I'll continue with the linear regression example, though not assume a polynomial model. First, some notation. Let β denote our population coefficient vector, and b its estimate. Let ||b|| denote the vector norm of b. Let p denote the number of predictor variables. Again, if we have p predictor variables, we will get a perfect fit in the training set if p = n-1.

If you are familiar with the LASSO, you know that we may be able to do better than computing b to be the OLS estimate; a shrunken version of OLS may do better. Well, which shrunken version? How about the minimum-norm solution?

Before the interpolation point, our unique OLS solution is the famous

$$b = (X'X)^{-1} X'Y$$

This can also be obtained as $b = X^{-} Y$ where X^{-} is a *generalized inverse* of X. It's the same as the classic formula before interpolation, but it can be used after the interpolation point as well. And the key point is then that one implementation of this, the *Moore-Penrose inverse*, **gives the minimum-norm solution**.

This minimum-norm property reduces variance. So, not only will MP allow us to go past the interpolation point, it will also reduce variance, possibly causing the L-C curve to **descend a second time**! We now have a double U-shape (there could be more).

And if we're lucky, in this second U-shape, we may reach a lower minimum than we had in the original one. If so, it will have paid to overfit!

There some subtleties here. The minimum norm means, minimum among all solutions for fixed p. As p increases, the space over which we are minimizing grows richer, thus more opportunities for minimizing the norm. On the other hand, the variance of that minimum norm solution is increasing with p. Meanwhile the bias is presumably staying constant, since all solutions have a training set error of 0. So the dynamics underlying this second U would seem to be different from those of the first.

Empirical illustration:

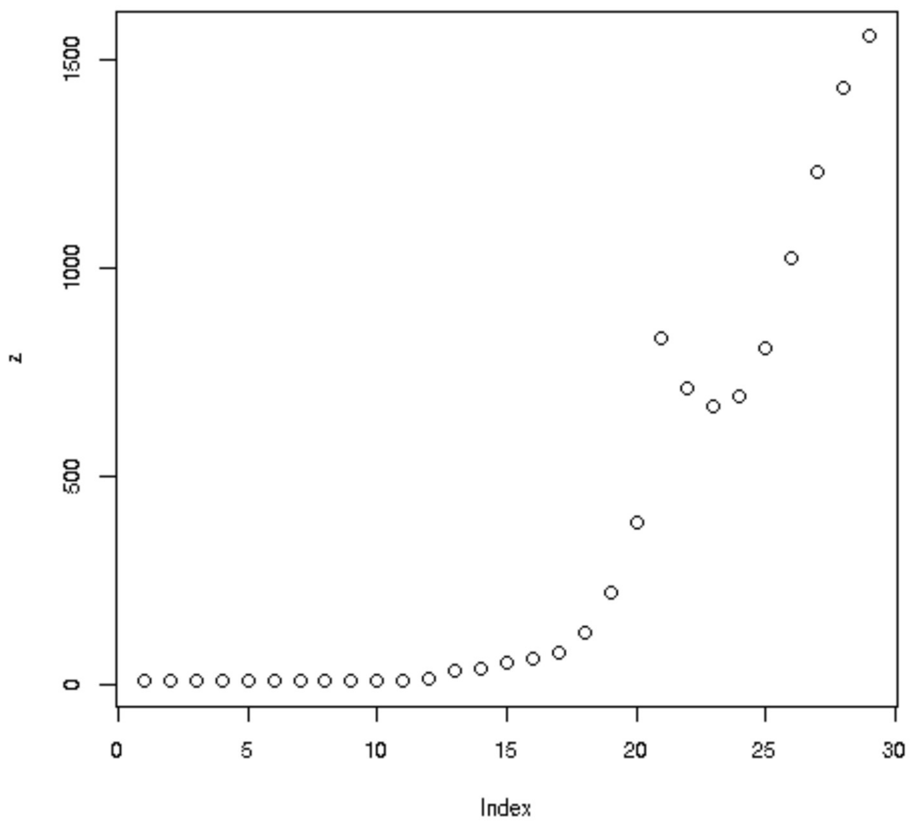
Here I'll work with the Million Song dataset. It consists of 90 audio measurements made on about 500,000 songs (not a million) from 1922 to 2011. The goal is to predict the year of release from the audio.

I took the first p predictors, varying p from 2 to 30, and fit a quadratic model, with $O(p^2)$ predictors resulting.

One of the newer features of regtools is its qe*-series ("Quick and Easy") of functions. They are extremely simple to use, all having the call format **qeX(d,'yname'),** to predict the specified Y in the data frame **d**. Again, the emphasis is on simplicity; the above call is all one needs, so for example there is no preliminary code for defining a model. They are all wrappers for standard R package functiona, and are paired with **predict(**) and **plot()** wrappers.

Currently there are 9 different machine learning algorithms available, e.g **qeSVM(**) and **qeRF()**, for SVM and random forests. Here we will use **qePoly()**, which wraps our **polyreg** package. Note by the way that the latter correctly handles dummy variables (e.g. no powers of a dummy are formed). Note too that **qePoly()** computes the Moore-Penrose inverse if we are past interpolation, using the function **ginv()** from the MASS package.

Again to make this experiment on a meaningful scale, I generated random training sets of size n = 250, and took the rest of the data as the test set. I used from 2 to 30 predictor variables, and used Mean Absolute Prediction Error as my accuracy criterion. Here are the results,



Well, sure enough, there it is, the second U. The interpolation point is between 22 and 23 predictors (there is no “in-between” configuration), where there are 265 parameters, overfitting our n = 250.

Alas, the minimum in the second U is not lower than that of the first, so overfitting hasn’t paid off here. But it does illustrate the concept of double-descent.

Code:

```
overfit <-  function(nreps,n,maxP)
 {
    load('YearData.save')
    nas <- rep(NA,nreps*(maxP-1))
    outdf <- data.frame(p=nas,mape=nas)
    rownum <- 0
    for (i in 1:nreps) {
       idxs <- sample(1:nrow(yr),n)
       trn <- yr[idxs,]
       tst <- yr[-idxs,]
       for (p in 2:maxP) {
          rownum <- rownum + 1
          out<-qePoly(trn[,1:p+1)],
             'V1',2,holdout=NULL)
          preds <- predict(out,tst[,-1])
          mape <- mean(abs(preds - tst[,1]))
          outdf[rownum,1] <- p
          outdf[rownum,2] <- mape
          print(outdf[rownum,])
       }
    }
    outdf  #run through tapply() for the graph
 }
```