11

Economic and Financial Applications

The latest Mathematica versions include multiple capabilities related to economics and finance. The new functionality makes it easy to perform financial visualizations and to access both historical and real time data. The commands related to stock markets and financial derivatives would be especially useful to readers interested in investment portfolio management. Additionally, the functions for optimization, with applications in economics and many other fields, have been improved significantly. A new function to solve the Traveling Salesman Problem in a very efficient way has been included. All of these topics will be discussed in the sections below using real world examples. This chapter should be read along with Chapter 6, the one covering probability and statistics.

11.1 Financial Information

11.1.1 Introduction

Mathematica, as part of its computable data functionality, includes the command `FinancialData` to access finance-related information. Although in theory the program can display thousands of indicators, in reality the accessible ones are mostly those related to the US markets. Those readers who may need reliable access to other markets or real time financial data would probably be interested in taking a look at the Wolfram Finance Platform, a new product from Wolfram Research (the developers of *Mathematica*) that focuses on finance: http://www.wolfram.com/finance-platform/.

In any case, although the information may not be directly accessible from `FinancialData`, nowadays it would be easy to access it. For example, many bank platforms enable their clients to download data in an Excel format that could then be imported into *Mathematica* and analyzed using any of the multiple functions related to finance:

http://reference.wolfram.com/mathematica/guide/Finance.html

11.1.2 FinancialData

`FinancialData[ "name", "`*property*`",` {*start, end, interval*}] gives the value of the specified property for the financial entity “name” (it could be an index, a stock, a commodity, etc.) for a given period of time (by default it returns the most recent value). We recommend to read the documentation pages to know more about this function’s capabilities.

- The symbols ("name") used are the ones available in http://finance.yahoo.com/ since Yahoo Finance supplies the information to the function. In the website we can find that the symbol for the Standard & Poor's 500 index, that includes the 500 largest public companies in terms of market capitalization that trade in either NYSE or NASDAQ is: ^GSPC (we can also use SP500). We can check it as follows:

```
FinancialData["SP500", "Name" ]
```

```
S&P 500
```

- If we want to see the price (with a few minutes delay) we can type:

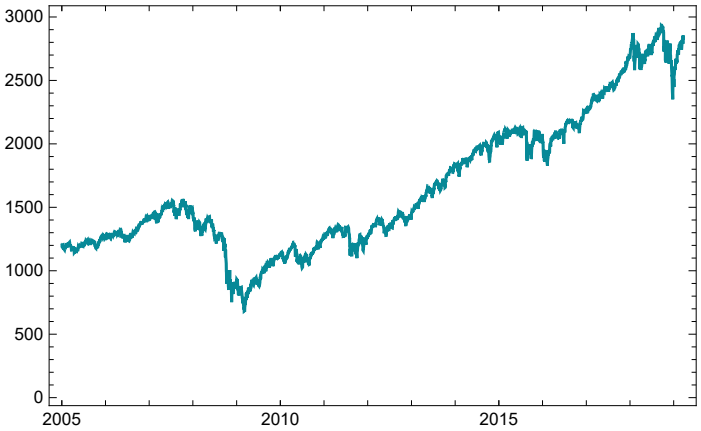
```
FinancialData["^GSPC"]
```

```
2269.
```

- We can also access historical data. In this example we show the trajectory of the S&P 500 index from January 1, 2000 to December 31, 2016.

```
In[•]:= DateListPlot[FinancialData["^GSPC",
          {"January 1 2005", "March 25 2019"}], Joined → True]
```

Out[•]=



11.1.3 Visualization

- The graph below displays the trajectory of GE in the NYSE . For each trading day, we can see the open, high, low and close prices.

```
In[•]:= CandlestickChart[{"NYSE:GE", {{2018, 01, 1}, {2018, 12, 31}}}]
```

Out[•]=

In the world of professional stock market investors, there's a group that uses chart analysis as its main tool for making investment decisions. Its members are known as chartists and their techniques are known as technical analysis. They closely study the past performance of stocks or indices to find trends and support and resistance levels. These people associate certain graph shapes to future markets ups and downs. Although there's no

sound mathematical theory justifying their predictions, all economic newspapers have a section dedicated to this type of analysis.

- The powerful function `InteractiveTradingChart`, introduced in *Mathematica* 8, includes practically all the functionality that a chartist may require. In this example, we display GE's share price history for a given period. Notice that in the upper area of the graph, we can see a trimester at a time and using the bottom part we can move through the entire period. We can also choose the time interval (days, weeks or months) and even what chart type and indicators to show.

```
In[•]:= InteractiveTradingChart[{"NYSE:GE", {{2018, 10, 1}, {2019, 03, 25}}}]
```

11.1.4 Automatic Adjustments

With *Mathematica* we can automatically fit our financial data to a given distribution (by default the program will use the maximum likelihood method) with the function `EstimatedDistribution`.

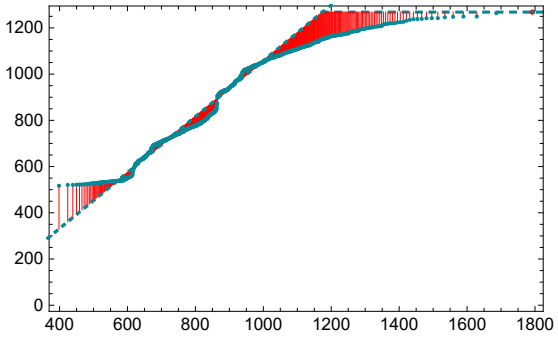
- Let's fit Google's share price to a lognormal distribution after downloading the data.

```
In[•]:= googleStock =
          FinancialData["GOOG", {{2015, 3, 10}, {2019, 02, 25}, "Day"}, "Value"];

In[•]:= dist = EstimatedDistribution[googleStock, LogNormalDistribution[μ, σ]]

Out[•]= LogNormalDistribution[6.74006, 0.235848]
```

- The fit is poor on both tails. As a matter of fact, all the efforts that have been made trying to predict financial markets have produced unsatisfactory results. These markets exhibit what Mandelbrot (the father of fractal geometry) called "wild randomness" (single observations can impact the total in a very disproportionate way).

```
In[•]:= QuantilePlot[googleStock, dist, Filling → Automatic, FillingStyle → Red]
```

Out[•]=



11.2 Financial Functions

The functions related to financial calculations can be found in guide/Finance. We will give an overview of some of them in this section.

11.2.1 Bonds

```
Clear["Global`*"]
```

Bonds are financial assets that we can also analyze with *Mathematica*. When an organization needs capital, instead of borrowing from banks, they can issue bonds. Bonds can be considered a type of loan in which the lenders are the ones purchasing them. Those lenders can usually be anyone. There are different types of bonds. Normally they pay a fixed amount of interest at predetermined time intervals: annually, half-annually, quarterly or even monthly. Some bonds issued for short periods of time (less than one year), such as *Treasury Bills*, don't pay interest explicitly, they are sold at a discount from their par value.

For analyzing bonds, *Mathematica* has the function `FinancialBond`. It takes the following arguments (some of them are optional):

"FaceValue"	face value, par value
"Coupon"	coupon rate, payment function
"Maturity"	maturity or call/put date
"CouponInterval"	coupon payment interval
"RedemptionValue"	redemption value
"InterestRate"	yield to maturity or yield rate
"Settlement"	settlement date
"DayCountBasis"	day count convention

Let's see some examples.

- The yield to maturity of a €1,000 30-year bond maturing on February 28, 2019, with a coupon of 6% paid quarterly that was purchased on January 6, 2013 for €900 would be:

```
In[•]:= FindRoot[FinancialBond[{"FaceValue" → 1000, "Coupon" → 0.05,
          "Maturity" → {2019, 2, 28}, "CouponInterval" → 1/4},
         {"InterestRate" → y, "Settlement" → {2013, 1, 6}}] == 900, {y, .1}]
```

```
Out[•]= {y → 0.0701618}
```

11.2.2 The Black–Scholes Equation

In this example we first compute the price of a derivative using *Mathematica*'s existing function and then compare it with the result obtained by solving the Black–Scholes equation directly

(http://www.wolfram.com/language/11/partial-differential-equations/find-the-value-of-a-european-call-option.html).

- Let's start by using FinancialDerivative to compute the price of a European vanilla call option with the data from the previous section:

```
In[•]:= spotPrice = 9.87; euribor12m = 0.00058; vol = 0.528374;

In[•]:= FinancialDerivative[{"European", "Call"}, {"StrikePrice" -> spotPrice,
         "Expiration" -> 1}, {"InterestRate" -> euribor12m, "Volatility" -> vol,
         "CurrentPrice" -> spotPrice}]

Out[•]= 2.05882
```

- Now, let's do the same computation using the Black–Scholes equation (To create more visually appealing documents you can convert the input cell to the traditional format: **Cell ▸ Convert to ▸ TraditionalForm**):

```
In[•]:= BlackScholesModel =
```

$$\left\{-r\,c(t,s)+r\,s\,\frac{\partial c(t,s)}{\partial s}+\frac{1}{2}s^2\sigma^2\,\frac{\partial^2 c(t,s)}{\partial s^2}+\frac{\partial c(t,s)}{\partial t}==0,\ c(T,s)==\max(s-k,0)\right\};$$

- After solving the boundary value problem we get:

```
In[•]:= (dsol = c[t, s] /.
          DSolve[BlackScholesModel, c[t, s], {t, s}][[
          1]]) // TraditionalForm
```

Out[•]//TraditionalForm=

$$\frac{1}{2}e^{-rT}\left(s\,e^{rT}\,\text{erfc}\left(\left(2\log(k)+\left(2r+\sigma^2\right)(t-T)-2\log(s)\right)\Big/\left(2\sqrt{2}\ \sigma\sqrt{T-t}\right)\right)-\right.$$
$$\left.k\,e^{rt}\,\text{erfc}\left(\left(2\log(k)+\left(2r-\sigma^2\right)(t-T)-2\log(s)\right)\Big/\left(2\sqrt{2}\ \sigma\sqrt{T-t}\right)\right)\right)$$

- Finally, we calculate the price of the derivative and compare it with the one previously obtained. We can see it's the same:

```
In[•]:= dsol /. {t -> 0, s → spotPrice, k -> spotPrice, σ -> vol, T -> 1,
         r ->  euribor12m}

Out[•]= 2.05882
```

11.3 Optimization

11.3.1 What is Constrained Optimization?

The optimization problems that we discuss in this section are related to finding an optimum (maximum or minimum, depending on the case) for an objective function of *n* variables, given *r* constraints.

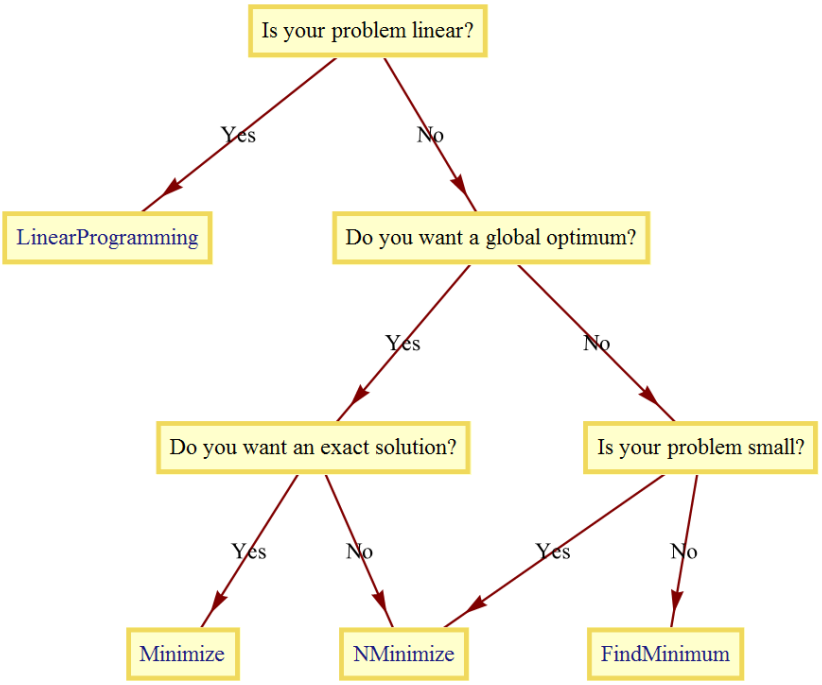
This type of problems is common in several fields such as economics, finance and engineering as we will show using different examples. You can find an excellent tutorial in the documentation pages: "Introduction to Constrained Optimization in the Wolfram Language" (tutorial/ConstrainedOptimizationIntroduction).

To solve them, *Mathematica* has the following functions:

Function	Application	Description
FindMinimum / FindMaximum	Local Numerical optimization	Linear programming methods, non-linear, interior points, and the use of second derivatives
NMinimize / NMaximize	Global Numerical optimization	Linear programming methods, Nelder-Mead, *differential evolution*, *simulated annealing*, random search
Minimize / Maximize	Exact Global optimization	Linear programming methods, algebraic cylindrical decomposition, Lagrange multipliers and other analytical methods, integer linear programming.
LinearProgramming	Linear optimization	Simplex, modified simplex, interior point

The following tree can help us decide the most appropriate function to find the minimum (To find the maximum we would use the functions `Maximize`, `NMaximize` or `FindMaximum` instead).

- The command below draws the tree. When creating a document, you may be interested in seeing only the result. To hide the input just click directly on the output marker cell.



In many cases the problem can be solved using several functions. With the exception of `LinearProgramming`, the syntax for the rest of the functions is very similar: **f**[{objective function, *constraints*}, {*variables*}].

```
Clear["`Global`*"]
```

Let's see some examples.

- Given the objective function (ob) with constraints c1 and c2:

```
var = {x, y}, ob = x + y, c1 = 0 ≤ x ≤ 1, c2 = 0 ≤ y ≤ 2
```

- The maximum can be found using any of the previously mentioned commands:

```
{NMaximize[{ob,c1,c2}, var], Maximize[{ob,c1,c2}, var],
FindMaximum[{ob,c1,c2}, var]}
```

```
{{3., {x → 1., y → 2.}}, {3, {x → 1, y → 2}}, {3., {x → 1., y → 2.}}}
```

- We can see that the maximum occurs when {x → 1, y → 2} and is 3. We can verify it (remember that "/." is used for substitutions).

```
x + y /. {x → 1, y → 2}
```

```
3
```

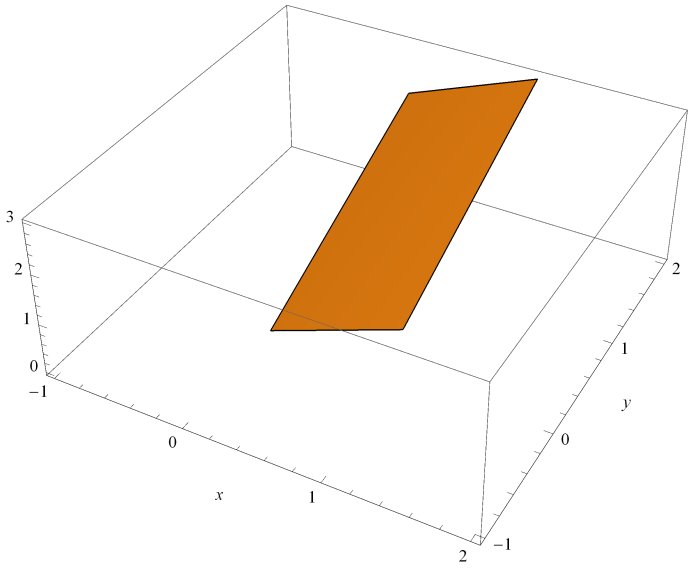
- The minimum can be computed as follows:

```
Minimize[{ob,c1,c2}, var]
```

```
{0, {x → 0, y → 0}}
```

- We can interpret the problem graphically by drawing the plane $x + y$ only in the region: $\{0 \le x \le 1, 0 \le y \le 2\}$.

```
Plot3D[x + y, {x, -1, 2}, {y, -1, 2},
 RegionFunction -> Function[{x, y}, 0 ≤ x < 1 && 0 ≤ y < 2],
 AxesLabel → Automatic, Mesh → None]
```



The graph clearly shows that the maximum is in (1,2) and the minimum in (0,0). Use the mouse to move the graph around to see it from different perspectives.

In certain situations we'd like the variables to take only integer values. To add this extra requirement in *Mathematica,* we use the built-in symbol **Integers**.

- Let's consider the following model in which we want to minimize the objective function (ob) (notice that equality when typing constraints is "=="):

```
Clear["`Global`*"]

var = {x, y};
ob = x + 2 y;
c1 = -5 x + y == 7;
c2 = x + y ≥ 26;
c3 = x ≥ 3;
c4 = y ≥ 3;
```

```
NMinimize[{ob, c1, c2, c3, c4, var ∈ Integers}, var]
```

```
{58., {x → 4, y → 27}}
```

Since this is a linear problem, we can also use `LinearProgramming`. As a matter of fact, this command is the most appropriate one for linear problems, especially if the number of variables is large. The syntax is: `LinearProgramming`[*c*, *m*, *b*] . This function finds the vector **x** that minimizes the quantity **c.x** subject to the constraints **m.x** ≥ **b** and **x** ≥ 0. We can limit the values that the variables (some or all of them) can take to just integers with `LinearProgramming`[..., Integers].

- For comparison purposes, let's solve the same problem using `LinearProgramming`:

$x + 2y$ → c : {1, 2}

$-5x + y == 7$ → m1 : {−5, 1} b1 : {7, 0}

$x + y \geq 26$ → m2 : {1, 1} b2 : {26, 1}

Notice that the syntax to indicate the type of constraint is as follows: $\{b_i, 0\}$ if $m_i . x == b_i$; $\{b_i, 1\}$ if $m_i . x \geq b_i$ and $\{b_i, -1\}$ if $m_i . x \leq b_i$.

```
LinearProgramming[{1, 2}, {{-5, 1}, {1, 1}}, {{7, 0}, {26, 1}},
  {{3, Infinity}, {4, Infinity}}, Integers] // Quiet
```

```
{4, 27}
```

Next we'll show some examples for nonlinear optimization.

```
Clear["`Global`*"]
```

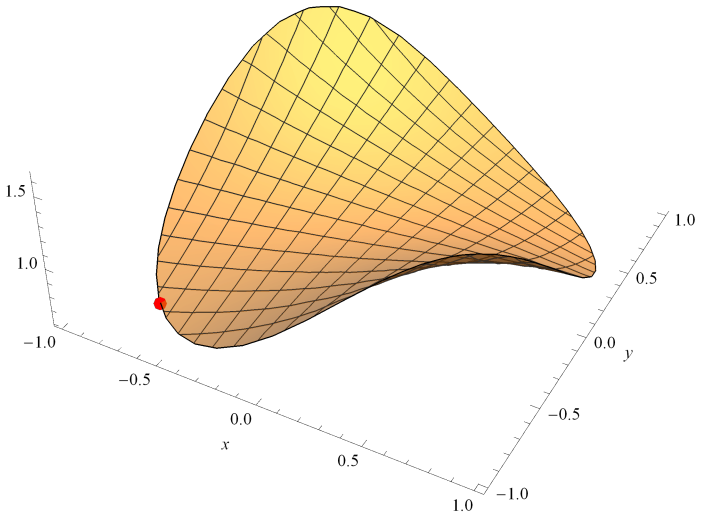
- Here is another example of nonlinear optimization, in this case for finding the global minimum (`Minimize`). The objective function is Exp(−*x y*) with the constraint that *x*, *y* ∈ a Circle centered in {0, 0} and with a radius *r* = 1. We show the result, both numerically and graphically.

```
m = Minimize[{Exp[-x y], {x, y} ∈ Disk[]}, {x, y}]
```

$$\left\{\frac{1}{\sqrt{e}}, \left\{x \to -\frac{1}{\sqrt{2}}, y \to -\frac{1}{\sqrt{2}}\right\}\right\}$$

```
g = Plot3D[Exp[-x y], {x, y} ∈ Disk[], Axes → True,
    Boxed → False, PlotStyle → Opacity@0.6, AxesLabel → Automatic];
```

```
Show[g,
 Graphics3D[{PointSize@Large, Red, Point[{x, y, m[[1]]} /. m[[2]]]}]]
```



11.3.2 Application: Portfolio Optimization

A bank commissions a consulting firm to analyze the most profitable way to invest €10,000,000 for two years given the options in the table below. The table includes the expected rate of return (bi-annual) and associated risk.

Product (i)	Return (%), b_i	Risk (%) r_i
Mortgages	9	3
Mutual funds	12	6
Personal loans	15	8
Commercial loans	8	2
Certificates/Bonds	6	1

The capital not invested in any of the products will be placed in government bonds (assumed to be riskless) with a bi-annual rate of return of 3%. The objective of the consulting firm is to allocate the capital to each of the products to meet the following goals:

(a) Maximize the return per € invested.
(b) Keep the possibility of loss to a maximum of 5% of the total amount invested.
(c) Invest at least 20% in commercial loans.
(d) Allocate to mutual funds and personal loans an amount no larger than the one invested in mortgages.

- Variables: x_i is the percentage of capital invested in product i. The amount placed in government bonds can be considered as a new product with an expected return, in percentage, $b_6 = 3$ and risk $r_6 = 0$. Therefore:

```
In[•]:= var = {x1(*Mortgages*), x2(*Mutual funds*),
        x3(*Personal loans*), x4 (*Commercial loans*), x5
        (*Certificates/Bonds*), x6(*Government debt*)};
```

- Objective function to maximize: $\sum_{i=1}^{6} b_i x_i$.

```
In[•]:= of = 9 x1 + 12 x2 + 15 x3 + 8 x4 + 6 x5 + 3 x6;
```

- Constraint 1: All the money is invested: $\sum_{i=1}^{6} x_i = 1$.

```
In[•]:= c1 = x1 + x2 + x3 + x4 + x5 + x6 == 1;
```

- Constraint 2: The average risk is R = $\sum_{i=1}^{6} r_i x_i / \sum_{i=1}^{6} x_i \leq 5$ as $r_6 = 0$ then:

```
In[•]:= c2 = (3 x1 + 6 x2 + 8 x3 + 2 x4 + x5) / (x1 + x2 + x3 + x4 + x5) ≤ 5
```

$$\text{Out[•]=}\quad \frac{3 x_1 + 6 x_2 + 8 x_3 + 2 x_4 + x_5}{x_1 + x_2 + x_3 + x_4 + x_5} \leq 5$$

- Constraint 3: At least 20% of the capital has to be invested in commercial loans ($x_4 \geq 0.2$).

```
In[•]:= c3 = x4 ≥ 0.2;
```

- Constraint 4: The percentage invested in mutual funds (x_2) and personal loans (x_3) cannot be bigger than the one invested in mortgages (x_1).

```
In[•]:= c4 = x2 + x3 ≤ x1;
```

- Constraint 5: No percentage invested can be negative ($x_i \geq 0$).

```
In[•]:= c5 = Map[# ≥ 0 &, var];
```

- Using NMaximize, *Mathematica* tells us that we'd get a return of 11.2% investing 40% in mortgages, 40% in personal loans and 20% in commercial loans.

```
In[•]:= sol = NMaximize[{of, c1, c2, c3, c4, c5}, var]
Out[•]= {11.2, {x1 → 0.4, x2 → 0., x3 → 0.4, x4 → 0.2, x5 → 0., x6 → 0.}}
```

11.4 The Shortest Path Problem

11.4.1 The Traveling Salesman Problem

There are problems in mathematics that look very specialized and easy to solve at first sight but in reality they are very complicated and their resolution can have applications in a wide variety of fields. One of the most famous examples is the *Traveling Salesman Problem* (TSP). This problem can be defined as follows: A traveler would like to visit *N* cities starting and finishing in the same arbitrarily chosen one, minimizing the total distance traveled and passing through each city only once. If you think about it, there are many real-world problems that are actually equivalent to the TSP: route optimization (transportation, logistics), optimum network layout (trains, roads, electricity), even the connections inside microprocessors to minimize calculation time.

Mathematically, the problem consists of finding a permutation P = $\{c_0, c_1, ..., c_{n-1}\}$ such that $d_P = \sum_{i=1}^{n-1} d[c_i, c_{i+1}]$ would be a minimum, with d_{ij} representing the distance from city *i* to city *j*.

11.4.2 A Tour Around South American Cities

The next example is about organizing a trip to several cities in South America. In what order should we visit them to minimize the total distance traveled?

- The cities we're visiting are the following (we indicate both, the cities and their respective regions and countries to avoid ambiguity):

```
cities={Entity["City", {"Asuncion", "Asuncion", "Paraguay"}],
 Entity["City", {"Bogota", "DistritoCapital", "Colombia"}],
 Entity["City", {"RioDeJaneiro", "RioDeJaneiro", "Brazil"}],
 Entity["City", {"BuenosAires", "BuenosAires", "Argentina"}],
 Entity["City", {"Caracas", "DistritoCapital", "Venezuela"}],
 Entity["City", {"LaPaz", "LaPaz", "Bolivia"}],
 Entity["City", {"Lima", "Lima", "Peru"}],
 Entity["City", {"Montevideo", "Montevideo", "Uruguay"}],
 Entity["City", {"Quito", "Pichincha", "Ecuador"}],
 Entity["City", {"Santiago", "Metropolitana", "Chile"}]};
```

- Now we can calculate the best visiting order:

```
order = Last[FindShortestTour[GeoPosition[cities]]]
```

```
{1, 8, 4, 10, 6, 7, 9, 2, 5, 3, 1}
```

- Finally, we can display the route on a map:

```
GeoListPlot[cities[[order]], Joined → True]
```