Nonlinear Programming:

Zero-order, First-order, and Second-order Optimization Methods

Assoc. Prof. Karl Ezra Pilario, Ph.D.

Process Systems Engineering Laboratory
Department of Chemical Engineering
University of the Philippines Diliman

Outline

- Introduction to NLP
- Necessary and Sufficient Conditions for Optimality
- Convex Programming
- Methods for Solving NLP
    - One-Dimensional, Unconstrained NLP
    - Multivariable, Unconstrained NLP
    - Zero-order, First-order, Second-order Methods
    - Constrained NLP

Nonlinear Programming

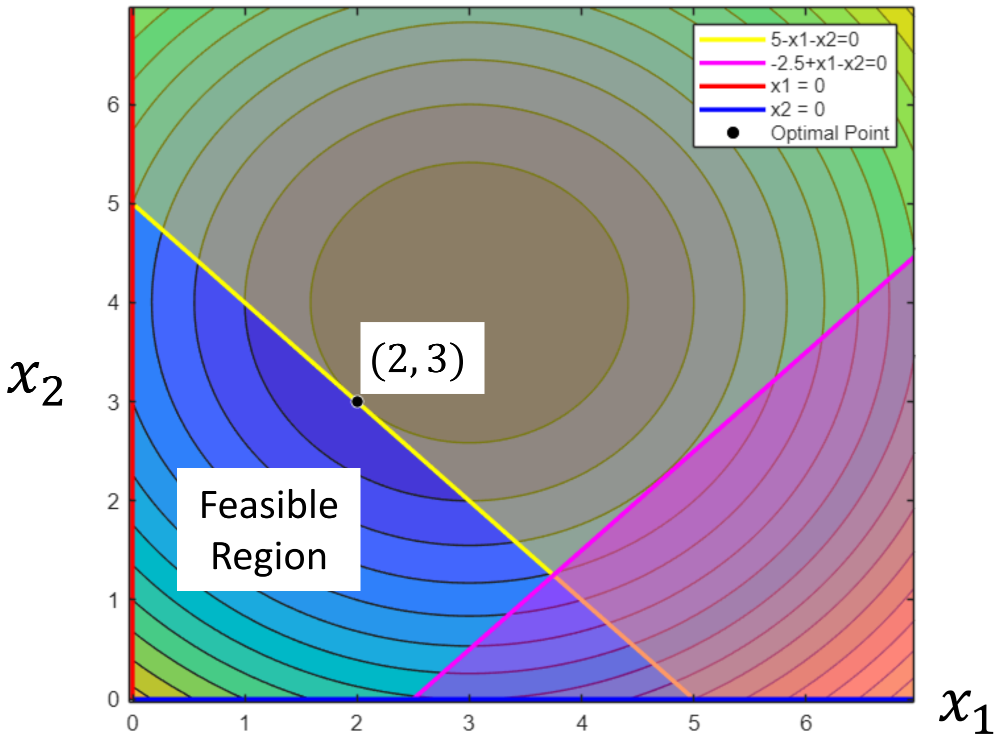
- The general form of a nonlinear program (NLP) is:

$$
\begin{aligned}
&\text{Minimize:} && f(\boldsymbol{x}) && \\
&\text{Subject to:} && h_i(\boldsymbol{x}) = 0 && i = 1, 2, \dots, l \\
& && g_j(\boldsymbol{x}) \le 0 && j = 1, 2, \dots, m \\
&\text{and} && x_k^L \le x_k \le x_k^U && k = 1, 2, \dots, n
\end{aligned}
$$

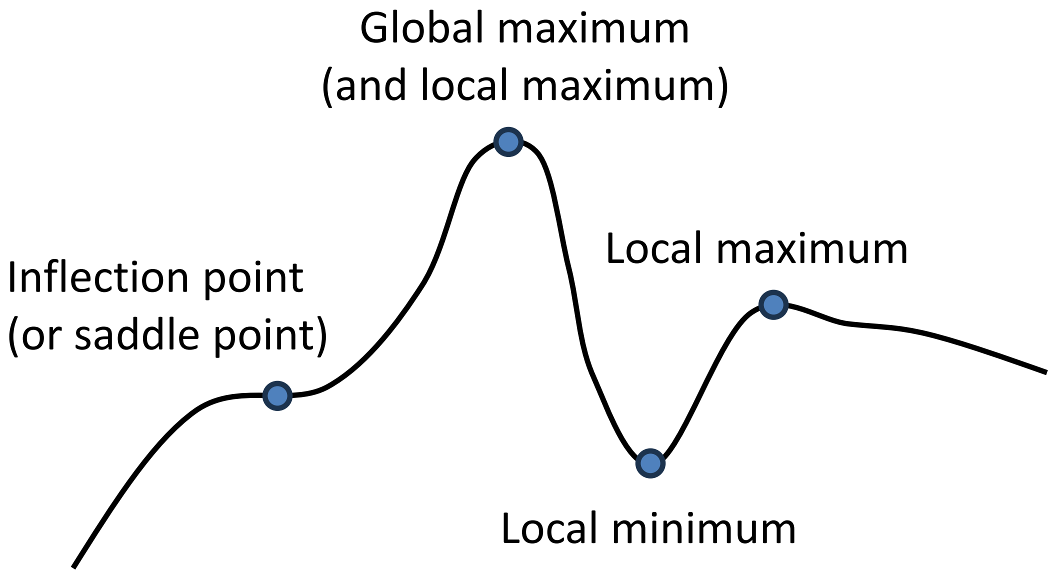
- At least one of $f(\boldsymbol{x})$, $g_j(\boldsymbol{x})$, or $h_i(\boldsymbol{x})$ is *nonlinear*.
- x_k^L and x_k^U are lower and upper bounds on x_k.

Example:

$$
\begin{aligned}
&\text{Minimize:} && f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 4)^2 \\
&\text{Subject to:} && x_1, x_2 \ge 0 \\
& && 5 - x_1 - x_2 \ge 0 \\
& && -2.5 + x_1 - x_2 \ge 0
\end{aligned}
$$



Nonlinear Programming



Local minimum

A point $\boldsymbol{x}^*$ such that no other point in the *vicinity* of $\boldsymbol{x}^*$ yields a value of $f(\boldsymbol{x})$ less than $f(\boldsymbol{x}^*)$:

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$$

Global minimum

A point $\boldsymbol{x}^*$ such that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ holds for any $\boldsymbol{x}$ in the n-dimensional space of $\boldsymbol{x}$.

Conditions for Optimality in Unconstrained NLPs

A point $\boldsymbol{x}^*$ is an **optimal solution** to an unconstrained NLP:

- ONLY IF $f(\boldsymbol{x})$ is twice differentiable at $\boldsymbol{x}^*$. **NECESSARY**
- ONLY IF $\nabla f(\boldsymbol{x}^*) = 0$ or $\boldsymbol{x}^*$ is a stationary point. **NECESSARY**
- IF $\boldsymbol{H}(\boldsymbol{x}^*)$ is *positive-definite* for a minimum to exist, and *negative-definite* for a maximum to exist. **SUFFICIENT**

$\boldsymbol{H}(\boldsymbol{x})$ is the Hessian matrix.

$$\mathbf{H}_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

$\boldsymbol{H}(\boldsymbol{x})$ is…	If all the eigenvalues of $\boldsymbol{H}(\boldsymbol{x})$ are…
Positive-definite	> 0
Positive-semidefinite	≥ 0
Negative-definite	< 0
Negative-semidefinite	≤ 0

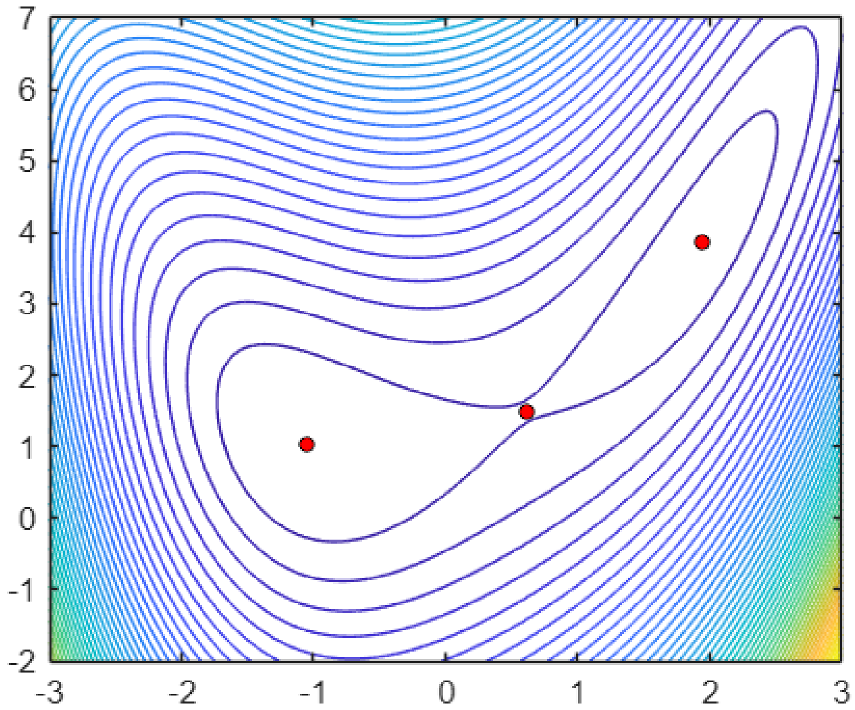
Nonlinear Programming

Example:

$$f(x_1, x_2) = 4 + 4.5x_1 - 4x_2 + x_1^2 + 2x_2^2 - 2x_1x_2 + x_1^4 - 2x_1^2x_2$$

This function has three stationary points $(1.941, 3.854)$, $(-1.053, 1.028)$, and $(0.6117, 1.4929)$. Evaluate the optimality at these points.

$$\boldsymbol{H}(x_1, x_2) = \begin{bmatrix} 2 + 12x_1^2 - 4x_2 & -2 - 4x_1 \\ -2 - 4x_1 & 4 \end{bmatrix}$$



Stationary point	$f(\boldsymbol{x})$	$f'(\boldsymbol{x})$	$\boldsymbol{H}(\boldsymbol{x})$	Eigenvalues		Evaluation
$(1.941, 3.854)$	0.9856	0.00	$\begin{bmatrix} 31.79 & -9.76 \\ -9.76 & 4.00 \end{bmatrix}$	0.9128	34.8810	**Local minimum**
$(-1.053, 1.028)$	-0.5134	0.00	$\begin{bmatrix} 11.19 & 2.21 \\ 2.21 & 4.00 \end{bmatrix}$	3.3743	11.8194	**Local minimum**
$(0.6117, 1.4929)$	2.8091	0.00	$\begin{bmatrix} 0.52 & -4.45 \\ -4.45 & 4.00 \end{bmatrix}$	-2.5161	7.0346	**Saddle point**

Nonlinear Programming

Unlike an LP, solving an NLP is hard because a minimum is **not guaranteed** to be the global minimum.

But if the NLP is **convex**, we have the following result.

Theorem

For an NLP whose objective function $f(\boldsymbol{x})$ is a **convex function** and each inequality constraint $g(\boldsymbol{x})$ is a convex function so that they form a **convex set**:

The local minimum of $f(\boldsymbol{x})$ is also a global minimum.

Analogously, if $f(\boldsymbol{x})$ is a **concave function** and the constraints $g(\boldsymbol{x})$ still form a **convex set**, then:

The local maximum of $f(\boldsymbol{x})$ is also a global maximum.

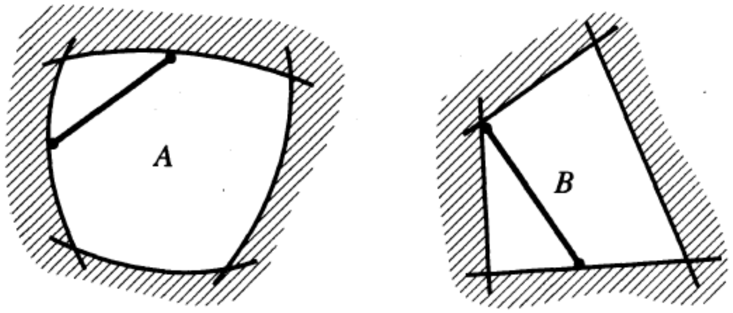
Definition: Convex Set

For every pair of points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ in a convex set, the point $\boldsymbol{x}$ given by a linear combination of the two points:

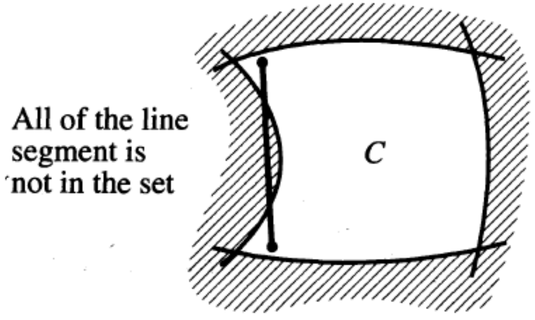
$$\boldsymbol{x} = \gamma \boldsymbol{x}_1 + (1-\gamma)\boldsymbol{x}_2, \qquad 0 \le \gamma \le 1$$

must also be in the set. Also, the intersection of any number of convex sets is a convex set.

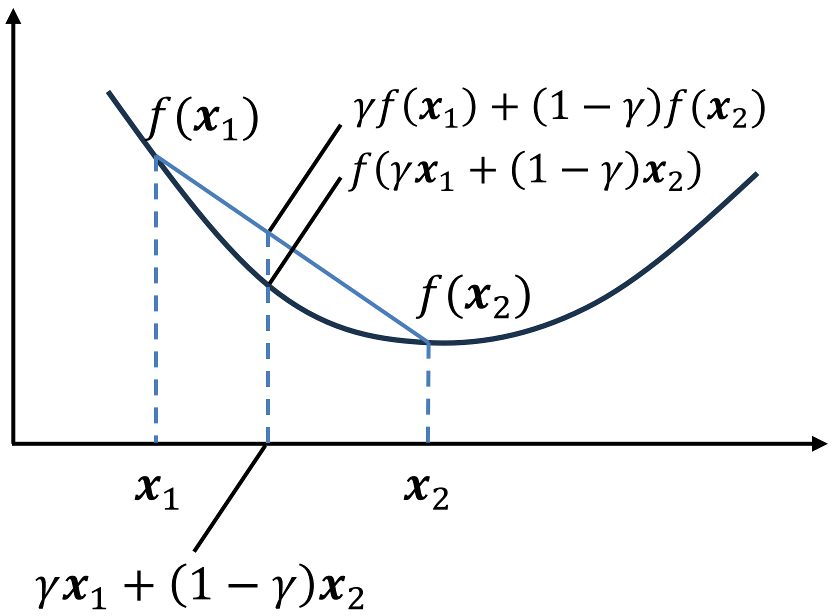
Examples of a Convex Set



Example of a Nonconvex Set



Nonlinear Programming



Definition: Convex Function

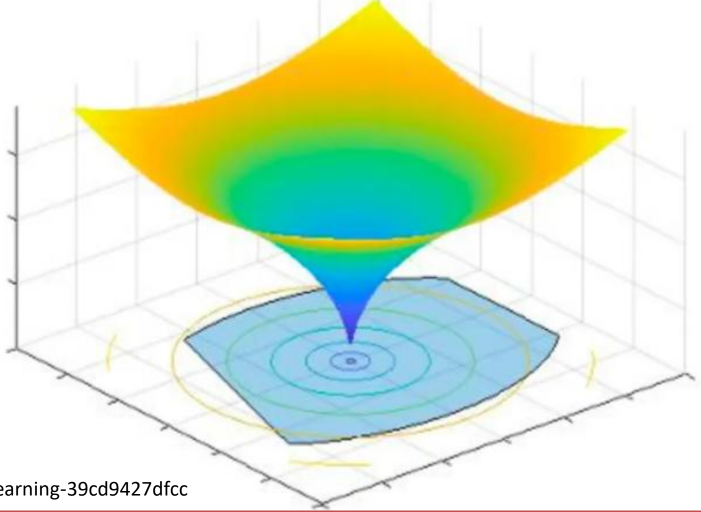
A function $f(\boldsymbol{x})$ defined on a convex set F is said to be a **convex function** if the following relation holds:

$$f(\gamma\boldsymbol{x}_1 + (1-\gamma)\boldsymbol{x}_2) \leq \gamma f(\boldsymbol{x}_1) + (1-\gamma)f(\boldsymbol{x}_2)$$

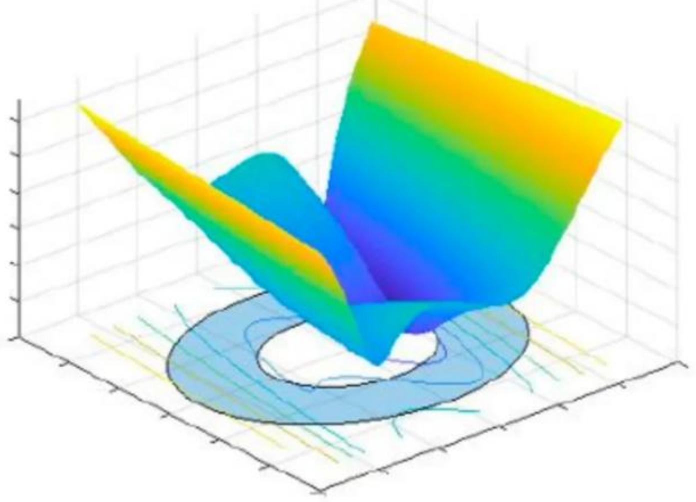
Where $0 \leq \gamma \leq 1$.

- If $f(\boldsymbol{x})$ is convex, then $-f(\boldsymbol{x})$ is concave.
- If only the inequality sign holds ($<$), the function is said to be *strictly convex*.
- If $f(\boldsymbol{x})$ is strictly convex, then $-f(\boldsymbol{x})$ is strictly concave.

A **convex** objective with **convex** constraints.



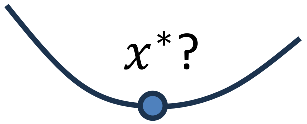
A **nonconvex** objective with **nonconvex** constraints.



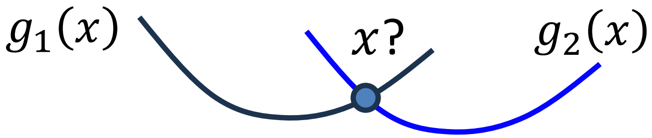
Source: https://rumn.medium.com/convex-vs-non-convex-functions-why-it-matters-in-optimization-for-machine-learning-39cd9427dfcc

One-Dimensional, Unconstrained NLP

Minimize: $f(x)$



Recall: Newton's Method for Root-Finding



Given: $f(x) = g_1(x) - g_2(x) = 0$

Initialization: One initial guess (x_0) and a Tolerance, Tol

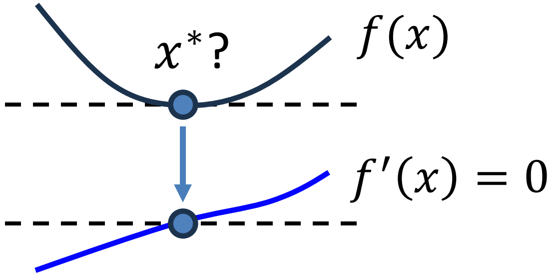
Set $x_{old} \leftarrow \text{Inf.}$
Set $x_{new} \leftarrow x_0$.
WHILE $|x_{new} - x_{old}| > Tol$:
    Assign: $x_{old} \leftarrow x_{new}$.
    Assign: $x_{new} \leftarrow x_{old} - \frac{f(x_{old})}{f'(x_{old})}$.
END WHILE
OUTPUT x_{new}

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

Newton's Method for Optimization

Minimize: $f(x)$

Find the roots: $f'(x) = 0$



Newton's Method

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

Quasi-Newton Method

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

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

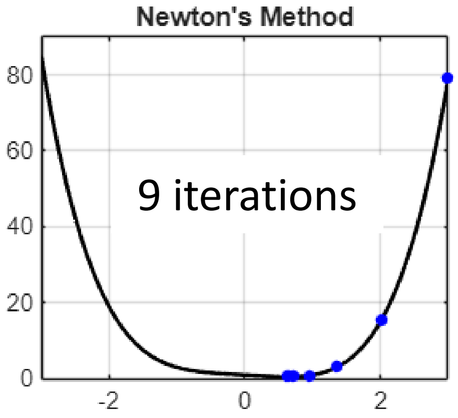
2nd Order finite difference

One-Dimensional, Unconstrained NLP

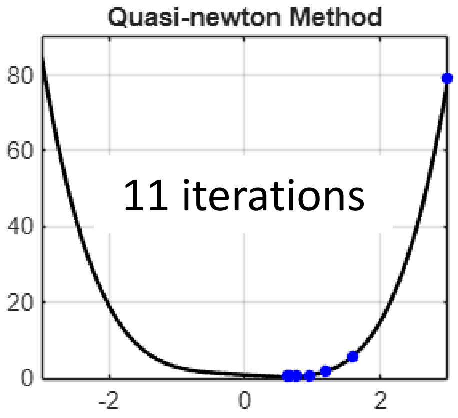
Example:

Minimize: $f(x) = x^4 - x + 1$

- Use Newton's Method starting with $x_0 = 3$.
- Use Quasi-newton method starting with $x_0 = 3$ and $x_1 = 2$.
- For both methods, iterate until the change in x is less than 10^{-7}.



$x^* = 0.63$



$x^* = 0.63$

```
f = @(x) x.^4-x+1;
df = @(x) 4*x.^3-1;
ddf = @(x) 12*x.^2;
```

```
% Use Newton's method
x0 = 3; x1 = Inf; ctr = 1;
while abs(x0-x1) > 1e-7
    x1 = x0;
    x0 = x1 - df(x1)/ddf(x1);
    ctr = ctr + 1;
end
disp(x0);
fprintf('%d\n', ctr);
```

```
% Use quasi-newton method
x0 = 3; x1 = 2; ctr = 1;
while abs(x0-x1) > 1e-7
    xn = x1 - (x0-x1)*df(x1)/(df(x0)-df(x1));
    x0 = x1; x1 = xn;
    ctr = ctr + 1;
end
disp(x0);
fprintf('%d\n', ctr);
```

Outline

- Introduction to NLP
- Necessary and Sufficient Conditions for Optimality
- Convex Programming
- Methods for Solving NLP
  - One-Dimensional, Unconstrained NLP
  - **Multivariable, Unconstrained NLP**
  - **Zero-order, First-order, Second-order Methods**
  - Constrained NLP

A Taxonomy of NLP Solvers

Zero-order Methods

Derivative-free, black-box

- Grid Search / Exhaustive Search
- Random Search
- Nelder-Mead Simplex

- Metaheuristic Search
    - Genetic Algorithms
    - Particle Swarm
    - Simulated Annealing
    - Differential Evolution
    - CMAES

- Bayesian Optimization / Surrogate-based

First-order Methods

Uses 1^{st} derivative information

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
    - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- Coordinate Descent

Second-order Methods

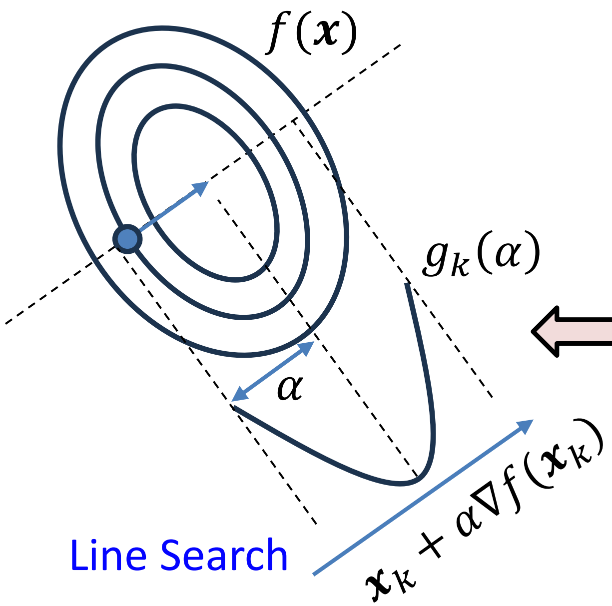
Uses 1^{st} and 2^{nd} derivative information

- *Newton's method*
- *Quasi-newton method (BFGS)*

- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)
- IPOPT

Multivariable, Unconstrained NLP

- **Steepest Descent or Gradient Descent**
- Stochastic Gradient Descent (SGD)
    - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- Coordinate Descent



Line Search

Steepest Descent / Gradient Descent

At each iteration, descend at the direction of the greatest rate of decrease in $f(\boldsymbol{x})$.

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

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent

Steps:

- Make an initial guess, $\boldsymbol{x}_0$
- Calculate the search direction $\nabla f(\boldsymbol{x}_k)$
- Calculate step size α by minimizing:

$$g_k(\alpha) = f(\boldsymbol{x}_k + \alpha\nabla f(\boldsymbol{x}_k))$$

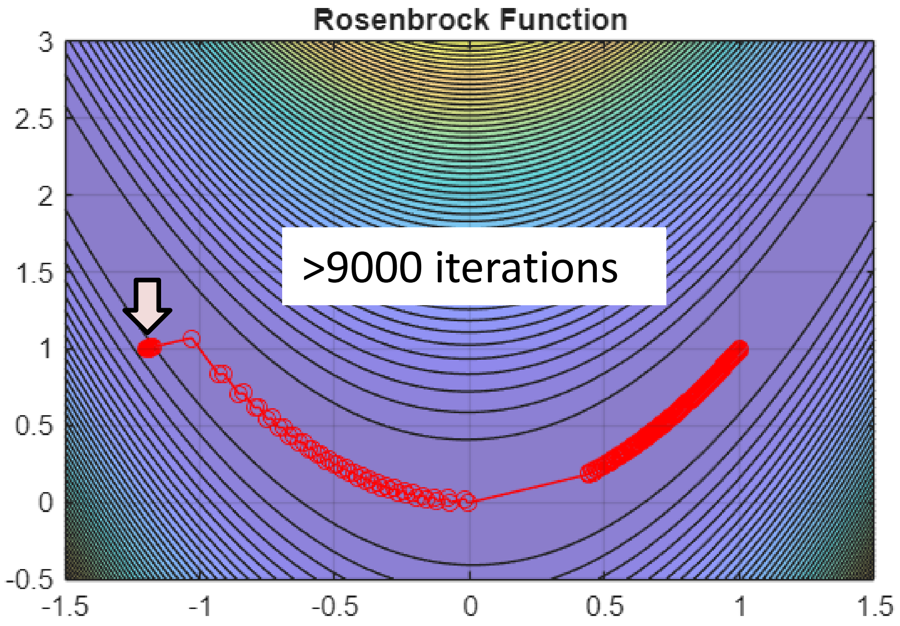
(e.g. using quasi-newton method)

- Update $\boldsymbol{x}_k$

Example:

Minimize $f(\boldsymbol{x})$ from $\boldsymbol{x}_0 = (-1.2, 1.0)$:

$$f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$



Multivariable, Unconstrained NLP

- Steepest Descent or Gradient Descent
- **Stochastic Gradient Descent (SGD)**
  - **RMSProp, Adam, etc.**
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- Coordinate Descent

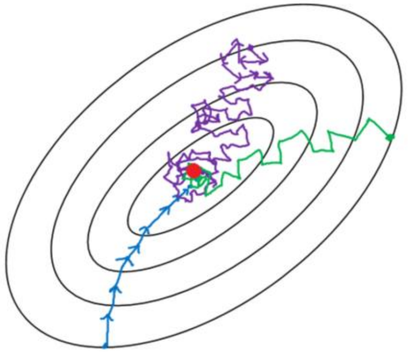
Batch vs. Stochastic Gradient Descent

Gradient descent (GD) is used to train neural nets.

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

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent

- **Batch GD** = all N data pts are used in $C(\boldsymbol{W}, \boldsymbol{b})$
- **Mini-batch GD** = a batch of $M < N$ data pts are used.
- **Stochastic GD** = only one random data pt is used.



Also, α_k is now called a *learning rate*. **RMSProp** and **Adam** are ways to decay α_k at every k.

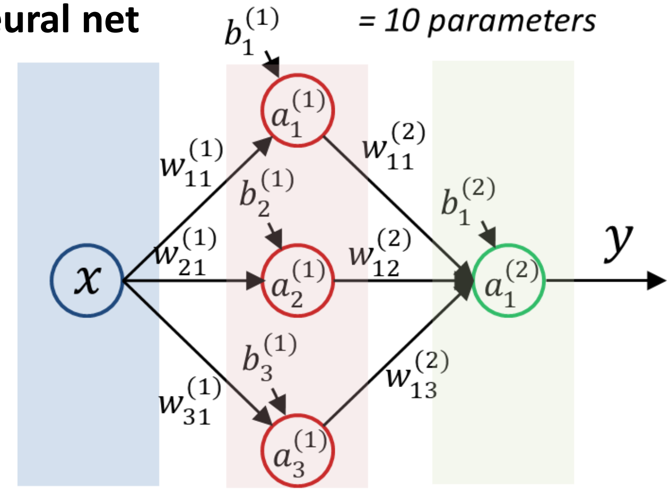
To train **neural nets** (from *machine learning*), we need to minimize $C(\boldsymbol{W}, \boldsymbol{b})$:

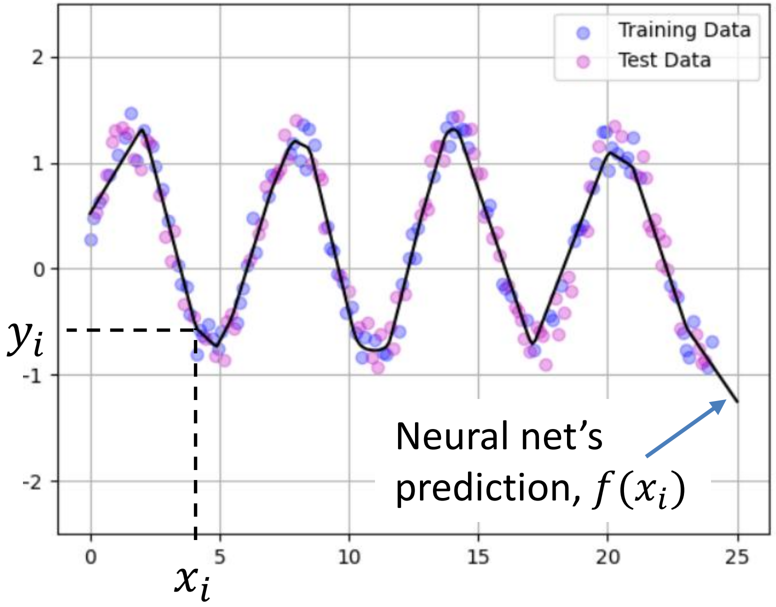
$$\min_{\boldsymbol{W},\boldsymbol{b}} \quad C(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(\boldsymbol{x}_i)\right)^2$$

$C(\boldsymbol{W}, \boldsymbol{b})$ = cost (mean squared error)
$[\boldsymbol{W};\ \boldsymbol{b}]$ = weights and biases
y_i = true values to be predicted
$f(\boldsymbol{x}_i)$ = neural net's prediction
N = no. of data points

Example: 1 hidden-layer neural net

6 weights, 4 biases = 10 parameters





Multivariable, Unconstrained NLP

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
  - RMSProp, Adam, etc.
- **Conjugate Gradient methods (e.g. Fletcher-Reeves)**
- Coordinate Descent

Conjugate Gradient Descent

This is the steepest descent update:

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

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent

This is the conjugate gradient update:

$$\boldsymbol{s}_0 = -\nabla f(\boldsymbol{x}_0) \quad \text{(search direction)}$$

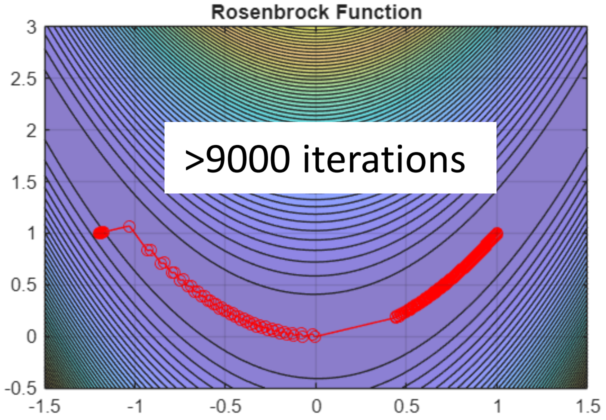
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$$

$$\boldsymbol{s}_{k+1} = -\nabla f(\boldsymbol{x}_{k+1}) + \boldsymbol{s}_k \frac{\nabla f(\boldsymbol{x}_{k+1})^T \nabla f(\boldsymbol{x}_{k+1})}{\nabla f(\boldsymbol{x}_k)^T \nabla f(\boldsymbol{x}_k)}$$

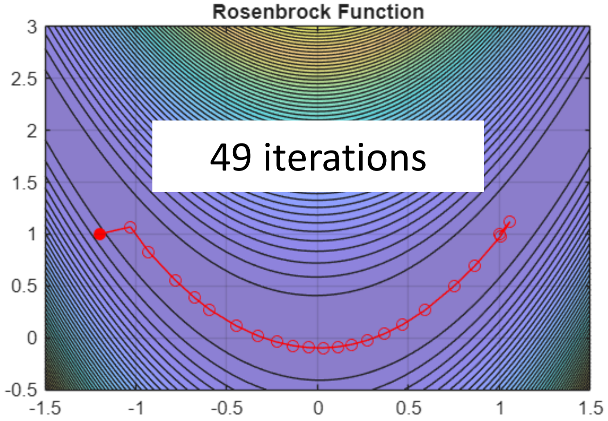
Example:

Minimize $f(\boldsymbol{x})$ from $\boldsymbol{x}_0 = (-1.2, 1.0)$: $f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

Steepest Descent



Conjugate Gradient Descent



- New search directions are *conjugate* to previous ones.
- Prevents “zigzagging”.
- Prevents undoing the progress made in previous steps.
- Allows convergence in at most N steps for an N-dimensional quadratic function.

Multivariable, Unconstrained NLP

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
  - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- **Coordinate Descent**

Coordinate Descent

This is the steepest descent update:

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

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent

Like steepest descent, but the search direction is aligned to one coordinate at a time.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \frac{\partial f(\boldsymbol{x}_k)}{\partial x_i}$$

What's the difference?

$$k = 1, \quad 2, \quad 3, \ldots$$

$$\nabla f(\boldsymbol{x}_k) \quad \begin{bmatrix} 2 \\ -1 \\ 3 \end{bmatrix}, \begin{bmatrix} -4 \\ 2 \\ 0 \end{bmatrix}, \begin{bmatrix} -2 \\ -5 \\ 1 \end{bmatrix} \ldots$$

$$\frac{\partial f(\boldsymbol{x}_k)}{\partial x_i} \quad \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -2 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 4 \end{bmatrix} \ldots$$

Steps:
- Make an initial guess, $\boldsymbol{x}_0$
- Calculate the search direction $\nabla f(\boldsymbol{x}_k)$
- Calculate step size α by minimizing:

$$g_k(\alpha) = f(\boldsymbol{x}_k + \alpha \nabla f(\boldsymbol{x}_k))$$

(e.g. using quasi-newton method)
- Update $\boldsymbol{x}_k$

Steps:
- Make an initial guess, $\boldsymbol{x}_0$
- Choose any coordinate among x_i.
- Calculate step size α by minimizing:

$$g_k(\alpha) = f\left(\boldsymbol{x}_k + \alpha \frac{\partial f(\boldsymbol{x}_k)}{\partial x_i}\right)$$

(e.g. using quasi-newton method)
- Update $\boldsymbol{x}_k$

Multivariable, Unconstrained NLP

- Steepest Descent or Gradient Descent
- Stochastic Gradient Descent (SGD)
    - RMSProp, Adam, etc.
- Conjugate Gradient methods (e.g. Fletcher-Reeves)
- **Coordinate Descent**

Coordinate Descent

This is the steepest descent update:

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

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent

Like steepest descent, but the search direction is aligned to one coordinate at a time.

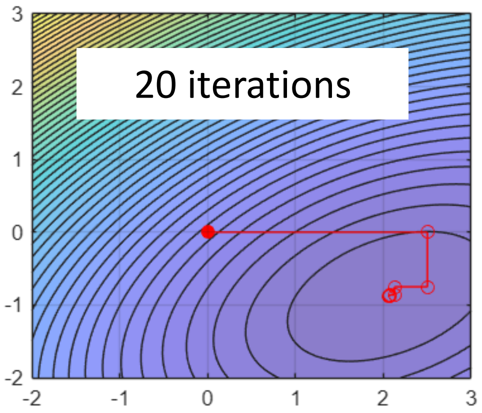
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \frac{\partial f(\boldsymbol{x}_k)}{\partial x_i}$$

Example:

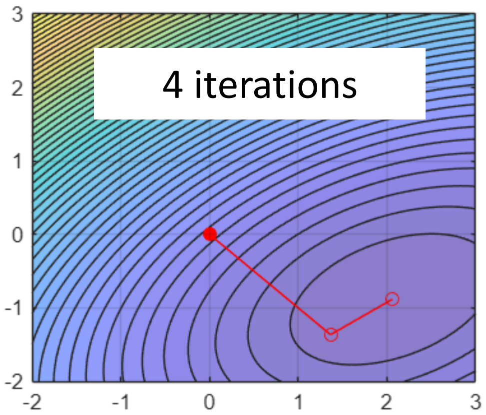
Minimize $f(\boldsymbol{x})$ from $\boldsymbol{x}_0 = (0.0, 0.0)$:

$$f(\boldsymbol{x}) = 0.06(x_1^2 - x_1 x_2) + 0.1x_2^2 - 0.3(x_1 - x_2)$$

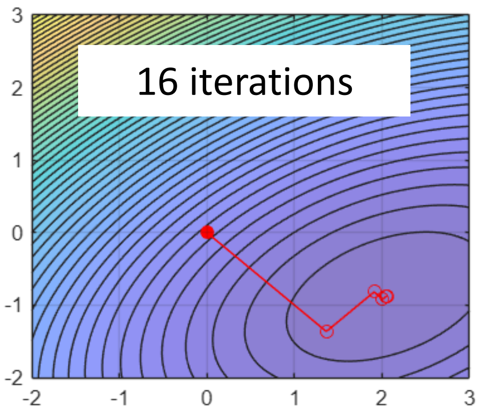
Coordinate Descent



Conjugate Gradient Descent



Steepest Descent



Multivariable, Unconstrained NLP

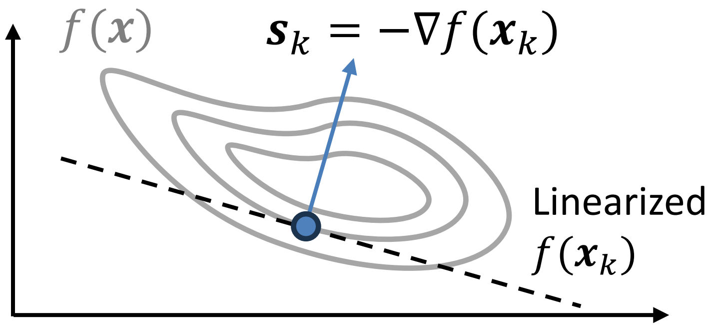
- **Newton's method**
- **Quasi-newton Method (BFGS)**
- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)
- IPOPT

Multivariable Quasi-Newton Methods

This is the steepest descent update: ⇨ This is the Newton update:

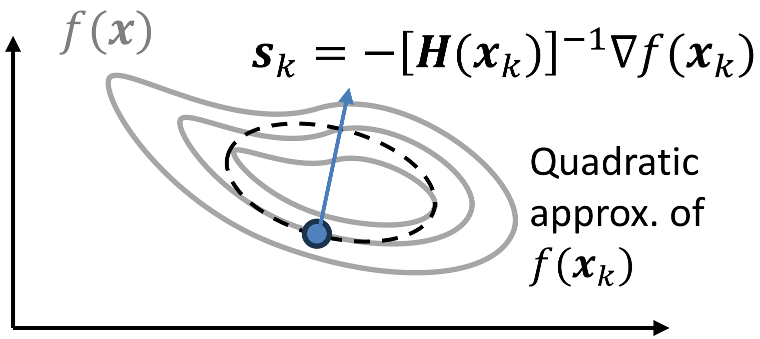
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent



$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k [\boldsymbol{H}(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent
$\boldsymbol{H}(x_k)$ = Hessian matrix



Multivariable, Unconstrained NLP

- **Newton's method**
- **Quasi-newton Method (BFGS)**
- Levenberg-Marquardt
- Trust Region Newton Methods
- Active Set and Interior-Point Methods
- Sequential Quadratic Programming (SQP)

Multivariable Quasi-Newton Methods

This is the Newton update:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k[\boldsymbol{H}(\boldsymbol{x}_k)]^{-1}\nabla f(\boldsymbol{x}_k)$$

α_k = step size at the kth iteration
$\nabla f(\boldsymbol{x}_k)$ = direction of steepest descent
$\boldsymbol{H}(x_k)$ = Hessian matrix

Quasi-Newton: **BFGS Algorithm**

(Broyden-Fletcher-Goldfarb-Shanno)

Steps:

- Make an initial guess, $\boldsymbol{x}_0$, and initial guess of inverse Hessian $\boldsymbol{\Gamma}_0 = \boldsymbol{I}$. [*steepest descent*]
- Calculate *search direction*, $\boldsymbol{s}_k = -\boldsymbol{\Gamma}_k \nabla f(\boldsymbol{x}_k)$
- Calculate step size, α, using line search:

$$g_k(\alpha) = f(\boldsymbol{x}_k + \alpha \boldsymbol{s}_k)$$

- Update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{s}_k$.
- Update $\boldsymbol{\Gamma}_k$ to $\boldsymbol{\Gamma}_{k+1}$.

$\boldsymbol{\Gamma}_k$ = kth approximation of the inverse Hessian

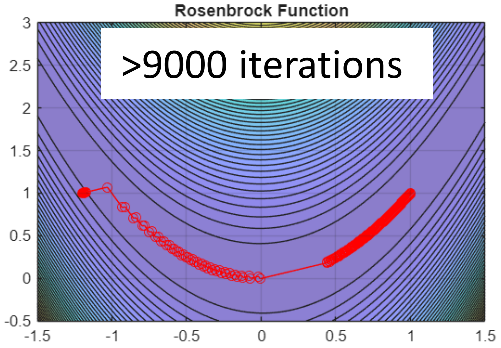
$$\boldsymbol{\Gamma}_{k+1} = \left(\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right)\boldsymbol{\Gamma}_k\left(\boldsymbol{I} - \frac{\boldsymbol{y}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right) + \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}$$

where $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$

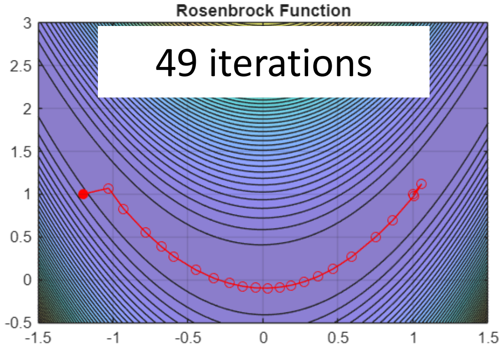
Example:

Minimize $f(\boldsymbol{x})$ from $\boldsymbol{x}_0 = (-1.2, 1.0)$: $f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

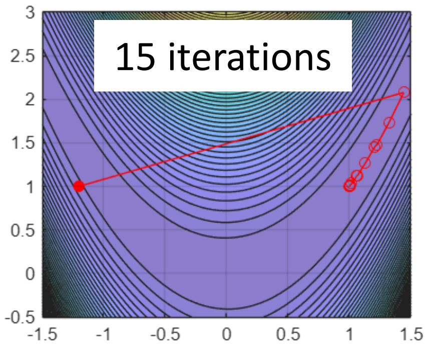
Steepest Descent



Conjugate Gradient Descent



BFGS



Outline

- Introduction to NLP
- Necessary and Sufficient Conditions for Optimality
- Convex Programming
- Methods for Solving NLP
    - One-Dimensional, Unconstrained NLP
    - Multivariable, Unconstrained NLP
    - Zero-order, First-order, Second-order Methods
    - Constrained NLP

Class Exercises

1. Minimize the Beale function from $\boldsymbol{x}_0 = (-2, -2)$:

$$f(\boldsymbol{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

2. Minimize the Booth function from $\boldsymbol{x}_0 = (0, 0)$:

$$f(\boldsymbol{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

Answers:

1. $f(3, 0.5) = 0$

2. $f(1, 3) = 0$