Discounting Library

The discounting library serves two purposes. First, it is used to discount cash flows back to present value and second, it provides a check that all required information for discounting is present in the user input.

# Libraries/Classes calling on Discounting Library

Validate/Read Library

Total Required Flows Class

Total Optional Flows Class

# Internal Library/Class dependencies

None

# External Library/Class Dependencies

None

# Functions

inflationRateCalc() – Returns the inflation rate from the nominal and real discount rates if the user has failed to provide an inflation rate

dRateNomCalc() – Returns the nominal discount rate from the inflation and real discount rates if the user has failed to provide a nominal discount rate

dRateRealCalc() – Returns the real discount rate from the inflation and nominal discount rates if the user has failed to provide a real discount rate

recurEscalationRateCorrection() – Returns the escalation rate in the event there is a discrepancy between the chosen analysis and the provided escalation rates (i.e. escalation rate is for Nominal but analysis is in Real dollars or vice versa)

quantEscalationCalc() – Returns escalation rate to use for a quantity type BCN object provided a quantityVariabilityRateType value exists, otherwise it is assumes no escalation occurs for the quantity

spv() – Returns the multiplier to convert a future value to a present value

presentValueCalc() – Returns the Present Value of a cash flow at a set timestep

escalatedQuantCalc() – Returns the escalated value of a non-monetary quantity

# Pseudo Code

Optional inputs are in italics

Begin Pseudocode

Import necessary libraries

Define inflationRateCalc(dRateNom, dRateReal)

Formula:

Return inflationRate

End inflationRateCalc

Define dRateNomCalc(inflationRate, dRateReal)

Formula:

Return dRateNom

End dRateNomCalc

Define dRateRealCalc(dRateNom,inflationRate)

Formula:

Return dRateReal

End dRateRealCalc

recurEscalationRateCorrection(analysisType, recurrenceVariabilityRateType, recurrenceVariabilityRateValues,*time*)

if escalation rate is required to be in real dollars

if recurrenceVariabilityRateValues is *float* (no *time* input required)

formula:

return

else if recurrenceVariabilityRateValues is *list of floats*

find appropriate list element based on *time* input

formula:

return

else if escalation rate is required to be in nominal dollars

if recurrenceVariabilityRateValues is *float* (no *time* input required)

formula:

return

else if recurrenceVariabilityRateValues is *list of floats*

find appropriate list element based on *time* input

formula:

return

end recurEscalationRateCorrection

define quantEscalationCalc(quantityVariabilityRateType, quantityVariabilityRateValues,*time*)

if quantityVariabilityRateValues is a *float*

formula:

else if quantityVariabilityRateValues is a *list of floats*

find appropriate list element based on *time* input

formula:

return

end quantEscalationRateCorrection

define spv(time,recurrenceVariabilityRateValues,discountRate)

if recurrenceVariabilityRateValues is *float*

formula:

else if recurrenceVariabilityRateValues is *list of floats*

formula:

where iteration through indices by 1 (note for programming purposes Python indexes lists starting at 0)

return

end spv

define discValueCalc(value,spv)

note that the spv value is based on a call on assumed to be done prior to the call to discValueCalc to get appropriate discounting value according to:

spv(time,recurrenceVariabilityRateValues,discountRate)

if there are no situations where we need an spv value without using it then it may make sense to fold *spv* and *discValueCalc* into a single function

formula:

return

end discValueCalc

define escalatedQuantCalc(quantity,quantityVariabilityRateType,quantityVariabilityRateValues,*time*)

calculate the escalation value (according to:

quantEscalationCalc(quantityVariabilityRateType,quantityVariabilityRateValues,*time*)

if quantityVariabilityRateValues is *float*

formula:

else if quantityVariabilityRateValues is *list of floats*

formula:

where iteration through indices by 1

calculate the escalated quantity value and return

end escalatedQuantCalc