Numerical Analysis

Basics

by Shangbang Long

shangbang.long@pku.edu.cn

Topics

- Scientific Computing
- Linear System
- Eigenvalue Problem
- Optimization
- Interpolation

Topic 1: Scientific Computing

Topic 1.1: Scientific Computing

Basic Idea

using advanced computing capabilities to solve complicated mathematical problems (usually with approximation)

General Strategy

- infinite -> finite
- differential -> algebraic
- nonlinear -> linear
- complicated -> approximated simple form

Topic 1.2: Approximation

Algorithm: computing surface area of Earth using formula $A = 4\pi r^2$

Involved approximation:

1. Earth is modeled as sphere, idealizing its true shape
2. Value for radius is based on empirical measurements and previous computations
3. Value for π is a rounded one.
4. Values for input data and results of arithmetic operations are rounded in computer

Topic 1.2.1: Error

Classification - 1

- **absolute error** = approximated value - true value
- **relative error** = absolute error / true value

Topic 1.2.2: Error

Classification - 2

- **truncation error**: resulting from algorithm (infinite -> finite)
- **rounding error**: resulting from limited precision representation

Example

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

- non-infinitesimal h results in truncation error.
- h much smaller than x results in rounding error.

Topic 1.3: Computer Arithmetics

Floating Point numbers

recall scientific notation:

$$12345 = 1.2345 \times 10^4$$

- mantissa: 1.2345
- exponent: 4

Floating Point numbers

Defined by 4 attributes:

- β: base / radix
- p: precision
- $[L, U]$: exponent range

$$x = \pm(\sum_{i=0}^{p-1} \frac{d_i}{\beta^i})\beta^E$$

where $d_i \in [0, \beta - 1], i = 0, \ldots, p - 1$,
$L \leq E \leq U$.

Floating Point numbers

limited p --> rounding error

Case: numerical anomoly in computing

$$lim_{n->\infty}(1+\frac{1}{n})^n$$

Topic 2: Linear System

Topic 2: Linear System

$$Ax = b$$

Analytical solution: $x = A^{-1}b$

Computing A^{-1} explicitly comes with problems:

1. Computation complexity: computing lots of determinants
2. Numerical instability: what if $|A| -> \frac{1}{\infty}$

Solving Triangular System

$$\begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix}$$

However, it's easy to solve when A is triangular...

First, solve x_3 from the last equation, then, substitute from the second equation and solve x_2, ...

LU Decomposition

Given the following linear system:

$$\boldsymbol{Ax} = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \boldsymbol{b}$$

I would like to transform A into triangular form, upper triangular, for example.

I need to eliminate any elements below the diagonal.

LU Decomposition

How to eliminate a certain segments of vectors?
Construct M_k from identity matrix as follows:

$$\boldsymbol{M_k a} = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -m_{k+1} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -m_n & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

where $m_i = a_i / a_k$, $i = k+1, \ldots, n$

LU Decomposition

$$\boldsymbol{M}_1\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix}$$

$$\boldsymbol{M}_1\boldsymbol{b} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 8 \\ 10 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix}$$

LU Decomposition

Similarly:

$$\boldsymbol{M_2 M_1 A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix}$$

$$\boldsymbol{M_2 M_1 b} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} = \boldsymbol{Mb}$$

LU Decomposition

By design:

$U = M_{n-1}...M_1 A = MA$ is upper triangular.

Let $L_i = M_i^{-1}$ which is lower triangular,
$L = L_1 L_2 ... L_{n-1} = M_1^{-1}...M_{n-1}^{-1}$ is lower triangular,

and $LU = M_1^{-1}...M_{n-1}^{-1} M_{n-1}...M_1 A = A$

a.k.a, $A = LU$

LU Decomposition

Other technical details...

- Partial/Complete Pivoting
- Cholesky Factorization for symmetric and positive definite A: $A = LL^T$

Topic 3: Eigenvalue Problem

Eigenvalue/vectors

$$Ax = \lambda x$$

Eigenvalue deomposition:

$$A = Q\Lambda Q^{-1}$$

Λ is the diagonal matrix whose diagonal elements are the corresponding eigenvalues

Eigenvalue/vectors

$$A^k = Q\Lambda^k Q^{-1}$$

Assume λ_1 is the largest eigenvalues, then:

$$lim_{k->\infty}\Lambda^k = \lambda_1^k * e_{1,1}$$

other eigenvalues just fade away...

which gives rise to the idea of **Power Iteration**

Power Iteration

Assume $x_0 = \sum \alpha_i v_i$

where v_is are eigen vectors, and αs represent weights. x_0 is a linear combination of v_is

$$x_k = Ax_{k-1} = A^2 x_{k-2} = \cdots = A^k x_0$$

$$= \sum_{i=1}^{n} \lambda_i^k \alpha_i v_i = \lambda_1^k \left(\alpha_1 v_1 + \sum_{i=2}^{n} (\lambda_i / \lambda_1)^k \alpha_i v_i \right)$$

x_k converges to multiple of eigenvector v_1 corresponding to dominant eigenvalue λ_1

Power Iteration

Algorithm:

1. randomly initialize x_0
2. compute: $x_{k+1} = Ax_k$
3. repeat step 2 until the ratio between x_{k+1} and x_k converges.

The convergent ratio is the eigenvalue

Power Iteration

The aforementioned algorithm may diverge as x gets larger...

Normalized Power Iteration

1. randomly initialize x_0
2. compute: $y_{k+1} = Ax_k$
3. normalize: $x_{k+1} = \frac{y_k}{||y_k||_\infty}$
4. repeat step 2 until x_k converges.

Then:

$\|y_k\|_\infty \to |\lambda_1|$, and $x_k \to v_1 / \|v_1\|_\infty$

Topic 4: Optimization

One-Dimentional Optimization

- Golden Section Search
- Successive Parabolic Interpolation
- Newton's Method

Golden Section Search

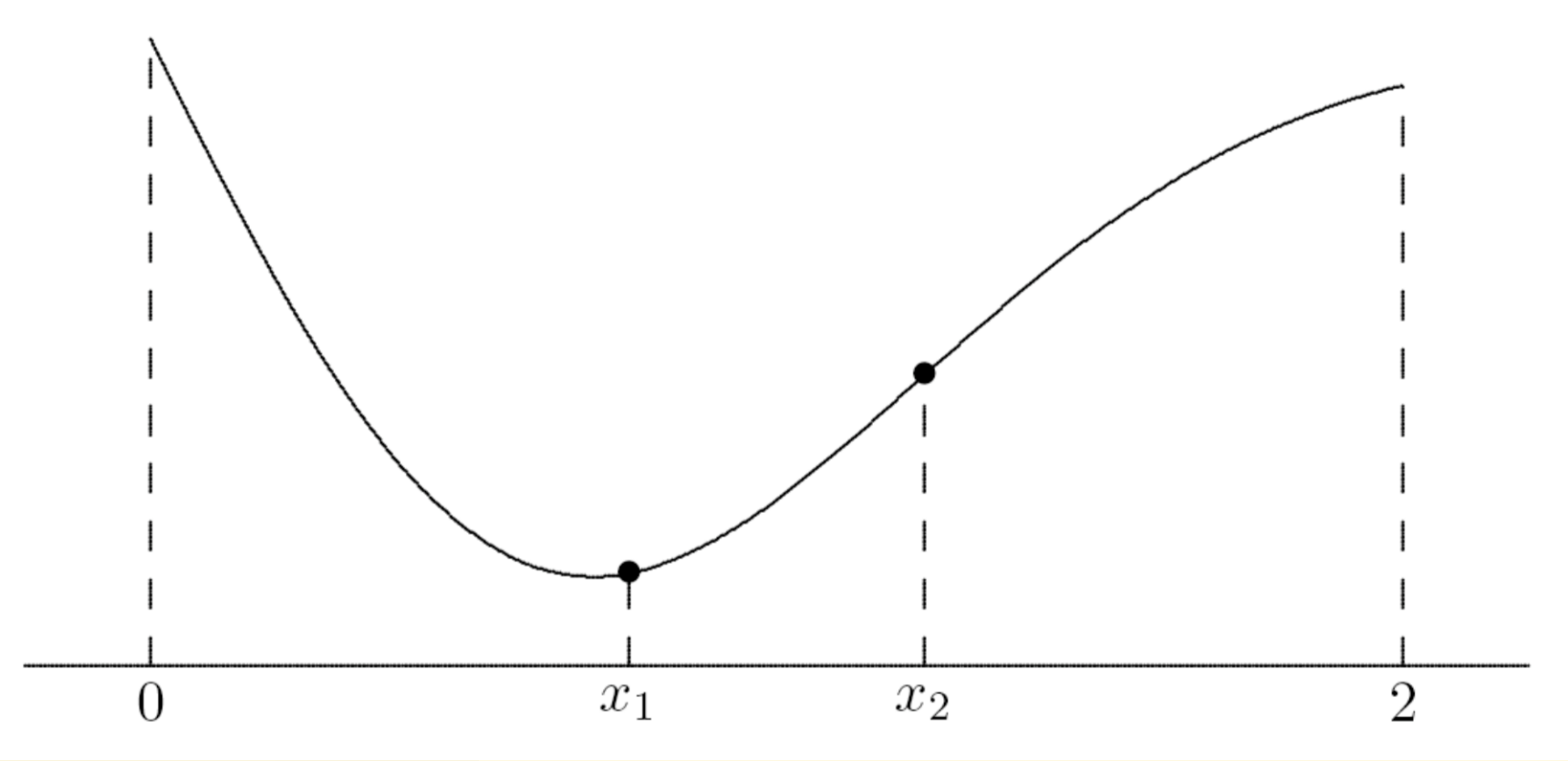
Unimodality

Real-valued function f is unimodal on interval $[a, b]$ if there is unique $x^* \in [a, b]$ such that $f(x^*)$ is minimum of f on
$[a, b]$, and f is strictly decreasing for $x \leq x^*$, strictly increasing for $x^* \leq x$

Golden Section Search

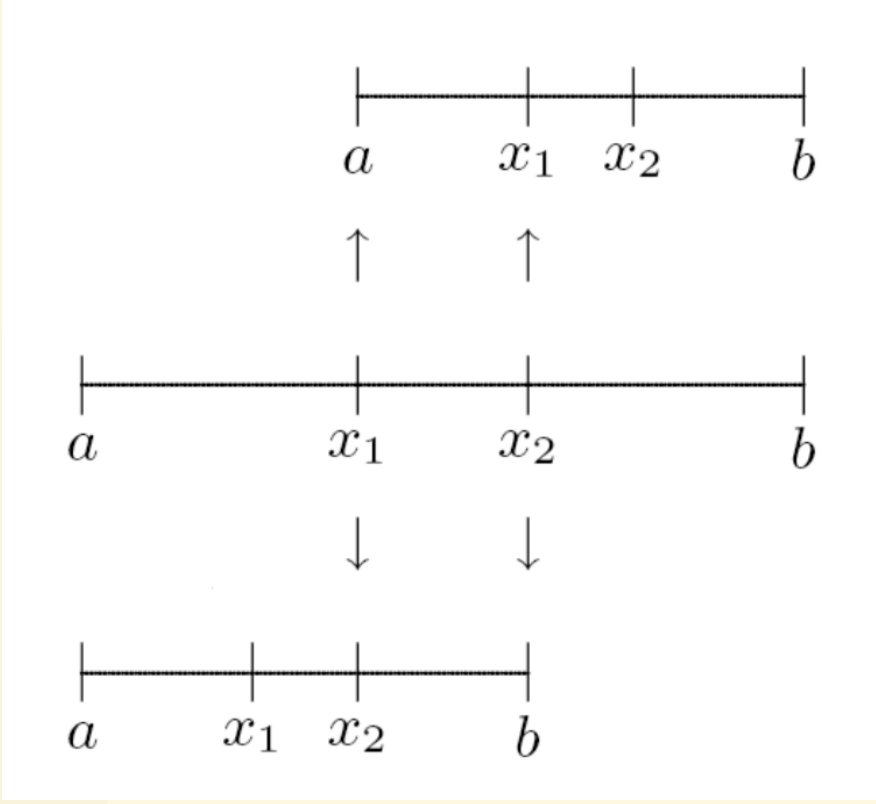
1. Suppose f is unimodal on $[a, b]$, and let x_1 and x_2 be two points within $[a, b]$, with $x_1 < x_2$.
2. Evaluating and comparing $f(x_1)$ and $f(x_2)$, we can discard either $(x_2, b]$ or $[a, x_1)$, with minimum known to lie in remaining subinterval.
3. Repeat 1-2, until $|x_1 - x_2| < \epsilon$

Golden Section Search



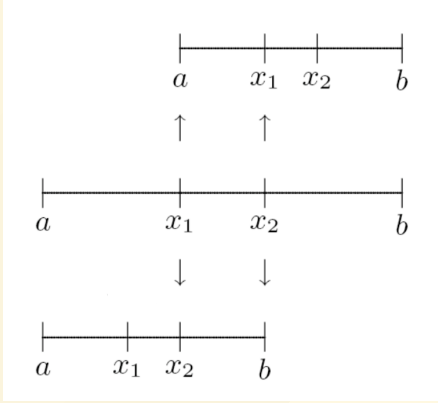
When $f(x_1) < f(x_2)$, it's impossible for the minimum to fall in $(x_2, 2]$

Why 'Golden'?



Reuse results from the last step, to reduce computational cost.

Why 'Golden'?



Let $x_2 - a = b - x_1 = \lambda, b - a = 1$(unit length)

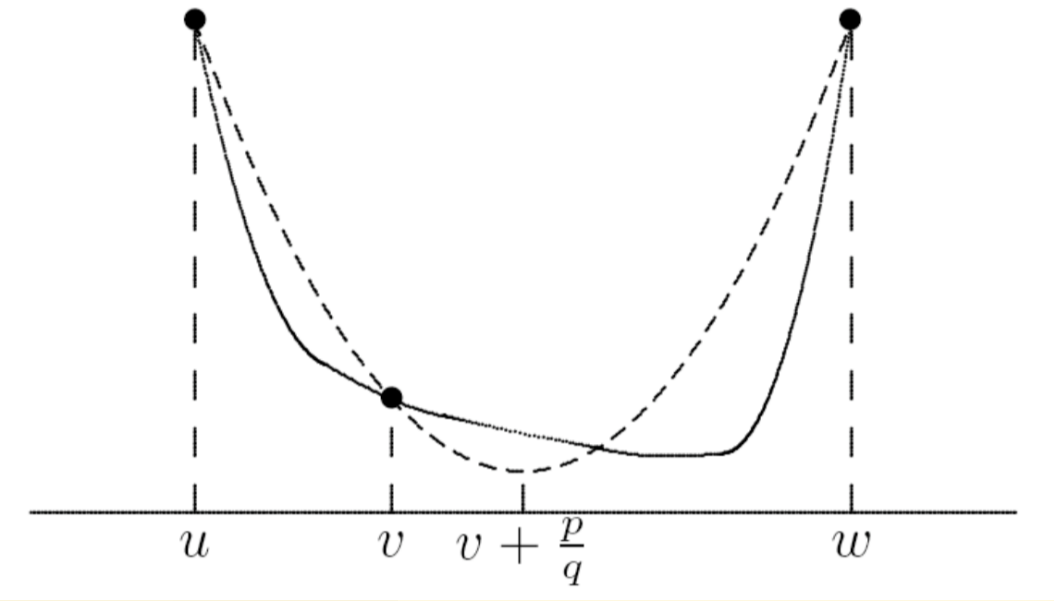
To reuse x_2 when $[a, x_1]$ is discarded, we have:

$$\frac{2\lambda-1}{\lambda} = \frac{x_2-x_1}{b-x_1} = \frac{x_{1,new}-a_{new}}{b_{new}-a_{new}} = \frac{x_1-a}{1} = 1 - \lambda$$

$\lambda \approx 0.618$ = golden ratio -1

Successive Parabolic Interpolation

Successive Parabolic Interpolation



1. Fit quadratic polynomial to three function values
2. Take minimum of quadratic to be new approximation to minimum of function
3. New point replaces oldest of three previous points and process is repeated until convergence

Newton's Methods

Based on truncated Taylor series:

$$f(x+h) \approx f(x) + f'(x)h + \frac{f''(x)}{2}h^2$$

Minimizing right hand side w.r.t. h:

$$h^* = -f'(x)/f''(x)$$

Therefore: $x_{k+1} = x_k - f'(x_k)/f''(x_k)$

Multi-Dimentional Optimization

Multi-Dimentional Optimization

- Gradient Descent
- Newton's Method
- Other Non-Convex Optimization

Gradient Descent

Gradient Descent

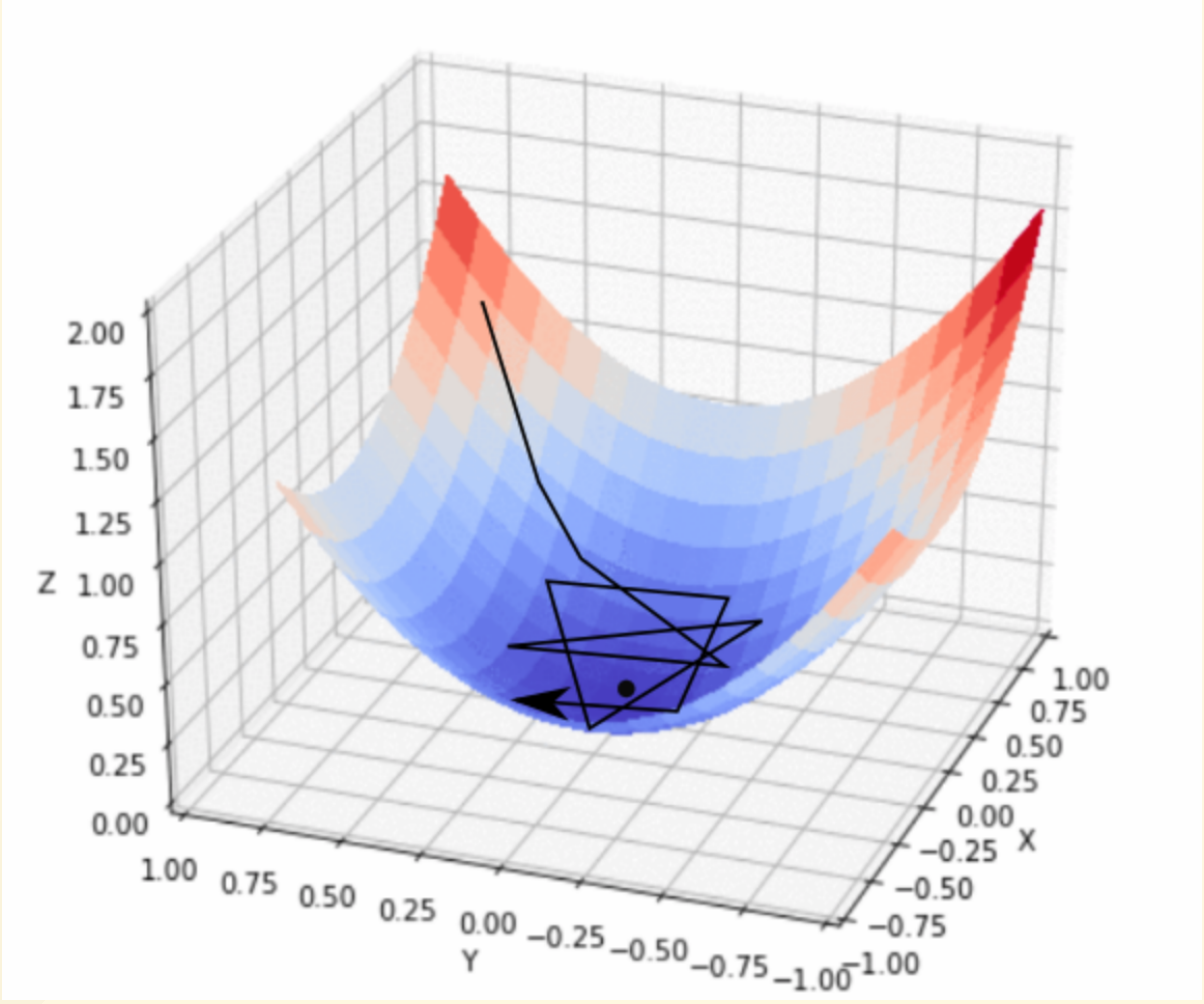
1. Start with randomly initialized X_0
2. Iteratively update towards the most rapidly descending direction indicated by first-order partial derivative:

$$X_{k+1} = X_k - \alpha_k \nabla f\left(X_k\right)$$

where α_k is called *step size* or *learning rate*, and is usually set as a constant.

It only converges to local minimum.

Gradient Descent



Newton's Method

Update rules:

$$x_{k+1} = x_k - H_f^{-1}(x_k)\,\nabla f(x_k)$$

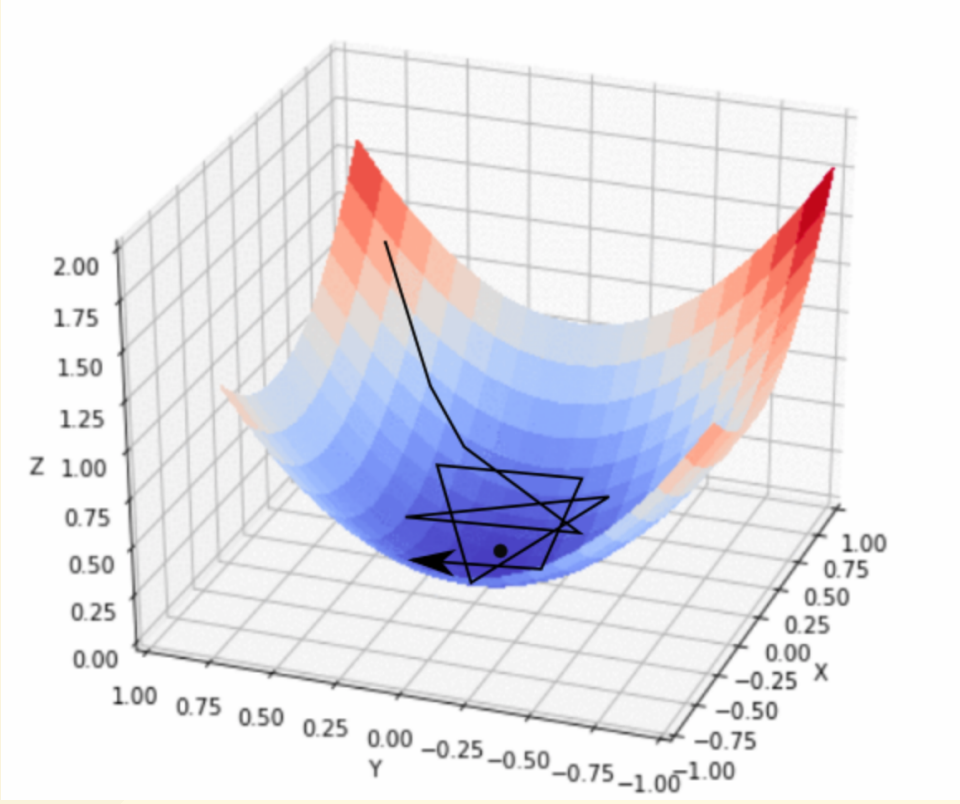
where H is the Hessian matrix of second partial derivatives of f,

$$\{H_f(x)\}_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

Other Non-Convex Optimization

- Momentum Gradient Descent
- Adagrad
- Adadelta
- Adam

Momentum Gradient Descent



In gradient descent: gradient resembles 'speed';

In momentum GD: gradient resembles 'acceleration'

Momentum Gradient Descent

1.Start with randomly initialized X_0.

2.Initialize momentum M_0 to zero vector.

3.Update momentum as the moving average of gradient g_t: $M_t = \mu * M_{t-1} + g_t$

4.Update parameters: $X_{k+1} = X_k - \alpha_k M_t$

where μ controls how fast momentum is updated.

It's **believed** that momentum GD can escape local minimum.

Adagrad

-- Adaptive Gradient Algorithm

Adagrad

Recall Newton's method:

$$x_{k+1} = x_k - H_f^{-1}\left(x_k\right) \nabla f\left(x_k\right)$$

As the number of parameters grows, it will be more time-consuming to comput H_f.

We can approximate H_f instead:

$$B_t := diag\left(\sum_{j=1}^{t} \nabla f_{i_j}\left(x_j\right) \cdot \nabla f_{i_j}\left(x_j\right)^\top\right)^{1/2}$$

Adagrad

The computation of B_t simply keep track of the squared first-order gradients. It can be computed as follows:

1. Initialization.
2. Update: $n_t = n_{t-1} + g_t^2$
3. $X_{k+1} = X_k - \frac{\alpha_k}{\sqrt{n_t + \epsilon}} * g_t$

It can regularize the gradients, preventing it from being too large or too small.

Adadelta

In Adagrad, n_t is monotonically increasing, and will finally stop the updates, when $\frac{\alpha_k}{\sqrt{n_t+\epsilon}} = 0$.

In Adadelta:

n_t is replaced with a decaying moving average:

$$n_t = \nu * n_{t-1} + (1 - \nu) * g_t^2$$

where ν is a hyper-parameter indicating the decay rate of n_t.

Adam: Adaptive Moment Estimation

-- Adadelta with Momentum

Adam

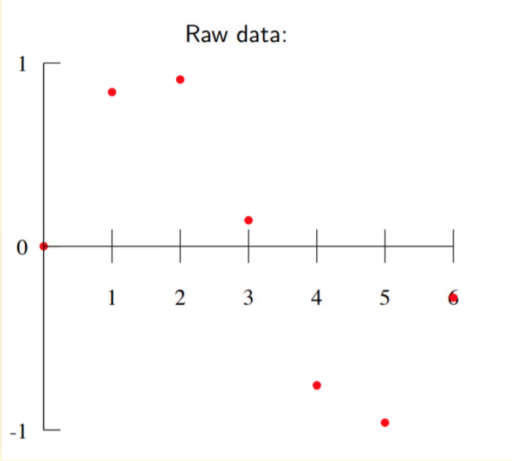
1. MA of gradient: $m_t = \mu * m_{t-1} + (1 - \mu) * g_t$
2. MA of second moment:
   $n_t = \nu * n_{t-1} + (1 - \nu) * g_t^2$
3. Adjustment: $\hat{m}_t = \frac{m_t}{1-\mu^t}, \hat{n}_t = \frac{n_t}{1-\nu^t}$
4. Update: $X_{k+1} = X_k - \frac{\hat{m}_t}{\sqrt{\hat{n}_t + \epsilon}} * \eta$

Remember: these are simply what people believe to be able to solve non-convex optimization, especially in training neural networks.

Topic 5: Interpolation

Interpolation

Interpolation is well discussed in Prof. Wang's lecture.



Recall: when we infer the continuous function from discretely sampled points, denoted as $\{(x_i, y_i)\}_{i=1}^{n}$, we can use 1) linear, 2) near neighbour, 3) polynomial, 4) spline, ...

_____polation

In this part, we will talk about **extrapolation**.

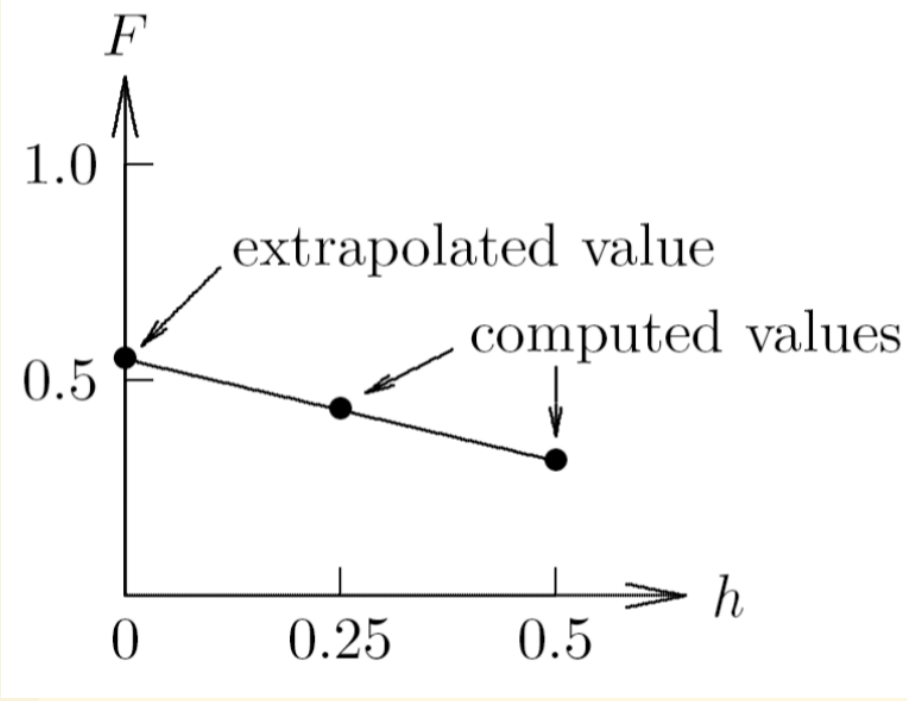


Image we want: $y^* = f(x^*), x^* < x_i$ or $x^* > x_i$ for any $x_i \in \{(x_i, y_i)\}_{i=1}^n$

Extrapolation

In many problems, such as numerical integration or differentiation, approximate value for some quantity is computed based on some step size h:

$$f'(x) = \lim_{h->0} \frac{f(x+h) - f(x)}{h}$$

Ideally, we would like to obtain limiting value as step size approaches zero, but we cannot take step size arbitrarily small because of excessive cost or rounding error.

Extrapolation

Based on values for nonzero step sizes, however, we may be able to estimate value for step size of zero.

One way to do this is called Richardson extrapolation.

Extrapolation

Let $F(h)$ denote value obtained with step size h. $F(0)$ is what we desire ideally.

If we compute value of F for some nonzero step sizes, and if we know theoretical behavior of $F(h)$ as $h -> 0$, then we can extrapolate from known values to obtain approximate value for $F(0)$.

Extrapolation

Suppose F is a linear function:

$$F(h) = a_0 + a_1 h + \mathcal{O}\left(h^2\right)$$

$a_0 = F(0)$ is out target.

Extrapolation

We compute:

1. $F(h) = a_0 + a_1 h + \mathcal{O}\left(h^2\right)$, and
2. $F(h/q) = a_0 + a_1 \frac{h}{q} + \mathcal{O}\left(h^2\right)$ for some positive integer q.

a_0 is then approximated as:

$$a_0 = F(h) + \frac{F(h) - F(\frac{h}{q})}{q^{-1} - 1}$$

Thanks!