Iteration, Likelihood, and All That

For Loops!

```
x<-1:100000
#
for (i in x){
  x[i]<-x[i] +1
}
```

For Loops: Costs & Benefits

Benefits:

1. Minimal code for repetitive actions
2. Can map same operation across a vectore, matrix, list, etc.

Costs:

1. Slow.
2. Lots of Code.

Speed: Many Operations Faster via *Vectorization*

```
1:10 + 1

#  [1]  2  3  4  5  6  7  8  9 10 11
```

Many Operations Faster via *Vectorization*

```
system.time(1:100000+1)

#    user  system elapsed
#   0.001   0.000   0.001


system.time({
  x<-1:100000
  for(i in x) x[i]<-x[i] +1
})

#    user  system elapsed
#   0.335   0.010   0.346
```

Vectorization Ubiquitos

```
dnorm(5, mean = 1:10, sd = 1)

#  [1] 1.338e-04 4.432e-03 5.399e-02 2.420e-01 3.989e-01
#  [6] 2.420e-01 5.399e-02 4.432e-03 1.338e-04 1.487e-06
```

Vectorization Ambiguous in Many Instances...

```
sampMean <- function (vec, size) mean(sample(vec, size))

sampMean(vec=1:5, size=1:5)

# [1] 1
```

How to Vectorize Ambiguous Functions

```r
sampMeanV <- Vectorize(sampMean,
                       vectorize.args="size")

sampMeanV(vec=1:5, size=1:5)

# [1] 3.000 3.000 3.333 3.000 3.000
```

The Guts of Vectorize

```
sampMeanV

# function (vec, size)
# {
#     args <- lapply(as.list(match.call())[-1L], eval, parent.frame())
#     names <- if (is.null(names(args)))
#         character(length(args))
#     else names(args)
#     dovec <- names %in% vectorize.args
#     do.call("mapply", c(FUN = FUN, args[dovec], MoreArgs = list(args[
#         SIMPLIFY = SIMPLIFY, USE.NAMES = USE.NAMES))
# }
# <environment: 0x1034f21f8>
```

Minimizing Code for Mapping Functions: the Apply Family

Take an object type - vector, matrix, list, etc., and map a function to every element, cleanly and quickly.

sapply for Vectors

```r
f <- function(x) x+1
sapply(1:5,f)

# [1] 2 3 4 5 6
```