efficientFrontier

May 4, 2020

0.1 Efficient Frontier / Minimum Variance Portfolio Stuff

table.dataframe { font-size:70%; } body { font-size:70%} /style>

do imports, keep variables in a dict as a workspace

[1]:
```
import datetime
import sympy as sym
from sympy.matrices import matrix_multiply_elementwise as mme
from sympy.plotting import plot as symplot
import IPython.display as disp
import numpy as np
from numpy import linalg as LA
print("imports done")

ws = {} # keep variables in a dict as a workspace
ws['dateStart'] = datetime.datetime.now().isoformat()[:16].replace(':','')
ws
```

```
imports done
```

[1]:
```
{'dateStart': '2020-05-04T1554'}
```

0.1.1 1 Efficient Frontier Formulae Manipulations

This note shows some manipulations for the hyperbola curve of the efficient frontier.

1. Re-arrange efficient frontier equation for risk(σ) and for return(μ)
2. Calculate intermediate scalars A,B,C,D from matrix of covariance and vector of returns, numerically and symbolically
3. Plot hyperbola of risk/return
4. Derive mimimum risk = hyperbola and apex

Reference (using their notation): Beste, Leventhal, Williams, & Dr. Qin Lu "Markowitz Review Paper" http://ramanujan.math.trinity.edu/tumath/research/studpapers/s21.pdf

1.1 hyperbola equation efficient frontier hyperbola:

$$\frac{\sigma^2}{1/C} - \frac{(\mu - A/C)^2}{D/C^2} = 1$$

where:

$$A = \mathbf{1}^T V^{-1} e = e^T V^{-1} \mathbf{1}$$

$$B = e^T V^{-1} e$$

$$C = \mathbf{1}^T V^{-1} \mathbf{1}$$

$$D = BC - A^2$$

e = expected returns vector

V = covariance matrix

$$\mathbf{1} = IdentityMatrix$$

e^T = e transpose

1.2: solve for sigma: σ

$$\frac{\sigma^2}{(1/C)} - \frac{(\mu - A/C)^2}{(D/C^2)} = 1$$

$$\frac{\sigma^2}{(1/C)} = 1 + \frac{(\mu - A/C)^2}{(D/C^2)}$$

divide by C:

$$\frac{\sigma^2}{(C/C)} = \frac{1}{C} + \frac{(\mu - A/C)^2}{(DC/C^2)}$$

$$\sigma^2 = \frac{1}{C} + \frac{(\mu - A/C)^2}{(D/C)}$$

$$\sigma^2 = \frac{1}{C} + \frac{(\mu - A/C)^2 C}{D}$$

$$\sigma^2 = \frac{1}{C} + \frac{\mu^2 C - 2\mu A + A^2/C}{D}$$

$$\sigma^2 = \frac{D + \mu^2 C^2 - 2\mu AC + A^2}{CD}$$

$$\sigma^2 = \frac{D + (\mu C - A)^2}{CD}$$

$$\sigma = \sqrt{\frac{D + (\mu C - A)^2}{CD}}$$

1.3: solve for mu: μ

$$\frac{\sigma^2}{(1/C)} - \frac{(\mu - A/C)^2}{(D/C^2)} = 1$$

$$\frac{\sigma^2}{(1/C)} - 1 = \frac{(\mu - A/C)^2}{(D/C^2)}$$

multiply through by \$ D/C^2 \$

$$\frac{\sigma^2 D}{(C^2/C)} - \frac{D}{C^2} = (\mu - A/C)^2$$

$$\sqrt{\frac{\sigma^2 D}{C} - \frac{D}{C^2}} = (\mu - A/C)$$

$$\sqrt{\frac{D(\sigma^2 C - 1)}{C^2}} = (\mu - A/C)$$

$$\mu = \frac{\sqrt{D(\sigma^2 C - 1)}}{C} + A/C$$

$$\mu = \frac{\sqrt{D(\sigma^2 C - 1)} + A}{C}$$

1.4: efficient frontier minimum σ efficient frontier hyperbola coordinates of minimum:

$$(\sigma, \mu) = (\sqrt{1/C}, (A/C))$$

1.5: (sigma, mu) coordinates at minimum σ

$$\sigma^2 = \frac{D + (\mu C - A)^2}{CD}$$

let $\mu = A/C$

$$\sigma^2 = \frac{D + ((A/C)C - A)^2}{CD}$$

$$\sigma^2 = \frac{D + (A - A)^2}{CD}$$

$$\sigma^2 = \frac{1}{C}$$

0.1.2 2: manipulate equation using sympy - solve for σ

take positive solution only

[2]:
```
mu, sigma, A, B, C, D = sym.symbols('mu sigma A B C D')
ws['sigmaEqn'] = sym.solve(sym.Eq( (sigma**2 / (1/C)) - ((mu - A/C)**2 / (D /
 ↪C**2)), 1) , sigma)[1] # [1]=> +ve soln
ws['sigmaEqn']
```

[2]:

$$\sqrt{\frac{A^2 - 2AC\mu + C^2\mu^2 + D}{CD}}$$

simplify:

[3]:
```
sym.factor(sym.Eq(sigma, ws['sigmaEqn']))
```

[3]:

$$\sigma = \sqrt{\frac{A^2}{CD} - \frac{2A\mu}{D} + \frac{C\mu^2}{D} + \frac{1}{C}}$$

check: [original form] minus [simplified (``factored'') form]:

[4]:
```
ws['sigmaEqn'] - ws['sigmaEqn']
```

[4]: 0

gives:

[5]:
```
sym.factor(ws['sigmaEqn'] - sym.factor(ws['sigmaEqn']))
```

[5]: 0

0.1.3 3 numerical example

Calculate A, B, C, & D, hence σ and μ, for a small example of 3 assets

sample annualized expected returns, in percent:

[6]:
```
ws['prec'] = 4 # number of digits of precision to display numerical values

ws['mu3'] = sym.Matrix(np.array([5.1, 7.0, 0.9]).T) # mu3 = sym.Matrix(mu3)
ws['mu3']
```

[6]:
$$\begin{bmatrix}5.1\\7.0\\0.9\end{bmatrix}$$

cor: sample correlations:

[7]:
```
ws['cor3'] = sym.Matrix([[  1.0,  0.5,  0.4],
                         [  0.5,  1.0, -0.1],
                         [  0.4, -0.1, 1.0]])
sym.N(ws['cor3'], ws['prec'])
```

[7]:
$$\begin{bmatrix}1.0 & 0.5 & 0.4\\0.5 & 1.0 & -0.1\\0.4 & -0.1 & 1.0\end{bmatrix}$$

vol: sample vols (stdev):

[8]:
```
ws['vol3'] = sym.Matrix([ 3.5,  4.2,  1.1])
sym.N(ws['vol3'], ws['prec'])
```

[8]:
$$\begin{bmatrix}3.5\\4.2\\1.1\end{bmatrix}$$

cov: compose to make covariance matrix:

[9]:
```
ws['cov3'] = mme(ws['vol3'] * ws['vol3'].T, ws['cor3']) # mme = element-wise␣
↪multiply
sym.N(ws['cov3'], ws['prec'])
```

[9]:
$$\begin{bmatrix} 12.25 & 7.35 & 1.54 \\ 7.35 & 17.64 & -0.462 \\ 1.54 & -0.462 & 1.21 \end{bmatrix}$$

check that

$$variance = vol^2$$

,

$$diag(cov) = vol^2$$

[10]:
```
sym.diag(*ws['vol3'])**2   # how to do sqrt of diagonal matrix in sympy?
```

[10]:
$$\begin{bmatrix} 12.25 & 0 & 0 \\ 0 & 17.64 & 0 \\ 0 & 0 & 1.21 \end{bmatrix}$$

check: get correlations back from covariance

in index subscript form:

$$r_{ij} = \frac{c_{ij}}{c_{ii} * c_{jj}}$$

in matrix form:

$$cor = vol^{-1} \times cov \times vol^{-1}$$

where

$$vol = \sqrt{diag(cov)}$$

as a diagonal matrix

vol:

[11]:
```
ws['oneOverVol'] = sym.diag(*ws['cov3'].diagonal())**(-0.5) # works!!!! using␣
↪sym.sqrt doesn't evaluate fully
ws['oneOverVol'] * ws['cov3'] * ws['oneOverVol']            # oneOverVol is␣
↪diagonal matrix so it's equal to its transpose
```

[11]:
$$\begin{bmatrix} 1.0 & 0.5 & 0.4 \\ 0.5 & 1.0 & -0.1 \\ 0.4 & -0.1 & 1.0 \end{bmatrix}$$

3.1 sample values for calculating sample hyperbolae scalars A,B,C,D A, B, C, D

calculated numerically as variables a,b,c,d:

vector of 3 ones:

[12]:
```
ws['ones3'] = sym.Matrix([1,1,1])
ws['ones3']
```

[12]:

$$\begin{bmatrix}1\\1\\1\end{bmatrix}$$

inverse of *cov*

[13]:
```
sym.N(ws['cov3']**(-1), ws['prec'])
```

[13]:

$$\begin{bmatrix}0.1497 & -0.06803 & -0.2164\\-0.06803 & 0.08818 & 0.1202\\-0.2164 & 0.1202 & 1.148\end{bmatrix}$$

check condition number of *cov* inverse:

[14]:
```
sym.N(LA.cond(np.array(ws['cov3']**(-1), dtype=float)), ws['prec'])
```

[14]:

27.53

calculate A from covariance $V(=cov)$, and $e(=mu)$:

$$A = \mathbf{1}^T V^{-1} e = e^T V^{-1} \mathbf{1}$$

[15]:
```
ws['a'] = ws['ones3'].T @ ws['cov3']**(-1) @ ws['mu3'] # = (mu3.T @ cov3**(-1)␣
↪@ ones3.T)
sym.N(ws['a'], ws['prec'])
```

[15]:

$$\begin{bmatrix}1.242\end{bmatrix}$$

calculate B from covariance $V(=cov)$, and $e(=mu) : B = e^T V^{-1} e$

[16]:
```
ws['b'] = ws['mu3'].T @ ws['cov3']**(-1) @ ws['mu3']
sym.N(ws['b'], ws['prec'])
```

[16]:

$$\begin{bmatrix}3.814\end{bmatrix}$$

calculate C from covariance

$$V(=cov) : C = \mathbf{1}^T V^{-1} \mathbf{1}$$

[17]:
```
ws['c']  = ws['ones3'].T @ ws['cov3']**(-1) @ ws['ones3']
sym.N(ws['c'], ws['prec'])
```

[17]:

$$\begin{bmatrix}1.057\end{bmatrix}$$

calculate D from covariance: $D = BC - A^2$

[18]:
```
ws['d'] = ws['b'] * ws['c'] - ws['a']**2
sym.N(ws['d'], ws['prec'])
```

[18]:

$$\begin{bmatrix}2.491\end{bmatrix}$$

3.2 hence calculate σ or μ using A, B, C, D calculate σ and μ from each other

$$\sigma = \sqrt{\frac{D + (\mu C - A)^2}{CD}}$$

and

$$\mu = \frac{\sqrt{D(\sigma^2 C - 1)} + A}{C}$$

e.g. for $\mu = 0.3$, $\sigma =$

[19]:
```
ws['sgma'] = ( (ws['d'] + (0.03 * ws['c'] - ws['a'])**2) / (ws['c']*ws['d'])␣
↪)**(0.5)
sym.N(ws['sgma'], ws['prec'])
```

[19]: $[1.226]$

e.g. for $\sigma = 3.086$, $\mu =???$

[20]:
```
print('mu from sigma)')
# check calculate back mu from sigma
((ws['d'] * ((2**2 * ws['c']) + sym.Matrix([[1]])) )**(0.5) + ws['a'] ) /␣
↪ws['c']

#sym.N(m, sPrec)
```

mu from sigma)

[20]: $[4.58803767855236]$

[21]:
```
# mimimum:
print("sigma, mu")
( (sym.Matrix([1]) / ws['c'])**(0.5), ws['a'] / ws['c'] )
```

sigma, mu

[21]: (Matrix([[0.972557105896013]]), Matrix([[1.17453312167395]]))

0.1.4 4: Symbolic Plot

sigma vs mu:

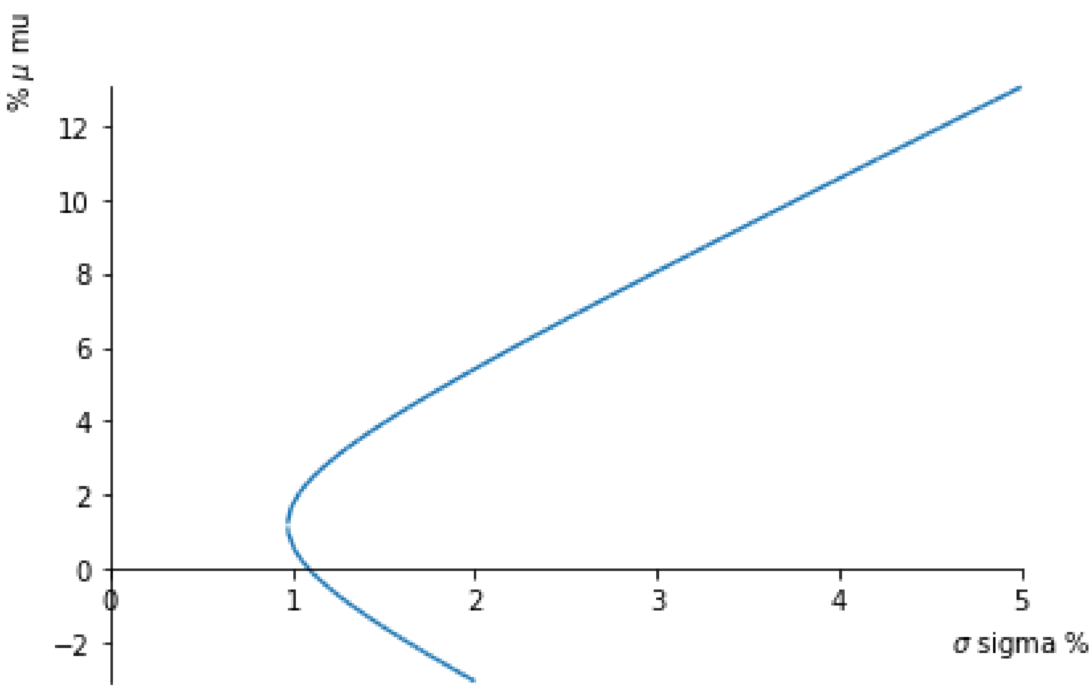
$$\sigma = \sqrt{\frac{D + (\mu C - A)^2}{CD}}$$

mu vs sigma

$$\mu = \frac{A + \sqrt{D(\sigma^2 C - 1)}}{C}$$

[22]:
```
symOne = sym.Matrix([[1]]) # just 1 in a sympy 1x1 Matrix
ws['fmuA'] = ( ws['a'] + ws['d'] * (sigma**2 * ws['c'] - symOne)**0.5 ) /␣
↪ws['c']  # upper half of hyperbola
ws['fmuB'] = ( ws['a'] - ws['d'] * (sigma**2 * ws['c'] - symOne)**0.5 ) /␣
↪ws['c']  # lower half of hyperbola

p0 = symplot(ws['fmuA'][0], (sigma, 0, 5.0), axis_center=(0.0,0.0), ylabel='%␣
↪$\mu$ mu', xlabel='$\sigma$ sigma %', show=False)
p1 = symplot(ws['fmuB'][0], (sigma, 0, 2.0), axis_center=(0.0,0.0), ylabel='%␣
↪$\mu$ mu', xlabel='$\sigma$ sigma %', show=False)
p0.append(p1[0])
p0.show()
```



0.2 5: Closed-form formulae

for small portfolio (3 assets) - covariance V

symbolic form of hyperbola in terms of asset covariances and returns:

(note the symmetric off-diagonal entries)

[23]:
```
u,s,A,B,C,D,E, s1,s2,s3, cv12,cv13,cv23, r0,r1,r2 = \
    sym.symbols('u s A B C D E  s1 s2 s3 cv12 cv13 cv23  r0 r1 r2')
```

```python
V = sym.Matrix([[s1**2, cv12,  cv13],
                [cv12,  s2**2, cv23],
                [cv13,  cv23,  s3**2]])
V
```

[23]:

$$\begin{bmatrix} s_1^2 & cv_{12} & cv_{13} \\ cv_{12} & s_2^2 & cv_{23} \\ cv_{13} & cv_{23} & s_3^2 \end{bmatrix}$$

inverse of covariance matrix V:

[24]:
```python
V**(-1)
```

[24]:

$$\begin{bmatrix} \frac{s_1^2 s_2^2 ((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)-(-cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)+cv_{13}(-cv_{12}^2+s_1^2 s_2^2))(cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)-cv_{13}(\cdots}{s_1^2(-cv_{12}^2+s_1^2 s_2^2)((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)} \cdots \\ \frac{-cv_{12}((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)(cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)-cv_{13}(-cv_{12}^2+s_1^2 s_2^2)\cdots}{(-cv_{12}^2+s_1^2 s_2^2)((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)} \cdots \\ \frac{cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)-cv_{13}(-cv_{12}^2+s_1^2 s_2^2)}{(-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2} \cdots \end{bmatrix}$$

check multiply: $V \times V^{-1}$

[25]:
```python
sym.MatMul(V, V.inv(),doit=False)
```

[25]:

$$\begin{bmatrix} s_1^2 & cv_{12} & cv_{13} \\ cv_{12} & s_2^2 & cv_{23} \\ cv_{13} & cv_{23} & s_3^2 \end{bmatrix} \begin{bmatrix} \frac{s_1^2 s_2^2 ((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)-(-cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)+cv_{13}(-cv_{12}^2+s_1^2 s_2^2))(cv_{12}(-\cdots}{s_1^2(-cv_{12}^2+s_1^2 s_2^2)((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)} \cdots \\ \frac{-cv_{12}((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)(cv_{12}(-cv_{12}cv_{13}+cv_{2}\cdots}{(-cv_{12}^2+s_1^2 s_2^2)((-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2)} \cdots \\ \frac{cv_{12}(-cv_{12}cv_{13}+cv_{23}s_1^2)-cv_{13}(-cv_{12}^2+s_1^2 s_2^2)}{(-cv_{12}^2+s_1^2 s_2^2)(-cv_{13}^2+s_1^2 s_3^2)-(-cv_{12}cv_{13}+cv_{23}s_1^2)^2} \cdots \end{bmatrix}$$

=

[26]:
```python
sym.simplify(V @ V.inv())
```

[26]:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[27]:
```python
#sym.ask(sym.Q.symmetric(V.inv())) # only avaulable in SageMath/Cocalc?
```

[28]:
```python
vi00,vi11,vi22, vi01,vi02,vi12 = sym.symbols('vi00, vi11, vi22, vi01, vi02,␣
↪vi12')
sA,sB,sC,sD = sym.symbols('sA, sB, sC, sD') # keep separate from symbols␣
↪A,B,C,D above
Vi = sym.Matrix([[vi00, vi01, vi02],
                 [vi01, vi11, vi12],
                 [vi02, vi12, vi22]])
Vi
```

[28]:

$$\begin{bmatrix} vi_{00} & vi_{01} & vi_{02} \\ vi_{01} & vi_{11} & vi_{12} \\ vi_{02} & vi_{12} & vi_{22} \end{bmatrix}$$

returns: E =

[29]:
```python
E = sym.Matrix([r0,r1,r2])
E
```

[29]:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \end{bmatrix}$$

hence closed form formulas for A, B, C, D:

$$A = \mathbf{1}^T V^{-1} e = e^T V^{-1} \mathbf{1} =$$

[30]:
```python
sA = ws['ones3'].T @ V.inv() @ E
sA
```

[30]:

$$\Bigg[r_0 \Bigg(\frac{cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)-cv_{13}\left(-cv_{12}^2+s_1^2s_2^2\right)}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2} + \frac{-cv_{12}\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s \cdots\right.}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv \cdots\right.\right.}$$

$ B = e^T V^{-1} e = $

[31]:
```python
# B
sB = E.T @ V.inv() @ E
sB
```

[31]:

$$\Bigg[r_0 \Bigg(\frac{r_0\left(s_1^2s_2^2\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2\right)-\left(-cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)+cv_{13}\left(-cv_{12}^2+s_1^2s_2^2\right)\right)\left(cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s \cdots\right.\right.\right.}{s_1^2\left(-cv_{12}^2+s_1^2s_2^2\right)\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2\right)}$$

$$C = \mathbf{1}^T V^{-1} \mathbf{1} =$$

[32]:
```python
sC = ws['ones3'].T @ V.inv() @ ws['ones3']
sC
```

[32]:

$$\Bigg[\frac{s_1^2\left(-cv_{12}^2+s_1^2s_2^2\right)}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2} - \frac{2s_1^2\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2} - \frac{-cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)+ \cdots}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)- \cdots}$$

$ D = BC - A^2 $

[33]:
```python
sD = sB @ sC - sA**2
sD
```

[33]:

$$\Bigg[- \Bigg(r_0 \Bigg(\frac{cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)-cv_{13}\left(-cv_{12}^2+s_1^2s_2^2\right)}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2} + \frac{-cv_{12}\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2\right)-\left(-cv_{12}cv_{13}+ \cdots\right.}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)- \cdots\right.}$$

[34]:
```python
# hmmmm
sym.factor(sD)
```

[34]:

$$\left[-\left(r_0\left(\frac{cv_{12}\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)-cv_{13}\left(-cv_{12}^2+s_1^2s_2^2\right)}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2}+\frac{-cv_{12}\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\left(-cv_{12}cv_{13}+cv_{23}s_1^2\right)^2\right)-\left(-cv_{12}cv_{13}+c\right.}{\left(-cv_{12}^2+s_1^2s_2^2\right)\left(\left(-cv_{12}^2+s_1^2s_2^2\right)\left(-cv_{13}^2+s_1^2s_3^2\right)-\right.}\right.\right.\right.$$

workspace final contents:

[35]:
```python
ws['dateEnd'] = datetime.datetime.now().isoformat()[:16].replace(':','')

for k in sorted(ws): print("%10s" % k, type(ws[k]))
```

```
         a <class 'sympy.matrices.dense.MutableDenseMatrix'>
         b <class 'sympy.matrices.dense.MutableDenseMatrix'>
         c <class 'sympy.matrices.dense.MutableDenseMatrix'>
      cor3 <class 'sympy.matrices.dense.MutableDenseMatrix'>
      cov3 <class 'sympy.matrices.dense.MutableDenseMatrix'>
         d <class 'sympy.matrices.dense.MutableDenseMatrix'>
   dateEnd <class 'str'>
 dateStart <class 'str'>
      fmuA <class 'sympy.matrices.immutable.ImmutableDenseMatrix'>
      fmuB <class 'sympy.matrices.immutable.ImmutableDenseMatrix'>
       mu3 <class 'sympy.matrices.dense.MutableDenseMatrix'>
oneOverVol <class 'sympy.matrices.dense.MutableDenseMatrix'>
     ones3 <class 'sympy.matrices.dense.MutableDenseMatrix'>
      prec <class 'int'>
      sgma <class 'sympy.matrices.immutable.ImmutableDenseMatrix'>
  sigmaEqn <class 'sympy.core.power.Pow'>
      vol3 <class 'sympy.matrices.dense.MutableDenseMatrix'>
```

[ ]: