Statistics 243: *class notes*

William J. De Meo

October 22, 1997

1 More About the Debugger

The following abreviations are useful when using the debugger:

```
d short
D long
S char*
F double
```

For example, to display the first 20 doubles in an array x, do:

```
&x[0]/20F
call command
```

You might want to write a subroutine which prints a matrix in a nice way. Then you can call it from the debugger:

```
double **;
void printmat(double *x, long nr, long nc);
```

In debugger, do:

```
(dbx) call printmat(x,5,4)
```

Consider making a file called print.c where you put all your print routines.

In the debugger, you can assign a variable a value before resuming program operation:

```
assign variable = value
```

You will get tired of typing cotinue, run, step, etc. so there is an alias facility:

alias name "string", e.g.

```
alias s step
alias n next
alias e print
```

store it in a file in your home directory called .dbxinit.

Another feature available is the source command: `source filename`

Suppose you put a stop in main and a stop in sub1. Then do: `step`

```
printx
12
```

```
cont
⋮
printx
75938492
```

So memory got trashed somewhere around *vdots*. Make an alias:

```
alias m ‘‘step; print x’’
```

and put a bunch of m’s in your file. The output will show you the line where the value of x has changed.

2 Elimination Techniques

Gaussian Elimination
Performs elementary row operations on a matrix containing the left and right hand sides of a system of equations.

Adjust algorithm:

```
for k=1 to p
   b=a_{kk}
   for i=1 to 2p
      a_{ki} = a_{ki}/b  /* if there’s a problem with this step, we’ll know */
   end
   for i=1 to p and i\neq k
      b=a_{ik}
      for j=1 to 2p
         a_{ij} = a_{ij} - b*a_{kj}
      end
   end
end
```

We have been keeping track of more than we need. We will see a more efficient way in a minute. First, consider the augmented matrix

$$\hat{X} = [X : y]_{n\times(p+1)}$$

$$\hat{X}^t\hat{X} = \begin{bmatrix} X^tX & X^ty \\ y^tX & y^ty \end{bmatrix}$$

Adjust algorithm applied to the first p rows transforms this matrix to

$$\begin{bmatrix} I & (X^tX)^{-1}X^ty \\ 0 & y^t(I - X(X^tX)^{-1}X)y \end{bmatrix}$$

and the bottom right entry contains the residual sum of squares. The matrix is now,

$$\begin{bmatrix} I & \hat{\beta} \\ 0 & RSS \end{bmatrix}$$

We modify it to become the sweep(k) algorithm. Operates on one column at a time. The first sweep produces the beta hat in the top right entry that we would get from regressing Y on only the first variable x_1. The cool thing is that if we sweep a few columns, say the first three, then decide we don’t want the second column, resweep the second column – that removes the second columns effect.

2.1 The sweep(k) algorithm

```
let d = a_{kk}
for i=1 to p
a_{ki} = a_{ki}/d
end
 for i=1 to p, i \neq k
 b = a_{ik}
  for j=1 to p
  a_{ij} = a_{ij} - b*a_{kj}
  end
 a_{ik} = -b/d /* key step: the (ik)th element of the inverse is -b/d */
 end
a_{kk}= 1/d /* key step */
```

Now our sweep operator does

$$\begin{bmatrix} X^tX & X^tY \\ y^tX & y^ty \end{bmatrix}$$

we get

$$\begin{bmatrix} (X^tX)^{-1} & (X^tX)^{-1}X^ty \\ -y^tX(X^tX)^{-1} & y^t(I - X(X^tX)^{-1}X)y \end{bmatrix}$$