Week 2 – R Data Modeling

Why functions?

- Data structures tie related values into one object.
- Functions tie related commands into one object.
- In both cases: easier to understand, easier to work with, easier to build into larger things

For example, here is an Excel look-alike NPV function

```
# Net Present Value function Inputs:
# vector of rates (rates) with 0 as
# the first rate for time 0, vector
# of cash flows (cashflows) Outputs:
# scalar net present value
NPV.1 <- function(rates, cashflows) {
    NPV <- sum(cashflows/(1 + rates)^(seq_along(cashflows) -
        1))
    return(NPV)
}
```

Generate data internal to the function

- Use `seq_along` to generate time index of cashflows.
- Be sure to subtract 1 from this sequence as starting cashflow is time 0.

Our functions get used just like the built-in ones:

```r
rates <- c(0, 0.08, 0.06, 0.04)  # first rate is always 0.00
cashflows <- c(-100, 200, 300, 10)
NPV.1(rates, cashflows)
```

```
## [1] 361.0741
```

Go back to the declaration and look at the parts:

```r
# Net Present Value function Inputs:
# vector of rates (rates) with 0 as
# the first rate for time 0, vector
# of cash flows (cashflows) Outputs:
# scalar net present value
NPV.1 <- function(rates, cashflows) {
    NPV <- sum(cashflows/(1 + rates)^(seq_along(cashflows) -
        1))
    return(NPV)
}
```

Interfaces: the **inputs** or **arguments**; the **outputs** or **return value**

- Calls other functions `sum`, `seq_along()`, operators `/`, `+`, `^` and `-` . could also call other functions we've written
- `return()` explicitly says what the output is: good documentation . alternately, return the last evaluation; explicit returns are better documentation

Comments: Not required by R, but always a Good Idea

One-line description of purpose - Listing of arguments - Listing of outputs

What should be a function?

- Things you're going to re-run, especially if it will be re-run with changes
- Chunks of code you keep highlighting and hitting return on
- Chunks of code which are small parts of bigger analyses
- Chunks which are very similar to other chunks

Named and default arguments

```
# Internal Rate of Return (IRR)
# function Inputs: vector of cash
# flows (cashflows), scalar
# interations (maxiter) Outputs:
# scalar net present value
IRR.1 <- function(cashflows, maxiter = 1000) {
    t <- seq_along(cashflows) - 1
    # rate will eventually converge to
    # IRR
    f <- function(rate) (sum(cashflows/(1 +
        rate)^t))
    # use uniroot function to solve for
    # root (IRR = rate) of f = 0 c(-1,1)
    # bounds solution for only positive
    # or negative rates select the root
    # estimate
    return(uniroot(f, c(-1, 1), maxiter = maxiter)$root)
}
```

Default argument

- `maxiter` controls the number of iterations.
- We can eliminate this argument if we want (perhaps at our peril!)

```r
# Here are the cashflows for a 3\%
# coupon bond bought at a hefty
# premium
cashflows <- c(-150, 3, 3, 3, 3, 3, 3,
    3, 103)
IRR.1(cashflows)
```

```
## [1] -0.02554088
```

```r
IRR.1(cashflows, maxiter = 100)
```

```
## [1] -0.02554088
```

Negative Interest Rates

- We get a negative IRR or yield to maturity on this net present value $= 0$ calculation.

Shoot the trouble

Problem: Odd behavior when arguments aren't as we expect

```
NPV.1(c(0.1, 0.05), c(-10, 5, 6, 100))
```

```
## [1] 86.10434
```

We do get a solution, but...

- What does it mean? What rates correspond with what cashflows?
- *Solution*: Put sanity checks into the code.
- Use `stopifnot(some logical statment)` is TRUE.

```r
# Net Present Value function Inputs:
# vector of rates (rates) with 0 as
# the first rate for time 0, vector
# of cash flows (cashflows), length
# of rates must equal length of
# cashflows Outputs: scalar net
# present value
NPV.2 <- function(rates, cashflows) {
    stopifnot(length(rates) == length(cashflows))
    NPV <- sum(cashflows/(1 + rates)^(seq_along(cashflows) -
        1))
    return(NPV)
}
```

`stopifnot` TRUE error handling

- Arguments to `stopifnot()` are a series of logical expressions which should all be TRUE.
- Execution halts, with error message, at *first* FALSE.

```
NPV.2(c(0.1, 0.05), c(-10, 5, 6, 100))
```

Hit (not too hard!) the `Escape` key on your keyboard

This will take you out of `Browse[1]>` mode and back to the console prompt `>`.

What the function can see and do

- Each function has its own environment.
- Names here override names in the global environment.
- Internal environment starts with the named arguments.
- Assignments inside the function only change the internal environment.
- Names undefined in the function are looked for in the environment the function gets called from.

Try this . . .

Your company is running a £100 project in the EU. You must post 25% collateral in a Landesbank using only high-quality government securities. You find a high-quality gilt fund that will pay 1.5% (coupon rate) annually for three years.

Questions

1. How much would you pay for this collateral if the rate curve (yield to maturity of cash flows) is (from next year on. . .)

```r
rates <- c(-0.001, 0.002, 0.01)
```

2. Suppose a bond dealer asks for 130% of notional collateral value for this bond. What is the yield on this transaction (IRR)? Would you buy it?
3. What is the return on this collateral if you terminate the project in one year and liquidate the collateral (i.e., sell it for cash) if the yield shifts down by 0.005? This is a "parallel" shift, which is finance for: "take each rate and deduct 0.005."

Thinking. . .

Some answers

Build rates and cash flows across the 3-year time frame:

```
(rates <- c(0, rates))
```

```
## [1]  0.000 -0.001  0.002  0.010
```

```
collateral.periods <- 3
collateral.rate <- 0.25
collateral.notional <- collateral.rate *
    100
coupon.rate <- 0.015
cashflows <- rep(coupon.rate * collateral.notional,
    collateral.periods)
cashflows[collateral.periods] <- collateral.notional +
    cashflows[collateral.periods]
(cashflows <- c(0, cashflows))
```

```
## [1]  0.000  0.375  0.375 25.375
```

What just happened. . .

1. Append a `0` to the rate schedule so we can use the `NPV.2` function.
2. Parameterize the term sheet (terms of the collateral transaction),
3. `rep()` coupon cash flows.
4. Add notional value repayment to the last cash flow.

Find the present value of the bond using `NPV.2`:

```r
(Value.0 <- NPV.2(rates, cashflows))
```

```
## [1] 25.3776
```

The answer is £25.378

or `Value.0 / collateral.notional` times the notional value.

The yield to maturity averages the forward rates across the bond cash flows. This is one interpretation of the Internal Rate of Return ("IRR").

```
cashflows.IRR <- cashflows
collateral.ask <- 130
cashflows.IRR[1] <- -(collateral.ask/100) *
    collateral.notional
# mind the negative sign!
(collateral.IRR.1 <- IRR.1(cashflows.IRR))
```

```
## [1] -0.07112366
```

You end up paying over 7% per annum for the privilege of owning this bond! You call up the European Central Bank, report this overly hefty haircut on your project. You send out a request for proposal to other bond dealers. They come back with an average asking price of 109 (109% of notional).

```
cashflows.IRR <- cashflows
collateral.ask <- 109
cashflows.IRR[1] <- -(collateral.ask/100) *
    collateral.notional
(collateral.IRR.1 <- IRR.1(cashflows.IRR))
```

```
## [1] -0.01415712
```

That's more like it: about 140 basis points (1.41% x 100 basis points per percentage) cost (negative sign).

Unwind the project, and the collateral transaction, in 1 year. Let's suppose the yield curve in 1 year has parallel shifted down by 0.005.

```
rate.shift <- -0.005
rates.1 <- c(0, rates[-2]) + rate.shift
cashflows.1 <- c(0, cashflows[-2])
(Value.1 <- NPV.2(rates.1, cashflows.1))
```

```
## [1] 25.37541
```

```
(collateral.return.1 <- Value.1/(-cashflows.IRR[1]) -
    1)
```

```
## [1] -0.0687923
```

Looks much more than a break-even return on the collateral transation:

```
(collateral.gainloss <- collateral.notional *
    collateral.return.1) * 1e+06
```

```
## [1] -1719807
```

```
# adjust for millions of euros
```

That's probably someone's salary. . . (in pounds sterling).

Whitman
SCHOOL of MANAGEMENT
SYRACUSE UNIVERSITY

MBA@SYRACUSE

Mind the Interface!

- Interfaces mark out a controlled inner environment for our code;
- Interact with the rest of the system only at the interface.
- Advice: arguments explicitly give the function all the information.
  - Reduces risk of confusion and error
  - Exception: true universals like π
- Likewise, output should only be through the return value. More about breaking up tasks and about environments later

Further reading:

Herbert Simon, *The Sciences of the Artificial*

Whitman
SCHOOL *of* MANAGEMENT

SYRACUSE UNIVERSITY

MBA@SYRACUSE