Efficiently processing data files

```
In[139]:=
Clear["Global`*"]; DateString[]
SetDirectory[NotebookDirectory[]];
Out[139]=
Fri 20 Feb 2026 11:02:27
```

Functions

Functions are a handy way to bundle up a possibly complex calculation (or series of calculations) into a single command. They are especially well-suited for repetitive operations.

Example: Reading a LoggerPro data file.

First, take a quick look at the first few lines to see what you are up against:

```
In[141]:=
FilePrint["TO-20250220-undamped-1.csv", 3]
"Latest: Time (s)","Latest: Potential (V)"

0,0.495300292969

0.002,0.49072265625
```

This is a comma-separated file, so import it like this. Look at the first few lines of the imported data to make sure it is fine.

```
In[142]:=
rawdata = Import["TO-20250220-undamped-1.csv", "CSV"];
```

[[ ]] takes Part of an array. Range[3] prints out {1, 2, 3}, so we will be taking the first 3 elements of the array. TableForm[ ] merely prints it out in a table format. (MatrixForm is similar.)

```
In[143]:=
TableForm[rawdata⟦Range[3]⟧]
Out[143]//TableForm=
Latest: Time (s)    Latest: Potential (V)
0.                  0.4953
0.002               0.490723
```

We want to select all the lines where there are 2 elements and each element is a number. Fiddle with just the first 3 lines until we are sure we have it right.

```
In[144]:=
Select[rawdata⟦Range[3]⟧, Length[#] == 2 && NumberQ[#⟦1⟧] && NumberQ[#⟦2⟧] &]
Out[144]=
{{0., 0.4953}, {0.002, 0.490723}}
```

There is a more compact way to do those number tests on each element of a list: VectorQ[expression, test] gives True only if *test* yields True when applied to each of the elements in *expression*.

In[145]:=
```
Select[rawdata⟦Range[3]⟧, Length[#] == 2 && VectorQ[#, NumberQ] &]
```
Out[145]=
```
{{0., 0.4953}, {0.002, 0.490723}}
```

So we can apply that to all the raw data. Again, just as a visual check, we can look at the first few lines.

In[146]:=
```
data = Select[rawdata, Length[#] == 2 && VectorQ[#, NumberQ] &];
TableForm[data⟦Range[3]⟧] (* Again, look at the first few lines. *)
```
Out[147]//TableForm=
```
0.       0.4953
0.002    0.490723
0.004    0.490723
```

Lastly, I'd like to convert the voltages to angles by applying my calibration factor. (The voltage offset won't be relevant for us since it has likely drifted anyway.)

In[148]:=
```
dθdV = 0.558; (* From my voltage calibration curve *)
```
In[149]:=
```
θvst = Table[{data⟦i, 1⟧, dθdV * data⟦i, 2⟧}, {i, 1, Length[data] }];
TableForm[θvst⟦Range[3]⟧]
```
Out[150]//TableForm=
```
0.       0.276378
0.002    0.273823
0.004    0.273823
```

Bundling it all up as a function: Think about what you ideally would like to do. In this case, I would like to write something like:

θvst = getFile["TO-20250220-undamped-1.csv", dθdV].

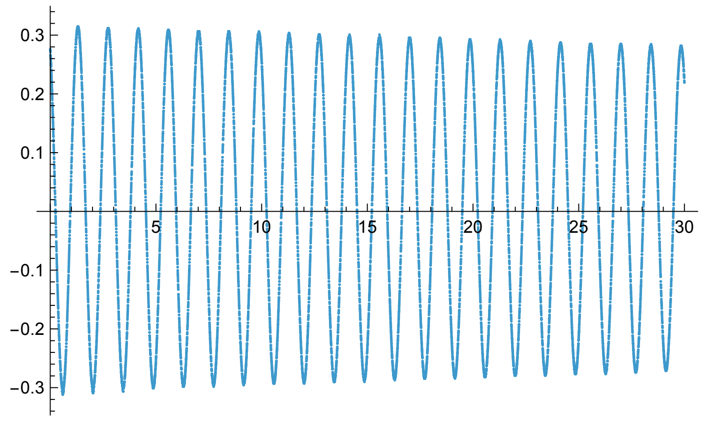
The way I did it here, I used temporary arrays 'rawdata', and 'data', which I didn't need afterwards. I could get rid of them by squishing everything into one command, but I can also use a 'Module', which allows you to specify such temporary variables which are discarded after the function runs.

In[151]:=
```
(* Read in Voltage vs. Time file from LoggerPro and convert the voltages to
 angular displacement. Items listed in green are private to this function. *)
getFile[filename_, dθdV_] := Module[
  {rawdata, data}, (* List of temporary variables for this function *)
  rawdata = Import[filename, "CSV"];
  (* Individual commands get separated by semicolons *)
  data = Select[rawdata, Length[#] == 2 && VectorQ[#, NumberQ] &];
  (* The function returns the last line executed, so don't suppress that
   with a semicolon. *)
  Table[{data⟦i, 1⟧, dθdV * data⟦i, 2⟧}, {i, 1, Length[data] }]
  ]
```

In[152]:=

```
(* Test it out *)
θvst = getFile["TO-20250220-undamped-1.csv", dθdV];
ListPlot[θvst]
```

Out[153]=



Example: Finding the undamped frequencies

In[154]:=

```
model[θ0_, γ_, ω_, ϕ_, θoff_, t_] := θ0 Exp[- γ t / 2] Sin[ω t + ϕ] + θoff
```

For the torsional oscillator, we want to find the frequency for each of our 5 trials. Again, try the commands one by one, and then bundle them all up in a handy function.

Recall that we often had to give a guess to help NonlinearModelFit[ ] find the right solution. For our next experiment, that will be important, so let’s include a guess here:

In[155]:=

```
ωguess = 5;
fit = NonlinearModelFit[θvst,
  model[θ0, γ, ω, ϕ, θoff, t], {θ0, γ, {ω, ωguess}, ϕ, θoff}, t]
```

Out[155]=

```
FittedModel[ 0.00719 + 0.311 ≪1≫ Sin[2.08 + ≪1≫] ]
```

In[156]:=

```
fit["BestFitParameters"]
```

Out[156]=

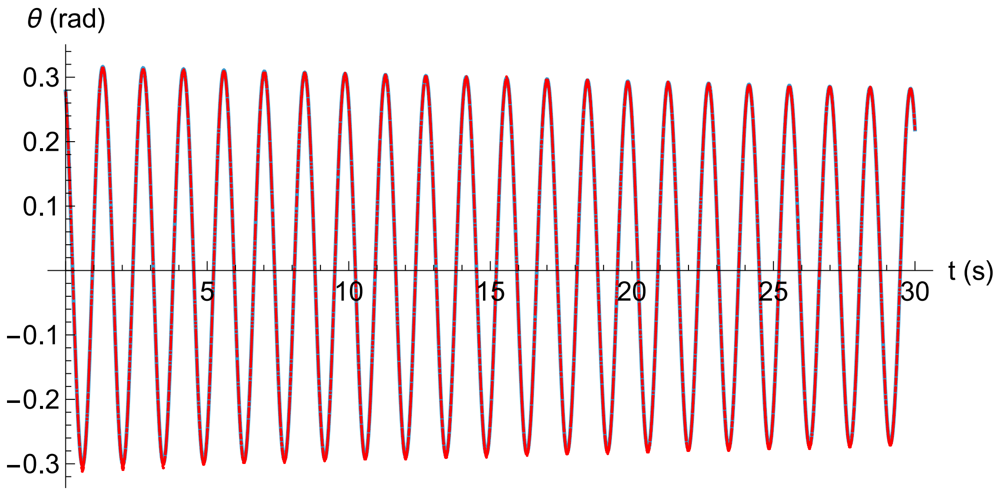
```
{θ0 → 0.310506, γ → 0.00814831, ω → 4.40419, ϕ → 2.08286, θoff → 0.0071934}
```

Check it out visually to make sure it is reasonable. (Putting the theory first and the data second will draw the dots on top of the theory plot, making it visually clearer. The other options here just stretch it out nicely and make clear labels.)

In[157]:=

```
Show[{
  Plot[fit[t], {t, 0, 30}],
  ListPlot[θvst, PlotStyle → {Red, PointSize[0.002]} ]
 },
 LabelStyle → Larger, AxesLabel → {"t (s)", "θ (rad)"},
 AspectRatio → 1/2, ImageSize → Scaled[0.8]
]
```

Out[157]=



To assign one of the fit parameters to a result, such as the frequency, use the /. notation:

In[158]:=

```
ω /. fit["BestFitParameters"]
```

Out[158]=

```
4.40419
```

Bundling it up into a function. Note that all lines in the function (except the last) end in a ‘;’ to suppress the output.

In[159]:=

```
getωfit[data_, ωguess_] := Module[
  {fit, θ0, γ, ω, ϕ, θoff, t, ωfit},
  fit = NonlinearModelFit[data,
    model[θ0, γ, ω, ϕ, θoff, t], {θ0, γ, {ω, ωguess}, ϕ, θoff}, t];
  ωfit = ω /. fit["BestFitParameters"]
 ]
```

In[160]:=

```
(* Test it out.  (It is not very sensitive to our guess.)*)
getωfit[θvst, 5]
```

Out[160]=

```
4.40419
```

You can even chain the two functions together:

In[161]:=
```
getωfit[getFile["TO-20250220-undamped-1.csv", dθdV], 5]
```
Out[161]=
```
4.40419
```

Operating on multiple files

Of course all this work is only worthwhile if you are going to do it multiple times. In this experiment, I have 5 files. Mathematica can list out the filenames easily, if you used an easy file name system. The FIlenames[ ] command returns a list (in curly braces) of files matching a specification. Using the ‘*’ wildcard, I can list out my five files:

In[162]:=
```
FileNames["TO-20250220-undamped-*.csv"]
```
Out[162]=
```
{TO-20250220-undamped-1.csv,
 TO-20250220-undamped-2.csv, TO-20250220-undamped-3.csv,
 TO-20250220-undamped-4.csv, TO-20250220-undamped-5.csv}
```

You can use that list to generate a Table of *ω* values: The {f, FileNames[]} construct
assigns the variable ‘f’ to each of the elements in the FileNames list in turn. (I like to break things out over multiple lines to make the groupings clearer.)

In[163]:=
```
ω0s = Table[
   getωfit[getFile[f, dθdV], 5],
   {f, FileNames["TO-20250220-undamped-*.csv"]}
  ]
```
Out[163]=
```
{4.40419, 4.40434, 4.40426, 4.40422, 4.40429}
```

In[164]:=
```
Mean[ω0s]
```
Out[164]=
```
4.40426
```

In[165]:=
```
δω0 = StandardDeviation[ω0s] / Sqrt[Length[ω0s]]
```
Out[165]=
```
0.0000266678
```

Now it is very easy to repeat all that with my damped oscillation files:

In[166]:=
```
ωvs = Table[
   getωfit[getFile[f, dθdV], 5],
   {f, FileNames["TO-20250220-damped-*.csv"]}
  ]
```
Out[166]=
```
{4.41284, 4.41285, 4.41277, 4.41271, 4.41272}
```

In[167]:=

```
Mean[ωvs]
```

Out[167]=

```
4.41278
```

In[168]:=

```
δωv = StandardDeviation[ωvs] / Sqrt[Length[ωvs]] (* Uncertainty *)
```

Out[168]=

```
0.0000287787
```

Repeating to find the *γ* values

In[169]:=

```
getγfit[data_, ωguess_] := Module[
  {fit, θ0, γ, ω, ϕ, θoff, t, γfit},
  fit = NonlinearModelFit[data,
    model[θ0, γ, ω, ϕ, θoff, t], {θ0, γ, {ω, ωguess}, ϕ, θoff}, t];
  γfit = γ /. fit["BestFitParameters"]
 ]
```

In[170]:=

```
γs = Table[
  getγfit[getFile[f, dθdV], 5],
  {f, FileNames["TO-20250220-damped-*.csv"]}
 ]
```

Out[170]=

```
{0.141641, 0.141499, 0.141431, 0.141397, 0.141265}
```

In[171]:=

```
Mean[γs]
```

Out[171]=

```
0.141447
```

In[172]:=

```
δγ = StandardDeviation[γs] / Sqrt[Length[γs]] (* Uncertainty *)
```

Out[172]=

```
0.0000617774
```

Bundling both together -- returning more than one value from a function.

You can also have Mathematica return both *ω* and *γ* from the fit with a single function call. The function should return a list (in curly braces):

In[173]:=
```
getωγfit[data_, ωguess_] := Module[
  {fit, θ0, γ, ω, ϕ, θoff, t, ωfit, γfit},
  fit = NonlinearModelFit[data,
    model[θ0, γ, ω, ϕ, θoff, t], {θ0, γ, {ω, ωguess}, ϕ, θoff}, t];
  ωfit = ω /. fit["BestFitParameters"];
  γfit = γ /. fit["BestFitParameters"];
  {ωfit, γfit} (* Return a list*)
 ]
```

In[174]:=
```
Table[
 getωγfit[getFile[f, dθdV], 5],
 {f, FileNames["TO-20250220-damped-*.csv"]}
]
```

Out[174]=
```
{{4.41284, 0.141641}, {4.41285, 0.141499},
 {4.41277, 0.141431}, {4.41271, 0.141397}, {4.41272, 0.141265}}
```

Using Mathematica’s Around[ ] function for Uncertainties

You can use the Around function to express a number with its uncertainty.

In[175]:=
```
{ω0avg, δω0} = {Mean[ω0s], StandardDeviation[ω0s] / Sqrt[Length[ω0s]]}
```

Out[175]=
```
{4.40426, 0.0000266678}
```

In[176]:=
```
{γavg, δγ} = {Mean[γs], StandardDeviation[γs] / Sqrt[Length[γs]]}
```

Out[176]=
```
{0.141447, 0.0000617774}
```

In[177]:=
```
ω0 = Around[ω0avg, δω0]
```

Out[177]=
```
4.404258 ± 0.000027
```

In[178]:=
```
γ = Around[γavg, δγ]
```

Out[178]=
```
0.14145 ± 0.00006
```

You can use the ‘Around’ function to do calculations with uncertainty. For example, here is the function for computing Q:

In[179]:=
```
Q[ω0_, γ_] := ω0 / γ
```

```
In[180]:=
myQ = Q[ω0, γ]
Out[180]=
31.137 ± 0.014
```

Or, calculating the uncertainty the long way:

```
In[181]:=
```

$$Q[\omega 0avg , \gamma avg] * Sqrt\left[\left(\frac{\delta\omega 0}{\omega 0avg}\right)^2 + \left(\frac{\delta\gamma}{\gamma avg}\right)^2\right]$$

```
Out[181]=
0.0136006
```

This also works for messier equations, as long as the uncertainties are all “small”.

You can extract parts of an Around object with the “Value” and “Uncertainty” keys:

```
In[182]:=
myQ
Out[182]=
31.137 ± 0.014

In[183]:=
myQ["Value"]
Out[183]=
31.1372

In[184]:=
myQ["Uncertainty"]
Out[184]=
0.0136006
```