CHAPTER 8

Introduction to S-Plus

S-Plus is a very powerful program for doing statistics research and data analysis, including the ability to easily generate random numbers, manipulate arrays of various dimensions, and to produce very high quality graphics.

The standard reference for using S-Plus is "An Introduction to S and S-Plus" by Phil Spector (a PhD graduate of our department, now at UB-Berkeley, `spector@stat.berkeley.edu`) published by Duxbury Press in 1994.

Starting, Stopping, and Using S-Plus

To start S-Plus, one need only enter the command `S+` from the Unix prompt. When it starts, the user sees a prompt (a `>` sign). To exit S-Plus and return to the Unix prompt, one enters the command `q()`.

An example of what can be done using S-Plus is (anything following a "pound sign" is a comment and is ignored by S-Plus):

```
> X11()                           # open a graphics window on the screen
> x <- rnorm(100)                 # generate N(0,1) sample of size 100
> x                               # saying the name of a variable or fctn
                                  # causes it to be displayed on the screen
> plot(x,type="l")                # plot x(i) versus i
> postscript("out.ps")            # open a file to get postscript graph
> plot(x,type="l")                # send plot to file
> dev.off(dev.clr())              # close postscript file
> !lpr -Php2 out.ps               # send postscript file to hp2 printer
> X <- matrix(rnorm(1000),100,10) # form 100 by 10 matrix of N(0,1)'s
> XTX <- t(X) %*% X               # get X'X
```

Keeping Track of an S-Plus Session

Every variable or function you define during a session is saved in a special form in a directory called `.Data` somewhere in your directory structure. If you have created such a directory in the directory where you start S-Plus, then the program will use that one. If not, it will create a `.Data` in your home directory or use the one that is already there if there is one. This is a very nice feature as everything you've done is still there the next time you use S-Plus (unless of course you've redefined something).

Subscripting

One of the most important parts of S-Plus is subscripting. Let `X` be a matrix and `A` be a three dimensional array. Then `X[i,j]` is the (i,j)th element of `X`, while `X[i,]` and `X[,j]` are the ith row and jth column respectively. This can be extended to three (and higher) dimensional arrays as well, eg, `A[,,i]` is the ith matrix in the three dimensional array and `A[3,,4]` is the third row of the fourth matrix.

You can pick out parts of arrays more generally by specifying each index you want. For example, if `U` is a vector of U(0,1)'s, you can pick out the elements that are less than 0.25 by saying `U[U<0.25]`, since `U<0.25` is a vector of T's and F's.

There are a million tricks one can learn about subscripting and they are crucial to using S-Plus efficiently.

S-Plus Functions

S-Plus has an on-line help system. To get help about a particular command, enter `help(name)` where `name` is the name of the command. In the following table, we list the names of some of the more useful functions contained in S-Plus.

`function(s)`	Purpose
Distributions: Random numbers (prefix `r`), pdf (prefix `d`), cdf (prefix `p`), and quantile (prefix `q`):	
`beta, binom, cauchy, chisq, exp, f, gamma, geom, lnorm`	
`lnorm, logis, nbinom, norm, pois, t, unif, weibull`	
Numerical operators	
`+, -, *, /, ^`	
`%*%, %/%, %%`	Matrix multiplication, integer divide, modulus
Form arrays	
`c`	`c(x,y,0,4,z)` forms vector
`rep`	`x <- rep(0,100)` gives (0,...,0)
`:`	`1:n` gives (1,...,n); `n:1` gives (n,...,1), etc.
`seq`	`seq(from=0,to=1,by=.1)` or `seq(from=0,to=1,len=11)`
`sample`	`sample(n)` gives random permutation of (1,...,n)
`diag`	Create diagonal matrix or extract diagonal elements
`cbind`	Combine vectors into matrix having the vectors as columns
`rbind`	Combine vectors into matrix having the vectors as rows
`col`	`col(A)` same size as `A` but with column numbers
`row`	`row(A)` same size as `A` but with row numbers
Comparison operators	
`>, <, <=, ==, !=`	
Logical operators	
`&, |, &&, ||, !`	Elementwise and, or; control and, or; unary not
Mathematical functions	
`abs, exp, gamma, lgamma, log, log10, sign, sqrt`	
Trigonometric functions	
`cos, sin, tan, acos, asin, atan, cosh, sinh, tanh, acosh, asinh, atanh`	
Complex number functions	
`Arg, Conj, Mod, Re, Im`	

Rounding and truncating

`round, signif, trunc, ceiling, floor`

Looping

`if, for, while, repeat, next, break`

Sorting and ranking

`sort`	Sort an array
`rev`	Reverse the order of an array
`order`	ith element of `order(x)` is index of ith smallest `x`
`rank`	Return ranks of an array

Functions for simple statistics

`cor, cumsum, mean, median, min, max, prod, range, sum, var`

`quantile`	$X_{(i)} = Q((i-1)/(n-1))$
`sample`	`sample(n)` gives a random permutation of $1, \ldots, n$

Functions for categorical data

`cut`	Create categories from continuous variables
`pretty`	Creates convenient break points from continuous variable
`split`	Break up an array based on value of categorical variable
`table`	Count by categories

Data management

`append, duplicated, match, pmatch, replace, unique`

Matrix or array functions

`nrow, ncol`	Number of rows and columns of matrix
`dim`	Vector of dimensions of a multiway array
`chol`	Cholesky decomposition
`crossprod`	Kronecker product
`eigen`	Eigenanalysis
`outer`	Outer product of two vectors
`scale`	Scale the columns of a matrix
`solve`	Solve system of equations or invert a matrix
`qr`	QR Orthogonalization
`t`	Matrix transpose

Character manipulation

`abbreviate`	Generate abbreviations of character values
`cat`	Display values on screen or send to file
`grep`	Search for patterns in characters
`nchar`	Number of characters in a string
`paste`	Put numbers into character strings
`substring`	Extract parts of character values

Statistical inference

```
binom.test, chisq.test, cor.test, fisher.test, friedman.test, kruskal.test
mantelhaen.test, mcnemar.test, prop.test, t.test, var.test, wilcox.test
```

Statistical Modeling

`lm`	Linear models and regression
`aov`	Analysis of variance

`glm`	Generalized linear models
`gam`	Generalized additive models
`loess`	Local regression models
`tree`	Tree-based models
`nls`	Nonlinear regression

Time series analysis

Multivariate analysis

Reading from files

`scan, dget`

Writing to screen or printer

`print, sink, cat, round, paste, format, write, data.dump, data.restore`

Open graphics devices

`X11, motif, tek4014, postscript` First two for workstations, next for over modem, last for printer

Manipulating graphics devices

`dev.cur, dev.list, dev.next, dev.prev, dev.off, dev.set`

Graphics parameters

High level graphics functions

`barplot, boxplot, contour, hist, pie, plot, qqnorm, qqplot, tsplot, usa`

`coplot`	Separate plots for different ranges
`dotchart, faces`	Plot multivariate data
`pairs`	Scatterplos for all possible pairs of columns in a matrix
`persp`	3-D perspective plot
`plclust`	Plot of cluster trees from `hlclust`

Low level graphics (add things to existing plot)

`lines, points, segments, polygon, arrows, symbols, text`
`axis, axes, box, legend, mtext, title`

`tslines, tspoints`	Lines and points on time series plots
`abline`	Add regression line to plot
`perspp`	Project 3-D points to 2-D points to be used with `persp`
`frame`	Advance to next figure
`labclust`	Add labels to cluster plot
`stamp`	Add time stamp to a plot

An Example

I have files called `oz80.dat`, `oz81.dat`, and so on, up to `oz93.dat`, which contain hourly readings of ozone concentrations for one of the years 1980 through 1993 at between 11 and 13 monitoring stations around Harris County, Texas. Each line in each file starts with the numbers of the month, day, and hour for that line's observation, followed by the ozone level at the different stations. Thus the file for 1993 (a non-leap year) contains 8,760 lines (365 days times 24 hours per day). In 1993 there were 11 monitoring stations active. To read the 8,760 by 14 matrix, we would say

```
oz <- matrix(scan("oz93.dat"),8760,14,byrow=T)
```

The `scan` function reads all 8,760 times 14 values in the file `oz93.dat` into a single vector (it will read across the lines) and then the `matrix` function will put the elements of that vector into the 8760 by 14 matrix `oz` "by row."

If a monitoring site was malfunctioning at any time, the value 8888 or 9999 was inserted into the data file. Thus to count the number of missing values at each site and display them on the screen, we could do

```
for(i in 1:11) print(sum(oz[,i+3]>=8888))
```

which shows the power and terseness of S-Plus syntax. The `oz[,i+3]` will form a vector containing the `i+3`rd column of `oz` (remember that the first three columns of `oz` are month, day, hour). Then `oz[,i+3]>=8888` will form a vector of `T`'s and `F`'s corresponding to elements that are greater than or equal to 8888 (ie, missing) or less than 8888 (not missing). Then applying the `sum` function to that vector of `T`'s and `F`'s will treat the `T`'s as 1's and the `F`'s as 0's, and thus we get the number of missing values.

Arrays Are Stored By Column

If you define a matrix by `X <- matrix(1:9,3,3)`, you get

```
  1  4  7
  2  5  8
  3  6  9
```

Similarly, if you do `A <- array(1:16,c(2,2,4))` the four matrices are

```
 a[,,1]    A[,,2]    A[,,3]     A[,,4]
  1  3      5  7      9  11     13  15
  2  4      6  8     10  12     14  16
```

Writing Your Own Functions

An important part of using S-Plus is writing our own functions. If we include the following code in a file (maybe called `myfuncs.S`)

```
plot4 <- function(x)
{
  par(mfrow=c(2,2))        # divide graphics window into 2 by 2 regions
  hist(x)                  # put histogram in upper left
  boxplot(x)               # boxplot in upper right
  f <- density(x)          # get kernel density estimate
  plot(f$x,f$y,type="l")   # plot it
  qqnorm(x)                # qq plot in lower left
}
```

and we said `source("myfuncs.S")` from the S-Plus prompt, the fucntion `plot4` would be available from then on to be used anytime we used S-Plus.

In writing functions, you need to keep the following points in mind:

1. Note the structure of a function. The first line has the function name followed by the `<-`, followed by a list of arguments. The "body" of the function is contained within matching left and right curly brackets.

2. A function can return a single object (which often is a "list," see the next section) "in its name." There is a `return` statement (see the example below) but the last thing defined in the function is returned by default.

3. If an argument is changed in the function, its value in the calling function is not changed.

4. Arguments can have default values, eg, `rnorm <- function(n,mean=0,sd=1)` can be used by saying `x <- rnorm(100)` or `x <- rnorm(100,5)` to get $\mu = 5$ and $\sigma = 1$, or `x <- rnorm(100,sd=4)` to get $\mu = 0$ and $\sigma = 4$.

The Concept of a List

Like most programming languages (other than fortran), S-Plus has the ability to group together into one logical structure several variables. The structure in S-Plus is called a "list." For example, if we write the following function

```
myreg <- function(X,y)
{
  beta   <- solve(t(X) %*% X, t(X) %*% y)
  fitted <- X %*% beta
  res    <- y - fitted
  ssq    <- sum(res * res) / (nrow(X) - ncol(X))
  Rsq    <- 1 - var(res)/var(y)
  tstat  <- beta / sqrt(ssq * diag(solve(t(X) %*% X)))
  pval   <- 2 * pt(1 - abs(tstat))
  return(list(coeff=beta, yhat=fitted, res=res, ssq=ssq, Rsq=Rsq,
              tstat=tstat, pval=pval))
}
```

and we have the Hald Data on a file called `hald1.dat` without the title and n and m lines, we could say

```
  zz      <- scan("hald1.dat",list(y=0,x1=0,x2=0,x3=0,x4=0))
  n       <- length(zz$y)
  haldout <- myreg( cbind( rep(1,n),zz$x1,zz$x2,zz$x3,zz$x4)), zz$y)
```

and then, for example, we could plot the residuals by saying

```
plot(haldout$res)
```

Interfacing with C or Fortran

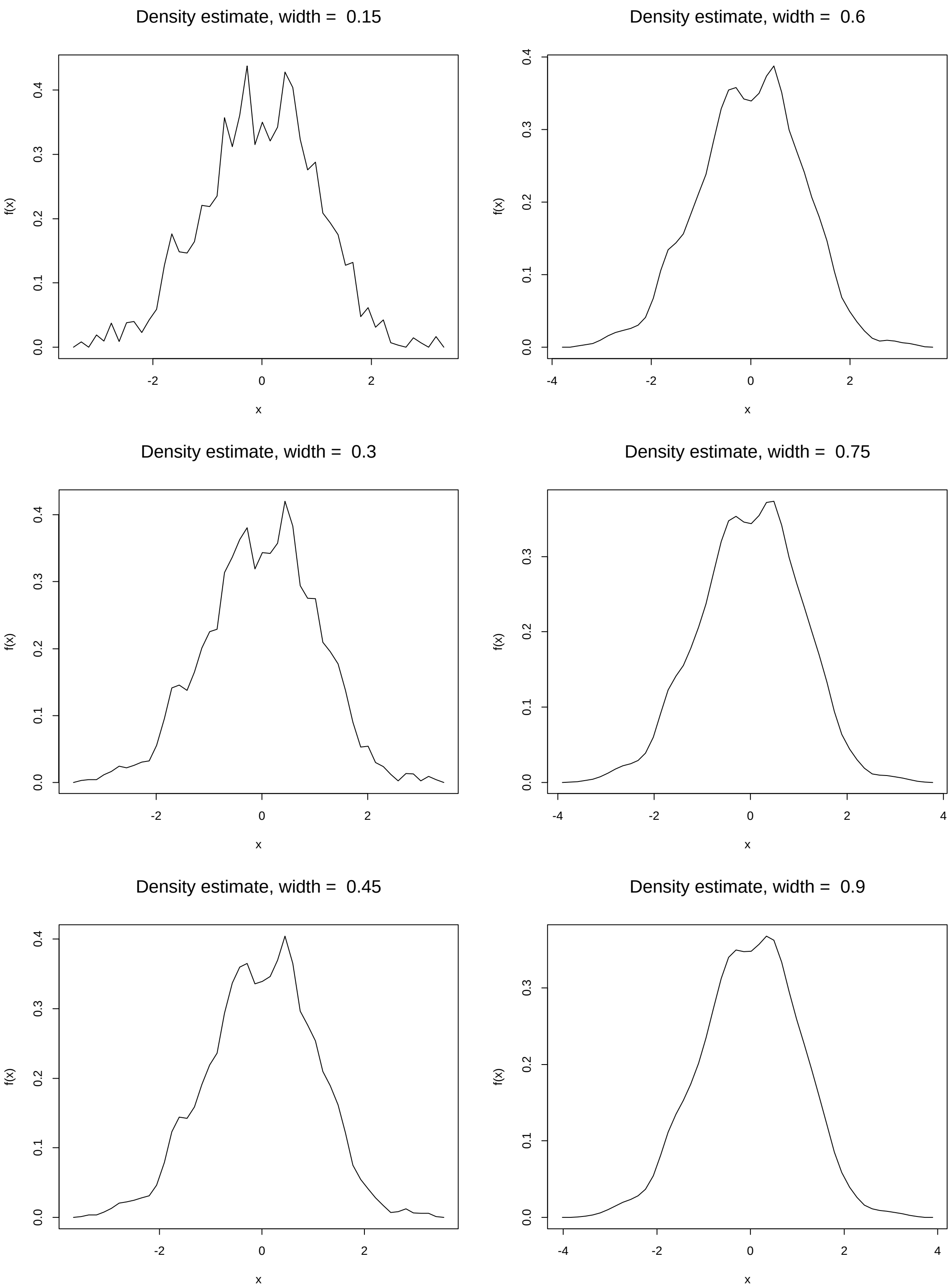
It is also possible to create our own functions in Fortran or C (see pg. 130 of Spector's book for example).

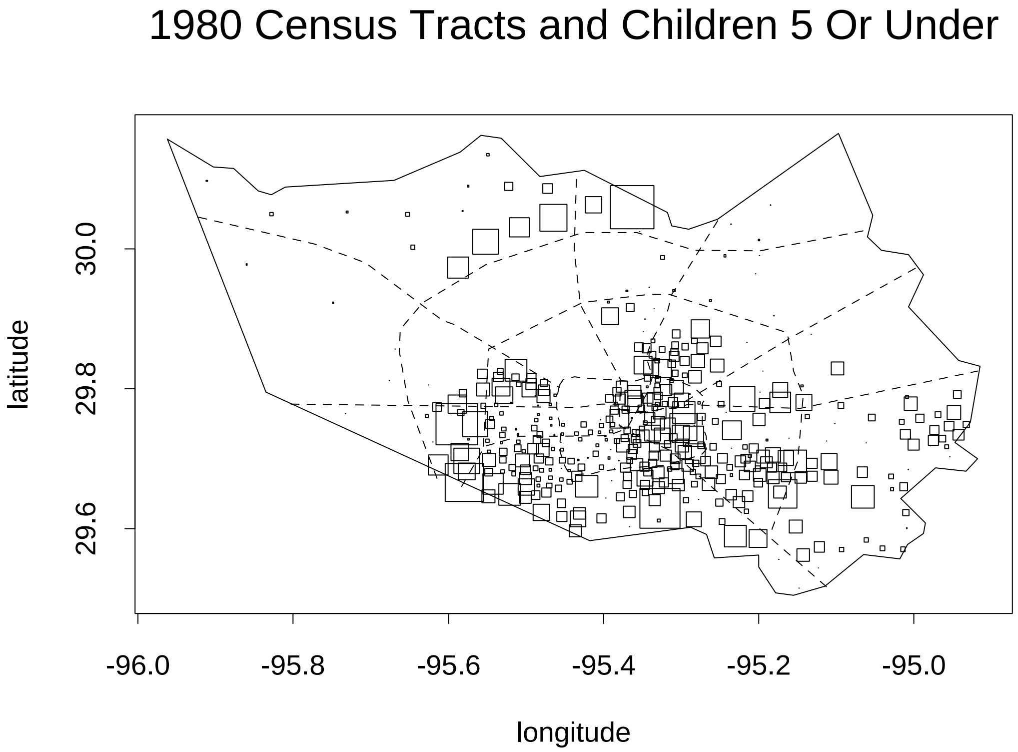
Two Graphics Examples

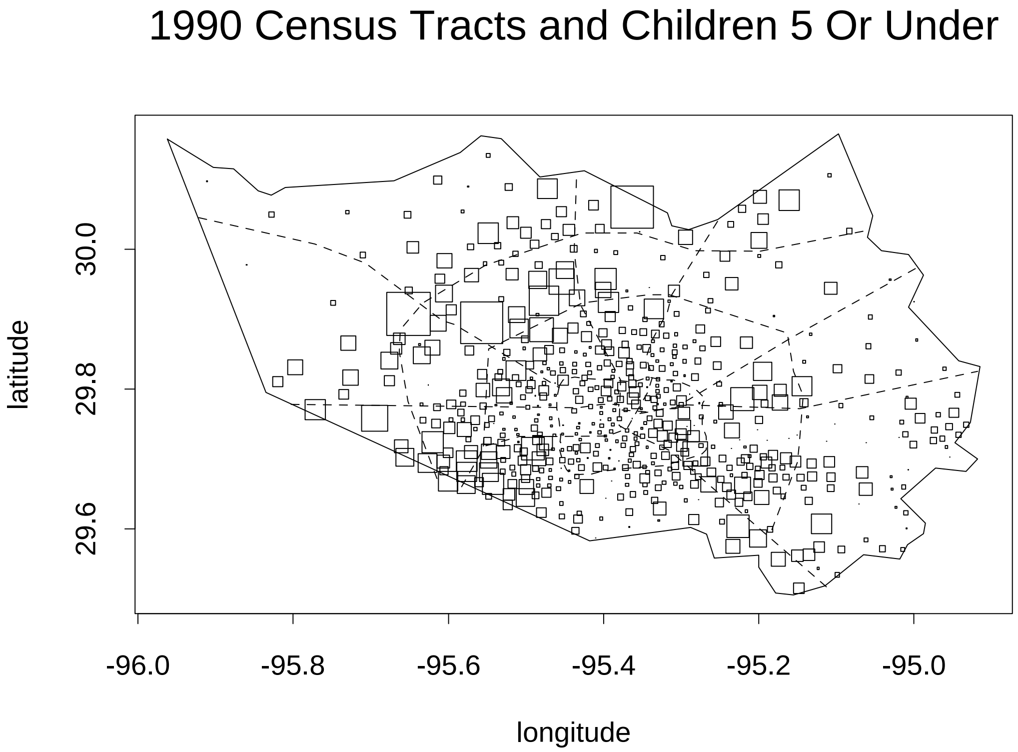
Kernel Density Estimate

The following function produces the page of density estimate plots.

```
plot.dens <- function()
{
  par(mfcol=c(3,2))
  x <- rnorm(1000)
  for(i in 1:6) {
    plot(density(x,width=.15*i),type='l',xlab='x',ylab='f(x)',
         main=paste("Density estimate, width = ",round(.15*i,3)))
  }
}
```


Density estimate, width = 0.15
Density estimate, width = 0.6
Density estimate, width = 0.3
Density estimate, width = 0.75
Density estimate, width = 0.45
Density estimate, width = 0.9
f(x)
x


1980 Census Tracts and Children 5 Or Under
latitude
30.0
29.8
29.6
-96.0
-95.8
-95.6
-95.4
-95.2
-95.0
longitude


1990 Census Tracts and Children 5 Or Under
latitude
30.0
29.8
29.6
-96.0
-95.8
-95.6
-95.4
-95.2
-95.0
longitude

Census Data Example

The code given below produces the plot given on the previous page. Here are the first ten lines of the three files the example uses:

```
First 10 lines of kids80.dat
-----------------------------
 121.00    29.7576  -95.3633   69.
 201.01    29.7729  -95.3339  295.
 201.02    29.7655  -95.3277  622.
 202.00    29.7567  -95.2899 1098.
 203.01    29.7807  -95.3107  464.
 203.02    29.7835  -95.2997  707.
 203.03    29.7774  -95.2997  286.
 204.00    29.7785  -95.3211  260.
 205.01    29.7865  -95.3433  511.
 205.02    29.7850  -95.3351  664.

First 10 lines of kids90.dat
-----------------------------
 12100  29.7576 -95.3633    12
 20101  29.7729 -95.3339   273
 20102  29.7655 -95.3277   502
 20210  29.7657 -95.2966   833
 20220  29.7477 -95.2833     1
 20301  29.7807 -95.3107   337
 20302  29.7835 -95.2997   689
 20303  29.7774 -95.2997   312
 20400  29.7785 -95.3211   172
 20501  29.7865 -95.3433   340

First 10 lines of roads.dat
-----------------------------
-95.962 30.157
-95.835 29.795
-95.418 29.583
-95.288 29.602
-95.267 29.592
-95.257 29.558
-95.200 29.562
-95.200 29.545
-95.178 29.508
-95.155 29.505
```

```
plot.census <- function()
{

  kids80 <- scan("kids80.dat",list(tract=0,y=0,x=0,kids=0))
  kids90 <- scan("kids90.dat",list(tract=0,y=0,x=0,kids=0))

  par(mfrow=c(2,1),mar=c(4,4,4,4))

  drawroads(main="1980 Census Tracts and Children 5 Or Under")

  symbols(kids80$x,kids80$y,squares=kids80$kids,add=T,inches=.3)

  drawroads(main="1990 Census Tracts and Children 5 Or Under")

  symbols(kids90$x,kids90$y,squares=kids90$kids,add=T,inches=.3)

}

drawroads <- function(lty=3,cex=1.0,main="")
{

  nl <- c(47,11,13,11,8,6,11,40,6)

  nstart <- c(1,cumsum(nl)[1:8]+1)    # 1,48,59,etc.
  nend   <- c(nstart[2:9]-1,sum(nl))  # 47,58,etc

  plot(roads$x[nstart[1]:nend[1]],roads$y[nstart[1]:nend[1]],type="l",
       xlab="longitude",ylab="latitude",cex=cex,main=main)

  for(i in 2:9) lines(roads$x[nstart[i]:nend[i]],
                      roads$y[nstart[i]:nend[i]],lty=lty)

}
```

```
read.roads <- function()
{

  roads <- scan("roads.dat",list(x=0,y=0))

  return(roads)

}
```