Interpolation, extrapolation, etc.

Prof. Ramin Zabih

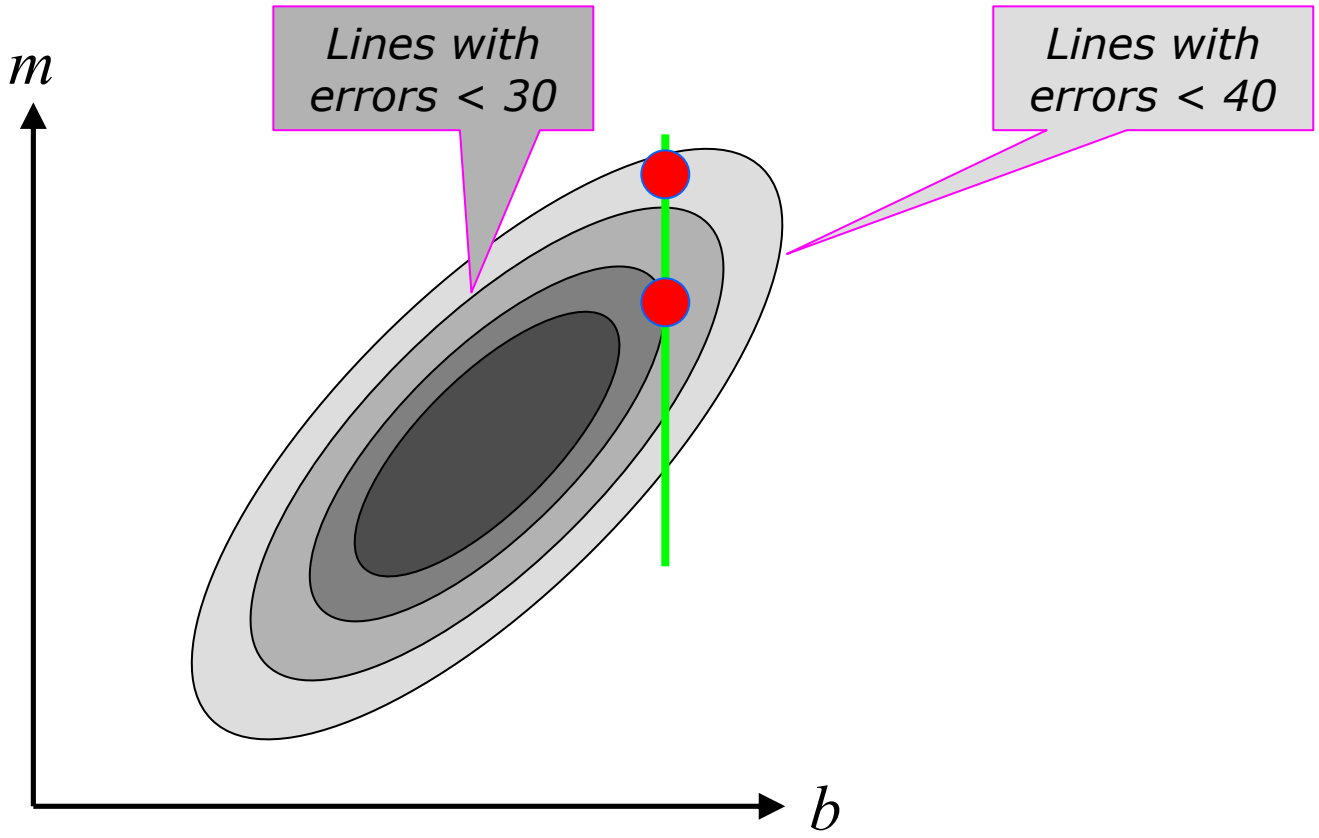
http://cs100r.cs.cornell.edu

Cornell University
Computer Science

Administrivia

- Assignment 5 is out, due Wednesday
- A6 is likely to involve dogs again
- Monday sections are now optional
  - Will be extended office hours/tutorial
- Prelim 2 is Thursday
  - Coverage through today
  - Gurmeet will do a review tomorrow

Hillclimbing is usually slow



Extrapolation

- Suppose you only know the values of the distance function f(1), f(2), f(3), f(4)
  - What is f(5)?
- This problem is called extrapolation
  - Figuring out a value outside the range where you have data
  - The process is quite prone to error
  - For the particular case of temporal data, extrapolation is called prediction
  - If you have a good model, this can work well

Interpolation

- Suppose you only know the values of the distance function f(1), f(2), f(3), f(4)
  - What is f(2.5)?
- This problem is called interpolation
  - Figuring out a value inside the range where you have data
    - But at a point where you don't have data
  - Less error-prone than extrapolation
- Still requires some kind of model
  - Otherwise, crazy things could happen between the data points that you know

Analyzing noisy temporal data

- Suppose that your data is temporal
  - Radar tracking of airplanes, or DJIA
- There are 3 things you might wish to do
  - Prediction: given data through time T, figure out what will happen at time T+1
  - Estimation: given data through time T, figure out what actually happened at time T
    - Sometimes called filtering
  - Smoothing: given data through time T, figure out what actually happened at time T-1
    - Or even earlier

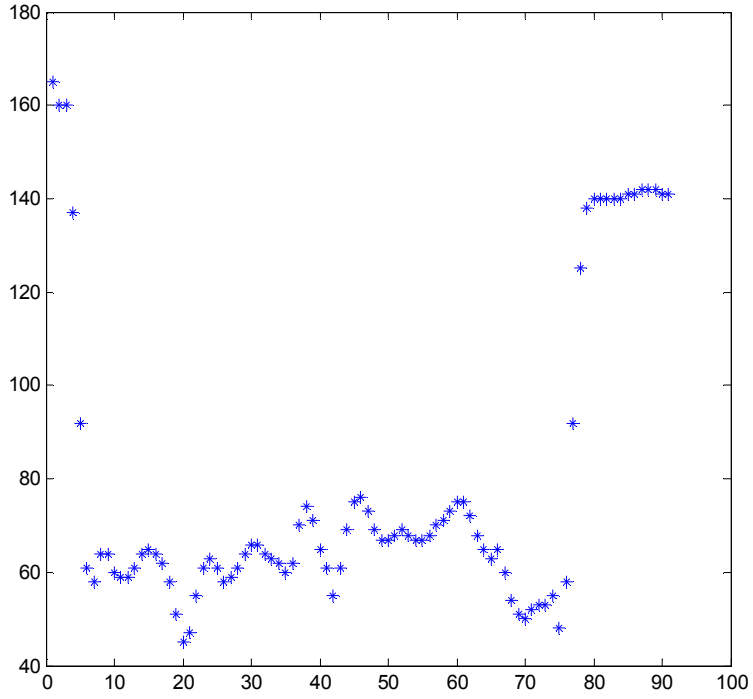
Image interpolation

- Suppose we have a picture and want to make a bigger one
  - I.e., higher resolution
- Lots of applications (if it could be done…)
  - Example: take a low-resolution picture on your cellphone, then create a high-resolution version to display on your computer monitor
- Note that image extrapolation is clearly much harder
  - What is outside the picture? Who knows??

Scanline interpolation

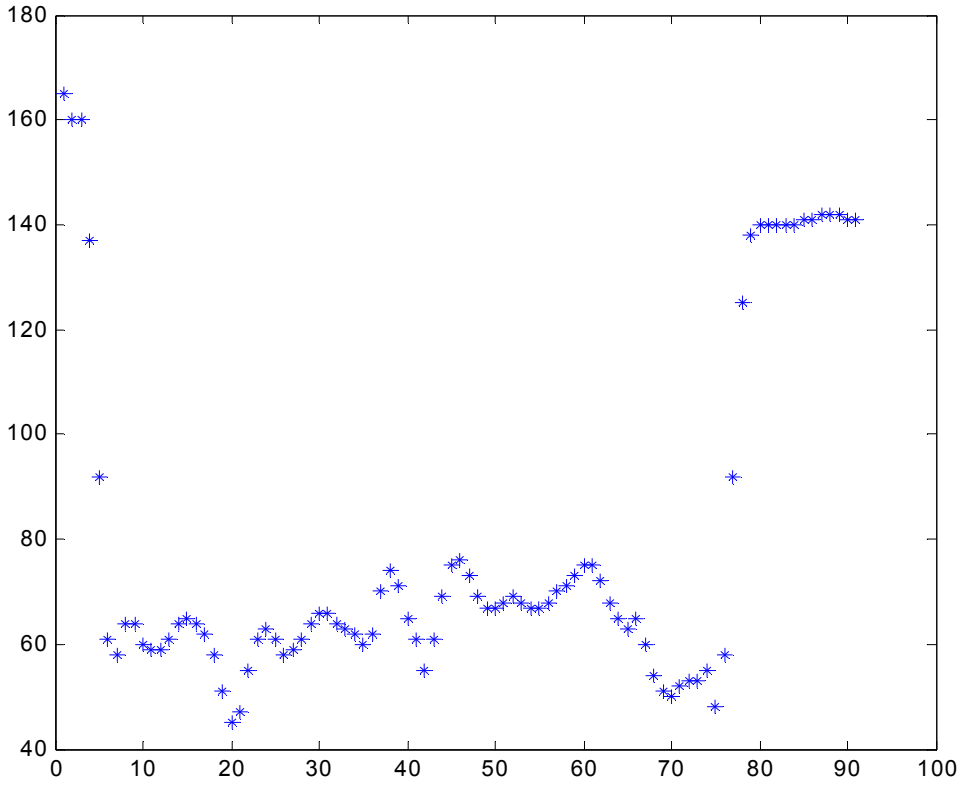
- Let's just consider a single row
  - In an image, this is called a scanline
    - Terminology comes from CRT's
- Suppose we have 100 data points
  - x values are 1, 2, 3, … , 100
  - We know f(1), f(2), … , f(100)
- How do we get another 99 data points?
  - x values of 1, 1.5, 2, 2.5, …, 99.5, 100
  - We need f(1.5), f(2.5), … , f(99.5)

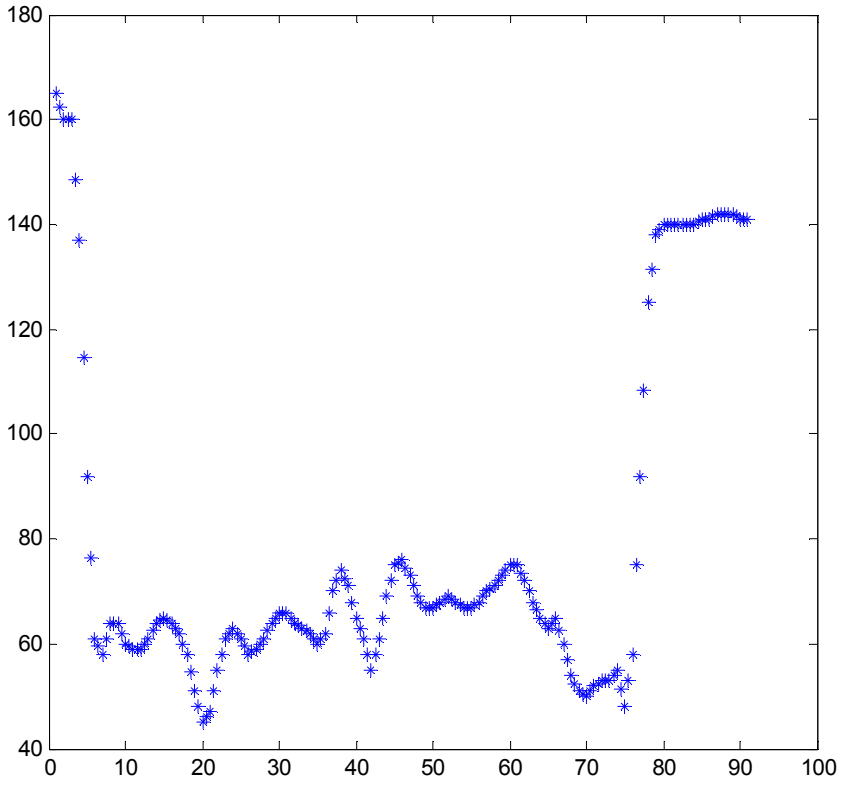
Motivating example



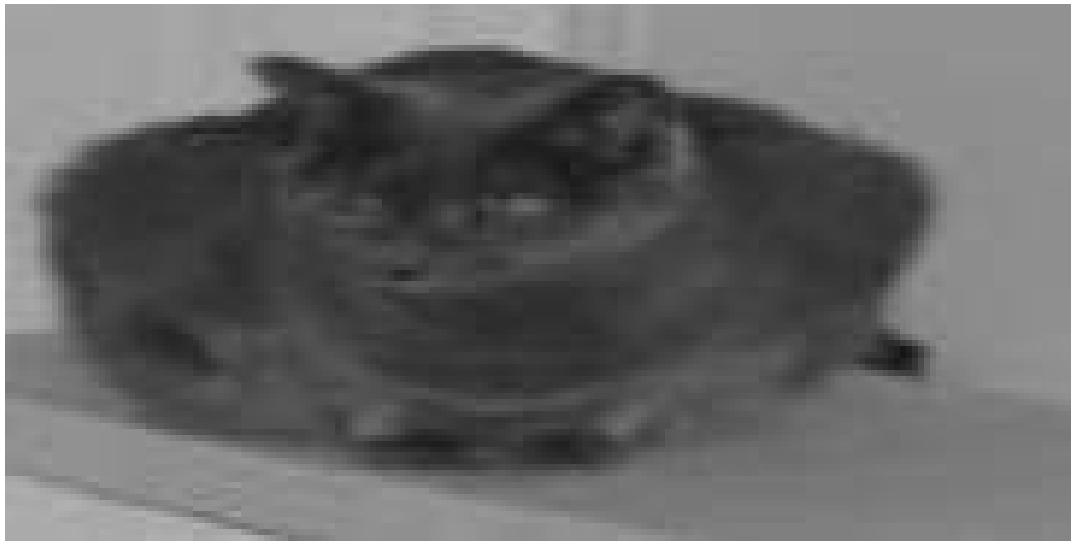
Obvious solution

- f(2.5) is the average of f(2) and f(3)





Results



Linear interpolation

- Suppose we want to interpolate f(2.2)?
  - As before, depend on f(2) and f(3)
  - If f(2) = 100, and f(3) = 200, f(2.2) = 120
- With some algebra we can generalize this
  - Be sure to always check the boundary cases!

$$x' \in [x_l, x_r] \quad \Longrightarrow$$

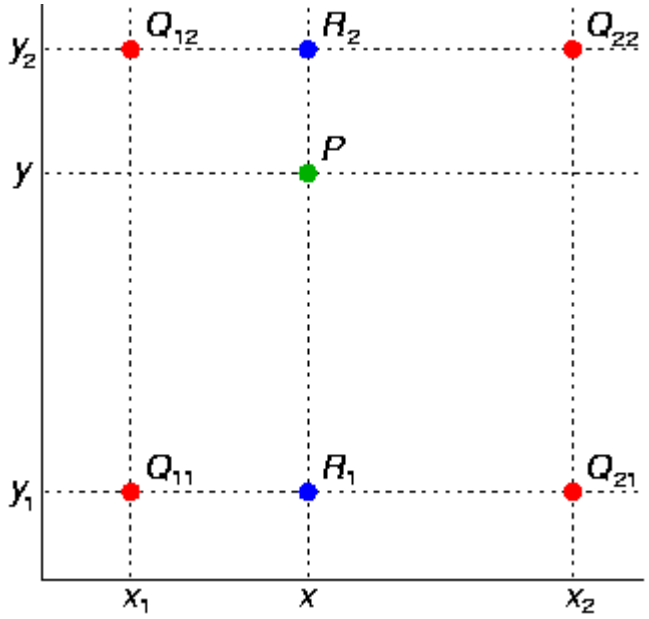
$$f(x') = \quad f(x_l) + (f(x_r) - f(x_l))\frac{x' - x_l}{x_r - x_l}$$

Nearest neighbor interpolation

- There is an even simpler technique
  - Which will be useful when we look at color object recognition (real soon now…)
- We can simply figure out which known value we are closest to, and use that
  - f(2.2) = f(2), f(2.9) = f(3), f(2.6) = f(3), etc.
- This is a fast way to get a bad answer

Bilinear interpolation

- What about in 2D?
  - Interpolate in x, then in y
- Example
  - We know the red values
  - Linear interpolation between red values gives us the blue values
  - Linear interpolation between the blue values gives us the answer
- This is order-independent



Limits of interpolation

- Can you prove that it is impossible to interpolate soundly?
  - An algorithm is **sound** if it always gives the right answer
    - This is widely considered a desirable property
- Suppose I claim to have a sound way to produce an image with 4x as many pixels
  - Sound, in this context, means that it gives what a better camera would have captured
  - Can you *prove* this cannot work?
- Related to impossibility of compression

Example algorithm that can't exist

- Consider a compression algorithm, like zip
  - Take a file F, produce a smaller version F'
  - Given F', we can uncompress to recover F
  - This is lossless compression, because we can "invert" it
    - MP3, JPEG, MPEG, etc. are not lossless
- Claim: there is no such algorithm that always produces a **smaller** file F'

Proof of claim

- Pick a file F, produce F’ by compression
  - F’ is smaller than F, by assumption
- Now run compression on F’
  - Get an even smaller file, F’’
- At the end, you’ve got a file with only a single byte (a number from 0 to 255)
  - Yet by repeatedly uncompressing this you can eventually get F
- However, there are more than 256 different files F that you could start with!