Global Iterative Solution —
Distributed processing of the attitude updating

L. Lindegren (21 May 2001)
SAG-LL-37

ABSTRACT. The attitude updating algorithm given in GAIA-LL-24 (v. 2) is modified to allow distributed processing in which each processor takes care of a subset of the sources according to the sky partitioning. A detailed (but very simplified) numerical example is provided. The same principle can be used to distribute the calibration and global processing.

1 Introduction

In GAIA-LL-34 (v. 2), detailed algorithms in the form of fortran subroutines were provided for a number of processes in the Global Iterative Solution (GIS) to be implemented in the on-going GAIA Data Access and Analysis Study (GDAAS). It is recalled that the GIS basically consists of the following processes to be executed cyclically until convergence:

- source processing – one source (astronomical object) at a time;
- attitude processing – one attitude interval (some 12 hr) at a time;
- calibration – one CCD at a time;
- global processing – needs to consider the whole mission.

It is envisaged that the GIS will be executed in a distributed environment with several processes running in parallel. In order to minimize large-scale data shuffling it seems natural to associate a specific area on the sky (following the HTM division) with each local processor. This makes it very straightforward to distribute the source processing, since that only requires access to a very small area of the sky at a time. It is less obvious if and how the other processes can be distributed without a dramatic increase in overhead.

In this document the attitude processing is described in some detail, including a ‘worked example’ using the subroutines supplied with GAIA-LL-34 (v. 2). All the processes mentioned above are instances of non-linear least-squares estimation, implemented (as proposed for the GIS) by the classical method using normal equations. The general idea for distributed processing using normal equations is discussed in Sect. 2 and then applied to the attitude processing as an example involving fairly many unknowns (Sect. 3). The same method can be used for the calibration and global processing; here the normal equations only involve one or a few unknowns, so the situation is much simpler than for the attitude processing.
2 Least-squares by distributed processing

All the processes described above solve a non-linear least-squares problem by the use of normal equations. The non-linearity arises primarily from the need to take care of outliers by means of downweighting. Let \( p \) be the parameter vector to be estimated and \( m = \dim(p) \) the number of unknowns (parameters). For given observations \( o \) (with \( \dim(o) \gg m \)) the quantity to minimize is

\[
Q(p) = \sum_k \rho \left( \frac{o_k - f(t_k|p)}{\sigma_k} \right)
\]

where \( f \) is the observation model, \( \rho \) the loss function, \( \sigma \) the standard error assigned to each observation, and \( k \) the observation index. For ordinary least-squares the loss function is simply \( \rho(z) = z^2 \), but in order to accommodate outliers this should be modified by a factor which reduces the weight of observations with large \( |z| \). Tentatively, the Huber metric has been proposed,

\[
\rho(z) = z^2 \times \begin{cases} 
1 & \text{if } |z| \leq c \\
\frac{c(2|z| - c)}{2} & \text{otherwise}
\end{cases}
\]

(not the typo in Eq. 2 of GAIA-LL-34.2). Problem (1) can be solved iteratively from an initial approximation \( p \) by means of the equations:

\[
A_{ij} = \sum_k \frac{w_k \partial f_k \partial f_k}{\sigma_k^2} \frac{\partial^2 f_k}{\partial p_i \partial p_j}, \quad i, j = 1 \ldots m, 
\]

\[
b_i = \sum_k \frac{w_k \partial f_k}{\sigma_k^2} (o_k - f_k), \quad i = 1 \ldots m,
\]

\[
x = A^{-1}b,
\]

\[
p \leftarrow p + x.
\]

This is iterated until \( |x| \) is negligible. Here \( w_k \) is the weight reduction factor calculated for each observation based on the size of the normalized residual \( z_k \equiv (o_k - f_k)/\sigma_k \).\(^1\) The linear system of equations \( Ax = b \) is called the normal equations (\( A^{-1} \) is also an estimate of the covariance of the resulting \( p \)). Since \( A \) is symmetric of size \( m \times m \) and \( b \) is of size \( m \times 1 \) the total number of reals to be stored for the normal equations is (at most) \( m(m + 3)/2 \). (For the attitude updating \( A \) is band-diagonal which means that the actual storage is much smaller and increases only linearly with \( m \).)

Figure 1 is a schematic illustration of the steps necessary to perform the non-linear least-squares estimation according to this method. In addition to the normal equations, some additional statistics are usually calculated in step 2, here exemplified by the total number of observations \( n \) and the mean weight reduction factor \( w \).

\(^1\)Actually, the downweighting is not done quite as intended in the attached subroutines, i.e. what is done does not correspond to the Huber metric. This is of little consequence at the present stage, but eventually the treatment of outliers should be considered more carefully, which probably will lead to a modification at least of the function \texttt{whuber}.
Define
- starting approximation for the parameters (p)
- observational data (o)
- solution criterion (e.g. number of iterations)

Step 0: Initialize least-squares problem

\[ A = 0 \]
\[ b = 0 \]
\[ n = 0 \]
\[ wsum = 0 \]

(etc.)

Step 1: Initialize normal equations

\[ A := A + (\text{new data}) \]
\[ b := b + (\text{new data}) \]
\[ n := n + 1 \]
\[ wsum := wsum + (\text{new data}) \]

(etc.)

loop over observations

Iterate a few times, or until \(|x|\) is negligible

Step 2: Accumulate normal equations

\[ x := A^{-1}b \]
\[ p := p + x \]
\[ w := wsum/n \]

(etc.)

Step 3: Solve n.e. and update parameters

After convergence, output p, n, w and any other relevant statistic

Figure 1. Flow diagram for a typical non-linear least-squares problem such as the attitude updating. An important consideration is that the accumulated arrays (A, b and various statistics) are relatively small and of constant size for a given problem. For instance, A and b together need at most \(m(m + 3)/2\) reals, if \(m\) is the dimension of \(p\). For the attitude updating \(A\) is a sparse (band-diagonal) matrix which can be stored much more compactly. The algorithm \texttt{attupd} takes full advantage of this property.

The convenience of the normal-equations approach derives partly from the facts (a) that their size is typically much less than what is needed to store the observations — thus it is more convenient to send normal equations around than observations —, and (b) that the sums in Eq. (2) can be accumulated independently for disjoint sets of observations \((k)\), and then co-added before solving the normals and updating the parameter estimate. If the observational data are stored locally on \(N\) different nodes (each with its own processor or processors), then an obvious modification of this scheme is as shown in Figure 2.

3 A worked example

A numerical example demonstrates how the attitude updating subroutine \texttt{attupd} from \texttt{GAIA-LL-34.2} was modified to allow distributed processing.
3.1 The source catalogue

In order to avoid having to generate and store a huge source catalogue, a seeded source approach is used. This means that the source identification number (an integer in the range from 1 to $2^{31} - 1 \approx 2 \times 10^9$) is used to seed a random number generator which then provides the source parameters according to a given distribution model. The model used here is extremely simple: the position $(\alpha, \delta)$ is drawn from a uniform distribution over the sky while the remaining four astrometric parameters (parallax, proper motion and radial velocity) are set to zero. The subroutine `getstar` returns the six astrometric parameters for a given source number. Note that `getstar` uses a separate, very primitive random
number generator called \texttt{ran01}.

### 3.2 Observational data for input

Simple geometric observations in the form of angular coordinates \((\eta, \zeta)\) were generated for a time interval of one spin period (3 hr). The text file \texttt{obs.prn} is a list of 4253 simulated observations for the time interval 0.01 < \(t\) < 0.135 days. The nominal scanning law was used as ‘true’ attitude. Both Astro fields were used, each of size 0.66 x 0.66 deg, and a basic angle of 106 deg. The path of each source across the several CCDs was not simulated; instead selected observations were taken at random time intervals. The total number of observations per time interval is much smaller that for the real mission, but sufficient to fix the attitude for the chosen knot frequency. The observational standard errors were taken to be constant, viz. 0.1 mas along-scan and 2 mas across-scan per observation. For each observation, \texttt{obs.prn} gives the time (days), the source number, the field used (1 or 2), the observed along-scan coordinate \(\eta\) (rad), the standard error \(\sigma_\eta\) (rad), the observed across-scan coordinate \(\zeta\) (rad), and the standard error \(\sigma_\zeta\) (rad). The first five lines of \texttt{obs.prn} are shown by the following table:

<table>
<thead>
<tr>
<th>(t)</th>
<th>ID</th>
<th>Field</th>
<th>(\eta)</th>
<th>(\sigma_\eta)</th>
<th>(\zeta)</th>
<th>(\sigma_\zeta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.010000261968169</td>
<td>704896274</td>
<td>1.9287524364094</td>
<td>0.0000000004848</td>
<td>-0.03412891250</td>
<td>0.00000009663</td>
<td></td>
</tr>
<tr>
<td>0.010339194255986</td>
<td>1069884866</td>
<td>1.9287588761347</td>
<td>0.0000000004848</td>
<td>0.049428430912</td>
<td>0.00000009663</td>
<td></td>
</tr>
<tr>
<td>0.010428866597720</td>
<td>152831700</td>
<td>15.36289341948</td>
<td>0.0000000004848</td>
<td>0.02671468696</td>
<td>0.00000009663</td>
<td></td>
</tr>
<tr>
<td>0.01062886802226</td>
<td>775760992</td>
<td>1.922209749243</td>
<td>0.0000000004848</td>
<td>0.03064915462</td>
<td>0.00000009663</td>
<td></td>
</tr>
<tr>
<td>0.0107636899763</td>
<td>1165974252</td>
<td>6.360519578936</td>
<td>0.0000000004848</td>
<td>0.011669667034</td>
<td>0.00000009663</td>
<td></td>
</tr>
</tbody>
</table>

### 3.3 Starting approximation

The starting approximation for the attitude spline is given by the file \texttt{q_initial.prn}, which contains one line per knot. The columns are: time (days), \(S_{1n}\), \(S_{2n}\), \(S_{3n}\), and \(S_{4n}\), where \(S_{mn}\) are the basic spline coefficients defined in Sect. 5.3 of GAIA-LL-34.2. Note that the first column contains the adopted knot sequence, including three points on either side of the time interval over which the attitude is defined.

<table>
<thead>
<tr>
<th>(t)</th>
<th>(S_{1n})</th>
<th>(S_{2n})</th>
<th>(S_{3n})</th>
<th>(S_{4n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0086111</td>
<td>1.3828462</td>
<td>-7.6065143</td>
<td>3.5427477</td>
<td>-5.4030911</td>
</tr>
<tr>
<td>0.0097047</td>
<td>1.2963815</td>
<td>-7.6219960</td>
<td>3.4797236</td>
<td>-5.444048</td>
</tr>
<tr>
<td>0.0095704</td>
<td>1.2077432</td>
<td>-7.6364644</td>
<td>3.4162249</td>
<td>-5.4842619</td>
</tr>
<tr>
<td>0.0100000</td>
<td>1.1199439</td>
<td>-7.6498904</td>
<td>3.3622617</td>
<td>-5.6237383</td>
</tr>
<tr>
<td>0.0104626</td>
<td>1.0319963</td>
<td>-7.6523096</td>
<td>3.2879423</td>
<td>-5.65624696</td>
</tr>
</tbody>
</table>

### 3.4 Fitting the attitude spline (non-distributed processing)

The following main program implements the method in Figure 1 by successive calls to the subroutine \texttt{attupd} (supplied with GAIA-LL-34.2).

\footnote{This routine should be kept separate from the more sophisticated routines used to generate observation noise and similar things where a good degree of randomness is important. These latter routines should never be reinitialized (given a new seed) within a simulation.}
PROGRAM fit_attitude
  c
  c  Fit an attitude spline to the observations in file_obs.
  c
  c  An initial approximation of the attitude (as well as the
  c  knot sequence) is defined in file_ini. The final attitude
  c  estimate (after niter least-squares iterations) is given in
  c  file_out.
  c
  c  N.B.: the parameter N must chosen such that N+4 does not exceed
  c  the number of rows in file_ini; moreover, file_obs must contain
  c  sufficient observations covering the time interval tau(2:N-1),
  c  where tau(-1:N+2) is the knot sequence defined by the first
  c  column of file_ini.
  c
  c  L Lindegren (Land Observatory) 19 May 2001
  c
  IMPLICIT REAL*8 (a-h,o-z)
  INCLUDE 'const_math.h'
PARAMETER (N = 273, NB = 4*M, LDA = 16)
  REAL*8 tau(-1:N+2), bsc0(4,N), bsc1(4,N), wa(LDA,NB), wb(NB),
         + wc(NB), wd(NB), glb(1), apar(6),
         + wc(2), fobs(2), ferr(2)
CHARACTER file_obs*80, file_ini*80, file_out*80
  lu0 = 10
  lu1 = 11
  file_obs = 'obs.prn'
  file_ini = 'q_initial.prn'
  file_out = 'q stale.prn'
glb(1) = 1d0
  c------------------------
  c STEP 0 -
  c------------------------
  c Define knot sequence (tau), initial spline approximation (bsc0)
  c and number of iterations (niter):
  c
  OPEN(lu0,file=file_ini)
  DO i = -1, N+2
     IF (i.GE. 1 .AND. i.LE. N) THEN
        READ(lu0,*) tau(i), (bsc0(j,i),j=1,4)
     ELSE
        READ(lu0,*) tau(i)
     ENDIF
  ENDDO
CLOSE(lu0)
  tbeg = tau(2)
  tend = tau(N-1)
  niter = 6

  c This is the main iteration loop for the non-linear
  c least-squares -- in each loop, bsc0 is improved to bsc1:
  c
  DO iter = 1, niter
  c------------------------
  c STEP 1 -
  c------------------------
  c Initialize normal equations etc:
  c
  mode = 0
              CALL attupd(N, tau, bsc0, mode, glb, apar, tobs, fobs, ferr,
             + wa, wb, wc, wd, bsc1, lerr, wr)

  c Loop over the observations, accepting those in [tbeg,tend]:
  c
  OPEN(lu0,file=file_obs)
  nobs = 0
  CONTINUE
  READ(lu0,*,END=200) tobs, id, ia, (fobs(i),ferr(i),i=1,2)

  6
IF (tobs .LT. tbeg) GOTO 100
IF (tobs .GT. tend) GOTO 200
CALL getstar(id, apar)

------------------
c STEP 2 -
------------------
c Update normal equations:
c
mode = 2
CALL attpd(N, tau, bsc0, mode, glb, apar, tobs, fobs, ferr,
+ wa, wb, wc, wd, bsc1, ierr, wr)
IF (ierr .NE. 0) print *, 'ierr,tobs=', ierr, tobs
nobs = nobs + 1
GOTO 100

c No more observations (or tobs exceeds tend):
c
200 CONTINUE
CLOSE(1n0)
------------------
c STEP 3 -
------------------
c Solve normal equations and update spline coefficients:
c
mode = 3
CALL attpd(N, tau, bsc0, mode, glb, apar, tobs, fobs, ferr,
+ wa, wb, wc, wd, bsc1, ierr, wr)
WHITE(*,('(x,a,i10)') 'iteration = ', iter
WHITE(*,('(x,a,i10)') 'attpd ierr = ', ierr
WHITE(*,('(x,a,2110)') 'nobs = ', nobs, nobs
WHITE(*,('(x,a,2f10.6)') 'mean weight reduction = ', wr

c Calculate the RMS update and (in preparation for the
c next iteration) copy bsc1 to bsc0:
c
rms = 0.0
DO i = 1, N
   rms = rms + (bsc1(i,1) - bsc0(i,1))**2
   bsc0(i,1) = bsc1(i,1)
ENDDO
rms = dsgpt(rms/dble(N+4))
WHITE(*,('(x,a,g12.6)') 'RMS update (uas) = ', rms/uas

ENDO
------------------
c OUTPUT -
------------------
c Output updated spline data to file_out:
c
OPEN(file=file_out)
DO i = -1, M+2
   IF (i .GE. 1 .AND. i .LE. M) THEN
      WRITE(lu1, '(f15.9,4f20.16)') tau(i), (bscl[j,i],j=1,4)
   ELSE
      WRITE(lu1, '(f15.9,4f20.16)') tau(i), (0d0, j=1,4)
   END IF
ENDDO
CLOSE(11)
WRITE(*,('(x,a,a)') 'spline data written to: ', file_out('1:20)
END

Executing fit_attitude generates the following standard output (out_fit_attitude),
where the convergence of the five iterations can be seen. Between the last two iterations
the RMS update of the spline coefficients $S_{mn}$ is below 0.02 $\mu$as.
iteration - 1
attupd ierr - 0
nobs - 4253 4253
mean weight reduction - .116187 .997737
RMS update (us) - 3100.21

iteration - 2
attupd ierr - 0
nobs - 4253 4253
mean weight reduction - .998937 .999168
RMS update (us) - 416.262

iteration - 3
attupd ierr - 0
nobs - 4253 4253
mean weight reduction - .998948 .999170
RMS update (us) - 3.86799

iteration - 4
attupd ierr - 0
nobs - 4253 4253
mean weight reduction - .998948 .999170
RMS update (us) - 3.173265-01

spline data written to: qdata1.prn

The first five lines of the output data file qdata1.prn are shown below. The meaning of the columns is the same as for the starting approximation approximation.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$S_{1n}$</th>
<th>$S_{2n}$</th>
<th>$S_{3n}$</th>
<th>$S_{4n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.00861110</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
</tr>
<tr>
<td>.009074070</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
<td>.0000000000000000</td>
</tr>
<tr>
<td>.009637040</td>
<td>.120774323604927</td>
<td>-753864181196079</td>
<td>.341622804870468</td>
<td>-.548426210331236</td>
</tr>
<tr>
<td>.010000000</td>
<td>.1119943890276697</td>
<td>-75408943499448236</td>
<td>.3352361624729914</td>
<td>-.5523738710087985</td>
</tr>
<tr>
<td>.010462960</td>
<td>.1031996526570365</td>
<td>-756230940480843</td>
<td>.3287842226712601</td>
<td>-.5562469662737534</td>
</tr>
</tbody>
</table>

3.5 Fitting the attitude spline (distributed processing)

In attupd the same subroutine is used for steps 1, 2 and 3, as controlled by the argument mode. In order to implement the scheme in Figure 2, the routine is broken up into three separate routines called attupd ini (initialize), attupd acc (accumulate) and attupd fin (finalize). These are contained in the file attupd.dis.f.

It would be possible to run these routines from a single main program (as for attupd), but in order to bring out more clearly the modularity and the interaction between the different modules, they are here actually run by separate main programs called att.ini, att.acc and att.fin. Standard input is used to control e.g. the processor number and data are exchanged between them by means of temporary binary files (*.uf). This structure in principle allows the attitude fitting to be run on a cluster without special parallelization code, although it would not be very efficient due to the use of disk files for data exchange.
Here the NPROC = 4 processes are actually run in sequence rather than in parallel, using the following shell script (att.bat):

```bash
#!/bin/csh
#
# unix batch program to illustrate distributed processing
# of attitude determination.
#
# NITER iterations are made, each consisting of three steps:
# 1. initialize data (attIni) - only needed for first iteration
# 2. accumulate normals (attAcc)
# 3. finalize and solve normals (attFin)
# After the last iteration, results are written to a text file.
#
# The second step is divided into NPROC separate processes,
# here run sequentially, but they could also run in parallel.
# Step 3 has to wait until all NPROC processes are ready.
#
# define number of iterations and number of distributed processes:
set NITER = 5
set NPROC = 4

# Step 1 (initialize):
echo $NPROC $attIni

# this is the iteration loop:
set ITER = 1
while ($ITER <$> $NITER )

# Step 2 (here written explicitly for IPROC = 1, NPROC):
  echo 1 $attAcc
  echo 2 $attAcc
  echo 3 $attAcc
  echo 4 $attAcc

# Step 3:
  echo $ITER $NPROC $attFin
  @ ITER++
end

# convert final result (qdata.uf) to text file (qdata.prn)
# for easier inspection:
print qdata

# cleanup:
rm work*.uf obs*.uf qdata.uf
```

Executing att.bat generates standard output (out.att.bat) which is identical to the output from fit_attitude except for the name of the output data file. The output data file (qdata.prn) is also virtually identical to the previous one (qdata1.prn). This merely shows that the distribution of steps 1 and 2 is mathematically (if not quite numerically) equivalent to the original algorithm. If there is some slight numerical difference between qdata.prn and qdata1.prn it will be caused by rounding errors in the accumulation of the normal equations.
4 Files distributed with this document

The files available with this document are as follows:

--- program units for non-distributed processing (Sect. 3.4):
fit_attitude.f
out_fit_attitude
qdata1.prm

--- program units for distributed processing (Sect. 3.5):
att_acc.f main program to accumulate normals
att_bat batch program to run att_ini, att_acc, att_fin
att_fin.f main program to finalize attitude updating
att_ini.f main program to initialize attitude updating
attupd_dis.f subroutines for attitude updating
out_att_bat standard output from att_bat
print_qdata.f convert binary output data to printable
qdata.prm output data file

--- general routines:
const_astr.h include file for astronomical constants
const_math.h include file for mathematical constants
getstar.f return source parameters (Sect. 3.1)
LL34.f subroutine package (see below)
makefile makefile for compiling all programs
obs.prm input observational data (Sect. 3.2)
q_initial.prm initial spline approximation (Sect. 3.3)

Remark: the subroutine package LL34.f (v. 2.1) used here is in principle the same as
distributed with GAIA-LL-34.2, except that some unnecessary print statements — used
for debugging and then left there by mistake — have been removed.