Session 2
Sep. 5, 2019

0.2 Scientific Python

- optimized for vector/matrix computations (near machine speed although interpreted language)
- functionality comprises all aspects of scientific computing

ToDo:

Install Anaconda package (version 3.x) (Windows, Linux, Apple)

↳ SciPy package included
↳ spyder editor (development environment)

Code examples in git folder

1. Basics of Financial Math

1.1 Time Value of Money

r = annual interest rate

FV = future value, PV = present value

after n years: $FV = PV(1+r)^n$

$PV = FV(1+r)^{-n}$

if interest is compounded m times per year: $FV = PV \cdot \left(1+\frac{r}{m}\right)^{n \cdot m}$

terminology:
- BEY (bond-equivalent yield)
  annualized yield, compounded semi-annually
- MEY (mortgage-equivalent yield)
  annualized yield, compounded monthly

effective annual interest rate: $(1+r_{eff})^n = \left(1+\frac{r}{m}\right)^{n \cdot m}$

$$\Rightarrow r_{eff} = \left(1+\frac{r}{m}\right)^m - 1$$

↳ allows to compare instruments with different compounding standards

Example: $r = 10\%$, compounded twice a year

$$r_{eff} = \left(1+\frac{0.1}{2}\right)^2 - 1 = 0.1025 = 10.25\%$$

Is r_{eff} always bigger than r? $(r>0)$

$$r_{eff} = \left(1+\frac{r}{m}\right)^m - 1 = 1 + m\frac{r}{m} + \underbrace{rest}_{>0} - 1 > r \quad \text{Yes}$$

$\left(\frac{r}{m} \geq -1 \text{ also true with Bernoulli's inequality}\right)$

continuous compounding: take $\lim_{m\to\infty}$

$$FV = PV \lim_{m\to\infty} \left(1+\frac{r}{m}\right)^{m\cdot n} = PV \cdot e^{rn}$$

1.2 General Cash Flows

n years, r is the yearly interest rate

at end of year there is cash flows $C_1, C_2, \ldots, C_n$

$$PV = \frac{C_1}{(1+r)} + \frac{C_2}{(1+r)^2} + \ldots + \frac{C_n}{(1+r)^n} = \sum_{i=1}^{n} \frac{C_i}{(1+r)^i}$$

with $x = \frac{1}{1+r}$ $\Rightarrow$ have to evaluate polynomials $\sum_{i=1}^{n} C_i x^i$

Implementation in python:

- best: use vectorized operations

  $i = \text{arange}(1, n+1)$, $C = \text{given array}$

  use dot product for vectors: $PV = \text{dot}(C, x^{**}i)$

  (C: vector, $x^{**}i$: vector)

- explicit loop (slow)
- Horner's scheme:

$$PV = \Big(\big((C_n x + C_{n-1}) x + C_{n-2} \big) x + \ldots + C_1 \Big) x$$

- polyval (fct. from SciPy), uses optimized Horner's scheme
  ↳ look up documentation

Annuity: $C_i = C \ \forall i$

ordinary annuity (usually assumed): pays C at end of the year

$$FV = \sum_{i=0}^{n-1} C(1+r)^i$$

geometric series: $x \sum_{i=0}^{n} x^i = \sum_{i=1}^{n+1} x^i = x^{n+1} - 1 + \sum_{i=0}^{n} x^i$

$$(x-1) \sum_{i=0}^{n} x^i = x^{n+1} - 1 \quad \Rightarrow \sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1 - x}$$

$$\Rightarrow FV = C \left(\frac{(1+r)^n - 1}{r} \right)$$