

R Tutorial 2

Econ 520

1. Vectors

In R, a variable can hold a vector of numbers. (You can also define matrices and arrays, as we'll see later on.) This is very handy because we can do a set of operations simultaneously on a large set of numbers.

An easy way to construct a vector is to use the "c" command. This "concatenates" numbers into a vector:

```
> x<- c(1,3,5,10)
```

This defines a vector (1,3,5,10) and stores the entire vector in x. If you just type "x" at the prompt, you will see the entire vector:

```
> x
[1] 1 3 5 10
```

Many arithmetic operations work element-by-element on vectors. For example:

```
> x
[1] 1 3 5 10
> x+1
[1] 2 4 6 11
> 3*x
[1] 3 9 15 30
```

Also some functions work element-by-element on vectors:

```
> exp(x)
[1] 2.718282 20.085537 148.413159 22026.465795
> log(x)
[1] 0.000000 1.098612 1.609438 2.302585
> sin(x)
[1] 0.8414710 0.1411200 -0.9589243 -0.5440211
```

Some functions in R work directly on a vector to return a scalar. For example:

```
> sum(x)
[1] 19
> mean(x)
[1] 4.75
> sd(x)
[1] 3.86221
```

These calculate the sum of the elements in x, the mean (average) of the elements, and the (sample) standard deviation.

Another useful type of calculation involves logical vectors. A logical vector is a vector where each element can be either "TRUE" or "FALSE." Logical vectors can be constructed using the concatenation function:

```
> lv1 <- c(FALSE, TRUE, FALSE, TRUE)
> lv1
[1] FALSE TRUE FALSE TRUE
```

or by applying a logical test to a vector, for example:

```
> x
[1] 1 3 5 10
> lv2 <- x > 4
> lv2
[1] FALSE FALSE TRUE TRUE
```

Here, lv2 is a vector of the same size as x, with FALSE indicating that the corresponding element of x is not strictly greater than 4. (For weak inequality, you can use <= and >=)

Two logical vectors of the same length can be combined using logical operators (& for "and", | for "or"):

```
> lv1
[1] FALSE TRUE FALSE TRUE
> lv2
```

```
[1] FALSE FALSE TRUE TRUE
```

```
> lv1 & lv2
```

```
[1] FALSE FALSE FALSE TRUE
```

```
> lv1 | lv2
```

```
[1] FALSE TRUE TRUE TRUE
```

Some numerical functions can be applied to logical vectors:

```
> sum(lv2)
```

```
[1] 2
```

2. Indexing vectors

Recall that we defined:

```
> x
```

```
[1] 1 3 5 10
```

We can access elements of the vector individually, by using `x[n]`, where `n` is the position of the element in the vector:

```
> x[1]
```

```
[1] 1
```

```
> x[2]
```

```
[1] 3
```

```
> x[4]
```

```
[1] 10
```

Notice that `x[1]` is the first element in the vector `x`. (In some programming languages, vector indices start at 0, but in R, they start at 1.)

We can take subvectors using the ":" notation:

```
> x[1:3]
```

```
[1] 1 3 5
```

We can also reorder the elements. Here, "reverseindex" is a vector containing the numbers 1 through 4, but in reverse sequence. Writing `x[reverseindex]` asks R to construct the vector (`x[4]`, `x[3]`, `x[2]`, `x[1]`):

```
> reverseindex <- c(4,3,2,1)
```

```
> x[reverseindex]
```

```
[1] 10 5 3 1
```

Another way to extract subvectors is using a logical index vector:

```
> logindex <- x > 4
```

```
> logindex
```

```
[1] FALSE FALSE TRUE TRUE
```

```
> x[logindex]
```

```
[1] 5 10
```

This creates a vector of those elements of `x` that are greater than 4.

3. Random Numbers

R has extensive facilities for working with random numbers and probability distributions.

Suppose we are interested in the Uniform(0,4) distribution. The function `dunif(x,min,max)` gives the PDF of a uniform distribution with lower bound equal to `min`, upper bound equal to `max`, evaluated at `x`:

```
> dunif(.5, min=0, max=4)
```

```
[1] 0.25
```

The function `pnunif(x, min, max)` gives the probability of being less than or equal to `x` (i.e., the CDF):

```
> pnunif(.5, min=0, max=4)
```

```
[1] 0.125
```

Finally, the function `runif(n, min, max)` generates `n` (independent) random draws from the Uniform(`min`,`max`) distribution. Since we don't necessarily want to view all of them on the screen, we can store them in a vector variable for later use:

```
> Udraws <- runif(100,min=0,max=4)
```

```
> mean(Udraws)
[1] 1.882392
```

Notice that `mean(Udraws)` gives the sample average of the random draws in `Udraws`. This should be fairly close to the expected value of 2. We see that the mean is 1.88, a little off from the expectation. If we had taken, say, 100,000 draws instead of just 100, we would likely be closer:

```
> Udraws2 <- runif(100000, min=0, max=4)
> mean(Udraws2)
[1] 1.990044
```

R has similar functions available for various families of distributions. For example, `dnorm(x,mean, sd)` gives the PDF of a normal random variable with a certain mean and a certain standard deviation, `rnorm` generates normal random variables, and so on. Other families of probability distributions in R include:

- Beta (`dbeta`, `pbeta`, `rbeta`)
- Binomial (`dbinom`)
- Cauchy (`dcauchy`)
- Chi-squared (`dchisq`)
- Exponential (`dexp`)
- Gamma (`dgamma`)
- Geometric (`dgeom`)
- Log Normal (`dlnorm`)
- Poisson (`dpois`)