Cargo-Cult Complexity Testing

Tran Ma - Ambiata

THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN

INTRODUCTION TO ALGORITHMS

THIRD EDITION

The master theorem

The master method depends on the following theorem.

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

The master theorem

The master method depends on the following theorem.

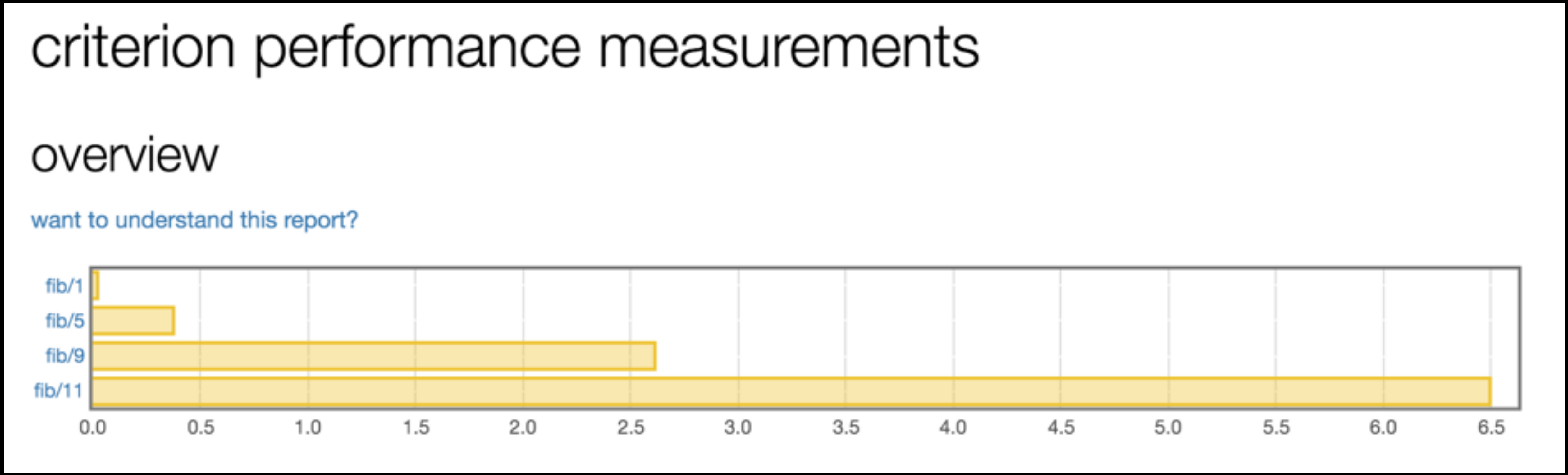
Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

Just Run It And See



Generate Inputs

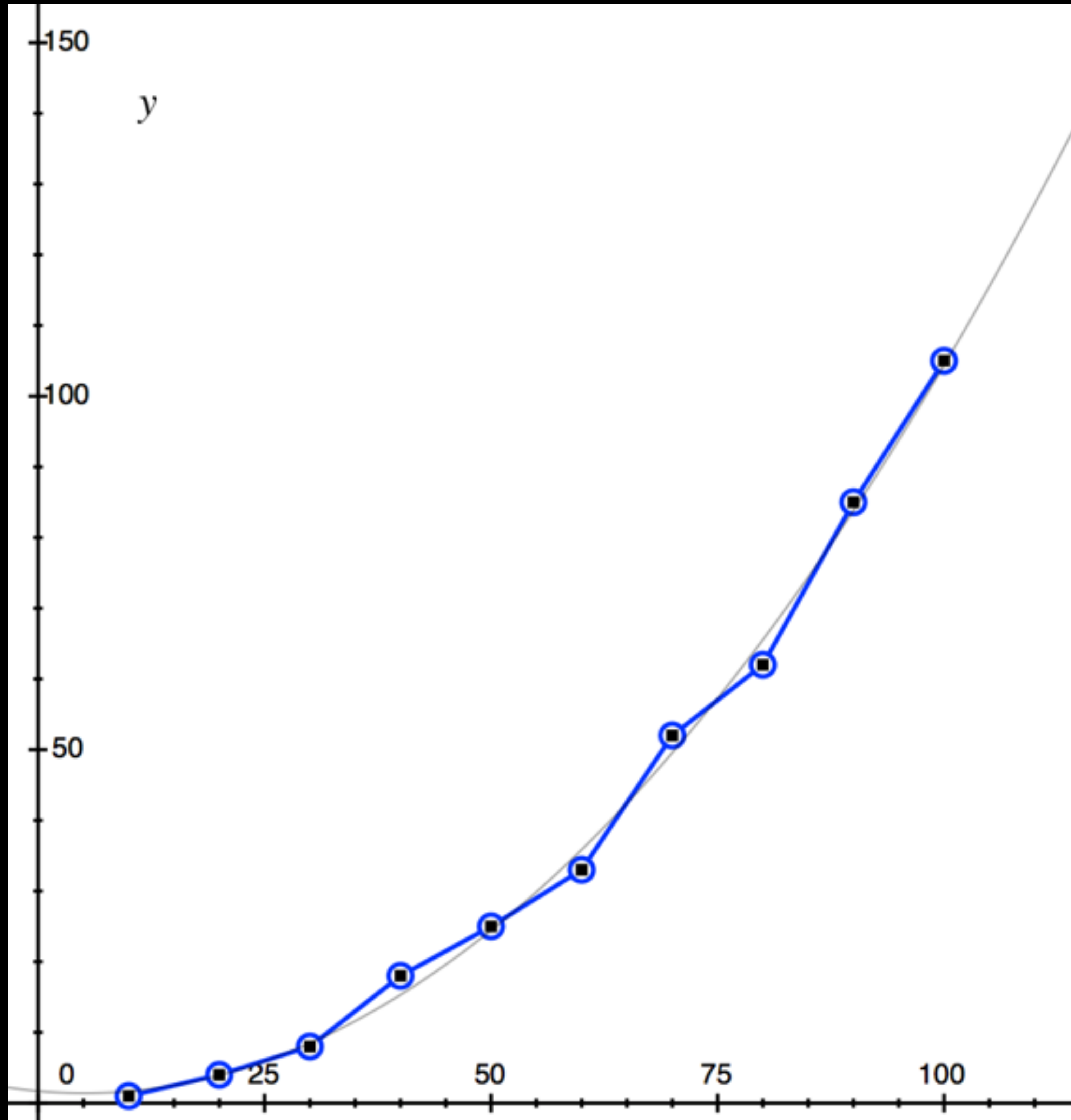
Integer input size n:

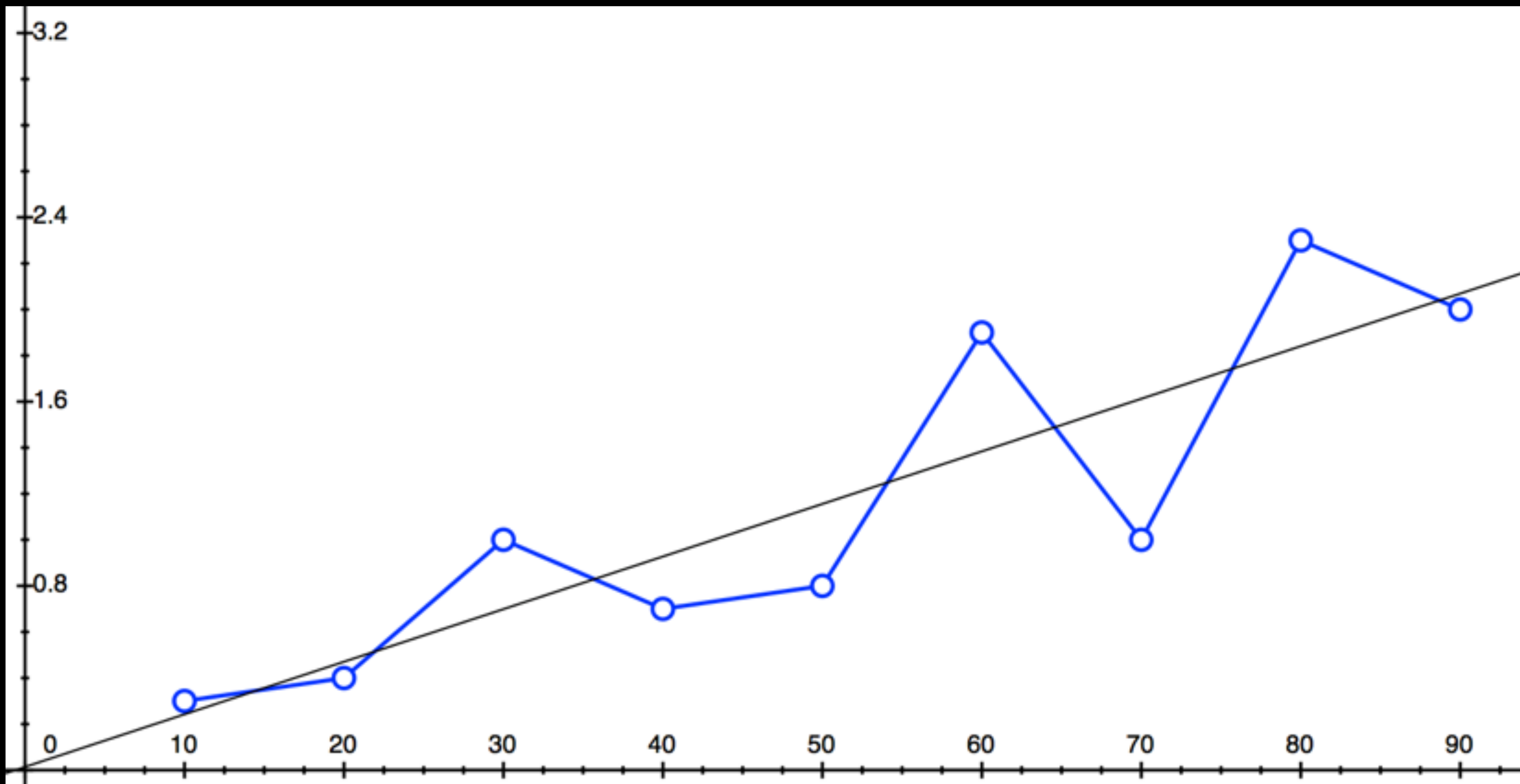
```
:: Gen Int
return n
```

List input size n:

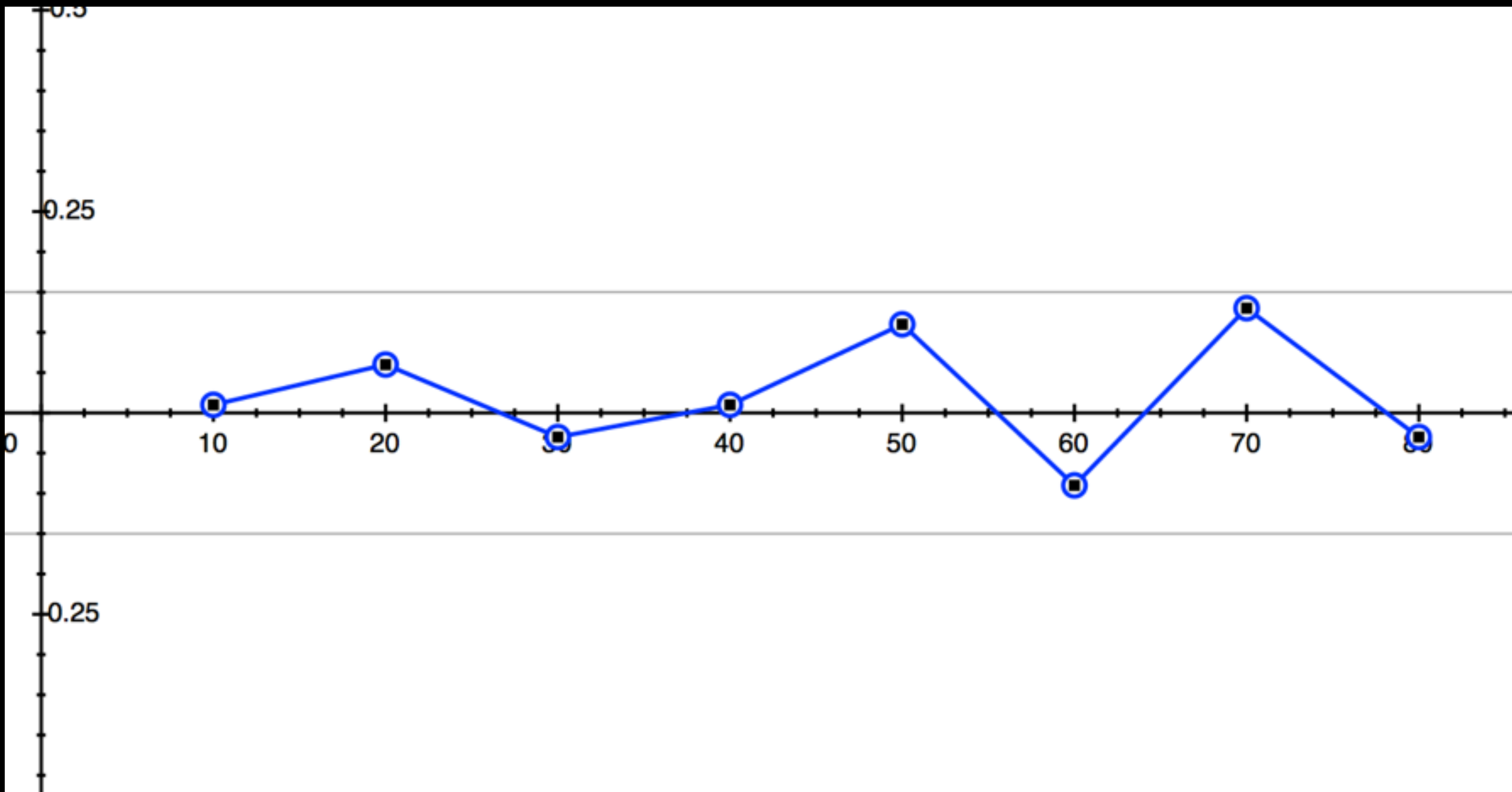
```
:: Arbitrary a => Gen [a]
take n <$> arbitrary
```

Demo 1
(generate points)


150
y
100
50
0
25
50
75
100



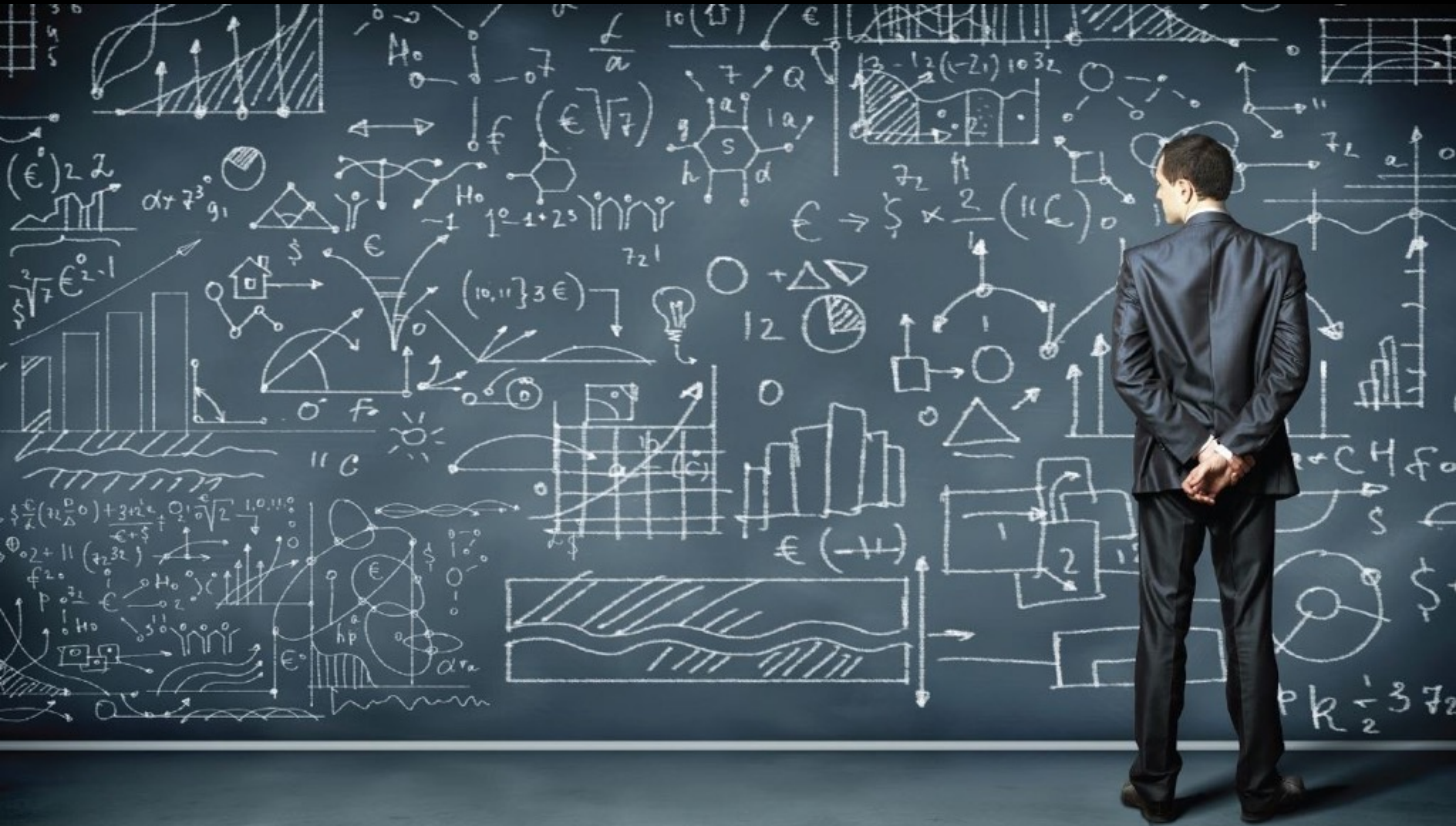
3.2
2.4
1.6
0.8
0
10
20
30
40
50
60
70
80
90


0.25
-0.25
0
10
20
40
50
60
70

Find a polynomial order that fits best

```haskell
-- | Estimate the polynomial order for some points, given
--   a margin of error.
--
polyOrder :: Double -> [Point] -> Maybe Order
polyOrder epsilon points@(_:_:_:_)
  | isConstant points
  = Just 0
  | otherwise
  = fmap succ $ polyOrder epsilon $ deriv points
  where
   isConstant derivs
    = sd (fmap snd derivs) < epsilon
polyOrder _ _
  = Nothing
```

Demo 2
(naive)



Use curve-fitting to try and fit every type of curve

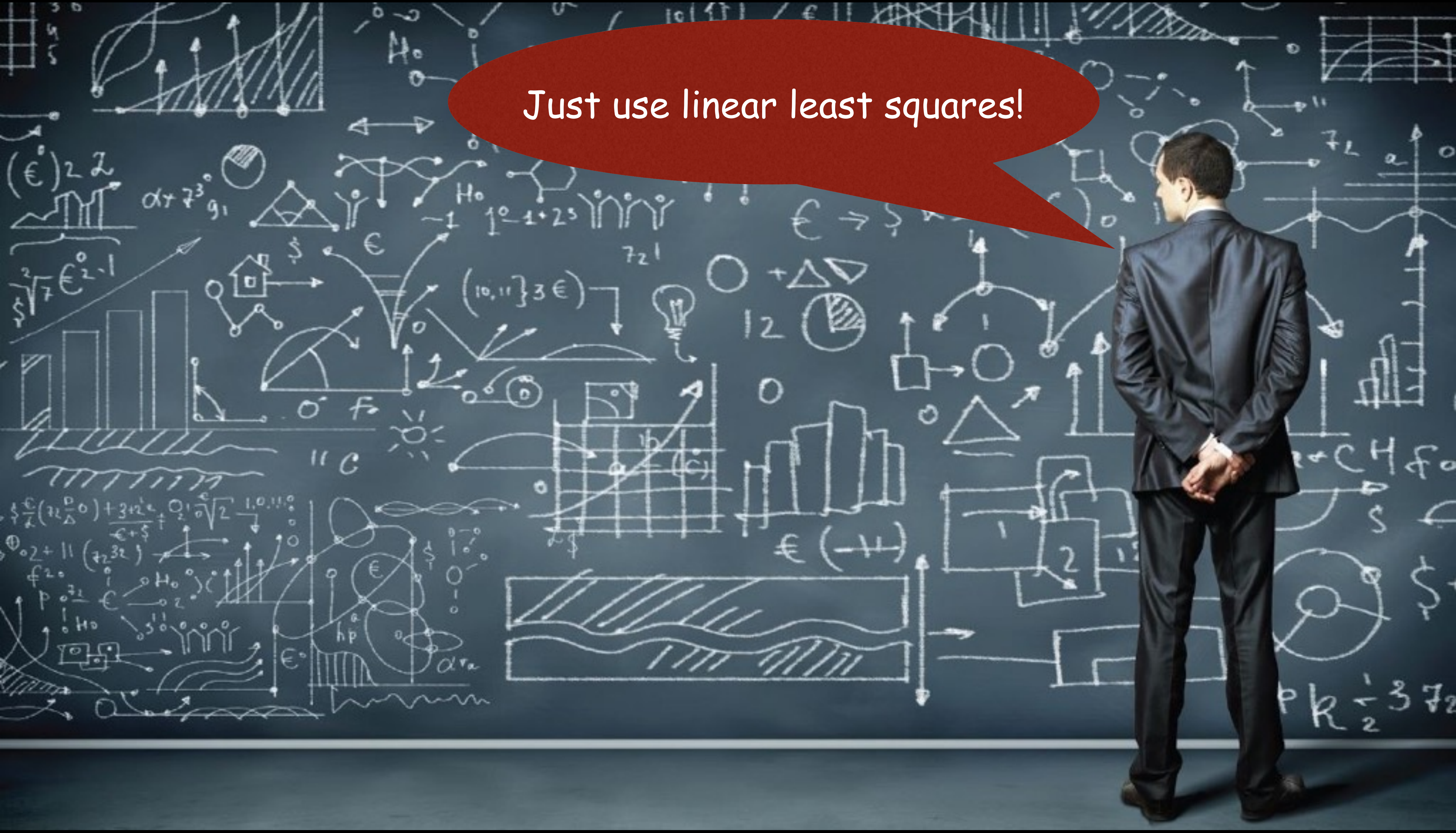
Use the R-squared statistic to determine their goodness of fit!

Curve Fitting
with nls()?

R's non-linear least squares method for curve-fitting

Curve Fitting with nls()?

"If there was a strategy that was both good and general -- one that always worked - it would already be implemented in every nonlinear least squares program and starting values would be a non-issue."


Just use linear least squares!

lm()

```
> summary(lm(y ~ I(x^2)+I(x)))

Call:
lm(formula = y ~ I(x^2) + I(x))

Residuals:
       Min         1Q     Median         3Q        Max 
-2.627e-04 -1.723e-04 -1.097e-05  1.229e-04  5.871e-04 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.641e-04  1.669e-04   2.182  0.04439 *  
I(x^2)       1.073e-05  1.867e-06   5.747    3e-05 ***
I(x)        -1.497e-04  3.843e-05  -3.895  0.00129 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0002174 on 16 degrees of freedom

Multiple R-squared:  0.8397,  Adjusted R-squared:  0.8197

F-statistic: 41.91 on 2 and 16 DF,  p-value: 4.361e-07
```

HaskellR

```
[r|
  summary(lm(y ~ I(x^2) + I(x)))$adj.r.squared
|]
```

HaskellR

```haskell
lm' :: Order -> [Double] -> [Double] -> R s (R.SomeSEXP s)
lm' order x y
  = do [r| x = x_hs |]
       [r| y = y_hs |]
       case order of
         Constant    -> [r|summary(lm(y ~ 1))$adj.r.squared|]
         LogN        -> [r|summary(lm(y ~ I(log(x))))$adj.r.squared|]
         NLogN       -> [r|summary(lm(y ~ I(x * log(x))))$adj.r.squared|]
         Linear      -> [r|summary(lm(y ~ I(x)))$adj.r.squared|]
         Quadratic   -> [r|summary(lm(y ~ I(x^2) + I(x)))$adj.r.squared|]
         Cubic       -> [r|summary(lm(y ~ I(x^3) + I(x^2) + I(x)))$adj.r.squared|]
         Quartic     -> [r|summary(lm(y ~ I(x^4) + I(x^3) + I(x^2) + I(x)))$adj.r.squared|]
         Exp         -> [r|summary(lm(log(y) ~ x))$adj.r.squared|]
```

Demo 3
(the sorts)

Demo 4
(fibonacci)

Demo 5
(codensity!?)

“Honourable” mention

計

る

Hakaru

```haskell
-- | Given a complexity order and some data, generate a curve of
--   that order that fit the data.
--
curveFit :: Order -> Fit -> IO (Double -> Double)
curveFit thing (Fit xs ys dropn taken)
 = do l <- mcmc (measureForOrder thing xs ys)
                (map (Just . toDyn . Lebesgue) ys)
      let means = expectations $ take taken $ drop dropn l
      return $ mkCurve thing means
```

/end