Optimization theory: from a machine learning perspective

James G. Shanahan [1,2]

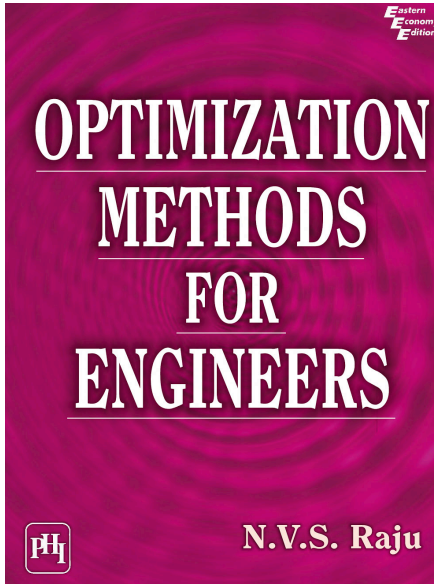
[1]Church and Duncan Group,

[2]School of Informatics, Computing and Engineering, Indiana University

EMAIL: James_DOT_Shanahan_AT_gmail_DOT_com

References

- **OPTIMIZATION METHODS FOR ENGINEERS** – N.V.S. Raju, 2015
  - Analytical versus numerical approaches



Lecture Outline

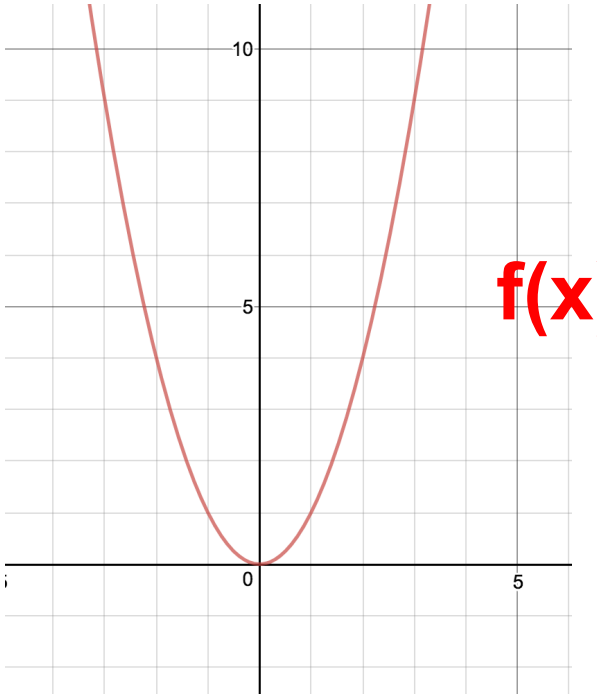
- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Optimization Theory

- **Optimization can be defined as the process of finding the conditions (aka solution) that give the maximum or minimum of a function.**
  - In the simplest case, an optimization problem involves maximizing or minimizing a real function by selecting input values of the function and computing the corresponding values of the function.

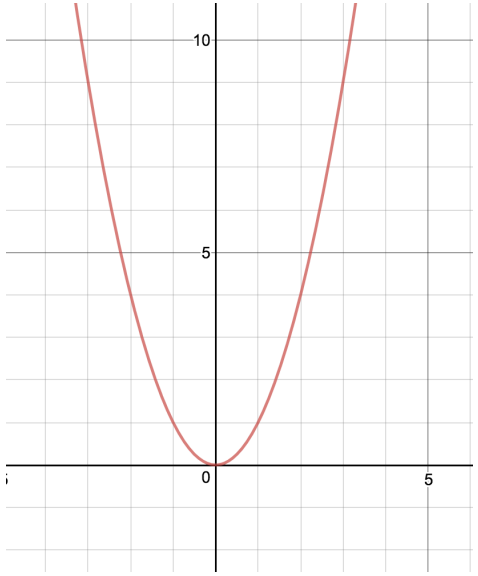


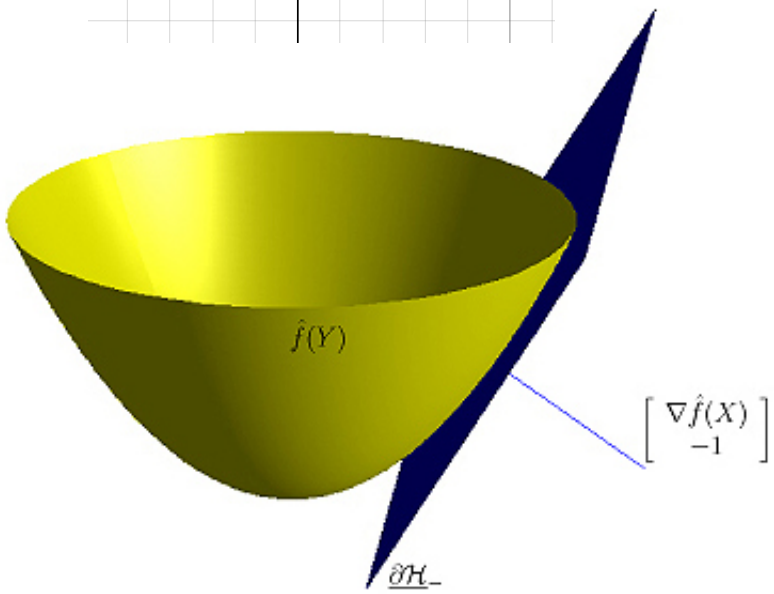
$f(x) = x^2$

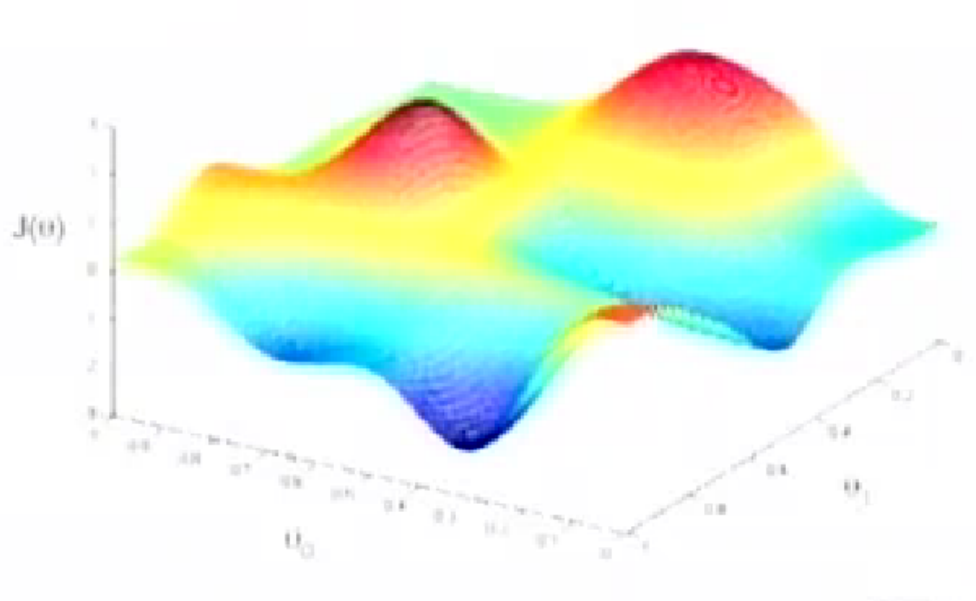
Optimization

- In mathematics, computer science and operations research, **mathematical optimization**, is the selection of a best element (with regard to some criterion) from some set of available alternatives.[1]
- In the simplest case, an optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function.
- The generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics.
- More generally, optimization includes finding "best available" values of some objective function given a defined domain (or input), including a variety of different types of objective functions and different types of domains.

Optimization Theory





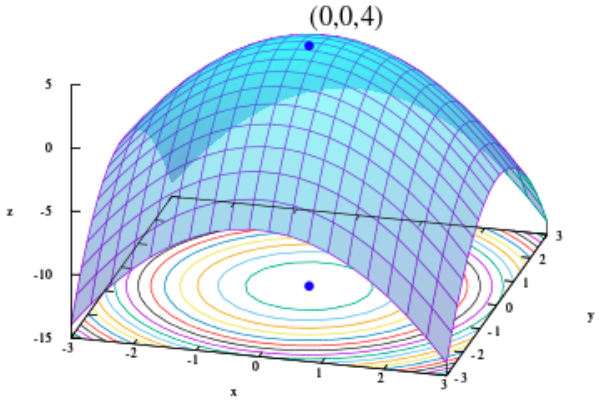


Unconstrained optimization

- **Find the maximizer of the a function**

Maximize $f(x, y) = -(x^2 + y^2) + 4$.

Contour map



Graph of a paraboloid given by $z = f(x, y) = -(x^2 + y^2) + 4$. The global maximum at $(x, y, z) = (0, 0, 4)$ is indicated by a blue dot.

Optimization?

- **Finding (one or more) minimizer, aka solution, of a function subject to constraints**

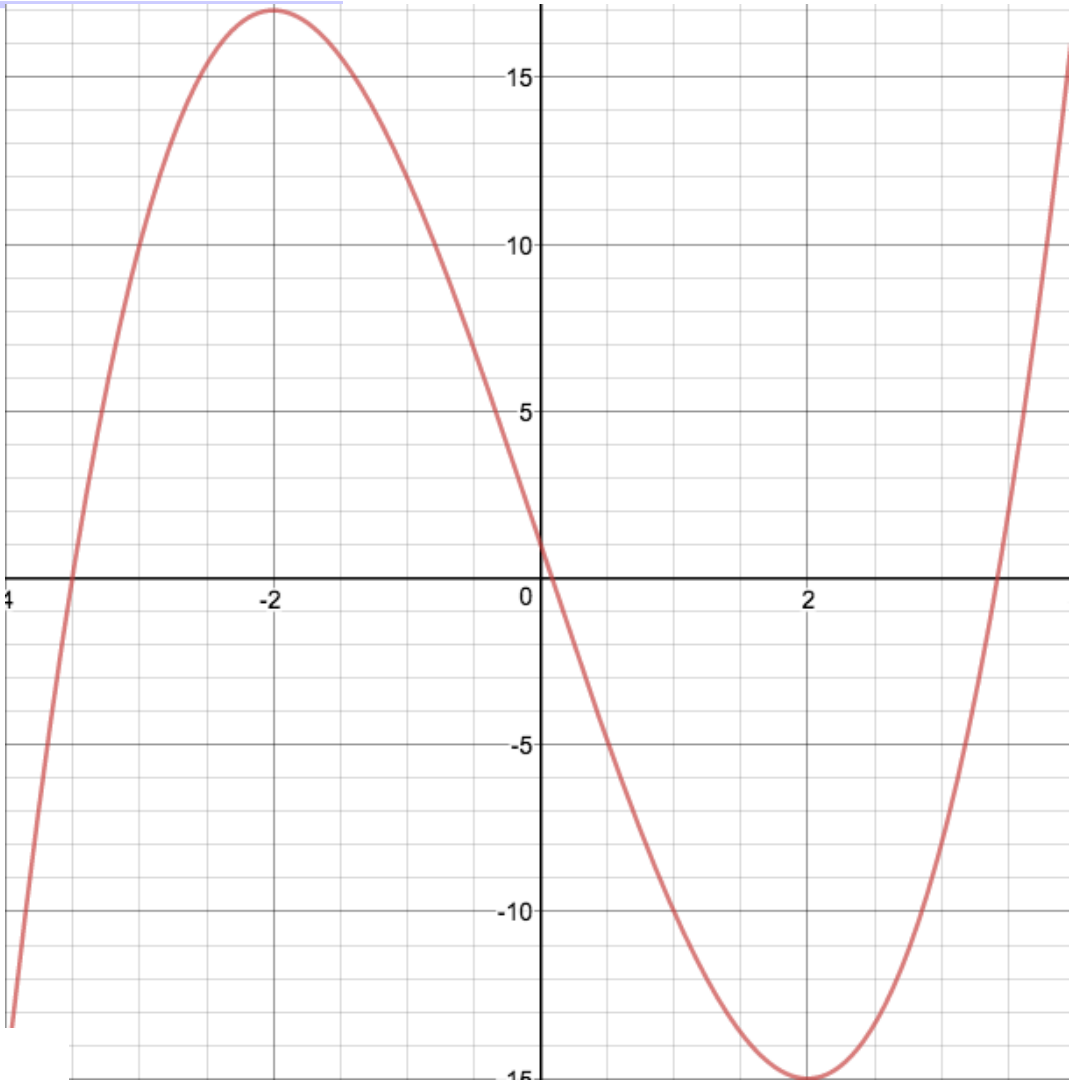
$$\arg\min_x f_0(x)$$
$$\text{s.t.} f_i(x) \leq 0, i = \{1, \ldots, k\}$$
$$h_j(x) = 0, j = \{1, \ldots l\}$$

- **Most of the machine learning problems are, in the end, optimization problems.**

Find the optimal solutions to a problem

- https://www.desmos.com/calculator

$$f(x)=x^3 - 12x+1$$



An **optimal solution** is a feasible **solution** where the objective function reaches its maximum (or minimum) value – for example, the most profit or the least cost.

A globally **optimal solution** is one where there are no other feasible **solutions** with better objective function values.

minimum (–f(x)) == maximum(f(x)

- If a point x^* corresponds to the minimum value of the function $f(x)$, the same point also corresponds to the maximum value of the negative of the function, $-f(x)$.
- Thus optimization can be taken to mean minimization since the maximum of a function can be found by seeking the minimum of the negative of the same function.

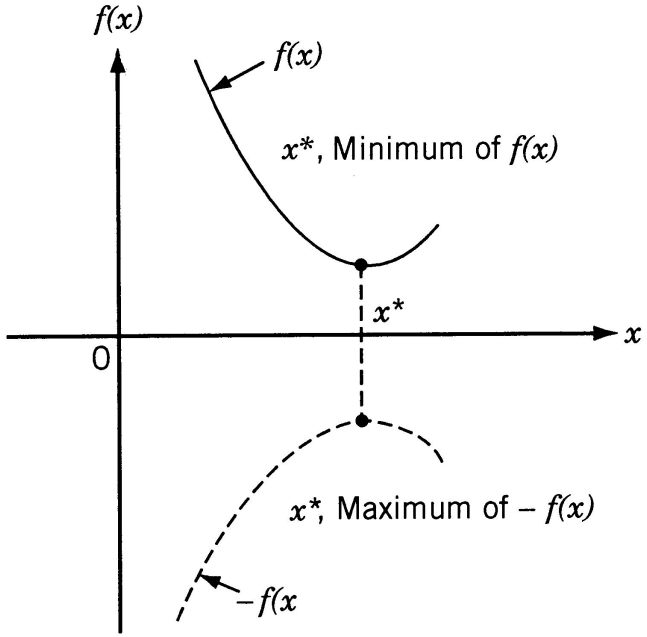


Figure 1.1 Minimum of $f(x)$ is same as maximum of $-f(x)$.

Optimization is huge field

Linear optimization [edit]

Main articles: Linear programming and Simplex algorithm

Linear programming was developed to aid the allocation of resources in firms and in industries during the 1930s in Russia and during the 1940s in the United States. During the Berlin airlift (1948), linear programming was used to plan the shipment of supplies to prevent Berlin from starving after the Soviet blockade.[65][66]

Nonlinear programming [edit]

See also: Nonlinear programming, Lagrangian multiplier, Karush–Kuhn–Tucker conditions, and Shadow price

Extensions to nonlinear optimization with inequality constraints were achieved in 1951 by Albert W. Tucker and Harold Kuhn, who considered the nonlinear optimization problem:

Minimize $f(x)$ subject to $g_j(x) \leq 0$ and $h_j(x) = 0$ where
$f(\cdot)$ is the function to be minimized
$g_j(\cdot)$ $(j = 1, \ldots, m)$ are the functions of the m *inequality constraints*
$h_j(\cdot)$ $(j = 1, \ldots, l)$ are the functions of the l equality constraints.

In allowing inequality constraints, the Kuhn–Tucker approach generalized the classic method of Lagrange multipliers, which (until then) had allowed only equality constraints.[67] The Kuhn–Tucker approach inspired further research on Lagrangian duality, including the treatment of inequality constraints.[68][69] The duality theory of nonlinear programming is particularly satisfactory when applied to convex minimization problems, which enjoy the convex-analytic duality theory of Fenchel and Rockafellar; this convex duality is particularly strong for polyhedral convex functions, such as those arising in linear programming. Lagrangian duality and convex analysis are used daily in operations research, in the scheduling of power plants, the planning of production schedules for factories, and the routing of airlines (routes, flights, planes, crews).[69]

WWII was a big catalyst for OR

Table 1.1 Chief Contributors of Optimization Methods

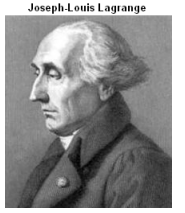
Period	Optimization Method	Chief Contributors
1928	Game theory	von Neumann
1940s	Mean value theorems	Lagrange, Taylor, Cauchy
1940s	Differential calculus methods	Newton, Lagrange, Cauchy
1940	Operations research (basics)	McCloskey and Trefethen
1947	Simplex method and linear programming	G.B. Dantzig
1960s	Big-M (penalty) simplex method	Dantzig, Charnes
1960s	Variations in calculus	Bernoulli, Euler, Lagrange, and Weirstrass
1951	Necessary and sufficient conditions	Kuhn and Tucker
1960s	Unconstrained optimization (numerical)	Zoutendijk, Rosen, Carroli and McCormick
1957	Principle of optimality and dynamic programming problems	Richard Bellman
1960s	Geometric programming	Duffin, Zener and Peterson
1960s	Integer programming	Gomory
1961	Goal (multi-objective) programming	Charnes and Cooper
1990s	Simulated annealing (SA), genetic algorithm (GA), neural networks	—

1. Introduction

Historical development

- **Isaac Newton (1642-1727)**
  (The development of differential calculus methods of optimization)

- **Joseph-Louis Lagrange (1736-1813)**
  (Calculus of variations, minimization of functionals, method of optimization for constrained problems)



- **Augustin-Louis Cauchy (1789-1857)**
  (Solution by direct substitution, steepest descent method for unconstrained optimization)

1. Introduction

Historical development

- Leonhard Euler (1707-1783)
  (Calculus of variations, minimization of functionals)

- Gottfried Leibnitz (1646-1716)
  (Differential calculus methods of optimization)

İsim: Gottfired Wilhelm von Leibniz

1. Introduction

Historical development

- **George Bernard Dantzig (1914-2005)**
  (Linear programming and Simplex method (1947))

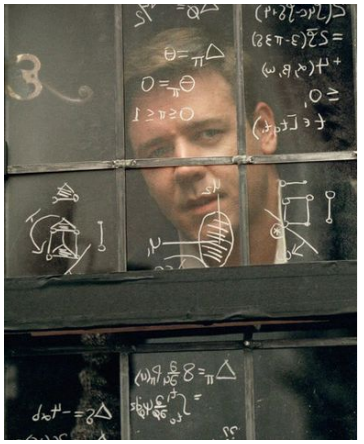
- **Richard Bellman (1920-1984)**
  (Principle of optimality in dynamic programming problems)

- **Harold William Kuhn (1925-)**
  (Necessary and sufficient conditions for the optimal solution of programming problems, game theory)

1. Introduction

Historical development

- **Albert William Tucker (1905-1995) (Necessary and sufficient conditions for the optimal solution of programming problems, nonlinear programming, game theory: his PhD student was John Nash)**



- **Von Neumann (1903-1957) (game theory)**

❑ Variants of Mathematical Optimization Model

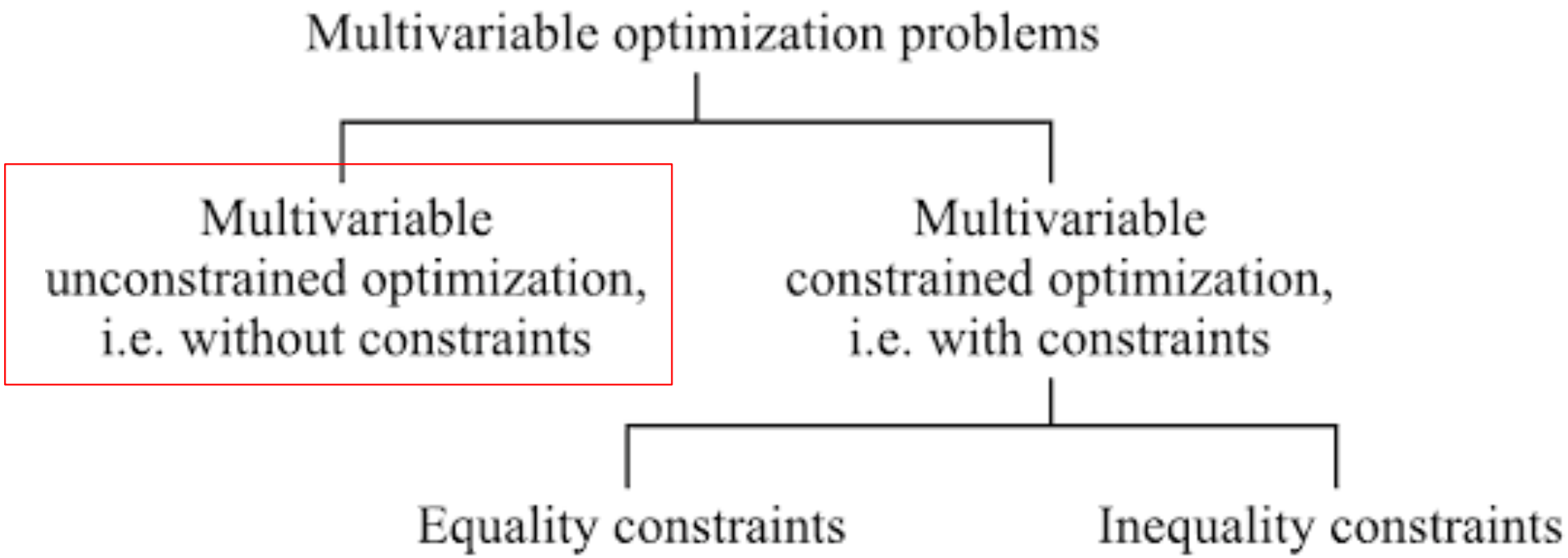
- ❍ Convex(objective fcn: convex, constraint: convex)→Linear Programming
- ❍ Integer (some or all variables: integer values) → Integer programming
- ❍ Quadratic(Objective fcn: quadratic) → Quadratic programming
- ❍ Nonlinear(Objective fcn or constraints: nonlinear) → Nonlinear programming
- ❍ Stochastic(some constraints: random variable) → Stochastic programming

…

❑ Solution Techniques for Mathematical Optimization

- ❍ Optimization algorithms(fixed steps): Simplex algorithm, variants of Simplex, …
- ❍ Iterative methods(converged solution): Newton's method, Interior point methods, Finite difference, Numerical analysis, Gradient descent, Ellipsoid method, …
- ❍ Heuristics(approximated solution): Nelder-Mead simplicial heuristic, Genetic algorithm, Differential Search algorithm, Dynamic relaxation, …

Univariate/Multivariate optimization



Taxonomy of multivariate optimization problems

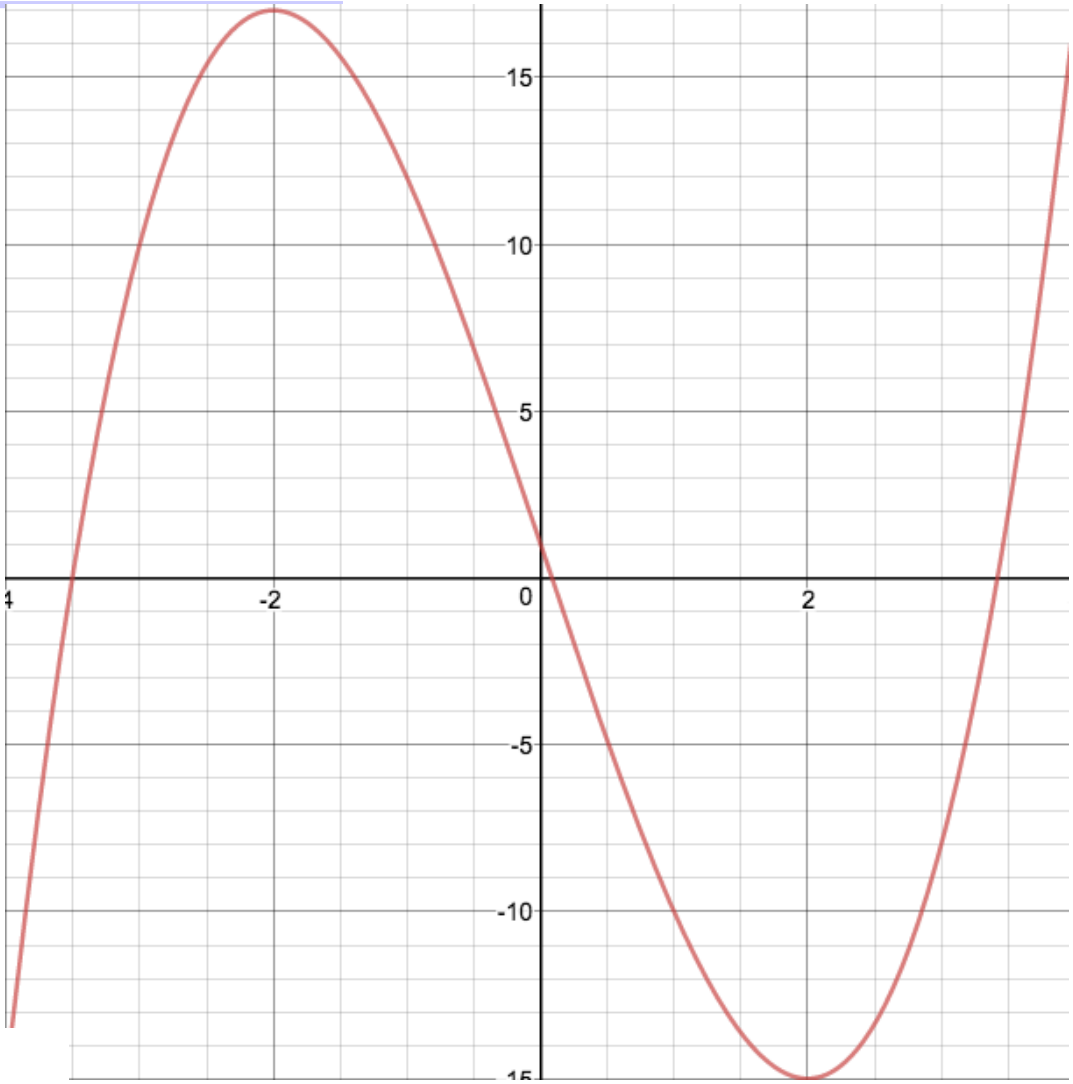
Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Find the optimal solutions to a problem

- https://www.desmos.com/calculator

$$f(x)=x^3 - 12x+1$$



An **optimal solution** is a feasible **solution** where the objective function reaches its maximum (or minimum) value – for example, the most profit or the least cost.

A globally **optimal solution** is one where there are no other feasible **solutions** with better objective function values.

Maximum vs Minimum

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

f'(x=x*) =0 at maximum and minimum

These zero points are the roots!

Second derivative f''(x) = 6x

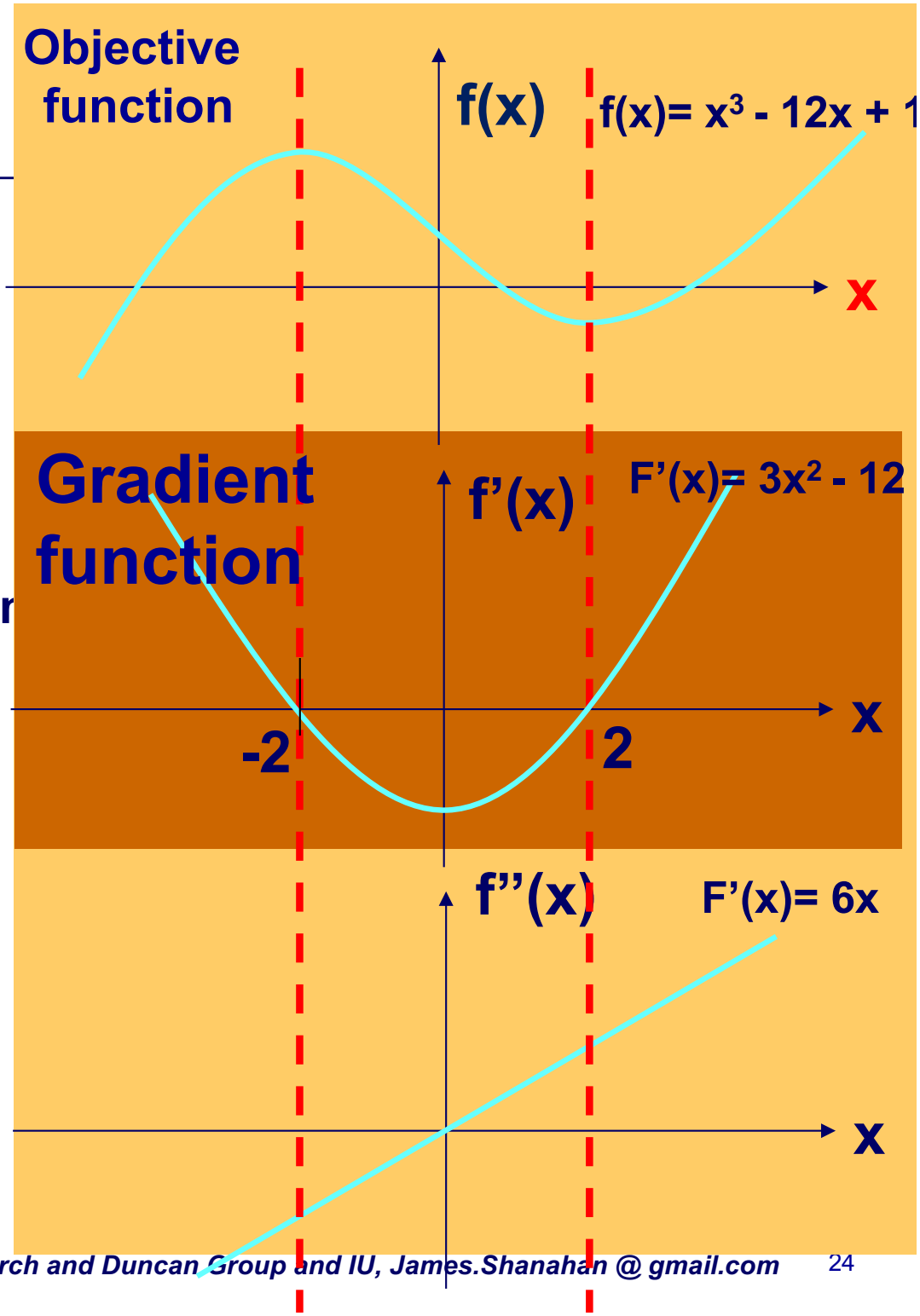
Gives the "rate of change of gradient"

SOC

If f''(x=x*) < 0 then maximum

f''(x=x*) > 0 then minimum

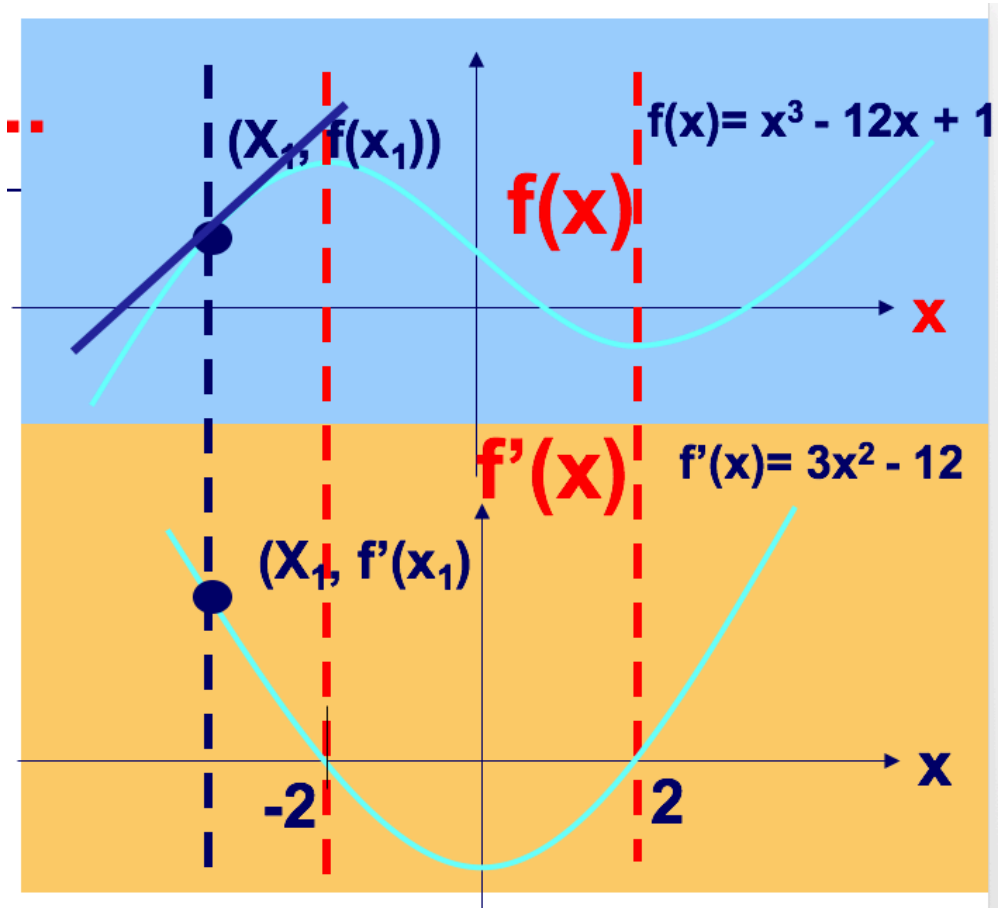
f''(x=x*) == 0 then ???



From turning points to roots

K-12

- **Locating turning points (in the original function) thru root finding algorithms in the first derivative function**



Finding the roots of gradient function f'(x)

- **An important problem in mathematics and statistics is finding values of x to satisfy f(x) = 0.**
  - Such values are the roots of the equation and also known as the zeros of f(x).
- **Can solve analytically via closed form as we did above OR**
  - In quadratic cases
    - Factoring
    - Use the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- **Determine the roots of an equation or multiple equations numerically**

Finding turning points of f(x) by hand via f'(x)=0

$f(x) = 2x^3 - 3x^2 - 12x + 6$ # Function

STEP 1 $f'(x) = 6x^2 - 6x - 12$ # Gradient formula

STEP 2 Since the gradient of the tangent at the turning point is 0

STEP 3: Find Roots

Solve an quadratic equation using factoring

$6x^2 - 6x - 12 = 0$

$x^2 - x - 2 = 0$

$(x - 2)(x - 1) = 0$

$x = 2$ or $x = -1$ Insert into f(x)

STEP 4

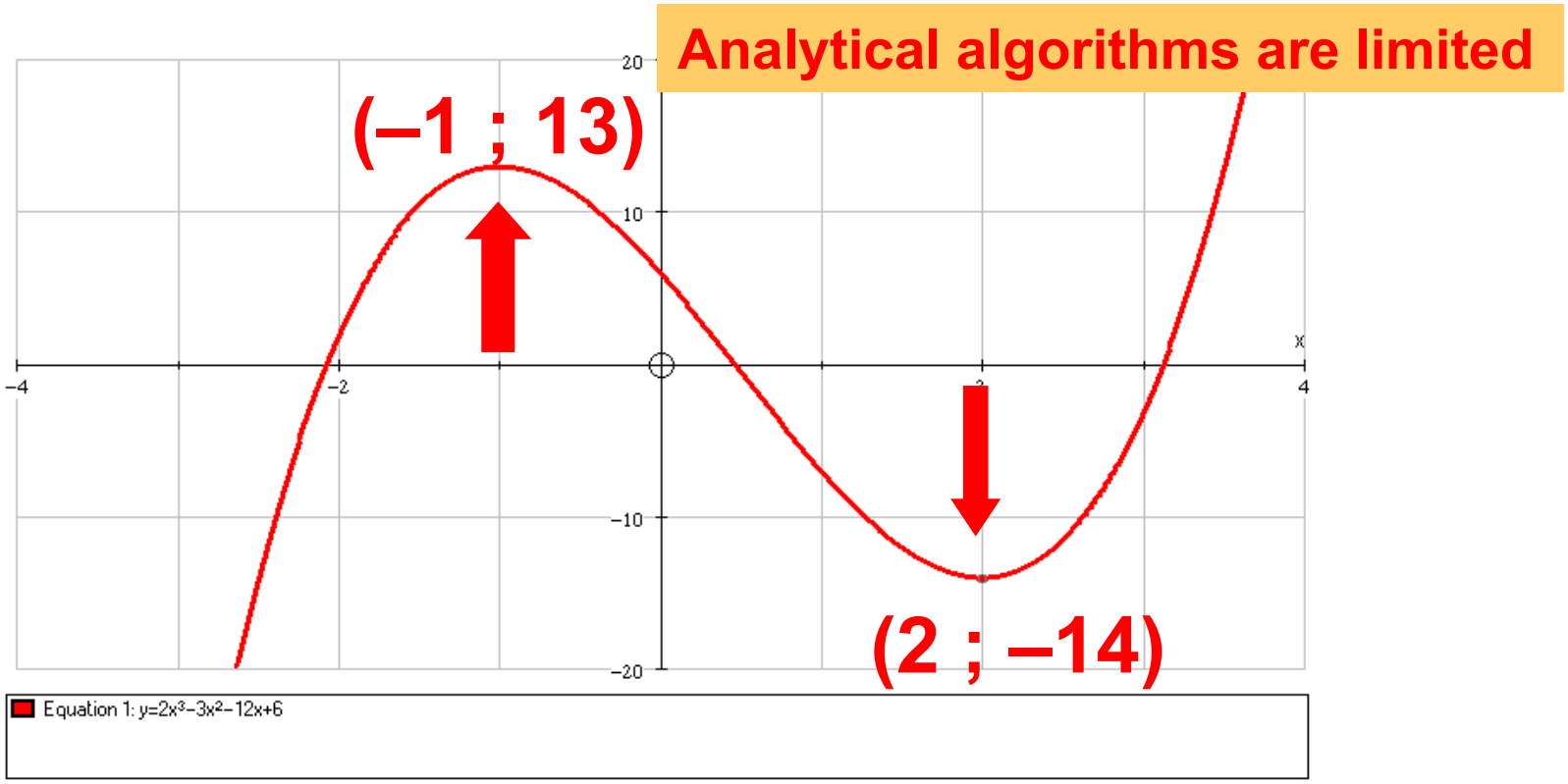
When $x = 2$, $f(2) = 2(2)^3 - 3(2)^2 - 12(2) + 6 = -14$

When $x = -1$, $f(-1) = 2(-1)^3 - 3(-1)^2 - 12(-1) + 6 = 13$

Turning Points (-1, 13) and (2, -14)

Calculate the coordinates of the turning points of the graph

USING CALCULUS and Gradient Descent



Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Finding the roots of gradient function f'(x)

- **An important problem in mathematics and statistics is finding values of x to satisfy f(x) = 0.**
  - Such values are the roots of the equation and also known as the zeros of f(x).
- **Can solve analytically via closed form as we did above OR**
  - In quadratic cases
    - Factoring
    - Use the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- **Determine the roots of an equation or multiple equations numerically**

Finding the roots of gradient function f'(x)

In our case f(x) is f'(x) since we wish to solve f'(x)=0

- **An important problem in mathematics and statistics is finding values of x to satisfy f(x) = 0.**
  - Such values are the roots of the equation and also known as the zeros of f(x).
- **Can solve analytically via closed form as we did above OR**
  - In quadratic cases
    - Factoring
    - Use the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
- **Determine the roots of an equation or multiple equations algorithmically**
  - Newton's Method Module (numerical gradient and analytical gradient)
    - Newton's method or the Newton-Raphson method is a procedure or algorithm for approximating the zeros of a function f (or, equivalently, the roots of an equation f(x) = 0).
  - **Bisection Method Module [wont discuss here]**
  - **Regula Falsi Method Module [wont discuss here]**
  - **Fixed Point Iteration Module [wont discuss here]**
  - **Secant Method Module [wont discuss here]**

Root Finding Algorithms References

- **Can solve analytically via closed form as we did above OR**
  - In quadratic cases
    - Factoring
    - Use the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

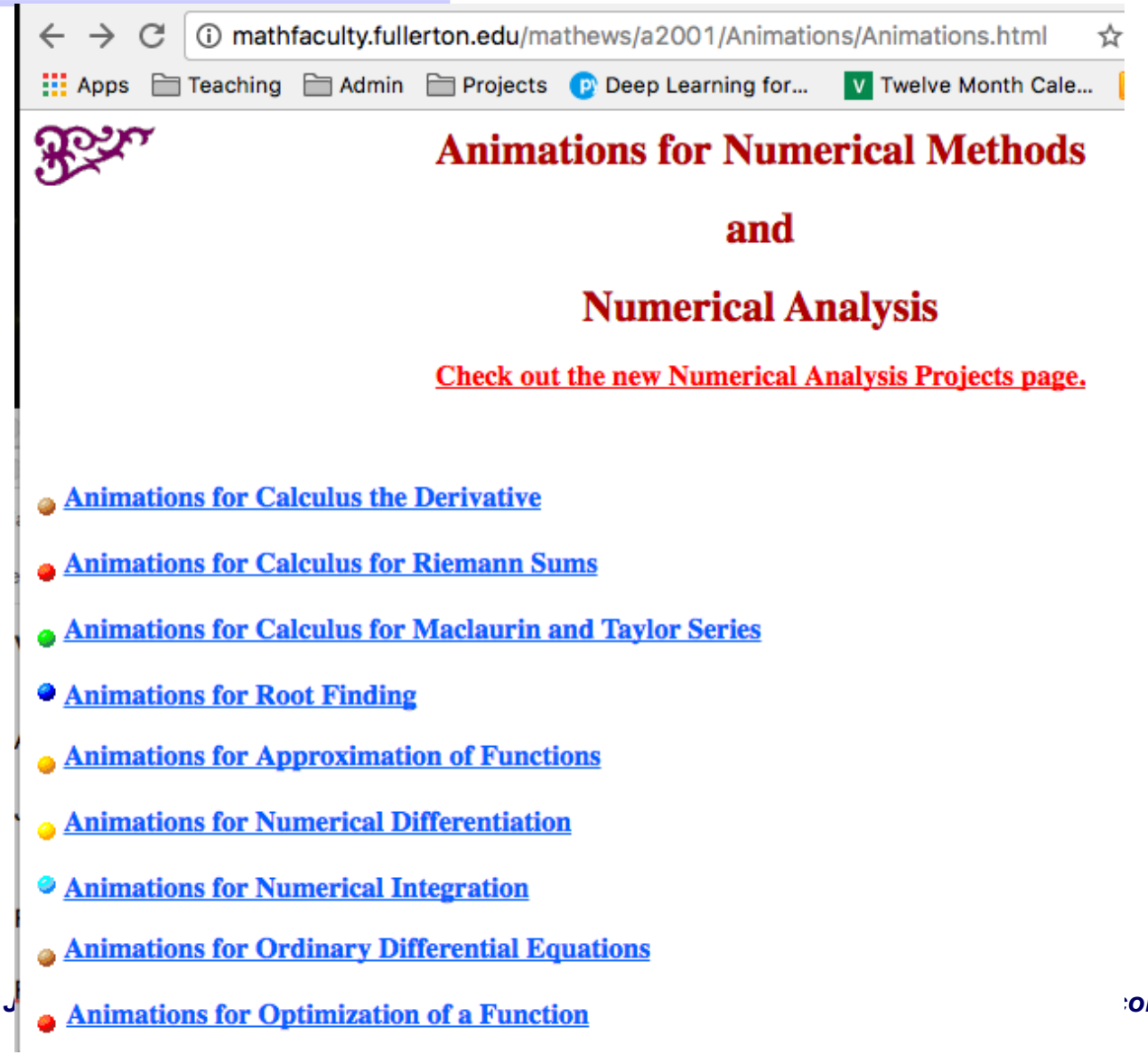
Refs

- **Determine the roots of an equation or multiple equations algorithmically**
  - Newton's Method Module (numerical gradient and analytical gradient)
    - Newton's method or the Newton-Raphson method is a procedure or algorithm for approximating the zeros of a function f (or, equivalently, the roots of an equation f(x) = 0).
  - **Bisection Method Module [wont discuss here]**
  - **Regula Falsi Method Module [wont discuss here]**
  - **Fixed Point Iteration Module [wont discuss here]**
  - **Secant Method Module [wont discuss here]**
- **Click for Animations of the different approaches**
  - http://mathfaculty.fullerton.edu/mathews/a2001/Animations/Animations.html

Root Finding Algorithms and more

- http://mathfaculty.fullerton.edu/mathews/a2001/Animations/Animations.html

Refs



Newton-Raphson Method: A History

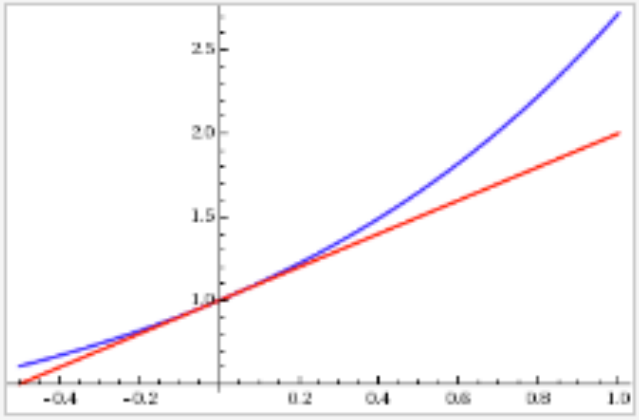
- **Solving a nonlinear equation of the form f(x)=0**
- **Isaac Newton developed an initial version of this algorithm in 1669 and published it in 1685; Raphson tweaked it 1690**

- **Extending it to a system of two equations**
  - In 1740, Thomas Simpson described Newton's method as an iterative method for solving general nonlinear equations using fluxional calculus, essentially giving the description above
  - In the same publication, Simpson also gives the generalization to systems of two equations and notes that Newton's method can be used for solving optimization problems by setting the gradient to zero.

http://en.wikipedia.org/wiki/Newton%27s_method

Taylor Series tell us everything

If a real-valued function *f* is differentiable at the point *a* then it has a linear approximation at the point *a*. This means that there exists a function h_1 such that



Graph of $f(x) = e^x$ (blue) with its linear approximation $P_1(x) = 1 + x$ (red) at $a = 0$.

Approximate f(x) with the Taylors series f(x) ~ f(a)+f'(a)(x-a) + f''(a)(x-a)^2/2…..

$$f(x) = f(a) + f'(a)(x-a) + h_1(x)(x-a), \qquad \lim_{x \to a} h_1(x) = 0.$$

Here

$$P_1(x) = f(a) + f'(a)(x-a)$$

is the linear approximation of *f* at the point *a*. The graph of $y = P_1(x)$ is the tangent line to the graph of *f* at $x = a$. The error in the approximation is

$$R_1(x) = f(x) - P_1(x) = h_1(x)(x-a).$$

Finding the roots f(x) i atively

In our case f(x) is f'(x) since w f'(x)=0

- **An important problem i statistics is findi = 0.**
  - Such values a and also known as
- **Can s d above OR**
- **numerically determine on or multiple equations**
  - or the Newton-Raphson method is a algorithm for approximating the zeros of a (or, equivalently, the roots of an equation f(x) = 0).
  - ecting Method [wont discuss here]
  - Secant Method [wont discuss here]

Taylor Series tell us everything we need to know about finding a min or max f(x)=f(a)+f'(a)(x-a) + f''(a)(x-a)²/2..... Goal: find roots of f'(x) so we can locate Turning Points (candidate Max, Min) (Note Newton uses f'(x) and f''(x)) And use f''(x) to determine if candidate is a max or min

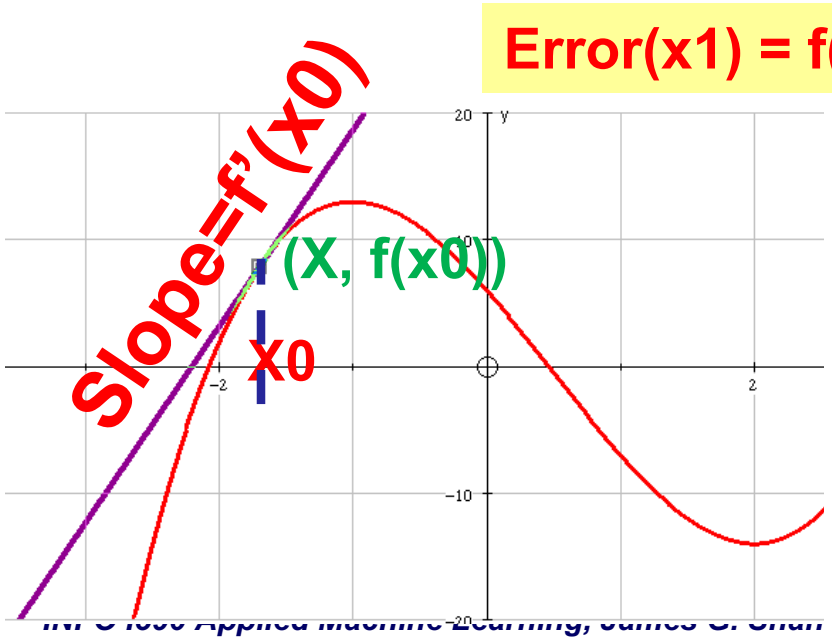
High level algo for finding optimal solutions

- **Taylor Series tell us everything we need to know about finding a min or max $f(x)=f(a)+f'(a)(x-a) + f''(a)(x-a)^2/2\ldots..$**
- **Given f, an univariate objective function, determine optimal solutions**
- **Algorithm to determine maxima or minima**
  - **Find roots of f'(x)** [roots of gradient function]
    - Use an iterative algo
  - **Locate Turning Points** (candidate Max, Min) (Note Newton uses f'(x) and f''(x)) [yields that value of the objective function i.e., $f(x_{root})$
  - **Is candidate max or min?**
    - use f''(x) to determine if candidate is a max or min

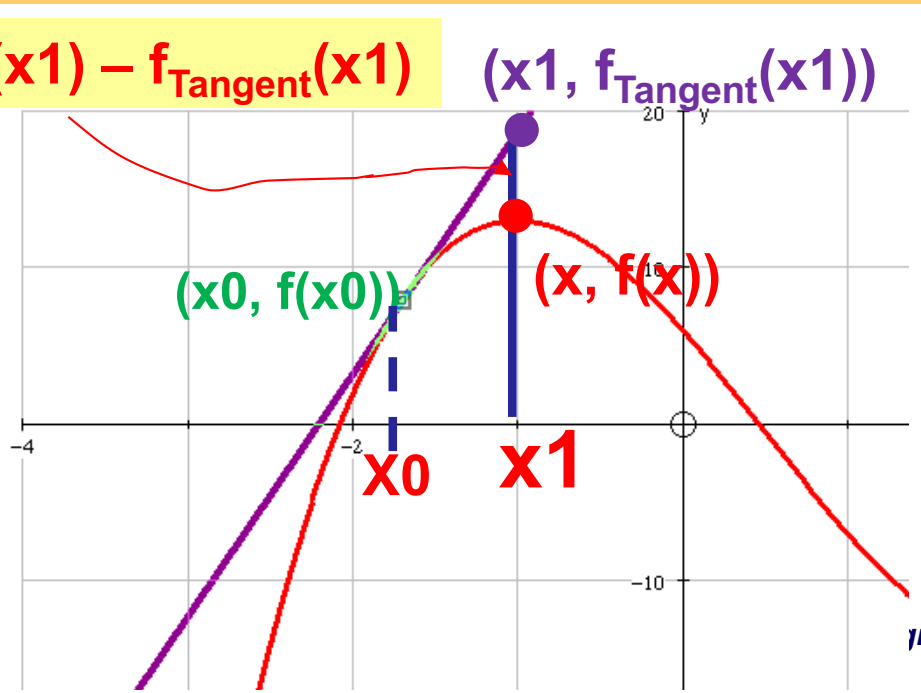
Recall: approximate a curve using a Tangent

- **Given a point on the curve, (x0, f(x0)) and a slope, f'(x0), we can calculate the equation of the tangent at (x0, f(x0))**
  - y - y0 = f'(x0)(x - x0) ## y-y0 = m(x-x0)
  - f(X) = f(x0) + f'(x0)(X-x0) where X is a free variable
- **Then for any X1 in the neighbourhood of X0 we can approximate f(x1) it by the tangent at (x, f(x0)),**
  - i.e., f(**x1**) ~ $f_{tangent}$(**x1**) =f(x0) + f'(x0)(**X1**-x0)

Linear approximations can be good when in the neighborhood



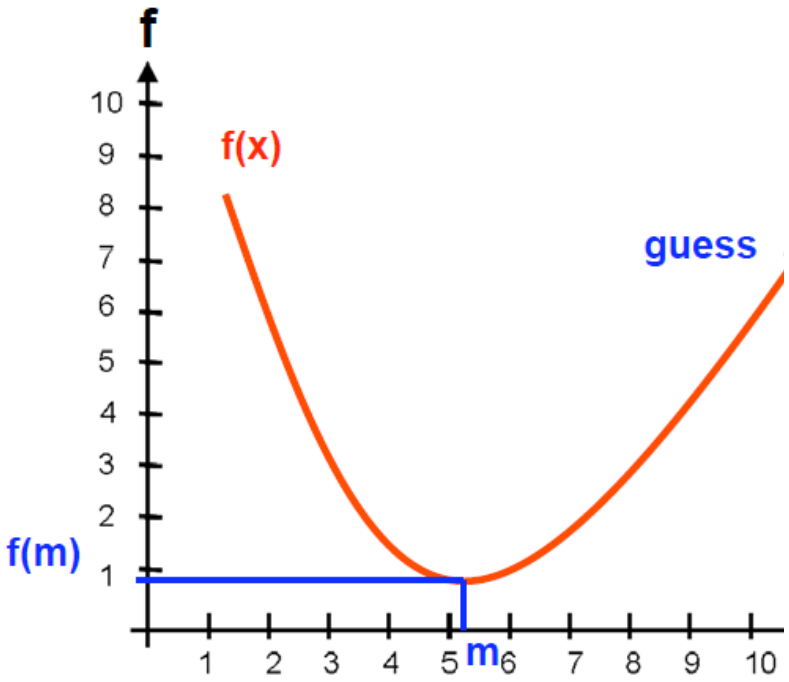
Error(x1) = f(x1) – $f_{Tangent}$(x1)



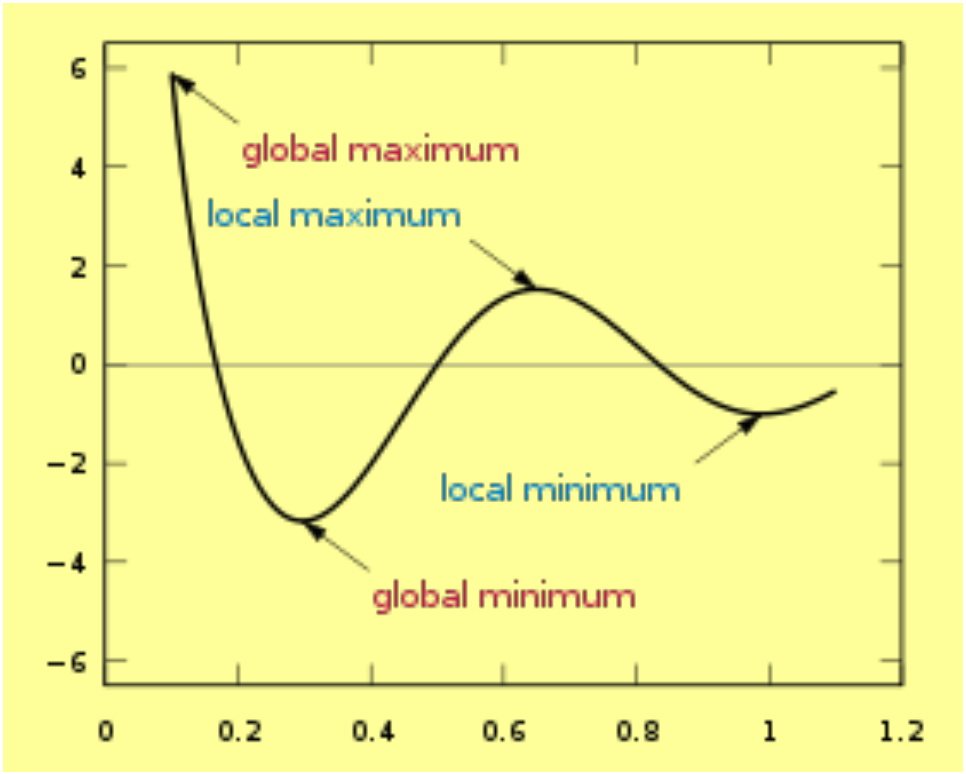
Focus on Convex Univariate problems

For the moment

Convex problem



Non-Convex problem



Local and global maxima and minima for $\cos(3\pi x)/x$, $0.1 \leq x \leq 1.1$

Nonlinear Equations – Iterative Methods

Find the roots

- **Computers can NOT solve the roots in closed form (easily)**
- **Iterative Algorithm**
  - Start from an initial value x^0 as a candidate root (and also bracket the extrema).
  - Generate a sequence of iterate x^{n-1}, x^n, x^{n+1} which hopefully converges to the solution x^* (the root of f(x))
  - Iterates are generated according to an iteration function $F: x^{n+1}=F(x^n)$

Question

- When does it converge to correct solution ?
- What is the convergence rate ?

Given a Pt. and Slope..

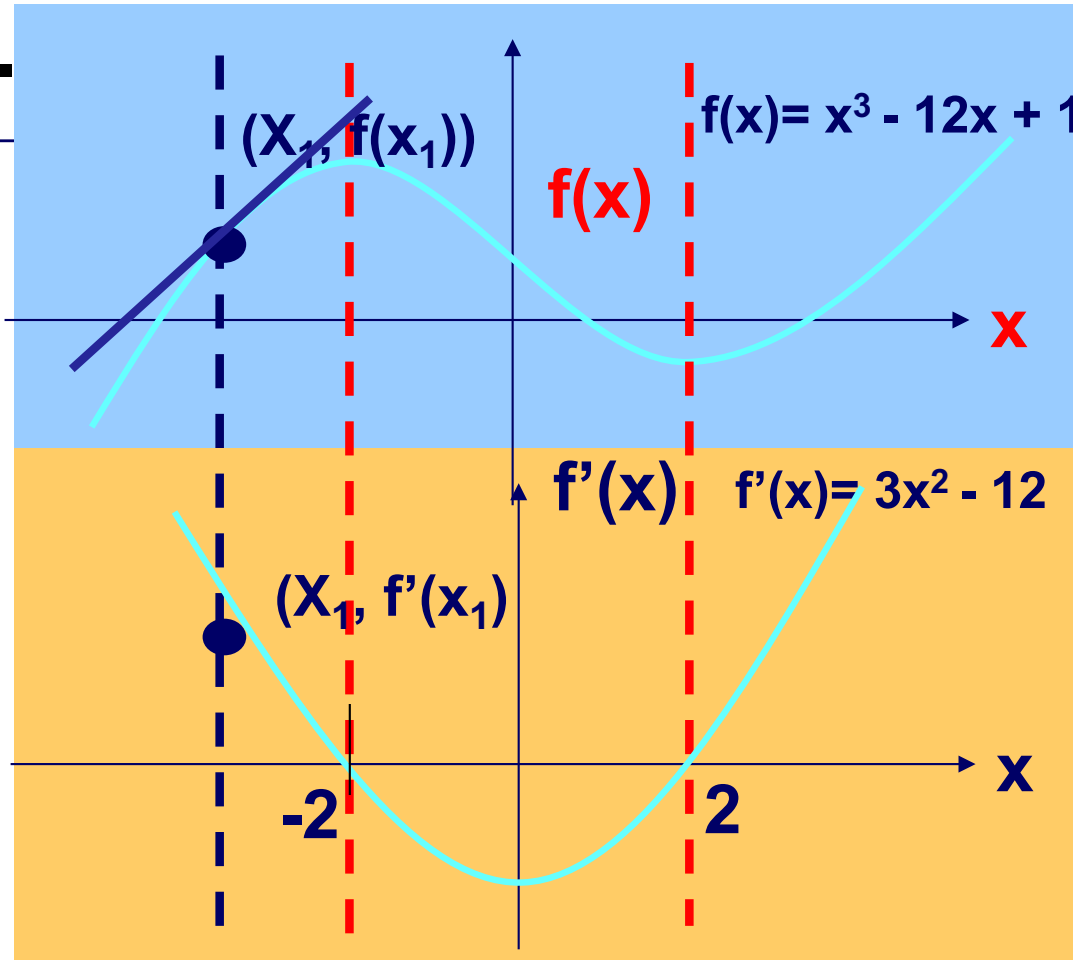
$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

[=0 at maximum and minimum]

Using $(x_1, f(x_1))$ and $m=f'(x_1)$
And the equation formula
y-y0=m(x-x0)



Find roots of f'(x) to give us candidate turning points

Maximum vs Minimum

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

f'(x=x*) =0 at maximum and minimum

These zero points are the roots!

Second derivative f''(x) = 6x

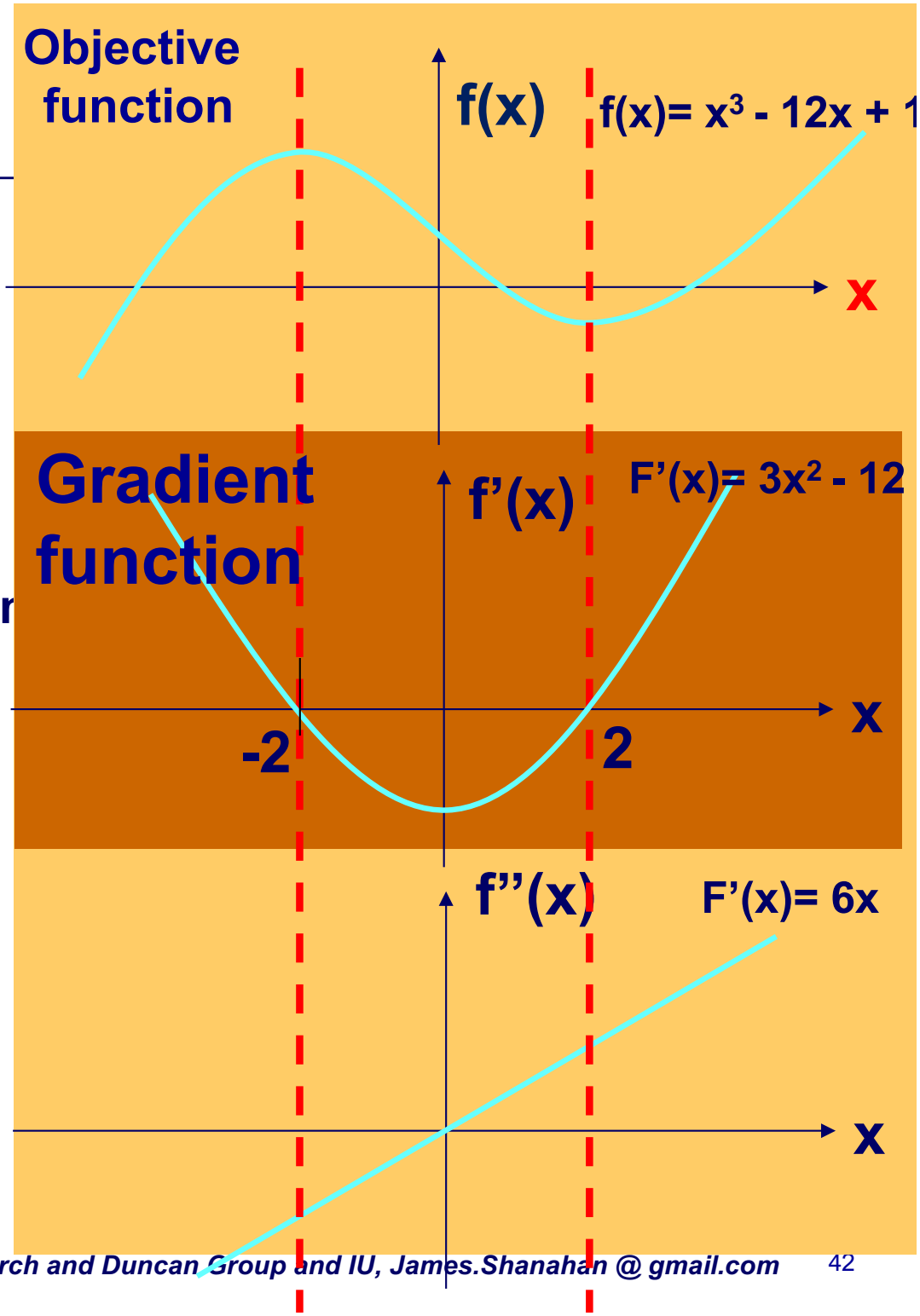
Gives the "rate of change of gradient"

SOC

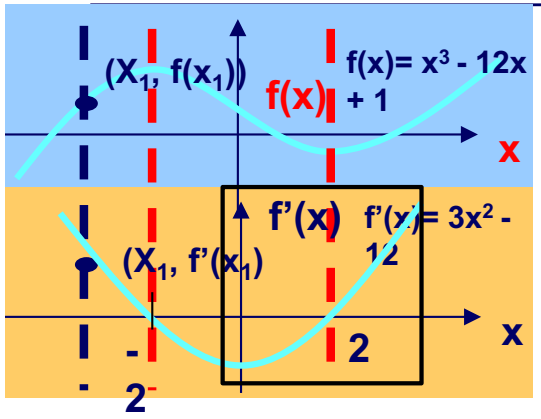
If f''(x=x*) < 0 then maximum

f''(x=x*) > 0 then minimum

f''(x=x*) == 0 then ???



Newton-Raphson Method – Root Finding of derivative/gradient function f'(x)



1. Initial guess: x^0 Letting i=0 $x^i = x^0$
2. Approximate f(x) by tangent at $(x^i, f(x^i))$ *# $(x^0, f(x^0))$ for the first iteratio*
3. Find where $f_{Tangent\_x0}(x) = 0$, i.e., x^{i+1}; better approx. of the root (x^*)
4. Repeat until convergence

E.g., X does not change

Newton-Raphson (NR) Method

Consists of linearizing the system.

Want to solve f(x)=0 $\rightarrow$ Replace f(x) with its linearized version and solve.

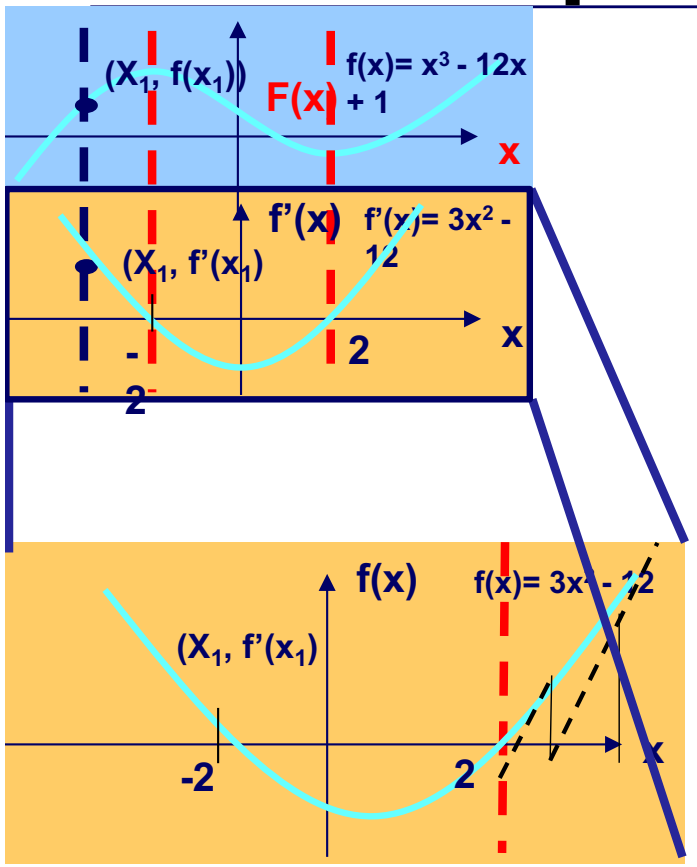
$$f(x) = f(x^*) + \frac{df}{dx}(x^*)(x - x^*) \qquad 1^{st} \text{ order Taylor Series}$$

$$f(x^{k+1}) = f(x^k) + \frac{df}{dx}(x^k)(x^{k+1} - x^k)$$

$$\Rightarrow x^{k+1} = x^k - \left[\frac{df}{dx}(x^k)\right]^{-1} f(x^k) \qquad \text{Iteration function}$$

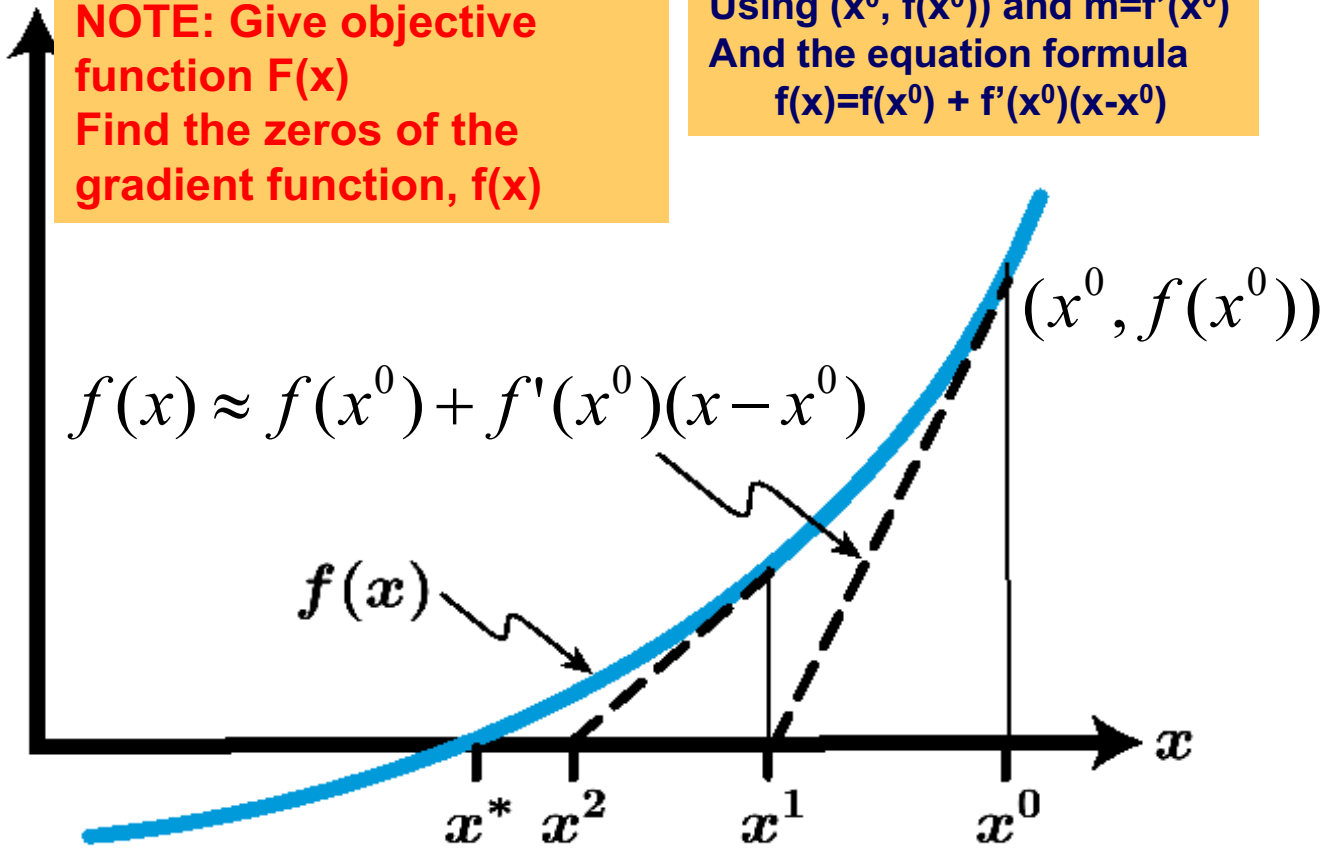
Note: at each step need to evaluate *f* and *f'*

Newton-Raphson Method – Graphical View



NOTE: Give objective function F(x)
Find the zeros of the gradient function, f(x)

Using $(x^0, f(x^0))$ and $m=f'(x^0)$
And the equation formula
$f(x)=f(x^0) + f'(x^0)(x-x^0)$



1. Initial guess: x^0 Letting i=0 $x^i = x^0$
2. Approximate f(x) by tangent at $(x^i, f(x^i))$ *# $(x^0, f(x^0))$ for the first iteration*
3. Find where $f_{Tangent\_x0}(x) = 0$, i.e., x^{i+1}; better approx. of the root (x^*)
4. Repeat until convergence

Newton-Raphson (NR) Method

Consists of linearizing the system.

Want to solve f(x)=0 $\rightarrow$ Replace f(x) with its linearized version and solve.

$$f(x) = f(x^*) + \frac{df}{dx}(x^*)(x - x^*) \qquad 1^{st} \text{ order Taylor Series}$$

$$f(x^{k+1}) = f(x^k) + \frac{df}{dx}(x^k)(x^{k+1} - x^k)$$

$$\Rightarrow x^{k+1} = x^k - \left[\frac{df}{dx}(x^k)\right]^{-1} f(x^k) \qquad \text{Iteration function}$$

Note: at each step need to evaluate *f* and *f'*

Deriving Newton-Raphson Method

Find the zeros of the gradient function, f(x), given objective function F(x)

- **Solving a nonlinear equation of the form f(x)=0**
- **Generate a sequence of iterate x^{n-1}, x^{n}, x^{n+1} which hopefully converges to the solution x^* (the root of f(x))**

Approximate f(x) with a 1st order Taylor Series around x^i

1 $$f(x) \approx f(x^0) + f'(x^0)(x - x^0) \quad \forall x \text{ surrounding } x^0$$

2 $$f(x^{i+1}) \approx f(x^i) + f'(x^i)(x^{i+1} - x^i) \quad \forall x \text{ surrounding } x^i$$

3 $$0 = f(x^i) + f'(x^i)(x^{i+1} - x^i) \qquad \text{We desire a root (i.e., } f(x^{i+1}) = 0)$$

4 $$f'(x^i)(x^{i+1} - x^i) = -f(x^1)$$

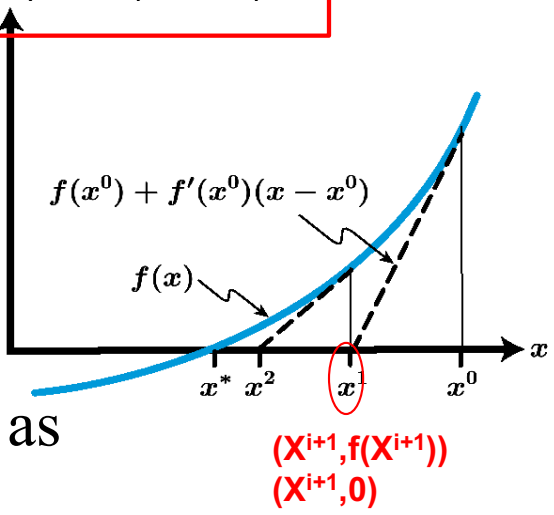
Eq2. Solve for x^{i+1}

5 $$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i$$

Iteration function

6 $$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$

sometimes written as



Newton-Raphson (NR) Method

Consists of linearizing the system.

Want to solve f(x)=0 $\rightarrow$ Replace f(x) with its linearized version and solve.

$$f(x) = f(x^*) + \frac{df}{dx}(x^*)(x - x^*) \qquad 1^{st}\ order\ Taylor\ Series$$

$$f(x^{k+1}) = f(x^k) + \frac{df}{dx}(x^k)(x^{k+1} - x^k)$$

$$\Rightarrow x^{k+1} = x^k - \left[\frac{df}{dx}(x^k)\right]^{-1} f(x^k) \qquad Iteration\ function$$

Note: at each step need to evaluate *f* and *f'*

Newton-Raphson Method – Algorithm

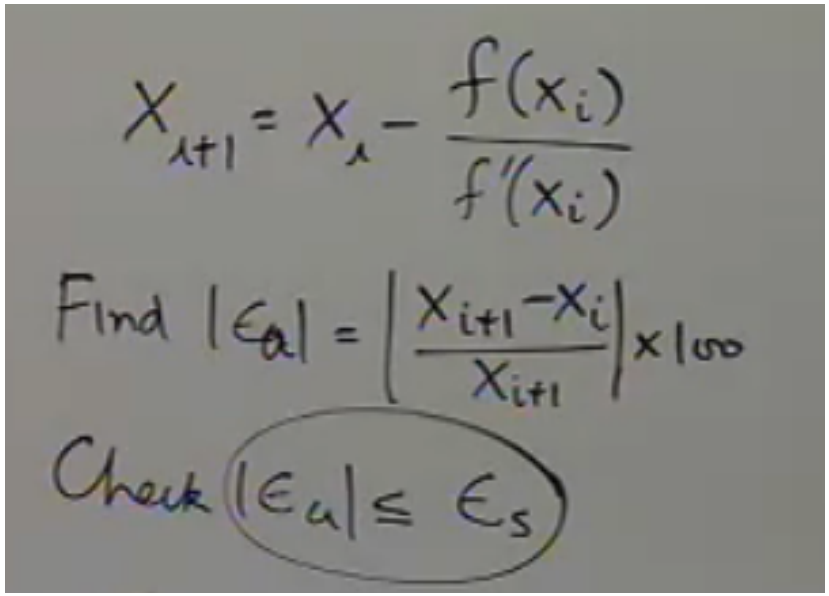
$x^1, x^2, x^3 \ldots, \ldots, x^*$

Define iteration

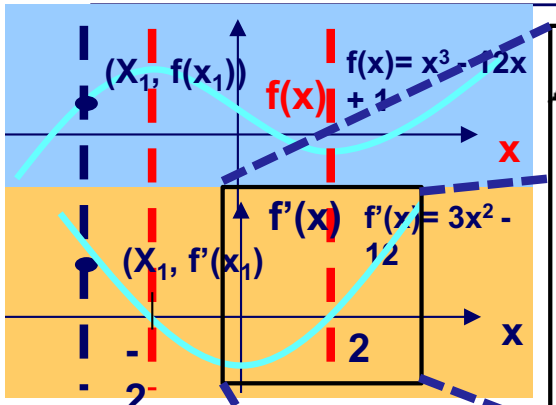
Do $k = 0$ to ….

$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$

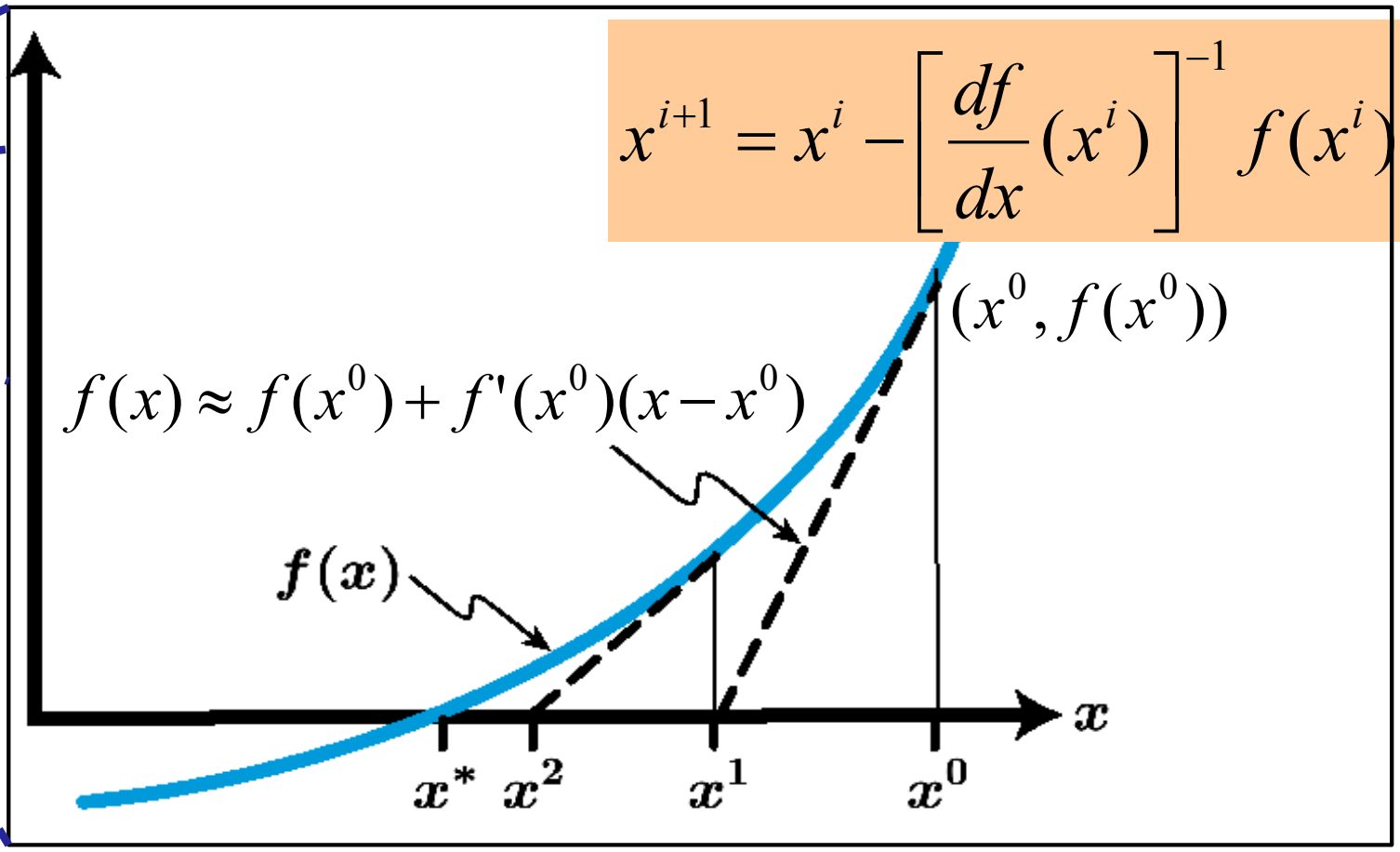
until convergence



Newton-Raphson Method – Root Finding of derivative/gradient function f'(x)



$$x^{i+1} = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i)$$



1. Initial guess: x^0 Letting i=0 $x^i = x^0$
2. Approximate f(x) by tangent at $(x^i, f(x^i))$ *# $(x^0, f(x^0))$ for the first iteratio*
3. Find where $f_{Tangent\_x0}(x) = 0$, i.e., x^{i+1}; better approx. of the root (x^*)
4. Repeat until convergence

E.g., X does not change

Quiz: Do one NR Step, i.e., Find 2nd approx.

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i) \quad Iteration\ function$$

Taking 1 as the first approximation of a root of $x^3+2x-4=0$, use the Newton-Raphson method to calculate the second approximation of this root.

$f(x)=x^3+2x-4$ $\quad$ $f'(x)=3x^2+2$

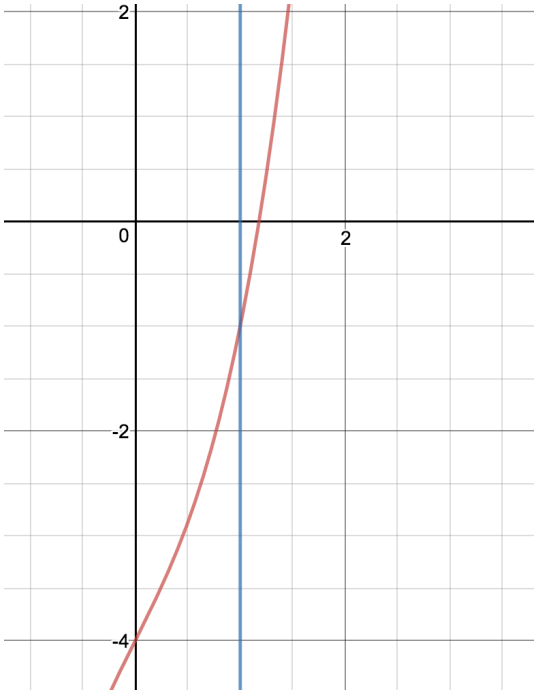
$f(1)=1+2-4=-1$

$f'(1)=3+2=5$

$$x_2 = 1 - \frac{-1}{5} = 1 + \frac{1}{5} = 1.2$$

Do one Newton-Raphson step

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i) \quad Iteration\ function$$

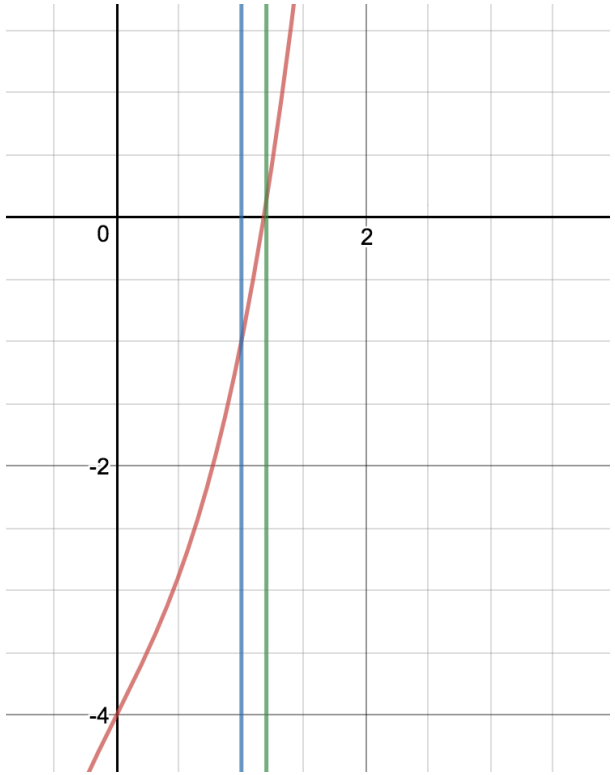


$$f(x)=x^3+2x-4$$

$$f'(x)=3x^2+2$$

Do one Newton-Raphson step

$$x^{i+1} = -\frac{f(x^i)}{f'(x^i)} + x^i = x^i - \left[\frac{df}{dx}(x^i)\right]^{-1} f(x^i) \quad \textit{Iteration function}$$



$$f(x)=x^3+2x-4$$

$$f'(x)=3x^2+2$$

$$f(1)=1+2-4=-1$$

$$f'(1)=3+2=5$$

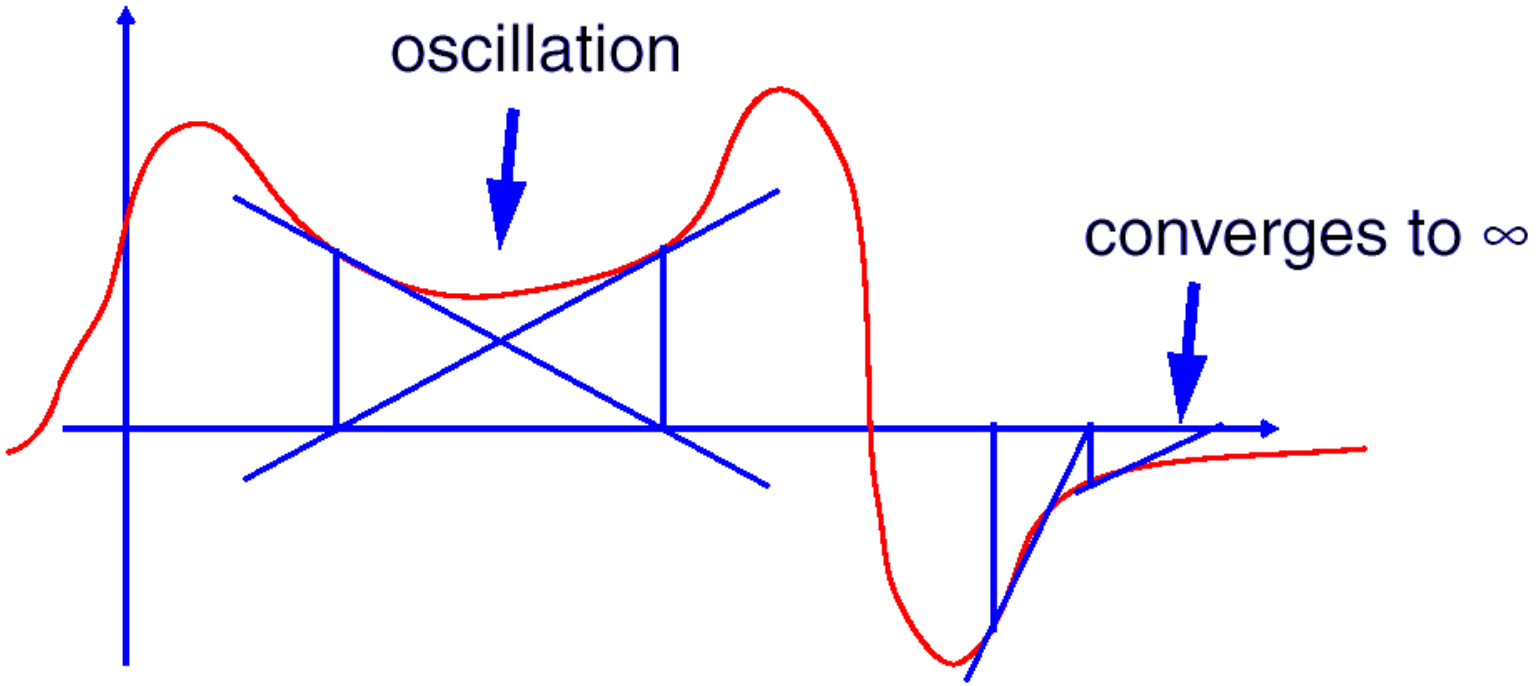
$$x_2=1-\frac{-1}{5}=1+\frac{1}{5}=1.2$$

Newton-Raphson Method – Convergence

Local Convergence

Convergence Depends on a Good Initial Guess

Example:



Operational Issues:Quasi-Newton

Suppose that the objective f is a function of multiple arguments, $f(w_1, w_2, \ldots w_p)$. Let's bundle the parameters into a single vector, $\vec{w}$. Then the Newton update is

$$\vec{w}_{n+1} = \vec{w}_n - H^{-1}(w_n)\nabla f(\vec{w}_n) \quad (16)$$

Calculating gradient and Hessian not very time-consuming but calculating the inverse of H is!

where ∇f is the **gradient** of f, its vector of partial derivatives $[\partial f/\partial w_1, \partial f/\partial w_2, \ldots \partial f/\partial w_p]$, and H is the **Hessian** of f, its matrix of second partial derivatives, $H_{ij} = \partial^2 f/\partial w_i \partial w_j$.

Calculating H and ∇f isn't usually very time-consuming, but taking the inverse of H is, unless it happens to be a diagonal matrix. This leads to various **quasi-Newton** methods, which either approximate H by a diagonal matrix, or take a proper inverse of H only rarely (maybe just once), and then try to update an estimate of $H^{-1}(w_n)$ as w_n changes. (See section 8.3 in the textbook for more.)

In R, have a look at

?optim #method=BFGS

[http://www.stat.cmu.edu/~cshalizi/350/2008/lecture 29/lecture-29.pdf]

[Hand, Manilla, Smith, Data Mining, Section 8.3]

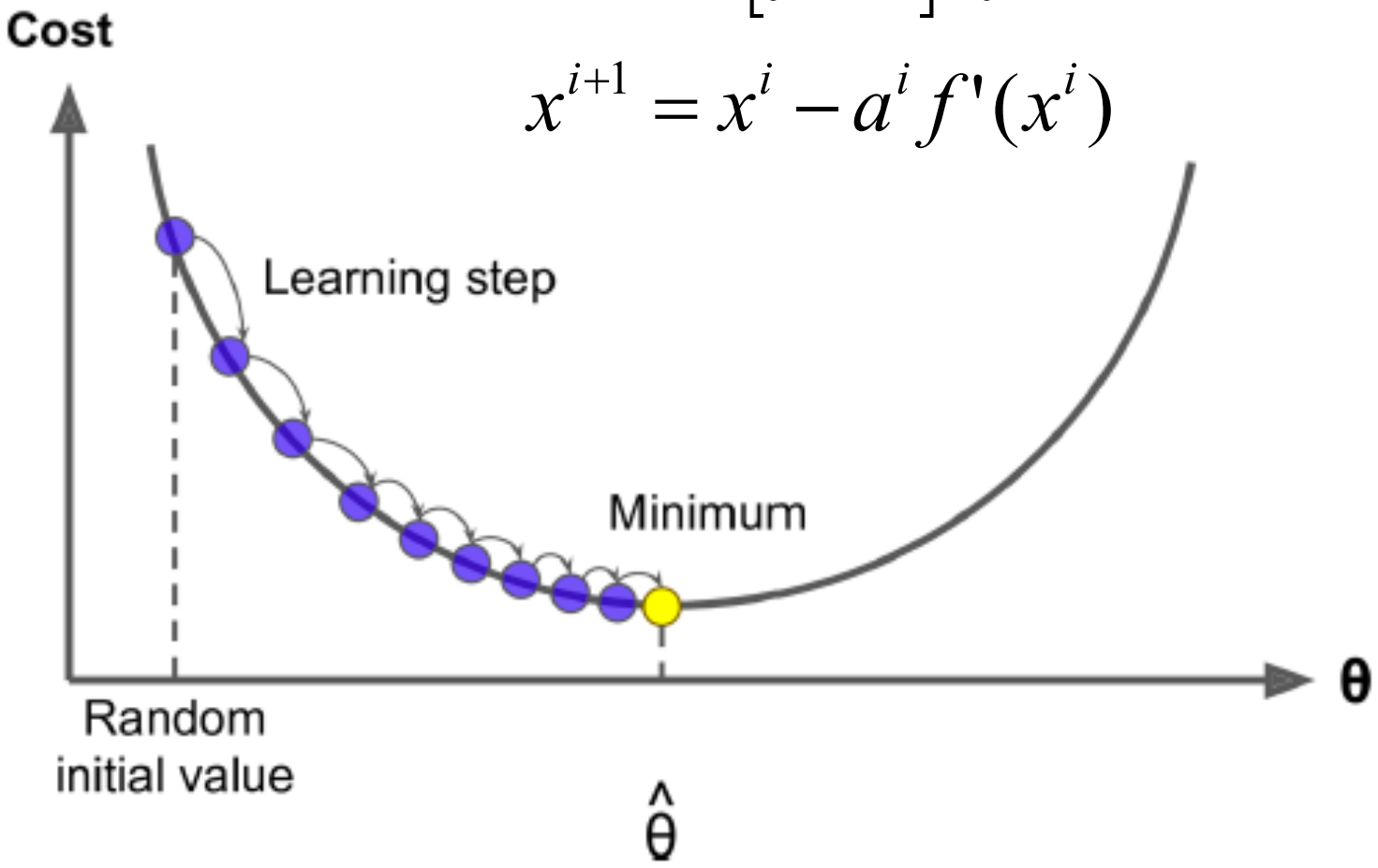
Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Gradient provides direction and f''(x) the stepsize

$$x^{i+1} = x^i - \left[f''(x^i) \right]^{-1} f'(x^i)$$

$$x^{i+1} = x^i - a^i f'(x^i)$$



Gradient Descent (a simpler root finder) in 1D

$$x^{i+1} = x^i - \frac{f'(x^i)}{f''(x^i)} \qquad \textit{Iteration function}$$

Newton-Raphson In 1-Dimension

$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i)$$

- **Calculating f''(x), the Hessian H, and inverting it is complex so simpler algorithms have been developed such gradient descent**

$$x^{i+1} = x^i - a^i f'(x^i)$$

Gradient Descent
Iterate until X does not change

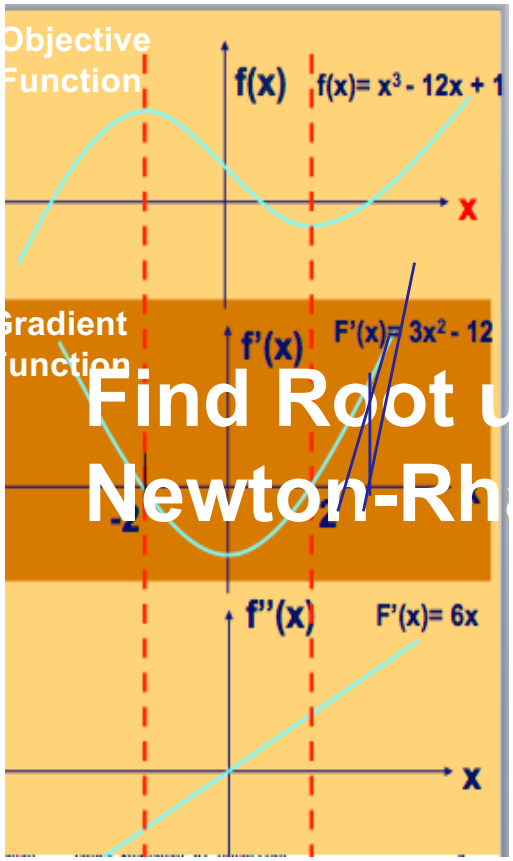
- **How large should I step in the positive gradient direction (gradient ascent)**
  - or in the negative gradient direction (gradient descent)
  - Convergence criteria. E.g., decision vector X does not change that much

Gradient Descent (a simpler root finder)

GIVEN: Minimize f(x)

STEP 1: Find the zeros of the gradient function f'(x)

- **Using Bisection method; OR Newton-Raphson; OR Gradient Descent**



$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \left[H(x^i)\right]^{-1} J(x^i) \qquad \text{Multivariate Case}$$

Newton-Ra

Calculating f''(x), the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)

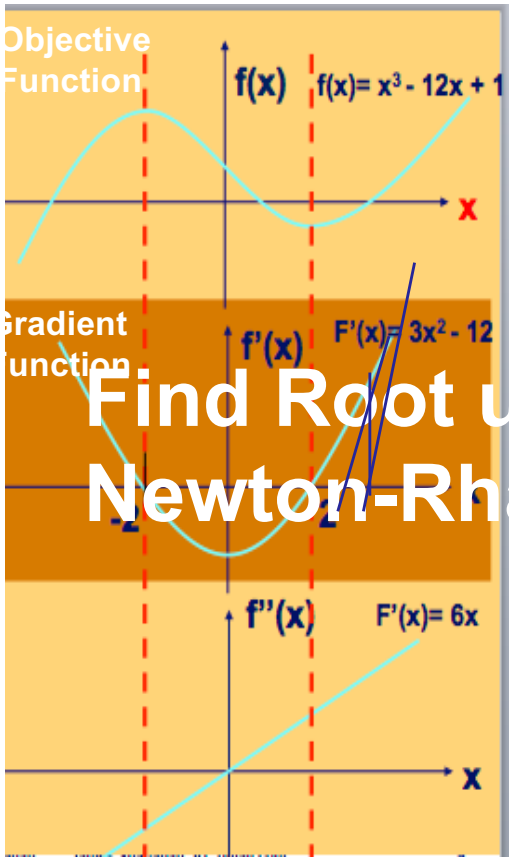
- **Convergence criteria. E.g., decision vector X does not change that much or error does not change**

Gradient Descent (a simpler root finder)

GIVEN: Minimize f(x)

STEP 1: Find the zeros of the gradient function f'(x)

- **Using Bisection method; OR Newton-Raphson; OR Gradient Descent**



$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \left[H(x^i)\right]^{-1} J(x^i) \qquad \text{Multivariate Case}$$

Newton-Ra

Calculating f''(x), the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

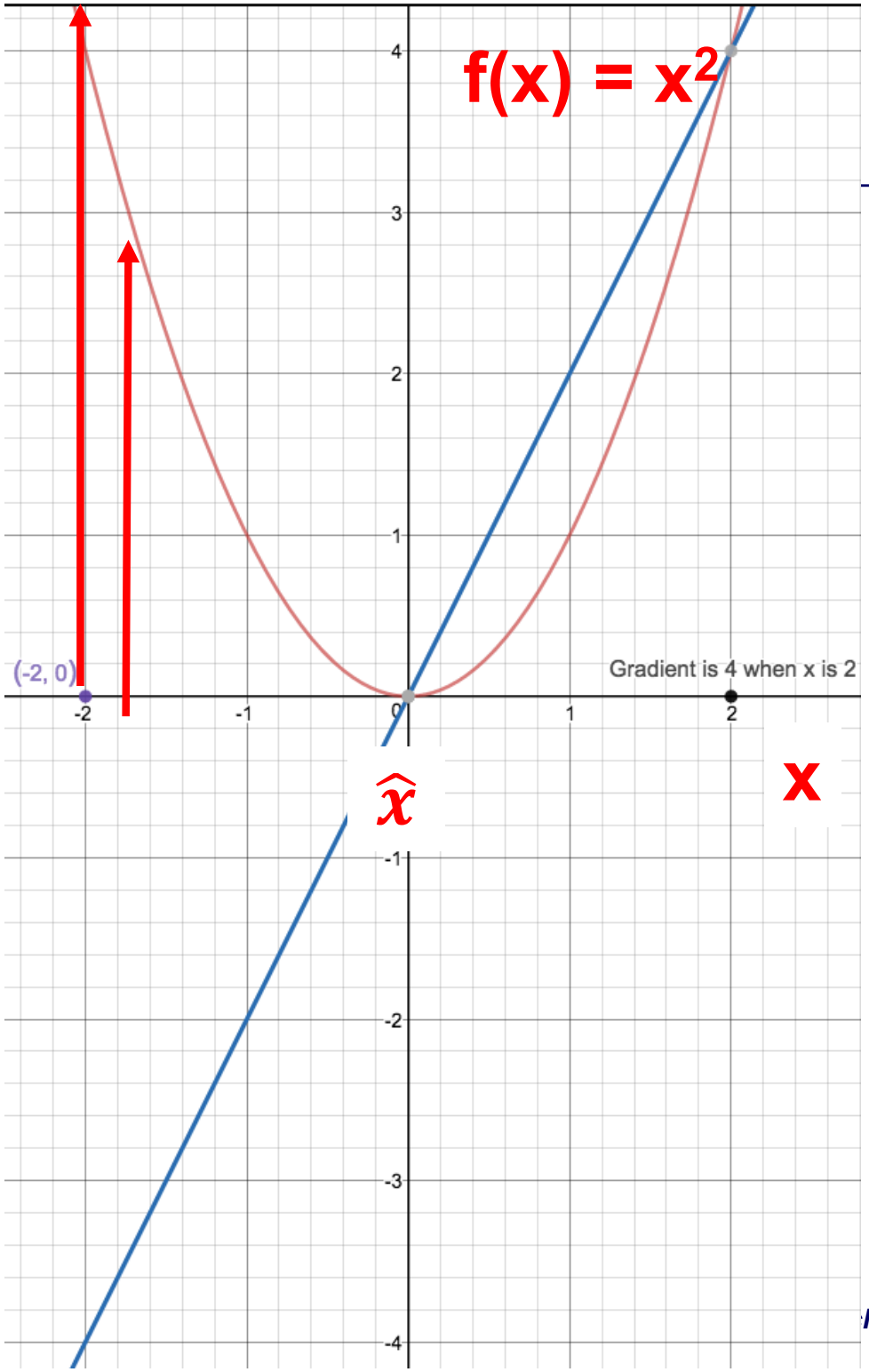
$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

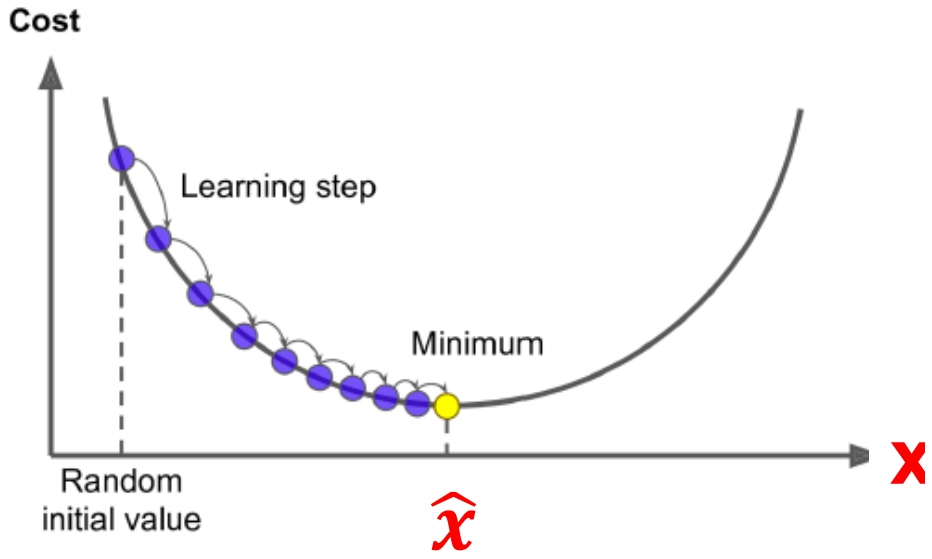
- or in the negative gradient direction (gradient descent)

- **Convergence criteria. E.g., decision vector X does not change that much or error does not change**



Gradient update at x=-2

$$x^{i+1} = x^i - \alpha^i f'(x^i)$$ Univariate case



$$x = x - 0.1\frac{df(x)}{dx}$$ **Eq (1)**

$$x = x - 0.1\frac{dx^2}{dx}$$ **Eq (2)**

$$x = x - 0.1 * 2x$$ **Eq (3)**

$$x = -2 - 0.1 * (2 * -2)$$ **Eq (4)**

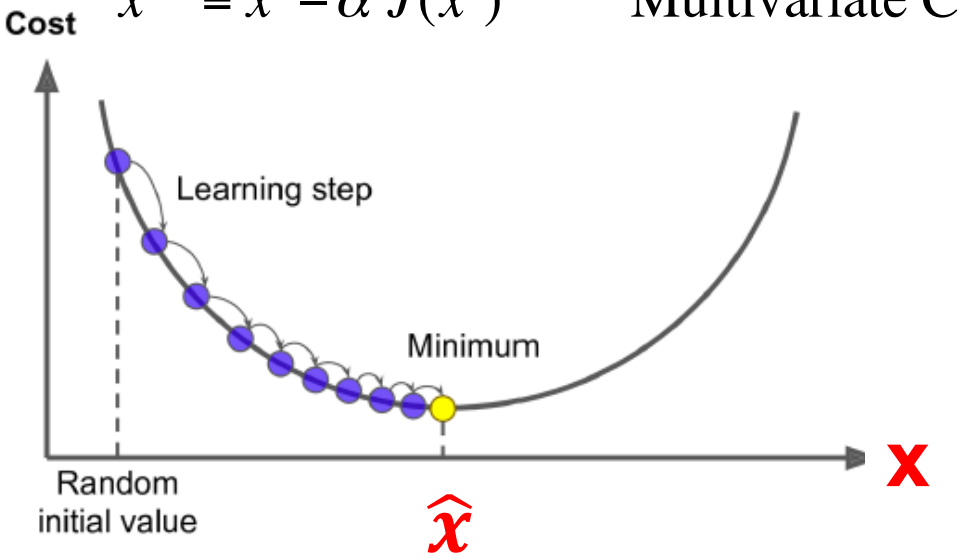
$$x = -2 - 0.1 * -4$$ **Eq (5)**

$$x = 1.6$$

Gradient update at x=-2

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$



$$x = x - \frac{df(x)}{dx} \qquad (1)$$

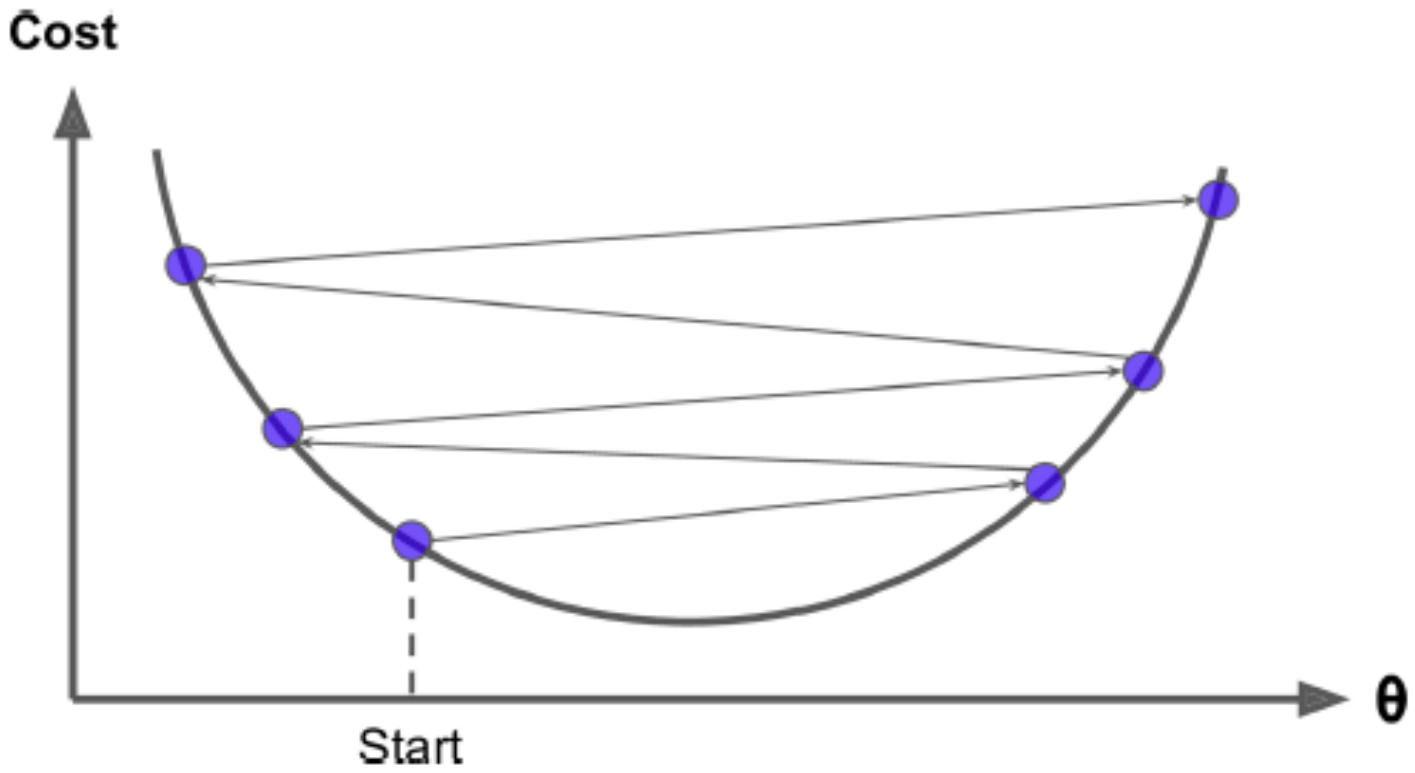
$$x = x - \frac{dx^2}{dx} \qquad (2)$$

$$x = x - 2x \qquad (3)$$

$$x = 2 - 2\times -2 \qquad (4)$$

$$x = 2 + 4 \qquad (5)$$

Update rate (aka learning rate)



Optimize the Learning Rate: α

- **Fixed learning rate**
  - While it is more common to run stochastic gradient descent as we have described it and with a fixed learning rate
- **Dynamic, decreasing learning rate**
  - by slowly letting the learning rate decrease to zero as the algorithm runs, it is also possible to ensure that the parameters will converge to the global minimum rather then merely oscillate around the minimum

- **Or it can be calculated**

$$\alpha = \frac{\mathbf{h}^{\mathbf{T}}\mathbf{h}}{\mathbf{h}^{\mathbf{T}}\mathbf{H}\mathbf{h}} = \frac{\nabla f(x_0)^{\mathbf{T}} \nabla f(x_0^{\mathbf{T}})}{\nabla f(x_0)^{\mathbf{T}} H \nabla f(x_0^{\mathbf{T}})}$$

Gradient Descent

Initialization: Select ϵ and any initial trial solution x'. Go first to the stopping rule.
Iteration:

1. Express $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ as a function of t by setting

$$x_j = x_j' + t\left(\frac{\partial f}{\partial x_j}\right)_{\mathbf{x}=\mathbf{x}'}, \qquad \text{for } j = 1, 2, \ldots, n,$$

and then substituting these expressions into $f(\mathbf{x})$.

2. Use the one-dimensional search procedure (or calculus) to find $t = t^*$ that maximizes $f(\mathbf{x}' + t\,\nabla f(\mathbf{x}'))$ over $t \geq 0$.
3. Reset $\mathbf{x}' = \mathbf{x}' + t^*\,\nabla f(\mathbf{x}')$. Then go to the stopping rule.

Stopping rule: Evaluate $\nabla f(\mathbf{x}')$ at $\mathbf{x} = \mathbf{x}'$. Check if

$$\left|\frac{\partial f}{\partial x_j}\right| \leq \epsilon \qquad \text{for all } j = 1, 2, \ldots, n.$$

If so, stop with the current $\mathbf{x}'$ as the desired approximation of an optimal solution $\mathbf{x}^*$. Otherwise, perform another iteration.

Steepest Descent Method: example

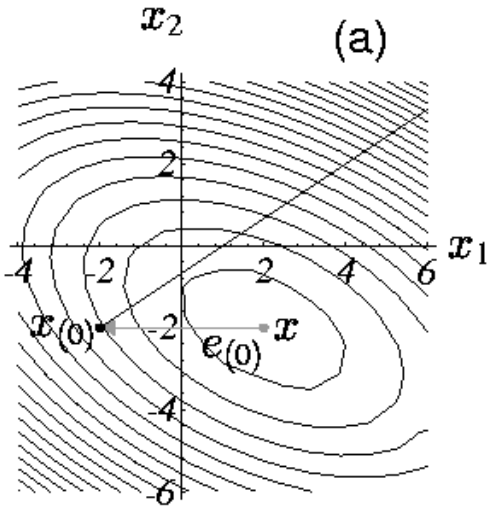
$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

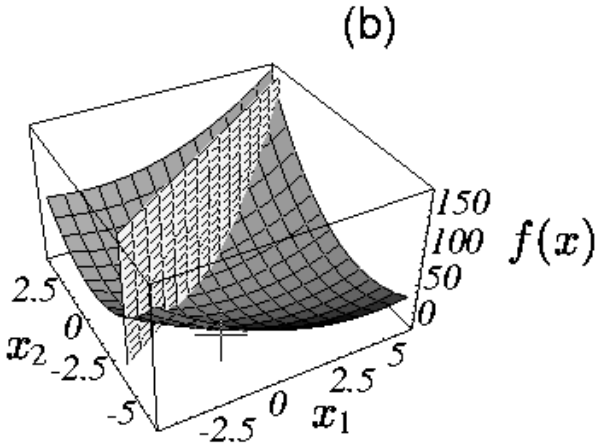
a) Starting at (-2,-2) take the direction of steepest descent of f

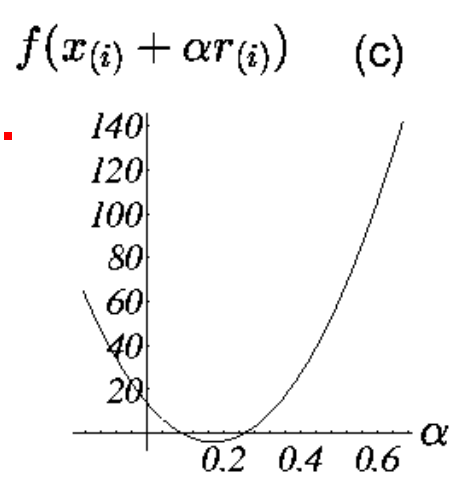
b) Find the point on the intersection of these two surfaces that minimizes f

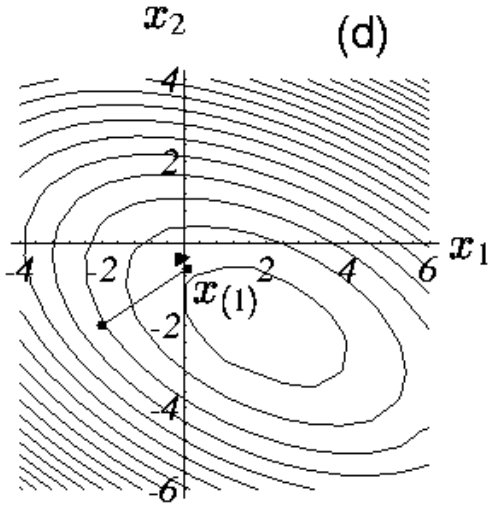
c) Intersection of surfaces (a plane).

d) The gradient at the bottommost point is orthogonal to the gradient of the previous step









Find α so that

$$\varphi(\alpha) = \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$

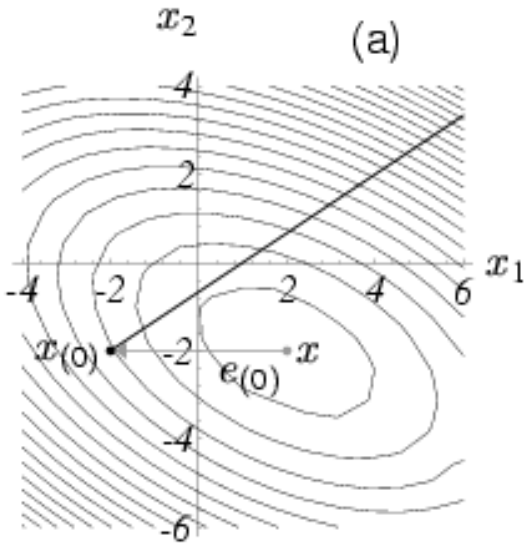
is minumum

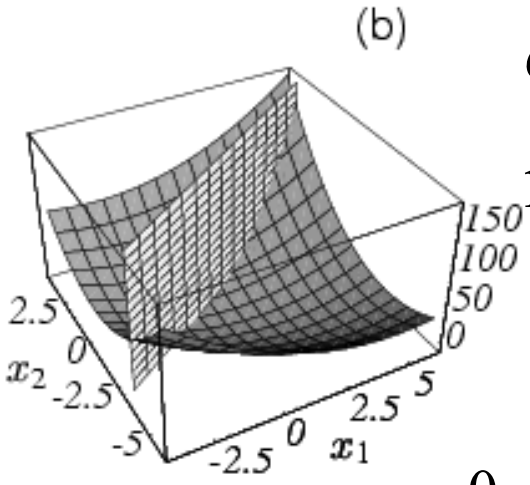
A *line search* is a procedure that chooses α to minimize f along a line. Figure 6(b) illustrates this task: we are restricted to choosing a point on the intersection of the vertical plane and the paraboloid. Figure 6(c) is the parabola defined by the intersection of these surfaces. What is the value of α at the base of the parabola?

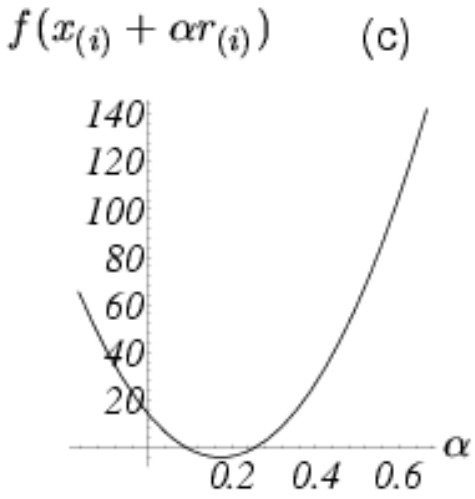
From basic calculus, α minimizes f when the *directional derivative* $\frac{d}{d\alpha} f(x_{(1)})$ is equal to zero. By the chain rule, $\frac{d}{d\alpha} f(x_{(1)}) = f'(x_{(1)})^T \frac{d}{d\alpha} x_{(1)} = f'(x_{(1)})^T r_{(0)}$. Setting this expression to zero, we find that α should be chosen so that $r_{(0)}$ and $f'(x_{(1)})$ are orthogonal (see Figure 6(d)).

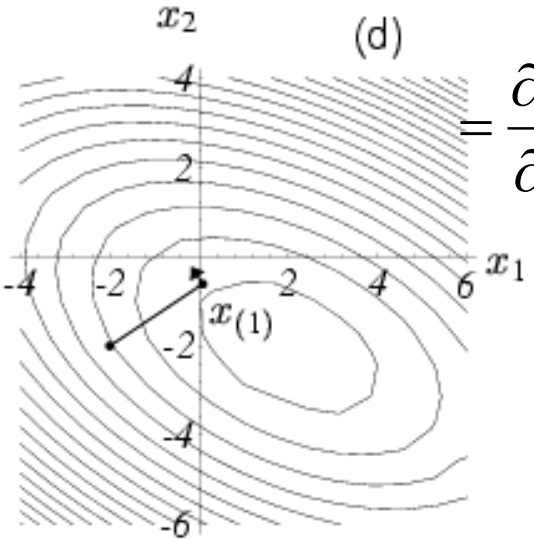
Line search: Find Minimum (2 functions)

$\varphi(\alpha) = F(\mathbf{x}+\alpha\mathbf{h})$, $\mathbf{x}$ and $\mathbf{h}$ fixed, $\alpha \geq 0$.









Find α so that

$$\varphi(\alpha) = \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$

is minumum

Use chain rule df/du*du/dx

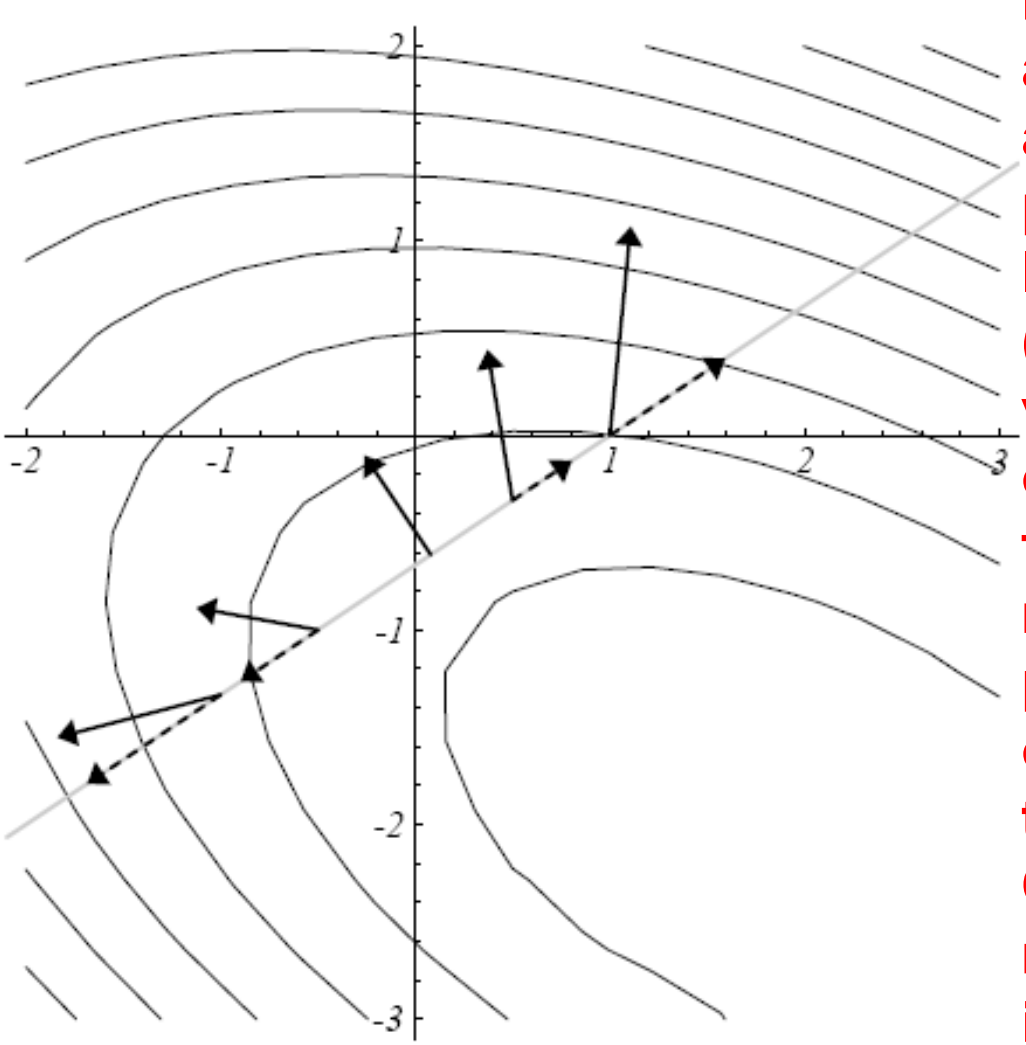
$$0 = \frac{\partial \varphi(\alpha)}{\partial \alpha} = \frac{\partial \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))}{\partial \alpha}$$

$$= \frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha} = \nabla f(\mathbf{x}_0)^{\mathrm{T}} \mathbf{F}'(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$

Line search

$$0 = \frac{\partial \varphi(\alpha)}{\partial \alpha} = \frac{\partial \mathbf{F}(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))}{\partial \alpha}$$

$$= \frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha} = \nabla f(\mathbf{x}_0)^{\mathrm{T}} \mathbf{F}'(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$



[Duda and Hart Stork page 226]

The gradient candidate $\nabla f(x_0)^i$ is shown at several locations along the search line (solid arrows). Each gradient's projection onto the line [in our line search] is also shown (dotted arrows). The gradient vectors represent the direction of steepest increase of f (our function that is being minimized), and the projections represent the rate of increase as one traverses the search line.

On the search line, f is minimized where the gradient is orthogonal to the search line.

Notes on previous two slides

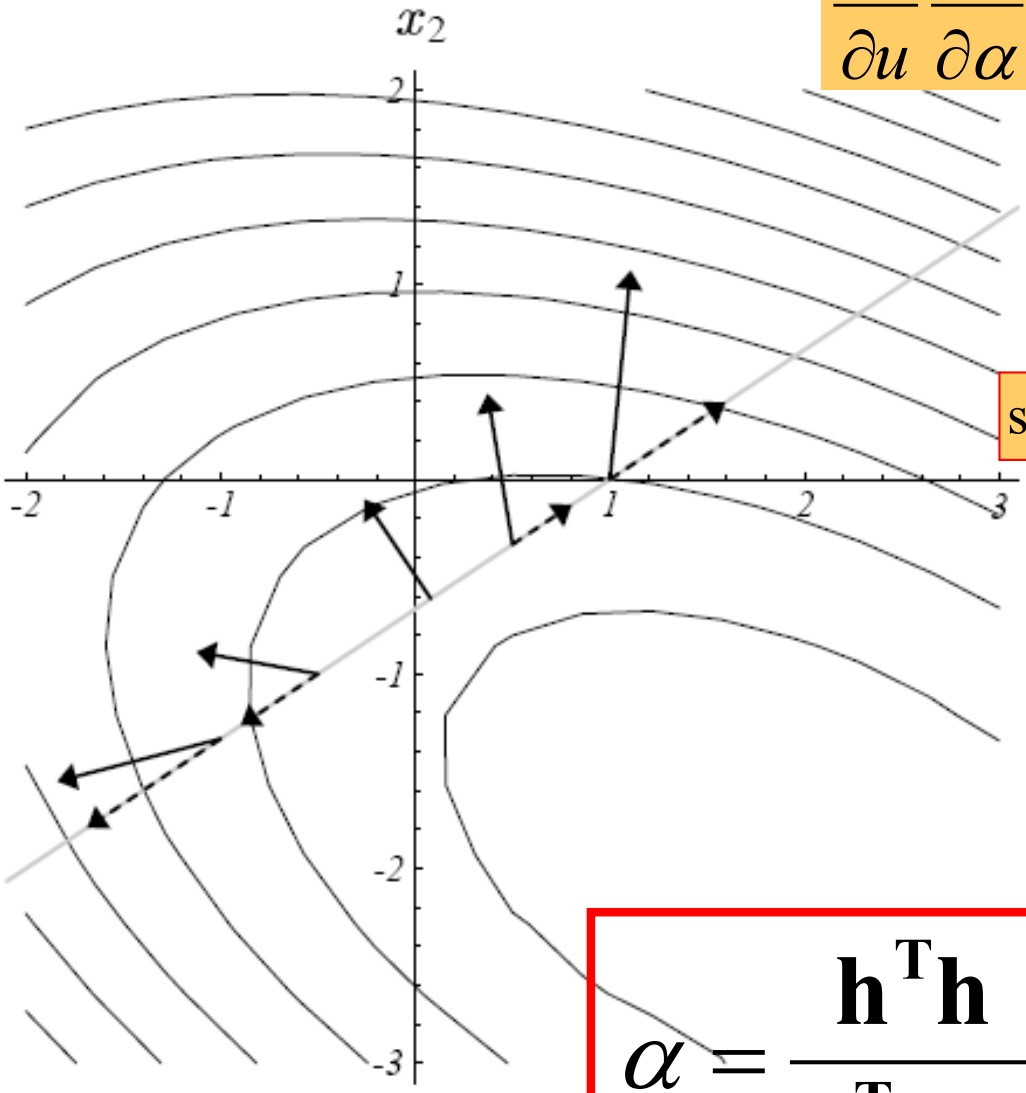
- There is an intuitive reason why we should expect these vectors to be orthogonal at the minimum.
- Figure on previous slide shows the gradient vectors at various points along the search line. The slope of the parabola (Figure (c) of the second previous slide) at any point is equal to the magnitude of the projection of the gradient onto the line (Figure on previous slide,dotted arrows). These projections represent the rate of increase of f as one traverses the search line.
- f is minimized where the projection is zero—where the gradient is orthogonal to the search line.

Line search

$$F(\mathbf{x}+\alpha\mathbf{h}) = F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) + O(\alpha^2)$$
$$\simeq F(\mathbf{x}) + \alpha\mathbf{h}^\top \mathbf{F}'(\mathbf{x}) \quad \text{for } \alpha \text{ sufficiently small.}$$

Taylors Theorem

$$\frac{\partial \mathbf{F}}{\partial u}\frac{\partial u}{\partial \alpha} = \nabla f(\mathbf{x}_0)^{\mathbf{T}} \mathbf{F}'(\mathbf{x}_0 + \alpha \cdot \nabla f(\mathbf{x}_0))$$



$$\text{Let } \mathbf{h} = \nabla f(\mathbf{x}_0)$$

$$\mathbf{h}^{\mathbf{T}} \mathbf{F}'(\mathbf{x}_0 + \alpha\mathbf{h}) = 0$$

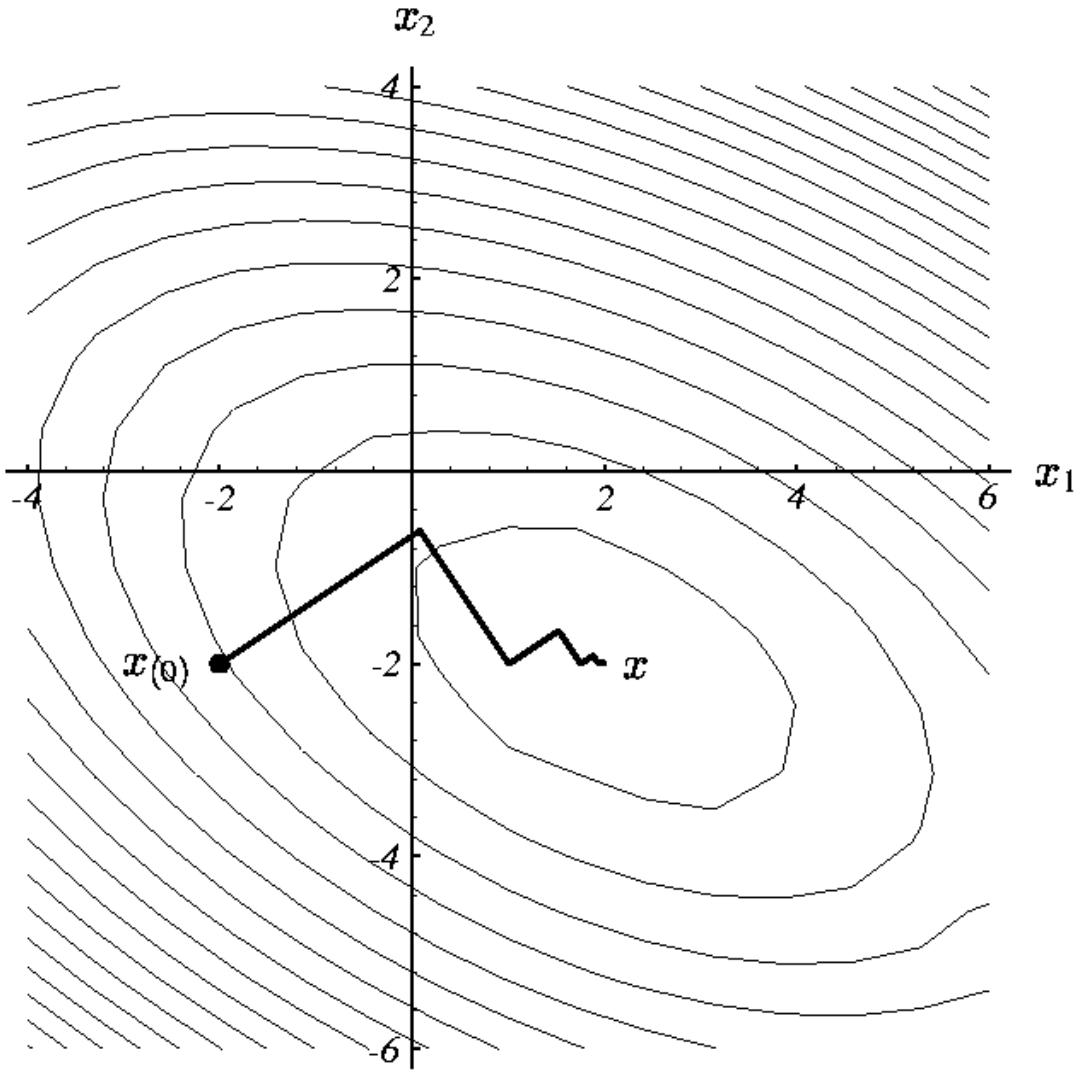
since $\mathbf{F}'(\mathbf{x}_0 + \alpha\mathbf{h}) = \mathbf{F}'(\mathbf{x}_0) + \alpha\mathbf{F}''(\mathbf{x}_0)^{\mathbf{T}}\mathbf{h}$

$$\mathbf{h}^{\mathbf{T}} \mathbf{F}'(\mathbf{x}_0 + \alpha\mathbf{h})$$
$$= \mathbf{h}^{\mathbf{T}} \left(\mathbf{F}'(\mathbf{x}_0) + \alpha\mathbf{F}''(\mathbf{x}_0)^{\mathbf{T}}\mathbf{h} \right)$$
$$= -\mathbf{h}^{\mathbf{T}}\mathbf{h} + \alpha\mathbf{h}^{\mathbf{T}}\mathbf{H}\mathbf{h} = 0$$

$$\alpha = \frac{\mathbf{h}^{\mathbf{T}}\mathbf{h}}{\mathbf{h}^{\mathbf{T}}\mathbf{H}\mathbf{h}} = \frac{\nabla f(x_0)^{\mathbf{T}} \nabla f(x_0^{\mathbf{T}})}{\nabla f(x_0)^{\mathbf{T}} H \nabla f(x_0^{\mathbf{T}})}$$

[Duda and Hart Stork page 226]

Iterations of Steepest Descent Method



Gradient Descent Example

Example. Consider the following two-variable problem:

Maximize $f(\mathbf{x}) = 2x_1x_2 + 2x_2 - x_1^2 - 2x_2^2$.

Thus,

$$\frac{\partial f}{\partial x_1} = 2x_2 - 2x_1,$$
$$\frac{\partial f}{\partial x_2} = 2x_1 + 2 - 4x_2.$$

2(0) – 2(0)=0
2(0) + 2 – 2(0)=2
$\nabla f^1 = (0,2)$

We also can verify (see Appendix 2) that $f(\mathbf{x})$ is concave. To begin the gradient search procedure, suppose that $\mathbf{x} = (0, 0)$ is selected as the initial trial solution. Because the respective partial derivatives are 0 and 2 at this point, the gradient is

$\nabla f(0, 0) = (0, 2)$. $\nabla f(x_1)$ $X^1=(0,0)$

Therefore, to begin the first iteration, set

$$x_1 = 0 + t(0) = 0,$$
$$x_2 = 0 + t(2) = 2t,$$

and then substitute these expressions into $f(\mathbf{x})$ to obtain

$$f(\mathbf{x}' + t\,\nabla f(\mathbf{x}')) = f(0, 2t) = 2(0)(2t) + 2(2t) - 0^2 - 2(2t)^2 = 4t - 8t^2.$$

Linesearch

Because

$$f(0, 2t^*) = \max_{t \geq 0} f(0, 2t) = \max_{t \geq 0} \{4t - 8t^2\}$$

and

$$\frac{d}{dt}(4t - 8t^2) = 4 - 16t = 0,$$

it follows that

$$t^* = \frac{1}{4},$$

$$\alpha = \frac{\mathbf{h}^T\mathbf{h}}{\mathbf{h}^T\mathbf{H}\mathbf{h}} = \frac{\nabla f(x_0)^T \nabla f(x_0^T)}{\nabla f(x_0)^T H \nabla f(x_0^T)}$$

$$\alpha = \frac{(0,2)^T(0,2)}{(0,2)^T\begin{bmatrix} -2 & 2 \\ 2 & -4 \end{bmatrix}(0,2)} = \frac{4}{(2,-8)^T(0,2)} = -\frac{4}{16}$$

so

Reset $\mathbf{x}' = (0, 0) + \frac{1}{4}(0, 2) = \left(0, \frac{1}{2}\right)$. x^2

For this new trial solution, the gradient is

$\nabla f\left(0, \frac{1}{2}\right) = (1, 0)$. $\nabla f(x_2)$ Gradient at x

Thus, for the second iteration, set

$$\mathbf{x} = \left(0, \frac{1}{2}\right) + t(1, 0) = \left(t, \frac{1}{2}\right),$$

so

$$f(\mathbf{x}' + t\,\nabla f(\mathbf{x}')) = f\left(0 + t, \frac{1}{2} + 0t\right) = f\left(t, \frac{1}{2}\right) = (2t)\left(\frac{1}{2}\right) + 2\left(\frac{1}{2}\right) - t^2 - 2\left(\frac{1}{2}\right) = t - t^2 + \frac{1}{2}.$$

Because

$$f\left(t^*, \frac{1}{2}\right) = \max_{t \geq 0} f\left(t, \frac{1}{2}\right) = \max_{t \geq 0} \left\{t - t^2 + \frac{1}{2}\right\}$$

and

$$\frac{d}{dt}\left(t - t^2 + \frac{1}{2}\right) = 1 - 2t = 0,$$

then

$$t^* = \frac{1}{2},$$

so

Reset $\mathbf{x}' = \left(0, \frac{1}{2}\right) + \frac{1}{2}(1, 0) = \left(\frac{1}{2}, \frac{1}{2}\right)$. x^3

Maximum vs Minimum

For unconstrained opt

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

f'(x=x*) =0 at maximum and minimum

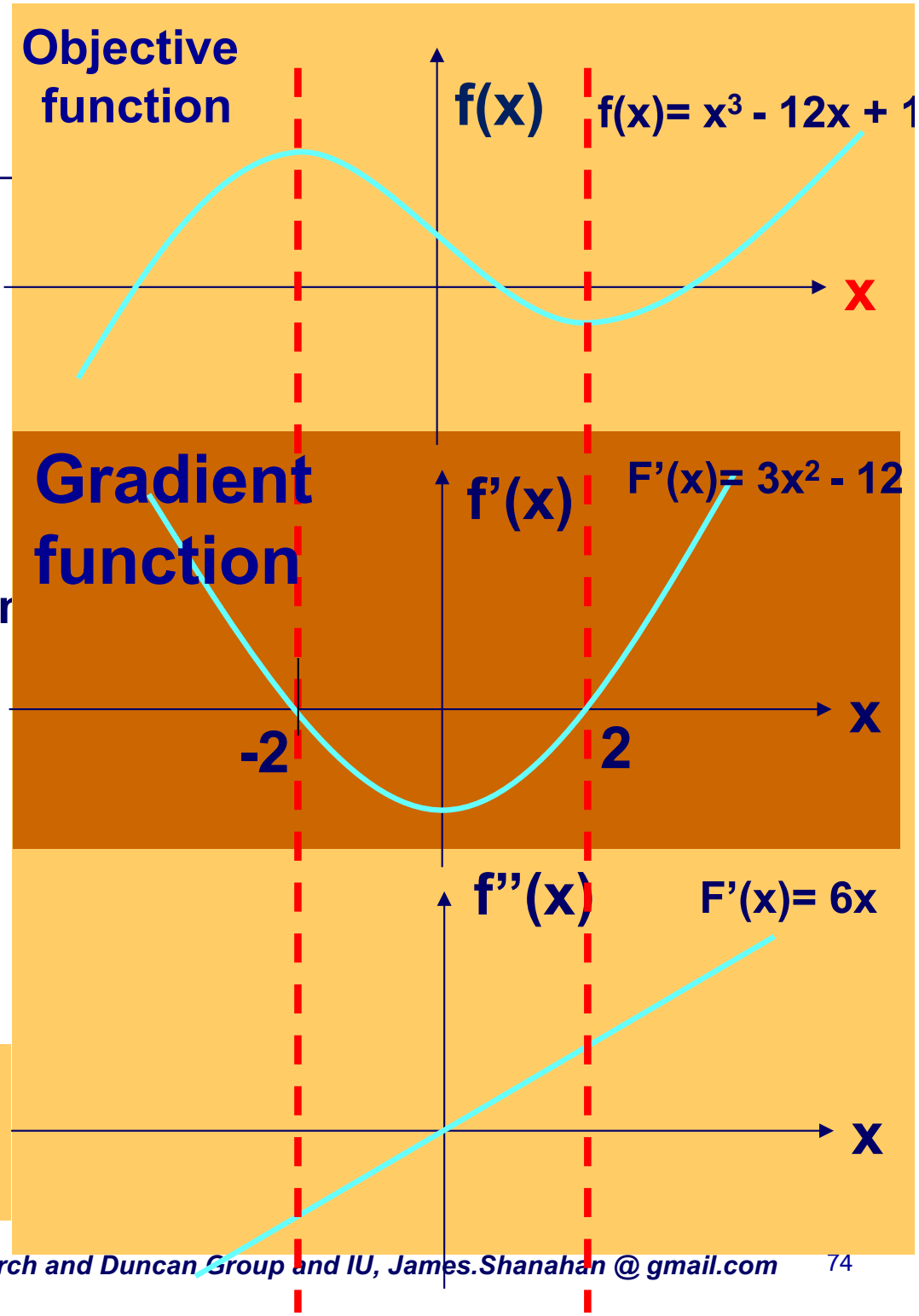
These zero points are the roots!

Second derivative f''(x) = 6x

Gives the "rate of change of gradient"

SOC

If f''(x=x*) < 0 then maximum
If f''(x=x*) > 0 then minimum
If f''(x=x*) >= 0 then ???



Maximum vs Minimum

For unconstrained opt

$f(x) = x^3 - 12x$

First derivati

$f'(x) = 3x^2 - 12$

FOC

f'(x=x*) =0 at maximum

These zero points are th

Second derivative f

Gives the "rate

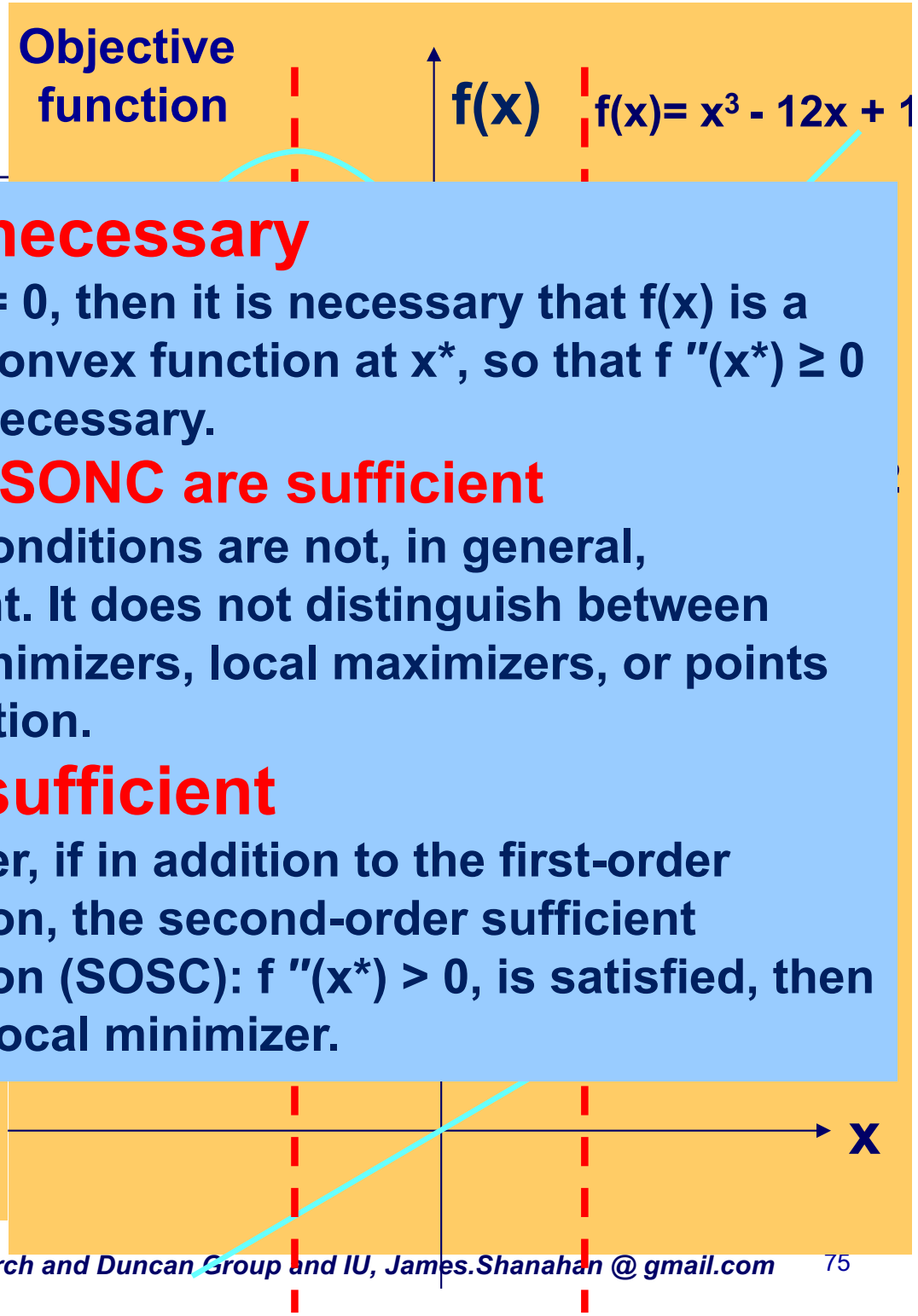
change of gradi

SOC

If f''(x=x*) < 0 then maximum

If f''(x=x*) > 0 then minimum

If f''(x=x*) >= 0 then ???



SONC: necessary

- If $f'(x^*) = 0$, then it is necessary that f(x) is a locally convex function at x*, so that $f''(x^*) \geq 0$ is also necessary.

FOC and SONC are sufficient

- These conditions are not, in general, sufficient. It does not distinguish between local minimizers, local maximizers, or points of inflection.

SOSC: sufficient

- However, if in addition to the first-order condition, the second-order sufficient condition (SOSC): $f''(x^*) > 0$, is satisfied, then x* is a local minimizer.

FOC vs SONC vs SOSC

A differentiable function f of one variable defined on an interval $F = [a\ e]$. If an interior-point $\bar{x}$ is a local/global minimizer, then $f'(\bar{x}) = 0$; if the left-end-point $\bar{x} = a$ is a local minimizer, then $f'(a) \geq 0$; if the right-end-point $\bar{x} = e$ is a local minimizer, then $f'(e) \leq 0$. first-order necessary condition (FONC) summarizes the three cases by complementarity conditions:

$$a \leq x \leq e,\ f'(x) = y^a + y^e,\ y^a \geq 0,\ y^e \leq 0,\ y^a(x-a) = 0,\ y^b(x-e) = 0.$$

If $f'(\bar{x}) = 0$, then it is necessary that $f(x)$ is a locally convex function at $\bar{x}$, so that $f''(\bar{x}) \geq 0$ is also necessary. This is called the second-order necessary condition (SONC), which we would explored further.

These conditions are not, in general, sufficient. It does not distinguish between local minimizers, local maximizers, or points of inflection. However, if in addition to the first-order condition, the second-order sufficient condition (SOSC): $f''(\bar{x}) > 0$, is satisfied, then $\bar{x}$ is a local minimizer.

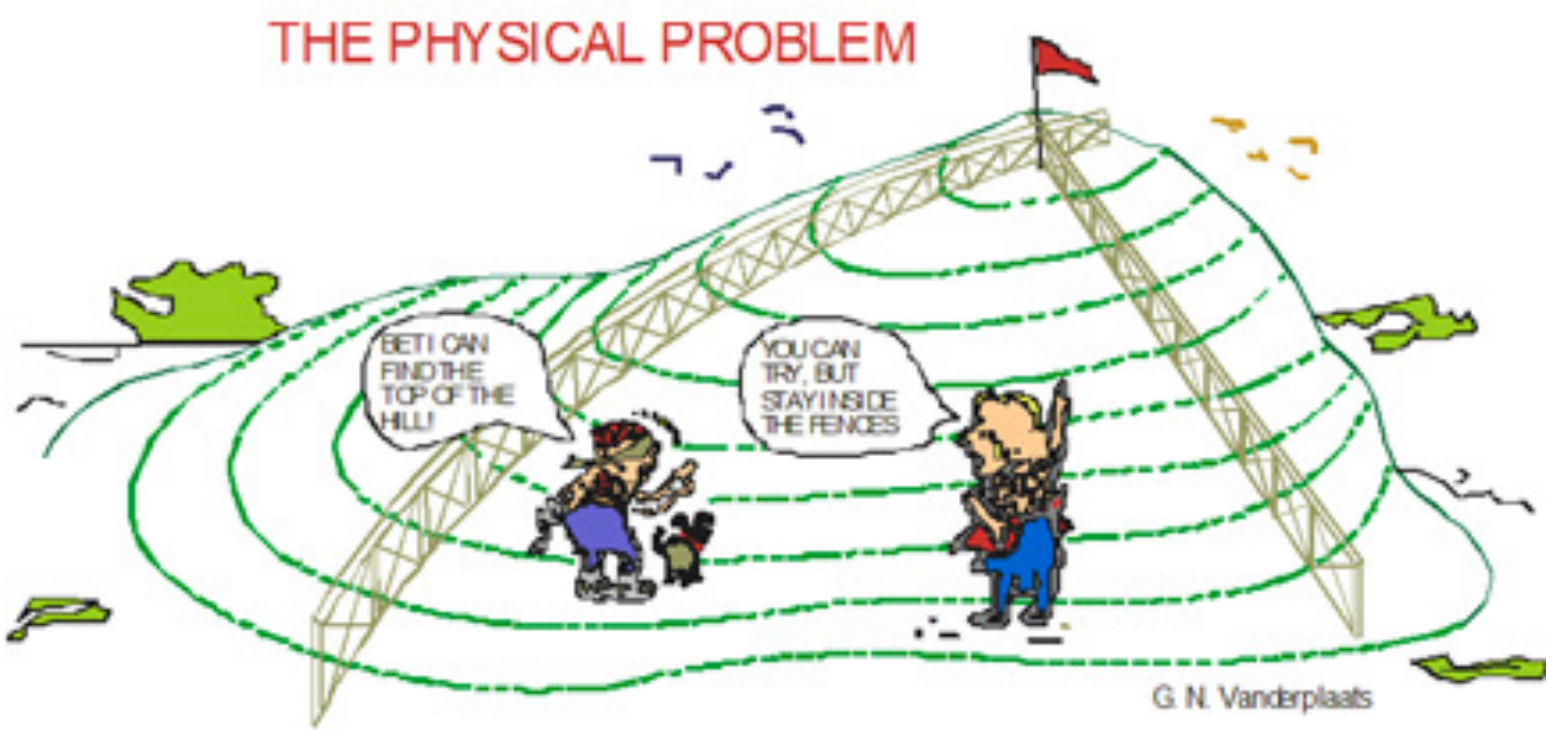
If the function is convex, the first-order necessary condition is already sufficient.

https://web.stanford.edu/class/msande311/lecture06.pdf

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Terminology



Maximize: $Y = f(X_1, X_2)$ (objective)

Subject to:

$$\left.\begin{aligned} F_1 &= f_1(X_1, X_2) \le 0 \\ F_2 &= f_2(X_1, X_2) \le 0 \end{aligned}\right\}$$

(constraints)

$$X = \begin{Bmatrix} X_1 \\ X_2 \end{Bmatrix}$$

(design variables)

‹ DERIVATIVES OF MULTIVARIABLE FUNCTIONS

Partial derivative and gradient (articles)

- Introduction to partial derivatives
- Second partial derivatives
- The gradient
- Directional derivatives (introduction)
- Directional derivatives (going deeper)

Next tutorial
Differentiating parametric curves

The gradient

The gradient stores all the partial derivative information of a multivariable function. But it's more than a mere storage device, it has several wonderful interpretations and many, many uses.

What you need to be familiar with before starting this lesson:

- Partial derivatives
- Vector fields
- Contour maps—only necessary for one section of this lesson.

What we're building toward

- The gradient of a scalar-valued multivariable function $f(x, y, \dots)$, denoted ∇f, packages all its partial derivative information into a vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

In particular, this means ∇f is a vector-valued function.

Unconstrained multivariate optimization

- **Analytical algorithm**

- **Find optima using iterative numerical algorithms**

Characterizing Extrema in Multi-Dims

- **FOC**
  - Gradient function is zero (i.e., the objective function is at an extremum). A vector of zeros
- **Is x^* an optimum?**
- **SOC**
  - Can be established by checking if the Hessian is positive definite (second partial derivative f''(x*) i.e., $\delta^2 F/dx_i dx_J$ evaluated at x*)
  - A sufficient condition for an extreme point x* (i.e., $F(x^*) = 0$) to be a minimum is to have a positive definite Hessian at x* (i.e., has positive (nonzero) eigenvalues)

Characterizing Extrema in Multi-Dims

A sufficient condition for a stationary point x^* to be an extreme point is that the matrix of second partial derivatives (Hessian matrix) of $f(x)$ evaluated at x^* is:

(i) Positive definite when x^* is a relative minimum point.
(ii) Negative definite when x^* is a relative maximum point.

For more details see:
Optimization Methods For Engineers
- **Publisher:** PHI Learning
- **ISBN:** 9788120347441
- **Author:** R.V.S. Raju

Hessian matrix

- **The Hessian matrix of a function of n independent variables f (x_1, x_2, …, x_n) is the matrix of second derivatives of f with respect to the independent variables. That is:**

$$H_f = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nn} \end{bmatrix}$$

We will usually omit naming the function in question when there's no room for doubt, so in the above case we'd just use H (instead of H_f) to refer to the Hessian of the function f.

6.5 Working Rule (Algorithm) for Unconstrained Multivariable Optimization

From Theorems 6.1 and 6.2, we can obtain the following procedure to solve the problems of multivariable functions without constraints.

Step 1: Check the function to ensure that it belongs to the category of multivariable unconstrained optimization problems, i.e. it should contain more than one variable, say, $x_1, x_2, \ldots, x_n$ and no constraints.

Step 2: Find the first partial derivatives of the function w.r.t. $x_1, x_2, \ldots, x_n$.

Step 3: Equate all the first partial derivatives (obtained from Step 2) to zero to find the values of $x_1^*, x_2^*, \ldots, x_n^*$.

Step 4: Find all the second partial derivatives of the function.

Step 5: Prepare the Hessian matrix. For example, if we have two variables x_1, x_2, in $f(x)$, then the Hessian matrix is

$$J_{(x_1^*, x_2^*)} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

Step 6: Find the values of determinants of square sub-matrices

$$J_1, J_2, \ldots, J_n$$

In our example,
$$J_1 = \left| \frac{\partial^2 f}{\partial x_1^2} \right|$$

and
$$J_2 = \begin{vmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} \end{vmatrix}$$

Step 7: Evaluate whether x_1^*, x_2^* are relative maxima or minima depending on the positive or negative nature of J_1, J_2 and so on.
Thus, the sign of J_1 decides whether J is positive or negative while the sign of J_2 decides definite or indefiniteness of J (Table 6.1). More clearly,

If (i) $J_1 > 0, J_2 > 0$, then J is positive definite and x^* is a relative minimum.
(ii) $J_1 > 0, J_2 < 0$, then J is indefinite and x^* is a saddle point.
(iii) $J_1 < 0, J_2 > 0$, then J is negative definite and x^* is a relative maximum.
(iv) $J_1 < 0, J_2 < 0$, then J is indefinite and x^* is a saddle point.

Table 6.1 presents the sign notation of J.

Table 6.1 Sign Notation of J

J_2 \ J_1	J_1 positive or $J_1 > 0$ (J is positive)	J_1 negative or $J_1 < 0$ (J is negative)
J_2 positive or $J_2 > 0$ (J is definite)	J is positive definite and x^* is a relative minimum	J is negative definite and x^* is a relative maximum
J_2 negative or $J_2 < 0$ (J is indefinite)	J is indefinite and x^* is a saddle point	J is indefinite and x^* is a saddle point

Analytical Algo for unconstrained multivariate optimization

7 Steps

Analytical Algo for unconstrained multivariate optimization

6.5 Working Rule (Algorithm) for Unconstrained Multivariable Optimization

From Theorems 6.1 and 6.2, we can obtain the following procedure to solve the problems of multivariable functions without constraints.

Step 1: Check the function to ensure that it belongs to the category of multivariable unconstrained optimization problems, i.e. it should contain more than one variable, say, $x_1, x_2, \ldots, x_n$ and no constraints.

Step 2: Find the first partial derivatives of the function w.r.t. $x_1, x_2, \ldots, x_n$.

Step 3: Equate all the first partial derivatives (obtained from Step 2) to zero to find the values of $x_1^*, x_2^*, \ldots, x_n^*$.

Step 4: Find all the second pa

Step 5: Prepare the Hessian m
then the Hessian matri

FOC

Step 6: Find

In our example,

and

Step 7: Evaluate whether x_1^*, x

SOSC

(ii) $J_1 > 0, J_2 < 0,$
(iii) $J_1 < 0, J_2 > 0,$
(iv) $J_1 < 0, J_2 < 0,$ then J is indefinite and x^* is a saddle point.

Table 6.1 presents the sign notation of J.

Table 6.1 Sign Notation of J

J_2 \ J_1	J_1 positive or $J_1 > 0$ (J is positive)	J_1 negative or $J_1 < 0$ (J is negative)
J_2 positive or $J_2 > 0$ (J is definite)	J is positive definite and x^* is a relative minimum	J is negative definite and x^* is a relative maximum
J_2 negative or $J_2 < 0$ (J is indefinite)	J is indefinite and x^* is a saddle point	J is indefinite and x^* is a saddle point

Step 1: Check the function to ensure that it belongs to the category of multivariable unconstrained optimization problems, i.e. it should contain more than one variable, say, $x_1, x_2, \ldots, x_n$ and no constraints.

Step 2: Find the first partial derivatives of the function w.r.t. $x_1, x_2, \ldots, x_n$.

Step 3: Equate all the first partial derivatives (obtained from Step 2) to zero to find the values of $x_1^*, x_2^*, \ldots, x_n^*$.

Step 4: Find all the second partial derivatives of the function.

Step 5: Prepare the Hessian matrix. For example, if we have two variables x_1, x_2, in $f(x)$, then the Hessian matrix is

$$J_{(x_1^*, x_2^*)} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \dfrac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$$

Step 6: Find the values of determinants of square sub-matrices $J_1, J_2, \ldots, J_n$

A sufficient condition for an extreme point x* (i.e., F(x*) = 0) to be a minimum is to have a positive definite Hessian at x*

The values of J_1 and J_2 and the nature of the extreme points are given in Table 6.2.

Table 6.2 Sign Notation of J in Illustration 6.2

Point x	Value of J_1	Value of J_2	Nature of J	Nature of x	$f(x)$
(0, 0)	+ 4	+ 32	Positive definite	Relative minimum	6
(0, −8/3)	+ 4	−32	Indefinite	Saddle point since $\partial f/\partial x_1 = \partial f/\partial x_2$	418/27
(−4/3, 0)	−4	−32	Indefinite	Saddle point	194/27
(−4/3, −8/3)	−4	+ 32	Negative definite	Relative maximum	50/3

Example: Analytical Algo. for unconstraine multivariate optimization

Illustration 6.3 Find the extreme values of the function $f(x_1, x_2) = x_1^2 - x_2^2$.

Solution Applying the necessary conditions $\partial f/\partial x_1 = 0$ and $\partial f/\partial x_2 = 0$, we have

$$\frac{\partial f}{\partial x_1} = 2x_1 = 0 \quad \Rightarrow \quad x_1 = 0$$

$$\frac{\partial f}{\partial x_2} = -2x_2 = 0 \quad \Rightarrow \quad x_2 = 0$$

FOC

Now, applying the sufficient conditions

$$\frac{\partial^2 f}{\partial x_1^2} = 2, \quad \frac{\partial^2 f}{\partial x_2^2} = -2 \quad \text{and} \quad \frac{\partial^2 f}{\partial x_1 \partial x_2} = 0$$

Therefore, the Hessian matrix $J = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$

Thus at $x_1 = 0$, $x_2 = 0$.

We have $J_1 = |2| = 2$ and $J_2 = \begin{vmatrix} 2 & 0 \\ 0 & -2 \end{vmatrix} = -4$

SOSC are not met

Since J_1 is positive (+2) and J_2 is negative (−4), the nature of J is indefinite and the point at $x_1 = 0$, $x_2 = 0$ is a saddle point, and also the value the function at this saddle point is $f(0,0) = 0$.

Note: At the saddle point if one of the values of x_1 or x_2 is fixed, the other may show extremity.

Lecture Outline

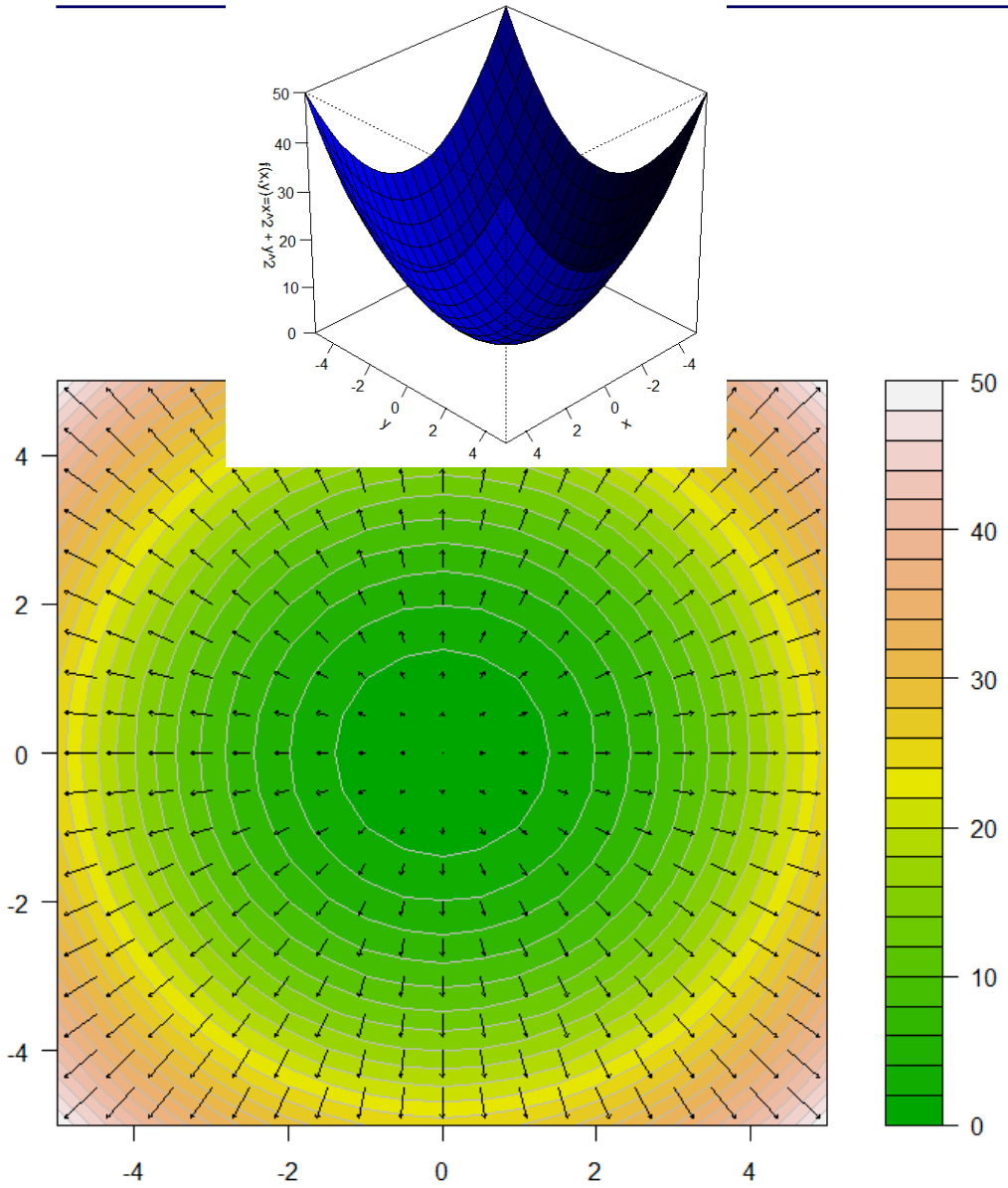
- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

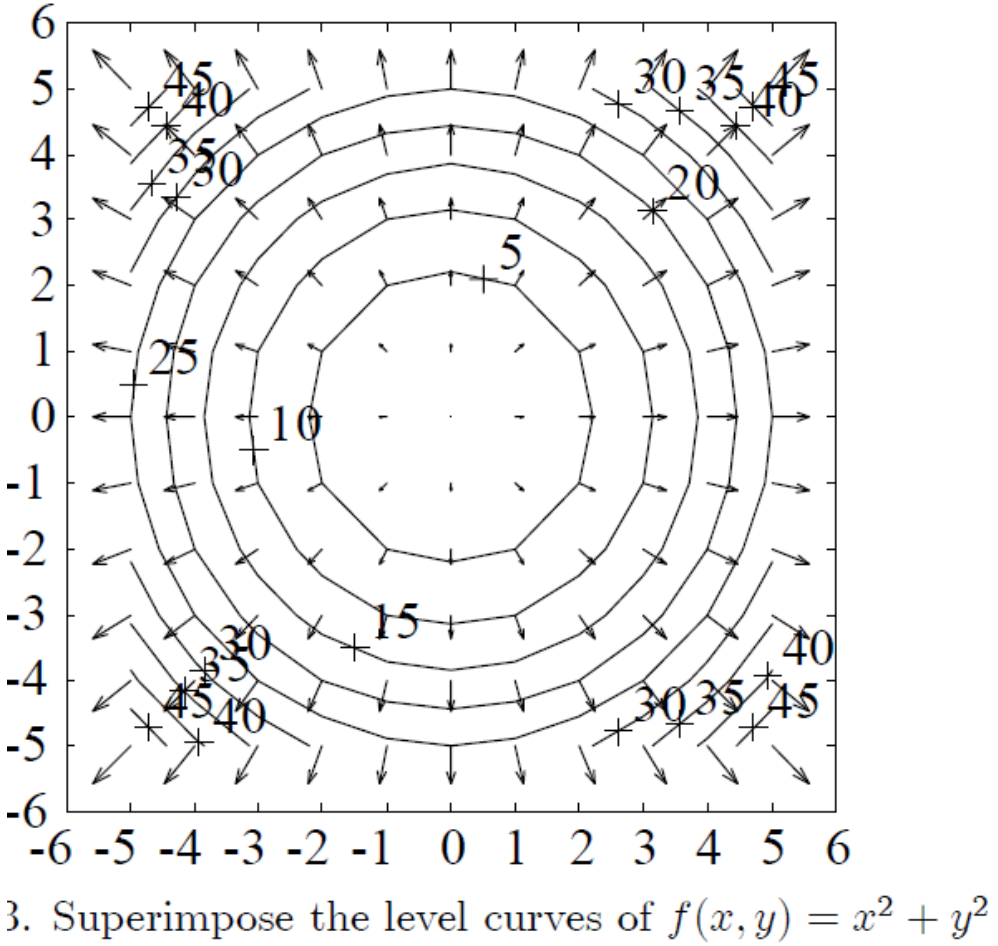
Unconstrained multivariate optimization

- **Analytical algorithm**

- **Find optima using iterative numerical algorithms**

Gradient Vector Plots of f(x,y)=x^2+y^2



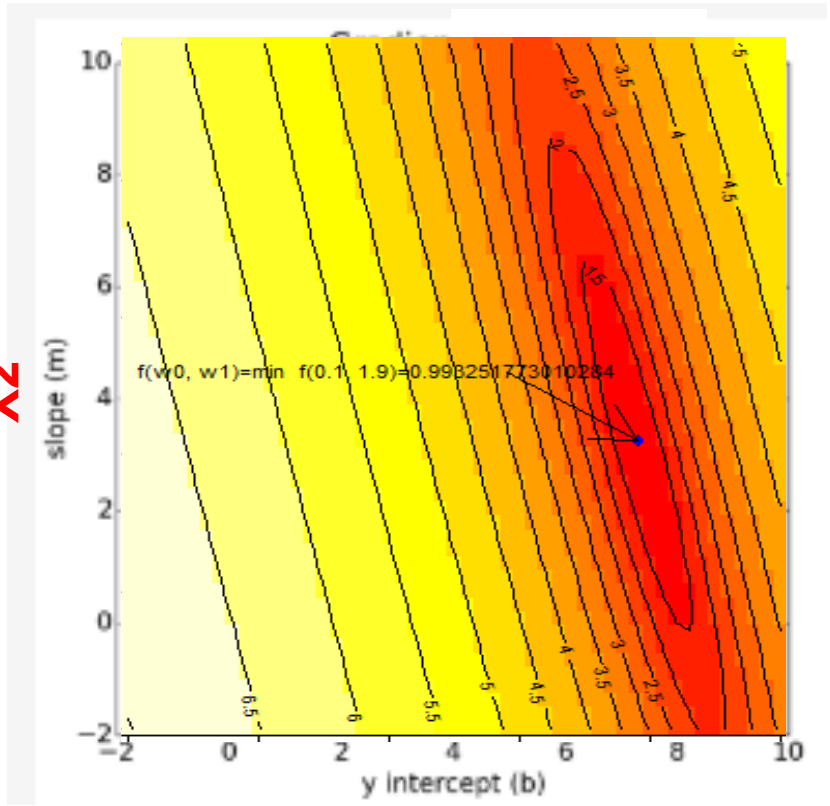


3. Superimpose the level curves of $f(x, y) = x^2 + y^2$

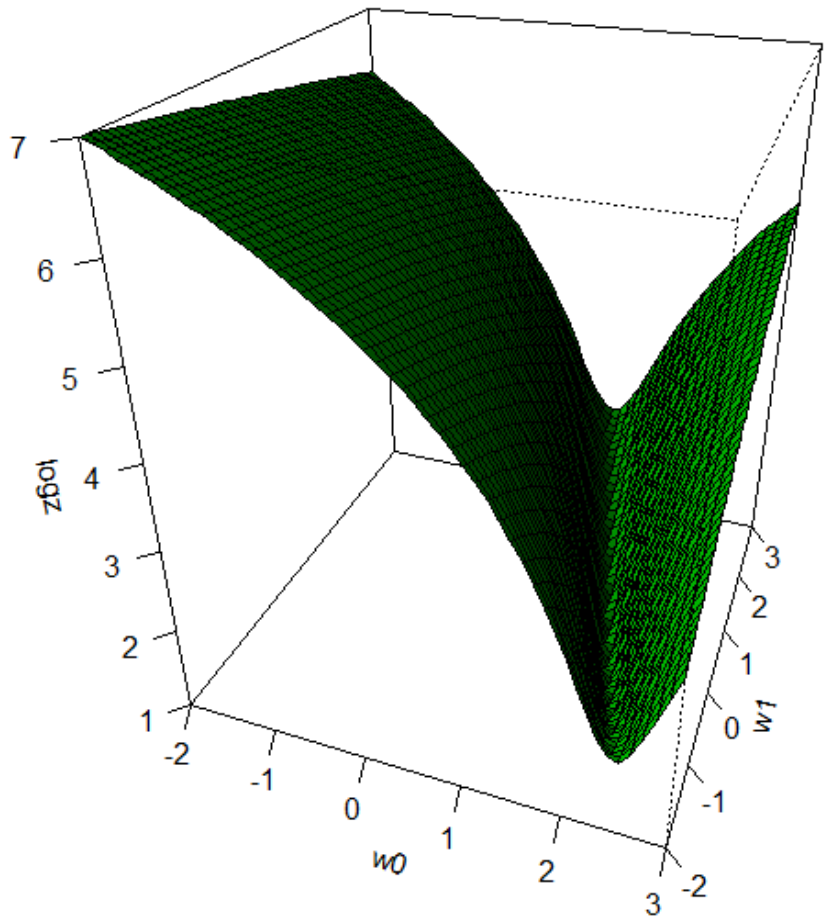
http://online.redwoods.cc.ca.us/instruct/darnold/MULTCALC/grad/grad.pdf

Function

F(X1, X2)

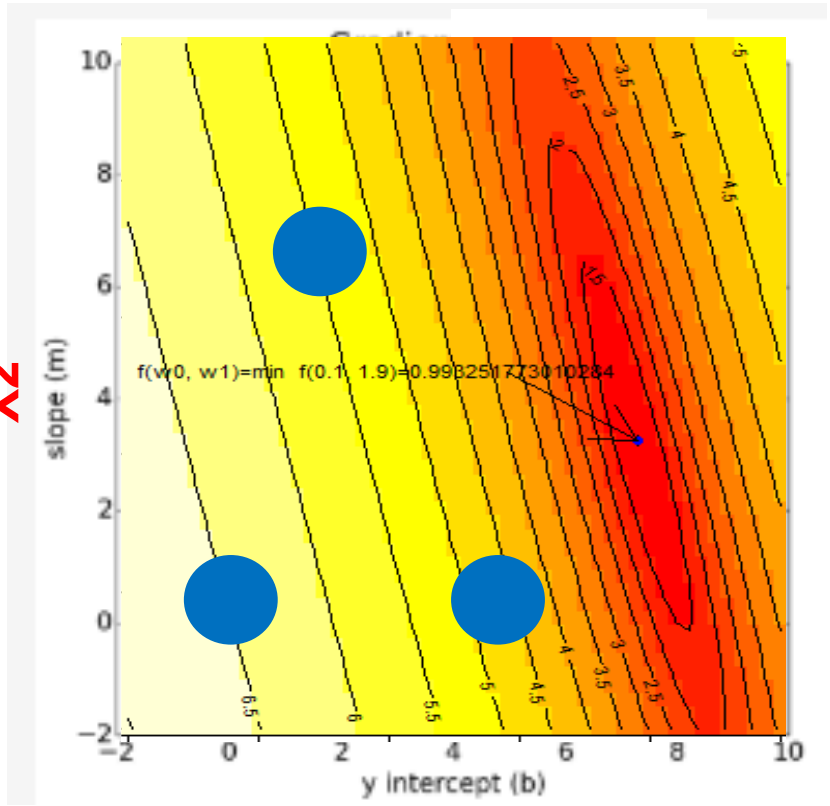


X1



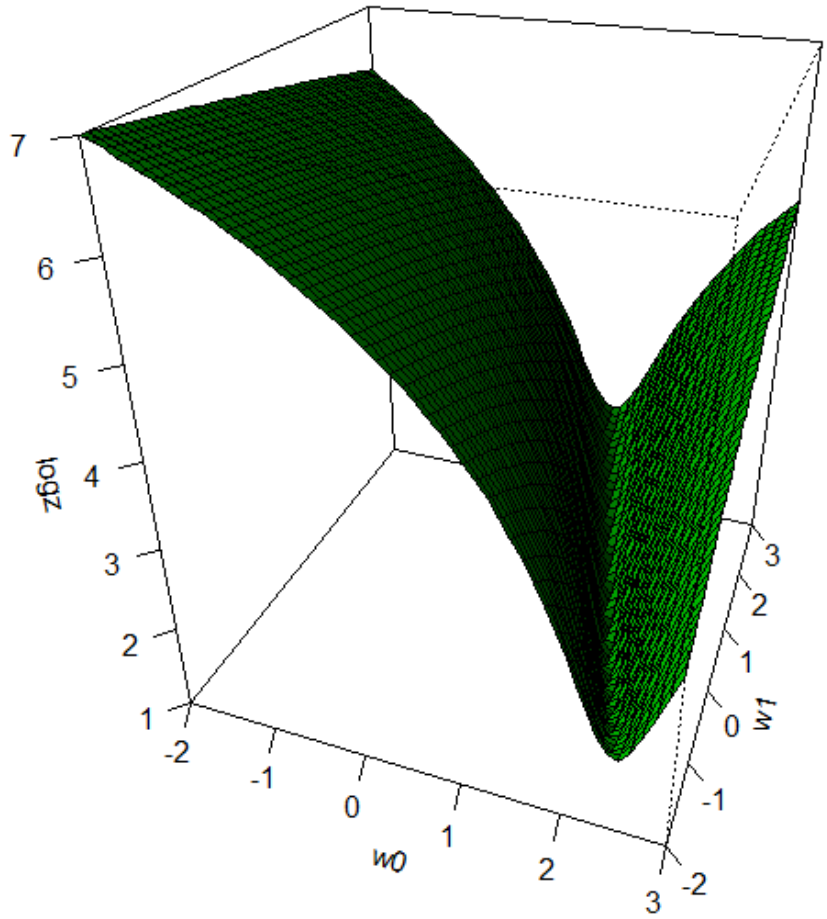
Explore randomly

F(X1, X2)



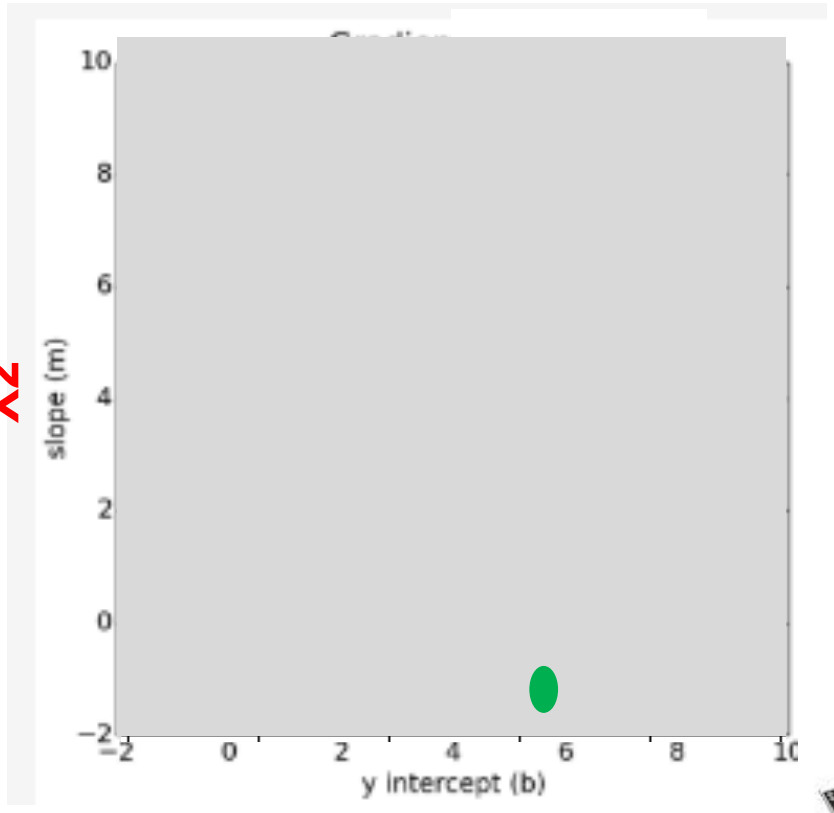
X2

X1



Explore using the gradient compass

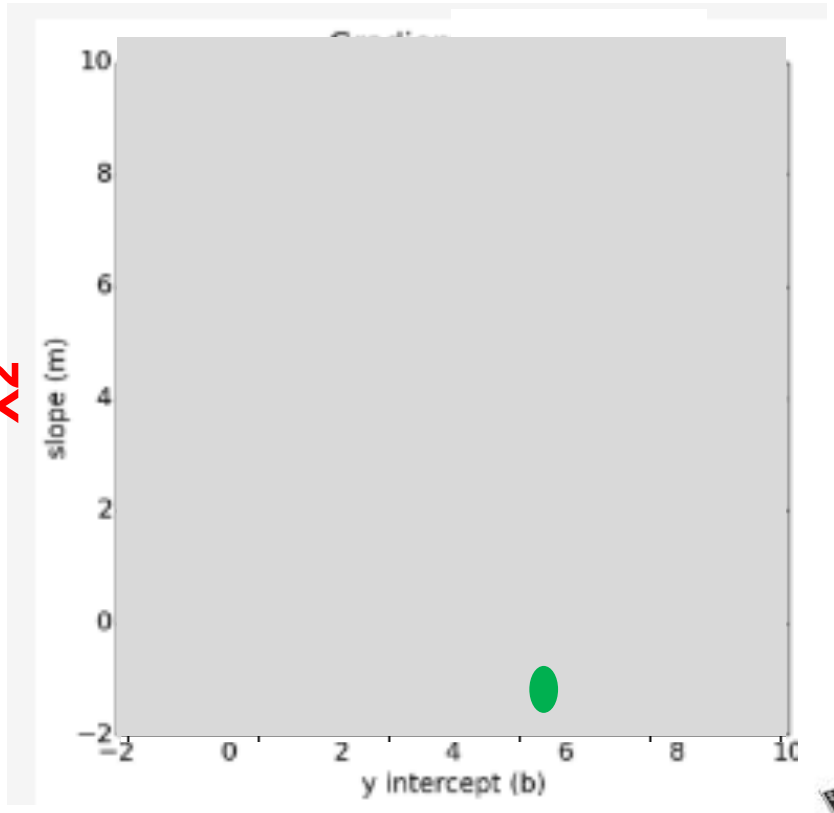
F(X1, X2)



tells which way is up/down without a map

Explore using the gradient compass

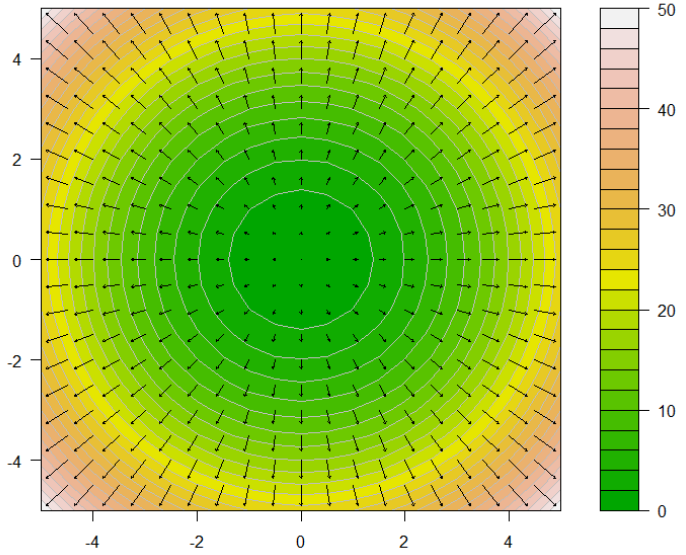
F(X1, X2)



X2

X1

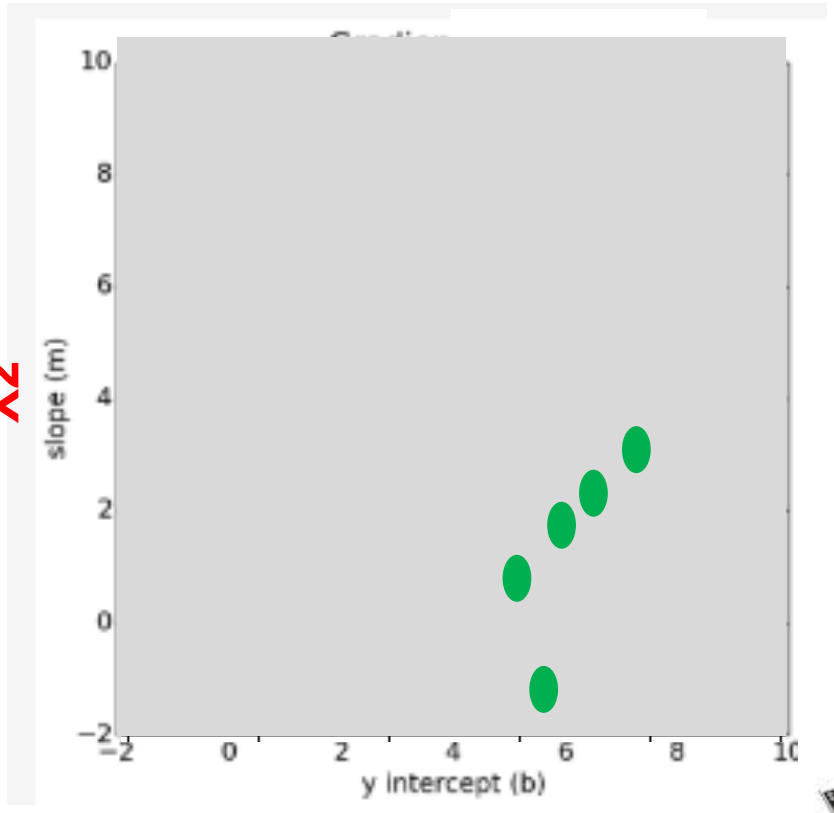
tells which way is up/down without a map



Gradient vector plo

Explore using the gradient compass

F(X1, X2)



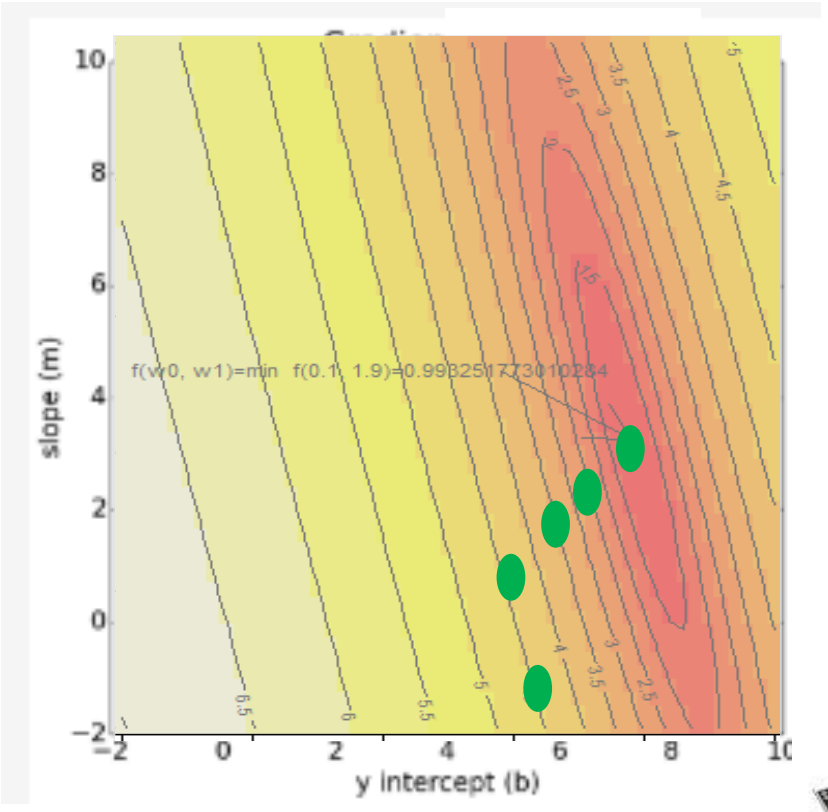
X2

X1

tells which way is up/down without a map

Explore using the gradient compass

F(X1, X2)



tells which way is up/down without a map

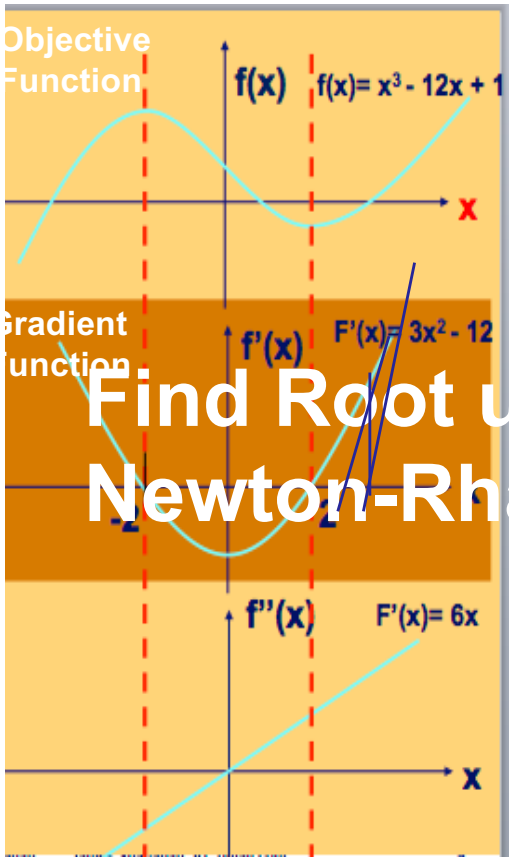
X1

Newton-Raphson

GIVEN: Minimize f(x)

STEP 1: Find the zeros of the gradient function f'(x)

- **Using Bisection method; OR Newton-Raphson; OR Gradient Descent**



$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \left[H(x^i)\right]^{-1} J(x^i) \qquad \text{Multivariate Case}$$

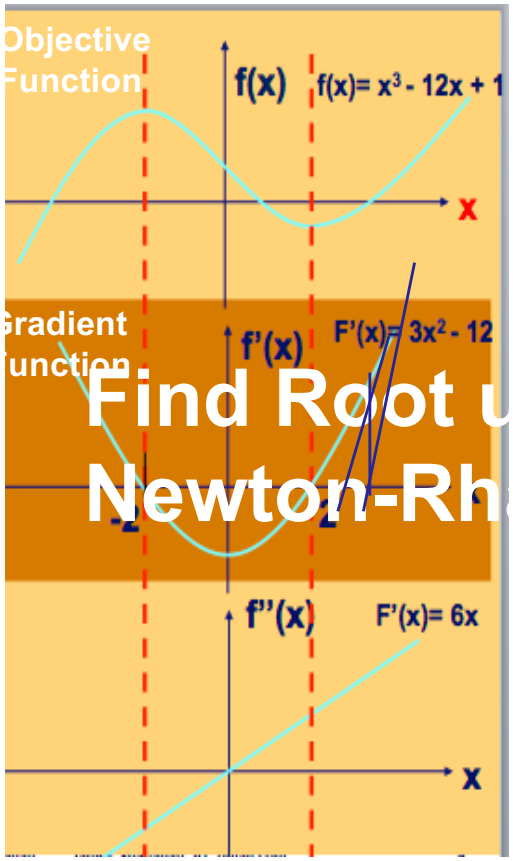
Calculating f''(x), the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

Gradient Descent (a simpler root finder)

GIVEN: Minimize f(x)

STEP 1: Find the zeros of the gradient function f'(x)

- **Using Bisection method; OR Newton-Raphson; OR Gradient Descent**



$$x^{i+1} = x^i - \left[f''(x^i)\right]^{-1} f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \left[H(x^i)\right]^{-1} J(x^i) \qquad \text{Multivariate Case}$$

Newton-Ra

Calculating f''(x), the Hessian H in multivariate case, and inverting it is complex so simpler algorithms have been developed such as gradient descent

learning rate

$$x^{i+1} = x^i - \alpha^i f'(x^i) \qquad \text{Univariate case}$$

$$x^{i+1} = x^i - \alpha^i J(x^i) \qquad \text{Multivariate Case}$$

Gradient Descent

How large should I step in the positive gradient direction (gradient ascent)

- or in the negative gradient direction (gradient descent)

- **Convergence criteria. E.g., decision vector X does not change that much or error does not change**

Gradient Descent in a nutshell

- http://www2.econ.iastate.edu/classes/econ500/hallam/documents/Opt_Simple_Multi_000.pdf

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Constrained optimization

- **Mathematical optimization problem:**

$$\text{minimize } f_0(x)$$
$$\text{subject to } g_i(x) \le b_i, \quad i = 1,\ldots,m$$

  - $f_0 : \boldsymbol{R^n} \rightarrow \boldsymbol{R}$: objective function
  - $x=(x_1,\ldots,x_n)$: design variables (unknowns of the problem, they must be linearly independent)
  - $g_i : \boldsymbol{R^n} \rightarrow \boldsymbol{R}$: $(i=1,\ldots,m)$: inequality constraints

Optimization can be defined as the process of finding the conditions that give the maximum or minimum of a function.

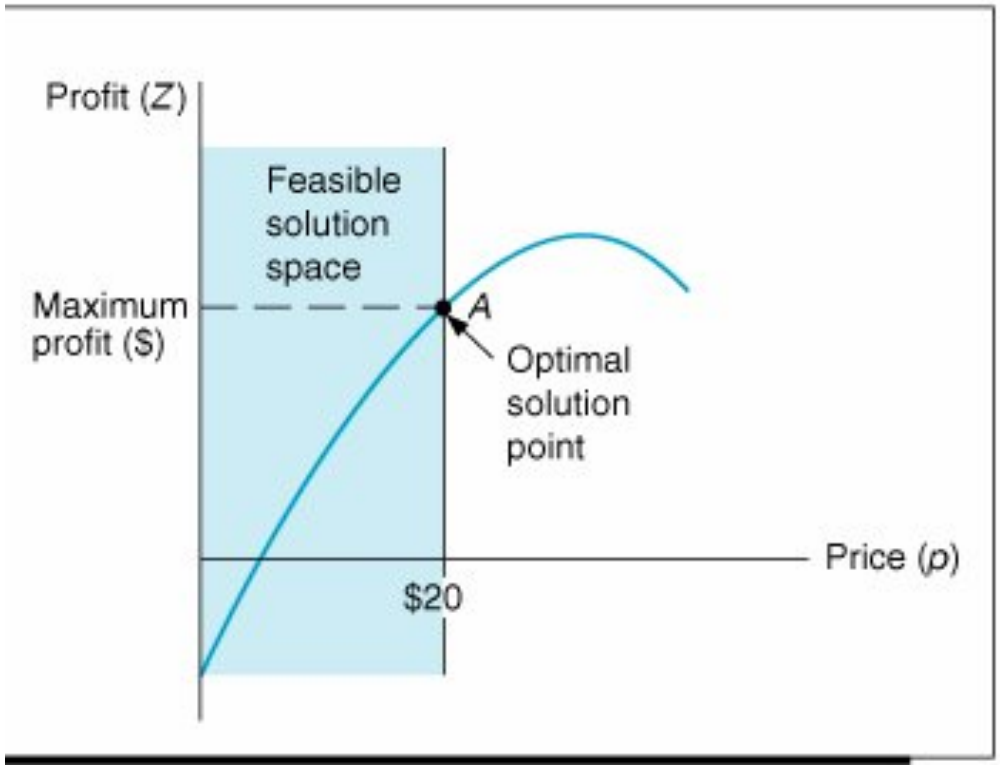
- **The problem is a constrained optimization problem**

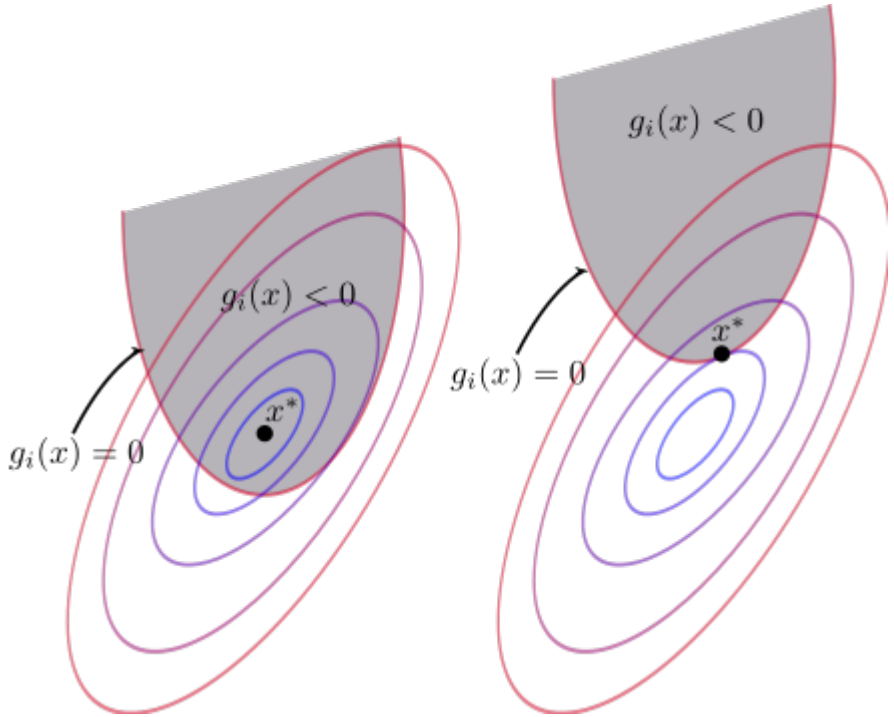
constrained optimization

- Finding the minimizer of a function subject to constraints:

$$\begin{aligned} \underset{x}{\text{minimize}} \; & f_0(x) \\ \text{s.t.} \; & f_i(x) \leq 0, \; i = \{1, \ldots, k\} \\ & h_j(x) = 0, \; j = \{1, \ldots, l\} \end{aligned}$$

- Example: Stock market. "Minimize variance of return subject to getting at least \$50."





Let's simplify to convex problems

- **One global maximum or minimum of a univariate function, e.g., f(x) =x^2**
  - Will provide more formal definition shortly
- **Assume function f(x) =x^2, find the x value that minimizes f(x)**

$$\arg\min_{x \in \Omega_X} f(x)$$

The value of *x* that maximises *f(x).* For example,

$$\arg\min_{x \in \{1,2,-3\}} f(x^2) = 1$$

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

Convex Optimization

- **Convex** minimization is a subfield of **optimization** that studies the problem of minimizing **convex** functions over **convex** sets. The convexity makes **optimization** easier than the general case since local minimum must be a global minimum, and first-order conditions are sufficient conditions for optimality.

Let's simplify to convex problems

- **One global maximum or minimum of a univariate function, e.g., f(x) =x^2**
  - Will provide more formal definition shortly
- **Assume function f(x) =x^2, find the x value that minimizes f(x)**

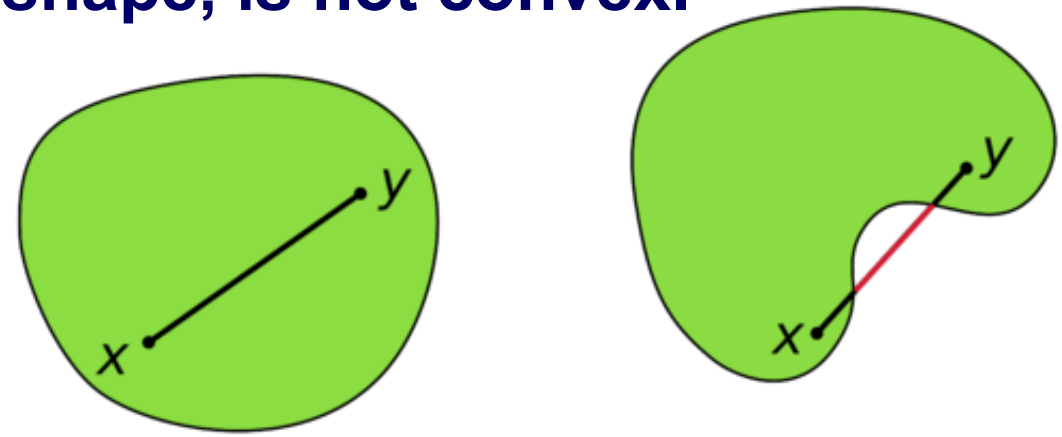
$$\arg\min_{x \in \Omega_X} f(x)$$

The value of *x* that maximises *f(x).* For example,

$$\arg\min_{x \in \{1,2,-3\}} f(x^2) = 1$$

Convexity

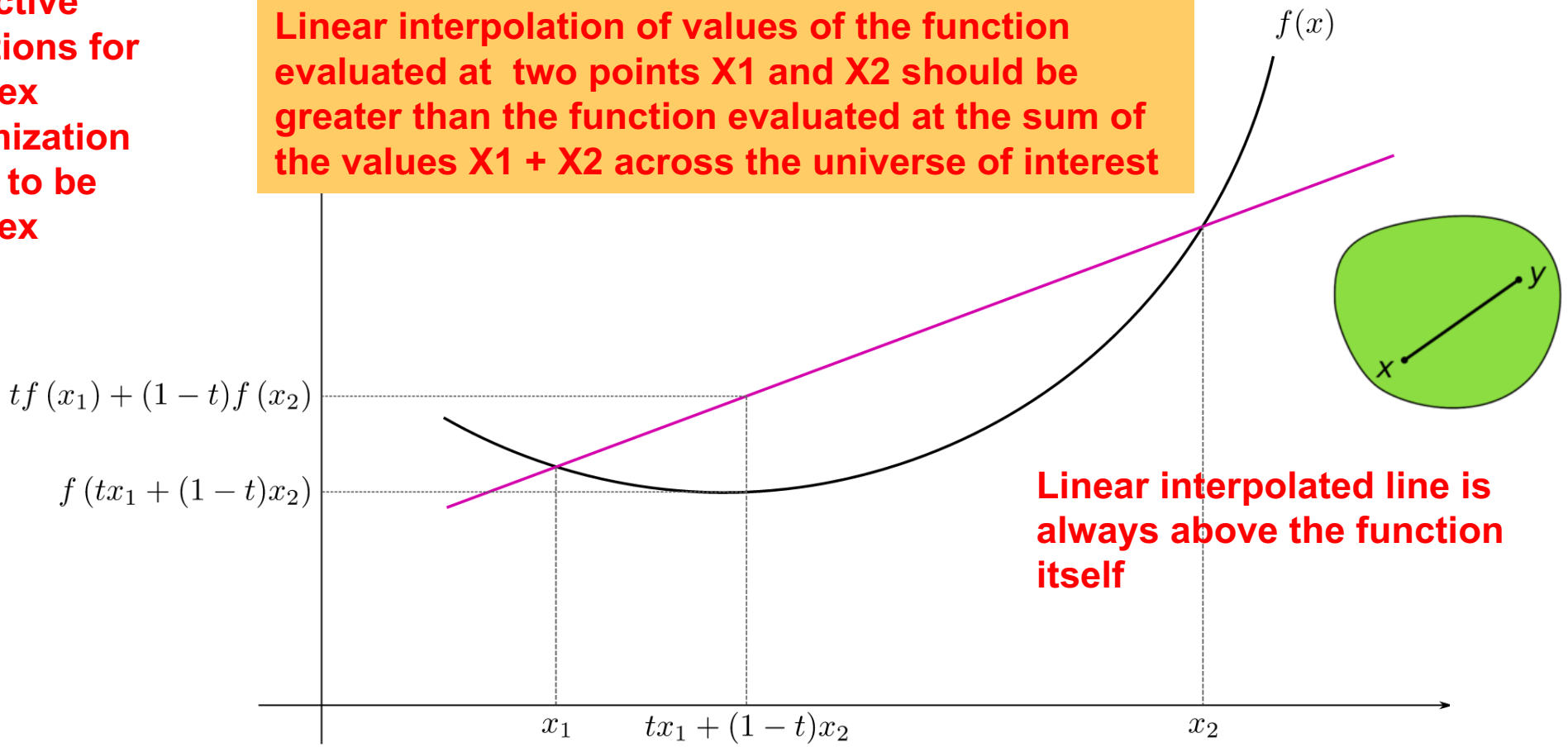
- In Euclidean space, an object is convex if for every pair of points within the object, every point on the straight line segment that joins them is also within the object.
- For example, a solid cube is convex, but anything that is hollow or has a dent in it, for example, a crescent shape, is not convex.



Minimization: Convex Functions in 1D

Objective functions for convex optimization need to be convex

Linear interpolation of values of the function evaluated at two points X1 and X2 should be greater than the function evaluated at the sum of the values X1 + X2 across the universe of interest



Linear interpolated line is always above the function itself

Let X be a convex set in a real vector space and let $f : X \to \mathbf{R}$ be a function.

- f is called **convex** if:

$$\forall x_1, x_2 \in X, \forall t \in [0,1] : \qquad f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2).$$

- f is called **strictly convex** if:

$$\forall x_1 \neq x_2 \in X, \forall t \in (0,1) : \qquad f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2).$$

- A function f is said to be (strictly) concave if $-f$ is (strictly) convex.

Convex Optimization Problems

Definition

An optimization problem is *convex* if its objective is a convex function, the inequality constraints f_j are convex, and the equality constraints h_j are affine

$$\begin{aligned} \underset{x}{\text{minimize}}\ & f_0(x) && \text{(Convex function)} \\ \text{s.t.}\ & f_i(x) \leq 0 && \text{(Convex sets)} \\ & h_j(x) = 0 && \text{(Affine)} \end{aligned}$$

Theorem

$\nabla f(x) = 0$ *if and only if* x *is a global minimizer of* $f(x)$.

Proof.

- $\nabla f(x) = 0$. We have

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

- $\nabla f(x) \neq 0$. There is a direction of descent.

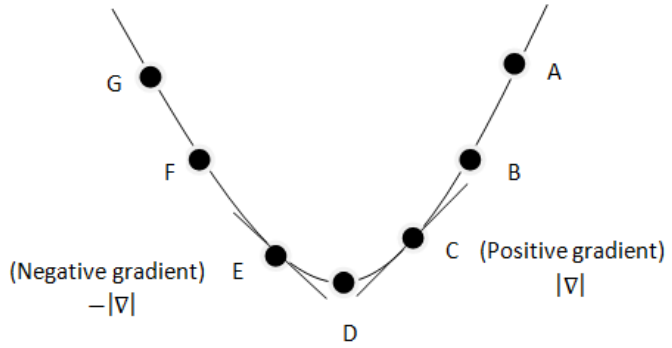
Convex Optimization Problems

More formally

Definition

An optimization problem is *convex* if its objective is a convex function, the inequality constraints f_j are convex, and the equality constraints h_j are affine

$$\begin{aligned} \underset{x}{\text{minimize}}\ & f_0(x) && \text{(Convex function)} \\ \text{s.t.}\ & f_i(x) \leq 0 && \text{(Convex sets)} \\ & h_j(x) = 0 && \text{(Affine)} \end{aligned}$$



Theorem

$\nabla f(x) = 0$ *if and only if* x *is a global minimizer of* $f(x)$.

Proof.

- $\nabla f(x) = 0$. We have

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

- $\nabla f(x) \neq 0$. There is a direction of descent.

First order condition of convexity
Around a minimum, everywhere f(y)>= f(x)+NablaF(x)(y-x)
For it to be minimum if NablaF(x) is zero otherwise we violate the constraint

First order condition of convexity

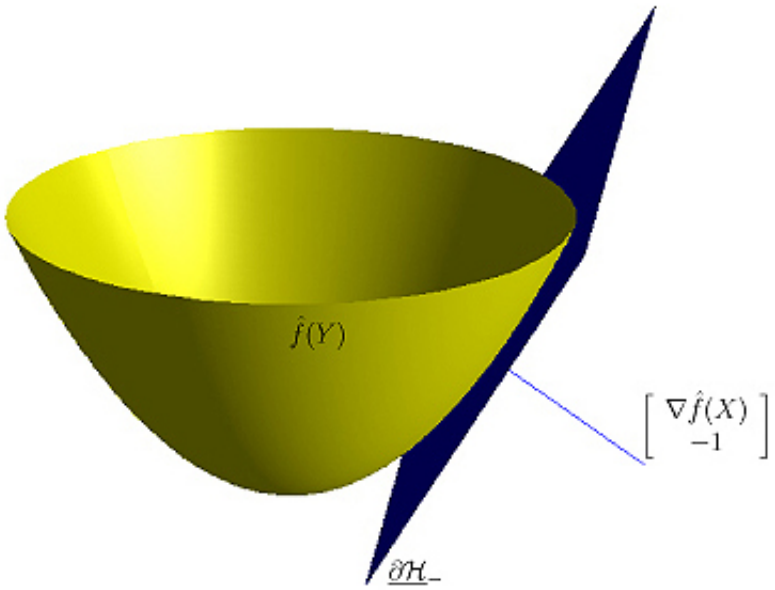
- **This is called First order condition of convexity**

3.1.3 First-order conditions

Suppose f is differentiable (*i.e.*, its gradient ∇f exists at each point in **dom** f, which is open). Then f is convex if and only if **dom** f is convex and

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{3.2}$$

holds for all $x,\ y \in$ **dom** f. This inequality is illustrated in figure 3.2.



http
vxbook/bv_cvxbook.pdf

Convexity: First-order condition

A real-valued differentiable function is **convex** *iff*

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

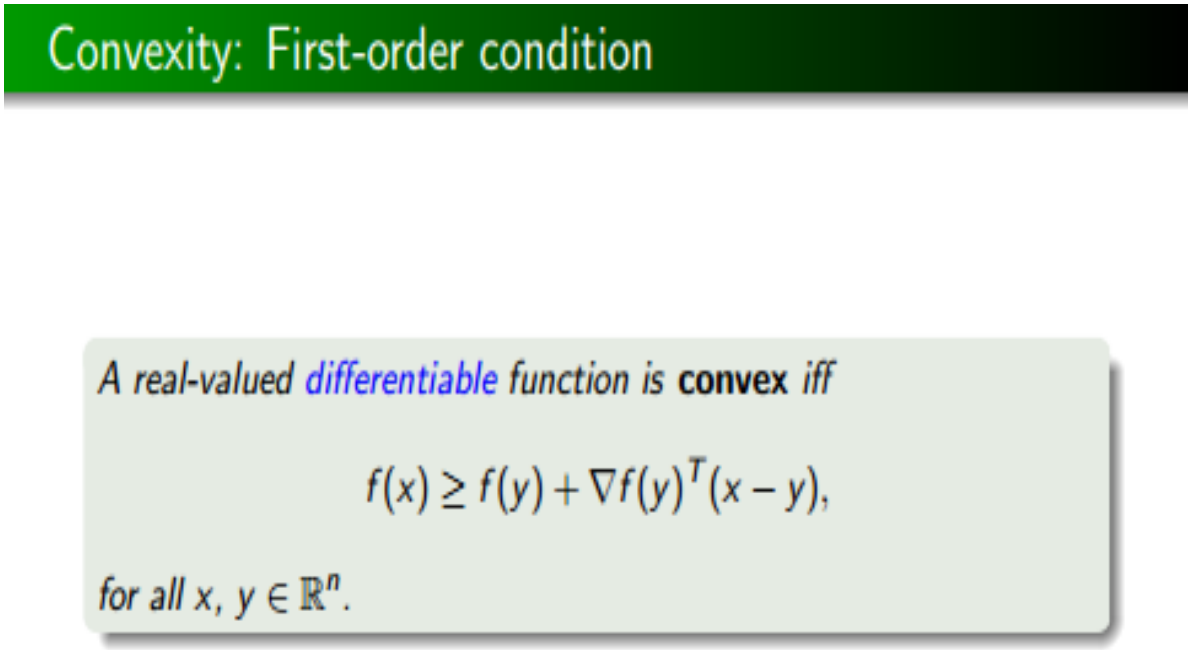
for all $x, y \in \mathbb{R}^n$.

- The function is globally *above the tangent* at y.

http://www.cs.ubc.ca/~schmidtm/MLSS/convex.pdf

First order condition of convexity rigorous proof

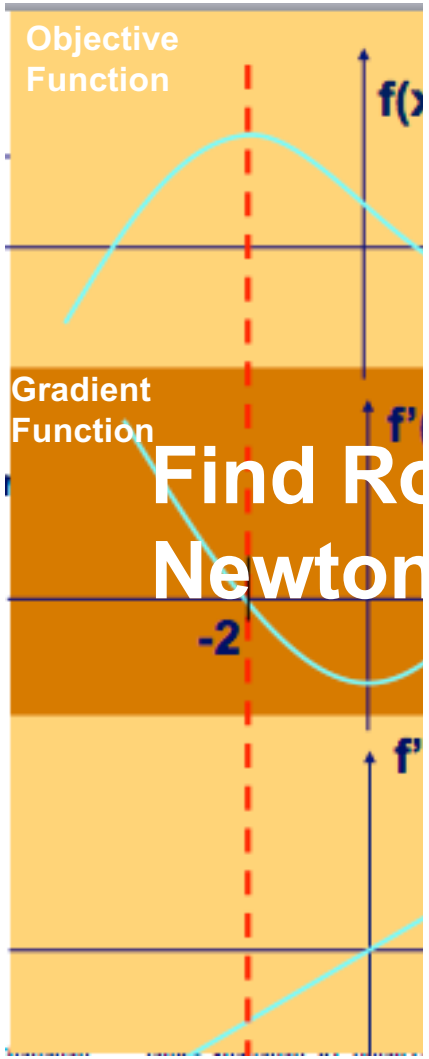
- **See here for more details**
  - http://mathgotchas.blogspot.com/2011/10/proof-for-first-order-condition-of.html



Convex opt: Maximization Problems

- Max, concave, negative f''(x)

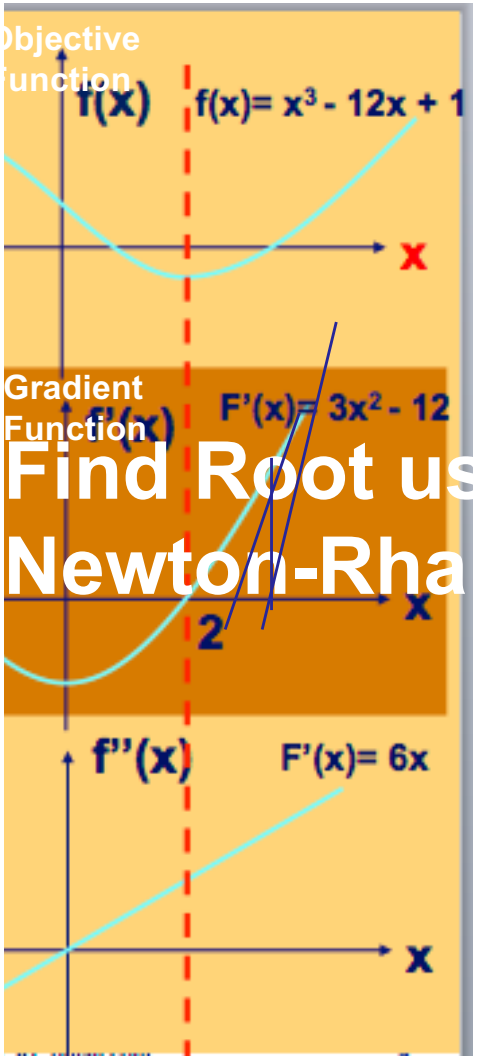
- Establish where f'(x)=0
- And we have found the max



Convex opt: Minimization Problems

- Min, convex, f''(x) <0

- Establish where f'(x)=0
- And we have found the min



Primal and Dual Spaces

- Start with optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}}\ & f_0(x) \\ \text{s.t.}\ \ & f_i(x) \leq 0,\ i = \{1, \ldots, k\} \\ & h_j(x) = 0,\ j = \{1, \ldots, l\} \end{aligned}$$

- Form *Lagrangian* using Lagrange multipliers $\lambda_i \geq 0$, $\nu_i \in \mathbb{R}$

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{k} \lambda_i f_i(x) + \sum_{j=1}^{l} \nu_j h_j(x)$$

- Form *dual function*

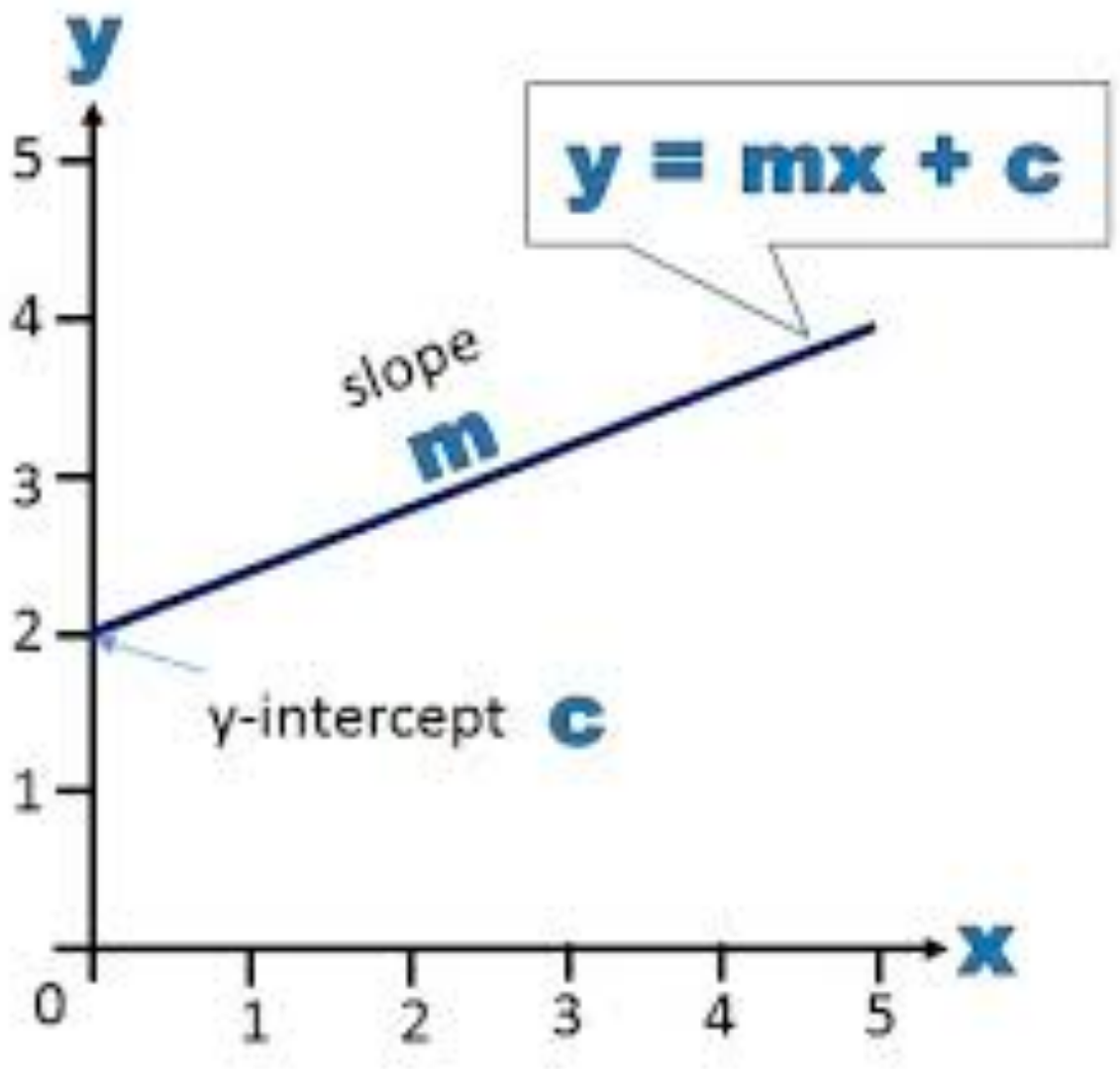
$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) = \inf_x \left\{ f_0(x) + \sum_{i=1}^{k} \lambda_i f_i(x) + \sum_{j=1}^{l} \nu_j h_j(x) \right\}$$

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

- **As machine learners why do we care about optimization theory?**

Equation of a line



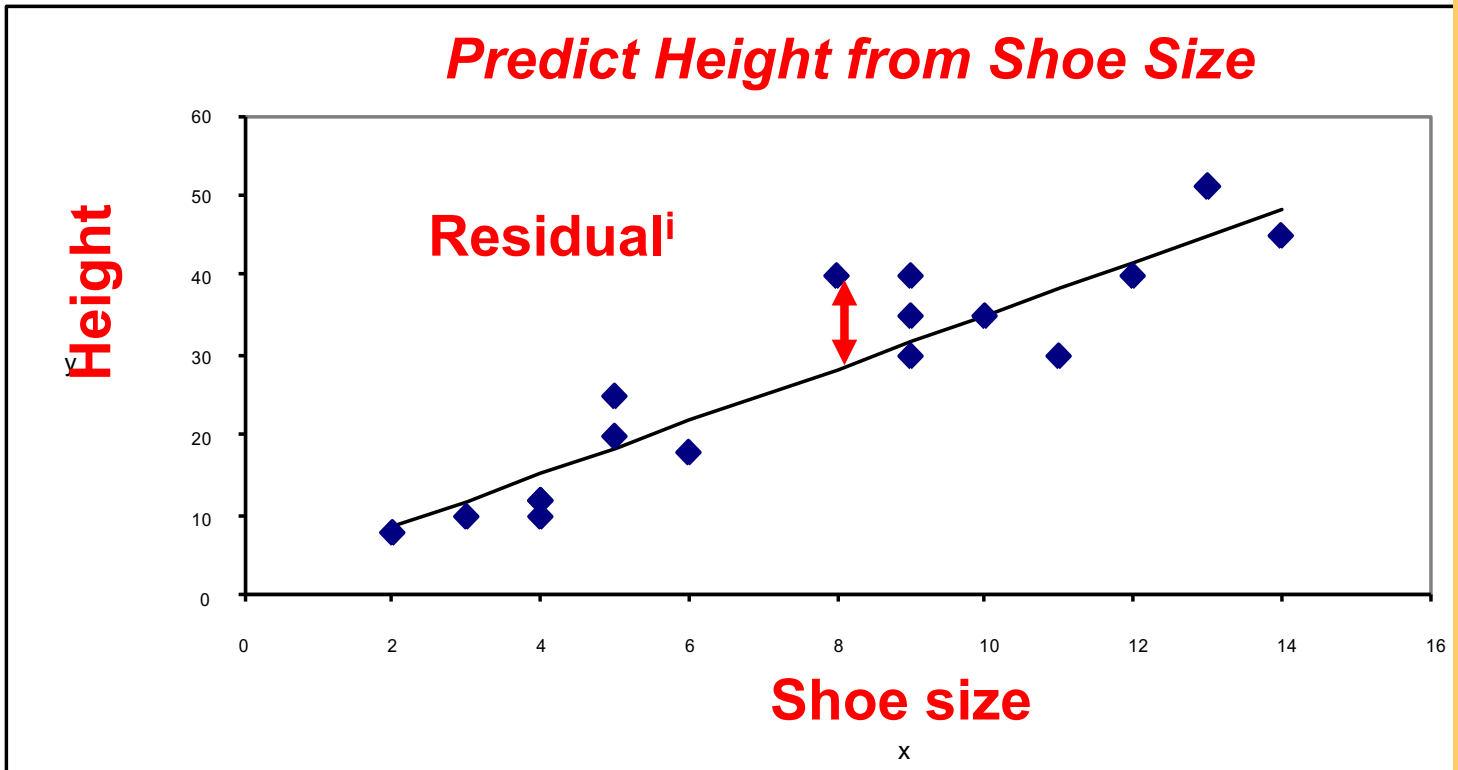
$$f(x) = mx + b$$

In math an Equation says 1,(

Equation of a line

$$\text{Residual}^{\text{i}} = \left(WX^i - y^i\right) \longrightarrow \text{Residual}^{\text{i}} = \left(WX^i - y^i\right)^2$$

Squared error loss gives us a twice differentiable function and thus we can use convex optimization



y=mx + b
$y=w_1 * x + w_0$

Where w1, w0, slope, intercept

are model parameters

Each pair yields a different sum of squares error

Machine Learning: Regression

Machine Learning (ML):"*a computer program that improves its performance at some task through experience*" [Mitchell 1997]

GIVEN: Input data is a table of attribute values and associated class values (in the case of supervised learning)
GOAL: Approximate $f(x_1,\ldots,x_n)$->y

Mimimize $MSE = \frac{1}{n}\sum \text{Residual}^i = \frac{1}{n}\sum \left(WX^i - y^i\right)^2$ **Y is real valued**

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					76
…	…	…	…	…	…
L (aka m)	0	4	...	8	97

Maximum vs Minimum

$$f(x)= x^3 - 12x + 1$$

First derivative

$$f'(x) = 3x^2 - 12$$

FOC

f'(x=x*) =0 at maximum and minimum

These zero points are the roots!

Second derivative f''(x) = 6x

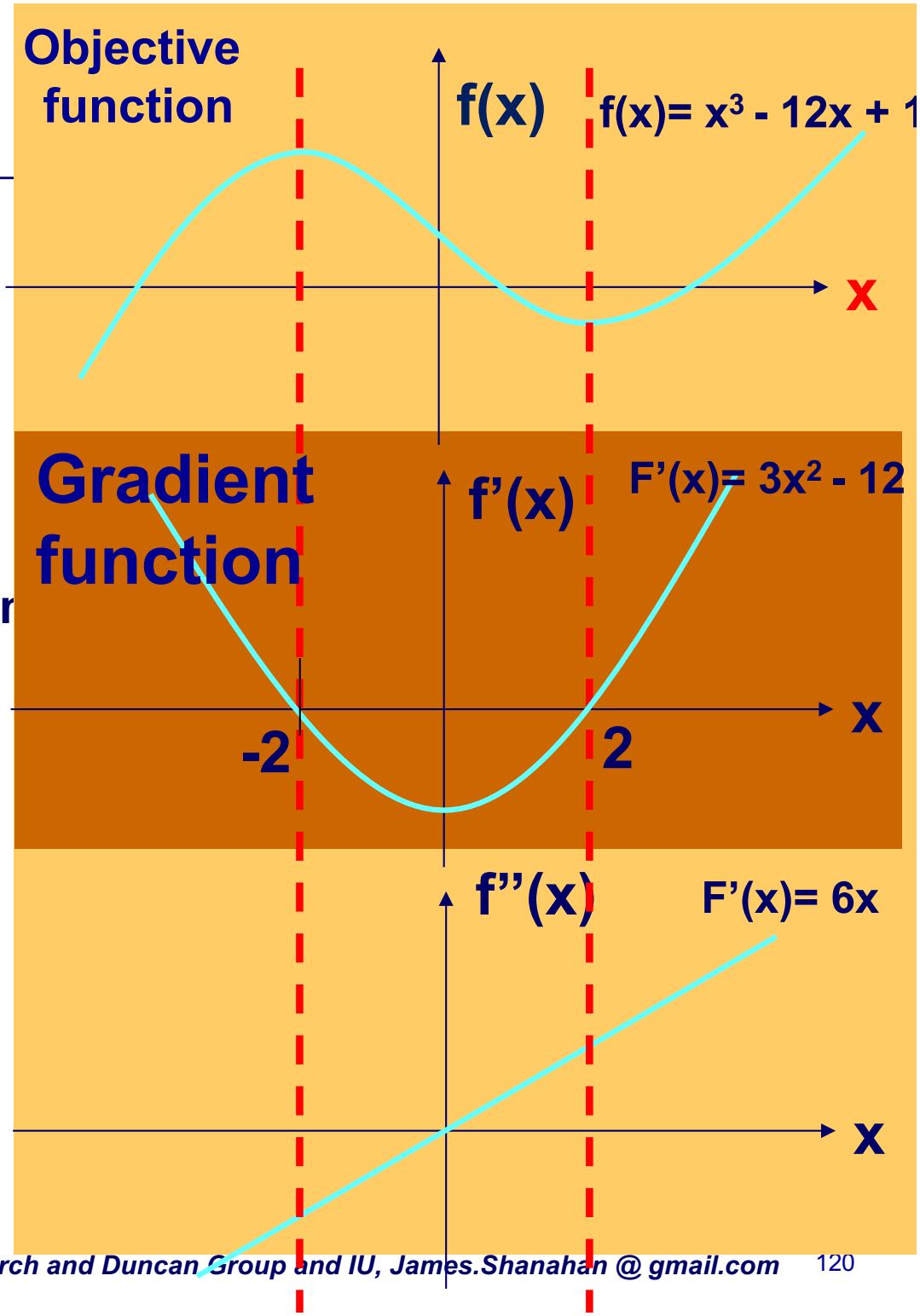
Gives the "rate of change of gradient"

SOC

If f''(x=x*) < 0 then maximum

f''(x=x*) > 0 then minimum

f''(x=x*) == 0 then ???



Y=m X x + b X 1
y=θ_1x + θ_0 1

vs Minimum

$$f(\theta)= \theta^3 - 12\theta + 1$$

First derivative

$$f'(\theta) = 3\theta^2 - 12$$

In convex optimisation what can we say about roots?

$F'(\theta)= 3\theta^2 - 12$
$0 = 3\theta^2 - 12$
$\Theta = 2$ or -2

FOC

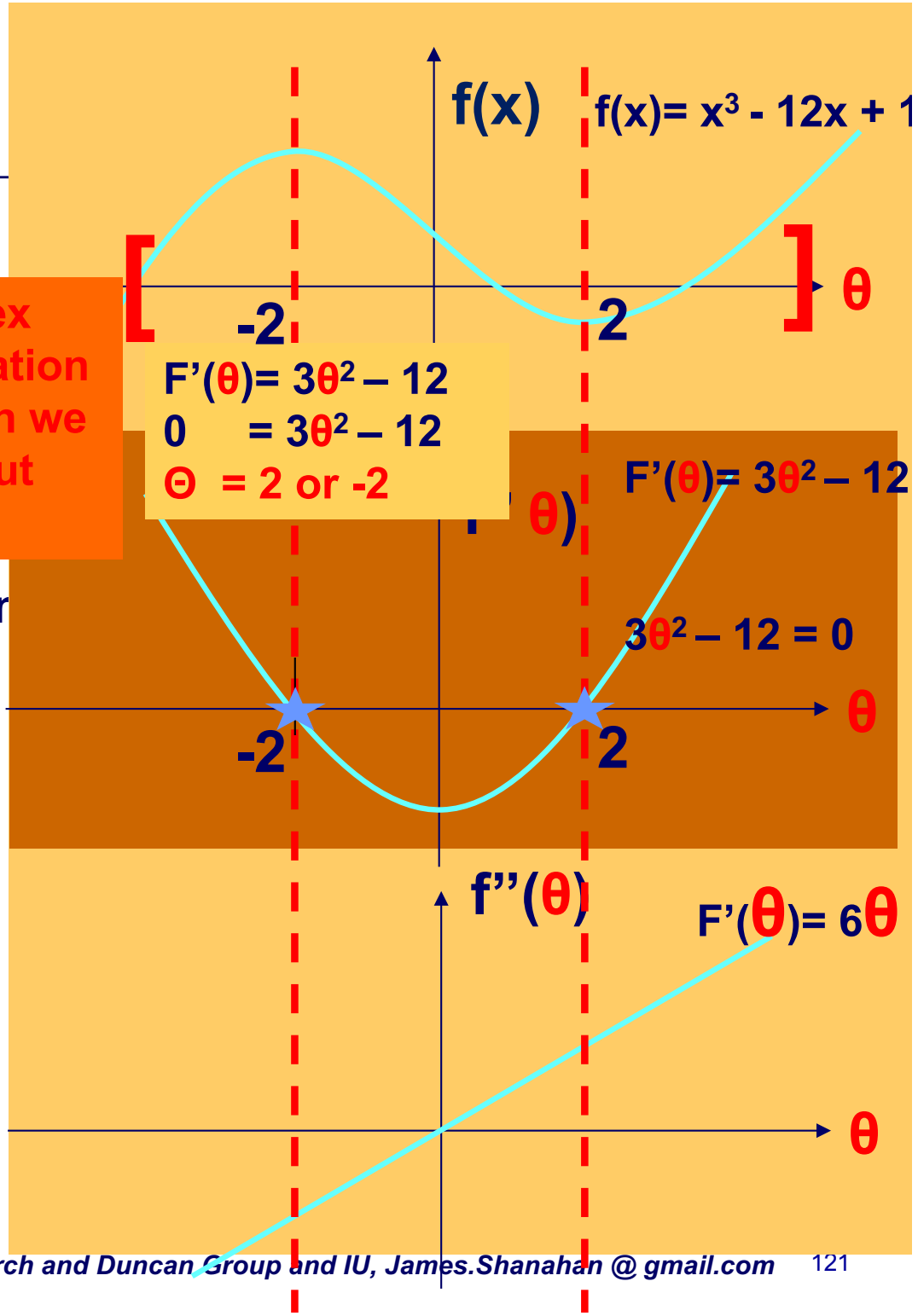
f'(θ=θ*) =0 at maximum and minimur
These zero points are the roots!

Second derivative $f''(\theta) = 6\theta$

Gives the "rate of change of gradient"

SOC

If f''(x=x*) < 0 then maximum
f''(x=x*) > 0 then minimum
f''(x=x*) == 0 then ???



Linear Regression: gradient is more complex

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					76
…	…	…	…	…	…
L (aka m)	0	4	…	8	97

$$\hat{y} = h_\theta(\mathbf{x}) = \theta^T \cdot \mathbf{x}$$

- θ is the model's *parameter vector*, containing the bias term θ_0 and the feature weights θ_1 to θ_n.
- θ^T is the transpose of θ (a row vector instead of a column vector).
- $\mathbf{x}$ is the instance's *feature vector*, containing x_0 to x_n, with x_0 always equal to 1.
- $\theta^T \cdot \mathbf{x}$ is the dot product of θ^T and $\mathbf{x}$.
- h_θ is the hypothesis function, using the model parameters θ.

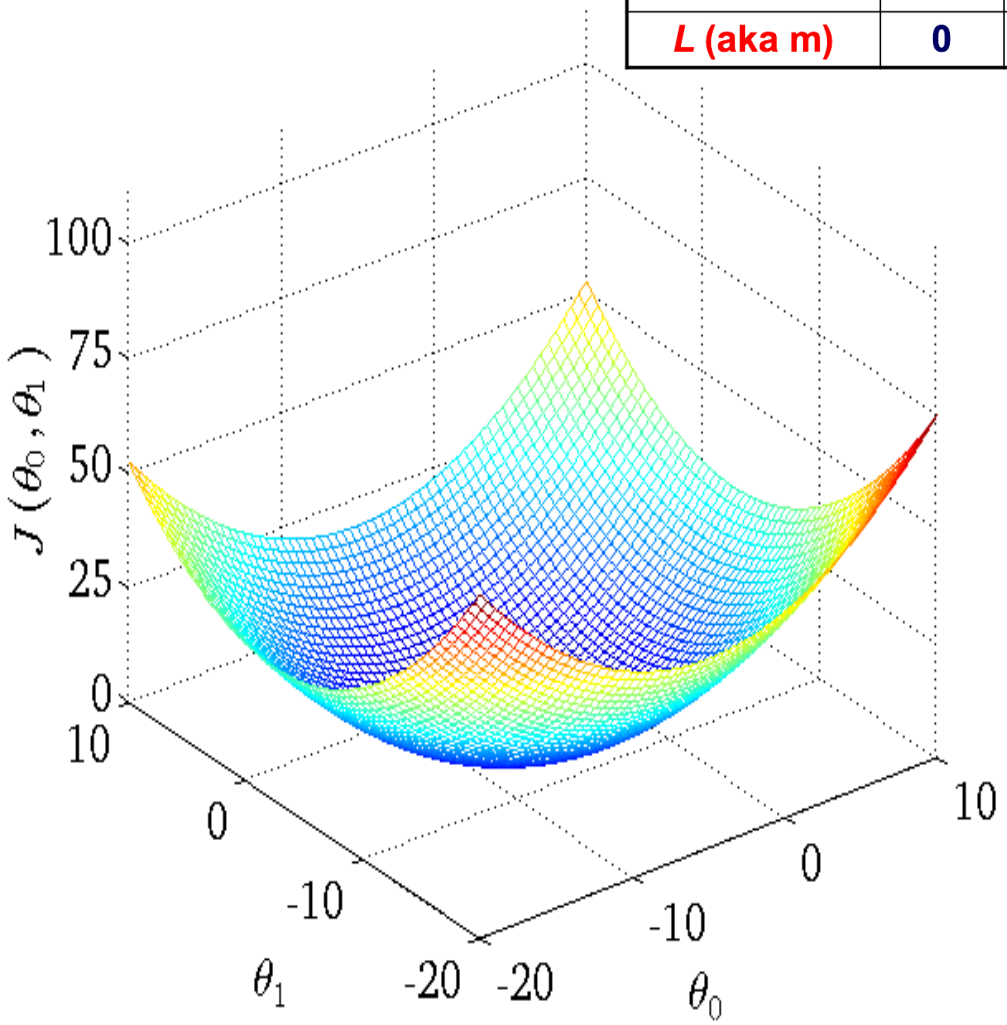
$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

Single variable $$\frac{\partial}{\partial \theta_j}\mathrm{MSE}(\theta) = \frac{2}{m}\sum_{i=1}^{m}\left(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}\right)x_j^{(i)}$$

Error Surface for OLS: Price~Size

$$J_q(W, X_1^L) = \frac{1}{m}\sum_{i=1}^{m}(W^T X_i - y_i)^2$$

Instance\Attr	x_1	x_2	…	x_n	y
1	3	0	..	7	73
2					76
…	…	…	…	…	…
L (aka m)	0	4	...	8	97



Gradient descent for weight optimization in linear regression

Vanilla gradient descent, aka batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

As we need to calculate the gradients for the whole dataset to perform just *one* update, batch gradient descent can be very slow and is intractable for datasets that don't fit in memory. Batch gradient descent also doesn't allow us to update our model *online*, i.e. with new examples on-the-fly.

In code, batch gradient descent looks something like this:

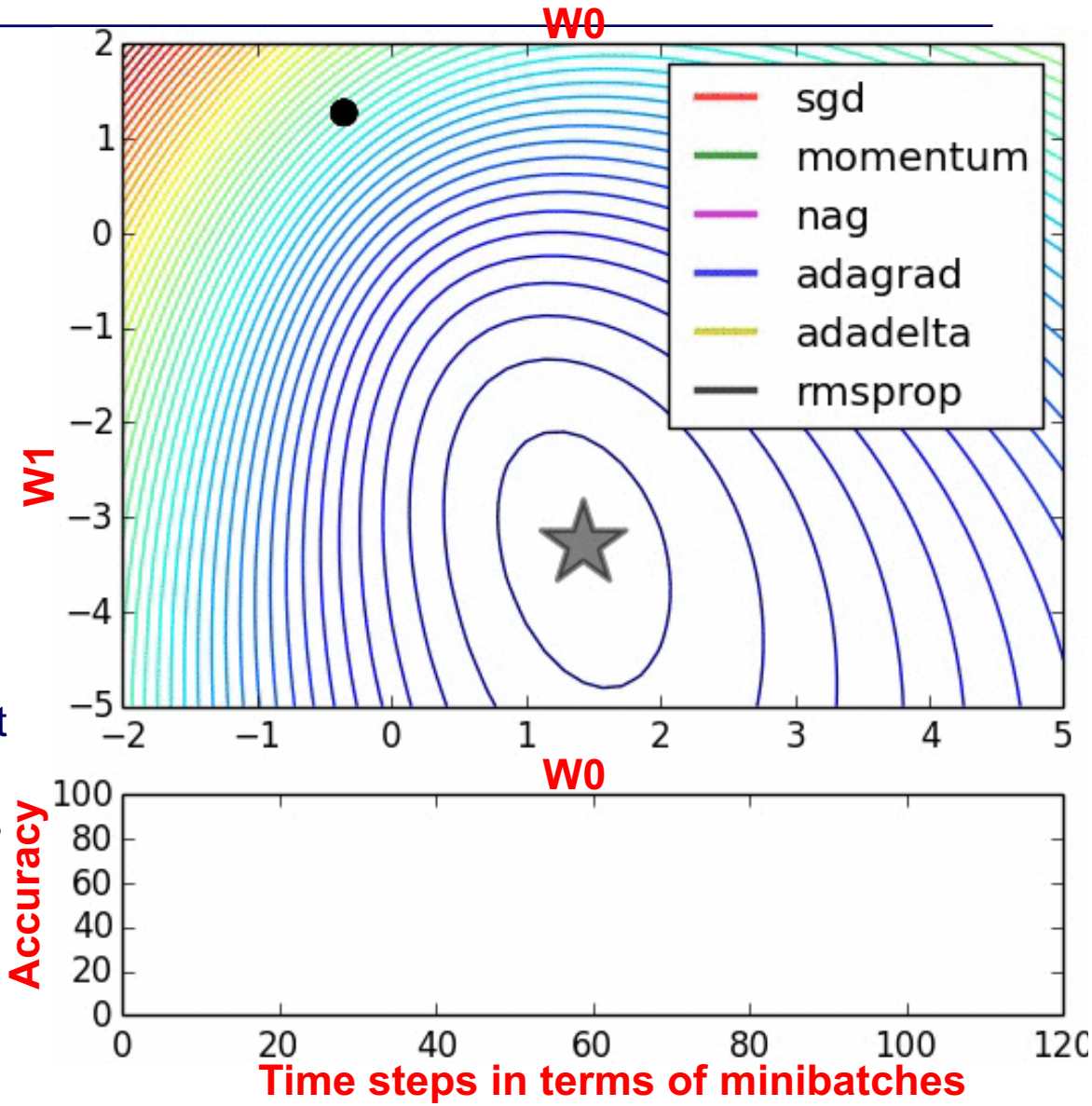
```
for i in range(nb_epochs):
  params_grad = evaluate_gradient(loss_function, data, params)
  params = params - learning_rate * params_grad
```

For a pre-defined number of epochs, we first compute the gradient vector `params_grad` of the loss function for the whole dataset w.r.t. our parameter vector `params`. Note that state-of-the-art deep learning libraries provide automatic differentiation that efficiently computes the gradient w.r.t. some parameters. If you derive the gradients yourself, then gradient checking is a good idea. (See here for some great tips on how to check gradients properly.)

We then update our parameters in the direction of the gradients with the learning rate determining how big of an update we perform. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

The effects of different GD update formulas

- **Noisy moons:** 2 class dataset
- This is logistic regression on noisy moons dataset from sklearn which shows the smoothing effects of momentum based techniques (which also results in over shooting and correction).
- Minibatches:
  - The error surface is visualized as an average over the whole dataset empirically, but the trajectories show the dynamics of minibatches on noisy data.
- The bottom chart is an *accuracy plot*.



(image credits to Alec Radford)

[http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html]

Animations comparing optimization algorithms

- Alec Radford has created some great animations comparing optimization algorithms SGD, Momentum, NAG, Adagrad, Adadelta, RMSprop (unfortunately no Adam) on low dimensional problems. Also check out his presentation on RNNs.
- The error surface is visualized as an average over the whole dataset empirically, but the trajectories show the dynamics of minibatches on noisy data. The bottom chart is an accuracy plot."

LR closed-form solution: Normal Equation

- **To find the value of θ that minimizes the cost function, there is a closed-form solution —in other words, a mathematical equation that gives the result directly. This is called the Normal Equation**

$$\hat{\theta} = \left(\mathbf{X}^T \cdot \mathbf{X}\right)^{-1} \cdot \mathbf{X}^T \cdot \mathbf{y}$$

- $\hat{\theta}$ is the value of θ that minimizes the cost function.
- y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$.

Now let's compute $\hat{\theta}$ using the Normal Equation. We will use the `inv()` function from NumPy's Linear Algebra module (`np.linalg`) to compute the inverse of a matrix, and the `dot()` method for matrix multiplication:

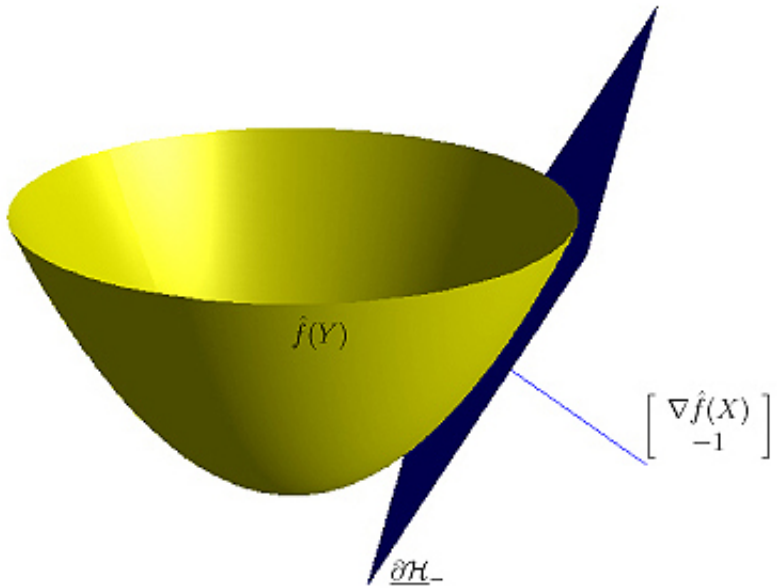
```
X_b = np.c_[np.ones((100, 1)), X]  # add x0 = 1 to each instance
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

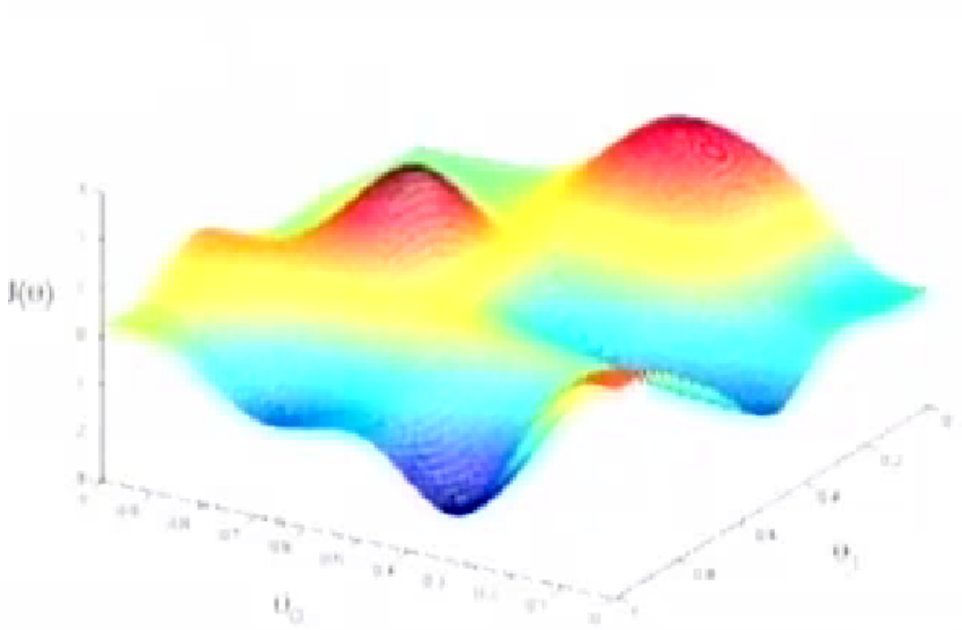
The actual function that we used to generate the data is $y = 4 + 3x_0 + \text{Gaussian noise}$. Let's see what the equation found:

```
>>> theta_best
array([[ 4.21509616],
       [ 2.77011339]])
```

Intuition

- Theory
- Geometry
- Code





Why do we care?

Optimization is at the heart of many (most practical?) machine learning algorithms.

- Linear regression:

$$\underset{w}{\text{minimize}} \quad \|Xw - y\|^2$$

- Classification (logistic regresion or SVM):

$$\underset{w}{\text{minimize}} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i x_i^T w)\right)$$

$$\text{or} \quad \|w\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad \xi_i \geq 1 - y_i x_i^T w, \xi_i \geq 0.$$

But wait there's more ..

- Maximum likelihood estimation:

$$\underset{\theta}{\text{maximize}} \sum_{i=1}^{n} \log p_\theta(x_i)$$

- Collaborative filtering:

$$J = \tfrac{1}{2} R(Y - X\Theta^T)^2 + \tfrac{1}{2}\lambda(||X||^2 + ||\Theta||^2)$$

- k-means:

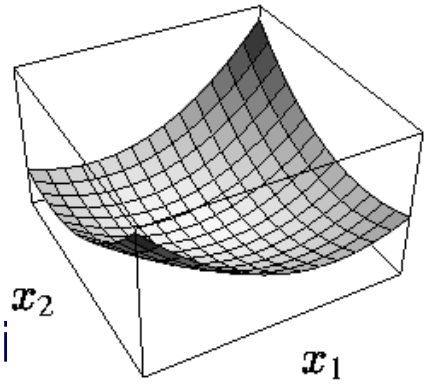
$$\underset{\mu_1,\ldots,\mu_k}{\text{minimize}} \; J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - \mu_j\|^2$$

- And more (graphical models, feature selection, active learning, control)

Convex versus Non-Convex ML

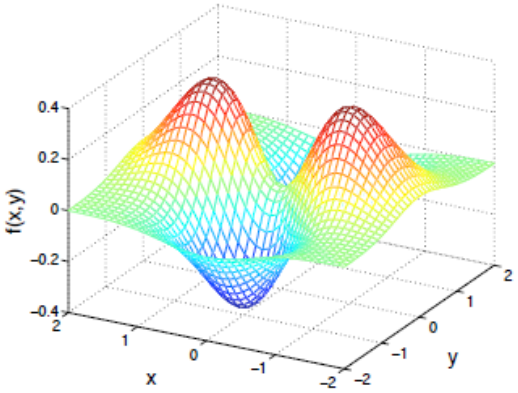
- **Many methods in machine learning are based on finding parameters that minimise some objective function. Very often, the objective function is a weighted sum of two terms: a cost function and regularization term.**
  - In statistics terms the (log-)likelihood and (log-)prior.
  - If both of these components are convex, then their sum is also convex.
- **Examples of convex optimisation problems in ML:**
  - linear regression/ Ridge regression, with Tikhonov regularisation, etc
  - sparse linear regression with L1 regularisation, such as lasso
  - support vector machines
  - parameter estimation in linear-Gaussian time series (Kalman filter and friends)
- **Typical examples of non-convex optimization in ML are**
  - neural networks
  - maximum likelihood mixtures of Gaussians

General Approach to Finding Extrema



- **Well-behaved version spaces**
  - Convex or concave function (±definiteness)
  - Algorithms seek a local extrema knowing that it wi
    - If f() is a concave function then local maximum is a global maximum
    - If f() is a convex function then local minimum is a global minimum
    - Newton-Raphson, Gradient Descent, Conjugate Gradient Descent

- **Otherwise**
  - We resort to local approximations
    - Hill-Climbing
    - Simulated annealing
    - Commonly used in Neural Networks



Optimization terms and Concepts

- Variable
- Feasible region
- Solution (feasible point)
- Optimal solution (best point)
- Global and local optimality
- Optimality conditions
- Duality
- Direct methods
- Numerical methods
- Heuristics

‹ DERIVATIVES OF MULTIVARIABLE FUNCTIONS

Partial derivative and gradient (articles)

- Introduction to partial derivatives
- Second partial derivatives
- The gradient
- Directional derivatives (introduction)
- Directional derivatives (going deeper)

Next tutorial
Differentiating parametric curves

The gradient

The gradient stores all the partial derivative information of a multivariable function. But it's more than a mere storage device, it has several wonderful interpretations and many, many uses.

What you need to be familiar with before starting this lesson:

- Partial derivatives
- Vector fields
- Contour maps—only necessary for one section of this lesson.

Review your calculus

What we're building toward

- The gradient of a scalar-valued multivariable function $f(x, y, \ldots)$, denoted ∇f, packages all its partial derivative information into a vector:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$

https://www.khanacademy.org/math/multivariable-calculus/multivariable-derivatives/partial-derivative-and-gradient-articles/a/introduction-to-partial-derivatives

In particular, this means ∇f is a vector-valued function.

Lecture Outline

- **Introduction**
- **Optimization Theory Introduction**
- **Unconstrained optimization**
  - Univariate unconstrained optimization
    - Finding optimal solutions via Root finding using Newton-Raphson
    - Finding optimal solutions via Root finding using Gradient descent
  - Multivariate unconstrained optimization
    - Analytical versus numerical optimization (gradient descent)
- **Constrained optimization**
- **Convex Optimization**
- **Machine learning as an optimization problem**
- **Summary**

End of lecture

Appendix: Roots of a system of nonlinear equations

- **Find the solution (aka ROOT) of a system of nonlinear equations**

Consider a nonlinear system of equations:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

suppose we have the function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

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

and the objective function

$$F(\mathbf{x}) = \frac{1}{2} G^{\mathrm{T}}(\mathbf{x}) G(\mathbf{x}) = \frac{1}{2} \left(\left(3x_1 - \cos(x_2 x_3) - \frac{3}{2} \right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1 \right)^2 + \left(\exp(-x_1 x_2) + 20x_3 + \frac{1}{3}(1 \right.$$

With initial guess

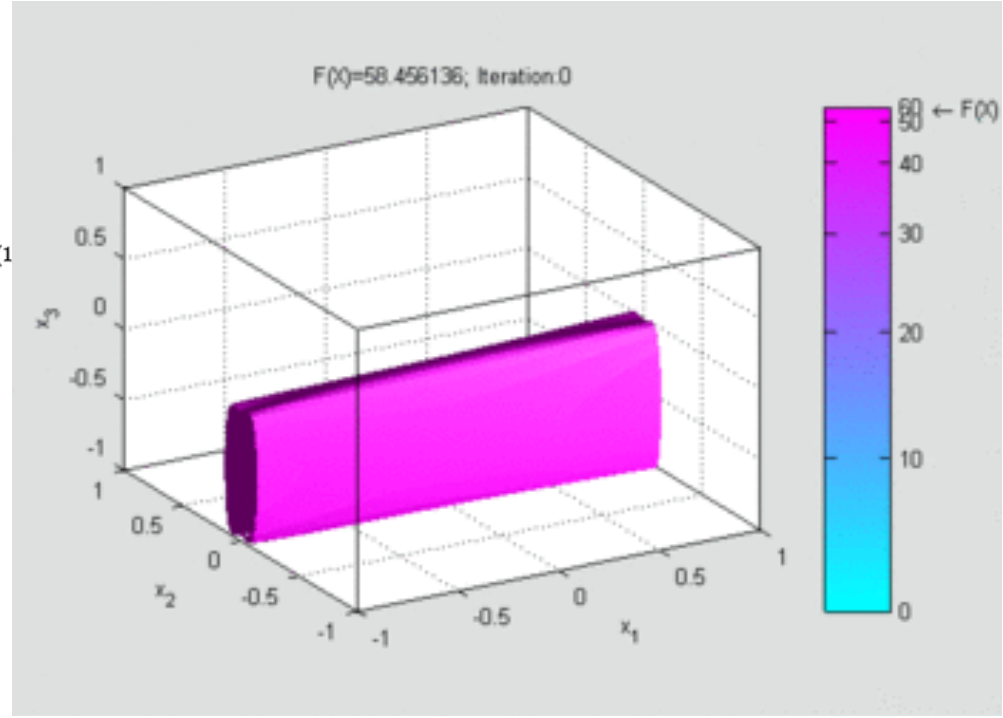
$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

$$\nabla F(\mathbf{x}^{(0)}) = J_G(\mathbf{x}^{(0)})^{\mathrm{T}} G(\mathbf{x}^{(0)})$$



Solving for multiple nonlinear equations

- **Refs**
  - https://en.wikipedia.org/wiki/Gradient_descent
  - https://math.stackexchange.com/questions/1983713/gradient-descent-to-solve-nonlinear-systems

In the example given, the nonlinear system of equations is:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - 3/2 = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + (10\pi - 3)/3 = 0 \end{cases}$$

Then G is defined:

$$G(\vec{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - 3/2 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + (10\pi - 3)/3 \end{bmatrix} = \begin{bmatrix} g_1(\vec{x}) \\ g_2(\vec{x}) \\ g_3(\vec{x}) \end{bmatrix}$$

which is a vector in $\mathbb{R}^3$.

Note that the equation is solved when $G = 0$. So the idea is to minimize G as much as possible (i.e. make it close to 0). The most logical way to do that is to simply minimize the norm of G (i.e. $||G||$). So we define:

$$F(\vec{x}) = \frac{1}{2}||G(\vec{x})||_2^2 = \frac{1}{2}G^T(\vec{x})G(\vec{x}) = \frac{1}{2}\left(g_1^2(\vec{x}) + g_2^2(\vec{x}) + g_3^2(\vec{x})\right)$$

where we square the norm because the transformation preserves ordering, so there is no need for the extra computational cost of the square root. Also, constant coefficients don't matter in the objective function, so we multiply by 1/2 for mathematical "niceness" (it will cancel the 2s that come from differentiating F to compute ∇F, which has squares).

Solution of a non-linear system

Gradient descent can also be used to solve a system of nonlinear equations. Below is an example that shows how to use the gradient descent to solve for three unknown variables, x_1, x_2, and x_3. This example shows one iteration of the gradient descent.

Consider a nonlinear system of equations:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

suppose we have the function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

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

and the objective function

$$F(\mathbf{x}) = \frac{1}{2} G^{\mathrm{T}}(\mathbf{x}) G(\mathbf{x}) = \frac{1}{2} \left(\left(3x_1 - \cos(x_2 x_3) - \frac{3}{2} \right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1 \right)^2 + \left(\exp(-x_1 x_2) + 20x_3 + \frac{1}{3}(10\pi - 3) \right)^2 \right)$$

With initial guess

$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

$$\nabla F(\mathbf{x}^{(0)}) = J_G(\mathbf{x}^{(0)})^{\mathrm{T}} G(\mathbf{x}^{(0)})$$

The Jacobian matrix $J_G(\mathbf{x}^{(0)})$

$$J_G = \begin{bmatrix} 3 & \sin(x_2 x_3) x_3 & \sin(x_2 x_3) x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{bmatrix}$$

Then evaluating these terms at $\mathbf{x}^{(0)}$

$$J_G\left(\mathbf{x}^{(0)}\right) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}, \qquad G(\mathbf{x}^{(0)}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}$$

So that

$$\mathbf{x}^{(1)} = 0 - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix}$$

and

$$F\left(\mathbf{x}^{(0)}\right) = 0.5\left((-2.5)^2 + (-1)^2 + (10.472)^2\right) = 58.456$$

Now a suitable γ_0 must be found such that $F(\mathbf{x}^{(1)}) \leq F(\mathbf{x}^{(0)})$. This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$ which gives

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}$$

Evaluating at this value,

$$F\left(\mathbf{x}^{(1)}\right) = 0.5\left((-2.48)^2 + (-1.00)^2 + (6.28)^2\right) = 23.306$$

The decrease from $F(\mathbf{x}^{(0)}) = 58.456$ to the next step's value of $F(\mathbf{x}^{(1)}) = 23.306$ is a sizable decrease in the objective function. Further steps would reduce its value until a solution to the system was found.

Consider a nonlinear system of equations:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

suppose we have the function

$$G(\mathbf{x}) = \begin{bmatrix} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

where

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

and the objective function

$$F(\mathbf{x}) = \frac{1}{2} G^{\mathrm{T}}(\mathbf{x}) G(\mathbf{x}) = \frac{1}{2} \left(\left(3x_1 - \cos(x_2 x_3) - \frac{3}{2} \right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1 \right)^2 + \left(\exp(-x_1 x_2) + 20x_3 + \frac{1}{3}(10\pi - 3) \right)^2 \right)$$

With initial guess

$$\mathbf{x}^{(0)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We know that

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \gamma_0 \nabla F(\mathbf{x}^{(0)})$$

where

$$\nabla F(\mathbf{x}^{(0)}) = J_G(\mathbf{x}^{(0)})^{\mathrm{T}} G(\mathbf{x}^{(0)})$$

The Jacobian matrix $J_G(\mathbf{x}^{(0)})$

$$J_G = \begin{bmatrix} 3 & \sin(x_2x_3)x_3 & \sin(x_2x_3)x_2 \\ 8x_1 & -1250x_2 + 2 & 0 \\ -x_2\exp(-x_1x_2) & -x_1\exp(-x_1x_2) & 20 \end{bmatrix}$$

Then evaluating these terms at $\mathbf{x}^{(0)}$

$$J_G\left(\mathbf{x}^{(0)}\right) = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 20 \end{bmatrix}, \qquad G(\mathbf{x}^{(0)}) = \begin{bmatrix} -2.5 \\ -1 \\ 10.472 \end{bmatrix}$$

So that

$$\mathbf{x}^{(1)} = 0 - \gamma_0 \begin{bmatrix} -7.5 \\ -2 \\ 209.44 \end{bmatrix}$$

and

$$F\left(\mathbf{x}^{(0)}\right) = 0.5\left((-2.5)^2 + (-1)^2 + (10.472)^2\right) = 58.456$$

Now a suitable γ_0 must be found such that $F(\mathbf{x}^{(1)}) \leq F(\mathbf{x}^{(0)})$. This can be done with any of a variety of line search algorithms. One might also simply guess $\gamma_0 = 0.001$ which gives

$$\mathbf{x}^{(1)} = \begin{bmatrix} 0.0075 \\ 0.002 \\ -0.20944 \end{bmatrix}$$

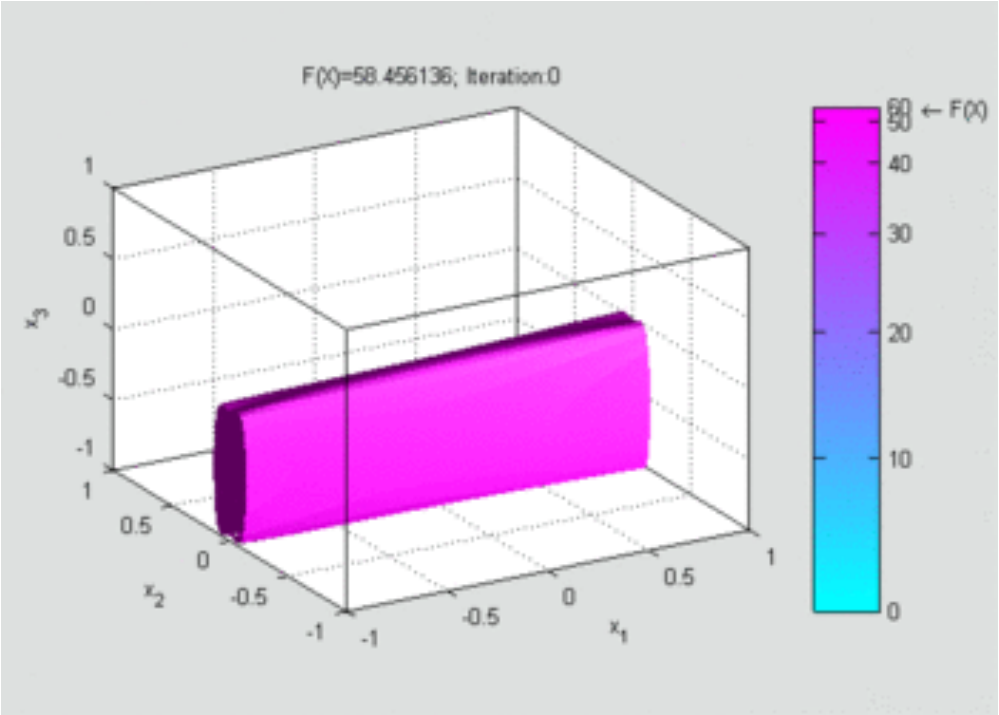
Evaluating at this value,

$$F\left(\mathbf{x}^{(1)}\right) = 0.5\left((-2.48)^2 + (-1.00)^2 + (6.28)^2\right) = 23.306$$

The decrease from $F(\mathbf{x}^{(0)}) = 58.456$ to the next step's value of $F(\mathbf{x}^{(1)}) = 23.306$ is a sizable decrease in the objective function. Further steps would reduce its value until a solution to the system was found.

Solution of a non-linear system: Animation

- An animation showing the first 83 iterations of gradient descent applied to this example. Surfaces are isosurfaces of $F(x^{(n)})$ at current guess $x^{(n)}$, and arrows show the direction of descent. Due to a small and constant step size, the convergence is slow.



MATLAB [edit]

The following MATLAB code demonstrates a concrete solution for solving the non-linear system of equations presented in the previous section:

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0 \\ \exp(-x_1 x_2) + 20x_3 + \frac{10\pi - 3}{3} = 0 \end{cases}$$

```
% Multi-variate vector-valued function G(x)
G = @(x) [
    3*x(1) - cos(x(2)*x(3)) - 3/2           ;
    4*x(1)^2 - 625*x(2)^2 + 2*x(2) - 1      ;
    exp(-x(1)*x(2)) + 20*x(3) + (10*pi-3)/3];

% Jacobian of G
JG = @(x) [
    3,                    sin(x(2)*x(3))*x(3),   sin(x(2)*x(3))*x(2) ;
    8*x(1),               -1250*x(2)+2,          0                   ;
    -x(2)*exp(-x(1)*x(2)), -x(1)*exp(-x(1)*x(2)), 20                 ];

% Objective function F(x) to minimize in order to solve G(x)=0
F = @(x) 0.5 * sum(G(x).^2);

% Gradient of F (partial derivatives)
dF = @(x) JG(x).' * G(x);

% Parameters
GAMMA = 0.001;    % step size (learning rate)
MAX_ITER = 1000;  % maximum number of iterations
FUNC_TOL = 0.1;   % termination tolerance for F(x)

fvals = [];       % store F(x) values across iterations
progress = @(iter,x) fprintf('iter = %3d: x = %-32s, F(x) = %f\n', ...
    iter, mat2str(x,6), F(x));

% Iterate
iter = 1;         % iterations counter
x = [0; 0; 0];    % initial guess
fvals(iter) = F(x);
progress(iter, x);
while iter < MAX_ITER && fvals(end) > FUNC_TOL
    iter = iter + 1;
    x = x - GAMMA * dF(x);  % gradient descent
    fvals(iter) = F(x);     % evaluate objective function
    progress(iter, x);      % show progress
end

% Plot
plot(1:iter, fvals, 'LineWidth',2); grid on;
title('Objective Function'); xlabel('Iteration'); ylabel('F(x)');

% Evaluate final solution of system of equations G(x)=0
disp('G(x) = '); disp(G(x))

% Output:
%
% iter =   1: x = [0;0;0]                          , F(x) = 58.456136
% iter =   2: x = [0.0075;0.002;-0.20944]          , F(x) = 23.306394
% iter =   3: x = [0.015005;0.0015482;-0.335103]   , F(x) = 10.617030
% ...
% iter = 187: x = [0.683335;0.0388258;-0.52231]    , F(x) = 0.101161
% iter = 188: x = [0.684666;0.0389831;-0.522302]   , F(x) = 0.099372
%
% (converged in 188 iterations after exceeding termination tolerance for F(x))
```