Question 1

In [1]:
```
import numpy as np
import pandas as pd
from gurobipy import *
factor = pd.read_csv('FactorBetas.csv')

init = [.06, .06, .009, .05, .055, .065, .051, .005, .055, .15, .1, .07, .15, .1, .02]
bm   =  [.002, .1, 0, .3, .06, .07, .06, 0, .12, .008, .05, .05, .08, .05, .05]
```

In [2]:
```
def optimize (max_trade = 15, LogToConsole = False):
    model = Model()
    buy = model.addVars(15, vtype = GRB.CONTINUOUS, ub = 0.25, lb = 0, name = 'buy')
    sell = model.addVars(15, vtype = GRB.CONTINUOUS, ub = 0.25, lb = 0, name = 'sell')
    yb = model.addVars(15, vtype = GRB.BINARY, name = 'yb')
    ys = model.addVars(15, vtype = GRB.BINARY, name = 'ys')
    y_zero = model.addVars(15, vtype = GRB.BINARY, name = 'y_zero')

    def Variance():
        Mkt = 0
        HML = 0
        SMB = 0
        idio_vol = 0
        for i in range(15):
            weight = (init[i] + buy[i] - sell[i])
            weight_diff = weight - bm[i]
            Mkt += weight_diff * factor['Beta Market'][i]
            HML += weight_diff * factor['Beta HML'][i]
            SMB += weight_diff * factor['Beta SMB'][i]
            idio_vol += 0.1 * weight_diff *0.1 * weight_diff
        return (0.1 * Mkt * 0.1 * Mkt)  + (0.2 * HML* 0.2 * HML) + (0.2 * SMB)*(0.2 * SMB) + idio_vol

    def sum_trade():
        s = 0
        for i in range(15):
            s = s + yb[i] + ys[i]
        return s

    def sum_weights():
        s = 0
        for i in range(15):
            weight = init[i] + buy[i] - sell[i]
            s += weight
        return s

    model.setObjective(Variance(), GRB.MINIMIZE)
    model.addConstr(sum_weights() == 1)
    for i in range(15):
        model.addConstr(buy[i] <= 0.25 * yb[i])
        model.addConstr(buy[i] >= 0.01 * yb[i])
        model.addConstr(sell[i] <= 0.25 * ys[i])
        model.addConstr(yb[i] + ys[i] <= 1)
        model.addConstr((2*y_zero[i]-1)*(init[i] + buy[i] - sell[i]) >= ys[i]*y_zero[i]*0.01)
        if init[i] >= 0.01:
            model.addConstr(sell[i] >= 0.01 * ys[i])
        else:
            model.addConstr(sell[i] == init[i] * ys[i])

    model.addConstr(sum_trade() <= max_trade)
    model.Params.LogToConsole = LogToConsole
    model.optimize()
    model.printAttr("X")
    return model.ObjVal
```

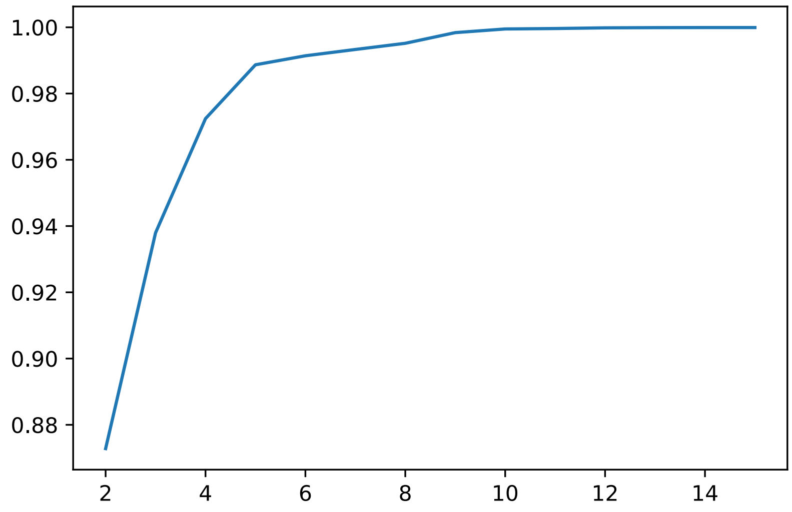
In [3]:
```
var = []
for i in range(0,16):
    var.append(optimize(i))
```

```
Using license file C:\gurobi903\key\gurobi.lic
Academic license - for non-commercial use only
```

In [4]:
```
print("Fontier")
reduction = 1 - np.asarray(var)/var[0]
pd.Series(reduction)[2:].plot()
```

```
Fontier
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ecaf9ce6a0>
```



In [5]:
```
print(reduction)
```

```
[0.         0.         0.87281945 0.93795751 0.97243138 0.98868018
 0.99139915 0.99330319 0.99516927 0.99838919 0.99949548 0.99962708
 0.99983422 0.99989574 0.99992171 0.99992171]
```

In [6]:
```
optimize(5,True)
```

```
Parameter LogToConsole unchanged
   Value: 1  Min: 0  Max: 1  Default: 1
Gurobi Optimizer version 9.0.3 build v9.0.3rc0 (win64)
Optimize a model with 77 rows, 75 columns and 210 nonzeros
Model fingerprint: 0xd1a970a3
Model has 461 quadratic objective terms
Model has 15 quadratic constraints
Variable types: 30 continuous, 45 integer (45 binary)
Coefficient statistics:
  Matrix range     [5e-03, 1e+00]
  QMatrix range    [1e-02, 2e+00]
  QLMatrix range   [1e-02, 1e+00]
  Objective range  [4e-04, 7e-02]
  QObjective range [1e-03, 1e+00]
  Bounds range     [3e-01, 1e+00]
  RHS range        [1e+00, 5e+00]
  QRHS range       [5e-03, 1e-01]
Found heuristic solution: objective 0.0081566
Presolve removed 4 rows and 2 columns
Presolve time: 0.00s
Presolved: 146 rows, 116 columns, 434 nonzeros
Presolved model has 461 quadratic objective terms
Variable types: 56 continuous, 60 integer (60 binary)

Root relaxation: objective 1.088725e-07, 296 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0    0.00000    0   32    0.00816    0.00000   100%     -    0s
H    0     0                       0.0008970    0.00000   100%     -    0s
H    0     0                       0.0008305    0.00000   100%     -    0s
     0     0    0.00000    0   23    0.00083    0.00000   100%     -    0s
H    0     0                       0.0008155    0.00000   100%     -    0s
H    0     0                       0.0007909    0.00000   100%     -    0s
     0     0    0.00000    0   23    0.00079    0.00000   100%     -    0s
     0     0    0.00000    0   23    0.00079    0.00000   100%     -    0s
     0     0    0.00000    0   23    0.00079    0.00000   100%     -    0s
H    0     0                       0.0005305    0.00000   100%     -    0s
H    0     0                       0.0004414    0.00000   100%     -    0s
     0     2    0.00000    0   22    0.00044    0.00000   100%     -    0s
*   64    35              11       0.0003890    0.00002  93.6%  14.6    0s
H   76    43                       0.0002784    0.00003  90.7%  14.0    0s
*   78    43               8       0.0002365    0.00003  89.0%  14.3    0s
*   89    33               8       0.0002339    0.00004  81.4%  14.1    0s
*  154    33              11       0.0002285    0.00006  75.8%  13.1    0s
*  157    33              11       0.0002062    0.00006  73.2%  13.2    0s
*  219    46              15       0.0002058    0.00006  73.1%  13.1    0s
*  230    46              16       0.0001338    0.00006  58.7%  12.9    0s
*  250    30              14       0.0000923    0.00006  37.5%  13.0    0s

Cutting planes:
  Cover: 14
  Implied bound: 15
  MIR: 8
  Flow cover: 3

Explored 308 nodes (3978 simplex iterations) in 0.23 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 9.23316e-05 0.000133781 0.00020575 ... 0.000441419

Optimal solution found (tolerance 1.00e-04)
Best objective 9.233155504707e-05, best bound 9.233155504707e-05, gap 0.0000%

    Variable            X
-------------------------
      buy[3]     0.242115
      buy[8]     0.065684
     sell[0]    0.0612085
     sell[9]         0.15
    sell[12]    0.0965905
       yb[3]            1
       yb[8]            1
       ys[0]            1
       ys[9]            1
```

```
       ys[12]      1
    y_zero[1]      1
    y_zero[2]      1
    y_zero[3]      1
    y_zero[4]      1
    y_zero[5]      1
    y_zero[6]      1
    y_zero[7]      1
    y_zero[8]      1
   y_zero[10]      1
   y_zero[11]      1
   y_zero[12]      1
   y_zero[13]      1
   y_zero[14]      1
```

```
Out[6]: 9.233155504707359e-05
```

At least 5 trades are needed to reduce the variance by 97.5% The corresponding trades are:

notice that the index starts from 0 to 14 instead of 1 to 15

```
   buy[3]    0.242115
   buy[8]    0.065684
  sell[0]   0.0612085
  sell[9]        0.15
 sell[12]   0.0965905
```

Question 2

max $E_0[\ln W_T]$ s.t. $\begin{cases} W_T \geq 0.9 \\ E_0[\xi_T W_T] = W_0 \end{cases}$

Using Lagrange multiplier: max $E_0[\ln W_T - \lambda(\xi_T W_T - W_0)]$

s.t. $W_T \geq 0.9$

FOC: $\frac{1}{W_T^*} - \lambda \xi_T = 0$ $\quad W_T^* = \frac{1}{\lambda \xi_T}$ when $W_T \geq 0.9$ i.e. $\xi_T \leq \frac{1}{0.9\lambda}$

$\Rightarrow W_T^* = \begin{cases} \frac{1}{\lambda \xi_T} & \text{when } \xi_T \leq \frac{1}{0.9\lambda} \\ 0.9 & \text{o/w.} \end{cases}$

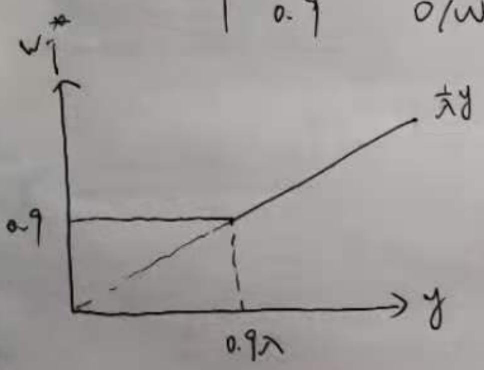
$\frac{d\xi_T}{\xi_T} = -r\,dt - \eta\,dB$

consider $y = \frac{1}{\xi}$ $\quad \frac{\partial y}{\partial T} = 0$ $\quad \frac{\partial y}{\partial \xi} = -\frac{1}{\xi_T^2}$ $\quad \frac{\partial^2 y}{\partial \xi^2} = \frac{2}{\xi_T^3}$

$\Rightarrow \frac{dy}{y} = (r + \eta^2)\,dt + \eta\,dB$

$W_T^* = \begin{cases} \frac{1}{\lambda} y_T & \text{when } y_T \geq 0.9\lambda \\ 0.9 & \text{o/w.} \end{cases}$ where $dy/y = (r + \eta^2)\,dt + \eta\,dB$.

$\boxed{\text{price of } y \text{ at } 0}$ $y_0 = \frac{1}{\xi_0} = 1.$



essentially, this is a portfolio of longing $\frac{1}{\lambda}y$ and $\frac{1}{\lambda}$ put option with $\boxed{\text{expected return } (r + \eta^2)}$, $\sigma = \eta$, $T = 2$, $S = y_0$, $K = 0.9\lambda$

$\frac{1}{\lambda}y_0 + \frac{1}{\lambda}\text{Put}_{,BSM}(r, \eta, 2, y_0, 0.9\lambda) = W_0$

All inputs except λ are known. Letting computer to find the value of λ:

$\lambda^* = 1.13719.$

essentially solution can be found using dichotomy

(d) $\lambda(T) = F(T) S_T^{-\eta/6}$, $F(T) = \dfrac{\exp(-(r+\frac{\eta^2}{2})t)}{S_0^{-\eta/6} \exp[-\frac{\eta}{6}(\mu-\frac{\sigma^2}{2})t]}$

$$y_T = \frac{1}{\lambda_T} = \frac{1}{F(T)} S_T^{\eta/6}$$

The portfolio is $\frac{1}{\lambda^*} y$ and $\frac{1}{\lambda^*}$ put option

$$\frac{\partial W_t^*}{\partial S_t} = \frac{1}{\lambda^*} \frac{dy}{dS} + \frac{1}{\lambda^*} \frac{\partial put}{\partial y} \cdot \frac{dy}{dS}$$

$$\frac{dy}{dS} = \frac{1}{F(T)} \frac{\eta}{6} S_T^{\frac{\eta-6}{6}} \qquad \lambda^* = 1.13719$$

$$\frac{\partial put}{\partial y} = \Delta p = -N(-d_1) \ , \ d_1 = \frac{\ln(y/0.9\lambda^*) + (r+\frac{\eta^2}{2})T}{\eta\sqrt{T}}$$

at $t=0$, $T=2$, $r=0.04$, $\eta = \frac{0.1-0.04}{0.2} = 0.3$, $\lambda^* = 1.13719$

$$\Rightarrow \frac{\partial put}{\partial y} = -0.3398 \ , \ \left.\frac{dy}{dS}\right|_{t=0} = \frac{\eta}{6} = 1.5$$

$$\frac{\partial W_t^*}{\partial S_t} = 1.767279$$

Question 2 - Codes

In [7]:
```
import math
import scipy.stats as stat
import numpy as np
from scipy.optimize import fsolve

def BSM_put(S,sigma,r,K,T):
    x1 = (math.log(S/K) + T * r) / (sigma * math.sqrt(T)) + 0.5 * sigma * math.sqrt(T)
    x2 = (math.log(S/K) + T * r) / (sigma * math.sqrt(T)) - 0.5 * sigma * math.sqrt(T)
    delta = stat.norm.cdf(x1)
    B = - K * math.exp(-r * T) * stat.norm.cdf(x2)
    C = S * delta + B
    P = C + K * math.exp(-r * T) - S

    return P

def _solver(lmbd):
    mu = 0.1
    r = 0.04
    T = 2
    sigma = 0.2
    eta = (mu - r) /sigma

    put = BSM_put(S = 1, sigma = eta, r = r, T = 2, K = 0.9 * lmbd)
    return 1/lmbd + put/lmbd - 1

ans = fsolve(_solver,1)[0]
print(ans)
```

```
1.1371877964962607
```

In [8]:
```
S = 1
K = 0.9 * ans
mu = 0.1
r = 0.04
T = 2
sigma = 0.2
eta = (mu - r) /sigma


x1 = (math.log(S/K) + T * r) / (sigma * math.sqrt(T)) + 0.5 * eta * math.sqrt(T)
stat.norm.cdf(-x1)
```

Out[8]: 0.3398187119335354

In [9]:
```
1/ans * 1.5 + 1/ans * 1.5 * stat.norm.cdf(-x1)
```

Out[9]: 1.7672789613926458

In [ ]: