Smoothing - part 3

- Penalized splines is not the only way to estimate $f(x)$ when $y = f(x) + \epsilon$
- Two others are kernel smoothing and the Lowess (Loess) smoother
- I'll only talk about Lowess
- The Lowess algorithm does the same thing as fitting a penalized spline
- Lowess is more ad-hoc. Only practical difference I've found is that Lowess includes an outlier detection/downweighting step
- Lowess is LOcally WEighted polynomial regreSSion
- Original paper is Cleveland, W.S. (1979) Robust locally weighted regression and smoothing scatterplots. JASA 74:829-836

Lowess concepts

- any function, $f(x)$, is closely approximated by $\beta_0 + \beta_1 x$ over a small enough range of x
- so pick a point, x, may or may not be an x in the data
- fit a linear regression to the subset of the data with x values near x
- predict $f(x)$ from $\hat{\beta}_0 + \hat{\beta}_1 x$
- repeat for as many x's as needed to estimate $f(x)$
- A couple of refinements:
    - Use weighted regression with weights dependent on distance from x
    - "Robustify" the fit by downweighting the influence of regression outliers (far vertically from $\hat{f}(x)$)

- A simple algorithm that doesn't work well:
- Consider all points within specified distance d of x
- Fit OLS regression to these points
- Imagine fitting the curve to 5 points around x
- Then shifting to $x + 0.01$ that now includes 6 points
- that 6th point is within d of $x + 0.01$ but not within d of x
- Result is a very "jumpy" fitted curve
  because that 6'th point has a very large influence on the fitted value
- Lowess incorporates weighting so points $\approx d$ from x have some influence, but not much.
- Influence increases as x gets close to a data point
- Also hard to specify an appropriate d: depends on range of x values

Lowess details

- What is "close to *x*"?
  - Define close in terms of a fraction of the data points
  - f = smoothing parameter, $f \geq 0$.
  - For $0 \leq f \geq 1$, f is a fraction of data points.
  - For demonstration, consider small $f = 0.2$
  - For observation i, compute the n values: $d_{ij} = \mid x_i - x_j \mid \ \forall j$
  - The observations with the $f \times n$ smallest d_{ij} are "close" to d_{ij}.
    (Pictures on next two slides)
  - "close" means close if points dense in local neighborhood
    (e.g. $x_{43} = 10.8$)
    not so close if points sparse in local neighborhood ($x_{11} = 2.2$)
  - define h_i as the f quantile of d_{ij} for specified i
    $h_{11} = 2.94$, $h_{43} = 1.10$

Points close (f=0.2) to obs at age = 2.2

Age

log serum C-peptide conc.

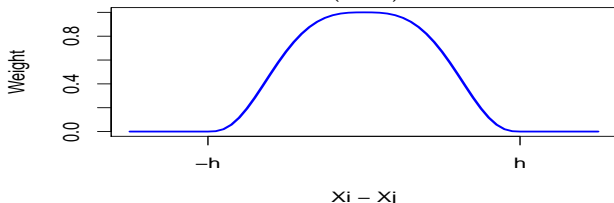**Points close (f=0.2) to obs at age = 10.8**

Age

log serum C–peptide conc.
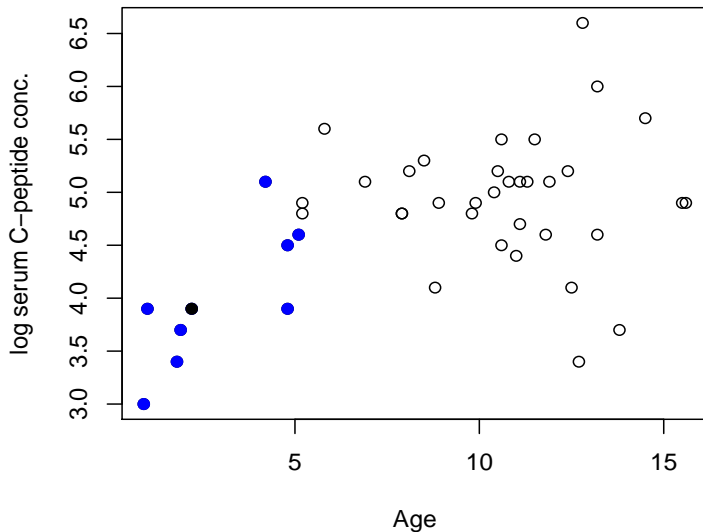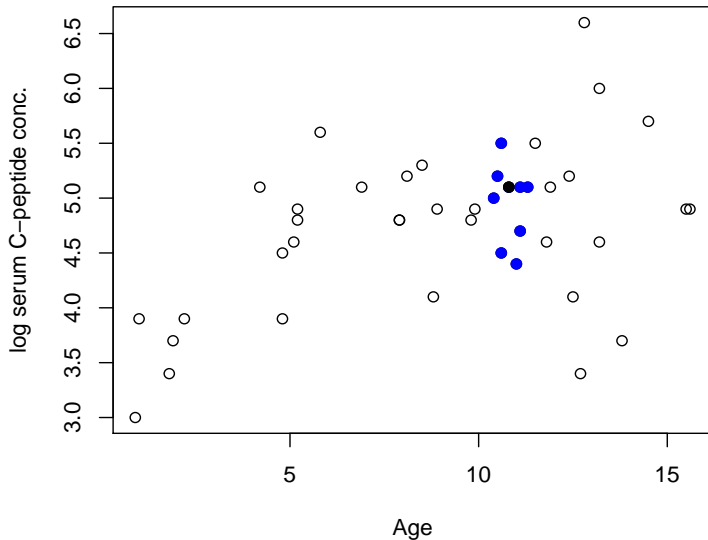
Lowess details

- How to weight points?
    - Cleveland proposed tricube function

$$T(t) = \begin{cases} (1 - \mid t^3 \mid)^3 & \text{for } \mid t \mid < 1 \\ 0 & \text{for } \mid t \mid \geq 1 \end{cases}$$

- weight for point x_j is $T\left(\frac{x_i - x_j}{h_i}\right)$

$x_i - x_j$

- Can have $f > 1$
  In this case, $h_i = f^{1/p} \max(d_{ij})$, where $p = \#$ X variables
- large f, e.g. $f \approx 100$, give $w_i = 1$ to all points for all x_i

Lowess details

- What form of local polynomial to fit?
    - Default is often quadratic, can also use linear
- Putting together the pieces:
- estimate $f(x_i)$ by:
    - computing weights, $w_j(x_i)$, for each observation j based on closeness to x_i
    - fitting weighted regression with
      $\text{E } Y = \beta_0 + \beta_1 x$ or $\text{E } Y = \beta_0 + \beta_1 x + \beta_2 x^2$
    - estimating $f(x_i)$ as $\hat{\beta}_0 + \hat{\beta}_1 x_i$ or $Y = \hat{\beta}_0 + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2$
    - Smoothness of estimated $\hat{f}(x_i)$ depends on smoothing parameter f
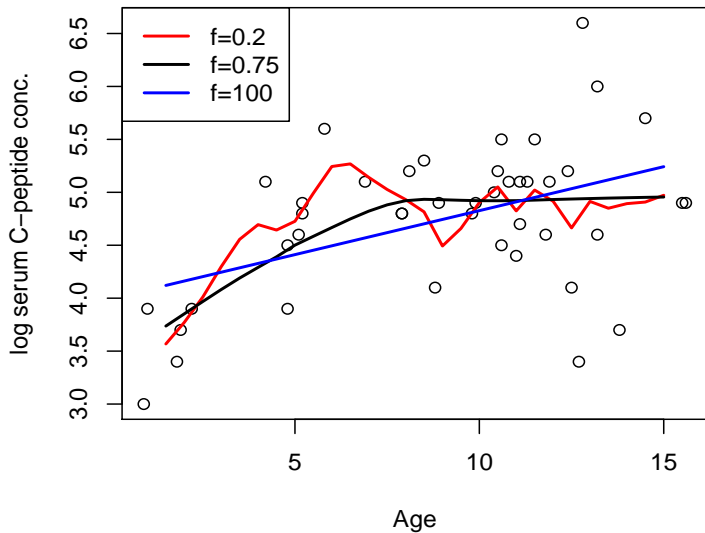
Loess to assess lack of fit

- Can use loess to assess lack of fit
  E $Y = \beta_0 + \beta_1 x$ is linear loess with $f \to \infty$
  E $Y = \beta_0 + \beta_1 x + \beta_2 x^2$: quadratic loess with $f \to \infty$
- Model comparison:
    - is a straight line appropriate?
      compare linear loess with $f = 100$ to linear loess with small f
    - is a quadratic curve appropriate?
      compare quadratic loess with $f = 100$ to quad. loess with small f
- Need to estimate model df and error df associated with a loess fit
- Intuitively depends on smoothing parameter
    - Smaller f means wigglier curve, larger model df, smaller error df
    - Very large f, e.g. ≈ 100 means model df = 2 or 3 and error df = $n - 2$ or $n - 3$
- Take same approach as we did with penalized regression splines

- $\hat{f}(x)$ is a linear combination of the observed y's, $l_i\, y$
- so $\hat{y} = Sy$ where each row of S is the appropriate l_i
- S is a smoother matrix, equivalent to P_X for a Gauss-Markov model
- define model df as $tr(S)$
- define error df as $n - 2tr(S) + tr(SS')$
- Degrees of freedom for diabetes data (n=43 obs.)

| | Linear loess | | | Quadratic loess | | |
span	model	error	total	model	error	total
0.2	11.00	27.89	38.89	17.49	21.8	39.33
0.75	3.04	39.02	42.06	4.61	37.51	42.12
100	2.00	41.00	43.00	2.99	39.99	42.98

- Both df are approximate because S usually not idempotent

"Robustifying" the fit

- Regression outliers (unusually large or small Y given the X value) can have a big influence, especially when considering only a few points in a local neighborhood
- Idea derived from "M-estimation", a type of robust regression goal: general pattern, less (or not) affected by individual values
- Provides a mechanism to reduce the influence of one (or more) outliers without deleting any observations
- Concept: (for either linear regression or smoothing) wt. regr. where the wt's depend on the size of the residual
- Downweight points with large absolute residuals
- A common choice is the bisquare weight function

$$B(t) = \begin{cases} (1 - t^2)^2 & t \leq 1 \\ 0 & t > 1 \end{cases}$$

- Cleveland suggests $\delta_i = B(e_i/6m)$, m is the median of $| e_1 |, | e_2 |, \cdots, | e_n |$

- M estimation fits very well with local estimation by weighting
- The loess algorithm to estimate $\hat{f}(x_i)$:
  - fit preliminary regression and calculate residuals, e_i
  - calculate $w_j(x_i)$ for each obs. based on f, fraction of points
  - calculate δ_i for downweighting residuals
  - WLS with weights = $w_j(x_i) \times \delta_i$
  - repeat last three steps, once or until convergence
  - estimate $\hat{y}(x_i)$
- In my experience, the choice of smoothing parameter is crucial
  choice of loess or penalized splines much less
  both give similar smooths, when similar amounts of smoothing
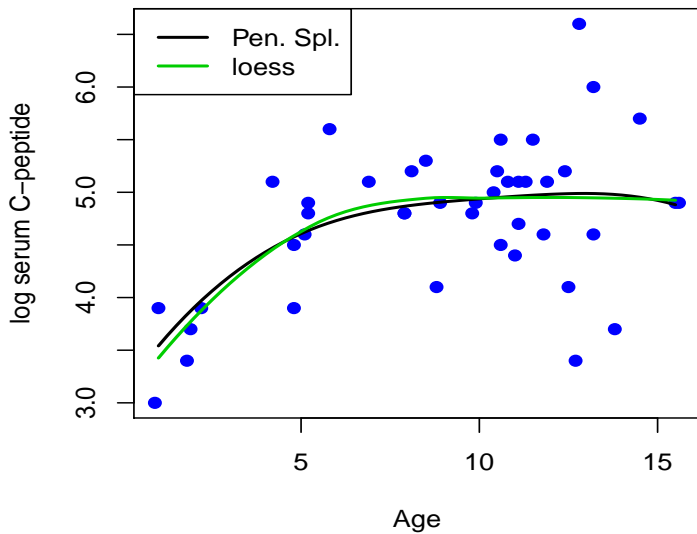
Computing for loess

```
# at least two functions to smooth using the loess alg
#  lowess()   older version, still used
#  loess()    what I use and will demonstrate

# use the diabetes data to illustrate
diabetes <- read.csv('diabetes.csv')

# basic call
diab.lo <- loess(y ~ age, data=diabetes)

# printing the object provides useful info
diab.lo

# and summary() provides more
summary(diab.lo)
```

Computing for loess

```
# the equivalent # parameters is not trace(S).
#   I don't understand what additional calculation is

# default fit is local quadratic (degree = 2) with f =

# useful helper functions:
#  predict, residuals both behave as you would expect.
#  Can specify newdata= in usual way

plot(diabetes$age,diabetes$y)
lines(seq(0,16,0.5),predict(diab.lo,newdata=data.frame

# can change span or default degree
temp <- loess(y~age,data=diabetes, span=100, degree=2)
lines(seq(0,16,0.5),predict(temp,newdata=data.frame(ag
```

Computing for loess

```
temp <- loess(y~age,data=diabetes, span=100, degree=1)
lines(seq(0,16,0.5),predict(temp,newdata=data.frame(ag

# can also use anova to do model comparison
anova(temp, diab.lo)

# one catch is that both models must be loess objects
diab.lm <- lm(y~age,data=diabetes)
anova(diab.lm, diab.lo)    # doesn't work

# so fit linear using loess() with degree=1 and large
# or quadratic using degree=2 and large span
```

Computing for loess

```
# default is no outlier downweighting.  add this by sp
#  family='symmetric'  (Default is family='gaussian'

diab.lo2 <- loess(y~age, data=diabetes, family='symmet

plot(diabetes$age,diabetes$y,col=4,pch=19)
lines(seq(0,16,0.5),predict(diab.lo,newdata=data.frame
lines(seq(0,16,0.5),predict(diab.lo2,newdata=data.fram
  col=4)
# very similar fits in this case because no really obv
```

Methods for large P, small N problems

- Regression has (at least) three major purposes:
  1. Estimate coefficients in a pre-specified model
  2. Discover an appropriate model
  3. Predict values for new observations
- Regression includes classification
  because classification can be viewed as a logistic regression
- Up to now (500, 511), have worked with data with small to moderate number of observations, small number of potentially important variables.
- What if many potential variables?
- e.g. 10,000 variables and 100 observations, $p >> n$
- Linear regression with all 10,000 variables will not work ($X'X$ singular)
- Forward stepwise or all subsets regression will not work well when predictors are highly correlated

- Often prediction is the most important goal, but need 1) and 2) to do prediction
- Practical methods sometimes called data mining.
- Huge area of current research: Statistics, Computer Science
- Large number of potential methods
- We look very quickly at four methods
  1. Principal Components Regression
  2. Partial Least Squares Regression
  3. Ridge Regression
  4. Lasso
- Support Vector Machines, Neural Networks, and elastic nets are other popular (or upcoming) techniques that we will not talk about.
- Hastie, Tibshirani, and Friedman, 2009, *The Elements of Statistical Learning* is my primary source for this material. Also discusses SVM's, NN's, and EN's.

Examples

- Arcene data:
  - Use abundance of proteins in blood to predict whether an individual has cancer (ovarian or prostate).
  - Training data: 10,000 variables, 100 subjects: 44 with cancer, 56 without
  - Validation data: to assess performance of model on new data (out-of-sample prediction) 100 subjects
- Corn grain data:
  - Use the near-infrared absortion spectrum to predict % oil in corn grain
  - Data from ISU grain quality lab, provided by Dr. Charlie Hurbugh
  - Training Data 1: 20 samples from 2008
  - Training Data 2: 337 samples from 2005 - 2008
  - Validation Data: 126 samples from 2009, 2010
  - 38 variables, all highly correlated: pairwise 0.79 - 0.9999

Principle Components Regression

- Useful for:
    - Reducing # X variables
    - Eliminating collinearity
- Example:
    - Corn Grain data, training set 2
    - 337 samples, 38 X variables
- X variables highly correlated: large off-diagonal elements of $X'X$.
- Remember the spectral (aka eigen) decomposition: $A = UDU'$, where $U'U = I$
- Center all X variables: $X_i^c = X_i - \mu_i$
- Apply decomposition to $X^{c'}X^c = UDU'$
- Compute new X variables: $X^* = XU$
- This is a rigid rotation around (0,0)
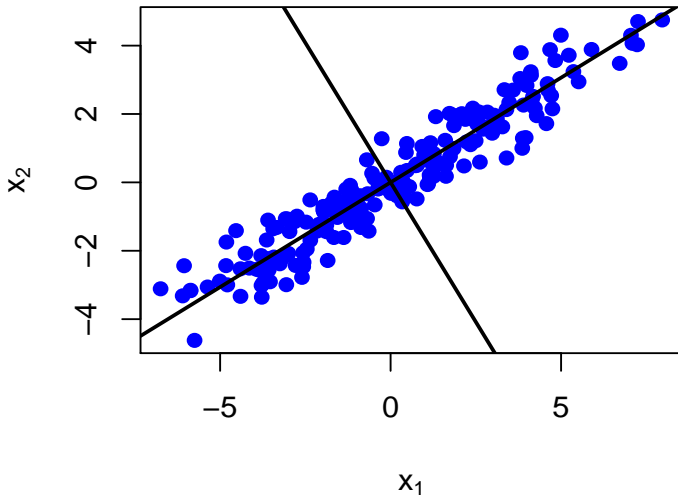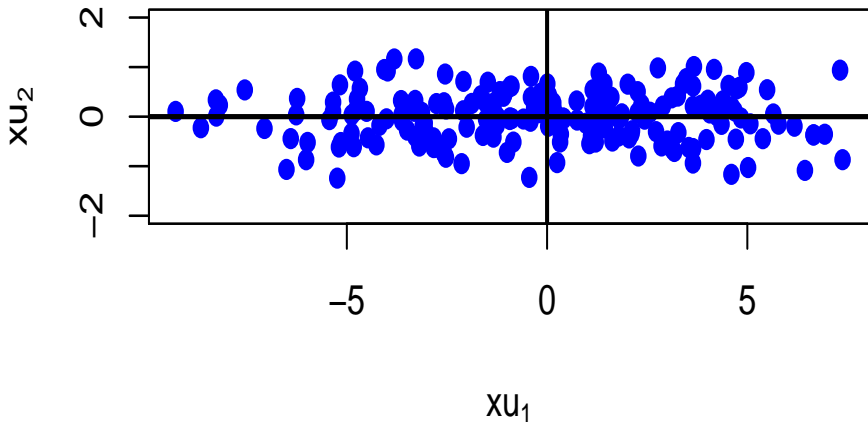
- Why use X^*? They are uncorrelated!
- So no multicollinearity problem.
- Estimate $\hat{\boldsymbol{\beta}}^* = (X^{*'}X)^- X'y$
- Now, want $\hat{\boldsymbol{\beta}}$ for original X variables
- $\hat{\boldsymbol{\beta}} = U\hat{\boldsymbol{\beta}}^*$ because E $Y = X^* b^* = X U b^*$
- If use k PCA axes, obtained from k X variables, can show that the $\hat{\boldsymbol{\beta}}$ are the same as the $\hat{\boldsymbol{\beta}}$ from OLS regression, assuming no numerical issues due to multicollinearity
- However, principal components have a second use.

- Subsets of the *k* PCA axes provide a reduced rank approximation of $\boldsymbol{X}$
- Some background:
- If X's are centered, $\boldsymbol{X}'\boldsymbol{X}/(N-1)$ is the variance-covariance matrix of the X's
- If X's are centered and scaled (so var $X_i = 1$), $\boldsymbol{X}'\boldsymbol{X}/(N-1)$ is the correlation matrix of the X's
- "Total variance" = $\sum_{i=1}^{k} VarX_i = tr(\boldsymbol{X}'\boldsymbol{X})/(N-1)$
- If $\boldsymbol{X}$ centered, total variance $= tr(D)/(N-1)$, because $tr(\boldsymbol{X}'\boldsymbol{X}) = tr(\boldsymbol{UDU}') = tr(\boldsymbol{DU}'\boldsymbol{U}) = tr(\boldsymbol{DI}) = tr(\boldsymbol{D})$
- If $\boldsymbol{X}$ centered and scaled, $tr(D) = k(N-1)$
- Eigenvalues (diagonal elements of D) customarily sorted from largest (D_{11} to smallest D_{kk}
- A subset of the *k* eigenvalues and associated eigenvectors provides a reduced rank approximation to $\boldsymbol{X}'\boldsymbol{X}$.

- Consider just $(XU)_1$ in the 2 X example:
- tr($X' X$)) = 2718.7, $diag(D)$ = [2669.1, 49.6]
- Most (98.2% = 2669.1/2718.7 of the total variability in the two variables is "explained" by the first rotated variable
- To see this, consider reconstructing X from just the first column of X^*.
- Since $X^* = XU$, $X^* U^{-1} = X$
- The 'one axis' reconstruction of X is $X_1 = X_1^* U_1$, where X_1^* is the first column of X^* and U_1 is the first row of U
- Aside: when U is full rank, it has a unique inverse. Since $U' U = I$, that inverse is U'.

x[,1]
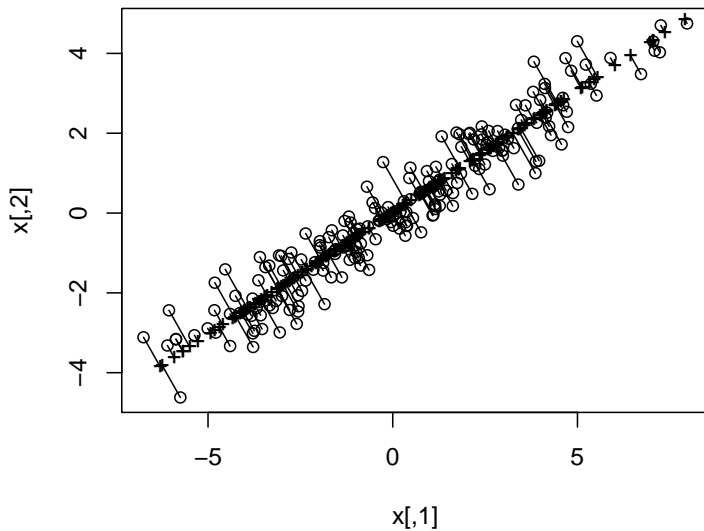
- Another way to think about PCA, motivated by previous plot.
- The X^* values for PCA axis 1 are the projections of X onto a vector. Could project onto any vector.
- An extreme choice on next page
- Would like to identify the vector for which the variance of the projected values is maximized.
- PCA axis $\nu_m = \alpha$ for which Var($X\alpha$) is maximized subject to:
    1. $||\alpha|| = 1$, avoids trivial changes; Var $X\,2\alpha$ = 4 Var $X\alpha$.
    2. $\alpha'\nu_i = 0$, $\forall i = 1, \ldots, m-1$, ensures that axis m is orthogonal to all previous axes (1, 2, ..., m-1)
- The set of ν_m is the matrix of eigenvectors U!
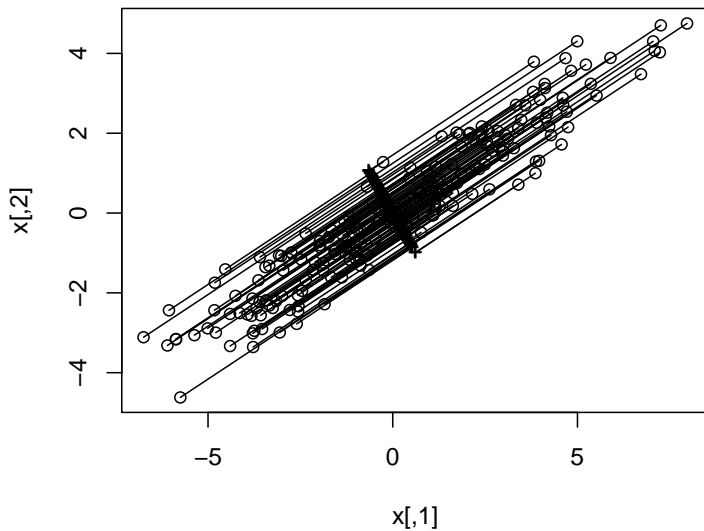
- Would have a problem if tried to use X_1 as two X variables in a regression! $X_1'X_1$ is singular.
- Don't need to: use only X_1^* (one column) in the regression
- Suggests a way to fit a regression with more X variables than observations (e.g. Corn data from 2008 only: 20 obs, 38 X variables)
- Do a PCA on $X'X$ to create X^*. Fit a sequence of regressions:
    - only first column of X^*, i.e. X_1^*
    - first two columns of X^*, i.e. $[X_1^*, X_2^*]$
    - first three columns, ... many columns
- Use cross validation or GCV to choose optimum number of PCA axes to include in the regression.

# Partial Least Squares (PLS) regression

- The PCA at the center of PC Regression only considers the variability in $\boldsymbol{X}$.
- No reason that the linear combination of $\boldsymbol{X}$ that has the largest variance (PCA axis 1) also has to be effective at predicting $\boldsymbol{Y}$
- PCA axes 5 and 7 may be the most highly associated with Y.
- PLS finds axes that simultaneously explain variation in $\boldsymbol{X}$ and predict Y
- The PLS axis ν_m is the α that maximizes

$$Corr^2(\boldsymbol{y}, \boldsymbol{X}\alpha) \; Var(\boldsymbol{X}\alpha),$$

subject to:

1. $||\alpha|| = 1$, avoids trivial changes; Var $\boldsymbol{X}\, 2\alpha$ = 4 Var $\boldsymbol{X}\alpha$.
2. $\alpha'\nu_i = 0$, $\forall i = 1, \ldots, m-1$, ensures that axis m is orthogonal to all previous axes (1, 2, ..., m-1)

- Contrast with PCA problem: max Var($\boldsymbol{X}\alpha$)

- Including the correlation with *y* seems appealing.
- My practical experience is that the max Var aspect of the problem dominates the max Corr, so PLS often very similar to PCR.
- Example: Predicting %Oil in corn grain.
- *X* is absorbance at 38 wavelengths in the Near Infrared. Data for 10 samples:

Wavelength
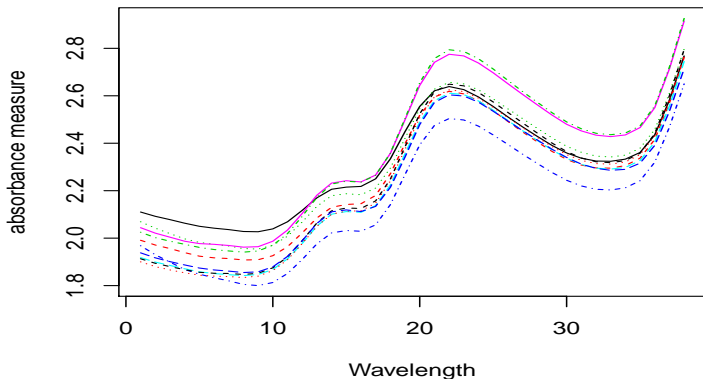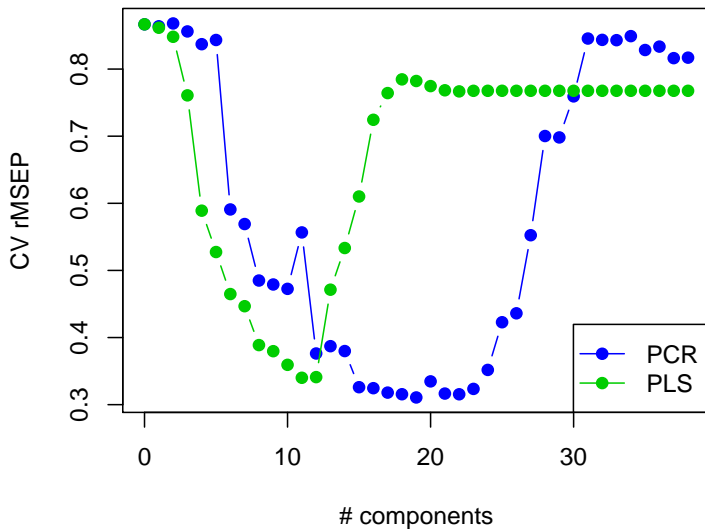
Ridge Regression

- An old idea (Hoerl and Kennard, 1970, Techn.) currently in revival
- Originally proposed as a solution to multicollinearity
- Problem is that two or more X variables are correlated
- i.e. some off diagonal elements of $X'X$ are large relative to the diagonal elements
- The ridge regression fix: add an arbitrary constant to the diagonal elements
- i.e. estimate $\hat{\boldsymbol{\beta}}$ by

$$\hat{\boldsymbol{\beta}} = (X'X + \lambda I)^{-1} X' Y$$

- Choice of λ changes $\hat{\boldsymbol{\beta}}$ and fit of model
- No justification for one choice or another
- But, we've seen expressions like $(X'X + \lambda I)^{-1} X' Y$ very recently when we talked about penalized least squares.

- Penalized splines: $(X'X + \lambda^2 D)^{-1} X' Y$ is the solution to the penalized LS problem:

$$min(\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) + \lambda^2 \sum u_j^2$$

- The ridge regression estimates are the solution to a penalized LS problem!

$$min\left[(\mathbf{y} - \mathbf{X}\beta)'(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta'\beta\right]$$

- This insight has lead to the recent resurgence of interest
- Concept of the penalty makes sense. One consequence of multicollinearity is large variance in β. The penalty helps keep β's closer to zero.
- Another version of penalized regression:
- Instead of using $\lambda \sum \beta_i^2$ as the penalty, use $\sum |\beta|$.
- The first is called an L_2 penalty; the second an L_1 penalty

The LASSO

- The LASSO: find β that minimize:

$$(\boldsymbol{y} - \boldsymbol{X}\beta)'(\boldsymbol{y} - \boldsymbol{X}\beta) + \lambda\beta'\boldsymbol{1}$$

- Seems like a trivial change.
- Turns out to have major and useful consequences
- Because of the shape of the surface of constant penalty
- Consider a regression with 2 X variables. Y, X_1 and X_2 have mean 0.
- Correct model is E $Y = \beta_1 X_1 + \beta_2 X_2$, $\beta_1 = 4, \beta_2 = 0.5$
- Consider the contours of $SS = (Y - \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2)^2$ for a grid of (β_1, β_2)
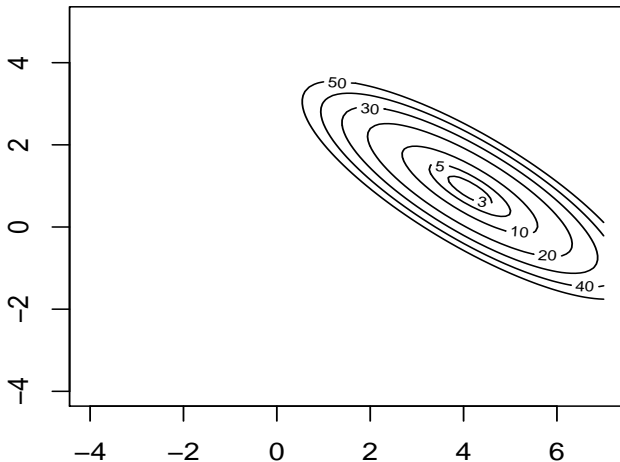
- Now, think about penalized LS by considering the penalty as a constraint
- What is the LS estimate of (β_1, β_2) given the constraint that $\beta_1^2 + \beta_2^2 = 4$
- Given by the (β_1, β_2) on the constraint surface for which the LS surface is a minimum
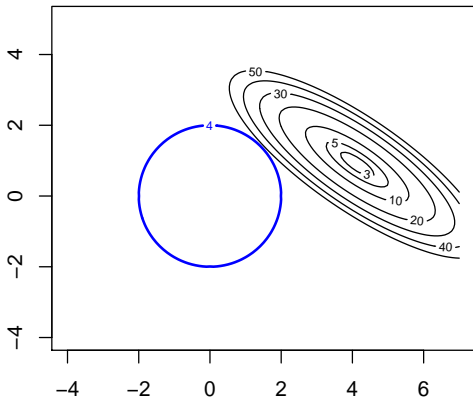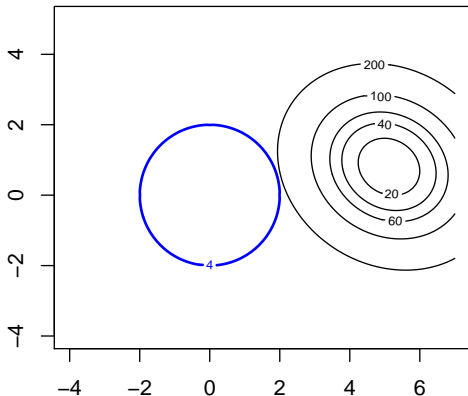
- Changing to L1 penalty changes the shape of the constraint surface
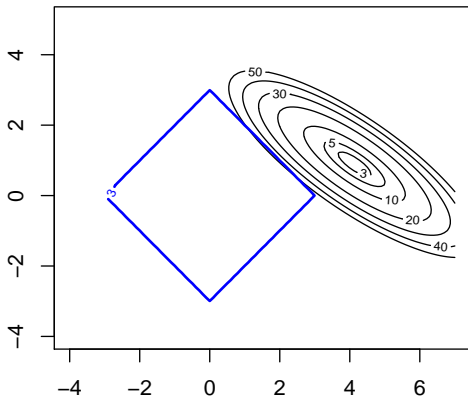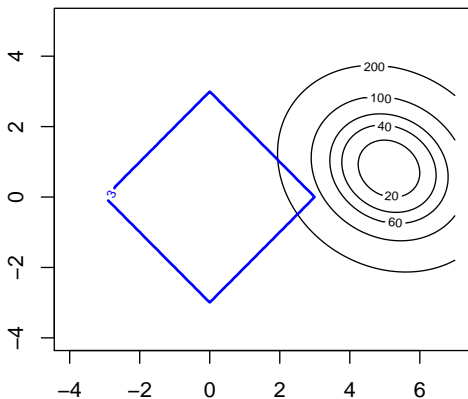

- In this case, $\hat{\beta}_1 \approx 2$, $\hat{\beta}_2 \approx 1$

- The shape of the LASSO (L1 penalty) tends to force $\hat{\beta}$ to 0
- 2nd data set, $\beta_1 = 5$, $\beta_2 = 1$

- $\hat{\beta}_1 \approx 1.9$, $\hat{\beta}_2 \approx 0.7$

- But, the L1 penalty forces $\hat{\beta}_2$ to 0

- In this case, $\hat{\beta}_1 = 3$, $\hat{\beta}_2 = 0$

- In practice, the LASSO seems to give better prediction equations than do PCR and PLS.
- Provides a mechanism to do variable selection .
- This has been just a quick introduction to all these methods. There are many details that I have omitted.

Classification and Regression Trees

- Another option for problems with complicated interactions
- A very quick way to get a boat-load of X variables is to consider all possible interactions:
  10 X's: 45 cross-products and 120 three-variable interactions!
- Could imagine estimating
  $f(X_1, X_2, \ldots, X_k, X_1 X_2, \ldots X_{k-1} X_k, X_1, X_2, X_3, \ldots X_{k-2} X_{k-1} X_k \ldots)$
- Gets hard beyond $k = 4$: "Curse of dimensionality"
- In high dimensions, every point is isolated (see next slide)
- Solutions:
  1. Assume effects are additive E $Y = f(X_1) + f(X_2) + \ldots + X_k$
  2. What if you expect interactions or contingent response to a variable?
     - Contingent response:
       if $X_1 \geq$ e.g. 25, Y unrelated to X_2
       if $X_2 <$ e.g. 25, Y depends on X_1 and X_2
- Tree models provide a way to model contingent responses parsimoniously
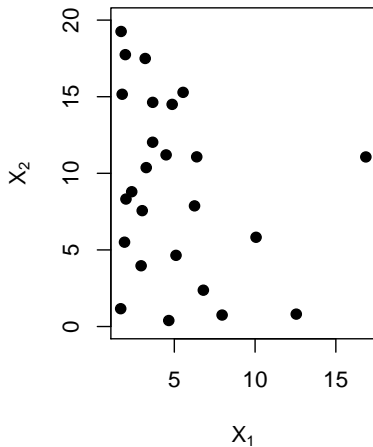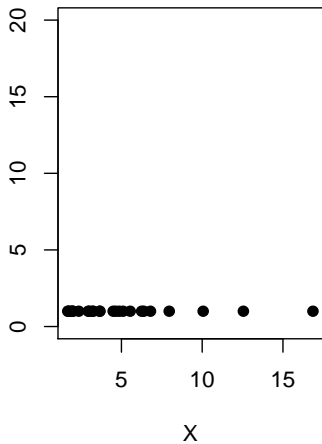
- Best known method is the Classification and Regression Tree (CART)
- Breiman et al. (1984), *Classification and Regression Trees*
- Concept: (for 1 X variable)
  - Any E $Y = f(X)$ can be approximated by a sequence of means

$$
f(X) = \begin{cases}
\mu_1 & X \leq k_1 \\
\mu_2 & k_1 > X \leq k_2 \\
\vdots & \vdots \\
\mu_K & k_{K-1} \leq X < k_K \\
\mu_{K+1} & k_K \leq X
\end{cases}
$$
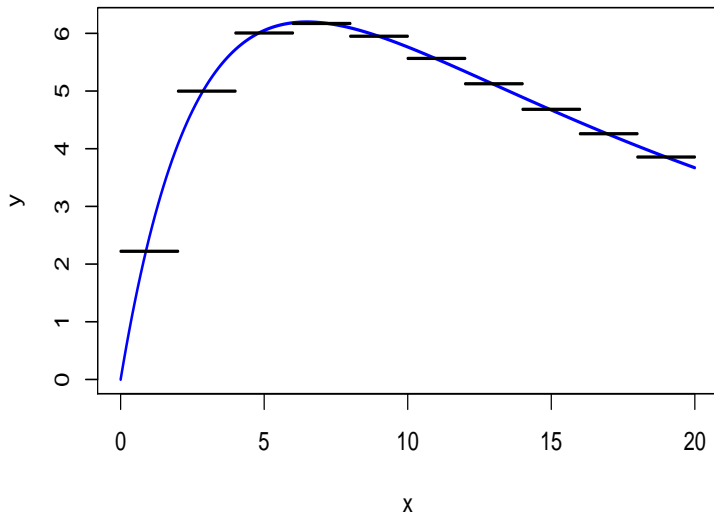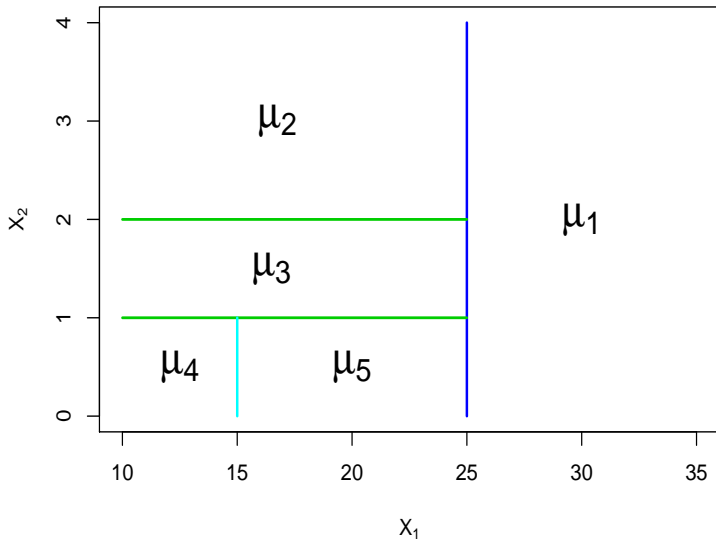
- Given Y, assumed continuous, and a set of p potentially useful covariates $\{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_p\}$, how do we find a subset of k $\boldsymbol{X}$'s and a set of split points $\boldsymbol{s}$ that define an optimal partition of regions $R = \{R_1, R_2, \ldots R_M\}$.

- Criterion: for continuous $\boldsymbol{Y}$, least squares

$$\sum_{j=1}^{M} \sum_{i \in R_j} (Y_i - \mu_j)^2 \tag{1}$$

  - Focus is on finding the partition.
  - If partition R known, the region means $\{\mu_1, \mu_2, \ldots \mu_M\}$ that minimize (1) are easy to estimate
  - They are the sample average for each region

- Difficult part is finding the optimal partition into regions

- i.e. (previous picture): how to discover which covariates are $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, and where to split each covariate?

- use a greedy algorithm: make a series of locally optimal choices (best next step), in the hope that together they provide the globally optimal solution.
- Given all the data, 1st step is to split into two half-planes based on variable X_j and split point s:

$$R_1(j, s) = \{X : X_j \leq s\}$$
$$R_2(j, s) = \{X : X_j > s\}$$

that minimize

$$SS = \sum_{i:X \in R_1(j,s)} (Y_i - \bar{Y}_1)^2 + \sum_{i:X \in R_2(j,s)} (Y_i - \bar{Y}_2)^2$$

- This search is quite fast, because the model is constant mean within region
- If $X_j = 1$, 1.2, 1.6, 2, 5, any s between 1.6 and 2 has same SS. Only need to consider one s, e.g. $s=$ average of two adjacent X values.

- Now consider R_1 alone: split into two half planes, R_{11} and R_{12}
- Consider R_{11} alone: split into two half planes R_{111} and R_{112}
- Continue until either:
  - reduction in SS after splitting is small
  - small number of points in region
    5 is a commonly used minimum number of points.
    will not split a region with 5 or fewer points
- Back up one level and consider the other side of the split
- Split it as appropriate
- Result is a tree giving the partition into regions and the associated $\hat{Y}$ for each region

- Example: Minneapolis housing price data from Kutner et al.
- Data: Sale price of a house and 11 characteristics (e.g. size, lot size, # bedrooms, ….
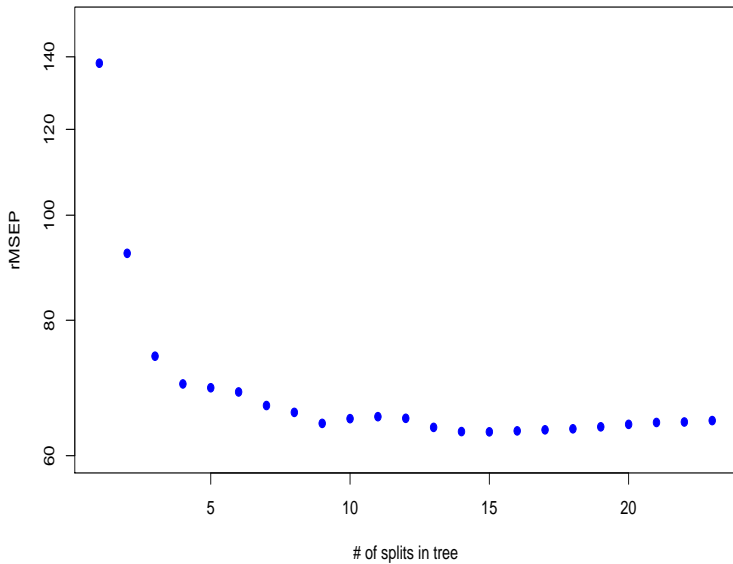- Goal: predict expected sale price given the characteristics
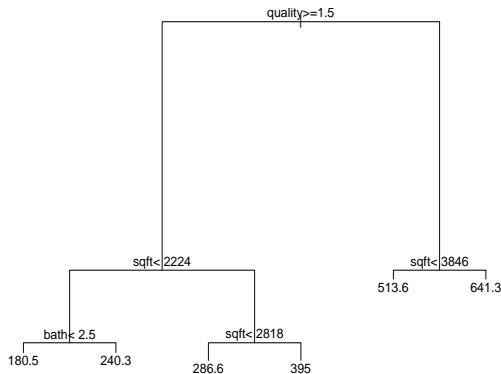
- Suggests # bathrooms important for small houses, not others

- How large/complex a tree should you produce?
- A tree allowing smaller decrease in SS per split
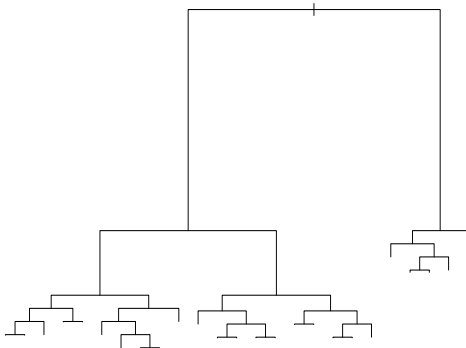
- What is "out of sample" prediction error for each size of tree (# of splits)?
- estimate by ten-fold cross-validation

# of splits in tree

rMSEP

- Familiar pattern from model selection in multiple regression:
- Simple models have large rMSEP: mostly bias
- Models that are too complex have slightly larger rMSEP: overfitting the training data set
- General advice: fit too large a tree then prune it back
  Increases the chance that find good global optimum (details in Hastie, Tibshirani and Friedman, 2009, *The Elements of Statistical Learning*, p 308.)
- Other general advice: If you believe two (or more variables) have additive effects, don't use a tree.
    - Split on X_1, then each subset of the data is split on X_2
    - Same X_2 split in both sides. Tree can't take advantage.
    - Use a GAM
- Trees are low bias, high variance predictors / classifiers

Classification Tree

- response is 0/1 or 1/2/.../K
- Want to classify an observation given X
- Same strategy: divide covariate space into regions
- For each region and class, calculate $\hat{p}_{mk}$, the proportion of observations in region m in class k
- Classify the region as type k if k is the most frequent class in the region
- Replace SS (node impurity for continuous data) with one of
  - Misclassification rate: $1 - \sum \hat{p}_{mk^*}$, where k^* is the set of classes other than the true class
  - Gini index: $\sum \hat{p}_{mk}(1 - \hat{p}_{mk})$
  - Deviance: $-\sum \hat{p}_{mk} \log \hat{p}_{mk}$

Random Forests

- Crazy-seeming idea that works extremely well for a low bias, high variance classifier (like a tree)
    - Draw a NP bootstrap sample of observations (resample with replacement)
    - Construct classification or regression tree T_b
    - Repeat B times, giving T_1, T_2, ... T_B
- For a specified set of covariates, $\boldsymbol{X}$, each tree gives a prediction $\hat{Y}_b$
- The Random Forest prediction is
    - regression: Average of the B tree-specific predictions
    - classification: most frequent predicted class among the B trees.
- Example of a general technique called *Bootstrap Aggregation* or *bagging*

- Why this works: Consider a regression. Assume $\hat{Y} \sim (Y, \sigma^2)$
- If tree predictions are independent, Var $\bar{\hat{Y}} = \sigma^2/B$
- If some positive pair-wise correlation, ρ, Var $\bar{\hat{Y}} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- Still less than σ^2, but better if ρ close to 0
- One final detail that reduces correlation between trees (and hence between their predictions)
  - At each split in each bootstrap tree, randomly choose $m \leq p$ variables as splitting candidates. Often $m \approx \sqrt{p}$
- Does this work: Minneapolis house data, 10 fold c.v.

Method	rMSEP
Tree	64.86
Forest	55.39

- Disadvantage:
  no convenient way to summarize / describe the model

Forests or trees?

- One of my current research collaborations: predicting invasiveness of woody trees and shrubs
- The horticulture industry wants to import shrubs and trees from other countries.
  - Showy flowers, interesting form, survives in nasty places
- Some, e.g. buckthorn, become invasive pests, invading native forests.
- The USDA Ag and Plant Health Inspection Service (APHIS) has to decide whether to permit importation of new plants.
- The million $ question: will it be invasive. ????
  - Various "seat-of-the-pants" models
  - A CART model beats all of them in predicting invasiveness in IA
  - And Chicago, and northern IN, and southern MN, and northern MO.
  - Nice, clean tree that we have published so others can use our results
    (See next slide)
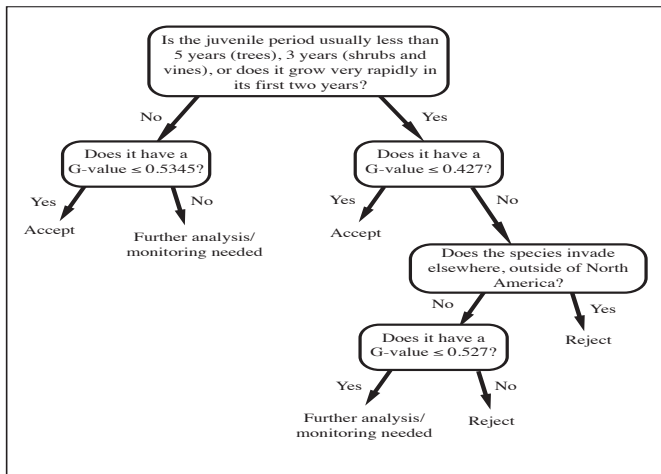
- A random forest makes even better predictions
- But, have 1000 trees. Can't show them.
- The model lives in a computer.
- We're currently writing a proposal to put the RF model on a web server, so everyone can use it.

Numerical Optimization

- Huge field, many different techniques, active research area
- Has impact in just about every scientific field
- Applied statisticians need a basic familiarity with optimization
- and how to help it when the usual doesn't work
- That's my goal in this section
- Will talk about:
    - golden section search
    - Gauss-Newton (have already done this)
    - Newton-Raphson
    - "derivative-free" methods (e.g. BFGS)
    - Nelder-Mead
    - the Expectation-Maximization (EM) algorithm
- Will talk about minimization (max f = min $-f$)

Golden section search

- For a unimodal function of a scalar (univariate) function
- Requires two starting values, *a* and *b*. Finds location of the minimum of $f(x)$ in [*a*, *b*]
- The concept:
    - Choose two points *c* and *d* s.t. $a < c < d < b$
    - Calculate $f(a)$, $f(b)$, $f(c)$ and $f(d)$
    - if $f(c) < f(d)$, minimum is somewhere between *a* and *d* replace *b* with *d*
    - if $f(c) > f(d)$, minimum is somewhere between *c* and *b* replace *a* with *c*
- A picture will make this much clearer
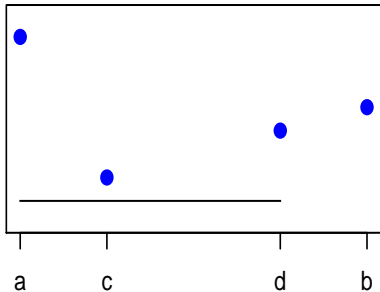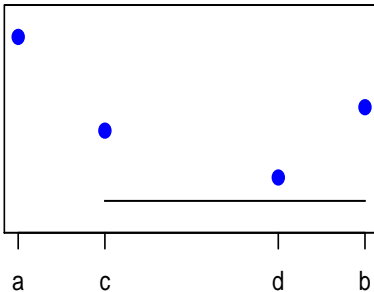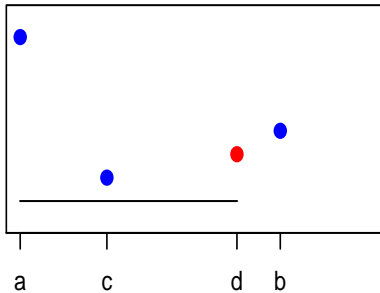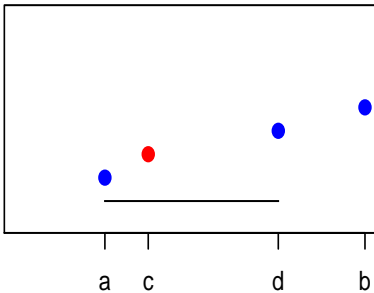
- So, how to pick c and d?
- Convergence is faster when:
  - $c - a + d - c = b - c$ i.e. $d - a = b - c$
  - ratios of distances between points in "old" and "new" configurations are equal
- result is when $\left(\frac{c-a}{d-b} \right)^2 = \frac{c-a}{d-b} + 1$
- i.e. $\frac{c-a}{b-d} = \frac{1+\sqrt{5}}{2}$
- This is the "golden ratio" of mathematical fame, which gives this method its name.
- Iterate until the bracketing region is sufficiently small
- Usually defined relative to minimum achievable precision for the operating system
- N.B. Only works for univariate problems!

Practical issues

- This algorithm is very robust. It has always worked for me.
- If you have multiple minima in [a, b], algorithm finds one of them
  Not necessarily the correct (smallest) minimum.
- Variation finds roots of a function, i.e. x s.t. $f(x) = 0$
  - Need to choose a and b s.t. $f(a)$ and $f(b)$ have opposite signs
  - i.e. $f(a) > 0$ and $f(b) < 0$
  - Which guarantees (for a continuous function) there is a root in [a, b]
- I find profile likelihood confidence intervals by finding roots of
  $\log L(\theta_{mle}) - \log L(\theta) - \chi^2_{1-\alpha,DF}/2 = 0$
- find one endpoint by finding root between [$small$, θ_{mle}] and the
  other by finding root between [θ_{mle}, $large$]

Minimization for functions of vectors

- We've already seen one algorithm for minimization of a function of vectors: Gauss-Newtown (see part 5, slides 47, 48)
- Another way to write one step of the Gauss-Newton iteration:
  - $\beta^{(i+1)} = \beta^{(i)} + (\boldsymbol{D}'\boldsymbol{D})^{-1}\boldsymbol{D}^i(\boldsymbol{y} - f(\boldsymbol{x}, \beta))$
  - where $\boldsymbol{D}$ is the matrix of derivatives of f w.r.t. each element of β evaluated at each X
- A related algorithm: Newton-Raphson
  - to find a root of a function, i.e. x s.t. $f(x) = 0$
  - If $f(x)$ is linear, the root is $x - \frac{f(x)}{f'(x)}$, where $f'(x) = \frac{df}{dx} \mid x$
  - If $f(x)$ is non-linear, can find the root by successive approximation
    $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$
    $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \ldots$
  - We need to find the minimum/maximum of a function, i.e. where $f'(x) = 0$, so our iterations are:
    - $x_1 = x_0 - \frac{f'(x_0)}{f''(x_0)}$ \qquad $x_2 = x_1 - \frac{f'(x_1)}{f''(x_1)}, \ldots$

Newton-Raphson and Fisher scoring algorithms

- For functions of vector valued parameters, the scalar derivatives are replaced by gradient vectors and Hessian matrices:
    - $\boldsymbol{X}_1 = \boldsymbol{X}_0 - \boldsymbol{H}_0^{-1}\boldsymbol{D}_0$
      $\boldsymbol{X}_2 = \boldsymbol{X}_1 - \boldsymbol{H}_1^{-1}\boldsymbol{D}_1, \ldots,$
    - where $\boldsymbol{D}_i$ is the gradient vector evaluated at $\boldsymbol{X}_i$ and $\boldsymbol{H}_i$ is the Hessian matrix evaluated at $\boldsymbol{X}_i$.

- Fisher scoring: replace the observed Hessian matrix with its expected value
    - E $-\boldsymbol{H}^{-1}$ is the Fisher information matrix from Stat Theory
    - Requires calculating the expectation
    - In expectation, E $\boldsymbol{H}$ = E $\boldsymbol{D}'\boldsymbol{D}$
    - Historically useful because gave simple expressions for hand-worked iterative calculations for many discrete distributions
    - In many cases, Newton-Raphson works better than Fisher scoring

Practical considerations

- i.e. does it matter which method choose?
  - Newton-Raphson requires the Hessian (2nd derivative) matrix
  - Easy to compute if have analytic second derivatives
  - If have to compute gradient and/or Hessian numerically, considerably numerical inaccuracy in Hessian (more below)
  - To me, not much of a practical difference
  - Numerical analysts focus on this sort of issue; I don't.
  - For me: what is provided / used by software trumps theoretical concerns.
  - John Tukey: practical power = product of statistical power and probability a method will be used.

"Derivative-free" methods

- What if you don't (or don't want to) calculate an analytic gradient vector or Hessian matrix?
- For Gauss-Newton: replace the analytic gradient with a numeric approximation
  - Two possibilities:
    - "forward" differences: $\frac{df}{dx} \mid x_0 \approx \frac{f(x_0+\delta)-f(x_0)}{\delta}$
    - "central" differences: $\frac{df}{dx} \mid x_0 \approx \frac{f(x_0+\delta)-f(x_0-\delta)}{2\delta}$
  - "central" differences are better approximations, but require more computing when X is large
- For Newtown-Raphson, also need to approximate the Hessian (2nd derivative) matrix
  numerical accuracy issues are a serious concern: differences of differences are subject to serious roundoff error
- various algorithms reconstruct the Hessian matrix from a sequence of estimated derivatives
- The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is the most popular (for good reasons).

Dealing with problems

- What if the algorithm doesn't converge?
    - First, and foremost, check that any analytic quantities (derivatives, Hessian matrices) are correct.
        - Many times I've made a sign error in one of my calculations
        - Has serious consequences; usually stops algorithm from making progress!
        - Compare analytic derivatices (or Hessian) to numerical approximations.
        - Should be close
    - Consider different starting values
        - if start at a "bad" place, e.g. gradient essentially flat there, will not make progress
- Working on the above two things usually fixes problems.
- If not, consider a different algorithm, e.g. Nelder-Mead, which does not depend on derivatives

Step-halving

- One practical detail
- G-N, N-R, and BFGS all have the form:
  $\boldsymbol{X}^{(i+1)} = \boldsymbol{X}^{(i)} + $ "step"
- Step specifies both the direction to move and the amount to move.
- Sometimes the direction is reasonable, but the amount is not.
- Picture on next slide (NR iterations), started at (50,0.2)
- Practical solution: Step-halving
  - when $f(\boldsymbol{X}^{i+1}) > f(\boldsymbol{X}^i)$, i.e. algorithm 'getting worse'
  - replace "step" with "step"/2
  - if that's still too much, repeat (so now 1/4 original step)
- N-R step starting at (54, 0.5) goes to (41, 0.68)
  lnL from 62.9 down to 62.0
- Red dot would be the 'step-halved' next try, lnL > 64
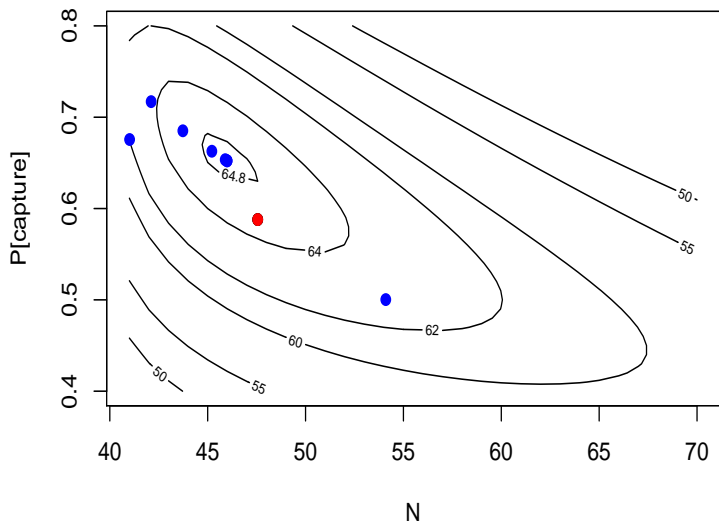
Nelder-Mead algorithm

- Concepts, not details
- *N* is the length of the parameter vector
- Construct a $N + 1$ simplex in the parameter space (picture on next slide)
- Find the worst vertex (largest if minimizing, smallest if maximizing)
- Replace it by a new point
    - Basic step is reflection (another picture)
    - But also expansion, contraction and reduction
    - (As I said, details don't matter)
- Fundamentally different algorithm from others
- Doesn't directly use any gradient information
- So, what do I use?
    - BFGS is my standard default choice
    - If that struggles, I switch to Nelder-Mead