Programming for FinTech: Finance Problems

Module 1: R Programming

Prof. Matthew G. Son

Vectorized operations

Q1. Present Value

You are given a series of future cash flows from an investment as follows (in dollars):

```
cash_flows <- c(500, 700, 800, 1000, 1200)
```

- The cash flows occur at the end of each year (today is $t = 0$ and `cash_flow[[1]]` is at $t = 1$).
- The annual interest rate (discount rate) is 5%.

Generate below to calculate present value vector.

```
years <- 1:5
r <- 0.05
present_values <- cash_flows / (1 + r)^years
```

Q1-1. What is the length of input vectors, `cash_flows`, `years`, `r`? Check them programmatically.

Q1-2. How was it possible to calculate present value though inputs are not in the same length?

Functions

Net Present Value

Q3. NPV calculator: variable cash flows

NPV calculation formula is expressed as:

$$NPV = CF_0 + \sum_{t=1}^{N} \frac{CF_t}{(1+r)^t}$$

Write a `npv_calculator()` function that takes three inputs and calculates NPV:

- `cf0`: cash flow today
- `cash_flows`: a vector of cash flows occur at the end of each year
- `rate`: interest rate

Confirm your `npv_calculator()` works by replicating below result:

```
npv_calculator(
  cf0 = 150,
  cash_flows = c(50, 150, 200, 100),
  rate = 0.05
  ) # 561.3512
```

Q4. NPV calculator: flat cash flows

Based on the formula given below, write a function named `npv_growing_annuity()` that:

$$NPV = CF_0 + \sum_{t=1}^{N} \frac{CF_1(1+g)^{t-1}}{(1+r)^t}$$

- Takes `cf0`, `cf1`, `growth_rate`, `n_years`, and `rate` as arguments.
- The default value of `cf0` and `growth_rate` is set to zero.

> **Tip**
>
> Utilize `years <- 1:n_years` and `years - 1` in your function.

Confirm your `npv_growing_annuity()` works by replicating below results:

```r
npv_growing_annuity(
  cf1 = 300,
  n_years = 5,
  rate = 0.05
) # 1298.843
```

```r
npv_growing_annuity(
  cf0 = -1000,
  cf1 = 300,
  n_years = 5,
  growth_rate = 0.04,
  rate = 0.05
) # 401.6185
```

Cost of Equity (Gordon Dividend Model)

Q5. Simple Gordon model

Write a function named `gorden_coe()` with arguments `div0`, `p0`, `g`. Confirm that you replicate below result.

Basic Gordon dividend discount model is:

$$P_0 = \Sigma_{t=1}^{\infty} \frac{Div_0 * (1+g)}{r_E - g} = \frac{Div_0 * (1+g)}{(1+r_E)} + \frac{Div_0 * (1+g)^2}{(1+r_E)^2} + ...$$

That is:

$$r_E = \frac{Div_0 * (1+g)}{P_0} + g$$

- provided $g < r_E$

```r
gordon_coe(div0 = 3, p0 = 130, g = 0.07)
```

```
[1] 0.09469231
```

Function + Vectorized Operations

Q6. Black-Scholes Option Pricing

The Black-Scholes formula for a European call and put option is given by:

$$C = S_0 \Phi(d_1) - K e^{-rT} \Phi(d_2)$$
$$P = K e^{-rT} \Phi(-d_2) - S_0 \Phi(-d_1)$$

where

$$d_1 = \frac{\ln(S_0/K) + \left(r + \frac{\sigma^2}{2}\right) T}{\sigma \sqrt{T}}$$
$$d_2 = d_1 - \sigma \sqrt{T}$$

Below is the bsm pricing function from above equations.

```r
bsm_price <- function(S0, K, r, T, sigma, type = "call") {
  d1 <- (log(S0 / K) + (r + 0.5 * sigma^2) * T) / (sigma * sqrt(T))
  d2 <- d1 - sigma * sqrt(T)

  if (type == "call") {
    return(S0 * pnorm(d1) - K * exp(-r * T) * pnorm(d2))
  } else if (type == "put") {
    return(K * exp(-r * T) * pnorm(-d2) - S0 * pnorm(-d1))
  } else {
    stop("Invalid option type. Use 'call' or 'put'.")
  }
}
```

Now, imagine you are analyzing the following options:

Option ID	Stock Price (S0)	Strike Price (K)	Risk-Free Rate (r)	Time to Maturity (T, years)	Volatility (sigma)	Option Type
1	100	100	0.05	1	0.20	call
2	105	100	0.05	0.5	0.25	call
3	110	100	0.05	2	0.30	put

Use above function to calculate BSM price estimates.

Control Structure

Under construction

Regressions

Under construction

Cost of Equity (CAPM)

Textual analysis : newspaper headlines