The Mathematica® Journal

Selected Financial Applications

Ramesh Adhikari

This article shows how to use some of Mathematica's built-in financial functions and define new functions useful for the practical analysis of real-world financial data. The main topics covered are linear programming and its application in bond portfolio management, conditional value-at-risk minimization, introductory time-series analysis, simulation, bootstrapping, robust equity portfolio optimization and artificial intelligence.

1. Introduction

Our main objective is to apply the Wolfram Language to solve financial models. We do not explain financial concepts or any mathematical background related to the financial applications introduced in this article. Nor do we introduce Mathematica. Wellin [1] gives a good introduction to programming in Mathematica. Mathematical background related to the models discussed here can be found in standard textbooks, including the ones cited.

First, we define three supporting functions used in the rest of this article.

`getReturnsData`

The `getReturnsData` function downloads historical stock returns given its four arguments:

- a list of one or more ticker symbols
- the start date as a date object
- the end date as a date object
- the period as the frequency of data

```
In[29]:=
      getReturnsData[symbols : {__String}, startDate_,
        endDate_, period_] :=
       Module[
        {pricedata, mergedpricedata, allnonmissingprices},
        pricedata =
         FinancialData[#, {startDate, endDate, period}] & /@
          symbols;
        mergedpricedata = TimeSeriesThread[# &, pricedata,
          ResamplingMethod → Missing[]];
        allnonmissingprices =
         Select[ QuantityMagnitude[mergedpricedata["Values"]],
          ! MemberQ[#, QuantityMagnitude[Missing[]]] &];
        Differences@allnonmissingprices / Most@allnonmissingprices
       ]
```

basicStats

The basicStats function computes 11 basic descriptive statistics given a list or a matrix of returns as its input argument.

```
In[30]:= basicStats[list_List] :=
       Module[
        {statsFunctions, statisticsnames, temporaryvariable},
        statisticsnames = {"N", "Mean", "Minimum Value",
          "Maximum Value", "25th Percentile Value",
          "50th Percentile Value", "75th Percentile Value" ,
          "Standard Deviation", "Variance", "Skewness",
          "Kurtosis"};
        statsFunctions =
         Flatten[{Length[#], Mean[#], Min[#], Max[#],
            Quartiles[#], StandardDeviation[#], Variance[#],
            Skewness[#], Kurtosis[#]}] &;
        temporaryvariable = If[Depth@list == 2, {#} & /@ list, list];
        Transpose@
         Prepend[statsFunctions /@ Transpose@temporaryvariable,
          statisticsnames]
        ]
```

◻ visualHistogram

The `visualHistogram` function was taken from Stack Exchange Network (https://mathematica.stackexchange.com/questions/194234/plot-of-histogram-similar-to-output-from-risk) and modified. The function takes a vector of numerical data and returns a histogram with a handle on each side. You can drag these two thin vertical gray lines to vary the percentage of data within two values.

```
In[36]:= visualHistogram[data_] := DynamicModule[
           {haldle, minvalue, maxvalue, histogram, yaxisrange},
           Manipulate[
            Show[
             histogram,
             Plot[PDF[haldle, x],
              {x, xaxisrange⟦1, 1⟧, xaxisrange⟦2, 1⟧},
              Exclusions → None, Filling → {1 → Axis},
              FillingStyle → Red, PlotRange → yaxisrange,
              PlotStyle → Directive[Red]],
             Axes → None, Frame → True,
             GridLines → {{xaxisrange⟦1, 1⟧, xaxisrange⟦2, 1⟧},
               Automatic}, Method → {"GridLinesInFront" → True},
             PlotLabel → StringForm["Percent of Data Inside = ``%",
               100 Round[Abs[CDF[haldle, xaxisrange⟦2, 1⟧] -
                   CDF[haldle, xaxisrange⟦1, 1⟧]], 0.001]]
            ],
            {{xaxisrange, {{minvalue, 0}, {maxvalue, 0}}},
             Locator, Appearance → None},
            Initialization :→ (
              haldle = HistogramDistribution[data];
              {minvalue, maxvalue} = First[haldle["Domain"]];
              histogram = Histogram[data, Automatic, "PDF",
                ChartBaseStyle → Directive[EdgeForm[None], Black]];
              yaxisrange =
               Last[
                Through[{Min, Max}@#] & /@
                   Transpose[Join @@ Cases[# // ToBoxes,
                      RectangleBox[x_, y_, ___] :→ {x, y}, ∞]] &@
                 histogram];
             )
           ]
          ]
```

□ Example

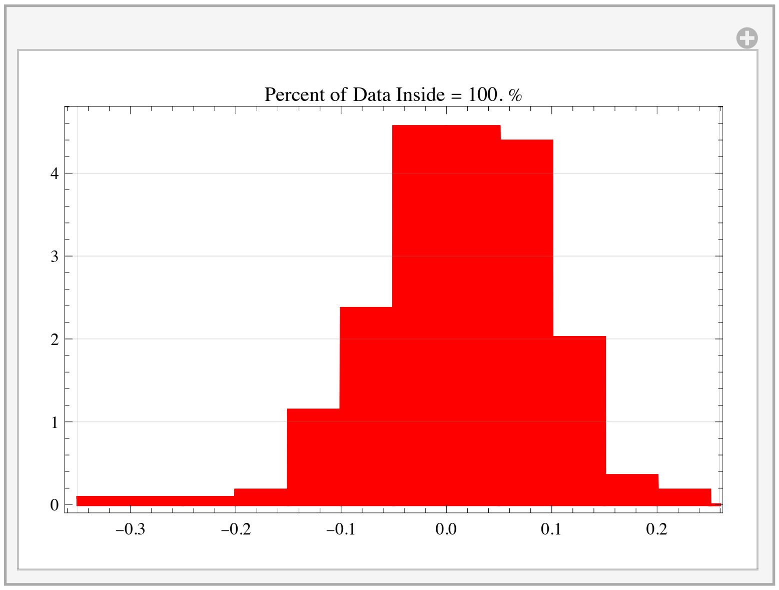
For example, this downloads monthly historical stock returns for Boeing Company (BA), Apple Inc. (AAPL) and NVIDIA Corporation (NVDA) for the period May 1, 2000, to May 30, 2019, and computes descriptive statistics. We chose returns of the Boeing Company for the histogram. Drag the handles at the far ends of the histogram (the thin vertical lines) to see the percentage of values that lie within two values.

```
In[32]:= returns1 = getReturnsData[{"BA", "AAPL", "NVDA"},
            {2000, 5, 1}, {2019, 5, 30}, "Month"];
```

```
In[33]:= Column[{
           Text@TableForm[basicStats@returns1,
             TableHeadings →
              {None, {"Statistics", "BA", "AAPL", "NVDA"}}],
           visualHistogram[returns1[[All, 1]] ]
          }, Alignment → Center]
```

Statistics	BA	AAPL	NVDA
N	228	228	228
Mean	0.013036	0.025185	0.0272098
Minimum Value	−0.345703	−0.577436	−0.567623
Maximum Value	0.226094	0.453783	0.833916
25th Percentile Value	−0.0344627	−0.0352143	−0.0603671
50th Percentile Value	0.0170044	0.0284498	0.0249775
75th Percentile Value	0.06912	0.0964163	0.117196
Standard Deviation	0.0811269	0.115341	0.176888
Variance	0.00658158	0.0133036	0.0312893
Skewness	−0.526538	−0.563102	0.504439
Kurtosis	4.41683	6.64109	5.59732

Out[33]=



The article is organized as follows:

- Section 2 introduces linear programming and applies it to bond portfolio management.
- Section 3 discusses mean-conditional value-at-risk portfolio optimization.
- Section 4 shows how to use built-in functions for introductory financial time-series analysis.
- Section 5 describes how simulation can be used in capital budgeting.
- Section 6 applies bootstrapping to risk management and financial planning.
- Section 7 describes robust equity portfolio optimization.
- Section 8 introduces functions useful for machine learning.
- Finally, the last section concludes the discussion.

2. Linear Programming and Applications

In this section, we illustrate a few applications of linear programming to financial problems similar to those in Cornuéjols, Peña and Tütüncü [2]. A linear program is an optimization problem whose purpose is to minimize or maximize a linear objective function subject to linear constraints. We first decide the decision variables, objective function and constraints. Then we find the values of the decision variables to optimize the objective function.

For vectors $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{d} \in \mathbb{R}^p$ and matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{D} \in \mathbb{R}^{p\times n}$, we can specify a general linear program in the following form:

$$\begin{array}{lll} \underset{\mathbf{x}}{\text{minimize}} & \mathbf{c}^{\mathsf{T}}\,\mathbf{x} & \text{(objective function)} \\ \text{subject to} & \mathbf{A}\,\mathbf{x} = \mathbf{b} & \text{(equality constraints)} \\ & \mathbf{D}\,\mathbf{x} \geq \mathbf{d} & \text{(inequality constraints)} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} & \text{(lower and upper bounds)} \end{array}$$

We can convert any $\leq$ constraint to $\geq$ by adding auxiliary variables.

The Wolfram Language has built-in functions to solve linear optimization problems with real variables. They include:

- `LinearProgramming`
- `FindMinimum`
- `FindMaximum`
- `NMinimize`
- `NMaximize`
- `Minimize`
- `Maximize`

For large-scale problems, the most flexible and efficient of these is `LinearProgramming`. The others are appropriate for solving linear programs written in terms of equations.

□ Example Using `FindMinimum`

We consider an example very similar to the one presented in [2]. Assume that we have obligations to pay cash flows in the next eight years as shown in the following table. The first row shows years and the second row shows the amount of cash to be paid each year.

Year	1	2	3	4	5	6	7	8
Obligations	100	200	100	200	800	1200	400	1000

Also assume that we have five government bonds available to invest with the cash flows from the obligations and current prices given as follows.

Year	Bond 1	Bond 2	Bond 3	Bond 4	Bond 5
1	2.50	5.00	3.00	4.00	3.50
2	2.50	5.00	3.00	4.00	3.50
3	2.50	5.00	3.00	4.00	3.50
4	2.50	5.00	3.00	4.00	3.50
5	102.50	5.00	3.00	4.00	3.50
6	–	105.00	3.00	4.00	3.50
7	–	–	103.00	4.00	3.50
8	–	–	–	104.00	103.50
Price	102.40	110.80	96.94	114.70	96.63

To make a portfolio that minimizes the overall cost and still meets all the expected future cash payments, we can decide on how to allocate the funds by converting this problem into a linear program and solving it. Assume that x_i is the number of bonds i to purchase.

We define four variables for convenience.

```
In[2]:= m = {
          {2.5, 5, 3, 4, 3.5},
          {2.5, 5, 3, 4, 3.5},
          {2.5, 5, 3, 4, 3.5},
          {2.5, 5, 3, 4, 3.5},
          {102.5, 5, 3, 4, 3.5},
          {0, 105, 3, 4, 3.5},
          {0, 0, 103, 4, 3.5},
          {0, 0, 0, 104, 103.5}
        };
```

```
In[3]:= c = {102.36, 110.83, 96.94, 114.65, 96.63};
```

```
In[11]:= xvector = x# & /@ Range@5
```

```
Out[11]= {x1, x2, x3, x4, x5}
```

```
In[12]:= p = 100 {1, 2, 1, 2, 8, 12, 4, 10}

Out[12]= {100, 200, 100, 200, 800, 1200, 400, 1000}
```

Then the problem can be stated as follows.

Minimize	$102.36\,x_1 + 110.83\,x_2 + 96.94\,x_3 + 114.65\,x_4 + 96.63\,x_5$
subject to	$2.5\,x_1 + 5\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 100$ $2.5\,x_1 + 5\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 200$ $2.5\,x_1 + 5\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 100$ $2.5\,x_1 + 5\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 200$ $102.5\,x_1 + 5\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 800$ $105\,x_2 + 3\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 1200$ $103\,x_3 + 4\,x_4 + 3.5\,x_5 \geq 400$ $104\,x_4 + 103.5\,x_5 \geq 1000$
and	$0 \leq x_1, x_2, \ldots, x_5$

This solves the problem.

```
In[16]:= FindMinimum[
          {
           c.xvector,
           And @@ Thread[GreaterEqual[m.xvector, p]] &&
            Thread[0 <= xvector]
          },
          xvector, Method → "LinearProgramming"]

Out[16]= {5007.15,
          {x1 → 6., x2 → 28.1036, x3 → 3.55518, x4 → 0., x5 → 9.66184}}
```

Same Example Using `LinearProgramming`

The problem can also be solved using the `LinearProgramming` function. One of its most useful forms is `LinearProgramming[c, m, b, bounds]`, where:

- `c` is the vector of coefficients of the objective function.
- `m` is the matrix of coefficients in the constraints.
- `b` is a two-column matrix representing the constants on the right side of the constraints and the direction of inequality.

1	≥
0	=
−1	≤

- `bounds` is a two-column matrix of lower and upper bounds for the decision variables.

The variables `c` and `m` were defined before; here they are displayed in matrix form.

```
In[16]:= c // MatrixForm

Out[16]//MatrixForm=
 ( 102.36 )
 ( 110.83 )
 (  96.94 )
 ( 114.65 )
 (  96.63 )
```

```
In[17]:= m // MatrixForm

Out[17]//MatrixForm=
 (   2.5    5    3    4    3.5  )
 (   2.5    5    3    4    3.5  )
 (   2.5    5    3    4    3.5  )
 (   2.5    5    3    4    3.5  )
 ( 102.5    5    3    4    3.5  )
 (   0    105    3    4    3.5  )
 (   0      0  103    4    3.5  )
 (   0      0    0  104  103.5  )
```

We define `b` from `p` and `bounds`.

```
(b = {#, 1} & /@ p) // MatrixForm

Out[18]//MatrixForm=
 (  100  1 )
 (  200  1 )
 (  100  1 )
 (  200  1 )
 (  800  1 )
 ( 1200  1 )
 (  400  1 )
 ( 1000  1 )
```

```
In[19]:= (bounds = Table[{0, Infinity}, {5}]) // MatrixForm

Out[19]//MatrixForm=
 ( 0  ∞ )
 ( 0  ∞ )
 ( 0  ∞ )
 ( 0  ∞ )
 ( 0  ∞ )
```

This solves the problem.

```
In[20]:= Module[
          {solution},
          solution = LinearProgramming[c, m, b, bounds];
          {c.solution, solution}
         ]

Out[20]= {5007.15, {6., 28.1036, 3.55518, 0., 9.66184}}
```

Mathematica’s built-in capabilities to solve linear programming problems can be used in a wide variety of financial problems. We refer interested readers to [2].

3. Mean-CVaR Optimization

In this section, we solve the mean-CVaR portfolio problem, which was proposed by Rockafellar and Uryasev [3]. CVaR optimization does not depend on any assumption of how returns are distributed, but it works for the normal distribution. We summarize the linear programming formulation of the CVaR problem.

Following [3], $P(\mathbf{r})$ is the joint density function of the underlying asset returns $\mathbf{r} = (r_1, r_2, \ldots, r_n)^{\mathrm{T}}$, where r_i is the return on asset i; $f(\mathbf{w}, \mathbf{r})$ is the loss associated with the decision vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$, where w_i is the proportion of money invested in asset i; and α is the $(1-\beta)$-quantile of the loss distribution.

Then the CVaR can be defined as:

$$\min_{\mathbf{w},\alpha}\left(\alpha + \frac{1}{1-\beta}\,\mathrm{E}(f(\mathbf{w},\mathbf{r})-\alpha)^+\right), \text{ where E is expectation and } (x)^+ = \begin{cases} x & x>0 \\ 0 & x \le 0 \end{cases}$$

For a sample size q, the CVaR is approximately:

$$\min_{\mathbf{w},\alpha}\left(\alpha + \frac{1}{q(1-\beta)} \sum_{k=1}^{q} (-\mathbf{w}^{\mathrm{T}} r_k - \alpha)^+\right).$$

The problem can be restated as a linear optimization problem by introducing auxiliary variables, one for each observation in the sample:

Find $\min_{\mathbf{w},\mathbf{z},\alpha}\left(\alpha + \frac{1}{q(1-\beta)} \sum_{k=1}^{q} z_k\right)$,

subject to $\mathbf{w}^{\mathrm{T}} r_k + \alpha + z_k \ge 0$ and $z_k \ge 0$ for $k = 1, \ldots, q$.

As a linear program, the problem is:

Find $\min_{\mathbf{w},\mathbf{z},\alpha}\left(\alpha + \frac{1}{q(1-\beta)} \sum_{k=1}^{q} z_k\right)$,

subject to

$\mathbf{w}^{\mathrm{T}} r_k + \alpha + z_k \ge 0$ and $z_k \ge 0$ for $k = 1, \ldots, q$,

$\sum_{i=1}^{n} w_i = 1$ and $w_i \ge 0$ for $i = 1, \ldots, n$,

$-\mathbf{w}^{\mathrm{T}} \boldsymbol{\mu} = -R^*$,

where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$, $\mu_i = \mathrm{E}(r_i)$ and R^* is the target optimal portfolio return.

We define the function `mCVaROptimization` to estimate the optimal weights that minimize CVaR. It takes three arguments:

• the returns matrix (`returns`)

• the target portfolio return (`targetRMeanReturn`)

• the confidence level (β), between 0.9 and 0.99

```
In[21]:= mCVaROptimization[returns_List, targetRMeanReturn_, β_] :=
          Module[
           {m, n, vector, A, Aeq, b, bounds, solution, weights},
           {m, n} = Dimensions[returns];
           vector =
            Flatten[{1, ConstantArray[0, n],
              1 / (m (1 - β)) ConstantArray[1, m]}];
           A =
            -Flatten /@
              Transpose@{ConstantArray[1, m], returns,
                IdentityMatrix@m};
           Aeq =
            (PadRight[#, 1 + m + n] & /@
              {Prepend[-Mean@returns, 0], {0, 1, 1, 1}});
           b = ArrayFlatten@{
              {Transpose@{ConstantArray[0, m], ConstantArray[-1, m]}},
              {Transpose[{{-targetRMeanReturn, 1}, {0, 0}}]}
             };
           bounds = Transpose@{
              ConstantArray[0, 1 + m + n],
              Flatten[{Infinity, ConstantArray[1, n],
                ConstantArray[Infinity, m]}]
             };
          solution = LinearProgramming[vector,
             ArrayFlatten@{{A}, {Aeq}}, b, bounds,
             Method → "Simplex"];
           weights = solution[[2 ;; n + 1]];
           Text@Column[{
              "Long-Only Portfolio\n",
              Row[{ "Weights = ", solution[[2 ;; n + 1]]}],
              Row[{ "Portfolio Mean = ",
                NumberForm[Mean[returns].weights, {4, 4}]}],
              Row[{"Portfolio Variance = ",
                NumberForm[weights.Covariance[returns].weights,
                 {4, 4}]}],
              Row[{"Value-at-Risk = ",
                NumberForm[solution[[1]], {4, 4}]}],
             Row[{"Conditional Value-at-Risk = ",
                NumberForm[solution.vector]}]
             }]
          ]
```

Example

This downloads monthly returns of three stocks over the period May 1, 2000, to May 30, 2019, and computes the CVaR-based optimal weights and associated values given the target portfolio return and confidence level.

```
In[22]:= returns3 = getReturnsData[{"AAPL", "BA", "WMT"},
           {2000, 5, 1}, {2019, 5, 30}, "Month"];
```

```
In[23]:= mCVaROptimization[returns3, 0.0052, 0.95]

         Long-Only Portfolio

         Weights = {0., 0.142715, 0.857285}
Out[23]= Portfolio Mean = 0.0052
         Portfolio Variance = 0.0023
         Value-at-Risk = 0.0731
         Conditional Value-at-Risk = 0.100818
```

The function computes optimal weights for a long-only portfolio. It can easily be modified to account for short-selling.

4. Introductory Time-Series Analysis and Forecasting

Data collected over time is common in finance. Mathematica has many built-in functions to model the stochastic nature of financial time series and to forecast the future value of a series. This section gives examples of functions that are useful for model specification, estimation, diagnostics and forecasting of univariate time-series data.

Constructing and Visualizing Time Series

The first step in any exploratory analysis is to construct and plot time series. You can use the built-in functions `TimeSeries` and `TemporalData` to construct financial time series as pairs $\{t_i, v_i\}$. There are two formats for each:

- `TimeSeries[{{`t_1, v_1`}, {`t_2, v_2`}, …}]`
- `TimeSeries[{{`t_1, t_2, …`}, {`v_1, v_2, …`}}]`
- `TemporalData[{{`t_1, v_1`}, {`t_2, v_2`}, …}]`
- `TemporalData[{{`t_1, t_2, …`}, {`v_1, v_2, …`}}]`

Time-series data can be manipulated using many built-in functions; see the documentation.

```
In[24]:= ? TimeSeries*
```

```
Out[24]=
System`
TimeSeries                 TimeSeriesMap          TimeSeriesRescale
TimeSeriesAggregate        TimeSeriesMapThread    TimeSeriesShift
TimeSeriesForecast         TimeSeriesModel        TimeSeriesThread
TimeSeriesInsert           TimeSeriesModelFit     TimeSeriesWindow
TimeSeriesInvertibility    TimeSeriesResample
```

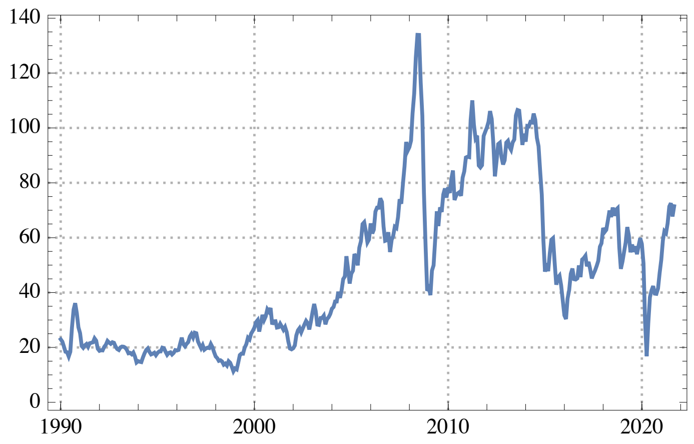
Once we create a time series, we can use functions like `ListLinePlot` or `DateListPlot` to visualize it.

We illustrate the historical global price of WTI Crude (POILWTIUSDM). First, we download the historical price of WTI crude oil since January, 1990, and then make a plot. Use the API key 207071a5f2e90e7816259d3c32c1ab81 if needed.

```
In[17]:= timeSeriesData =
          TimeSeriesWindow[
           ServiceConnect["FederalReserveEconomicData"][
            "SeriesData", "ID" → {"POILWTIUSDM"}],
           {{1990, 01, 01}, Today}];


         DateListPlot[timeSeriesData, PlotTheme → "Detailed"]
```

Out[18]=



▫ Model Fitting and Its Diagnostics

The built-in function `TimeSeriesModelFit` supports both linear and nonlinear time-series models. It fits an automatically selected parametric model to a time series. We can customize the model fit specification by changing its options. The currently supported families of models are:

- autoregressive (AR)
- moving-average (MA)
- autoregressive moving-average (ARMA)
- autoregressive integrated moving-average (ARIMA)
- seasonal autoregressive moving-average (SARMA)
- seasonal integrated autoregressive moving-average (SARIMA)
- autoregressive conditionally heteroscedastic (ARCH)
- generalized autoregressive conditionally heteroscedastic (GARCH)

You can find descriptions of these models in any time-series books, including Tsay [4].

Although `TimeSeriesModelFit` selects the model automatically, there are many built-in functions for choosing appropriate values for the parameterizations for a given family and checking the appropriateness of the fitted models.

Next, we are going to show the use of some tools for model specification and checking the adequacy of fitted models.

Use `AutocorrelationTest` to test whether the data is autocorrelated. (Use `PartialCorrelationFunction` to estimate the partial correlation function of the data.)

```
In[19]:= AutocorrelationTest[timeSeriesData, Automatic,
           "HypothesisTestData"]["TestDataTable", All]

                   | Statistic  P-Value
Out[19]= Box-Pierce | 1922.59    0.
         Ljung-Box  | 1949.75    0.
```

Use `UnitRootTest` to test whether data comes from an autoregressive time-series process with unit root.

```
In[20]:= UnitRootTest[timeSeriesData, Automatic,
           "HypothesisTestData"]["TestDataTable", All]

                           | Statistic   P-Value
         Dickey-Fuller F   | -0.627312   0.544584
Out[20]= Dickey-Fuller T   | -0.379336   0.547254
         Phillips-Perron F | -1.60259    0.379706
         Phillips-Perron T | -0.740976   0.394823
```

A number of other tools are available for model specification:

• Akaike information criterion (AIC)

• Finite sample corrected AIC (AICc)

• Bayesian information criterion (BIC)

• Schwartz–Bayes information criterion (SBC)

To choose the appropriate model for the `timeSeriesData`, we can do the following. (For information on `ARIMAProcess`, see the Mathematica help system.)

```
In[21]:= TimeSeriesModelFit[timeSeriesData,
             Method → {Automatic, "SelectionCriterion" → #}][
           "CandidateSelectionTable"] & /@
          {"AIC", "AICc", "BIC", "SBC"}
```

```
           Candidate                AIC            Candidate                AICc
        1  ARIMAProcess[1, 1, 0]  1127.81       1  ARIMAProcess[1, 1, 0]  1129.92
        2  ARIMAProcess[2, 1, 0]    1129.78     2  ARIMAProcess[2, 1, 0]    1131.94
        3  ARIMAProcess[1, 1, 1]    1130.25     3  ARIMAProcess[1, 1, 1]    1132.41
        4  ARIMAProcess[2, 1, 1]    1132.26     4  ARIMAProcess[2, 1, 1]    1134.48
Out[21]= {5  ARIMAProcess[0, 1, 1]    1138.59  , 5  ARIMAProcess[0, 1, 1]    1140.7   ,
        6  ARIMAProcess[2, 2, 2]    1143.46     6  ARIMAProcess[2, 2, 2]    1145.76
        7  ARIMAProcess[2, 2, 3]    1144.02     7  ARIMAProcess[2, 2, 3]    1146.41
        8  ARIMAProcess[3, 2, 2]    1144.89     8  ARIMAProcess[2, 2, 1]    1147.13
        9  ARIMAProcess[2, 2, 1]    1144.91     9  ARIMAProcess[3, 2, 2]    1147.28
        10 ARIMAProcess[3, 2, 3]    1144.92     10 ARIMAProcess[3, 2, 3]    1147.4

           Candidate                BIC            Candidate                SBC
        1  ARIMAProcess[1, 1, 0]  1150.56       1  ARIMAProcess[1, 1, 0]  1139.64
        2  ARIMAProcess[2, 1, 0]    1158.96     2  ARIMAProcess[2, 1, 0]    1145.55
        3  ARIMAProcess[1, 1, 1]    1159.42     3  ARIMAProcess[1, 1, 1]    1146.02
        4  ARIMAProcess[0, 1, 1]    1161.26     4  ARIMAProcess[0, 1, 1]    1150.42
        5  ARIMAProcess[2, 1, 1]    1167.6   ,  5  ARIMAProcess[2, 1, 1]    1151.97  }
        6  ARIMAProcess[1, 2, 1]    1176.11     6  ARIMAProcess[1, 2, 1]    1162.89
        7  ARIMAProcess[2, 2, 1]    1180.08     7  ARIMAProcess[2, 2, 1]    1164.62
        8  ARIMAProcess[1, 2, 2]    1184.05     8  ARIMAProcess[2, 2, 2]    1167.11
        9  ARIMAProcess[2, 2, 2]    1184.62     9  ARIMAProcess[1, 2, 2]    1168.64
        10 ARMAProcess[1, 2]        1193.86     10 ARMAProcess[1, 2]        1178.59
```

Once the model has been specified, you can estimate its parameters with `TimeSeriesModelFit` and you can assess its goodness of fit through analysis of residuals.

This estimates the parameters of the model for `timeSeriesData`.

```
In[22]:= timeseriesmodel = TimeSeriesModelFit[timeSeriesData,
            {"ARIMA", {1, 1, 0}}];
```

```
In[24]:= timeseriesmodel["ParameterTable"]
```

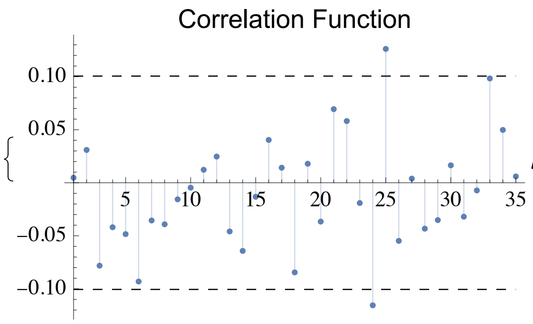
Out[24]=

	Estimate	Standard Error	t-Statistic	P-Value
a_1	0.384374	0.0472958	8.12702	3.10992×10^{-15}

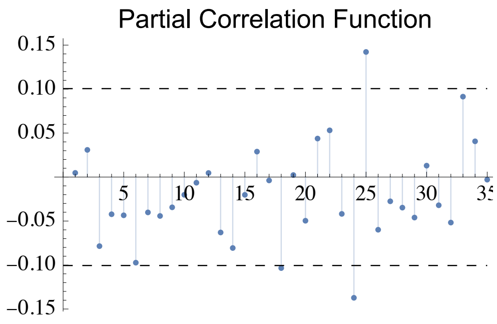
Use `AFCPlot`, `PFCPlot` or `LjungBoxPlot` of the residuals to assess the whiteness of the model residuals.

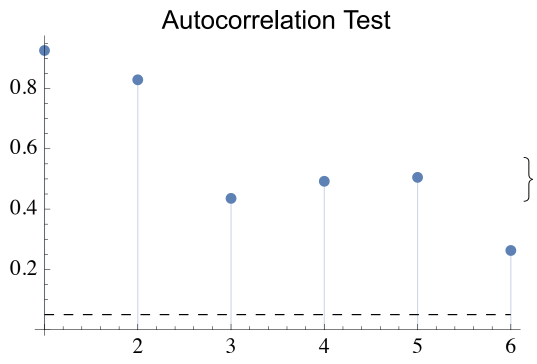
```
In[25]:= timeseriesmodel /@ {"ACFPlot", "PACFPlot", "LjungBoxPlot"}
```

Out[25]= {



,





}

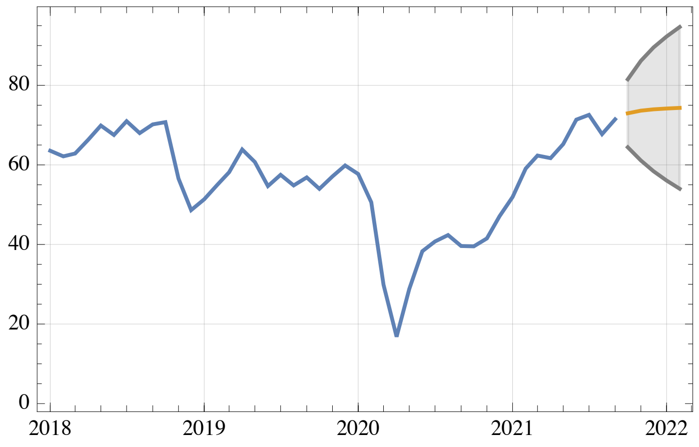
◻ Forecasting

A primary objective of building a time-series model is to forecast its future values. Prediction limits are important to assess the potential accuracy of the forecast. We can use `TimeSeriesForecast` to forecast unobserved future values. The function takes the methods `"AR"`, `"Covariance"` and `"Kalman"`. We use mean-squared errors to get the precision of our prediction.

This calculates and plots the forecast for the next 5 months of the series within 95% confidence limits.

```
In[26]:= Module[
  {forecast, error, quantile, uband, lband},
  forecast = TimeSeriesForecast[timeseriesmodel, {1, 5}];
  error = Sqrt@forecast@"MeanSquaredErrors";
  quantile = Quantile[NormalDistribution[],
    1 - 1 / 2 × (1 - .95)];
  uband = TimeSeriesThread[{1, quantile}.# &,
    {forecast, error}];
  lband = TimeSeriesThread[{1, -quantile}.# &,
    {forecast, error}];
  DateListPlot[
   {
    TimeSeriesWindow[timeSeriesData,
     {{2018, 1, 1}, Last@timeSeriesData["Dates"]}],
    forecast,
    uband,
    lband
   },
   PlotStyle → {Automatic, Automatic, Gray, Gray},
   GridLines → {Automatic, Automatic}, Filling → {3 → {4}}]
 ]
```

Out[26]=



ARCH and GARCH Models

The autoregressive conditional heteroskedasticity (ARCH) model and the generalized ARCH model (GARCH) are often used to get a volatility forecast of a time series. You can also use the built-in function `TimeSeriesModelFit` to estimate parameters of these volatility models. Most volatility models are based on using returns that are obtained after subtracting unconditional mean returns. For our illustration, we de-mean our returns data. The parameters of the model are typically estimated with maximum likelihood.

To estimate a GARCH model for WTI Crude data, we find the de-mean data and estimate the model as follows.

```
In[34]:= returns4 = 100 Differences[Log[timeSeriesData]]["Values"];
         TimeSeriesModelFit[returns4 - Mean[returns4],
           {"GARCH", {1, 1}}]["ParameterTable"]
```

Out[35]=

	Estimate	Standard Error	t-Statistic	P-Value
α_1	0.673461	0.0563082	11.9603	1.67988×10^{-28}
β_1	0.	0.0761721	0.	0.5

You can use the built-in functions `ARProcess`, `MAProcess`, `ARMAProcess`, `SARIMAProcess`, `ARCHProcess` and `GARCHProcess` to simulate time-series data and for risk management. `EstimatedProcess` is another important function to estimate the parameters of a process given its data and model specification.

5. Simulating Data and Financial Application

A financial model consisting of fixed relations and variables may not be accurate because most relationships between financial variables are random. Therefore, we must be able to incorporate stochasticity. Monte Carlo simulation is widely used to represent the true features of random modeling. Simulation modeling is a computer-based modeling technique that mimics a real-life situation and helps to incorporate uncertainties in input variables. Such techniques give a distribution of a forecast variable, not just a single value. Therefore, it is very useful when we are uncertain about future outcomes. In this section, we give examples for simulating data using Mathematica and show an application in capital budgeting.

`RandomVariate` is a powerful function to get data from built-in statistical distributions, including those that are:

- continuous or discrete
- univariate or multivariate
- parametric or derived
- defined by data

Example in Project Risk Analysis

Consider a situation in which you have to evaluate an investment by forecasting the present value of its future cash flows.

We define a function `presentValueOfCashFlows` to compute the present value of future cash flows. Here are its 11 arguments and values for an example. Assume that the revenue and terminal value both follow a triangular distribution and that gross margin follows a uniform distribution.

description	argument	example
minimum expected revenue	`minExpectedRevenue`	$100
maximum expected revenue	`maxExpectedRevenue`	$400
most likely expected revenue	`mostLikelyExpectedRevenue`	$200
minimum terminal value	`minTerminalValue`	$0
maximum terminal value	`maxTerminalValue`	$500
most likely terminal value	`mostLikelyTerminalValue`	$300
minimum gross profit margin	`minGrossProfitMargin`	15%
maximum gross profit margin	`maxGrossProfitMargin`	35%
life of the project	`projectLife`	5 years
cost of capital	`interestRate`	11%
tax rate	`taxRate`	21%

```
In[27]:= presentValueOfCashFlows[minExpectedRevenue_,
   maxExpectedRevenue_, mostLikelyExpectedRevenue_,
   minTerminalValue_, maxTerminalValue_,
   mostLikelyTerminalValue_, minGrossProfitMagrin_,
   maxGrossProfitMagrin_, projectLife_, interestRate_,
   taxRate_] := Module[
   {revenue, terminalValue, grossMargin, cashFlows},
   revenue = RandomVariate[
     TriangularDistribution[
      {minExpectedRevenue, maxExpectedRevenue},
      mostLikelyExpectedRevenue], projectLife];
   terminalValue =
    RandomVariate[TriangularDistribution[
      {minTerminalValue, maxTerminalValue},
      mostLikelyTerminalValue], 1];
   grossMargin =
    RandomVariate[UniformDistribution[
      {minGrossProfitMagrin, maxGrossProfitMagrin}],
     projectLife];
   cashFlows =
    (1 - taxRate) (revenue - (1 - grossMargin) revenue);
   TimeValue[Cashflow@cashFlows, interestRate, 0] +
              terminalValue
    ----------------------------------------
    (1 + interestRate)^Length@cashFlows ]
```

For the given values, we simulate the data 10,000 times. (This takes a minute or so.)

```
In[28]:= presentValueOfCashFlows =
           Table[presentValueOfCashFlows[100, 400, 200, 0, 500,
              300, 0.15, 0.15, 5, 0.11, 0.21], 10 000] // Flatten;
```

We can summarize the data as follows.

```
In[34]:= Text@TableForm[basicStats@presentValueOfCashFlows,
           TableHeadings → {None, {"Stats", "Value"}}]
```

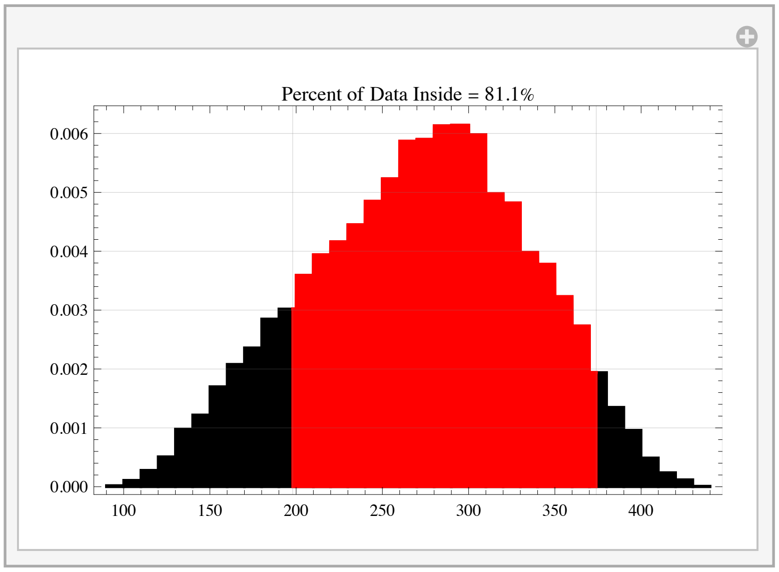
Out[34]=

Stats	Value
N	10 000
Mean	270.944
Minimum Value	91.2346
Maximum Value	434.381
25th Percentile Value	225.688
50th Percentile Value	274.813
75th Percentile Value	316.916
Standard Deviation	63.3651
Variance	4015.13
Skewness	−0.168023
Kurtosis	2.43941

Here is a histogram of the distribution of the present value of the cash flows. Drag the handles (thin vertical lines) on either side of the red region to see the percentage of cash flows that falls within the range of data.

```
In[37]:= visualHistogram@presentValueOfCashFlows
```

Out[37]=



This is just one example of simulation. We can define similar functions to compute the value of a firm using different valuation models:

• discounted cash flows

• residual operating income valuation

• abnormal growth in earnings valuation

There are many other areas of finance where simulation can be used.

6. Bootstrapping and Financial Application

We can apply the bootstrap approach in several contexts in finance. When the data is limited and the true distribution of the population is unknown, we can generate the sampling distribution of a statistic by generating many new samples from the original data and use the empirical distribution for statistical inference. This is called bootstrapping. Performing a bootstrap analysis in Mathematica is very straightforward using the `RandomChoice` or `RandomSample` functions with or without replacement.

Performing bootstrap analysis entails two steps. First, we define a function that computes the statistic of interest. Second, we estimate the statistic of interest by repeatedly sampling observations (usually 10,000 times or more) from the original sample with replacement. Then we can use the distribution of sample statistics to infer an appropriate decision.

In this section, we illustrate the use of bootstrapping through two examples.

Example 1

We consider estimating the distribution of an equally weighted portfolio's conditional value-at-risk (CVaR) using weekly returns of Walmart stock (WMT) and Procter & Gamble (PG) over the period January 1, 1982, to March 30, 2019.

Step 1

We get the historical weekly returns data.

```
In[38]:= returns6 = getReturnsData[{"WMT", "PG"}, {1982, 01, 01},
            {2019, 04, 30}, "Week"];
```

Step 2

We define two functions.

The function `conditionalVaR` computes the conditional value-at-risk given the returns data.

```
In[39]:= conditionalVaR[list_List, prob_ : 0.05] :=
          Mean[Sort[list][[ ;; Floor[prob Length@list] ]] ]
```

The function `bootstrapCVaR` returns a distribution of conditional value-at-risk measures given the size of the sample.

```
In[41]:= bootstrapCVaR[returns_, size_] := Module[
           {n = Last@Dimensions@returns6},
           conditionalVaR[RandomSample[returns6, size].
             Table[1 / n, n]]
          ]
```

We use these functions to get distributions of CVaR with 10,000 observations.

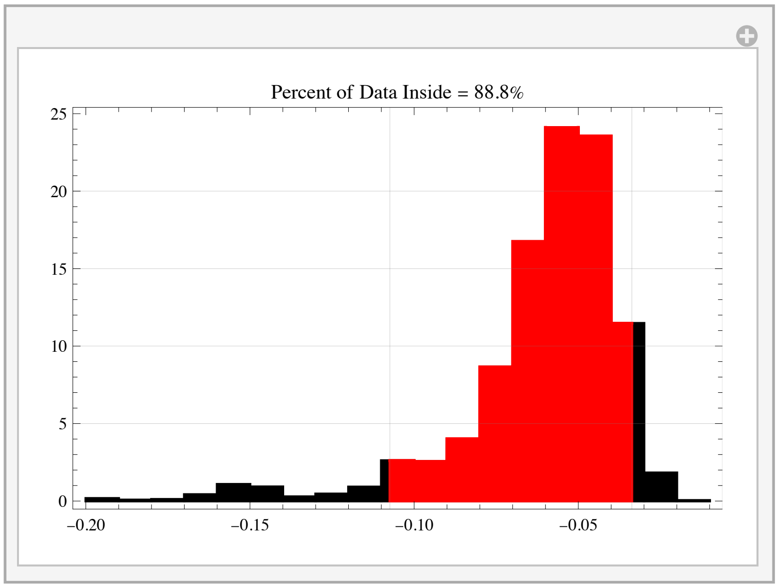
```
In[42]:= cvars = Table[bootstrapCVaR[returns6, 50], 10 000];
```

We summarize the data.

```
In[43]:= Text@Column[
           {
            TableForm[basicStats@cvars],
            visualHistogram[cvars]
           },
           Center]
```

```
N                         10 000
Mean                      −0.0606506
Minimum Value             −0.199207
Maximum Value             −0.0163968
25th Percentile Value     −0.0679713
50th Percentile Value     −0.0550866
75th Percentile Value     −0.0453298
Standard Deviation        0.0246071
Variance                  0.00060551
Skewness                  −2.06658
Kurtosis                  8.75516
```

Out[43]=



Example 2

The next example focuses on retirement planning using the bootstrapping concept. Assume that we want to calculate the terminal value of the following retirement portfolio. The savings are invested equally in two market indices: the S&P 500 Index and the NASDAQ 100 Index. Assume that future returns would be a random draw from past returns. The initial deposit is $1000. Monthly saving for the next 20 years is $1,500. The number of retirement years is 15. During the 15 years of retirement, $2000 will be withdrawn monthly. Starting at the 10th year, $30,000 will be withdrawn annually for three years.

Define the function `futureValueofRetirementPortfolio`, which calculates the terminal value of a retirement portfolio. It takes 10 arguments:

- returns of the stocks in which money is equally invested (`returns`)
- initial portfolio value (`initialValue`), which must be positive
- `serviceYears`–years in service
- `saving`–periodic saving
- `frequency`–frequency of contribution per year; coded as 12 for monthly data and 1 for annual data
- `retirementYears`–number of retirement years
- `income`–periodic income during retirement years
- `bigWithdrawal`–big annual withdraw amount during planning period
- `bigWithdrawalYear`–big withdrawal starting year; annual and in successive years with no gap
- `bigWithdrawalNumber`–number of annual big withdrawals

```
In[44]:= futureValueofRetirementPortfolio[returns_List,
           initialValue_, serviceYears_, saving_, frequency_,
           retirementYears_, income_, bigAnnualwithdraw_,
           bigWithdrawalPeriod_, bigWithdrawalNumber_] := Module[
           {cashFlows, numberOfAssets, weights, randomReturns},
           cashFlows = Flatten[{
              initialValue, Table[saving, frequency serviceYears],
              Table[-income, frequency retirementYears]}];
           cashFlows[[
             Table[bigWithdrawalPeriod frequency + frequency i,
              {i, bigWithdrawalNumber}]]] += -bigAnnualwithdraw;
           numberOfAssets = Last@Dimensions@returns;
           weights = Table[1 / numberOfAssets, numberOfAssets] // N;
           randomReturns =
            Flatten@Table[RandomChoice[returns, 1].weights,
              Length@cashFlows];
           TimeValue[Cashflow@cashFlows, randomReturns,
            (serviceYears + retirementYears) frequency]
          ]
```

Using the values given, this computes the terminal value.

```
In[45]:= monthlyreturns = getReturnsData[{"WMT", "PG"},
            {1982, 01, 01}, {2019, 04, 30}, "Month"];
```

```
In[46]:= futureValueofRetirementPortfolio[monthlyreturns, 1000,
          20, 1500, 12, 15, 2000, 30000, 10, 3]
```

```
Out[46]= 1.02382 × 10^7
```

7. Mean-Variance Optimization under Uncertainty

Optimization under uncertainty (or robust optimization) is another approach that helps to get solutions that are good for most realizations of data. Many financial problems fit into this framework; for example, the mean-variance portfolio optimization problem. Here uncertainty may arise due to many factors:

• uncertainty in the mean vector

• fluctuations in the covariation matrix

• variability of risk in the market over time

• imprecise model approximation

Feasibility depends on both the decision variable **w** and the uncertain vector **r**; uncertainty can be introduced in the expected value, the variance or both.

This section considers a mean-variance problem that allows some degree of variation in returns and covariances.

The standard form of a mean-variance problem is expressed in terms of the information about the expected returns and the covariance structure of the returns. For given asset returns $\mathbf{r} = (r_1, r_2, \ldots r_n)^{\mathrm{T}}$, where r_i is the return on asset i; and the decision vector $\mathbf{w} = (w_1, w, \ldots w_n)^{\mathrm{T}}$, where w_i is the proportion of money invested in asset i; the mean variance problem is written as:

$$\underset{\mathbf{w}}{\text{Minimize}}\ \frac{1}{2}\,\mathbf{w}^{\mathrm{T}}\,\mathbf{V}\,\mathbf{w} - \theta\,\mathbf{w}^{\mathrm{T}}\,\mathbf{M}$$

subject to $\mathbf{1}^{\mathrm{T}}\,\mathbf{w} = 1$.

Here $\mathbf{V} = \begin{pmatrix} \sigma_{11} & \sigma_{12} & \ldots & \sigma_{1\,n} \\ \sigma_{21} & \sigma_{22} & \ldots & \sigma_{2\,n} \\ \vdots & \vdots & & \vdots \\ \sigma_{n\,1} & \sigma_{n\,2} & \ldots & \sigma_{n\,n}^2 \end{pmatrix}$, where $\sigma_{i,j}$ is the covariance between securities i and j;

$\theta \geq 0$ is the risk coefficient; $\mathbf{M} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{pmatrix}$, where μ_i is the average return on security i; and **1** is an $n \times 1$ column vector of ones, $\mathbf{1} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$.

It is important to find the portfolio with the maximum Sharpe ratio, which can be obtained by solving the following problem; assume that $\theta = 1$ and denote the risk-free rate by r_{rf}:

$$\underset{\mathbf{w}}{\text{Maximize}}\ \mathbf{w}^{\mathrm{T}} (\mathbf{M} - r_{\text{rf}}\, \mathbf{1}) - \frac{1}{2}\, \mathbf{w}^{\mathrm{T}}\, \mathbf{V}\, \mathbf{w}$$

subject to $\mathbf{1}^{\mathrm{T}}\, \mathbf{w} = 1$.

The analytical solution to this basic problem for portfolio optimal portfolio weights $\mathbf{w}^*$ is

$$\mathbf{w}^* = \frac{\mathbf{V}^{-1} (\mathbf{M} - r_{\text{rf}}\, \mathbf{1})}{\mathbf{1}^{\mathrm{T}}\, \mathbf{V}^{-1} (\mathbf{M} - r_{\text{rf}}\, \mathbf{1})}.$$

We define the function `weightsVector` to compute $\mathbf{w}$. It takes the arguments returns data (`returns`) and risk-free rate (`riskFree`) and returns an optimal weight vector.

```
In[47]:= weightsVector[returns_, riskFree_] := Module[{n, x},
           n = Last@Dimensions@returns;
           x = Inverse[Covariance@returns].
              (Mean@returns - riskFree);
           x / ConstantArray[1, n].x]
```

One way to get a robust solution to the mean-variance problem is to sample data in several scenarios to estimate parameters. Assuming that the returns have multivariate normal distribution, we define a function `robustTangencyPortfolios` to compute tangency portfolios with simulated data. The function takes four arguments:

• `returns`–returns data

• `riskFree`–risk-free rate

• `sampleSize`–sample size

• n–number of iterations

It returns a distribution of optimal Sharpe ratios.

```
In[48]:= robustTangencyPortfolios[returns_, riskFree_, sampleSize_,
           n_] := Module[{mean, variance, distribution, weights},
           mean = Mean@returns;
           variance = Covariance@returns;
           weights =
            Table[weightsVector[
               RandomVariate[MultinormalDistribution[mean, variance],
                sampleSize], riskFree], n];
         Table[(weights⟦i⟧.mean - riskFree) /
             Sqrt[weights⟦i⟧.variance.weights⟦i⟧],
            {i, Length@weights}]
          ]
```

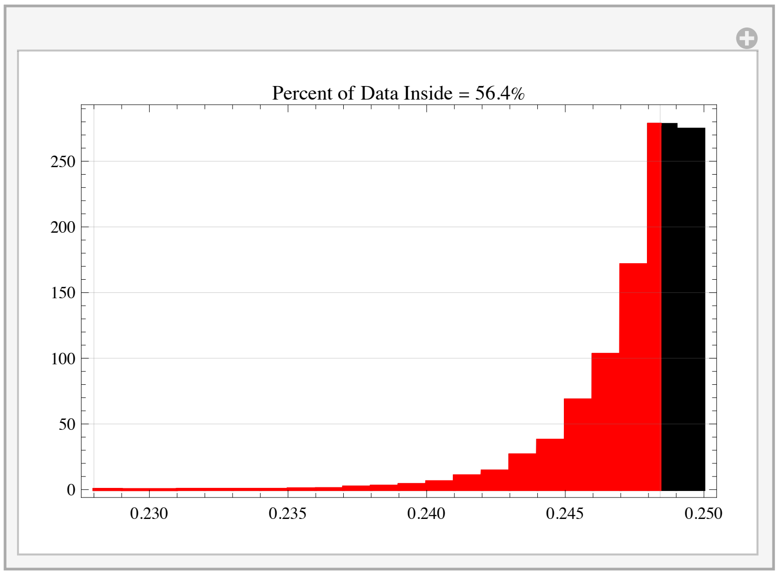
We apply those two functions to simulated data and get the distribution of portfolio means that maximize the Sharpe ratio.

We download historical monthly returns as before, compute the distribution of maximum Sharpe ratios and draw a histogram.

```
In[49]:= returns7a = getReturnsData[{"BA", "AAPL", "WMT"},
            {1982, 01, 01}, {2019, 04, 30}, "Month"];
```

```
In[50]:= visualHistogram[
          SharpeRatios = robustTangencyPortfolios[returns7a,
            0.02 / 12, 2000, 10 000]]
```

Out[50]=



Another way to get robustness in the parameter estimate is to introduce some kinds of uncertainty; interval uncertainty sets and ellipsoidal uncertainty sets are the most commonly used. Following Kim, Kim and Fabozzi [5] (using **V** and **M** instead of their $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$), the interval uncertainty set for the expected returns can be defined as:

$$\left\{\mathbf{M} : \left|M_i - \hat{M}_i\right| \leq \delta_i \text{ for } i = 1, 2, \ldots, n\right\},$$

where $\mathbf{M} = (M_1, \ldots, M_n)$, the variable $\hat{M}_i$ is an estimate of the expected return and δ_i is a constant used to control the expected returns of stock i.

The mean-variance problem with box uncertainty can be written as:

$$\underset{\mathbf{w}}{\text{Minimize}}\ \underset{\mathbf{M}}{\text{Maximize}}\ \frac{1}{2}\mathbf{w}^{\mathsf{T}}\,\mathbf{V}\,\mathbf{w} - \theta\,\mathbf{w}^{\mathsf{T}}\,\mathbf{M}$$

subject to $\mathbf{w}^{\mathsf{T}}\,\mathbf{1} = 1$ and where $\mathbf{M}$ is such that $\left|M_i - \hat{M}_i\right| \leq \delta_i$ for $i = 1, 2, \ldots, n$.

This objective function can be modified as follows (see [5] for the derivation and explanation of the notation $\mathbf{V_T}$ and $\mathbf{M_T}$):

$$\underset{\mathbf{x}}{\text{Minimize}}\ \mathbf{x}^{\mathrm{T}}\,\mathbf{V_T}\,\mathbf{x} - \theta\,\mathbf{M_T}^{\mathrm{T}}\,\mathbf{x}$$

subject to $\mathbf{x}^{\mathrm{T}}\,\mathbf{T}^{\mathrm{T}}\,\mathbf{1} = 1$ and $\mathbf{x} \geq 0$, where $\mathbf{T}$ is the transformation matrix $\mathbf{T} = [\mathbf{I}_n, -\mathbf{I}_n]$ for $\mathbf{I}_n$ a unit matrix of size $n \times n$; for example, when $n = 2$, $\mathbf{T} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$.

We define the function `robustPortfolioOptimizationWithBoxUncertainty` to find the solution to the mean variance optimization problem with box uncertainty. This function takes three arguments:

- `returns`–a matrix of stock returns
- θ–the risk coefficient
- α–the confidence level for the uncertainty set

```
In[52]:= robustPortfolioOptimizationWithBoxUncertainty[returns_,
           θ_, α_] := Module[
           {m, n, mean, variance, δ, i, ii, μ, σ, weights,
            solution, solutionweights},
           {m, n} = Dimensions@returns;
           mean = Mean@returns;
           variance = Covariance@returns;
           δ = -Quantile[NormalDistribution[], (1 - α) / 2] ×
             Sqrt@Variance@returns / Sqrt@m;
           i = IdentityMatrix[n];
           ii = ArrayFlatten@{{i, -i}};
           μ = θ (mean.ii - δ.ArrayFlatten[{{i, i}}]);
           σ = Transpose[ii].variance.ii;
           weights = w# & /@ Range[2 n];
           solution =
            NMinimize[{2 weights.σ.weights - θ weights.μ,
               Total[ii.weights] == 1, Map[-1 ≤ # ≤ 1 &, weights]},
              weights];
           solutionweights = ii.(weights /. Last@solution);
           Text@Column[{
              Row[{"Weights = ", solutionweights}],
              Row[{"Portfolio Mean = ",
                 NumberForm[mean.solutionweights, {4, 4}]}],
              Row[{"Portfolio Variance = ",
                 NumberForm[
                  Sqrt[solutionweights.variance.solutionweights],
                  {4, 4}]}]
             }]
          ]
```

We compute optimal weights for the following portfolio.

```
In[53]:= returns7b = getReturnsData[{"BA", "AAPL", "WMT"},
            {2010, 01, 01}, {2019, 04, 30}, "Month"];

In[54]:= robustPortfolioOptimizationWithBoxUncertainty[returns7b,
          1, 0.95]

         Weights = {0.464677, 0.462604, 0.0727189}
Out[54]= Portfolio Mean = 0.0189
         Portfolio Variance = 0.0527
```

In [5], an ellipsoidal uncertainty set on expected returns is defined as:

$$\left\{\mathbf{M} : \left(\mathbf{M} - \hat{\mathbf{M}}\right)^{\mathrm{T}} \mathbf{V}_{\mathbf{M}}^{-1} \left(\mathbf{M} - \hat{\mathbf{M}}\right) \leq \delta^2\right\},$$

where $\mathbf{V}_{\mathbf{M}}$ is the covariance matrix of estimation error of expected returns and δ controls the size of the ellipsoid. With this uncertainty set, the mean variance problem can be written as:

$$\underset{\mathbf{w}}{\text{Minimize}}\ \mathbf{w}^{\mathrm{T}} \mathbf{V}_{\mathbf{M}}\, \mathbf{w} - \theta\left(\hat{\mathbf{M}}^{\mathrm{T}} \mathbf{w} - \delta \sqrt{\mathbf{w}^{\mathrm{T}} \mathbf{V}_{\mathbf{M}}\, \mathbf{w}}\right),$$

subject to $\mathbf{w}^{\mathrm{T}} \mathbf{1} = 1$.

The covariance matrix of estimation errors can be approximated in several ways using the sample covariance matrix of stock returns. Assuming that the samples of stock returns are independent and identically distributed, [5] defines $\mathbf{V}_{\mathbf{M}} = \mathbf{V} / n$, where $\mathbf{V}$ is the covariance matrix of stock returns and n is the sample size.

Define `robustPortfolioOptimizationWithEllipsoidalUncertainty` to compute an optimal portfolio with an ellipsoidal uncertainty. It takes three arguments:

- returns (`returns`)
- value of risk coefficient (θ)
- confidence level for the uncertainty set (α)

```
In[55]:= robustPortfolioOptimizationWithEllipsoidalUncertainty[
           returns_, θ_, α_] := Module[
           {m, n, mean, variance, σ, δ, weights, solution,
            solutionweights},
           {m, n} = Dimensions[returns];
           mean = Mean@returns;
           variance = Covariance@returns;
           σ = variance / m;
           δ = Sqrt@Quantile[ChiSquareDistribution@n, α];
           weights = w# & /@ Range[n];
           solution =
            NMinimize[
             {weights.variance.weights - θ weights.mean +
               θ δ Sqrt[weights.σ.weights], Total@weights == 1,
              Map[-1 ≤ # ≤ 1 &, weights]}, weights];
```

```
   solutionweights = weights /. Last@solution;
   Text@Column[{
      Row[{"Weights = ", solutionweights}],
      Row[{"Portfolio Mean = ",
        NumberForm[mean.solutionweights, {4, 4}]}],
      Row[{"Portfolio Variance = ",
        NumberForm[
         Sqrt[solutionweights.variance.solutionweights],
         {4, 4}]}]
     }]
 ]
```

We compute optimal weights for our usual portfolio subject to an ellipsoidal uncertainty set.

```
In[56]:= returns7c = getReturnsData[{"BA", "AAPL", "WMT"},
            {2000, 01, 01}, {2019, 04, 30}, "Month"];

In[57]:= robustPortfolioOptimizationWithEllipsoidalUncertainty[
          returns7c, 1, 0.95]

         Weights = {0.333093, 0.330501, 0.336406}
Out[57]= Portfolio Mean = 0.0138
         Portfolio Variance = 0.0548
```

We can use the built-in function `RobustConvexOptimization` to optimize the portfolio problem, introducing uncertainty in the mean returns or in the covariance.

```
μ = Mean[returns7c];
Σ = Covariance[returns7c];
n = Length[μ];
meanUncertaintyConstraint = 0.01 ≤ θm ≤ 0.3;
return = (μ + θm).w;
risk = Inactive[Norm][CholeskyDecomposition[Σ].w];
weightConstraints = {0 ≤ w ≤ 1, Total[w] == 1};

RobustConvexOptimization[
 risk - return,
 ForAll[θm, θm ∈ Vectors[n, Reals],
  {weightConstraints, meanUncertaintyConstraint}],
 {w},
 {"PrimalMinimumValue", "PrimalMinimizerRules"},
 MaxIterations → 10000]

Out[66]= {0.025654, {w → {0.257885, 0.165993, 0.576122}}}
```

Similarly, we can introduce uncertainty in risk and solve the optimization problem using this same function. See the documentation for an example.

8. Artificial Intelligence

With increasing computational resources and larger datasets, machine learning or artificial intelligence is a growing field in finance. A recent book by Dixon, Halperin and Bilokon [6] is a good reference on theory and applications.

Although the Wolfram Language includes a wide range of functions that work on many types of data, including numerical, categorical, time series, textual, image and audio, we focus on the `Predict` function and time-series data only. The `Predict` function uses input data and returns a predictor function that can be used to forecast the value of dependant variables given the values of independent variables. In this section, we show two examples using financial time series.

One Predictor for Stock Returns

This example uses the Aruoba–Diebold–Scotti (ADS) business conditions index as one predictor of percentage change in the S&P 500 index. We define the function `getADSAndSP500Data`. It downloads the ADS index from the Federal Reserve Bank of Philadelphia and the S&P 500 index returns and then merges them for use in the `Predict` function, given the start and end date as arguments.

```
In[58]:= getADSAndSP500Data[dateStart_, dateEnd_] :=
          Module[{data, adsData, adsDate, adsTimeSeries,
            sp500Returns, mergedData},
           data =
            Import[
              "https://www.philadelphiafed.org/-/media/frbp/assets/
                surveys-and-data/ads/ads_index_most_current
                _vintage.xlsx?la=en&hash=6
                DF4E54DFAE3EDC347F80A80142338E7",
              "Data"]⟦1, 2 ;;⟧;
           adsData = Cases[data⟦All, 2⟧, Except[Last[data⟦All, 2⟧]]];
           adsDate =
            DateObject /@ Map[ToExpression,
              StringSplit[data⟦All, 1⟧⟦1 ;; Length[adsData]⟧, ":"]];
           adsTimeSeries = TimeSeriesWindow[
             TimeSeries[adsData, {adsDate}], {dateStart, dateEnd}];
           sp500Returns =
            Differences[
             Log[FinancialData["^GSPC", "Close",
               {dateStart, dateEnd}]]];
           mergedData = TimeSeriesThread[Rule[First[#], Last[#]] &,
             {adsTimeSeries, sp500Returns},
             ResamplingMethod → Missing[]];
           DeleteCases[mergedData["Values"], x_ → Missing[]]
          ]
```

Using this function, we download the data for the period January 30, 1970, to September 30, 2019, and generate a prediction function. (This takes several minutes.)

```
In[59]:= model1data = getADSAndSP500Data[{1970, 01, 30},
            {2019, 09, 30}];
         model1 = Predict[model1data];
```

Once the prediction function `model1` is generated, we can use it to predict the future value of the stock index value. For instance, here it predicts percentage change in the S&P 500 index if ADS is 0.9, 2 or -1.2.

```
In[61]:= model1[{0.9, 2, -1.2}]

Out[61]= {-0.000188061, 0.00113712, 0.00011818}
```

■ *Several Predictors for Stock Returns*

In this example, we use five monthly macro variables to predict percentage change in the value of the S&P 500 index:

• USSLIND–the leading index for the United States

• UMCSENT–the University of Michigan consumer sentiment index

• CFNAIMA3–the Chicago Fed national activity index: three-month moving average

• MICH–the University of Michigan inflation expectation index

• T10Y2YM–10-year Treasury constant maturity minus 2-year Treasury constant maturity

The function `getMonthlyFedAndSP500Data` takes a list of macroeconomic series IDs (`macroSeriesIds`), start date and end date as input arguments. It returns values for specified macroeconomic variables and S&P 500 index returns in the format suitable for the `Predict` function. The Federal Reserve Bank of St. Louis may require the API key to download its data. The API key can be obtained freely by creating a user account at https://fred.stlouisfed.org (click "my account" and follow the instructions). Use the API key 207071a5f2e90e7816259d3c32c1ab81 if needed.

```
In[62]:= mergeData[a_List, b_List] := Module[
           {rules, keys},
           rules = Apply[# → {##2} &, {a, b}, {2}];
           keys = Union @@ Keys@rules;
           Join[List /@ keys, ##, 2] & @@
             (Lookup[#, keys, Missing[] & /@ #[[1, 2]]] & /@ rules)
          ]
```

```
In[63]:= getMonthlyFedAndSP500Data[macroSeriesIds : {__String},
           dateStart_, dateEnd_] := Module[
           {connect, dataMerged1, dataMerged2, temporaryData1,
            temporaryData2, data, timeSeriesData, sp500,
            sp500TimeSeries, finalData},
           connect = ServiceConnect["FederalReserveEconomicData"];
           dataMerged1 = TimeSeriesWindow[
             connect["SeriesData", "ID" → {macroSeriesIds⟦1⟧}],
             {dateStart, dateEnd}];
           dataMerged2 =
            Transpose@{dataMerged1["Dates"]⟦All, 1, {1, 2}⟧,
              dataMerged1["Values"]};
           Do[
            temporaryData1 = TimeSeriesWindow[
              connect["SeriesData", "ID" → {macroSeriesIds⟦i⟧}],
              {dateStart, dateEnd}];
            temporaryData2 =
             Transpose@{temporaryData1["Dates"]⟦All, 1, {1, 2}⟧,
               temporaryData1["Values"]};
            dataMerged2 = mergeData[dataMerged2, temporaryData2],
            {i, 2, Length@macroSeriesIds}
           ];
           data = Select[dataMerged2, ! MemberQ[#, Missing[]] &];
           timeSeriesData = TimeSeries[data⟦All, 2 ;;⟧,
             {data⟦All, 1⟧}];
           sp500 = Differences@
             Log@FinancialData["^GSPC", "Close",
               {dateStart, dateEnd, "Month"}];
           sp500TimeSeries = TimeSeries[100 sp500["Values"],
             {sp500["Dates"]⟦All, 1, {1, 2}⟧}];
           finalData = TimeSeriesThread[Rule[First[#], Last[#]] &,
             {timeSeriesData, sp500TimeSeries},
             ResamplingMethod → Missing[]];
           DeleteCases[finalData["Values"], {___} → Missing[]]
          ]
```

Using this function, we download five macro variables as well as the S&P 500 index returns over the period January 30, 1983, to May 30, 2019.

```
In[64]:= model2data = getMonthlyFedAndSP500Data[
            {"USSLIND", "UMCSENT", "CFNAIMA3", "MICH", "T10Y2YM"},
            {1983, 01, 30}, {2019, 06, 30}];
```

We can generate the prediction function using the data and predict the value. `Predict` can take the `Method` option to specify which regression method to use.

```
In[65]:= model2 = Predict[model2data, Method → "NearestNeighbors"];
         model2[{1.66, 78, 0.04, 4, 1.02}]

Out[66]= 0.873431
```

We have shown some applications of only the built-in function `Predict`. However, the Wolfram Language comes with many other built-in functions that are useful in classification, discriminant analysis and neural networks. You can use these tools to learn from the data and build models to extract useful information. We encourage you to explore more about machine learning in Mathematica.

9. Conclusion

As financial data becomes increasingly available, serious data analysis requires knowing software to manipulate large datasets. Aside from demonstrating many built-in functions, we introduced many custom functions especially designed for technical computation of financial data. Mathematica can do much more than what we have shown in this article. The Wolfram Language in general and Mathematica in particular are well-suited to implement sophisticated financial models, including pricing securities, trading strategies, simulation, optimization, risk management and time-series analysis [7, 8]. Mathematica's built-in knowledge is also very useful for asset pricing models based on estimating the stochastic discount factor using the generalized method of moments.

References

[1] P. Wellin, *Essentials of Programming in Mathematica*, Cambridge, UK: Cambridge University Press, 2016.

[2] G. Cornuéjols, J. Peña and R. Tütüncü, *Optimization Methods in Finance*, 2nd ed., New York: Cambridge University Press, 2018.

[3] R. T. Rockafellar and S. Uryasev, "Optimization of Conditional Value-at-Risk", *The Journal of Risk*, **2**(3), 2000 pp. 21–41. https://doi.org/10.21314/JOR.2000.038.

[4] R. S. Tsay, *An Introduction to Analysis of Financial Data with R*, Hoboken, NJ: Wiley, 2013.

[5] W. C. Kim, J. H. Kim and F. J. Fabozzi, *Robust Equity Portfolio Management*, Hoboken, NJ: Wiley, 2016.

[6] M. Dixon, I. Halperin and P. Bilokon, *Machine Learning in Finance from Theory to Practice*, Cham, Switzerland: Springer, 2020.

[7] A. L. Lewis, *Option Valuation under Stochastic Volatility: With Mathematica Code*, Newport Beach, CA: Finance Press, 2000.

[8] A. L. Lewis, *Option Valuation under Stochastic Volatility II: With Mathematica Code*, Newport Beach, CA: Finance Press, 2016.

R. Adhikari, “Selected Financial Applications,” *The Mathematica Journal*, 2021. doi.org/10.3888/tmj.23-5.

About the Author

Ramesh Adhikari is an Associate Professor of Finance at Humboldt State University. Prior to coming to HSU, he taught undergraduate and graduate students at Tribhuvan University and worked at the Central Bank of Nepal. He was also a research fellow at Osaka Sangyo University, Osaka, Japan. He earned a Ph.D. in Financial Economics from the University of New Orleans. He is interested in the areas of computational finance and high-dimensional statistics.

Ramesh Adhikari
School of Business, Humboldt State University
1 Harpst Street
Arcata, CA 95521
ramesh.adhikari@humboldt.edu