ANPASS is a program for performing least-squares analysis. It was originally written in Fortran but has since been translated to Go and then to Rust by Brent R. Westbrook. This documentation corresponds to the Rust version only. In principle, ANPASS can be used to solve any polynomial or even linear regression problem, but in practice it is only used for determining the force constants in quartic force fields. As such, this will be the running example of usage in this document.

1 Overview

A quartic force field (QFF) is a fourth-order Taylor series expansion of the internuclear potential energy portion of the Watson Hamiltonian of the form

$$V = \frac{1}{2}\sum_{ij} F_{ij}\Delta_i\Delta_j + \frac{1}{6}\sum_{ijk} F_{ijk}\Delta_i\Delta_j\Delta_k + \frac{1}{24}\sum_{ijkl} F_{ijkl}\Delta_i\Delta_j\Delta_k\Delta_l \tag{1}$$

This can be rephrased as the matrix equation $\mathbf{XF} = \mathbf{V}$, where $\mathbf{F}$ is the (initially unknown) vector of force constants that minimizes the least squares difference $|\mathbf{XF} - \mathbf{V}|$, $\mathbf{V}$ is a vector of potential energies, and $\mathbf{X}$ is itself a matrix composed of the elements X_{ik} given by the expression

$$X_{ik} = \prod_{j}^{N} x_{ij}^{e_{jk}} \tag{2}$$

In turn, the x_{ij} for the QFF example are the jth components of the ith displacement, and the e_{jk} are the kth exponent in the jth row of exponents. This is similar to the basic polynomial regression problem given by

$$A = \begin{bmatrix} x_1^m & \dots & x_1^2 & x_1 & 1 \\ x_2^m & \dots & x_2^2 & x_2 & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ x_n^m & \dots & x_n^2 & x_n & 1 \end{bmatrix} \tag{3}$$

except that $N = 1$ in the basic form, and k is $\begin{bmatrix} m & m-1 & \dots & 0 \end{bmatrix}$ instead of varying in arbitrary ways. In other words, the exponents are given by the vector $\begin{bmatrix} m & m-1 & \dots & 0 \end{bmatrix}$ instead of the matrix with elements e_{jk}, and the x_{ij} are really the vector of elements x_i.

With this formulation, ANPASS can be used to solve the mth-order regression problem. For example, a linear regression problem for i data points is represented by $e = \begin{bmatrix} 1 & 0 \end{bmatrix}$. The one caveat to this is that the resulting function is only printed in the file `fort.9903` in the units expected for a QFF, so if you want the raw values back, divide your coefficient by 4.359813653. A full example of such a fitting is given in Appendix A.

2 Input format

The layout of the ANPASS input file is as follows:

2.1 Header

```
!INPUT
TITLE
H2O 2A1 F12-TZ
PRINT
99
```

```
INDEPENDENT VARIABLES
3
DATA POINTS
69   -2
(3F12.8,f20.12)
```

The only line read by the Rust version is the last. This line is parsed by the regular expression:

```
"(?i)^\s*\((\d+)f[0-9.]+,f[0-9.]+\)\s*$
```

which looks for a line of this general form and also captures the first number before an `f`. This first number is used as the number of columns of displacements in the next section and signals the beginning of that section.

2.2 Displacements

This section is composed of displacements in the first $N-1$ columns, followed by an optional column of energies. If the number of columns minus one is equal to the number parsed in the aforementioned regular expression, the first $N-1$ columns are taken as displacements, while the last column is taken as the corresponding energy. Otherwise, all N columns are taken as part of the displacement matrix. This allows for only the displacements to be input as part of a template.

```
-0.00500000 -0.00500000 -0.01000000       0.000128387078
-0.00500000 -0.00500000  0.00000000       0.000027809414
-0.00500000 -0.00500000  0.01000000       0.000128387078
-0.00500000 -0.01000000  0.00000000       0.000035977201
-0.00500000 -0.01500000  0.00000000       0.000048243883
-0.00500000  0.00000000 -0.01000000       0.000124321064
-0.00500000  0.00000000  0.00000000       0.000023720402
-0.00500000  0.00000000  0.01000000       0.000124321065
-0.00500000  0.00500000 -0.01000000       0.000124313373
```

2.3 Unknowns

This section simply lists the number of columns in the matrix that follows. This is necessary since the exponents are wrapped after 16 columns, so it would be possible for all of them to blend into a single row. This section looks like:

```
UNKNOWNS
22
```

2.4 Exponents

This section gives the exponents for the polynomial equation to be fit. It has the form given in the listing below.

```
FUNCTION
0    1    0    2    1    0    0    3    2    1    0    1    0    4    3    2
1    0    2    1    0    0
0    0    1    0    1    2    0    0    1    2    3    0    1    0    1    2
3    4    0    1    2    0
0    0    0    0    0    0    2    0    0    0    0    2    2    0    0    0
0    0    2    2    2    4
```

2.5 Stationary point

This section requests a refitting of the energies to a known stationary point. It looks like:

```
STATIONARY POINT
-0.000045311426      -0.000027076533       0.000000000000      -0.000000002131
```

Like the Displacement section above, the first $N-1$ columns are displacements used to bias the displacements in that section, while the last column is an energy used to bias each of the energies before performing the refitting. If this section is not present, a stationary point search is undertaken and a stationary point will be printed in the output.

3 Fitting

The first task of ANPASS is to find the vector F (of force constants in the QFF problem) that minimizes $|XF - V|$. This minimization process is often referred to as "fitting" since the discrete data points in the input are being fit to the continuous function described by the X matrix. With the matrix formulation further described in Section 1, the fitting process simply requires solving the ordinary least squares equation given in Eqn. 4.

$$F = (X^\intercal X)^{-1} X^\intercal V \tag{4}$$

Since $X^\intercal X$ is always positive semidefinite, the Cholesky decomposition can be used to compute its inverse. The rest of the solution is then straightforward.

4 Stationary Point

Newton's method is used to find the stationary point of the function described by the force constants, **F**, and the exponents, **E**. The basic idea of Newton's method is to find the stationary point by an iterative approach:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{5}$$

In words, update the current guess at the stationary point, x_k, by subtracting the ratio of the first derivative at that guess, $f'(x_k)$, and the second derivative at the same point, $f''(x_k)$.

In this case, **x** is actually a vector, and so is the first derivative ($f'(x_k)$) or gradient, **g**. The elements of **g** are computed as shown in Eqn. 6, where N is the number of unknowns (columns in **E**), and M is the number of variables (rows in **E** or columns in **X**).

$$g_i = \sum_j^N \left(e_{ij} F_j x_i^{e_{ij}-1} \prod_{k \neq i}^M x_k^{e_{kj}} \right) \tag{6}$$

Intuitively, this is the same as a derivative of a more conventional polynomial. For example, the derivative of $f = 4x^2yz$ with respect to x is $\frac{\partial f}{\partial x} = 2 \cdot 4xyz$. Mapping this on to Eqn. 6 shows that $2 \rightarrow e_{ij}$, $4 \rightarrow F_j$, $x_i \rightarrow x$, and the other $x_k \rightarrow y$ and z. Thus, $g_i = \frac{\partial f}{\partial x_i}$.

Similarly, the second derivative, written as $f''(x_k)$ in Eqn. 5, corresponds to the Hessian matrix, **H**, with diagonal elements shown in Eqn. 7 and off-diagonal elements shown in Eqn. 8. As in the case of the gradient, the subscript i runs from 1 to M. Note that **H** is symmetric, so the second subscript, l, for the off-diagonal elements, only runs to $i - 1$.

$$H_{ii} = \sum_{j}^{N} \left((e_{ij} - 1) e_{ij} F_j x_i^{e_{ij}-2} \prod_{k \neq i}^{M} x_k^{e_{kj}} \right) \tag{7}$$

$$H_{il} = H_{li} = \sum_{j}^{N} \left(e_{ij} e_{lj} F_j x_i^{e_{ij}-1} x_l^{e_{lj}-1} \prod_{k \neq i, k \neq l} x_k^{e_{kj}} \right) \tag{8}$$

Again, these both resemble the forms for normal second derivatives of polynomials, $\frac{\partial^2}{\partial x^2}$ in the first case and $\frac{\partial^2}{\partial x \partial y}$ in the second case, for example.

With the gradient and Hessian in hand, all that remains is to invert the Hessian matrix, multiply this inverse by the gradient, and subtract the resulting vector from the previous guess for the solution:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma \mathbf{H}^{-1} \mathbf{g} \tag{9}$$

Comparing Eqn. 9 to Eqn. 5, it becomes clear that there is an additional factor, γ. This represents an adjustable step size in the direction found by Newton's method. In ANPASS $\gamma = 0.5$, but other choices may work as well. Additionally, ANPASS chooses an initial guess of $\vec{0}$ for the value of $\mathbf{x}_0$.

Once every element of $\delta = \gamma \mathbf{H}^{-1} \mathbf{g}$ is less than the desired cutoff (1.1×10^{-8} in this case), the algorithm terminates. As a safeguard against infinite looping, the algorithm also terminates after 100 iterations if convergence is not reached. Upon successful termination, the stationary point is characterized to determine whether it is a maximum, minimum, or saddle point. This is straightforward given the final Hessian matrix. If all of the eigenvalues are negative, the stationary point is a maximum; if all of the eigenvalues are positive, the stationary point is a minimum; and if there is a mix of positive and negative eigenvalues, the stationary point is a saddle point.

5 Re-fitting

With the coefficients, $\mathbf{F}$, and the stationary point, $\mathbf{x}$, determined, the function can be re-fit to the new stationary point by evaluating the function at $\mathbf{x}$ and biasing each of the displacements and energies by subtracting $\mathbf{x}$ from each displacement and the energy at that displacement, $V(\mathbf{x})$, from each corresponding energy. Finally, these biased values can be fit again, using the same procedure described in Section 3.

6 Output

The force constants resulting from the re-fitting are then written out in the format expected by INTDER. In particular, the INTDER force constant I_{abcd} is given by Eqn. 10, where f is 1.0 if e_{ji} is 0 or 1, 2.0 if e_{ji} is 2, 6.0 if e_{ji} is 3, and 24.0 if e_{ji} is 4. In other words, f is the coefficient of the term in the Taylor series expansion of order e_{ji}. α is the conversion factor to get the force constants into the proper units, 4.359813653.

$$I_{abcd} = \alpha F_i \prod_{j}^{M} f \tag{10}$$

After constructing the force constants in this manner, they are written to the file `fort.9903`, again as expected by INTDER. It has the straightforward form shown in the listing below.

```
    0    0    0    0       0.000000000008
    1    0    0    0       0.000000094903
    2    0    0    0       0.000000008695
    1    1    0    0       8.360863692412
```

```
  2    1    0    0      0.364250381719
  2    2    0    0      0.705590041837
  3    3    0    0      8.562725561910
  1    1    1    0    -41.638868371768
  2    1    1    0     -0.611029974345
  2    2    1    0     -0.447356783198
  2    2    2    0     -0.701565547377
  3    3    1    0    -41.484904978169
  3    3    2    0      0.392943158463
  1    1    1    1    181.917347385520
  2    1    1    1     -0.292134838234
  2    2    1    1      0.372522603575
  2    2    2    1      1.034069183159
  2    2    2    2     -0.655831803345
  3    3    1    1    182.206178031047
  3    3    2    1     -1.233550191335
  3    3    2    2     -0.820459302767
  3    3    3    3    183.621273959614
```

A Linear regression example

The complete input for the linear regression example described in the Overview is shown below. The Fortran version fails to run on this example, and the Go version panics because it can't find a stationary point (since lines don't have stationary points). However, the Go version will still write `fort.9903`, which contains the coefficients of the line.

Listing 1: anpass.in

```
!INPUT
TITLE
H2O 2A1 F12-TZ
PRINT
99
INDEPENDENT VARIABLES
3
DATA POINTS
11   -2
(1F12.8,f20.12)
0.00000000       2.300000000000
1.00000000       3.400000000000
2.00000000       7.600000000000
3.00000000       8.100000000000
4.00000000       9.400000000000
5.00000000      13.600000000000
6.00000000      14.500000000000
7.00000000      15.900000000000
8.00000000      18.600000000000
9.00000000      21.700000000000
10.00000000      21.800000000000
UNKNOWNS
2
FUNCTION
1    0
END OF DATA
!FIT
!STATIONARY POINT
!END
```

Listing 2: fort.9903

```
  1     0     0     0       8.894019852120
  0     0     0     0       9.789763384464
```

Again, the results found here have been converted to the units needed for a QFF, so dividing them both by 4.359813653 gives the desired equation $y = 2.04x + 2.24$.