

# Tricks of the Trade

This is a column of programming tricks and techniques, most of which, we hope, will be contributed by our readers. You are encouraged to submit ideas to this column.

Edited by Paul Abbott

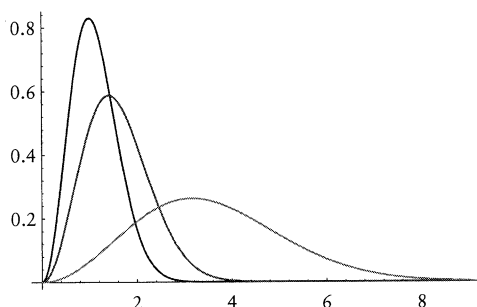
## Maxwellian Speed Distribution

The distribution of molecular speeds is described in any standard physics text (see, for example, *Fundamentals of Physics*, by D. Halliday, R. Resnick, and J. Walker, 4th ed., Wiley, New York, 1993). Consider the Maxwellian probability distribution function (pdf):

```
In[1]:= MaxwellianPDF[v_, a_:1] = v^2 Exp[-a v^2]/
      Integrate[v^2 Exp[-a v^2], {v, 0, Infinity}]
```

$$\text{Out[1]} = \frac{4 a^{3/2} v^2}{E^a v \text{ Sqrt}[\text{Pi}]}$$

```
In[2]:= Plot[Evaluate[MaxwellianPDF[v, #]& /@ {1, 0.5, 0.1}],
      {v, 0, 9}, PlotRange -> All,
      PlotStyle -> GrayLevel /@ {0.2, 0.4, 0.6}];
```



To generate random numbers in the range  $[0, \infty)$  with a (normalised) Maxwellian pdf, we first need to obtain the cumulative distribution function (cdf).

```
In[3]:= MaxwellianCDF[w_] = Integrate[MaxwellianPDF[v, a],
      {v, 0, w/Sqrt[a]}] // Expand
```

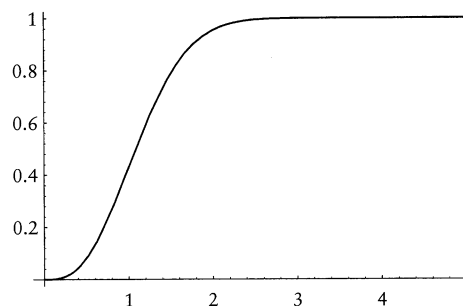
$$\text{Out[3]} = \frac{-2 w}{E^w \text{ Sqrt}[\text{Pi}]} + \text{Erf}[w]$$

We must then find the inverse cdf and apply it to a uniformly distributed random number in the range  $[0, 1]$ . (See Chapter 26.8 of *Handbook of Mathematical Functions*, edited by M. Abramowitz and I. Stegun.)

## Inverse Cumulative Distribution

An easy way to numerically construct the inverse cdf is to Plot the cdf and extract the list of sampled points to produce a table of values,

```
In[4]:= values = {};
      Plot[With[{s = MaxwellianCDF[w]},
      AppendTo[values, {w, s}]; s], {w, 0, 5},
      PlotRange -> All];
```



then reverse the coordinates and interpolate the result:

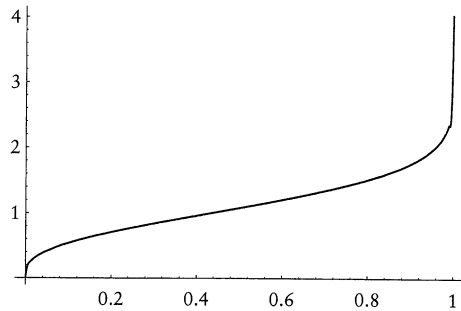
```
In[6]:= InverseCDF = Interpolation[Reverse /@ values]; values = .
```

After reading off the minimum and maximum ordinate values,

```
In[7]:= {tmin, tmax} = InverseCDF[[1]]
      Out[7]= {0., 1.}
```

we can plot the inverse cdf:

```
In[8]:= Plot[InverseCDF[t], {t, tmin, tmax}];
```



### Statistics and Expectation Values

After producing a table of 5000 Maxwellian distributed random numbers,

```
In[9]:= data = Table[InverseCDF[Random[]], {5000}];
```

and loading the package

```
In[10]:= << Statistics`DataManipulation`
```

we can “bin” our data using `BinCounts`,

```
In[11]:= ?BinCounts
```

`BinCounts[data, {xmin, xmax, dx}]` gives a list of the number of elements in data that lie in bins from `xmin` to `xmax` in steps of `dx`. `BinCounts[{{x1,y1}, {x2,y2}, ...}, {xmin, xmax, dx}, {ymin, ymax, dy}]` gives an array of bin counts.

into bins of specified width:

```
In[12]:= width = 0.1;
```

```
In[13]:= bindata = Transpose[
  {Range[width/2, 5, width],
   BinCounts[data, {0, 5, width}] / width / Length[data]}];
```

First, we check that the binned data is normalized:

```
In[14]:= Plus @@ (Last /@ bindata) width
```

```
Out[14]= 1.
```

The numerical (velocity) expectation value  $\langle v \rangle$  is

```
In[15]:= (First /@ bindata).(Last /@ bindata) width
```

```
Out[15]= 1.12464
```

which should be compared with the “exact” value:

```
In[16]:= Integrate[MaxwellianPDF[v] v, {v, 0, Infinity}] // N
```

```
Out[16]= 1.12838
```

Similarly, the expectation value  $\langle v^2 \rangle$

```
In[17]:= ((First /@ bindata)^2).(Last /@ bindata) width
```

```
Out[17]= 1.50082
```

should be compared with

```
In[18]:= Integrate[MaxwellianPDF[v] v^2, {v, 0, Infinity}] // N
```

```
Out[18]= 1.5
```

### Visualization

Using the package

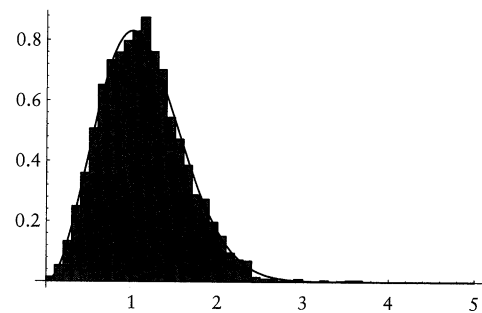
```
In[19]:= << Graphics`Graphics`
```

and modifying our data format,

```
In[20]:= genbindata = bindata /. {a_, b_} -> {a, b, width};
```

we can produce a bar chart of our binned data and compare it with the exact distribution.

```
In[21]:= Show[GeneralizedBarChart[genbindata, PlotRange -> All],
  Plot[MaxwellianPDF[v,1], {v, 0, 5}];
```



### Nonlinear Fitting

Lastly, we can use the package

```
In[22]:= << Statistics`NonlinearFit`
```

to see how well our data is fitted by the Maxwellian pdf:

```
In[23]:= NonlinearFit[bindata,
  a MaxwellianPDF[v, b] + c, v, {a, b, c}]
```

```
Out[23]= {a -> 0.989323, b -> 1.0028, c -> 0.00213543}
```

### $f[a,x]$ versus $f[a][x]$

The notations  $f[a,x]$  or  $f[a][x]$  can often be used interchangeably. However, here are two examples where the second syntax has a clear advantage.

### Wronskian

Consider the linearly independent solutions to a second-order homogenous differential equation:

```
In[1]:= DSolve[y''[x] + y[x] == 0, y[x], x]
```

```
Out[1]= {{y[x] -> C[2] Cos[x] - C[1] Sin[x]}}
```

We can write the general linear combination as

$$\text{In[2]= } y[i\_][x\_ ] := c[i,1] \text{ Cos}[x] + c[i,2] \text{ Sin}[x]$$

This notation makes it easier to compute derivatives,

$$\text{In[3]= } y[1]'[x]$$

$$\text{Out[3]= } c[1, 2] \text{ Cos}[x] - c[1, 1] \text{ Sin}[x]$$

and hence admits a natural implementation of the Wronskian determinant:

$$\text{In[4]= } \text{Wronskian}[u\_ , v\_][x\_ ] := u[x] v'[x] - u'[x] v[x]$$

$$\text{In[5]= } \text{Wronskian}[y[1], y[2]][x] // \text{Simplify}$$

$$\text{Out[5]= } -(c[1, 2] c[2, 1]) + c[1, 1] c[2, 2]$$

### Dynamic Programming

Suppose that you have a function of several parameters, say

$$\text{In[6]= } q[n\_ \text{Integer}, y\_ \text{Real}] := \text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]$$

If this integral needs to be computed a large number of times, one might think of using dynamic programming:

$$\text{In[7]= } q[n\_ \text{Integer}, y\_ \text{Real}] := q[n, y] = \\ \text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]$$

However, it is inefficient to store the integrals for values of the real parameter  $y$ . The major saving is in computing the symbolic integral only once for each value of  $n$ . This can be achieved using the (pure function) syntax

$$\text{In[8]= } q[n\_ ] := q[n] = \text{Function}[y, \\ \text{Evaluate}[\text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]]]$$

After computing

$$\text{In[9]= } q[0][0.3]$$

$$\text{Out[9]= } 0.349859$$

one finds that the symbolic integral for  $n=0$  has been computed and saved:

$$\text{In[10]= } ?q$$

Global`q

$$q[0] = \text{Function}[y$, -1 + E^y$]$$

$$q[n\_ \text{Integer}, y\_ \text{Real}] :=$$

$$q[n, y] = \text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]$$

$$q[n\_ ] := q[n] =$$

$$\text{Function}[y, \text{Evaluate}[\text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]]]$$

This syntax is easily compiled for faster numerical computation by replacing `Function` with `Compile`:

$$\text{In[11]= } \text{Remove}[q]$$

$$\text{In[12]= } q[n\_ ] := q[n] = \text{Compile}[y, \\ \text{Evaluate}[\text{Integrate}[x^n \text{ Exp}[x], \{x, 0, y\}]]]$$

$$\text{In[13]= } q[0][0.3]$$

$$\text{Out[13]= } 0.349859$$

### Kepler Equation

The Kepler equation,  $s = u + e \sin s$ , arises in celestial mechanics. For  $e$  a small quantity, this equation can be solved using power series. Defining  $s = u + a$  reduces the equation to  $a = e \sin(u + a)$ . Introduce the quantity  $a_k = e \sin(u + a_{k-1}) + O(e^{k+1})$ :

$$\text{In[1]= } a[k\_ ] := a[k] = e \text{ Sin}[u + a[k-1]] + 0[e]^(k+1)$$

Then  $a_{k-1} \rightarrow a_k \rightarrow a$  as  $k \rightarrow \infty$ , and hence the Kepler equation is formally satisfied. Note that dynamic programming ( $a[k\_ ] := a[k] = \dots$ ) is used to store the intermediate computations. Defining the initial condition,

$$\text{In[2]= } a[0] = 0;$$

one obtains

$$\text{In[3]= } a[4]$$

$$\text{Out[3]= } \text{Sin}[u] e + \text{Cos}[u] \text{ Sin}[u] e^2 + \\ (\text{Cos}[u]^2 \text{ Sin}[u] - \frac{\text{Sin}[u]^3}{2}) e^3 + \\ (\frac{-7 \text{Cos}[u] \text{Sin}[u]^3}{6} + \text{Cos}[u] (\text{Cos}[u]^2 \text{Sin}[u] - \frac{\text{Sin}[u]^3}{2})) \\ e^4 + 0[e]^5$$

This result can be converted to a Fourier series expansion using `Expand`, `Normal`, and `Collect`:

$$\text{In[4]= } \text{Expand}[\text{Normal}[\%], \text{Trig} \rightarrow \text{True}]$$

$$\text{Out[4]= } e \text{ Sin}[u] - \frac{e^3 \text{ Sin}[u]}{8} + \frac{e^2 \text{ Sin}[2 u]}{2} - \frac{e^4 \text{ Sin}[2 u]}{6} + \\ \frac{3 e^3 \text{ Sin}[3 u]}{8} + \frac{e^4 \text{ Sin}[4 u]}{3}$$

$$\text{In[5]= } \text{Collect}[\%, \text{Table}[\text{Sin}[n u], \{n, 1, 3\}]]$$

$$\text{Out[5]= } (e - \frac{e^3}{8}) \text{ Sin}[u] + (\frac{e^2}{2} - \frac{e^4}{6}) \text{ Sin}[2 u] + \frac{3 e^3 \text{ Sin}[3 u]}{8} + \\ \frac{e^4 \text{ Sin}[4 u]}{3}$$

The self-consistency of the series solution (that  $a = e \sin(u + a)$ ) is easily checked:

```
In[6]:= a[5] - e Sin[u + a[5]]
```

```
Out[6]:= 0[e]6
```

### Solving Inverse Problems Using a Green Function

A powerful and general method for solving inhomogenous differential equations such as

$$L[f(x)] = g(x),$$

where  $L$  is a differential operator with homogenous boundary conditions

$$f(a) = 0, f(b) = 0,$$

is provided by the Green function  $G(x, y)$ , which formally satisfies

$$L[G(x, y)] = \delta(x - y),$$

where  $\delta$  is the Dirac delta function (distribution), and the (homogenous) boundary conditions

$$G(a, y) = 0, G(b, y) = 0, G(x, a) = 0, G(x, b) = 0.$$

Some readers may object to calling  $G(x, y)$  a Green function rather than a Green's function. To quote from J.D. Jackson, *Classical Electrodynamics* (Wiley, New York, 1975, p. viii):

Of minor note is the change from Maxwell's equations and a Green's function to the Maxwell equations and a Green function. The latter boggles some minds but is in conformity with other usage (Bessel function, for example). It is still Green's theorem, however, because that's whose theorem it is.

Green functions are due to George Green (1793–1841), an English mathematician whose work was not appreciated during his lifetime – largely due to his unusual methodology.

The strength of the method is that  $G(x, y)$  is *independent* of the inhomogenous part  $g(x)$  of the differential equation. The solution to the differential equation is

$$f(x) = \int_a^b G(x, y)g(y)dy.$$

The following example is motivated by the article "Solving inverse problems using the Maximum Entropy Principle," by H.G. Miller and D.W. von Oertzen (in *Programming and Mathematical Techniques in Physics*, edited by Yu. Yu. Lobanov and E.P. Zhidkov, World Scientific, Singapore, 1994).

Consider the differential equation describing a forced simple harmonic oscillator:

$$x''(t) + x(t) = g(t),$$

along with the boundary conditions

$$x(0) = x(\pi/2) = 0.$$

The (region-dependent) Green function, independent of  $g(t)$ , is

$$G(t, \tau) = \begin{cases} -\cos \tau \sin t, & a \leq t \leq \tau \leq b \\ -\cos t \sin \tau, & a \leq \tau \leq t \leq b \end{cases}$$

This can be implemented as

```
In[1]:= Green[1][t_, tau_] = -Cos[tau] Sin[t];
Green[2][t_, tau_] = -Cos[t] Sin[tau];
Green[t_, tau_] /; 0 <= t <= tau <= N[Pi/2] :=
  Green[1][t, tau];
Green[t_, tau_] /; 0 <= tau <= t <= N[Pi/2] :=
  Green[2][t, tau]
```

We can include the equality  $t = \tau$  because the Green function is continuous:

```
In[5]:= Green[1][t, tau] - Green[2][t, tau] /. t -> tau
Out[5]:= 0
```

Writing the differential operator as

```
In[6]:= L[x_] := D[x, {t, 2}] + x
```

it is easily verified that the Green function satisfies the homogenous differential equation in each region,

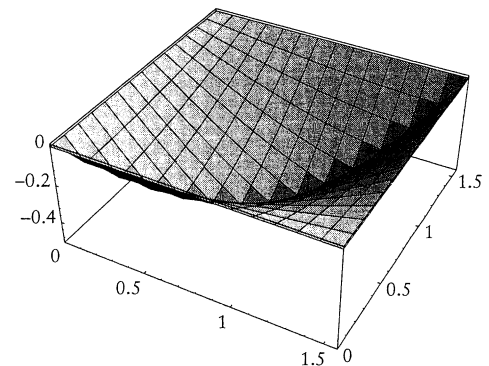
```
In[7]:= L[Green[1][t, tau]] == L[Green[2][t, tau]] == 0
Out[7]:= True
```

and vanishes along the boundaries:

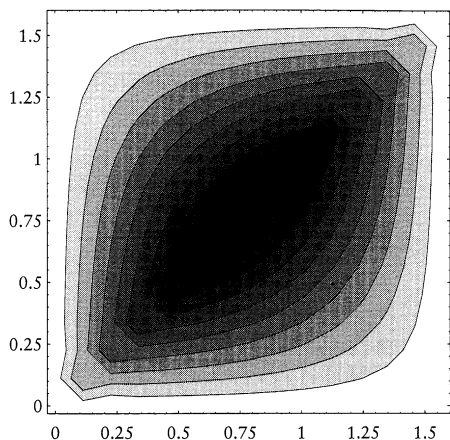
```
In[8]:= Green[1][0, tau] == Green[1][t, Pi/2] ==
  Green[2][t, 0] == Green[2][Pi/2, tau] == 0
Out[8]:= True
```

Visualizing the Green function reveals the behavior on the boundaries and the symmetry about the diagonal line  $t = \tau$ :

```
In[9]:= Plot3D[Green[t, tau], {t, 0, Pi/2}, {tau, 0, Pi/2}]
```



```
In[10]:= ContourPlot[Green[t, tau], {t, 0, Pi/2}, {tau, 0, Pi/2}]
```



The Green function has a discontinuous derivative along the line  $t = \tau$ . The difference across this line is

```
In[11]:= D[Green[2][t, tau], t] - D[Green[1][t, tau], t] /.
t -> tau // Simplify
```

```
Out[11]= 1
```

Consider the *inverse problem* of determining the inhomogeneous part  $g(t)$  of the differential equation by measuring  $x(t)$  at a finite number of times. After discretizing the equation

$$x(t) = \int_a^b G(t, \tau)g(\tau)d\tau,$$

values of  $g_j = g(\tau_j)$  can be computed from the inverse of the (discretized) Green function:

$$x(t_i) = \sum_{j=1}^n G(t_i, \tau_j) g(\tau_j)\delta\tau_j \Rightarrow x_i = G_{ij}\delta_j g_j \Rightarrow g_j = (G_{ij}\delta_j)^{-1} x_i.$$

The pseudo-inverse (or Moore-Penrose inverse) is required because the matrix  $G_{ij}$  is, in general, rectangular:

```
In[12]:= ?PseudoInverse
PseudoInverse[m] finds the pseudoinverse of a rectangular
matrix.
```

Consider a particular forcing term:

```
In[13]:= g[t_] = E^-t Sin[10 t];
```

The solution to the differential equation is

```
In[14]:= x[t_] =
Integrate[Green[2][t, tau] g[tau], {tau, 0, t}] +
Integrate[Green[1][t, tau] g[tau], {tau, t, Pi/2}] //
Simplify
```

```
Out[14]= (10 Cos[t] - 10 E^t Cos[10 t] - 10 E^Pi/2 Sin[t] -
49 E^t Sin[10 t]) / 5002
```

It is easy to verify that this function satisfies the equation,

```
In[15]:= (x''[t] + x[t] // Simplify) == g[t]
```

```
Out[15]= True
```

and the boundary conditions:

```
In[16]:= x[0] == x[Pi/2] == 0
```

```
Out[16]= True
```

Suppose that, at the times  $t_i$ ,

```
In[17]:= ti = Range[0, 1.5, 0.15];
```

we make measurements of the displacements  $x_i = x(t_i)$ :

```
In[18]:= xi = x /@ ti // N;
```

Then, to compute the driving term at the times

```
In[19]:= delj = N[Pi/50];
```

```
In[20]:= tauj = Range[delj, Pi/2, delj] // N
```

```
Out[20]= {0.0628319, 0.125664, 0.188496, 0.251327, 0.314159,
0.376991, 0.439823, 0.502655, 0.565487, 0.628319,
0.69115, 0.753982, 0.816814, 0.879646, 0.942478,
1.00531, 1.06814, 1.13097, 1.19381, 1.25664, 1.31947,
1.3823, 1.44513, 1.50796, 1.5708}
```

the discretized product  $G_{ij}\delta_j$  is

```
In[21]:= Apply[Green[##]&, Outer[List, ti, tauj], {2}] delj;
```

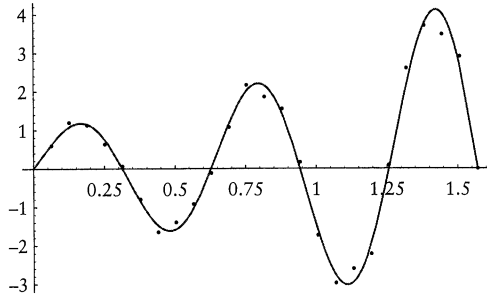
Computing the pseudo-inverse yields the solution to the inverse problem:

```
In[22]:= PseudoInverse[%] . xi
```

```
Out[22]= {0.598546, 1.19473, 1.12583, 0.634954, 0.059995,
-0.795513, -1.64788, -1.39195, -0.917474, -0.118439,
1.07218, 2.16464, 1.86116, 1.55034, 0.165687, -1.73135,
-2.97166, -2.59968, -2.21745, 0.0885683, 2.60381,
3.71391, 3.49089, 2.91582, -3.09335 10^-14}
```

Here is the comparison between the exact and discretized solutions:

```
In[23]:= Plot[g[t], {t, 0, Pi/2},
  PlotStyle -> RGBColor[1,0,0],
  Epilog -> ListPlot[Transpose[{tau, %}][[1]] ]
```

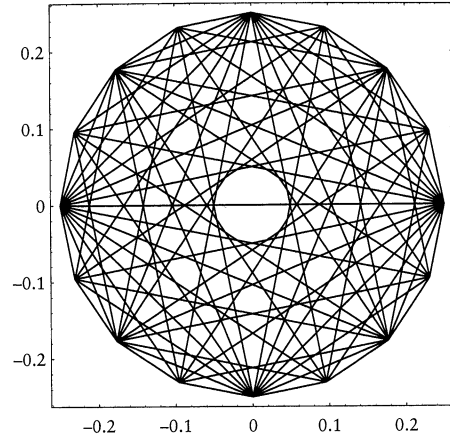


### Plots of FFT of IdentityMatrix

Here is a function for visualizing the Fourier transforms of identity matrices as Argand diagrams, inspired by an article entitled “Yet another look at the FFT,” appearing in *Cleve’s Corners, Collected Columns from the MathWorks Newsletter* (The MathWorks Inc., [info@mathworks.com](mailto:info@mathworks.com)):

```
In[1]:= FFTPlot[n_] :=
  With[{ft = Fourier /@ N[IdentityMatrix[n]]},
    ListPlot[Flatten[
      Transpose[{Re[#], Im[#]}]& /@ ft, 1],
      PlotJoined -> True,
      AspectRatio -> Automatic,
      Axes -> None,
      Frame -> True,
      FrameTicks -> Automatic ]; ]
```

```
In[2]:= FFTPlot[16]
```



```
In[3]:= FFTPlot[17]
```

