Dynamic Analysis for Real Time Imaging

This brief sketch is an adjunct to the GeoPIXE manual. Refer to the manual for a guide to spectrum fitting and building the Dynamic Analysis matrix.

The process of using a Dynamic Analysis matrix for real-time image projection works something like this:

1. For each event, use the detector number to index to the correct energy calibration table for this detector.

2. Convert detector pulse-height channel number to energy (keV) using its calibration.

3. Now using the energy calibration of the DA matrix (often the same as the first detector in the array, but with a different offset), convert this energy to channel, or column in the DA matrix.

4. This column contains increments, one for each element, to make to the elemental images at the current X,Y position.

Some details modify this a little. It is nice to have the images quantitative and record ppm-flux units regardless of how many detectors are selected from a detector array. This is done by scaling the matrix column values down by the effective multiplicity, which is the sum of the "relative sensitivity factors". Similarly, we scale the variance image increments down by this number squared. This has not been done below.

This C code fragment illustrates the process for a single detector:

```
/*     Assume we have an event (N,E,X,Y) from detector N:

       matrix is array is         float  DA[columns][rows]
       matrix cal is              float  DA_calA, DA_calB

       detector calibrations:     float  calA[], calB[]

       images are arrays:         float  image[el][y][x]
       variance arrays:           float  var[el][y][x]
       number of elements:        long   N_el
*/
  long column;
  float e, f, dinc;

  e = E * calA[N] + calB[N];                    /* event energy     */
  column = (e - DA_calB) / DA_calA;             /* DA matrix col    */

  for (i=0; i<N_el; i++) {
    dinc = DA[column][i];

    image[i][Y][X] = image[i][Y][X] + dinc;
    var[i][Y][X] = var[i][Y][X] + dinc*dinc;
  }
```

These are quantitative image in the sense that we can at any time extract estimates of average elemental concentration (e.g. within an arbitrary shaped area, or along a profile traverse) by integrating an area of the `image` array and dividing by the integrated beam flux for the same area. Similarly, one-sigma uncertainty comes from integrating the variance array `var`, taking the square-root, and dividing by flux. Hence, concentration colour-bar legends can be applied in real-time, and interactive concentrations can be extracted from images, even during data collection (for [partly] completed parts of the image area).

The ‘export_DA.pro’ procedure dumps a binary file containing the matrix and other details, such as energy calibration and the identity of the active detectors.

Note that byte-ordering has already been fixed for a Little Endian target processor, such as a PC (Linux or Windows). This option is set in the ‘Export_DA2.pro’ routine.

The file has this format:

File is dumped as follows:

```
       long   version;                              /* version number   */
       struct header       header;                  /* header           */
       unsigned byte       b[N_el][ns];             /* element labels   */
       float               DA[N_el][size];          /* DA matrix        */
```

where the header struct is:

```
struct header
{
       long   N_el;         /* number of elements                        */
       long   size;         /* number of columns in DA matrix            */
       float  DA_calA;      /* energy cal gain of DA matrix              */
       float  DA_calB;      /* energy offset of DA matrix cal            */
       long   ns;           /* number of characters in element names     */
};
```

where the ‘b’ strings are null terminated and typically contain 4 chars (ns = 4, including NULL). All energy calibration data is in keV.

Note that the matrix does not usually extend all the way to zero energy, or zero original channel number. Only the fitted energy range is usually included. So the mapping using calibration of the matrix and detector elements is very important to pick up the correct column.

The first element is usually “Bac”, a truncation of “Back”, which is the background image and maps the variation of the underlying background component as fitted to the master spectrum. Other element labels are standard element mnemonics (2 chars plus NULL), perhaps with an appended ‘L’ or ‘M’ to indicate the use of L or M lines (3 chars plus NULL).

Back is a useful diagnostic, as any missing elements (i.e. elements present but omitted from the build of the DA matrix) usually contribute to the background image. Watch out for intense spots here that may indicate a concentration of an element that was initially not noticed but is actually significant in a small area of the image.

Some Refinements

Rather than calculate ‘e’ from initial channel number and then ‘column’ from ‘e’, a set of lookup tables ‘clookup’ can be built initially, one for each active detector:

```
  for (j=0; j<N_det; j++) {
    for (i=0; i<size; i++) {
       x = (float)i;
       clookup[j][i] = (long))(calA[j]*x + calB[j]-DA_calB)/DA_calA + 0.5);

       if (clookup[j][i] < 0)||( clookup[j][i] >=size) {
         clookup[j][i] = -1;
       }
    }
```

```
}
```

where the addition of 0.5 provides some simple rounding to form '`clookup`'. And then the main data projection loop from above simplifies to:

```
j = lookup[N];
f = 1./(float)N_det;
If (j >= 0) {
  column = clookup[j][E];

  if (column >= 0) {
    for (i=0; i<N_el; i++) {
      dinc = DA[column][i] * f;

      image[i][Y][X] = image[i][Y][X] + dinc;
      var[i][Y][X] = var[i][Y][X] + dinc*dinc;
    }
  }
}
```