Moving ADAS Infrastructure to Python

Overview, approach and objectives

ADAS Workshop, 30th September 2014

Allan Whiteford

256 Kelvin Limited
(0.02 eV Limited for fusion people)

Contents

- Some background.
- Is Python the future?
- GUI programs.
- Callable ADAS:
  - running programs,
  - extracting data,
  - reading ADF files.
- Searching for data.
- Some (radical?) suggestions.
- Next steps.

Some background

- Core ADAS has a heavy reliance on IDL:
  - costly for ADAS members,
  - element of risk (if it stops being produced),
  - sometimes a sticking point for new members,
  - was about the only viable option in 1994!
- 20 years later we have more options, we certainly wouldn't pick IDL if we had to make the decision in 2014.
- Foresight and ongoing policy since the inception of the project has kept the core code in Fortran.
- Converting from IDL to something else doesn't require a re-write of (much) atomic physics.

Is Python the future?

- Current position:
  - emerging (some would say emerged) as the dominant high level language for scientific computing,
  - extensive support for arrays in the form of NumPy,
  - higher level bundles like SciPy are available,
  - already used significantly in fusion and astrophysics.
- Licensing:
  - Python is available at no cost,
  - permissive license doesn't restrict ADAS licensing options,
  - may need to be careful with (some) GPL libraries.

GUI programs, two approaches

- Native Python GUI:
  - numerous options but TkInter and wxPython are probably the front-runners,
  - GTK and Qt also possibilities,
  - on top of that require separate plotting libraries.
- Web-based GUI:
  - using python as the back-end to an HTML/JS front end,
  - slightly more complex in terms of moving parts,
  - same requirement of separate plotting libraries,
  - closer to current industry approaches in many respects.

GUI approaches, pros and cons

- Native Python GUI has significant technology advantages:
  - easier to use, less technology to understand and maintain,
  - no complications about local and remote files,
  - will (in theory) give faster performance.
- Web-based GUI gives more flexibility in what we can do:
  - all the user needs is a web browser,
  - can install ADAS centrally in a lab,
  - can use ADAS over the Internet,
  - can also run locally.

(We're trying the latter, more ambitious, approach for the pilot but not yet fully committed to it.)

Different approaches for different series

- ADAS Series 2, 3, 4, 7 and most of 8 will quite simply convert a screen at a time;
  - the **input** → **process** → **output** model works well here.
- We could do that with other series but, rather, we see this as an opportunity to:
  - mothball or completely re-work series 1,
  - make series 5 more interactive, focusing on data exploration,
  - push series 6 towards extended-ADAS and/or re-think it.
- Scope of the current work is to do a full conversion on ADAS205 as a ‘representative’ program and produce suggestions and mock ups for how series 5 programs might look.

Callable ADAS

- This project will **not** seek to retire, deprecate or otherwise endanger calling ADAS from IDL;
  - run_adas405 + read_adf15 $\rightarrow$ feeding in to your IDL code to model a plasma to give an integrated line emission is safe!
- We want to build an alternative and complementary interface in Python with the following key attributes
  - Pythonic API — not just a routine by routine conversion,
  - performant — almost certainly running on NumPy,
  - fully compatible with IPython,
  - building blocks for GUI code,
  - able to replace IDL for those who want to do that.

What would it look like?

- We looked (and are still looking) at:
  - SunPy,
  - FAC,
  - MDS+,
  - DAG,
  - PyQuante,
  - efit.py.
- Some of these aren't very Pythonic in nature - you can see the Fortran origins shining through with a hidden state stored deep in common blocks.

What would it look like?

- So we looked are more core libraries slightly outside of science:
  - curl,
  - sqlite,
  - a POV-ray library,
  - PyODE,
  - lots of game programming libraries.
- Then we pretty much ignored them all!
- We want something which feels like Python but still feels like ADAS.
- (Hence, among other things, everything will still be numbered.)

Ok, let’s look at what it actually looks like...

ADAS Python API

```
        from ADAS import ADF
        adf15 = ADF(15,'/path/to/adf15.dat')
        adf15.temp=[1,2,5,10,20,50,100]
        adf15.dens=[1e13,1e13,1e13,1e13,1e13,1e13]
        adf15.block=1
        coeff=adf15.extract();          # like read_adf15
#or
        raw=adf15.raw();                # like xxdata_15
```

Hidden features:

- If you ask your question using Python lists, you'll get the answer in a list (easy for beginners... and to make talks look simpler).
- If you ask your question using NumPy arrays, you'll get the answer in an array (better performance and all the benefits of NumPy).

ADAS Python API (continued)

```
from ADAS import ADAS
equil = ADAS(405)
equil.temp=[1,2,5,10,20,50,100]
equil.dens=[1e13,1e13,1e13,1e13,1e13,1e13]
equil.elem='he'
equil.calculate();
stages=equil.stages;
frac=equil.frac;
print frac
```

- Same story with regards to lists vs NumPy.
- Can also do:

```
from ADAS import ADAS405
equil = ADAS405()
```

ADAS Python API (continued)

- Ok, not all things have numbers...

```
from ADAS.atomic import continuo
brem=continuo()
brem.z0=6
brem.z1=7
brem.tev=3000
brem.wave=6000
brem.calculate()
contff=brem.contff
contin=brem.contin
```

- or:

```
contff , contin = brem.calculate()
```

Midplane emission through a fusion device

- Import some things:

```
from ADAS import ADAS
from ADAS import ADF
import numpy as np
from scipy.integrate import simps
```

- Set up grid of radial points on the midplane:

```
a = 0.7
r=a*(np.arange(101)/50.0 - 1.0)
```

- Create temperature and density profiles along the midplane:

```
temp=10.0 + 3e3 * (1-(r/a)**2)
dens=1e10 + 1e13 * (1-(r/a)**2)**0.5
```

- Calculate fractional ionisation balance of Ne along midplane:

```
pop = ADAS(405)
pop.temp=temp
pop.dens=dens
pop.elem='ne'
pop.calculate()
frac=pop.frac
```

- Extract emissivity coefficient for $1s^2 2p\ ^2P - 1s^2 2s\ ^2S$ transition:

```
adf = ADF(15,'/.../adas/adf15/pec96#ne/pec96#ne_pju#ne7.dat')
adf.block=1
adf.temp=temp
adf.dens=dens
adf.extract()
coeff=adf.coeff
```

- Calculate emissivity profile:

```
emissivity=frac[:,7]*coeff*dens
```

- Calculate emission by (naive) integration

```
emission=simps(emissivity,r)
```

- Output the answer:

```
print emission
```

- This calculation gives me an answer of **19.82** using our prototype code.
- IDL gives me **20.23** for the equivalent commands.
- Difference is just down to the sloppy integration at the end, not ADAS — inspection of 'emissivity' shows them as identical.

An External API?

- Using a combination of:
  - a well defined callable ADAS API,
  - our interactive programs being web-based.
- We get, almost for free, an ‘external’ API which could be used for:
  - a dedicated ADAS node in a modelling cluster which does on-demand atomic physics (e.g. ITM-style scenario),
  - similarly but on a lab-wide setting (just as there is an API to get experimental data there can be one to get atomic data),
  - optionally (not a personal fan) remote calculations and data provision over the internet.

What would it look like?

Send a JSON file *(you could use XML if you want)* like this:

```
{'prog': 'adas405',
 'elem': 'he',
  te:    [   1,   2,   5,  10,  20,  50, 100],
  dens:  [1e13,1e13,1e13,1e13,1e13,1e13,1e13]
}
```

Get back a JSON file like this:

```
{'frac':
  [[1.0E+00,6.3E-01,1.0E-03,3.4E-06,3.5E-08,6.7E-10,6.2E-11],
   [3.3E-06,3.7E-01,9.4E-01,3.4E-02,1.1E-03,7.7E-05,2.1E-05],
   [5.8E-27,8.1E-10,5.9E-02,9.7E-01,1.0E+00,1.0E+00,1.0E+00]],
 'stage':['He+0','He+1','He+ 2']
}
```

Searching for data

- Navigating the ADAS directory structure is complicated, even for experienced users.
- We already have OPEN-ADAS which allows searching by:
  - full text ‘intelligent’ search,
  - cross dataset by ion,
  - by specific data class,
  - cross dataset by wavelength of interest.
- Moving to a web-based interactive front-end will allow for lots of re-use of OPEN-ADAS searching.
- Note this doesn’t really resolve the perennial question of ‘which data should I use?’

Verb → Noun or Noun → Verb?

- Traditionally, one selects an ADAS program then selects a file.
- However, on a conventional computer system you tend to locate a file then open it.
- No reason why we can't, for users who want it, turn ADAS inside out whereby you navigate by data then have options of processing the data in various ADAS programs
- Similarly, the output of ADAS205 (i.e. `countour.pass`) is typically fed into ADAS207. Giving the user the ability for this to happen automatically will aid in initial comprehension of ADAS workflow.
- This is one possible change but only where users want it — you would still be able to select by program.

More interactive programs

- ADAS series 5 is used for data exploration and extraction.
- Tied to the **input** $\rightarrow$ **process** $\rightarrow$ **output** paradigm.
- This is fine for interrogation (from a time reversal point of view, ADAS503 is just an interactive version of read_adf15).
- Being able to interactively alter the transitions, ranges etc. and have the data appear in real time would help with the exploratory aspects.
- Output options would still exist for extraction.
- And also perhaps producing an embeddable read_adf15 command or program following a similar model as laid down in ADAS605.

Summary

- Next steps:
  - produce working prototypes of all of the above,
  - argue (extensively) with Martin over naming conventions,
  - circulate the pilot prototypes more widely,
  - await feedback.
- Please let me or Martin know if you'd be willing to test out the various programs when they are ready.
- It is likely that this pilot will inform the decision of the steering committee as to whether committing to a full conversion at this time is prudent or not.

Thank you!