) > \phi_k(\alpha),\ \forall \alpha \in [0, \beta)$. Since α_k minimizes ϕ_k, it follows that

$$F(\mathbf{x}_{k+1}) = \phi_k(\alpha_k) \leq \phi_k(\beta) < \phi_k(0) = F(\mathbf{x}_k)$$

Deriving Steepest Gradient Descent for quadratic programming: Let A be a symmetric positive definite matrix. Solve $F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$. We can do this by solving $\nabla F(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$. The **residual** is $\mathbf{r} = \nabla F(\mathbf{x}_k) = A\mathbf{x}_k - \mathbf{b}$. From

$$\alpha_k = argmin_\alpha F(\mathbf{x}_k - \alpha\nabla F(\mathbf{x}_k)) = argmin_\alpha F(\mathbf{x}_k - \alpha\mathbf{r})$$

We can then apply our definition of F to expand this. Thus we get

$$\alpha_k = argmin_\alpha \frac{1}{2}(\mathbf{x}_k - \alpha\mathbf{r}_k)^T A(\mathbf{x}_k - \alpha\mathbf{r}_k) - \mathbf{b}^T(\mathbf{x}_k - \alpha\mathbf{r}_k)$$

Label this as $\phi_k(\alpha)$. Since $\phi_k'(\alpha) = 0$ we get that

$$\phi_k'(\alpha) = (\mathbf{x}_k - \alpha\mathbf{r}_k)^T A(-\mathbf{r}_k) + \mathbf{b}^T(\mathbf{r}_k) = 0$$

We can solve this equation for

$$\alpha = \frac{\|\mathbf{r}_k\|_2^2}{\mathbf{r}_k^T A\mathbf{r}_k}$$

Here $< \mathbf{x},\ \mathbf{y} >_A = \mathbf{x}^T A\mathbf{y}$, and $\|\mathbf{x}\|_A^2 = < \mathbf{x},\ \mathbf{x} >_A$. Thus $\alpha = \frac{\|\mathbf{r}_k\|_2^2}{\|\mathbf{r}_k\|_A^2}$

Steepest Gradient Descent for quadratic programming:

1. Pick an $\mathbf{x}_0$
2. $\mathbf{r}_k = A\mathbf{x}_k - \mathbf{b}$
3. $\alpha_k = \frac{\|\mathbf{r}_k\|_2^2}{\|\mathbf{r}_k\|_A^2}$
4. $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{r}_k$

Example: Look as his notes, they are pretty clear.

Condition number: If A is symmetric and positive definite then

$$Cond(A) = \frac{\lambda_{max}}{\lambda_{min}}$$

2.4 Lecture 6

If Eigenvector guess is good

If you are lucky enough to get a point $\mathbf{x}_k$ such that there will be exactly one step until argmin is found, then $\mathbf{x}_k - \mathbf{x}^*$ is an eigenvector of A. Demonstration of consistency: $A(\mathbf{x}_k - \mathbf{x}^*) = \lambda(\mathbf{x}_k - \mathbf{x}^*)$

$$\mathbf{r}_k = A\mathbf{x}_k - \mathbf{b} = A\mathbf{x}_k - A\mathbf{x}^* = A(\mathbf{x}_k - \mathbf{x}^*) = \lambda(\mathbf{x}_k - \mathbf{x}^*)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{r}_k$$

$$\alpha_k = \frac{\|\mathbf{r}_k\|_2^2}{\|\mathbf{r}_k\|_A^2} = \frac{\|\lambda(\mathbf{x}_k - \mathbf{x}^*)\|_2^2}{\|\mathbf{r}_k\|_A^2}$$

This ultimately reduces to $\alpha_k = \frac{1}{\lambda}$. Thus

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{\lambda}\mathbf{r}_k = \mathbf{x}_k - \frac{1}{\lambda}(A\mathbf{x}_k - \mathbf{b})$$

Which reduces to $\mathbf{x}_k - \mathbf{x}_k + \mathbf{x}^* = \mathbf{x}^*$.
Thus, the claim has been proven.

Note: If $A = \{\lambda\}_{i=0}^n * I$, it follows that one step solves the problem.

General case (A is symmetric positive definite):

Define $\|\mathbf{e}\|_A = \sqrt{e^T A e}$. Define $\kappa = \frac{\lambda_{max(A)}}{\lambda_{\min(A)}}$

Then $\|\mathbf{x}_k - \mathbf{x}^*\|_A \leq (\frac{\kappa-1}{\kappa+1})^k \|\mathbf{x}_0 - \mathbf{x}^*\|_A$
(Only the power is k, the rest are κ)

Since $0 < \frac{\kappa-1}{\kappa+1} < 1$, the sequence $\{\mathbf{x}_k\}$ will converge.

Theorem 2.5 *For a quadratic problem if* $\min_{\mathbf{x}} F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ *and A is symmetric positive definite then the steepest descent algorithm converges for any* $\mathbf{x}_0$.

Order of convergence:

We say that the sequence $\{x_k\}$ converges to $\mathbf{x}^*$ with order of convergence p if for some constant C, the following holds:

$$\lim_{k\to\infty} \frac{\|\mathbf{x}_{k+1} - \mathbf{x}^*\|}{\|\mathbf{x}_k - \mathbf{x}^*\|^p} = C$$

Example: $\frac{1}{k}$ and α^k, $\forall \alpha \in (0,1)$, both converge linearly.
But $x_k = \alpha^{(2^k)}$ $\forall \alpha \in (0,1)$ converges quadratically.
Proof: $\lim_{k\to\infty} \frac{\|\mathbf{x}_{k+1}-0\|}{\|\mathbf{x}_k-0\|^2} = \lim_{k\to\infty} \frac{\alpha^{(2^{k+1})}}{\left(\alpha^{(2^k)}\right)^2} = 1$

Order of Steepest descent algorithm for quadratic programming:

The order of convergence is 1 in general, as can be see from

$$\|\mathbf{e}_k\|_A \leq \left(\frac{\kappa-1}{\kappa+1}\right)^k \|\mathbf{e}_0\|_A$$

How to choose α:

For a general optimization problem, how do you choose step size? I.e. solve $\alpha_k = argmin_{\alpha \geq 0} F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k))$
Let $\phi_k(\alpha) = F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k))$ and assume F and that and ϕ are differentiable and convex.

- Bisection method:
  Start with [a, b]
  1. If $\phi_k'(m) > 0$, recurse with [a, m]
  2. If $\phi_k'(m) < 0$, recurse with [m, b]
  3. If $\phi_k'(m) = 0$, stop
- Newton's method:
  Solve $\phi_k'(\alpha) = 0$ (First order necessary condition)
  $\alpha_{i+1} = \alpha_i - \frac{\phi_k'(\alpha_i)}{\phi_k''(\alpha_i)}$ He derived this very clearly in his notes.
- Secant method:
  If F and $\phi_k(\alpha)$ are not second order differentiable:

$$\phi_k''(\alpha_i) \approx \frac{\phi'(\alpha_i) - \phi'(\alpha_{i-1})}{\alpha_i - \alpha_{i-1}}$$

  Thus we have

$$\alpha_{i+1} = \alpha_i - \frac{\phi'(\alpha_I)}{\frac{\phi'(\alpha_i) - \phi'(\alpha_{i-1})}{\alpha_i - \alpha_{i-1}}} = \alpha_i - \frac{\phi'(\alpha_i)\big(\alpha_i - \alpha_{i-1}\big)}{\phi'(\alpha_i) - \phi'(\alpha_{i-1})}$$

Line search using newtons method:

Let $\phi_k(\alpha) = F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k))$.

- Guess β_0 (our guess for α_k).
- Run newton's method on $\phi_k(\alpha)$ to find β_j, for some high number j.
- Define $\alpha_k = \beta_j$
- Then run **one** gradient decent step
- Repeat

Wolfe conditions:

1. Armijo Condition:
$\phi_k(\alpha_k) \leq \phi_k(0) + C_1 \alpha_k \phi_k'(0)$

2. Curvature Condition:
$\phi_k'(\alpha_k) \geq C_2 \phi_k'(0)$
$\nabla F(x^{(k)} + \alpha_k p_k)^T p_k \geq C_2 \nabla F(x^{(k)})^T p^{(k)}$

If your line search function satisfies the two conditions above, your line search will converge.

Chapter 3

Conjugate gradient method

3.1 Lecture 7

Review:

Let $A \in \mathbb{R}^{nxn}$ where A is symmetric and positive definite (represented $A > 0$). We wish to find $\min F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T b$, with a First Order Necessary Condition of $A\mathbf{x}^* - \mathbf{b} = 0$. To do this, we can utilize Steepest Gradient Descent.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$$

$$\nabla F(\mathbf{x}_k) = \mathbf{r}_k = A\mathbf{x}_k - \mathbf{b}$$

$$\alpha_k = argmin_\alpha F(\mathbf{x}_k - \alpha \mathbf{r}_k)$$

Since we are solving a quadratic programming problem:

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T A \mathbf{r}_k}$$

Convergence: Since $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$, for quadratic programming, the Steepest Gradient Descent error convergence is as follows:

$$\|\mathbf{e}_i\|_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i ||\mathbf{e}_0||_A$$

Here $\kappa = \frac{\lambda_{max(A)}}{\lambda_{min(A)}}$ (Condition number) This is faster than normal gradient descent, but it still takes an arbitrary number of iterations to get 'close' to the solution. How can we solve this faster?

Conjugate gradient method

For a quadratic programming problem where A is a symmetric positive definite matrix such that $A \in \mathbb{R}^{nxn}$, we are guaranteed that the Conjugate Gradient method can find the solution in at most n steps!

Let $\{\mathbf{d}_k\}_{k=1}^{n-1}$ be a set of orthogonal vectors.
In other words, $< \mathbf{d}_i, \mathbf{d}_j >= 0,\ i \neq j$. Let our 'guess update' step be

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{d}_k$$

We shall determine what α_k is below.

$$\mathbf{e}_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}^* = \mathbf{x}_k - \alpha_k \mathbf{d}_k - \mathbf{x}^* = \mathbf{e}_k - \alpha_k \mathbf{d}_k$$

If we require that $\mathbf{e}_{k+1} \perp \mathbf{d}_0, \mathbf{d}_1, ...\mathbf{d}_k$ we get that

$$\mathbf{e}_{k+1}^T \mathbf{d}_k = 0 \iff (\mathbf{e}_k - \alpha_k \mathbf{d}_k)^T \mathbf{d}_k = 0$$

Solving for α_k, we get that

$$\alpha_k = \frac{\mathbf{d}_k^T \mathbf{e}_k}{\mathbf{d}_k^T \mathbf{d}_k^T}$$

Why require $\mathbf{e}_{k+1} \perp \mathbf{d}_0, \mathbf{d}_1, ...\ \mathbf{d}_k$? Well, if $\mathbf{e}_{k+1} \perp \mathbf{d}_0, \mathbf{d}_1, ...\ \mathbf{d}_k$, then what if $k = n-1$? In that case $e \perp \mathbf{d}_0, \mathbf{d}_1, ...\ \mathbf{d}_{n-1}$. But since $\{\mathbf{d}_k\}_{k=0}^{n-1}$ is a set of n orthogonal vectors, then in an nxn space, no non-zero vector can be orthogonal to $\mathbf{d}_k,\ \forall k \in [0, n-1]$. In other words, after n iterations, $\mathbf{e} = 0$!

Important notes:

1. Be mindful that $\{\mathbf{d}_k\}_{k=0}^{n-1}$ is set of n vectors with indexing starting at $k = 0$.
2. This works because of a simple lemma: Orthogonal vectors are linearly independent.

Current Problems:

1. We need to pre-compute $\mathbf{d}_k$ (Not too hard)
2. α_k requires knowledge of $\mathbf{e}_k$ (This is a big problem)

Let's tackle the biggest first: knowledge of $\mathbf{e}_k$:
Consider: $A\mathbf{e}_k = A(\mathbf{x}_k - \mathbf{x}^*) = A\mathbf{x}_k - \mathbf{b} = \mathbf{r}_k$. So, what if we substituted our normal dot product for an A-induced dot product? In other words

$$< \mathbf{x}, \mathbf{y} >_A = \mathbf{x}^T A \mathbf{y}$$

If we do so, then

$$\alpha_k = \frac{< \mathbf{d}_k, \mathbf{e}_k >_A}{< \mathbf{d}_k, \mathbf{d}_k >_A} = \frac{\mathbf{d}_k^T A \mathbf{e}_k}{\mathbf{d}_k^T A \mathbf{d}_k} = \frac{\mathbf{d}_k^T \mathbf{r}_k}{\mathbf{d}_k^T A \mathbf{d}_k}$$

Now we need not know $\mathbf{e}_k$ as we can calculate $\mathbf{r}_k$!

Theorem: $\mathbf{e}_{k+1} \perp_A \mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_k$
i.e. - $\langle \mathbf{e}_{k+1}, \mathbf{d}_i \rangle_A = e_{k+1}^T A \mathbf{d}_i = 0$, $i = 0, ..., k$

Corollary: If $\mathbf{e}_n \perp_A \mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_{n-1}$ and $d_0, ..., d_n$ forms a basis of $\mathbb{R}^n$ then $\mathbf{e}_n = 0$, which means the algorithm finishes in at most n steps

Remarks:

- $\alpha_k = arg\min_\alpha F(\mathbf{x}_k - \alpha \mathbf{d}_k)$
- $F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k - \alpha \mathbf{d}_k) = \min_\alpha F(\mathbf{x}_k - \alpha \mathbf{d}_k)$
- $F(\mathbf{x}_{k+1}) = \min_{\alpha_0, \alpha_1, ..., \alpha_k} F(\mathbf{x}_0 - \sum_{i=0}^k \alpha_i \mathbf{d}_i)$

3.2 Lecture 8

Refining the Conjugate Gradient Method:
For $\min \frac{1}{2}\mathbf{x}^T A x - \mathbf{b}^T x$, $x \in \mathbb{R}^n$, we can compute $\mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_{n-1} \in \mathbb{R}^n$ such that $\langle \mathbf{d}_i, \mathbf{d}_j \rangle_A = \mathbf{d}_i^T A \mathbf{d}_j = \mathbf{0}$, if $i \neq j$ We can now redefine $\mathbf{x}_k$ and derive $\mathbf{e}_{k+1}$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$$

$$\mathbf{e}_{k+1} = \mathbf{e}_k + \alpha_k \mathbf{d}_k$$

Assuming $\mathbf{e}_{k+1} \perp_A \mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_k$. Since $0 = \mathbf{e}_{k+1}^T A \mathbf{d}_k = \mathbf{e}_k^T A \mathbf{d}_k + \alpha_k \mathbf{d}_k^T A \mathbf{d}_k$ then it follows that:

$$\begin{aligned} \alpha_k &= \frac{-\mathbf{d}_k^T A \mathbf{e}_k}{\mathbf{d}_k^T A \mathbf{d}_k} \\ &= \frac{-\mathbf{d}_k^T (A\mathbf{x}^* - \mathbf{b})}{\mathbf{d}_k^T A \mathbf{d}_k} \\ &= \frac{-\mathbf{d}_k^T \mathbf{r}_k}{\mathbf{d}_k^T A \mathbf{d}_k} \end{aligned}$$

Why would we want to do this?

Because there are n vectors in the sequence of $\{\mathbf{d}_k\}$ and we are in the n dimension; thus, if $\mathbf{e}_{k+1} \perp_A \{\mathbf{d}_k\}$, then it must be the 0 vector!

How to produce $\{\mathbf{d}_k\}_{k=0}^{n-1}$?

Using Gram-Schmidt Orthogonalization, we are able to produce a sequence of orthogonal vectors using a sequence of linearly independent vectors $\{\mathbf{u}_k\}_{k=0}^{n-1}$.

$$\mathbf{d}_0 = \mathbf{u}_0, \quad \mathbf{d}_k = \mathbf{u}_k - \sum_{i=0}^{k-1} Proj_{\mathbf{d}_i}(\mathbf{u}_k), \ where \ Proj_{\mathbf{x}}(\mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\mathbf{x}^T\mathbf{x}}\mathbf{x}$$

We can use a modified version of the Gram-Schmidt Orthogonalization known as Gram-Schmidt A-Orthogonalization. We simply replace

$$Proj_{\mathbf{d}}(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{d}}{\mathbf{d}^T\mathbf{d}}\mathbf{d} = \frac{\langle \mathbf{x}, \mathbf{d}\rangle}{\langle \mathbf{d}, \mathbf{d}\rangle}\mathbf{d}$$

with

$$Proj_{\mathbf{d}}^{A}(x) = \frac{\mathbf{x}^T A\mathbf{d}}{\mathbf{d}^T A\mathbf{d}}\mathbf{d} = \frac{\langle \mathbf{x}, \mathbf{d}\rangle_A}{\langle \mathbf{d}, \mathbf{d}\rangle_A}\mathbf{d}$$

A better way to find $\mathbf{d}_k$:

To be memory/runtime efficient we would ideally want to be able to calculate $\mathbf{d}_k$ only using $\mathbf{d}_{k-1}$. Given $\mathbf{d}_0, \mathbf{d}_1, ..., \mathbf{d}_{n-1}$, where $\langle \mathbf{d}_i, \mathbf{d}_j\rangle_A = 0$ for $i \neq j$, we know that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k, \quad \alpha_k = \frac{-\mathbf{d}_k^T\mathbf{r}_k}{\mathbf{d}_k^T A\mathbf{d}_k}, \quad \mathbf{r}_k = A\mathbf{x}_k - \mathbf{b}$$

If construction of $\mathbf{d}_k$ depends on all the previous vectors before it, how can we make a smart choice of $\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{n-1}$ such that we can calculate $\mathbf{d}_k$ simply with $\mathbf{d}_{k-1}$? We can achieve this by selecting $\mathbf{u}_k$ such that:

$$\mathbf{u}_k = -\nabla F(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k = -\mathbf{r}_k$$

Let $\mathbf{g}_k = -\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$. Thus, $\mathbf{g}_k = \mathbf{u}_k$. Therefore:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k, \quad \alpha_k = \frac{-\mathbf{d}_k^T\mathbf{r}_{(k)}}{\mathbf{d}_k^T A\mathbf{d}_k} = \frac{\mathbf{d}_k^T\mathbf{g}_k}{\mathbf{d}_k^T A\mathbf{d}_k}$$

From this it follows that:

$$\begin{aligned}
\mathbf{b} - A\mathbf{x}_{k+1} &= \mathbf{b} - A\mathbf{x}_k - \alpha_k A\mathbf{d}_k \\
\mathbf{g}_{k+1} &= \mathbf{g}_k - \alpha_k A\mathbf{d}_k \\
\mathbf{d}_{k+1} &= \mathbf{g}_{k+1} + \beta_k\mathbf{d}_k \\
\beta_k &= \frac{-\mathbf{g}_{(k+1)}^T A\mathbf{d}_k}{\mathbf{d}_k^T A\mathbf{d}_k}
\end{aligned}$$

Thus, we have have found a way to construct $\mathbf{d}_{k+1}$ such that it only requires $\mathbf{d}_k$!

3.3 Lecture 9

Conjugate Gradient Method:

In order to solve $\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T a\mathbf{x} - \mathbf{b}^T\mathbf{x} = F(\mathbf{x})$, A is symettric positive definite:

Initialize:

$$\mathbf{x}_0 = 0,\ \mathbf{g}_0 = b,\ \mathbf{d}_0 = \mathbf{g}_0$$

Repeat: for $k = 0, 1, 2, ..., n-1$

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{d}_k^T A \mathbf{d}_k}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{g}_{k+1} = \mathbf{g}_k - \alpha_k A \mathbf{d}_k$$

$$\beta_k = \frac{-\mathbf{g}_{k+1}^T A \mathbf{d}_k}{\mathbf{d}_k^T A \mathbf{d}_k} = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}$$

$$\mathbf{d}_{k+1} = \mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$$

Summary:

So far, we have gone over three methods to solve a quadratic programming problem: $F(\mathbf{x}) = \min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T a\mathbf{x} - \mathbf{b}^T\mathbf{x}$

1. Gradient Descent Method with a fixed step alpha:

   $$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k)$$

   No need to restrict to quadratic programming. But *if* it is then, $\nabla F(\mathbf{x}_k) = A\mathbf{x}_k - \mathbf{b}$

2. Steepest Gradient Descent Method:

   $$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$$

   $$\alpha_k = argmin_\alpha F(\mathbf{x}_k - \alpha \nabla F(\mathbf{x}_k))$$

   Use a numerical method to find α_k, such as newtons, secant, or bisection. No need to restrict to quadratic programming. But *if* it is, then $\alpha_k = \frac{r_k^T r_k}{r_k^T A r_k},\ r_k = A\mathbf{x}_k - \mathbf{b}$

3. Conjugate Gradient method:
   Current version designed for quadratic programming. Will stop at most n-step for a problem with A is nxn.

Can we do better?

Gradient descent method only uses 0th order, $F(\mathbf{x})$, and 1st order, $\nabla F(\mathbf{x})$, information. Thus, generally, it's order of convergence is 1.

What about a method that uses 0th, 1st, and 2nd order information, and that general has 2nd order convergence?

Chapter 4

Newton's method

4.1 Lecture 10

Matlab note: To solve $A\mathbf{x} = \mathbf{b}$ do **not** use $\mathbf{x} = A^{-1}\mathbf{b}$. Instead use $\mathbf{x} = A\backslash B$ because it is more computationally efficient

Convergence of Newton's method:
Will converge quadratically is $\mathbf{x}_0$ is close enough to $\mathbf{x}^*$
Ie the order of convergence is 2

Derivation of Newton's Method: Assuming $F \in \mathbb{C}^2$

Consider $\min_x F(x) = x^2 + \frac{1}{2}x^3$. Ideally, we would like to find d such that $x_{k+1} = x_k + d \approx x^*$ Via the Taylor expansion of $F(x_k + d)$ we have: $F(x_k + d) = F(x_k) + dF'(x_k) + \frac{1}{2}d^2F''(x_k) + O(d^3) \approx F(x^*)$.

We would like to solve for d such that $F(x^*) \approx F(x_k + d)$. Thus we want to solve $F(x^*) \approx F(x_k + d) \approx F(x_k) + dF'(x_k) + \frac{1}{2}d^2F''(x_k)$ for d. Taking the derivative on each side, we get $0 = F'(x_k) + dF''(x_k)$. From there we get that $d = -\frac{F'(x_k)}{F''(x_k)}$. Thus, we have derived newton's formula for one dimension: $x_{k+1} = x_k + d = x_k - \frac{F'(x_k)}{F''(x_k)}$. We can generalize this for n-dimensions.

Derivation for n dimensions: Since the Hessian of F is symmetric: The Taylor series for higher dimensions is

$$F(\mathbf{x}_k + \mathbf{d}) \approx F(\mathbf{x}_k) + \mathbf{d}^T\nabla F(\mathbf{x}_k) + \frac{1}{2}\mathbf{d}^T\nabla^2 F(\mathbf{x}_k)\mathbf{d} + \dots$$

Using this equation and applying the same method as we did above, we get that $\mathbf{0} = \mathbf{0} + \nabla F(\mathbf{x}_k) + \nabla^2 F(\mathbf{x}_k)\mathbf{d}$. Thus

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\nabla^2 F(\mathbf{x}_k)\right)^{-1}\nabla F(\mathbf{x}_k)$$

Newton's method Geometrically: He approximates $F(\mathbf{x})$ at $\mathbf{x}_k$ with a Taylor approximation, call it $T(\mathbf{x})$. He then finds $argmin\, T(\mathbf{x})$ of the Taylor series and sets $\mathbf{x}_{k+1}$ to that. That is what one Taylor 'update' does.

How fast is Newton's Method for quadratic programming:

If A is symmetric positive definite, then the first order necessary condition is that $\nabla F(\mathbf{x}^*) = A\mathbf{x}^* - \mathbf{b} = \mathbf{0}$. Thus $\mathbf{x}^* = A^{-1}\mathbf{b}$. Due to the fact that $\nabla F = A\mathbf{x} - \mathbf{b}$ and $\nabla^2 F(\mathbf{x}) = A$, it follows that, $\mathbf{x}_{k+1} = \mathbf{x}_k - A^{-1}(A\mathbf{x}_k - \mathbf{b})$. Thus $\mathbf{x}_1 = \mathbf{x}_1 - A^{-1}\Big(A\mathbf{x}_0 - \mathbf{b}\Big) = A^{-1}\mathbf{b}$. Since $\mathbf{x}_1$ doesn't depend on $\mathbf{x}_0$, we have converged. In other words: One step for quadratic programming.

Theorem 4.1 *Assuming $F \in \mathbb{C}^2$ and $\nabla^2 F$ is Lipschitz continuous; if $\mathbf{x}_0$ is sufficiently close to $\mathbf{x}^*$, $\nabla F(\mathbf{x}^*) = 0$, and $\nabla^2 F(\mathbf{x}^*) > 0$, then $\exists C \in \mathbb{R}$ such that $||\mathbf{x}_{k+1} - \mathbf{x}^*|| \leq C||\mathbf{x}_k - \mathbf{x}^*||^2$. In other words, convergence is quadratic.*

Pros of Newton's method:

1. Will always converge for a strictly convex function
2. Newton's method is more efficient than gradient descent
3. Normally quadratic convergence
4. For quadratic programming, one step convergence

Cons of Newton's method:

1. Calculating $\nabla^2 F(\mathbf{x})$ is often expensive
2. If $\mathbf{x}^{(0)}$ is not close enough to x^*, $\{\mathbf{x}_k\}$ may diverge
3. Requires $F \in \mathbb{C}^2$ and that $\nabla^2 F$ is positive definite
4. It is not necessarily a 'descent method' (Ie, it is not necessarily true that $F(\mathbf{x}_{k+1}) \leq F(\mathbf{x}_k)$)

Theorem 4.2 *Let $\{\mathbf{x}^{(k)}\}$ be a sequence generated by Newton's method. If $\nabla^2 F(\mathbf{x}^{(k)}) > 0$ and $\nabla F(\mathbf{x}^{(k)}) \neq 0$ then the direction $\mathbf{d}_k = \left(\nabla^2 F(\mathbf{x}^{(k)})\right)^{-1} \nabla F(\mathbf{x}^{(k)})$ is a 'descent' direction. Ie, $\exists \beta > 0$ such that $F(\mathbf{x}^{(k)} + \alpha \mathbf{d}_k) < F(\mathbf{x}^{(k)})$, $\forall \alpha \in (0, \beta)$*

Guaranteed descent:

Newton's method with line search gives: $\nabla^2 F(\mathbf{x}^{(k)})\mathbf{d}_k = -\nabla F(\mathbf{x}^{(k)})$, and $\alpha_k = argmin\ F(\mathbf{x}^{(k)} + \alpha * \mathbf{d}_k)$ where $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{d}_k$ If we choose alpha like this, then it is true that $F(\mathbf{x}^{(k+1)}) < F(\mathbf{x}^{(k)})$; thus we have a guaranteed 'descent method'

Levenberg-Marquardt modification:

What if $\nabla^2 F(x)$ is not positive definite? We can use shifting to solve

this! Instead of using the standard Newton method, we can use the following $\mathbf{x}_{k+1} = \mathbf{x}_k - (\nabla^2 F(\mathbf{x}_k) + u_k I)^{-1} \nabla F(\mathbf{x}_k)$, where u_k is some number big enough to make $(\nabla^2 F(\mathbf{x}_k) + u_k I)$ positive definite.

You will notice here that as u_k approaches 0, this will function as Newton's Method, but as u_k approaches ∞, it will gradient descent method.

4.2 Lecture 11

Recall: To evaluate $(\nabla^2 F)^{-1}$ is very time consuming.

Nonlinear least squared problem

We are given a function $y = \alpha sin(\omega t + \phi)$ and a given set of inputs, $t_1, t_2, ...t_m$. We wish to find α, ω, ϕ such that the squared error is minimized. In other words, we want

$$\min_{\alpha,\omega,\phi} \sum_{i=1}^{m} \Big(y_i - \alpha sin(\omega t_i + \phi) \Big)^2$$

For simplicity, let

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \alpha \\ \omega \\ \phi \end{bmatrix}, \; r_i(\mathbf{x}) = y_i - \alpha sin(\omega t_i + \phi)$$

Now we want to find $\min F(\mathbf{x}) = \sum_{i=1}^{m} r_i(\mathbf{x})^2$. From this function, we calculate the gradient and hessian to be

$$\nabla F = \left\{ 2 \sum_{i=1}^{m} r_i(\mathbf{x}) \frac{\partial r_i}{\partial x_j} \right\}$$

$$\nabla^2 F = \left\{ 2 \sum_{i=1}^{m} \left(\frac{\partial r_i}{\partial x_k} \frac{\partial r_i}{\partial x_j} + r_i(\mathbf{x}) + \frac{\partial^2 r_i}{\partial x_k \partial x_j} \right) \right\}$$

To simplify this, let

$$\mathbf{r}(\mathbf{x}) = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}, \quad J(\mathbf{x}) = \begin{bmatrix} \frac{\partial r_1}{\partial x_1}, \frac{\partial r_1}{\partial x_2}, \frac{\partial r_1}{\partial x_3} \\ \frac{\partial r_2}{\partial x_1}, \frac{\partial r_2}{\partial x_2}, \frac{\partial r_2}{\partial x_3} \\ \vdots \quad \vdots \quad \vdots \\ \frac{\partial r_m}{\partial x_1}, \frac{\partial r_m}{\partial x_2}, \frac{\partial r_m}{\partial x_3} \end{bmatrix}$$

After doing this, one can tell that

$$\nabla F = 2J(\mathbf{x})^T \mathbf{r}(\mathbf{x})$$

For reasons that will become apparent later, let the matrix S be:

$$S(\mathbf{x}) = \left[\sum_{i=1}^{m} r_i(\mathbf{x}) \frac{\partial^2 r_i(\mathbf{x})}{\partial x_k \partial x_j} \right]$$

Doing so allows us to say that

$$\nabla^2 F = 2\big(J^T J + S(\mathbf{x})\big)$$

Now that we have all of the pieces, we can re-write Newton's method for the nonlinear least squares problem as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \big(J(\mathbf{x}_k)^T J(\mathbf{x}_k) + S(\mathbf{x}_k)\big)^{-1} J(\mathbf{x}_k)^T \mathbf{r}(\mathbf{x}_k)$$

For brevity, if we remove all the $(\mathbf{x}_k)$s we have:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \big(J^T J + S\big)^{-1} J^T \mathbf{r}$$

Gauss-Newton's method:

What if we drop $S(\mathbf{x}_k)$? It contains the second order information, *but* takes a long time to calculate. If we drop it, we get the Gauss-Newton's method.

$$J(\mathbf{x}) = \left[\frac{\partial r_i}{\partial x_j} \right]$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (J^T J)^{-1} J^T \mathbf{r}$$

Facts about this method:

1. Because we dropped S, we no longer have quadratic convergence, however this approach is often super linear. In other words, it is often that $1 < order\ of\ convergence < 2$.
2. This method only requires first order differentiable.
3. $J^T J$ is semi-positive definite. If it is not invertible, we can apply the Levenberg-Marquardt modification once more: At each step we choose a μ_k such that adding $\mu_k I$ to $J^T J$ will cause $J^T J + \mu_k I$ to be positive definite. For this method, we use the formula:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (J^T J + \mu_k I)^{-1} J^T \mathbf{r}$$

Chapter 5

Barzilai-Borwein method

5.1 lecture 11

The Barzilai-Borwein method (BB step size)

It is a gradient descent method with special step size, meaning it has the form $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$. The computation cost of a single step is similar to the standard gradient method, but performance is significantly better.

We want to solve $\min_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x})$. We now have many methods to do so, lets look at three:

1. Gradient descent $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x}_k)$
2. Newton's method $\mathbf{x}_{k+1} = \mathbf{x}_k - \nabla^2 F(\mathbf{x}_k)^{-1} \nabla F(\mathbf{x}_k)$
3. BB method: choose α_k such that $\alpha_k \nabla F(\mathbf{x}_k) \approx \nabla^2 F(\mathbf{x}_k)^{-1} \nabla F(\mathbf{x}_k)$

For clarity, let's introduce a few new notations. Let

$$\mathbf{p}_k = \nabla F(\mathbf{x}_k), \quad H_k = \nabla^2 F(\mathbf{x}_k)$$

Thus, for the BB method, we want to find α_k such that $\alpha_k \mathbf{p}_k \approx H_k^{-1} \mathbf{p}_k$

Consider the quadratic programming problem $\min F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ where A is symmetric positive definite. Here one can see that Newton's method is $H_k^{-1}\mathbf{p}_k = A^{-1}\mathbf{p}_k$. Thus for this problem we would like to find α_k such that $\alpha_k \mathbf{p}_k \approx A^{-1}\mathbf{p}_k$. In other words, we want to find α_k such that $(\frac{I}{\alpha_k})^{-1}\mathbf{p}_k \approx A^{-1}\mathbf{p}_k$.

Notation: Let $\mathbf{S}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$ and $\mathbf{y}_{k-1} = \mathbf{p}_k - \mathbf{p}_{k-1}$

Thus $\mathbf{y}_{k-1} = A\mathbf{x}_k - \mathbf{b} - (A\mathbf{x}_{k-1} - \mathbf{b}) = A\mathbf{S}_{k-1}$, for quadratic programming. From this we can find that we would like to choose α_k such that

$\alpha_k^{-1}\mathbf{S}_{k-1} \approx \mathbf{y}_{k-1}$

From that we get the least squared problem:

$$\alpha_k^{-1} = \arg\min_{\beta} \frac{1}{2}||\beta\mathbf{S}_{k-1} - \mathbf{y}_{k-1}||^2$$

5.2 Lecture 12

Deriving BB step size:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{p}_k$$

We are looking for an α_k such that $\alpha_k\mathbf{p}_k \approx H_k^{-1}\mathbf{p}_k$. For quadratic programming, A is symmetric positive definite. Since $H_k = A$, we are looking for α_k such that $\alpha_k\mathbf{p}_k \approx A^{-1}\mathbf{p}_k$

Recall that last class we stated that:

$$\mathbf{S}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1} = \Delta\mathbf{x}_{k-1}$$

$$\mathbf{y}_{k-1} = \mathbf{p}_k - \mathbf{p}_{k-1} = \Delta\mathbf{p}_k = A\mathbf{x}_k - \mathbf{b} - (A\mathbf{x}_k - \mathbf{b}) = A\Delta\mathbf{x}_{k-1}$$

Thus we derive the secant equation, $\Delta\mathbf{p}_{k-1} = A\Delta\mathbf{x}_{k-1}$. We are now looking for an α_k such that $\frac{1}{\alpha_k}\Delta\mathbf{x}_{k-1} \approx \Delta\mathbf{p}_{k-1}$. In other words we have $\mathbf{x}_k = \alpha_k\Delta\mathbf{p}_{k-1}$ To solve this, we can solve one of the following least squared problems:

1. $\alpha_k^{-1} = argmin_\beta||\beta\Delta\mathbf{x}_{k-1} - \Delta\mathbf{p}_{k-1}||_2^2$
2. $\alpha_k = argmin\ ||\Delta\mathbf{x}_{k-1} - \alpha\Delta\mathbf{p}_{k-1}||_2^2$

Solving problem 1, one has that

$$\alpha_k = \frac{\Delta\mathbf{x}_{k-1}^T\Delta\mathbf{x}_{k-1}}{\Delta\mathbf{x}_{k-1}^T\Delta\mathbf{p}_{k-1}}$$

Solving problem 2, one has that

$$\alpha_k = \frac{\Delta\mathbf{x}_{k-1}^T\Delta\mathbf{p}_{k-1}}{\Delta\mathbf{p}_{k-1}^T\Delta\mathbf{p}_{k-1}}$$

You can use *either* α for BB step size.

Algorithm for BB-step size:

Regardless, of which you choose, we get the following algorithm. To $\min_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x})$ where $F \in \mathbb{C}^1$, do the following:

1. Guess $\mathbf{x}_0$
2. $\mathbf{p}_0 = \nabla F(\mathbf{x}_0)$
3. $\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{p}_0$ where α_0 is some small number.
4. $\mathbf{p}_1 = \nabla F(\mathbf{x}_1)$
5. Repeat the following:

   a. $\mathbf{p}_k = \nabla F(\mathbf{x}_k)$

   b. $\Delta \mathbf{x}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$

   c. $\Delta \mathbf{p}_{k-1} = \mathbf{p}_k - \mathbf{p}_{k-1}$

   d. $\alpha_k = \begin{cases} \alpha_k = \frac{\Delta \mathbf{x}_{k-1}^T \Delta \mathbf{x}_{k-1}}{\Delta \mathbf{x}_{k-1}^T \Delta \mathbf{p}_{k-1}} \\ \alpha_k = \frac{\Delta \mathbf{x}_{k-1}^T \Delta \mathbf{p}_{k-1}}{\Delta \mathbf{p}_{k-1}^T \Delta \mathbf{p}_{k-1}} \\ \text{Alternate} \end{cases}$

   Where you choose whatever alpha you wish of the list above.

   e. $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}$

BB Step Size Example:

Consider $\min F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x}$, $A = \begin{bmatrix} 6, 4 \\ 4, 6 \end{bmatrix}$

First, we will select an initial guess $\mathbf{x}_0$, a small α_0 and then make $\mathbf{p}_0$:

$$\alpha_0 = 0.1, \quad \mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 2 \end{bmatrix}, \quad \mathbf{p}_0 = A\mathbf{x}_0 = \begin{bmatrix} 6, 4 \\ 4, 6 \end{bmatrix} \begin{bmatrix} 0.5 \\ 2 \end{bmatrix} = \begin{bmatrix} 11 \\ 14 \end{bmatrix}$$

Now, we can calculate $\mathbf{x}_1$ and $\mathbf{p}_1$:

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{p}_0 = \begin{bmatrix} 0.5 \\ 2 \end{bmatrix} - 0.1 \begin{bmatrix} 11 \\ 14 \end{bmatrix} = \begin{bmatrix} -0.6 \\ 0.6 \end{bmatrix}$$

$$\mathbf{p}_1 = A\mathbf{x}_1 = \begin{bmatrix} 6, 4 \\ 4, 6 \end{bmatrix} \begin{bmatrix} -0.6 \\ 0.6 \end{bmatrix} = \begin{bmatrix} -1.2 \\ 1.2 \end{bmatrix}$$

Then, we calculate our first $\Delta \mathbf{x}$ and Δp:

$$\Delta \mathbf{x}_0 = \mathbf{x}_1 - \mathbf{x}_0 = \begin{bmatrix} -1.1 \\ -1.4 \end{bmatrix}, \quad \Delta \mathbf{p}_0 = \mathbf{p}_1 - \mathbf{p}_0 = \begin{bmatrix} -12.2 \\ -12.8 \end{bmatrix}$$

Now we can calculate the value for α_1 (remember, we can only freely choose α_0)

$$\alpha_1 = \frac{\Delta \mathbf{x}_0^T \Delta \mathbf{x}_0}{\Delta \mathbf{x}_0^T \Delta \mathbf{p}_0} = \frac{317}{3134} \approx 0.1011$$

Finally, we simply repeat the following as needed

$$\alpha_k = \frac{\Delta \mathbf{x}_{k-1}^T \Delta \mathbf{x}_{k-1}}{\Delta \mathbf{x}_{k-1}^T \Delta \mathbf{p}_{k-1}}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k$$

$$\mathbf{p}_{k+1} = A\mathbf{x}_{k+1}$$

$$\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$$

$$\Delta\mathbf{p}_k = \mathbf{p}_{k+1} - \mathbf{p}_k$$

Convergence of BB step size

1. For quadratic functions, BB step size has R-linear convergence. This is: $||\mathbf{x}_k - \mathbf{x}^*||_2 \leq C_k$, $\lim_{k\to\infty} C_k = 0$
2. For 2D quadratic function, BB step size has super-linear convergence. $\lim_{k\to\infty} \frac{||\mathbf{x}_k - \mathbf{x}^*||}{||\mathbf{x}_{k-1} - \mathbf{x}^*||} = 0$
3. General function: On a general unconstrained differentiable problem, pairing BB step size with a non-monotone line search works very well.

Chapter 6

Quasi-Newton Method

6.1 Lecture 12

Recall Newton's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla^2 F(\mathbf{x}_k)^{-1} \nabla F(\mathbf{x}_k)$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k H_k^{-1} \mathbf{p}_k$$

If $\alpha_k = arg\min_{\alpha \geq 0} F(\mathbf{x}_k - \alpha H_k^{-1}\mathbf{p}_k)$ and $H_k > 0$, then $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$. Calculating H_k^{-1} is expensive.

Goal of Quasi-Newton: To generate a sequence of matrices $Q_k \approx H_k^{-1}$ or $B_k \approx H_k$

Construction of Q_k

Given $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k Q_k \mathbf{p}_k$, we want to ensure $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$

$$\begin{aligned} F(\mathbf{x}_{k+1}) &= F(\mathbf{x}_k) + \nabla F(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) + O(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2) \\ &= F(\mathbf{x}_k) + \mathbf{p}_k^T(-\alpha_k Q_k \mathbf{p}_k) + O(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2) \\ &= F(\mathbf{x}_k) - \alpha_k \mathbf{p}_k^T Q_k \mathbf{p}_k + O(\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2) \end{aligned}$$

Thus, we can deduce we want $Q_k = Q_k^T$ and $Q_k > 0$

Theorem 6.1 (Theoretical guidance of Q_k) *Assuming $F \in \mathbb{C}^1$ and $\mathbf{p}_k = \nabla F(\mathbf{x}_k) \neq 0$. If $Q_k = Q_k^T$ and Q_k is positive definite, then if an update is preformed where $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k Q_k \mathbf{p}_k$ and $\alpha_k = arg\min_{\alpha \geq 0} F(\mathbf{x}_k - \alpha Q_k \mathbf{p}_k)$, it is true that $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$.*

Motivation in the simple quadratic problem

Consider $\min \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ where $A = A^T$ and $A > 0$. Here it is clear that $\mathbf{p}_k = A\mathbf{x}_k - \mathbf{b}$ and $H_k = A$. Thus we have

$$\mathbf{p}_{k+1} - \mathbf{p}_k = A\mathbf{x}_{k+1} - \mathbf{b} - (A\mathbf{x}_k - \mathbf{b})$$

$$\Delta\mathbf{p}_k = A\Delta\mathbf{x}_k \text{ or } A^{-1}\Delta\mathbf{p}_i = \Delta\mathbf{x}_i \quad i = 0, 1, 2, \dots, k$$

Thus, we can deduce we want $Q_{k+1}\Delta\mathbf{p}_i = \Delta\mathbf{x}_i$, where $i = 0, 1, ..., k$ or $\Delta\mathbf{p}_i = \beta_{k+1}\Delta\mathbf{x}_i$, where $i = 0, 1, ..., k$

In the context of quadratic programming where A is symmetric, positive definite, and square, it is true that if $\Delta\mathbf{p}_0, \Delta\mathbf{p}_1, ..., \Delta\mathbf{p}_{n-1}$ are linearly independent, then $Q_n = A^{-1}$ which means $\mathbf{x}_{n+1}$ should be the true solution of our equation: $\min_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$

Theorem 6.2 (More theoretical guidance of Q_k) *For the quadratic problem $\frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, where $A > 0$, the following holds. If $Q_{k+1}\Delta\mathbf{p}_i = \Delta\mathbf{x}_i$ for $i = 0, 1, ..., k+1$, then $Q_i\mathbf{p}_i = d_i$. In other words $\langle d_i, d_j\rangle_A = 0$ where $i \neq j$. This can be used in $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha_i d_i$.*

6.2 Lecture 13

Recall that we want to construct $Q_k \approx H_k^{-1}$ or a $B_k \approx H_k$ such that B_k^{-1} is easy to compute.

We also require these values satisfy two conditions:

1. $Q_k = Q_k^T$ and $Q_k > 0$. When this is combined with a line search for α (recall our equation is $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k Q_k\mathbf{p}_k$), we are guaranteed that $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$

2. $Q_k\Delta\mathbf{p}_i = \Delta\mathbf{x}_i$, $i = 0, 1, ...k-1$, where $\Delta\mathbf{x}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$ and $\Delta\mathbf{p}_i = \mathbf{p}_{i+1} - \mathbf{p}_i = \nabla F(\mathbf{x}_{i+1}) - \nabla F(\mathbf{x}_i)$. This guarantees that we have the secant equation: This guarantees us that this method will converge in at most n steps.

We have three methods that iteratively update Q_k that satisfy these.

Single rank symmetric algorithm (SRS)

This method is a rank one correction formula. Starting from $Q_0 = I$ (Q_0 must be spd) $Q_1 = Q_0 + \beta\mathbf{z}_0\mathbf{z}_0^T$ Here $Q_i \in \mathbb{R}^{nxn}$, $\mathbf{z} \in \mathbb{R}^n$, $\beta_0 \in \mathbb{R}$

Example: $Q_0 + 2\begin{bmatrix}1\\2\\3\\\vdots\\n\end{bmatrix}\begin{bmatrix}1,2,3,\ldots,n\end{bmatrix}$, here Q_0 is being added to a rank-1

General formula:

$$Q_{k+1} = Q_k + \beta_k \mathbf{z}_k \mathbf{z}_k^T$$

$$\mathbf{z}_k = \Delta\mathbf{x}_k - Q_k \Delta\mathbf{p}_k$$

$$\beta_k = \frac{1}{\Delta\mathbf{p}_k^T(\Delta\mathbf{x}_k - Q_k\Delta\mathbf{p}_k)}$$

Facts about this method:

1. Q_k is symmetric
2. It satisfies the secant equation
3. However this formula *cannot* guarantee Q_k is always positive definite.

Using a Rank 1 algorithm to solve $\min F(\mathbf{x})$

To solve $\min_{\mathbf{x}} F(\mathbf{x})$, select $\mathbf{x}_0$ and Q_0 where $Q_0 = Q_0^T$ and $Q_0 > 0$. (Generally, select I). Then calculate $\mathbf{p}_0$ from $\mathbf{x}_0$. Loop the following for $k = 0, 1, \ldots, n-1$

$$\mathbf{d}_k = Q_k \mathbf{p}_k$$

$$\alpha_k = argmin_\alpha F(\mathbf{x}_k - \alpha \mathbf{d}_k)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{d}_k$$

$$\Delta\mathbf{x}_k = -\alpha_k \mathbf{d}_k$$

$$\Delta\mathbf{p}_k = \mathbf{p}_{k+1} - \mathbf{p}_k$$

$$Q_{k+1} = Q_k + \frac{(\Delta\mathbf{x}_k - Q_k\Delta\mathbf{p}_k)(\Delta\mathbf{x}_k - Q_k\Delta\mathbf{p}_k)^T}{\Delta\mathbf{p}_k^T(\Delta\mathbf{x}_k - Q_k\Delta\mathbf{p}_k)}$$

Note: For a quadratic programming problem, $\alpha_k = \frac{<\mathbf{p}_k, \mathbf{d}_k>}{<\mathbf{d}_k, \mathbf{d}_k>_A}$

Example:

Let $F(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^2 + \frac{1}{2}\mathbf{x}_2^2 + 3 = \frac{1}{2}\mathbf{x}^T \begin{bmatrix}2,0\\0,1\end{bmatrix} x + 3$. Furthermore, let $\mathbf{p}_k = \nabla F(\mathbf{x}_k) = \begin{bmatrix}2,0\\0,1\end{bmatrix}\mathbf{x}_k$. We begin by guessing $\mathbf{x}_0$ and Q_0, then calculating $\mathbf{p}_0$

$$\mathbf{x}_0 = \begin{bmatrix}1\\2\end{bmatrix} \quad Q_0 = \begin{bmatrix}1,0\\0,1\end{bmatrix} \quad \mathbf{p}_0 = \begin{bmatrix}2,0\\0,1\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix} = \begin{bmatrix}2\\2\end{bmatrix}$$

From there we can do the following calculations:

$$\mathbf{d}_0 = Q_0\mathbf{p}_0 = \begin{bmatrix}2\\2\end{bmatrix}$$

$$\alpha_0 = \frac{\langle \mathbf{p}_0, \mathbf{d}_0\rangle}{\langle \mathbf{d}_0, \mathbf{d}_0\rangle_A} = \frac{\begin{bmatrix}2,2\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}}{\begin{bmatrix}2,2\end{bmatrix}\begin{bmatrix}2,0\\2,1\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}} = \frac{2}{3}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0\mathbf{d}_0 = \begin{bmatrix}1\\2\end{bmatrix} - \frac{2}{3}\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}\frac{-1}{3}\\\frac{2}{3}\end{bmatrix}$$

$$\Delta\mathbf{x}_0 = \mathbf{x}_1 - \mathbf{x}_0 = \begin{bmatrix}\frac{-4}{3}\\\frac{-4}{3}\end{bmatrix}$$

$$\Delta\mathbf{p}_0 = \mathbf{p}_1 - \mathbf{p}_0 = \begin{bmatrix}\frac{-8}{3}\\\frac{-4}{3}\end{bmatrix}$$

$$Q_1 = Q_0 + \frac{(\Delta\mathbf{x}_0 - Q_0\Delta\mathbf{p}_0)(\Delta\mathbf{x}_0 - Q_0\Delta\mathbf{p}_0)^T}{\Delta\mathbf{p}_0^T(\Delta\mathbf{x}_0 - Q_0\Delta\mathbf{p}_0)} = \begin{bmatrix}\frac{1}{2},0\\0,1\end{bmatrix}$$

Now we can repeat the steps to find the answer.

$$\mathbf{d}_1 = Q_1\mathbf{p}_1 = \begin{bmatrix}\frac{-1}{3}\\\frac{2}{3}\end{bmatrix}$$

$$\alpha_1 = 1$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha_1\mathbf{d}_1 = \begin{bmatrix}0\\0\end{bmatrix}$$

Davidon-Fletcher-Powell (DFP)

This method uses rank two perturbation. The method is as follows

$$Q_{k+1} = Q_k + \mathbf{u}_k\mathbf{u}_k^T + \mathbf{v}_k\mathbf{v}_k^T, \quad \mathbf{u}_k, \mathbf{v}_k \in \mathbb{R}^{nxn}$$

$$Q_{k+1} = Q_k + \frac{\Delta\mathbf{x}_k\Delta\mathbf{x}_k^T}{\Delta\mathbf{x}_k^T\Delta\mathbf{p}_k} - \frac{(Q_k\Delta\mathbf{p}_k)(Q_k\Delta\mathbf{p}_k)^T}{\Delta\mathbf{p}_k^T(Q_k\Delta\mathbf{p}_k)}$$

Facts about this method:

1. Q_k is symmetric positive definite.
2. It satisfies the secant equation
3. However, Q_k might become nearly singular.

Broyden-Flecher-Goldfarb-Shanno (BFGS)

This method uses rank two perturbation *on* $B_k \approx H_k$, where $B_k = B_k^T$ and B_k is symmetric positive definite. The method is as follows:

$$\Delta \mathbf{p}_i = B_k \Delta \mathbf{x}_i, \quad i = 0, i, ...k-1$$

$$B_{k+1} = B_k + \frac{\Delta \mathbf{p}_k \Delta \mathbf{p}_k^T}{\Delta \mathbf{x}_k^T \Delta \mathbf{p}_k} - \frac{B_k \Delta \mathbf{x}_k \Delta \mathbf{x}_k^T B_k}{\Delta \mathbf{x}_k^T B_k \Delta \mathbf{x}_k}$$

Facts about this method:

1. B_k is spd
2. $\Delta \mathbf{p}_i = B_k \Delta x_i, \ i = 0, 1, ...k-1$

Lemma 1 (Sherman-Morrison) *If* $A \in \mathbb{R}^{n \times n}$ *is non-singular and* $1 + \mathbf{v}^T A^{-1} \mathbf{u} \neq 0$ *then* $A + \mathbf{u}\mathbf{v}^T$ *is also non-singular.*

$$(A + \mathbf{u}\mathbf{v}^T)^{-1} = A^{-1} - \frac{(A^{-1}\mathbf{u})(\mathbf{u}^T A^{-1})}{1 + \mathbf{v}^T A^{-1} \mathbf{u}}$$

Therefore BFGS:

$$Q_{k+1} = Q_k + \left(1 + \frac{\Delta \mathbf{p}_k^T Q_k \Delta \mathbf{p}_k}{\Delta \mathbf{p}_k^T \Delta \mathbf{x}_k}\right) \frac{\Delta \mathbf{x}_k \Delta \mathbf{x}_k^T}{\Delta \mathbf{x}_k^T \Delta \mathbf{p}_k} - \frac{Q_k \Delta \mathbf{p}_k \Delta \mathbf{x}_k^T + \Delta \mathbf{x}_k \Delta \mathbf{p}_k^T Q_k}{\Delta \mathbf{p}_k^T \Delta \mathbf{x}_k}$$

Theorem 6.3 (convergence) *If* F *is strongly convex, BFGS with backtracking line search converges for any* $\mathbf{x}_0$ *and any* Q_0 *that is spd.*
If $\nabla^2 F$ *is Lipschitz, then local convergence is super linear. That is:*

$$\lim_{k \to \infty} C_k = 0 \quad \textit{where} \quad ||\mathbf{x}_{k+1} - \mathbf{x}^*||_2 \leq C_k ||\mathbf{x}_k - \mathbf{x}^*||_2$$

Chapter 7

Linear programming

7.1 Lecture 14

Goal: Determine values of decision variables that maximize or minimize a linear objective function, where decision variables are subject to linear constraints

Decision variables What we are trying to solve for.

We define $A\mathbf{x} \geq \mathbf{b}$ to be:

$$A\mathbf{x} \geq \mathbf{b} \iff \sum_{j=1}^{n} a_{ij}x_j \geq b_i, \quad i = 1, 2, \dots \ m$$

History of L.P. solutions

1. Simplex: number of steps $\approx O(e^n)$
2. Khachiyam: Polynomial time
3. Karmatar: Polynomial time but better. The interior point method

Formal definition of Linear programming:

$\min_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x})$ such that $\mathbf{x} \geq \mathbf{0},\ G(\mathbf{x}) = 0,\ H(\mathbf{x}) \geq 0$ where F, G, H are all linearly dependent on $\mathbf{x}$. This can be converted to the standard form:

Standard form:

$$\min_{\mathbf{x}\in\mathbb{R}^T} A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} \text{ where } \mathbf{c} \in \mathbb{R}^n$$

Quick note: The loose inequality is necessary; otherwise this may be an open set and may not be solve-able.

Manufacturing Example

The 16 Gb iPod requires two 8 Gb chips
The 8 Gb iPod requires 8 Gb chip

Weekly resources are limited to:
at most 800 units of 3.5 inch LCD
at most 1000 units of 8 Gb chips
50 hours of total labor time
4 mins of labor are required to make a 16 Gb iPod
3 mins of labor are required to make a 8 Gb iPod

For marketing reasons:
Total production of ipod cannot exceed 700
Number of 16 Gb iPod cannot exceed Number of 8 Gb iPod by more than 300

Profit
16 dollars each for 16 Gb iPod
10 dollars each for 8 Gb iPod

Current Production plan is as follows:
450 of 16 Gb iPod + 100 of 8 Gb iPod = $8200

We are looking to maximize the profits.

Decision Variables
x_1 = weekly produced units of 16 Gb iPod
x_2 = weekly produced units of 8 Gb iPod

Objective function: $max_{x1,x2} 16x1 + 10x2$
constraints:

x_1, $x_2 > 0$

LCD $x_1 + x_2 \leq 800$

memory $2x_1 + x_2 \leq 1000$

labor $4x_1 + 6x_2 \leq 3000$

Marketing total $x_1 + x_2 \leq 700$

Marketing mix $x_1 - x_2 \leq 350$

Diet Example

There are n different food types (price of j-th food is c_j).
There are m basic nutrients.

To achieve a balanced diet, need to receive at least b_i units of the i-th nutrient.
Assume each unit of j-th type of food contains a_ij units of the i-th nutrient.

Let x_j = the number of units of food j

We are looking to minimize the cost of the diet:
c_1 x_1, c_2 x_2, ..., c_n x_n
such that:
a_11 x_1 + a_12 x_2 + ... + a_1n x_n >= b_1
a_21 x_1 + a_22 x_2 + ... + a_2n x_n >= b_2
...
a_m1 x_1 + a_m2 x_2 + ... + a_mn x_n >= b_m
and x_1 >= 0, x_2 >= 0, ..., x_n >= 0

C = (c_1; c_2; ...; c_n)
x = (x_1; x_2; ...; x_n)
A = (a_11, a_12, ..., a_1n; ... ; a_m1, a_m2, ..., a_mn)
b = (b_1; b_2; ...; b_n)

min C^x
s.t.
Ax >= b
x >= 0

A manufacturing company has plants in cities A_1, A_2, A_3.
The company distributes its products to cities B_1, B_2, B_3, B_4.

	B_1	B_2	B_3	B_4	supply
A_1	$7	$10	$14	$18	30
A_2	$7	$11	$12	$6	40
A_3	$5	$18	$15	$9	30
Demand	20	20	25	35	

Denote x_ij as the number of products from A_i to B_j
Denote c_ij as the price of moving 1 unit from A_i to B_j

Then, the objective function is
sum_{i, j} x_ij c_ij
s.t.
x_ij >= 0
x * 1 = (30; 40; 30)
x^T * 1 = (20; 20; 25; 35)

*Geometric Point of View
example:
max x_1 + 5 x_2

```
s.t.
5 x_1 + 6 x_2 <= 30
3 x_1 + 2 x_2 <= 12
x_1 >= 0
x_2 >= 0

Standard Form Equivalent:
max_x C^T X
s.t.
Ax <= b
x >= 0
C = (1, 5)
A = (5, 6;3, 2)
b = (3, 0;1, 2)
```

hyperplane $\mathbb{R}^n$: $\{x \in \mathbb{R}^n \mid a^T x = b\}$
where $a = (a_1; a_2; ...; a_n) \in \mathbb{R}^n$ and $b \in \mathbb{R}$

half space : $\{x \in \mathbb{R}^n \mid a^T x \geq b\}$ or $\{x \in \mathbb{R}^n \mid a^T x \leq b\}$

7.2 Lecture 15

Geometric idea:

At a very basic geometric level: We can constrain our problem to a polytope (hyper polygon). From then we can locate the extreme points in the shape (imagine corner points). Then we either check each extreme point to find which optimizes the objective function or keep checking points until a certain criteria which ensures the objective function is optimized.

What we will do:

1. We will design a numerical algorithm to transform (iterate) between vertices of the polytope.
2. Design a stopping criteria (an efficiently method to verify that our solution optimizes the objective function)

Definitions:

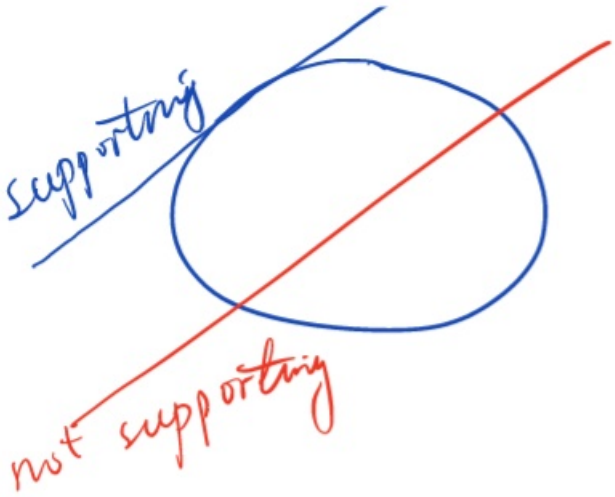
Hyperplane: $\mathbf{x} \in \mathbb{R}^n$ such that $\sum_{i=1}^{n} a_i x_i = \mathbf{c}$ where $\mathbf{c} \in \mathbb{R}$

Half space: $\mathbf{x} \in \mathbb{R}^n$ such that $\mathbf{a}^T\mathbf{x} \geq \mathbf{b}$. In other words, it is either 'side' of a hyperplane that divides a space

Convex set: ω is called convex if $\forall \mathbf{x}, \mathbf{y} \in \omega,\ \forall \alpha \in [0, 1]$ it is true that $\alpha\mathbf{x} + (1 - \alpha)\mathbf{y} \in \omega$). In other words: you can pick any two points in the

convex set and the entire line segment between those two points will be in the set.

Supporting hyperplane: A supporting hyperplane of a convex set is a hyperplane that passes through a point in the convex set, where all of the convex set is contained within said hyperplane and one of the two half spaces of the hyperplane. For a smooth curve, you can think of it as a tangent plane; however, for a convex set like a square the supporting hyperplane could not be called tangent at the vertex.



Convex set fact: The intersection of convex sets is also convex.

General idea: To solve the optimization problem, set the objective function equal to a constant such that it will create a supporting hyperplane of the polytope created by the constraints. In the case of a minimization, the polytope will be in the positive half space of the hyperplane; and in the case of a maximization, the negative half space.

Standard form of L.P.

All L.P. problems can be written in this form:

$$\min \mathbf{C}^T\mathbf{x} \text{ such that } A\mathbf{x} = \mathbf{b},\ x_i \geq 0,\ \forall x_i \in \mathbf{x},\ A \in \mathbb{R}^n$$

Note: Any maximization problem can be made into a minimization problem by negating the objective function

Example:

$$\min(-x_1 - 5x_2) \text{ such that:}$$

$$5x_1 + 6x_2 \leq 30$$
$$3x_1 + 2x_2 \leq 12$$
$$x_1, x_2 \geq 0$$

We can now introduce **slack variables**:

$$5x_1 + 6x_2 + y_1 = 30, \text{ where } y_1 \geq 0$$
$$3x_1 + 2x_2 + y_2 = 12, \text{ where } y_2 \geq 0$$

This leads us to:

$$\min(-x_1 - 5x_2) \text{ such that } 5x_1 + 6x_2 \leq 30$$
$$3x_1 + 2x_2 + y_2 = 12$$
$$x_1, x_2, y_1, y_2 \geq 0$$

We can further generalize this by renaming y_1 to x_3 and y_2 to x_4

$$A = \begin{bmatrix} 5, 6, 1, 0 \\ 3, 2, 0, 1 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} -1 \\ -5 \\ 0 \\ 0 \end{bmatrix}$$

Thus we have $\min \begin{bmatrix} -1, -5, 0, 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ such that $A\mathbf{x} = \begin{bmatrix} 30 \\ 12 \end{bmatrix}$ and $\mathbf{x} \geq \mathbf{0}$

More generally:

$\min \mathbf{C}^T\mathbf{x}$ such that $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq \mathbf{0}$ is equivalent to $\min \mathbf{C}^T\mathbf{x}$ such that $A\mathbf{x} + \mathbf{y} = \mathbf{b}$ and $\mathbf{x}, \mathbf{y} \geq \mathbf{0}$

If we let, $\mathbf{x}_{new} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$, $A_{new} = \begin{bmatrix} A, I \end{bmatrix}$ and $\mathbf{C}_{new} = \begin{bmatrix} \mathbf{C} \\ \mathbf{0} \end{bmatrix}$ then we have $\min \mathbf{C}_{new}^T \mathbf{x}_{new}$ such that $A_{new}\mathbf{x}_{new} = \mathbf{b}$ and $\mathbf{x}_{new} \geq \mathbf{0}$.

Quick Note: When we require $A\mathbf{x} \geq \mathbf{b}$, we can use **surplus** variables. Here you subtract $\mathbf{y}$ instead of adding it. In this case: $A_{new} = \begin{bmatrix} A, -I \end{bmatrix}$

It is also common to have both slack and surplus variables, such as is required below:

$$\min x_1 + 5x_2 \text{ such that:}$$
$$5x_1 + 6x_2 \leq 30$$
$$3x_1 + 2x_2 \geq 12$$

$$x_1, x_2 \geq 0,\ A = \begin{bmatrix} 5, 6, 1, 0 \\ 3, 2, 0, -1 \end{bmatrix}$$

Surplus/Slack variables:

We can adjust our inequalities to become equalities by adding a value (slack variables) or subtracting a value (surplus variables) without affecting the final solution. In other words, if we have $\mathbf{x} \leq \mathbf{c}$, we can say $\mathbf{x} + \mathbf{y} = \mathbf{c}$ for some $\mathbf{y} \geq \mathbf{0}$. This $\mathbf{y}$ is called a slack variable. If $\mathbf{x} \geq \mathbf{c}$, we could say $\mathbf{x} - \mathbf{y} = \mathbf{c}$. Here $\mathbf{y}$ is a surplus variable.

In the case of a constraint involving an absolute value, we can simply split the constraint into two sub-constraints. For example:
$\|x_2\| \leq 2$ can be split into $x_2 \leq 2$ and $x_2 \geq -2$

In the case of a negative decision varible, we can simply substitute a negative version of the variable into the equation. For example:

$$\min(x_1 - x_2) \text{ such that } 3x_1 - x_2 = -5,\ x_2 \leq 2,\ x_2 \geq -2,\ x_1 \leq 0$$

can be transformed into

$$\min(-x_1' - x_2) \text{ such that } -3x_1' - x_2 = -5,\ \ x_2 + y_1 = 2,\ \ x_2 - y_2 = -2$$

$$x_1', y_1, y_2 \geq 0$$

But x_2 is still unconstrained, so we can constrain it by noting that $x_2 = u - v$, where $u \geq 0$ and $v \geq 0$. Doing we, we have:

$$\min -x_1' - x_2 \text{ such that:}$$

$$-3x_1' - x_2 = -5$$

$$u - v + y_1 = 2$$

$$u - v - y_2 = -2$$

$$x_1',\ y_1,\ y_2,\ u,\ v\ \geq 0$$

7.3 Lecture 16

Two questions we must answer to solve LP problems:

1. How do we move from one extreme to another?
2. How do we know when to stop?

Extreme point of a convex set: $x \in \mathbf{\Omega}$ is an extreme point of there do *not* exists two points $y_1, y_2 \in \mathbf{\Omega}$ such that $x = \alpha y_1 + (1-\alpha)y_2$ where $\alpha \in (0,1)$. In other words, you cannot chose two points, $y_1, y_2 \in \mathbf{\Omega}$ such that x is on a line connecting y_1 and y_2.

Consider a L.P. problem:

$$\min \mathbf{C}^T\mathbf{x} \text{ such that } A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}$$

Here $\mathbf{x}, \mathbf{C} \in \mathbb{R}^n$, $A \in \mathbb{R}^{mxn}, m \leq n$, and $Rank(A) = m$.

Defining terminology:

Define $\mathbf{\Omega} = \{\mathbf{x} \in \mathbb{R}^n \text{ such that } A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$. Thus we want $\min_{\mathbf{x} \in \mathbf{\Omega}} \mathbf{C}^T\mathbf{x}$. Here $\mathbf{\Omega}$ is our constrained domain.

$$\text{Note that } A = \left[a_1, a_2, \ldots, a_n\right],\ a_i \in \mathbb{R}^n$$

$$\text{Let } B \subseteq \{1, 2, 3, ..., n\}, \text{ such that B has m elements: } |B| = m$$

$$\text{We define } A_B = \left[a_{i_1}, a_{i_2}, \ldots, a_{i_m}\right], \text{ where } B = i_1, i_2, \ldots, i_m$$

In other words, B is a set of index values of A's columns, and A_B is the matrix containing the columns enumerated by B.
Note that here, B is invertible.

Consider an equation: $A_B\mathbf{x}_B = b$. Since $rank(A) = m$, then it follows that $det(A_B) \neq 0$. Thus A_B is inevitable and $\mathbf{x}_B = A_B^{-1}\mathbf{b}$. This suggests a solution of $A\mathbf{x} = \mathbf{b}$

The entire index set is $\{1, 2, ..., n\} = \{B, B^C\}$. By this we mean that the index set is the union of B and B^C. We can now define a vector $\mathbf{x}_{new}$ such that

$$\mathbf{x}_{new}(i) = \begin{cases} \mathbf{x}(i)\ if \mathbf{x}(i) \in B \\ 0\ else \end{cases}$$

For example, if $B = \{2, 4\}$, and $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ then we have that $\mathbf{x}_{new} = \begin{bmatrix} 0 \\ x_2 \\ 0 \\ x_4 \end{bmatrix}$. We will denote this $\mathbf{x}_{new}$ as $\begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix}$.

Claim $\mathbf{x}_{new}$ is a solution of $A\mathbf{x} = \mathbf{b}$

Why? Well: $A\mathbf{x} = \left[A_B, A_{B^C}\right] \begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix} = A_B\mathbf{x}_B = A_BA_B^{-1}b = \mathbf{b}$

Define $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} A_B^{-1}\mathbf{b} \\ \mathbf{0} \end{bmatrix}$. We call $\mathbf{x}$ a basic solution to $A\mathbf{x} = \mathbf{b}$

We call $\{a_i\}_{i \in B}$ basic columns, and $\mathbf{x}_B$ basic variables.

Furthermore, if $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix} \geq \mathbf{0}$, then $\mathbf{x}$ is called a basic feasible solution.

In other words, a BFS means $\mathbf{x} \in \mathbf{\Omega} = \{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$ and $\mathbf{x}$ is basic.

In the case of the above example:

$$A = \begin{bmatrix} 1,5,1,0,0 \\ 2,1,0,1,0 \\ 1,1,0,0,1 \end{bmatrix},\ B = \{3,4,5\},\ b = \begin{bmatrix} 40 \\ 20 \\ 12 \end{bmatrix}$$

Basic columns $A_B = \begin{bmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{bmatrix}$

Basic variables $\mathbf{x}_B = A_B^{-1}\mathbf{b} = \begin{bmatrix} 40 \\ 20 \\ 12 \end{bmatrix}$

Basic feasible solution $\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 40 \\ 20 \\ 12 \end{bmatrix}$

If some of $\mathbf{x}_B = \mathbf{0}$, then $\begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix}$ is called degenerate basic solution

If some of $\mathbf{x}_B = \mathbf{0}$ and $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ 0 \end{bmatrix} \geq \mathbf{0}$, then $\mathbf{x}$ is called a degenerate basic feasible solution

Geometrically: $\mathbf{x}$ is a BFS if and only if $\mathbf{x}$ is an extreme point of $\{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$

How many BFSs:

Given $A \in \mathbb{R}^{nxn},\ m \leq n,\ rank(A) = m$ we can tell that

$$\#BFS \leq \binom{n}{m} = \frac{n!}{m!(n-m)!}$$

Theorem 7.1 (Fundamental Theorem of L.P.) *For any standard LP problem:*

1. If there exists a feasible solution (i.e. $\mathbf{\Omega} \neq \emptyset$), then there exists a basic feasible solution.

2. If these exists an optimal feasible solution (not necessarily unique), then there exists an optimal basic feasible solution.

To solve $A\mathbf{x} = \mathbf{b}$ we must solve two issues:

1. Suppose $\mathbf{x}_0$ is a BFS. Find δ such that $\mathbf{x}_k + \epsilon\delta$ is a BFS and that $\mathbf{C}^T(\mathbf{x}_k + \epsilon\delta) \leq \mathbf{C}^T(\mathbf{x}_k)$
2. We need an easy to check stopping criteria

Simplex method:

Assume $\mathbf{x}_0 = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix}$ is a BFS where $B \subset \{1, 2, \ldots, n\}$, the number of elements in B is m, and $\mathbf{x}_B = A_B^{-1}b$. First we shall construct an 'edge' direction; we will call this direction δ_j and define it as: For $j \in B^C$

$$\delta_j \in \mathbb{R}^m \text{ such that } \forall i \in \{1, 2, ..., n\}, \text{ where } \delta_j(i) = \begin{cases} -A_B^{-1}\mathbf{a_j}(i) & \text{if } \ i \in B \\ 1 & \text{if } \ i = j \\ 0 & \text{else} \end{cases}$$

Since $j \in B^C$, we will have multiple possible δ_j at each step, although only one will be chosen. Which depends on what the value of j is.

$$A(\mathbf{x}_0 + \epsilon\delta^j) = A\mathbf{x}_0 + A\epsilon\delta^j = \mathbf{b} + \mathbf{0} = \mathbf{b}$$

Note: this property assures us that our succeeding solutions will always satisfy $A\mathbf{x}_k = \mathbf{b}$

7.4 Lecture 17

Recall that our representation of the simplex method is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon\delta_j, \ j \in B^C$$

$$\delta_j \in \mathbb{R}^m \text{ such that } \forall i \in \{1, 2, ..., n\}, \text{ where } \delta_j(i) = \begin{cases} -(A_B^{-1}\mathbf{a_j})(i) & \text{if } \ i \in B \\ 1 & \text{if } \ i = j \\ 0 & \text{else} \end{cases}$$

Also recall that: $A\delta_j = \mathbf{0}, \ \forall j \in B^C$. Thus

$$A\mathbf{x}_{k+1} = A(\mathbf{x}_k + \epsilon\delta_j) = A\mathbf{x}_k + \mathbf{0} = \mathbf{b}$$

In other words, if $\mathbf{x}_k$ is a solution of $A\mathbf{x} = \mathbf{b}$, then $\mathbf{x}_{k+1}$ is also a solution.

Now we have two questions to answer:

1. How do we choose j?
2. How do ϵ such that $\mathbf{C}^T\mathbf{x}_{k+1} < \mathbf{C}^T\mathbf{x}_k$

Deriving $\bar{\mathbf{C}}$:

We require both

$$x_{k+1} = \mathbf{x}_k + \epsilon\delta_j \geq \mathbf{0}$$

$$\mathbf{C}^T\mathbf{x}_k \geq \mathbf{C}^T\mathbf{x}_{k+1} = \mathbf{C}^T(\mathbf{x}_k + \epsilon\delta_j) = \mathbf{C}^T\mathbf{x}_k + \epsilon\mathbf{C}^T\delta_j$$

In other words, since $\epsilon \geq 0$, we must have $\mathbf{C}^T\delta_j \leq \mathbf{0}$.

$\mathbf{C}^T\delta_j$ is known as the **reduced cost**

Using his notation we have that

$$\mathbf{C}^T\delta_j = \mathbf{C}^T \begin{bmatrix} -A_B^{-1}\mathbf{a}_j \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \mathbf{C}_j - \mathbf{C}_B^T A_B^{-1}$$

$$j_0 = argmin_{j \in B^C}\mathbf{C}^T\delta_j$$

Thus, we can define $\bar{\mathbf{C}}$ as follows:

$$\bar{\mathbf{C}} = \mathbf{C}^T - \mathbf{C}_B^T A_B^{-1} A$$

$$\bar{\mathbf{C}}(j) = \begin{cases} \mathbf{C}_j - \mathbf{C}_B^T A_B^{-1}\mathbf{a}_j & \text{if } j \in B^C \\ 0 & \text{if } j \in B \end{cases}$$

Theorem 7.2 *If* $\mathbf{x} = \begin{bmatrix} \mathbf{x}_B \\ \mathbf{0} \end{bmatrix}$ *is BFS, then:* $\mathbf{x}$ *is optimal if and only if* $\bar{\mathbf{C}} \geq \mathbf{0}$.
In other words, if it is not possible for $\bar{\mathbf{C}}$ *to be negative, then* $\mathbf{x}$ *is optimal*

Deriving ϵ:

Suppose that for some j, $\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon\delta_j$, where $\epsilon > 0$. We require that $\mathbf{x}_{k+1} \geq \mathbf{0}$. From this we have two cases:

Case 1: $\delta_j \geq \mathbf{0}$. Then for any $\epsilon > 0$ we are guaranteed that both $A\mathbf{x}_{k+1} = \mathbf{b}$ and $\mathbf{x}_{k+1} \geq \mathbf{0}$. In this situation, the objective function has no lower bound as we can choose an arbitrarily large epsilon which means there is no optimal solution as we can always decrease it further. This is known as an unbounded LP. One example of this is:

$$\min -x_1,\ x_1 \geq 0$$

Case 2: $\delta_j \not\geq \mathbf{0}$. In other words, $\exists i$ such that $\delta_j(i) < \mathbf{0}$.

$$\text{Using his notation: } \mathbf{x}_k + \epsilon\delta_j = \begin{bmatrix} \mathbf{x}_B(1) \\ \vdots \\ \mathbf{x}_B(m) \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \epsilon \begin{bmatrix} \delta_j(1) \\ \vdots \\ \delta_j(m) \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

We want $\mathbf{x}_k + 1$ to have m non-zero entries (because this is required for $\mathbf{x}_{k+1}$ to be a BFS). As δ_j will always have a 1 at the j-th index, and $\epsilon > 0$, we have that δ_j will have $m + 1$ non-zero entries. Thus, we must choose an epsilon such that *exactly one* entry of δ_j will cancel out with one entry of $\mathbf{x}_k$.

Ratio Test:

$$\epsilon = min_i \left\{ \frac{\mathbf{x}_B(i)}{-\delta_j(i)} \text{ such that } i \in B,\ \delta_j(i) < 0 \right\}$$

The i that satisfies the above will be denoted as i_0

Remarks:

1. x_{k+1} is a BFS.
2. The number of nonzero elements in $\mathbf{x}_{k+1} = m$. Therefore, in $\mathbf{x}_{k+1}$, our B is redefined as:

$$B = B \cup \{j\} \backslash \{i_0\}$$

Thus $\mathbf{x}_{k+1}$ is the BFS corresponding to $B \cup \{j\} \backslash \{i_0\}$

Simplex Summary:

1. Choose $\mathbf{x}_0$ that is a BFS
2. Check if $\bar{\mathbf{C}} = \mathbf{C}^T - \mathbf{C}_B^T A_B^{-1} A \geq \mathbf{0}$. If $\bar{\mathbf{C}} \geq 0$ then stop, $\mathbf{x}$ is an optimizer.
3. Otherwise: Choose j such that $j = arg\min \bar{\mathbf{C}}(j)$
4. Construct δ_j
5. If $\delta_j \geq \mathbf{0} \rightarrow$ LP is unbounded

6. If $\delta_j \ngeq \mathbf{0}$ then $\epsilon = min_i \left\{ \frac{\mathbf{x}_B(i)}{-\delta_j(i)} \text{ such that } i \in B,\ \delta_j(i) < 0 \right\}$

7. Update $B = B \cup \{j\} \backslash i_0$

8. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon \delta_j$

Example:

$$\max\ 2x_1 + 5x_2 \ \text{ such that:}$$

$$x_1 \leq 4$$

$$x_2 \leq 6$$

$$x_1 + x_2 \leq 8$$

$$x_1, x_2 \geq 0$$

First we make this into standard form:

$$\min\ -2x_1 - 5x_2 \ \text{ such that:}$$

$$x_1 + x_3 = 4$$

$$x_2 + x_4 = 6$$

$$x_1 + x_2 + x_5 = 8$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

Now we have:

$$A = \begin{bmatrix} 1,0,1,0,0 \\ 0,1,0,1,0 \\ 1,1,0,0,1 \end{bmatrix},\ \mathbf{b} = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix},\ \mathbf{C} = \begin{bmatrix} -2 \\ -5 \\ 0 \\ 0 \\ 0 \end{bmatrix},\ \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

$$\min\ \mathbf{C}^T\mathbf{x} \ \text{ such that } \ A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}$$

Initialization: Fix a basis B and a BFS

$$B = \{3,4,5\},\ A_B = \begin{bmatrix} 1,0,0 \\ 0,1,0 \\ 0,0,1 \end{bmatrix},\ \mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 6 \\ 8 \end{bmatrix}$$

Now, check the reduced cost:

$$\bar{\mathbf{C}} = \mathbf{C}^T \mathbf{C}_B^T A_B^{-1} A = \begin{bmatrix} -2, -5, 0, 0, 0 \end{bmatrix}$$

$$j = arg\min_j \bar{\mathbf{C}}(j) = 2$$

This means we will later add 2 into our basis

$$\delta_j = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}$$

Now, use the ratio test

$$\epsilon = \min_i \left\{ \frac{\mathbf{x}_k(i)}{-\delta_j(i)} \text{ such that } i = 4, 5 \right\} = \min_i \left\{ \frac{6}{1}, \frac{8}{1} \right\} = 6$$

Next we update B:

$$B = \{3, 4, 5\} \cup \{2\} \backslash \{4\} = \{2, 3, 5\}$$

$$\mathbf{x}_1 = \mathbf{x}_0 + \epsilon\delta_j = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 6 \\ 8 \end{bmatrix} + 6 \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \\ 4 \\ 0 \\ 2 \end{bmatrix}$$

If you check $\mathbf{x}_1$, you will find that it is also a BFS corresponding to $\{2, 3, 5\}$. You can then repeat these steps.

7.5 Lecture 18

Pseudo code of the simplex method:

1. Convert the problem to standard form.
   - $A \in \mathbb{R}^{mxn}$
   - $Rank(A) = m$
2. Initialization:
   a. Find an index set B that gives some basic solution and its corresponding BFS, denoted x_0.
   b. Compute $\bar{\mathbf{C}}^T = \mathbf{C}^T - \mathbf{C}_B^T A_B^{-1} A$
3. Loop while $\bar{\mathbf{C}} \ngeq \mathbf{0}$

   $j = argmin\{\bar{\mathbf{C}}_j\}$

$$\delta_j(i) = \begin{cases} -A_B^{-1}\mathbf{a_j}(i) & \text{if } i \in B \\ 1 & \text{if } i = j \\ 0 & \text{else} \end{cases}$$

If $\delta_j \geq \mathbf{0}$ The problem is unbounded, otherwise:

$$\epsilon = \min \left\{ \frac{\mathbf{x}(i)}{-\delta_j(i)} \mid i \in B,\ \delta_j(i) < 0 \right\}, \text{ achieved at } i_0$$

$$B = B \cup \{j\} \backslash \{i_0\}$$

$$\bar{\mathbf{C}}^T = \mathbf{C}^T - \mathbf{C}_B^T A_B^{-1} A$$

$$\mathbf{x} = \mathbf{x} + \epsilon \delta_j$$

Remarks:

If $\mathbf{x}$ is a degenerate BFS (ie, some of $x_B = 0$), then the step size ϵ could be 0. Thus no update for x. We can compensate for this by using an anti-cycle scheme. One of the more famous schemes is called Bland's rule.

How to choose an initial BFS (using a two phase method):

Recall the standard form: $\min \mathbf{C}^T\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}, \mathbf{x} >= \mathbf{0}$. We will call this problem (*)

Phase 1: Introduce an artificial problem

$$\min y_1 + y_2 + ... + y_m$$

$$\begin{bmatrix} A, I \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mathbf{b}. \text{ We will call this equation (**)}$$

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \geq \mathbf{0}$$

Facts:

1. We must ensure $\mathbf{b} \geq \mathbf{0}$ to ensure $\begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix}$ can be a BFS.
2. (*) has a BFS $\iff$ the associated (**) problem has an optimized feasible solution with objective function value 0.
3. $\begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix}$ is a (not necessarily optimal) solution of $\begin{bmatrix} A, I \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \mathbf{b}$

Ensuring $\mathbf{b} \geq \mathbf{0}$ example:

$$\begin{aligned} 5x_1 + 6x_2 + x_3 = 2 &\ \rightarrow\ 5x_1 + 6x_2 + x_3 = 2 \\ x_1 + 8x_2 + x_4 = -5 &\ \rightarrow\ -x_1 - 8x_2 - x_4 = 5 \\ x_1, x_2, x_3, x_4 \geq 0 &\ \rightarrow\ x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

We are negating both sides of the second constraint in order to ensure that all elements of $\mathbf{b}$ are positive so that $\begin{bmatrix}\mathbf{0}\\ \mathbf{b}\end{bmatrix}$ can be guaranteed to be a BFS.

If we apply the simplex method to (**) using the initial guess $\mathbf{x} = \begin{bmatrix}\mathbf{0}\\ \mathbf{b}\end{bmatrix}$, this will lead to an optimizer of (**). The first n entries of this function are a BFS of (*).

Phase 2: Use the first n entries of the solution to (**) and the corresponding B set as an initial guess for (*). Then apply the simplex method.

Two Phase Example: $\min x_1 + x_3 + 5x_4$

$$\begin{aligned} 5x_1 + 6x_2 + x_3 = 2 &\rightarrow 5x_1 + 6x_2 + x_3 = 2 \\ x_1 + 8x_2 + x_4 = -5 &\rightarrow -x_1 - 8x_2 - x_4 = 5 \\ x_1, x_2, x_3, x_4 \geq 0 &\rightarrow x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Phase 1: Introduce an artificial problem

$$\min y_1 + y_2$$
$$5x_1 + 6x_2 + x_3 + y_1 = 2$$
$$x_1 + 8x_2 + x_4 + y_2 = -5$$
$$x_1, x_2, x_3, x_4, y_1, y_2 \geq 0$$
$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\2\\5\end{bmatrix}$$

Then, apply the simplex method.

Phase 2:

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} = \begin{bmatrix}x_1\\x_2\\x_3\\x_4\\0\\0\end{bmatrix}$$

Here the number of zeros in $\{x_1, x_2, x_3, x_4\} = m = 2$. B is the index set of nonzero values in $\{x_1, x_2, x_3, x_4\}$. Finally, $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$ serves as the initial values of the original problem

Chapter 8

Fundamentals of non-linear programming

8.1 Lecture 18

$$\min_{x \in \mathbb{R}^n} F(x) \text{ such that:}$$
$$h_i(x) = 0,\ i = 1, 2, ...m$$
$$g_j(x) \leq 0,\ i = 1, 2, ...p$$

Generally: no effective way to solve this problem

Definitions:

- Feasible points:

$$\mathbf{x} \in \mathbb{R}^n \text{ such that:}$$
$$h_i(\mathbf{x}) = 0,\ i = 1, 2,\ \dots\ m$$
$$g_j(\mathbf{x}) \leq 0,\ j = 1, 2,\ \dots\ p$$

- Standard Form:

$$F : \mathbb{R}^n \to \mathbb{R} \quad \vec{h} : \mathbb{R}^n \to \mathbb{R}^m \quad \vec{g} : \mathbb{R}^n \to \mathbb{R}^p$$

$$\min_{\mathbf{x} \in \mathbb{R}^n} F(\mathbf{x}) \text{ such that:}$$
$$\mathbf{h}(\mathbf{x}) = \mathbf{0},\ \mathbf{g}(\mathbf{x}) \leq \mathbf{0}$$

Example:

$$\min(x_1 - 1)^2 + x_2 - 2 \text{ such that:}$$
$$x_2 - x_1 = \mathbf{0},\ x_1^2 + x_2 \leq 2$$

First: equality constraints

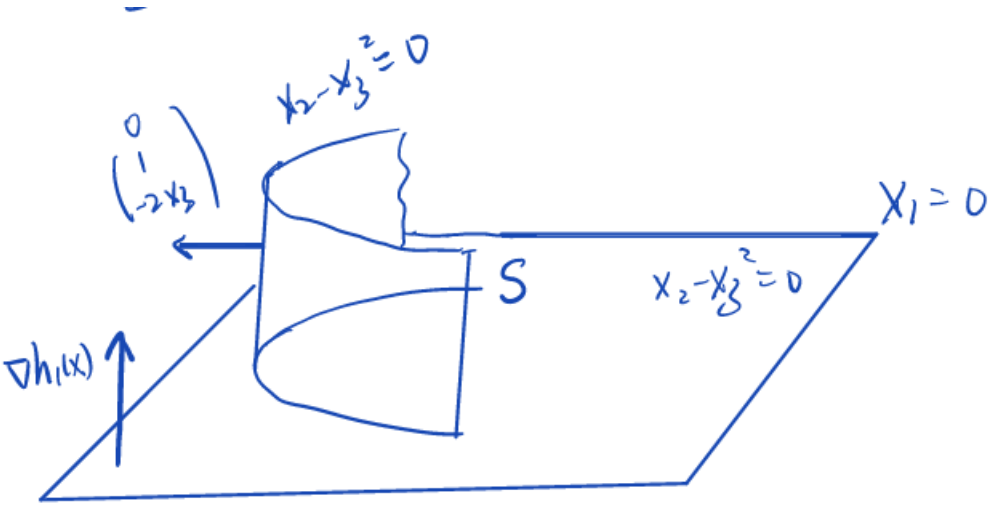
$$min_{\mathbf{x}\in\mathbb{R}^n} F(\mathbf{x}) \text{ such that:}$$
$$h_1(\mathbf{x}) = 0, h_2(\mathbf{x}) = 0, \ldots, h_m(\mathbf{x}) = 0$$
$$\text{Equivalently } \mathbf{h}(\mathbf{x}) = \mathbf{0}$$

Assume $h_i \in C^1$ for $i = 1, ..., m$

Regular points: Any $\mathbf{x}$ that satisfies $h(\mathbf{x}) = \mathbf{0}$ where $\{\nabla h_1(\mathbf{x}), ..., \nabla h_m(\mathbf{x})\}$ are linearly independent. Equivalently $D\vec{h}(\mathbf{x}) = \begin{bmatrix} \nabla h_1^T(\mathbf{x}) \\ ... \\ \nabla h_m^T(\mathbf{x}) \end{bmatrix} = \frac{\partial h_i}{\partial \mathbf{x}_j}$. This is known as the Jacobian Matrix). Here, $D\vec{h}(\mathbf{x})$ is a full rank matrix.

Example

$$h_1(\mathbf{x}) = x_1 \; h_2(\mathbf{x}) = x_2 - x_3^2$$
$$\text{Feasible points } = \{\mathbf{x} \in \mathbb{R}^3 \mid h_1(\mathbf{x}) = 0, \; h_2(\mathbf{x}) = 0 \}$$



$$\nabla h_1(\mathbf{x}) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \; \nabla h_2(\mathbf{x}) = \begin{bmatrix} 0 \\ 1 \\ -2x_3 \end{bmatrix}$$

Note: in this case, all $\mathbf{x}$ that satisfy $h(\mathbf{x}) = \mathbf{0}$ will be regular points because no matter our choice of x_3, the gradients of $h_1(\mathbf{x})$ and $h_2(\mathbf{x})$ will remain linearly independent

$$\mathbf{S} = \{ \mathbf{x} \mid h_1(\mathbf{x}) = 0, \; h_2(\mathbf{x}) = 0 \}$$

$$dim(\mathbf{S}) = 3 - 2 = 1$$

If $\mathbf{x}$ is a regular point, and $S = \{ \mathbf{x} \mid h_i(\mathbf{x}) = 0 \}$, for $0 \leq i \leq m$ then $dim(\mathbf{S}(\mathbf{x})) = n - m$