Efficiently processing data files

In[97]:=
```
Clear["Global`*"]; DateString[]
SetDirectory[NotebookDirectory[]];
```
Out[97]=
```
Wed 26 Feb 2025 14:14:31
```

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[99]:=
```
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[100]:=
```
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[101]:=
```
TableForm[rawdata⟦Range[3]⟧]
```
Out[101]//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[102]:=
```
Select[rawdata⟦Range[3]⟧, Length[#] == 2 && NumberQ[#⟦1⟧] && NumberQ[#⟦2⟧] &]
```
Out[102]=
```
{{0., 0.4953}, {0.002, 0.490723}}
```

It’s not necessary here, but it turns out there’s a 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[103]:=

```
Select[rawdata[[ Range[3] ]], Length[#] == 2 && VectorQ[#, NumberQ] &]
```

Out[103]=

```
{{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[104]:=

```
data = Select[rawdata, Length[#] == 2 && VectorQ[#, NumberQ] &];
TableForm[data[[Range[3]]]] (* Again, look at the first few lines. *)
```

Out[105]//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:

In[106]:=

```
dθdV = 0.558; (* From my voltage calibration curve *)
```

In[107]:=

```
θvst = Table[{data[[i, 1]], dθdV * data[[i, 2]]}, {i, 1, Length[data] }];
TableForm[θvst[[ Range[3] ]] ]
```

Out[108]//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.

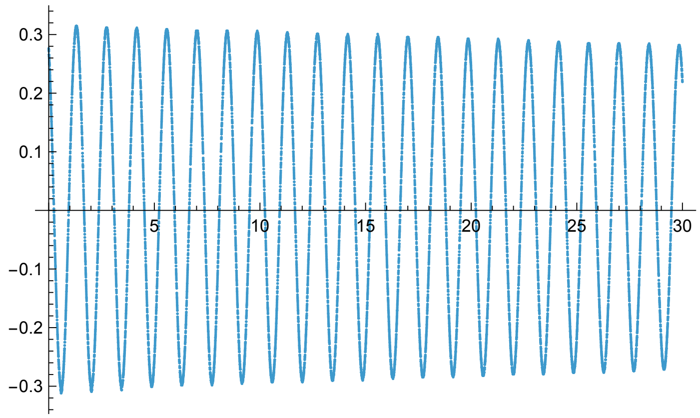
In[109]:=

```
(* 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[110]:=

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

Out[111]=



Example: Finding the undamped frequencies

In[112]:=

```
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.

In[113]:=

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

Out[113]=

```
FittedModel[ 0.00719 + 0.311 ≪1≫ Sin[1.06 − ≪1≫] ]
```

In[114]:=

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

Out[114]=

```
{θ0 → -0.310506, γ → 0.00814831, ω → 4.40419, ϕ → -1.05874, θoff → 0.0071934}
```

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

In[115]:=

```
ω /. fit["BestFitParameters"]
```

Out[115]=

```
4.40419
```

Bundling it up:

In[116]:=
```
getωfit[data_] := Module[
  {fit, ωfit},
  fit =
   NonlinearModelFit[data, model[θ0, γ, ω, ϕ, θoff, t], {θ0, γ, ω, ϕ, θoff}, t];
  ωfit = ω /. fit["BestFitParameters"]
 ]
```

In[117]:=
```
(* Test it out*)
getωfit[θvst]
```

Out[117]=
```
4.40419
```

You can even chain the two functions together:

In[118]:=
```
getωfit[getFile["TO-20250220-undamped-1.csv", dθdV]]
```

Out[118]=
```
4.40419
```

Operating on multiple files

Of course this 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 FIle-names[ ] command returns a list (in curly braces) of files matching a specification. Using the '*' wild-card, I can list out my five files:

In[119]:=
```
FileNames["TO-20250220-undamped-*.csv"]
```

Out[119]=
```
{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.

In[120]:=
```
ω0s = Table[getωfit[getFile[f, dθdV]],
  {f, FileNames["TO-20250220-undamped-*.csv"]}]
```

Out[120]=
```
{4.40419, 4.40434, 4.40426, 4.40422, 4.40429}
```

In[121]:=
```
Mean[ω0s]
```

Out[121]=
```
4.40426
```

In[122]:=
```
δω0 = StandardDeviation[ω0s] / Sqrt[Length[ω0s]]
```
Out[122]=
```
0.0000266678
```

Expressing the uncertainty with Around[ ]

In[123]:=
```
ω0 = Around[Mean[ω0s], δω0]
```
Out[123]=
```
4.404258 ± 0.000027
```

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

In[124]:=
```
ωvs = Table[getωfit[getFile[f, dθdV]], {f, FileNames["TO-20250220-damped-*.csv"]}]
```
Out[124]=
```
{4.41284, 4.41285, 4.41277, 4.41271, 4.41272}
```

In[125]:=
```
Mean[ωvs]
```
Out[125]=
```
4.41278
```

In[126]:=
```
δωv = StandardDeviation[ωvs] / Sqrt[Length[ωvs]]
```
Out[126]=
```
0.0000287787
```

In[127]:=
```
ωv = Around[Mean[ωvs], δωv]
```
Out[127]=
```
4.412777 ± 0.000029
```

Mathematica knows how to propagate uncertainty in many straightforward calculations.

In[128]:=
```
ω0 - ωv
```
Out[128]=
```
-0.00852 ± 0.00004
```

The uncertainty is the same as we get with the standard propagation formula:

In[129]:=
```
Sqrt[δω0^2 + δωv^2]
```
Out[129]=
```
0.000039235
```