ADAPTIVE SIMULATED ANNEALING (ASA) ©

Lester Ingber

Lester Ingber Research
P.O. Box 857
McLean, VA 22101

ingber@alumni.caltech.edu

1. GNU General Public License (GPL)

This Adaptive Simulated Annealing (ASA) code is being made available under a GNU COPYING "copyleft" license, and is owned by Lester Ingber[1]. Please read the copy of this license contained in this directory. Its intent is to make this code publicly available to the widest audience while maintaining the integrity of the basic algorithm.

2. Documentation

2.1. Table of Contents

A Table of Contents of the three levels of headers with their page numbers is located at the end of this document. This may be placed after the first title page, or left at the end for quick reference.

2.2. readme.ms

The readme.ms file is used to prepare other documentation files using UNIX® MS macros.

2.3. README and README+

README is an ASCII file that can be previewed on your screen or sent to an ASCII lineprinter.

README+ is README without any filters to strip off underlining and bold enhancements. This is uuencoded to the file README+.uu in order to pass through the shar utility. If you have downloaded ASA-shar or ASA-shar.Z, to use this file, 'uudecode README+.uu' will leave README+.

2.4. asa.[13nl] Manpage

The README or README+ file can be copied to a file named asa.[l3], and asa.[13] can be installed as MANPATH/cat1/asa.1 or MANPATH/cat3/asa.3, where MANPATH is the place your man directory is located. If you do not have any cat[13] directories on your system, then installing a copy of README or README+ as MANPATH/man[13nl]/asa.[13nl], choosing one of the suffixes in [13nl] for your choice of directory and asa file name, should work fine on most machines. (However, passing asa.[13nl] which is the equivalent of README[+] through man may strip out some items like \"asa_out\".) You likely can avoid some further undesirable formatting by man by placing '.nf' on the first line of this file.

2.5. README.ps

README.ps is a PostScript® formatted file which may be previewed on your screen if you have the proper software, or it may be sent to a PostScript® printer to produce hardcopy.

2.6. Additional Documentation

CHANGES is a terse record of major changes made in the ASA code. NOTES is a collection of recommended enhancements, modifications, comments, caveats, etc., that might be of interest.

The file asa_new in ftp.alumni.caltech.edu: /pub/ingber is a list of major changes in ASA since the last announcement to the ASA_list. This can be used as a quick guide to determine if you should download the latest ASA code in the archive before the next announcement.

An addendum to NOTES is the file asa_papers in ftp.alumni.caltech.edu: /pub/ingber, listing some (p)reprints that have used ASA or its precursor VFSR. This separation of information is to minimize updating versions of the ASA directory due to changes in this section.

It is certain that there is much research to be done on determining optimal or even reasonable ASA parameters, e.g., the set of Program Options, for different classes of systems, especially in higher dimensional spaces of user parameters. (In the NOTES file are some comments on how you might use ASA recursively to determine the optimal set of some Program Options for a given system.) A major purpose of making this code publicly available is to motivate more of this research, and thus make the code more useful to a wider audience.

2.7. Parallelizing ASA and PATHINT Project (PAPP)

The file /pub/ingber/MISC.DIR/parallel.txt contains an update of the Parallelizing ASA and PATHINT Project (PAPP). No code will be released until it has passed some reasonable tests and has reasonable documentation.

2.8. Additional Information

Sorry, I cannot assume the task of mailing out hardcopies of code or papers. My volunteer time assisting people with their queries on my codes and papers must be limited to electronic mail correspondence. Commercial consulting appointments can be made by contacting me via e-mail, mail, or calling 1.800.L.INGBER.

3. Availability of ASA Code

3.1. Caltech

The latest Adaptive Simulated Annealing (ASA) code and some related (p)reprints can be retrieved via anonymous ftp from ftp.alumni.caltech.edu [131.215.139.234] in the /pub/ingber directory. This archive also can be accessed via WWW path http://alumni.caltech.edu/~ingber/ or ftp://ftp.alumni.caltech.edu/pub/ingber/.

Interactively [brackets signify machine prompts]:

```
[your_machine%] ftp ftp.alumni.caltech.edu
[Name (...):] anonymous
[Password:] your_e-mail_address
[ftp>] cd pub/ingber
[ftp>] binary
[ftp>] ls
[ftp>] get file_of_interest
[ftp>] quit
```

The 00index file contains an index of the other files and information on getting gzip and unshar for DOS®, MAC®, UNIX®, and VMS® systems.

The latest version of ASA, ASA−x.y (x and y are version numbers), can be obtained in several formats. ASA−shar.Z is a compress'd shar'd file of the current code. For the convenience of users who do not have any uncompress/gunzip utility, there is a file ASA−shar which is an uncompress'd copy of ASA−shar.Z; if you do not have sh or shar, you still can delete the first−column X's and separate the files at the END_OF_FILE locations. There are tar'd versions in compress and gzip format, ASA.tar.Z and ASA.tar.gz, respectively. There also is a current zip'd version, ASA.zip, in which all files have been processed through unix2dos. Directory /pub/ingber/0lower.dir contains links to these files for some PC users who may have difficulty with upper case.

Patches ASA−diff−x1.y1−x2.y2.Z up to the present version can be prepared, if a good case for doing so is presented. These may be concatenated as required before applying. If you require a specific patch that is not contained in the archive, contact ingber@alumni.caltech.edu.

3.2. Electronic Mail

If you do not have ftp access, get information on the FTPmail service by: mail ftpmail@decwrl.dec.com, and send only the word "help" as the body of the message. You will receive the information in /pub/ingber/UTILS.DIR/ftpmail.txt. Similarly, from a BITNET site, send the word "help" as the body of a message to bitftp@pucc (bitftp@pucc.bitnet if from an Internet site).

If any of the above are not possible, and if your mailer can handle large files (please test this first), the code or papers you require can be sent as uuencode'd compress'd files via electronic mail. If you have gzip, resulting in smaller files, please state this.

3.3. ASA Mailing List

If you wish to be placed on the electronic mailing ASA_list to receive major updates between public announcements of new versions, please send e−mail stating this request to asa−request@alumni.caltech.edu. Update notices are sent to the ASA_list about every month or two, more frequently if warranted, e.g., in cases of important bug fixes; these notices are the only e−mail sent to the ASA_list. This is highly recommended if you plan to use ASA on complex systems, as there is ongoing research using and testing ASA by many users. To unsubscribe from this list, simply send an electronic mail with this request to asa−request@alumni.caltech.edu.

4. Background

4.1. Context

The ASA code was first developed in 1987 as Very Fast Simulated Reannealing (VFSR) to deal with the necessity of performing adaptive global optimization on multivariate nonlinear stochastic systems[2]. VFSR was recoded and applied to several complex systems, in combat analysis[3], finance[4], and neuroscience[5]. The first applications to combat analysis used code written in RATFOR and converted into FORTRAN. Other applications since then have used new code written in C. (The NOTES file contains some comments on interfacing ASA with FORTRAN codes.) A paper has indicated how this technique can be enhanced by combining it with some other powerful algorithms, e.g., to produce an algorithm for parallel computation[6]. In November 1992, the VFSR C-code was rewritten, e.g., changing to the use of long descriptive names, and made publicly available as version 6.35 under the same GNU license as this ASA code[7].

Beginning in January 93, many adaptive features were developed, largely in response to users' requests, leading to this ASA code. ASA has been examined in the context of a review of methods of simulated annealing using annealing versus quenching (faster temperature schedules than permitted by basic heuristic proof of ergodicity)[8]. ASA is now used world-wide across many disciplines[9].

4.2. Outline of ASA Algorithm

Details of the ASA algorithm are best obtained from the published papers. There are three parts to its basic structure.

4.2.1. Generating Probability Density Function

In a D-dimensional parameter space with parameters p^i having ranges $[A_i, B_i]$, about the k'th last saved point (e.g, a local optima), p^i_k, a new point is generated using a distribution defined by the product of distributions for each parameter, $g^i(y^i; T_i)$ in terms of random variables $y^i \in [-1, 1]$, where $p^i_{k+1} = p^i_k + y^i(B_i - A_i)$, and "temperatures" T_i,

$$g^i(y^i; T_i) = \frac{1}{2(|y^i| + T_i)\ln(1 + 1/T_i)} \ .$$

4.2.2. Acceptance Probability Density Function

The cost functions, $C(p_{k+1}) - C(p_k)$, are compared using a uniform random generator, $U \in [0, 1)$, in a "Boltzmann" test: If

$$\exp[-(C(p_{k+1}) - C(p_k))/T_{\mathrm{cost}}] > U \ ,$$

where T_{cost} is the "temperature" used for this test, then the new point is accepted as the new saved point for the next iteration. Otherwise, the last saved point is retained.

4.2.3. Reannealing Temperature Schedule

The annealing schedule for each parameter temperature, T_i from a starting temperature T_{i0}, is

$$T_i(k_i) = T_{0i} \exp(-c_i k_i^{1/D}) \ .$$

This is discussed further below.

The annealing schedule for the cost temperature is developed similarly to the parameter temperatures. However, the index for reannealing the cost function, k_{cost}, is determined by the number of accepted points, instead of the number of generated points as used for the parameters. This choice was made because the Boltzmann acceptance criteria uses an exponential distribution which is not as fat-tailed as the ASA distribution used for the parameters. This schedule can be modified using several OPTIONS. In particular, the Pre-Compile DEFINE_OPTIONS USER_COST_SCHEDULE permits quite arbitrary functional modifications for this annealing schedule.

As determined by the Program Options selected, the parameter "temperatures" may be periodically adaptively reannealed, or increased relative to their previous values, using their relative first derivatives with respect to the cost function, to guide the search "fairly" among the parameters.

4.3. Efficiency Versus Necessity

ASA is not necessarily an "efficient" code. For example, if you know that your cost function to be optimized is something close to a parabola, then a simple gradient Newton search method most likely would be faster than ASA. ASA is believed to be faster and more robust than other simulated annealing techniques for *most* complex problems with multiple local optima; again, be careful to note that some problems are best treated by other algorithms. If you do not know much about the structure of your system, and especially if it has complex constraints, and you need to search for a global optimum, then this ASA code is heartily recommended to you.

5. Outline of Use

Set up the ASA interface: Your program should be divided into two basic modules. (1) The user calling procedure, containing the cost function to be minimized (or its negative if you require a global maximum), here is contained in user.c and user.h. (2) The ASA optimization procedure, here is contained in asa.c and asa.h. The file asa_user.h contains definitions and macros common to both asa.h and user.h. Furthermore, there are some options to explore/read below. It is assumed there will be no confusion over the standard uses of the term "parameter" in different contexts, e.g., as an element passed by a subroutine or as a physical coefficient in a cost function.

ASA has been run successfully on many machines under many compilers. To check out your own system, you can run 'make' (or the equivalent set of commands in the Makefile), and compare your asa_out and user_out files to the test_asa and test_usr files, respectively, provided with this code. (For these runs, TIME_CALC=TRUE, discussed below, was added to the compilation options.) No attempt was made to optimize any compiler, so that the test runs do not really signify any testing of compilers or architectures; rather they are meant to be used as a guide to determine what you might expect on your own machine.

The major sections below describe the compilation procedures, the Program Options available to you to control the code, the use of templates to set up your user module and interface to the asa module, and how to submit bug reports.

If you already have your own cost function defined, as a quick guide to get started, you can search through user.c for all occurrences of "MY_COST" to insert the necessary definitions required to run ASA.

6. Makefile/Compilation Procedures

The PostScript® README.ps and ASCII README and README+ files were generated using 'make doc'. The Makefile describes some options for formatting these files differently. Use 'make' or 'make all' to compile and run asa_run, the executable prepared for the test function. Examine the Makefile to determine the "clean" options available.

Since complex problems by their nature are often quite unique, it is unlikely that the default parameters are just right for your problem. However, experience has shown that if you *a priori* do not have any reason to determine your own parameters, then you might do just fine using these defaults, and these are recommended as a first-order guess. These defaults can be changed simply by adding to the DEFINE_OPTIONS line in the Makefile, by passing options on your command line, and by changing structure elements in the user or asa module as described below. Depending on how you integrate ASA into your own user modules, you may wish to modify this Makefile or at least use some of these options in your own compilation procedures.

Note that the Makefile is just a convenience, not a necessity, to use ASA. E.g., on systems which do not support this utility, you may simply compile the files following the guidelines in the Makefile, taking care to pass the correct DEFINE_OPTIONS to your compilation commands at your shell prompt. Still another way, albeit not as convenient, is to make the desired changes in the asa_user.h, and asa.h or user.h files as required.

Since the Makefile contains comments giving short descriptions of some options, it should be considered as an extension of this documentation file. For convenience, most of this information is repeated below. However, to see how they can be used in compilations, please read through the Makefile.

For example, to run the ASA test problem using the gcc compiler, you could just type at your "%" prompt:

```
% gcc -g -DASA_TEST=TRUE -o asa_run user.c asa.c -lm
% asa_run
```

You may have to feed different options to your own compiler. The resulting asa_out file should only differ from the test_asa file by having different values for the OPTIONS_FILE and TIME_CALC lines, and by not having a few lines marking the CPU time.

If you have defined your own cost function within the "MY_COST" guides in user.c, then ASA_TEST should be set to FALSE (the default if ASA_TEST is not defined in your compilation lines or in the Makefile). The code for ASA_TEST=TRUE is given just above these guides as a template to use for your own cost function.

7. User Options

The DEFINE_OPTIONS are organized into two groups: Pre-Compile Options and (Pre-Compile) Printing Options. In addition, there are some alternatives to explore under Compiler Choices and Document Formatting. Below are the DEFINE_OPTIONS with their defaults. The Program Options are further discussed in other sections in this document.

7.1. Pre-Compile DEFINE_OPTIONS

7.1.1. OPTIONS_FILE=TRUE

You can elect to read in the Program Options from asa_opt by setting OPTIONS_FILE=TRUE. OPTIONS_FILE=TRUE can be set in the Makefile in compilation commands or in asa_user.h.

7.1.2. ASA_LIB=FALSE

Setting ASA_LIB=TRUE will facilitate your running asa() as a library call from another program, calling asa_main() in user.c. In the templates provided, all initializations and cost function definitions are set up in user.c. For example, you may wish to have some data read in to a module that calls asa_main(), then parses out this information to the arrays in asa_main() and initialize_parameters (and possibly recur_initialize_parameters). In conjunction with setting printout to stdout (see ASA_OUT and USER_ASA_OUT), this can be a convenient way of using the same asa_run executable for many runs.

7.1.3. HAVE_ANSI=TRUE

Setting HAVE_ANSI=FALSE will permit you to use an older K&R C compiler. This option can be used if you do not have an ANSI compiler, overriding the default HAVE_ANSI=TRUE. If you use HAVE_ANSI=FALSE, change CC and CDEBUGFLAGS as described in the Makefile.

7.1.4. IO_PROTOTYPES=TRUE

Some machines do not like any other I/O prototyping other than those in their own include files, e.g., like one Convex-120 that was tested. Other machines, like a Dec-3100 under Ultrix complained that the ANSI I/O prototypes were inconsistent. A Sun under gcc gave warnings if no I/O prototypes were present. Therefore, the defaults in asa_user.h use K&R system prototypes even for the ANSI compiler, for fprintf, fflush, fclose, and exit, and for fscanf in user.h. Setting IO_PROTOTYPES=FALSE will comment out even these declarations. This also has worked on an Indigo and on a Cray C90/UNICOS 8.0.

7.1.5. TIME_CALC=FALSE

Some systems do not have the time include files used here; others have different scales for time. Setting TIME_CALC=TRUE will permit use of the time routines. In the NOTES are some contributed code that should be useful for some particular systems.

7.1.6. TIME_STD=FALSE

Some systems, e.g., hpux, use other Unix-standard macros to access time. Setting TIME_STD=TRUE when using TIME_CALC=TRUE will use these time routines instead.

7.1.7. INT_LONG=TRUE

Some smaller systems choke on ‘long int’ and this option can be set to INT_LONG=FALSE to turn off warnings and possibly some errors. The cast LONG_INT is used to define ‘int’ or ‘long int’ appropriately.

7.1.8. INT_ALLOC=FALSE

The cast on *number_parameters is set to ALLOC_INT which defaults to LONG_INT. On some machines, ALLOC_INT might have to be set to int if there is a strict requirement to use an (unsigned) int for calloc, while ‘long int’ still can be used for other aspects of ASA. If ALLOC_INT is to be set to int, set INT_ALLOC to TRUE.

7.1.9. SMALL_FLOAT=1.0E-18

SMALL_FLOAT is a measure of accuracy permitted in log and divide operations in asa, i.e., which is not precisely equivalent to a given machine’s precision. There also are Pre-Compile DEFINE_OPTIONS to separately set constants for minimum and maximum doubles and precision permitted by your machine. Experts who require the very best precision can fine-tune these parameters in the code.

Such issues arise because the fat tail of ASA, associated with high parameter temperatures, is very important for searching the breadth of the ranges especially in the initial stages of search. However, the parameter temperatures require small values at the final stages of the search to converge to the best solution, albeit this is reached very quickly given the exponential schedule proven in the referenced publications to be permissible with ASA. Note that the test problem in user.c is a particularly nasty one, with 1E20 local minima and requiring ASA to search over 12 orders of magnitude of the cost function before correctly finding the global minimum. Thus, intermediate values disagree somewhat for SMALL_FLOAT=1.0E-12 from the settings using SMALL_FLOAT=1.0E-18 (the default); they agree if SMALL_FLOAT=1.0E-12 while also setting MIN_DOUBLE=1.0E-18. The results diverge when the parameter temperatures get down to the range of E-12, limiting the accuracy of the SMALL_FLOAT=1.0E-12 run.

7.1.10. MIN_DOUBLE=SMALL_FLOAT

You can define your own machine’s minimum positive double here if you know it.

7.1.11. MAX_DOUBLE=1.0/SMALL_FLOAT

You can define your own machine’s maximum double here if you know it.

7.1.12. EPS_DOUBLE=SMALL_FLOAT

You can define your own machine's maximum precision here if you know it.

7.1.13. NO_PARAM_TEMP_TEST=FALSE

If NO_PARAM_TEMP_TEST is set to TRUE, then all parameter temperatures less than EPS_DOUBLE are set to EPS_DOUBLE, and no exit is called.

7.1.14. NO_COST_TEMP_TEST=FALSE

If NO_COST_TEMP_TEST is set to TRUE, then a cost temperature less than EPS_DOUBLE is set to EPS_DOUBLE, and no exit is called.

7.1.15. SELF_OPTIMIZE=FALSE

The user module contains a template to illustrate how ASA may be used to self-optimize its Program Options. This can be very CPU-expensive and is of course dependent on how you define your recursive cost function (recur_cost_function in the user module). The example given returns from recur_cost_function the number of function evaluations taken to optimization the test cost_function, with the constraint to only accept optimizations of the cost_function that are lower than a specified value. A few lines of code can be uncommented in user.c to force a fast exit for this demo; search for FAST EXIT. This example uses OPTIONS_FILE=FALSE (the default) in the Pre-Compile Options; note that OPTIONS_FILE=TRUE here would set Program Options from asa_opt for the top level program, not for the Program Options for the cost_function().

This can be useful when approaching a new system, and it is suspected that the default ASA Program Options are not at all efficient for this system. It is suggested that a trimmed cost function or data set be used to get a reasonable guess for a good set of Program Options. ASA has demonstrated that it typically is quite robust under a given set of Program Options, so it might not make too much sense to spend lots of resources performing additional fine tuning of the these options. Also, it is possible you might crash the code by permitting ranges of Program Options that cause your particular cost_function to return garbage to asa().

7.1.16. ASA_TEST=FALSE

Setting ASA_TEST to TRUE will permit running the ASA test problem. This has been added to the DEFINE_OPTIONS in the Makefile so that just running make will run the test problem for the new user.

Searching user.c for "MY_COST" provides a guide to the user for additional code to add for his/her own system. Just above each occurrence of these guides is the corresponding code for ASA_TEST=TRUE. Keeping the default of ASA_TEST set to FALSE permits such changes without overwriting the test example.

7.1.17. ASA_TEMPLATE=FALSE

There are several templates that come with the ASA code, used to test several OPTIONS. To permit use of these OPTIONS without having to delete these extra tests, these templates are wrapped with ASA_TEMPLATE. To use tests associated with these OPTIONS, which can be determined by reading the code, just set ASA_TEMPLATE to TRUE in the Makefile or in your compilation procedures.

Note that running the ASA test problem in user.c is not affected by ASA_TEMPLATE. To use your own cost function, you must at least rewrite relevant portions of cost_function() and initialize_parameters() in user.c.

7.1.18. OPTIONAL_DATA=FALSE

It can be useful to return additional information to the user module from the asa module. When OPTIONAL_DATA is set to true, an additional Program Option pointer, *asa_data, is available in USER_DEFINES *OPTIONS to gather such data.

In one ASA_TEMPLATE provided (see the set of DEFINE_OPTIONS used in the Makefile), OPTIONAL_DATA is used together with SELF_OPTIMIZE to find the set of ASA parameters giving the (statistically) smallest number of generated points to solve the ASA test problem, assuming this were run several times with different random seeds for randflt in user.c (e.g., changing "seed" in myrand). Here, asa_data[0] is used as a flag to print out asa_data[1] in user.c, set to *best_number_generated_saved in asa.c.

7.1.19. USER_COST_SCHEDULE=FALSE

The function used to control the cost_function temperature schedule is of the form test_temperature in asa.c. If the user sets the Pre-Compile DEFINE_OPTIONS USER_COST_SCHEDULE to TRUE, then this function of test_temperature can be controlled, adaptively if desired, in user.c in cost_schedule() (and in recur_cost_schedule() if SELF_OPTIMIZE is TRUE) by setting USER_COST_SCHEDULE to TRUE. The names of these functions are set to the relevant pointer in user.c, and can be changed if desired, i.e.,

```
USER_OPTIONS->cost_schedule = user_cost_schedule;
RECUR_USER_OPTIONS->cost_schedule = recur_user_cost_schedule;
```

7.1.20. USER_REANNEAL_FUNCTION=FALSE

In asa.h, the macro

```
#define \
        REANNEAL_FUNCTION(temperature, tangent, max_tangent) \
            (temperature * (max_tangent / tangent))
```

is used to determine the new temperature, subject to further tests in reanneal(). This is the default if USER_REANNEAL_FUNCTION is FALSE.

If the user sets the Pre-Compile DEFINE_OPTIONS USER_REANNEAL_FUNCTION to TRUE, then the function controlling the new reannealed temperature can be controlled, adaptively if desired using USER_OPTIONS, in user.c in user_reanneal(), and in recur_user_reanneal() if SELF_OPTIMIZE is TRUE. The names of these functions are set to the relevant pointer in user.c, and can be changed if desired, i.e.,

```
USER_OPTIONS->reanneal_function = user_reanneal;
RECUR_USER_OPTIONS->reanneal_function = recur_user_reanneal;
```

7.1.21. ASA_SAMPLE=FALSE

When ASA_SAMPLE is set to TRUE, data is collected by ASA during its global optimization process to importance-sample the user's variables. Five OPTIONS become available to monitor the sampling: n_accepted, bias_acceptance, *bias_generated, average_weights, and limit_weights.

If average_weights exceeds the user's choice of limit_weights, then the ASA_OUT file will contain additional detailed information, including temperatures and biases for each current parameter. To facilitate extracting importance-sampled information from the file printed out by the asa module, all relevant lines start with :SAMPLE[|:|#|+].

Many Monte Carlo sampling techniques require the user to guess an appropriately decreasing "window" to sample the variable space. The fat tail of the ASA generating function, and the decreasing effective range of newly accepted points driven by exponentially decreasing temperature schedules, removes this arbitrary aspect of such sampling.

However, note that, albeit local optima are sampled, the efficiency of ASA optimization most often leads to poor sampling in regions whose cost function is far from the optimal point; many such points may be important contributions to algorithms like integrals. Accordingly, ASA_SAMPLE likely is best used to explore new regions and new systems.

To increase the sampling rate and thereby to possibly increase the accuracy of this algorithm, use one or a combination of the various OPTIONS available for slowing down the annealing performed by ASA.

7.1.22. ASA_PARALLEL=FALSE

When ASA_PARALLEL is set to TRUE, parallel blocks of generated states are calculated of number equal to the minimum of OPTIONS->gener_block and OPTIONS->gener_block_max. For most systems with complex nonlinear cost functions that require the fat tail of the ASA distribution, leading to high generated to acceptance ratios, this is the most CPU intensive part of ASA that can benefit from parallelization.

The actual number calculated is determined by a moving average, determined by OPTIONS->gener_mov_avr, of the previous numbers of OPTIONS->gener_block of generated states required to find a new best accepted state. If and when OPTIONS->gener_mov_avr is set to 0, then OPTIONS->gener_block is not changed thereafter.

7.2. Printing DEFINE_OPTIONS

7.2.1. ASA_PRINT=TRUE

Setting this to FALSE will suppress all printing within asa.

7.2.2. ASA_OUT=\"asa_out\"

The name of the output file containing all printing from asa. If you wish to attach a process number use ASA_OUT=\"asa_out_$$\". (Use ASA_OUT=\"asa_out_$$$$\" if this is set in the Makefile.) If ASA_OUT=\"STDOUT\" then ASA will print to stdout.

7.2.3. USER_ASA_OUT=FALSE

When USER_ASA_OUT is set to TRUE, an additional Program Option pointer, *asa_out_file, is used to dynamically set the name(s) of the file(s) printed out by the asa module. (This overrides any ASA_OUT settings.) In user.c, if USER_OPTIONS->asa_out_file = "STDOUT";, then ASA will print to stdout.

In one ASA_TEMPLATE provided (see the set of DEFINE_OPTIONS used in the Makefile), USER_ASA_OUT is used to generate multiple files of separate ASA runs. (If USER_OPTIONS->QUENCH_PARAMETERS and/or USER_OPTIONS->QUENCH_COST is set to TRUE in user.c, then this ASA_TEMPLATE will separate runs with different quenching values.)

7.2.4. ASA_PRINT_INTERMED=TRUE

This option is only effective if ASA_PRINT is TRUE. Setting ASA_PRINT_INTERMED to FALSE will suppress much intermediate printing within asa, especially arrays which can be large when the number of parameters is large. Printing at intermediate stages of testing/reannealing has been turned off when SELF_OPTIMIZE is set to TRUE, since there likely can be quite a bit of data generated; this can be changed by explicitly setting ASA_PRINT_INTERMED to TRUE in the Makefile or on your compilation command lines.

7.2.5. ASA_PRINT_MORE=FALSE

Setting ASA_PRINT_MORE to TRUE will print out more intermediate information, e.g., new parameters whenever a new minimum is reported. As is the case whenever tangents are not calculated by choosing some ASA options, normally the intermediate values of tangents will not be up to date.

7.3. Program OPTIONS

```
typedef struct {
        LONG_INT LIMIT_ACCEPTANCES;
        LONG_INT LIMIT_GENERATED;
        int LIMIT_INVALID_GENERATED_STATES;
        double ACCEPTED_TO_GENERATED_RATIO;

        double COST_PRECISION;
        int MAXIMUM_COST_REPEAT;
        int NUMBER_COST_SAMPLES;
        double TEMPERATURE_RATIO_SCALE;
        double COST_PARAMETER_SCALE;
        double TEMPERATURE_ANNEAL_SCALE;
        int USER_INITIAL_COST_TEMP;
        double *user_cost_temperature;

        int INCLUDE_INTEGER_PARAMETERS;
        int USER_INITIAL_PARAMETERS;
        ALLOC_INT SEQUENTIAL_PARAMETERS;
        double INITIAL_PARAMETER_TEMPERATURE;
        int RATIO_TEMPERATURE_SCALES;
        double *user_temperature_ratio;
        int USER_INITIAL_PARAMETERS_TEMPS;
        double *user_parameter_temperature;

        int TESTING_FREQUENCY_MODULUS;
        int ACTIVATE_REANNEAL;
        double REANNEAL_RESCALE;
        LONG_INT MAXIMUM_REANNEAL_INDEX;

        double DELTA_X;
        int DELTA_PARAMETERS;
        double *user_delta_parameter;
        int USER_TANGENTS;
        int CURVATURE_0;

        int QUENCH_PARAMETERS;
        double *user_quench_param_scale;
        int QUENCH_COST;
        double *user_quench_cost_scale;

#if OPTIONAL_DATA
        double *asa_data;
#endif
#if USER_ASA_OUT
        char *asa_out_file;
#endif
#if USER_COST_SCHEDULE
        double (*cost_schedule) ();
#endif
#if USER_REANNEAL_FUNCTION
        double (*reanneal_function) ();
#endif
#if ASA_SAMPLE
```

```
            int n_accepted;
            double bias_acceptance;
            double *bias_generated;
            double average_weights;
            double limit_weights;
#endif
#if ASA_PARALLEL
            int gener_mov_avr;
            LONG_INT gener_block;
            LONG_INT gener_block_max;
#endif
   }
USER_DEFINES;
```

Note that two ways are maintained for passing the Program Options. Check the comments in the NOTES file. It may be necessary to change some of the options for some systems. Read the NOTES file for some ongoing discussions and suggestions on how to try to optimally set these options. Note the distinction between trying to speed up annealing/quenching versus trying to slow down annealing (which sometimes can speed up the search by avoiding spending too much time in some local optimal regions). Templates are set up in ASA to accommodate all alternatives. Below, the defaults are given in square brackets [].

(A) user module.
When using ASA as part of a large library, it likely is easiest to make these changes within the user module, e.g., using the template placed in user.c. The Program Options are stored in the structure USER_DEFINES *OPTIONS (named USER_DEFINES *USER_OPTIONS in the user module).

(B) asa module.
It likely is most efficient to use a separate data file in the asa module, avoiding repeated compilations of the code, to test various combinations of Program Options, e.g., using the file asa_opt when OPTIONS_FILE=TRUE in the Makefile or on your compilation command lines.

7.3.1. OPTIONS->LIMIT_ACCEPTANCES[10000]

The maximum number of states accepted before quitting. All the templates in ASA have been set to use LIMIT_ACCEPTANCES=1000 to illustrate the way these options can be changed. If LIMIT_ACCEPTANCES is set to 0, then no limit is observed. This can be useful for some systems that cannot handle large integers. (To exit at a specific number of generated points, see the discussion at LIMIT_INVALID_GENERATED_STATES below.)

7.3.2. OPTIONS->LIMIT_GENERATED[99999]

The maximum number of states generated before quitting. If LIMIT_GENERATED is set to 0, then no limit is observed. This can be useful for some systems that cannot handle large integers. (To exit at a specific number of generated points, see the discussion at LIMIT_INVALID_GENERATED_STATES below.)

7.3.3. OPTIONS->LIMIT_INVALID_GENERATED_STATES[1000]

This sets limits of repetitive invalid generated states, e.g., when using this method to include constraints. This also can be useful to quickly exit asa() if this is requested by your cost function: Setting the value of LIMIT_INVALID_GENERATED_STATES to 0 will exit at the next calculation of the cost function (possibly after a few more exiting calls to calculate tangents and curvatures). For example, to exit asa() at a specific number of generated points, set up a counter in your cost function, e.g., similar to the one in the test function in user.c. For all calls >= the limit of the number of calls to the cost function, terminate by setting USER_OPTIONS->LIMIT_INVALID_GENERATED_STATES = 0 and setting *cost_exit = FALSE. (Note that the number of calls counted will include those calls used to set up some initializations.)

7.3.4. OPTIONS->ACCEPTED_TO_GENERATED_RATIO[1.0E-6]

The least ratio of accepted to generated states. If this value is encountered, then the usual tests, including possible reannealing, are initiated even if the timing does not coincide with the set TESTING_FREQUENCY_MODULUS (defined below). All the templates in ASA have been set to use ACCEPTED_TO_GENERATED_RATIO=1.0E-4 to illustrate the way these options can be changed.

7.3.5. OPTIONS->COST_PRECISION[1.0E-18]

This sets the precision required of the cost function if exiting because of reaching MAXIMUM_COST_REPEAT.

7.3.6. OPTIONS->MAXIMUM_COST_REPEAT[5]

The maximum number of times that the cost function repeats itself before quitting.

7.3.7. OPTIONS->NUMBER_COST_SAMPLES[5]

The number of cost function values sampled to determine the initial cost function temperature.

7.3.8. OPTIONS->TEMPERATURE_RATIO_SCALE[1.0E-5]

This scale is a guide to the expected cost temperature of convergence within a small range of the global minimum. As explained in the ASA papers, and as outlined in the NOTES, this is used to set the rates of annealing. Here is a brief description in terms of the temperature schedule outlined above.

As a useful physical guide, the temperature is further parameterized in terms of quantities m_i and n_i, derived from an "expected" final temperature (which is not enforced in ASA), T_{fi},

$$T_{fi} = T_{0i} \exp(-m_i) \text{ when } k_{fi} = \exp n_i \ ,$$

$$c_i = m_i \exp(-n_i / D) \ .$$

However, note that since the initial temperatures and generating indices, T_{0i} and k_i, are independently scaled for each parameter, it usually is reasonable to simply take $\{c_i, m_i, n_i\}$ to be independent of the index i, i.e., to be $\{c, m, n\}$ for all i.

In asa.c,

$$m = -\log(\text{TEMPERATURE\_RATIO\_SCALE}) \ .$$

This can be overridden if RATIO_TEMPERATURE_SCALES (further discussed below) is set to TRUE, and then values of multipliers of $-\log(\text{TEMPERATURE\_RATIO\_SCALE})$ are used in asa.c. These multipliers are calculated in the user module as USER_OPTIONS->user_temperature_ratio[] (and passed to OPTIONS->user_temperature_ratio[] in the asa module). Then,

$$m_i = m \, \text{OPTIONS}- > \text{user\_temperature\_ratio[i]} \ .$$

For large numbers of parameters, TEMPERATURE_RATIO_SCALE is a very influential Program Option in determining the scale of parameter annealing. It likely would be best to start with a larger value than the default, to slow down the annealing.

7.3.9. OPTIONS->COST_PARAMETER_SCALE[1.0]

This is the ratio of cost:parameter temperature annealing scales. As explained in the ASA papers, and as outlined in the NOTES, this is used to set the rates of annealing.

In terms of the algebraic development given above for the TEMPERATURE_RATIO_SCALE, in asa.c,

$$c_{\text{cost}} = c \, \text{COST\_PARAMETER\_SCALE} \ .$$

COST_PARAMETER_SCALE is a very influential Program Option in determining the scale of annealing of the cost function.

7.3.10. OPTIONS->TEMPERATURE_ANNEAL_SCALE[100.0]

This scale is a guide to achieve the expected cost temperature sought by TEMPERATURE_RATIO_SCALE within the limits expected by LIMIT_ACCEPTANCES. As explained in the ASA papers, and as outlined in the NOTES, this is used to set the rates of annealing.

In terms of the algebraic development given above for the TEMPERATURE_RATIO_SCALE, in asa.c,

$$n = \log(\text{TEMPERATURE\_ANNEAL\_SCALE}) .$$

For large numbers of parameters, TEMPERATURE_ANNEAL_SCALE probably should at least initially be set to values greater than *number_parameters, although it will not be as influential as TEMPERATURE_RATIO_SCALE.

7.3.11. OPTIONS->USER_INITIAL_COST_TEMP[FALSE]

Setting USER_INITIAL_COST_TEMP to TRUE permits you to specify the initial cost temperature. This can be useful in problems where you want to start the search at a specific scale.

7.3.12. OPTIONS->user_cost_temperature

If USER_INITIAL_COST_TEMP is TRUE, a pointer, OPTIONS->user_cost_temperature, is used to adaptively initialize parameters temperatures. If this choice is elected, then user_cost_temperature[] must be initialized (named USER_OPTIONS->user_cost_temperature[] in the user module). (If USER_INITIAL_COST_TEMP is FALSE, then the pointer *user_cost_temperature must be included in *OPTIONS, but it need not be initialized.)

7.3.13. OPTIONS->INCLUDE_INTEGER_PARAMETERS[FALSE]

Include integer parameters in derivative and reannealing calculations. This is useful when the parameters can be analytically continued between their integer values, or if you set the parameter increments to integral values by setting the DELTA_PARAMETERS option to TRUE, as discussed further below.

7.3.14. OPTIONS->USER_INITIAL_PARAMETERS[FALSE]

ASA always requests that the user guess initial values of starting parameters, since that guess is as good as any random guess the code might make. The default is to use the ASA distribution about this point to generate an initial state of parameters and value of the cost function that satisfy the user's constraints. If USER_INITIAL_PARAMETERS is set to TRUE, then the first user's guess is used to calculate this first state.

7.3.15. OPTIONS->SEQUENTIAL_PARAMETERS[-1]

The ASA default for generating new points in parameter space is to find a new point in the full space, rather than to sample the space one parameter at a time as do most other algorithms. This is in accord with the general philosophy of sampling the space without any prior knowledge of ordering of the parameters. However, if you have reason to believe that at some stage(s) of search there might be some benefit to sampling the parameters sequentially, then set SEQUENTIAL_PARAMETERS to the parameter number you wish to start your annealing cycle, i.e., ranging from 0 to (*parameter_dimension - 1). Then, ASA will cycle through your parameters in the order you have placed them in all arrays defining their properties, keeping track of which parameter is actively being modified in OPTIONS->SEQUENTIAL_PARAMETERS, thereby permitting adaptive changes. Any negative value for SEQUENTIAL_PARAMETERS will use the default ASA algorithm. Upon exiting asa(), SEQUENTIAL_PARAMETERS is reset back to its initial value.

7.3.16. OPTIONS->INITIAL_PARAMETER_TEMPERATURE[1.0]

The initial temperature for all parameters. This is overridden by use of the USER_INITIAL_PARAMETERS_TEMPS option.

7.3.17. OPTIONS->RATIO_TEMPERATURE_SCALES[FALSE]

Different rates of parameter annealing can be set with RATIO_TEMPERATURE_SCALES set to TRUE. This requires initializing an array in the user module as discussed below.

7.3.18. OPTIONS->user_temperature_ratio

If RATIO_TEMPERATURE_SCALES is TRUE, a pointer, OPTIONS->user_temperature_ratio, is used to adaptively set ratios of scales used to anneal the parameters in the cost function. This can be useful when some parameters are not being reannealed, or when setting the initial temperatures (using USER_INITIAL_PARAMETERS_TEMPS set to TRUE) is not sufficient to handle all your parameters properly. This typically is not encountered, so it is advised to look elsewhere at first to improve your search. If this choice is elected, then user_temperature_ratio[] must be initialized (named USER_OPTIONS->user_temperature_ratio[] in the user module). (If RATIO_TEMPERATURE_SCALES is FALSE, then the pointer *user_temperature_ratio must be included in *OPTIONS, but it need not be initialized.)

7.3.19. OPTIONS->USER_INITIAL_PARAMETERS_TEMPS[FALSE]

Setting USER_INITIAL_PARAMETERS_TEMPS to TRUE permits you to specify the initial parameter temperatures. This can be useful in constrained problems, where greater efficiency can be achieved in focussing the search than might be permitted just by setting upper and lower bounds.

7.3.20. OPTIONS->user_parameter_temperature

If USER_INITIAL_PARAMETERS_TEMPS is TRUE, a pointer, OPTIONS->user_parameter_temperature, is used to adaptively initialize parameters temperatures. If this choice is elected, then user_parameter_temperature[] must be initialized (named USER_OPTIONS->user_parameter_temperature[] in the user module). (If USER_INITIAL_PARAMETERS_TEMPS is FALSE, then the pointer *user_parameter_temperature must be included in *OPTIONS, but it need not be initialized.)

7.3.21. OPTIONS->TESTING_FREQUENCY_MODULUS[100]

The frequency of testing for periodic testing and reannealing.

7.3.22. OPTIONS->ACTIVATE_REANNEAL[TRUE]

This permits reannealing to be part of the fitting process. This might have to be set to FALSE for systems with very large numbers of parameters just to decrease the number of function calls.

7.3.23. OPTIONS->REANNEAL_RESCALE[10.0]

The reannealing scale used when MAXIMUM_REANNEAL_INDEX is exceeded.

7.3.24. OPTIONS->MAXIMUM_REANNEAL_INDEX[50000]

The maximum index (number of steps) at which the initial temperature and the index of the temperature are rescaled to avoid losing machine precision. ASA typically is quite insensitive to the value used due to the dual rescaling.

7.3.25. OPTIONS->DELTA_X[0.001]

The fractional increment of parameters used to take numerical derivatives when calculating tangents for reannealing. This is overridden when DELTA_PARAMETERS is set to TRUE, as discussed further below.

Note that this can cause evaluations of your cost function outside a range when a parameter being sampled is at the boundary. However, only values of parameters within the ranges set by the user are actually used for acceptance tests.

7.3.26. OPTIONS->DELTA_PARAMETERS[FALSE]

Different increments, used during reannealing to set each parameter's numerical derivatives, can be set with DELTA_PARAMETERS set to TRUE. This requires initializing an array in the user module as discussed below.

7.3.27. OPTIONS->user_delta_parameter

If DELTA_PARAMETERS is TRUE, a pointer, OPTIONS->user_delta_parameter, is used to adaptively set increments of parameters used to take pseudo-derivatives (numerical derivatives). For example, this can be useful to reanneal integer parameters when a choice is made to permit their derivatives to be taken. If this choice is elected, then OPTIONS->user_delta_parameter[] must be initialized (named USER_OPTIONS->user_delta_parameter[] in the user module). (If DELTA_PARAMETERS is FALSE, then the pointer *user_delta_parameter must be included in *OPTIONS, but it need not be initialized.)

7.3.28. OPTIONS->USER_TANGENTS[FALSE]

By default, asa() calculates numerical tangents (first derivatives) of the cost function for use in reannealing and to provide this information to the user. However, if USER_TANGENTS is set to TRUE, then when asa() requires tangents to be calculated, a value of *valid_state_generated_flag (called *cost_flag in ASA_TEST in user.c) of FALSE is set and the cost function is called. The user is expected to set up a test in the beginning of the cost function to sense this value, and then calculate the tangents[] array (containing the derivatives of the cost function, or whatever sensitivity measure is desired to be used for reannealing) to be returned to asa(). An example is provided with the ASA_TEMPLATE for ASA_SAMPLE.

7.3.29. OPTIONS->CURVATURE_0[FALSE]

If the curvature array is quite large for your system, and you really do not use this information, you can set CURVATURE_0 to TRUE which just requires a one-dimensional curvature[0] to be defined to pass to the asa module (to avoid problems with some systems). This is most useful, and typically is necessary, when minimizing systems with large numbers of parameters since the curvature array is of size number of parameters squared.

If you wish to calculate the curvature array periodically, every reannealing cycle determined by OPTIONS->TESTING_FREQUENCY_MODULUS, then set OPTIONS->CURVATURE_0 to -1.

7.3.30. OPTIONS->QUENCH_PARAMETERS[FALSE]

This Program Option permits you to alter the basic algorithm to perform selective "quenching," i.e., faster temperature cooling than permitted by the ASA algorithm. This can be very useful, e.g., to quench the system down to some region of interest, and then to perform proper annealing for the rest of the run. However, note that once you decide to quench rather than to truly anneal, there no longer is any statistical guarantee of finding a global optimum. Furthermore, once you decide to quench there are many more alternative algorithms you might wish to choose for your system.

Setting QUENCH_PARAMETERS to TRUE can be extremely useful in very large parameter dimensions. As discussed in the first 1989 VFSR paper, the heuristic statistical proof of finding the global optimum reduces to the following: The parameter temperature schedules must suffice to insure that the product of individual generating distributions,

$$g = \prod_i g^i \ ,$$

taken at all annealing times, indexed by k, of not generating a global optimum, given infinite time, is such that

$$\prod_k (1 - g_k) = 0 \ ,$$

which is equivalent to

$$\sum_k g_k = \infty \ .$$

For the ASA temperature schedule, this is satisfied as

$$\sum_k \prod^D 1/k^{-1/D} = \sum_k 1/k = \infty \ .$$

Now, if the temperature schedule above is redefined as

$$T_i(k_i) = T_{0i} \exp(-c_i k_i^{Q/D}) \ ,$$

$$c_i = m_i \exp(-n_i Q/D) \ ,$$

in terms of the "quenching factor" Q, then the above proof fails if $Q > 1$ as

$$\sum_k \prod^D 1/k^{-Q/D} = \sum_k 1/k^Q < \infty \ .$$

This simple calculation shows how the "curse of dimensionality" arises, and also gives a possible way of living with this disease which will be present in any algorithm that substantially samples the parameter space. In ASA, the influence of large dimensions becomes clearly focussed on the exponential of the power of k being $1/D$, as the annealing required to properly sample the space becomes prohibitively slow. So, if we cannot commit resources to properly sample the space ergodically, then for some systems perhaps the next best procedure would be to turn on quenching, whereby Q can become on the order of the size of number of dimensions. In some cases tried, a small system of only a few parameters can be used to determine some reasonable Program Options, and then these can be used for a much larger space scaled up to many parameters. This can work in some cases because of the independence of dimension of the generating functions.

7.3.31. OPTIONS->user_quench_param_scale

If QUENCH_PARAMETERS is TRUE, a pointer, OPTIONS->user_quench_param_scale, is used to adaptively set the scale of the temperature schedule. If this choice is elected, then OPTIONS->user_quench_param_scale[] must be initialized (named USER_OPTIONS->user_quench_param_scale[] in the user module), and values defined for each dimension. (If QUENCH_PARAMETERS is FALSE, then the pointer *user_quench_param_scale must be included in *OPTIONS, but it need not be initialized.) The default in the asa module is to assign the annealing value of 1 to all elements that might be defined otherwise. If values are selected greater than 1 using this Program Option, then quenching is enforced.

7.3.32. OPTIONS->QUENCH_COST[FALSE]

If QUENCH_COST is set to TRUE, the scale of the power of $1/D$ temperature schedule used for the acceptance function can be altered in a similar fashion to that described above when QUENCH_PARAMETERS is set to TRUE. However, note that this OPTION does not affect the annealing proof of ASA, and so this may used without damaging the statistical ergodicity of the algorithm. Even greater functional changes can be made using the Pre-Compile DEFINE_OPTIONS USER_COST_SCHEDULE.

7.3.33. OPTIONS->user_quench_cost_scale

If QUENCH_COST is TRUE, a pointer, OPTIONS->user_quench_cost_scale, is used to adaptively set the scale of the temperature schedule. If this choice is elected, then OPTIONS->user_quench_cost_scale[0] must be initialized (named USER_OPTIONS->user_quench_cost_scale[0] in the user module). (If QUENCH_COST is FALSE, then the pointer *user_quench_cost_scale must be included in *OPTIONS, but it need not be initialized.) The default in the asa module is to assign the annealing value of 1 to this element that might be defined otherwise.

OPTIONS->user_quench_cost_scale may be changed adaptively without affecting the ergodicity of the algorithm, within reason of course. This might be useful for some systems that require different approaches to the cost function in different ranges of its parameters. Note that increasing this parameter beyond its default of 1.0 can result in rapidly locking in the search to a small region of the cost function, severely handicapping the algorithm. On the contrary, you may find that slowing the cost temperature schedule, by setting this parameter to a value less than 1.0, may work better for your system.

7.3.34. OPTIONS->asa_data

If the Pre-Compile Option OPTIONAL_DATA[FALSE] is set to TRUE, an additional Program Option pointer, OPTIONS->asa_data, becomes available to to return additional information to the user module from the asa module. This information communicates with the asa module, and memory must be allocated for it in the user module. An example is given in user.c when SELF_OPTIMIZE is TRUE.

7.3.35. OPTIONS->asa_out_file

If you wish to have the printing from the asa module be sent to a file determined dynamically from the user module, set the Pre-Compile Printing Option USER_ASA_OUT[FALSE] to TRUE, and define the Program Option *asa_out_file in the user module. (This overrides any ASA_OUT settings.) An example of this use for multiple asa() runs is given in the user module.

7.3.36. OPTIONS->cost_schedule

If USER_COST_SCHEDULE[FALSE] is set to TRUE, then (*cost_schedule) () is created as a pointer to the function user_cost_schedule() in user.c, and to recur_user_cost_schedule() if SELF_OPTIMIZE is set to TRUE.

7.3.37. OPTIONS->reanneal_function

If USER_REANNEAL_FUNCTION[FALSE] is set to TRUE, then (*reanneal_function) () is created as a pointer to the function user_reanneal() in user.c, and to recur_user_reanneal() if SELF_OPTIMIZE is set to TRUE.

7.3.38. OPTIONS->n_accepted

If ASA_SAMPLE is set to TRUE, n_accepted contains the current number of points saved by the acceptance criteria. This can be used to monitor the sampling.

7.3.39. OPTIONS->bias_acceptance

If ASA_SAMPLE is TRUE, this is the bias of the current state from the Boltzmann acceptance test described above.

7.3.40. OPTIONS->bias_generated

If ASA_SAMPLE is TRUE, a pointer, OPTIONS->bias_generated, contains the the biases of the current state from the generating distributions of all active parameters, described above. OPTIONS->bias_generated[] must be initialized in the user module.

7.3.41. OPTIONS->average_weights

IF ASA_SAMPLE is TRUE, this is the average of the weight[] array holding the products of the inverse asa generating distributions of all active parameters.

For example, OPTIONS->n_accepted can be used to monitor changes in a new saved point in the cost function, and when OPTIONS->average_weights reaches a specified number (perhaps repeated several times), the cost function could return an invalid flag from the cost function to terminate the run. When the average_weights is very small, then additional sampled points likely will not substantially contribute more information.

7.3.42. OPTIONS->limit_weights

If ASA_SAMPLE is set to TRUE, limit_weights is a limit on the value of the average of the weight[] array holding the inverse asa generating distribution. When this lower limit is crossed, asa will no longer send sampling output to be printed out, although it still will be calculated. As the run progresses, this average will decrease until contributions from further sampling become relatively unimportant.

7.3.43. OPTIONS->gener_mov_avr

If ASA_PARALLEL is set to TRUE, gener_mov_avr determines the window of the moving average of sizes of parallel generated states required to find new best accepted states. A reasonable number for many problems is 3.

If and when OPTIONS->gener_mov_avr is set to 0, then OPTIONS->gener_block is not changed thereafter.

7.3.44. OPTIONS->gener_block_max

If ASA_PARALLEL is set to TRUE, gener_block is an initial block size of parallel generated states to calculate to determine a new best accepted state.

7.3.45. OPTIONS->gener_block_max

If ASA_PARALLEL is set to TRUE, gener_block_max is an initial maximum block size of parallel generated states to calculate to determine a new best accepted state. This can be changed adaptively during the run.

This can be useful if your parallel code assigns new processors "on the fly," to compensate for some cost functions being more CPU intensive, e.g., due to boundary conditions, etc. Then OPTIONS->gener_block_max may be larger than the number of physical processors, e.g., if OPTIONS->gener_block would call for such a size.

8. User Module

This module includes user.c, user.h, and asa_user.h. You may wish to combine them into one file, or you may wish to use the ASA module as one component of a library required for a large project.

8.1. int main(int argc, char **argv) | int asa_main()

In main(), set up your initializations and calling statements to asa. The files user.c and user.h provide a sample program, as well as a sample cost function for your convenience. If you do not intend to pass parameters into main, then you can just declare it as main() without the argc and argv arguments, deleting other references to argc and argv.

If ASA_LIB is set to TRUE, then asa_main() is used as a function call instead of main().

If SELF_OPTIMIZE is set to TRUE, then the first main()/asa_main() in user.c is closed off, and a different main()/asa_main() procedure in user.c is used.

8.2. void initialize_parameters(

```
double *cost_parameters,
double *parameter_lower_bound,
double *parameter_upper_bound,
double *cost_tangents,
double *cost_curvature,
ALLOC_INT *parameter_dimension,
int *parameter_int_real,
USER_DEFINES * USER_OPTIONS)
```

Before calling asa, the user must allocate storage and initialize some of the passed parameters. A sample procedure is provided as a template. In this procedure the user should allocate storage for the passed arrays and define the minimum and maximum values. Below is detailed all the parameters which must be initialized. If your arrays are of size 1, still use them as arrays as described in user.c. Alternatively, if you define ‘int user_flag’, then pass &user_flag.

As written above, these are the names used in the user module. All these parameters could be passed globally in the user module, e.g., by defining them in user.h instead of in main() in user.c, but since the asa module only passes local parameters to facilitate recursive use, this approach is taken here as well.

8.3. void recur_initialize_parameters(

double *recur_cost_parameters,
double *recur_parameter_lower_bound,
double *recur_parameter_upper_bound,
double *recur_cost_tangents,
double *recur_cost_curvature,
ALLOC_INT *recur_parameter_dimension,
int *recur_parameter_int_real,
USER_DEFINES * RECUR_USER_OPTIONS)

This procedure is used only if SELF_OPTIMIZE is TRUE, and is constructed similar to initialize_parameters().

8.4. double user_cost_function(

double *x,
double *parameter_minimum,
double *parameter_maximum,
double *tangents,
double *curvature,
ALLOC_INT *number_parameters,
int *parameter_type,
int *valid_state_generated_flag,
int *exit_status,
USER_DEFINES *OPTIONS)

8.4.1. user_cost_function

You can give any name to user_cost_function as long as you pass this name to asa; it is called cost_function in the user module. This function returns a real value which ASA will minimize. In cases where it seems that the ASA default parameters are not very efficient for your system, you might consider modifying the cost function being optimized. For example, if your actual cost function is of the form of an exponential to an exponential, you might do better using the logarithm of this as user_cost_function.

8.4.2. *x

x (called cost_parameters in the user module) is an array of doubles representing a set of parameters to evaluate.

8.4.3. double *parameter_minimum

8.4.4. double *parameter_maximum

These two arrays of doubles are passed. Since ASA works only on bounded search spaces, these arrays should contain the minimum and maximum values each parameter can attain. If you aren’t sure, try a factor of 10 or 100 times any reasonable values. The exponential temperature annealing schedule should quickly sharpen the search down to the most important region.

Passing the parameter bounds in the cost function permits some additional adaptive features during the search. For example, setting the lower bound equal to the upper bound will remove a parameter from consideration. In the user module these bounds are named parameter_lower_bound and parameter_upper_bound.

8.4.5. double *tangents

This array of doubles is passed. On return from asa this contains the first derivatives of the cost function with respect to its parameters. These can be useful for determining the value of your fit. In this implementation of ASA, the tangents are used to determine the relative reannealing among parameters.

8.4.6. double *curvature

This array of doubles is passed next. On return from asa, for real parameters, this contains the second derivatives of the cost function with respect to its parameters. These also can be useful for determining the value of your fit.

When the DEFINE_OPTIONS CURVATURE_0 option is set to TRUE the curvature calculations are bypassed. This can be useful for very large spaces.

8.4.7. ALLOC_INT *number_parameters

An integer containing the dimensionality of the state space is passed next (called parameter_dimension in the user module). (If you define ‘ALLOC_INT number_parameters’, pass &number_parameters.) The arrays x (representing cost_parameters), parameter_lower_bound, parameter_upper_bound, cost_tangents, and parameter_int_real (below) are to be of the size *number_parameters. The array curvature which may be of size the square of *number_parameters.

8.4.8. int *parameter_type

This integer array is passed next (passed as parameter_int_real in the user module). Each element of this array (each flag) can be: REAL_TYPE (-1) (indicating the parameter is a real value), INTEGER_TYPE (1) (indicating the parameter can take on only integer values), REAL_NO_REANNEAL (-2), or INTEGER_NO_REANNEAL (2). The latter two choices signify that no derivatives are to be taken with respect to these parameters. For example, this can be useful to exclude discontinuous functions from being reannealed.

8.4.9. *valid_state_generated_flag

valid_state_generated_flag is the address of an integer, named cost_flag in the user module. In user_cost_function(), *cost_flag should be set to FALSE (0) if the parameters violate a set of user defined constraints (e.g., as defined by a set of boundary conditions) or TRUE (1) if the parameters represent a valid state. If *cost_flag is returned to asa() as FALSE, no acceptance test will be attempted, and a new set of trial parameters will be generated.

If another algorithm suggests a way of incorporating constraints into the cost function, then this modified cost function can be used as well by ASA, or that algorithm might best be used a front-end to ASA.

If OPTIONS->USER_TANGENTS[FALSE] has been set to TRUE, then asa() expects the user to test the value of *valid_state_generated_flag that enters from asa(). If *valid_state_generated_flag enters with a value of FALSE, then the user is expected to calculate the tangents[] array (called cost_tangents[] in ASA_TEST in user.c) before exiting that particular evaluation of the cost function. An example is provided with the ASA_TEMPLATE for ASA_SAMPLE.

8.4.10. int *exit_status

The address of this integer is passed to asa. On return it contains the code for the reason asa exited.

NORMAL_EXIT = 0. Given the criteria set largely by the DEFINE_OPTIONS, the search has run its normal course.

P_TEMP_TOO_SMALL = 1. A parameter temperature was too small using the set criteria. Often this is an acceptable status code. You can omit this test by setting NO_PARAM_TEMP_TEST to TRUE as one of your Pre-Compile Options; then values of the parameter temperatures less than EPS_DOUBLE are set to EPS_DOUBLE.

C_TEMP_TOO_SMALL = 2. The cost temperature was too small using the set criteria. Often this is an acceptable status code. You can omit this test by setting NO_COST_TEMP_TEST to TRUE as one of your Pre-Compile Options; then a value of the cost temperature less than EPS_DOUBLE is set to EPS_DOUBLE.

COST_REPEATING = 3. The cost function value repeated a number of times using the set criteria. Often this is an acceptable status code.

TOO_MANY_INVALID_STATES = 4. Too many repetitive generated states were invalid using the set criteria. This is helpful when using *cost_flag, as discussed above, to include constraints.

An exit code of 9, defined by exit(9), has been set in case any of the calloc memory allocations fails. Note that just relying on such a simple summary given by *exit_status can be extremely deceptive, especially in highly nonlinear problems. It is *strongly* suggested that the user set ASA_PRINT=TRUE before any production runs. An examination of some periodic output of ASA can be essential to its proper use.

8.4.11. USER_DEFINES *OPTIONS

All Program Options are defined in this structure. Since Program Options are passed to asa and the cost function, these may be changed adaptively.

The Program Options also can be read in from a separate data file, asa_opt, permitting efficient tuning/debugging of these parameters without having to recompile the code. This option has been added to the asa module.

8.5. double recur_cost_function(

double *recur_cost_parameters,
double *recur_parameter_lower_bound,
double *recur_parameter_upper_bound,
double *recur_cost_tangents,
double *recur_cost_curvature,
int *recur_parameter_dimension,
int *recur_parameter_int_real,
int *recur_cost_flag,
int *recur_exit_code,
USER_DEFINES * RECUR_USER_OPTIONS)

This procedure is used only if SELF_OPTIMIZE is TRUE, and is constructed similar to cost_function().

8.6. double user_random_generator()

A random number generator function must be selected. It may be as simple as one of the UNIX® random number generators (e.g. drand48), or may be user defined, but it should return a real value within [0,1) and not take any parameters. A good random number generator, randflt, and its auxiliary routines are provided with the code in the user module.

8.7. void initialize_rng()

Most random number generators should be "warmed-up" by calling a set of dummy random numbers.

8.8. double user_cost_schedule(
double test_temperature,
USER_DEFINES * USER_OPTIONS);

If USER_COST_SCHEDULE[FALSE] is set to TRUE, then this function must define how the new cost temperature is calculated during the acceptance test. The default is to return test_temperature. For example, if you sense that the search is spending too much time in local minima at some stage of search, e.g., dependent on information gathered in USER_OPTIONS, then you might return the square root of test_temperature, or some other function, to slow down the sharpening of the cost function acceptance test.

8.9. double recur_user_cost_schedule(
double test_temperature,
USER_DEFINES * RECUR_USER_OPTIONS);

If USER_COST_SCHEDULE[FALSE] and SELF_OPTIMIZE[FALSE] both are set to TRUE, then this function must define how the new cost temperature is calculated during the acceptance test. As discussed above for user_cost_schedule(), you may modify the default value of test_temperature returned by this function, e.g., dependent on information gathered in RECUR_USER_OPTIONS.

8.10. double user_reanneal(
double current_temp,
double tangent,
double max_tangent,
USER_DEFINES * USER_OPTIONS);

If USER_REANNEAL_FUNCTION[FALSE] is set to TRUE, then this function must define how the new temperature is calculated during reannealing.

8.11. double recur_user_reanneal(
double current_temp,
double tangent,
double max_tangent,
USER_DEFINES * RECUR_USER_OPTIONS);

If USER_REANNEAL_FUNCTION[FALSE] and SELF_OPTIMIZE[FALSE] both are set to TRUE, then this function must define how the new temperature is calculated during reannealing.

8.12. final_cost = asa(
cost_function,
randflt,
cost_parameters,
parameter_lower_bound,
parameter_upper_bound,
cost_tangents,
cost_curvature,
parameter_dimension,
parameter_int_real,
cost_flag,
exit_code,
USER_OPTIONS);

This is the form of the call to asa from user.c. A double is returned to the calling program as whatever it is named by the user, e.g., final_cost. It will be the minimum cost value found by asa.

8.13. double asa(

double (*user_cost_function) (
double *, double *, double *, double *, double *,
ALLOC_INT *, int *, int *, int *, USER_DEFINES *),
double (*user_random_generator) (void),
double *parameter_initial_final,
double *parameter_minimum,
double *parameter_maximum,
double *tangents,
double *curvature,
ALLOC_INT *number_parameters,
int *parameter_type,
int *valid_state_generated_flag,
int *exit_status,
USER_DEFINES * OPTIONS)

This is how asa is defined in the ASA module, contained in asa.c and asa_user.h. All but the user_cost_function, user_random_generator, and parameter_initial_final parameters have been described above as they also are passed by user_cost_function().

8.13.1. double (*user_cost_function) ()

The parameter (*user_cost_function*) () is a pointer to the cost function that you defined in your user module.

8.13.2. double (*user_random_generator) ()

A pointer to the random number generator function, defined in the user module, must be passed next.

8.13.3. double *parameter_initial_final

An array of doubles is passed (passed as cost_parameters in the user module). Initially, this array holds the set of starting parameters which should satisfy any constraints or boundary conditions. Upon return from the asa procedure, the array will contain the best set of parameters found by asa to minimize the user's cost function. Experience shows that any guesses within the acceptable ranges should suffice, since initially the system is at high annealing temperature and ASA samples the breadth of the ranges. The default is to have asa generate a set of initial parameters satisfying the user's constraints. This can be overridden using USER_INITIAL_PARAMETERS=TRUE, to have the user's initial guess be the first generated set of parameters.

8.14. void print_time(char *message, FILE * ptr_out)

As a convenience, this subroutine and its auxiliary routine aux_print_time are provided in asa.c to keep track of the time spent during optimization. Templates in the code are provided to use these routines to print to output from both the asa and user modules. These routines can give some compilation problems on some platforms, and may be bypassed using one of the DEFINE_OPTIONS. It takes as its parameters a string which will be printed and the pointer to file to where the printout is directed. An example is given in user_cost_function to illustrate how print_time may be called periodically every set number of calls by defining PRINT_FREQUENCY in user.h. See the NOTES file for changes in these routines that may be required on some particular systems.

8.15. void sample(FILE * ptr_out, FILE * ptr_asa)

When ASA_TEMPLATE and ASA_SAMPLE are set to true, using data collected in the ASA_OUT file, this routine illustrates how to extract the data stored in the ASA_OUT file and print it to the user module.

9. Bug Reports

I volunteer my time to make every reasonable effort to maintain only current versions of the asa module, to permit the code to compile without "error," not necessarily without compiler "warnings." The user module is offered only as a guide to accessing the asa module. The NOTES file will contain updates for some standard machines. I welcome your bug reports and constructive critiques regarding this code. If you are having problems, it might help if you enclose relevant portions your ASA_OUT file.

"Flames" will be rapidly quenched.

10. References

[1] L. Ingber, "Adaptive Simulated Annealing (ASA)," [ftp.alumni.caltech.edu: /pub/ingber/ASA−shar, ASA−shar.Z, ASA.tar.Z, ASA.tar.gz, ASA.zip], Lester Ingber Research, McLean, VA (1993).

[2] L. Ingber, "Very fast simulated re-annealing," *Mathl. Comput. Modelling,* 12, pp. 967-973 (1989).

[3] L. Ingber, H. Fujio, and M.F. Wehner, "Mathematical comparison of combat computer models to exercise data," *Mathl. Comput. Modelling,* 15, pp. 65-90 (1991).

[4] L. Ingber, "Statistical mechanical aids to calculating term structure models," *Phys. Rev. A,* 42, pp. 7057-7064 (1990).

[5] L. Ingber, "Statistical mechanics of neocortical interactions: A scaling paradigm applied to electroencephalography," *Phys. Rev. A,* 44, pp. 4017-4060 (1991).

[6] L. Ingber, "Generic mesoscopic neural networks based on statistical mechanics of neocortical interactions," *Phys. Rev. A,* 45, pp. R2183-R2186 (1992).

[7] L. Ingber and B. Rosen, "Very Fast Simulated Reannealing (VFSR)," [ringer.cs.utsa.edu: /pub/rosen/vfsr.Z], University of Texas, San Antonio, TX (1992).

[8] L. Ingber, "Simulated annealing: Practice versus theory," *Mathl. Comput. Modelling,* 18, pp. 29-57 (1993).

[9] M. Wofsey, "Technology: Shortcut tests validity of complicated formulas," *The Wall Street Journal,* CCXXII, p. B1 (24 September 1993).

Table of Contents

$Id: readme.ms,v 4.2 1994/10/23 23:35:08 ingber Exp ingber $