1

An introduction to *Mathematica*

Overview of basic operations · Numerical computation · Symbolic computation · Graphics and visualization · Data import and analysis · Dynamic and interactive computation · Programming · Starting up Mathematica · Notebook interface · Entering input · Mathematical expressions · Syntax of functions · Lists · Dealing with errors · Help and documentation

Mathematica is a very large and seemingly complex system. It contains thousands of functions for performing various tasks in science, mathematics, engineering, and many other disciplines. These tasks include numerical and symbolic computation, programming, data analysis, knowledge representation, and visualization of information. In this introductory chapter, we give a sense of its breadth and depth by looking at some computational and programming examples drawn from a variety of fields. The last part of the chapter covers basic topics in getting started, including how to enter and evaluate expressions, how to deal with errors, and how to get help, with pointers to the documentation system. Users already familiar with *Mathematica* could lightly skim this chapter.

1.1 Overview of basic operations

Numerical and symbolic computation

On a very basic level, *Mathematica* can be thought of as a sophisticated calculator. With it you can enter mathematical expressions and compute their values.

```
In[1]:= Sqrt[2.0 * 10 Pi] (10/E)^10
Out[1]= 3.5987 * 10^6
```

You can store values in memory to be used in subsequent computations. For example, the following three inputs compute the Lorentz factor for an object moving at half the speed of light.

```
In[2]:= c = 299 792 458 Meter/Second;

In[3]:= v = c/2

Out[3]= 149 896 229 Meter/Second

In[4]:= N[Sqrt[1 - v^2/c^2]]

Out[4]= 0.866025
```

Yet *Mathematica* differs from calculators and simple computer programs in its ability to calculate exact results and to compute to an arbitrary degree of precision.

```
In[5]:= 1/2 + 1/3 + 1/5 + 1/7 + 1/11 + 1/13

Out[5]= 40 361/30 030

In[6]:= 2^1024

Out[6]= 179 769 313 486 231 590 772 930 519 078 902 473 361 797 697 894 230 657 273 ⁖
        430 081 157 732 675 805 500 963 132 708 477 322 407 536 021 120 113 879 ⁖
        871 393 357 658 789 768 814 416 622 492 847 430 639 474 124 377 767 893 ⁖
        424 865 485 276 302 219 601 246 094 119 453 082 952 085 005 768 838 150 ⁖
        682 342 462 881 473 913 110 540 827 237 163 350 510 684 586 298 239 947 ⁖
        245 938 479 716 304 835 356 329 624 224 137 216

In[7]:= N[Sin[2017 × 2^(1/5)], 40]

Out[7]= -0.9999999999999999978567771261060983259068 5
```

One of the most significant features of *Mathematica* is its ability to manipulate and compute with symbolic expressions. For example, you can factor polynomials and simplify trigonometric expressions.

```
In[8]:= Factor[x^7 - 1]

Out[8]= (-1 + x) (1 + x + x^2 + x^3 + x^4 + x^5 + x^6)
```

```
In[9]:= TrigReduce[Sin[3 θ]^5]
```

$$\text{Out[9]}= \frac{1}{16}\left(10 \sin[3\theta] - 5 \sin[9\theta] + \sin[15\theta]\right)$$

You can simplify expressions using assumptions about variables contained in those expressions. For example, if k is assumed to be an integer, $\sin(2\pi k + x)$ simplifies to $\sin(x)$.

```
In[10]:= Assuming[k ∈ Integers, Simplify[Sin[2 π k + x]]]
Out[10]= Sin[x]
```

Functions are available for solving systems of equations, for example, this solves a symbolic 2×2 linear system.

```
In[11]:= LinearSolve[{{a11, a12}, {a21, a22}}, {x1, x2}]
```

$$\text{Out[11]}= \left\{\left\{\frac{a_{22}\,x_1 - a_{12}\,x_2}{-a_{12}\,a_{21} + a_{11}\,a_{22}}\right\}, \left\{\frac{a_{21}\,x_1 - a_{11}\,x_2}{a_{12}\,a_{21} - a_{11}\,a_{22}}\right\}\right\}$$

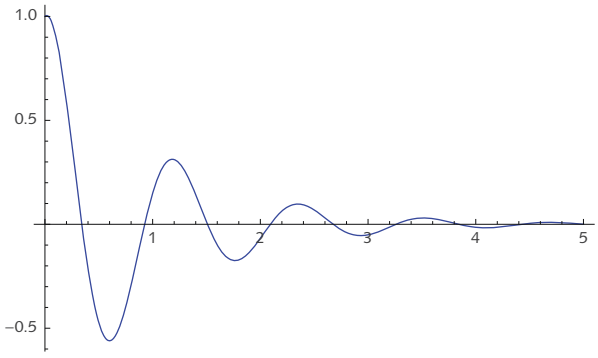
You can solve and plot solutions to differential equations, for example, a system representing a linear damped pendulum.

```
In[12]:= soln = DSolve[{y''[x] + 2 y'[x] + 30 y[x] == 0,
           y[0] == 1, y'[0] == 1 / 2}, y[x], x]
```

$$\text{Out[12]}= \left\{\left\{y[x] \to \frac{1}{58} e^{-x}\left(58 \cos\left[\sqrt{29}\,x\right] + 3\sqrt{29}\,\sin\left[\sqrt{29}\,x\right]\right)\right\}\right\}$$

```
In[13]:= Plot[y[x] /. soln, {x, 0, 5}, PlotRange → All]
```

Out[13]=



You can create and then operate on functions that are defined piecewise.

```
In[14]:= sinc[x_] = Piecewise[ {{1, x == 0}}, Sin[x] / x]
```

$$\text{Out[14]=}\begin{cases} 1 & x == 0 \\ \frac{\text{Sin}[x]}{x} & \text{True} \end{cases}$$

```
In[15]:= Integrate[sinc[x^2] / x, x]
```

$$\text{Out[15]=}\ \frac{\text{CosIntegral}\left[x^2\right]}{2} - \frac{\text{Sin}\left[x^2\right]}{2\,x^2}$$

One of the advantages of working symbolically is that you can quickly see underlying formulas and algorithms at work. For example, this computes a present value for an annuity of 36 payments of $500 using a symbolic effective interest rate.

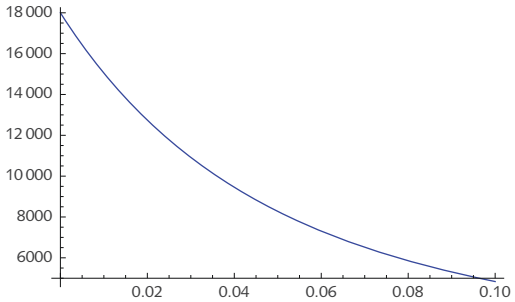
```
In[16]:= presentValue = TimeValue[Annuity[500, 36], r, 0]
```

$$\text{Out[16]=}\ \frac{500\left(-1 + (1 + r)^{36}\right)}{r\,(1 + r)^{36}}$$

A plot clearly shows the relationship between the interest rate and the present value of the annuity.

```
In[17]:= Plot[presentValue, {r, 0.0, 0.10}]
```

Out[17]=



In fact, symbolic expressions are very general objects – you can work with them as you would any expression.

```
In[18]:= Factor[  ^7 - 1]
```

$$\text{Out[18]=}\ \left(-1 + \text{  }\right)\left(1 + \text{  } + \text{  }^2 + \text{  }^3 + \text{  }^4 + \text{  }^5 + \text{  }^6\right)$$

```
In[19]:= Rotate[Style["Mathematica", "Text"], 45 Degree]
```

Out[19]=

Graphics and visualization

Visualizing functions or sets of data often provides greater insight into their structure and properties. *Mathematica* has a wide range of visualization capabilities, including two- and three-dimensional plots of functions or datasets, contour and density plots of functions of two variables, bar charts, histograms and other charting functions for data, and many other functions for specialized areas such as statistical analysis, financial analysis, wavelets, and others. In addition, with the *Mathematica* programming language you can construct graphical images "from the ground up" using primitive elements, as we will see in Chapter 10.

Here is a stream plot of the vector field $\{\cos(1 - x + y^2), \sin(1 + x^2 - y)\}$.

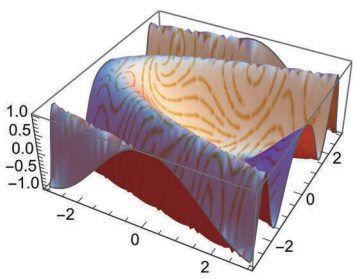
```
In[20]:= strm = StreamPlot[{Cos[-1 - x + y^2], Sin[1 + x^2 - y]},
            {x, -3, 3}, {y, -3, 3}, Frame → None]
```

Out[20]=

This plot can be thought of as a symbolic expression that can then be used in other expressions, such as a texture on a surface.

```
In[21]:= Plot3D[Sin[-1 - x + y^2], {x, -3, 3},
            {y, -3, 3}, PlotStyle → Texture[strm], Mesh → None]
```

Out[21]=



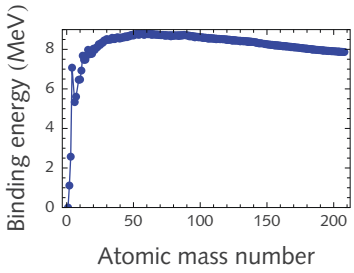
Of course, discrete data, requiring analysis and visualization, are commonly what you will work with. Here we import isotope data and then plot the atomic mass number against the binding energy for all stable isotopes.

```
In[22]:= data = Outer[IsotopeData[#1, #2] &, IsotopeData["Stable"],
            {"MassNumber", "BindingEnergy", "Symbol"}];

In[23]:= Take[data, 8]
Out[23]= {{1, 0., 1H}, {2, 1.112283, 2H}, {3, 2.572681, 3He},
          {4, 7.073915, 4He}, {6, 5.332345, 6Li}, {7, 5.606291, 7Li},
          {9, 6.462758, 9Be}, {10, 6.475071, 10B}}

In[24]:= ListLinePlot[data[[All, {1, 2}]],
          Mesh → All, PlotRange → {0, 9}, Frame → True,
          FrameLabel → {Style["Atomic mass number", 9],
            Style["Binding energy (MeV)", 9]}]
```

Out[24]=



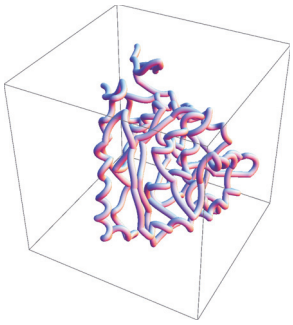
Data from many possible sources – imported from a collector, a database, an online source – can be used directly. For example, this imports the positions of the atoms on a human protein, grouped by amino acid residue.

```
In[25]:= positions = ProteinData["PAH", "AtomPositions", "Residue"];
         Take[positions[[All, 2]], 8]
Out[26]= {{-2540.6, 3683.2, 1606.4}, {-2198.3, 3551.7, 1698.},
          {-2103.1, 3212.1, 1554.4}, {-2017.6, 2937.4, 1804.2},
          {-1649.6, 2912.8, 1901.5}, {-1451.6, 2689.9, 2141.6},
          {-1397.1, 2827.3, 2492.5}, {-1198.3, 2531.6, 2626.9}}
```

These data can then be used to visualize the conformation of the protein backbone by running a Bézier curve through the data and wrapping that curve in a tube.

```
In[27]:= Graphics3D[Tube[BezierCurve[positions[[All, 2]]], 80]]
```

Out[27]=



Working with data

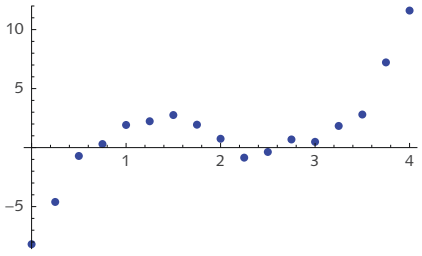
A typical workflow with many kinds of data involves: import, cleaning/filtering, analysis, visualization, export of results. The data itself can take many different forms: tabular/numerical data, images, sound files, movies, HTML pages, and many other types. Once the data are in *Mathematica*, the statistical and visualization tools can be applied to analyze and visualize them. For example, this imports some sample data from a spreadsheet.

```
In[28]:= data = Import["sampledata.xlsx", {"Data", 1}]
Out[28]= {{0., -8.18672}, {0.25, -4.6057},
  {0.5, -0.709252}, {0.75, 0.300171}, {1., 1.91848},
  {1.25, 2.2322}, {1.5, 2.7596}, {1.75, 1.94169},
  {2., 0.748574}, {2.25, -0.852022}, {2.5, -0.368416},
  {2.75, 0.690119}, {3., 0.488073}, {3.25, 1.83513},
  {3.5, 2.80307}, {3.75, 7.2199}, {4., 11.6129}}
```

A plot of the raw data gives a quick picture of the behavior.

```
In[29]:= ListPlot[data]
```

Out[29]=



This fits the data with a linear model using the basis functions x, x^2, and x^3.

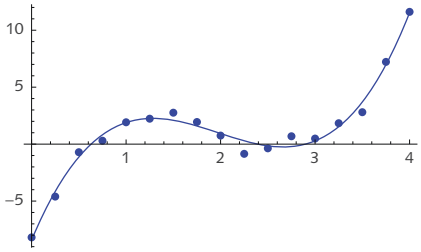
```
In[30]:= model = LinearModelFit[data, {x, x^2, x^3}, x];
```

```
In[31]:= model["BestFit"]
Out[31]= -8.42456 + 19.815 x - 11.4307 x^2 + 1.93117 x^3
```

This shows the model together with the raw data.

```
In[32]:= Show[
          Plot[model[x], {x, 0, 4}, PlotRange → All],
          ListPlot[data]
         ]
```

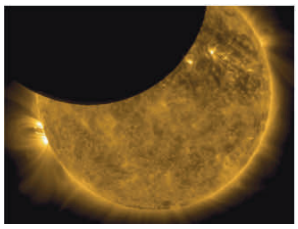
Out[32]=



Data can be imported directly from the internet; in the following, we import an image from a NASA website and operate on it using built-in image processing tools.

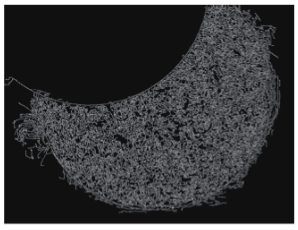
```
In[33]:= sun = Import[
            "http://www.nasa.gov/images/content/491318main_week27-
              transit_946-710.jpg"]
```

Out[33]=



```
In[34]:= EdgeDetect[sun]
```

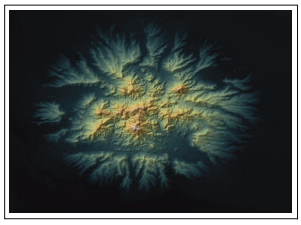
Out[34]=



In the following, three-dimensional digital elevation data contained in an archive from the USGS National Elevation Dataset are used to reconstruct a surface.

In[35]:=
```
Import["NED_40638016.zip"]
```

Out[35]=



In[36]:=
```
Import["NED_40638016.zip", "CoordinateSystemInformation"]
```

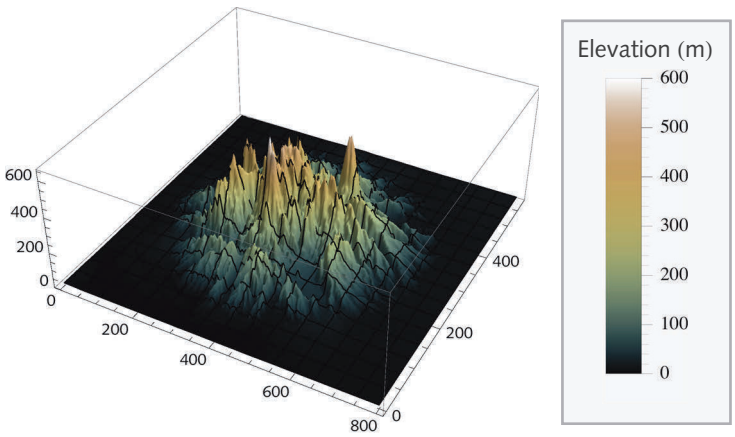
Out[36]=
```
GEOGCS → {NAD83, DATUM → {North_American_Datum_1983, SPHEROID →
      {GRS 1980, 6 378 137, 298.257, AUTHORITY → {EPSG, 7019}},
    TOWGS84 → {0, 0, 0, 0, 0, 0, 0}, AUTHORITY → {EPSG, 6269}},
  PRIMEM → {Greenwich, 0, AUTHORITY → {EPSG, 8901}},
  UNIT → {degree, 0.0174533, AUTHORITY → {EPSG, 9108}},
  AXIS → {Lat, NORTH}, AXIS → {Long, EAST},
  AUTHORITY → {EPSG, 4269}}
```

In[37]:=
```
elevations = Import["NED_40638016.zip", {"ARCGrid", "Data"}];
Dimensions[elevations]
```

Out[38]=
```
{575, 799}
```

In[39]
```
ListPlot3D[elevations, MaxPlotPoints → 300,
 ColorFunction → "Topographic", PlotRange → All,
 PlotLegends → BarLegend[{All, {0, 600}}, LegendLabel →
    Style["Elevation (m)", "Menu"], LegendFunction → "Panel"]]
```

Out[39]=



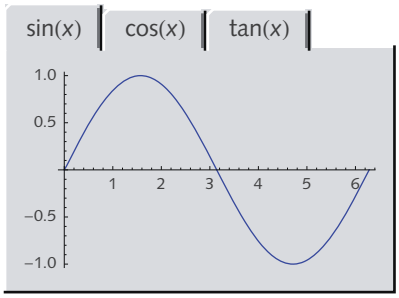
Dynamic interactivity

In addition to the computational tools such as those described above, *Mathematica* also contains tools for creating dynamic interfaces to interact with expressions with which you are working. In this section we will give a few short examples of what is possible, waiting until Chapter 11 for a methodical look at how to program these elements.

Several functions are available to create interfaces in which you manipulate parameters dynamically through controls such as sliders, tabs, checkboxes, pulldown menus, and other mouse-driven interfaces.

In[40]:=
```
TabView[
 Table[TraditionalForm[f[x]] → Plot[f[x], {x, 0, 2 π}],
  {f, {Sin, Cos, Tan}}]]
```

Out[40]=



You can interact with plots directly through the use of dynamic `Locator` objects. In the following example, moving the points with your mouse will cause the fitted model and its plot to be dynamically updated.

In[41]:=
```
Manipulate[
 model = LinearModelFit[pts, {x, x^2, x^3}, {x}];
 Plot[model[x], {x, 0, 1}, PlotRange → 2],
 {{pts, {{.1, .6}, {.2, -.4}, {.45, 0.3},
    {0.56, 0.1}, {.92, .25}}}, Locator}]
```

Out[41]=