Lecture 12: Solving Linear Systems.
Overview for lecture 12

1. R commands for matrices and vectors (reference slides)

2. Solving linear systems $Ax = b$.
   
   (a) Forwards and Backwards substitution
   (b) Solving $Ax = b$ for full rank $A$ using LU factorization
   (c) Regression.
   (d) Over-determined systems. Numerical stability and QR factorization.
Solving linear systems

Suppose $A$ is a real $n \times p$ matrix of rank $p$ with $p \leq n$, and entries $a_{i,j}$, and $b$ is an $n \times 1$ real vector.

Many important numerical problems reduce to

$$Ax = b$$

for $x$.

If $p < n$, then the system is over-determined. We come back to this case later. We will look at how the equations $Ax = b$ may be solved when $p = n$ so that $A^{-1}$ exists and $x = A^{-1}b$.

R has a function `solve(A)` returning $A^{-1}$ so we could compute

$$x = \text{solve}(A) \%*\% b.$$ 

We will see that this is inefficient and numerically unstable, and find that the best method depends on the properties of $A$. 
Forward and Backward elimination

Suppose $A$ is lower triangular so that $a_{i,j} = 0$ for $i > j$. Solve $Ax = b$ for $x$ using forward substitution. Chop the $n$ equations in $Ax = b$ into blocks

$$A = \begin{pmatrix} a_{11} & 0_{1 \times (n-1)} \\ A_{21} & A_{22} \end{pmatrix}$$

Here $A_{21} = A_{2:n,1}$ is $(n-1) \times 1$ and $A_{22} = A_{2:n,2:n}$ is itself lower triangular and $(n-1) \times (n-1)$. Now $Ax = b$ is

$$\begin{pmatrix} a_{11} & 0_{1 \times (n-1)} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_{2:n} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_{2:n} \end{pmatrix}$$

The top row of the matrix says $a_{11}x_1 = b_1$ so $x_1 = b_1/a_{11}$.
The bottom block of the matrix has \((n - 1)\) rows

\[
\begin{pmatrix}
A_{21} & A_{22}
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_{2:n}
\end{pmatrix}
= b_{2:n}
\]

\[A_{21}x_1 + A_{22}x_{2:n} = b_{2:n}\]

\[A_{22}x_{2:n} = b_{2:n} - A_{21}x_1\]

\[\tilde{A}\tilde{x} = \tilde{b} \quad \text{now } (n - 1) \times (n - 1)\]

We are left with a smaller version of the problem we started with.

It took \(2(n - 1) + 1\) additions, subtractions, multiplications and divisions (called 'flops') to solve for \(x_1\) and calculate \(\tilde{A}\) and \(\tilde{b}\). Since \(\sum_{i=1}^{n}(2i - 1) = n^2\), forward solving is \(n^2\) flops.

\(R\) has \texttt{forwardsolve(A,b)} for forward elimination for \(n \times n\) lower triangular \(A\) and \(n \times 1\) \(b\). There is \texttt{backsolve(A,b)} for backward elimination on upper triangular \(A\).
LU factorization

The most efficient method for solving $Ax = b$ for a general full rank $n \times n$ square matrix is to factorize

$$A = LU$$

into a lower $L$ and upper $U$ triangular matrices * at a cost of $2n^3/3 + O(n^2)$ flops (we haven't proven this, it's just assertion) and then solving $LUx = b$ by setting $y = Ux$ and then

solving $Ly = b$ (forwards)

and then

solving $Ux = y$ (backwards).

The function `solve(A,b)` uses this method. The two elimination steps take $2n^2$ flops so the leading term in the number of flops is $2n^3/3$.

*if there is no $LU$ factorization we seek $A = PLU$ with $P$ a permutation.
Normal linear models

Consider the aids data

```r
> d = read.table("AIDS.txt")
> head(d)
cases time time.sq
1 185 1 1
2 200 2 4
3 293 3 9
4 374 4 16
5 554 5 25
6 713 6 36
> (n<-dim(d)[1])
[1] 25
```
Suppose we want to fit the normal linear regression model

\[ y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, \quad i = 1, 2, \ldots, n \]

with \( y_i \) the number of cases in month \( x_i \), and \( \varepsilon_i \sim N(0, \sigma^2) \) iid normal errors. In vector form the model is

\[
\begin{pmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_n \\
\end{pmatrix}
= \begin{pmatrix}
  1 & x_1 & x_1^2 \\
  1 & x_2 & x_2^2 \\
  \vdots & \vdots & \vdots \\
  1 & x_n & x_n^2 \\
\end{pmatrix}
\begin{pmatrix}
  \alpha \\
  \beta_1 \\
  \beta_2 \\
\end{pmatrix} + \begin{pmatrix}
  \varepsilon_1 \\
  \varepsilon_2 \\
  \vdots \\
  \varepsilon_n \\
\end{pmatrix}
\]

or

\[ y = X\theta + \varepsilon \]

with \( \theta = (\alpha, \beta_1, \beta_2)^T \) etc.
The $R$ commands to fit this normal linear model are

d.lm=lm(cases ~time+time.sq,data=d)
summary(d.lm)

Here `d.lm` is a list full of results from the model fit output by `lm()`. Notice the R formula notation `cases~time+time.sq`.

The columns of `summary(d.lm)` output give $\hat{\theta}_i$, an estimate $\hat{\sigma}_i$ of the error in $\hat{\theta}_i$, and columns for the test $H_0: \theta_i = 0$.

If the model is good, the regression should interpolate the data with normal residuals $y - X\hat{\theta}$. We can check this using a normal qq-plot for the residuals,

`qqnorm(residuals(d.lm)); qqline(residuals(d.lm))`. 
What's inside the `lm()` box?

The equations $X\theta = y$ are over-determined (more equations than variables, $n > p$, we can't expect a solution), so minimize $R(\theta) = (y - X\theta)^T(y - X\theta)$; get $X\theta$ as close as we can to $y$.

$$R(\theta) = \sum_{i=1}^{n} (y_i - \alpha - \beta_1 x_i - \beta_2 x_i^2)^2$$

$$= (y - X\theta)^T(y - X\theta)$$

$$= (X\theta)^T X\theta - 2y^T X\theta + y^T y$$

Taking partial derivatives wrt $\theta$ and imposing $\frac{\partial R}{\partial \theta} = 0$ ($p$ equations) leads to the $p$ normal equations

$$X^T X \theta = X^T y$$

for $\theta$ in this over-determined system. This is $Ax = b$ with $A = X^T X$, $x = \theta$ and $b = X^T y$. 
Solving the normal equations using QR factorization

We could use LU factorization to solve the normal equations. However QR factorization is usually best as it is more stable numerically.

\[
X = \begin{pmatrix}
1 & -1 \\
0 & 10^{-10} \\
0 & 0
\end{pmatrix}
\]

\[
X^T X = \begin{pmatrix}
1 & -1 \\
-1 & 1 + 10^{-20}
\end{pmatrix}
\]

At machine precision $1 + 10^{-20}$ and 1 are equal so $X^T X$ appears to be singular. Any method (like LU) that solves $(X^T X) \theta = X^T y$ by first computing $X^T X$ will fail on this problem.

Instead, factorize $X = QR$ ($Q$ is $n \times p$ and orthogonal, so $Q^T Q = I_{p \times p}$, and $R$ is $p \times p$, upper triangular, and has positive entries on
the diagonal). This takes $2np^2$ flops (assertion). Since

$$X^T X = R^T Q^T Q R,$$

the normal equations

$$X^T X \theta = X^T y$$

are

$$R^T R \theta = R^T Q^T y.$$

We can solve these by

solving $R \theta = Q^T y$ (backwards)

$(np+p^2$ flops) for an overall leading order cost of $2np^2$ flops. The functions `qr.solve(X,y)` and `lm()` use this method. LU would take $np^2$ but may fail.
In R,

\[ X = \text{cbind}(\text{rep}(1,n), d\text{\$time}, d\text{\$time.sq}) \]

followed by

\[ d\text{.theta} = \text{qr.solve}(X, d\text{\$cases}) \]

to give the regression parameters.