

## COMPUTER SCIENCE 51

Spring 2009  
cs51.seas.harvard.edu

Prof. Greg Morrisett  
Prof. Ramin Zabih

computer  
science 51



## Today's topics

- Streams and impossible programs
- Fitting 2D points with line of best fit
- Interpolating images
  - A different impossible program



## Impossible streams

- Simple questions about streams are actually impossible for a computer
  - Example: count how often a number appears in a stream
    - This is easy in a list!

```
; Gap between successive stream elements
(define (gap s)
  (lcons (- (lcar (lcdr s)) (lcar s))
        (gap (lcdr s))))
(define prime-gap (gap primes))
; Fields medal for implementing counts function
(define twin-primes-false-proof (counts 2 prime-gap))
```

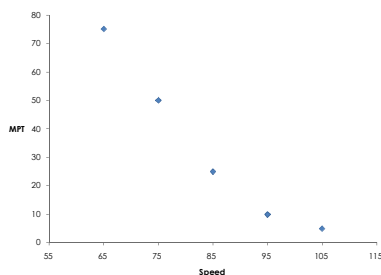


## Computation and data

- For the next few lectures we are going to look at the CS of “real-world” data
  - Anything actually measured
    - Physics, economics, chemistry, government...
- How can we use computers to make sense of data?
  - Especially for large data sets
- Real data always has errors in it
  - So we need error-tolerant algorithms!



## Example data

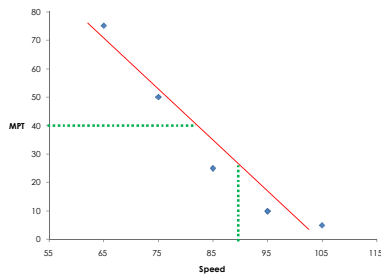


## Understanding data

- What is the relationship between two quantities?
  - (Model) fitting
  - For 2D data, we'll look at lines
- Later: find structure in the data
- Fitting often used for interpolation
  - What happened between data points?
  - Extrapolation: before/after data points
    - Generally harder (includes prediction)



## Interpolation example



– Note how hard extrapolation would be!

## Fitting via optimization

- Key technique: optimization
  - Think about Scheme's argmin function!
    - And, is there a way to make this lazy??
  - $\text{argmin}(F,L)$ : the element  $x$  in the list  $L$  such that  $(F x)$  is smallest

```
(define (argmin f l)
  (letrec ([choose (lambda (x y)
                    (if (< (f x) (f y)) x y))]
    [loop (lambda (cur r)
            (if (empty? r)
                cur
                (loop (choose cur (car r))
                    (cdr r)))]))
    (loop (car l) (cdr l))))
```

## Using argmin for fitting

- Define a measure of line badness
  - Use argmin to find best line (least bad)
- Some subtleties:
  - Lines are defined by 2 numbers. But this makes the code harder to read.
  - There are infinitely many lines!
- We could make argmin “lazy” by using streams instead of lists
  - But in general this doesn't help!

## Smarter argmin?

- General case: nothing smart to do
  - Have to run  $F$  on every element of  $L$ 
    - Exhaustive search!
  - Any way to end early?
- Suppose the elements of  $(\text{map } F L)$  decrease and then increase
  - Sort of like a parabola
    - $(\text{argmin id } (\text{list } 5 \ 3 \ 2 \ 1 \ 4 \ 6 \ 10 \ 42))$
- Can we use this to be smarter?

## Use the “dip”

- Can stop when we see the increase

```
(define (argmin-dip f l)
  (letrec ([choose (lambda (x y)
                    (if (< (f x) (f y)) x y))]
    [loop (lambda (cur r)
            (cond [(empty? r) cur]
                  ; Found increase, so done
                  [(> (f (car r)) (f cur)) cur]
                  [else
                   (loop (choose cur (car r))
                       (cdr r))]])
    (loop (car l) (cdr l))))
```

## Can even be lazy

- Use infinite streams
  - Force only what you need

```
(define (argmin-dips f s)
  (letrec ([choose (lambda (x y)
                    (if (< (f x) (f y)) x y))]
    [loop (lambda (cur r)
            (cond [(empty? r) cur]
                  [(> (f (lcar r)) (f cur)) cur]
                  [else
                   (loop (choose cur (lcar r))
                       (lcdr r))]])
    (loop (lcar s) (lcdr s))))
```

### Another way to write this

- Make a small change
  - As long as it helps reduce f

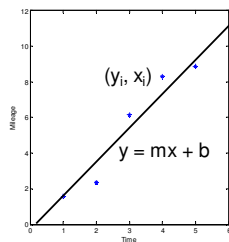
```
(define (minimize f delta)
  (letrec ([cur (f 1.0)] ; got to start somewhere)
    [loop (lambda (cur)
            (cond [(< (f (+ cur delta)) (f cur))
                  (loop (+ cur delta))]
                  [(< (f (- cur delta)) (f cur))
                  (loop (- cur delta))]
                  [else cur]))])
    (loop cur)))
```

### Stream of approximations

- Capture the history of the computation as a stream

```
(define (minimizes f delta)
  (letrec ([cur (f 1.0)]
          [loop (lambda (cur)
                  (cond [(< (f (+ cur delta)) (f cur))
                        (cons cur (loop (+ cur delta)))]
                        [(< (f (- cur delta)) (f cur))
                        (cons cur (loop (- cur delta)))]
                        [else (cons cur empty)])])])
    (loop cur)))
```

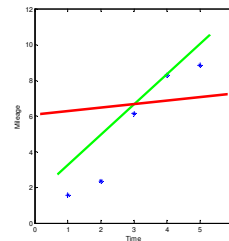
### Linear fitting



- Best line won't necessarily pass through any data point
- This is a bad (but traditional) way to represent a line

### Error for a model?

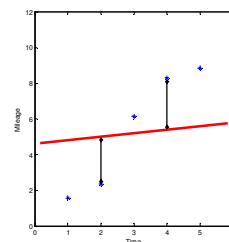
- What makes a line good versus bad?
  - This is actually a very subtle question



### Residual errors

- The difference between what the model predicts and what we observe is called a residual error
  - Consider the data point (x,y)
  - The model m,b predicts (x,mx+b)
  - The residual is  $y - (mx + b)$
- 1D residuals can be easily visualized
  - Vertical distance to the line

### LS fitting and residuals



$$Err(m, b) = \sum_i [y_i - (mx_i + b)]^2$$

## Greedy algorithm

- To find the LS squares line:
- Start with some initial  $(m,b)$
- Make a small change to  $(m,b)$ 
  - See if  $\text{Err}(m,b)$  goes down
  - If so, make that your new  $(m,b)$
- Done when small changes don't reduce  $\text{Err}(m,b)$

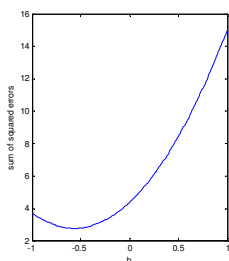


## Why least squares?

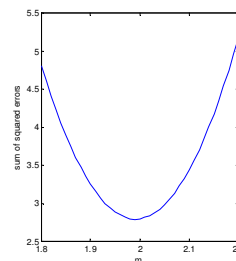
- There are lots of sensible ways to compute goodness of fit
- Why do we always use least squares?
- This is a deep topic
  - But we will point out two things that are special about least squares



## Fix $m=2$ , change $b$

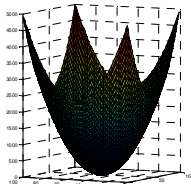


## Fix $b=-0.5$ , change $m$



## Some error functions are easy

- You will notice that the squared error has a **single** minimum value
  - The plots I just showed you demonstrate this is true in 1D
    - But it's true in 2D also



## Consequences

- For error functions like this, if we get an answer where a small change doesn't improve it, we know we are right!
  - Some issue with what "small" means
- Our LS-hillclimbing method will always converge to the right answer
  - By slowly rolling downhill
  - Hard to analyze how long it will take

