Introduction to Scientific Computing

Combinatorial Optimizations

Combinatorial Optimization

- **Examples**
  - ***Sudoku Puzzles*** (Constraint satisfaction problems or CSPs defined as a set of objects whose state must satisfy a number of constraints.)
  - Finding ***Shortest Path*** through a map (graph)
  - ***Travelling Salesman Problem*** (TSP): given a list of cities and their pairwise distances, the task is to find a shortest possible tour that visits each city exactly once.
  - ***Vehicle Routing Problem*** (VRP): seeking to service a number of customers with a fleet of vehicles. Often the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. Implicit is the goal of minimizing the cost of distributing the goods.

Sudoku Puzzles

- Goal to fill in all of the entries in a 9x9 table with 1-9 values subject to the constraint that every row, column and sub-square contains all of the digits 1-9.

L25-1

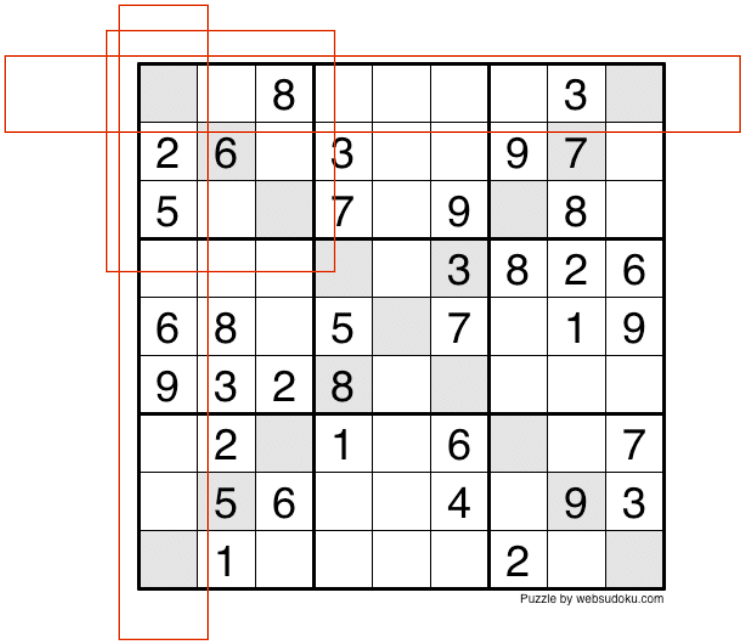
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Solving Sudoku Puzzles

- **A brute force *recursive* method involves finding the unmarked entries deciding what values they can take on and trying *each* possibility in turn**

sudoku.m, L25-2



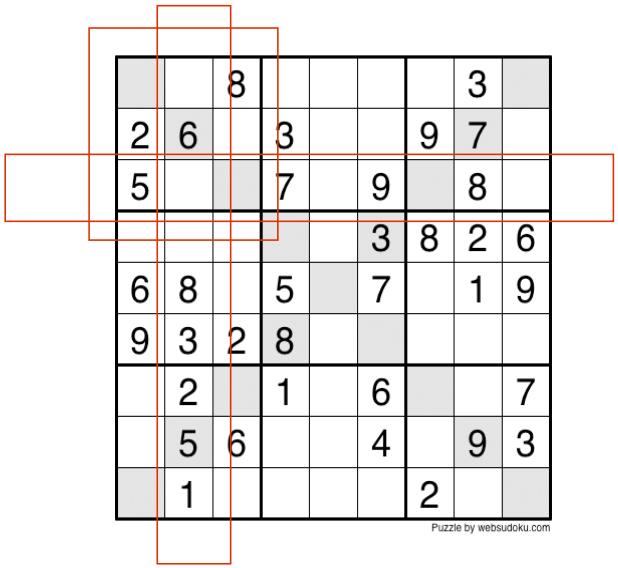
Useful functions

```
union(A,B);
setdiff(A,B);
```

Solving Sudoku Puzzles – a new version

- **A *recursive* method involves finding the unmarked entries deciding what values they can take on and trying *each* possibility in turn, however, starting with the *easiest* entries with the fewest possibilities.**

sudoku2.m, 25-3



Sorting

Sorting

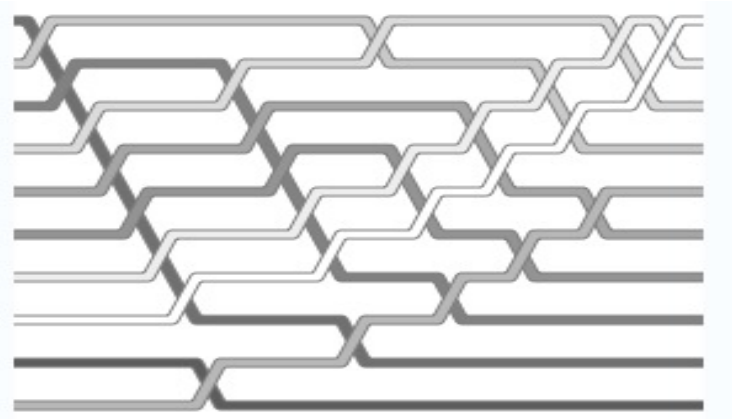
- **A sorting algorithm is an algorithm that puts elements of a list in certain order, for example numerical order.**

- **Efficient sorting is important to optimizing the use of other algorithms (such as search and merge algorithms).**

- **More formally, the output must satisfy two conditions:**
  - The output is in none-decreasing order (each element is no smaller than the previous element according to the desired total order);
  - The output is a permutation, or reordering, of the input.

Bubble Sort (text 11.3.1)

- Bubble sort is a simplistic method of sorting data that is used in computer science education. The algorithm starts at the beginning of the data set. It compares the first two elements, and if the first is greater than the second, it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass. The cost for bubble sort is $O(n^2)$.



Smaller numbers rise to the top of the list, like bubbles of air in water. (Largest one sinks to the bottom.)

Bubble Sort (text 11.3.1)

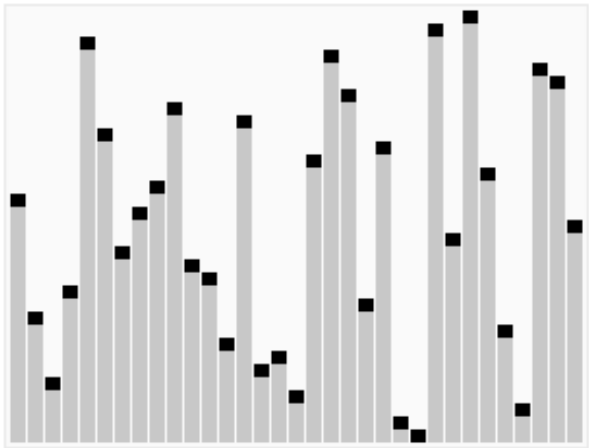
```
•n=length(x); sorted = 0;

•while ~sorted
•    sorted = 1;
•    for j=1:n
•        if x(j) > x(j+1)
•           x(j) ←→ x(j+1)
•           sorted = 0;
•        end
•    end
•end
```

Quicksort

- **Quicksort sorts by employing a divide and conquer strategy to divide a list into two sub-lists.**

1. Pick an element, called a *pivot*, from the list.
2. Reorder the list so that all elements which are less than the pivot come before the pivot and so that all elements greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.



- **If at each step we choose the median as the pivot then it works in O(n log n).**

MATLAB sort

- 1D

```
[y,I] = sort(x)
  x = input
  y = sorted array
  I = permutation index
```

- 2D

```
y = sort(x, dim, mode)
  dim: selects a dimension along which to sort.
mode: selects the direction of the sort
      'ascend' results in ascending order
     'descend' results in descending order
```

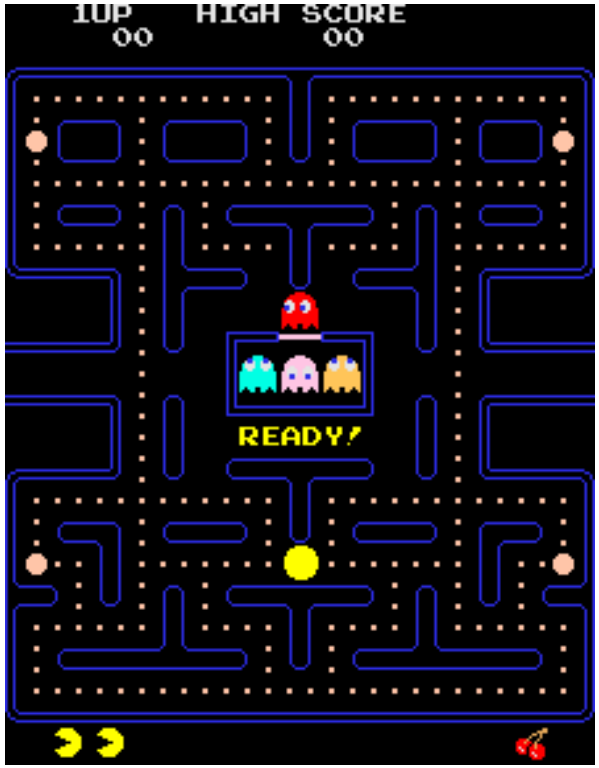
Shortest Path Problems

•Shortest Path problems show up in a number of guises:

- finding directions on google maps

- finding routes that avoid traffic

- guiding ghosts back home in Pac-Man



The Grassfire algorithm

```
function distance = grassfire (occupancy, dest_row, dest_col)
```

Create a distance array of the same size as the occupancy array
Initialize all of the entries in the distance array to infinity
Set the distance corresponding to the destination cell to zero

Loop

For each cell
Find the neighbor of this cell with the smallest distance value
Set my distance value equal to this minimum value + 1
(Remember that occupied cells in the grid have a distance ∞)
Break out of the loop when you go through a pass where none of the distance values change – remember that you can use the break function in Matlab to exit a loop

End loop

Matrix Computation

Linear Matrix Equation

- Consider the linear matrix equation

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

or

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Solution

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}$$

augmented matrix equation

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_n \end{array}\right]$$

Gaussian Elimination - example

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

(a) Forward Elimination

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

(b) Back Substitution

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 2 & 1 & 0 & 7 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ 0 & 0 & -1 & 1 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 2 & 0 & 0 & 4 \\ 0 & \frac{1}{2} & 0 & \frac{3}{2} \\ 0 & 0 & -1 & 1 \end{array}\right] \quad \begin{array}{r} x_1 = 2 \\ \Rightarrow \; x_2 = 3 \\ x_3 = -1 \end{array}$$

Gaussian Elimination with w/o Pivoting

$$\begin{bmatrix} 20 & 15 & 10 \\ -3 & -2.249 & 7 \\ 5 & 1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 45 \\ 1.751 \\ 9 \end{bmatrix} \quad \Rightarrow \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

(a) Forward Elimination

$$\left[\begin{array}{ccc|c} 20 & 15 & 10 & 45 \\ -3 & -2.249 & 7 & 1.751 \\ 5 & 1 & 3 & 9 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 20 & 15 & 10 & 45 \\ 0 & 0.001 & 8.5 & 8.501 \\ 0 & -2.75 & 0.5 & -2.25 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 20 & 15 & 10 & 45 \\ 0 & 0.001 & 8.5 & 8.501 \\ 0 & 0 & 23375.5 & 23375.45 \end{array}\right]$$

(b) Back Substitution

$$\left[\begin{array}{ccc|c} 20 & 15 & 10 & 45 \\ 0 & 0.001 & 8.5 & 8.501 \\ 0 & 0 & 23375.5 & 23375.45 \end{array}\right]$$

$$\begin{aligned} x_1 &= 0.9625 \\ \Rightarrow x_2 &= 1.05 \\ x_3 &= 0.999995 \end{aligned}$$

Gaussian Elimination with Pivoting

(a) Forward Elimination

$$\left[\begin{array}{cccc} 20 & 15 & 10 & 45 \\ -3 & -2.249 & 7 & 1.751 \\ 5 & 1 & 3 & 9 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 20 & 15 & 10 & 45 \\ 0 & 0.001 & 8.5 & 8.501 \\ 0 & -2.75 & 0.5 & -2.25 \end{array}\right] \rightarrow \left[\begin{array}{cccc} 20 & 15 & 10 & 45 \\ 0 & -2.75 & 0.5 & -2.25 \\ 0 & 0.001 & 8.5 & 8.501 \end{array}\right]$$

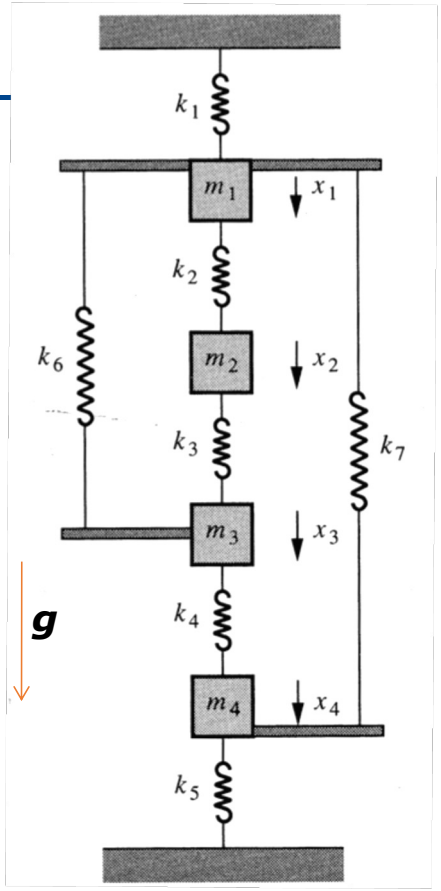
$$\rightarrow \left[\begin{array}{cccc} 20 & 15 & 10 & 45 \\ 0 & -2.75 & 0.5 & -2.25 \\ 0 & 0 & 8.50018 & 8.50018 \end{array}\right]$$

(b) Back Substitution

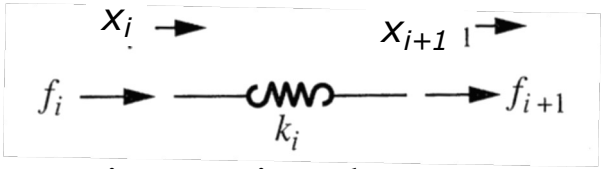
$$\left[\begin{array}{cccc} 20 & 15 & 10 & 45 \\ 0 & -2.75 & 0.5 & -2.25 \\ 0 & 0 & 8.50018 & 8.50018 \end{array}\right] \qquad \begin{aligned} x_1 &= 1 \\ \Rightarrow x_2 &= 1 \\ x_3 &= 1 \end{aligned}$$

Example Spring Mass System

- Consider a typical spring mass system, shown in the figure at the right. Gravity is pointing downward.
- Solve the displacements in each of the springs, x_1, x_2, x_3 and x_4.
- Take $k_1=k_3=k_6=k$, $k_2=k_4=k_5=2k$, and $k_7=3k$; all masses to be $m_1=m_2=m_3=m_4=m$.



Spring Mass System



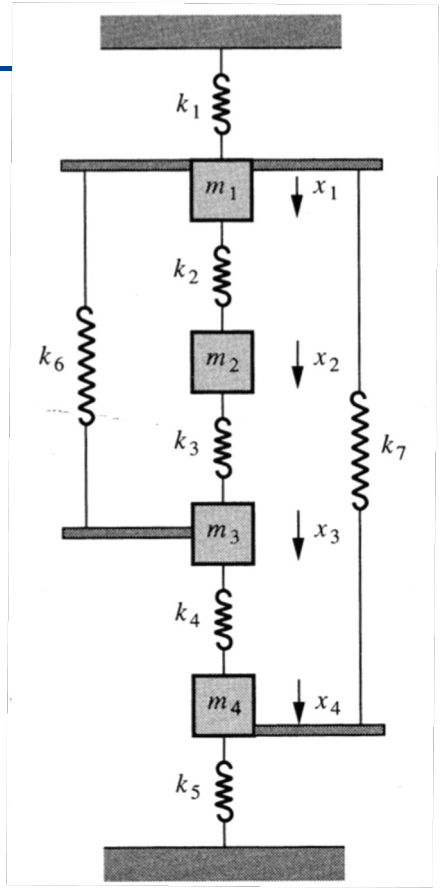
For a given spring element, at equilibrium the force and displacement are related by

$$f_i = k_i(x_i - x_{i+1})$$
$$f_{i+1} = k_i(x_{i+1} - x_i)$$

Force balance at each mass:

$$f_1 - f_2 - f_6 - f_7 - m_1 g = 0$$
$$f_2 - f_3 - m_2 g = 0$$
$$f_3 + f_6 - f_4 - m_3 g = 0$$
$$f_4 + f_7 - f_5 - m_4 g = 0$$

$$f_1 = k_1 x_1$$
$$f_2 = k_2(x_2 - x_1)$$
$$f_3 = k_3(x_3 - x_2)$$
$$f_4 = k_4(x_4 - x_3)$$
$$f_5 = -k_5 x_4$$
$$f_6 = k_6(x_3 - x_1)$$
$$f_7 = k_7(x_4 - x_1)$$



Spring Mass System

- Final equations:

$$k_1x_1 - k_2(x_2 - x_1) - k_6(x_3 - x_1) - k_7(x_4 - x_1) = m_1g$$
$$k_2(x_2 - x_1) - k_3(x_3 - x_2) = m_2g$$
$$k_3(x_3 - x_2) + k_6(x_3 - x_1) - k_4(x_4 - x_3) = m_3g$$
$$k_4(x_4 - x_3) + k_7(x_4 - x_1) + k_5x_4 = m_4g$$

$or,$

$$(k_1 + k_2 + k_6 + k_7)x_1 - k_2x_2 - k_6x_3 - k_7x_4 = m_1g$$
$$-k_2x_1 + (k_2 + k_3)x_2 - k_3x_3 = m_2g$$
$$-k_6x_1 - k_3x_2 + (k_3 + k_6 + k_4)x_3 - k_4x_4 = m_3g$$
$$-k_7x_1 - k_4x_3 + (k_4 + k_7 + k_5)x_4 = m_4g$$

Spring Mass System

- Matrix equation:

$$\begin{pmatrix} k_1+k_2+k_6+k_7 & -k_2 & -k_6 & -k_7 \\ -k_2 & k_2+k_3 & -k_3 & 0 \\ -k_6 & -k_3 & k_3+k_6+k_4 & -k_4 \\ -k_7 & 0 & -k_4 & k_4+k_7+k_5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = mg \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 7 & -2 & -1 & -3 \\ -2 & 3 & -1 & 0 \\ -1 & -1 & 4 & -2 \\ -3 & 0 & -2 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \left(\frac{mg}{k}\right) \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1.4702 \\ 1.8874 \\ 1.7219 \\ 1.2649 \end{pmatrix} \left(\frac{mg}{k}\right)$$

Example Flow in a Pipe

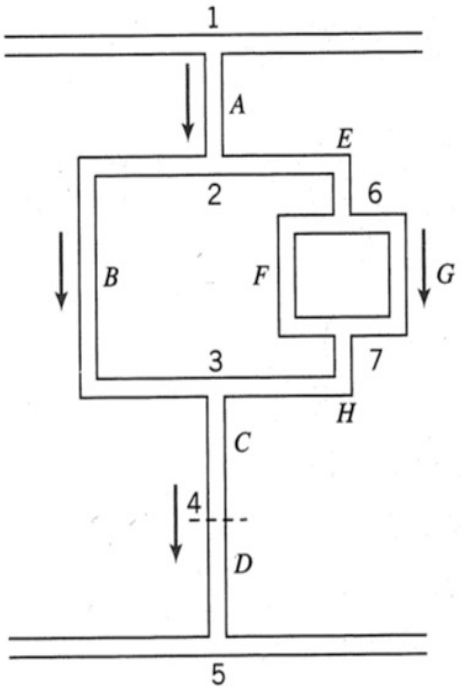
- In the section of a cast-iron pipe network shown at the right, the static pressure head available at point 1 is 100 ft of water, and point 5 is a drain. Find the flow rates in each pipe.
- Ignore gravity effects, ignore minor losses.
- Over each pipe section, pressure drop and mean flow velocity is related by

$$\Delta p = \rho f \frac{L}{D}\frac{V^2}{2}$$

where ρ is water density, and

$$f = \frac{64}{\mathrm{Re}} \quad \text{laminar } \mathrm{Re} < 3000 \qquad \mathrm{Re} = \frac{\rho V D}{\mu}$$

$$\frac{1}{\sqrt{f}} = -2.0\log\left(\frac{e/D}{3.7} + \frac{2.51}{\mathrm{Re}\sqrt{f}}\right) \quad \text{turbulent } \mathrm{Re} > 3000$$



A: L = 10 ft, D = 1.5 in.
B: L = 20 ft, D = 1.5 in.
C: L = 10 ft, D = 2 in.
D: L = 10 ft, D = 1.5 in.
E: L = 5 ft, D = 1.5 in.
F: L = 10 ft, D = 1 in.
G: L = 10 ft, D = 1.5 in.
H: L = 5 ft, D = 2 in.

Overconstrained Systems

Least Squares Method

Overconstrained (overdetermined) Systems

- A linear system of equations is overconstrained if there are more equations than unknowns, eg.

$$\begin{cases} x - y = 0 \\ y = 2 \\ x = 1 \end{cases} \quad \text{solved in Matlab} \Rightarrow \begin{matrix} x = 1.3333 \\ y = 1.6667 \end{matrix} \quad ?$$

- Consider the residual of the equations:

$$J = (x - y)^2 + (y - 2)^2 + (x - 1)^2$$

Least squares method

- Its minimization requires that

$$\begin{aligned} \partial J/\partial x &= 2(x - y) + 2(x - 1) = 0 \\ \partial J/\partial y &= -2(x - y) + 2(y - 2) = 0 \end{aligned} \Rightarrow \begin{matrix} 2x - y = 1 \\ -x + 2y = 2 \end{matrix} \Rightarrow \begin{matrix} x = 4/3 \\ y = 5/3 \end{matrix}$$

Least Squares Method

- For an overconstrained linear system

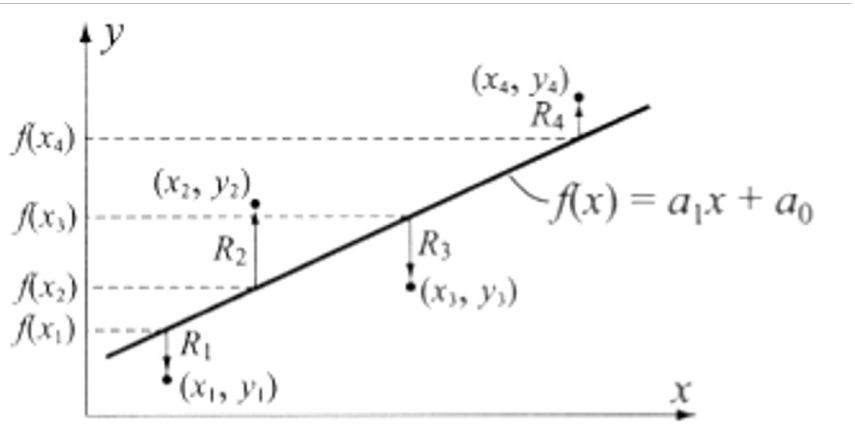
$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \qquad m>n$$

- Minimization of the residual of the equations,

$$R = \sum_{i=1}^{m} \left(a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n - b_i \right)^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right)^2$$

requires that

$$\frac{\partial R}{\partial x_k} = \sum_{i=1}^{m} 2a_{ik} \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right) = 0 \quad \rightarrow \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ik}a_{ij} \right) x_j = \sum_{i=1}^{m} a_{ik}b_i \quad \rightarrow \sum_{j=1}^{n} \bar{a}_{kj}x_j = \bar{b}_k$$



- Polynomial fit – `p = polyfit(x,y,n)`
- Square of residual
- Minimizing R requires $$R = \sum_{i=1}^{m}\left[f(x_i) - y_i\right]^2 = \sum_{i=1}^{m}\left[(a_1x_i + a_0) - y_i\right]^2$$

$$\frac{\partial R}{\partial a_1} = \sum_{i=1}^{m} 2x_i\left[(a_1x_i + a_0) - y_i\right] = 0 \qquad \frac{\partial R}{\partial a_0} = \sum_{i=1}^{m} 2\left[(a_1x_i + a_0) - y_i\right] = 0$$

- Least Squares $\dfrac{\partial R}{\partial a_1} = 0, \quad \dfrac{\partial R}{\partial a_0} = 0$

$$\sum_{i=1}^{m} x_i\left[\left(a_1 x_i + a_0\right) - y_i\right] = 0 \qquad \sum_{i=1}^{m}\left[\left(a_1 x_i + a_0\right) - y_i\right] = 0$$

$$\left(\sum_{i=1}^{m} x_i^{\,2}\right) a_1 + \left(\sum_{i=1}^{m} x_i\right) a_0 = \left(\sum_{i=1}^{m} x_i y_i\right) \qquad \left(\sum_{i=1}^{m} x_i\right) a_1 + (m) a_0 = \left(\sum_{i=1}^{m} y_i\right)$$

$$\begin{bmatrix} \sum_{i=1}^{m} x_i^{\,2} & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & m \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_i y_i \\ \sum_{i=1}^{m} y_i \end{bmatrix} \Rightarrow \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} x_i^{\,2} & \sum_{i=1}^{m} x_i \\ \sum_{i=1}^{m} x_i & m \end{bmatrix}^{-1} \begin{bmatrix} \sum_{i=1}^{m} x_i y_i \\ \sum_{i=1}^{m} y_i \end{bmatrix}$$

- Least Squares in Matlab (linear fit)

$$a_1 x_i + a_0 = y_i \quad i = 1,2,\ldots,m$$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \quad \Rightarrow \quad \mathbf{A} \cdot \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \mathbf{b}$$

$$\begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \mathbf{A} \setminus \mathbf{b}$$

Matlab

- Least Squares in Matlab (quadratic fit)

$$a_2 x_i^2 + a_1 x_i + a_0 = y_i \quad i = 1,2,\ldots,m$$

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m^2 & x_m & 1 \end{bmatrix} \cdot \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \quad \Rightarrow \quad \mathbf{A} \cdot \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \mathbf{b}$$

$$\begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \mathbf{A} \setminus \mathbf{b}$$

Matlab

- Least Squares in Matlab (other fit)

$$a_2(\sin x_i) + a_1(\cos x_i) = y_i \quad i = 1,2,\ldots,m$$

$$\begin{bmatrix} \sin x_1 & \cos x_1 \\ \sin x_2 & \cos x_2 \\ \vdots & \vdots \\ \sin x_m & \cos x_m \end{bmatrix} \cdot \begin{bmatrix} a_2 \\ a_1 \end{bmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \quad \Rightarrow \quad \mathbf{A} \cdot \begin{bmatrix} a_2 \\ a_1 \end{bmatrix} = \mathbf{b}$$

$$\begin{bmatrix} a_2 \\ a_1 \end{bmatrix} = \mathbf{A} \setminus \mathbf{b}$$

Matlab