# Handout: Handy Computer Tools

T. Satogata: January 2017 USPAS Accelerator Physics

January 2017

This is a description of a few computational tools that I've found to be useful as a working physicist. This handout is more to note that these tools exist, and to indicate how they might be used in class. It is not meant to replace full documentation for any of these tools, nor is it meant to imply that any of these tools are any better or worse than what you might already be using!

## 1 Google

Google functions very well as a single-line scientific calculator. There are many useful constants that Google knows, such as $\epsilon_0$ (`epsilon_0`), $\mu_0$ (`mu_0`), $\pi$ (`pi`), the elementary charge (`elementary charge`), and proton and electron masses (`m_p`) and (`m_e` respectively). Google will also intraconvert most units transparently, though it does not seem to handle Gauss very well (`G` is the gravitational constant). The `*` for multiplication is optional.

A few examples:

- `(4*pi*epsilon_0)*(hbar*c)/(elementary charge)^2`
  → `137.035999` (Fine structure constant, $\alpha^{-1}$)
- `(1 Tesla)*(1 m) in Volts/c`
  → `299 792 458 volts / c` (a useful unit conversion)
- `mu_0(50)(300 A)/(50 cm)`
  → `0.037699 teslas` (field of a 50-turn, 300 A, 50 cm long solenoid)

A nice Arxiv paper on the use of Google calculator for physics is located here: https://arxiv.org/pdf/physics/0411198.pdf.

## 2 Mathematica

**Mathematica** is the industry standard among symbolic manipulation codes, and it also does arbitrary-precision calculations. If you are fortunate enough to have a license (usually through a lab or student/educational discount), you probably already know how to use it. Matrices are entered with curly brackets, e.g. `{{1,L},{0,1}}`, and matrix multiplication and vector dot products are performed using a period, e.g. "`a.b`". Functions are capitalized and use square brackets to surround arguments, and `Simplify[]` is used to aggressively simplify algebraic expressions. One common stumbling block for a beginner is that the trace function is `Tr[]`; `Trace[]` is used to trace variables, not to calculate matrix traces.

## 3 Wolfram Alpha

The **Wolfram Alpha** "Computational Knowledge Engine" includes an interface to some simple but powerful and convenient Mathematica tools, including numerical and symbolic

```
In[12]:= qdh = {{1, 0}, {1 / (2 * f), 1}}
In[13]:= qfh = {{1, 0}, {-1 / (2 * f), 1}}
In[14]:= d = {{1, L / 2}, {0, 1}}
In[21]:= Simplify[qfh.d.qdh.qdh.d.qfh]
```

$$\text{Out[21]=}\ \left\{\left\{1-\frac{L^2}{8\,f^2},\ L+\frac{L^2}{4\,f}\right\},\ \left\{\frac{L\,(-4\,f+L)}{16\,f^3},\ 1-\frac{L^2}{8\,f^2}\right\}\right\}$$

```
In[31]:= Tr[Simplify[qfh.d.qdh.qdh.d.qfh]]
```

$$\text{Out[31]=}\ 2-\frac{L^2}{4\,f^2}$$

Figure 1: A simple Mathematica FODO cell calculation.

analysis:

`http://www.wolframalpha.com`

Its big drawback is that you cannot define variables, but it has two big advantages: it's free, and it's in a browser so it's portable. It's extremely useful for one-off calculations, particularly when you can edit your entry in a text editor and then cut and paste it into the Alpha query window. See Fig. (2) for a simple FODO cell matrix calculation in Alpha.

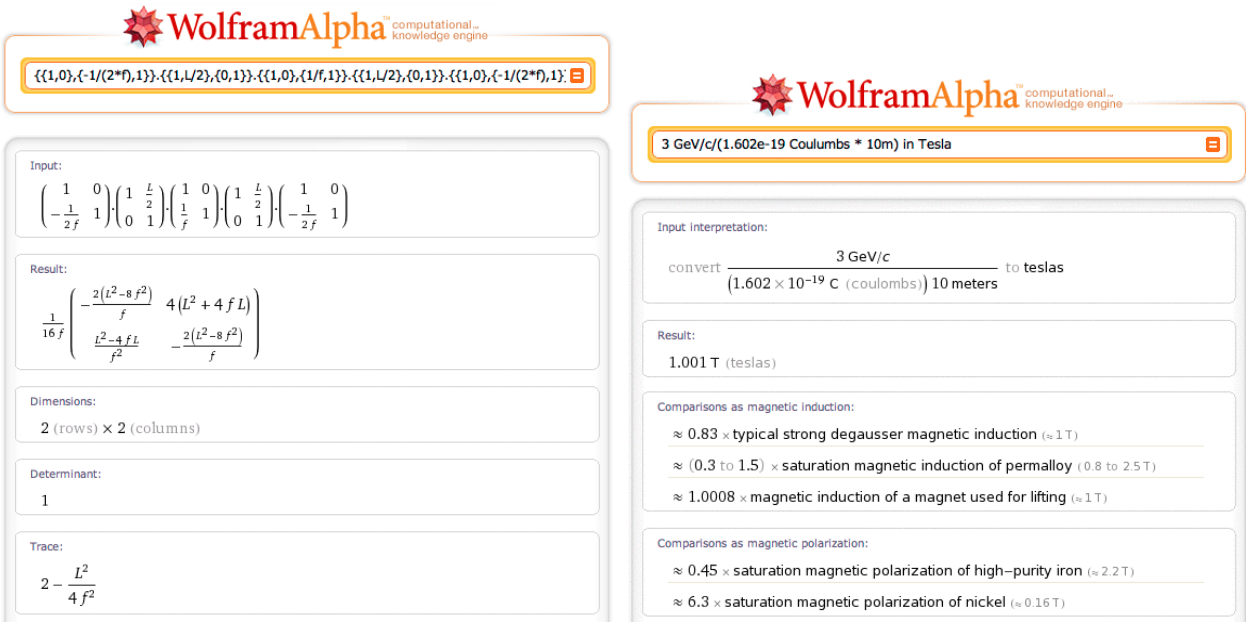


Figure 2: A simple FODO cell calculation, and a calculation of required magnetic field to bend a $q = e$, $p$ =3 GeV/c particle through a bending radius of $\rho$ =10 m in Wolfram Alpha.

Both **Google** and Alpha also serve as excellent symbolic calculators and unit converters. For example, type

$$\texttt{3 GeV/c / (1.602e - 19 Coulombs * 10m) in Tesla} \qquad (1)$$

to calculate that it takes a 1 T magnet to bend a $q = e$, $p$ =3 GeV/c particle with a bending radius of $\rho$ =10 m. Also see Fig. (2). These nice online unit-friendly calculators will often get confused about certain symbols. For example, they get confused between $e$ as the base of natural logarithms vs the charge of the electron. In the above example I just used a number I know, but Alpha also will recognize it if you type `q_e`, while Google will recognize it if you type `elementary charge`.

Alpha also gives some comparisons to magnetic induction and polarization. Such a magnet

would be comfortably under saturation if made out of good iron, but would be horribly saturated and probably of bad field quality if made out of nickle.

## 4 yacas

One of Wolfram Alpha's drawbacks is that it only evaluates single statements; it's not really a full free interpreter. Fortunately there are a few free symbolic algebra interpreters out there of varying complexity. Perhaps the most lightweight is **yacas**, yet another computer algebra system, available with source code from

`http://www.yacas.org`

yacas went through a period of dormancy but is being maintained again, and it has a reasonably active user community. You can download the source code and compile it fairly easily for a simple command-line interpreter. There is also a console view (beneath the "Yacas Online" link on the above website) that can be used from a web browser connected to the internet. The syntax is close to Mathematica, but `:=` is used for variable assignment, functions delineate arguments with parentheses, and `*` is used for matrix multiplication.

Yacas also does not simplify algebraic expressions by default, so you will find yourself using `Simplify()` in almost every evaluation. Yacas also has a hard time automatically reducing polynomial fractions, as seen in Fig. (3). Nonetheless, it is useful for simple symbolic and algebraic calculations, and it can also perform numerical evaluation. However, yacas does not integrate with common plotting programs such as **gnuplot**.

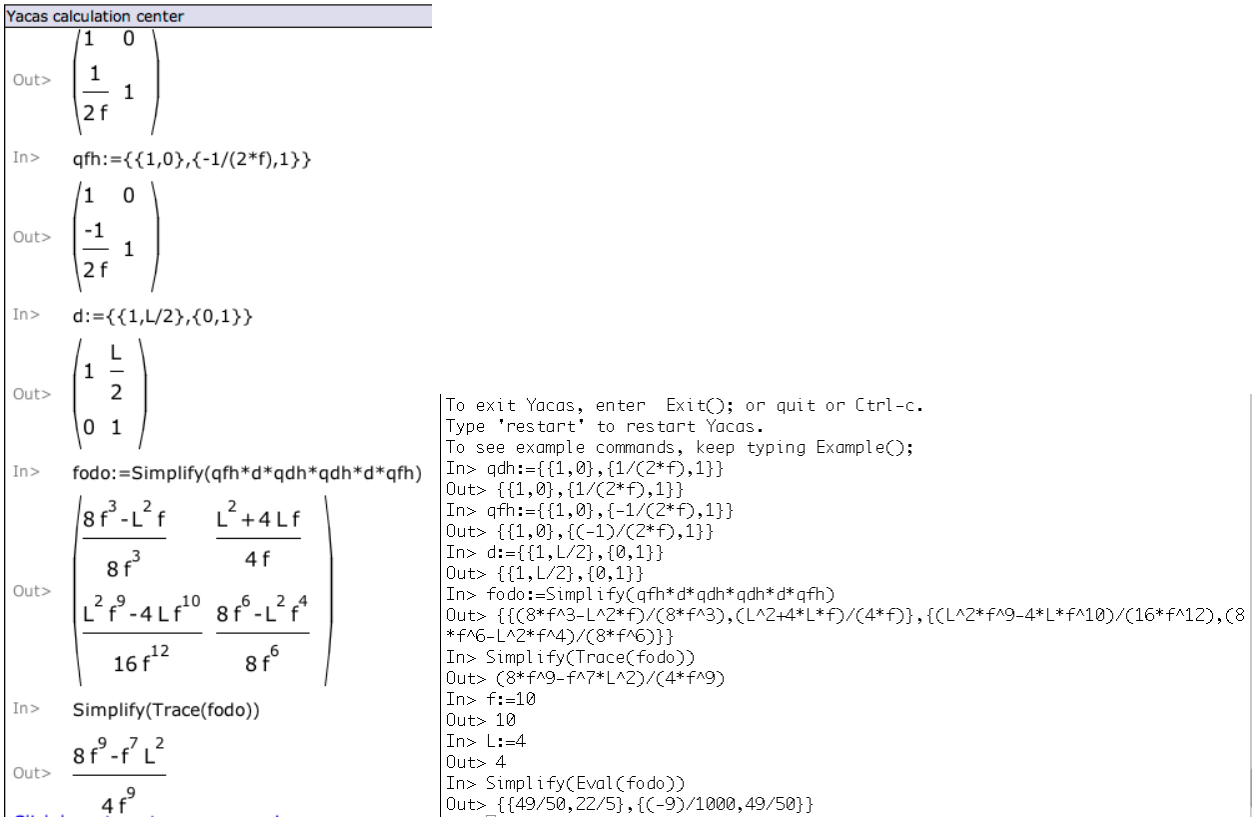


Figure 3: A simple FODO cell calculation in the web interface (left) and terminal interface (right) to yacas. The terminal interface example also shows numerical evaluation.

# 5 maxima

**maxima** is much more heavyweight (the user's guide is nearly 1000 pages long), but is also very well-supported and free. It is derived from Macsyma, a symbolic algebra system developed by MIT and DOE from the late 1960's to 2000. It is available with source code from

`http://maxima.sourceforge.net`

maxima's symbolic interpreter is implemented in LISP, no surprise since it originated at MIT. The syntax is thus a bit more arcane to those who are familiar with the Mathematica expression style; it will be more familiar to those who program elisp in emacs. For example, assignments are made with `:`, not with an equals sign as in Mathematica or `:=` as in yacas. Matrices are declared with the `matrix()` function. Instead of Simplify(), you should generally use `factor(expand())` around most algebraic expressions to simplify them as much as possible. All statements must end with a semicolon; you can get very confused if you forget it as the interpreter will look like it's hung waiting for your input. An example interpreter session is shown in Fig. (4).

```
(%i1) d: matrix([1,L/2],[0,1]);
                                  [    L ]
                                  [ 1  - ]
(%o1)                             [    2 ]
                                  [      ]
                                  [ 0  1 ]
(%i2) qdh: matrix([1,0],[1/(2*f),1]);
                                  [  1   0 ]
                                  [        ]
(%o2)                             [  1     ]
                                  [ ---  1 ]
                                  [ 2 f    ]
(%i3) qfh: matrix([1,0],[-1/(2*f),1]);
                                 [   1    0 ]
                                 [          ]
(%o3)                            [    1     ]
                                 [ - ---  1 ]
                                 [   2 f    ]
(%i4) fodo: factor(expand(qfh.d.qdh.qdh.d.qfh));
                          [    2      2                 ]
                          [   L  - 8 f   L (L + 4 f)    ]
                          [ - ---------  -----------    ]
                          [         2         4 f       ]
                          [     8 f                     ]
(%o4)                     [                             ]
                          [                   2      2  ]
                          [ L (L - 4 f)     L  - 8 f    ]
                          [ -----------   - ---------    ]
                          [         3            2      ]
                          [    16 f          8 f        ]
(%i5) load(functs)
;
(%o5)   /opt/local/share/maxima/5.24.0/share/simplification/functs.mac
(%i6) tracematrix(fodo);
                                    2      2
                                   L  - 8 f
(%o6)                             - ---------
                                         2
                                    4 f
```

Figure 4: A simple FODO cell calculation in the terminal interface for maxima.

maxima also has the advantage of integrating well with **gnuplot**, so it can be used to plot fairly complex functions (see pp. 117 and onward of the manual). There are also many nice GUI programs available, such as **wxmaxima**; these give a much nicer display than maxima from the command line.

# 6 Matlab

You may also be familiar with **Matlab**, a numerical analysis program that is widely used particularly for signal processing and excellent numerical matrix analysis tools. It excels at analysis of large sparse matrices. Matlab does not perform any symbolic algebra. Matlab can be even more costly than Mathematica, but it is particularly useful for design and modeling of time-varying systems with a powerful graphical environment called **Simulink**.

The **AT accelerator toolbox**, developed at SLAC and available at

`http://ssrl.slac.stanford.edu/at`

is particularly popular for accelerator modeling and high-level controls at many light sources around the world. It is implemented fully within Matlab, and requires Matlab licenses to run.

# 7 Octave

Since Matlab can be expensive (several thousands of dollars for a single license), there is a free program called **Octave** that replicates much of the numerical and matrix analysis with a very similar syntax. Octave is part of the GNU project but is not directly developed or supported by the FSF (Free Software Foundation). Octave is installed on the Linux systems in many computer labs. Octave does not have built-in graphics like Matlab does, so Matlab user interface code (including AT) is not trivially portable to Octave. However, Octave can produce many plots that Matlab can by invoking gnuplot as a plotting tool.

Matlab and Octave have a syntax much closer to C. Assignment is done with = and matrix multiplication is done with *. Matrices are surrounded with square brackets, with commas separating elements and semicolons separating rows. A semicolon at the end of a statement also suppresses output. A simple numerical FODO cell example, including calculation of eigenvalues and phase advance/cell, is shown in Fig. (5).

```
octave-3.0.3:1> f=11.1312;            # focal length [m]
octave-3.0.3:2> l=29.79;              # fodo cell length [m]
octave-3.0.3:3> qdh=[1,0;1/(2*f),1];  # Half defocusing quad matrix
octave-3.0.3:4> qfh=[1,0;-1/(2*f),1]; # Half focusing quad matrix
octave-3.0.3:5> d=[1,l/2;0,1];        # Drift matrix
octave-3.0.3:6> fodo=qfh*d*qdh*qdh*d*qfh  # fodo cell, printed
fodo =

    0.104703   49.721456
   -0.019892    0.104703

octave-3.0.3:7> [V,lambda]=eig(fodo)  # Find and print eigenvalues, eigenvectors
V =

   0.99980 + 0.00000i   0.99980 - 0.00000i
  -0.00000 + 0.02000i  -0.00000 - 0.02000i

lambda =

   0.10470 + 0.99450i   0.00000 + 0.00000i
   0.00000 + 0.00000i   0.10470 - 0.99450i

octave-3.0.3:8> # Calculate phase advance/cell mu for this FODO cell next
octave-3.0.3:8> mu=acos(real(lambda(1,1)))*360/(2*pi)
mu =  83.990
```

Figure 5: A simple numerical FODO cell calculation in Octave.