Documentation of C Inversion Library

Paul O'Brien (paul.obrien@aero.org)

November 17, 2010

Contents

1 Introduction

This document describes the functions provided by the inversion library "invlib.so" or "invlib.dll" for determination of a best-fit energy spectrum from a multi-channel measurement.

2 Changes

- 11/17/2010 Function descriptions expanded and new section added for Python interface and test suite (S. Morley)
- 5/28/2010 Updated discussion of principal components transform and maximum likelihood problem setup. Added figures showing example from CRRES p.c. model
- 5/27/2010 Created `pc_spec_inv` and `pc_spec_inv_multi` and associated documentation. Reorganized documentation a little.
- 1/6/2010 Created Matlab wrapper and fixed documentation for `omni2uni`.
- 3/13/2009 Fixed some pointer bugs in `ana_spec_inv_multi`, tested and fixed invlib.m.
- 3/10/2009 Added "Vampola" method for angular inversion (`omni2uni` and `wide2uni`).
- 3/9/2009 Fixed failure to initialize `ell` in `ana_spec_inv`.
- 3/5/2009 Added option to let `ana_spec_inv` do compute ΔE form `Egrid`. Minor fix to Makefile.
- 3/4/2009 Created invlib.m, changed δt to δt in equations.
- 3/3/2009 Made modifications to `ana_spec_inv` and `ana_spec_inv_multi` to support return of estimated counts.

  Modified definition of `support_data` to hold estimated counts for each analytical function after fit parameters.

  Added output `lambda` to hold estimated counts for combined fit.
- 3/2/2009 added `support_data` to `ana_spec_inv` and `ana_spec_inv_multi`.
- 3/2/2009 fixed bug in specinv that caused PLE to only do PL.
- 2/17/2009 Created `ana_spec_inv_multi`
- 2/13/2009 Changed `void *outFilePtr` to `char *outFile`, expanded meaning of verbose input code to allow output to a file (codes 4 and 5), and added a new output error value (-502) for unable to open the output file. (Thanks for Reiner Friedel for the suggestion)
- 9/2/2008 Added double relativistic Maxwellian (RM2) and Appendix A.
- 8/11/2008 Added power law with exponential tail (PLE) analytical spectrum to `ana_spec_inv`. Note: now `real_params` must have length 3 to use PLE.
- 4/25/2008 Library and documentation created and released.

3 Spectral Inversion Functions

The library provides several spectral inversion function, `ana_spec_inv`, `ana_spec_inv_multi`, `pc_spec_inv`, and `pc_spec_inv_multi`. These routines provide either analytical spectral inversions (power-laws and such), or principal component spectral inversions (PC's derived from observations or simulations). The `_multi` variants support multiple calls with the same energy response functions but different values for the counts in each data channel and duration of the time accumulation.

3.1 Calculations Performed

In continuous form, the spectral inversion functions solve

$$\vec{y} \approx \vec{\lambda} = \delta t \int_0^\infty \vec{G}(E) f(E) dE + \vec{b}, \tag{1}$$

where $\vec{y}$ is a vector of observed counts, $\vec{\lambda}$ is a vector of expected counts, δt is the integration time, $\vec{G}$ is a vector of energy-geometric factors (response functions), and $f(E)$ is the differential particle flux at energy E, and $\vec{b}$ is a vector of expected background counts. In this equation, all vectors in (1) have length N_y, the number of energy channels. Equation (1) is solved by parameterizing $f(E) = f(E;\vec{q})$ and determining the maximum likelihood value of the N_q free parameters $\vec{q}$. The maximum likelihood equation for $\vec{q}$ is invariably nonlinear, and so a nonlinear optimizer (a nonlinear minimizer) must be used once the problem is set up.

Throughout this subsection, we use the one-based linear algebra convention for vector and matrix subscripts.

3.1.1 Numerical Problem to be Solved

The first step toward a numerical solution to (1) is to discretize the integral:

$$\begin{aligned} \vec{y} &\approx \vec{\lambda} = \underline{\underline{H}}\vec{f} + \vec{b}, && (2) \\ H_{ij} &\approx \delta t G_i(E_j)\Delta E_j, && (3) \\ f_j &= f(E_j). && (4) \end{aligned}$$

Where $\vec{f}$ is now defined on a grid in energy, with N_E points, and $\underline{\underline{H}}$ has size $N_y \times N_E$. For a sufficiently fine grid in ΔE_j, the approximation for H_{ij} above should work well. However, slightly better choices are available if the grid is not coarse or unevenly spaced. It is usually worthwhile to use at least a trapezoidal integral, regardless of the grid:

$$\Delta E_j = \begin{cases} (E_{j+1} - E_j)/2 & j = 1 \\ (E_j - E_{j-1})/2 & j = N_E \\ (E_{j+1} - E_{j-1})/2 & \text{otherwise} \end{cases} \tag{5}$$

The plateau energy weight (not recommended) would be given by:

$$\Delta E_j = \begin{cases} (E_{j+1} - E_j) & j = 1 \\ (E_j - E_{j-1}) & j = N_E \\ (E_{j+1} - E_{j-1})/2 & \text{otherwise} \end{cases} \tag{6}$$

3.1.2 Measurement Penalty Functions

The next step is to define a measurement "penalty" function based on the likelihood of observing counts $\vec{y}$ given expected counts $\vec{\lambda}$. The observations differ from the expected counts due to a variety of possible error processes. The most common two errors are Poisson error and calibration error.

3.1.2.1 Poisson Error Penalty Function. The Poisson error arises from the finite number of counts observed, and is often referred to as "counting statistics" or "counting error." The likelihood function, or the probability of observing y given λ from a Poisson counting process, is given by:

$$p^{(\text{Poisson})}(y|\lambda) = \frac{\lambda^y e^{-\lambda}}{y!}. \tag{7}$$

The penalty function is given by the negative natural log of the likelihood function, with terms that do not depend on λ removed:

$$\ell^{(\text{Poisson})}(\lambda) = -\ln p(y|\lambda)^{(\text{Poisson})} = \lambda - y\ln\lambda + \text{ constants}. \tag{8}$$

We remove the terms that do not depend on λ because they will not affect our solution (which, varies λ by varying $\vec{f}$ to minimize the penalty function).

We will eventually also need the first and second derivatives with respect to λ:

$$\begin{aligned} \frac{d\ell^{(\text{Poisson})}}{d\lambda} &= 1 - y/\lambda, && (9) \\ \frac{d^2\ell^{(\text{Poisson})}}{d\lambda^2} &= y/\lambda^2. && (10) \end{aligned}$$

3.1.2.2 Relative Error Penalty Function. The relative error process arises from incomplete knowledge of the instrument calibration, either due to incomplete preflight testing, or changes in the instrument that occur during launch or on orbit, or due to assumptions made about the angular response of a wide-angle or omni-directional sensor relative to the local angular flux distribution.

Relative errors are assumed to have a Gaussian shape in the log of the counts:

$$p(y|\lambda)^{(\text{Relative})} = \frac{\exp[-((\ln y - \ln\lambda)/\delta y)^2/2]}{\sqrt{2\pi}y\delta y}. \tag{11}$$

The relative error penalty function is:

$$\ell^{(\text{Relative})}(\lambda) = -\ln p(y|\lambda)^{(\text{Relative})} = ((\ln y - \ln\lambda)/\delta y)^2/2 + \text{ constants}. \tag{12}$$

The derivatives with λ are:

$$\frac{d\ell^{(\text{Relative})}}{d\lambda} = (\ln\lambda - \ln y)/(\delta y)^2/\lambda, \tag{13}$$

$$\frac{d^2\ell^{(\text{Relative})}}{d\lambda^2} = (1 + \ln y - \ln\lambda)/(\lambda\delta y)^2. \tag{14}$$

3.1.2.3 Selecting a Penalty Function for Each y. In order to keep things simple, we select either the Poisson or relative error penalty function for each y, whenever the relative Poisson counting error $(1/\sqrt{y})$ is larger than the Gaussian relative error (δy).

$$\ell_i = \begin{cases} \ell^{(\text{Poisson})} & y < (\delta y)^{-2} \\ \ell^{(\text{Relative})} & \text{otherwise} \end{cases} \tag{15}$$

Thus, the final function to be minimized is:

$$\ell(\vec{\lambda}) = \sum_i \ell_i(\lambda_i). \tag{16}$$

This equation assumes that the error processes in the N_y channels are independent; i.e., the deviations $y_i - \lambda_i$ are not correlated with each other.

The derivatives are given by:

$$\frac{\partial\ell}{\partial\lambda_i} = \frac{\partial\ell_i}{\partial\lambda_i}, \tag{17}$$

$$\frac{\partial^2\ell}{\partial\lambda_i\lambda_k} = \delta_{ik}\frac{\partial\ell_i}{\partial\lambda_i}. \tag{18}$$

3.1.3 Analytical Spectral Functions

For the functions `ana_spec_inv` and `ana_spec_inv_multi`, we use analytical (parametric) spectral functions. We do not have any prior information with which to constrain the parameters of these analytical functions, so we will postpone discussion of prior model penalty functions until section 3.1.4.

3.1.3.1 Power Law Spectrum. The power law (PL) spectrum is given by:

$$f^{(\text{PL})}(E) = \exp(q_1 - q_2\ln E), \tag{19}$$

where we have chosen this parameterization so that there is no positivity constraint on q_1 and to reduce numerical effects of large free parameters, e.g., if we'd defined q_1 as a prefactor.

Its derivatives are:

$$\frac{\partial f^{(\text{PL})}}{\partial q_1} = f(E), \tag{20}$$

$$\frac{\partial f^{(\text{PL})}}{\partial q_2} = -\ln E f(E), \tag{21}$$

$$\frac{\partial^2 f^{(\text{PL})}}{\partial q_1^2} = f(E), \tag{22}$$

$$\frac{\partial^2 f^{(\mathrm{PL})}}{\partial q_1 \partial q_2} = -\ln E f(E) = \frac{\partial^2 f^{(\mathrm{PL})}}{\partial q_2 \partial q_1}, \tag{23}$$

$$\frac{\partial^2 f^{(\mathrm{PL})}}{\partial q_2^2} = (\ln E)^2 f(E). \tag{24}$$

3.1.3.2 Exponential Spectrum. The exponential (EXP) spectrum is given by:

$$f^{(\mathrm{EXP})}(E) = \exp(q_1 + q_2 E), \tag{25}$$

where, again, we have chosen this parameterization so that there is no positivity constraint on q_1 and to reduce numerical effects of large free parameters. Its derivatives are:

$$\frac{\partial f^{(\mathrm{EXP})}}{\partial q_1} = f(E), \tag{26}$$

$$\frac{\partial f^{(\mathrm{EXP})}}{\partial q_2} = E f(E), \tag{27}$$

$$\frac{\partial^2 f^{(\mathrm{EXP})}}{\partial q_1^2} = f(E), \tag{28}$$

$$\frac{\partial^2 f^{(\mathrm{EXP})}}{\partial q_1 \partial q_2} = E f(E) = \frac{\partial^2 f^{(\mathrm{EXP})}}{\partial q_2 \partial q_1}, \tag{29}$$

$$\frac{\partial^2 f^{(\mathrm{EXP})}}{\partial q_2^2} = E^2 f(E). \tag{30}$$

3.1.3.3 Relativistic Maxwellian Spectrum. The relativistic Maxwellian (RM) spectrum is given by:

$$f^{(\mathrm{RM})}(E) = E(1 + E/E_0/2)\exp(q_1 + q_2 E), \tag{31}$$

where, again, we have chosen this parameterization so that there is no positivity constraint on q_1 and to reduce numerical effects of large free parameters. The constant E_0 is the rest energy of the particle species (511 keV for electrons, 938 MeV for protons). Its derivatives are given by the same expressions as for the exponential spectrum (26)-(30).

3.1.3.4 Double Relativistic Maxwellian Spectrum. The double relativistic Maxwellian (RM2) spectrum is given by:

$$f^{(\mathrm{RM2})}(E) = E(1 + E/E_0/2)\left[\exp(q_1 + q_2 E) + \exp(q_3 + q_4 E)\right], \tag{32}$$

where, again, we have chosen this parameterization so that there is no positivity constraint on q_1 or q_3 and to reduce numerical effects of large free parameters. The constant E_0 is the rest energy of the particle species (511 keV for electrons, 938 MeV for protons). Its derivatives are given by the straightforward extension of (26)-(30).

3.1.3.5 Power Law Spectrum with Exponential Tail. The power law spectrum with exponential tail (PLE) is given by:

$$f^{(\mathrm{PLE})}(E) = \begin{cases} \exp(q_1 - q_2 \ln E) & E \le E_{\mathrm{break}}, \\ \exp(q_1 - q_2 \ln E_{\mathrm{break}} - (E - E_{\mathrm{break}})/E_0) & E > E_{\mathrm{break}}. \end{cases} \tag{33}$$

where we have chosen this parameterization so that there is no positivity constraint on q_1 and to reduce numerical effects of large free parameters, e.g., if we'd defined q_1 as a prefactor. Nominal values for inner zone protons are $E_0 = 345$ MeV, $E_{\mathrm{break}} = 100$ MeV.

Its derivatives are:

$$\theta = \begin{cases} -\ln E & E \le E_{\mathrm{break}}, \\ -\ln E_{\mathrm{break}} & E > E_{\mathrm{break}}. \end{cases} \tag{34}$$

$$\frac{\partial f^{(\mathrm{PLE})}}{\partial q_1} = f(E), \tag{35}$$

$$\frac{\partial f^{(\mathrm{PLE})}}{\partial q_2} = \theta f(E), \tag{36}$$

$$\frac{\partial^2 f^{(\mathrm{PLE})}}{\partial q_1^2} = f(E), \tag{37}$$

$$\frac{\partial^2 f^{(\mathrm{PLE})}}{\partial q_1 \partial q_2} = \theta f(E) = \frac{\partial^2 f^{(\mathrm{PLE})}}{\partial q_2 \partial q_1}, \tag{38}$$

$$\frac{\partial^2 f^{(\mathrm{PLE})}}{\partial q_2^2} = \theta^2 f(E). \tag{39}$$

3.1.3.6 Fit Errors and Combination of Multiple Spectral Fits. For a given spectral function $f^{(k)}(E)$, the fit is the minimization of $\ell^{(k)}(\vec{\lambda})$ with respect to $\vec{q}^{(k)}$, which obtains the maximum likelihood estimate $\hat{q}^{(k)}$. Each fit is performed with multivariate minimization routines provided by the Gnu Scientific Library. Some of these minimizations require gradients of $\ell^{(k)}$ with respect to $\vec{q}^{(k)}$, which are given by (without the (k) superscripts):

$$\begin{aligned} \frac{\partial \ell}{\partial q_m} &= \sum_i \frac{\partial \ell_i}{\partial \lambda_i} \sum_j \frac{\partial \lambda_i}{\partial f_j} \frac{\partial f_j}{\partial q_m}, \\ &= \sum_i \frac{\partial \ell_i}{\partial \lambda_i} \sum_j H_{ij} \frac{\partial f_j}{\partial q_m}. \end{aligned} \tag{40}$$

The Hessian, used for computing the error bar, is given by:

$$\frac{\partial^2 \ell}{\partial q_m \partial q_{m'}} = \sum_i \frac{\partial^2 \ell_i}{\partial \lambda_i^2} \sum_j H_{ij} \frac{\partial f_j}{\partial q_m} \sum_{j'} H_{ij'} \frac{\partial f_{j'}}{\partial q_{m'}} + \sum_i \frac{\partial \ell_i}{\partial \lambda_i} \sum_j H_{ij} \frac{\partial^2 f_j}{\partial q_m \partial q_{m'}}. \tag{41}$$

We treat the error on the resulting flux as having a log-normal distribution with standard deviation (i.e., standard error) given by the energy-dependent expression:

$$\begin{aligned} \sigma_{\ln f^{(k)}(E)} &= \sqrt{\sum_m \sum_{m'} \frac{\partial \ln f^{(k)}}{\partial q_m^{(k)}} \operatorname{cov}(q_m^{(k)}, q_{m'}^{(k)}) \frac{\partial \ln f^{(k)}}{\partial q_{m'}^{(k)}}}, \\ &= \sqrt{\sum_m \sum_{m'} \frac{1}{f^{(k)}} \frac{\partial f^{(k)}}{\partial q_m^{(k)}} \operatorname{cov}(q_m^{(k)}, q_{m'}^{(k)}) \frac{1}{f^{(k)}} \frac{\partial f^{(k)}}{\partial q_{m'}^{(k)}}}, \end{aligned} \tag{42}$$

$$\operatorname{cov}(q_m^{(k)}, q_{m'}^{(k)}) = \begin{pmatrix} & \vdots & \\ \cdots & \left.\frac{\partial^2 \ell^{(k)}}{\partial q_m^{(k)} \partial q_{m'}^{(k)}}\right|_{\hat{q}^{(k)}} & \cdots \\ & \vdots & \end{pmatrix}^{-1}, \tag{43}$$

$$p(\ln f^{(k)}(E)) = \frac{1}{\sqrt{2\pi}\sigma_{\ln f^{(k)}}} \exp\left[-\frac{1}{2}\left(\frac{\ln f(E) - \ln f^{(k)}(E)}{\sigma_{\ln f^{(k)}(E)}}\right)^2\right] \tag{44}$$

The approximation of an error covariance by the inverse of the Hessian of the penalty function is equivalent to approximating the curvature of the penalty function near the maximum likelihood value with the curvature of a Gaussian, i.e., a second-order Taylor series expansion of the penalty function.

We choose to combine the analytical fits according to $\ell^{(k)}$:

$$w_k = \frac{\exp(-\ell^{(k)} - N_q^{(k)})}{\sum_k \exp(-\ell^{(k)} - N_q^{(k)})}, \tag{45}$$

where $N_q^{(k)}$ is the length of $\vec{q}^{(k)}$ (always 2 in the spectral functions used so far). When evaluating the exp functions above, it is best first to subtract of the least $\ell^{(k)}$ from all the $\ell^{(k)}$ to avoid floating point overflow. This combination process is equivalent to assuming (reasonably) there there are multiple possibilities for each f_j, with probability given by $\exp(-\ell^{(k)})$.

We then have a probability distribution that combines the log-normal probability distributions for the individual fits:

$$p(\ln f^{\mathrm{combined})}(E)) = \sum_k w_k p(f^{(k)}(E)), \tag{46}$$

$$
\begin{aligned}
\ln \hat{f}(E) &= \left\langle \ln f^{(\text{combined})}(E) \right\rangle = \sum_k w_k \ln f^{(k)}(E), && (47) \\
\delta \ln \hat{f}(E) &= \sqrt{\operatorname{var} \ln f^{(\text{combined})}(E)} \\
&= \sqrt{\sum_k w_k \left(\sigma^2_{\ln f^{(k)}(E)} + \ln^2 f^{(k)}(E) \right) - \left\langle \ln f^{(\text{combined})}(E) \right\rangle^2}. && (48)
\end{aligned}
$$

When the function returns, `flux[j]` $= \hat{f}(E_j)$, and `dlogflux[j]` $= \delta \ln \hat{f}(E_j)$.

3.1.4 Principal Component Spectral Function

For the functions `pc_spec_inv` and `pc_spec_inv_multi`, we use data-derived (discrete) spectral basis functions to parameterize the spectrum and associated prior information to constrain those parameters.

We begin by assuming a data set of log fluxes:

$$
x_j(t) = \ln f(E_j, t), \tag{49}
$$

with a log-normal distribution:

$$
p(\vec{x}) = \frac{1}{\sqrt{(2\pi)^{N_E} |\underline{\underline{\Sigma}}|}} \exp\left[-\frac{1}{2} \left(\vec{x} - \bar{\vec{x}} \right)^T \underline{\underline{\Sigma}}^{-1} \left(\vec{x} - \bar{\vec{x}} \right) \right] \tag{50}
$$

We compute the mean log flux as:

$$
\bar{\vec{x}} = \langle \vec{x}(t) \rangle, \tag{51}
$$

where $\langle \cdot \rangle$ implies an average over time.

We compute the covariance of log flux as:

$$
\underline{\underline{\Sigma}} = \left\langle \left(\vec{x}(t) - \bar{\vec{x}} \right) \left(\vec{x}(t) - \bar{\vec{x}} \right)^T \right\rangle. \tag{52}
$$

We eigenfactor $\underline{\underline{\Sigma}}$ as:

$$
\begin{aligned}
\underline{\underline{\Sigma}} &= \underline{\underline{V}}\,\underline{\underline{D}}\,\underline{\underline{V}}^T, && (53) \\
\underline{\underline{V}}^T \underline{\underline{V}} &= \underline{\underline{I}}, && (54) \\
D_{ij} &= \delta_{ij} d_i. && (55)
\end{aligned}
$$

The columns of $\underline{\underline{V}}$ are orthonormal basis vectors, and the values $\vec{d}$ on the diagonal of $\underline{\underline{D}}$ are the amount of variance of the log flux spectrum explained by each vector.

Therefore, for any time, we can write:

$$
\begin{aligned}
\vec{x}(t) &= \bar{\vec{x}} + \underline{\underline{V}} \vec{q}(t), && (56) \\
f(E_j, t) &= \exp\left[\bar{x}_j + \sum_m V_{jm} q_m(t) \right]. && (57)
\end{aligned}
$$

The gradient and Hessian of f and $\ln f$ with respect to $\vec{q}$ are:

$$
\begin{aligned}
\frac{\partial f(E_j)}{\partial q_m} &= f(E_j) V_{jm}, && (58) \\
\frac{\partial^2 f(E_j)}{\partial q_m \partial q_{m'}} &= f(E_j) V_{jm} V_{jm'}, && (59) \\
\frac{\partial \ln f(E_j)}{\partial q_m} &= V_{jm}, && (60) \\
\frac{\partial^2 \ln f(E_j)}{\partial q_m \partial q_{m'}} &= 0. && (61)
\end{aligned}
$$

We can then determine the distribution for $\vec{q}$ from (50):

$$\begin{aligned}
\vec{q} &= \underline{\underline{V}}^T(\vec{x}-\bar{\vec{x}}), && (62)\\
\underline{\underline{\Sigma}}^{-1} &= \underline{\underline{V}}\,\underline{\underline{D}}^{-1}\underline{\underline{V}}^T, && (63)\\
|\Sigma| &= |\underline{\underline{D}}|, && (64)\\
p(\vec{q}) &= p(\vec{x})|\underline{\underline{V}}| = p(\vec{x}) = \frac{1}{\sqrt{(2\pi)^{N_q}|\underline{\underline{D}}|}}\exp\left[-\frac{1}{2}\vec{q}^T\underline{\underline{D}}^{-1}\vec{q}\right]. && (65)
\end{aligned}$$

The $|\underline{\underline{V}}| = 1$ factor is included for completeness: it conserves probability under the change of variables from $\vec{x}$ to $\vec{q}$. The $\vec{q}$'s are therefore uncorrelated, nominally Gaussian variables with zero mean and variance given by the corresponding entries in $\vec{d}$.

We denote the negative log likelihood penalty function associated with the prior information as $\hat{\ell}$:

$$-\ln p(\vec{q}) = \hat{\ell}(\vec{q}) = \frac{1}{2}\vec{q}^T\underline{\underline{D}}^{-1}\vec{q} + \quad \text{constants}. \tag{66}$$

Its gradient and Hessian are:

$$\begin{aligned}
\frac{\partial\hat{\ell}}{\partial q_m} &= \frac{q_m}{d_m}, && (67)\\
\frac{\partial^2\hat{\ell}}{\partial q_m \partial q_{m'}} &= \frac{\delta_{mm'}}{d_m}. && (68)
\end{aligned}$$

The prior information penalty function $\hat{\ell}$ is added to the measurement penalty function (16) to give:

$$\ell(\vec{q}) = \hat{\ell}(\vec{q}) + \sum_i \ell_i(\lambda_i(\vec{q})). \tag{69}$$

(In the analytical spectral fitting, there was no prior information to constrain the solution $\vec{q}$, so $\hat{\ell}$ was zero). Because of the presence of prior information it is now technically possible to have more free parameters than constraints (i.e., $N_q > N_y$ is allowed).

As before, a nonlinear optimizer is used to minimizer $\vec{\ell}$ with respect to $\vec{q}$. The solution is denoted $\hat{q}$. The error covariances $\operatorname{cov}\delta\vec{q}$ and $\operatorname{cov}\delta\vec{x}$ (log flux) are given by:

$$\begin{aligned}
\operatorname{cov}(\delta q_m, \delta q_{m'}) &= \begin{pmatrix} & \vdots & \\ \cdots & \left.\frac{\partial^2\ell}{\partial q_m \partial q_{m'}}\right|_{\hat{q}} & \cdots \\ & \vdots & \end{pmatrix}^{-1}, && (70)\\
\operatorname{cov}(\delta x_j, \delta x_{j'}) &= \sum_{m,m'} \left.\frac{\partial \ln f(E_j)}{\partial q_m}\right|_{\hat{q}} \operatorname{cov}(\delta q_m, \delta q_{m'}) \left.\frac{\partial \ln f(E_{j'})}{\partial q_{m'}}\right|_{\hat{q}} = \sum_{m,m'} V_{jm}\operatorname{cov}(\delta q_m, \delta q_{m'})V_{j'm'}. && (71)
\end{aligned}$$

Finally the standard error of the natural log flux is:

$$\delta \ln f(E_j) = \delta x_j = \sqrt{\operatorname{cov}(\delta x_j, \delta x_j)}. \tag{72}$$

3.1.5 Why does this work?

Maximum likelihood methods are developed in several places; here we provide only a brief summary of how one sets up a maximum likelihood problem. We wish to solve for some unknowns $\vec{q}$ given some information $\vec{y}$. The maximum likelihood solution $\hat{q}$ maximizes the probability of $\vec{q}$ conditioned on $\vec{y}$. In the common notation of probability theory, we maximize $p(\vec{q}|\vec{y})$:

$$p(\vec{q}|\vec{y}) = \frac{p(\vec{q},\vec{y})}{p(\vec{y})} = p(\vec{y}|\vec{q})\frac{p(\vec{q})}{p(\vec{y})}. \tag{73}$$

In practice, we minimize the negative log likelihood:

$$-\ln p(\vec{q}|\vec{y}) = -\ln p(\vec{y}|\vec{q}) - \ln p(\vec{q}) + \ln p(\vec{y}). \tag{74}$$

The measurement penalty functions $\ell_i(\lambda_i(\vec{q}))$ provide $-\ln p(\vec{y}|\vec{q})$:

$$-\ln p(y_i|\lambda_i(\vec{q})) = \ell_i(\lambda_i(\vec{q})) + \quad \text{constants.} \tag{75}$$

The prior information penalty function $\hat{\ell}(\vec{q})$ provides $-\ln p(\vec{q})$:

$$-\ln p(\vec{q}) = \hat{\ell}(\vec{q}) + \quad \text{constants.} \tag{76}$$

We note that $p(\vec{y})$ does not depend on $\vec{q}$, and can be dropped from the negative log likelihood expression, like other constants. Therefore,

$$-\ln p(\vec{q}|\vec{y}) = \ell(\vec{q}) = \hat{\ell}(\vec{q}) + \sum_i \ell_i(\lambda_i(\vec{q})) + \quad \text{constants.} \tag{77}$$

In the vicinity of the solution $\hat{q}$, we can Taylor expand $\ell(\vec{q})$:

$$\ell(\vec{q}) = \ell(\hat{q}) + \sum_m \left.\frac{\partial \ell}{\partial q_m}\right|_{\hat{q}_m} (q_m - \hat{q}_m) + \sum_{m,m'} (q_{m'} - \hat{q}_{m'}) \left.\frac{\partial^2 \ell}{\partial q_m \partial q_{m'}}\right|_{q_m, \hat{q}_{m'}} (q_m - \hat{q}_m) + \mathcal{O}(||\vec{q} - \hat{q}||^3). \tag{78}$$

The first term is a constant, and the second term is zero (because we evaluate the derivative at the local minimum in ℓ). If we truncate the series in the quadratic term, we have:

$$\ell(\vec{q}) = \sum_{m,m'} (q_{m'} - \hat{q}_{m'}) \left.\frac{\partial^2 \ell}{\partial q_m \partial q_{m'}}\right|_{q_m, \hat{q}_{m'}} (q_m - \hat{q}_m) + \quad \text{constants.} \tag{79}$$

This is just the expression for the negativelog likelihood of a Gaussian distribution in $\delta\vec{q} = \vec{q} - \hat{q}$ with covariance $(\operatorname{cov}\delta\vec{q})$ given by:

$$\operatorname{cov}(\delta\vec{q}) = \begin{bmatrix} & \vdots & \\ \cdots & \left.\frac{\partial^2 \ell}{\partial q_m \partial q_{m'}}\right|_{\hat{q}} & \cdots \\ & \vdots & \end{bmatrix}^{-1} \tag{80}$$

3.1.6 Example

The Matlab function `invlib_pc_spec_inv.m` generates two figures. The first (Figure 1) is a principal component model generated from CRRES HEEF and MEA fluxes at $L \sim 4-5$. The second (Figure 2) compares analytical spectral inversion results to those for principal component spectral inversion. Note that in the "10-PC" case, there are more free parameters than unknowns and the solution is still stable. The ICO channels provide little constraint below about 0.8 MeV.

3.2 C language prototypes

The C language prototypes for the spectral inversion functions can be found in "invlib.h".

3.2.1 ana_spec_inv

The first library function is `ana_spec_inv` with the following C prototype:

```
int ana_spec_inv(const double *y, const double *dy,
                 const double *Egrid, const double *H, const double *b,
                 const long int *int_params, const double *real_params,
                 char *outFile, double *Eout,
                 double *flux, double *dlogflux,
 double *lambda, double *support_data);
```

This function takes a set of counts `y` (per accumulation interval), estimated relative error on those counts `dy`, and measurement matrix `H` defined on energy grid `Egrid`, with estimated background counts `b`, and returns `flux` on output energy grid `Eout`. The user can select from a variety of analytical functions that will be individually fit to the data, and then a "weighted expected value" method will be used to combine multiple analytical fits. The user can also control which nonlinear minimization solver is used, and whether and where progress messages are printed. The function returns a code that indicates success or one of several failure modes. There is a multi-case version `ana_spec_inv_multi` described at the end of this section.

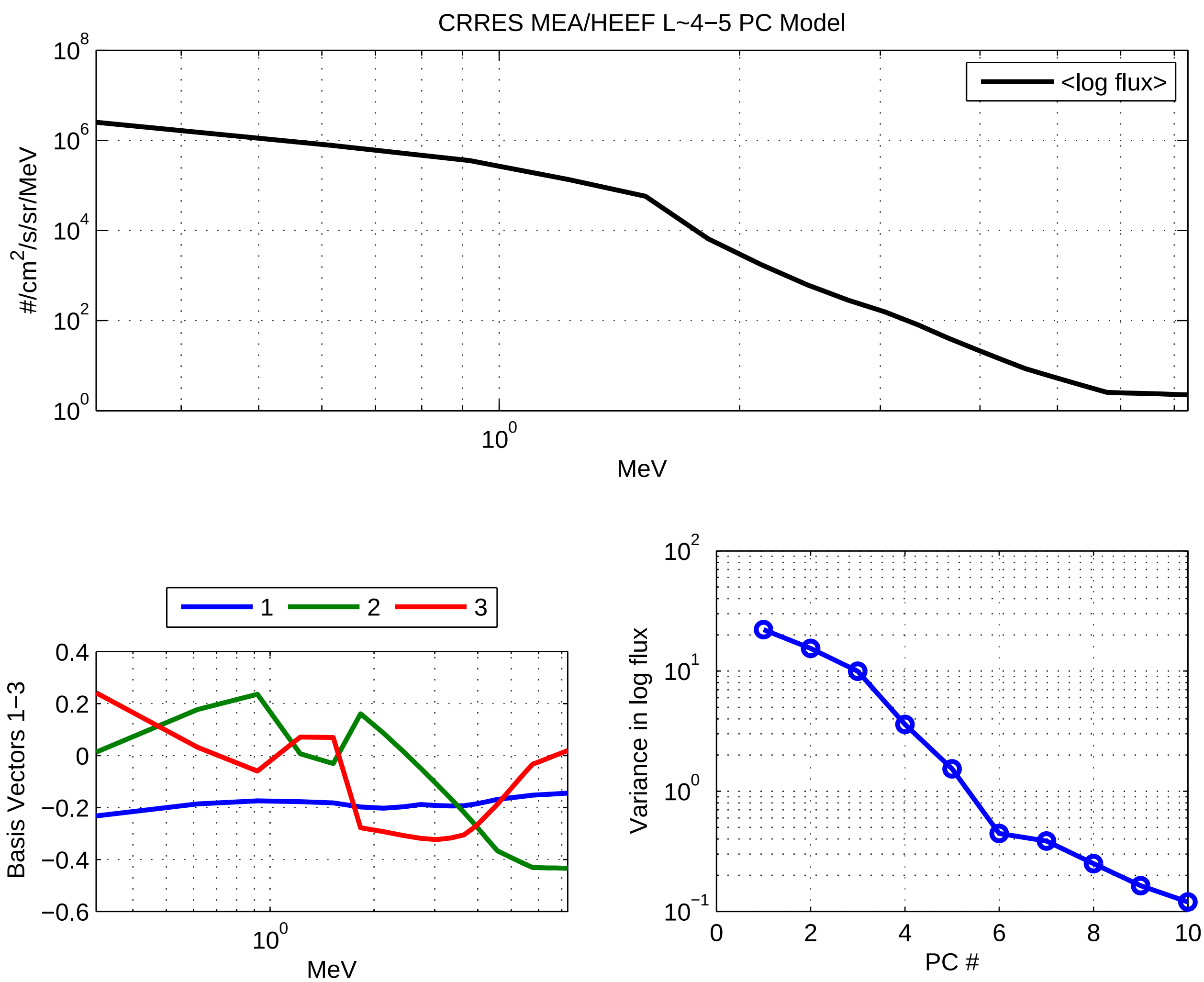

Figure 1: Principal component derived by combining CRRES MEA and HEEF differential channels.

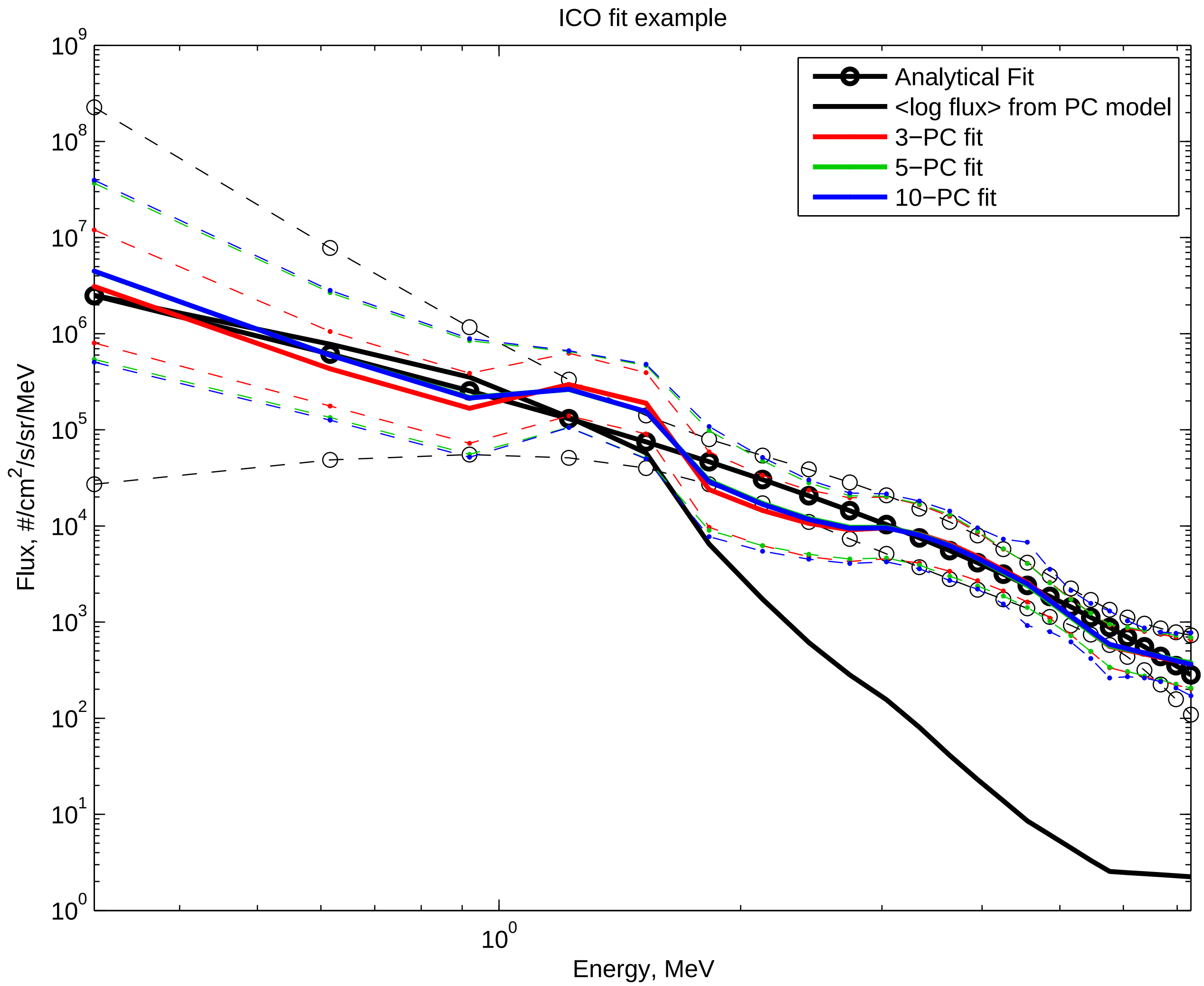

Figure 2: An example of fitting power law (PL) and exponential (EXP) spectra or the CRRES principal components model to the counts in 5 ICO electron channels to a simulated exponential with Poisson noise. Dashed lines indicate 1 standard error above and below the fits. (The 10-PC fit falls right on top ofthe 5-PC fit.)

3.2.2 ana_spec_inv_multi

The library also provides a multi-case function, `ana_spec_inv_multi`, with the following C prototype:

```
int ana_spec_inv_multi(const long int Ntimes,
       const double *y, const double *dy,
       const double *Egrid, const double *H0,
       const double *dt, const double *b,
       const long int *int_params, const double *real_params,
       char *outFile, double *Eout,
       double *flux, double *dlogflux,
       double *lambda, double *support_data,
       int *result_codes);
```

This function calls `ana_spec_inv` `Ntimes`, in each case advancing the `y` and `b` pointers by `NY` (i.e., moving one row forward, if `y` and `b` are interpreted as `Ntimes`×`NY` row-major matrices), and, similarly, advancing `flux` and `dlogflux` by `NE`. `lambda`, if requested, is advanced by `NY`. `support_data`, if requested, is advanced by $(2 + \texttt{ASI\_MAX\_NQ} + \texttt{NY}) \times (\texttt{ASI\_MAX\_POW2} + 1)$, where `ASI_MAX_POW2` is presently 4 since 2^4 is the largest function bitmap value, and is defined in "specinv.h" for C users.

`result_codes` provides the functional return values. The primary difference between `ana_spec_inv` and `ana_spec_inv_multi` is that the latter requires the aggregation time (`dt`) be passed in separately from the instrument response function, where for `ana_spec_inv` the response function must include the aggregation time. For each call to `ana_spec_inv`, a temporary `H` will be set to `H0`×`dt[t]`, to reflect a possibly changing integration time. The other inputs are passed directly to `ana_spec_inv`, without alteration.

The return code of `ana_spec_inv_multi` is typically the first non-success result code returned by the multiple calls `ana_spec_inv`. Of course, if all calls to `ana_spec_inv` succeed, the return code from `ana_spec_inv_multi` will indicate success. The only additional exception is if `Ntimes`< 1, in which case the empty input array error code will be returned (-102).

3.2.3 pc_spec_inv

```
int pc_spec_inv(const double *y, const double *dy, const double *Egrid,
const double *H, const double *b,
const double *mean_log_flux, const double *basis_vectors,
const double *basis_variance,
const long int *int_params, const double *real_params,
char *outFile, double *flux, double *dlogflux,
double *lambda, double *support_data);
```

This function behaves like `ana_spec_inv` except that instead of specifying a list of analytical spectral functions to try, prior model information is passed in (`mean_log_flux`, `basis_vectors`, and `basis_variance`). This function does not have a separate energy grid `Eout` for output flux: `Eout` ≡ `Egrid`. As we will see below, the input `int_params` and the output `support_data` have slightly different meanings for this function.

3.2.4 pc_spec_inv_multi

```
int pc_spec_inv_multi(const long int Ntimes,
      const double *y, const double *dy,
      const double *Egrid, const double *H0,
      const double *dt,const double *b,
      const double *mean_log_flux, const double *basis_vectors,
      const double *basis_variance,
      const long int *int_params, const double *real_params,
      char *outFile,
      double *flux, double *dlogflux,
      double *lambda, double *support_data,
      int *result_codes);
```

Like `pc_spec_inv`, this function behaves like its counterpart `ana_spec_inv_multi` except that it calls `pc_spec_inv_mult` `Ntimes`. The same prior model information is used for each inversion, and the structure of `support_data` is slightly different.

3.3 Function Arguments

- `y` (input) `y[NY*t+i]` = counts in channel `i` in case `t`.
- `dy` (input) `dy[i]` = relative error for channel `i` (RMS error of ln `y[i]` from calibration) (dimensionless).
- `Egrid` (input) `Egrid[j]` = nominal energy of `j`$^{\text{th}}$ grid point (e.g., in keV).
- `H` (input) `H[NY*j+i]` = response of channel `i` to flux at energy `j` (e.g., in keV cm^2 sr s).
- `dt` (input) `dt[t]` = integration time for case `t`.
- `b` (input) `b[NY*t+i]` = expected background counts in channel `i` for case `t`.
  Best fit when `y[i]` $\approx$ `b[NY*t+i]` + $\sum_j$ `H[NY*j+i]*dt[t]*flux[NE*t+j]`.
- `mean_log_flux` (input) `mean_log_flux[j]` = mean natural log flux at energy `j` (e.g., flux in keV cm^2 sr s).
- `basis_vectors` (input) `basis_vectors[`N_q`*j+m]` = V_{jm} (note 0-based index in C vs 1-based index in linear algebra).
- `basis_variance` (input) `basis_variance[m]` = d_m variance in natural log flux explained by m^{th} basis vector.
- `int_params` (input) integer parameters, length = 10. (TBR)

  `int_params[0]` Number of energy channels, `NY`.

  `int_params[1]` Number of energy grid points, `NE`.

  `int_params[2]` *depends on function:*

  `ana_spec_inv` or `ana_spec_inv_multi`: Number of energy output points `NEout`.

  `pc_spec_inv` or `pc_spec_inv_multi`: Number of basis vectors, N_q.

  `int_params[3]` *depends on function:*

  `ana_spec_inv` or `ana_spec_inv_multi`: Spectral Function Bit Mask (combined via bitwise OR):

  [1] Power law (PL),

  [2] Exponential (EXP),

  [4] Relativistic Maxwellian (RM),

  [8] Power law with exponential tail (PLE).

  [16] Double Relativistic Maxwellian (RM2),

  `pc_spec_inv` or `pc_spec_inv_multi`: Number of basis vectors to actually use ($N_{q'} \leq N_q$).

  `int_params[4]` Choice of minimizer (choose one):

  [0] Broyden-Fletcher-Goldfarb-Shanno, BFGS (recommended),

  [1] Conjugate Fletcher-Reeves, Conjugate FR,

  [2] Conjugate Polak-Ribiere, Conjugate PR,

  [3] Nelder-Mead Simplex.

  `int_params[5]` Maximum number of iterations by minimizer (recommend 10,000).

  `int_params[6]` Verbose setting (choose one):

  [0] no text output,

  [1] text output to standard output stream,

  [2] text output to standard error stream,

  [3] text output to outFile (assumes outFile is actually a `FILE *`).

  [4] text output to outFile, overwrite existing file

  [5] text output to outFile, append to existing file

  `int_params[7]` Energy integral weighting setting

  [0] H already includes ΔE

[1] H needs to be multiplied by ΔE. Compute ΔE using trapezoidal rule.

[2] H needs to be multiplied by ΔE. Compute ΔE using plateau rule.

`int_params[8]` reserved.

`int_params[9]` reserved.

- `real_params` (input) real parameters, length 10, not used by `pc_spec_inv` or `pc_spec_inv_multi` (TBR)

  `real_params[0]` = rest energy of particle species.

  `real_params[1]` = E_{break} used by PLE.

  `real_params[2]` = E_0 used by PLE.

  `real_params[3]` reserved.

  `real_params[4]` reserved.

  `real_params[5]` reserved.

  `real_params[6]` reserved.

  `real_params[7]` reserved.

  `real_params[8]` reserved.

  `real_params[9]` reserved.

- `outFile` (input) provides filename or `FILE *` for verbose output (see verbose setting above). Null terminated string.
- `Eout` (output) `Eout[j]` = nominal energy of `j`$^{\mathrm{th}}$ grid point for output flux (e.g., in keV).
- `flux` (output) `flux[NE*t+j]` = inverted flux at `j`$^{\mathrm{th}}$ output energy grid point (e.g., in #/keV/cm^2/sr/s), case `t`.
- `dlogflux` (output) `dlogflux[NE*t+j]` = standard error of natural log of `flux[NE*t+j]` (dimensionless).
- `lambda` (output) `lambda[NY*t+i]` = estimated counts from combined fit for `y[NY*t+i]` (ignored if `NULL`).
- `support_data` (output) support data (ignored if `NULL`)

  if function is `ana_spec_inv` or `ana_spec_inv_multi`:

  stride (`(...)` below) for each t is `(2+ASI_MAX_NQ+NY)*(ASI_MAX_POW2+1)`.

  `support_data[(...)*t+(2+ASI_MAX_NQ+NY)*k]` ℓ for the spectral function with bit mask 2^k (dimensionless).

  `support_data[(...)*t+(2+ASI_MAX_NQ+NY)*k+1]` k^{th} weight (dimensionless).

  `support_data[(...)*t+(2+ASI_MAX_NQ+NY)*k+2+m]` m^{th} fit parameter for k^{th} function (dimensionless).

  `support_data[(...)*t+(2+ASI_MAX_NQ+NY)*k+2+ASI_MAX_NQ+i]` expected counts for `y[i]`. (usually dimensionless)

  if function is `pc_spec_inv` or `pc_spec_inv_multi`:

  stride (`(...)` below) for each t is $1 + N_{q'}$

  `support_data[(...)*t` ℓ

  `support_data[(...)*t+1+m]` m^{th} fit parameter (dimensionless).

Notes:

1. Except where noted, array and matrix indices above are zero-based, following C convention rather than one-based linear algebra convention.
2. All matrices are passed into/out of C in "row major" format in the "linear algebra" sense. That is, each `double *` points to a sequence of contiguous rows of the corresponding linear algebra matrix.
3. For `ana_spec_inv` and `pc_spec_inv`, assume $t = 0$ to compute array indices.

4. `ana_spec_inv` operates on *counts* not *count rate*—
5. Missing counts can be replaced with NaN, and the channel will be ignored.
6. If `dy` is unknown, use ln(2)/2 for factor of 2 95% confidence bounds, or 0 for Poisson error only.
7. Energy units must be consistent throughout: e.g., fluxes in `/keV`, energy grid points in `keV`, and rest energy in `keV`.
8. `H` is stored in row-major format, i.e., a sequence of contiguous rows of $\underline{\underline{H}}$.
9. $H_{ij} \approx \delta t G_i(E_j)\Delta E_j$, where δt is the integration time, $G_i(E_j)$ is the geometric factor for channel i at energy j, and ΔE_j is the energy bandwidth (weighting in the numerical integral) of the j^{th} grid point. Alternatives are possible: for example, the first and last columns of $\underline{\underline{H}}$ can be divided by 2 to approximate trapezoidal integration if the energy grid is uniform.
10. It is best not to fit both an exponential and a relativistic Maxwellian in combination with a power law (or any other spectral function we add later) spectrum. Because exponential and relativistic Maxwellians are so similar, they'll combine to overrule the power law, inadvertently giving two votes to roughly the same spectrum.
11. `outFile` is ignored for verbose settings 0, 1, and 2.
12. `N_{q'}` is used by `pc_spec_inv` and `pc_spec_inv_multi` to fit fewer principal components than N_q, which may be desirable for speed or stability.
13. The 95% confidence interval on `flux[j]` is given by `flux[j]` $\times$ exp($\pm$1.9600 $\times$ `dlogflux[j]`).
14. For `ana_spec_inv` and `ana_spec_inv_multi`, entries in `support_data` for fit functions not used (i.e., are not set in the spectral function bit mask) are returned without modification.
15. `ASI_MAX_NQ` is 10 at the moment, providing for up to 10 free parameters in some future fit function. It may change, but that's unlikely. `specinv.h` defines `ASI_MAX_NQ`, `ASI_MAX_POW2`, and some handy macros for C users.

3.4 Return Codes

1 Success! No Error.

0 Unknown error (this only happens if there's a bug).

-101 NULL passed where pointer expected.

-102 One or fewer valid data points; e.g., (`NY`≤ 1), or `NE`≤ 1 or `NEout`< 1.

-103 one or more invalid (NaN or infinite) in input arrays, or some `y[i]`< 0, `dy[i]`< 0, `Egrid[j]`≤ 0, or `Egrid[j]`≤ 0.

-104 no counts in any channel.

-201 No functions selected in function bit map.

-202 Invalid function selected in function bit map.

-301 Relativistic Maxwellian (RM) requested with NULL `real_params` or `real_params[0]`≤ 0.

-302 Power law with exponential tail (PLE) requested with NULL or negative `real_params`.

-401 Invalid minimizer selected.

-402 Invalid iterations requested (zero or negative).

-501 Invalid value of verbose provided or NULL value provided for user-requested stream (outFile).

-502 User-requested verbose output file (outFile) could not be opened.

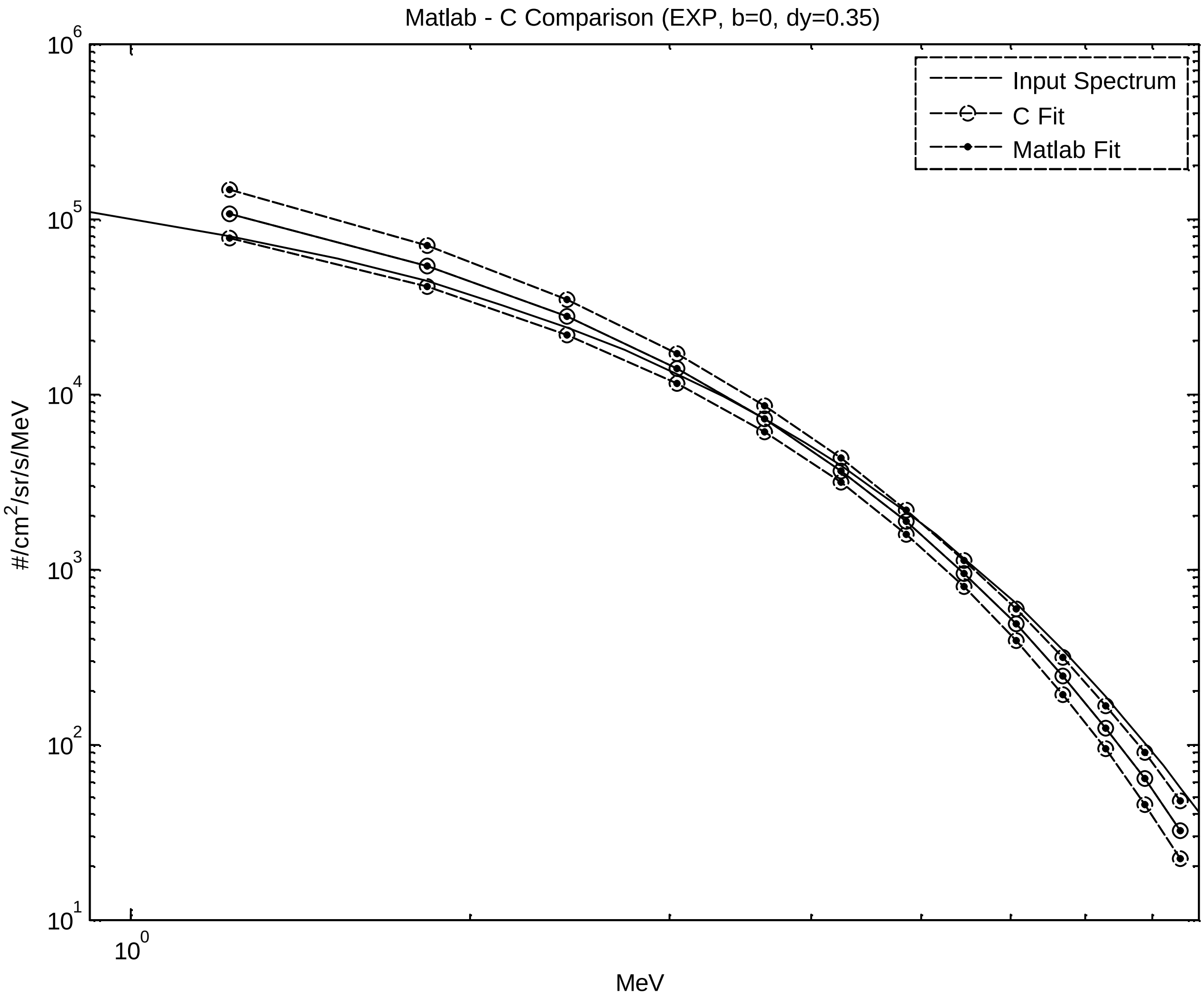

Figure 3: An example of fitting power law (PL) and exponential (EXP) spectra to the counts in 5 ICO electron channels to a simulated exponential with Poisson noise. Dotted lines indicate 1 standard error above and below the fits.

3.5 Validation

The routine `ana_spec_inv` and many of its dependencies have been validated against a Matlab counterpart "EnergySpectralInversionAnalytical.m". The C test code for `ana_spec_inv` is "specinv_test.c" and the Matlab test code is "specinv_test.m". The program"specinv_test.exe" reads the text file "specinv_test.in1" and produces the text file "specinv_test.out1", which is then compared to the same spectral inversion solved in Matlab. The results are shown in Figure 3.

Validation has consisted of validation of individual "private" routines found in the various '.c" files but not shared via "invlib.h", as well as the shared `ana_spec_inv routine`. The C and Matlab codes agree to within 5 or 6 decimal places for inputs of noisy power-law and exponential distributions fit to combinations of power-law, exponential and relativistic Maxwellian spectra.

4 Angular Inversion Functions

This section describes the angular inversion functions for single-channel omnidirectional and wide-angle measurements.

The following C prototypes can be found "invlib.h":

```
int omni2uni(const double *omniflux, const double *dlogomniflux,
```

```
            const long int *int_params,
            const double *real_params,
            char *outFile,
            double *uniflux, double *dloguniflux);
```

Returns estimated locally-mirroring unidirectional flux.

```
int wide2uni(const double *wideflux, const double *dlogwideflux,
            const double *PAgrid, const double *H,
            const long int *int_params,
            const double *real_params,
            char *outFile,
            const long int *ialpha0,
            double *uniflux, double *dloguniflux);
```

Returns estimated flux at `PAgrid[ialpha0]`, where `H` describes angular dependence on grid `PAgrid`.

4.1 Calculations Performed - TEM-1 Method

The omnidirectional or wide-angle flux is given by:

$$f_{\text{iso}} = \int_0^{2\pi} \int_0^{\pi} h(\alpha, \phi) f_{\text{uni}}(\alpha) \sin \alpha d\alpha d\phi. \tag{81}$$

We denote the wide-angle or omnidirectional flux f_{iso} under the assumption that `wideflux` or `omniflux` is given in the form of a isotropic unidirectional flux (e.g., $\#/\text{cm}^2/\text{s/sr/keV}$). For the wide-angle approximation, one has a bit more flexibility: it is only necessary to provide `H` and `wideflux` in appropriate units so that the units of `uniflux` will be the units of `wideflux` divided by the units of `H`.

The omnidirectional flux is a special case of the wide-angle flux problem:

$$h(\alpha, \phi) = \frac{1}{4\pi}. \tag{82}$$

First, we discretize this equation:

$$\begin{aligned} f_{\text{iso}} &\approx \sum_i H_i f_i, & (83) \\ H_i &\approx \int_0^{2\pi} h(\alpha_i, \phi) d\phi \sin \alpha_i \Delta \alpha_i, & (84) \\ f_i &= f_{\text{uni}}(\alpha_i). & (85) \end{aligned}$$

TEM-1 follows Vette's AE-8 atmospheric cutoff:

$$\frac{B_m}{B_0} = \begin{cases} 0.6572 L_m^{3.452} & L_m < 2.4 \\ 0.196 L_m^{4.878} & 2.4 \leq L_m < 3.0 \\ 1.4567 L_m^{3.050} & L_m > 3.0 \end{cases} \tag{86}$$

Any flux outside (i.e., with B_m/B_0 beyond the limit) is treated as implicitly zero by removing it from the $\vec{f}$. By implication, there is no electron flux at any pitch angle for $L_m < 1.1293$. Requesting a unidirectional flux at a local pitch angle that mirrors beyond the cutoff will result in an error code (see Return Codes, section 4.4).

Next, we set the problem up as a constrained maximum likelihood equation in the log fluxes:

$$\begin{aligned} x_i &= \ln f_i, & (87) \\ y &= \texttt{wideflux}, & (88) \\ \delta y &= \texttt{dlogwideflux}, & (89) \\ \lambda &= \sum_i H_i f_i, & (90) \\ \ell &= \Lambda(\ln \lambda - \ln y) + \frac{1}{2}(\vec{x} - \vec{\mu})^T \underline{\underline{\Sigma}}^{-1} (\vec{x} - \vec{\mu}). & (91) \end{aligned}$$

The LaGrange multiplier Λ ensures that the local minimum in ℓ satisfies $y = \lambda$, i.e., the inverted angular distribution exactly reconstructs the observed wide-angle flux.

Note: I also tried using a typical log-normal penalty function on y based on δy, but that approach causes the solution $\vec{x}$ to be systematically biased toward $\vec{\mu}$. This is a typical problem with maximum likelihood. I believe we ought to treat separately the uncertainty in the wide-angle measurement and the uncertainty in the conversion to unidirectional flux. This LaGrange multiplier approach does just that.

The second term in (91) is the likelihood of a particular local pitch angle distribution at the specified energy and location according to TEM1. TEM1 provides the median $m_{50}(E, \alpha_e q, L_m)$, 95$^{\text{th}}$ percentile $m_{95}(E, \alpha_e q, L_m)$, and spatial correlation coefficient $\rho(\alpha_{eq,i}, \alpha_{eq,i'}|E, L_m)$, and we can convert between α and α_{eq} using the B/B_0 parameter.

$$\begin{aligned}
\sin\alpha_{eq,i} &= \frac{\sin\alpha_i}{\sqrt{B/B_0}}, && (92)\\
\mu_i &= \ln m_{50}(E, \alpha_{eq,i}, L_m), && (93)\\
\sigma_i &= (\ln m_{95}(E, \alpha_{eq,i}, L_m) - \mu)/\Phi^{-1}(0.95), && (94)\\
\Phi^{-1}(0.95) &= 1.6448536270, && (95)\\
\Sigma_{ii'} &= \sigma_i\sigma_{i'}\rho(\alpha_{eq,i}, \alpha_{eq,i'}|E, L_m). && (96)
\end{aligned}$$

To solve for ℓ will require the usual gradients and derivatives:

$$\begin{aligned}
\frac{\partial\lambda}{\partial x_i} &= H_i f_i && (97)\\
\frac{\partial\ell}{\partial x_i} &= (\Lambda/\lambda)H_i f_i + \sum_{i'}\left(\underline{\underline{\Sigma}}^{-1}\right)_{ii'}(x_{i'} - \mu_{i'}), && (98)\\
\frac{\partial\ell}{\partial\Lambda} &= \ln\lambda - \ln y, && (99)\\
\frac{\partial^2\ell}{\partial x_i\partial x_{i'}} &= (\Lambda/\lambda)H_i f_i\delta_{ii'} - (\Lambda/\lambda^2)H_i f_i H_{i'} f_{i'} + \left(\underline{\underline{\Sigma}}^{-1}\right)_{ii'}, && (100)\\
\frac{\partial^2\ell}{\partial x_i\partial\Lambda} &= H_i f_i/\lambda, && (101)\\
\frac{\partial^2\ell}{\partial\Lambda^2} &= 0. && (102)
\end{aligned}$$

The initial guess is $y/(\vec{H}^T\exp(\vec{\mu}))$, i.e., the median fluxes are rescaled to reconstruct y. The fit for $\vec{x}$ is performed with multivariate root finding routines provided by the Gnu Scientific Library, producing the maximum likelihood value $(\hat{\vec{x}}^T, \hat{\Lambda})$, which finds simultaneous zeros of (98) and (99).

Thus the estimated flux is:

$$\texttt{uniflux} = \exp(\hat{x}_{i_{\alpha_0}}). \tag{103}$$

We discard all the other components of $\hat{\vec{x}}$ because we want to conserve the number of observations (one cannot get something for nothing) for subsequent analysis. Doing otherwise would risk over-weighting the "typical" pitch angle distribution shape in subsequent analysis of the inverted flux.

The uncertainty on the log flux is given by:

$$\begin{aligned}
\texttt{dloguniflux} &= \sqrt{(\delta y)^2 + (\delta\ln f_{i_{\alpha_0}})^2}, && (104)\\
\delta\ln f_{i_{\alpha_0}} &= \delta x_{i_{\alpha_0}} = \sqrt{Q_{i_{\alpha_0}, i_{\alpha_0}}}, && (105)\\
\underline{\underline{Q}} &= \begin{pmatrix} & \vdots & & \\ \cdots & \left.\frac{\partial^2\ell}{\partial x_i\partial x_{i'}}\right|_{(\hat{\vec{x}}^T,\hat{\Lambda})} & \cdots & \left.\frac{\partial\ell}{\partial\Lambda}\right|_{(\hat{\vec{x}}^T,\hat{\Lambda})} \\ & \vdots & & \\ & \left.\frac{\partial\ell}{\partial\Lambda}\right|_{(\hat{\vec{x}}^T,\hat{\Lambda})} & & 0 \end{pmatrix}^{-1}. && (106)
\end{aligned}$$

Note that $\underline{\underline{Q}}$ is the inverse of the Hessian of ℓ in the $(\vec{x}^T, \Lambda)$ super-space, and `dloguniflux` is a combination of the angular inversion error and the measurement error `dlogwideflux`, as if they were independent (not a bad assumption).

4.2 Calculations Performed - Vampola Method

The Vampola method is based on a $\sin^n \alpha$ fit, where n depends on the L shell, in this case McIlwain L_m in Olson-Pfitzer Quiet. Table 1 gives the coefficients provided in *Vampola* [1996]. For L_m in bounds, linear interpolation is used. For L_m out of bounds, the nearest boundary value is used.

Table 1: Exponent in $\sin^n \alpha$ fits from *Vampola* [1996]

L_m	n
3.00	5.380
3.25	5.078
3.50	4.669
3.75	3.916
4.00	3.095
4.25	2.494
4.50	2.151
4.75	1.998
5.00	1.899
5.25	1.942
5.50	1.974
5.75	1.939
6.00	1.970
6.25	2.136
6.50	1.775
6.75	1.438
7.00	1.254
7.25	1.194
7.50	1.046
7.75	0.989
8.00	0.852

Whereas the TEM-1 method required a constrained minimization, the Vampola method requires only a numerical integral:

$$f_{\mathrm{uni}}(\alpha) = f_0 \sin^n \alpha, \tag{107}$$

$$f_{\mathrm{iso}} \approx \sum_j H_j f_0 \sin^n \alpha_j, \tag{108}$$

$$\texttt{uniflux} = \frac{f_{\mathrm{iso}}}{\sum_j H_j f_0 \sin^n \alpha_j} \sin^n \alpha_0. \tag{109}$$

The Vampola method also uses the AE8 atmospheric cutoff, like TEM-1. Finally, because Vampola did not provide error estimates on n, the output error estimate (`dloguniflux`) for the Vampola method is the input error (`dlogomniflux` or `dlogwideflux`).

4.3 Function Arguments

4.3.1 omni2uni

- `omniflux` (input) Estimated isotropic unidirectional flux (for TEM-1 method, use #/cm^2/sr/s/keV).
- `dlogomniflux` (input) relative error for `omniflux` (dimensionless).
- `int_params` (input) integer parameters, length = 5 (TBR).

  `int_params[0]` `NA` = number of grid points for angular integral.

  `int_params[1]` angular inversion method:

  [-1] - TEM-1.

  [-2] - Vampola.

int_params[2] Verbose setting (choose one):

[0] no text output,

[1] text output to standard output stream,

[2] text output to standard error stream,

[3] text output to outFile (assumes outFile is actually a FILE *).

[4] text output to outFile, overwrite existing file

[5] text output to outFile, append to existing file

int_params[3] Choice of root finder (choose one):

[0] Powell's Hybrid method, scaled (recommended),

[1] Powell's Hybrid method

[2] Newton's method

[3] Pseudo-global Newton's method

[4] Powell's Hybrid method w/ numerical Hessian, scaled

[5] Powell's Hybrid method w/ numerical Hessian

[6] Discrete Newton's method (numerical Hessian)

[7] Broyden's algorithm (numerical Hessian)

int_params[4] Maximum number of iterations by root finder (recommend 1,000).

- real_params (input) real parameters, length 3 (TBR).

  real_params[0] = Energy of particle flux, keV

  real_params[1] = B/B_0 of spacecraft location, dimensionless.

  real_params[2] = L_m, McIlwain L for locally mirroring particle, in Olson-Pfitzer Quiet.

- outFile (input) provides filename or FILE * for verbose output (see verbose setting above). Null terminated string.
- uniflux (output) Locally-mirroring unidirectional flux, (e.g., in #/keV/cm^2/sr/s).
- dloguniflux (output) Standard error of natural log of uniflux (dimensionless).

4.3.2 wide2uni

- wideflux (input) Estimated isotropic unidirectional flux (for TEM-1 method, use #/cm^2/sr/s/keV).
- dlogwideflux (input) relative error for wideflux (dimensionless).
- PAgrid (input) PAgrid[i] local pitch angle at grid point i, degrees, must be monotonically increasing grid.
- H (input) H[i] Angular weighting for unidirectional flux at grid point i, dimensionless.
- int_params (input) integer parameters, length = 5 (TBR), same as omni2uni except:

  [0] = NA number of pitch angles in grid.

- real_params (input) same as omni2uni.
- outFile (input) same as omni2uni.
- ialpha0 (input) zero-based index of pitch angle grid point for uniflux output (e.g., grid point nearest sensor bore sight).
- uniflux (output) same as omni2uni.
- dloguniflux (output) same as omni2uni.

Notes:

1. Pointers are used throughout for ease of access from FORTRAN, which can only pass arguments by reference (i.e., pointers).
2. Array and matrix indices above are zero-based, following C convention rather than one-based linear algebra convention.
3. `omni2uni` and `wide2uni` operate on estimated isotropic unidirectional flux.
4. Alternatively, `H` and `wideflux` can be defined in terms of omnidirectional flux, so long as the units of `wideflux/H` give units of `uniflux`.
5. If `dlogomniflux` or `dlogwideflux` is unknown, use ln(2)/2 for factor of 2 95% confidence bounds.
6. `omniflux`, `wideflux`, `dlogomniflux`, or `dlogwideflux` negative or zero will result in an error (see Return Codes, section 4.4).
7. `outFile` is ignored for verbose settings 0, 1, and 2.
8. The 95% confidence interval on `uniflux` is given by `uniflux` $\times \exp(\pm 1.9600 \times$ `dloguniflux`).

4.4 Return Codes

1 Success! No Error.

0 Unknown error (this only happens if there's a bug).

-101 NULL passed where pointer expected.

-102 One or fewer valid data points (`NY`≤ 1), or `NE`≤ 1 or `NEout`< 1.

-103 One or more invalid (NaN or infinite) in input arrays, or some `y[i]`< 0, `dy[i]`< 0, `Egrid[j]`≤ 0, or `Egrid[j]`≤ 0.

-104 Negative or zero input `omniflux`, `wideflux`, `dlogomniflux`, `dlogwideflux`.

-401 Invalid minimizer or root finder selected.

-402 Invalid iterations requested (zero or negative).

-501 Invalid value of verbose provided or user-requested file stream NULL (outFile).

-502 User-requested verbose output file (outFile) could not be opened.

-601 Output pitch angle requested out of range (index out of range or B_m beyond cutoff).

-602 Invalid angular inversion method requested.

4.5 Validation

The routine `omni2uni` has been validated for the TEM-1 method by running it for various cases and examining the output for sanity (not very sophisticated validation). The C test code is is "omni2uni_test.c", which passes a single test case to `omni2uni` and prints it output.

Here is a transcript of `omni2uni_test.exe`:

```
[PROMPT]> ./omni2uni_test.exe
omni2uni_test:inputs = omni=1.00000e+04 (dlog=3.46574e-01) @ [300.0 keV, B/B0=400.00, Lm=6.60]
fzero Invoked with solver hybridsj, MaxIter=1000
fzero: 1/1000: 0.313671:
fzero: 2/1000: 0.0140756:
fzero: 3/1000: 1.27369e-06:
fzero: 4/1000: 3.37084e-09:
fzero: completed after 4/1000: success
omni2uni_test:inputs = omni=1.00000e+04 (dlog=3.46574e-01) @ [300.0 keV, B/B0=400.00, Lm=6.60]
omni2uni_test:result = 1, uni=2.73852e+04 (dlog=3.46656e-01)
[PROMPT]>
```

In this case, because $B/B0 >> 1$, there is a substantial loss cone, thus creating a significant pitch-angle anisotropy, which results in a much larger locally-mirroring flux than would be assumed from isotropy alone (by a factor of 2.7...).

5 Dependencies and Compiling

The inversion library routines rely on the GNU Scientific Library, available for free download at http://www.gnu.org/software/gsl The GSL base library and the GSL CBLAS library are required (both come with a typical GSL install).

The library is built for the gcc compiler suite. A makefile ("Makefile") is provided. The `make` command alone will display the helps required to build the `invlib` shared object (DLL).

The compilation may generate some `warning` and `Info` messages:

```
warning: comparison between signed and unsigned
warning: assignment discards qualifiers from pointer target type
Info: resolving _gsl_multimin_fdfminimizer_conjugate_fr by linking to
 __imp__gsl_multimin_fdfminimizer_conjugate_fr (auto-import)
Info: resolving _gsl_multimin_fdfminimizer_conjugate_pr by linking to
 __imp__gsl_multimin_fdfminimizer_conjugate_pr (auto-import)
Info: resolving _gsl_multimin_fminimizer_nmsimplex by linking to
 __imp__gsl_multimin_fminimizer_nmsimplex (auto-import)
Info: resolving _gsl_multimin_fdfminimizer_vector_bfgs2 by linking to
 __imp__gsl_multimin_fdfminimizer_vector_bfgs2 (auto-import)
```

I have not figured out how to make a stand-alone DLL in cygwin, I think because GSL libraries depend on "cygwin1.dll". However, I have created a DLL with MSYS/MINGW, and the makefile has commands to do this.

6 High-level language interfaces to INVLIB

6.1 Python

An interface to the C inversion library has been written in Python to allow rapid scripting and testing of the library. The package is compatible with Python versions $\geq$2.6, including Py3k.

To install the Python package, both Python and NumPy must be installed first. If MatPlotLib is installed additional visualization tools will be available. In the directory containing the inversion library, type:

```
python setup.py install --home=<dir>
```

where the `--home` option defines a non-standard installation directory. To install in the default location (by omitting the flag) may require administrative privileges. To access the interface from within Python, import the package `pyinvlib`.

The unit-testing suite included with this interface, "invlib_test.py", can be run either directly from the checked-out SVN repository or from the install directory. It includes tests for a number of methods within the interface, as well as basic regression tests for the underlying C library. Currently this represents the most comprehensive test suite for the C inversion library and it is recommended that the library

As usual, the documentation for `pyinvlib` is contained in the source code in the' form of docstrings. From a Python environment, this can be accessed using the Python built-in `help()` function (within the iPython interpreter simply appending a `?` will do). e.g.,

```
>>> import pyinvlib as pinv
>>> help(pinv.SpecInv)

Help on class SpecInv in module pyinvlib:

class SpecInv(InvBase)
 |  Spectral Inversion class using INVLIB
 |
 |  This class is designed to hold all necessary inputs to *_spec_inv* from
 |  the INVLIB library. The calls to INVLIB routines are implemented as
```

```
|  object methods.
|
|  Example use:
|  import package (required)
|  >>> import pyinvlib as pinv
|  instantiate and populate attributes
|  >>> dum = pinv.SpecInv(verbose=True)
|  >>> dum.counts = [4.15524e+02, 3.70161e+02, 2.42137e+02, 2.12097e+02,
|  1.47379e+02, 1.40524e+02, 9.37500e+01, 5.08064e+01, 1.93548e+01, 3.22581e+00]
|  >>> dum.dcounts=[0.3466]*10
|  >>> dum.readRespFunc(fname='sopa_LANL-97A_t2_HSP_elec.001')
|  >>> dum.setParams(fnc=8)
|  >>> dum.anaSpecInv()
|  plot the output spectrum with errorbars (using matplotlib)
|  >>> fig = dum.plot()
|
|  Method resolution order:
|      SpecInv
|      InvBase
|      __builtin__.object
|
|  Methods defined here:
...
```

The package defines two new classes for the user, `SpecInv` and `AngInv`, for spectral inversion and angular inversion, respectively. (Note: these are sub-classed from the `InvBase` class, which is not designed to be instantiated and provides jointly inherited behavior).

For comprehensive and up to date documentation on the Python interface, either read the help from the Python environment or use a package like "epydoc" to automatically generate comprehensive documentation in either HTML or PDF form.

A Adding a new spectral form

The following modifications must be made to add a new spectral form:

In `specinv.h`,

- add a new `ASI_FXN_<NEW>` constant, the next power of two.
- increment `ASI_FXN_POW2` constant,
- insert `ASI_FXN_<NEW>` in bitwise or of all allowed functions (`ASI_FXN_ALL`)

In `invlib_const.h`, add any new error codes associated with improper options/settings for new analytical spectrum.

In `specinv.c`,

- create new `flux_<new>` function, following template from existing analytical flux functions.
- add input checks to `ana_spec_inv`, e.g., checking for existence and positivity of rest mass parameter in `params`.
- add case for `ASI_FXN_<NEW>` to `switch(fxn_bit)` in `ana_spec_inv`.

In `invlib.tex` (this file), add appropriate documentation:

- Note changes in section 2, "Changes".
- Add definition and derivatives of new spectral function to section 3.1.3.
- Update argument list for `ana_spec_inv`, section 3.3.

Test compilation with `make` commands. Recompile this file with `pdflatex`.

References

Vampola, A.L. Outer zone energetic electron environment update, *Final Report of ESA/ESTEC/WMA/P.O. 151351*, ESA-ESTEC, Noordwijk, The Netherlands, 1996.