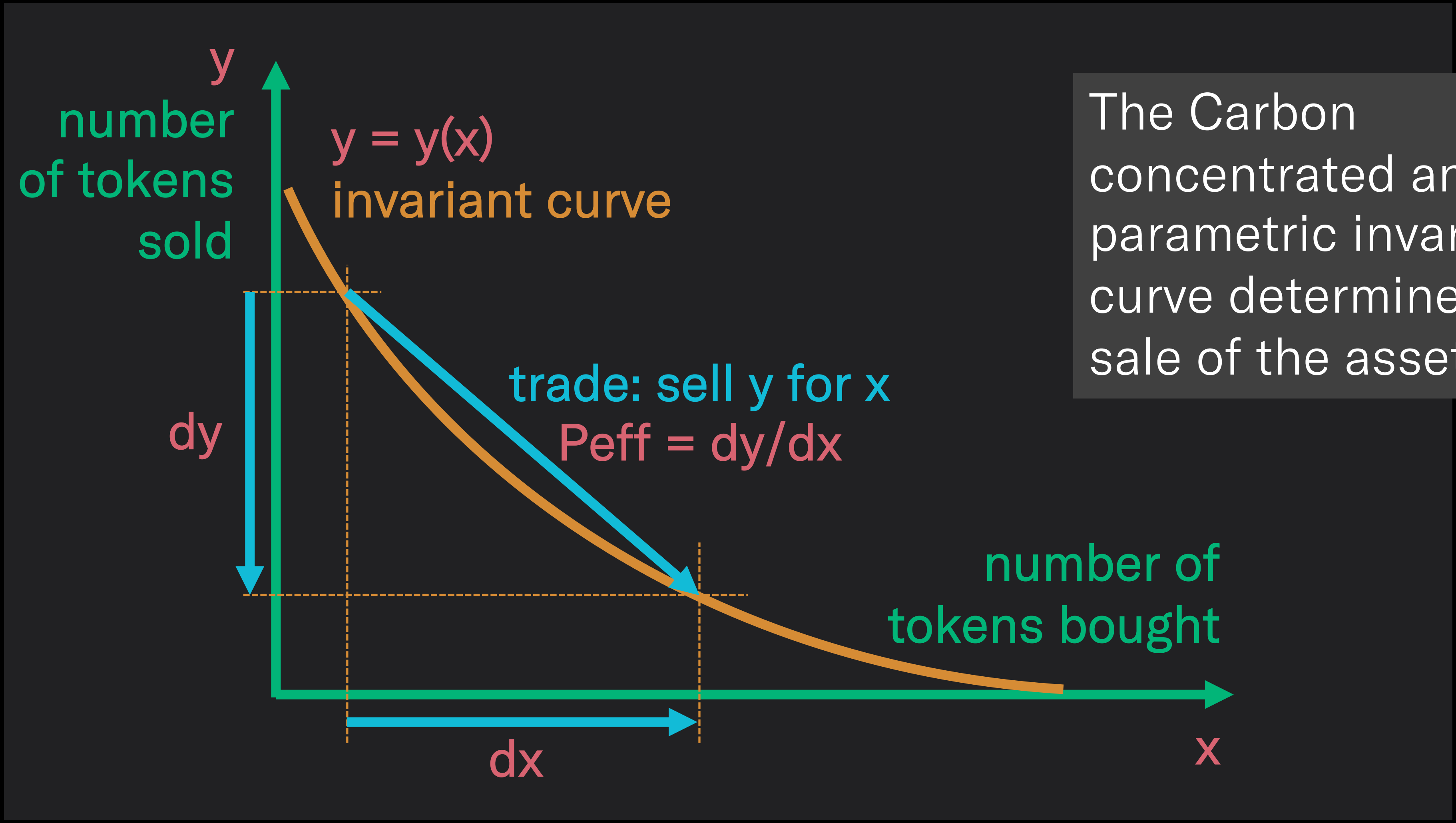
CARBON

Simulating Carbon

v1.0 (22 January 2023)

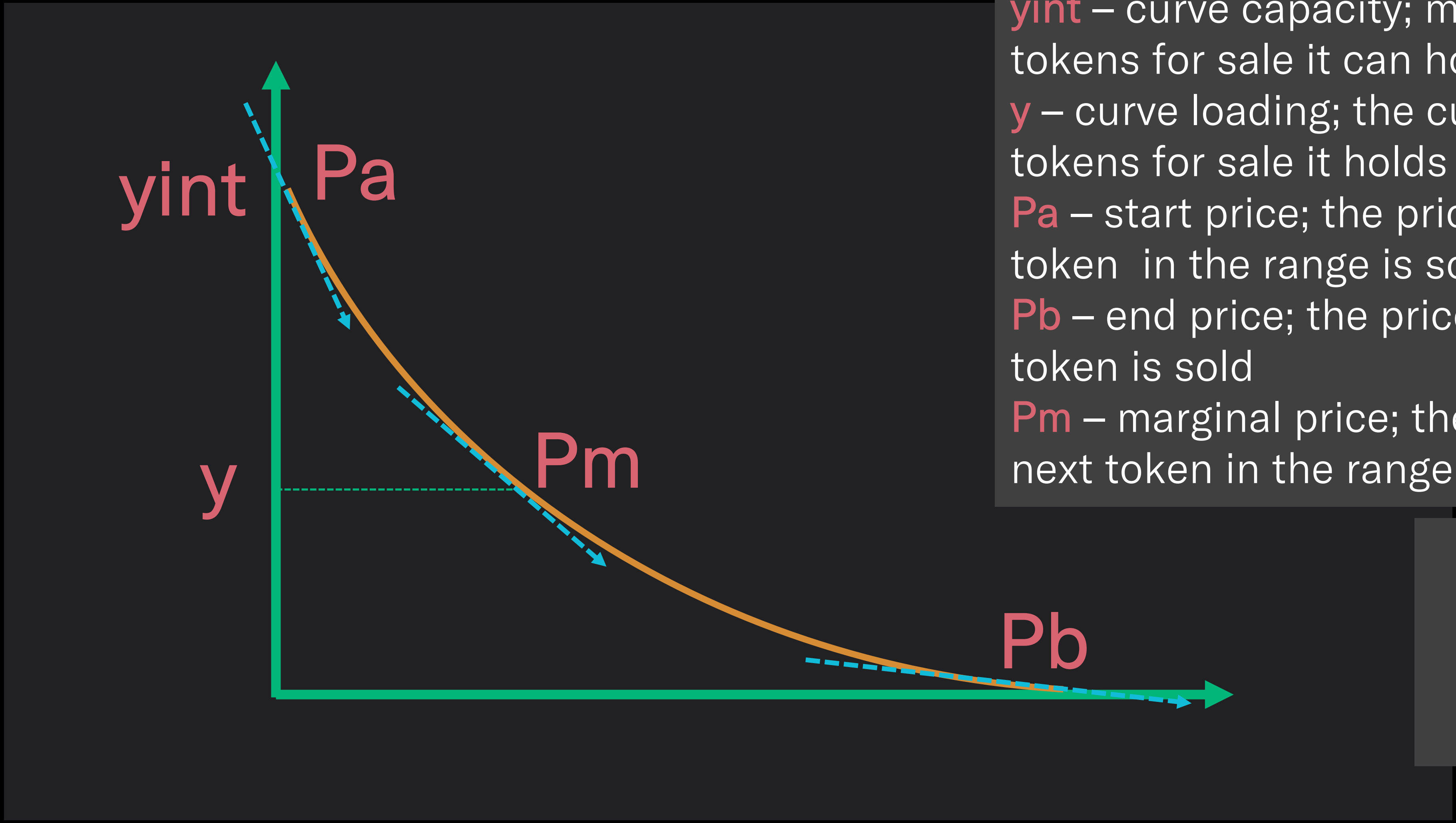
Carbon basics

The Carbon invariant curve



The Carbon concentrated and parametric invariant curve determines the sale of the asset y

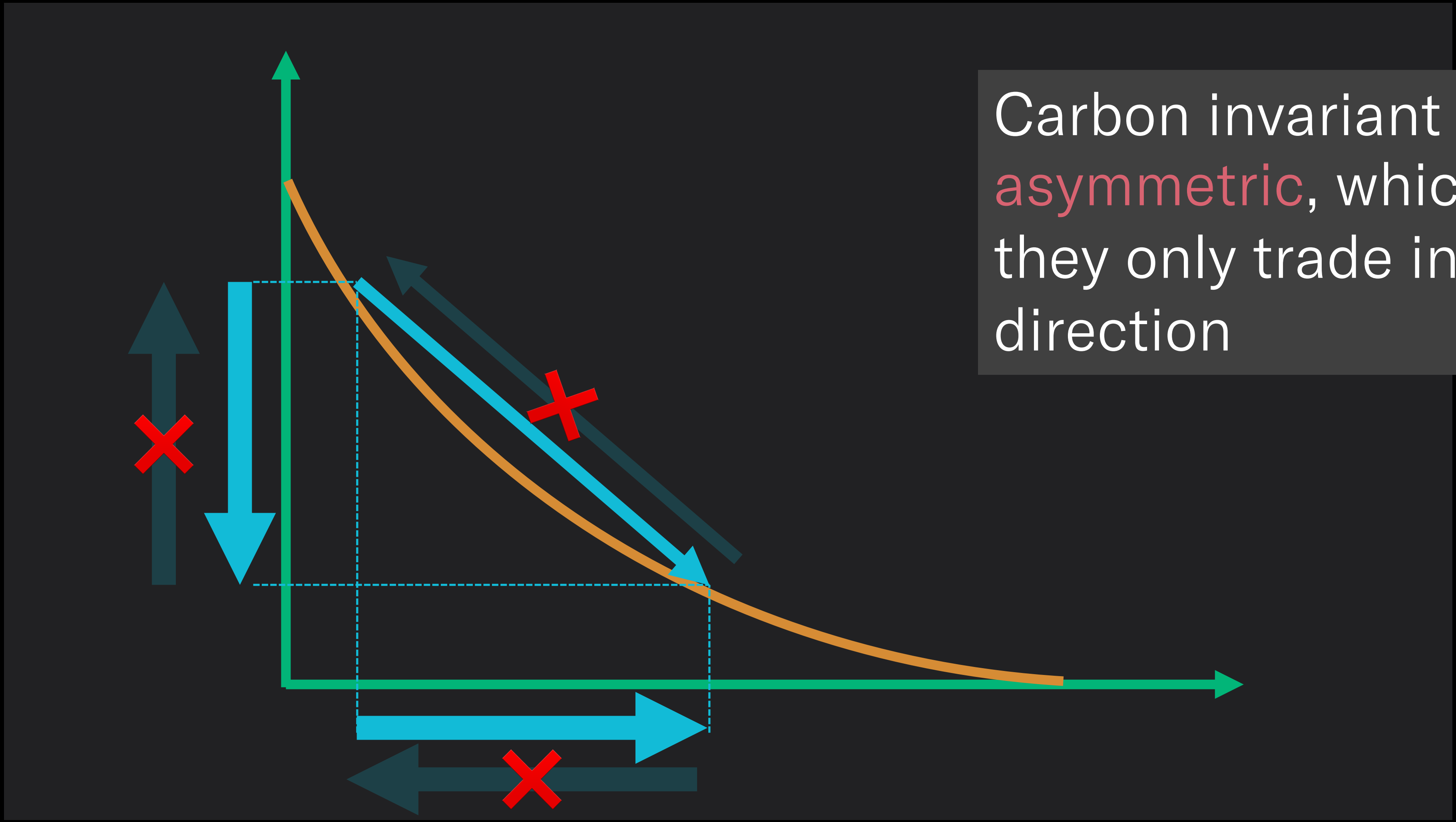
Carbon curve parameters



yint – curve capacity; maximum number of tokens for sale it can hold
y – curve loading; the current number of tokens for sale it holds
Pa – start price; the price at which the first token in the range is sold
Pb – end price; the price at which the last token is sold
Pm – marginal price; the price at which the next token in the range is sold

(1) Only four of five params independent.
(2) Natural price quote dy/dx, to be converted to pair convention

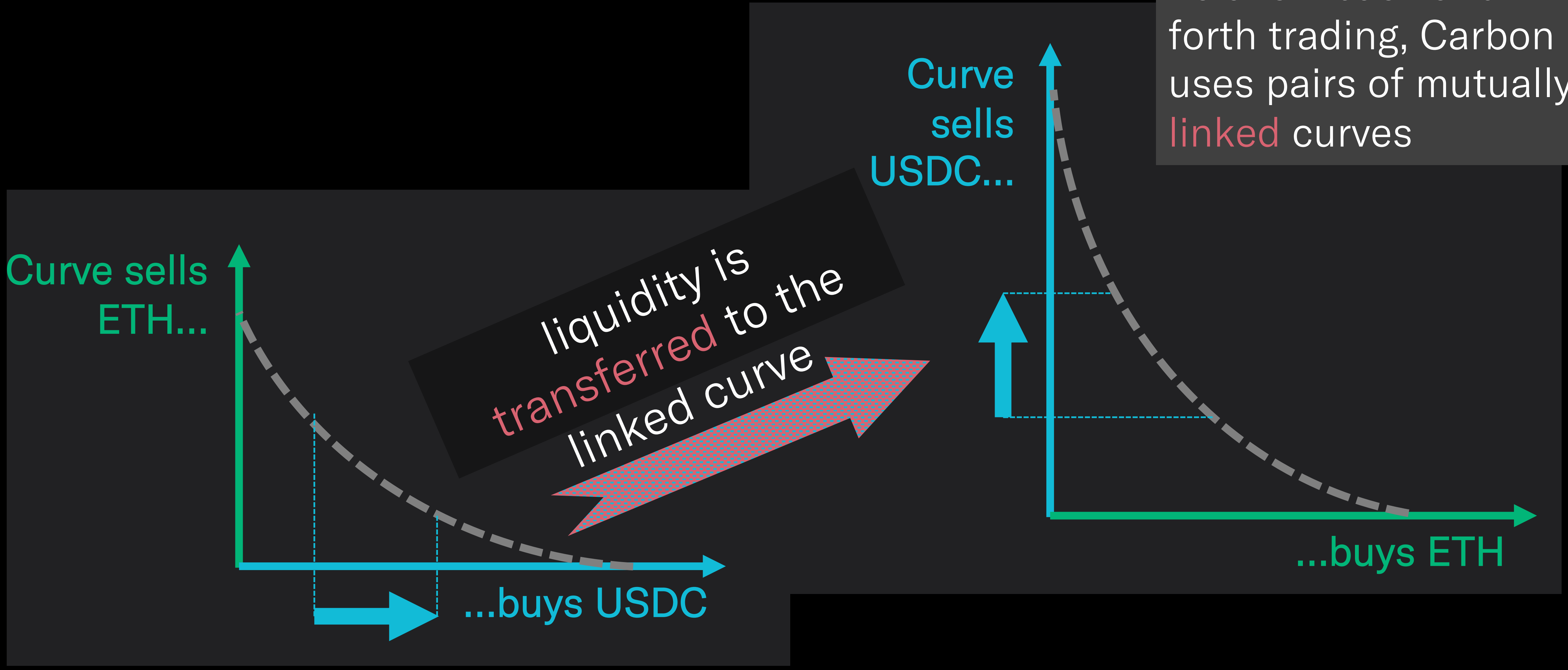
Carbon asymmetric curve



Carbon invariant curves are asymmetric, which means they only trade in one direction

Carbon linked curves

To allow back and forth trading, Carbon uses pairs of mutually linked curves



Minimum viable simulation

Minimum viable simulation

Initialize the simulator

```
Sim = CarbonSimulatorUI(pair="ETH/USDC")
```

Use ETH/USDC as default pair

Add a Carbon strategy

```
Sim.add_strategy(“ETH”,
                     1, 1500, 2000,
                  1000, 1250, 1000)
```

Sell ETH between 1,500-2,000; seed with 1 ETH
Buy ETH between 1,250-1,000; seed with 1,000 USDC

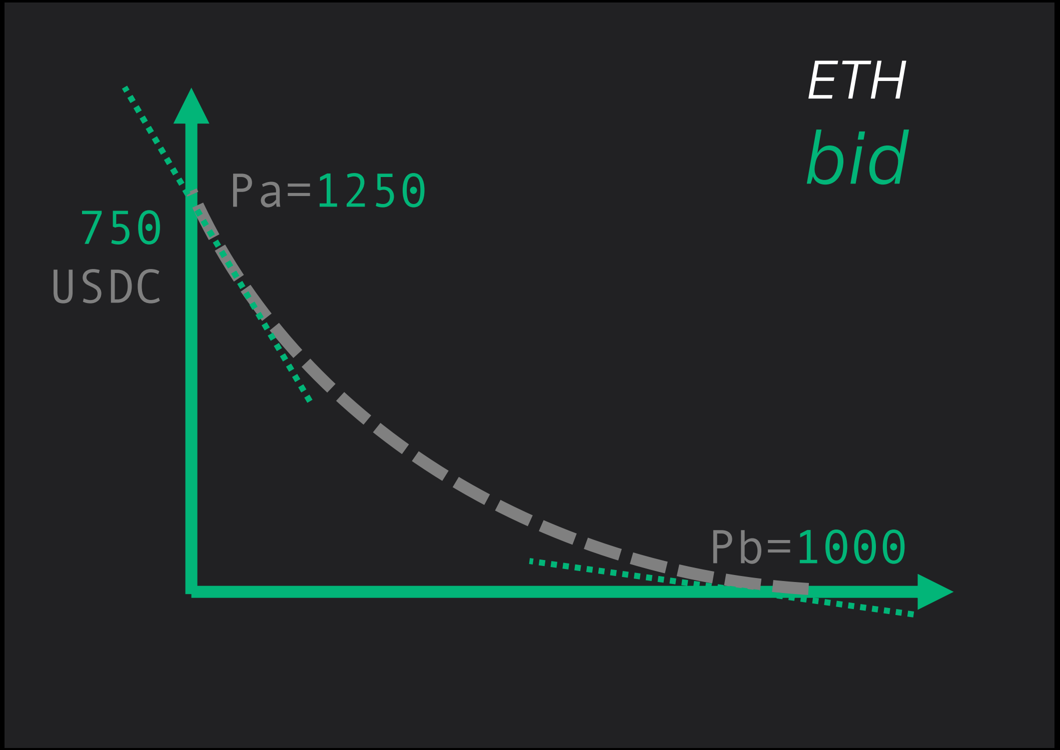
Execute a trade

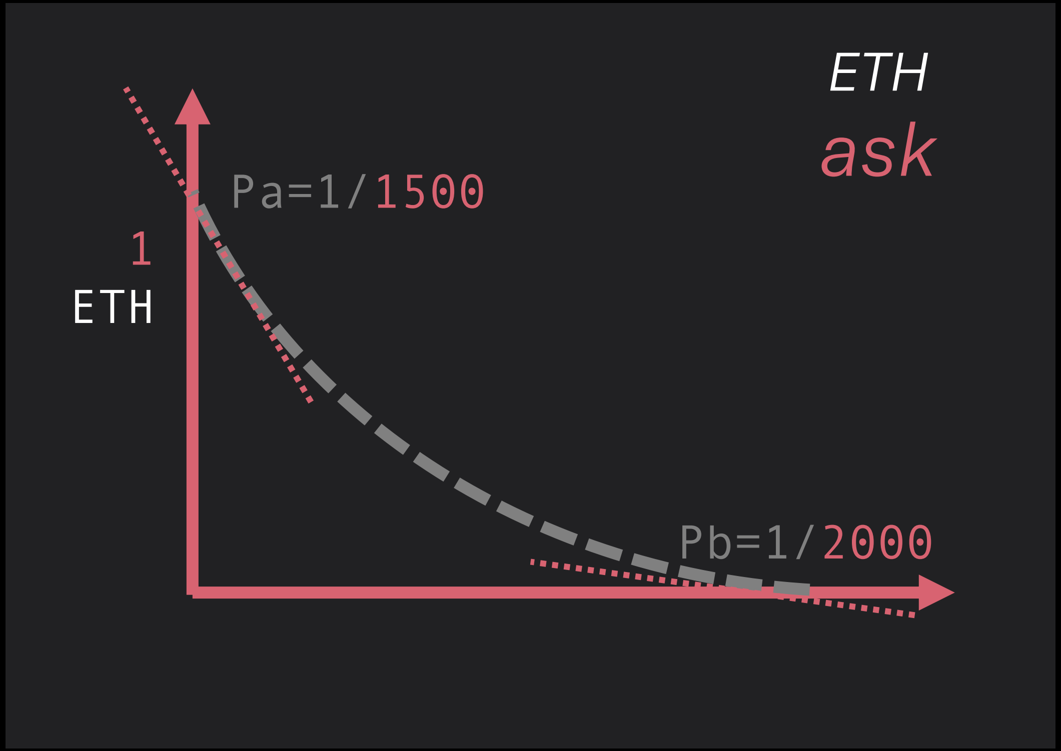
```
Sim.amm_sells(“ETH”, 0.5)
```

Sell 0.5 ETH

Adding a Carbon strategy

```
add_strategy(“ETH”, 1, 1500, 2000, 750, 1250, 1000)
```

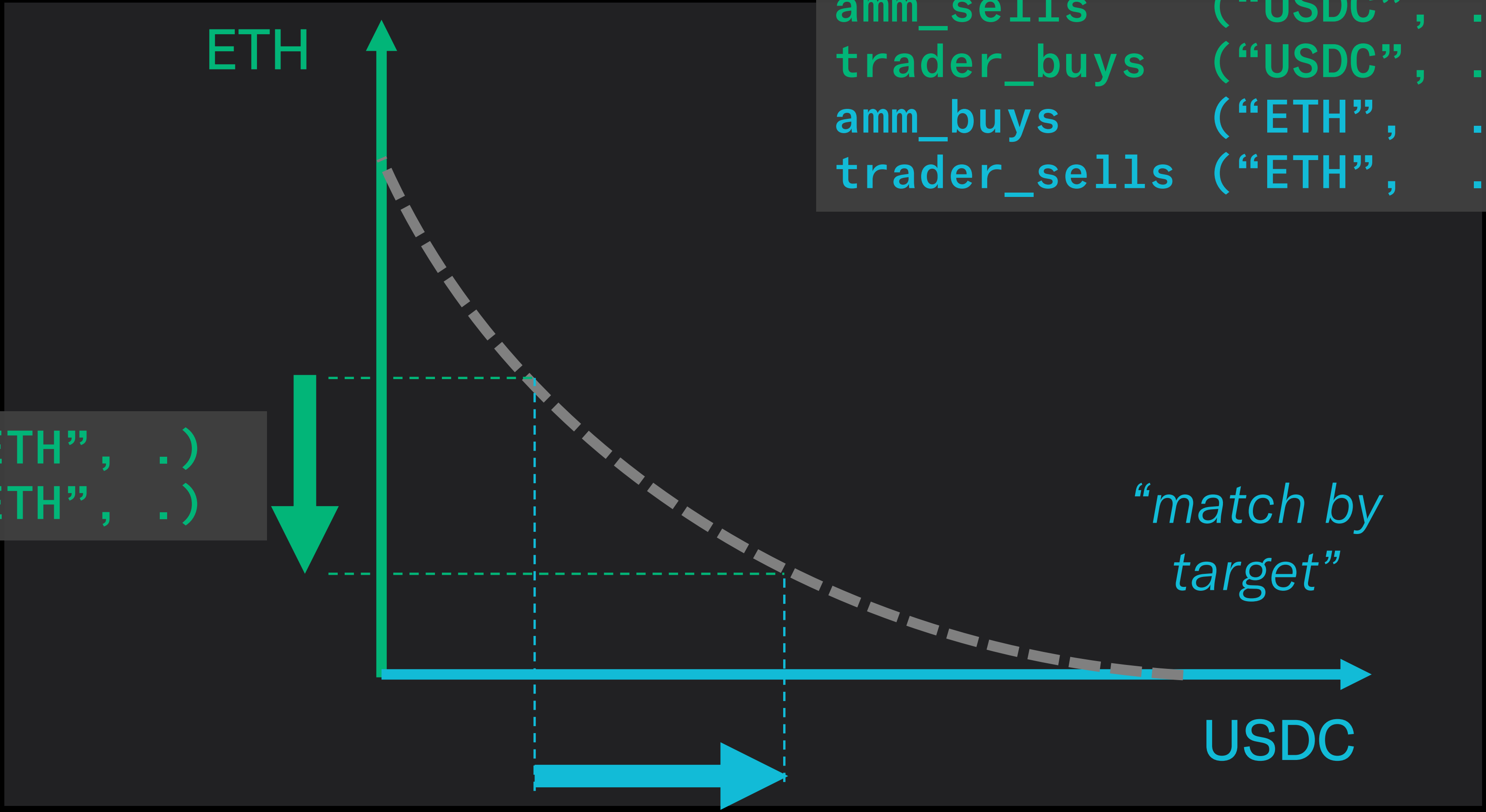




Executing a trade

```
LINKED CURVE
============
amm_sells     (“USDC”, .)
trader_buys   (“USDC”, .)
amm_buys      (“ETH”,  .)
trader_sells  (“ETH”,  .)
```

“match by source”



```
amm_sells    (“ETH”, .)
trader_buys  (“ETH”, .)
```

“match by target”

```
amm_buys     (“USDC”, .)
trader_sells (“USDC”, .)
```

Simulation example

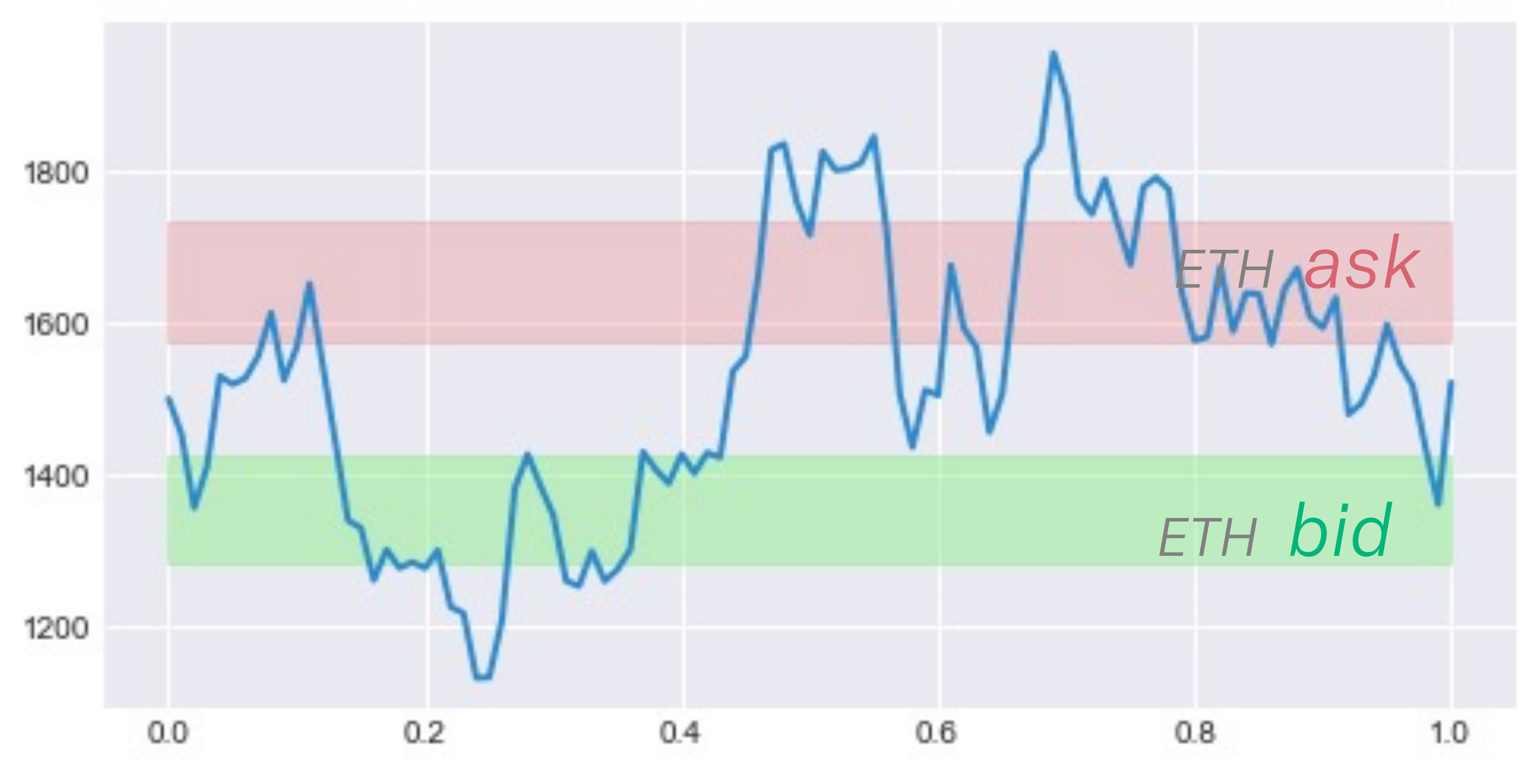
detailed sim using dyfromp_f function (Demo 7-1)

Demo 7-1

Carbon simulation run

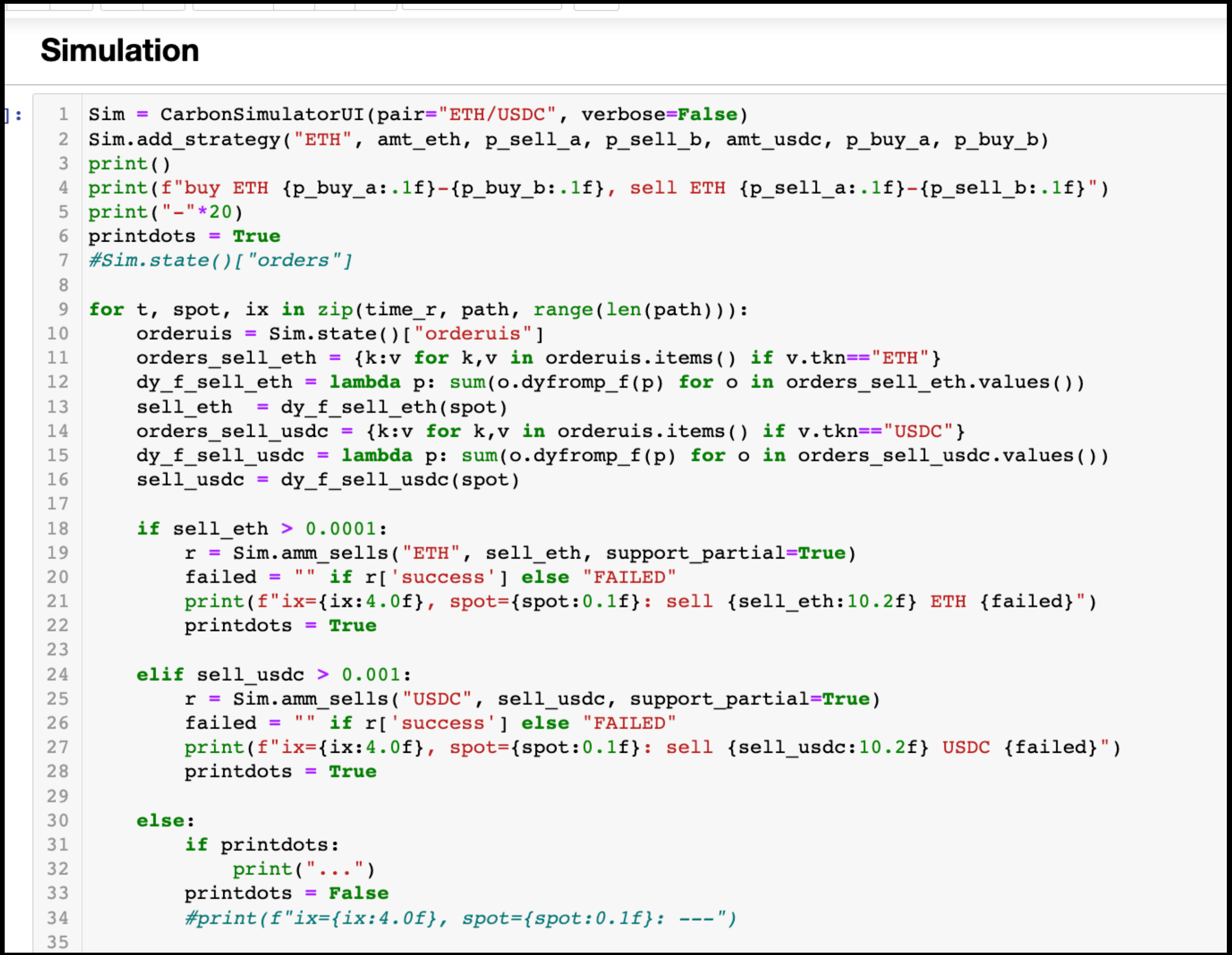
```
    ETH bid 1282-1425 (750 USDC)
             spot 1500
    ETH ask 1575-1732 (  1 ETH )
```

```
buy ETH 1425.0-1282.5, sell ETH 1575.0-1732.5
--------------------
...
ix=   2, spot=1356.7: sell     472.93 USDC
...
ix=   8, spot=1612.9: sell       0.34 ETH
...
ix=  11, spot=1651.3: sell       0.33 ETH
...
ix=  14, spot=1340.0: sell     950.72 USDC
ix=  15, spot=1330.4: sell     109.97 USDC
ix=  16, spot=1261.5: sell     551.29 USDC
...
ix=  46, spot=1663.8: sell       1.08 ETH
ix=  47, spot=1826.8: sell       0.78 ETH
...
ix=  99, spot=1361.2: sell    1356.49 USDC
...
--------------------
ix=   0,  spot=1500.0: 1.0 ETH 1000.0 USDC (=2500.0 USDC)
ix= 100,  spot=1521.4: 1.0 ETH 1715.7 USDC (=3197.5 USDC)
```



Demo 7-1

Carbon simulation run



Simulation

```
Sim = CarbonSimulatorUI(pair="ETH/USDC", verbose=False)
Sim.add_strategy("ETH", amt_eth, p_sell_a, p_sell_b, amt_usdc, p_buy_a, p_buy_b)
print()
print(f"buy ETH {p_buy_a:.1f}-{p_buy_b:.1f}, sell ETH {p_sell_a:.1f}-{p_sell_b:.1f}")
print("-"*20)
printdots = True
#Sim.state()["orders"]

for t, spot, ix in zip(time_r, path, range(len(path))):
    orderuis = Sim.state()["orderuis"]
    orders_sell_eth = {k:v for k,v in orderuis.items() if v.tkn=="ETH"}
    dy_f_sell_eth = lambda p: sum(o.dyfromp_f(p) for o in orders_sell_eth.values())
    sell_eth  = dy_f_sell_eth(spot)
    orders_sell_usdc = {k:v for k,v in orderuis.items() if v.tkn=="USDC"}
    dy_f_sell_usdc = lambda p: sum(o.dyfromp_f(p) for o in orders_sell_usdc.values())
    sell_usdc = dy_f_sell_usdc(spot)

    if sell_eth > 0.0001:
        r = Sim.amm_sells("ETH", sell_eth, support_partial=True)
        failed = "" if r['success'] else "FAILED"
        print(f"ix={ix:4.0f}, spot={spot:0.1f}: sell {sell_eth:10.2f} ETH {failed}")
        printdots = True

    elif sell_usdc > 0.001:
        r = Sim.amm_sells("USDC", sell_usdc, support_partial=True)
        failed = "" if r['success'] else "FAILED"
        print(f"ix={ix:4.0f}, spot={spot:0.1f}: sell {sell_usdc:10.2f} USDC {failed}")
        printdots = True

    else:
        if printdots:
            print("...")
        printdots = False
        #print(f"ix={ix:4.0f}, spot={spot:0.1f}: ---")

```

Initialize the simulator and load the strategy; then loop over the spot values

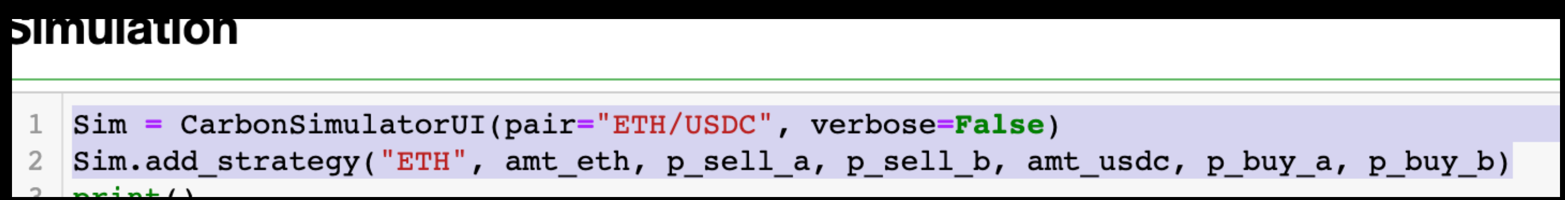
In the loop, determine how much ETH (and USDC) to sell to get to a certain price...

...and execute the transaction

Carbon simulation run

Demo 7-1

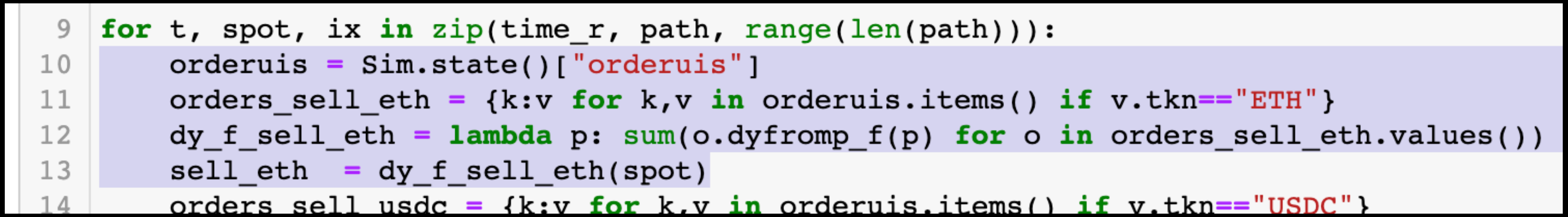
Initialize the simulator and load the strategy; then loop over the spot values



Simulation

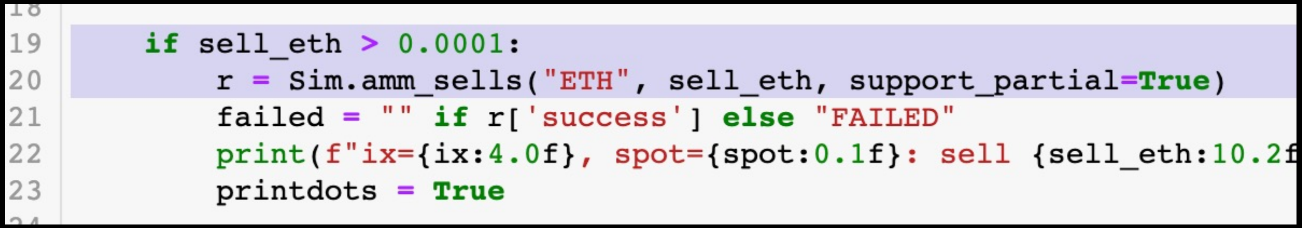
```
Sim = CarbonSimulatorUI(pair="ETH/USDC", verbose=False)
Sim.add_strategy("ETH", amt_eth, p_sell_a, p_sell_b, amt_usdc, p_buy_a, p_buy_b)
print()
```

In the loop, determine how much ETH (and USDC) to sell to get to a certain price…



```
for t, spot, ix in zip(time_r, path, range(len(path))):
    orderuis = Sim.state()["orderuis"]
    orders_sell_eth = {k:v for k,v in orderuis.items() if v.tkn=="ETH"}
    dy_f_sell_eth = lambda p: sum(o.dyfromp_f(p) for o in orders_sell_eth.values())
    sell_eth   = dy_f_sell_eth(spot)
    orders_sell_usdc = {k:v for k,v in orderuis.items() if v.tkn=="USDC"}
```

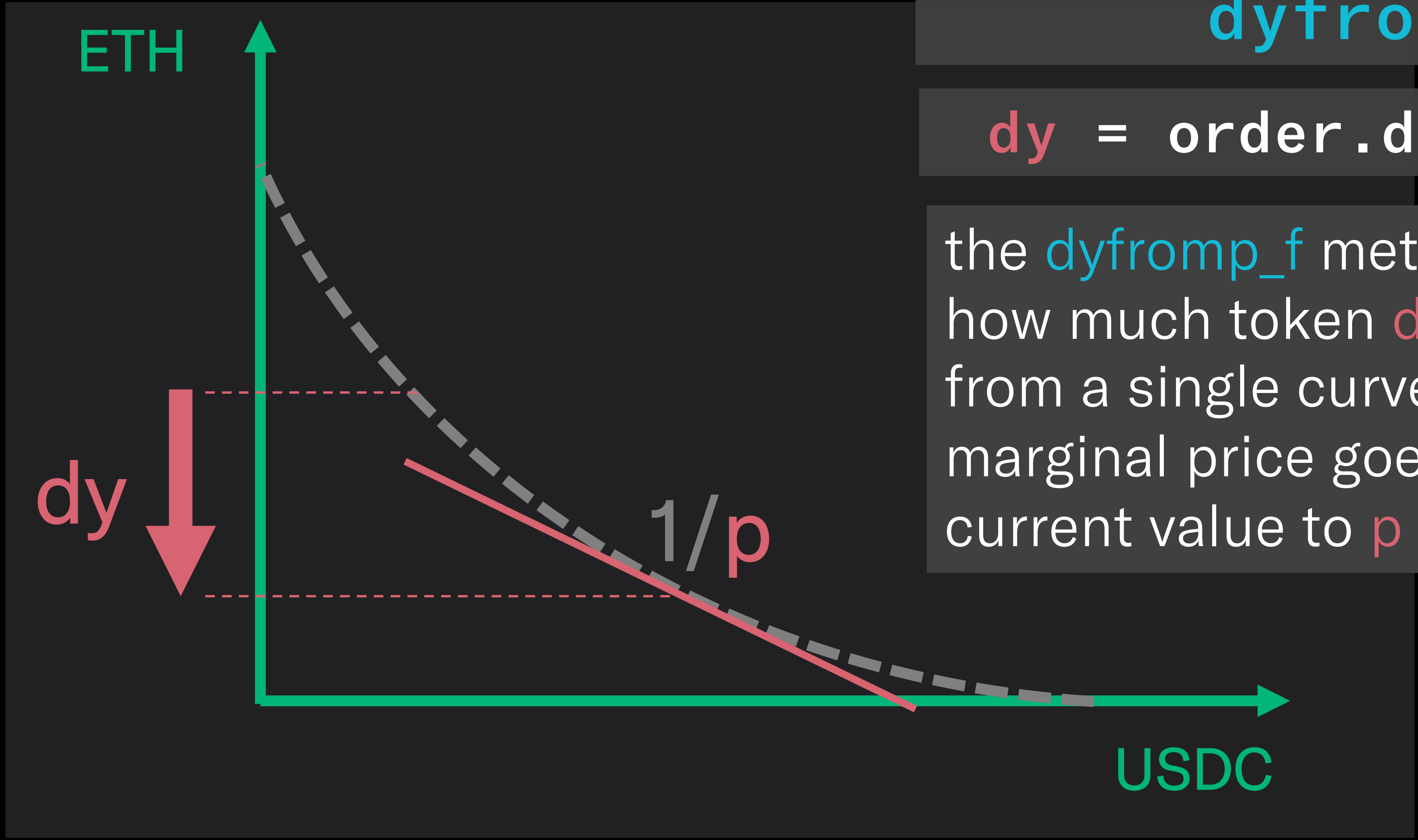
…and execute the transaction



```
    if sell_eth > 0.0001:
        r = Sim.amm_sells("ETH", sell_eth, support_partial=True)
        failed = "" if r['success'] else "FAILED"
        print(f"ix={ix:4.0f}, spot={spot:0.1f}: sell {sell_eth:10.2f
        printdots = True
```

Demo 7-1

Liquidity released by price



dyfromp_f

```
dy = order.dyfromp_f(p)
```

the dyfromp_f method calculates how much token dy will be sold from a single curve when the marginal price goes from its current value to p

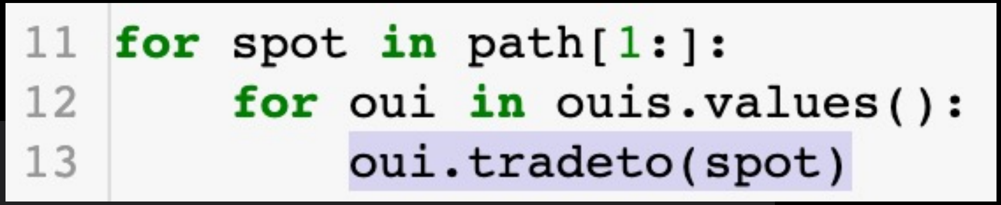
Simulation example

fast sim using tradeto function (Demo 7-1; NBTest 50 ,51)

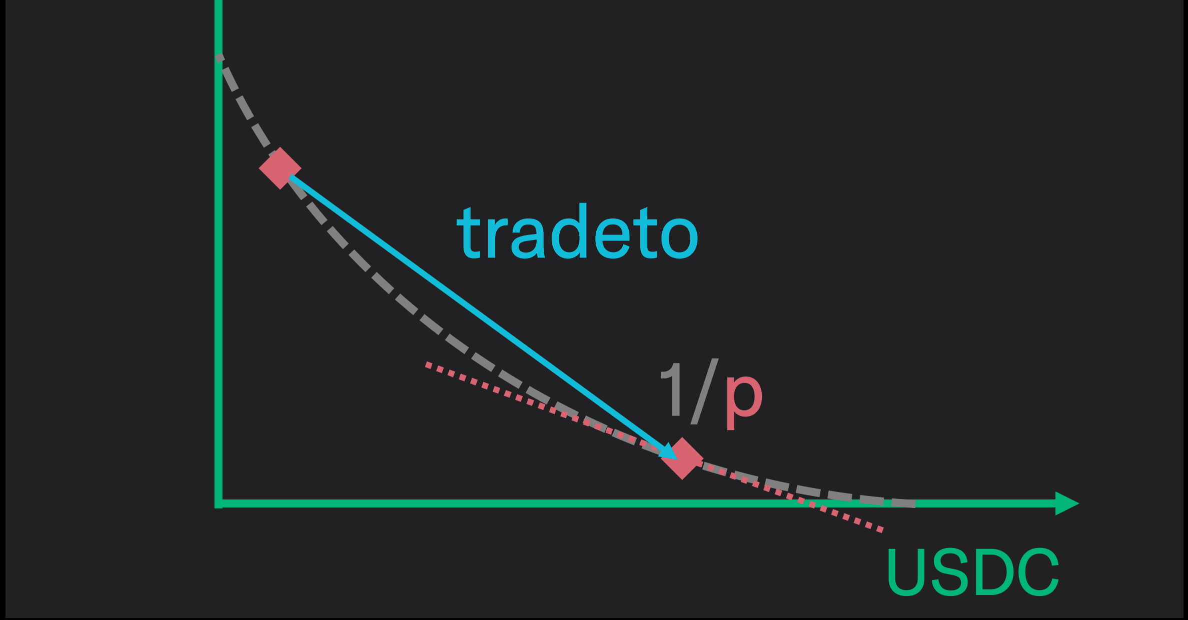
Using the tradeto function

Demo 7-1

Demo 7-2



```
for spot in path[1:]:
    for oui in ouis.values():
        oui.tradeto(spot)
```



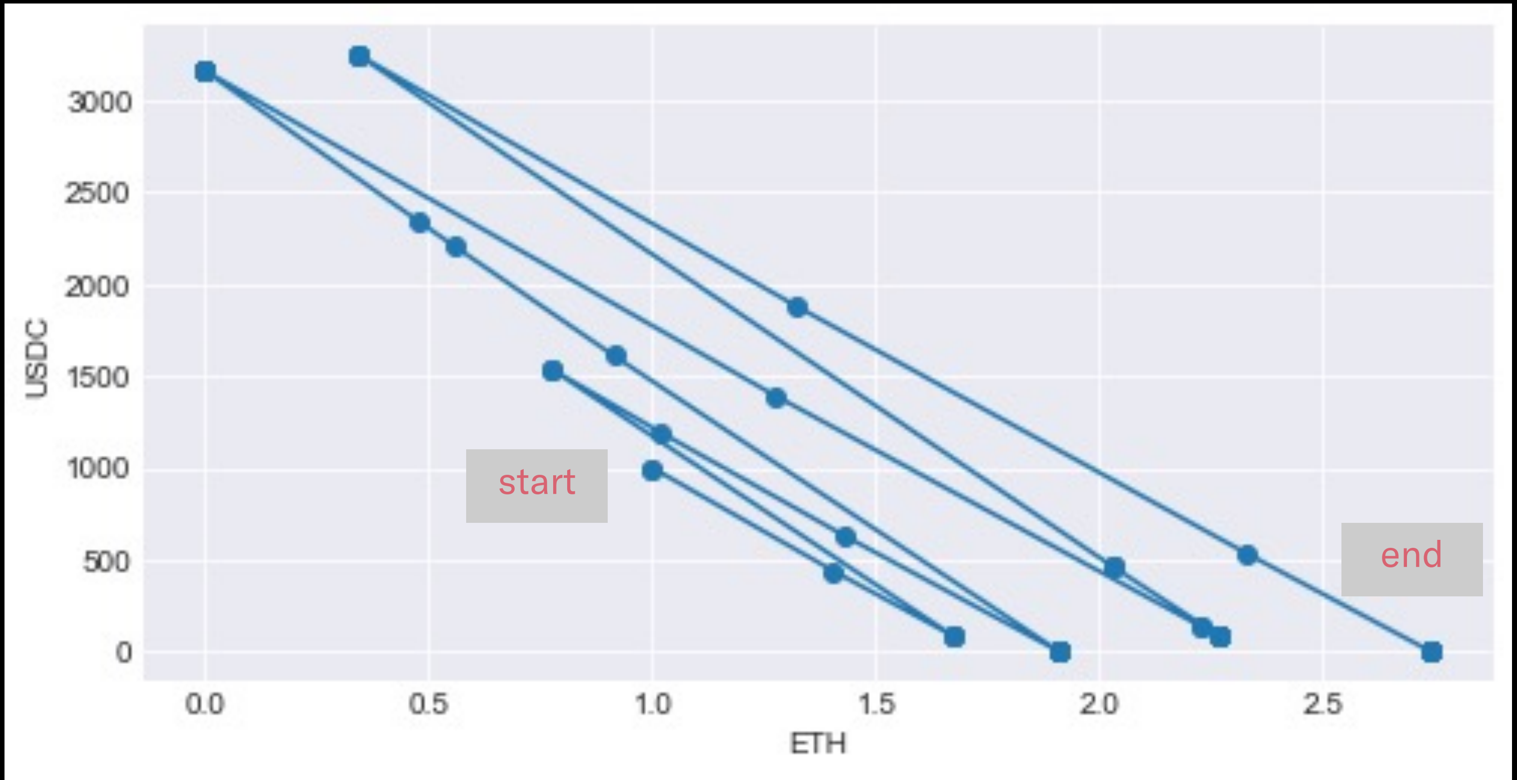
`tradeto(p)`

`order.tradeto(p)`

the tradeto method changes the state of an order, trading it to the marginal price p. the collateral received is placed with the linked order, if present

Portfolio path chart

Demo 7-1

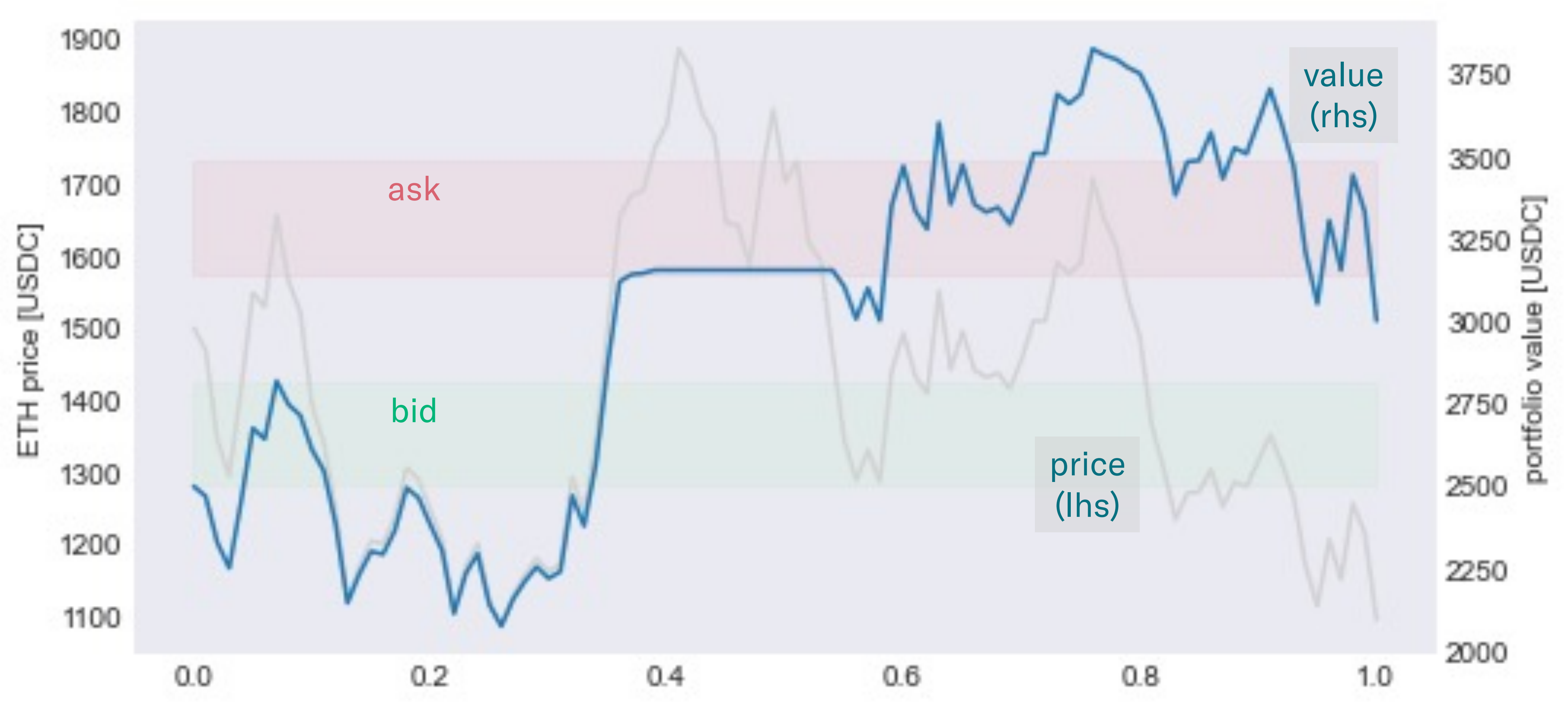


the tradeto method changes the state of an order, trading it to the marginal price p. the collateral received is placed with the linked order, if present

Portfolio value evaluation chart

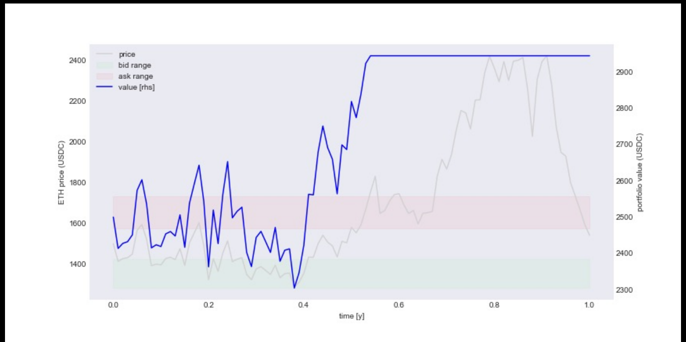
Demo 7-1

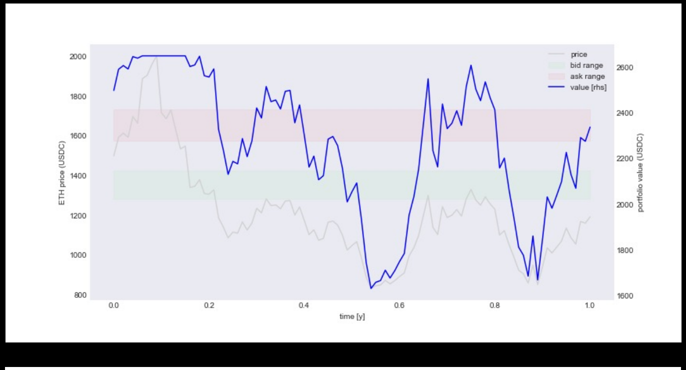
Demo 7-2

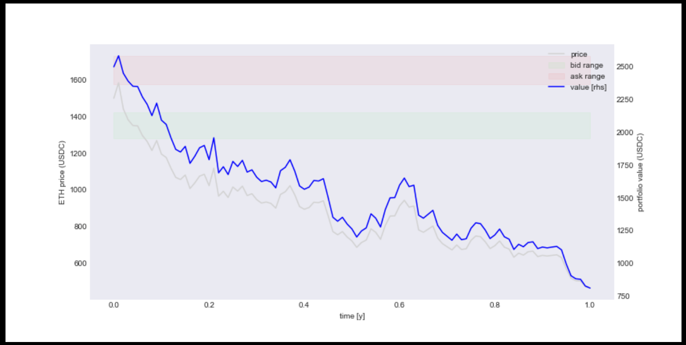


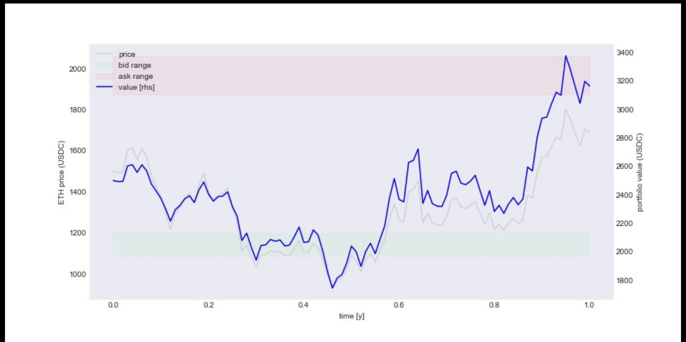
Various simulation runs

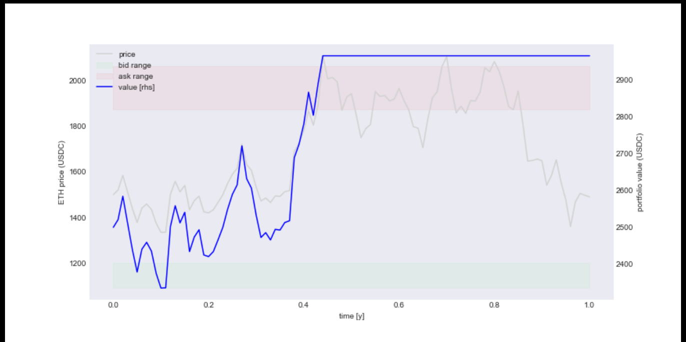
Demo 7-2

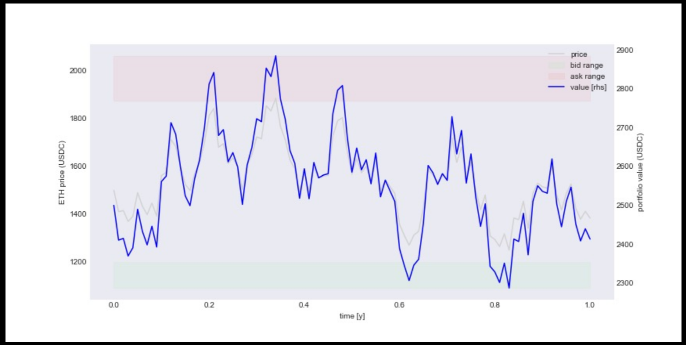


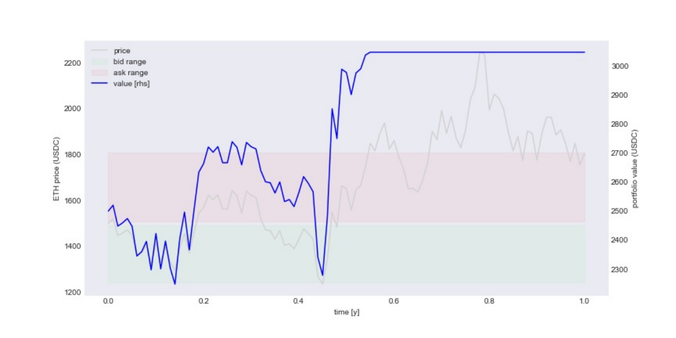


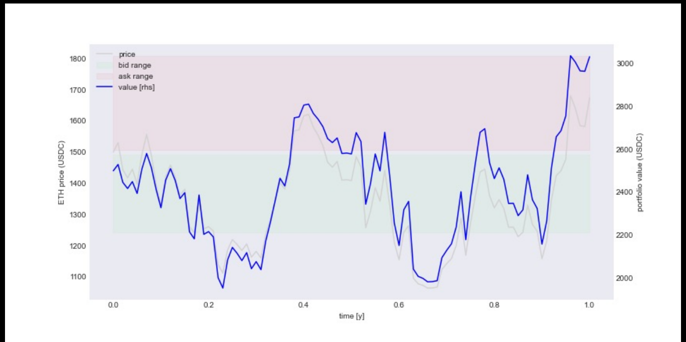


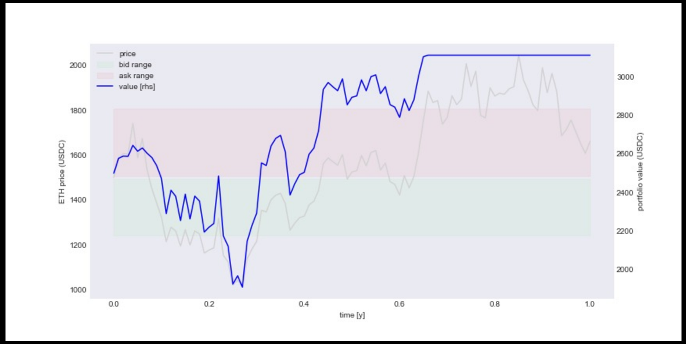












Frozen simulations

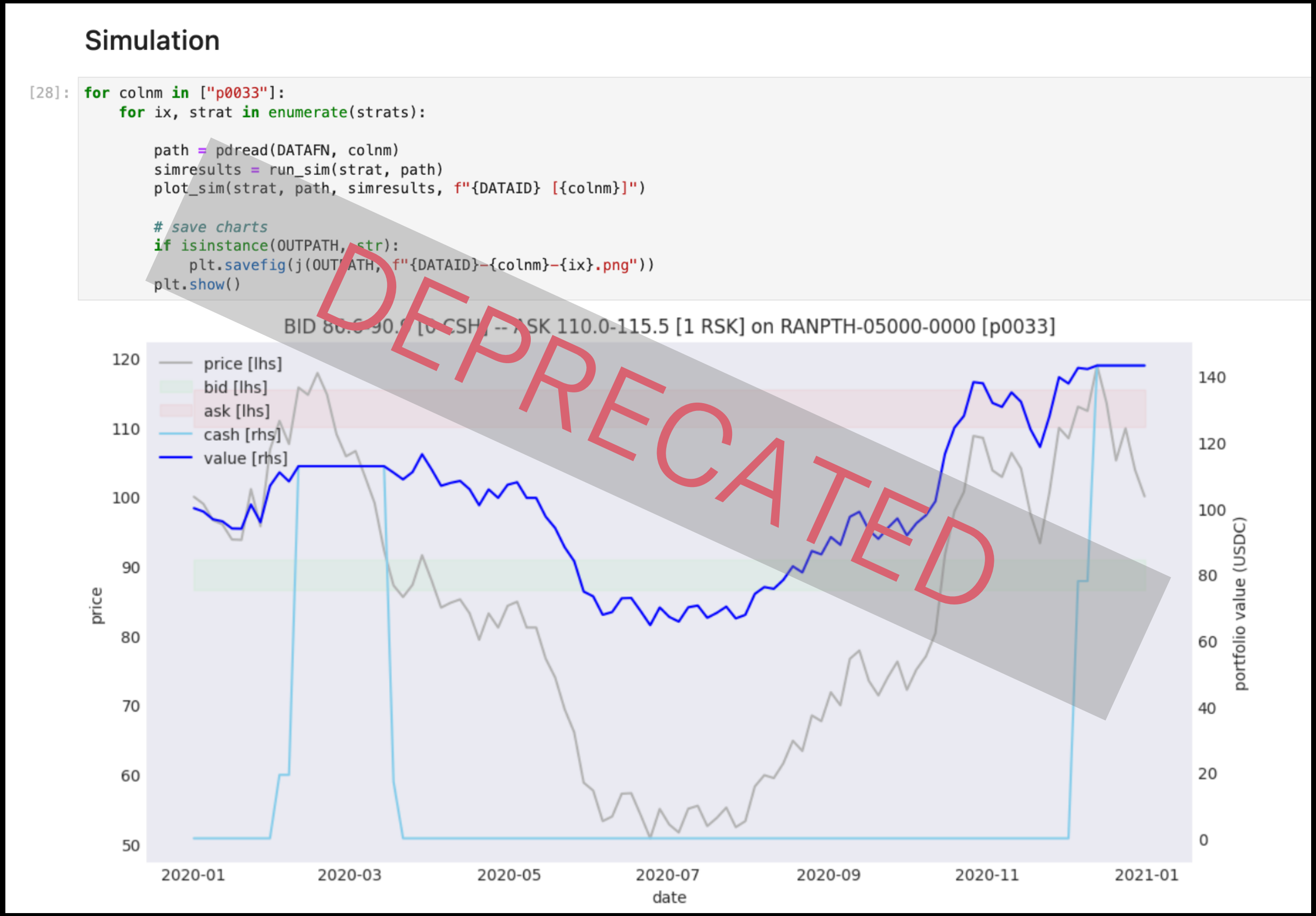
sims based on fixed set of paths for reproducibility (Demo 7-3)

Binder

- Binder (mybinder.org) is docker-based execution platform for Jupyter notebooks based on publicly available git repoes
- Carbon maintains a repo with analysis' specific for binder, carbon-simulator-binder; this repo does not contain a Carbon distribution, but relies on the latest version released on PyPi
- The Binder repo contains a large number of workbooks, so it can be overwhelming; look at the TOC file for specific links to workspaces that have been optimized

Sim on Binder (deprecated)

Demo 7-3
(frozen_20230121)



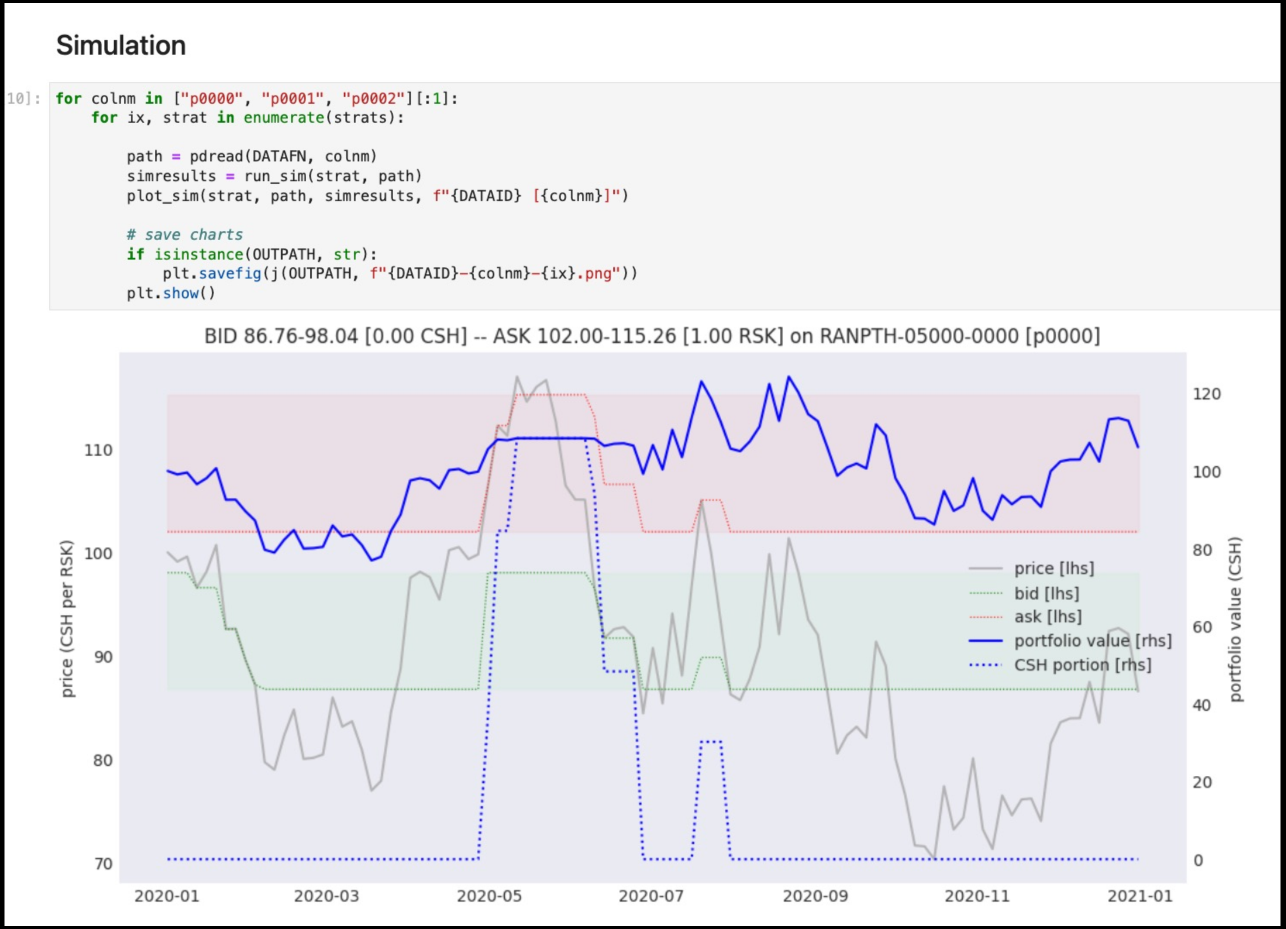
Simulation

```
[28]: for colnm in ["p0033"]:
          for ix, strat in enumerate(strats):

              path = pdread(DATAFN, colnm)
              simresults = run_sim(strat, path)
              plot_sim(strat, path, simresults, f"{DATAID} [{colnm}]")

              # save charts
              if isinstance(OUTPATH, str):
                  plt.savefig(j(OUTPATH, f"{DATAID}-{colnm}-{ix}.png"))
              plt.show()
```

the above link refers to a previous version of the analysis notebook; it is now deprecated, but as it is frozen it is a backup in case there is an issue with the more recent distributions

Simulation on Binder

Demo 7-3 (frozen_202302)



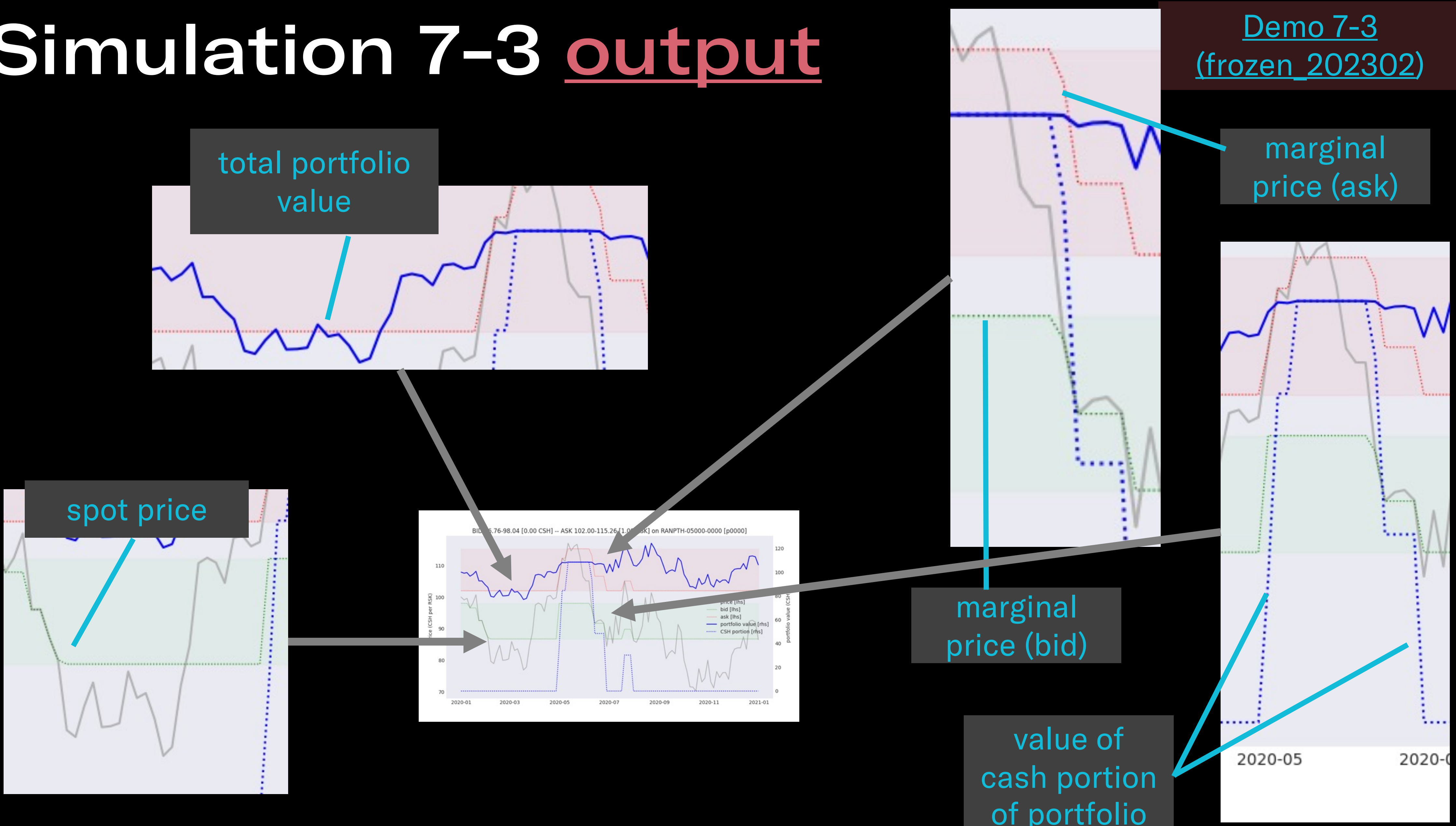
Simulation

```
10]: for colnm in ["p0000", "p0001", "p0002"][:1]:
         for ix, strat in enumerate(strats):

             path = pdread(DATAFN, colnm)
             simresults = run_sim(strat, path)
             plot_sim(strat, path, simresults, f"{DATAID} [{colnm}]")

             # save charts
             if isinstance(OUTPATH, str):
                 plt.savefig(j(OUTPATH, f"{DATAID}-{colnm}-{ix}.png"))
             plt.show()
```

- the above link refers to the latest working version of the simulation workbook on binder
- there are multiple books in this directory, post-fixed with a, b, c etc
- the latest books may be slightly ahead of the pypi releases, so they may not run on Binder until main release has caught up (typically, a few days max)

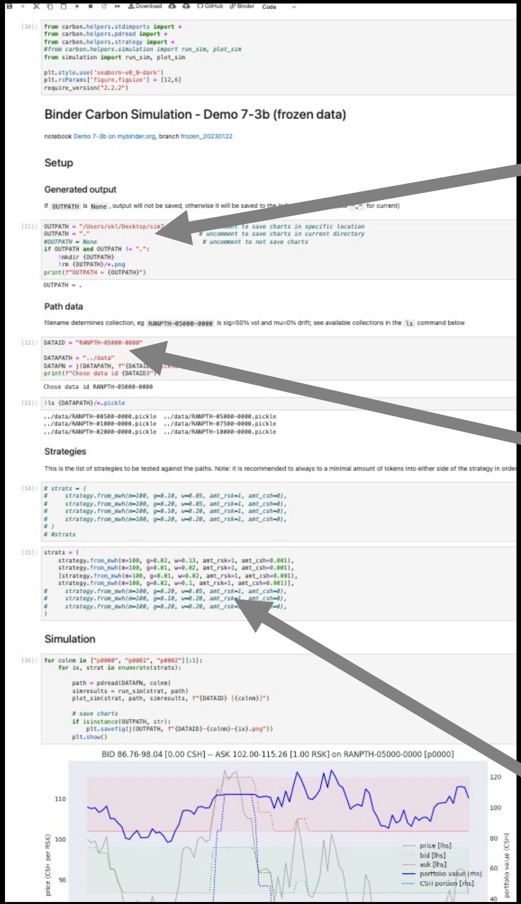
Simulation 7-3 output

[Demo 7-3 (frozen_202302)]



Simulation 7-3 inputs

Demo 7-3 (frozen_202302)



```
[10]: from carbon.helpers.stdimports import *
      from carbon.helpers.pdread import *
      from carbon.helpers.strategy import *
      #from carbon.helpers.simulation import run_sim, plot_sim
      from simulation import run_sim, plot_sim

      plt.style.use('seaborn-v0_8-dark')
      plt.rcParams['figure.figsize'] = [12,6]
      require_version("2.2.2")
```

Binder Carbon Simulation - Demo 7-3b (frozen data)

notebook Demo 7-3b on mybinder.org, branch frozen_20230122

Setup

Generated output

If `OUTPATH` is `None`, output will not be saved, otherwise it will be saved to the indi... (use `"."` for current)

```
[11]: OUTPATH = "/Users/skl/Desktop/sim7...   # uncomment to save charts in specific location
      OUTPATH = "."                           # uncomment to save charts in current directory
      #OUTPATH = None                          # uncomment to not save charts
      if OUTPATH and OUTPATH != ".":
          !mkdir {OUTPATH}
          !rm {OUTPATH}/*.png
      print(f"OUTPATH = {OUTPATH}")

OUTPATH = .
```

Path data

filename determines collection, eg `RANPTH-05000-0000` is sig=50% vol and mu=0% drift; see available collections in the `ls` command below

```
[12]: DATAID = "RANPTH-05000-0000"

      DATAPATH = "../data"
      DATAFN = j(DATAPATH, f"{DATAID}.pickle")
      print(f"Chose data id {DATAID}")

Chose data id RANPTH-05000-0000

[13]: !ls {DATAPATH}/*.pickle

../data/RANPTH-00500-0000.pickle  ../data/RANPTH-05000-0000.pickle
../data/RANPTH-01000-0000.pickle  ../data/RANPTH-07500-0000.pickle
../data/RANPTH-02000-0000.pickle  ../data/RANPTH-10000-0000.pickle
```

Strategies

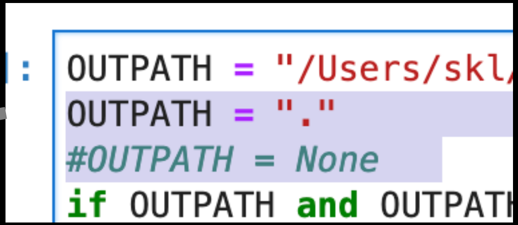
This is the list of strategies to be tested against the paths. Note: it is recommended to always to a minimal amount of tokens into either side of the strategy in order

```
[14]: # strats = (
      #     strategy.from_mwh(m=100, g=0.10, w=0.05, amt_rsk=1, amt_csh=0),
      #     strategy.from_mwh(m=100, g=0.20, w=0.05, amt_rsk=1, amt_csh=0),
      #     strategy.from_mwh(m=100, g=0.10, w=0.20, amt_rsk=1, amt_csh=0),
      #     strategy.from_mwh(m=100, g=0.20, w=0.20, amt_rsk=1, amt_csh=0),
      # )
      # #strats

[15]: strats = (
          strategy.from_mwh(m=100, g=0.02, w=0.13, amt_rsk=1, amt_csh=0.001),
          strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
          [strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
          strategy.from_mwh(m=100, g=0.02, w=0.1, amt_rsk=1, amt_csh=0.001)],
      #     strategy.from_mwh(m=100, g=0.20, w=0.05, amt_rsk=1, amt_csh=0),
      #     strategy.from_mwh(m=100, g=0.10, w=0.20, amt_rsk=1, amt_csh=0),
      #     strategy.from_mwh(m=100, g=0.20, w=0.20, amt_rsk=1, amt_csh=0),
      )
```

Simulation

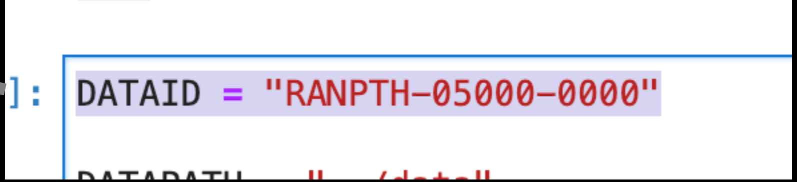
```
[16]: for colnm in ["p0000", "p0001", "p0002"][:1]:
          for ix, strat in enumerate(strats):

              path = pdread(DATAFN, colnm)
              simresults = run_sim(strat, path)
              plot_sim(strat, path, simresults, f"{DATAID} [{colnm}]")

              # save charts
              if isinstance(OUTPATH, str):
                  plt.savefig(j(OUTPATH, f"{DATAID}-{colnm}-{ix}.png"))
              plt.show()
```

saved charts



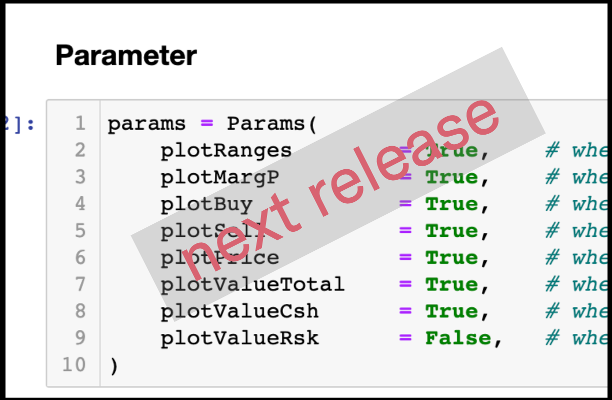
```
]: OUTPATH = "/Users/skl/
   OUTPATH = "."
   #OUTPATH = None
   if OUTPATH and OUTPATH
```

price data



```
]: DATAID = "RANPTH-05000-0000"

   DATAPATH = "../data"
```

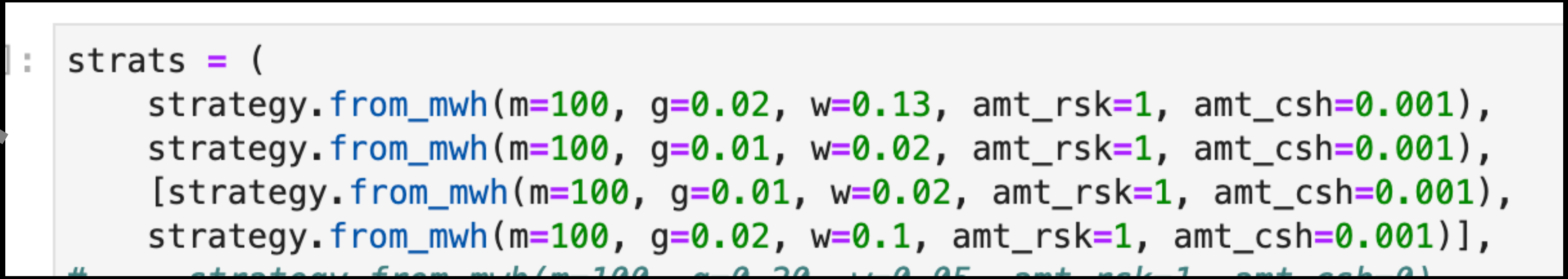
chart content



Parameter

```
]: params = Params(
       plotRanges          = True,      # whe
       plotMargP           = True,      # whe
       plotBuy             = True,      # whe
       plotSell            = True,      # whe
       plotPrice           = True,      # whe
       plotValueTotal      = True,      # whe
       plotValueCsh        = True,      # whe
       plotValueRsk        = False,     # whe
   )
```

Carbon strategies



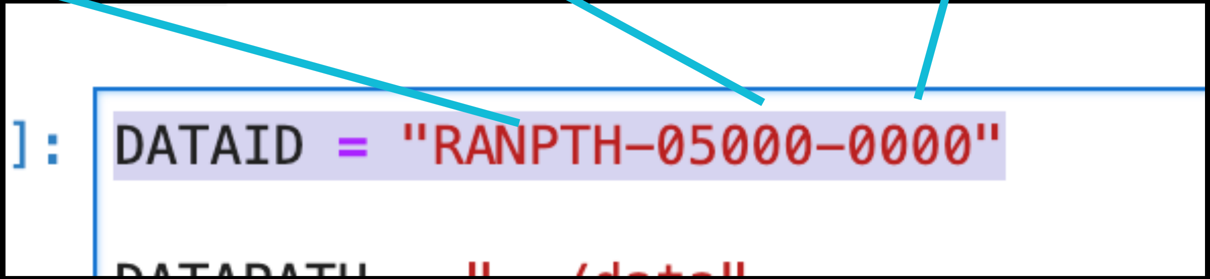
```
]: strats = (
       strategy.from_mwh(m=100, g=0.02, w=0.13, amt_rsk=1, amt_csh=0.001),
       strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
       [strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
       strategy.from_mwh(m=100, g=0.02, w=0.1, amt_rsk=1, amt_csh=0.001)],
   #     strategy.from_mwh(m=100, g=0.20, w=0.05, amt_rsk=1, amt_csh=0),
```

Simulation data

collection of random paths

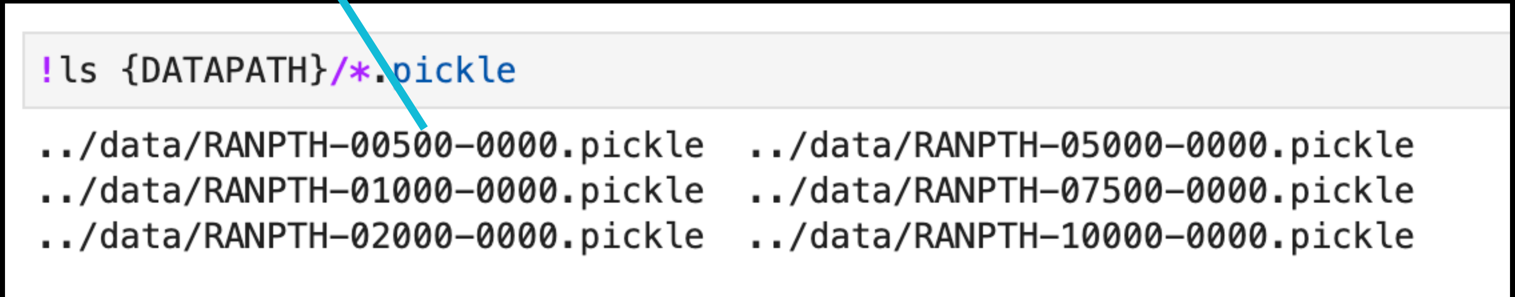
vol sigma = 50%

drift mu = 0%



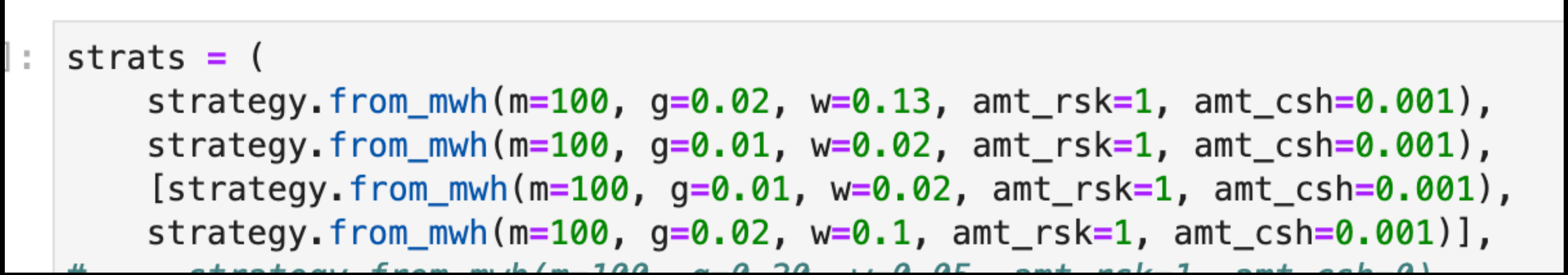
```
]: DATAID = "RANPTH-05000-0000"

   DATAPATH = "../data"
```

all available data (we will add actual market data soon)

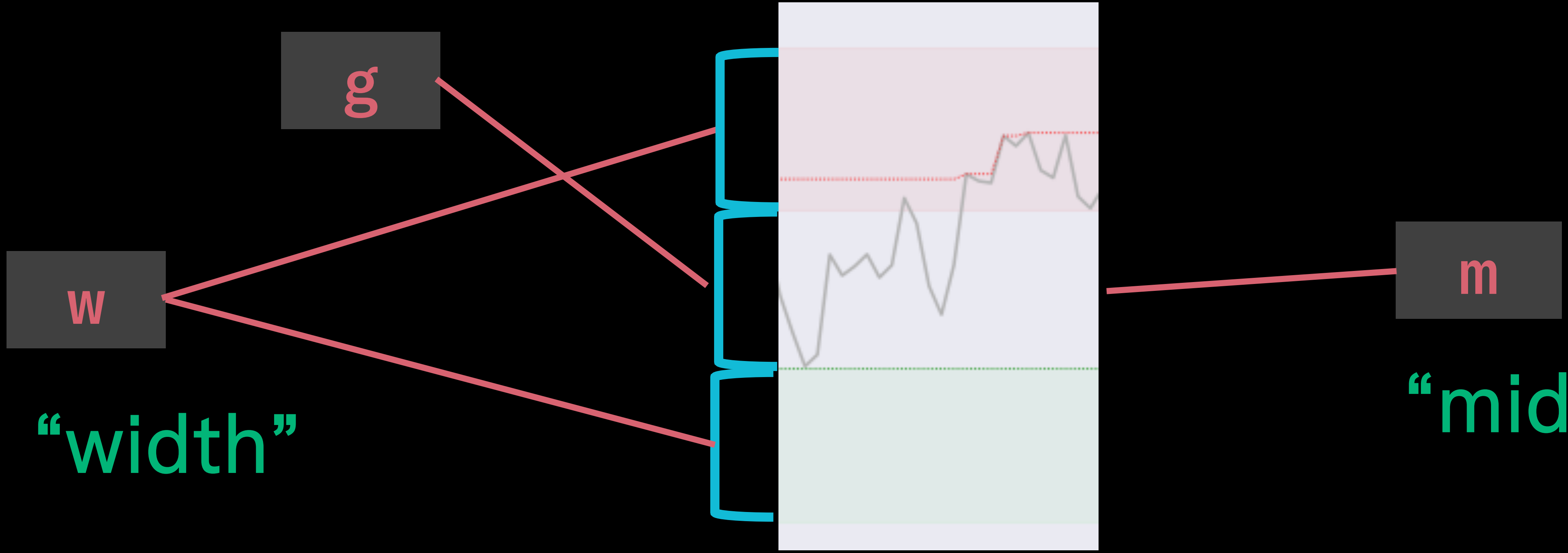


```
!ls {DATAPATH}/*.pickle
../data/RANPTH-00500-0000.pickle  ../data/RANPTH-05000-0000.pickle
../data/RANPTH-01000-0000.pickle  ../data/RANPTH-07500-0000.pickle
../data/RANPTH-02000-0000.pickle  ../data/RANPTH-10000-0000.pickle
```

Parameterizing strategies

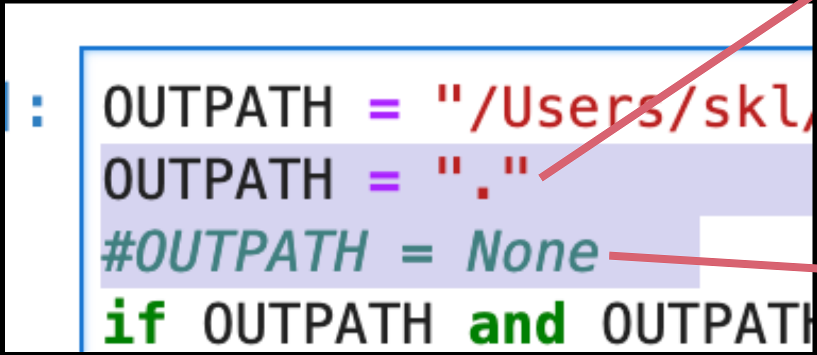


```
]: strats = (
        strategy.from_mwh(m=100, g=0.02, w=0.13, amt_rsk=1, amt_csh=0.001),
        strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
        [strategy.from_mwh(m=100, g=0.01, w=0.02, amt_rsk=1, amt_csh=0.001),
        strategy.from_mwh(m=100, g=0.02, w=0.1, amt_rsk=1, amt_csh=0.001)],
#       strategy.from_mwh(m=100, g=0.20, w=0.05, amt_rsk=1, amt_csh=0)
```



Preserving charts

- as png file
- single Word docx



```
OUTPATH = "/Users/skl/
OUTPATH = "."
#OUTPATH = None
if OUTPATH and OUTPATH
```

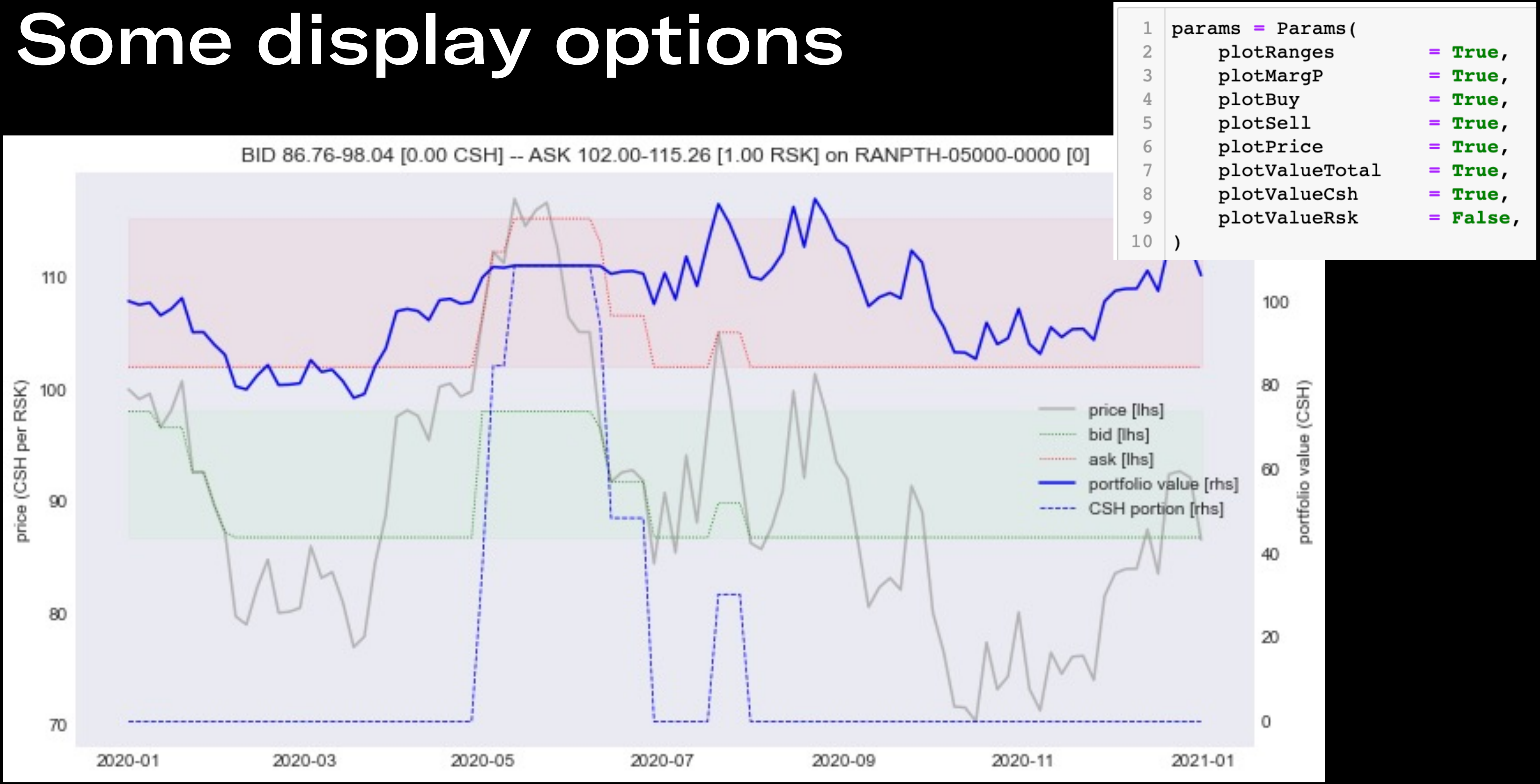
outpath=“.”

every chart generated is written into the output directory; all charts are combined into a single Word document that can be downloaded

outpath=None

charts are not saved

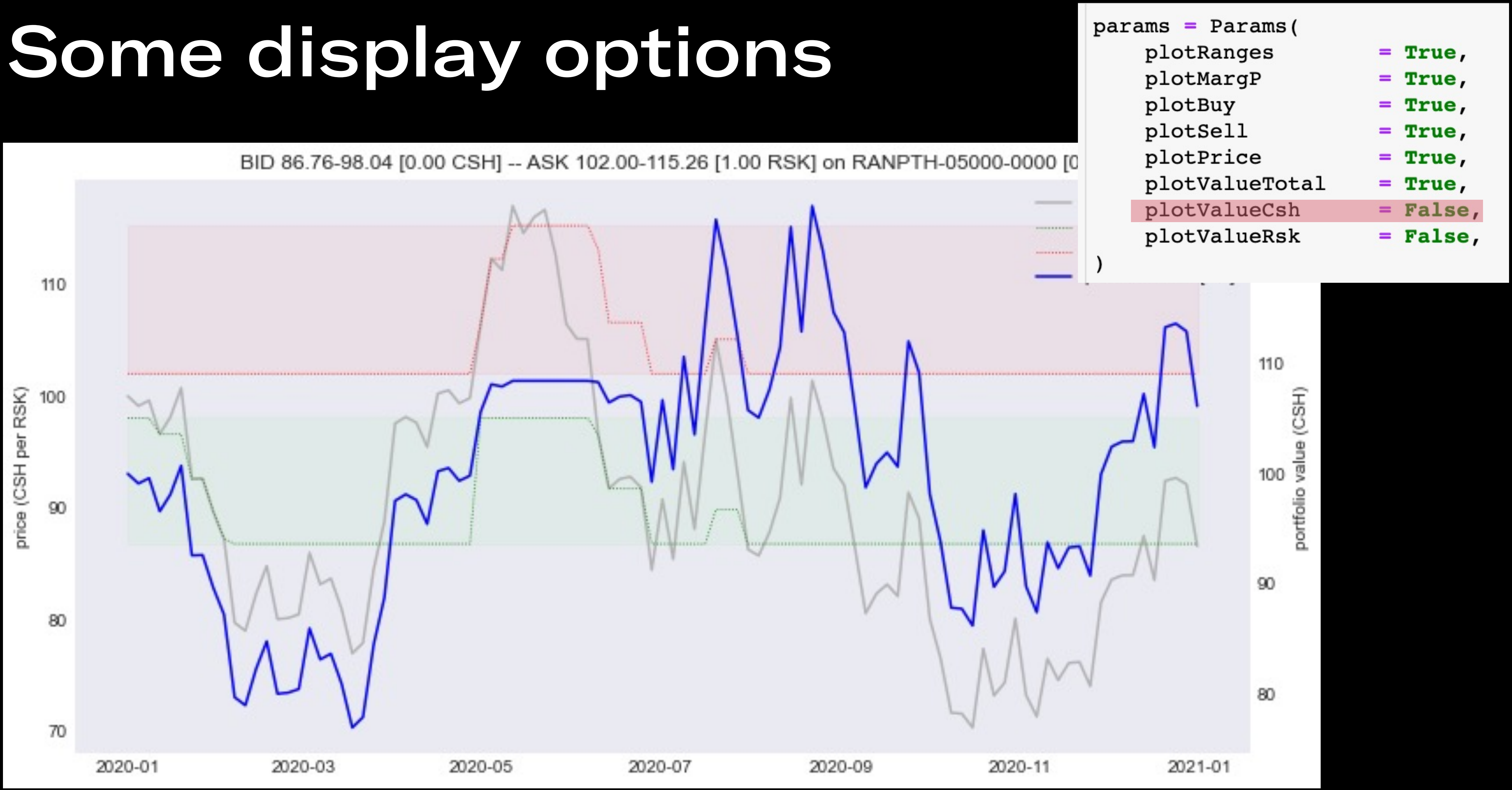
Some display options



BID 86.76-98.04 [0.00 CSH] -- ASK 102.00-115.26 [1.00 RSK] on RANPTH-05000-0000 [0]

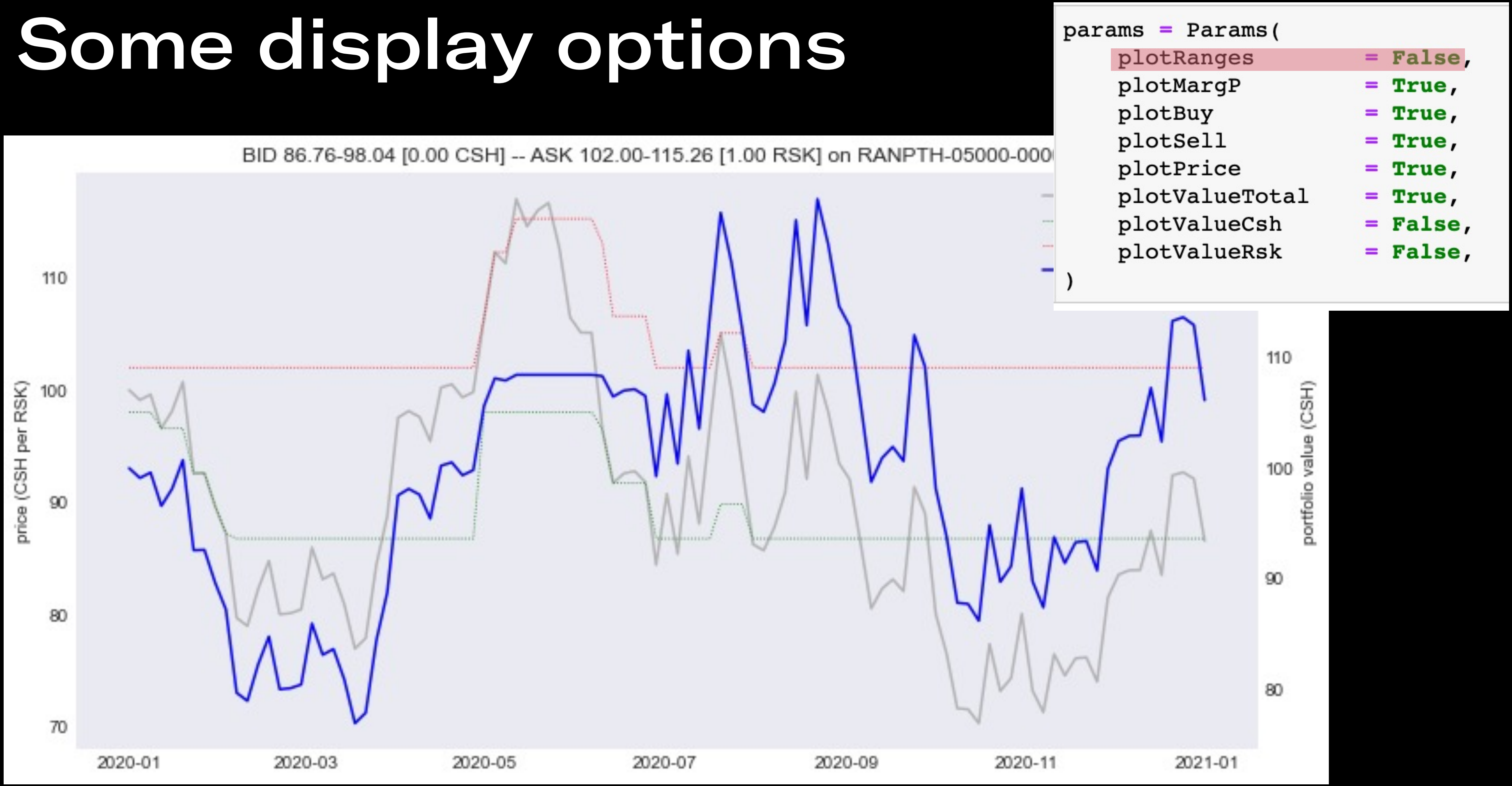
```
params = Params(
    plotRanges          = True,
    plotMargP           = True,
    plotBuy             = True,
    plotSell            = True,
    plotPrice           = True,
    plotValueTotal      = True,
    plotValueCsh        = True,
    plotValueRsk        = False,
)
```

Some display options



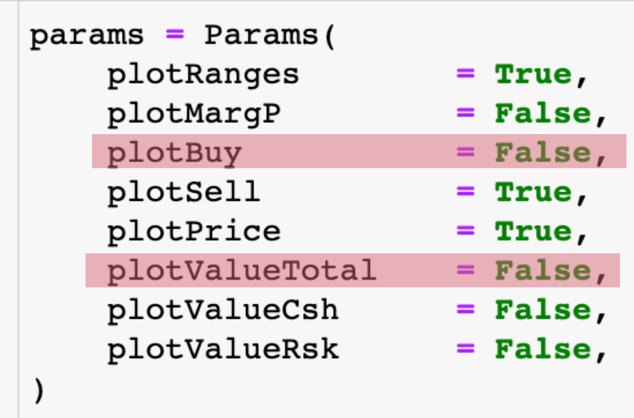
```
params = Params(
    plotRanges          = True,
    plotMargP           = True,
    plotBuy             = True,
    plotSell            = True,
    plotPrice           = True,
    plotValueTotal      = True,
    plotValueCsh        = False,
    plotValueRsk        = False,
)
```

Some display options

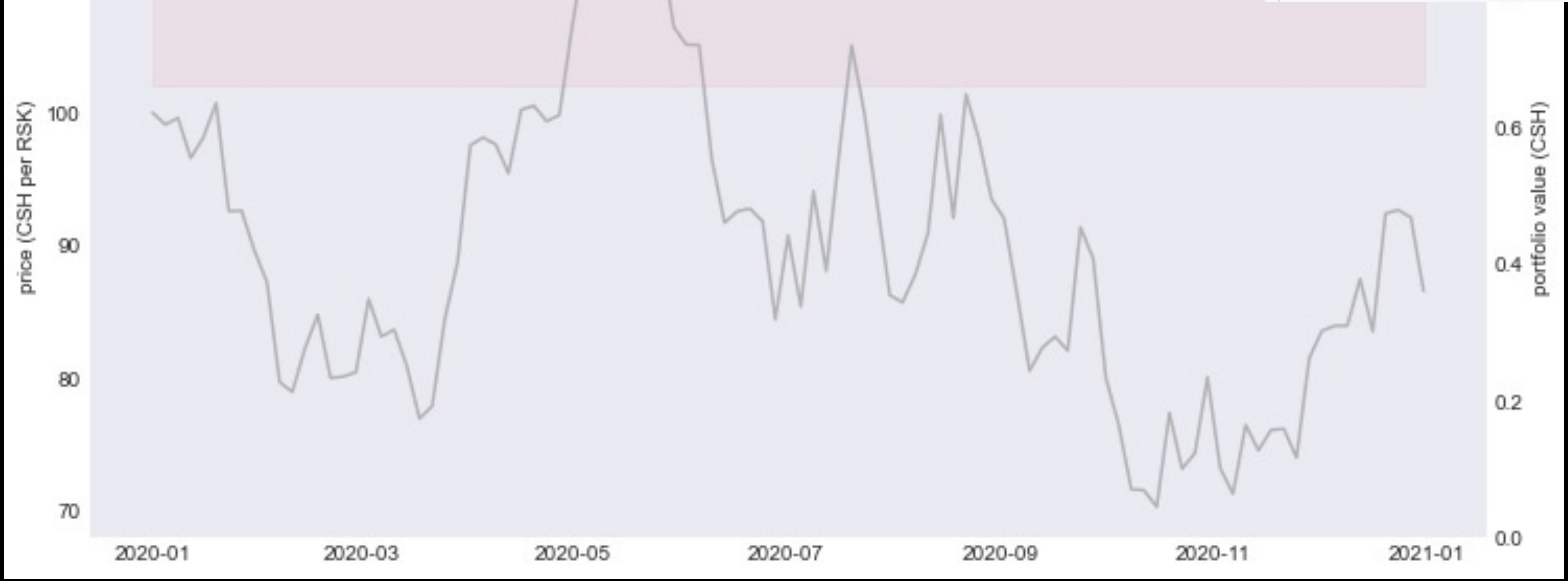


```
params = Params(
    plotRanges          = False,
    plotMargP           = True,
    plotBuy             = True,
    plotSell            = True,
    plotPrice           = True,
    plotValueTotal      = True,
    plotValueCsh        = False,
    plotValueRsk        = False,
)
```

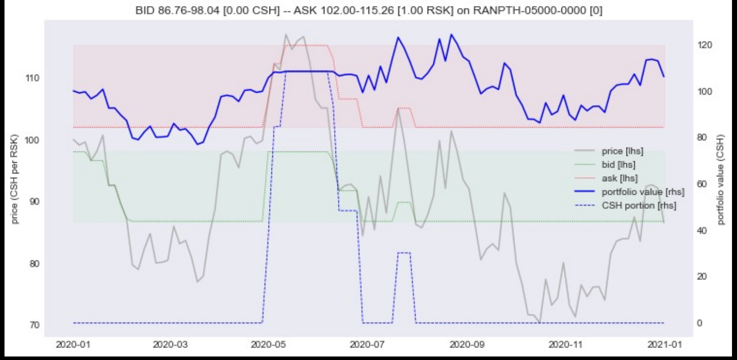
Some display options

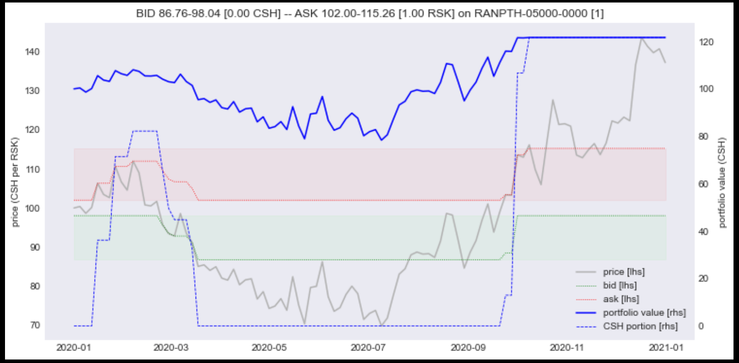


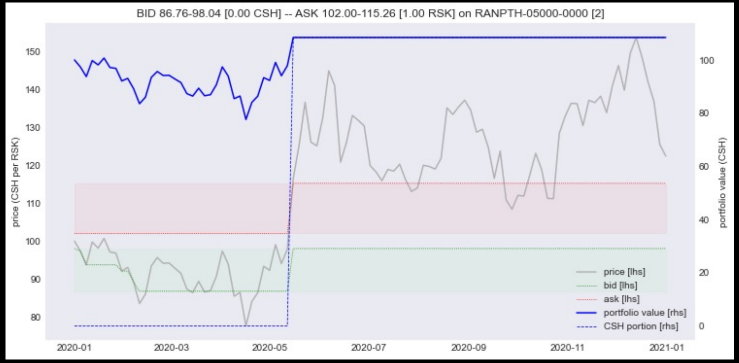
```
params = Params(
    plotRanges          = True,
    plotMargP           = False,
    plotBuy             = False,
    plotSell            = True,
    plotPrice           = True,
    plotValueTotal      = False,
    plotValueCsh        = False,
    plotValueRsk        = False,
)
```

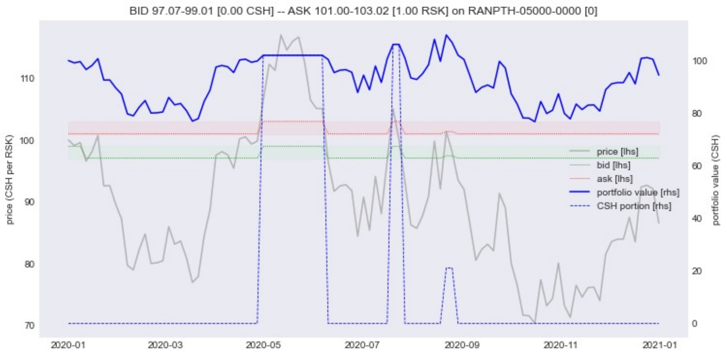


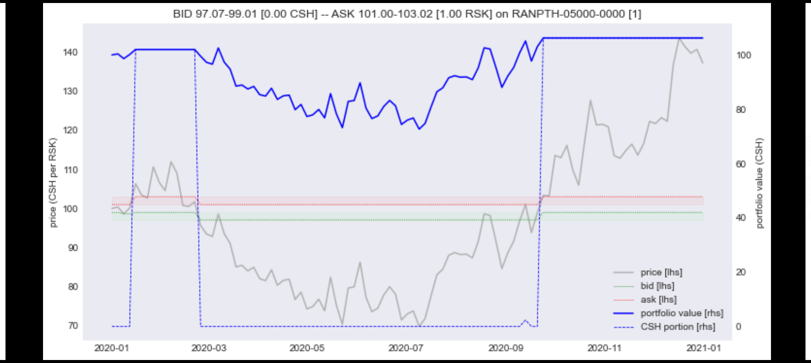
Some example runs

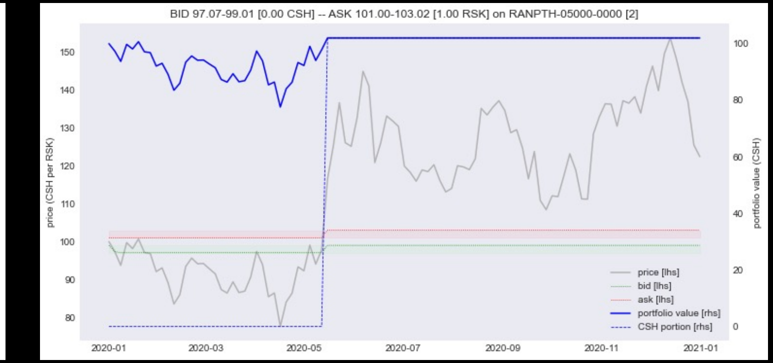












Appendix

References

- This presentation on github
- Carbon Whitepaper on carbondefi.xyz
- Carbon Simulator on github and binder
- Simulator example Demo 7-1 on github and binder
- Simulator example Demo 7-2 on github and binder

NBTest versus Demo notebooks

Demo

- Demo notebooks cover a specific use Carbon use case
- They only contain user-readable demo code, they are not part of the testing pipeline
- Demo books are roughly grouped by thematic area; eg 7-x books are about trading charts
- Demo notebooks are located in [resources/demo]()

NBTest

- NBTest notebooks are developed in line with features of the Carbon library
- They may contain some demo code, but most importantly they contain tests
- NBTest books are sequentially numbered, by time of development of the associated feature
- NBTest notebooks are located in [resources/NBTest]()