1. Vectors for ADQL

Markus Demleitner
msdemlei@ari.uni-heidelberg.de

- Context: ADQL
- Arrays? Vectors?
- Proposed Operations
- Application to Gaia XP Spectra

Funded by e-inf-astro, BMBF FKZ 05A20VH5

2. Context: ADQL

The Astronomical Data Query Language ADQL is

- . . . a SQL derivative intended to shield astronomers from all the variations of RDBMSes
- . . . used with the Table Access Protocol on almost 150 TAP services (among them VizieR, Gaia, Simbad)
- . . . arguably the closest thing we have to interoperable code-to-data at the moment.

If you're new to ADQL: Grab a TOPCAT and head over to http://docs.g-vo.org/adql.

3. Vectors

"Vector" here is supposed to mean "one-dimensional array". Aplogies to all physicists.

1D arrays in database tables in the GAVO Heidelberg data centre:

- spectra (flux, errors, spectral coordinate in separate arrays)
- time series (also as multi-array)
- photometric points (with other per-band metadata)
- per-axis metadata on cubes
- error models including covariances
- (theoretically, geometries, but these are not really vectors in the database)

To get an idea of what sorts of non-string arrays a TAP service has, try

```
SELECT * FROM tap_schema.columns
WHERE arraysize IS NOT NULL
  AND (NOT datatype LIKE '%char' AND arraysize!='*')
```

At this point, not many TAP services carry interesting array data, and only few (but: ARI-Gaia does!) support anything like the proposed vector extension.

4. Operations I: Componentwise

Where I can copy numpy, I do, so there is:

- `vec[index]` – component access. Since I think most SQL engines work that way, index is for now 1-based.
- `vec1[+-*/]vec2` – component-wise arithmetic.

In component-wise operations of vectors of unequal length, we currently raise no error but pad the shorter with NaNs. Footgun?

All operations assume floating point data.

5. Operations II: Vector Operations

These are operations actually well-defined for true vectors:

- `scalar*vec`, `vec*scalar`, `vec/scalar` – normal scalar multiplication
- `arr_dot(vec1,vec2)` – dot product. By our padding rule, that's NaN for unequal-length vectors.

Would a 3D cross product be sufficiently necessary to be included here?

6. Operations III: Array Aggregation

These are the well-known SQL aggregate functions, where each vector is an aggregate:

- `arr_avg(arr)` – the arithmetic mean of. . .
- `arr_max(arr)` – the largest among. . .
- `arr_min(arr)` – the smallest among. . .
- `arr_sum(arr)` – the sum of. . .
- `arr_stddev(arr)` – the standard deviation of. . .

. . . the *elements* of an array.

There's also `arr_count(arr)` in analogy to SQL COUNT: the "array length".

7. Operations IV: Aggregate Arrays

The well-known SQL aggregate functions are very useful for arrays, too, where they work based on the componentwise operators. That's

- AVG
- MIN
- MAX
- SUM

More ADQL aggregate functions have been proposed (e.g., histograms); where they can be written in terms of vector components, they should work for arrays, too.

8. Operations V: Array Map

It turns out the numpy ufuncs (basically, componentwise application of normal functions) are *really* are too useful to not have something equivalent. Currently implemented:

`arr_map(power(10, x), mags)`

– that is, you write an expression over a literal `x`.

Yeah, that's a bit lame, not only because it might shadow actual SQL identifiers. Perhaps use the array name instead of `x`?

9. Operations VI: Slicing

Currently not implemented but clearly necessary: Slicing.

- `nflux[4:7]` – perhaps too close to Python given we probably want to keep 1-based indexes?
- `arr_slice(nflux, 4, 7)`?

Postgres has Python syntax but its own semantics (1-based indexes, upper bound included). We'll certainly confuse *someone*.

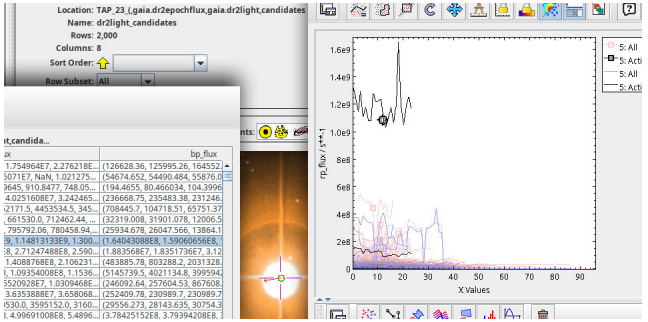

Fig. 1

10. Example: Epoch Photometry

Imagine you are looking for stars changing their colour in Gaia DR2 epoch photometry. Well:

```
with candidates as (
  select top 2000 * from (
    select source_id, rp_flux, bp_flux,
      arr_stddev(bp_flux-rp_flux) as colordiff,
      arr_avg(rp_flux_error) as fluxerror
    from gaia.dr2epochflux
    where arr_stddev(bp_obs_time-rp_obs_time)<0.01
      and arr_count(rp_flux)>5) as q
  order by colordiff desc)
select candidates.*, ra, dec, phot_g_mean_mag
from gaia.dr2light
join candidates using(source_id)
```

What's going on here?

The first tricky thing is `arr_stddev(bp_obs_time-rp_obs_time)<0.01`. This filters out records for which the photometry in RP and BP isn't roughly at the same times; either because measurements are missing (in which case this is NaN) or because measurements are too far apart.

Then there ie `arr_stddev(rp_flux-bp_flux)`. This computes BP-RP colour for all the observations and computes the standard deviation of the resulting values. That's a reasonable metric for "variable in colour".

With `arr_count(rp_flux)>5`, I'm throwing out lightcurves dominated by too few points, and `arr_avg(rp_flux_error)` lets me diagnose later whether I have really noisy data.

The rest is fairly standard ADQL fare; I'm selecting the 2000 light curves showing the highest colour variablity and pull position and G-magnitude out of gaia_source (or its lite cousin in this case).

11. Playing with the result

TOPCAT has recently grown a bunch of nice features to deal with array-like data:

(cf. Fig. 1)

What did I do here?

The key is the *Add XYArray Control* entry from the Layers menu in TOPCAT's plane plots. Add these for both the RP and the BP arrays. You then want to configure light tones for their main

subset and something strong for the *Activated* subsets (also, check them). Also enable *Handles* in the Form tabs: that lets you select curves and have them reflected in table views.

Outside of the plot, you want to start Aladin, enable its DSS and Simbad planes and check "transmit coordinates" in TOPCAT's activation actions. This lets you select lightcurves and in Aladin see what the object looks like and what Simbad has to say about it.

The thing in the figure is g Her, a mag 2 (so: Gaia photometry may be a bit wonky in the red) AGB star that shows, by our metric, the strongest colour variability. The next match is V Mon, classified by Simbad as "S Star", presumably also on the AGB. Places 3 to 5 are held by T Lep, S Pic, and S Vir, all known Mira variables. Place 6 is known to Simbad as IRAS 15373-6101, listed as an LPV *candiate*, followed by ζ Gem, a (wow) Cepheid.

Whether we can learn something from this except that AGB stars change their colours a lot? Presumably not without being a lot more careful with the errors (and in the case ζ Gem, it seems there is some instrumental trouble). But you may get the idea.

12. Major Open Questions

- Slicing, ufuncs as above
- Value pairs/interpolation for unevenly sampled data (e.g., spectral/flux, time/flux).

13. Read on

The (editable) proposal[1]

A blog post on working with spectral arrays[2]

A tutorial we'll soon rewrite to use server-side arrays[3]

. . . Thanks!

1 https://wiki.ivoa.net/twiki/bin/view/IVOA/ADQLVectorMath

2 https://blog.g-vo.org/a-proposed-vector-extension-for-adql.html

3 http://www.g-vo.org/tutorials/dfbs.pdf