18-660: Numerical Methods for Engineering Design and Optimization

Xin Li
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213

Electrical & Computer ENGINEERING

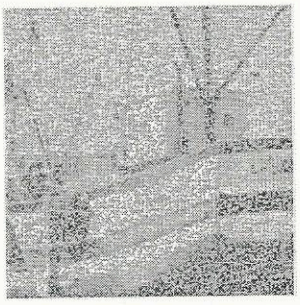
Outline

- Project 2: Image Recovery
  - Objective
  - Methods
  - Submission details

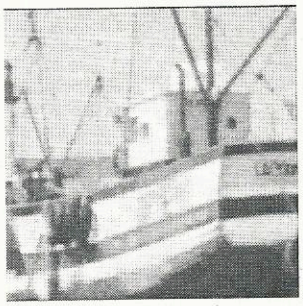
Objective

- Apply compressed sensing to recover a full image from a small number of sampled pixels

Original image



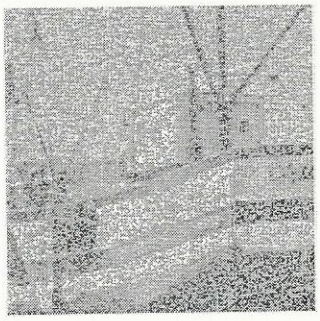
Sampled image



Recovered image

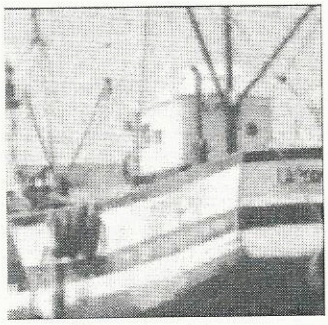
Application

- Corrupted image recovery



Corrupted image

Recover

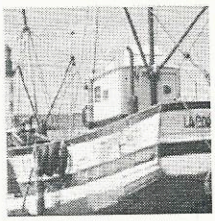


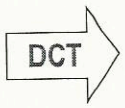
Recovered image

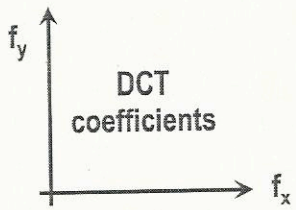
Outline

- Objective
- Methods
- Submission details

2-D DCT Transform







$$g(x,y) = \sum_{u=1}^{P}\sum_{v=1}^{Q} \alpha_u \alpha_v \cos\frac{(2x-1)(u-1)\pi}{2P} \cos\frac{(2y-1)(v-1)\pi}{2Q} G(u,v)$$

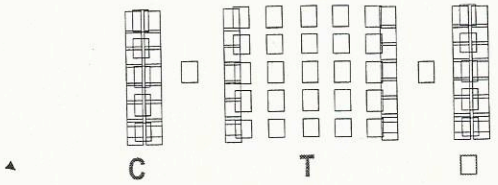
Image pixel — Transformation — DCT coefficient

$$x,u = 1,2,\ldots,P$$

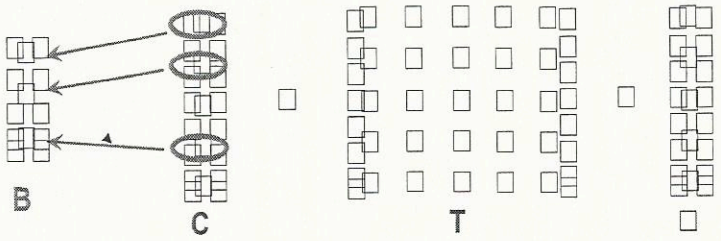
$$y,v = 1,2,\ldots,Q$$

$$\alpha_u = \begin{cases} \sqrt{1/P} & u = 1 \\ \sqrt{2/P} & 2 \le u \le P \end{cases}$$

$$\alpha_v = \begin{cases} \sqrt{1/Q} & v = 1 \\ \sqrt{2/Q} & 2 \le v \le Q \end{cases}$$



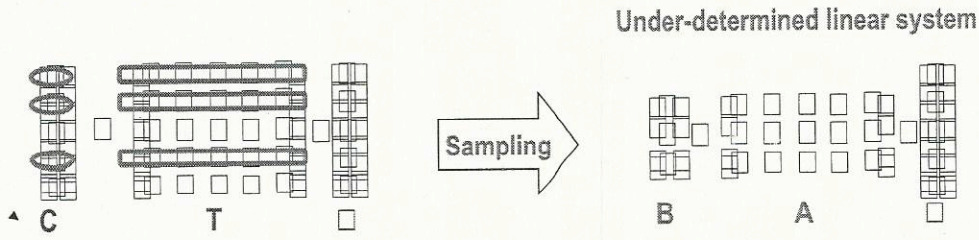
Compute DCT Coefficients



- Transformation matrix T is known
- How to compute DCT coefficients?
  - If the full image C is given: $\alpha = T^{-1} C$
  - If only a few samples of C (sampled vector B) is given, can we get an approximate solution of α?

Compute DCT Coefficients

- Sampled image leads to an underdetermined linear system: $B = A \alpha$



- Compute DCT coefficients α by *solving an underdetermined linear system*: $B = A \alpha$

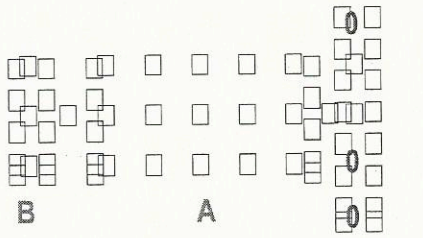
Compressed sensing

- Once DCT coefficients are computed, recover the full image by $C = T \alpha$

Image Recovery

- Apply compressed sensing to determine approximate DCT coefficients from sampled pixels
- Apply inverse DCT transform to recover the full image

Compressed Sensing



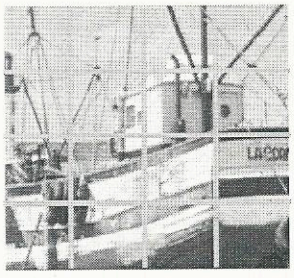
- Solve the underdetermined linear system: B = A
- If α is sparse:
  - Find the sparse solution by L_0-norm regularization

$$\min \quad \|A\alpha - B\|_2^2$$
$$\text{S.T.} \quad \|\alpha\|_0 \leq \varepsilon$$

Implementation Issues

- α is expected to be sparse
- How to solve the optimization with L_0-norm regularization?
- How to determine □?

Construct Sparse DCT Coefficients



- DCT coefficients of a large image are often not sparse
- Solution: break a large image into small blocks
  - Each small block corresponds to few non-zero DCT coefficients
  - You can try different block size K in your own experiments
  - 8x8 block is suggested for the small test image
  - 16x16 block is suggested for the large test image

Solve L_0-norm Regularization

- **Use Orthogonal Matching Pursuit (OMP) to solve:**

$$\min_{\square} \quad \|A\square\square B\|_2^2$$
$$\text{S.T.} \quad \|\square\|_0 \;\square\square$$

  - Step 1: Set F = B, □ = { } and p = 1
  - Step 2: Calculate inner product values $\square_i$ = <A_i, F>
  - Step 3: Identify the index s for which $|\square_s|$ takes the largest value
  - Step 4: Update □ by □ = □ □ {s}
  - Step 5: Approximate F by the linear combination of {A_i; i □ □}

$$\min_{\square_i, i\square\square} \left\| \square_{i\square\square} \square_i \square A_i \square B \right\|_2^2$$

  - Step 6: Update F

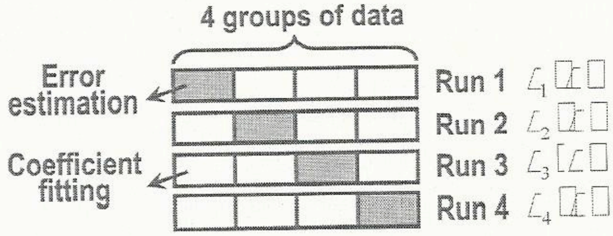
$$F \square B \square \square_{i\square\square} \square_i \square A_i$$

  - Step 7: If p < □, p = p+1 and go to Step 2. Otherwise, stop.

$$\square_i \square 0 \quad \square i \square\square\square$$

Determine □ by Cross Validation

- **For each □ in a given list**
  - Calculate DCT coefficients from the training set
  - Estimate approximation "error" from the testing set
    - Use mean square error as a measurement of the "error"
  - Example: 4-fold cross validation



$$Error \square\square\square[\square_1 \square\square\square \square_2 \square\square\square \square_3 \square\square\square \square_4 [\square\square]/4$$

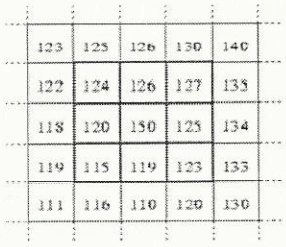
- **Select □ with the minimum error**

Cross Validation with Random Subsets

- **N-fold cross validation**
  - Distribute all data into N folds such that the numbers of samples in all folds are equal
  - Take one fold as the test set at each iteration
  - Training-and-test process is repeated for N times
- **Cross validation with random subsets**
  - At each iteration, randomly draw *m* samples to form the test set and use all other samples as the training set
    - Use *m = floor(S/6)* in this project, where *S* is the total number of samples
  - Repeat the training-and-test process for M times
    - Use *M = 20* in this project
- **Apply cross validation with random subsets in this project**
  - When the data set is small, using this method with large M is more accurate than N-fold cross validation
  - It is easy to implement if the data set cannot be divided equally into N folds

Median Filter

- **Median filtering is to replace each pixel in an image by the median of its neighborhood**
- **Median filtering algorithm where filter size is m□n:**
  - Sort all pixel values in an m□n block, centered at (x,y), to find the median
  - Replace the pixel value f(x,y) by the median



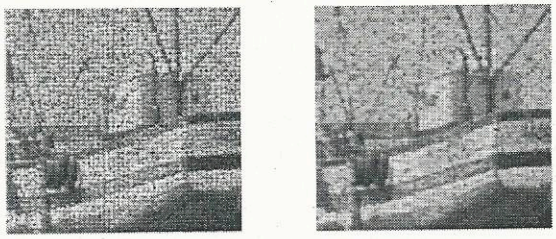
Neighbourhood values:

115, 119, 120, 123, 124, 125, 126, 127, 150

Median value: 124

Median Filter

- **Apply median filter (MF) to improve the quality of recovered images**
  - You can use MATLAB function medfilt2
  - Set filter size 3□3
- **Compare the error of the recovered image w/ MF and w/o MF**



Recovered images (4□4 block)
(Left: w/o median filter, Right: w/ median filter)

Summary of Image Recovery

- **Read in an image**
- **Break image into blocks where block size is K□K**
- **For each block:**
  - Randomly sample a few pixels where sample size is S
    - No repetition for each pixel
  - Compute DCT coefficients from the samples
    - Use OMP algorithm
    - □ Is determined by cross validation using random subsets
  - Apply inverse DCT transform to recover the block
- **Combine all recovered blocks into a full image**
  - Apply median filter to improve image quality

Critical Functions

- The following three functions are critical for your implementation and grading:
  - Data sampling
  - OMP solver
  - Optimal □selection via cross validation

Quality Measurement

- Mean square error between recovered image and original image is used to measure the quality of recovery
  - $\frac{1}{W \cdot H} \sum_{\substack{1 \le x \le W \\ 1 \le y \le H}} [\hat{g}(x,y) - g(x,y)]^2$
  - W : image width

    H : image height
  - $\hat{g}(x,y)$: pixels of recovered image

    $g(x,y)$: pixels of original image

Outline

- Objective
- Methods
- Submission details

Project Files

All files for this project can be found from the distributed package

- A report template
  - Proj2.doc
- Two test images
  - fishing_boat.bmp
  - lena.bmp
- Three MATLAB functions
  - imgRead.m
  - imgShow.m
  - imgRecover.m

MATLAB Functions

- **imgRead.m**
  - Load test image: e.g. A = imgRead('lena.bmp')
  - Input: input file name
  - Output: H-by-W matrix
    - A(i,j): image pixel at i-th row and j-th column
    - W: image width
    - H: image height
- **imgShow.m**
  - display image: e.g. imgShow(B);
  - Input: H-by-W matrix
    - B(i,j): image pixel at i-th row and j-th column
    - W: image width
    - H: image height

MATLAB Functions

- **imgRecover.m**
  - Should be implemented by you
  - Syntax: imgOut = imgRecover(imgIn, blkSize, numSample)
  - Input:
    - imgIn: H-by-W input matrix
    - blkSize: block size, i.e., K on Slide 12
    - numSample: number of samples per block, i.e., S on Slide 15
  - Output: H-by-W matrix
    - Recovered image without median filtering

Project Submission

- **You should zip the MATLAB code (.m) and figures (.fig) into a single file and submit it to the course web site**
  - Your code must work on Linux cluster without any modification
  - Follow instructions specified in the WORD template
- **You should also submit a PDF report (at most 4 pages) to the course web site**
  - Follow instructions specified in the WORD template

Grading Criteria

- **75% for MATLAB code and results**
- **25% for project report**

18-660: Numerical Methods for Engineering Design and Optimization

Xin Li
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213

Electrical & Computer ENGINEERING

Overview

- **Nonlinear Equation Solver**
  - Newton-Raphson method
  - Binary search

Nonlinear Algebraic Equation

- Many physical systems are nonlinear and must be mathematically described as a nonlinear equation

$$F(x) = 0$$

- Example: nonlinear ordinary differential equation

$$F[\dot{x}(t), x(t), u(t)] = 0 \quad x(0) = X$$

$x(t)$: N-dimensional vector of unknown variables

$u(t)$: Vector of input sources

F: Nonlinear operator

X: Initial condition

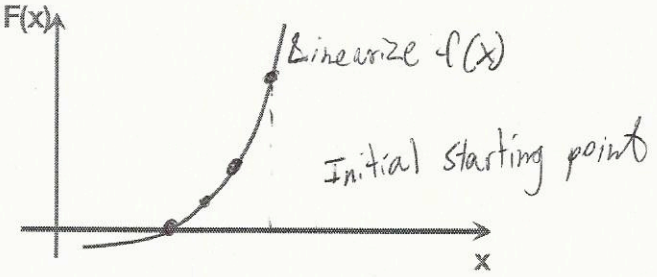
Nonlinear Algebraic Equation

$$F(x) = 0$$

- Closed-form solution cannot be derived for a general nonlinear equation
- Iterative algorithm must be applied to find an approximate solution (i.e., numerical solution)
- Newton-Raphson method is a widely-used algorithm to solve nonlinear algebraic equation

Newton-Raphson Method

- A one-dimensional example



$F(x)=0$

K-th iteration $F(x^{(k)}) \neq 0$

update solution $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$

Newton-Raphson Method

- Taylor series approximation
  - For a convergent series and sufficiently small $\Delta x^{(k)}$

$$F[x^{(k)} + \Delta x^{(k)}] = F[x^{(k)}] + F'[x^{(k)}] \cdot \Delta x^{(k)} + \cdots$$

$x^{(k+1)}$

- Newton-Raphson method relies on 1st-order Taylor expansion
  - The function F(•) must be smooth and its derivative exists

Newton-Raphson Method

- **Apply first-order Taylor expansion**

$$F\left[x^{(k+1)}\right] \approx F\left[x^{(k)}\right] + F'\left[x^{(k)}\right] \cdot \underbrace{\left[x^{(k+1)} - x^{(k)}\right]}_{\Delta x^{(k)}} = 0$$

$$\Delta x^{(k)} = -\frac{F\left[x^{(k)}\right]}{F'\left[x^{(k)}\right]}$$

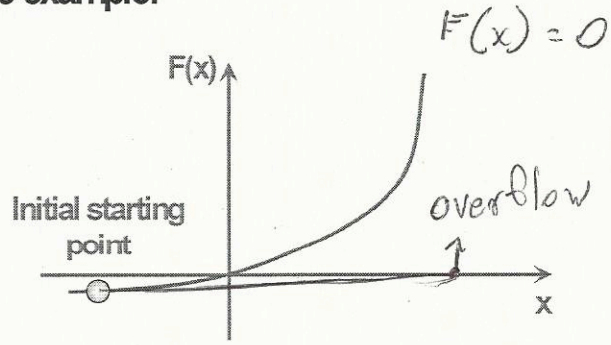
$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$$

$$= x^{(k)} - \frac{F\left[x^{(k)}\right]}{F'\left[x^{(k)}\right]}$$

- **Global convergence is not guaranteed in general**
  - Many techniques exist to improve convergence

Newton-Raphson Method

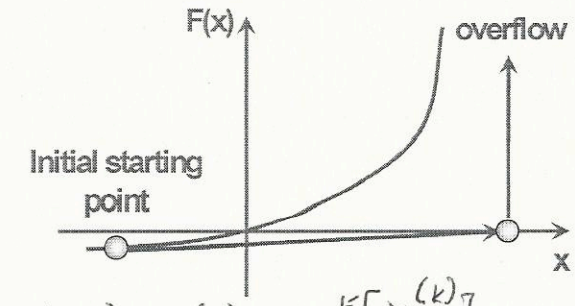
- **A simple example:**



$$x^{(k+1)} = x^{(k)} - \frac{F\left[x^{(k)}\right]}{F'\left[x^{(k)}\right]}$$

Newton-Raphson Method

- A simple example:



$$x^{(k+1)} = x^{(k)} - \lambda \frac{F[x^{(k)}]}{F'[x^{(k)}]} \qquad (0 < \lambda < 1)$$

Newton-Raphson Method

$$F\left[x^{(k+1)}\right] \approx F\left[x^{(k)}\right] + F'\left[x^{(k)}\right] \cdot \left[x^{(k+1)} - x^{(k)}\right] = 0$$

Accurate if $x^{(k)}$ and $x^{(k+1)}$ are close

- Start Newton-Raphson iteration from a good initial solution
- In practice, a good initial solution may be unknown
- An alternative approach is to randomly select multiple initial solutions and apply Newton-Raphson method to each of them

A Simple Example

$$F(x) = \log(x) = 0$$

$$F'(x) = \frac{1}{x}$$

- **Start from an initial solution $x^{(0)} = 0.1$**

$$F[x^{(0)}] = \log(0.1) = -2.3$$

$$F'[x^{(0)}] = \frac{1}{0.1} = 10$$

A Simple Example

$$F(x) = \log(x) = 0 \qquad x^{(0)} = 0.1$$

$$F'(x) = \frac{1}{x} \qquad F[x^{(0)}] = -2.30$$

$$F'[x^{(0)}] = 10$$

- **Iteration #1**

$$F[x^{(1)}] \approx F[x^{(0)}] + F'[x^{(0)}] \cdot [x^{(1)} - x^{(0)}] = 0$$

$$-2.3 + 10[x^{(1)} - 0.1] = 0$$

$$x^{(1)} = 0.33$$

$$F[x^{(1)}] = \log(0.33) = -1.11$$

$$F'[x^{(1)}] = \frac{1}{0.33} = 3$$

A Simple Example

$$F(x) = \log(x) = 0 \qquad x^{(1)} = 0.33$$

$$F'(x) = \frac{1}{x} \qquad F\left[x^{(1)}\right] = -1.11$$

$$F'\left[x^{(1)}\right] = 3$$

- **Iteration #2**

$$F\left[x^{(2)}\right] \approx F\left[x^{(1)}\right] + F'\left[x^{(1)}\right] \cdot \left[x^{(2)} - x^{(1)}\right] = 0$$

$$-1.11 + 3\left[x^{(2)} - 0.33\right] = 0$$

$$x^{(2)} = 0.7$$

$$F\left[x^{(2)}\right] = \log(0.7) = -0.36$$

$$F'\left[x^{(2)}\right] = \frac{1}{0.7} = 1.43$$

A Simple Example

$$F(x) = \log(x) = 0 \qquad x^{(2)} = 0.70$$

$$F'(x) = \frac{1}{x} \qquad F\left[x^{(2)}\right] = -0.36$$

$$F'\left[x^{(2)}\right] = 1.43$$

- **Iteration #3**

$$F\left[x^{(3)}\right] \approx F\left[x^{(2)}\right] + F'\left[x^{(2)}\right] \cdot \left[x^{(3)} - x^{(2)}\right] = 0$$

$$-0.36 + 1.43\left[x^{(3)} - 0.7\right] = 0$$

$$x^{(3)} = 0.95$$

$$F\left[x^{(3)}\right] = \log(0.95) = -0.05$$

$$F'\left[x^{(3)}\right] = \frac{1}{0.95} = 1.05$$

Newton-Raphson Method

- **Extend to multi-dimensional case**

$$F(x)=0 \quad \begin{cases} F_1(x)=0 \\ F_2(x)=0 \\ \vdots \\ F_N(x)=0 \end{cases} \qquad X=\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

  - Apply first-order Taylor expansion for all F_i's

$$F_1\left[x^{(k+1)}\right]=F_1\left[x^{(k)}\right]+\left.\frac{\partial F_1}{\partial x_1}\right|_k \cdot\left[x_1^{(k+1)}-x_1^{(k)}\right]+\left.\frac{\partial F_1}{\partial x_2}\right|_k \cdot\left[x_2^{(k+1)}-x_2^{(k)}\right]+\cdots=0$$

$$F_2\left[x^{(k+1)}\right]=F_2\left[x^{(k)}\right]+\left.\frac{\partial F_2}{\partial x_1}\right|_k \cdot\left[x_1^{(k+1)}-x_1^{(k)}\right]+\left.\frac{\partial F_2}{\partial x_2}\right|_k \cdot\left[x_2^{(k+1)}-x_2^{(k)}\right]+\cdots=0$$

$$\vdots$$

Newton-Raphson Method

- **Re-write into a matrix form**

$$\begin{bmatrix} F_1\left[x^{(k+1)}\right] \\ F_2\left[x^{(k+1)}\right] \\ \vdots \end{bmatrix}=\begin{bmatrix} F_1\left[x^{(k)}\right] \\ F_2\left[x^{(k)}\right] \\ \vdots \end{bmatrix}+\begin{bmatrix} \left.\frac{\partial F_1}{\partial x_1}\right|_k & \left.\frac{\partial F_1}{\partial x_2}\right|_k & \cdots \\ \left.\frac{\partial F_2}{\partial x_1}\right|_k & \left.\frac{\partial F_2}{\partial x_2}\right|_k & \cdots \\ \vdots & \vdots & \vdots \end{bmatrix} \cdot \left(\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \end{bmatrix}-\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \end{bmatrix} \right)=0$$

$F[x^{(k+1)}]$ $F[x^{(k)}]$ $\frac{\partial F}{\partial x}\Big|_k$ $\Delta x^{(k)}$

jacobian matrix

Newton-Raphson Method

- The new $\Delta x^{(k)}$ can be solved by a linear solver

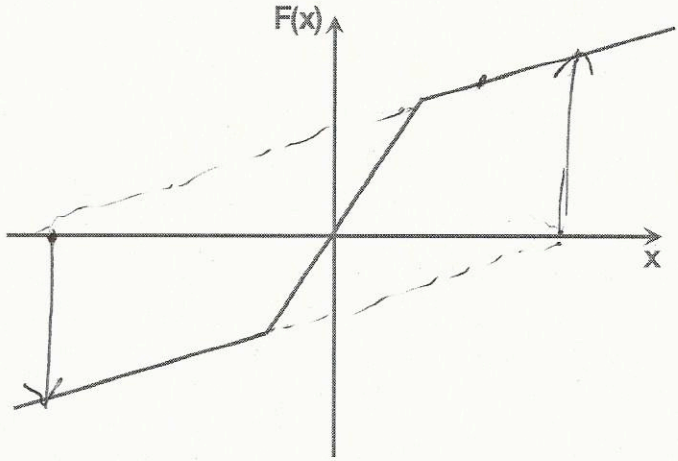
$$F\left[x^{(k+1)}\right] \approx F\left[x^{(k)}\right] + \left.\frac{\partial F}{\partial x}\right|_k \cdot \Delta x^{(k)} = 0$$

Vector matrix Vector

$$\Delta x^{(k)} = -\left[\left.\frac{dF}{dx}\right|_k\right]^{-1} \cdot F\left[x^{(k)}\right]$$

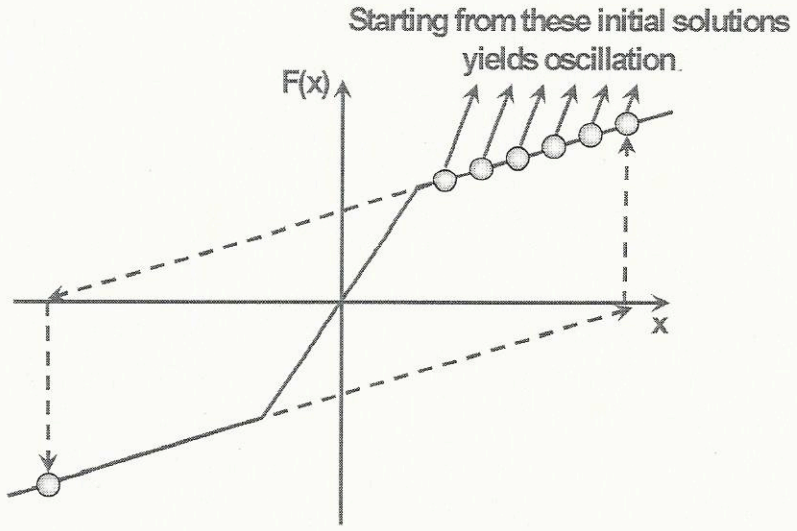
Newton-Raphson Method

- Newton-Raphson method is applicable, if and only if F(x) is smooth and its derivative exists
- A bad example: piecewise-linear function



Newton-Raphson Method

- If we use Newton-Raphson to solve the problem
  - 2nd order derivative is not continuous



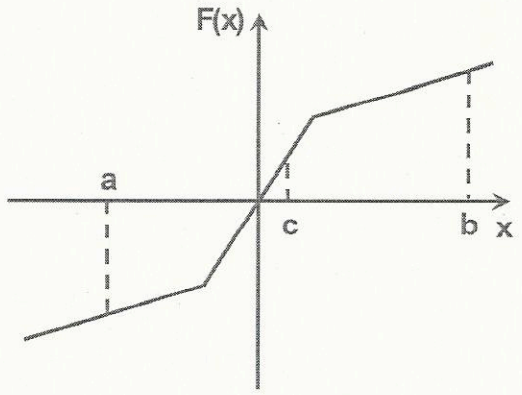
Binary Search

- Binary search is an alternative algorithm that is efficient for one-dimensional non-smooth but continuous function

- Key idea
  - Assume that the solution of F(x) = 0 is within an interval [a, b]
  - Iteratively shrink [a, b] to find the solution x

Binary Search

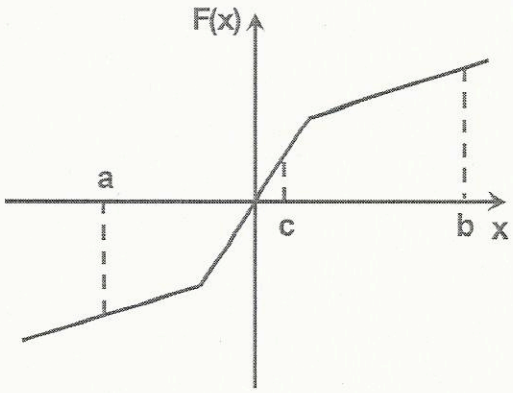
- Select initial interval [a, b] $F(a) \cdot F(b) < 0$
- Set c as the center of [a, b] $c = \frac{a+b}{2}$

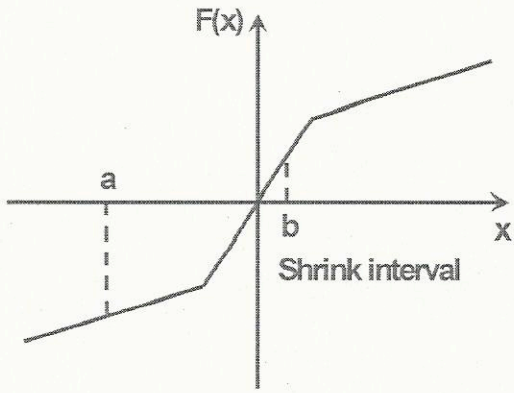


Binary Search

- If $F(a) \cdot F(c) < 0$
- Then $b = c$

Solution is within [a, c]

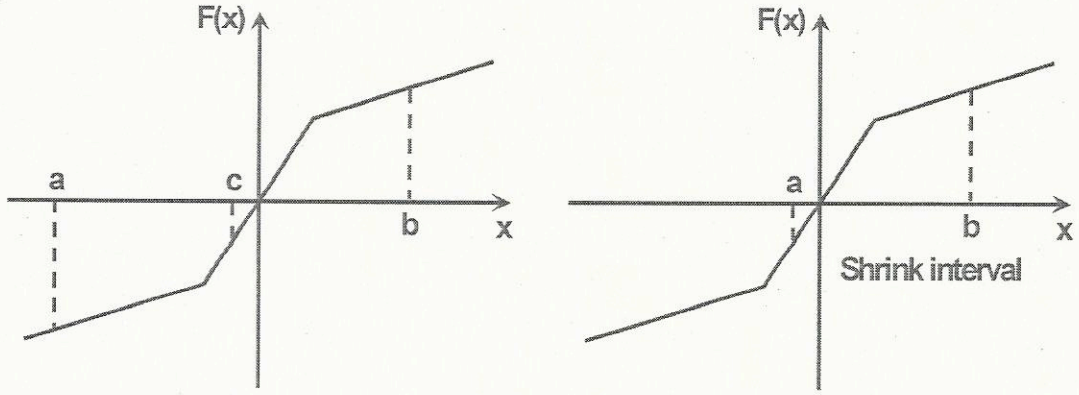




Binary Search

- If $F(b) \cdot F(c) < 0$
- Then $a = c$

Solution is within [c, b]



A Simple Example

$\log(x) = 0 \quad where \quad x \in [0,10]$

- Iteration #1

$a = 0$ $\quad \log(a) < 0$
$b = 10$ $\quad \log(b) > 0$
$c = 5$ $\quad \log(c) \geq 0$

$x \in [0, 5]$

- Iteration #2

$a = 0$ $\quad \log(a) < 0$
$b = 5$ $\quad \log(b) > 0$
$c = 2.5$ $\quad \log(c) > 0$

$x \in [0, 2.5]$

A Simple Example

$$\log(x) = 0 \quad where \quad x \in [0,10]$$

- **Iteration #3**

$a = 0$ $\quad \log(a) < 0$
$b = 2.5$ $\quad \log(b) > 0$
$c = 1.25$ $\quad \log(c) > 0$

$x \in [0, 1.25]$

- **Iteration #4**

$a = 0$ $\quad \log(a) < 0$
$b = 1.25$ $\quad \log(b) > 0$
$c = 0.625$ $\quad \log(c) < 0$

$x \in [0.625, 1.25]$

Continue till convergence is reached

Summary

- **Nonlinear equation solver**
  - Newton-Raphson method
  - Binary search