

Mathematica 6: data analysis, visualization and interaction

Roberto Cavaliere - roberto@wolfram.com

Wolfram Research - www.wolfram.com

Introduction

Mathematica's capabilities in statistics-related areas have steadily grown. It includes industrial-strength versions of the usual statistics and visualization capabilities, together with many important unique capabilities, made possible by its broad overall scope and extensibility, immense interconnected web of algorithms and integrated programming language and interface framework.

Mathematica provides integrated support both for classical statistics and for modern large-scale data analysis. Incorporating the latest numeric and computational geometry algorithms, Mathematica provides high-accuracy and high-reliability statistical results for datasets of almost unlimited size.

Beyond the internal algorithms and data analysis functionalities, Mathematica provides additional facilities useful in typical data analysis processes. For instance, it imports and exports data in more than one hundred standard formats, allowing the data exchange with other applications. It has a built-in high-level interface to all standard SQL databases, that allows immediate searching, reading and writing of arbitrary data and expressions. It gives access to a library of carefully cured and continually updated data maintained by Wolfram Research; for instance Financial Data, Chemical Data, Country and City Data and lots more.

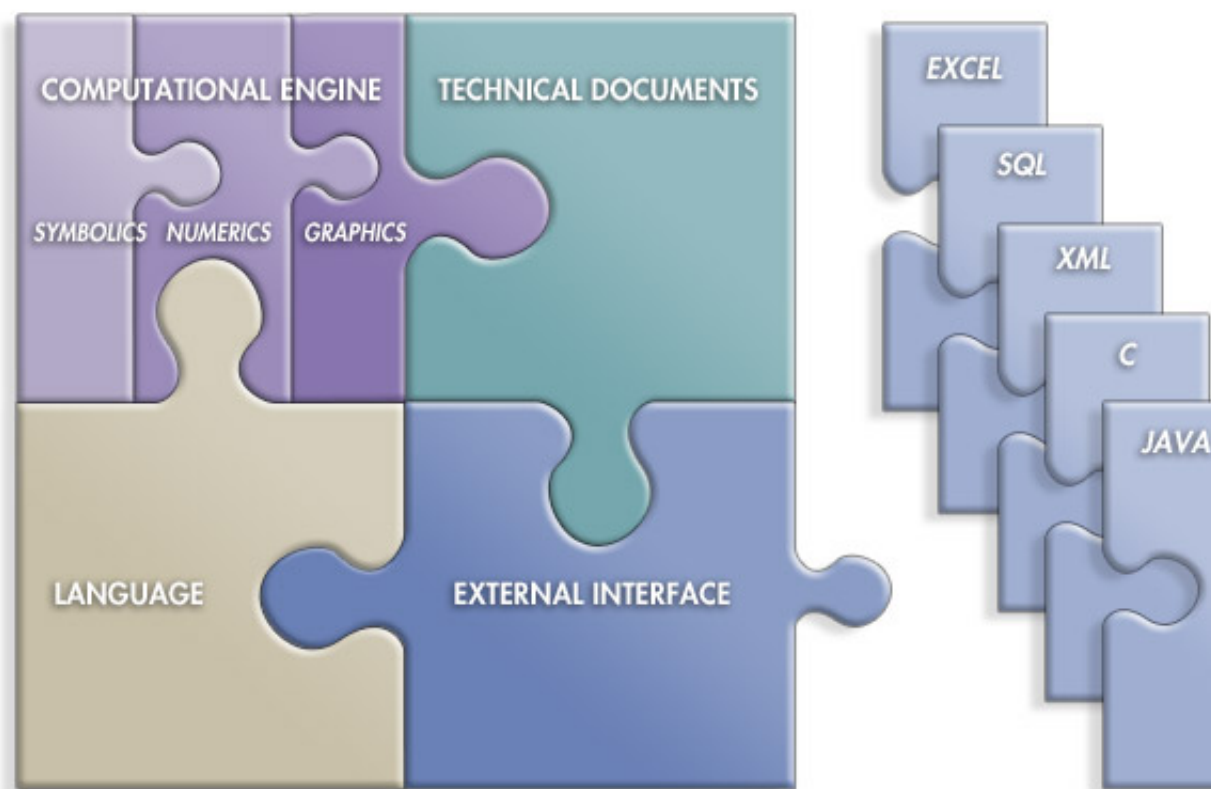
Finally, Mathematica automates the generation of compelling visualizations - and new technology in Mathematica 6 allows any output or computation to immediately become interactive and dynamic. Its integrated notebook mechanism, enable to create documents which can store a complete, formatted, editable, executable history of an analysis, complete with live graphics, interactive controls and arbitrary typeset tabular, mathematical and other material.

This talk

- Introduction to *Mathematica* basic philosophy
- Show a little of the technology in *Mathematica* 6
- Give some examples of how *Mathematica* works and what can do

What is *Mathematica*?

- One view...



- Use dichotomy: application and language
 - An application (use raw, immediately, yourself)
 - A language (encapsulated, extend use, for you or others).

But really...what IS *Mathematica*?

- A Complete system
- Fully integrated across all functionality

As a straightforward example of data import and processing, I will show an example of working with image data. Import a JPEG image showing a picture of bacteria and limestone accretions in the runoff from a geothermal hot spring.

```
In[1]:= Show[ Import[ToFileName[NotebookDirectory[], "Bacteria.JPG"], "JPG"], ImageSize -> 400]
```

Out[1]=



Extract a part of the image data from the Graphics object. The image data consists of a matrix of triples giving the values for each color channel at each pixel.

```
In[2]:= imagedata =
  Import[ToFileName[NotebookDirectory[], "Bacteria.JPG"], "Data"][[1 ;; 100, 51 ;; 150]]
```

A very large output was generated. Here is a sample of it:

```
{{{161, 135, 86}, {150, 121, 65}, <<96>>, {194, 156, 83}, {201, 163, 90}}, <<98>>, {<<1>>}}
```

Show Less

Show More

Show Full Output

Set Size Limit...

Show the part of the data that was extracted with the true colors.

```
In[3]:= imgdetail = Graphics[Raster[imagedata / 256.]]
```

```
Out[3]=
```



What I would like to do is get the size and location of the limestone accretion. There are many ways to approach this. The idea I show below is certainly not the best, but it demonstrates well the ability to explore using the mathematics and interactivity of *Mathematica*.

Compute the mean color of all the pixels shown in the image.

```
In[4]:= meancolor = Mean[Mean[imagedata]]
```

```
Out[4]= { 19 349 / 100 , 778 999 / 5000 , 390 591 / 5000 }
```

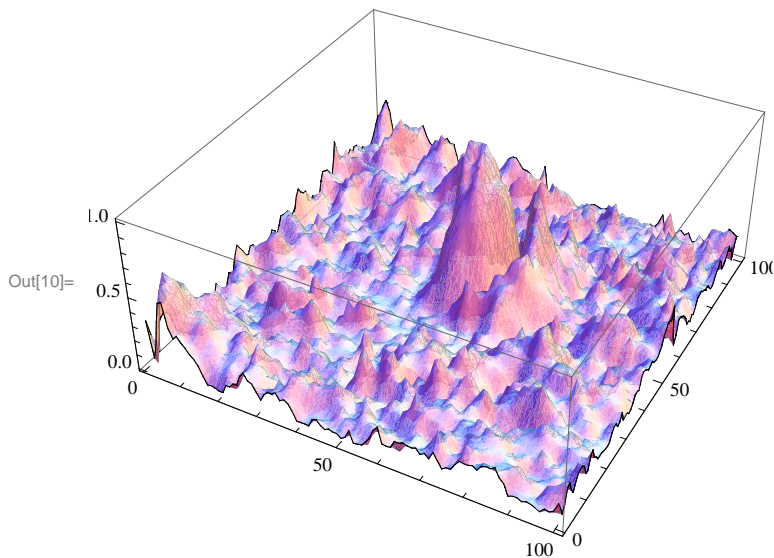
I am interested in highlighting those points with color different than the most prevalent dull yellow. Thinking of the color as a vector in 3-space, I want to highlight the points with different color directions. This can be done by taking the norm of the cross product with the mean color.

Compute the norm of the cross product of the color vector at each point with the mean color and normalize it so that the maximum value is 1.

```
In[5]:= orthogonal = Map[Norm[Cross[meancolor, #]] &, imagedata, {2}];
        orthogonal /= Max[orthogonal];
```

Show a surface plot of the resulting data.


```
In[10]:= ListPlot3D[orthogonal, PlotRange → All, Mesh → False]
```



Define a Gaussian model function.

```
In[7]:= Gaussian[a_, {x0_, y0_}][{x_, y_}] := e-a ((x-x0)2+(y-y0)2)
```

Compute a nonlinear least-squares best fit for a Gaussian to the data.

```
In[8]:= xydata = Flatten[MapIndexed[Flatten[{#2, #1}] &, orthogonal, {2}], 1]
```

A very large output was generated. Here is a sample of it:

$$\left\{ \left\{ 1, 1, \sqrt{\frac{214\,363\,651\,007\,321}{2\,063\,306\,248\,087\,497}} \right\}, \left\{ 1, 2, 16 \sqrt{\frac{5\,557\,467\,151}{294\,758\,035\,441\,071}} \right\}, \right. \\ \left. \ll 9997 \gg, \left\{ 100, 100, 109 \sqrt{\frac{1\,487\,060\,221}{687\,768\,749\,362\,499}} \right\} \right\}$$

Show Less

Show More

Show Full Output

Set Size Limit...

```
In[9]:= fit = FindFit[xydata, Gaussian[a, {x0, y0}][{x, y}], {{a, .1}, {x0, 50}, {y0, 60}}, {x, y}]
```

```
Out[9]= {a → 0.0094613, x0 → 59.0501, y0 → 53.4534}
```

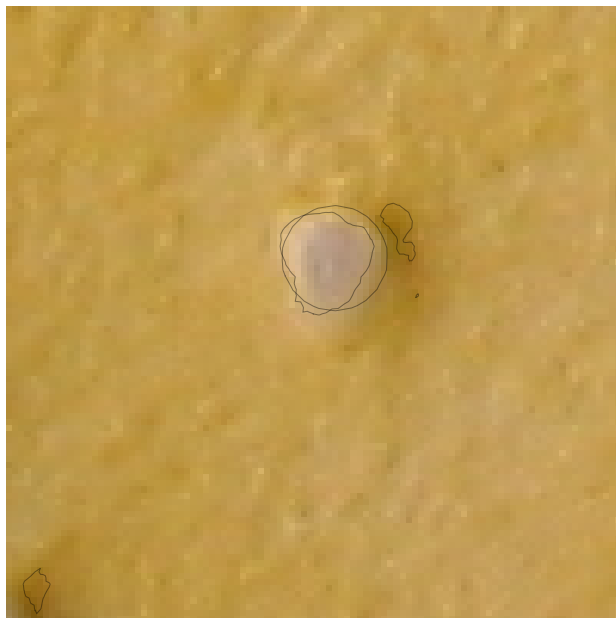
```
In[10]:= Gaussian[a, {x0, y0}][{x, y}] /. fit
```

```
Out[10]= e-0.0094613 ((-59.0501+x)2+(-53.4534+y)2)
```

Show contours for the data and the fit.

```
In[11]:= Show[imgdetail,
  ContourPlot[Evaluate[Gaussian[a, {x0, y0}][{y, x}] /. fit], {x, 0, 100},
    {y, 0, 100}, Contours -> {0.5}, ContourShading -> False, PlotRange -> All],
  ListContourPlot[orthogonal, Contours -> {0.5}, ContourShading -> False, PlotRange -> All]]
```

Out[11]=



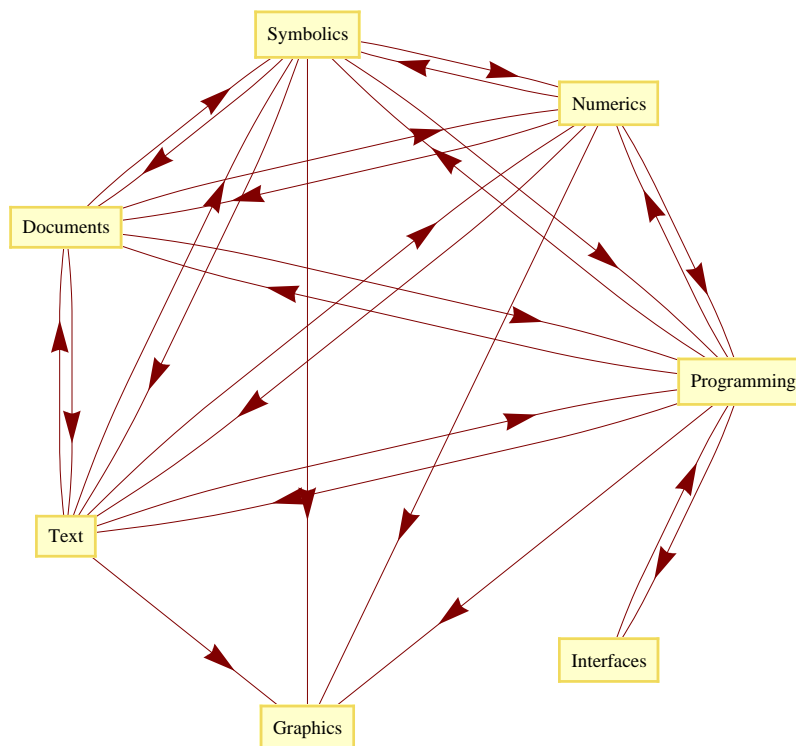
Why "The New Era"?

- "Instantly dynamic": new, unique and important
 - Means a whole new era of modelling, simulation and development.
 - Analogy: what can be done with DOS v Vista/MacOSX

- Revolutionary concept of "personal development environment"
 - Taking non-developers and enabling development
 - Analogy: Google making everyone personal librarian

■ Completing the "fully integrated" vision

```
parts = {"Programming", "Numerics", "Symbolics", "Documents", "Text", "Graphics"};
Apply[Mouseover,
  GraphPlot[#, DirectedEdges → True, VertexLabeling → True, Method → "CircularEmbedding",
    MultiedgeStyle → 0.04, SelfLoopStyle → None, ImageSize → 400] & /@
  {Join[DeleteCases[Flatten@Outer[Rule, parts, parts],
    (HoldPattern["Graphics" → _] | HoldPattern["Documents" → "Graphics"])]],
    {"Interfaces" → "Programming", "Programming" → "Interfaces"}],
  AppendTo[parts, "Interfaces"]; Flatten@Outer[Rule, parts, parts]]]
```



***Mathematica's* basic concept: everything is an expression**

Mathematica is built on a simple and efficient concept: everything is (internally) represented as an expression having the form **Name[arguments, options]**.

All objects we can use inside *Mathematica* is converted to an expression or expression of expressions (nesting). This makes *Mathematica* an extremely powerful and flexible language, and at the same time this is the "trick" for an immediate understanding of how the system works.

The version 6 has fully integrated this concept also for those elements, such graphics, that apparently have a different shape. To explore the internal representation of *Mathematica* objects, we have some functions like FullForm, InputForm TreeForm and so on.

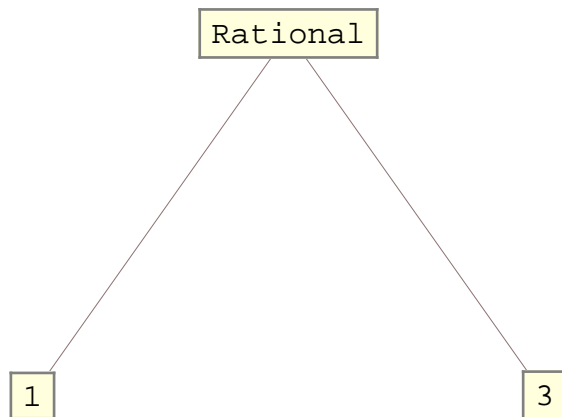
Here in the internal form of a rational number

```
In[12]:= FullForm[ $\frac{1}{3}$ ]
```

```
Out[12]//FullForm=  
Rational[1, 3]
```

```
In[13]:= TreeForm[ $\frac{1}{3}$ ]
```

```
Out[13]//TreeForm=
```



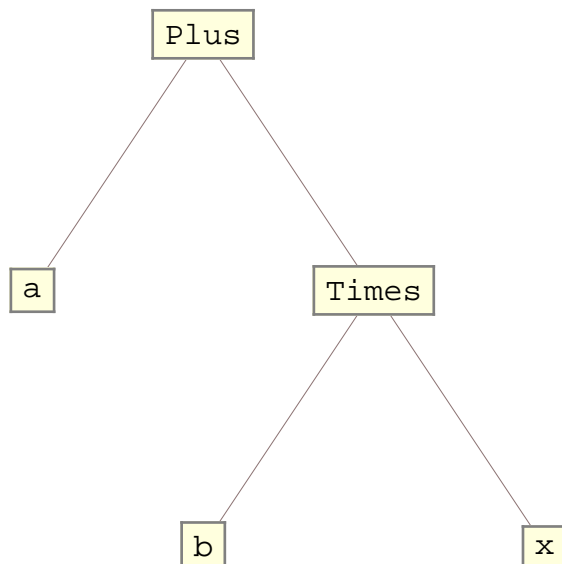
A binomial

```
In[14]:= FullForm[a + b x]
```

```
Out[14]//FullForm=  
Plus[a, Times[b, x]]
```

```
In[15]:= TreeForm[a + b x]
```

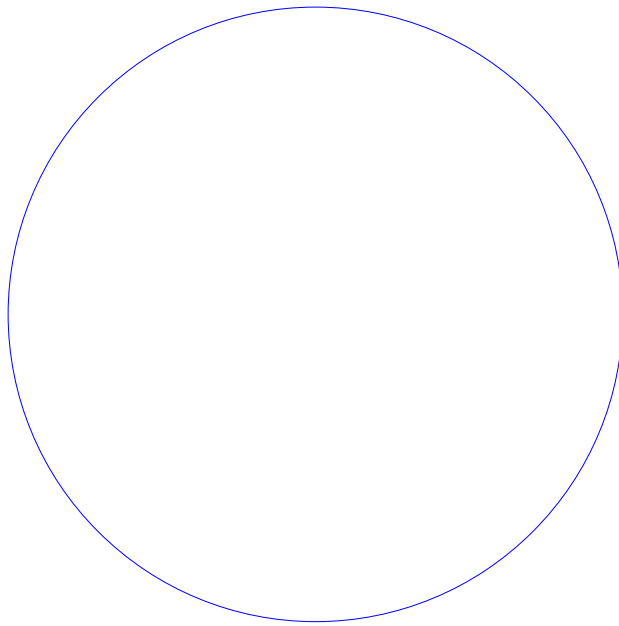
```
Out[15]//TreeForm=
```



A graphic

```
In[16]:= gr = Graphics[{Blue, Circle[{0, 0}]}]
```

```
Out[16]=
```



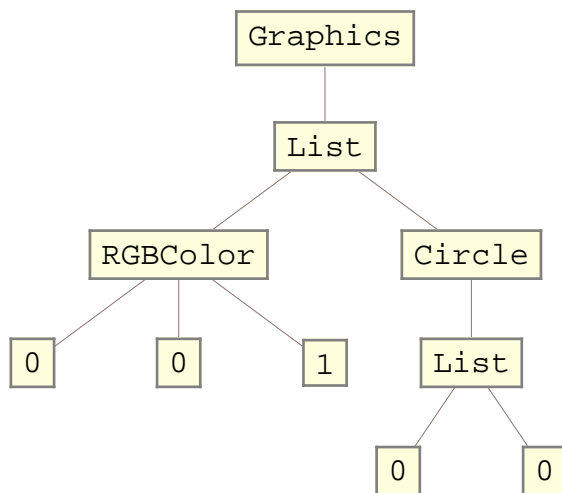
```
In[17]:= FullForm[gr]
```

```
Out[17]//FullForm=
```

```
Graphics[List[RGBColor[0, 0, 1], Circle[List[0, 0]]]]
```

```
In[18]:= TreeForm[gr]
```

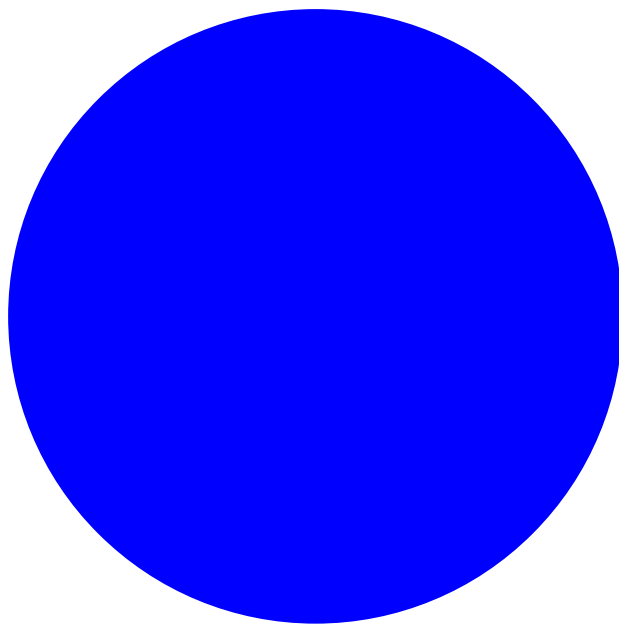
```
Out[18]//TreeForm=
```



Knowing the internal form is also useful to manage our results without to find complex alternatives or re-write pieces of code. If I want to replace the circle with a disk, I can use a replacement rule specifying that the objects Circle have to be replaced by the object Disk.

```
In[19]:= ReplaceAll[gr, Circle → Disk]
```

```
Out[19]=
```

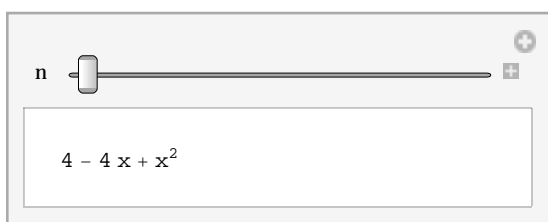


Even the front end (the *Mathematica* user interface module) manages objects with the same philosophy.

Here is a dynamic object enabling the manipulation of a polynomial expansion

```
In[20]:= man = Manipulate[Expand[(x - 2)^n], {n, 2, 10, 1}]
```

```
Out[20]=
```



I can ask for the InputForm and the TreeForm as well

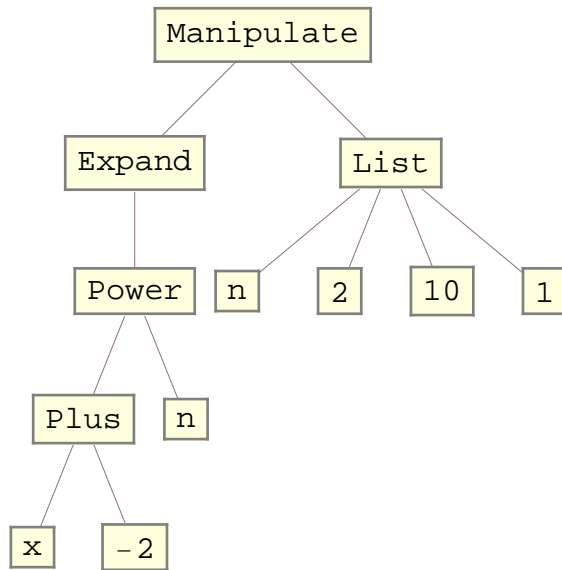
```
In[21]:= FullForm[man]
```

```
Out[21]//FullForm=
```

```
Manipulate[Expand[Power[Plus[x, -2], n]], List[n, 2, 10, 1]]
```

```
In[22]:= TreeForm[man]
```

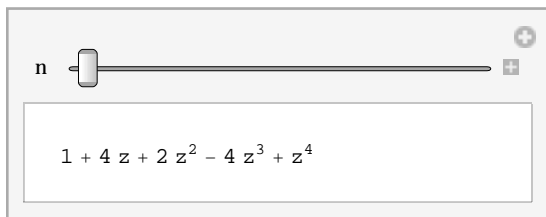
```
Out[22]//TreeForm=
```



So, if I want to change the x variable, I can use ReplaceAll once again

```
In[23]:= ReplaceAll[man, x -> (z - 1) ^ 2]
```

```
Out[23]=
```

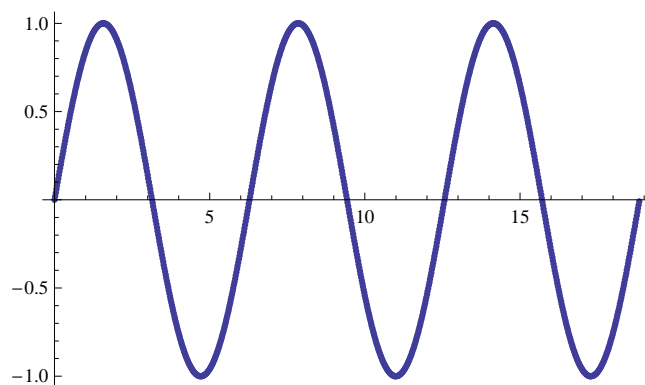


Another example.

Here is a list of points of the function Sin and the ListPlot image

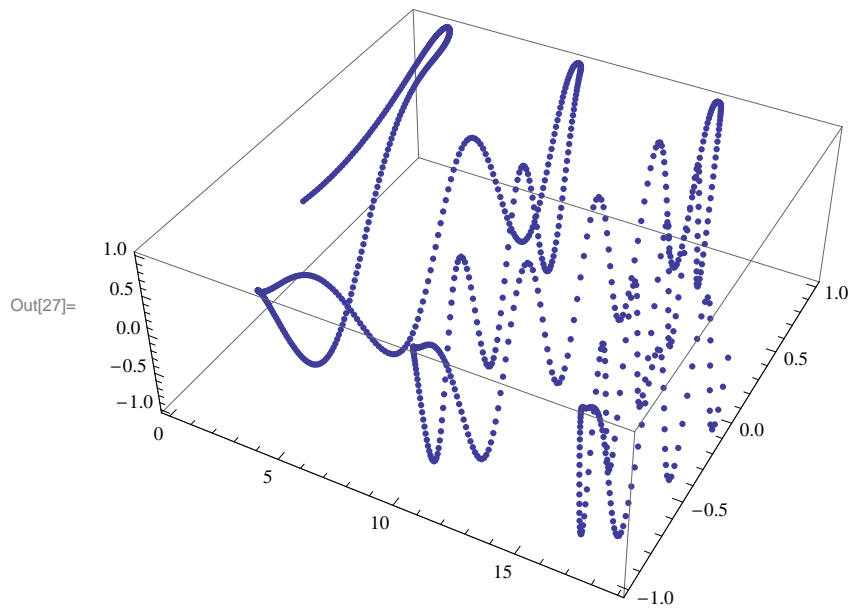
```
In[24]:= data = Table[{x, Sin[x]}, {x, 0, 6 Pi, 0.02}];
ListPlot[data]
```

```
Out[25]=
```



With an easy replacement rule, I generate a list of triple, where starting from the couple {x, y} the third value is a function of the first two

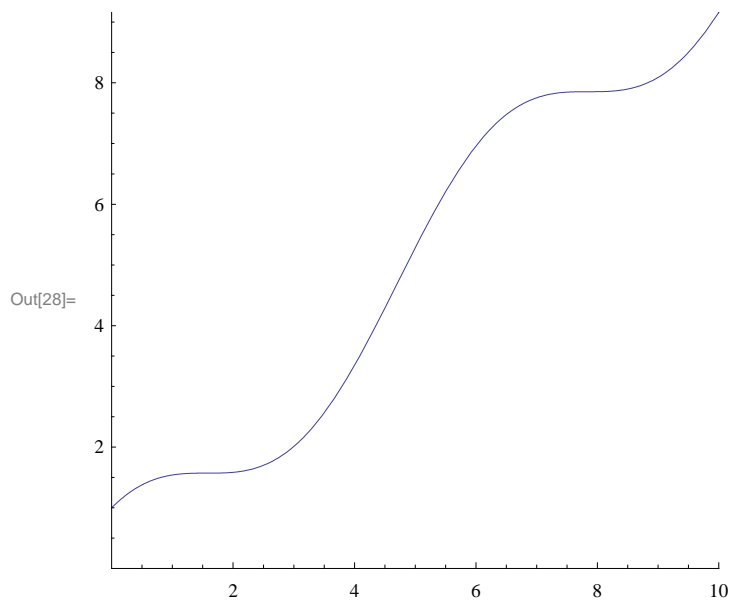
```
In[26]:= data = ReplaceAll[data, {x_, y_} -> {x, y, Sin[xy]}];
ListPointPlot3D[data]
```



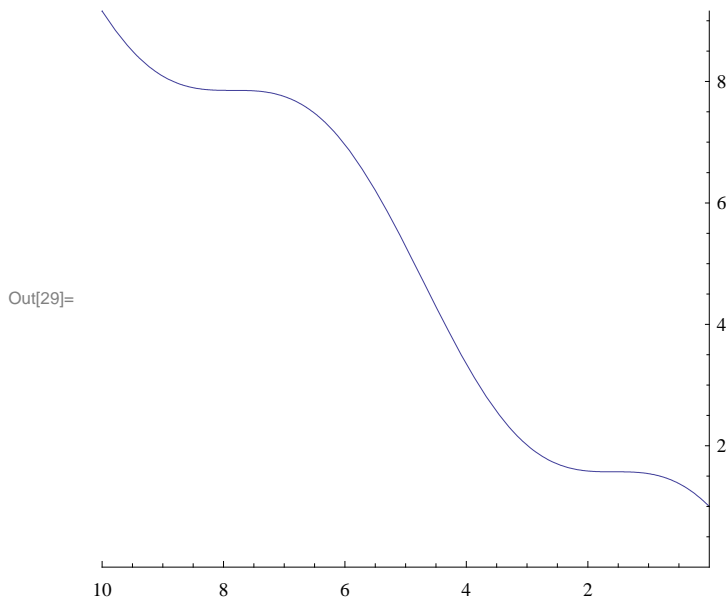
A more useful example

In astronomical field, sometimes it is needed to plot functions with reversed x axes (x values decreases from left to right). The flexibility of *Mathematica* allows to obtain this kind of result without to increase the complexity or the number of functions already built-in.

```
In[28]:= s = FullGraphics[Plot[Cos[x] + x, {x, 0, 10}]]
```



In[29]:= `ReplaceAll[s, {x_Real, y_Real} → {-x, y}]`



Dynamic interaction: a completely new feature in *Mathematica* 6

■ Examples

Here I generate two results, apparently identical: the solution of an equation with three parameters and the "Dynamic" version of the same solution

In[30]:= `Panel[Row[{Solve[a x^2 + b x + c == 0, x], Spacer[50], Dynamic[Solve[a x^2 + b x + c == 0, x]]}]]`

Out[30]=

$$\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\} \quad \{\{x \rightarrow 2\}\}$$

Now we can assign any value to one or more of the variable included in the solution, and see how the above cell change accordingly

In[31]:= `{a = 0, b = 1, c = -2}`

Out[31]= `{0, 1, -2}`

As you have seen, the first result is static, while the second one changes when a/b/c are changed. We can use some facilities to set their values. For instance a SetterBar

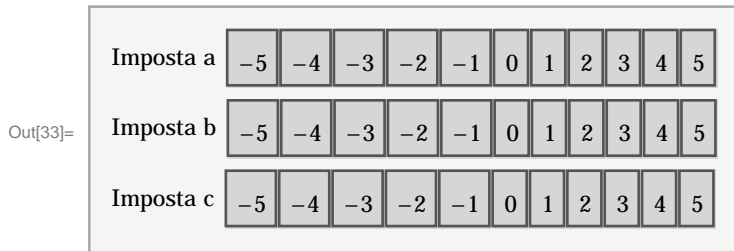
In[32]:= `SetterBar[Dynamic[a], {-1, 0, 1}]`

Out[32]=

| | | |
|----|---|---|
| -1 | 0 | 1 |
|----|---|---|

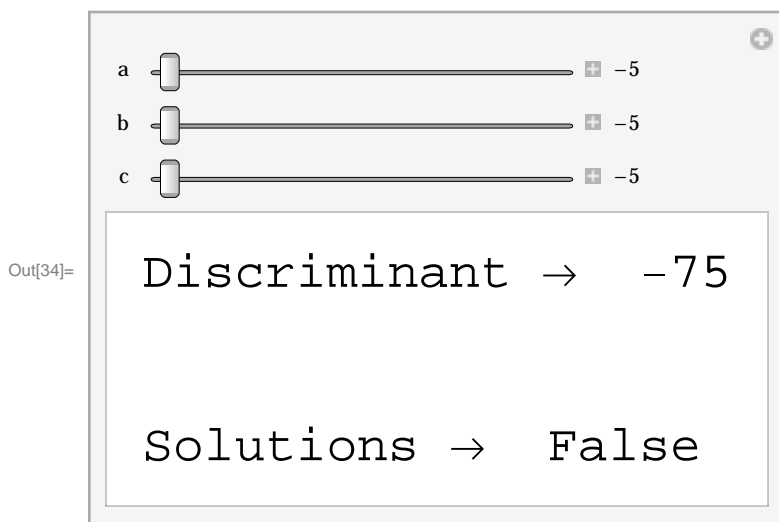
Here I arrange three controls, with SetterBar, one for each parameter

```
In[33]:= Panel[
  Column[
    {Row[{"Imposta a ", SetterBar[Dynamic[a], Range[-5, 5]]}],
      Row[{"Imposta b ", SetterBar[Dynamic[b], Range[-5, 5]]}],
      Row[{"Imposta c ", SetterBar[Dynamic[c], Range[-5, 5]]}]]]
```



Manipulate is a "macro" that allow an immediate generation of a special box where parameters can be manipulated by controllers.

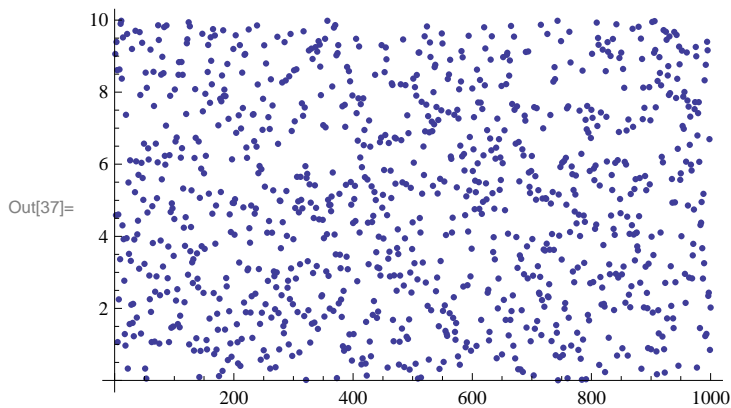
```
In[34]:= Manipulate[
  Style[Column[{Row[{"Discriminant → ", ToString[b^2 - 4 a c]}],
    Row[{"Solutions → ", Reduce[a x^2 + b x + c == 0, x, Reals]}], Spacings → 2], Large],
  {a, -5, 5, 1, Appearance → "Labeled"}, {b, -5, 5, 1, Appearance → "Labeled"},
  {c, -5, 5, 1, Appearance → "Labeled"}]
```



Another nice and useful new feature is Mouseover. It allow to overlap two objects and shows the first or the second depending on the position of the mouse pointer (outside or inside the object's box).

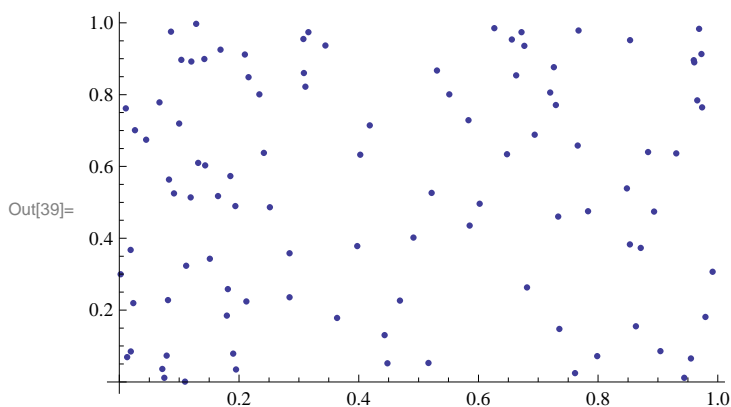
Here is a *double face* graphic, representing a ListPlot of a random data set and the BinCounts of its values. Mmove the mouse pointer inside/outside the image to see the "side B"

```
In[35]:= data = RandomReal[{0, 10}, 1000];
Needs["BarCharts`"];
Mouseover[ListPlot[data], BarChart[BinCounts[data, {0, 10, 1}]]]
```



Another example

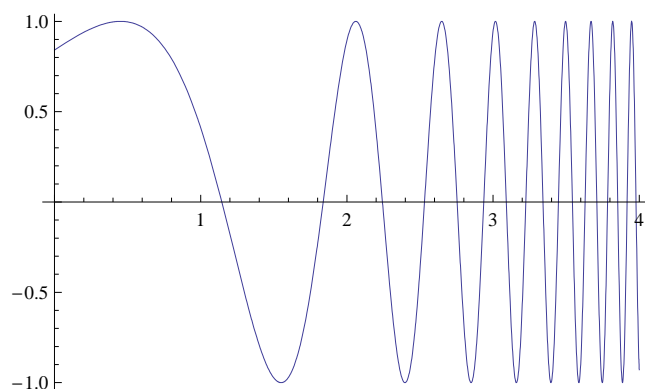
```
In[38]:= data = RandomReal[1, {100, 2}];
Mouseover[ListPlot[data],
  Tooltip[ListLinePlot[data[[Last@FindShortestTour[data]]], "Percorso minimo"]]
```



Note that in just one line of code we have used random number generation, a graphic function, a quite complex algorithm finding the shortest tour from a set of points and an interactivity function controlling the mouse position. This is an easy and intuitive example of *Mathematica* as an integrated system. The same philosophy for several different tasks and operations.

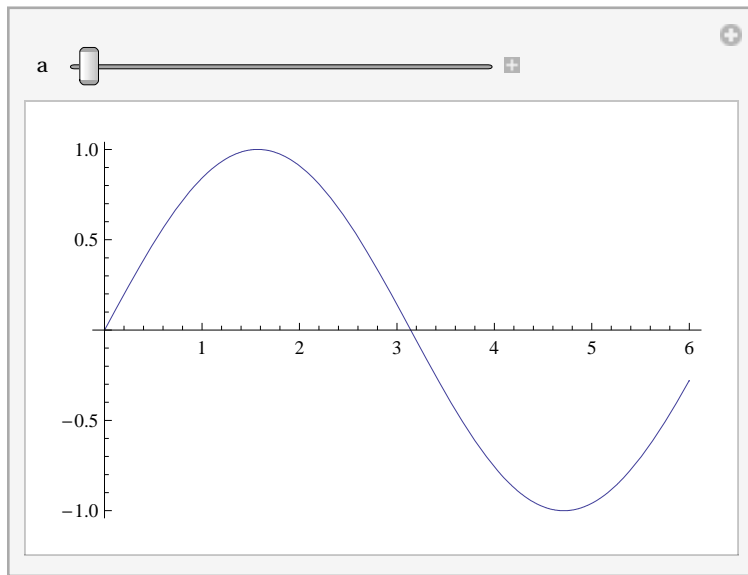
2D and 3D graphics: some examples and new features

```
Plot[Sin[e^x], {x, 0, 4}]
```



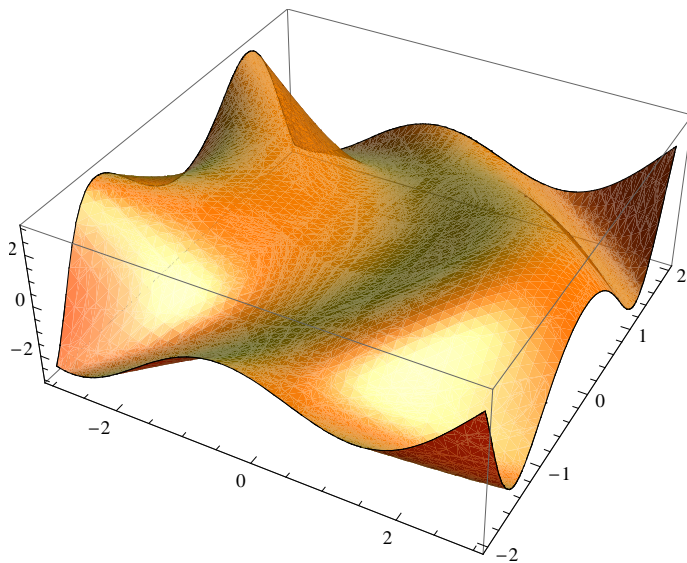
We can easily manipulate graphics using parameters and manipulators

```
Manipulate[Plot[Sin[x (1 + a x)], {x, 0, 6}], {a, 0, 2}]
```



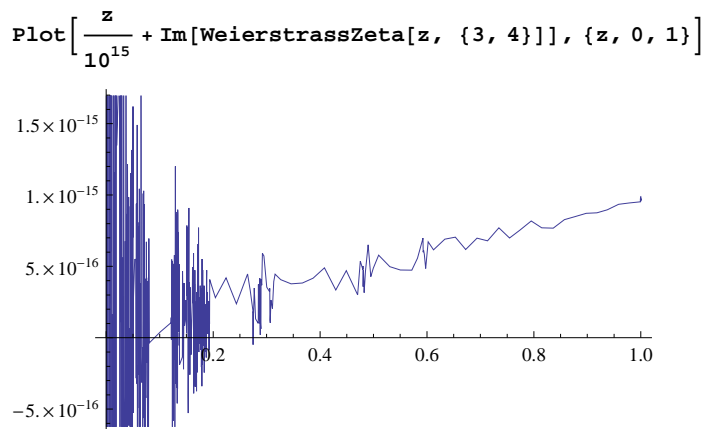
Functions of two variables

```
Plot3D[x Sin[x + y^2], {x, -3, 3}, {y, -2, 2},  
PlotStyle -> {Orange, Specularity[White, 20]}, Mesh -> None, PlotPoints -> 50]
```

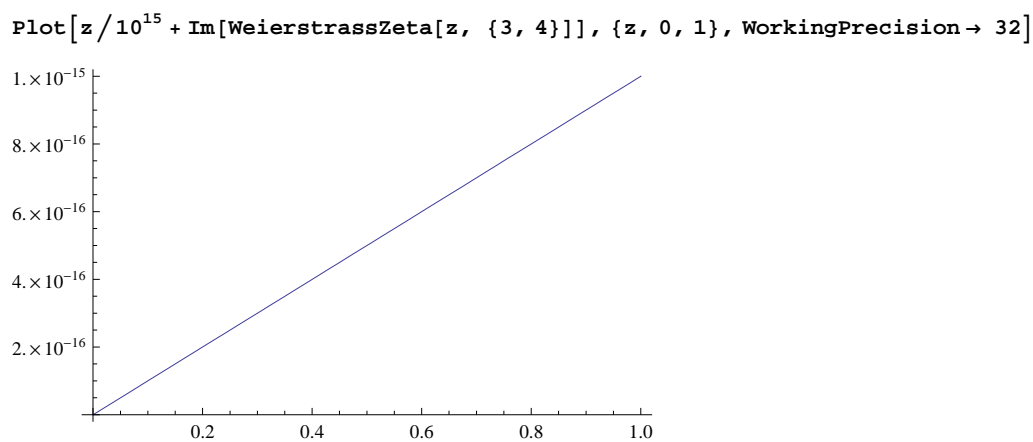


Mathematica's plot functions are now fully numeric and as a result allow the specification of `WorkingPrecision`. This will help clean up plots that would otherwise display some noise.

In this example the imaginary part of the `WeierstrassZeta` function is zero on the specified interval but numerical error creeps in leading to a lot of noise in the plot.



Setting the `WorkingPrecision` to 32 corrects the situation.

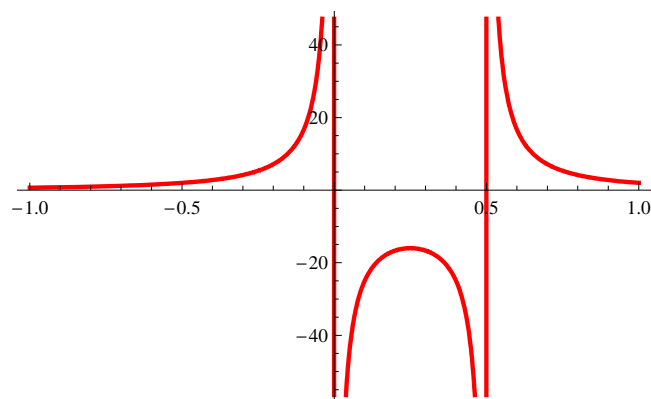


Plot: Singularity Removal

If a function has non-removable singularities at points x_i , the vertical lines that would normally appear in the plot can be avoided by including those points in the plot specification. This is similar to having `NIntegrate` pay special attention to individual points when integrating.

This function has singularities at 0 and $\frac{1}{2}$.

`Plot[$\frac{1}{x(x - \frac{1}{2})}$, {x, -1, 1}, PlotStyle → {Red, Thick}]`

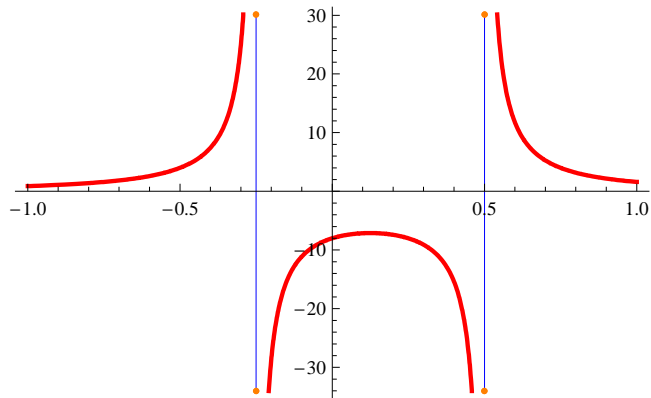


If we specify the exclusion values, we can also specify a style to use for those values. Giving the two colors, blue and

orange, creates a blue line with orange points at the end.

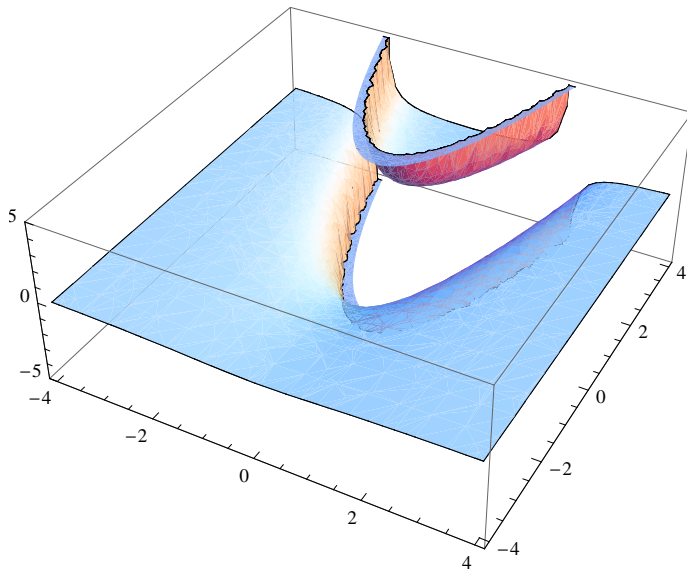
Here we specify the locations of the singularities. In this case the excluded values are indicated by blue lines with orange points at the minimum and maximum values.

```
Plot[ $\frac{1}{(x + \frac{1}{4})(x - \frac{1}{2})}$ , {x, -1, 1}, Exclusions -> {x == - $\frac{1}{4}$ , x ==  $\frac{1}{2}$ },  
ExclusionsStyle -> {Blue, Orange}, PlotStyle -> {{Red, Thick}}]
```

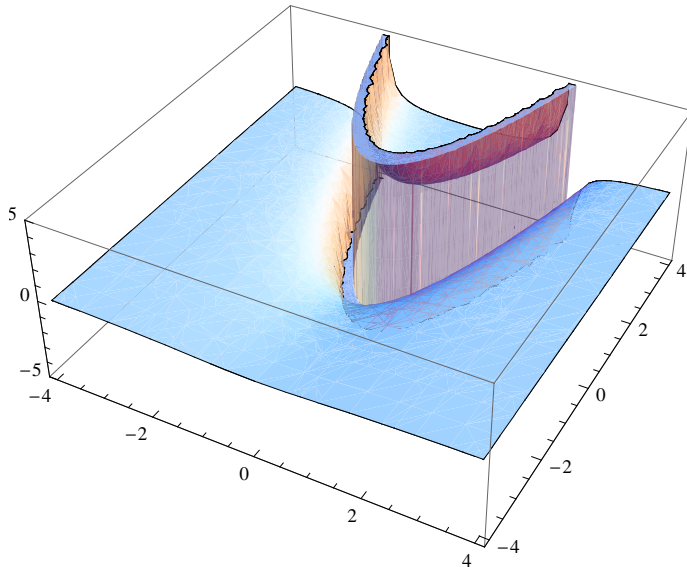


Similar functions are available for 3D graphics

```
Plot3D[ $\frac{2}{y - x^2}$ , {x, -4, 4}, {y, -4, 4}, Mesh -> None, PlotRange -> 5, Exclusions -> {y == x^2}]
```



```
Plot3D[ $\frac{2}{y-x^2}$ , {x, -4, 4}, {y, -4, 4}, Mesh → None, PlotRange → 5,
Exclusions → {y == x^2}, ExclusionsStyle → {Directive[Opacity[.5]]}]
```



Plot: Automatic Removal of Non-Real Values

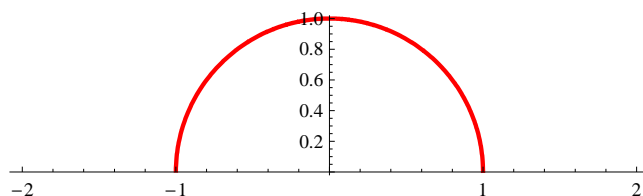
The visualization functions in *Mathematica* are defined to work with real-valued functions. Hence if a function becomes complex- or other-valued at some point, those values are excluded. With the adaptive sampling algorithm built into these functions, you will usually get fairly close to the real-valued barrier.

Here is an example of a function that returns complex values on the specified domain.

```
Table[ $\sqrt{1-x^2}$ , {x, -2, 2, 0.1}]
```

```
{0. + 1.73205 i, 0. + 1.61555 i, 0. + 1.49666 i, 0. + 1.37477 i, 0. + 1.249 i,
0. + 1.11803 i, 0. + 0.979796 i, 0. + 0.830662 i, 0. + 0.663325 i, 0. + 0.458258 i, 0.,
0.43589, 0.6, 0.714143, 0.8, 0.866025, 0.916515, 0.953939, 0.979796, 0.994987, 1.,
0.994987, 0.979796, 0.953939, 0.916515, 0.866025, 0.8, 0.714143, 0.6, 0.43589,
0. + 2.10734 × 10-8 i, 0. + 0.458258 i, 0. + 0.663325 i, 0. + 0.830662 i, 0. + 0.979796 i,
0. + 1.11803 i, 0. + 1.249 i, 0. + 1.37477 i, 0. + 1.49666 i, 0. + 1.61555 i, 0. + 1.73205 i}
```

```
Plot[ $\sqrt{1-x^2}$ , {x, -2, 2}, PlotStyle → {{Red, Thick}}, AspectRatio → Automatic]
```

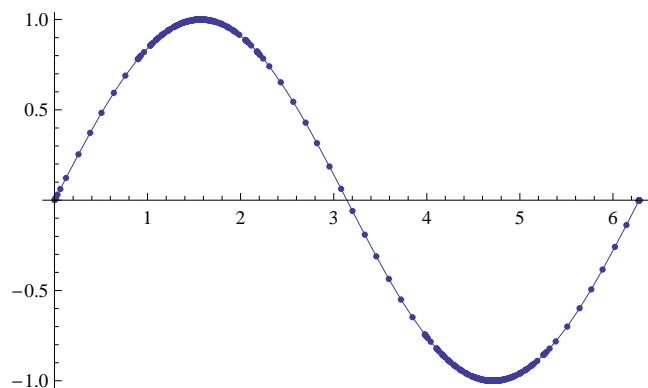


Plot: Adaptive Refinement

Mathematica uses an adaptive algorithm to determine how many points should be included in creating the plot. One of the determining factors is how many recursions are used to locate points used in the plot. As with other numeric functions, recursion leads to higher precision results.

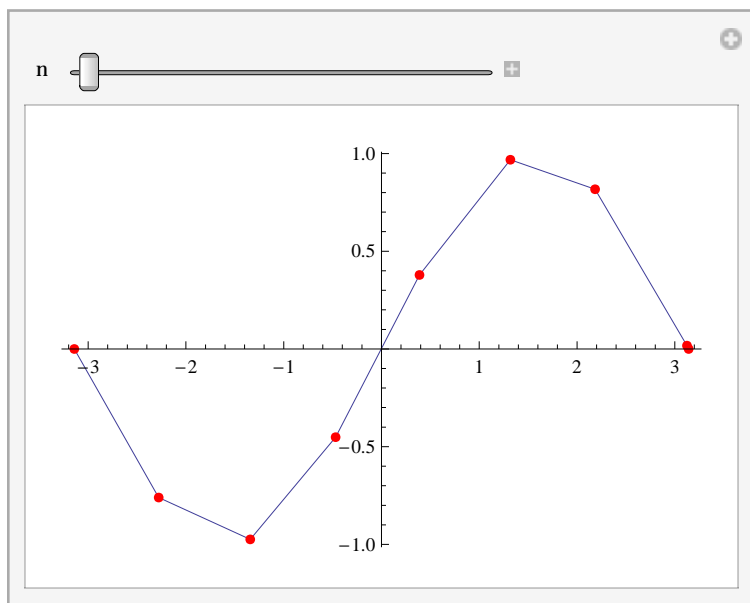
This shows all of the points used in the adaptive sampling routine for `Plot`.

```
Plot[Sin[t], {t, 0, 2  $\pi$ }, Mesh  $\rightarrow$  All]
```



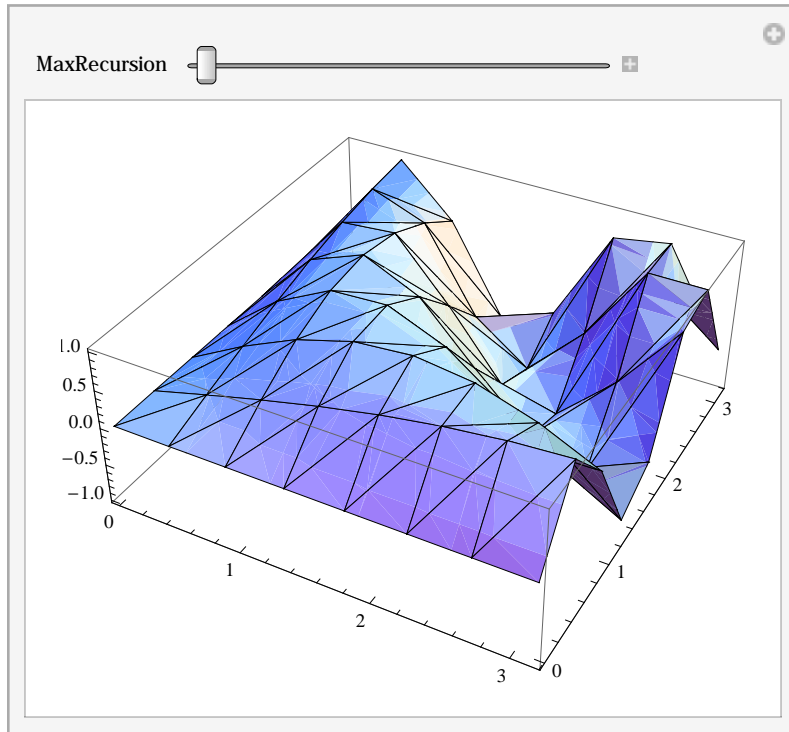
Adaptive refinement requires repeated sampling of the function which is done recursively. This `Manipulate` illustrates the process.

```
Manipulate[
Plot[Sin[x], {x, - $\pi$ ,  $\pi$ },
  MaxRecursion  $\rightarrow$  n,
  Mesh  $\rightarrow$  All, MeshStyle  $\rightarrow$  {Red, PointSize[0.015]}, PlotPoints  $\rightarrow$  8], {n, 0, 10, 1}]
```



Here is the same for 3D

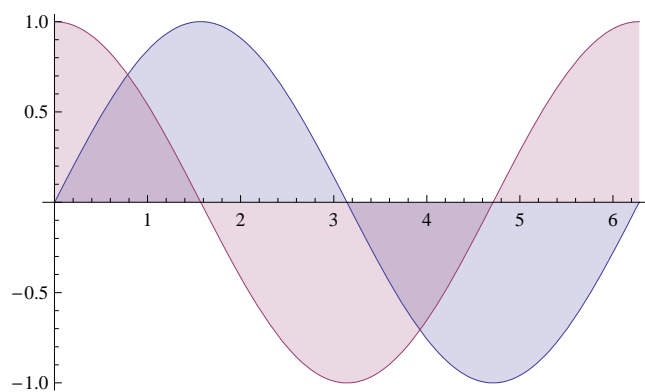
```
Manipulate[
Plot3D[Sin[x y], {x, 0,  $\pi$ }, {y, 0,  $\pi$ }, MaxRecursion  $\rightarrow$  r, PlotPoints  $\rightarrow$  8, Mesh  $\rightarrow$  All],
{{r, 0, "MaxRecursion"}, 0, 10, 1}]
```



Plot: Filling

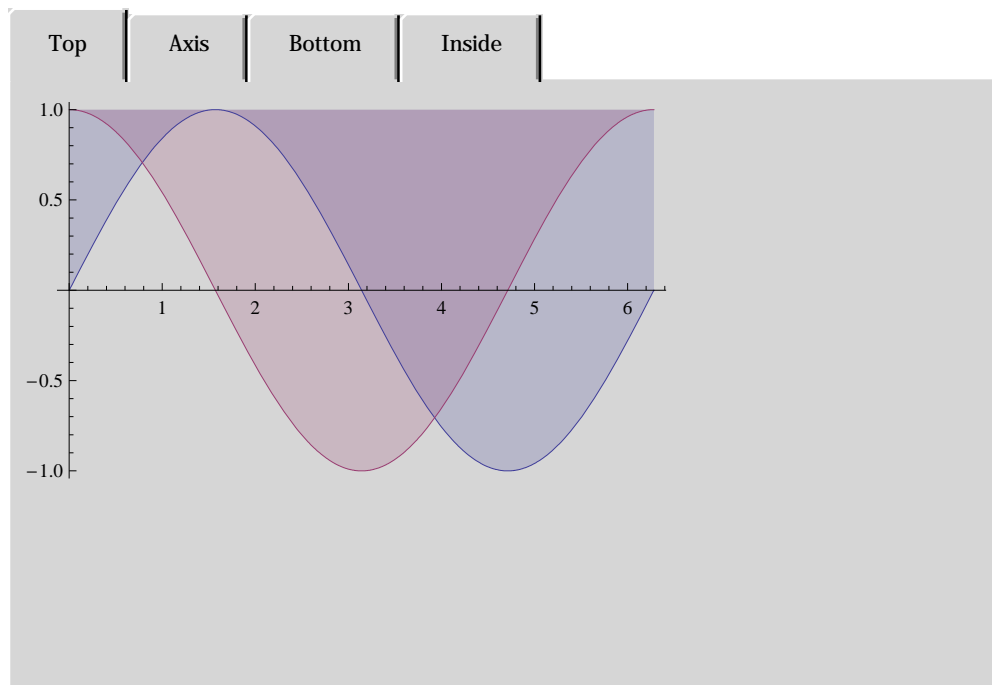
A new option, `Filling`, can be used to shade regions of a plot. This option essentially replaces the functionality that was present in the pre-Version 6 package, `Graphics`FilledPlot``.

```
Plot[{Sin[x], Cos[x]}, {x, 0,  $2\pi$ }, Filling  $\rightarrow$  Automatic]
```



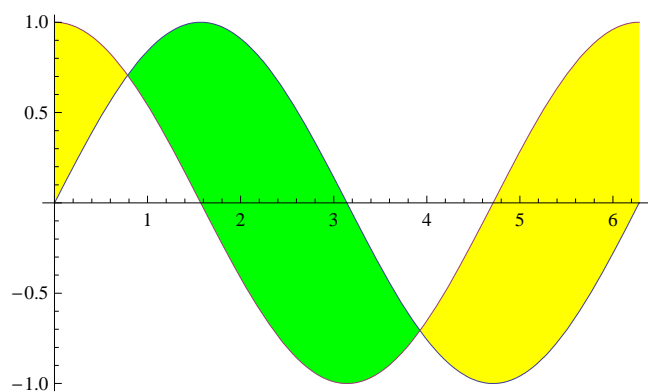
You can fill to the horizontal axis, or the top, or bottom of the plot.

```
TabView[{"Top" -> Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ }, Filling -> Top],
  "Axis" -> Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ }, Filling -> Axis],
  "Bottom" -> Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ }, Filling -> Bottom],
  "Inside" -> Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ }, Filling -> {1 -> {2}}]}]
```



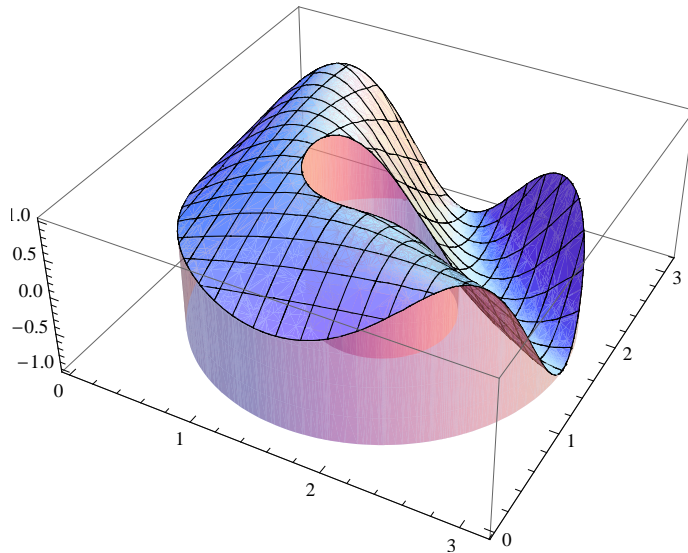
We add a different style in this example with the option `FillingStyle`. The colors change depending on which function value is greater.

```
Plot[{Sin[x], Cos[x]}, {x, 0, 2  $\pi$ }, Filling -> {1 -> {2}}, FillingStyle -> {Yellow, Green}]
```



Here is a 3D equivalent

```
Plot3D[Sin[x y], {x, 0, π}, {y, 0, π},
  RegionFunction -> (0.3 ≤ (#1 -  $\frac{\pi}{2}$ )2 + (#2 -  $\frac{\pi}{2}$ )2 ≤ 2 &), Filling -> Bottom]
```



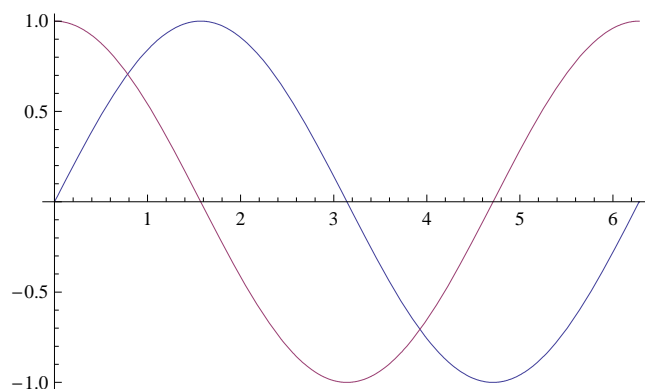
Tooltip and MouseOver

There is a new function, `Tooltip`, which can be used as a wrapper for the plot arguments. The argument to `Tooltip` will display when the mouse passes over the curves in the plot. You can use `Tooltip` to distinguish between vectors as in this example.

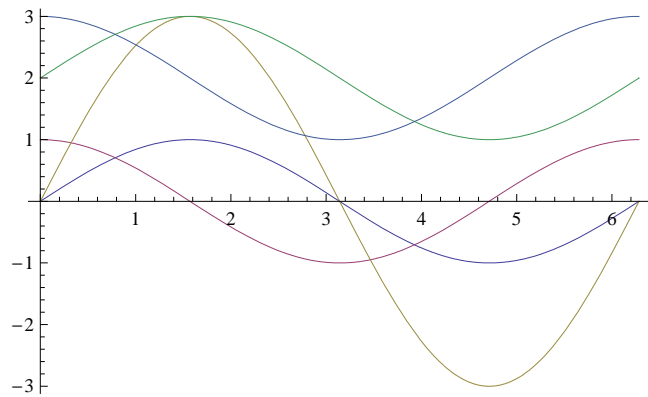
? `Tooltip`

`Tooltip[expr, label]` displays *label* as a tooltip while the mouse pointer is in the area where *expr* is displayed. >>

```
Plot[{Tooltip[Sin[x]], Tooltip[Cos[x]]}, {x, 0, 2 Pi}]
```

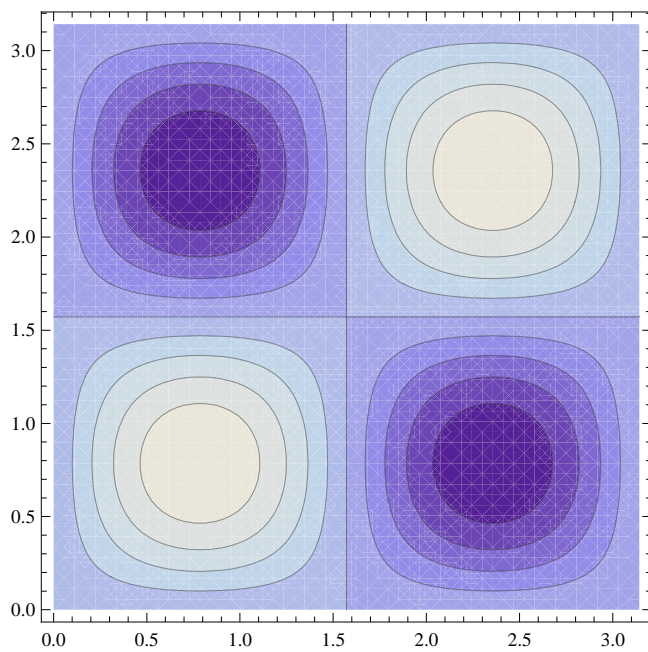


```
Plot[Evaluate[Map[Tooltip[#, TraditionalForm[#]] &,
  {Sin[x], Cos[x], 3 Sin[x], Sin[x] + 2, Cos[x] + 2}]], {x, 0, 2 π}]
```



ContourPlot show automatically Tooltips for contour lines

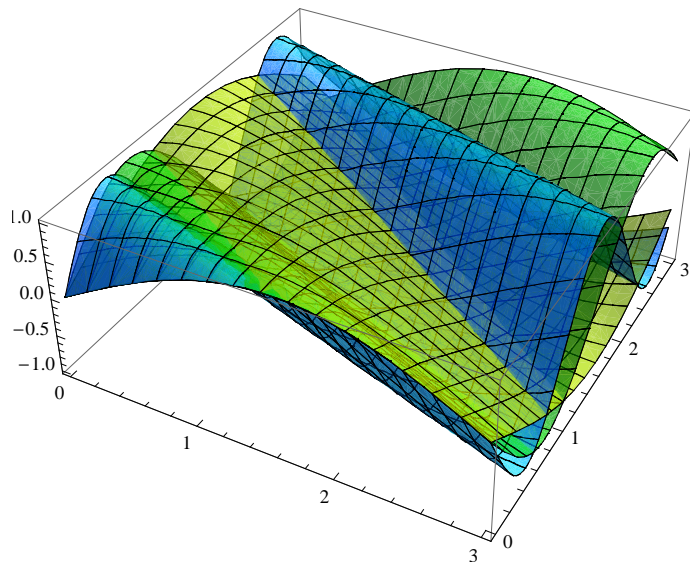
```
ContourPlot[Sin[2 x] Sin[2 y], {x, 0, π}, {y, 0, π}, ContourLabels → None]
```



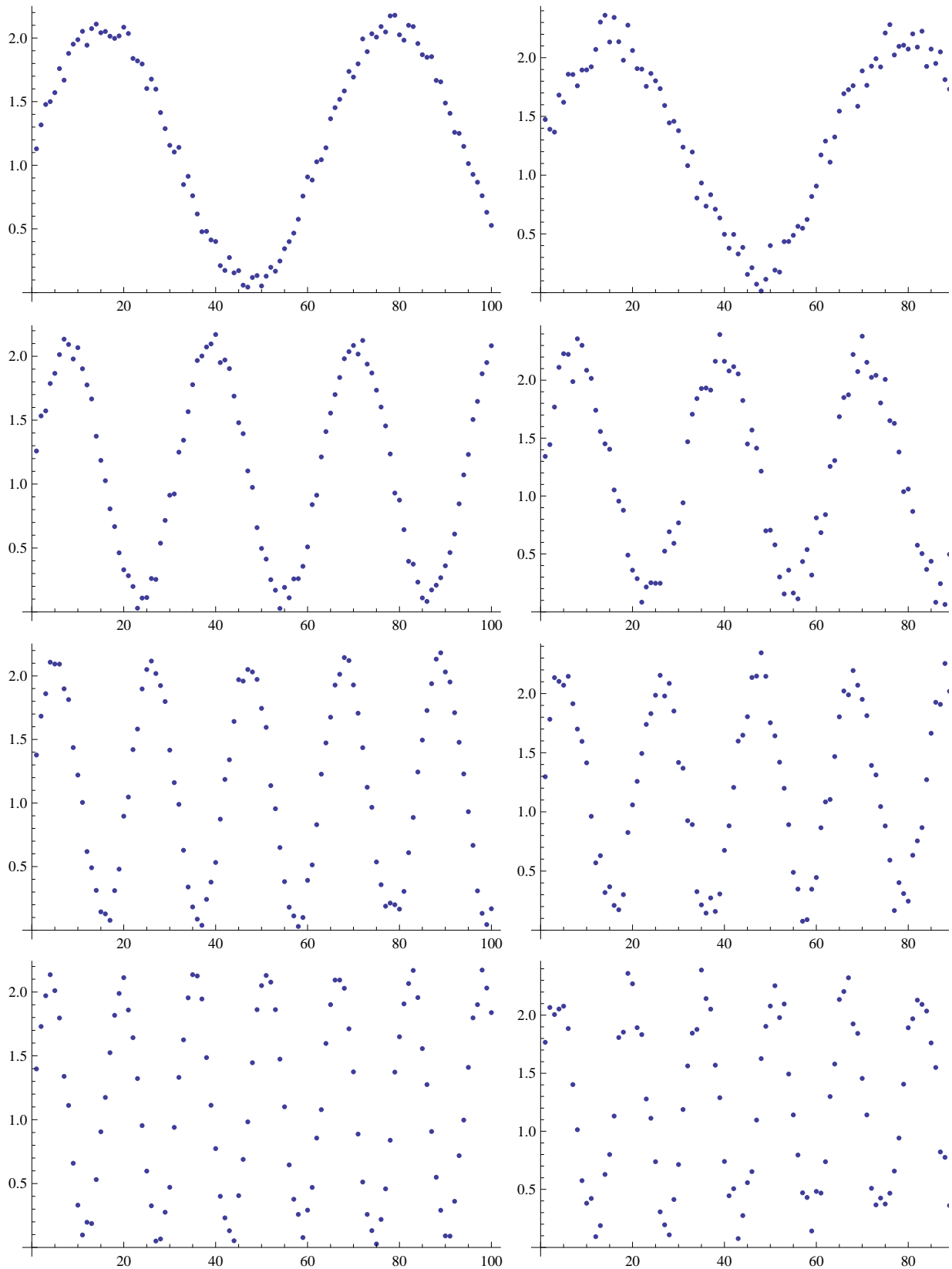
```

Plot3D[Evaluate[Table[Tooltip[Sin[x + ny], Plot[Sin[n x],
  {x, 0, 2 Pi}, Filling -> Axis, FillingStyle -> Pink, ImageSize -> 200]], {n, 3}]],
  {x, 0, 3}, {y, 0, 3}, PlotStyle -> Table[Opacity[.7, Hue[i / 6]], {i, 4}]]

```



```
Grid[Table[Mouseover[data = Table[1 + Sin[i r / 10], {r, 1, 100}] + RandomReal[t / 5, 100];
  ListPlot[data, PlotRange -> All, AxesOrigin -> {0, 0}], ListPlot[Abs[Fourier[data]],
  PlotRange -> All, Joined -> True], ImageSize -> All], {i, 1, 4}, {t, 1, 4}]
```



Graphics fully integrated into the whole system, just like any other object

Coin tossing experiments...

$$\text{coins} = \left\{ \text{1 Euro coin}, \text{2 Euro coin} \right\}$$

```
Button["Toss the coin", Print[coins[[RandomChoice[{1, 2}]]]]]
```

Toss the coin

```
t = Apply[Plus, (coins - 4)^2]
```

$$\left(-4 + \text{2 Euro coin} \right)^2 + \left(-4 + \text{1 Euro coin} \right)^2$$

```
t = Expand[%]
```

$$32 - 8 \text{ 2 Euro coin} + \left(\text{2 Euro coin} \right)^2 - 8 \text{ 1 Euro coin} + \left(\text{1 Euro coin} \right)^2$$

```
s = Apply[Plus, coins^2]
```

$$\left(\text{2 Euro coin} \right)^2 + \left(\text{1 Euro coin} \right)^2$$

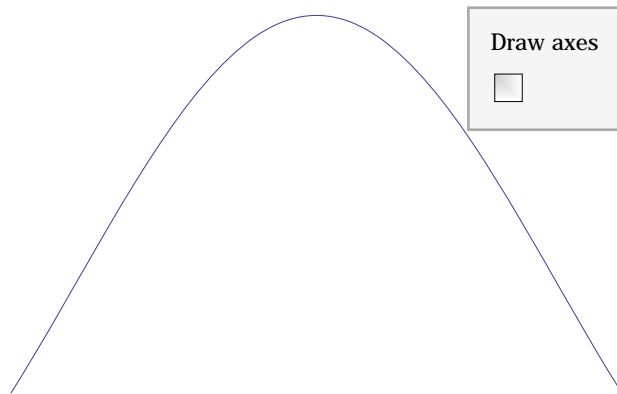
```
Solve[{t == 0, s == 1}, coins]
```

$$\left\{ \left\{ \text{1 Euro coin} \rightarrow \frac{33}{16} - \frac{31 i}{16}, \text{2 Euro coin} \rightarrow \frac{33}{16} + \frac{31 i}{16} \right\}, \right.$$

$$\left. \left\{ \left\{ \text{1 Euro coin} \rightarrow \frac{33}{16} + \frac{31 i}{16}, \text{2 Euro coin} \rightarrow \frac{33}{16} - \frac{31 i}{16} \right\} \right\} \right.$$

```
In[1]:= Dynamic[Plot[Cos[x], {x, -2, 2}, ImagePadding -> 1, Axes -> assi,
  Epilog -> Inset[Panel[Column[{"Draw axes", Checkbox[Dynamic[assi]]}], {1.5, .8}]]]
```

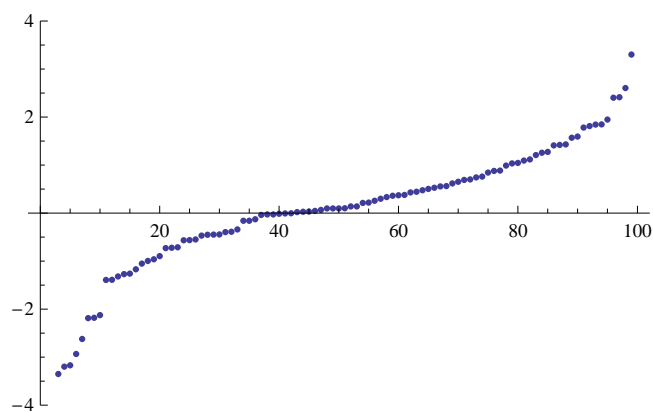
Out[1]=



Instantly dynamic

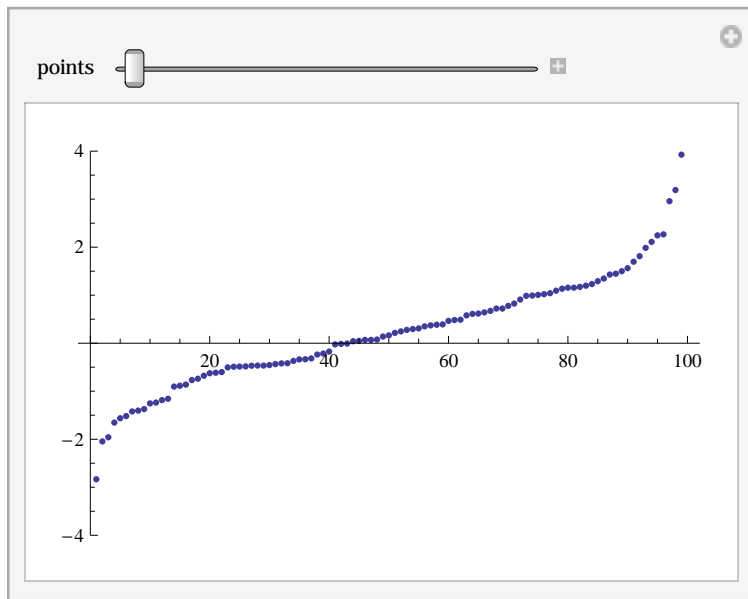
■ Interact with anything that makes sense

```
ListPlot[Sort[RandomReal[StudentTDistribution[5], {100}]], PlotRange -> {-4, 4}]
```



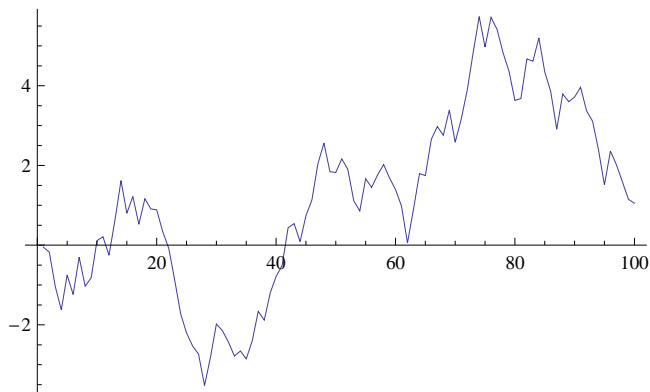
Adding interaction:


```
Manipulate[ListPlot[Sort[RandomReal[StudentTDistribution[5], {points}]],
  PlotRange → {-4, 4}], {points, 100, 1000, 10}]
```

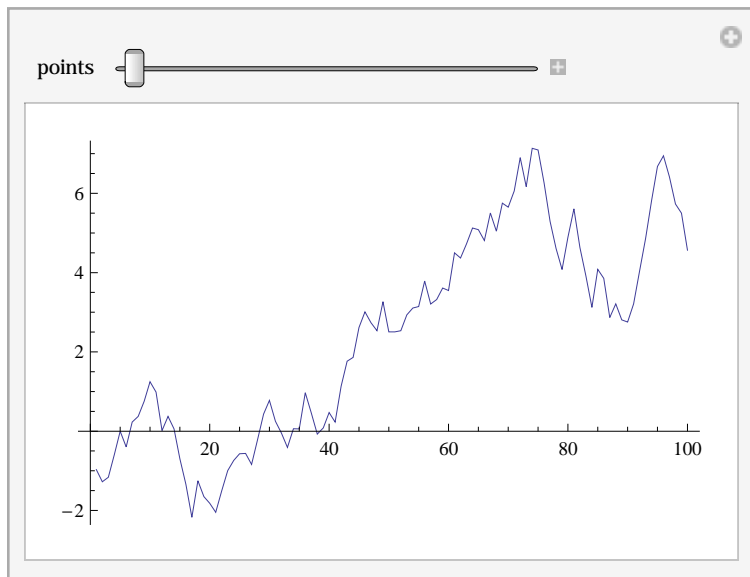


A random walk

```
ListLinePlot[Accumulate[RandomReal[{-1, 1}, 100]]]
```



```
Manipulate[ListLinePlot[Accumulate[RandomReal[{-1, 1}, points]]], {points, 100, 1000, 10}]
```

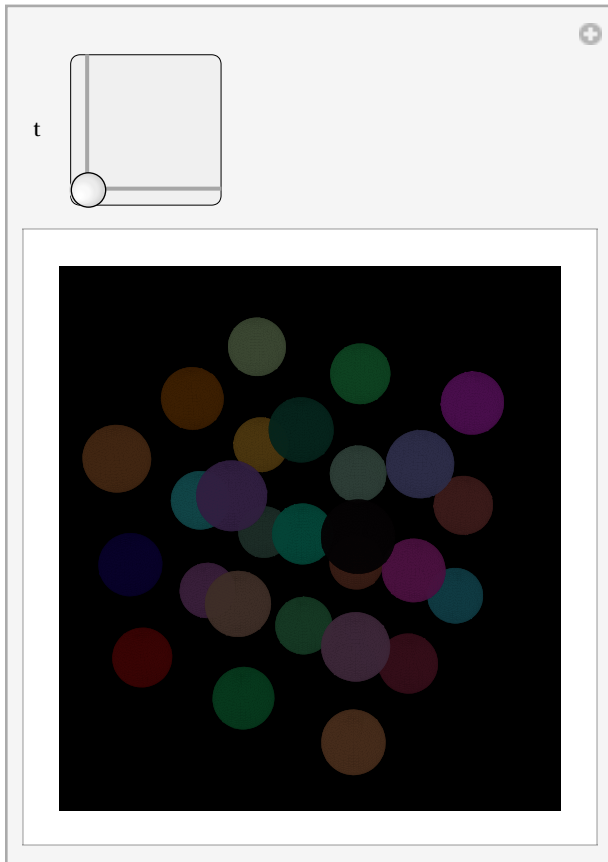


Fast response is needed for instant interactivity

■ Real-time lighting changes

```
In[2]:= myspheres = Table[{RGBColor[Random[], Random[], Random[]], Specularity[White, 128],
  Sphere[{x, y, z}, 1]}, {x, 0, 10, 4}, {y, 0, 10, 4}, {z, 0, 10, 4}];
Manipulate[Graphics3D[{White, PointSize[.02], Point[{t[[1]], t[[2]], 5}], myspheres},
  Background → Black, Boxed → False,
  Lighting → {RGBColor[.3, .3, .3], {White, {{{t[[1]], t[[2]], 5}, {0, 0, 0}}, 2}}},
  PlotRange → {{-1, 10}, {-1, 10}, {-1, 10}}, ImageSize → 500,
  {t, {-15, -15}, {20, 20}}, SaveDefinitions → True]
```

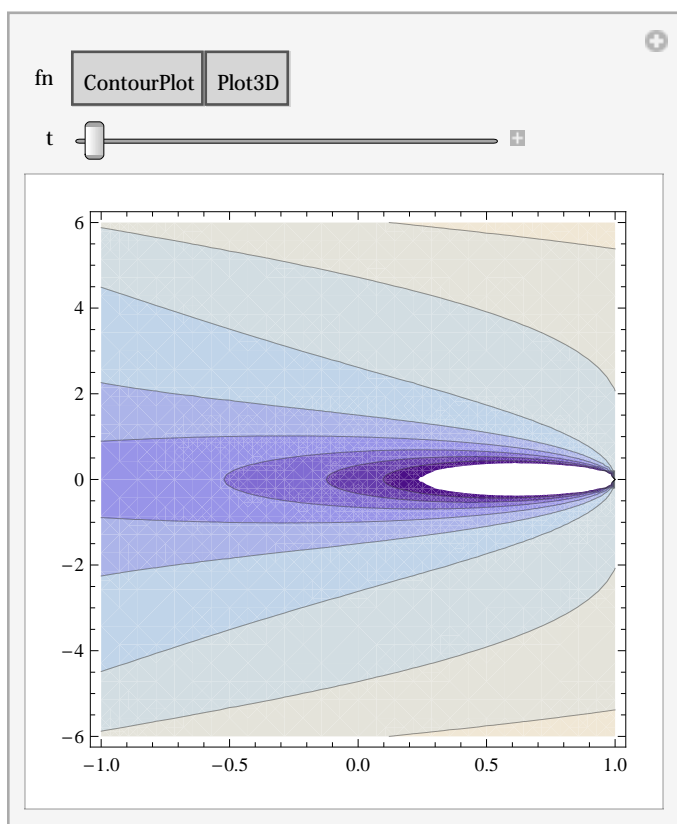
Out[2]=



■ Draft quality for expansive calculations

```
In[3]:= Manipulate[fn[Re[Zeta[x + I y]], {x, -t, 1}, {y, -6, 6}, ImageSize -> 500],
  {fn, {ContourPlot, Plot3D}}, {t, 1, 20}]
```

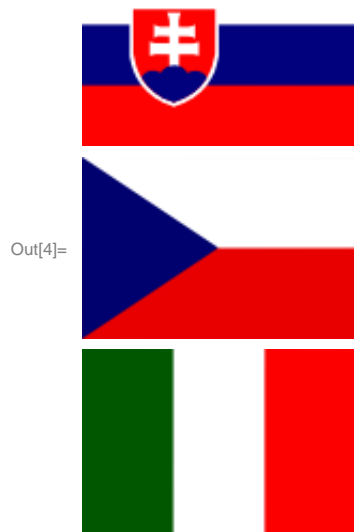
Out[3]=



Integrated data sources

Mathematica gives seamless immediate access to an ever-growing library of carefully curated and continually updated data maintained by Wolfram Research—all with a unified interface allowing complete interoperability and tight integration into *Mathematica* and its computational and data presentation capabilities.

```
In[4]:= Column[{Tooltip[CountryData["Slovakia", "Flag"], "Slovakia"],
  Tooltip[CountryData["CzechRepublic", "Flag"], "Czech Republic"],
  Tooltip[CountryData["Italy", "Flag"], "Italy"]}]]
```

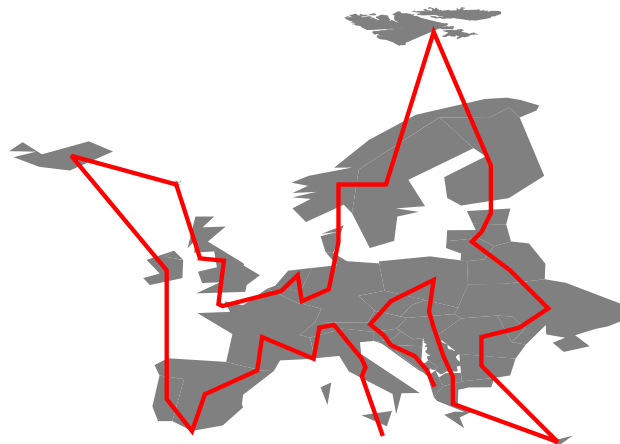


```
In[5]:= GraphPlot[Flatten[Thread[# -> CountryData[#, "BorderingCountries"]] & /@
  CountryData["SouthAmerica"]], VertexLabeling -> True]
```



```
In[6]:= Graphics[{Gray, CountryData[#, "SchematicPolygon"] & /@ CountryData["Europe"],
  Thick, Red, Line[#[[Last[FindShortestTour[#]]]]] &[
    Reverse[CountryData[#, "CenterCoordinates"]] & /@ CountryData["Europe"]]]}]
```

Out[6]=



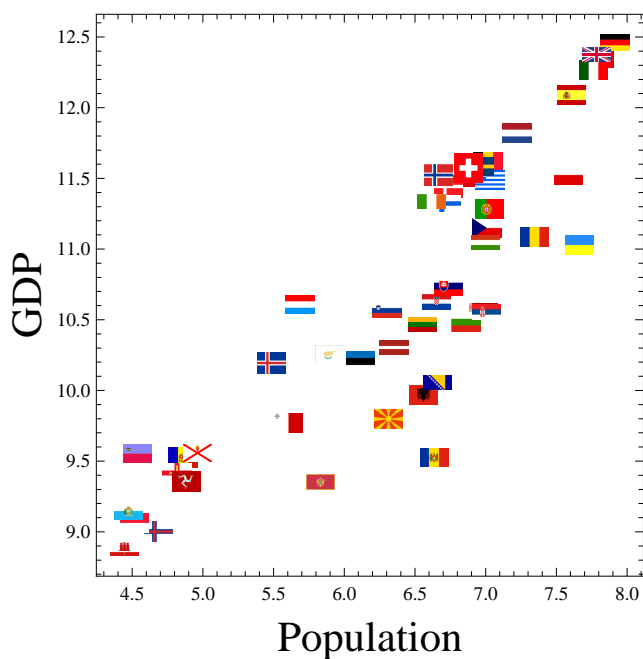
```
In[7]:= Button["Country data notebook",
  CreateDocument[CellGroup[{TextCell[CountryData[#, "Name"], "Section"],
    ExpressionCell[GraphicsRow[{CountryData[#, "Shape"],
      Quiet[DateListPlot[CountryData[#, {"Population"}, {1970, 2005}]]]],
      Frame -> All, ImageSize -> 450], "Output"]}], Closed] & /@ CountryData["Europe"],
  StyleDefinitions -> "Creative/NaturalColor.nb"], Method -> "Queued"]
```

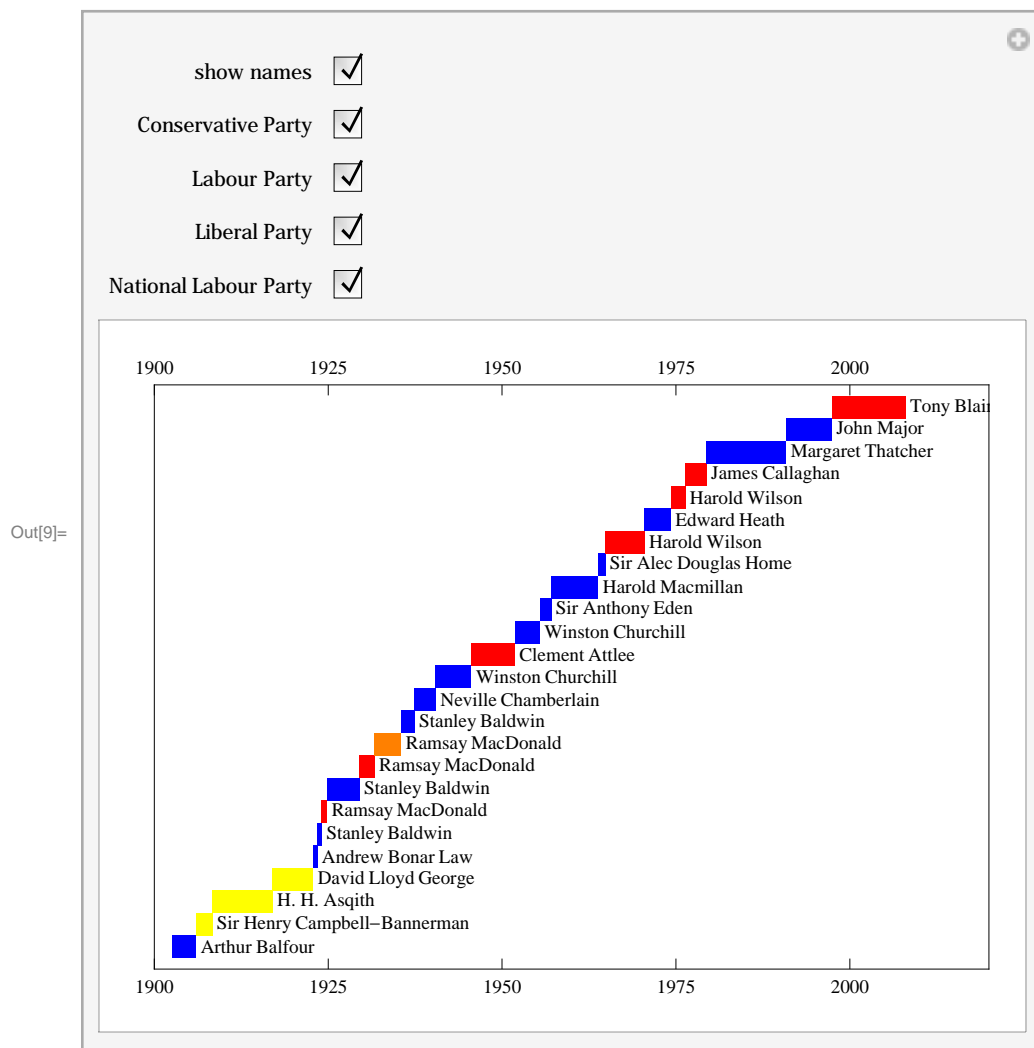
Out[7]=

Country data notebook

```
In[8]:= Graphics[Tooltip[Inset[CountryData[#, "Flag"], {Log[10, CountryData[#, "Population"]],
  Log[10, CountryData[#, "GDP"]]}, Center, 0.2], #] & /@
  DeleteCases[CountryData["Europe"], "VaticanCity" | "Svalbard"], Frame -> True,
  PlotRangePadding -> .2, FrameLabel -> {Style["Population", 20], Style["GDP", 20]}]
```

Out[8]=





Example of data analysis and dynamic interaction

Thanks to the complete integration of all *Mathematica*'s functionalities, we can analyse, manipulate and change data in just one box.

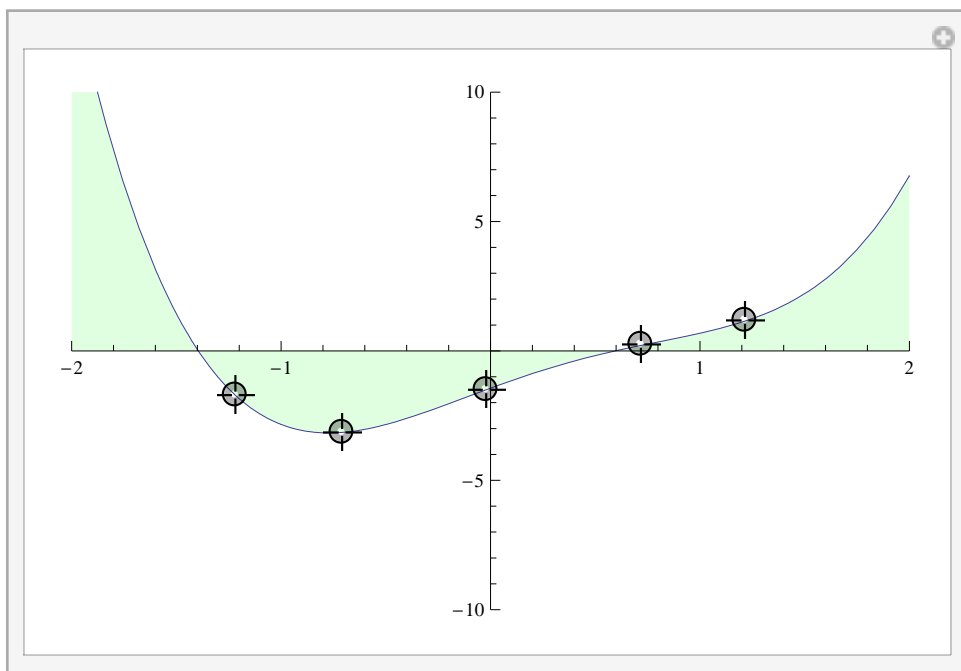
In this example there is a *InterpolatingPolynomial* applied on some starting points, then its graphic result and an interactive panel which let us to change "on the fly" pre-existing values or add new ones.

```

In[10]:= Manipulate[
  Plot[Evaluate[InterpolatingPolynomial[pts, x]], {x, -2, 2},
    PlotRange → {{-2, 2}, {-10, 10}}, Filling → Axis, FillingStyle → LightGreen,
    ImageSize → 400], {{pts, Table[{Random[], Random[]}, {5}]},
  {-2, -10}, {2, 10}, Locator, LocatorAutoCreate → True}]

```

Out[10]=



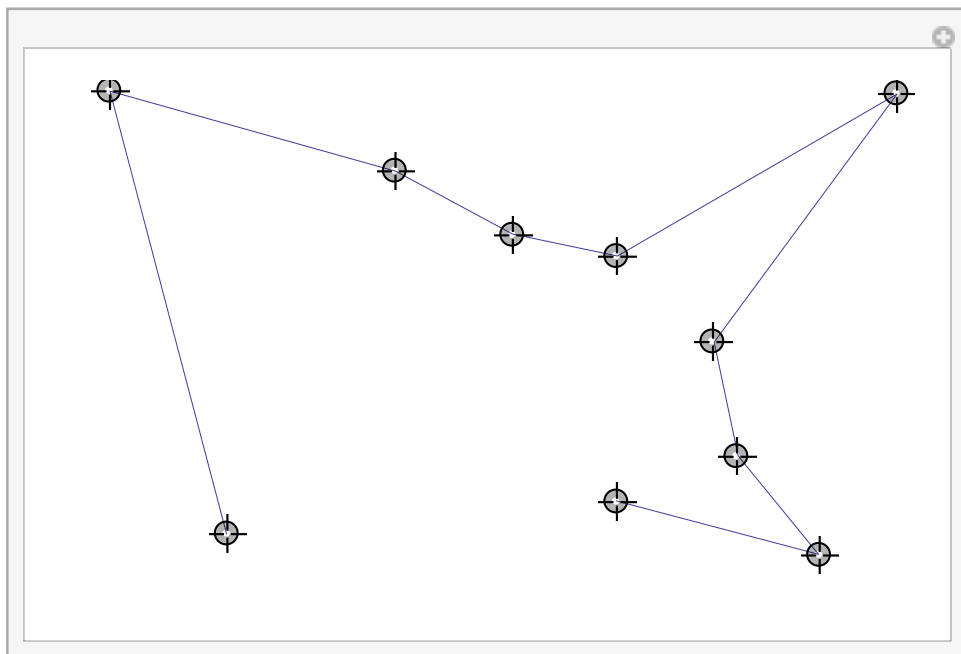
Here there is also the FindShortestTour included

```

In[11]:= Manipulate[ListLinePlot[pts[[Last[FindShortestTour[pts]]]], ImageSize → 400, Axes → False],
  {{pts, Table[{Random[], Random[]}, {10}]},
  {-2, -10}, {2, 10}, Locator, LocatorAutoCreate → True}]

```

Out[11]=



A descriptive Statistics Example

■ A Dataset

The following data are winning Maryland Pick-3 numbers gathered over a 32-week period in 1989 and 1990. The data are based on the Lottery data set from NIST's online Dataset Archives.

```
In[1]:= data = Import[ToFileName[NotebookDirectory[], "lotterydigits.txt"], "Table"];
```

There are 218 data elements represented as a list of digits.

```
In[2]:= Dimensions[data]
```

```
Out[2]= {218, 3}
```

Here are the first 5 winners.

```
In[3]:= data[[1 ;; 5]]
```

```
Out[3]= {{1, 6, 2}, {6, 7, 1}, {9, 3, 3}, {4, 1, 4}, {7, 8, 8}}
```

■ Some Qualitative Analyses

A question of interest may be: Were the Pick-3 results fair? That is to say, were the winning numbers roughly evenly distributed, or is there evidence that some numbers occurred with noticeably higher frequency.

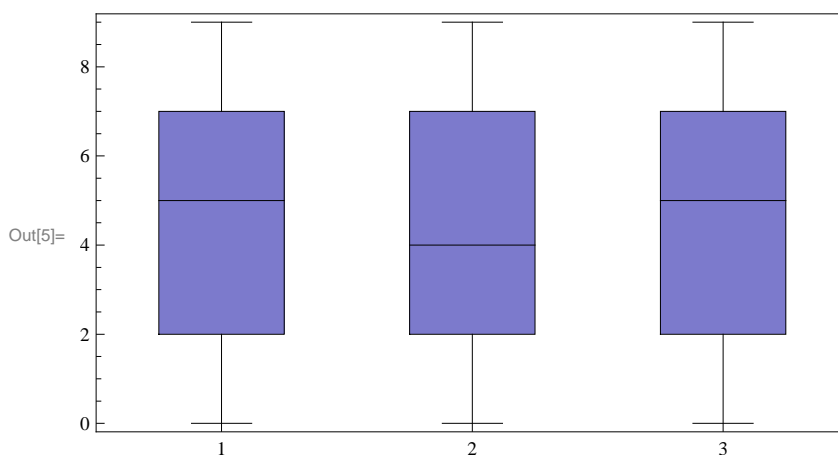
An initial qualitative investigation is likely to include some descriptive plots.

■ Distribution of Digits

```
In[4]:= << StatisticalPlots`
```

Box plots for the three digits might be used to access the location and spread of the three digits treated independently.

```
In[5]:= BoxWhiskerPlot[data]
```



Pie charts and bar charts could also be used to describe the frequency of individual digits.

```
In[6]:= << BarCharts`
```

```
In[7]:= << PieCharts`
```

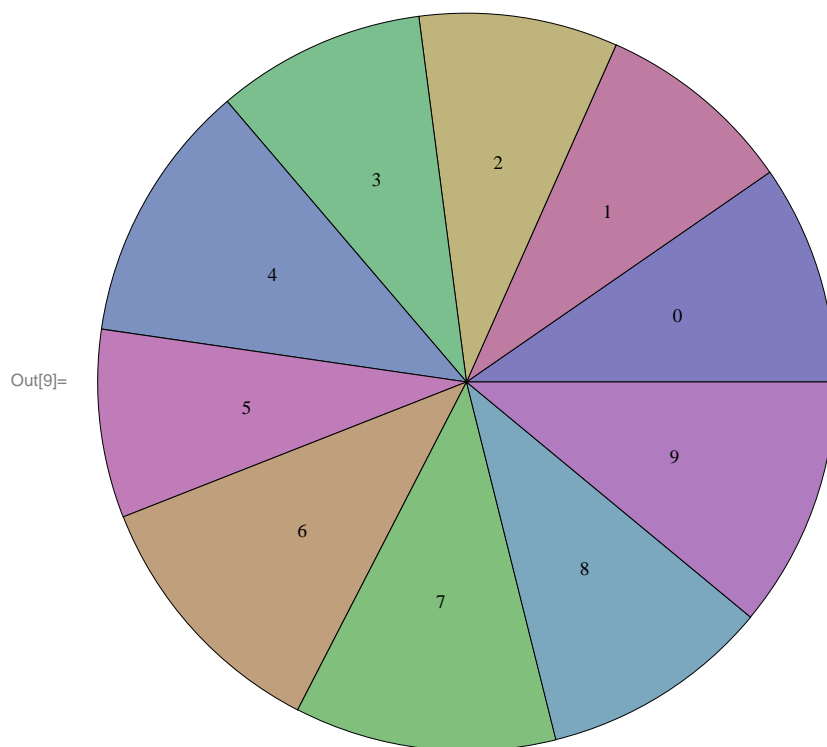
This gives a list of the counts for the first digits.

```
In[8]:= digit1 = Sort[Tally[data[[All, 1]]]]
```

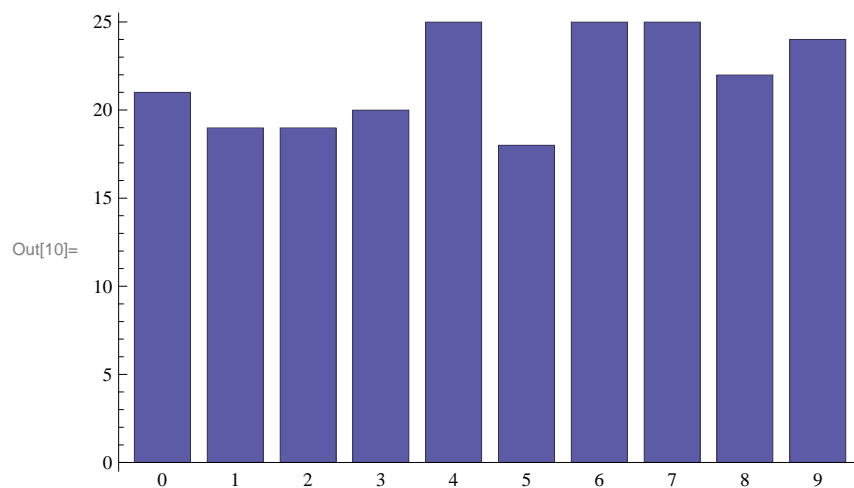
```
Out[8]= {{0, 21}, {1, 19}, {2, 19}, {3, 20}, {4, 25}, {5, 18}, {6, 25}, {7, 25}, {8, 22}, {9, 24}}
```

Pie and bar charts of the counts do not appear to show any large biases in the first digit.

```
In[9]:= PieChart[digit1[[All, 2]], PieLabels -> digit1[[All, 1]]]
```



```
In[10]:= BarChart[digit1[[All, 2]], BarLabels -> digit1[[All, 1]]]
```



Similar counts could be obtained for the second and third digits.

```
In[11]:= digit2 = Sort[Tally[data[[All, 2]]]]
```

```
Out[11]= {{0, 15}, {1, 24}, {2, 21}, {3, 26}, {4, 27}, {5, 20}, {6, 28}, {7, 22}, {8, 17}, {9, 18}}
```

```
In[12]:= digit3 = Sort[Tally[data[[All, 3]]]]
```

```
Out[12]= {{0, 26}, {1, 12}, {2, 26}, {3, 18}, {4, 23}, {5, 19}, {6, 18}, {7, 27}, {8, 30}, {9, 19}}
```

The counts of 12 and 30 look a little suspicious, but simulations show those counts are not really that extreme.

This approximates the probability of observing a smallest count less than or equal to 12.

```
In[13]:= << MultivariateStatistics`

In[14]:= Count[RandomInteger[MultinomialDistribution[218, Table[1/10, {10}]], 10 000],
  _? (Min[#] <= 12 &) ] / 10 000.

Out[14]= 0.124
```

This approximates the probability of observing a largest count greater than or equal to 30.

```
In[15]:= Count[RandomInteger[MultinomialDistribution[218, Table[1/10, {10}]], 10 000],
  _? (Max[#] >= 30 &) ] / 10 000.

Out[15]= 0.4002
```

Some "traditional" aspects

■ FindFit examples

```
In[16]:= ?? FindFit
```

`FindFit[data, expr, pars, vars]` finds numerical values of the parameters *pars* that make *expr* give a best fit to *data* as a function of *vars*. The data can have the form $\{\{x_1, y_1, \dots, f_1\}, \{x_2, y_2, \dots, f_2\}, \dots\}$, where the number of coordinates *x*, *y*, ... is equal to the number of variables in the list *vars*. The data can also be of the form $\{f_1, f_2, \dots\}$, with a single coordinate assumed to take values 1, 2,

`FindFit[data, {expr, cons}, pars, vars]` finds a best fit subject to the parameter constraints *cons*. >>

```
Attributes[FindFit] = {Protected}
```

```
Options[FindFit] = {AccuracyGoal → Automatic, Compiled → Automatic, EvaluationMonitor → None,
  Gradient → Automatic, MaxIterations → Automatic, Method → Automatic, NormFunction → Norm,
  PrecisionGoal → Automatic, StepMonitor → None, WorkingPrecision → Automatic}
```

A first example

```
In[17]:= data = Table[{t, { Sin[0.1` t] / e^{0.04` t}  t ≥ 0 + RandomReal[{-0.01`, 0.01`}] }}, {t, -5, 150}];

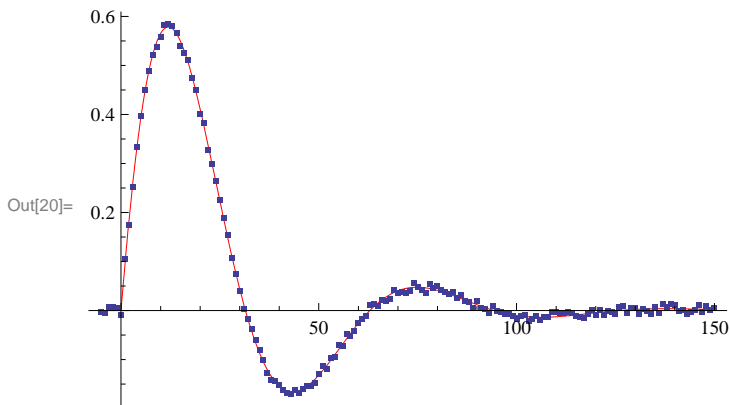
model = Which[t < 0, 0, True, Sin[b t] / e^{-a t}];

In[19]:= sol = FindFit[data, model, {{a, 1}, {b, 1}}, {t}, MaxIterations → 200]

Out[19]= {a → -0.0397316, b → 0.0998833}
```

This shows the final result compared with experimental data

```
In[20]:= Show[Plot[model /. sol, {t, -5, 150}, PlotStyle -> Red, PlotRange -> All],
  ListPlot[data, Joined -> False, PlotStyle -> {PointSize[0.008`]}], PlotRange -> All]
```



Another example.

```
In[21]:= MyFunction[a_, b_, c_, x_] := a If[x > 0, Cos[b x], Exp[c x]]
```

Here I make a list of values adding also a random component

```
In[22]:= Block[{ε = 0.1`, a = 1.2`, b = 3.4`, c = 0.98`},
  data = Table[{x, MyFunction[a, b, c, x] + ε RandomReal[{-0.5`, 0.5`}]}, {x, -5, 5, 0.1`}];
```

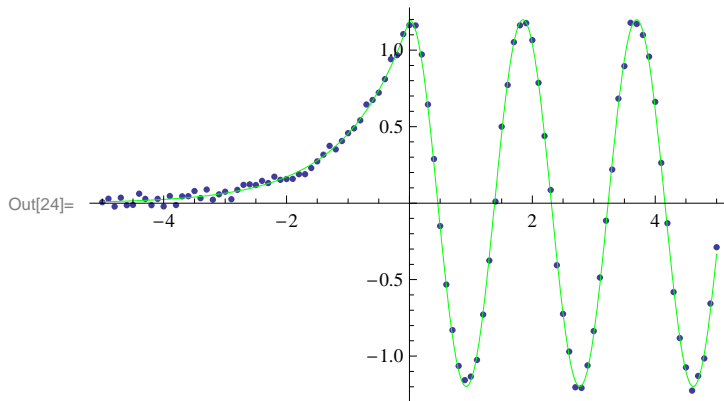
FindFit is used to find the data fitting

```
In[23]:= fit = FindFit[data, MyFunction[a, b, c, x], {{a, 1}, {b, 3}, {c, 1}}, x]
```

```
Out[23]= {a -> 1.19806, b -> 3.39931, c -> 0.971805}
```

This is the graphical result

```
In[24]:= Show[{ListPlot[data],
  Plot[MyFunction[a, b, c, x] /. fit, {x, -5, 5}, PlotStyle -> RGBColor[0, 1, 0]]}]
```



How we can discover intermediate steps? The option `EvaluationMonitor` does this for us

```
In[25]:= Options[FindFit]
```

```
Out[25]= {AccuracyGoal -> Automatic, Compiled -> Automatic, EvaluationMonitor -> None,
  Gradient -> Automatic, MaxIterations -> Automatic, Method -> Automatic, NormFunction -> Norm,
  PrecisionGoal -> Automatic, StepMonitor -> None, WorkingPrecision -> Automatic}
```

```
In[26]:= Reap[FindFit[data, MyFunction[a, b, c, x],
  {{a, 1}, {b, 3}, {c, 1}}, x, EvaluationMonitor->Sow[{a, b, c}]]]
```

```
Out[26]:= {{a -> 1.19806, b -> 3.39931, c -> 0.971805},
  {{{1., 3., 1.}, {0.620764, 3.29122, 0.38883}, {1.1178, 3.49485, 0.907617},
    {1.15837, 3.39585, 0.940838}, {1.19798, 3.39943, 0.972108}, {1.19806, 3.39931, 0.971804},
    {1.19806, 3.39931, 0.971805}, {1.19806, 3.39931, 0.971805}}}}
```

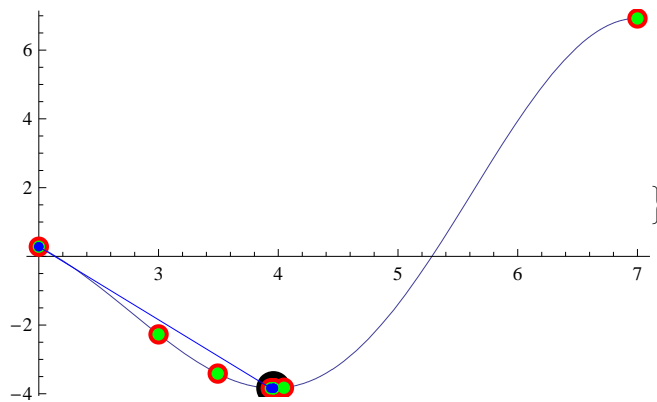
There are many other ways to further investigate the intermediate steps and internal algorithms behaviour. *Mathematica* provides many tools and in some cases additional documentation and/or packages to allow users "to explore and drive" the computation of their data.

This is an example of additional functions provided by the package `Optimization`UnconstrainedProblems``

```
In[27]:= Needs["Optimization`UnconstrainedProblems`"]
```

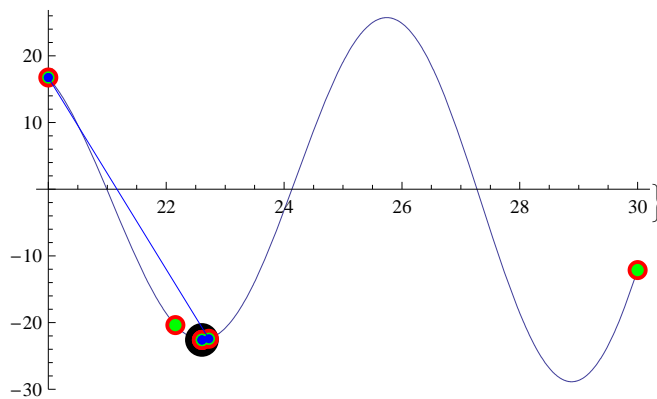
```
In[28]:= FindMinimumPlot[x Sin[x + 1], {x, 2}]
```

```
Out[28]:= {{-3.83922, {x -> 3.95976}}, {Steps -> 4, Function -> 9, Gradient -> 9},
```



```
In[29]:= FindMinimumPlot[x Sin[x + 1], {x, 20}]
```

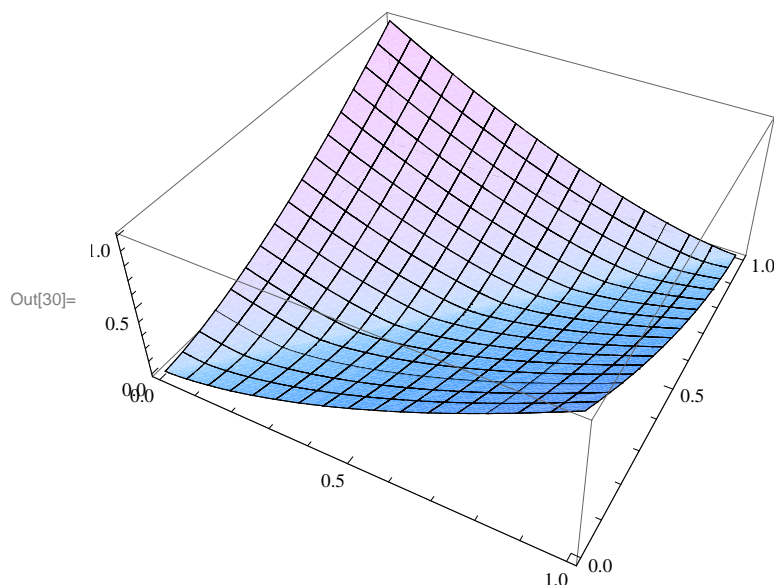
```
Out[29]:= {{-22.5841, {x -> 22.6062}}, {Steps -> 4, Function -> 7, Gradient -> 7},
```



Another example of `EvaluationMonitor` within the `NIntegrate`.

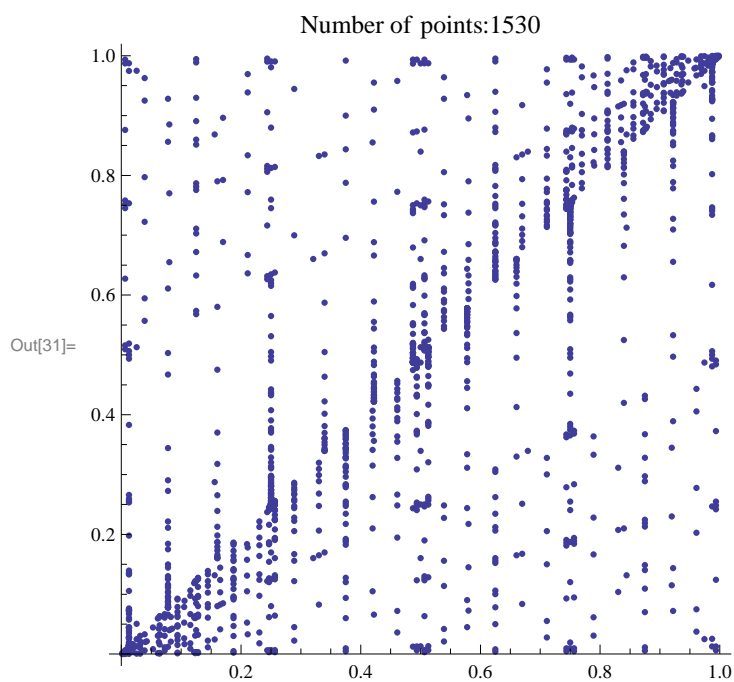
This is the function we want to integrate

```
In[30]:= Plot3D[Abs[x - y]3/2, {x, 0, 1}, {y, 0, 1},
  PlotPoints -> {50, 50}, ViewPoint -> {0.906, -1.672, 1.590}]
```



This is the graphic of points used by NIntegrate when the method used is the default Multidimensional

```
In[31]:= ListPlot[
  xys = Reap[NIntegrate[Abs[x - y]3/2, {x, 0, 1}, {y, 0, 1}, EvaluationMonitor -> Sow[{x, y}]]][[
    2, 1]], AspectRatio -> Automatic,
  PlotLabel -> "Number of points:" <> ToString[Length[xys]]]
```

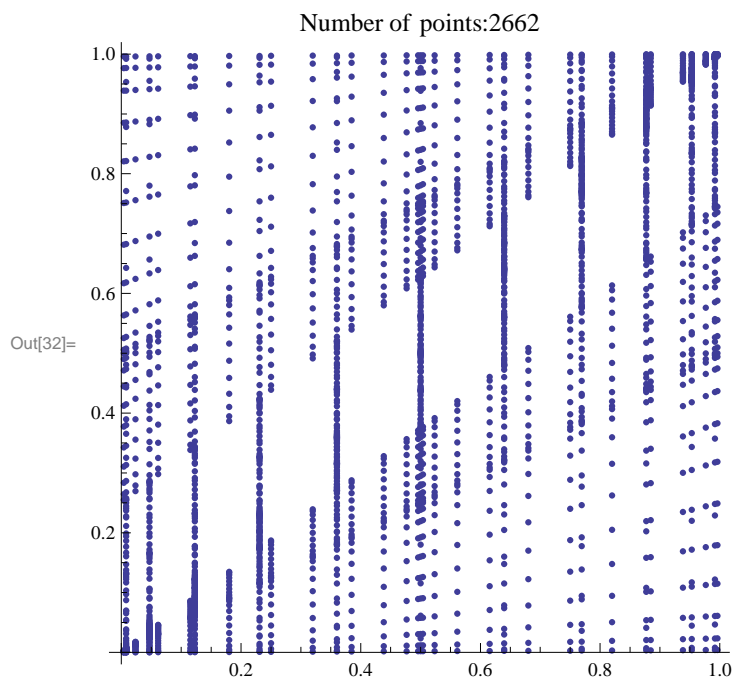


Here is the same for the GaussKronrod method

```

In[32]:= ListPlot[xys = Reap[NIntegrate[Abs[x - y]3/2, {x, 0, 1}, {y, 0, 1},
    Method → GaussKronrod, EvaluationMonitor → Sow[{x, y}]]][[2, 1]],
    AspectRatio → Automatic, PlotLabel → "Number of points:" <> ToString[Length[xys]]]

```

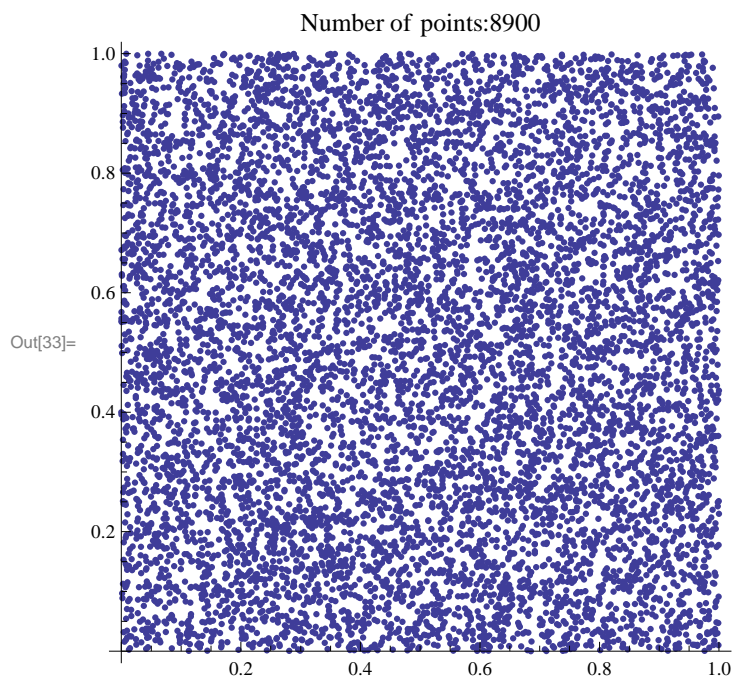


And finally the MonteCarlo method

```

In[33]:= ListPlot[xys = Reap[NIntegrate[Abs[x - y]3/2, {x, 0, 1},
    {y, 0, 1}, Method → MonteCarlo, EvaluationMonitor → Sow[{x, y}]]][[2, 1]],
    AspectRatio → Automatic, PlotLabel → "Number of points:" <> ToString[Length[xys]]]

```



There is also another interesting function named Monitor, that provides a temporary image of what is happening during a calculation

```
In[34]:= values = {};
Monitor[NIntegrate[Sqrt[x (1 - x)], {x, 0, 1},
  EvaluationMonitor -> (Pause[0.025]; AppendTo[values, x];)], ListPlot[values]]

Out[35]= 0.392699
```

Monitor NDSolve using an extrapolation method:

```
In[36]:= values1 = {}; values2 = {};
Monitor[NDSolve[{x'[t] + Sin[x[t]] == 0, x[0] == 1, x'[0] == 0}, x, {t, 0, 2 Pi},
  EvaluationMonitor -> (AppendTo[values1, {t, x[t]}];
  AppendTo[values2, {t, x'[t]}]; Pause[0.05];), Method -> "Extrapolation"],
  ListPlot[{values1, values2}]]

Out[36]= {{x -> InterpolatingFunction[{{0., 6.28319}}, <>]}}
```

Steps in parameter space for a nonlinear fit:

```
In[1]:= data = {{0.18, -0.13}, {0.84, -0.06}, {0.05, 0.88}, {0.24, -0.63}, {0.67, 0.93},
  {0.05, 0.88}, {0.65, 0.92}, {0.01, 0.99}, {0.17, -0.04}, {0.23, -0.55}};
lp = ListPlot[data, PlotRange -> All];

In[3]:= model[{a_, k_, w_, p_}] = a Exp[-k x] Sin[w x + p];
```

Monitor the evolution of the model over the steps with a graph:

```
In[4]:= Monitor[FindFit[data, model[{a, k, w, p}], {a, k, w, p}, x,
  StepMonitor -> (vars = {a, k, w, p}; Pause[0.3];)], Show[lp, Plot[model[vars], {x, 0, 1}]]]

Out[4]= {a -> 0.997279, k -> 0.104408, w -> -9.41643, p -> 1.56093}
```

Implement a simple LU decomposition with a slowdown parameter p :

```
In[5]:= LU[A_, p_ : 0] := Block[{m, n, L, U}, {m, n} = Dimensions[A]; {L, U} = {IdentityMatrix[n], A};
  Do[Pause[p]; L[[k ;; n, k]] = U[[k ;; n, k]] / U[[k, k]];
    U[[{k + 1} ;; n, k ;; n]] = U[[{k + 1} ;; n, k ;; n]] - L[[{k + 1} ;; n, {k}]] . U[[{k}, k ;; n]];
    , {k, 1, n - 1}];
  {L, U}]
```

Monitor an algorithm animation using MatrixForm or MatrixPlot:


```
In[6]:= Monitor[LU[RandomInteger[{1, 10}], {10, 10}], 1], MatrixForm /@ {L, U}]
```

```
Out[6]= {{ {1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, { $\frac{9}{8}$ , 1, 0, 0, 0, 0, 0, 0, 0, 0}, { $\frac{7}{8}$ ,  $\frac{13}{11}$ , 1, 0, 0, 0, 0, 0, 0, 0},
```

$$\left\{\frac{1}{8}, -\frac{31}{33}, \frac{41}{156}, 1, 0, 0, 0, 0, 0, 0\right\}, \left\{\frac{1}{2}, -\frac{4}{3}, -\frac{341}{156}, \frac{211}{76}, 1, 0, 0, 0, 0, 0\right\},$$

$$\left\{\frac{3}{8}, -\frac{5}{33}, \frac{1}{12}, \frac{13}{19}, \frac{332}{1619}, 1, 0, 0, 0, 0\right\}, \left\{\frac{9}{8}, \frac{65}{33}, \frac{229}{78}, -\frac{125}{38}, -\frac{1754}{1619}, -\frac{50208}{19031}, 1, 0, 0, 0\right\},$$

$$\left\{-\frac{3}{4}, \frac{46}{33}, \frac{283}{156}, -\frac{83}{38}, -\frac{1434}{1619}, -\frac{12866}{19031}, \frac{11617}{77568}, 1, 0, 0\right\},$$

$$\left\{-\frac{5}{4}, \frac{26}{33}, \frac{137}{156}, \frac{3}{38}, \frac{1418}{1619}, -\frac{10210}{19031}, -\frac{27367}{12928}, \frac{403998}{91085}, 1, 0\right\},$$

$$\left\{\frac{9}{8}, \frac{73}{33}, \frac{139}{39}, -\frac{74}{19}, -\frac{1960}{1619}, -\frac{6512}{19031}, -\frac{36311}{19392}, \frac{437684}{91085}, \frac{31423409}{51876118}, 1\right\}},$$

$$\{8, 9, 9, 1, 7, 7, 8, 2, 4, 10\}, \left\{0, -\frac{33}{8}, -\frac{65}{8}, -\frac{1}{8}, -\frac{31}{8}, -\frac{15}{8}, -1, \frac{11}{4}, -\frac{7}{2}, -\frac{29}{4}\right\},$$

$$\left\{0, 0, \frac{52}{11}, \frac{80}{11}, \frac{16}{11}, \frac{1}{11}, \frac{46}{11}, -2, \frac{106}{11}, \frac{97}{11}\right\},$$

$$\left\{0, 0, 0, \frac{76}{13}, \frac{121}{39}, -\frac{103}{156}, \frac{103}{26}, \frac{847}{78}, \frac{209}{78}, -\frac{683}{156}\right\},$$

$$\left\{0, 0, 0, 0, -\frac{1619}{228}, -\frac{1795}{912}, -\frac{485}{152}, -\frac{10421}{456}, \frac{407}{24}, \frac{20761}{912}\right\},$$

$$\left\{0, 0, 0, 0, 0, \frac{19031}{6476}, \frac{4675}{3238}, -\frac{2947}{3238}, \frac{23133}{3238}, -\frac{27561}{6476}\right\},$$

$$\left\{0, 0, 0, 0, 0, 0, \frac{77568}{19031}, \frac{223912}{19031}, -\frac{143855}{19031}, -\frac{320224}{19031}\right\},$$

$$\left\{0, 0, 0, 0, 0, 0, 0, \frac{91085}{9696}, \frac{587401}{77568}, -\frac{2783}{2424}\right\},$$

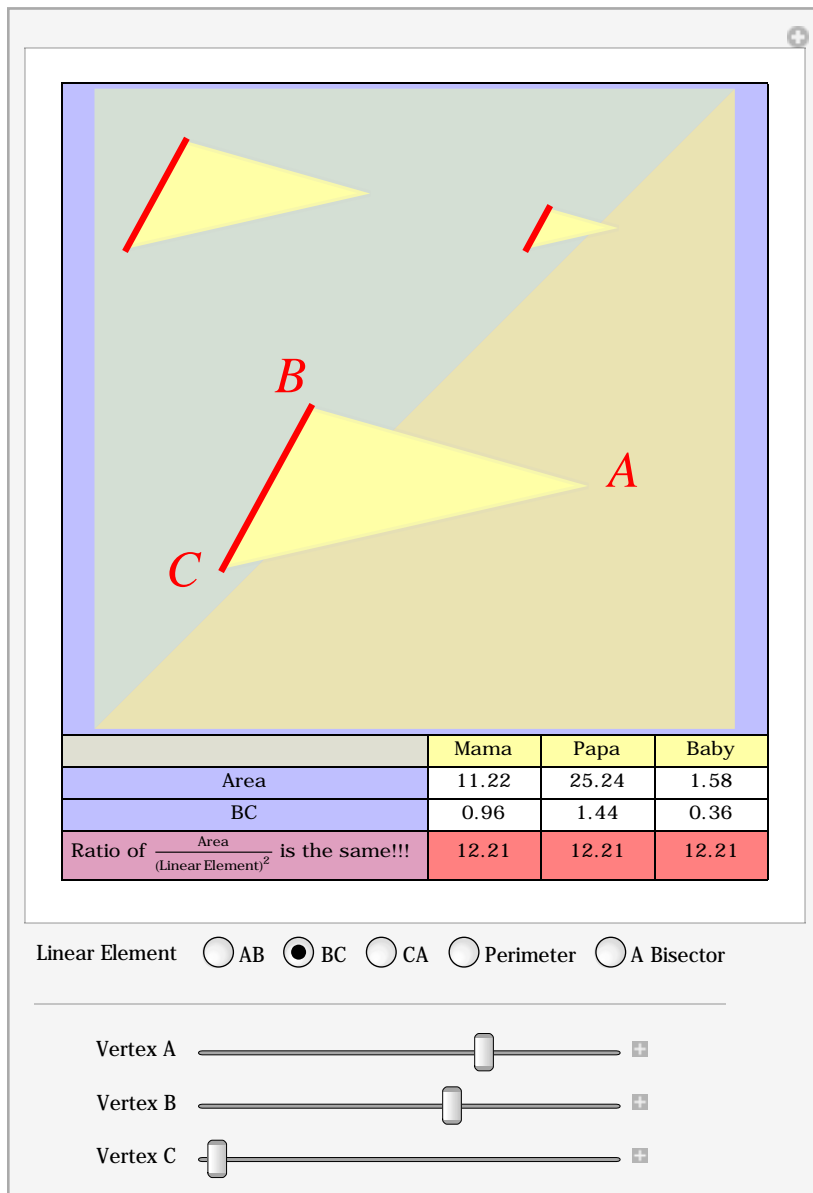
$$\left\{0, 0, 0, 0, 0, 0, 0, 0, -\frac{25938059}{364340}, -\frac{5823514}{91085}\right\}, \left\{0, 0, 0, 0, 0, 0, 0, 0, 0, \frac{107576643}{25938059}\right\}}\}$$

```
In[7]:= Monitor[LU[RandomReal[1, {100, 100}], .3],, MatrixPlot /@ {L, U}]
```

Everything fits with everything

```
Manipulate[DynamicModule[{lastpt = {0, 1}},
Graphics[{PointSize[0.1], Point[Dynamic[Refresh[If[pt != lastpt,
AppendTo[contents, {pt, {0, 0}}]; lastpt = pt], TrackedSymbols -> {pt}];
contents = Map[If[#[[1, 2]] >= 0, {#[[1]] - #[[2]], #[[2]] + {0, 0.001}},
{#[[1, 1]], 0}, {1, -0.8}#[[2]]] &, contents]; Map[First, contents]]],
PlotRange -> {{0, 1}, {0, 1}}, ImageSize -> 500]], {{pt, {0, 1}}, {0, 0},
{1, 1}, Locator, Appearance -> None},
{{contents, {}}, {0, 0}, ControlType -> None,
Appearance -> None},
SynchronousUpdating -> True]
```

Other examples (from demonstrations.wolfram.com)



Geometric objects that have the same shape and same size are called *congruent*. When they have the same shape but different sizes they are called *similar*. Here are three similar triangles called Mama Triangle, Papa Triangle, and Baby Triangle. You can change the position of their vertices with the sliders. Since their angles all change by the same amount they remain similar.

But notice that even though they are different sizes, the ratio of their areas to the square of ANY linear element is the same for all three triangles (watch the pink line in the table). A teenage Albert Einstein made use of this quality in his proof of the Pythagorean theorem.

