

The code on solving Abel equation: usage instructions

I.I.Antokhin

Sternberg State Astronomical Institute, Lomonosov Moscow State University

September 24, 2016

1 A general note

While some code is implemented to preven most evident user errors (like setting the initial approximation for z outside the region defined by the constraints etc.), the code is not fool proof. Making it such would take a lot of efforts and time which I do not have. Of course, in case of any problems I would be glad to help.

2 Files in the package

<code>prgrad_reg.c</code>	- functions to solve Fredholm and Abel equations
<code>wr_o_common.c</code>	- equation kernels for two types of Fredholm eqs. and Abel eq, also functions computing elliptical functions
<code>abel_prgrad_test.c</code>	- test program showing how to run functions from <code>prgrad_reg.c</code>
<code>Makefile</code>	- make file to compile <code>abel_prgrad_test</code>
<code>sim_power32.dat</code>	- simulated model data for the model 1 from my paper. Contains exact solution and exact input data for <code>abel_prgrad_test</code>
<code>sim_power32.abel</code>	- the results of solution for the data from <code>sim_power32.dat</code> with added Gaussian noise.
<code>abel_usage.pdf</code>	- this file

3 Compiling

The code is developed in Linux and compiled with gcc 4.8.3. There should be no problem compiling it with any ANSI C compiler. In a non-Linux OS you will have to replace the random generator functions `srandom` and `random` with your local versions. These functions are used in the test code to add Gaussian noise to the simulated data.

Note that in UNIX/Linux lines of text files are ended with `<LF>` while in Windows with `<CR><LF>`. If you use Windows you may want to convert the files to `<CR><LF>` format, otherwise you will see a file as a single line.

4 The structure of the calling (main) program

It is linear and very simple. See comments in the source code `abel_prgrad_test.c`. Outline:

```
Define necessary variables.
Read input data.
Calculate the uncertainty of input data and the sum of weights.
Compute the constraints matrix and its right-hand part.
```

Set initial approximation for z .
 Call the main function solving the equation.
 Calculate model Az
 Output the results.

In the test code I print all output to stdout. You may redirect it to a file.

Variables: In my code memory for all arrays is allocated dynamically. If you prefer, you may set most of 1D arrays (u , z , az , grids, weights etc.) in the calling program as static. Exceptions are the constraints matrix `con` and its right-hand part `b` which are allocated within the function `ptilrb` and must be defined as pointers (see the source code in `abel_prgrad_test.c` and comments below).

Read input data: For simplicity, in `abel_prgrad_test.c` I hard-coded reading them in the body of the `main` function. You may wish to write a separate function and call it from `main`. Same for setting algorithm parameters.

Uncertainty of input data and sum of weights: These are numeric representations of uncertainty

$$\delta^2 = \int_{x_0}^{x_m} \sigma^2(x) dx$$

(where $\sigma(x)$ is uncertainty of $u(x)$) and the integral of weights

$$sumv = \int_{x_0}^{x_m} v(x) dx$$

where $v(x)$ is weight of $u(x)$, e.g. $\sim 1/\sigma^2$

5 Description of functions to call from your main program

5.1 Constraints matrix

A priori constraints have the form

$$con \cdot \vec{z} \leq \vec{b}$$

where `con` is the constraints matrix and \vec{b} is the right-hand part. They are computed by

```
void ptilrb( int kernel_type, int switch_contype, int *n_con, double ***con,
            double **b, int n, double c2, int icore, int l, int *ierr );
```

Parameters:

```
kernel_type    - 1,2, or 3. 1 and 2 are kernels for Fredholm eq. when solving light curves
                  of WR+O binaries. For Abel equation, you must set it to 3.
switch_contype - type of constraints:
                  = 1 - monotonically non-increasing and non-negative
                      (constraints 2,3,4 also include this constraint)
                  = 2 - concave  $z'' \leq 0$ 
                  = 3 - convex  $z'' \geq 0$ 
                  = 4 - concave-convex with inflection point index  $l$ 
                  = 5 - non-negative (this is to use Tikhonov's regularization only),
                      used only with Abel eq., that is kernel_type=3
n_con          - the number of rows in con, computed within the function
```

con	- constraints matrix. Note that in the calling program you must declare it as "double **con". Computed within the function.
b	- right-hand part of constraints. In the calling program must be declared as "double *b". Computed within the function.
n	- dimension of z.
c2	- Is not used in this function, set to an arbitrary value. However, you will need it for initial approximation, so may set to the desired value before calling ptlrb.
icore	- For Abel equation, MUST be set to 0: icore=0.
l	- Index of inflection point for switch_contype=4. For other constraints, not used, set to an arbitrary value.
ierr	- return code. 0 if all is ok, 202 if memory for con or b could not be allocated.

Note that when calling ptlrb, you must use &con, &b and not just con, b.

5.2 Initial approximation

```
void initialapprox( int switch_contype, double c2, double *s, double *z, int l, int n )
```

switch_contype	- same as above
c2	- The value of z[0] of initial approximation. Should be roughly up to the expected scale of z. E.g. if you expect z to vary between 0 and 100, set c2=20 or 50 etc. Do not set it to e.g. 1e10 or 1e-10 as this may cause numerical problems.
s	- grid on s - array of size n. Argument of z. "s" == "r" from the paper. IMPORTANT: grid must be even (s[i+1]-s[i]=const)!!!
z	- Unknown function z. Array of size n.
l	- Index of the inflection point, see above.
n	- Size of grid on s. Dimension of s and z.

5.3 The main function

```
void ptizr_proj( int kernel_type, int switch_contype, int n_con, double **con, double *b,
double rstar, double *u0, double *v, double sumv, double *s, double *x,
double xmin, double xmax, int n, int m, double *z, double c2, double dl2,
double eps, double h, int adjust_alpha, double *alpha, char *metric,
int l, int icore, double ax, double *del2, int imax_reg, int *iter_reg,
int imax_minim, int *iter_minim, int verbose, int *ierr );
```

kernel_type	- Same as above.
switch_contype	- Same as above.
n_con	- Same as above.
con	- Same as above.
b	- same as above.
rstar	- Is not used for Abel equation, set to an arbitrary value.
u0	- Input data, right-hand part of Abel eq. Array of size m. =u from the paper.
v	- Weights of u0 data points.
sumv	- Sum of weights (see the code in abel_prgrad_test.c).
s	- Same as above.

x	- Argument of u0. Array of size m.
xmin	- Minimal value of x.
xmax	- Maximal value of x.
n	- Dimension of z.
m	- Dimension of u0. The number of input data points.
z	- Unknown function. Array of size n. On return, contains the solution.
c2	- Same as above.
delta2	- Uncertainty of input data (see above).
eps	- Threshold for solving $ Az-u ^2 - \delta^2 = 0$. See below.
h	- If known, uncertainty of the A operator. The difference between exact Az and its numeric approximation. At reasonably large n, it is safe to set h=0.0.
adjust_alpha	- If 1, search for optimal regularization parameter. If 0, solve at fixed alpha.
alpha	- Either alpha value to use or its initial value (see below).
metric	- Metric space to use. May be "L2", "W21", or "W22".
l	- Index of the inflection point as above.
icore	- As above. For Abel eq. set icore=0.
ax	- Not used for Abel eq. Set to an arbitrary value.
dl2	- Residual of the model relative to the input data.
imax_reg	- Max number of regularization iterations.
iter_reg	- Actual number of regularization iterations made.
imax_minim	- Max number of conjugate gradients minimization iterations made in the last regularization iteration (if any).
iter_minim	- Actual number of conjugate gr. iterations made.
verbose	- If set to 1, some additional stdout printing is done. Set to 0 to skip this.
ierr	- Return code. 1** - normal end, 2** - various errors = 100 - normal end, exact minimum is found = 101 - iterations finished by residual value = 102 - iterations finished by the norm of the gradient = 103 - alpha became equal to zero while doing iterations = 104 - when doing initial try at alpha=0, $\alpha^4 \geq \text{eps} \cdot \text{dl}2$, regularization impossible = 200 - initial approximation outside allowable range. = 201 - inconsistent adjust_alpha and alpha = 202 - errors allocating memory for working arrays = 203 - singular matrix of active constraints (when computing the projector) = 204 - initial alpha < 0 = 205 - while searching for the interval on alpha containing the solution, made imax_reg multiplications by 2.0, residual still negative. Initial regularization parameter too small. = 206 - max number of "big loop" (imax_minim) iterations reached, no solution. = 207 - in bisection method, imax_reg iterations done, still not reached exit criteria. = 208 - kernel_type <1 or >3.

Hopefully, the meaning of most of the parameters above is clear. A few comments on some of them:

Parameters controlling Tikhonov's regularization:

If `adjust_alpha=0` and `alpha=0`, no Tikhonov's regularization is done. The equation is solved on a compact set. If `adjust_alpha=0` and `alpha>0`, Tikhonov's regularization at this fixed value of `alpha` is performed. If `adjust_alpha=1`, `alpha` must be positive, otherwise the function will return an error. This positive value is used as the starting point in searching for the optimal `alpha`.

Optimal `alpha` is defined by the condition

$$an4 \equiv ||Az - u||^2 - \delta^2 = 0$$

The function on the left is a monotonically increasing function of **alpha** (see Fig.1). So the solution of the above equation is located between the points **alpha=0** and some **alpha_1** such that the function on the left is positive. The search is performed as follows:

1. Compute **an4** at **alpha=0**. If **an4<0**, proceed, otherwise exit (see below).
2. Compute **an4** at **alpha** set before calling **ptizr_proj**.
3. If **an4** is negative, multiply **alpha** by 2 and repeat (2).
4. Repeat (3) until **an4** becomes positive (at some **alpha_1**).
5. Now, the solution is between **alpha=0** and **alpha=alpha_1**.
6. Use the bisection method to solve the above equation.

In practical term, the solution is considered to be found if

$$||Az - u||^2 - \delta^2 < \epsilon\delta^2$$

Thus, variable **eps** sets the threshold for solution of the above equation.

There may be a situation when even at **alpha=0**

$$||Az - u||^2 - \delta^2 > 0$$

. This may happen e.g. if a priori constraints do not allow the model to fit the data very well. Or, if the uncertainty of input data δ^2 is underestimated. In this case, **an4** > 0 even at **alpha=0**. No Tikhonov's regularization is possible in this case.

Inflection point

If you want to use the concave-convex type of constraints, you must set the position (index) of the inflection point. If it is known, no problem. If not, you can search for it by repeatedly calling **ptizr_proj** with various values of **l** and choosing the optimal value by the minimum of **del2**. Clearly, **l>=1** and **l<=n-2** (recall that in C array indexes start from 0 so the first and last indexes of an array of the length **n** are 0 and **n-1**). Otherwise, **z** will be simply convex or concave. You can use these values of **l** as the end points of the **l** interval and find the optimal **l** by e.g. the golden section method. I have the code for the method but did not included it in the test code, as I wanted to keep it simple. In any case, it is easy to implement.

One further note has to be made. If you want to use Tikhonov's regularization with the concave-convex constraint, do not use it when searching for the optimal **l**. The reason is that with Tikhonov's regularization, if it is possible, you will always get **del2=delta^2** and will not be able to choose optimal **l** by the minimum of **del2**. Also, at different **l** the values of optimal **alpha** will be different and solutions are hard to compare. So search for the optimal **l** not using Tikhonov's regularization, and then run **ptizr_proj** once more with the optimal value of **l** (just found) AND Tikhonov's regularization switched on.

5.4 Computing model **u** (**Az**)

This is simply calculating the Abel integral with model **z**. To do this, first compute the kernel of Abel equation (of course, it is computed in **ptizr_proj**, but it is a local variable not seen outside **ptizr_proj**).

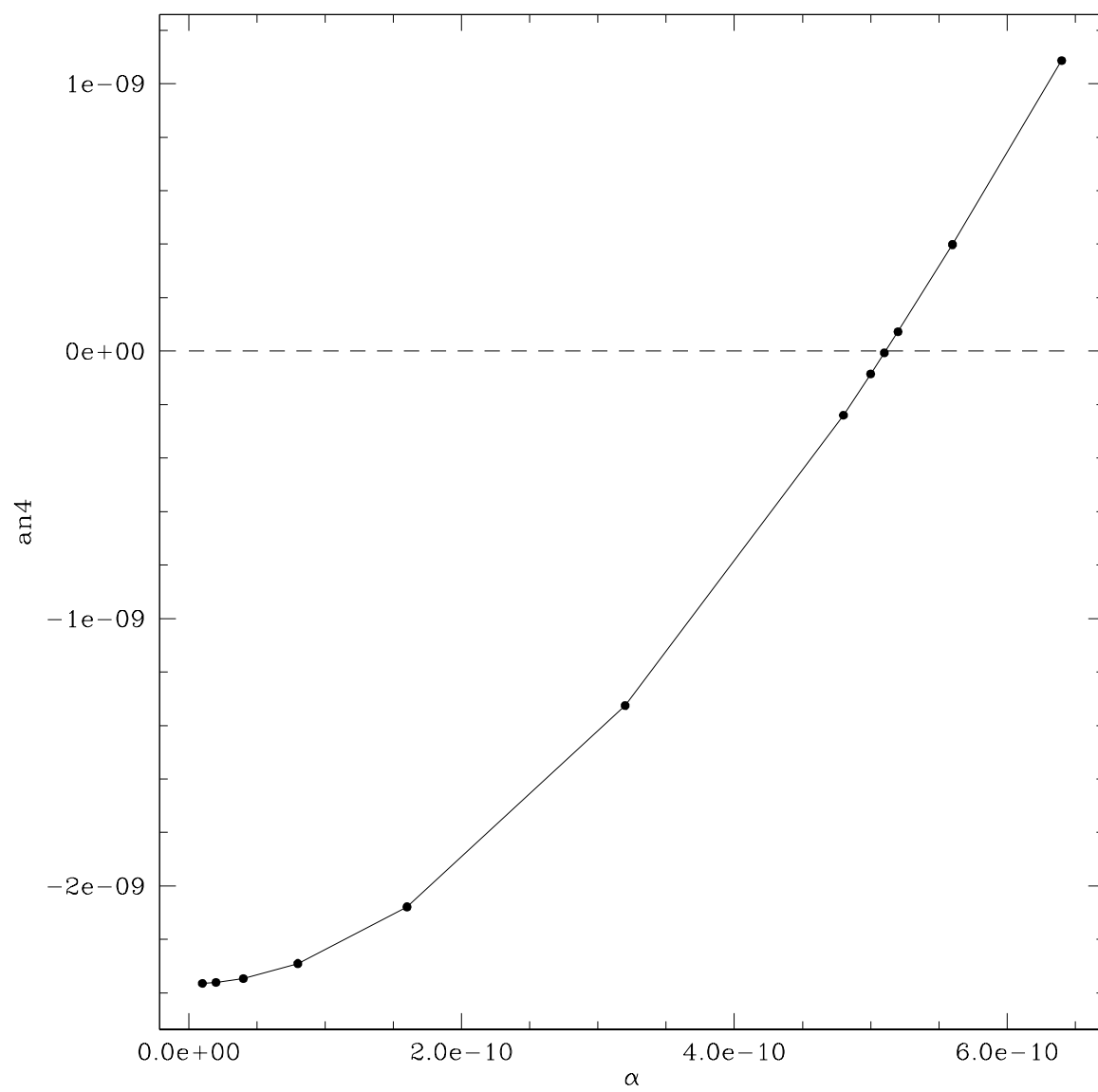


Figure 1: Dependence of $an4$ from α

Allocate memory for the kernel `a_abel`. Function `"matrix"` allocates memory for a 2D array with `m` rows and `n` columns,

```
double **a_abel = matrix( int m, int n )
```

Compute Az .

```
void pticr3( double **a, double *z, double *az, int n, int m )
```

See comments in the code of `abel_prgrad_test.c`.