AMS-512 Capital Markets and Portfolio Theory

Factor Models: Theory & Estimation

Robert J. Frey, Research Professor
Stony Brook University, Applied Mathematics and Statistics

frey@ams.sunysb.edu
http://www.ams.sunysb.edu/~frey

```
In[•]:= dir = NotebookDirectory[]

Out[•]= /Users/robertjfrey/Documents/Work/Stony Brook
          University/AMS/QF/public_html/Instruction/Spring2020/AMS512/Class05/
```

Set Up

Packages

Local Packages

```
In[•]:= << Local`SaveRecall`
```

Locales

Notebook

```
In[•]:= xResolveLocale[] := Module[
          {path, file, base, ext, proj, numb},
          {path, file} =
           Through[{FileNameDrop[#, -1] &, FileNameTake[#, -1] &}[NotebookFileName[]]];
          {base, ext} = Through[{FileBaseName, FileExtension}[file]];
          numb = Last@StringSplit[base, RegularExpression["\\D"]];
          proj = StringDrop[base, -StringLength[numb]];
          {path, file, base, ext, proj, numb}
         ]
```

```
In[∘]:= {sPath, sName, sBase, sExt, sProj, sNumber} = xResolveLocale[]

   ••• Last: {} has zero length and no last element.

   ••• StringLength: String expected at position 1 in StringLength[Last[{}]].

   ••• StringDrop: Sequence specification (+n, −n, {+n}, {−n}, {m, n}, or {m, n, s}) expected at position 2 in StringDrop[
         FactorModels, −StringLength[Last[{}]]].

Out[∘]= {/Users/robertjfrey/Documents/Work/Stony Brook
           University/AMS/QF/public_html/Instruction/Spring2020/AMS512/Class05,
         FactorModels.nb, FactorModels, nb, StringDrop[FactorModels,
          -StringLength[Last[{}]]], Last[{}]}
```

Calendar Functions

Both the Sort[], Ordering[] and Order[] functions behave in reasonable ways when dealing with dates in *Mathematica*'s {year, minth, day} format. They are used extensively and understanding how they work with dates is important to understanding how these functions work. For example, sorting an $n \times 2$ matrix vxSeries of date-datum pairs can be done as simply as vxSeries⟦Ordering[vxSeries⟦All, 1⟧]⟧

The convention for monthly calendars is to use the last calendar day of the month as the date value for each datum. This function converts a date in {year, month, day} format to the last day of the given year and month. Converting a sequence of dates is often accomplished by xEndOfMonth /@ vxCalendar.

Sometimes dates for monthly financial time series are reported using the last trading day of the month. The xEndOfMonth[] can be used to standardize month ending dates to be the literal last calendar day of the month.

```
In[∘]:= xEndOfMonth[{y_, m_, d_}] :=
          If[m == 12, {y, m, 31}, DateList[DayPlus[{y, m + 1, 1}, -1]]⟦1 ;; 3⟧];
```

Selecting dates out of a time series is normally done by using the xDataRangeSelect[] function. It uses Select[], does not assume that the dates are in any particular order and, therefore, works correctly whether they are or not. However, it does assume that the StartDate occurs at or before the EndDate.

```
In[∘]:= xDateRangeSelect[vxCalendar_, {StartDate_, EndDate_}] := Select[
           vxCalendar,
           (Order[StartDate, #] ≥ 0 && Order[EndDate, #] ≤ 0) &
          ];
```

The function xBuildMonthlyCalendar[] takes a specification in the fom of {{start year, start month}, {end year, end month}} and produces a calendar of month ending dates from the starting date to the ending date, inclusive. The calendar is intended to be complete and the dates are properly sorted.

```
In[•]:= xBuildMonthlyCalendar[
    {{iStartYear_, iStartMonth_}, {iEndYear_, iEndMonth_}}] := Module[
    {vviMonths, vviYears},
    vviMonths = Split[
      Mod[
       Range[0,
          ({iEndYear, iEndMonth} - {iStartYear, iStartMonth}).{12, 1}] + iStartMonth,
       12,
       1],
      #1 < #2 &
     ];
    vviYears = {#} & /@ Range[iStartYear, iEndYear];
    xEndOfMonth /@ Flatten[
      MapThread[Outer[{#1, #2, 01} &, #1, #2] &, {vviYears, vviMonths}],
      2
     ]
   ];
```

The function xCompleteMonthlyCalendarQ[] tests whether a given calendar completely covers its date range with sorted proper end of month dates.

```
In[•]:= xCompleteMonthlyCalendarQ[vxCalendar_] := xBuildMonthlyCalendar[
    Most /@ Through[{First, Last}[Sort[vxCalendar]]]] == vxCalendar
```

Time Series Functions

There are a two simple conventions used to represent time series. The first is a simple time series of scalar values usually designated something like vxSeries which a vector of {date, scalar} pairs. Although a matrix, the “vx” designation is used as being more representative of the fact that the object is conceptually a vector (v) of values (x ≡ date-scalar datum pairs). Sometimes an mx prefix will be used, because mxSeries is n × 2 matrix of mixed data.

The second time series representation is somewhat inaccurately called a “database” and usually designated something like dbSeries. It is a triple of {dates vector, names vector, data matirx}. Again, the indivdual data elements are assuned to be scalar. The triple can be thought of as record names, field names with each row of the matrix a record. This “db” format is also used more generally for any instance in which a matrix’s row and colums are named, and the “db” form can also be though of as a simple relational table.

Within the the capabilities of *Mathematica* it is easy to envision time series of extremely general form and function, *e.g.*, a time series of plots. We haven’t tried to address this larger application in this context.

The “Expand” functions take a base calendar (vxCalendar) and a time series (vxSeries/mxSeries or dbSeries) and produce a result whose calendar is the union of the base calendar and the time series’ calendar. Data with Missing[] values are inserted into the result as needed to expand the original time series. The result’s data are sorted by date even if the original time series’ data weren’t. If there are “holes” in the calendars, then no attempt is made to repair them. If this needs to be checked or accomplished, then it can be easily done with xCompleteMonthlyCalendarQ[] and xBuildMonthlyCalendar[] above.

```
In[•]:= xExpandCalendar[vxCalendar_, vxSeries_] := Sort[
          Join[
           vxSeries,
           Transpose[{#, Array[Missing[] &, Length[#]]}] &[
            Complement[vxCalendar, First /@ vxSeries]
           ]
          ]
         ];
```

```
In[•]:= xExpandCalendar[vxCalendar_, {vxDates_, vsNames_, mnData_}] := Module[
          {vxMissingCalendar, iMissingLength,
           vxNewCalendar, mnNewData, iColumns, viOrder},
          vxMissingCalendar = Complement[vxCalendar, vxDates];
          iMissingLength = Length[vxMissingCalendar];
          iColumns = Length[vsNames];
          vxNewCalendar = Join[vxDates, vxMissingCalendar];
          viOrder = Ordering[vxNewCalendar];
          vxNewCalendar = vxNewCalendar⟦viOrder⟧;
          mnNewData = Join[mnData, Array[Missing[] &, {iMissingLength, iColumns}]];
          mnNewData = mnNewData⟦viOrder⟧;
          {vxNewCalendar, vsNames, mnNewData}
         ];
```

The “Match” functions take a base calendar (vxCalendar) and a time series (vxSeries/mxSeries or dbSeries) and produce a result whose calendar is the base calendar. Dates in the time series that are not in the base calendar are dropped, and data with Missing[] values are inserted into the result as needed to cover dates in the base calendar but not in the time series. The result’s data are sorted by date even if the original time series’ data weren’t. They work by first applying the appropriate “Expand” function and then trimming the result according to the base calendar.

```
In[•]:= xMatchCalendar[vxCalendar_, vxSeries_] :=
         Pick[#, MemberQ[vxCalendar, First[#]] & /@ #] &[
          xExpandCalendar[vxCalendar, vxSeries]];
```

```
In[•]:= xMatchCalendar[vxCalendar_, {vxDates_, vsNames_, mnData_}] := Module[
          {dbNewData, vqPick},
          dbNewData = xExpandCalendar[vxCalendar, {vxDates, vsNames, mnData}];
          vqPick = MemberQ[vxCalendar, #] & /@ dbNewData⟦1⟧;
          {Pick[dbNewData⟦1⟧, vqPick], dbNewData⟦2⟧, Pick[dbNewData⟦3⟧, vqPick]}
         ];
```

```
In[•]:= xDailyToMonthly[vxSeries_] := Last /@ Split[vxSeries, (#1⟦1, 2⟧ == #2⟦1, 2⟧) &];
```

Linear Regression Plot

```
In[•]:= Options[xSimpleLinearRegressionPlot] = Join[{"qTooltip" → None}, Options[ListPlot]];
```

```
In[•]:= xSimpleLinearRegressionPlot[lmModel_, opts : OptionsPattern[]] :=
         Show[
          ListPlot[
           If[MatchQ[OptionValue["qTooltip"], None],
            lmModel["Data"],
            Thread[Tooltip[lmModel["Data"], OptionValue["qTooltip"]]]
           ],
           PlotStyle → {Gray, PointSize[Medium]},
           PlotRange → All,
           Frame → True
          ],
          ListPlot[
           Sort[{lmModel["Data"]⟦All, 1⟧, lmModel["PredictedResponse"]}ᵀ],
           Joined → True,
           PlotStyle → {{Black, Thick}},
           PlotRange → All
          ],
          ListPlot[
           Sort[{lmModel["Data"]⟦All, 1⟧, #}ᵀ] & /@
            Transpose[lmModel["MeanPredictionConfidenceIntervals"]],
           Joined → True,
           PlotStyle → {{Red, Dashed}},
           PlotRange → All
          ],
          ListPlot[
           Sort[{lmModel["Data"]⟦All, 1⟧, #}ᵀ] & /@
            Transpose[lmModel["SinglePredictionConfidenceIntervals"]],
           Joined → True,
           PlotStyle → {{Magenta, Dotted}},
           PlotRange → All
          ],
          PlotRange → All,
          Evaluate[FilterRules[{opts}, Options[ListPlot]]]
         ]
```

Formatting

This formatting functions focus on producing strings with basic space and layout issues resolved. Options to PaddedForm[] can be added to further tune the formatting. Additional formatting is usually accomplished by wrapping Style[] specifications around the strings producted by xFMT[], *e.g.*, to change the color of negative results. Three scalar versions are provided for handling Integers, Numerics and Strings. Two other versions handle vectors and matrices.

One forn of xFmt can be used to apply a list of formats to a list of fields so that a record can be formatted in a consistent manner. It can be applied to a vector of data which is interpreted as a single record or to a matrix whose rows are interpreted as records.

The default options are those of PaddedForm[] and the user can easily override them. The purpose of xFmt[] is to produce formatted strings, rather than the "display form" that *Mathematica* uses for NumberForm[] and related functions which tends to be a pain to deal with.

```
In[•]:= xFmt[x_?NumericQ, {w_Integer, d_Integer}, opts : OptionsPattern[]] :=
          ToString[PaddedForm[Chop[x, 10^-(1 + d)], {w, d},
            FilterRules[{opts, DigitBlock → {3, ∞}}, Options[PaddedForm]]]];
```

```
In[•]:= xFmt[x_Integer, w_Integer, opts : OptionsPattern[]] :=
          ToString[PaddedForm[x, {w, 0}, FilterRules[
             {opts, NumberPoint → "", DigitBlock → {3, ∞}}, Options[PaddedForm]]]];
```

```
In[•]:= xFmt[x_String, w_Integer] :=
          StringTake[StringJoin[x, StringJoin[Table[" ", {w}]]], w];
```

```
In[•]:= xFmt[x_?VectorQ, f_List] := xFmt @@ # & /@ Transpose[{x, f}];
```

```
In[•]:= xFmt[x_?MatrixQ, f_List] := xFmt[#, f] & /@ x;
```

Some examples: xFmt[] places leading zeroes to a signed integer and produces a string. Next, Style[] does whatever else is needed for the desired effect. Finally, If[] is used to conditionally modify the Style[] to show negative numbers in red. The result is carried along as {original number, formatted string} until the formatted string is finally presented as a result. *Mathematica* provides a rich set of functionality for formatting strings using Style[] so the decision was made to keep xFmt[] as simple as possible and rely on Style[] for everything else.

If you’re doing something complicated with formatting the best approach is written a general function that formats an individual value and then use Map[] or related functions to distribute the process over the individual elements.

```
In[•]:= If [First[#] < 0, Append[Last[#], Red], Last[#]] &[
         {#, Style[xFmt[#, 3, SignPadding → True, NumberPadding → "0"],
             FontSize → 18, FontFamily → "Verdana"]} &[-3]
        ]
        FullForm[%]

Out[•]= -003

Out[•]//FullForm= Style["-003", Rule[FontSize, 18], Rule[FontFamily, "Verdana"], RGBColor[1, 0, 0]]
```

Use a list of formatting specs to format a “record” or “line”.

```
In[•]:= xFmt[{"John", "Doe", 123, 123.43, -0.9823}, {10, 10, 5, {4, 1}, {4, 3}}]

Out[•]= {John      , Doe       ,   123,  123.4, -0.982}
```

Using a list of formatting specs to format a list of lines.

```
In[•]:= TableForm[xFmt[
          {{"John", "Doe", 123, 123.43, -0.9823}, {"Edward", "Smith", 34, 3024.4, 1.0004}},
          {10, 10, 5, {4, 1}, {4, 3}}], TableAlignments → Right]
Out[•]//TableForm=
        John          Doe                 123         123.4    -0.982
        Edward        Smith                34       3,024.0     1.000
```

MLE Factor Fit — Covariance Based

```
In[•]:= xInverse[mnB_, mnD_] := Module[
           {mnID},
           mnID = DiagonalMatrix[1 / Tr[mnD, List]];
           mnID - mnID.mnB.
             Inverse[IdentityMatrix[Last[Dimensions[mnB]]] + mnBᵀ.mnID.mnB].mnBᵀ.mnID
         ];
```

```
In[•]:= xInitializeFactorModel[mnObsCov_, iOrder_] := Module[
           {mnErrDiag, mnFactor},
           mnErrDiag = DiagonalMatrix[Diagonal[mnObsCov] / 2];
           mnFactor = (First[#].√#[[2]]) &[
             SingularValueDecomposition[mnObsCov - mnErrDiag, iOrder]
            ];
           {mnFactor, mnErrDiag}
         ];
```

```
In[•]:= xLogLik[iN_, mnOC_, mnIΣ_, nDΣ_] := -0.5 iN (Log[nDΣ] + Total[Diagonal[mnOC.mnIΣ]])
```

```
In[•]:= Options[xFactorFitMLE] = {"ToleranceGoal" → 10.^-8, "MaxIterations" → 400}
Out[•]= {ToleranceGoal → 1. × 10^-8, MaxIterations → 400}
```

```
In[•]:= xFactorFitMLE[iN_, mnObsCov_,
          {mnInitFactors_, mnInitErrDiag_}, OptionsPattern[]] := Module[
          {nDetCov, iF, mnErrDiag, mnFactors, mnInvCov,
           vnLogLikHistory, mnM, iMaxIter, mnProj, nTol},
          (* Resolve options and set up run parameters *)
          iMaxIter = OptionValue["MaxIterations"];
          nTol = OptionValue["ToleranceGoal"];
          iF = Last[Dimensions[mnInitFactors]];
          mnFactors = mnInitFactors;
          mnErrDiag = mnInitErrDiag;
          (* Pre-loop *)
          mnInvCov = xInverse[mnFactors, mnErrDiag];
          nDetCov = Det[mnFactors.mnFactorsᵀ + mnErrDiag];
          vnLogLikHistory = {xLogLik[iN, mnObsCov, mnInvCov, nDetCov]};
          For[iIter = 1, iIter ≤ iMaxIter, iIter++,
           (* E-Step *)
           mnProj = mnFactorsᵀ.mnInvCov;
           mnM = mnObsCov.Transpose[mnProj];
           (* M-Step *)
           mnFactors = mnM.Inverse[IdentityMatrix[iF] - mnProj.mnFactors + mnProj.mnM];
           mnErrDiag = DiagonalMatrix[Diagonal[mnObsCov - mnFactors.Transpose[mnM]]];
           (* Log likelihood *)
           mnInvCov = xInverse[mnFactors, mnErrDiag];
           nDetCov = Det[mnFactors.mnFactorsᵀ + mnErrDiag];
           vnLogLikHistory =
            Append[vnLogLikHistory, xLogLik[iN, mnObsCov, mnInvCov, nDetCov]];
           (* Termination test *)
           If[vnLogLikHistory⟦-1⟧ / vnLogLikHistory⟦-2⟧ - 1 ≤ nTol, Break[]];
          ];
          {{mnFactors, mnErrDiag}, vnLogLikHistory}
         ];
```

Condition Number

The condition number, a measure of its numerical stability, is the ratio of its smallest and largest eigenvalues.

```
In[•]:= xConditionNumber[mnM_] := Min[#] / Max[#] &[Eigenvalues[mnM]]
```

Quadratic Forms and Ellipsoids - A Little Linear Algebra and Geometry

```
In[•]:= mnΣ = {{1, -0.75}, {-0.75, 1}};
        MatrixForm[mnΣ]
```

```
Out[•]//MatrixForm=
  (   1     -0.75 )
  ( -0.75     1   )
```

```
In[•]:= CholeskyDecomposition[mnΣ]
Out[•]= {{1., -0.75}, {0., 0.661438}}
```

$$\mathbf{y}^T \mathbf{y} = \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} = \mathbf{x}^T \mathbf{P}^T \mathbf{P}\,\mathbf{x} \quad \Rightarrow \quad \mathbf{y} = \mathbf{P}\,\mathbf{x} \quad \Rightarrow \quad \mathbf{x} = \mathbf{P}^{-1}\,\mathbf{y}$$

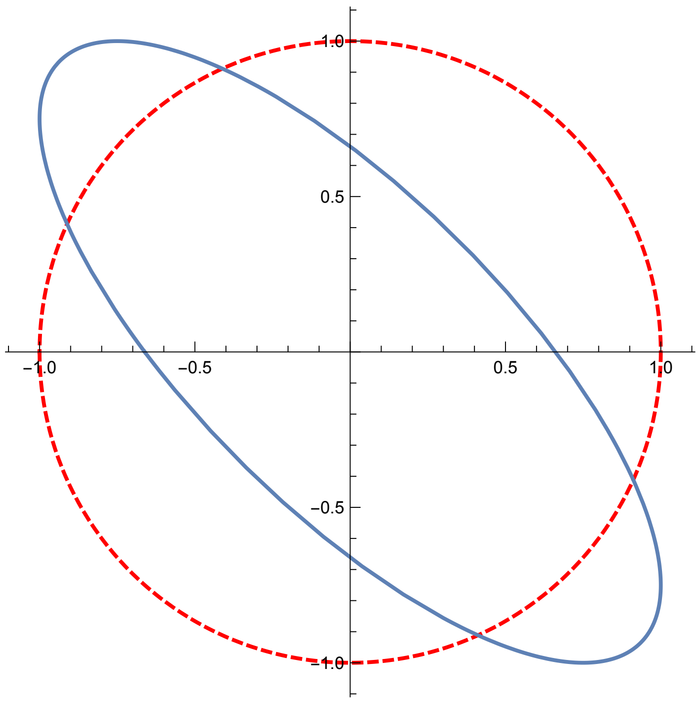
What we need, then, for the transformation matrix is the inverse of a "square root" of the inverse covariance matrix.

```
In[•]:= mnP = Inverse[CholeskyDecomposition[Inverse[mnΣ]]];
        MatrixForm[mnP]
```

```
Out[•]//MatrixForm=
  ( 0.661438  -0.75 )
  (    0.       1.  )
```

```
In[•]:= Show[
         ParametricPlot[{Cos[x], Sin[x]}, {x, -π, π}, PlotStyle → {Red, Dashed, Thick}],
         ParametricPlot[mnP.{Cos[x], Sin[x]}, {x, -π, π}, PlotStyle → {Thick}]
        ]
```

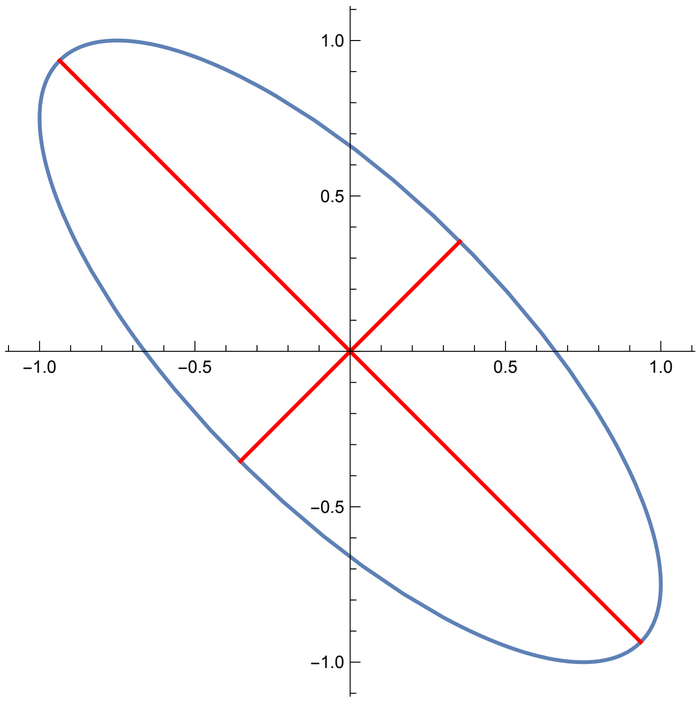
Out[•]=



```
In[•]:= {mnV, mnW, mnU} = SingularValueDecomposition[mnΣ];
        MatrixForm /@ {mnV, mnW, mnU}
```

$$\left\{\begin{pmatrix} -0.707107 & 0.707107 \\ 0.707107 & 0.707107 \end{pmatrix}, \begin{pmatrix} 1.75 & 0. \\ 0. & 0.25 \end{pmatrix}, \begin{pmatrix} -0.707107 & 0.707107 \\ 0.707107 & 0.707107 \end{pmatrix}\right\}$$

```
In[•]:= Show[
          ParametricPlot[mnP.{Cos[x], Sin[x]}, {x, -π, π}, PlotStyle → {Thick}],
          ListPlot[{#, -#} &[mnV⟦All, 1⟧ √mnW⟦1, 1⟧ ],
           Joined → True, PlotStyle → {Red, Thick}],
          ListPlot[{#, -#} &[mnV⟦All, 2⟧ √mnW⟦2, 2⟧ ], Joined → True, PlotStyle → {Red, Thick}]
        ]
```

Out[•]=


```
In[•]:= Det[mnΣ]
        Times @@ Tr[mnW, List]

Out[•]= 0.4375

Out[•]= 0.4375
```

Matrix Models

PCA

$$(r_i - r_f) = \sum_{j=1}^{m} p_{i,j}(f_j - r_f)$$

$$(\mathbf{r} - r_f\,\mathbb{1}) \;=\; \mathbf{P}\,(\mathbf{f} - r_f\,\mathbb{1})$$

$$\boldsymbol{\Sigma} \;=\; \mathbf{P}^T\,\mathbf{P}$$

There is a problem. There exists portfolios with zero variance in the model.

$$\exists\;\mathbf{x} \cdot \ni \cdot\; \mathbf{x}^T\,\mathbf{P}^T\,\mathbf{P}\,\mathbf{x} \;=\; 0$$

It is unlikely such portfolios exist in reality.

Factor Model

$$(r_i\; - r_f) \;=\; \sum_{j=1}^{m} b_{i,j}(f_j - r_f) + \varepsilon_i$$

$$(\mathbf{r} - r_f\,\mathbb{1}) \;=\; \mathbf{B}\,(\mathbf{f} - r_f\,\mathbb{1}) \;+\; \boldsymbol{\varepsilon}$$

$$\boldsymbol{\Sigma} \;=\; \mathbf{B}^T\,\mathbf{C}\,\mathbf{B} \;+\; \mathbf{D}$$

For orthonormal factors, *i.e.*, where $\mathbf{C} = \mathbf{I}$,

$$\boldsymbol{\Sigma} \;=\; \mathbf{B}^T\,\mathbf{B} \;+\; \mathbf{D}$$

Note that even if you hedge out “market” effects $\left(\mathbf{x}^T\,\mathbf{B}^T\,\mathbf{B}\,\mathbf{x} = \mathbf{0}\right)$ there is always a portfolio variance $(\mathbf{x}^T\,\mathbf{D}\,\mathbf{x} > \mathbf{0})$.

The Market Portfolio

The Market Portfolio, which here we mean of assets other than the risk free asset, is the proportional holdings of all such assets. That is, it is the capital-weighted sum of the proportions in all investors’ portfolios.

It is essentially a theoretical construct. Assets are assumed to be liquid, available to every investor, and infinitely divisible. All means all and includes stocks, bonds, precious metals, real estate, and so forth (See http://en.wikipedia.org/wiki/Market_portfolio).

While the market portfolio theoretically includes everything, in practice surrogates consisting of portfolios of widely available and liquid assets are typically used. Examples of stock indices might be the

- S&P 500 Index (http://en.wikipedia.org/wiki/S%26 P_ 500) for the United States or
- MSCI World Index (http://en.wikipedia.org/wiki/MSCI_World) for a more global perspective.

The market portfolio is in practice not known, but an appropriate surrogate is often used as representative of the market portfolio and serves as a benchmark against which investors can assess their performance.

Optimality of the Market Portfolio

We will not attempt a formal proof that the market portfolio is optimal. Instead we present a reasonable intuitive argument. However, that argument (as would a more formal proof) depends heavily on our assumptions:

- All investors are:
  - informed (knowing of all available investments along with their means and covariances),

- rational (being risk adverse), and
- mean-variance optimizers,

- The market portfolio consists of assets that are
  - liquid and
  - infinitely divisible.

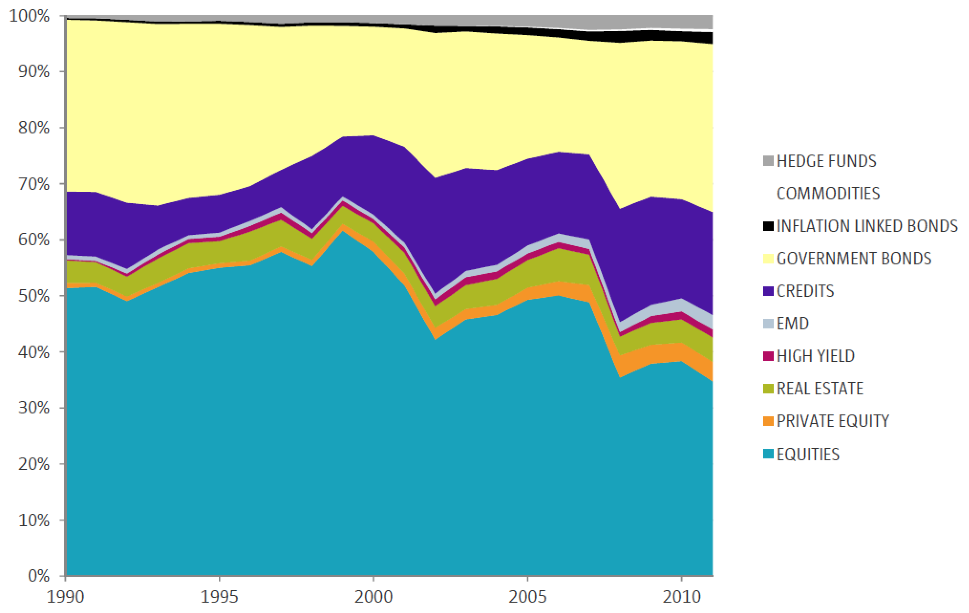
As covered earlier in class, investors, given a collection of risky assets need only consider a single allocation, the tangent portfolio. The risk-reward trade-offs appropriate for the investors are reflected by the adjusting their cash positions—deleveraging to decrease risk by holding cash and investing less in the tangent portfolio or leveraging to increase risk by borrowing cash and funding a larger holding in the tangent portfolio.

Why would investors only consider the full market portfolio and not a subset? Adding an investment for consideration cannot decrease the Sharpe ratio of a tangent portfolio; that would contradict its optimality. There is no benefit in excluding an asset from consideration.

All investors select, therefore, the identical tangent portfolio. That tangent portfolio is itself, therefore, the market portfolio. Thus, the market portfolio is not only efficient; it is the portfolio with the highest possible Sharpe ratio—it is *the* tangent portfolio.

Example - An Estimate of the Global Market Portfolio

Here is a graph which plots estimates of the global market portfolio of liquid investable assets. The authors' methodology is described in the paper cited below.



Strategic Asset Allocation : The Global Multi – Asset Portfolio 1959 – 2011,

by Ronald Q. Doeswijk, Trevin W. Lam, and Laurens A. P. Swinkels, 02 Nov 2012.

Retrieved 02 Mar 2014 from SSRN : http : // papers.ssrn.com / sol3 / papers.cfm ? abstract_id = 2 170 275.

Relevance of the Market Portfolio

Classroom discussion…

The Capital Asset Pricing Model

The Capital Asset Pricing Model (CAPM) is used to determine a theoretical rate of return for an asset based on that portion of its risk that cannot be diversified away (See http://en.wikipedia.org/wiki/Capital_asset_pricing _model).

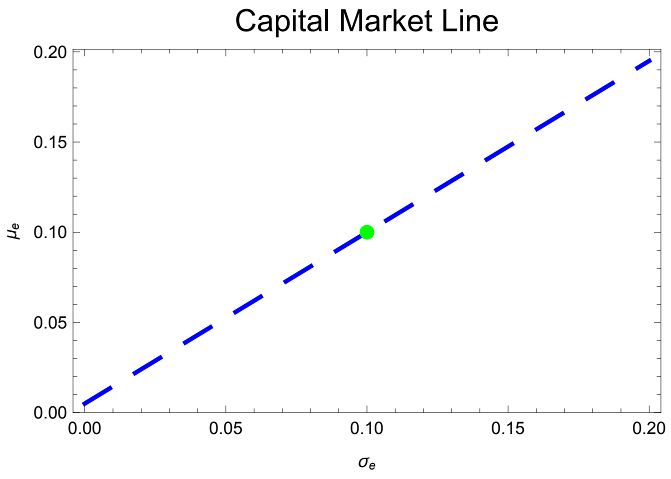
The CAPM decomposes the risk of an asset into two components: systematic risk (which captures its covariance with the market portfolio) and idiosyncratic risk (which is unique to the asset).

The Capital Market Line (Risk-Reward of Efficient Assets)

A consequence of the CAPM is that an efficient portfolio must lie on the (σ, μ)-line from the risk free rate through the market (tangent) portfolio. This is the *Capital Market Line*. For any efficient asset e, its expected return and risk are governed by the following relationship:

$$\mu_e = r_f + \frac{\mu_M - r_f}{\sigma_M} \sigma_e$$

In (σ, μ)-space, an example of the Capital Market Line is



The Security Market Line (Market Covariance and Expected Return)

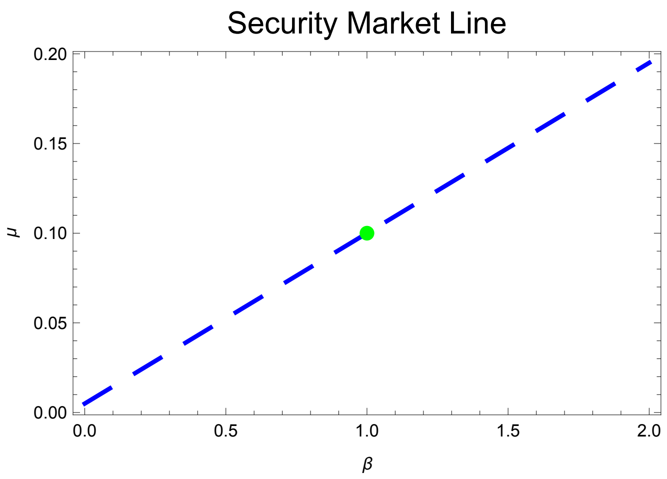
One implication of the CAPM and the security market line is that $\mu_i \propto \beta_i \propto \sigma_{i,M}$. Thus, in an efficient market only risk that is covariant with the market is compensated for in expected return.

$$r_i(t) - r_f = \beta_i (r_M(t) - r_f) + \epsilon_i(t)$$

$$\beta_i = \frac{\sigma_{i,M}}{{\sigma_M}^2}$$

The economic notion that is behind this is that idiosyncratic risks are uncorrelated with one another and can be largely diversified away. Systematic risks, however, cannot.

The line embodying this relationship is called the *Security Market Line*:



Also note that the value of $\beta_i = \sigma_{i,M}/\sigma_M{}^2$ is the solution to the linear regression problem with independent variable $(r_M(t) - r_f)$ and dependent variable $(r_i(t) - r_f)$.

CAPM Theorem

If the market portfolio M is efficient then for a given asset i its return satisfies

$$E[r_i(t) - r_f] = \beta_i(r_M(t) - r_f) + \epsilon_i(t)$$

$$\mu_i - r_f = \beta_i(\mu_M - r_f)$$

where

$$\beta_i = \frac{\sigma_{i,M}}{\sigma_M{}^2}$$

Proof : Consider a portfolio I composed of $0 \leq \alpha \leq 1$ invested in an asset i and (1-α) in the market portfolio; the mean and standard deviation of its return is

$$\mu_I = \alpha\,\mu_i + (1-\alpha)\,\mu_M$$

$$\sigma_I = \sqrt{\alpha^2\,\sigma_i{}^2 + 2\,\alpha\,(1-\alpha)\,\sigma_{i,M} + (1-\alpha)^2\,\sigma_M{}^2}$$

Note that $\alpha = 0$ corresponds to the market portfolio itself. *Note that for any value of $\alpha \neq 0$, the mean and standard deviation of the portfolio I must be below the capital market line; otherwise, the efficiency of the market portfolio would be contradicted*. Consider the following derivatives at $\alpha = 0$.

$$\frac{d\mu_I}{d\alpha} = \mu_i - \mu_M$$

$$\frac{d\sigma_I}{d\alpha} = \frac{\alpha\,\sigma_i{}^2 + (1-\alpha)\,\sigma_{i,M} + (\alpha-1)\,\sigma_M{}^2}{\sigma_\alpha}\Bigg|_{\alpha=0} = \frac{\sigma_{i,M} - \sigma_M{}^2}{\sigma_M}$$

$$\frac{d\mu_I}{d\sigma_I}\Bigg|_{\alpha=0} = \frac{(\mu_i - \mu_M)\,\sigma_M}{\sigma_{i,M} - \sigma_M{}^2}$$

As noted, the $\{\sigma_I, \mu_I\}$ curve is tangent to the security market line at the point of the market portfolio, $\alpha = 0$. Thus,

$$\frac{(\mu_i - \mu_M)\,\sigma_M}{\sigma_{i,M} - \sigma_M{}^2} = \frac{\mu_M - r_f}{\sigma_M}$$

Solving for $\mu_i - r_f$

$$\mu_i - r_f = \frac{\sigma_{i,M}}{\sigma_M{}^2}(\mu_M - r_f) = \beta_i(\mu_M - r_f)$$

Applications of the CAPM

The CAPM has many important applications in financial analysis, particularly in pricing and performance analysis. Although the real-world market certainly does not meet all of its assumptions, it has proved to be a useful approximation to guide investors. As with any model, slavishly following its prescriptions can lead to disaster.

Pricing Forms

Consider pricing an asset at time t based on its price one period hence.

$$\frac{\mathrm{E}[P(t+1)] - P(t)}{P(t)} = \mu_i = r_f + \beta(\mu_M - r_f) \implies P(t) = \frac{\mathrm{E}[P(t+1)]}{1 + r_f + \beta(\mu_M - r_f)}$$

$$\mu_i = r_f + \beta(\mu_M - r_f) \implies \mathrm{E}[P(t+1)] = P(t)(1 + r_f + \beta(\mu_M - r_f))$$

Thus, $r_f + \beta(\mu_M - r_f)$ represents a risk-adjusted interest rate that tells us how to price (*i.e.*, determine the present value of) the expected pay-off at $t+1$.

Jensen's Alpha

The basic form of the CAPM has no intercept term.

$$r_i(t) - r_f = \beta_i(r_M(t) - r_f) + \epsilon_i(t)$$

However, given actual data we can perform a linear regression and estimate the following regression line which includes an intercept term called Jensen's alpha.

$$r_i(t) - r_f = \alpha_i + \beta_i(r_M(t) - r_f) + \epsilon_i(t)$$

The value of α_i indicates how far above or below the security market line μ_i falls. Thus, it is a measure of whether the security i beats the market in risk-adjusted terms.

$$\mu_i - r_f = \alpha_i + \beta_i(\mu_M - r_f)$$

$$\text{BAD: intercept without risk adjustment} = \alpha_i + (1 - \beta_i)\, r_f$$

Sharpe Ratio

A positive α does not imply efficiency. A related measure is called the Sharpe ratio does measure relative efficiency by measuring the risk-to-reward relative to the capital market line.

$$\psi_i = \frac{\mu_i - r_f}{\sigma_i}$$

Mean-Variance Optimization under the CAPM

The CAPM provides us with a framework for better statistical estimation of the parameters required for mean-variance optimization. This not only leads to better estimates but

the structure of the CAPM can be exploited to produce more numerically stable and rapid portfolio optimizations. This is due to the fact the stability of both the likelihood function and the mean-variance quadratic program depend heavily on the structure of the covariance matrix.

Mathematical Development

$$r_i(t) - r_f = \beta_i (r_M(t) - r_f) + \epsilon_i(t), \quad \text{for } i = \{1, \ldots, n\}$$

The $\epsilon_i(t)$s are mean zero error terms

$$\mathrm{E}[\epsilon_i(t)] = 0, \quad \forall\, i$$

that are uncorrelated to the market and to each other.

$$\mathrm{Cov}[r_M(t), \epsilon_i(t)] = 0, \quad \forall\, i$$

$$\mathrm{Cov}[\epsilon_i(t), \epsilon_j(t)] = 0, \quad \forall\, i \neq j$$

Under the assumptions of the CAPM, the mean return of each asset is, therefore,

$$\mu_i - r_f = \beta_i (\mu_M - r_f)$$

and, noting how the cross-product terms are zero and drop out, the variance of each asset's return is

$$\sigma_i{}^2 = (\sigma_M \beta_i)^2 + \sigma_{\epsilon_i}{}^2$$

The *systematic risk* is that portion that arises out of its covariance with the market, $(\sigma_M \beta_i)^2$, and the *idiosyncratic risk* is that portion that is unique to the asset, $\sigma_{\epsilon_i}{}^2$.

Their covariances are based solely on their mutual connection to the market:

$$\sigma_{i,j} = \sigma_M{}^2 \beta_i \beta_j$$

Thus, a complete description of the market returns for the purposes of Markowitz mean-variance portfolio optimization involves the riskfree rate r_f, the market expected return μ_M and standard deviation σ_M and the vector of betas $\boldsymbol{\beta}$ and a diagonal covariance matrix of error variances $\mathbf{D}$:

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_i \\ \vdots \\ \beta_n \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} \sigma_{\epsilon_1}{}^2 & & & & \\ & \ddots & & & \\ & & \sigma_{\epsilon_i}{}^2 & & \\ & & & \ddots & \\ & & & & \sigma_{\epsilon_n}{}^2 \end{pmatrix}$$

From these we can easily construct the mean vector and covariance matrix for portfolio optimization:

$$\boldsymbol{\mu} - r_f \mathbb{1} = (\mu_M - r_f) \boldsymbol{\beta}$$

$$\boldsymbol{\Sigma} = \sigma_M{}^2 \boldsymbol{\beta} \boldsymbol{\beta}^T + \mathbf{D}$$

Example - Constructing a Covariance Matrix

Consider a simple three asset portfolio modeled under the CAPM. Their covariance matrix can be constructed as follows:

In[•]:=
```
Print[MatrixForm[σM^2 KroneckerProduct[{β1, β2, β3}, {β1, β2, β3}]],
  " + ", MatrixForm[DiagonalMatrix[{σϵ1, σϵ2, σϵ3}^2]]]
```

$$\begin{pmatrix} \beta_1^2 \sigma_M^2 & \beta_1 \beta_2 \sigma_M^2 & \beta_1 \beta_3 \sigma_M^2 \\ \beta_1 \beta_2 \sigma_M^2 & \beta_2^2 \sigma_M^2 & \beta_2 \beta_3 \sigma_M^2 \\ \beta_1 \beta_3 \sigma_M^2 & \beta_2 \beta_3 \sigma_M^2 & \beta_3^2 \sigma_M^2 \end{pmatrix} + \begin{pmatrix} \sigma_{\epsilon_1}^2 & 0 & 0 \\ 0 & \sigma_{\epsilon_2}^2 & 0 \\ 0 & 0 & \sigma_{\epsilon_3}^2 \end{pmatrix}$$

In[•]:=
```
MatrixForm[
  σM^2 KroneckerProduct[{β1, β2, β3}, {β1, β2, β3}] + DiagonalMatrix[{σϵ1, σϵ2, σϵ3}^2]
]
```

Out[•]//MatrixForm=

$$\begin{pmatrix} \beta_1^2 \sigma_M^2 + \sigma_{\epsilon_1}^2 & \beta_1 \beta_2 \sigma_M^2 & \beta_1 \beta_3 \sigma_M^2 \\ \beta_1 \beta_2 \sigma_M^2 & \beta_2^2 \sigma_M^2 + \sigma_{\epsilon_2}^2 & \beta_2 \beta_3 \sigma_M^2 \\ \beta_1 \beta_3 \sigma_M^2 & \beta_2 \beta_3 \sigma_M^2 & \beta_3^2 \sigma_M^2 + \sigma_{\epsilon_3}^2 \end{pmatrix}$$

Parsimony

The parameters for n assets required for a mean-variance estimation are n estimates for the mean, $n(n+1)/2$ unique estimates for the covariance, and the risk free rate, a total of $n(n+1)/2+n+1$. The CAPM requires the market mean and standard deviation, the risk free rate, n betas and n idiosyncratic (error) standard deviations, a total of $2\,n+3$.

A quick analysis shows how a direct estimate from the raw data compares with the CAPM:

In[•]:=

Parameter Count – n Assets

	Raw Data	CAPM
100	5151	203
250	31 626	503
500	125 751	1003
1000	501 501	2003

Out[•]=
```
Parameter Count – n Assets
{{5151, 203}, {31 626, 503}, {125 751, 1003}, {501 501, 2003}}
```

From many perspectives, parsimony is a significant advantage. The CAPM estimates are more statistically efficient. As long as it is a reasonable representation of the behavior of asset returns (which, alas, is not the best job we can do), it makes sense to use it as the basis for modeling returns, Recall that mean-variance optimization requires that the covariance matrix be inverted and the covariance matrix based on the CAPM tends to be much more stable numerically.

This is especially the case for large n where the resulting covariance is often singular or near-singular. This is not surprising. For large n unless we have very long time series a direct estimate may require us to estimate more parameters than we have data points. The results aren't likely to make much sense.

One possible deficiency of the CAPM is that it models systematic returns as being based on a single factor β. As we will see shortly, we can build similar models based on multiple factors that have the same benefits of parsimony but are more representative of the true character of asset returns.

Solution to a Simple Markowitz Model for the CAPM

Consider the following simple quadratic program representing the portfolios on the Capital Market Line and its solution to proportionality.

$$\min\left\{\frac{1}{2}\mathbf{x}^T\,\Sigma\,\mathbf{x} - \lambda\left(\boldsymbol{\mu} - r_f\,\mathbf{1}\right)^T\mathbf{x}\right\}$$

$$\Sigma\,\mathbf{x} - \lambda\left(\boldsymbol{\mu} - \mathbf{1}\,r_f\right) = \mathbf{0} \implies \lambda\left(\boldsymbol{\mu} - \mathbf{1}\,r_f\right) = \Sigma\,\mathbf{x}$$

If we assume the CAPM, then we can rewrite the covariance matrix as follows.

$$(\boldsymbol{\mu} - r_f\,\mathbf{1}) \propto \sigma_M{}^2\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{x} + \mathbf{D}\,\mathbf{x}$$

Next we multiply both sides with the inverse of the error covariance matrix.

$$\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1}) \propto \sigma_M{}^2\,\mathbf{D}^{-1}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{x} + \mathbf{x}$$

If we had a solution for $\boldsymbol{\beta}^T\,\mathbf{x}$, then we could solve for $\mathbf{x}$. Thus, we form an equation of the above, multiply both sides by $\boldsymbol{\beta}^T$and solve for $\boldsymbol{\beta}^T\,\mathbf{x}$.

$$\boldsymbol{\beta}^T\,\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1}) = \sigma_M{}^2\,\boldsymbol{\beta}^T\,\mathbf{D}^{-1}\,\boldsymbol{\beta}\,\boldsymbol{\beta}^T\,\mathbf{x} + \boldsymbol{\beta}^T\,\mathbf{x}$$

$$\boldsymbol{\beta}^T\,\mathbf{x} = \frac{\boldsymbol{\beta}^T\,\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1})}{1 + \sigma_M{}^2\,\boldsymbol{\beta}^T\,\mathbf{D}^{-1}\,\boldsymbol{\beta}}$$

We can now substitute the solution for $\boldsymbol{\beta}^T\,\mathbf{x}$ into the original expression and solve for $\mathbf{x}$ then drop the constant term that does not affect proportionality.

$$\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1}) \propto \sigma_M{}^2\,\mathbf{D}^{-1}\,\boldsymbol{\beta}\left(\frac{\boldsymbol{\beta}^T\,\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1})}{1 + \sigma_M{}^2\,\boldsymbol{\beta}^T\,\mathbf{D}^{-1}\,\boldsymbol{\beta}}\right) + \mathbf{x}$$

$$\mathbf{x} \propto \left(1 - \frac{\sigma_M{}^2\,\boldsymbol{\beta}^T\,\mathbf{D}^{-1}\,\boldsymbol{\beta}}{1 + \sigma_M{}^2\,\boldsymbol{\beta}^T\,\mathbf{D}^{-1}\,\boldsymbol{\beta}}\right)\mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1}) \implies \mathbf{x} \propto \mathbf{D}^{-1}(\boldsymbol{\mu} - r_f\,\mathbf{1})$$

We can substitute $(\mu_M - r_f)\boldsymbol{\beta}$ for $(\boldsymbol{\mu} - r_f\,\mathbf{1})$, rearrange terms and again drop the constant term.

$$\mathbf{x} \propto (\mu_M - r_f)\,\mathbf{D}^{-1}\,\boldsymbol{\beta} \implies \mathbf{x} \propto \mathbf{D}^{-1}\,\boldsymbol{\beta} \implies x_i \propto \frac{\beta_i}{\sigma_{\epsilon_i}{}^2}$$

If we note that $\beta_i = \sigma_{i,M}/\sigma_M{}^2$, then to proportionality we can drop the constant $1/\sigma_M{}^2$ and an alternate form is

$$x_i \propto \frac{\beta_i}{\sigma_{\epsilon_i}{}^2} \implies x_i \propto \frac{\sigma_{i,M}}{\sigma_{\epsilon_i}{}^2}$$

Obviously, if we normalize the solution to add to unity, then we have the market portfolio.

Example - CAPM Allocation

Consider the following CAPM parameters.

```
In[•]:= nRiskFree = 0.035;
nMarketReturn = 0.115;
nMarketVar = 0.009;
vnBeta = {0.85, 1.20, 0.95};
vnErrorVar = {0.021, 0.012, 0.015};
```

The expected return vector and covariance matrix can then be computed.

```
In[•]:= vnExpectedReturn = (nMarketReturn - nRiskFree) vnBeta + nRiskFree;
        MatrixForm[vnExpectedReturn]

        mnCovariance = nMarketVar Outer[Times, vnBeta, vnBeta] + DiagonalMatrix[vnErrorVar];
        MatrixForm[mnCovariance]
Out[•]//MatrixForm=
```

$$\begin{pmatrix} 0.103 \\ 0.131 \\ 0.111 \end{pmatrix}$$

```
Out[•]//MatrixForm=
```

$$\begin{pmatrix} 0.0275025 & 0.00918 & 0.0072675 \\ 0.00918 & 0.02496 & 0.01026 \\ 0.0072675 & 0.01026 & 0.0231225 \end{pmatrix}$$

We can use the complete form, $\mathbf{x} \propto \boldsymbol{\Sigma}^{-1} (\mu - r_f \mathbf{1})$, to solve for the optimal portfolio, normalizing the values so that they add to unity.

```
In[•]:= Timing[#/Total[#] &[Inverse[mnCovariance].(vnExpectedReturn - nRiskFree)]]
Out[•]= {0.000201, {0.198598, 0.490654, 0.310748}}
```

The alternate computation, $x_i \propto \beta_i / \sigma_{\epsilon_i}{}^2$, yields the same portfolio allocations.

$$x_i \propto \frac{\beta_i}{\sigma_{\epsilon_i}{}^2}$$

```
In[•]:= Timing[#/Total[#] &[vnBeta / vnErrorVar]]
Out[•]= {0.000025, {0.198598, 0.490654, 0.310748}}

In[•]:= %%[[1]] / %[[1]]
Out[•]= 8.04
```

Inverting the covariance matrix is a costly operation that is roughly cubic in the number of the securities in the portfolio, but the simplified solution scales linearly. Even with only 3 assets, exploiting the structure of the covariance matrix results in an order of magnitude speed-up. For a larger number of assets the speed-up can be many orders of magnitude. We will show later how a special covariance structure exists when we have more than just a single market factor and how it can be similarly exploited.

Multi-Factor Models

The CAPM has a single factor that explains the systematic exposure experienced by an asset or portfolio of assets. A better representation of returns may be realized by introducing multiple factors, e.g., a factor for each industry group or a factor to distinguish firms by an economic or market statistics such as total capitalization and book-to-market.

Let $\mathcal{I}$ denote the set of assets or securities, $\mathcal{K}$ denote the set of underlying factors with $f_k(t)$ the k^{th} factor's return, $b_{i,k}$ the *factor loading* of asset i to factor k, a_i an asset dependent intercept and $\xi_i(t)$ a zero mean error term, then the general form for a multi-factor model for an asset i is

$$r_i(t) = a_i + \sum_{k \in \mathcal{K}} b_{i,k}\, f_k(t) + \xi_i(t), \quad i \in \mathcal{I}$$

In matrix terms, with **B** the factor loading matrix,

$$\mathbf{r} = \mathbf{a} + \mathbf{B}\, \boldsymbol{f} + \boldsymbol{\xi}$$

The cardinality of $\mathcal{K}$ is less than than of $\mathcal{I}$, typically much less, i.e. $|\mathcal{K}| << |\mathcal{I}|$. Multi-factor models are used extensively in finance, particularly in the management of liquid market securities such as stocks and bonds, although their usefulness is by no means limited to those applications.

The CAPM is an example of a single factor model. Some factor models have only a small number of factors, such as the 3-factor Fama-French model described below. Others may have many more factors, such as the 72-factor BARRA Factor Model (See http://www.msci.com/resources/factsheets/Barra_US_Equity _Model _USE4.pdf) which has 60 industry factors and 12 style factors.

The Fama-French Model

The Fama-French Model (FFM) is a widely used three-factor model (see http://en.wikipedia.org/wiki/Fama–French_three-factor_model). Two groups of stocks seem to perform significantly better than expected from the CAPM.

- Stocks with smaller market capitalization seen to outperform ones with higher capitalization; hence, the return r_S which refers to the return of small-cap minus large cap stocks.
- Higher book-to-market ratios (value stocks) seem to outperform lower book-to-market ratios (growth stocks); hence, the return r_V which refers to the return of value stocks minus growth stocks (high minus low book-to-market).

The Fama-French model adds these two returns to the CAPM to produce a three-factor model:

$$r_i(t) - r_f = \alpha_i + \beta_i(r_M(t) - r_f) + \gamma_i\, r_S(t) + \delta_i\, r_V(t) + \epsilon_i(t), \quad i \in \mathcal{I}$$

Given the values of the returns r_M, r_S, and $r_V$as independent variables and an stock return r_i as the dependent variable, the parameters of the FFM are fit by linear regression. Studies have repeatedly shown that the FFM explains significantly more of a portfolio's return than does the CAPM.

There have been extensions to the FFM. For example, the Cahart Four-Factor Model (see http://en.wikipedi-a.org/wiki/Carhart_four-factor_model) adds a momentum term, identifying last month's winners and losers.

The APT

The Arbitrage Pricing Theory (APT) starts with the assumption that returns follow a multi-factor model. (See https://en.wikipedia.org/wiki/Arbitrage_pricing_theory). The APT argues that market forces at equilibrium will stabilize exposure to a factor proportional to its *risk-adjusted return*, i.e., its return net of the risk-free rate, an exposure that cannot be removed by diversification. The idiosyncratic error, however, can be diversified away and an investor is not compensated for that exposure. These factor exposures will yield the risk-adjusted return of the asset. The final form of the APT is, therefore,

$$r_i(t) - r_f = \sum_{k \in \mathcal{K}} b_{i,k}(f_k(t) - r_f) + \xi_i(t), \quad i \in \mathcal{I}$$

Merging the CAPM and the APT

The CAPM is similar to the APT, but it assumes each investor trades off risk and reward in mean-variance terms. This leads naturally to the notion of the tangent portfolio and, consequently, the market portfolio. It is driven by the demands of the investors.

The APT on the other hand estimates coefficients, factor loadings, that reflect the sensitivity of assets to various economic factors with arbitrage forces in the market driving over- and under-priced assets to their equilibrium values, but its assumptions are less restrictive and does not necessarily assume all investors preferences can be expressed by a single portfolio.

We can, however, relate the CAPM and APT as follows. First, model the factors by the CAPM, denoting the respective noise terms with η_k.

$$f_k(t) - r_f = \beta_k(r_M(t) - r_f) + \eta_k(t), \quad k \in \mathcal{K}$$

Apply the APT to the securities.

$$r_i - r_f = \sum_{j \in \mathcal{K}} b_{i,k}(f_k - r_f) + \xi_i, \quad i \in \mathcal{I}$$

Substitute the factor-level CAPM into the security-level APT.

$$\beta_k(r_M(t) - r_f) + \eta_k(t) \to f_k(t) - r_f, \quad k \in \mathcal{K}$$

$$r_i - r_f = \sum_{k \in \mathcal{K}} b_{i,k}\,\beta_k(r_M - r_f) + \sum_{k \in \mathcal{K}} b_{i,k}\,\eta_k + \xi_i, \quad i \in \mathcal{I}$$

The noise terms, being mean-zero and uncorrelated, can be coalesced into a single error term, ϵ_i.

$$r_i - r_f = \left(\sum_{k \in \mathcal{K}} b_{i,k}\,\beta_k\right)(r_M - r_f) + \epsilon_i, \quad i \in \mathcal{I}$$

$$\mu_i - r_f = \beta_i(\mu_M - r_f)$$

The equation above is just the CAPM for the security. Thus, the beta of an underlying security is simply the betas of the factors weighted by the security's factor exposures.

$$\beta_i = \sum_{k \in \mathcal{K}} b_{i,k}\,\beta_k, \quad i \in \mathcal{I}$$

Applications of Factor Models

Performance Analysis

We can extend the concept of Jensen's alpha introduced above for the CAPM to the APT

$$r_i(t) - r_f = \alpha_i + \sum_{k \in \mathcal{K}} b_{i,k}(f_k(t) - r_f) + \xi_i(t), \quad i \in \mathcal{I}$$

As before, the value of α_i indicates how far above or below the plane of the APT r_i falls. Thus, it is a measure of whether the asset i beats the market in risk-adjusted and factor -adjusted terms. By the term security we mean individual securities or portfolios of securities.

Style Analysis

Given an allocation **x** for a portfolio $\mathcal{P}$, we can perform a style analysis which tells us the sources of return for $\mathcal{P}$. This can be computed by weighting each asset’s loading by each asset’s allocation. The expressions below in matrix form show the portfolio’s exposure to factor k, (y_k):

$$y_k = \sum_{k \in \mathcal{K}} b_{i,k}\,x_i \quad \Longrightarrow \quad \mathbf{y} = \mathbf{B}^T\,\mathbf{x}$$

Mean-Variance Optimization

With $\phi_k = \mathrm{E}[f_k]$

$$\mu_i - r_f = \alpha_i + \sum_{k \in \mathcal{K}} b_{i,k}(\phi_k - r_f), \quad i \in \mathcal{I}$$

$$\boldsymbol{\mu} = \boldsymbol{\alpha} + \mathbf{B}\,\boldsymbol{\phi}$$

Let, as usual, **x** denote the allocation of assets in a portfolio $\mathcal{P}$. Let y_k be the total exposure of $\mathcal{P}$ to factor k. This can be computed by weighting each asset's loading to that factor by each asset's allocation. This results in the expression below and its generalization in matrix form.

$$y_k = \sum_{k \in \mathcal{K}} b_{i,k}\, x_i \quad \Longrightarrow \quad \mathbf{y} = \mathbf{B}^T \mathbf{x}$$

If the covariance matrix of factors is **C**, then the systematic variance associated with the factors is $\mathbf{y}^T \mathbf{C}\, \mathbf{y}$. Thus, we can substitute one expression into the other and get

$$\mathbf{y} = \mathbf{B}^T \mathbf{x} \quad \text{and} \quad \mathbf{y}^T \mathbf{C}\, \mathbf{y} \quad \Longrightarrow \quad (\mathbf{B}^T \mathbf{x})^T \mathbf{C}\, (\mathbf{B}^T \mathbf{x}) \quad \Longrightarrow \quad \mathbf{x}^T (\mathbf{B}\,\mathbf{C}\,\mathbf{B}^T)\, \mathbf{x}$$

The idiosyncratic variance is captured by the diagonal matrix of error variances **D**. The covariance matrix can therefore be expressed as

$$\boldsymbol{\Sigma} = \mathbf{B}\,\mathbf{C}\,\mathbf{B}^T + \mathbf{D}$$

The variance of a portfolio can be expressed by any of the three expression below.

$$\mathbf{x}^T \boldsymbol{\Sigma}\, \mathbf{x} \;=\; \mathbf{x}^T \mathbf{B}\,\mathbf{C}\,\mathbf{B}^T \mathbf{x} + \mathbf{x}^T \mathbf{D}\, \mathbf{x} \;=\; \mathbf{y}^T \mathbf{C}\, \mathbf{y} + \mathbf{x}^T \mathbf{D}\, \mathbf{x}$$

Consider a mean-variance optimization where the allocations must satisfy some constraint set $\mathcal{S}$

$$\mathcal{M} = \min_{\mathbf{x}} \left\{ \frac{1}{2} \mathbf{x}^T \boldsymbol{\Sigma}\, \mathbf{x} - \lambda\, \boldsymbol{\mu}^T \mathbf{x} \,\middle|\, \mathbf{x} \in \mathcal{S} \right\}$$

We can rewrite this in terms of both **x** and **y** by adding "accounting constraints" to ensure that **y** is constrained to be the factor exposures

$$\mathcal{M} = \min_{\mathbf{x},\mathbf{y}} \left\{ \frac{1}{2} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}^T \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} - \lambda \begin{pmatrix} \boldsymbol{\mu} \\ \mathbf{0} \end{pmatrix}^T \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \,\middle|\, \begin{pmatrix} \mathbf{B}^T & -\mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{0} \wedge \mathbf{x} \in \mathcal{S} \right\}$$

While it may seem that we have complicated things, the solution to a quadratic program essentially involves operations equivalent to inverting the covariance matrix. Simplifying the covariance matrix allows us to exploit its special structure and speed up solutions. We can simply things further by employing what we will call an *orthonormal multi-factor model* (⊥ MFM), i.e., where the factors are of unit variance and uncorrelated (hence, their covariance matrix is the identity matrix). The covariance matrix has the form

$$\boldsymbol{\Sigma} = \mathbf{B}\,\mathbf{B}^T + \mathbf{D}$$

and the portfolio optimization can be restated in a form in which the quadratic term is diagonalized

$$\mathcal{M} = \min_{\mathbf{x},\mathbf{y}} \left\{ \frac{1}{2} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}^T \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} - \lambda \begin{pmatrix} \boldsymbol{\mu} \\ \mathbf{0} \end{pmatrix}^T \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \,\middle|\, \begin{pmatrix} \mathbf{B}^T & -\mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{0} \wedge \mathbf{x} \in \mathcal{S} \right\}$$

We will demonstrate a closed-form solution for the above which exploits the special structure of the covariance matrix. Also, optimization code that is sophisticated enough to take advantage of sparse matrix structure will run considerably faster with the above formulation, often several orders of magnitude faster.

Parsimony

The parameters for n assets required for a mean-variance estimation are n estimates for the mean, $n(n-1)/2$ unique estimates for the covariance, and the risk-free rate, a total of $n(n-1)/2+n+1$. A multi-factor model requires the risk-free rate, k factor means and $k(k-1)/2$ covariances, the $n\times k$ factor loading matrix, n idiosyncratic (error) standard deviations, a total of $2\,n+3$.

A quick analysis shows how a direct estimate from the raw data compares with a four-factor model:

In[∘]:=

Parameter Count
4 Factors - *n* Assets

	Raw Data	Factor Model
100	5151	515
250	31 626	1265
500	125 751	2515
1000	501 501	5015

As with the CAPM, parsimony is a significant advantage. The benefits of statistical efficiency and numerical stability are similar to those delineated above for the CAPM.

Solution to a Simple Markowitz Model for the ⊥APT

Consider its solution to proportionality of the following simple quadratic program representing the tangent portfolio under an orthonormal APT-like multi-factor model (⊥APT). The linear algebra here closely follows the similar problem for the CAPM with only slight complications

$$\min\left\{\frac{1}{2}\mathbf{x}^T\,\Sigma\,\mathbf{x}-\lambda\left(\mu-r_f\,\mathbf{1}\right)^T\mathbf{x}\right\}$$

$$\Sigma\,\mathbf{x}-\lambda\left(\mu-\mathbf{1}\,r_f\right)=\mathbf{0}\quad\Longrightarrow\quad\lambda\left(\mu-\mathbf{1}\,r_f\right)=\Sigma\,\mathbf{x}$$

If we assume the ⊥ APT, then we can rewrite the covariance matrix as follows.

$$(\mu-r_f\,\mathbf{1})\;\propto\;\mathbf{B}\,\mathbf{B}^T\,\mathbf{x}+\mathbf{D}\,\mathbf{x}$$

Next we multiply both sides with the inverse of the error covariance matrix.

$$\mathbf{D}^{-1}(\mu-r_f\,\mathbf{1})\;\propto\;\mathbf{D}^{-1}\,\mathbf{B}\,\mathbf{B}^T\,\mathbf{x}+\;\mathbf{x}$$

If we had a solution for $\mathbf{B}^T\,\mathbf{x}$, then we could solve for $\mathbf{x}$. Thus, we form an equation of the above, multiply both sides by $\mathbf{B}^T$ and solve for $\mathbf{B}^T\,\mathbf{x}$.

$$\mathbf{B}^T\,\mathbf{D}^{-1}(\mu-r_f\,\mathbf{1})\;=\;\mathbf{B}^T\,\mathbf{D}^{-\mathbf{1}}\,\mathbf{B}\,\mathbf{B}^T\,\mathbf{x}+\mathbf{B}^T\,\mathbf{x}$$

$$\mathbf{B}^T\,\mathbf{x}=\left(\mathbf{B}^T\,\mathbf{D}^{-\mathbf{1}}\,\mathbf{B}+\mathbf{I}\right)^{-1}\,\mathbf{B}^T\,\mathbf{D}^{-1}(\mu-r_f\,\mathbf{1})$$

We can now substitute the solution for $\mathbf{B}^T\,\mathbf{x}$ into the original expression and solve for $\mathbf{x}$

$$\mathbf{D}^{-1}(\mu-r_f\,\mathbf{1})\;\propto\;\mathbf{D}^{-1}\,\mathbf{B}\left(\mathbf{B}^T\,\mathbf{D}^{-\mathbf{1}}\,\mathbf{B}+\mathbf{I}_{\mathcal{K}}\right)^{-1}\,\mathbf{B}^T\,\mathbf{D}^{-1}(\mu-r_f\,\mathbf{1})+\;\mathbf{x}$$

$$\mathbf{x}\propto\mathbf{D}^{-1}\left(\mathbf{I}_{\mathcal{I}}-\mathbf{B}\left(\mathbf{B}^T\,\mathbf{D}^{-\mathbf{1}}\,\mathbf{B}+\mathbf{I}_{\mathcal{K}}\right)^{-1}\,\mathbf{B}^T\,\mathbf{D}^{-1}\right)(\mu-r_f\,\mathbf{1})$$

We finally normalize the solution to sum to unity, and we have our solution.

It may seem that we have swapped a simple and straightforward equation for a tortured complicated one. The fact is that the solution above can be computed far faster than a general matrix inversion. Consider

- The matrix we must invert, $(\mathbf{B}^{\mathrm{T}}\,\mathbf{D}^{-1}\,\mathbf{B} + \mathbf{I}_{\mathcal{K}})$, has only the number of rows and columns as the number of factors; e.g., 3×3 in the FFM versus perhaps 1,000×1,000 in directly representing the assets.
- Code will run much faster if it exploits the fact that multiple diagonal matrices ($\mathbf{D}$, $\mathbf{I}_{\mathcal{I}}$, and $\mathbf{I}_{\mathcal{K}}$) appear throughout the computations; e.g., inverting $\mathbf{D}$ is trivial, and it is *never* necessary to explicitly represent diagonal matrices in their full form with extraneous zeros.
- Ordering the linear algebra can dramatically reduce both the number of operations and memory requirements. This is also the case when dealing with other computations involving the factor model; e.g., computing $(\mathbf{B}^{T}\,\mathbf{x})^{T}\,(\mathbf{B}^{T}\,\mathbf{x})$ requires materially less computation time and working memory than does the mathematically identical $\mathbf{x}^{T}(\mathbf{B}\,\mathbf{B}^{T})\,\mathbf{x}$.

A Small Investment Universe

Risky Assets

```
In[∘]:= vsTickers = FinancialData["^DJI", "Members"]
        vsNames = FinancialData[#, "Name"] & /@ vsTickers;
        TableForm[{vsTickers, vsNames}ᵀ, TableHeadings → {Range[Length[vsTickers]]}]

Out[∘]= {NYSE:MMM, NYSE:AXP, NASDAQ:AAPL, NYSE:BA, NYSE:CAT, NYSE:CVX, NASDAQ:CSCO, NYSE:KO,
         NYSE:DIS, NYSE:DWDP, NYSE:XOM, NYSE:GS, NYSE:HD, NASDAQ:INTC, NYSE:IBM,
         NYSE:JNJ, NYSE:JPM, NYSE:MCD, NYSE:MRK, NASDAQ:MSFT, NYSE:NKE, NYSE:PFE,
         NYSE:PG, NYSE:TRV, NYSE:UNH, NYSE:UTX, NYSE:VZ, NYSE:V, NASDAQ:WBA, NYSE:WMT}
```

Out[∘]//TableForm=

1	NYSE:MMM	3M
2	NYSE:AXP	American Express
3	NASDAQ:AAPL	Apple
4	NYSE:BA	Boeing
5	NYSE:CAT	Caterpillar
6	NYSE:CVX	Chevron
7	NASDAQ:CSCO	Cisco
8	NYSE:KO	Coca-Cola
9	NYSE:DIS	Disney
10	NYSE:DWDP	DowDuPont Inc
11	NYSE:XOM	Exxon Mobil
12	NYSE:GS	Goldman Sachs
13	NYSE:HD	Home Depot
14	NASDAQ:INTC	Intel
15	NYSE:IBM	IBM
16	NYSE:JNJ	Johnson & Johnson
17	NYSE:JPM	JPMorgan Chase
18	NYSE:MCD	McDonald's
19	NYSE:MRK	Merck & Co.
20	NASDAQ:MSFT	Microsoft
21	NYSE:NKE	Nike
22	NYSE:PFE	Pfizer
23	NYSE:PG	Procter & Gamble
24	NYSE:TRV	Travelers
25	NYSE:UNH	UnitedHealth
26	NYSE:UTX	United Technologies
27	NYSE:VZ	Verizon Communications
28	NYSE:V	Visa
29	NASDAQ:WBA	Walgreens Boots Alliance Inc
30	NYSE:WMT	Wal-Mart Stores

```
In[∘]:= mnDateRange = {{2008, 6, 30}, {2020, 02, 28}}

Out[∘]= {{2008, 6, 30}, {2020, 2, 28}}
```

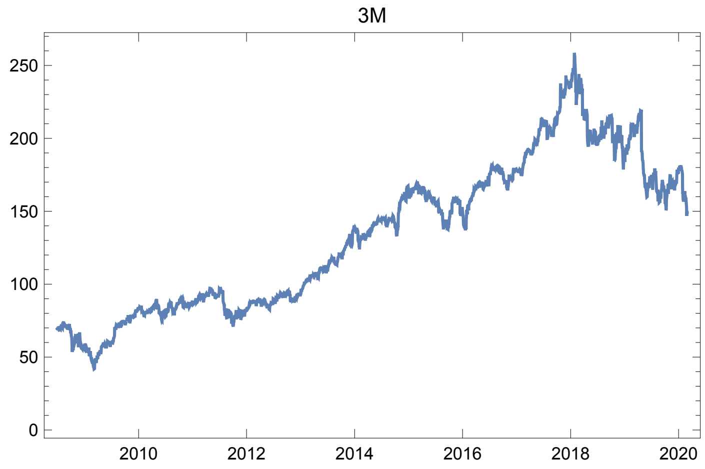
We create a working buffer to hold the individual price histories:

```
In[∘]:= buffer = (FinancialData[#, mnDateRange] &) /@ vsTickers;
```

```
In[]:= Length@buffer
       Head@First@buffer
Out[]= 30
Out[]= TemporalData
```

```
In[]:= DateListPlot[First@buffer, PlotLabel → First@vsNames]
```

Out[]=



```
In[]:= TableForm[({First@#, Last@#, Length@#} &[#["Dates"]]) & /@ buffer,
         TableHeadings → {vsNames, {"start", "end", "length"}}]
```

Out[]//TableForm=

	start	end
3M	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
American Express	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Apple	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Boeing	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Caterpillar	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Chevron	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Cisco	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Coca-Cola	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Disney	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
DowDuPont Inc	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Exxon Mobil	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Goldman Sachs	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Home Depot	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Intel	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4

IBM	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Johnson & Johnson	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
JPMorgan Chase	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
McDonald's	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Merck & Co.	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Microsoft	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Nike	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Pfizer	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Procter & Gamble	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Travelers	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
UnitedHealth	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
United Technologies	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Verizon Communications	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Visa	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Walgreens Boots Alliance Inc	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Wal-Mart Stores	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4

```
In[•]:= vxCalendar = Union @@ (#["Dates"] & /@ buffer);

In[•]:= completedBuffer = TimeSeries[#[vxCalendar], {vxCalendar}] & /@ buffer;

In[•]:= TableForm[({First@#, Last@#, Length@#} &[#["Dates"]]) & /@ completedBuffer,
         TableHeadings → {vsNames, {"start", "end", "length"}}]
```

Out[•]//TableForm=

	start	end
3M	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
American Express	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Apple	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Boeing	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Caterpillar	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Chevron	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Cisco	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4
Coca-Cola	Mon 30 Jun 2008 00:00:00 GMT−4.	Fri 28 Feb 2020 00:00:00 GMT−4

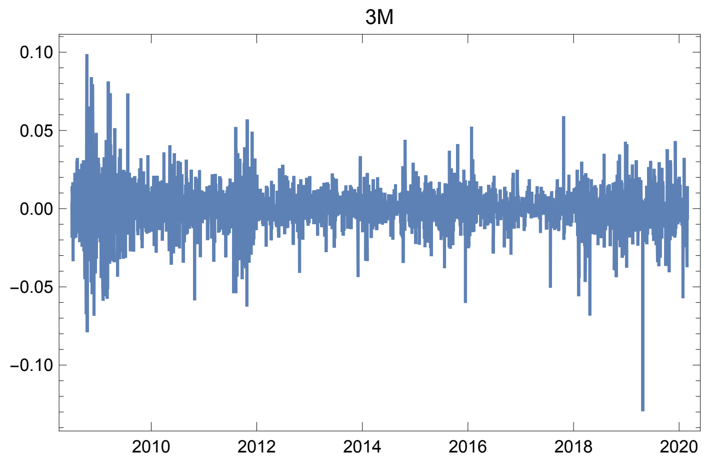
```
Disney                         Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
DowDuPont Inc                  Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Exxon Mobil                    Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Goldman Sachs                  Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Home Depot                     Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Intel                          Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
IBM                            Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Johnson & Johnson              Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
JPMorgan Chase                 Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
McDonald's                     Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Merck & Co.                    Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Microsoft                      Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Nike                           Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Pfizer                         Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Procter & Gamble               Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Travelers                      Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
UnitedHealth                   Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
United Technologies            Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Verizon Communications         Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Visa                           Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Walgreens Boots Alliance Inc   Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
Wal-Mart Stores                Mon 30 Jun 2008 00:00:00 GMT−4.    Fri 28 Feb 2020 00:00:00 GMT−4
```

```
In[∘]:= FreeQ[completedBuffer, _Missing]
Out[∘]= True
```

```
In[∘]:= tsDJIReturns =
          TimeSeries[Rest[#]/Most[#] - 1 &[#["Values"]], {Rest[#["Dates"]]}] & /@ completedBuffer;
```

```
In[•]:= DateListPlot[First@tsDJIReturns, PlotRange → All, PlotLabel -> First@vsNames]
```

Out[•]=



```
In[•]:= Export[FileNameJoin[{NotebookDirectory[], "tsDJIReturns.m"}], tsDJIReturns]

Out[•]= /Users/robertjfrey/Documents/Work/Stony Brook
          University/AMS/QF/public_html/Instruction/Spring2020/AMS512/Class05/tsDJIReturns
          .m

In[•]:= tsDJIReturns = Import[FileNameJoin[{NotebookDirectory[], "tsDJIReturns.m"}]];
```

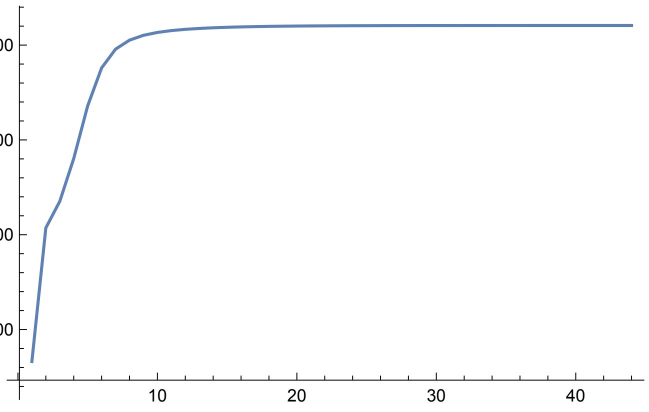
Building A Factor Model

```
In[•]:= Short[QuantityMagnitude[tsDJIReturns⟦1⟧["Values"]]]

Out[•]//Short= {0.00273031, ≪2935≫, -0.00612679}

In[•]:= mnDowReturns = Transpose[QuantityMagnitude[#["Values"]] & /@ tsDJIReturns];
        Dimensions@mnDowReturns

Out[•]= {2937, 30}
```

Estimating The Model

```
In[•]:= iN = Length[mnDowReturns]
        mnCov  =  Covariance[mnDowReturns];
        iOrder = 2;

Out[•]= 2937

In[•]:= {{mnFactors, mnErrDiag}, vnLogLikHistory} =
          xFactorFitMLE[iN, mnCov, xInitializeFactorModel[mnCov, iOrder]];
```

In[•]:=
```
ListPlot[vnLogLikHistory, Joined → True, PlotRange → All]
```

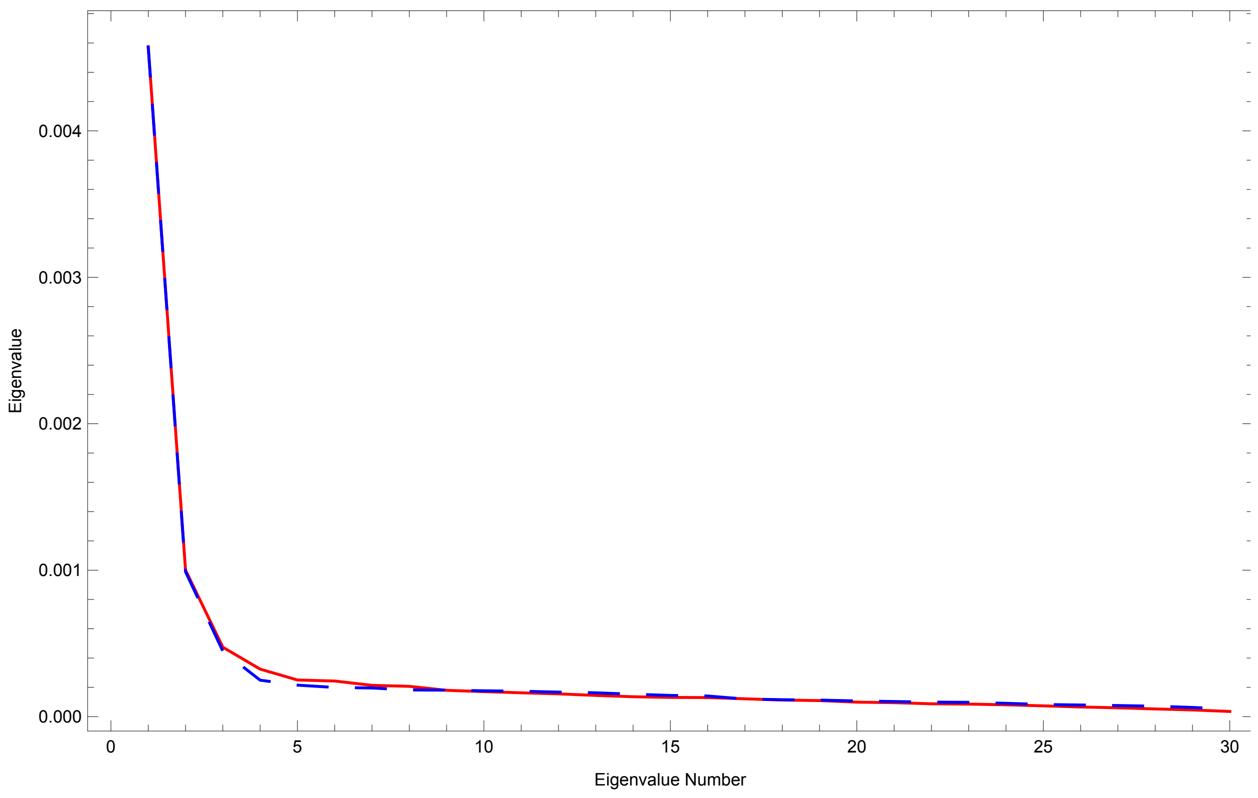
Out[•]=


In[•]:=
```
mnApproxCov = (mnFactors.Transpose[mnFactors]) + mnErrDiag;
```

Spectra of the Covariance Matrices

```
In[•]:= Show[{
  ListPlot[
   Eigenvalues[mnCov],
   Joined → True,
   Frame → True,
   FrameLabel → {
     "Eigenvalue Number",
     "Eigenvalue",
     StyleForm["Covariance Spectrum", FontSize → 14],
     ""
    },
   AxesOrigin → {0, 0},
   PlotRange → All,
   PlotStyle → {RGBColor[1, 0, 0], Thickness[0.0025]},
   ImageSize → 650
  ],
  ListPlot[
   Eigenvalues[mnApproxCov],
   Joined → True,
   PlotStyle → {RGBColor[0, 0, 1], Dashing[{0.025}], Thickness[0.0025]},
   PlotRange → All
  ]
 }]
```

Out[∘]=



Eigenvalue
0.004
0.003
0.002
0.001
0.000
0
5
10
15
20
25
30
Eigenvalue Number

In[•]:=
```
TableForm[mnFactors,
 TableHeadings → {vsTickers, {"Factor1", "Factor2", "Factor3", "Factor4"}}]
```

Out[•]//TableForm=
```
             | Factor1        Factor2
-------------+----------------------------
NYSE:MMM     | -0.0109675     0.000695423
NYSE:AXP     | -0.0175098     -0.0054299
NASDAQ:AAPL  | -0.0115749     -0.00181366
NYSE:BA      | -0.0129005     0.000282105
NYSE:CAT     | -0.0153229     -0.00200115
NYSE:CVX     | -0.0127622     0.00214959
NASDAQ:CSCO  | -0.0130748     -0.000150168
NYSE:KO      | -0.00695933    0.00387341
NYSE:DIS     | -0.0129838     0.000560539
NYSE:DWDP    | -0.0163066     -0.000622438
NYSE:XOM     | -0.0114314     0.0027511
NYSE:GS      | -0.0171427     -0.00743761
NYSE:HD      | -0.0114448     0.0000946597
NASDAQ:INTC  | -0.0126221     -0.00025582
NYSE:IBM     | -0.00969215    0.000261454
NYSE:JNJ     | -0.00721216    0.00387787
NYSE:JPM     | -0.0190327     -0.00843134
NYSE:MCD     | -0.0069542     0.00249321
NYSE:MRK     | -0.0094458     0.00459109
NASDAQ:MSFT  | -0.0121422     0.000561388
NYSE:NKE     | -0.0111966     0.000468603
NYSE:PFE     | -0.00933364    0.00320179
NYSE:PG      | -0.00710118    0.00386547
NYSE:TRV     | -0.0123327     0.000344473
NYSE:UNH     | -0.0123297     0.00226689
NYSE:UTX     | -0.0121086     0.000570029
NYSE:VZ      | -0.00811373    0.00286246
NYSE:V       | -0.0120252     -0.0021979
NASDAQ:WBA   | -0.00972265    0.00216229
NYSE:WMT     | -0.00645368    0.00336505
```

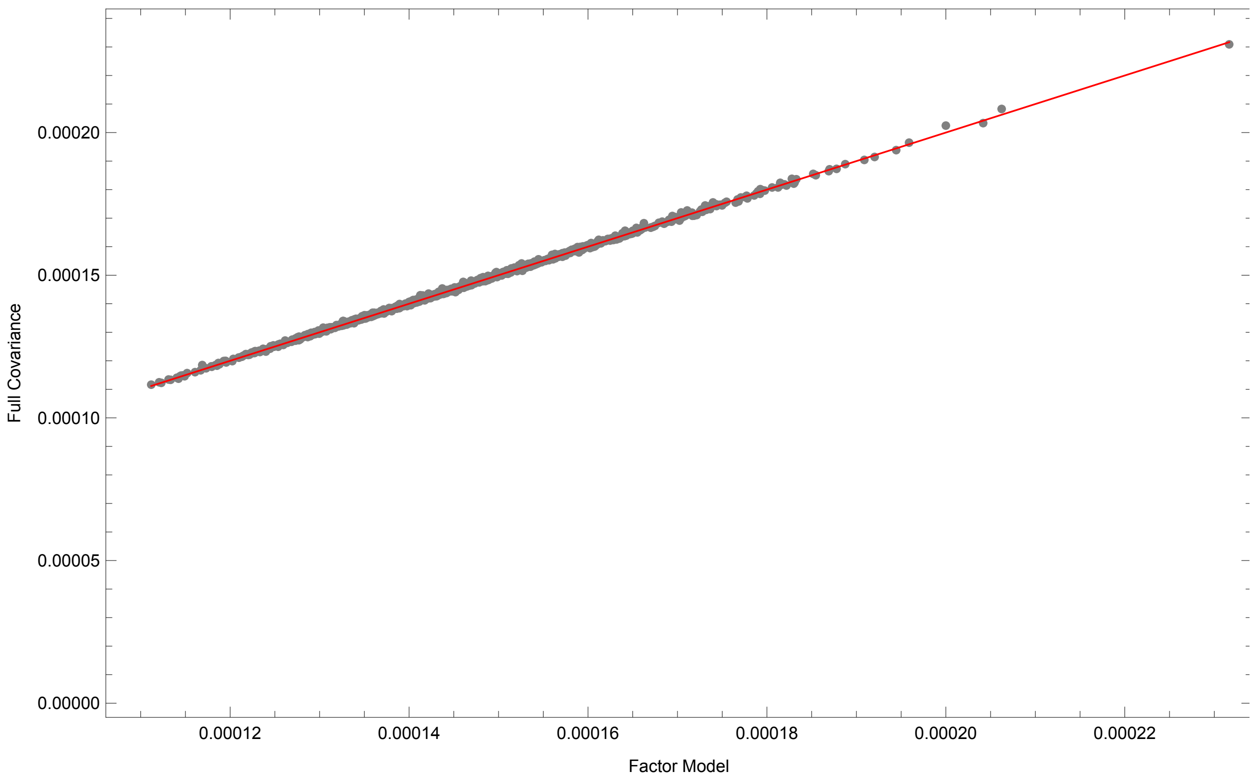
In[•]:=
```
xRandomSimplexDirichlet[d_, n_] := Block[
   {α},
   α = Array[1 &, d];
   Append[#, 1 - Total[#]] & /@ RandomVariate[DirichletDistribution[α], n]
  ];
```

Evaluating The Model

In[•]:=
```
mnSample = ({#1.mnApproxCov.#1, #1.mnCov.#1} &) /@ xRandomSimplexDirichlet[30, 1000];
```

In[∘]:=
```
Show[{
  ListPlot[
   mnSample,
   PlotRange → All,
   PlotStyle → {, PointSize → Medium, GrayLevel[0.5]},
   Frame → True,
   FrameLabel → {
     "Factor Model",
     "Full Covariance",
     StyleForm["Random Portfolio σ² Comparison", FontSize → 16],
     ""
    },
   ImageSize → 650
  ],
  Plot[
   x,
   {x, Min[Flatten[mnSample]], Max[Flatten[mnSample]]},
   PlotStyle → {RGBColor[1, 0, 0], Thickness[0.0015]}
  ]
 }]
```

Out[∘]=



```
In[•]:= Det[mnCov]
        Det[mnApproxCov]
Out[•]= 1.044 × 10^-115
Out[•]= 6.2673 × 10^-115
```

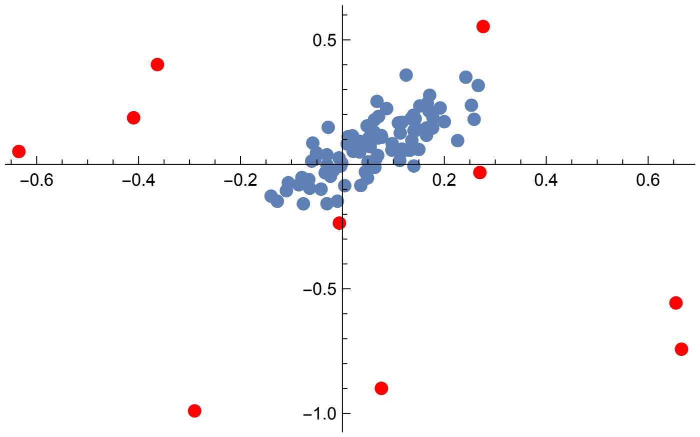
Robustness

Simulation

```
In[•]:= vnMNice = {0.07, 0.08};
        mnΣNice = Outer[Times, #, #] &[{0.1, 0.12}] {{1, 0.75}, {0.75, 1}};
        MatrixForm /@ {vnMNice, mnΣNice}
Out[•]= {( 0.07 ), ( 0.01   0.009  )}
         ( 0.08 )  ( 0.009  0.0144 )

In[•]:= vnMOutlier = {-0.05, -0.05};
        mnΣOutlier = Outer[Times, #, #] &[{0.5, 0.5}] × IdentityMatrix[2];
        MatrixForm /@ {vnMOutlier, mnΣOutlier}
Out[•]= {( -0.05 ), ( 0.25  0.   )}
         ( -0.05 )  ( 0.    0.25 )

In[•]:= mnSampleNice = RandomReal[MultinormalDistribution[vnMNice, mnΣNice], {90}];
        mnSampleOutlier =
          RandomReal[MultinormalDistribution[vnMOutlier, mnΣOutlier], {10}];
        mnSampleAll = RandomSample[Join[mnSampleNice, mnSampleOutlier]];

In[•]:= vnMAll = Mean[mnSampleAll];
        mnΣAll = Covariance[mnSampleAll];
        MatrixForm /@ {vnMAll, mnΣAll}
Out[•]= {( 0.0585044 ), ( 0.0265239   0.00176805 )}
         ( 0.0453004 )  ( 0.00176805  0.0479092  )
```

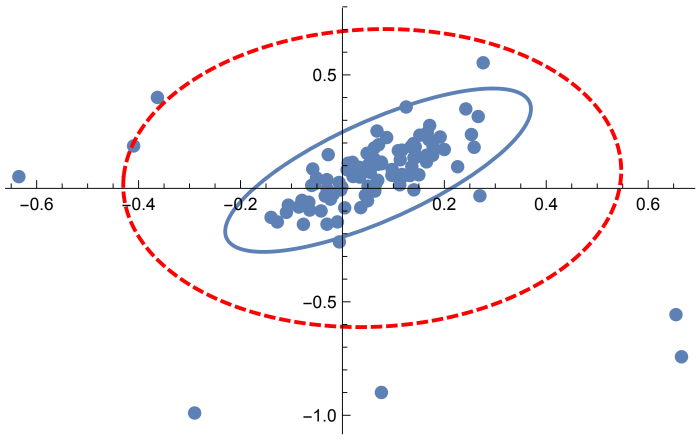
In[•]:=
```
Show[
 ListPlot[mnSampleNice, PlotRange → All, PlotStyle → {PointSize[Large]}],
 ListPlot[mnSampleOutlier, PlotRange → All, PlotStyle → {Red, PointSize[Large]}],
 PlotRange → All
]
```

Out[•]=



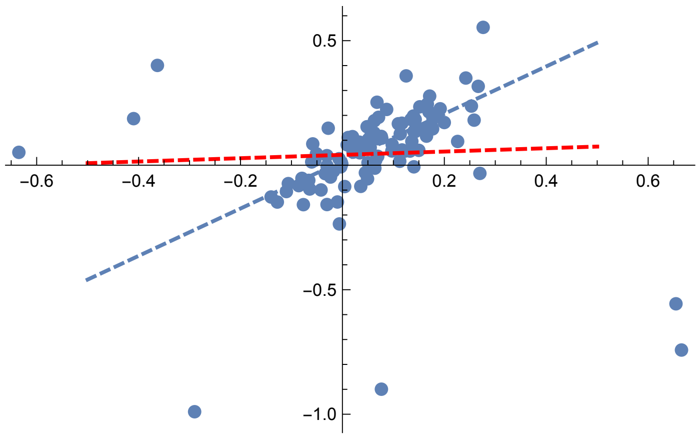
In[•]:=
```
Show[
 ListPlot[mnSampleAll, PlotRange → All, PlotStyle → {PointSize[Large]}],
 ParametricPlot[
  (3 Inverse[CholeskyDecomposition[Inverse[mnΣNice]]].{Cos[x], Sin[x]}) + vnMNice,
  {x, -π, π}, PlotStyle → {Thick}],
 ParametricPlot[
  (3 Inverse[CholeskyDecomposition[Inverse[mnΣAll]]].{Cos[x], Sin[x]}) + vnMAll,
  {x, -π, π}, PlotStyle → {Red, Dashed, Thick}],
 PlotRange → All
]
```

Out[•]=



```
In[•]:= vnLinearNice = PseudoInverse[{1, #} & /@ (Last /@ #)].(First /@ #) &[mnSampleNice]
        vnLinearAll = PseudoInverse[{1, #} & /@ (Last /@ #)].(First /@ #) &[mnSampleAll]

Out[•]= {0.0151398, 0.625384}

Out[•]= {0.0568326, 0.0369041}

In[•]:= modelAll = LinearModelFit[mnSampleAll, ξ, ξ];
        modelNice = LinearModelFit[mnSampleNice, ξ, ξ];
        vnLinearAll = modelAll["BestFitParameters"]
        vnLinearNice = modelNice["BestFitParameters"]

Out[•]= {0.0414006, 0.0666587}

Out[•]= {0.0161953, 0.951026}

In[•]:= Show[
         ListPlot[mnSampleAll, PlotRange → All, PlotStyle → {PointSize[Large]}],
         Plot[vnLinearNice.{1, x}, {x, -0.5, 0.5}, PlotStyle → {Dashed, Thick}],
         Plot[vnLinearAll.{1, x}, {x, -0.5, 0.5}, PlotStyle → {Red, Dashed, Thick}]
        ]
```

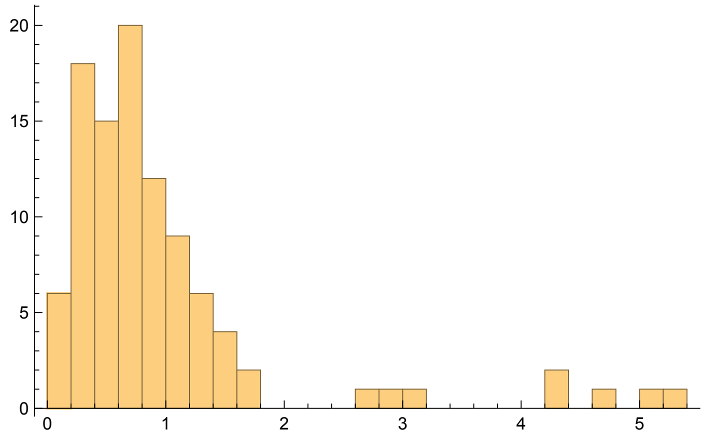
Out[•]=



Outlier Rejection

```
In[•]:= xDistances[x_] := Module[
          {iΣ, m},
          m = Mean[x];
          iΣ = Inverse[Covariance[x]];
          √((# - m).iΣ.(# - m)) & /@ x
         ]
```

```
In[•]:= Histogram[xDistances[mnSampleAll], PlotRange → All]
```

Out[•]=


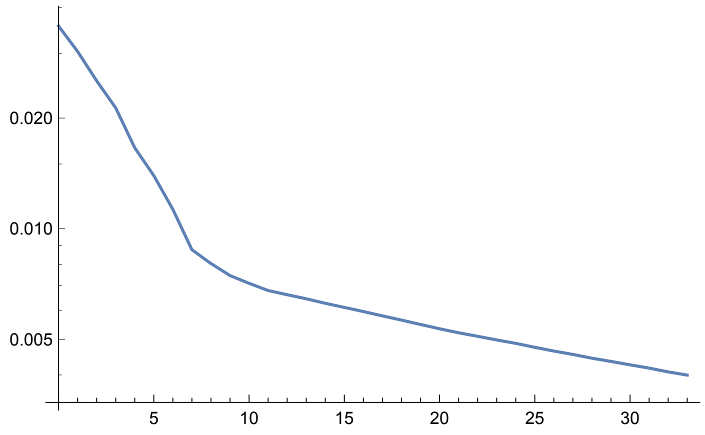
```
In[•]:= xDropFurthest[x_] := Module[
          {d},
          d = xDistances[x];
          Pick[x, # < Max[d] & /@ d]
         ]
```

```
In[•]:= xVolume[c_] := √Det[c] ;
```

```
In[•]:= mnSampleSubset = mnSampleAll;
        mnSubsetBoundary = {{0, xVolume[Covariance[mnSampleSubset]]}};
```

```
In[•]:= For[i = 1, i ≤ Floor[Length[mnSampleSubset] / 2], i++,
         mnSampleSubset = xDropFurthest[mnSampleSubset];
         mnSubsetBoundary =
          Append[mnSubsetBoundary, {i, xVolume[Covariance[mnSampleSubset]]}]
        ]
```

```
In[•]:= ListLogPlot[mnSubsetBoundary, Joined → True, PlotRange → All]
```

Out[•]=


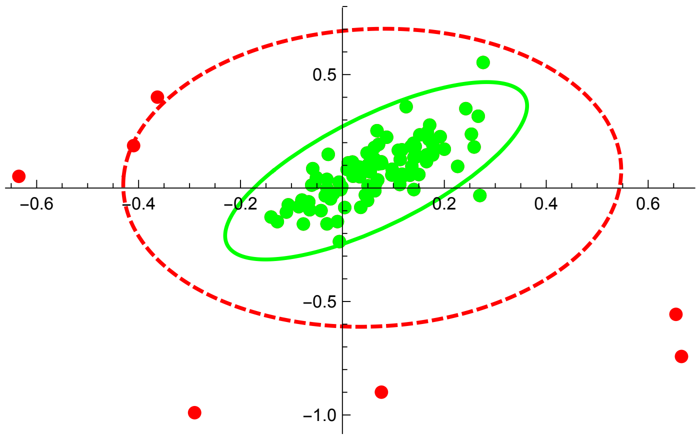
```
In[•]:= mnSampleRobust = Nest[xDropFurthest[#] &, mnSampleAll, 7];
```

In[•]:=
```
vnMRobust = Mean[mnSampleRobust];
mnΣRobust = Covariance[mnSampleRobust];
MatrixForm /@ {vnMRobust, mnΣRobust}
```

Out[•]=
```
{( 0.0661481 ), ( 0.00975192  0.00941791 )}
 ( 0.0761198 )  ( 0.00941791   0.0169724 )
```

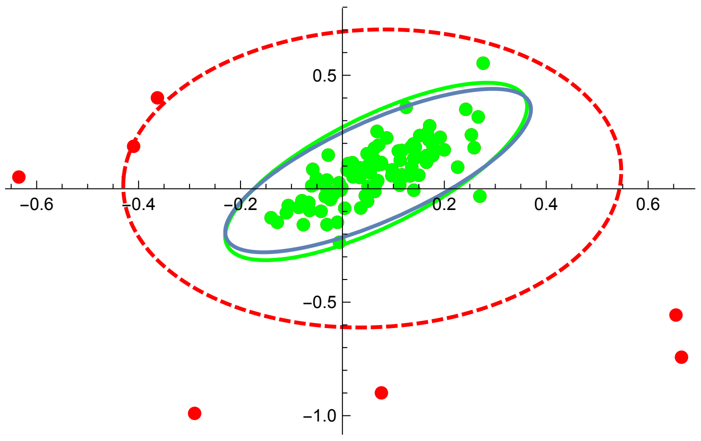
In[•]:=
```
Show[
 ListPlot[mnSampleAll, PlotRange → All, PlotStyle → {Red, PointSize[Large]}],
 ListPlot[mnSampleRobust, PlotRange → All, PlotStyle → {Green, PointSize[Large]}],
 ParametricPlot[
  (3 Inverse[CholeskyDecomposition[Inverse[mnΣRobust]]].{Cos[x], Sin[x]}) +
   vnMRobust, {x, -π, π}, PlotStyle → {Green, Thick}],
 ParametricPlot[(3 Inverse[CholeskyDecomposition[Inverse[mnΣAll]]].
      {Cos[x], Sin[x]}) + vnMAll, {x, -π, π}, PlotStyle → {Red, Dashed, Thick}],
 PlotRange → All
]
```

Out[•]=



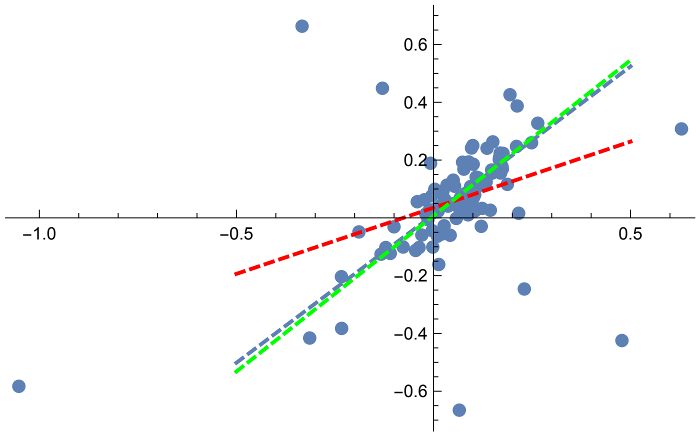
```
In[•]:= Show[
         ListPlot[mnSampleAll, PlotRange → All, PlotStyle → {Red, PointSize[Large]}],
         ListPlot[mnSampleRobust, PlotRange → All, PlotStyle → {Green, PointSize[Large]}],
         ParametricPlot[
          (3 Inverse[CholeskyDecomposition[Inverse[mnΣRobust]]].{Cos[x], Sin[x]}) +
           vnMRobust, {x, -π, π}, PlotStyle → {Green, Thick}],
         ParametricPlot[(3 Inverse[CholeskyDecomposition[Inverse[mnΣAll]]].
              {Cos[x], Sin[x]}) + vnMAll, {x, -π, π}, PlotStyle → {Red, Dashed, Thick}],
         ParametricPlot[(3 Inverse[CholeskyDecomposition[Inverse[mnΣNice]]].
              {Cos[x], Sin[x]}) + vnMNice, {x, -π, π}, PlotStyle → {Thick}],
         PlotRange → All
        ]
```

Out[•]=



```
In[•]:= vnLinearRobust = LinearModelFit[#, ξ, ξ]["BestFitParameters"] &[mnSampleRobust]
Out[•]= {0.0122373, 0.965749}
```

In[•]:=
```
Show[
 ListPlot[mnSampleAll, PlotRange → All, PlotStyle → {PointSize[Large]}],
 Plot[vnLinearNice.{1, x}, {x, -0.5, 0.5}, PlotStyle → {Dashed, Thick}],
 Plot[vnLinearAll.{1, x}, {x, -0.5, 0.5}, PlotStyle → {Red, Dashed, Thick}],
 Plot[vnLinearRobust.{1, x}, {x, -0.5, 0.5}, PlotStyle → {Green, Dashed, Thick}]
]
```

Out[•]=


In[•]:=
```
TableForm[{vnLinearAll, vnLinearNice, vnLinearRobust},
 TableHeadings → {{"All", "Nice", "Robust"}, {"Intercept", "Slope"}}]
```

Out[•]//TableForm=

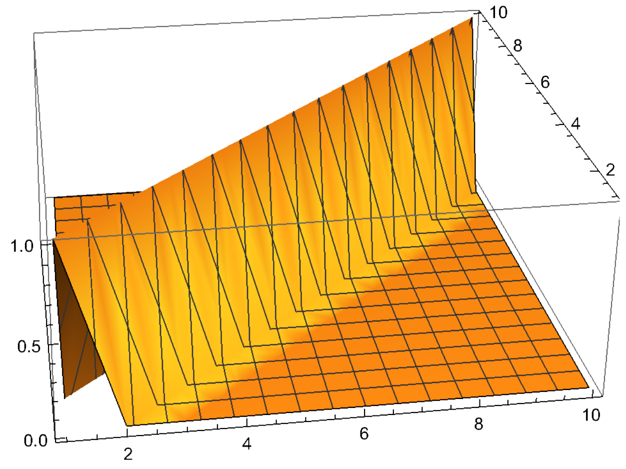
	Intercept	Slope
All	0.0414006	0.0666587
Nice	0.0161953	0.951026
Robust	0.0122373	0.965749

Sampling Effects on Covariance Spectra

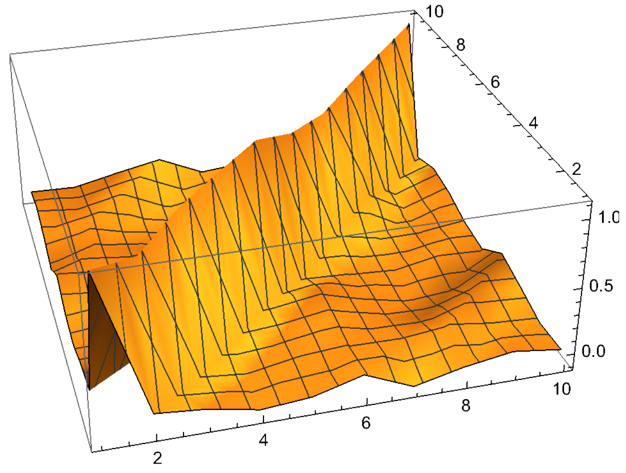
In[•]:=
```
vnSimple =
  RandomVariate[MultinormalDistribution[Array[0 &, 10], IdentityMatrix[10]], {200}];
```

In[•]:=
```
mnCov = Covariance[vnSimple];
ListPlot3D[IdentityMatrix[10], PlotRange → All]
ListPlot3D[mnCov, PlotRange → All]
ListPlot3D[mnCov - IdentityMatrix[10], PlotRange → All]
Tr[mnCov, List]
Histogram[Flatten[Table[mnCov⟦i, j⟧, {i, 1, Length[mnCov]}, {j, 1, i - 1}]]]
```

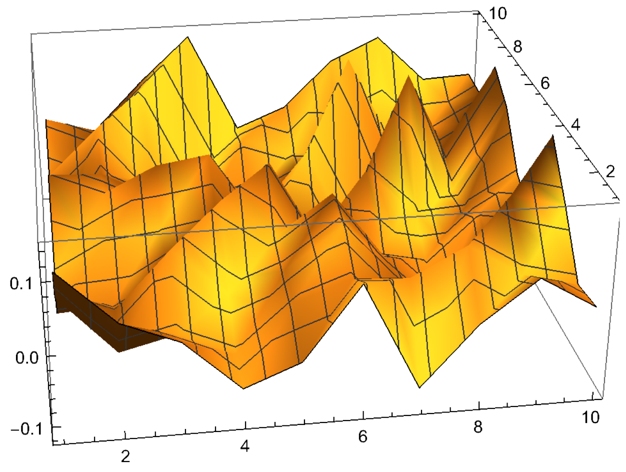
Out[•]=



Out[•]=

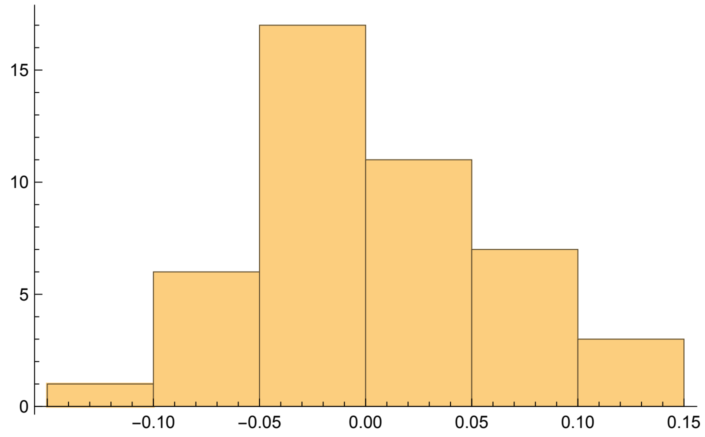


Out[•]=



```
Out[•]= {1.10468, 0.944054, 1.04568, 1.07474,
         1.09058, 0.930577, 0.87975, 0.952723, 1.00394, 1.05157}
```

Out[•]=

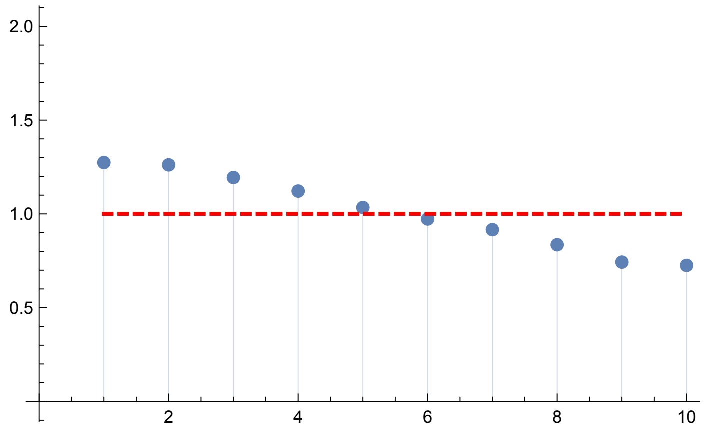


In[•]:=
```
Eigenvalues[IdentityMatrix[10]]
```

Out[•]=
```
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
```

In[•]:=
```
Show[
 ListPlot[Eigenvalues[mnCov], Filling → 0, PlotStyle → {PointSize[Large]}],
 ListLinePlot[Eigenvalues[IdentityMatrix[10]], PlotStyle → {Red, Dashed, Thick}],
 AxesOrigin → {0, 0},
 PlotRange → {{0, 10}, {0, 2}}
]
```

Out[•]=



Simple Mean-Variance Forms

A Simple Form

We wish to select a portfolio which achieves some minimum return

$$\mathcal{M} = \min_{x} \left\{ \frac{1}{2} \mathbf{x}^T \Sigma \mathbf{x} \;\middle|\; \boldsymbol{\mu}^T \mathbf{x} = r_{\text{targ}} \wedge 1^T \mathbf{x} = 1 \right\}$$

$$\mathcal{L}(\mathcal{M}) = \frac{1}{2} \mathbf{x}^T \Sigma \mathbf{x} - \lambda_{\text{ret}} \left(\boldsymbol{\mu}^T \mathbf{x} - r_{\text{targ}} \right) - \lambda_{\text{cap}} \left(\mathbf{1}^T \mathbf{x} - 1 \right)$$

$$\nabla \mathcal{L}(\mathcal{M}) = \boldsymbol{\Sigma}\,\mathbf{x} - \lambda_{\text{ret}}\,\boldsymbol{\mu} - \lambda_{\text{cap}}\,\mathbf{1} = \mathbf{0}$$

$$\mathbf{x} = \lambda_{\text{ret}}\,\boldsymbol{\Sigma}^{-1}\,\boldsymbol{\mu} + \lambda_{\text{cap}}\,\boldsymbol{\Sigma}^{-1}\,\mathbf{1}$$

$$1 = \mathbf{1}^T\,\mathbf{x} = \lambda_{\text{ret}}\,\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\boldsymbol{\mu} + \lambda_{\text{cap}}\,\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\mathbf{1} \;\Rightarrow \lambda_{\text{cap}} = \frac{1 - \lambda_{\text{ret}}\,\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\boldsymbol{\mu}}{\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\mathbf{1}}$$

$$\mathbf{x} = \lambda_{\text{ret}}\,\boldsymbol{\Sigma}^{-1}\,\boldsymbol{\mu} + \left(\frac{1 - \lambda_{\text{ret}}\,\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\boldsymbol{\mu}}{\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\,\mathbf{1}}\right)\boldsymbol{\Sigma}^{-1}\,\mathbf{1}$$

Tangent Portfolio

$$\mathbf{x}_{\text{tang}} = \left(\frac{1}{\mathbf{1}^T\,\boldsymbol{\Sigma}^{-1}\left(\boldsymbol{\mu} - \mathbf{1}\,r_f\right)}\right)\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \mathbf{1}\,r_f)$$