Chapter 9
Numerical Calculations

9.1 Approximate Numbers in *Mathematica*

Mathematica usually works with exact numbers, either symbolic (π, e) or rational, and derives exact analytical results. However, it can also perform approximate numerical calculations. Many problems cannot be solved analytically, but numerical solution is possible. On the other side, an analytical result can depend on symbolic parameters; in order to do a numerical calculation, you have to substitute some numerical values for all parameters. It is often useful to check the correctness of a complicated analytical derivation by a direct numerical calculation for a few sets of values of the parameters.

There are two kinds of approximate real numbers in *Mathematica*. The first one is *machine numbers*.

In[1] := *p* = *N*[Pi]
Out[1] = 3.14159
In[2] := FullForm[*p*]
Out[2]//FullForm =
3.141592653589793`

Precision is the number of significant decimal digits. For machine numbers it is a symbolic constant:

In[3] := Precision[*p*]
Out[3] = MachinePrecision
In[4] := *N*[MachinePrecision]
Out[4] = 15.9546

It is 53 bits (or about 16 decimal digits) of precision. In other languages (C, Fortran) such numbers are usually called double precision. Operations with such numbers in *Mathematica* are performed by hardware, as in C or Fortran. They are less efficient than in these languages, but nevertheless rather efficient.

In[5] := MachineNumberQ[*p*]
Out[5] = True

A. Grozin, *Introduction to Mathematica® for Physicists*, Graduate Texts in Physics,
DOI 10.1007/978-3-319-00894-3_9,

There are also *arbitrary-precision* numbers. Operations with them are implemented in software and are far less efficient.

In[6] := p = N[Pi, 25]
Out[6] = 3.14159265358979323846264

In[7] := FullForm[p]
Out[7]//FullForm =
3.14159265358979323846264`25.

In[8] := Precision[p]
Out[8] = 25.

Precision is a part of a value, not of a variable, as in some other languages. If it is equal to n, then the estimated relative error of the value is 10^{-n}. There is also *accuracy*—the number of significant decimal digits after the point. If it is equal to m, then the estimated absolute error of the value is 10^{-m}.

In[9] := Accuracy[p]
Out[9] = 24.5029

When approximate numbers are added or subtracted, the absolute errors are added. The difference of two approximately equal numbers has a lower precision than the operands.

In[10] := q = N[355/113, 20]
Out[10] = 3.1415929203539823009

In[11] := Accuracy[q]
Out[11] = 19.5029

In[12] := d = p − q
Out[12] = $-2.667641890624 \times 10^{-7}$

In[13] := {Precision[d], Accuracy[d]}
Out[13] = {12.929, 19.5028}

When approximate numbers are multiplied or divided, the relative errors are added.

In[14] := r = p/q
Out[14] = 0.9999999150863285520

In[15] := {Precision[r], Accuracy[r]}
Out[15] = {20., 20.}

In[16] := Clear[p, q, d, r]

This error handling sometimes may be too pessimistic. Let's consider an example [15]. The sequence $x_n = f(x_{n-1})$, $x_1 = 1$,

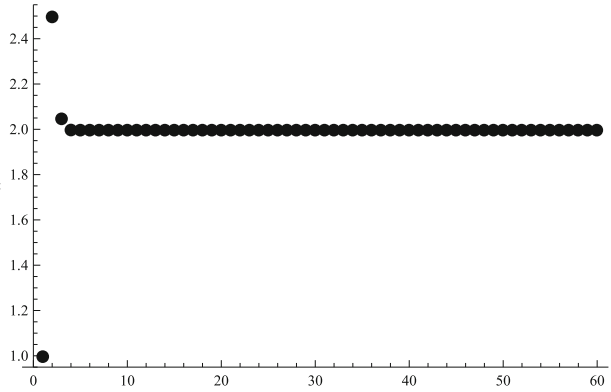
In[17] := f[x_] := (x^2 + 4)/(2 * x)

converges to 2. With machine numbers

```
In[18] := f[x_, n_] := Module[{t = Table[x, {n}]},
    Do[t[[i]] = f[t[[i - 1]]], {i, 2, n}]; t]
In[19] := x = f[1.0, 60];
```

```
In[20] := ListPlot[x, PlotRange->{0.95, 2.55},
      PlotMarkers->{Automatic, Medium}]
```

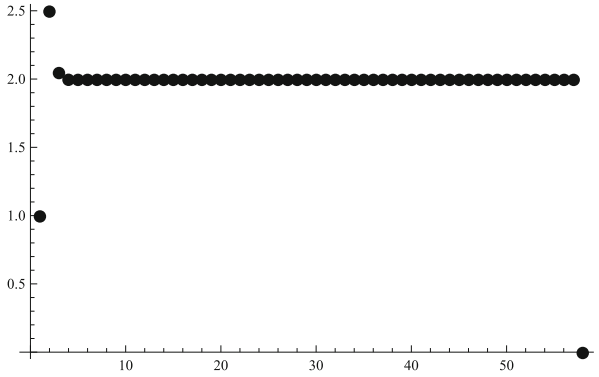
Out[20] =



this is indeed so. Now let x_1 be 1.0 with 17-digits precision:

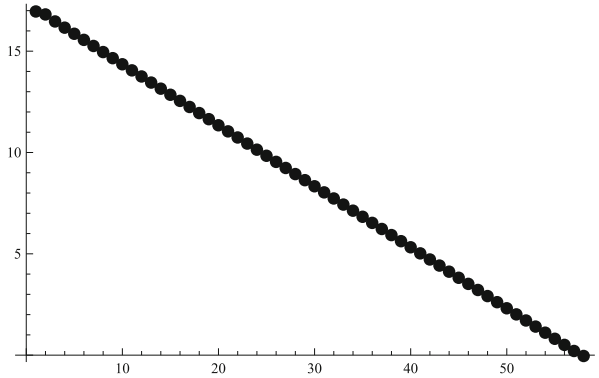
```
In[21] := x = f[1.0`17, 60];
In[22] := ListPlot[x, PlotRange->{-0.05, 2.55},
      PlotMarkers->{Automatic, Medium}]
```

Out[22] =




```
In[23] := ListPlot[Map[Precision, x],
      PlotMarkers->{Automatic, Medium}]
```

Out[23] =



The tail of this list is

In[24] := x[[55;;60]]

Out[24] = {2., 2., 0., 0., ComplexInfinity, Indeterminate}

In[25] := Map[Precision, x[[55;;60]]]

Out[25] = {0.84866, 0.54763, 0.2466, 0., ∞, ∞}

The initial precision is completely lost in 58 iterations.

In[26] := Clear[f, x]

9.2 Solving Equations

NSolve tries to solve equations numerically. They must not contain symbolic parameters, only numbers and unknowns. Only very limited classes of equations can be solved analytically; numerical solution is possible nearly always. The option Reals says to find only real roots.

In[27] := NSolve[x^5 + x + 1 == 0, x]

Out[27] = {{x → −0.754878}, {x → −0.5 − 0.866025i}, {x → −0.5 + 0.866025i},
{x → 0.877439 − 0.744862i}, {x → 0.877439 + 0.744862i}}

In[28] := NSolve[x^5 + x + 1 == 0, x, Reals]

Out[28] = {{x → −0.754878}}

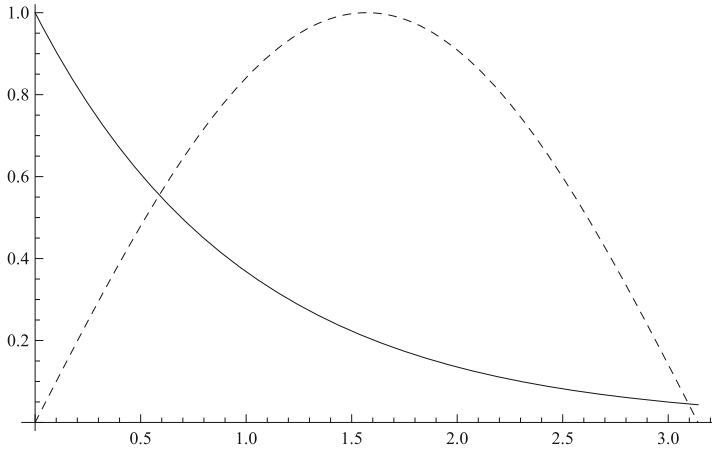
We can add an interval in which we want to find solutions.

In[29] := NSolve[{Exp[−x] == Sin[x], 0 < x < Pi}, x]

Out[29] = {{x → 0.588533}, {x → 3.09636}}

In[30] := Plot[{Exp[−x], Sin[x]}, {x, 0, Pi}]

Out[30] =



9.3 Numerical Integration and Summation

The function NIntegrate integrates numerically, without trying to do it analytically first—it uses an appropriate numerical method right away. Of course, integration limits must be numbers, and there must be no symbolic parameters.

In[31] := NIntegrate[Sin[x]/x, {x, 0, Infinity}]

Out[31] = 1.5708

The option PrecisionGoal states the desired precision of the result. If the result is close to 0 due to strong cancellations, it may be difficult to attain a high precision (i.e., a small relative error). Then it is better to specify AccuracyGoal, i.e., the desired absolute error. The option WorkingPrecision specifies the precision level at which internal calculations are done; it must be $\geqslant$ PrecisionGoal.

In[32] := NIntegrate[Exp[−x^2], {x, 0, Infinity}, WorkingPrecision−>30]

Out[32] = 0.886226925452758013649083741785

The integration method is selected automatically; however, we can specify it:

In[33] := NIntegrate[1/(1 + x ∗ y), {x, 0, 1}, {y, 0, 1},
Method−>"AdaptiveMonteCarlo"]

Out[33] = 0.822237

In[34] := i = NIntegrate[Log[x]^2/(x + 1), {x, 0, 1}, PrecisionGoal−>30,
WorkingPrecision−>35]

Out[34] = 1.8030853547393914280996072422671750

Suppose we suspect that the integral i is a linear combination of $\zeta(3)$ and 1 with rational coefficients. FindIntegerNullVector tries to find integer coefficients such that the linear combination vanishes:

In[35] := FindIntegerNullVector[{i, Zeta[3], 1}]

Out[35] = $\{2, -3, 0\}$

This means that $2\,i - 3\,\zeta(3) = 0$, i.e., our integral is $\frac{3}{2}\,\zeta(3)$. Of course, this is not a mathematical proof. However, if we increase precision, and the linear combination stays the same, we can be practically sure that the result is correct (this is called *experimental mathematics*).

NSum is similar.

In[36] := s = NSum[1/n^4, {n, 1, Infinity}, PrecisionGoal−>30,
WorkingPrecision−>35, NSumTerms−>30]

Out[36] = 1.0823232337111381915160036965412

If we suspect that the sum s is a linear combination of π^4 and 1 with rational coefficients, we can do

In[37] := FindIntegerNullVector[{s, Pi^4, 1}]

Out[37] = $\{90, -1, 0\}$

This means that our sum is, probably, $\frac{\pi^4}{90}$.

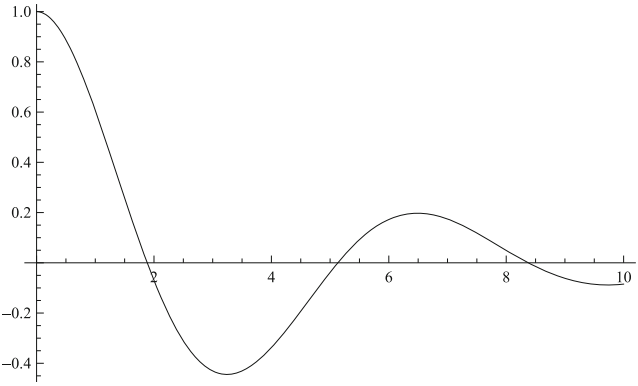
In[38] := Clear[i, s]

9.4 Differential Equations

NDSolve solves differential equations numerically, for a finite interval of the independent variable. It returns results in terms of InterpolatingFunction; this result can be numerically evaluated for any value of the independent variable in the given interval.

```
In[39]:= a = 1/2;
In[40]:= ns = NDSolve[{y''[t] + a * y'[t] + y[t] == 0, y'[0] == 0, y[0] == 1},
          y[t], {t, 0, 10}]
Out[40]= {{y[t] → InterpolatingFunction[{{0., 10.}}, <>][t]}}
In[41]:= Plot[y[t] /. ns[[1]], {t, 0, 10}]
```

Out[41] =



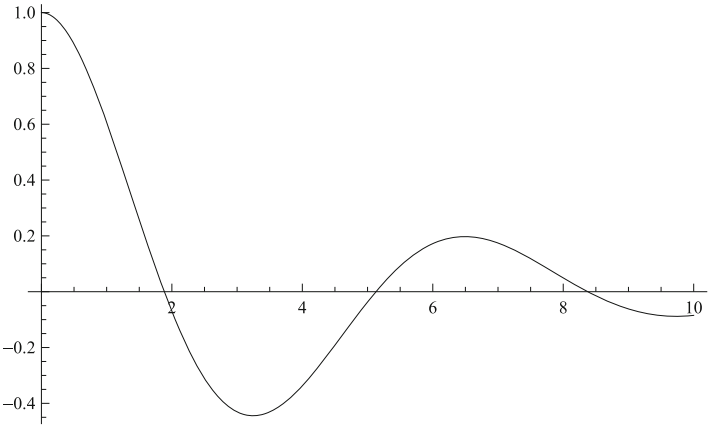
Let's compare with the analytical solution.

```
In[42]:= s = DSolve[{y''[t] + a * y'[t] + y[t] == 0, y'[0] == 0, y[0] == 1}, y[t], t]
```

$$\text{Out}[42] = \left\{\left\{y[t] \to \frac{1}{15} e^{-t/4}\left(15\text{Cos}\left[\frac{\sqrt{15}t}{4}\right] + \sqrt{15}\text{Sin}\left[\frac{\sqrt{15}t}{4}\right]\right)\right\}\right\}$$

```
In[43]:= Plot[y[t] /. s[[1]], {t, 0, 10}]
```

Out[43] =




```
In[44]:= Clear[a, s, ns]
```