

User guide to batch Solve

Karen D. Bayer

Wang Guangli

Leonid Petrov

Abstract:

This user's guide explains the use of the batch subsystem of Solve, the Mark IV VLBI Analysis System. It assumes that the reader is familiar with VLBI analysis using Solve's interactive subsystem. Readers should also be familiar with the technique of multigroup LSQ (or arc parameter elimination).

Table of contents:

- 1 [Overview](#)
 - 1.1 [Types of solutions](#)
 - 1.2 [Modes of solutions](#)
 - 1.3 [Combined Global Matrix \(CGM\)](#)
 - 1.4 [Arc files](#)
- 2 [Instructions for operating BATCH](#)
 - 2.1 [Preparations for the BATCH Run](#)
 - 2.2 [Setting up the BATCH control file](#)
 - 2.3 [Setting up data](#)
- 3 [Running BATCH](#)
 - 3.1 [Starting a run](#)
 - 3.2 [Monitoring a run](#)
 - 3.3 [Killing runs](#)
 - 3.4 [Cleaning up after aborted runs](#)
 - 3.5 [Restarting runs](#)

- 4 [BATCH'S output](#)
 - 4.1 [Spool file](#)
 - 4.2 [Warnings](#)
 - 4.3 [Output Section](#)
 - 4.4 [Global parameter output](#)
 - 4.5 [Local parameter output](#)
 - 4.6 [Appended files and CGM summary](#)
 - 4.7 [Progress file](#)
 - 4.8 [Sarfil](#)
 - 4.9 [Earth orientation plot file](#)
 - 4.10 [Covariance file](#)
 - 4.11 [Correlations file](#)
- 5 [Acknowledgments](#)
- 6 [References](#)
- 7 [Appendix](#)
 - 7.1 [BATCH's file directories](#)

1 Overview

Interactive Solve performs analysis of an individual session. The main purpose of batch Solve (referred thereafter to as BATCH) is to perform VLBI analysis of more than one session. The data are divided into sessions (referred also to as databases, or arcs), and processed sequentially. A session is typically 24 hours of data from a coordinated network of stations. Interactive

Solve is used for quality control, ambiguity resolution, editing and setting specific parameterization. BATCH is used for getting final results. It uses control files for detailed specification of the solution. BATCH takes care to set exactly the same a priori, calibration and contributions for all sessions which participate in the solution. BATCH performs its analysis using weighted multigroup LSQ (arc parameter elimination). It guarantees that all sessions of the solution are treated exactly the same way: the same a priori, calibration, contributions applied and the same style of parameterization. BATCH uses a built-in set of partials derivatives, calibrations, contributions supplied by CALC but it supports also a so-called user-mode which allow the use of specific user-written programs for computation non-standard partials, contributions and equations of constraints. This feature makes Solve a powerful tool for scientific researches.

1.1 Types of solutions

BATCH supports five solution types, INDEPENDENT, COMPLETE, FORWARD, BACK, SUPPRESSION, which fall into two solution categories: independent and global.

BATCH processes a series of sessions independently -- that is, the estimated parameters in one session are adjusted independently from parameters of another session if the INDEPENDENT solution type was requested. The results for a given session are identical to the results of running Solve interactively.

BATCH combines information from multiple sessions to estimate parameters when the solution of global category is requested. The parameters are divided into two classes: local (arc) and global. Local parameters are those which are adjusted using the observations of only one session. Each session has its own set of local parameters. Global parameters are those which are adjusted using all observations in all sessions. Some parameters can be either local or global in the same solution, depending on the purpose of the run.

BATCH produces the solutions in two steps. The first or forward step performs

matrix decomposition. The second or back step a) inverts the global parameters' matrix and b) uses it to estimate the global parameters, c) then it makes a backward substitution and compute for each session the local parameters as well as their formal uncertainties. This step also produces the solution residual statistics.

The straitforward way of making global solutions is to perform consequetely both steps. BATCH does it if solution type COMPLETE is requested. It is recommended way of making global solutions.

However, BATCH allows to specify one of incomplete global solutions types: FORWARD, GLOBAL_ONLY, BACK or SUPPRESSION. Solve provides tools for manipulations with incomplete solutions.

If the FORWARD solution type was requested then BATCH stops after completion of decomposition.

If the GLOBAL_ONLY solution type was requested then BATCH stops after inversion of CGM, obtaining global parameters, their covariance matrix and formal uncertainties, computation some statistics.

If SUPPRESSION solution type was requested then BATCH removes suppressed global parameters from a global parameter matrix. It is assumed that FORWARD run has been already done.

If BACK solution type was requested then BATCH inverts combined normal matrix of global parameters obtained and saved earler when user made the forward step. Then BATCH performs the back step of the global solution.

However, making global solution by several BATCH runs is a dangerous operation and should be done with great care. Making the global solution in two steps has sence when the control file in back and/or suppression runs differ from the control file for making forward runs. In one cases it is legitimite, in another cases not. The user should clearly understand what he/she is doing since there is a risk of getting wrong results.

BATCH distinguishes between regular (permanent) and test (temporary) forward

and complete solutions. BATCH writes these solutions' global parameter matrices to directories and catalogs which are, stable for regular solutions and frequently cleared for temporary solutions. The ID keyword in the BATCH control file's \$SETUP section determines whether a solution is a test or regular solution. The user can specify filename with path or without path if he/she need to keep the CGM. If a user is not going to keep the CGM then he/she can omit the name of the output CGM in the BATCH control file. Solve will generate the filename itself and write the output CGM on disk after processing the last experiment. The name of the directory is

a) if ID has the first 4 letters "test" then

 SCRATCH_DIR

else

 if an environment variable CGM_DIR is specified then

 b) \$CGM_DIR

 else

 c) CGM_DIR variable is defined in ../include/gsfcb.i

 endif

endif

1.2 Modes of solutions

Independent and global solutions may use slightly different algorithms for solving LSQ problems. The keyword FAST_MODE controls which algorithm is to be used.

The INDEPENDENT solution may use the fast B3D algorithm (preferable) if the user specified FAST_MODE B3D and it will use the slow algorithm if the user specified FAST_MODE NONE in his/her control file.

COMPLETE, FORWARD, GLOBAL_ONLY and BACK solution will use the fast B1B3D algorithm (preferable) if the user specified FAST_MODE B1B3D and it will use the slow B1D algorithm if the user specified FAST_MODE NONE.

Fast algorithms process the solution 2-20 times faster. Fast and non-fast algorithms are equivalent but results are slightly different due to the influence of rounding errors which are not the same in both algorithms.

In addition, the user can select TRAIN on NO TRAIN mode by specifying TRAIN YES or TRAIN NO (preferable) in the \$SETUP section of the control file.

BATCH calls a chain ("train") of executables when it is used in TRAIN mode. It has to read/write an intermediary CGM and other files extremely intensively. NB: TRAIN YES becomes an obsolete feature and may not be supported in the future.

All work is done by a single executable if BATCH is called in TRAIN NO mode. The user should specify the expected number of global parameters and an increment step in this mode. BATCH allocates dynamic memory at the beginning of the run in NO TRAIN mode which is computed on the basis of the expected number of global parameters which is unknown at the very beginning. If the actual number of global parameters would exceed that number then BATCH increases the number of expected parameters by the specified increment step and increases amount of allocated memory. The recommended value of the increment step is 128.

1.3 Combined Global Matrix (CGM)

The forward step performs matrix decomposition and saves accumulated information which the back steps later uses to estimate parameters. To allow multiple back steps to use the information, BATCH stores it in files called combined global matrix (CGM). The combined global matrix (CGM) is an intermediary result of matrix decomposition. It stores the information needed to estimate the global parameters and their covariances.

Forward steps always produce a CGM, either from the current solution alone

or by incorporating an input CGM as well. BATCH writes the intermediary CGM after processing each N session in the file \$WORK_DIR/CGMFxx and in principle lets the user use the partially completed CGM to estimate parameters from the information collected so far and to resume the solution from that point if it was interrupted (this trick is not recommended for ordinary runs).

The frequency of writing CGM is determined by the the user in a control file in the keyword SAVING_RATE. The size of the CGM may be tens of megabytes and frequent writing may slow down overall performance considerably. It is recommend that SAVING_RATE be set large value, e.g., 100, unless there are specific reasons to use a smaller value.

Back solutions always require an input CGM. (A complete solution's back step automatically uses the CGM just produced by its forward step.)

The following table summarizes the five solution types' usage of CGMs:

Solution Type	Input CGM	Output CGM
INDEPENDENT	no	no
COMPLETE	optional (1)	optional (2)
FORWARD	optional (1)	optional
GLOBAL_ONLY	optional (1)	optional (2)
BACK	yes	no
SUPPRESSION	yes	yes (3)

(1) - if used, the output CGM contains the input CGM plus information from the solution. Otherwise, the output CGM is based entirely on the solution.

(2) - BATCH completes the output CGM by the end of the forward step

and uses it for the back step.

- (3) - the output CGM is the input CGM minus suppressed global parameters.

1.4 Arc files

BATCH saves some matrices for each session which are produced during matrix decomposition when it performs a global solution. These matrices are used in the back step. BATCH may write or not to write them in so called arc files in the forward step. BATCH looks for the arc file in back step. If it finds an arc file related to the processed sessions it uses them, otherwise it repeats computations which have been done in forward step and creates them anew. When a back step uses arc files, it purges them, unless it is told to save arc files.

Saving arc files reduces execution time substantially, but the system should have enough space to accommodate them. Arc files are named [####xx, where xx are the Solve user initials of the analyst who runs Solve, and #### gives the session's position in that solution's list of sessions. Without any indication of which arc file contains which session, a back solution assumes that the arc files it needs are labeled with its initials and numbered in the order the sessions appeared in the solution. So the user must run every back solution he/she intends to run from one forward solution before running a new forward solution. Otherwise, the new solution will overwrite his/her arc files in a new order, and the remaining back solutions will access the wrong arcs.

On the other hand, a COMPLETE solution's back step automatically runs before any new forward step is run with its initials, so a complete solution's back step can safely use the arc files produced by its forward step. If the forward step fails to make the full set of arc files (for example, because it ran out of space), its back step automatically generates the missing information.

Suppression and independent solutions do not deal with arc files. Suppression solutions do not deal with sessions, and independent solutions process the sessions independently, so they do not need to save information related to the

specific experiments.

2 Instructions for operating BATCH

2.1 Preparations for the BATCH Run

BATCH gets its instructions from a control file. This file tells BATCH what data to use, what parameters to estimate and so on. Each solution requires a new control file tailored to that solution, and it is the user's responsibility to create it. See the description of BATOPT language for details. The control file should be written carefully. It is strongly recommended that all keywords be specified and that obsolete keywords or defaults not to be used. In practice the control file is rarely written anew: a user usually has several his/her favorite "master files" used as a template. He/she copies a master file to a new file then edits it.

Solve's interactive mode can get its data from databases and superfiles, but BATCH only uses superfiles.

2.2 Setting up the BATCH control file

Control files are divided into several sections, each determining how BATCH handles a different aspect of the solution (the data used, parameters estimated, etc.).

At times this part of the user's guide gives examples. These examples are only intended to illustrate syntax. Readers should not interpret them as recommendations for setting up their control files.

2.3 Setting up data

BATCH gets its data from "superfiles". Superfile is a name of a VLBI data format, which differs from a database format. Superfile doesn't have all the information kept in databases but only a subset needed for Solve. Each superfile has the same experiment name and version number as the session to which it corresponds. Superfiles are created from sessions by running program liptn.

The list of all available superfiles is in the file \$SAVE_DIR/SUPCAT .

To make superfiles, the user must run liptn to generate them from the corresponding sessions. Refer to the liptn documentation for details.

The user does not have to worry about where liptn puts the superfiles. liptn records their locations in the superfile catalog, and BATCH looks them up automatically.

3 Running BATCH

3.1 Starting a run

Solve guide 2

To start BATCH execute the command

```
solve <solve_initials> [<control_file>] [verbosity_level]  
[processor_index] [number_of_processors]
```

Here

<solve_initials> - two initials (XX) identifying the run and its files (SPLFXX on SPOOL_DIR, PRGFXX on Solve_WORK_DIR, etc.). Solve initials identifies temporary files which belong to different "solve users".

NB: A user may have more than one solve_initials. The set of solve initials which Solve recognizes is defined in the file \$SAVE_DIR/letok .

<control_file> - the file name containing the instructions for the BATCH run. File name after expansion should not have more than 128 characters.

The third argument can take values:

"verbose" -- (Default) Solve prints some information messages.

"silent" -- Solve doesn't prints information messages.

"silent-autorestart" -- The same as "verbose". If restart of the interrupted solution is possible, it will be done automatically without dialogue with a user.

"silent-autorestart" -- The same as "silent". If restart of the interrupted solution is possible, it will be done automatically without dialogue with a user.

"verbose-norestart" -- The same as "verbose". Solution will start a new regardless, whether restart is possible or not.

"silent-norestart" -- The same as "solent". Solution will start a new regardless, whether restart is possible or not.

The fourth and fifth optional arguments enable a rudimental multi-processor support: Solve decimates the experiment list and

processes only a part of experiments.

[processor_index] -- index of the processor used for the batch
Solve run in the independent mode. Should be
in the range [0, number_of_processors].
if set, Solve processes not all sessions in
the list but only one out of N, where N is
[number_of_processors]: mod(i,N), where "i" is
the session index.

[number_of_processors] -- The total number of processors which
Solve knows of.

3.2 Monitoring a run

Since Solve run may take long time it is impossible to resist a temptation
to learn what is going on. Program SMON is used for monitoring the global run.
Usage:

Usage: SMON <solve_initials> <work_dir> [<interval_update>]

where

<solve_initials> is the two character of the Solve user initials.

<work_dir> -- full path to the directory where solve scratch
file are located (\$WORK_DIR)

[<interval_update>] -- optional argument. Specifies time
interval in seconds for updating screen
output. Default is 1 second.

Example of the output:

```
picasso /disk4/vlbi/petrov/tests/job59
PT Solve: FORW 2(4) $88NOV09X <14> started (F) 0:02:50
```

Solve guide 2

Here:

picasso -- hostname where Solve is running.
/disk4/.. -- full name of the control file being executed.
PT -- Solve initials.
FORW -- program name which now is being executed.
2 -- index of the session in the arc-list which is being analyzed.
(4) -- total number of the sessions to be analyzed.
\$88NOV09X -- session name which is now being processed.
<14> -- version number of the session which is now being processed.
started -- status of the processing: one of
started
done
(F) -- mode:
(F) -- forward run in fast mode;
(B) -- backward run in fast mode;
(I) -- independent run in fast mode;
(f) -- forward run in non-fast mode;
(b) -- backward run in non-fast mode;
(i) -- independent run in non-fast mode;
0:02:50 -- amount of time elapsed since the last status update. A large amount of time may indicate that Solve is not active now because it terminated.
(Comment: the first line may not shown in the beginning of the batch run before parsing control file)

Another way to peek at the progress of the solution is to look at the progress file. BATCH records its progress in the progress file, PRGFXX on \$WORK_DIR. Each time BATCH completes processing a session it writes a line identifying the experiment name. By typing

```
tail $WORK_DIR/PRGFXX
```

the user can dump the last few sessions processed to find out how far he/she has gotten. However, in a complete solution, each sessions is processed twice,

once in each step, so the user must actually edit the progress file to check BATCH's progress.

3.3 Killing runs

Sometimes a user will want to start his/her run over or make some correction, then resume where he/she left off (recover). In each case, he/she must kill the enter program and any programs it called.

Sometimes these programs can be killed cleanly, with one command. UNIX considers the programs part of a single job. Users who run UNIX under the C or Korn shells can see which jobs they are running by typing `jobs`. This displays three pieces of information per job: the number of the job, its status and the run string which generated the job. For example, typing `jobs` might generate:

```
[1] -running   enter kb gkb004
[2] +stopped   enter kd gkb005
```

To kill a job, the user should type

```
kill -9 %n
```

Typing

```
ps -fu <login_id>
```

tells the user which programs he/she is currently executing. (Login-id is the user name used to log onto the UNIX computer.) If the user is running two or more BATCH runs, he/she will be running multiple copies of enter and so forth. In this case, he/she must pay special attention to `ps` to determine which copies to kill. He/she can identify the right copies by looking for the two characters that identify his/her run.

For example, typing

```
ps -fu kdb
```

might give the following output:

```

UID  PID  PPID  C   STIME TTY    TIME COMMAND
kdb 15559 13690  6 18:46:49 ttys5 0:00 ps -fu kdb
kdb 13690 13689  1 17:04:40 ttys5 0:01 -csh [csh]
kdb 15513 13690  0 18:45:18 ttys5 0:00 enter KB gkb004
kdb 15518 13690  0 18:45:25 ttys5 0:00 enter KD gkb005
kdb 15514 15513  0 18:45:18 ttys5 0:00 /mk3/bin/BATCH 0 4 0 0 KB
kdb 15519 15518  0 18:45:25 ttys5 0:00 /mk3/bin/BATCH 0 4 0 0 KD
kdb 15515 15514  0 18:45:18 ttys5 0:00 /mk3/bin/GTSUP 0 6 0 0 KB

```

In this example, the user is running two BATCH runs, one with initials KB (line 3) and one with initials KD (line 4). Suppose he/she wants to kill the KB run. Looking at the last column, the COMMAND column, and comparing the programs mentioned there to the above list of BATCH programs, there are five BATCH programs running -- two copies of enter, two of BATCH and one of GTSUP. Three of these belong to the KB run--the programs on lines 3, 5 and 7. These three must be killed in order to kill off the run.

To kill a program, the user should type

```
kill -9 <PID>
```

where PID is the number in column 2, the PID column. This should not be confused with column 3, the ppid number. In the above example, typing `kill -9 15515` kills GTSUP. One problem with killing the run program by program is that if the programs are killed in the wrong order, the unkillable programs may keep calling new programs, which in turn must be killed. Users should kill BATCH first, then the programs other than enter or enter, then enter. Even so, users should do a final `ps` afterwards to make sure every program has been killed.

Some programs may be impossible to kill with the kill command. These are called defunct programs. They can only be killed by rebooting the computer. This particular problem is still being studied, but in the meantime the user can ignore the program.

3.4 Cleaning up after aborted runs

Usually, when a background BATCH run aborts, it kills all of its programs. However, sometimes the run may not be able to clean up, due to uncontrollable events, such as system problems. (For example, sometimes runs have aborted because too many other runs were running.) So, when a background run aborts, the user should run `ps -fu` as a precaution, and use the kill command to kill any leftover programs.

3.5 Restarting runs

Once the user cleans up his/her killed/aborted run, he/she can resume processing where he/she stopped (recover), restart from the beginning or abandon the solution entirely. If the run aborted, the user can check his/her progress and error files (PRGFXX and ERRFXX on \$WORK_DIR) to find out where and why.

To restart BATCH, either for recovering or starting over, the user should retype his/her enter run string. If recovering is not possible then Solve starts solution anew. If recovering is possible then Solve asks the user whether he/she wants to recover.

Restarting is possible if

- 1) the user specifies the same control file as was in use in the interrupted solution,
- 2) the solution has not been completed,
- 3) BATCH saved the intermediary data at least once (i.e BATCH in forward step processed more sessions then the number specified SAVING_RATE or BATCH processed at least one session in backward mode),
- 4) the length of the control file (and arc-file if used) didn't change.

If all four conditions are true then Solve asks whether the user wants to recover, start over or cancel this attempt to run BATCH.

If at least one condition 1-3 is false Solve starts a new solution from the beginning without warning. If only condition 4 was violated then BATCH warns the user that recovering is not possible and asks a confirmation to start a solution anew.

Occasionally, when the user tries to restart, BATCH may claim that the run's initials are still in use. This indicates that killing/cleaning up the previous run failed to close the lock file (LOCKXX, on \$WORK_DIR, where XX are the run's initials.) Runs open their lock files with exclusive access, to prevent the running of multiple runs with the same initials. The user should make sure that every program from the previous run is dead. If so, then the user should run unlock, to release his/her run's lockfile. Program Unlock is located on \$SOLVE_DIR directory.

4 BATCH'S output

BATCH runs produce different combinations of output, depending on how the user sets up his/her BATCH control file. The main output of Solve is a spool file. The spool file for the global solution may have up to one million lines, so it is unlikely that somebody could print and read it. Finally, BATCH produces four additional files for further analysis: the progress file, the sarfil, the earth orientation plot file and the covariance or correlations

file.

4.1 Spool file

The spool file is SPLFXX on \$SPOOL_DIR, where XX are the two characters which identify the run. The spool file collects statistics, estimates and other information about the solution's global and local parameters. In addition, back, complete and suppression solutions append the solution's BATCH control file and progress file to the spool file, to gather a complete record of the run in one place.

BATCH produces a lot of information for each experiment. This amount is manageable for interactive solutions, which handle single sessions and were the original motivation for creating the spool file. However, many BATCH solutions include several thousands sessions, and their spool files are too large to be easily read. Several programs -- getpar, snoop, gsnoop and msnoop -- have been developed to extract selected information from spool files. A discussion of these programs is beyond the scope of this guide. Look at documentation about getpar and gsnoop.

The spool file consists of the following sections and subsections:

1. Warnings
2. Output section
 - a. Global parameter output
 - b. Local parameters output(These two types of output are produced in various combinations, as explained below)
3. Appended files and CGM summary
 - a. Copy of the run's BATCH control file
 - b. Copy of the run's progress file
 - c. CGM summary (list of names of the input CGM files)

4.2 Warnings

The first section of the spool file receives a warning message for every experiment which had bad temperature, pressure or relative humidity data, for which standard values had to be substituted or some other problems were disclosed.

4.3 Output Section

BATCH produces the global and local parameters output in various combinations, depending on the solution type and the FORWARD keyword in the BATCH control file's \$OUTPUT section. Complete and back solutions produce the output for the global parameters and then the output for the local parameters. The \$OUTPUT section's FORWARD keyword determines what output BATCH produces in a forward solution or a complete solution's forward step. Independent solutions just produce output for local parameters, since independent solutions do not work with global parameters. Suppression solutions do not work with sessions, so they do not produce parameter output.

4.4 Global parameter output

ADJST output

Global station parameters:
for each station:

Solve guide 2

- XYZ and UEN position and velocity
 - component information
- Velocity vector information
- Error ellipsoid information
- Correlations
- Station table
- Global source parameters
- Remaining global parameters

BASFE output

- Baseline component information:
 - vector magnitude, length, transverse and vertical, and sigmas
- Rates of change of components:
 - same information

First BATCH's ADJUST program produces information for any stations with globally estimated station parameters. ADJUST loops over the stations, producing all of a station's information at once.

For each station, ADJUST first prints a line for each of its XYZ and UEN position and velocity components. Each line contains the appropriate total, estimate, unscaled sigma and scaled sigma.

Next ADJUST combines the station's velocity components to print information about various velocity vectors and sigmas. First ADJUST prints totals for the velocity vector (as azimuth and elevation), the horizontal vector (as azimuth) and the horizontal vector's sigma. Then ADJUST prints the corresponding estimates.

Next ADJUST prints the station's error ellipsoid information: the azimuth, elevation and sigma for the error ellipsoid axis and the horizontal error ellipse axis.

Next, if the user has specified MINIMUM NO in his/her control file's \$OUTPUT section, ADJUST prints the correlations between the station's X, Y and Z position and velocity components. ADJUST prints the correlations for every pair

of these components.

Finally, if the user has specified STATION_TABLE YES in the \$OUTPUT section, ADJUST prints a table projecting the station's X, Y and Z position components at the beginning of each year from 1979 to 1992. The table takes two forms, depending on what the user specified for the \$OUTPUT section's MINIMUM keyword. MINIMUM YES prints one line per year, giving that year's projected X, Y and Z position totals and unscaled sigmas. MINIMUM NO prints four lines per year. The first line gives the six projected correlations between the X, Y and Z positions, and between each component's position and velocity. The remaining lines give each component's projected position total, estimate, unscaled sigma and scaled sigma.

ADJUST then produces totals, estimates and sigmas for any remaining global parameters, such as source coordinates. Each parameter's information is much the same as the information printed in Solve's interactive mode.

Finally, BATCH's BASFE program produces its part of the global output, if the BATCH control file's \$OUTPUT section specifies BASELINES YES. BASFE produces up to two sets of information -- information about the baseline components and information about their rates of change. BASFE produces the component information if the user estimates at least one station position globally. BASFE produces the rate of change information if the user also estimates station velocities.

If BASFE prints one of these types of information, it prints it for every pair of stations in the solution, even if no observations were actually made along that baseline. For each baseline and type of information (component or rate of change), BASFE prints the baseline vector's length, transverse and vertical components, the vector's magnitude, and each value's sigma. The sigmas are scaled by the combined weighted RMS delays and rates. BASFE orders the baselines based on the order of the stations' appearances in the solution.

4.5 Local parameter output

CRES output

- Calibrations
- Elevation cut offs
- General statistics
- Baseline statistics
- Source statistics

ADJUST output

- Flyby files
- local parameter totals, estimates, sigmas
- statistics
- Error ellipsoids
- Correlations

BASFE output

- Baseline component information
- Information for rates of change of components

BATCH's CRES program first lists the calibrations added to the theoreticals. CRES only produces the list if the BATCH control file's \$OUTPUT section specifies MINIMUM NO. Next CRES lists any elevation cut offs used in the session. Then CRES lists the general statistics over observations of the experiment. These include the number of observations used in the solution and the solution's fit. Finally, if MINIMUM NO is specified, CRES produces statistics for the local baselines and sources.

ADJUST first lists any flyby mapping files it used in the solution. These are the files selected through the BATCH control file's \$MAPPING section. Then ADJUST produces estimates and unscaled and scaled sigmas for all of the local parameters of the individual session. ADJUST also produces totals, constraint statistics and error ellipsoids for some of these parameters. ADJUST also produces correlations between the earth orientation and nutation parameters, but only if MINIMUM NO is specified, and only for certain earth orientation parameterization styles.

Finally, if the BATCH control file specifies BASELINES YES, BASFE produces

information about the local baselines. For every baseline containing at least one station whose positions are local parameters, BASFE produces that baseline's length, transverse and vertical components, the magnitude of the baseline vector and these values' sigmas.

4.6 Appended files and CGM summary

After the output section, BATCH appends several files:

- 1) Episodic motion control file,
- 2) Piece-wise station position file,
- 3) High-frequency eop control file,
- 4) Pressure loading control file,
- 5) Station position mod file,
- 6) Station velocity mod file,
- 7) Source position mod file,
- 8) Earth Orientation mod file,
- 9) Axis Offset mod file,
- 10) Program versions list,
- 11) solution's control file,
- 12) progress file with which is described below,
- 13) CGM summary (list of names of the input CGMs)

Files 1-9 are appended if a) they were used; b) MINIMUM NO was specified in the \$OUTPUT section of the control file.

4.7 Progress file

The progress file is PRGFXX on \$WORK_DIR, where XX are the two characters which identify the run. This file serves two purposes. First, BATCH writes a line to it each time it finishes processing a session, unless the user is running a suppression solution, which processes CGMs, not sessions. The session by session output lets the user check his/her progress file during the run to estimate when the run will finish. The output also makes it easy to identify experiments where fatal errors occur. Each line states how long it took BATCH to process the experiment and the time BATCH finished. At the end of the solution, BATCH also writes some overall solution statistics to the progress file, if the solution is a suppression, complete or back solution. Specifically, BATCH writes the fit statistics, the number of local (arc) and total parameters per session, and summaries of the baseline and source statistics, each for the entire solution.

Example:

```
TEST BY_SITE indp                                000515.232717
INDEPENDENT SOLUTION START                        000515.232815
  1 $79AUG03XX 25 ARC TIME:  0: 0: 6 cond# = .666E+07 000515.232821
  2 $80APR11XQ 46 ARC TIME:  0: 0: 6 cond# = .706E+07 000515.232827
```

The last column contains time of termination of session processing in UTC timescale. The previous column contains condition number of the matrix of local parameters of the session. (Condition number is zero when the session is processed in non-fast mode in global solution, since no information is collected). ARC_TIME means the physical time elapsed for processing that session.

NB: Qualifier FAST_DBG TIME in the \$SETUP section provides rather more detailed information about CPU and elapsed time taken by different parts of Solve for processing each session.

4.8 Sarfil

The sarfil (SARFXX on \$WORK_DIR) collects part of the spool file in a form that ARG, the automatic report generator, can use. However, ARG and the sarfil are gradually being replaced by report programs such as getpar, snoop and gsnoop, which pull data directly from the spool file. currently (2000.05.16) Sarfil is used by BATCH to produce overall statistics of the global run. Sarfil has binary format.

4.9 Earth orientation plot file

The earth orientation plot file, EOPLXX on \$WORK_DIR, collects the Earth orientation estimates and sigmas, approximately once per hour, for later plotting. (The user currently has to generate his/her own plotting control file.) The UT1/PM keyword in the BATCH control file's \$FLAGS section tells BATCH whether or not to generate this file.

4.10 Covariance file

The covariance file, CVRFxx on \$WORK_DIR, collects matrices of covariance data. The COVARIANCES keyword in the BATCH control file's \$OUTPUT section tells BATCH whether or not to generate this file.

NB: COVARIANCES is considered an obsolete and unsupported feature. Use CORRELATIONS instead.

4.11 Correlations file

Solve guide 2

The correlations file, CRLFXX on \$WORK_DIR, collects correlations coefficients between the estimates of the parameters that were specified. The CORRELATIONS keyword in the BATCH control file's \$OUTPUT section tells BATCH which correlations are to be computed and printed.

```
# Center:      BON ( Bonn Geodetic VLBI group )
# Analyst:     Leonid Petrov ( xxx )
# Machine:     picasso 9000/712 HP-UX B.10.20
# Executables: ./
# Solve initials: PE
# Solution ID: test_job1 (test of correlations)
# Control file: /disk4/vlbi/petrov/tests/job1
# Local time:  1999.10.11-10:07:05
#
# Type:        GLO_GLO Correlations
#
*
* This line is comment.
*
4  5 "GILCREEK X COMPONENT" "GILCREEK Y COMPONENT" -0.556215603
4  6 "GILCREEK X COMPONENT" "GILCREEK Z COMPONENT"  0.432941964
4  7 "GILCREEK X COMPONENT" "HARTRAO X COMPONENT" -0.217073754
4  8 "GILCREEK X COMPONENT" "HARTRAO Y COMPONENT" -0.549527974
4  9 "GILCREEK X COMPONENT" "HARTRAO Z COMPONENT"  0.639315466
4 10 "GILCREEK X COMPONENT" "HOBART26 X COMPONENT" -0.695608370
#
# Type:        GLO_LOC Correlations
# Database:     $99MAR15XH <5>
# Start_date:   2451253.08330000 1999.03.15-13:59:57.120
# Stop_date:    2451254.08120000 1999.03.16-13:56:55.679
#
1 389 "FORTLEZA X COMPONENT" "X WOBBLE 09903151400" -0.117249278
1 390 "FORTLEZA X COMPONENT" "Y WOBBLE 09903151400" -0.080747136
1 393 "FORTLEZA X COMPONENT" "LONGITUDE NUTATION "  0.021860446
```

Solve guide 2

```
1 394 "FORTLEZA X COMPONENT" "OBLIQUITY NUTATION " 0.033797252
2 389 "FORTLEZA Y COMPONENT" "X WOBBLE 09903151400" -0.136677586
2 390 "FORTLEZA Y COMPONENT" "Y WOBBLE 09903151400" 0.061179534
2 393 "FORTLEZA Y COMPONENT" "LONGITUDE NUTATION " -0.053816501
```

Output file consists of a) header -- lines which start from #;
b) comments -- lines which start from *; c) body. The body has records of fixed length. Fields:

- 1-5 -- index of the first parameter of the pair,
- 7-11 -- index of the second parameter of the pair,
- 15-34 -- name of the first parameter of the pair,
- 39-58 -- name of the second parameter of the pair,
- 61-73 -- correlation coefficient. Format: F12.9 .

(Of course, correlation cannot be obtained with precision better than 0.001, but more digits may appear useful for diagnostic purposes.)

Output file name is CORLxx where xx are Solve user initials and it is located in SOLVE scratch directory. The file is purged at the start of BATCH. New correlations are appended to the end of the file in order of their computation.

5 Acknowledgments

The authors acknowledge assistance from Ed Esfandiari, Mark Hayes, Ed Himwich, Clara Kuehn, Chopo Ma and Jim Ryan, all of whom are employed by or under contract to the Goddard Space Flight Center's Crustal Dynamics Project.

6 References

- 1) C. Ma et. al., Measurement of Horizontal Motions in Alaska Using Very Long Baseline Interferometry, Journal of Geophysical Research 95, 21991-22011, 1990.
- 2) L. Petrov "Multigroup LSQ and its generalization" (1997) Unpublished.
http://gemini.gsc.nasa.gov/solve_root/help/mglsq.ps
- 3) Solve online documentation. <http://gemini.gsc.nasa.gov/solve.html>

7 Appendix

File directories are the key to tailoring BATCH to a specific installation, and BATCH's system manager must be familiar with the details of BATCH's file structure. Users do not need to be as familiar, but will find some familiarity helpful. Because BATCH's file directories will probably differ among installations and are subject to change, this guide refers to BATCH's directories by uppercase names, listed in the following table. These names are the Fortran parameters BATCH uses to identify the directories. So readers can find out their installation's current directories by referring to the Solve parameter include file, \$MK4_ROOT/local/<CENTER_ABR>.lcl.

7.1 BATCH's file directories

PSOLVE_CGM_DIR = directory where BATCH creates regular CGMs. BATCH expects the full path to input CGMs, which consequently can be kept on

Solve guide 2

any directory.

Environment variable CGM_DIR supersedes this variable.

PSOLVE_PROG_DIR = directory where Solve programs, such as enter and liptn, are kept. If the user tries running one of these programs by typing just the program name, and he/she gets a "not found" message, the user should try typing the program name preceded by this directory.,

Environment variable SOLVE_DIR supersedes this variable.

PSOLVE_WORK_DIR = directory where the work files needed for a BATCH run are kept. For example, if the user is running under the XX initials, his/her error messages will go to ERRFXX on this directory.

Environment variable WORK_DIR supersedes this variable.

PSOLVE_SPOOL_DIR = directory where BATCH writes spool files.

Environment variable SPOOL_DIR supersedes this variable.

PSOLVE_SAVE_DIR = directory where various special Solve files are kept.

Environment variable PSOLVE_SAVE_DIR supersedes this variable.

Questions and comments about this guide should be sent to:

Leonid Petrov (Leonid.Petrov at nasa.gov)

Last update: 2020.06.18