defmodule Options.Main do

# mostly for debugging and exploration

def start() do

result = split(100)

IO.inspect(result)

IO.inspect(expand(result))

end

@doc """

iex> Options.Main.split(100, 2.0)

[50.0, 200.0]

"""

# split/2

# equity (stock) price progression, symetric

def split(s, gu \\ 2.0) do

with gd = 1 / gu, do: split(s, gu, gd)

end

@doc """

iex> Options.Main.split(100, 2.0, 0.5)

[50.0, 200.0]

"""

# split/3

# equity (stock) price progression

def split(s, gu, gd) do

[s \* gd, s \* gu]

end

@doc """

iex> Options.Main.bondp(100, 0.05, 1.0)

95.1229424500714

"""

# pres val of bf bond future

def bondp(bf, r, dt) do

bf \* :math.exp(-r \* dt)

end

@doc """

iex> Options.Main.callf([50, 200], 100)

[0, 100]

"""

# call future value

def callf([sd, su], ex) do

[max(0, sd - ex), max(0, su - ex)]

end

@doc """

iex> Options.Main.bondf([50, 200], [0, 100])

50

"""

# bond future value that satisfy hedge position w/o ratio

def bondf([sfd, \_sfu], [cfd, \_cfu]) do

if cfd <= 0.0 do

sfd

else

cfd

end

end

@doc """

iex> Options.Main.hedgef([50, 200], [0, 100])

[0, 150]

"""

# future hedge portfolio, bf bond future

def hedgef([sfd, sfu], [cfd, cfu]) do

with bf = bondf([sfd, sfu], [cfd, cfu]) do

[sfd - bf, sfu - bf]

end

end

@doc """

iex> Options.Main.hedgep(3.0, 2.0)

1.0

"""

# pres val hedge portfolio, stock present bond present

def hedgep(sp, bp) do

sp - bp

end

@doc """

iex> Options.Main.ratio([1.0, 2.0], [0.0, 1.0])

2.0

"""

# hedge future down, up, call etc.

def ratio([\_hfd, hfu], [\_cfd, cfu]) do

# may want to also add checks on the down values in future

hfu / cfu

end

@doc """

iex> Options.Main.callp(100, [50, 200], [0, 100], 100, 0.05, 5.0)

40.7066405642865

"""

# present value of call using forward call prices from tree

# sf stock present future down up, exercise price, hcr hedge call ratio

def callp(sp, [sfd, sfu], [cfd, cfu], ex, r, dt) do

with sf = [sfd, sfu],

cf = [cfd, cfu],

hcr = ratio(hedgef(sf, cf), cf),

bf = bondf(sf, cf),

bp = bondp(bf, r, dt) do

(sp - bp) / hcr

end

end

@doc """

iex> Options.Main.callpp(100, [50, 200], 100, 0.05, 1.0)

34.95901918330953

"""

# present value call, future call prices inferred

def callpp(sp, [sd, su], ex, r, dt) do

if ex >= sd and ex < su do

(sp - bondp(sd, r, dt)) \* (su - ex) / (su - sd)

else

if ex < sd do

sp - bondp(ex, r, dt)

else

0.0

end

end

end

@doc """

iex> [50.0, 200.0] |> Options.Main.expand()

[[25.0, 100.0], [100.0, 400.0]]

"""

# add layer to stock price progression

def expand([d, u]) do

if is\_float(d) or is\_integer(d) do

[split(d), split(u)]

else

[expand(d), expand(u)]

end

end

@doc """

iex> Options.Main.spread(100.0, 2)

[[[12.5, 50.0], [50.0, 200.0]], [[50.0, 200.0], [200.0, 800.0]]]

"""

# stock price progression to n levels

def spread(s \\ 100.0, n \\ 2) do

1..n

|> Enum.reduce(split(s), fn \_x, acc -> expand(acc) end)

end

@doc """

iex> [[[1.0, 1.0], [1.0, 1.0]], [[1.0, 1.0], [1.0, 1.0]]] |> Options.Main.depth()

3

"""

# depth of stock price prgression tree

def depth([x, \_y]) do

if is\_float(x) or is\_integer(x) do

1

else

1 + depth(x)

end

end

end