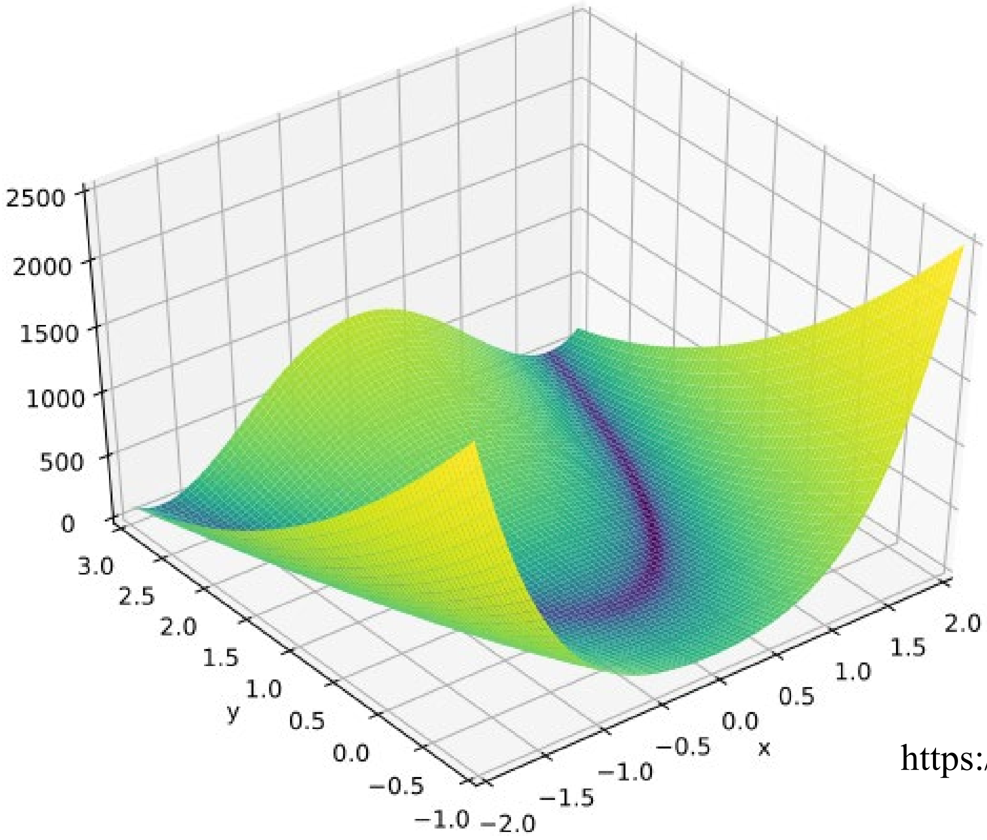
Function Optimization Algorithms

Fundamentals to Train Neural Nets



$$f(x, y) = (1-x)^2 + 100\left(y - x^2\right)^2$$

https://en.wikipedia.org/wiki/Rosenbrock_function

CJ Chung, F2025

Function Optimization - Essential technique for

- Many engineering design optimization problems
- Finance and Economics: Optimizing investment portfolios, pricing strategies, and resource allocation
- Operations and Logistics: optimizing route, resources, etc.
- Healthcare and Medicine: optimal medical treatments and managing healthcare resources efficiently.
- **ML/DL - Training Neural Networks: finding weight values to minimize the network error, optimizing model (continuous) parameters to improve accuracy and performance**
- …

Example: A function with 3 variables

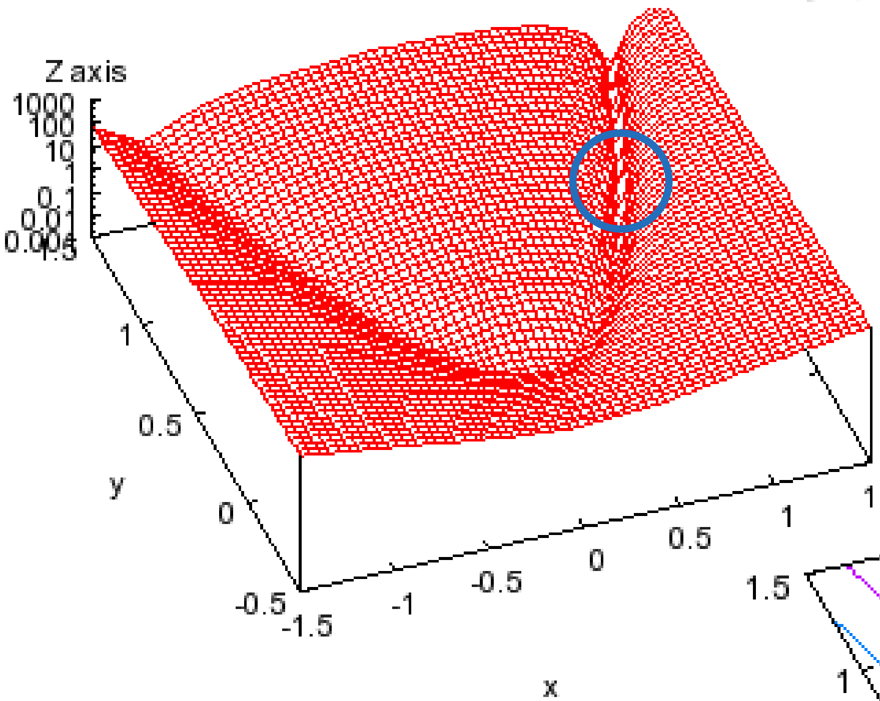
Minimize $f(x_1, x_2, x_3) = \sum_{i=1}^{3} x_i^2$,

subject to $-5 \le x_i \le 5,\ \ i = 1,2,3.$

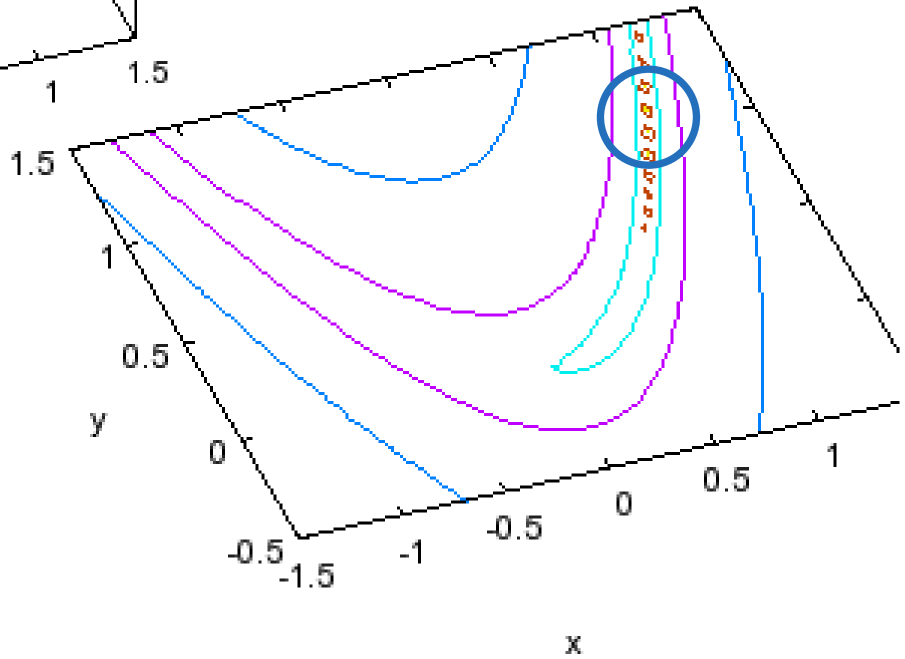
A global minimum 0 at $(x_1, x_2, x_3) = (0,0,0).$

Example: Rosenbrock function (valley)

$$f(x, y) = (1 - x)^2 + 100\left(y - x^2\right)^2$$



Global minimum 0 at (1,1)

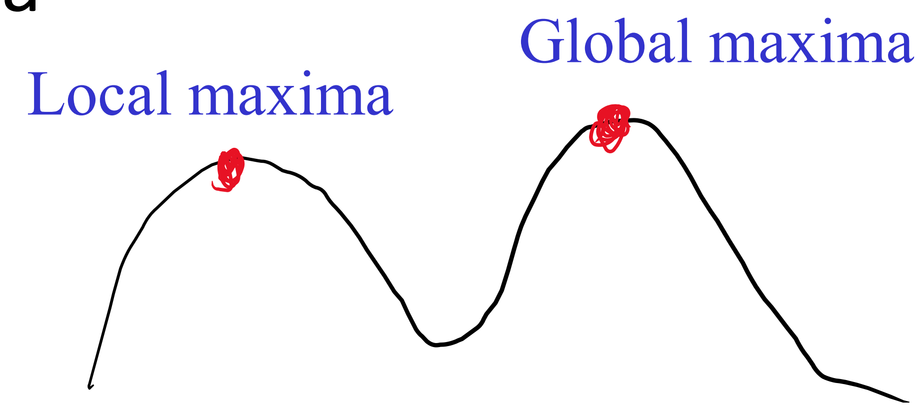


Function Optimization

- No explicit goal condition
- To find a structure having a max (or min) value of a function
- If we think of the data structures as “points” in a space, this function can be thought of as a landscape over the space
- A simple algorithm would be “Hill-climbing”, if maximization problem

Hill-climbing Idea

- Traverse by moving one point to that adjacent point having the highest elevation
- Terminates when there is no adjacent point having a higher elevation than the current point
- Can get stuck on local maxima



Minimization and Maximization

- If the problem is to minimize a function *f*, this is equivalent to maximize a function, -*f*

$$\min f(x) = \max\{-f(x)\}$$

Types of Optimization Problems

- Local / Global
- Combinatorial / Numerical
- Unconstrained / Constrained
- Single objective / Multi-objective
- Nonlinear / LP (Linear Programming: models are defined by linear relationships only) / Integer Programming

Non-linear function optimization

Unconstrained global minimization problems:

A pair $< S, f >$, where $S \subseteq R^n$ is a bounded set on R^n and

$$f : S \mapsto R$$

is an n-dimensional real-valued function.
The problem is to find a vector $x^* \in S$ such that

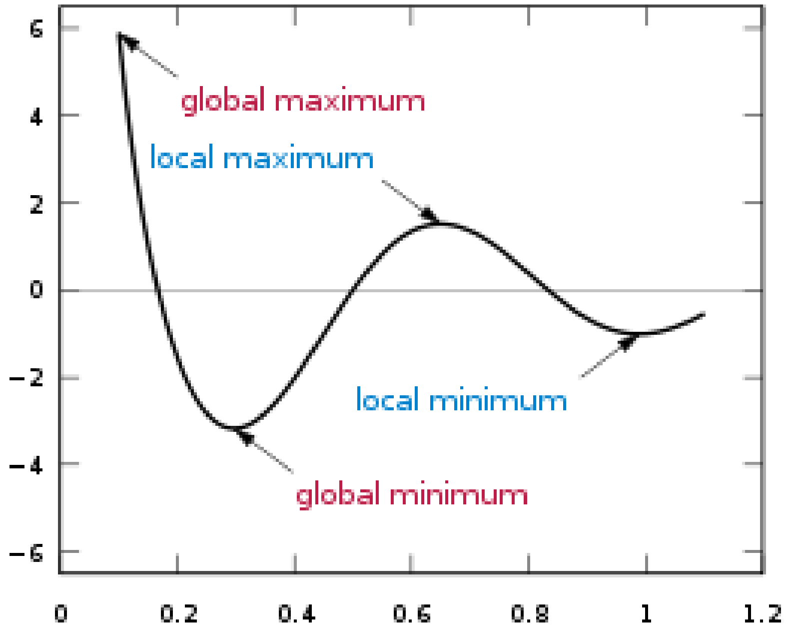
$$\forall x \in S: f(x^*) \leq f(x)$$

Note that objective function f does not need to be continuous. f could be a computer program function!

Maxima / Minima

- http://en.wikipedia.org/wiki/Maxima_and_minima
- Calculus based methods usually cannot escape local minima

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



Optimization/Search Methods

Calculus-Based

Enumerative

Guided random

Direct

Indirect

Simulated Annealing

…

Evolutionary

Eg.
- Hill Climbing
- **Gradient Descent (GD)**

This slide

local extrema are determined by finding where the gradient of the objective function is equal to 0

- **GA, ES, EP, CA, GP**
- **ES(1+1) with 1/5 rule**
- **CMA-ES**
- …

To be covered

Function optimization algorithms

- Using "direct" gradient information
  - (Steepest) Gradient Descent – simplest algorithm
  - Newton's
  - Conjugate Gradient
  - …
- Using random strategies
  - Simulated Annealing
  - Tabu Search
  - Particle swarm optimization
  - Evolutionary Computation - non-smooth function, especially the function is a computer program: GA, ES, CMA-ES, EP, CA, GP, …
- …

Steepest Gradient Descent (minimization)

- To have the function decrease at each iteration, i.e.

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

- Algorithm

Learning rate

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k g_k, \textit{ where}$$

$$g_k \equiv \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

Gradient
(directional information)

Steepest Gradient Descent (minimization) – Another Math Notation

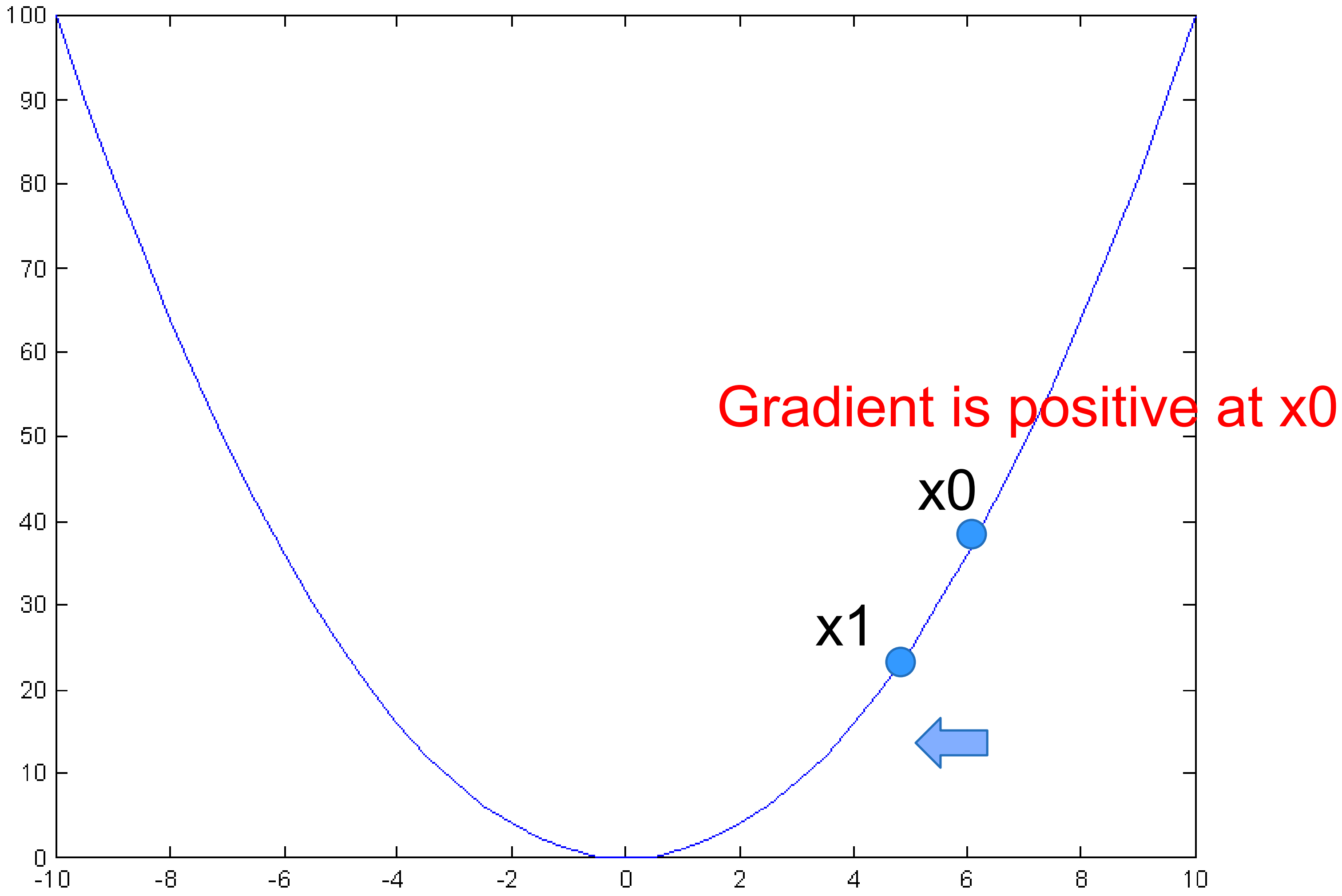
$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla F(\mathbf{x})\big|_{\mathbf{x}=\mathbf{x}_k}$$

"del" denotes the vector of partial derivatives of F

Steepest Gradient Descent (minimization) – Another Math Notation

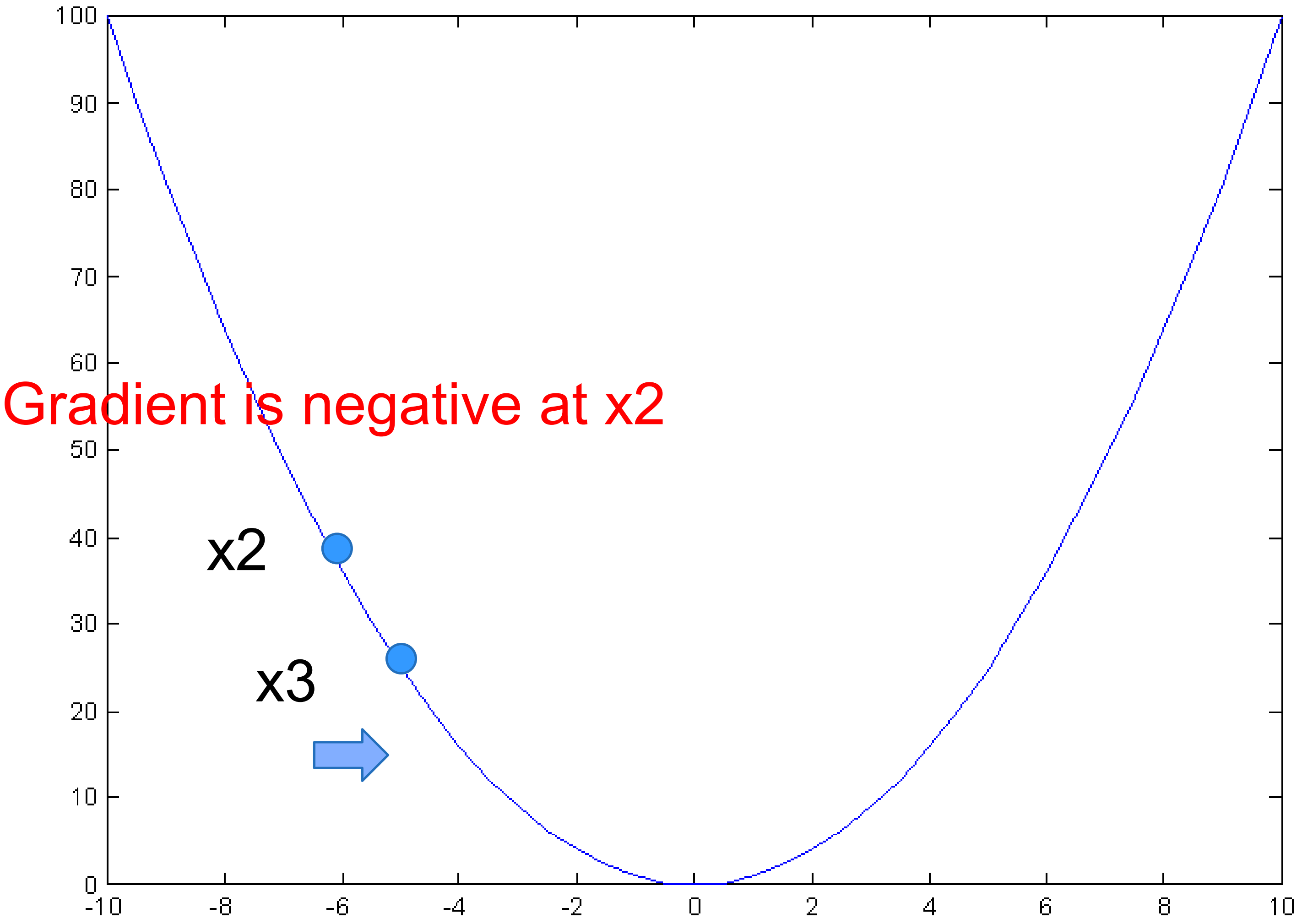
$$x_{k+1} = x_k - \alpha \frac{\partial F}{\partial x_k}$$

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



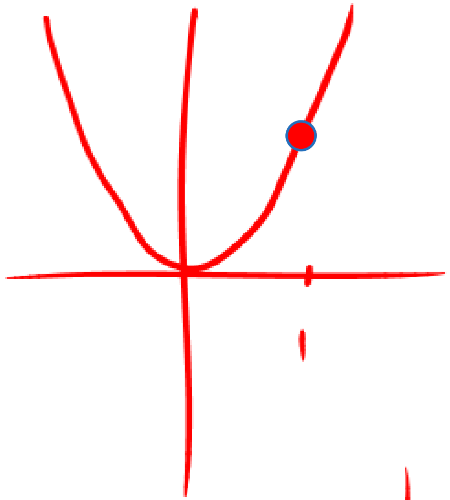
100
90
80
70
60
50
40
30
20
10
0
-10
-8
-6
-4
-2
0
2
4
6
8
10
Gradient is positive at x0
x0
x1

Moving west!


100
90
80
70
60
50
40
30
20
10
0
-10
-8
-6
-4
-2
0
2
4
6
8
10
Gradient is negative at x2
x2
x3

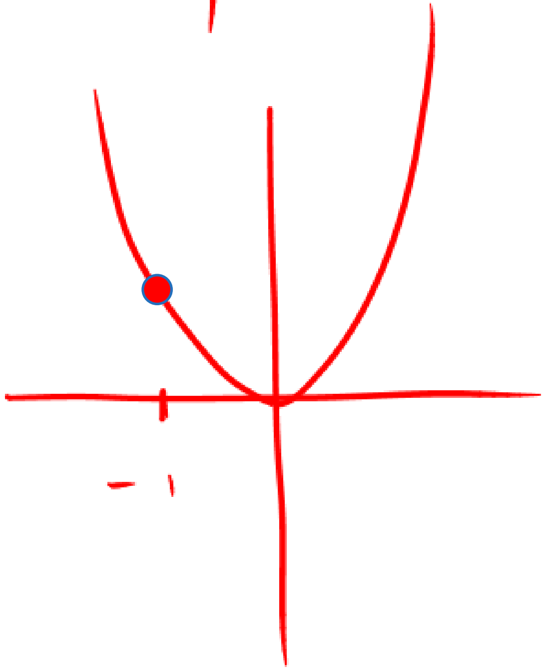
Moving east!

$f(x) = x^2 \qquad \alpha = 0.2 \qquad \underline{x_0 = 1}$



(i)

$$x_1 = x_0 - \alpha \cdot f'(x_0)$$
$$= 1 - 0.2 f'(1)$$
$$= 1 - 0.2 \cdot 2$$
$$= 1 - 0.4$$
$$= 0.6$$



(ii) $\underline{x_0 = -1}$

$$x_1 = x_0 - \alpha \cdot f'(x_0)$$
$$= -1 - 0.2 \cdot (-2)$$
$$= -1 + 0.4 = -0.6$$

An example of Gradient Descent Algorithm (1/3)

A function to be minimized: $F(\mathbf{x}) = x_1^2 + 25x_2^2$

Starting from an initial guess

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

The first step is to find the gradient:

Partial derivative of $F(\mathbf{x})$ with respect to x_1

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix}.$$

Evaluate the gradient at the initial guess

$$g_0 = \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} 1 \\ 25 \end{bmatrix}.$$

An example of Gradient Descent Algorithm (2/3)

A function to be minimized: $F(\mathbf{x}) = x_1^2 + 25x_2^2$

$$g = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix}$$

Assume $\alpha = 0.01$ (fixed). The 1st iteration of steepest descent algorithm would be

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha g_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.01 \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix}.$$

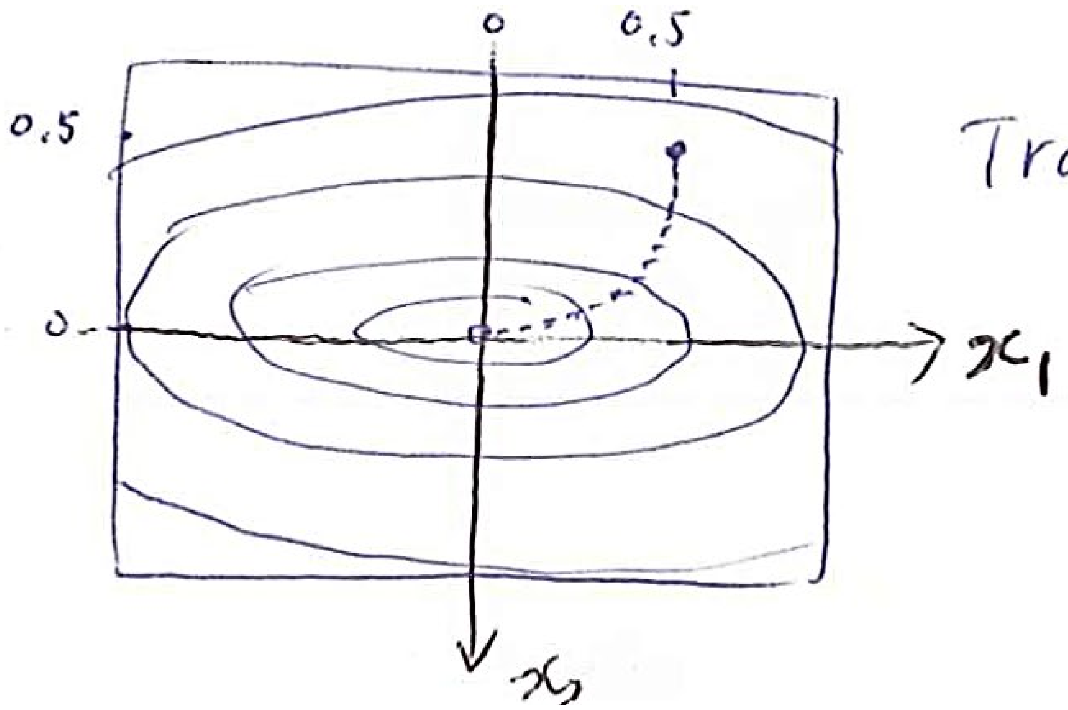
2nd iteration

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha g_1 = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix} - 0.01 \begin{bmatrix} 0.98 \\ 12.5 \end{bmatrix} = \begin{bmatrix} 0.4802 \\ 0.125 \end{bmatrix}$$

Continue until $\mathbf{x}_n$ is acceptable.

An example of Gradient Descent Algorithm (2/3)

A function to be minimized: $F(\mathbf{x}) = x_1^2 + 25x_2^2$



Trajectory for the algorithm when $\alpha = 0.01$

What are Advantages of Gradient Descent Algorithm?

- Efficient & fast
- Guaranteed to find a minima, if learning rate is small

What are Disadvantages of Calculus-based, for example, Gradient Descent Algorithm?

- Easily trapped in a local minima, not global minima
- (Some cases, it is hard to get partial derivatives!)
- Must find a partial derivative of the objective function, which may not always possible. Cannot be applied to unsmooth functions, non-differentiable functions, or computer program functions

What is the problem in the real-world optimization problems?

- Usually, we are given just data, not objective (math) function
- To find objective function of the real-world problem is difficult
- To find partial derivatives of some objective functions is hard or may not be always possible

HW1 – GD_HW2.ipynb (1/4)

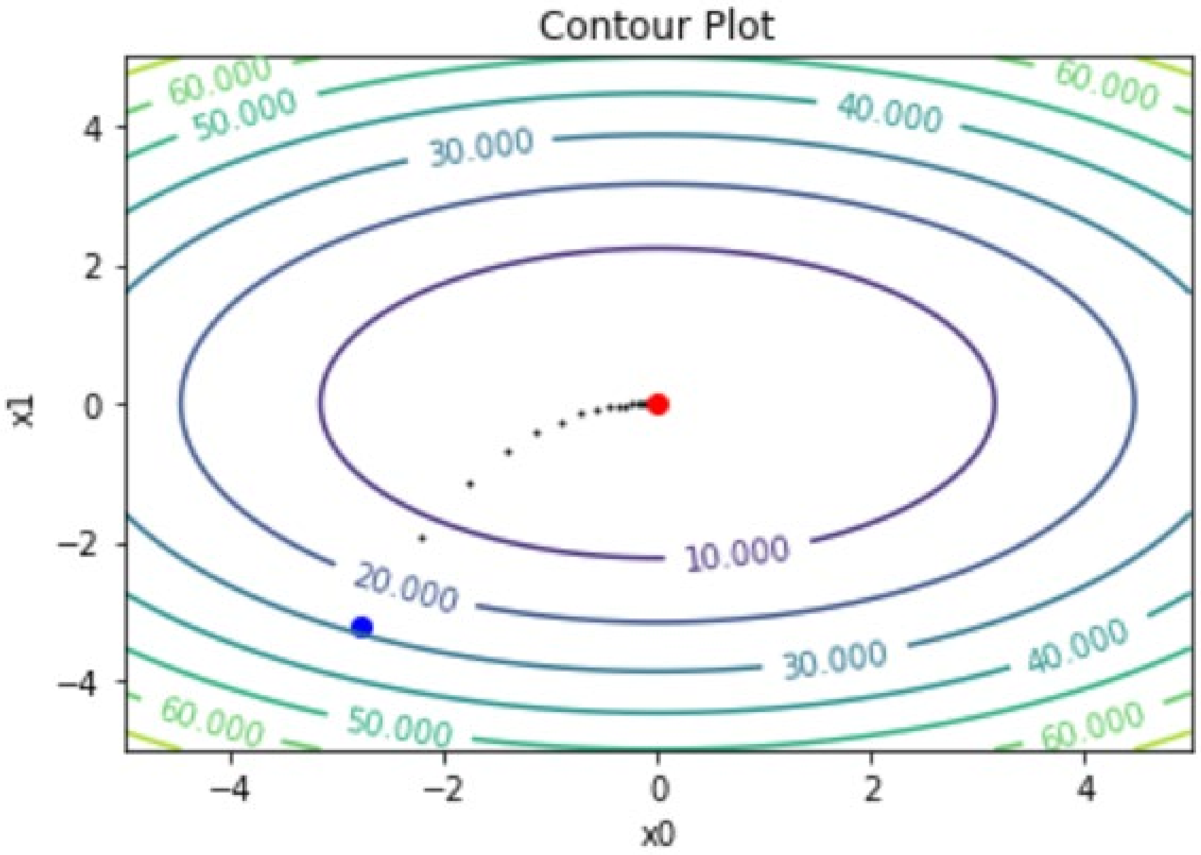
- “GD_HW1starter.ipynb” incomplete skeleton file is given in Canvas.
- First, complete the program to find a global minima of $f(x_0, x_1) = x_0^2 + 2x_1^2$ using Steepest Gradient Descent algorithm
- The function’s global minimum is at (x_0, x_1) = $(0, 0)$, where $f(x_0, x_1)$ = 0.
- A sample output data of the program should look like: ➔
- The output should be varying, since initial point will be generated at random

```
************************** Trial # =  1
[-2.2314268  -1.93731417] 12.4856
[-1.78514144 -1.1623885 ] 5.8890
[-1.42811315 -0.6974331 ] 3.0123
[-1.14249052 -0.41845986] 1.6555
[-0.91399242 -0.25107592] 0.9615
[-0.73119393 -0.15064555] 0.5800
[-0.58495515 -0.09038733] 0.3585
[-0.46796412 -0.0542324 ] 0.2249
[-0.37437129 -0.03253944] 0.1423
[-0.29949704 -0.01952366] 0.0905
[-0.23959763 -0.0117142 ] 0.0577
[-0.1916781  -0.00702852] 0.0368
[-0.15334248 -0.00421711] 0.0235
[-0.12267399 -0.00253027] 0.0151
[-0.09813919 -0.00151816] 0.0096
[-0.07851135 -0.0009109 ] 0.0062
[-0.06280908 -0.00054654] 0.0039
[-0.05024726 -0.00032792] 0.0025
[-0.04019781 -0.00019675] 0.0016
[-0.03215825 -0.00011805] 0.0010
[-2.57265994e-02 -7.08312755e-05] 0.0007
[-2.05812795e-02 -4.24987653e-05] 0.0004
Acceptable solution (last row above) found after 22 iterations.

System Success = 100.0% (1/1)
Total # of iterations used = 22
Avgerage # of iterations used = 22.0
```

HW1 – GD_HW2.ipynb (2/4)

- After making sure you get correct results, try to change `MaxTrial = 1` to `MaxTrial = 3`
- Update the code to introduce 2D contour lines and search points for each "Trial" by the GD algorithm, as shown right. Need to complete the function, "plot_contour(…)"
- Blue dot is the start point (-2.23…, -1.93…) and red dot is the acceptable point (-2.05812795e-02, -4.24987653e-05) found.

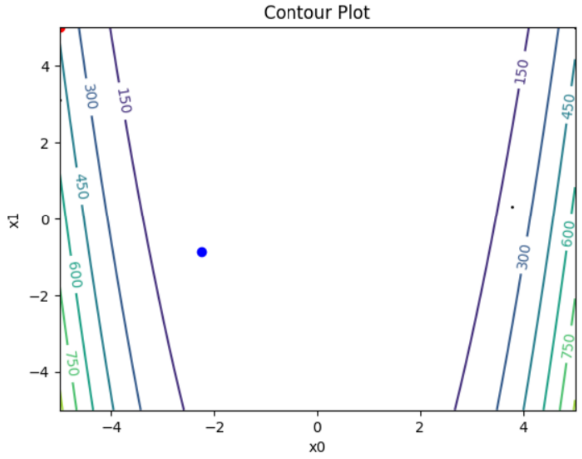


HW1 – GD_HW2.ipynb (3/4)

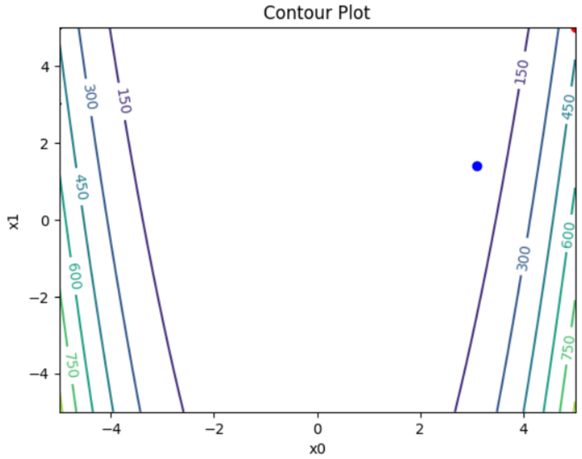
- Now introduce a new Rosenbrock function: $f(x_0, x_1) = (1 - x_0)^2 + \left(x_1 - x_0^2\right)^2$
- The function's global minimum is at $(x_0, x_1) = (1, 1)$, where $f(x_0, x_1) = 0$.
- Expected outputs for this function are:

```
************************** Trial # =  1
[-2.25777295 -0.8638732 ] 46.1515
[3.77758747 0.32840918] 202.0876
[-5.          3.11676077] 514.8762
[5. 5.] 416.0000
[-5.  5.] 436.0000
[5. 5.] 416.0000
[-5.  5.] 436.0000
   . . .

Max iterations reached without finding an acceptable solution.
```

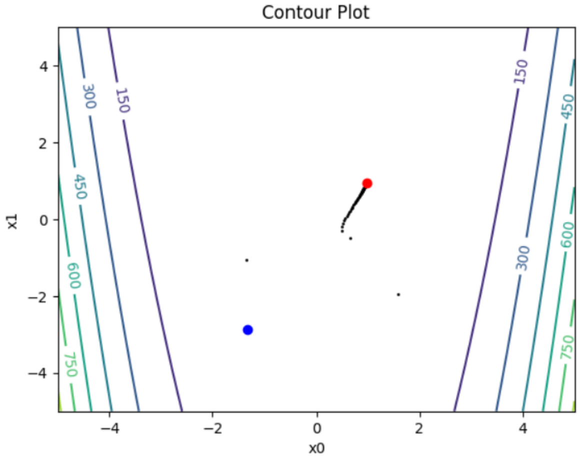



```
************************** Trial # =  2
[3.09293358 1.40050921] 71.0595
[-5.        3.033655] 518.5203
[5. 5.] 416.0000
[-5.  5.] 436.0000
[5. 5.] 416.0000
   . . .
Max iterations reached without finding an acceptable solution.
```




```
************************** Trial # =  3
[-1.32163154 -2.86422469] 26.6507
[ 1.58027743 -1.94203776] 20.0442
[-1.34191747 -1.05417486] 13.6351
[ 0.65889141 -0.48319139] 0.9578
[ 0.48534497 -0.29972554] 0.5514
   . . .

[0.9796013  0.95104141] 0.0005
Acceptable solution (last row above) found after 82 iterations.
```




```
System Success for Rosenbrock = 33.33% (1/3)
Total # of iterations used for Rosenbrock = 458
Average # of iterations used for Rosenbrock = 152.67
```

How to submit HW1?

- See “HW1” on Canvas for details

Techniques needed for HW1 to implement the Gradient Descent algorithm with a Contour Plot

- Python numpy library: “Numpy_intro.ipynb”
- Uniform random number generation using numpy: “Numpy_intro.ipynb”
- Scatter: “matplotlib_0.ipynb”
- Contour plot: “matplotlib_1contour.ipynb”

The above files are uploaded on Canvas

Review: Python Basics – Numpy

- Array oriented computing
- Linear Algebra
- Efficiently implemented multi-dimensional arrays
- Designed for scientific computation
- Used for plotting, Tensorflow, and ***Keras***
- Pronounced /ˈnʌmpaɪ/ (NUM-py) or sometimes /ˈnʌmpi/ (NUM-pee))

Review: Numpy append

Import numpy as np

```
narr = np.array([])
for i in range(5):
  narr = np.append(narr, i)
print(narr)
```

```
[0. 1. 2. 3. 4.]
```

Please test and study: Numpy_intro.ipynb

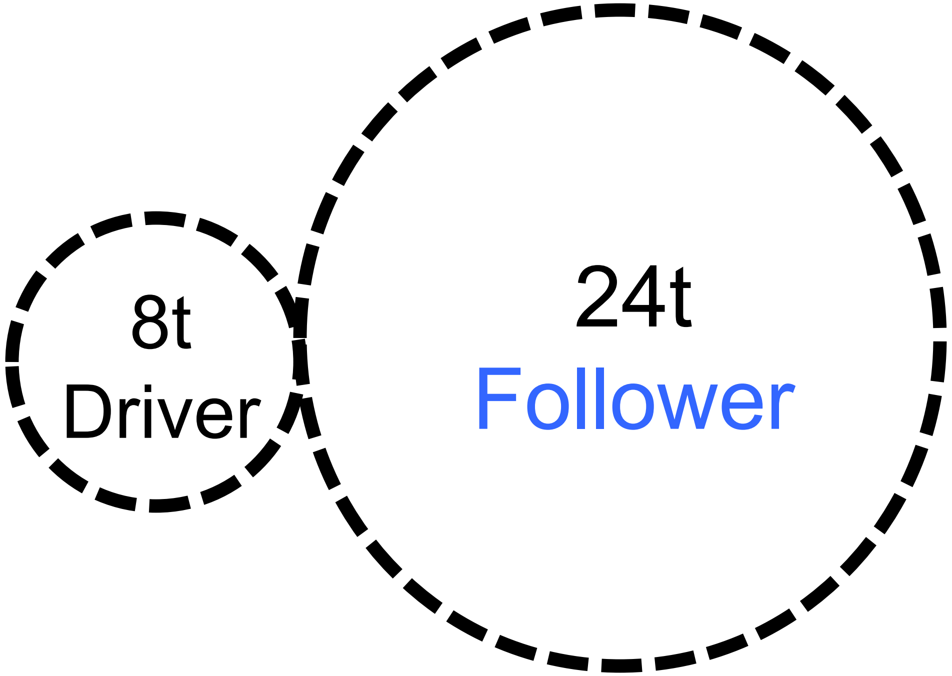
ES(1+1) with 1/5 rule

- ES: Evolution Strategy
- Important algorithm in this class
- Covered next week and another assignment

Another Old HW Example:

Gear Train Optimization Problem

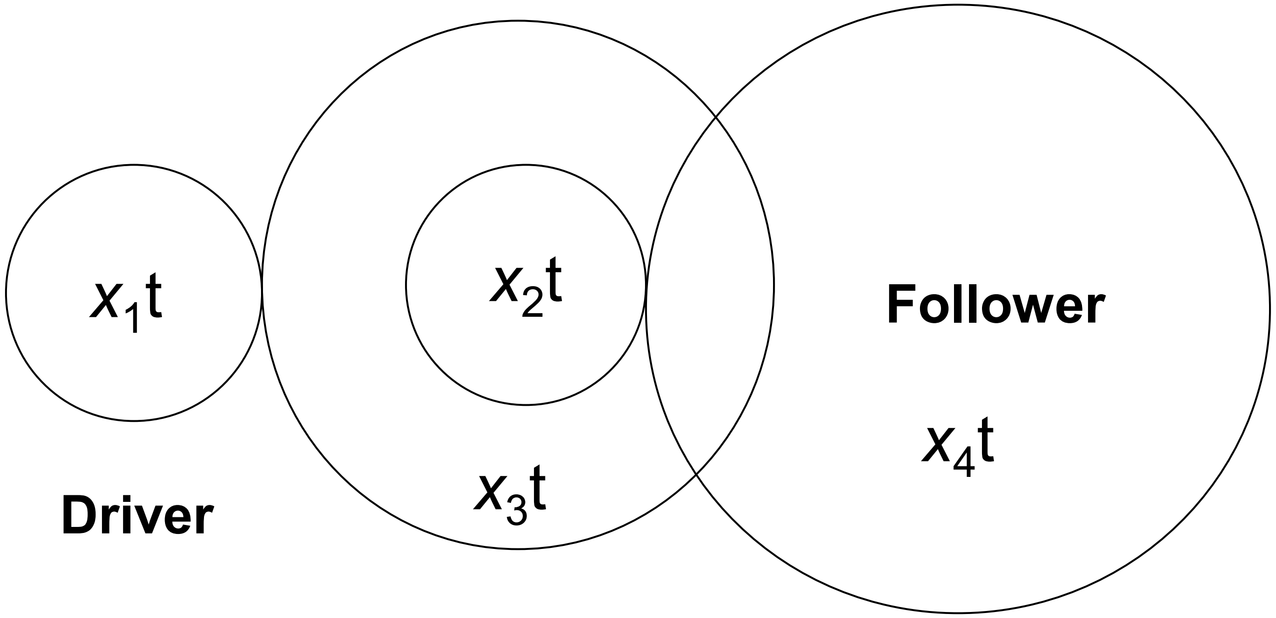
Gear Basics



Power (torque) of the Follower: $\tau_{24} = 24/8\ \tau_8$

Angular velocity of the Follower: $\omega_{24} = 8\omega_8/24$

Gear Train Design - An Optimization Problem [Kannan 1993]

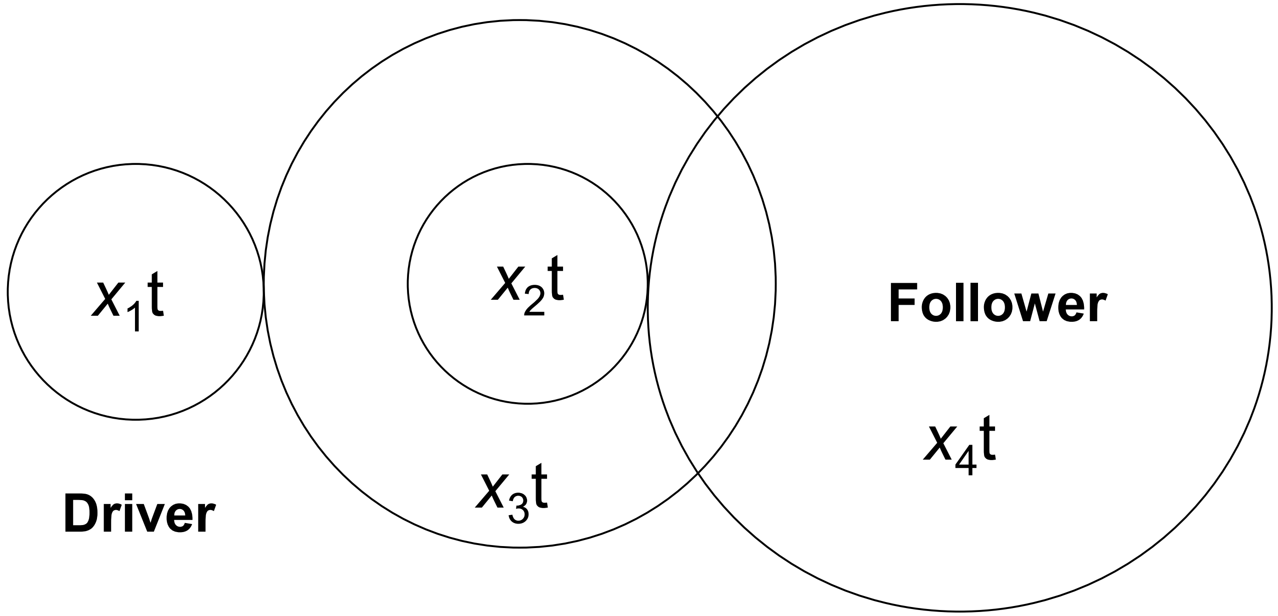


Objective:

to find the number of teeth for each gear
to produce 1/6.931 angular velocity ratio.

NLP (non linear programming) problem with strictly integer variables.

Gear Train Design Optimization Problem [Sandgren 1988]

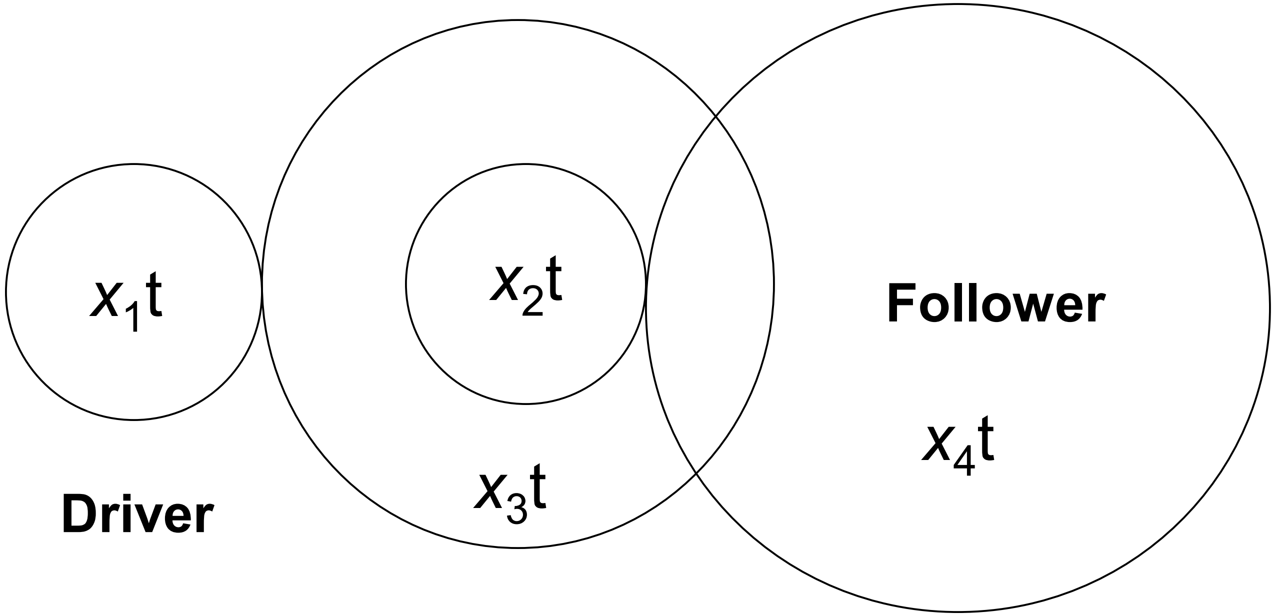


$$\text{Minimize } f(\mathbf{x}) = \left[\frac{1}{6.931} - \frac{x_1 x_2}{x_3 x_4}\right]^2$$

Subject to $12 \le x_1, x_2, x_3, x_4 \le 60,$

all $x_i\, s$ are integers.

Gear Train Design
Optimization Problem [Sandgren 1988]



- All possible gear teeth combinations: 49^4 = about 5.76 million

Review Question:

What is wrong with this minimization algorithm?

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k g_k, \textit{where}$$

$$g_k \equiv \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

Review Q:

What are Disadvantages of Calculus-based, for example, Gradient Descent Algorithm?

- Easily trapped in a local minima, not global minima
- (Some cases, it is hard to get partial derivatives!)
- Must find a partial derivative of the objective function, which may not always possible. Cannot be applied to unsmooth functions, non-differentiable functions, or computer program functions