

ADAPTIVE SIMULATED ANNEALING (ASA) ©

Lester Ingber

ingber@ingber.com
ingber@alumni.caltech.edu

Adaptive Simulated Annealing (ASA) is a C-language code developed to statistically find the best global fit of a nonlinear constrained non-convex cost-function over a D -dimensional space. This algorithm permits an annealing schedule for “temperature” T decreasing exponentially in annealing-time k , $T = T_0 \exp(-ck^{1/D})$. The introduction of re-annealing also permits adaptation to changing sensitivities in the multi-dimensional parameter-space. This annealing schedule is faster than fast Cauchy annealing, where $T = T_0/k$, and much faster than Boltzmann annealing, where $T = T_0/\ln k$. ASA has over 100 OPTIONS to provide robust tuning over many classes of nonlinear stochastic systems.

```
/******  
* Adaptive Simulated Annealing (ASA)  
* Lester Ingber <ingber@ingber.com>  
* Copyright © 1987-2012 Lester Ingber. All Rights Reserved.  
* The ASA-LICENSE file must be included with ASA code.  
*****/
```

\$Id: ASA-README.ms,v 28.12 2012/07/02 23:58:48 ingber Exp ingber \$

Table of Contents

1.	ASA–LICENSE	1
2.	Lester Ingber Research Terms of Use	1
3.	Documentation	2
3.1.	Table of Contents/Index	2
3.2.	ASA–README.ms and ASA–README	2
3.2.1.	ASA–README.txt and ASA–README+.txt	2
3.2.2.	asa.[13nl] Manpage	2
3.2.3.	ASA–README.ps and ASA–README.pdf	2
3.2.4.	ASA–README.html	2
3.3.	Additional Documentation	2
3.4.	Use of Documentation for Tuning	3
4.	Availability of ASA Code	4
4.1.	ingber.com	4
4.2.	Electronic Mail	4
5.	Background	4
5.1.	Context	4
5.2.	Outline of ASA Algorithm	6
5.2.1.	Simulated Annealing is NOT Simulated Quenching	6
5.2.2.	Generating Probability Density Function	6
5.2.3.	Acceptance Probability Density Function	6
5.2.4.	Reannealing Temperature Schedule	7
5.3.	Efficiency Versus Necessity	7
6.	Outline of Use	8
7.	ASA–Makefile/Compilation Procedures	8
7.1.	DLL ASA–Makefile	9
8.	Generic ASA Interfaces	9
9.	User Options	10
9.1.	Pre-Compile DEFINE_OPTIONS	10
9.1.1.	USER_COST_FUNCTION=cost_function	11
9.1.2.	RECUR_USER_COST_FUNCTION=recur_cost_function	11
9.1.3.	OPTIONS_FILE=TRUE	11
9.1.4.	OPTIONS_FILE_DATA=TRUE	11
9.1.5.	RECUR_OPTIONS_FILE=FALSE	11
9.1.6.	RECUR_OPTIONS_FILE_DATA=FALSE	11
9.1.7.	COST_FILE=TRUE	11
9.1.8.	ASA_LIB=FALSE	12
9.1.9.	HAVE_ANSI=TRUE	12
9.1.10.	IO_PROTOTYPES=FALSE	12
9.1.11.	TIME_CALC=FALSE	12
9.1.12.	TIME_STD=FALSE	12
9.1.13.	TIME_GETRUSAGE=TRUE	12
9.1.14.	INT_LONG=TRUE	12
9.1.15.	INT_ALLOC=FALSE	13
9.1.16.	SMALL_FLOAT=1.0E-18	13
9.1.17.	MIN_DOUBLE=SMALL_FLOAT	13
9.1.18.	MAX_DOUBLE=1.0/SMALL_FLOAT	13
9.1.19.	EPS_DOUBLE=SMALL_FLOAT	13
9.1.20.	CHECK_EXPONENT=FALSE	13
9.1.21.	NO_PARAM_TEMP_TEST=FALSE	13

9.1.22.	NO_COST_TEMP_TEST=FALSE	13
9.1.23.	SELF_OPTIMIZE=FALSE	14
9.1.24.	ASA_TEST=FALSE	14
9.1.25.	ASA_TEST_POINT=FALSE	14
9.1.26.	MY_TEMPLATE=TRUE	14
9.1.27.	USER_INITIAL_COST_TEMP=FALSE	14
9.1.28.	RATIO_TEMPERATURE_SCALES=FALSE	15
9.1.29.	USER_INITIAL_PARAMETERS_TEMPS=FALSE	15
9.1.30.	DELTA_PARAMETERS=FALSE	15
9.1.31.	QUENCH_PARAMETERS=FALSE	15
9.1.32.	QUENCH_COST=FALSE	16
9.1.33.	QUENCH_PARAMETERS_SCALE=TRUE	16
9.1.34.	QUENCH_COST_SCALE=TRUE	16
9.1.35.	ASA_TEMPLATE=FALSE	17
9.1.36.	OPTIONAL_DATA_DBL=FALSE	17
9.1.37.	OPTIONAL_DATA_INT=FALSE	17
9.1.38.	OPTIONAL_DATA_PTR=FALSE	17
9.1.39.	OPTIONAL_PTR_TYPE=USER_TYPE	18
9.1.40.	USER_COST_SCHEDULE=FALSE	18
9.1.41.	USER_ACCEPT_ASYMP_EXP=FALSE	18
9.1.42.	USER_ACCEPT_THRESHOLD=FALSE	18
9.1.43.	USER_ACCEPTANCE_TEST=FALSE	18
9.1.44.	USER_GENERATING_FUNCTION=FALSE	19
9.1.45.	USER_REANNEAL_COST=FALSE	19
9.1.46.	USER_REANNEAL_PARAMETERS=FALSE	20
9.1.47.	MAXIMUM_REANNEAL_INDEX=50000	20
9.1.48.	REANNEAL_SCALE=10.0	20
9.1.49.	ASA_SAMPLE=FALSE	21
9.1.50.	ADAPTIVE_OPTIONS=FALSE	21
9.1.51.	ASA_QUEUE=FALSE	21
9.1.52.	ASA_RESOLUTION=FALSE	22
9.1.53.	ASA_FUZZY=FALSE	22
9.1.54.	FITLOC=FALSE	22
9.1.55.	FITLOC_ROUND=TRUE	23
9.1.56.	FITLOC_PRINT=TRUE	23
9.1.57.	MULTI_MIN=FALSE	23
9.1.58.	ASA_PARALLEL=FALSE	23
9.1.59.	FDLIBM_POW=FALSE	24
9.1.60.	FDLIBM_LOG=FALSE	24
9.1.61.	FDLIBM_EXP=FALSE	24
9.2.	Printing DEFINE_OPTIONS	24
9.2.1.	USER_OUT=\"asa_usr_out\"	24
9.2.2.	INCL_STDOUT=TRUE	24
9.2.3.	ASA_PRINT=TRUE	24
9.2.4.	ASA_OUT=\"asa_out\"	24
9.2.5.	USER_ASA_OUT=FALSE	25
9.2.6.	ASA_PRINT_INTERMED=TRUE	25
9.2.7.	ASA_PRINT_MORE=FALSE	25
9.2.8.	G_FIELD=12 & G_PRECISION=7	25
9.2.9.	ASA_SAVE=FALSE	25
9.2.10.	ASA_SAVE_OPT=FALSE	26
9.2.11.	ASA_SAVE_BACKUP=FALSE	26
9.2.12.	ASA_PIPE=FALSE	26
9.2.13.	ASA_PIPE_FILE=FALSE	26

9.2.14.	ASA_EXIT_ANYTIME=FALSE	26
9.2.15.	SYSTEM_CALL=TRUE	27
9.3.	Program OPTIONS	27
9.3.1.	OPTIONS->Limit_Acceptances[10000]	30
9.3.2.	OPTIONS->Limit_Generated[99999]	30
9.3.3.	OPTIONS->Limit_Invalid_Generated_States[1000]	30
9.3.4.	OPTIONS->Accepted_To_Generated_Ratio[1.0E-6]	30
9.3.5.	OPTIONS->Cost_Precision[1.0E-18]	30
9.3.6.	OPTIONS->Maximum_Cost_Repeat[5]	30
9.3.7.	OPTIONS->Number_Cost_Samples[5]	30
9.3.8.	OPTIONS->Temperature_Ratio_Scale[1.0E-5]	31
9.3.9.	OPTIONS->Cost_Parameter_Scale_Ratio[1.0]	31
9.3.10.	OPTIONS->Temperature_Anneal_Scale[100.0]	31
9.3.11.	OPTIONS->User_Cost_Temperature	32
9.3.12.	OPTIONS->Include_Integer_Parameters[FALSE]	32
9.3.13.	OPTIONS->User_Initial_Parameters[FALSE]	32
9.3.14.	OPTIONS->Sequential_Parameters[-1]	32
9.3.15.	OPTIONS->Initial_Parameter_Temperature[1.0]	32
9.3.16.	OPTIONS->User_Temperature_Ratio	32
9.3.17.	OPTIONS->User_Parameter_Temperature	32
9.3.18.	OPTIONS->Acceptance_Frequency_Modulus[100]	32
9.3.19.	OPTIONS->Generated_Frequency_Modulus[10000]	33
9.3.20.	OPTIONS->Reanneal_Cost[1]	33
9.3.21.	OPTIONS->Reanneal_Parameters[TRUE]	33
9.3.22.	OPTIONS->Delta_X[0.001]	33
9.3.23.	OPTIONS->User_Delta_Parameter	34
9.3.24.	OPTIONS->User_Tangents[FALSE]	34
9.3.25.	OPTIONS->Curvature_0[FALSE]	34
9.3.26.	OPTIONS->User_Quench_Param_Scale	34
9.3.27.	OPTIONS->User_Quench_Cost_Scale	34
9.3.28.	OPTIONS->N_Accepted	35
9.3.29.	OPTIONS->N_Generated	35
9.3.30.	OPTIONS->Locate_Cost	35
9.3.31.	OPTIONS->Immediate_Exit[FALSE]	35
9.3.32.	OPTIONS->Best_Cost	36
9.3.33.	OPTIONS->Best_Parameters	36
9.3.34.	OPTIONS->Last_Cost	36
9.3.35.	OPTIONS->Last_Parameters	36
9.3.36.	OPTIONS->Asa_Data_Dim_Dbl	36
9.3.37.	OPTIONS->Asa_Data_Dbl	36
9.3.38.	OPTIONS->Asa_Data_Dim_Int	36
9.3.39.	OPTIONS->Asa_Data_Int	37
9.3.40.	OPTIONS->Asa_Data_Dim_Ptr	37
9.3.41.	OPTIONS->Asa_Data_Ptr	37
9.3.42.	OPTIONS->Asa_Out_File	37
9.3.43.	OPTIONS->Cost_Schedule	38
9.3.44.	OPTIONS->Asymp_Exp_Param	38
9.3.45.	OPTIONS->Acceptance_Test	38
9.3.46.	OPTIONS->User_Acceptance_Flag	38
9.3.47.	OPTIONS->Cost_Acceptance_Flag	38
9.3.48.	OPTIONS->Cost_Temp_Curr	38
9.3.49.	OPTIONS->Cost_Temp_Init	38
9.3.50.	OPTIONS->Cost_Temp_Scale	38
9.3.51.	OPTIONS->Prob_Bias	39

9.3.52.	OPTIONS->Random_Seed	39
9.3.53.	OPTIONS->Generating_Distrib	39
9.3.54.	OPTIONS->Reanneal_Cost_Function	39
9.3.55.	OPTIONS->Reanneal_Params_Function	39
9.3.56.	OPTIONS->Bias_Acceptance	39
9.3.57.	OPTIONS->Bias_Generated	39
9.3.58.	OPTIONS->Average_Weights	39
9.3.59.	OPTIONS->Limit_Weights	39
9.3.60.	OPTIONS->Queue_Size	40
9.3.61.	OPTIONS->Queue_Resolution	40
9.3.62.	OPTIONS->Coarse_Resolution	40
9.3.63.	OPTIONS->Fit_Local	40
9.3.64.	OPTIONS->Iter_Max	40
9.3.65.	OPTIONS->Penalty	40
9.3.66.	OPTIONS->Multi_Number	40
9.3.67.	OPTIONS->Multi_Cost	40
9.3.68.	OPTIONS->Multi_Params	41
9.3.69.	OPTIONS->Multi_Grid	41
9.3.70.	OPTIONS->Multi_Specify	41
9.3.71.	OPTIONS->Gener_Mov_Avr	41
9.3.72.	OPTIONS->Gener_Block	41
9.3.73.	OPTIONS->Gener_Block_Max	41
9.3.74.	OPTIONS->Random_Array_Dim	41
9.3.75.	OPTIONS->Random_Array	42
9.3.76.	OPTIONS->Asa_Recursive_Level	42
10.	User Module	42
10.1.	int main(int argc, char **argv)	42
10.2.	int asa_main(.	42
10.3.	int initialize_parameters(.	42
10.4.	int recur_initialize_parameters(.	43
10.5.	double cost_function(.	43
10.5.1.	cost_function	44
10.5.2.	*x	44
10.5.3.	double *parameter_lower_bound	44
10.5.4.	double *parameter_upper_bound	44
10.5.5.	double *cost_tangents	44
10.5.6.	double *cost_curvature	44
10.5.7.	ALLOC_INT *parameter_dimension	45
10.5.8.	int *parameter_int_real	45
10.5.9.	*cost_flag	45
10.5.10.	int *exit_code	45
10.5.11.	USER_DEFINES *OPTIONS	46
10.6.	double recur_cost_function(.	46
10.7.	double user_random_generator(.	47
10.8.	LONG_INT asa_seed(.	47
10.9.	double user_cost_schedule(.	47
10.10.	double recur_user_cost_schedule(.	47
10.11.	void user_acceptance_test(.	48
10.12.	void recur_user_acceptance_test(.	48
10.13.	double user_generating_distrib(.	48
10.14.	double recur_user_generating_distrib(.	48
10.15.	int user_reanneal_cost(.	49
10.16.	int recur_user_reanneal_cost(.	49
10.17.	double user_reanneal_params(.	49

10.18.	double recur_user_reanneal_params(.	49
10.19.	final_cost = asa(.	49
10.20.	double asa(.	50
10.20.1.	double (*user_cost_function) ()	50
10.20.2.	double (*user_random_generator) ()	50
10.20.3.	LONG_INT *rand_seed	50
10.20.4.	double *parameter_initial_final	50
10.21.	void print_time(char *message, FILE * ptr_out)	51
10.22.	void sample(FILE * ptr_out, FILE * ptr_asa)	51
10.23.	void adaptive_options(USER_DEFINES * USER_OPTIONS)	51
11.	Bug Reports and Help With ASA	51
12.	References	52

1. ASA–LICENSE

This Adaptive Simulated Annealing (ASA) code is being made available under conditions specified in the ASA–LICENSE file that comes with this code, and is owned by Lester Ingber[1]. Reference is properly given to the internet archive that first published the code. Please read the copy of the public ASA–LICENSE contained in the ASA directory. Its intent is to protect the integrity of the algorithm, promote widespread usage, and require reference to current source code. The ASA–LICENSE is so short it is repeated here:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

CONDITIONS

1. Redistributions of ASA source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must contain the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All modifications to the source code must be clearly marked as such. Binary redistributions based on modified source code must be clearly marked as modified versions in the documentation and/or other materials provided with the distribution.
4. Notice must be given of the location of the availability of the unmodified current source code, e.g.,
<http://www.ingber.com/>
in the documentation and/or other materials provided with the distribution or publication. ASA also is listed at
<http://alumni.caltech.edu/~ingber>
<http://asa-caltech.sourceforge.net>
<https://code.google.com/p/adaptive-simulated-annealing>
5. The name of Lester Ingber may not be used to endorse or promote products derived from this software without specific prior written permission.

DISCLAIMER

This software is provided by Lester Ingber and contributors “as is” and any expressed or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall Lester Ingber or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

2. Lester Ingber Research Terms of Use

Lester Ingber Research (LIR) develops projects in several areas of expertise documented in the [ingber.com](http://www.ingber.com) InterNet archive, e.g., this ASA code. Information on terms of use is in the file http://www.ingber.com/ingber_terms.html under WWW.

There is no charge for downloading and using codes or files in the [ingber.com](http://www.ingber.com) archive. In general, I have retained all rights such as copyrights to these codes and files, but they may be freely used by any person or group independent of affiliations, e.g., independent of academic or commercial affiliation.

Limited help assisting people with queries on my codes and papers is available only by electronic mail correspondence. Sorry, I cannot mail out hardcopies of code or papers.

3. Documentation

3.1. Table of Contents/Index

A compilation of the three levels of headers with their page numbers may be used as a Table of Contents placed after the first title page (as is done for ASA-README.ps, ASA-README.pdf and ASA-README.html below), or left at the end for quick reference (as is done for ASA-README.txt below).

3.2. ASA-README.ms and ASA-README

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

3.2.1. ASA-README.txt and ASA-README+.txt

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

ASA-README+.txt is ASA-README.txt without any filters to strip off underlining and bold enhancements.

3.2.2. asa.[13nl] Manpage

The ASA-README.txt or ASA-README+.txt file can be copied to a file named asa.[13], 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 ASA-README.txt or ASA-README+.txt 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 this asa.[13nl] through man may strip out additional “back-slash” characters, leading to missing words or unintended formatting. If such a file looks strange, compare it to the raw ASA-README.ms file to determine the true intended content. You likely can avoid some further undesirable formatting by man by placing ‘.nf’ on the first line of this file.

3.2.3. ASA-README.ps and ASA-README.pdf

ASA-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.

A PDF version ASA-README.pdf is prepared from ASA-README.ps.

3.2.4. ASA-README.html

ASA-README.html is an HTML version which enables easier access to subsections of this file. Cross-references have been kept local to this file, so you may view it under a local browser if you download the HTML source file.

The background image file asa_back.jpg referenced in ASA-README.html can be downloaded as http://www.ingber.com/asa_back.jpg from the ASA archive.

3.3. Additional Documentation

ASA-CHANGES is a terse record of major changes made in the ASA code. It has three sections, CHANGES, CONTRIBUTORS, and VERSION DATES.

ASA-NOTES is a collection of recommended enhancements, modifications, comments, caveats, etc., that might be of interest. There is a CONTENTS of sections headers that can be used to search on topics in your browser or editor.

There are three files in the ASA archive that should be considered as appendices to the ASA-NOTES file: http://www.ingber.com/asa_contrib.txt, http://www.ingber.com/asa_examples.txt, and http://www.ingber.com/asa_papers.html under WWW.

The file http://www.ingber.com/asa_contrib.txt in the ASA archive contains some code contributed by users. For example, references are giving to MATLAB and RLaB gateway routines to ASA (which include many important parts of the ASA user module), and to function support for ASA_PARALLEL. There is a CONTENTS of sections headers that can be used to search on topics in your browser or editor. In this file I have included the first 1987 VFSR code, the precursor to the ASA code, as used on a specific project, including the RATFOR vfsr.r and vfsr_com.r code, subsequently compiled into FORTRAN to run on a Lawrence Livermore supercomputer. I do not support this old RATFOR code.

The file http://www.ingber.com/asa_examples.txt in the ASA archive contains some example problems using ASA and some code to use in ASA. There is a CONTENTS of sections headers that can be used to search on topics in your browser or editor. This file contains some problems optimized using ASA, which can provide immediate examples on how you can optimize your own problem.

The file http://www.ingber.com/asa_papers.html is an addendum to the ASA-NOTES file in the ASA code, containing references to some patents and papers using ASA or its precursor VFSR.

The file [asa_new.txt](#) in the ASA archive is a list of major changes in ASA. The files ASA-README.txt, ASA-README.ps and ASA-README.pdf included with the code also are available independently as <http://www.ingber.com/ASA-README.txt>, <http://www.ingber.com/ASA-README.ps.gz>, <http://www.ingber.com/ASA-README.html>, <http://www.ingber.com/ASA-README.pdf>.

There is a set of ASA_TEMPLATE's available in the ASA-Makefile and in the user module (some also in the asa module) to illustrate use of particular OPTIONS, as listed under ASA_TEMPLATE below. You can search on these ASA_TEMPLATE's in your browser or editor to see how these are implemented. Note that some OPTIONS require your input, as described below, and code may fail until you add your own code. Once you have determined the most common set of DEFINE_OPTIONS you are likely to use, you might place these in your own TEMPLATE at the top of [asa_usr_asa.h](#) at the location specified, e.g.,

```
#if MY_TEMPLATE          /* MY_TEMPLATE_asa_user */
/* you can add your own set of #define here */
#define ... TRUE
#define ... 100
#endif
```

See http://www.ingber.com/utls_file_formats.txt for some links to information on gzip, PostScript, PDF, tar, and shar utilities. The file [utls_code.txt](#) in that directory gives short statements describing these files, which may be accessed as http://www.ingber.com/utls_code.html under WWW.

3.4. Use of Documentation for Tuning

I am often asked how I can help someone tune their system, and they send me their cost function or a list of the ASA OPTIONS they are using. Most often, the best help I can provide is based on my own experience that nonlinear systems typically are non-typical. In practice, that means that trying to figure out the nature of the cost function under sampling in order to tune ASA (or likely to similarly tune a hard problem under any sampling algorithm), by examining just the cost function, likely will not be as productive as generating more intermediate printout, e.g., setting ASA_PRINT_MORE to TRUE, and looking at this output as a “grey box” of insight into your optimization problem. Larger files with more information is provided by setting ASA_PIPE_FILE to TRUE. Treat the output of ASA as a simulation in the ASA parameter space, which usually is quite a different space than the variable space of your system.

For example, you should be able to see where and how your solution might be getting stuck in a local minima for a very long time, or where the last saved state is still fluctuating across a wide portion of your state space. These observations should suggest how you might try speeding up or slowing down annealing/quenching (see more below on modifying ASA with QUENCH OPTIONS) of the parameter space and/or tightening or loosening the acceptance criteria at different stages by modifying the OPTIONS, e.g., starting with the OPTIONS that can be easily adjusted using the [asa_opt](#) file.

The ASA-NOTES file that comes with the ASA code provides some guidelines for tuning that may provide some insights, especially the section Some Tuning Guidelines. An especially important guide is

to examine the output of ASA at several stages of sampling, to see if changes in parameter and temperatures are reasonably correlated to changes in the cost function. Examples of useful OPTIONS and code that often give quick changes in tuning in some problems are in the file http://www.ingber.com/asa_examples.txt under WWW. Some of the reprint files of published papers in the ingber.com provide other examples in harder systems, and perhaps you might find some examples of harder systems using ASA similar to your own in http://www.ingber.com/asa_papers.html under WWW. This is the best way to add some Art to the Science of annealing.

While the upside of using ASA is that it has many OPTIONS available for tuning, derived in large part from feedback from many users over many years, making it extremely robust across many systems, the downside is that the learning curve can be steep especially if the default settings or simple tweaking in `asa_opt` do not work very well for your particular system, and you then must turn to using more ASA OPTIONS. Most of these OPTIONS have useful guides in the ASA_TEMPLATES in `asa_usr.c`, as well as being documented here. If you really get stuck, you may consider working with someone else who already has climbed this learning curve and whose experience might offer quick help.

Tuning is an essential aspect of any sampling algorithm if it is to be applied to many classes of systems. It just doesn't make sense to compare sampling algorithms unless you are prepared to properly tune each algorithm to each system being optimized or sampled.

4. Availability of ASA Code

4.1. ingber.com

The latest Adaptive Simulated Annealing (ASA) code and some related papers can be accessed from the home page <http://www.ingber.com/> under WWW.

The home page <http://www.ingber.com/> under WWW, and the ASCII version `00index.txt`, contain an index of the other files.

The latest version of ASA, `ASA-x.y` (x and y are version numbers), can be obtained in two formats: <http://www.ingber.com/ASA.tar.gz> and <http://www.ingber.com/ASA.zip>. The tar'd version is compressed in gzip format, and `ASA.tar.gz`. In the zip'd version, `ASA.zip`, all files have been processed for DOS format.

Patches `ASA-diff-x1.y1-x2.y2` up to the present version can be prepared if a good case for doing so is presented, e.g. to facilitate updating your own modified codes. These may be concatenated as required before applying. If you require a specific patch, contact ingber@ingber.com.

4.2. Electronic Mail

If you do not have WWW, get the Guide to Offline Internet Access, returned by sending an e-mail to `mail-server@rtfm.mit.edu` with only the words "send usenet/news.answers/internet-services/access-via-email" in the body of the message. The guide gives information on using e-mail to access just about all InterNet information and documents. You will receive the information in `utils_access-via-email.txt` in the ASA archive.

5. Background

5.1. Context

Too often the management of complex systems is ill-served by not utilizing the best tools available. For example, requirements set by decision-makers often are not formulated in the same language as constructs formulated by powerful mathematical formalisms, and so the products of analyses are not properly or maximally utilized, even if and when they come close to faithfully representing the powerful intuitions they are supposed to model. In turn, even powerful mathematical constructs are ill-served, especially when dealing with multivariate nonlinear complex systems, when these formalisms are butchered into quasi-linear approximations to satisfy constraints of numerical algorithms familiar to particular analysts, but which tend to destroy the power of the intuitive constructs developed by

decision-makers.

In order to deal with fitting parameters or exploring sensitivities of variables, as models of systems have become more sophisticated in describing complex behavior, it has become increasingly important to retain and respect the nonlinearities inherent in these models, as they are indeed present in the complex systems they model. ASA can help to handle these fits of nonlinear models of real-world data.

It helps to visualize the problems presented by such complex systems as a geographical terrain. For example, consider a mountain range, with two “parameters,” e.g., along the North–South and East–West directions. We wish to find the lowest valley in this terrain. ASA approaches this problem similar to using a bouncing ball that can bounce over mountains from valley to valley. We start at a high “temperature,” where the temperature is an ASA parameter that mimics the effect of a fast moving particle in a hot object like a hot molten metal, thereby permitting the ball to make very high bounces and being able to bounce over any mountain to access any valley, given enough bounces. As the temperature is made relatively colder, the ball cannot bounce so high, and it also can settle to become trapped in relatively smaller ranges of valleys.

We imagine that our mountain range is aptly described by a “cost function.” We define probability distributions of the two directional parameters, called generating distributions since they generate possible valleys or states we are to explore. We define another distribution, called the acceptance distribution, which depends on the difference of cost functions of the present generated valley we are to explore and the last saved lowest valley. The acceptance distribution is used to decide whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on temperatures.

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]. The first published use of VFSR for a complex systems was in combat analysis, using a model of combat first developed in 1986, and then applied to exercise and simulation data in a series of papers that spanned 1988-1993[3]. 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 ASA–NOTES file contains some comments on interfacing ASA with FORTRAN codes.)

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 a “copyleft” GNU General Public License (GPL)[4], and copies were placed in NETLIB and STATLIB.

Beginning in January 93, many adaptive features were developed, largely in response to users’ requests, leading to this ASA code. Until 1996, ASA was located at <http://www.alumni.caltech.edu/~ingber/>. Pointers were placed in NETLIB and STATLIB to this location. ASA versions 1.1 through 5.13 retained the GPL, but subsequent versions through this one have incorporated a simpler ASA–LICENSE, based in part on a University of California license, that protects the integrity of the algorithm, promotes widespread usage, and requires reference to current source code. As the archive grew, more room and maintenance was required, and in February 1996 the site was moved to the present ingber.com location. Pointers were placed in the Caltech site to this location. <http://alumni.caltech.edu/~ingber> is the mirror homepage for the ASA site.

Mirrors of ASA (which may not always be current) are listed at <http://asa-caltech.sourceforge.net> (<http://asa-caltech.sf.net>) since January 2007, and at <https://code.google.com/p/adaptive-simulated-annealing> since in March 2001. FTP access to all archives was discontinued in July 2010.

In February 2011, beginning with Version 28.1, the License was simplified and brought into the framework described on http://en.wikipedia.org/wiki/BSD_License as the New BSD License.

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 -- see more below on modifying ASA with QUENCH OPTIONS)[5]. 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]. ASA is now used world-wide across many disciplines[7,8,9,10], including specific disciplines such as finance[11,12,13,14,15,16,17], neuroscience[18,19,20,21,22,23], and combat analyses[24,25,26,27,28,29,30]. Some papers illustrate the combined use of ASA for optimization and sampling[31,32,33,34,35,36]. The

http://www.ingber.com/asa_papers.html file in the ASA archive contains references to some patents and papers using ASA and VFSR.

5.2. Outline of ASA Algorithm

Details of the ASA algorithm are best obtained from the published papers. There are three parts to its basic structure: Generating Probability Density Functions, Acceptance Probability Density Functions, and Temperature Schedules.

5.2.1. Simulated Annealing is NOT Simulated Quenching

As an aside, it is very important to understand that Simulated Annealing (SA) is NOT Simulated Quenching (SQ). The issue is that SA is not just one algorithm -- it is a class of approaches to importance sampling.

Different SA algorithms may have different probability distributions they use to generate new states for consideration by the acceptance criteria. The often-cited SA proof of convergence depends on this generating probability distribution, which leads to different temperature schedules for different distributions.

If a schedule-1 derived for one SA-1 is faster than a schedule-2 derived for a different SA-2, but one uses schedule-1 with the SA-2 algorithm, then SQ-2 is being used and not SA-2. The proof for SA-2 will fail using schedule-1 (although the mixed algorithm might work just fine).

SQ is not a “bad” thing. It’s just that one cannot cite the SA proof of convergence, although the SQ being used might be really among the best algorithms being used for a given system. For example, there is no doubt that Newton’s method is superior to SA for parabolic objective functions. The point is that there are proofs that SA likely can do more for more complex problems, although it might not be the most efficient algorithm for a specific system.

See more below on how to modify ASA with QUENCH OPTIONS, easily permitting various degrees of quenching. This is especially useful if you determine that your optimal state(s) can be reached using SQ.

5.2.2. 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_k^i , 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 $p_{k+1}^i = p_k^i + y^i(B_i - A_i)$, and “temperatures” T_i ,

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

The DEFINE_OPTIONS USER_GENERATING_FUNCTION permits using an alternative to this ASA distribution function.

5.2.3. 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

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

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. The DEFINE_OPTIONS USER_ACCEPT_ASYMP_EXP or USER_ACCEPT_THRESHOLD permit using alternatives to this Boltzmann distribution function.

5.2.4. Reannealing Temperature Schedule

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

$$T_i(k_i) = T_{i0} \exp(-c_i k_i^{1/D}) .$$

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, and the Pre-Compile DEFINE_OPTIONS

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.

As determined by the Program Options selected, the reannealing of the cost temperature resets the scale of the annealing of the cost acceptance criteria as

$$T_{\text{cost}}(k_{\text{cost}}) = T_{0 \text{ cost}} \exp(-c_{\text{cost}} k_{\text{cost}}^{1/D}) .$$

The new $T_{0 \text{ cost}}$ is taken to be the minimum of the current initial cost temperature and the maximum of the absolute values of the best and last cost functions and their difference. The new k_{cost} is calculated taking T_{cost} as the maximum of the current value and the absolute value of the difference between the last and best saved minima of the cost function, constrained not to exceed the current initial cost temperature. This procedure essentially resets the scale of the annealing of the cost temperature within the scale of the current best or last saved minimum.

This default algorithm for reannealing the cost temperature, taking advantage of the ASA importance sampling that relates most specifically to the parameter temperatures, while often is quite efficient for some systems, may lead to problems in dwelling too long in local minima for other systems. In such case, the user may also experiment with alternative algorithms effected using the Reanneal_Cost OPTIONS, discussed below. For example, ASA provides an alternative calculation for the cost temperature, when Reanneal_Cost < -1 or > 1, that periodically calculates the initial and current cost temperatures or just the initial cost temperature, resp., as a deviation over a sample of cost functions.

These reannealing algorithms can be changed adaptively by the user as described below in the sections USER_REANNEAL_COST and USER_REANNEAL_PARAMETERS.

5.3. 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.

In the context of efficiency and necessity, the user should be alert to recognize that any sampling or optimization program generally should be considered as complementary, not as a substitute, to gaining knowledge of a particular system. Unlike relatively “canned” codes that exist for (quasi-)linear systems, nonlinear systems typically are non-typical. Often some homework must be done to understand the system, and tuning often is required of numerical algorithms such as ASA. For example, while principal component analyses (PCA) often suffices to generate good (quasi-)orthogonal or (quasi-)independent sets of parameters, this is not true for general nonlinear systems. While such innovations as reannealing take good advantage of ASA which offers independent distributions for each parameter, this generally

may not be a good substitute for a user-defined front-end, e.g., before the call to `asa()` or even embedded within the `cost_function()`, to interpret and define relevant parameters.

The ASA-NOTES file contains the sections @@Number of Generated States Required and @@Judging Importance-Sampling, recommending use of log-log plots to extrapolate the number of generated states required to attain a global minimum, possibly as a function of selected OPTIONS.

6. 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), is contained in `asa_usr.c`, `asa_usr.h` and `asa_usr_cst.c`. (2) The ASA optimization procedure, is contained in `asa.c` and `asa.h`. The file `asa_usr_asa.h` contains definitions and macros common to both `asa.h` and `asa_usr.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 ASA-Makefile), and compare your `asa_out` and `asa_usr_out` files to the `asa_test_asa` and `asa_test_usr` files, respectively, provided with this code. 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, you can insert it into `asa_usr_cst.c`. If you wish to insert more OPTIONS, as a quick guide to get started, you can search through `asa_usr.c` and the ASA-Makefile for all occurrences of “MY_TEMPLATE_” to insert the necessary definitions required to run ASA. If you use both `OPTIONS_FILE` and `OPTIONS_FILE_DATA` set to TRUE, then usually most such information can be placed in the `asa_opt` file, and then only the `cost_function()` must be inserted. The place to insert the `cost_function()` is marked by “MY_TEMPLATE_cost.”

7. ASA-Makefile/Compilation Procedures

The ASA-Makefile is intended to be a template for your own Makefile. For quick use, just copy this file to Makefile, which will be recognized by any standard make tool.

The PostScript® ASA-README.ps and ASCII ASA-README.txt and ASA-README+.txt files were generated using ‘make doc’. The ASA-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 ASA-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 ASA-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 ASA-Makefile or at least use some of these options in your own compilation procedures.

Note that the ASA-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 ASA-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_usr_asa.h`, and `asa.h` or `asa_usr.h` files as required.

Since the ASA–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 ASA–Makefile.

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

```
% cp ASA-Makefile Makefile
% gcc -g -DASA_TEST=TRUE -o asa_run asa_usr.c asa_usr_cst.c asa.c -lm
% asa_run
```

Note that if you use gcc across different levels of compiler optimization on different platforms, you may wish to invoke -ffloat-store to be sure to get the same precision in your results.

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

The easiest way for many users to quickly use ASA likely is to invoke the COST_FILE, OPTIONS_FILE, and OPTIONS_FILE_DATA OPTIONS (the default), using the files asa_usr_cst.c and asa_opt as templates. This is further described below and illustrated in the http://www.ingber.com/asa_examples.txt file in the section Use of COST_FILE on Shubert Problem.

7.1. DLL ASA–Makefile

Under Cygwin (cygwin.com), set ASA_LIB to TRUE and INCL_STDOUT to FALSE (OPTIONS described below), with the command

```
% make asadll
```

to produce a DLL to call asa_main() as a DLL function under windows. (Ignore any undefined references to _WinMain.) Note that per instructions given in the ASA–Makefile, -mno-cygwin -mrtld should be included in CFLAGS. If -mno-cygwin flag has been removed, use a mingw-targeted cross-compiler.

If paths are used to access files in code, under Windows use absolute paths with “\\” (double back slash) to separate folders/directories, instead of relative paths with “/” (single forward slash) separators as on other Unix platforms.

8. Generic ASA Interfaces

The sections above describe how to quickly adapt ASA for use in many problems. However, complex projects often require sophisticated use of multiple languages to handle data and multiple algorithms. ASA has many OPTIONS that enable users to interface ASA with such complex projects.

ASA should compile under C++ as well as under C compilers. For example, I regularly test this by running projects under both gcc and g++. This can be very useful when ASA is called from other C++ programs, e.g., when using ASA_LIB set to TRUE. The ASA-NOTES file has additional information, e.g., when using SUNPRO CC, etc.

I have led many projects that required ASA to interface with Java, Maple, Matlab, MySQL, etc. The approach briefly described below can be applied to any language that permits a simple interface to C code. This definitely requires some expert experience in C, so you may have to find a local C guru, since I cannot help you with your specific project.

Some specific interfaces have been prepared by other people, and I have included some of them in the asa_contrib.txt file. The tradeoff for their simple use is that these approaches are limited to using just a few ASA OPTIONS as they typically have trimmed down the ASA code.

The generic approach is to utilize at least the OPTIONS ASA_LIB and OPTIONAL_DATA_PTR, setting them to TRUE. ASA_LIB permits the entire ASA code to be called as a simple function. Its sole parameter can be a struct defined by OPTIONAL_PTR_TYPE, e.g., OPTIONAL_PTR_TYPE PROJECT, defined in asa_usr_asa.h. A small include file common to asa_usr_asa.h and to the larger complex project, e.g., project.h, is used to define the constituents of the the PROJECT struct. As described above

in the DLL ASA–Makefile sub-Section of Section 7, ASA_LIB can be used to create a DLL to be called by Windows programs.

A small C function, e.g., `project.c`, is to be used for the interface between ASA and the other language. Similarly, another small function also may be used to interface the project to handle the interface, e.g., `project.m`, `project.java`, `project.mpl`, etc. Inversely, the interface may (also) be between the cost function, e.g., in `asa_usr_cst.c` or `asa_usr.c`, and the project. Then the application below is used to pass information between the cost function and the other language.

The other language passes information and data to `project.c` required by ASA, where it is packed into the struct defined by `OPTIONAL_PTR_TYPE`. Multiple or recursive calls to ASA can be handled by including a flag in this struct, e.g., to turn on different cost functions. Also added to this struct are placeholders for the output of ASA required by the project. This struct is passed to the ASA code by calling `asa_main ()` defined in `asa_usr.c` with a parameter `PROJECT *Project`. In `asa_main ()`, in the section defining properties of `OPTIONAL_DATA_PTR`, the pointer to Project struct is set to the pointer path to `Asa_Data_Ptr`. `Asa_Data_Ptr` is now passed throughout the entire ASA code via the `OPTIONS` pointer, project parameters can be adaptively changed, etc. See the discussion under `Asa_Data_Ptr`.

After the call to `asa ()` in `asa_usr.c`, its output can be packed into the project struct, before memory is freed. The pointer `Asa_Data_Ptr` should be set to `NULL` instead of freed; see the comment in `asa_usr.c` at the place `Asa_Data_Ptr` is freed in the default code.

It is wise to create `#define PROJECT` and `#endif` pairs wherever changes to any ASA code are made, define `PROJECT` to `TRUE` in `asa_usr_asa.h`, so that it will be easy to modify updated ASA code, etc. Probably several such changes will have to be made in `asa_usr.c`. Control of `OPTIONS` likely will best be handled in `asa_usr_asa.h` than in the ASA–Makefile.

9. User Options

Program Options, i.e., the `USER_DEFINES` typedef on the `OPTIONS`, `USER_OPTIONS`, `RECUR_USER_OPTIONS`, etc., are turned on during the running of `asa ()`. The `DEFINE_OPTIONS` are compiled in by the use of arguments to the compilation or by setting them in the `asa_usr_asa.h` file. An example of the former is `Reanneal_Parameters`, and an example of the latter is `ASA_SAMPLE`. The basic code is kept small for most users by using the Pre–Compile `DEFINE_OPTIONS` to pull in additional `DEFINE_OPTIONS` only if required. The Program Options are intended to be used adaptively and/or to pull in additional code for cases where repeated or recursive use, e.g., when using `SELF_OPTIMIZE`, might be facilitated by having control of some Program Options at separate levels.

Note that even when the `DEFINE_OPTIONS` or Program Options are used to pull in new code, separate levels of control also can be achieved, albeit usually at the price of incurring some overhead in setting values at some levels of recursion or repeated calls. For example, in cases where new arrays or functions come into play, enough parameters are passed between the `asa` and user modules to calculate the defaults as well as different values adaptively. In some often used cases, separate `DEFINE_OPTIONS` are given, e.g., both `OPTIONS_FILE` and `RECUR_OPTIONS_FILE` exist. I have tried to strike some reasonable balance between these goals and constraints.

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.

Note that the Pre–Compile `DEFINE_OPTIONS` are all in capital letters, and the adaptive Program Options (under structure `USER_OPTIONS` in the user module and under structure `OPTIONS` in the `asa` module) are in capital and lower-case letters. In this file, often just the term `OPTIONS` may refer to the set of all options when the context is clear.

9.1. Pre-Compile DEFINE_OPTIONS

9.1.1. USER_COST_FUNCTION=cost_function

The default name of the cost function is `cost_function`. This can be changed in `asa_usr_asa.h` (or the ASA-Makefile) by defining `USER_COST_FUNCTION`. This of course requires compiling in the new cost function and its prototype.

9.1.2. RECUR_USER_COST_FUNCTION=recur_cost_function

When `SELF_OPTIMIZE` is `TRUE`, the default name of the recur cost function is `recur_cost_function`. This can be changed in `asa_usr_asa.h` (or the ASA-Makefile) by defining `RECUR_USER_COST_FUNCTION`. This of course requires compiling in the new cost function and its prototype.

9.1.3. OPTIONS_FILE=TRUE

You can elect to read in many of the Program Options from `asa_opt` by setting `OPTIONS_FILE=TRUE`. `OPTIONS_FILE=TRUE` can be set in the ASA-Makefile in compilation commands or in `asa_usr_asa.h`.

9.1.4. OPTIONS_FILE_DATA=TRUE

If `OPTIONS_FILE` is set to `TRUE`, then setting `OPTIONS_FILE_DATA` to `TRUE` permits reading most initialization data from `asa_opt`, i.e., number of parameters, minimum and maximum ranges, initial values, and integer or real types. This should suffice for most applications, just requiring insertion of the user's cost_function into `asa_usr_cst.c` or `asa_usr.c`.

If `OPTIONS_FILE`, `OPTIONS_FILE_DATA` and `QUENCH_COST` are `TRUE`, then `*User_Quench_Cost_Scale` is read in from `asa_opt`. If `OPTIONS_FILE`, `OPTIONS_FILE_DATA`, `QUENCH_COST`, and `QUENCH_PARAMETERS` are `TRUE`, then `*User_Quench_Cost_Scale` and `User_Quench_Param_Scale []` all are read in from `asa_opt`.

9.1.5. RECUR_OPTIONS_FILE=FALSE

When `SELF_OPTIMIZE` is `TRUE`, you can elect to read in many of the Program Options for the top-level program from `asa_opt_recur` (which you will have to create in the style of `asa_opt`), by setting `RECUR_OPTIONS_FILE=TRUE`.

9.1.6. RECUR_OPTIONS_FILE_DATA=FALSE

When `SELF_OPTIMIZE` is `TRUE`, if `RECUR_OPTIONS_FILE` is set to `TRUE`, then setting `RECUR_OPTIONS_FILE_DATA` to `TRUE` permits reading most initialization data from `asa_opt_recur` (which you will have to create in the style of `asa_opt`), i.e., number of parameters, minimum and maximum ranges, initial values, and integer or real types.

If `RECUR_OPTIONS_FILE`, `RECUR_OPTIONS_FILE_DATA` and `QUENCH_COST` are `TRUE`, then `*User_Quench_Cost_Scale` is read in from `asa_opt_recur`. If `RECUR_OPTIONS_FILE`, `RECUR_OPTIONS_FILE_DATA`, `QUENCH_COST`, and `QUENCH_PARAMETERS` are `TRUE`, then `*User_Quench_Cost_Scale` and `User_Quench_Param_Scale []` all are read in from `asa_opt_recur`.

9.1.7. COST_FILE=TRUE

If `COST_FILE` is set to `TRUE`, then you can use a separate file to define your cost function. When used together with `OPTIONS_FILE` and `OPTIONS_FILE_DATA` both set to `TRUE`, most users may be able to just use their own `asa_usr_cst.c` file for their `cost_function ()` together with the `asa_opt` data file, and not have to work through some of the examples and templates contained in `asa_usr.c`.

When `COST_FILE` is set to `TRUE`, the file `asa_usr_cst.c` contains `cost_function ()`. If you wish to change the name of `cost_function ()` in `asa_usr_cst.c`, then you must also change this name in the call to `asa ()` in `asa_usr.c` (search under "`asa ()`" and in the prototype listing in `asa_usr.h` (in the `HAVE_ANSI` set to `TRUE` or `FALSE` section as appropriate). You may wish to copy the appropriate parameter list in `asa_usr_cst.c` just before the `ASA_TEST` problem to be sure of using the proper format expected by `asa ()`.

in `asa.c`.

The http://www.ingber.com/asa_examples.txt file contains a section Use of `COST_FILE` on Shubert Problem which illustrates the simple modifications of ASA required to use `COST_FILE`.

9.1.8. ASA_LIB=FALSE

Setting `ASA_LIB=TRUE` will facilitate your running `asa ()` as a library call from another program, calling `asa_main ()` in `asa_usr.c`. In the templates provided, all initializations and cost function definitions are set up in the user module. 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.

When `ASA_LIB` is `TRUE`, another function becomes available in `asa_usr.c`, `asa_seed ()`, which can be used to change the initial seed used in runs made by `asa_main ()`. If this routine is not called, then the default initial seed is used. An example of using this routine when calling `asa_main ()` is given with `ASA_TEMPLATE_LIB`, using a `main ()` at the end of the `asa_usr.c` file.

As described in the DLL ASA–Makefile sub-Section of Section 7, the ASA–Makefile and `ASA_LIB` can be used to create a DLL for Windows programs.

9.1.9. 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 ASA–Makefile.

9.1.10. IO_PROTOTYPES=FALSE

Most newer operating systems do not like any other I/O prototyping other than those in their own include files. Other machines, like a Dec–3100 under Ultrix complain that the ANSI I/O prototypes were inconsistent. A Sun under 4.1.x gcc gave warnings if no I/O prototypes were present. The defaults in `asa_usr_asa.h` use newer system prototypes. `IO_PROTOTYPES=TRUE` will uncomment out declarations for such items as `fprintf`, `fflush`, `fclose`, `exit`, and `fscanf`.

9.1.11. 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.

9.1.12. TIME_STD=FALSE

Some systems, e.g., hpux and Cygwin (with `-mno-cygwin`), use other Unix–standard macros to access time. If `-mno-cygwin` flag has been removed, use a mingw-targeted cross-compiler. Setting `TIME_STD=TRUE` when using `TIME_CALC=TRUE` will use these time routines instead.

9.1.13. TIME_GETRUSAGE=TRUE

An additional module for using `TIME_CALC` set to `TRUE`, setting `TIME_GETRUSAGE` to `FALSE`, is more portable to compile across some platforms, e.g., Cygwin (with `-mno-cygwin`), but it can require different parameters for timing results. If `-mno-cygwin` flag has been removed, use a mingw-targeted cross-compiler. Comments have been placed in the code in `asa.c`.

9.1.14. 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.

9.1.15. 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`.

9.1.16. 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 `asa_usr_cst.c` and `asa_usr.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.

On some machines that have register variables assigned inconsistently with other doubles, there can arise some numerical differences in some systems. There has been no such problem found on Sun/Solaris 2.x using `gcc`, but some problems have been noticed on some Intel chips using different `gcc` optimizations.

As mentioned above, note that the use of `-ffloat-store` with `gcc/g++` is to maintain the same level of precision across levels of compiler optimization on different platforms.

9.1.17. MIN_DOUBLE=SMALL_FLOAT

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

9.1.18. MAX_DOUBLE=1.0/SMALL_FLOAT

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

9.1.19. EPS_DOUBLE=SMALL_FLOAT

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

9.1.20. CHECK_EXPONENT=FALSE

When `CHECK_EXPONENT` is set to `TRUE`, the macro `EXPONENT_CHECK(x)`, defined in `asa.h` in terms of `MIN_DOUBLE` and `MAX_DOUBLE`, checks that an exponent `x` is within a valid range and, if not, adjusts its magnitude to fit in the range.

9.1.21. 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.

9.1.22. 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.

9.1.23. 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 `asa_usr.c` to force a fast exit for this demo; search for `FAST_EXIT`. (Note that this also could be achieved by using `OPTIONS->Immediate_Exit` discussed below.)

The `ASA_TEMPLATE_SELF_OPT` example uses `OPTIONS_FILE=FALSE` in the Pre-Compile Options. Note that `DEFINE_OPTIONS OPTIONS_FILE=TRUE` and `OPTIONS_FILE_DATA=TRUE` here would take data from `asa_opt` for the lower-level program using the `cost_function ()`. Both `DEFINE_OPTIONS RECUR_OPTIONS_FILE` and `RECUR_OPTIONS_FILE_DATA` would have to be set to `TRUE` to use `asa_opt_recur` to read in both the `OPTIONS` and the `recur_cost_parameters` data (which you would have to write in the style of `asa_opt`) for the top-level `recur_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 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 ()`.

9.1.24. 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 `ASA-Makefile` so that just running `make` will run the test problem for the new user. No attempt was made to optimize any `OPTIONS` for the `ASA_TEST` problem as it appears in the standard code.

9.1.25. ASA_TEST_POINT=FALSE

The code used for the `ASA_TEST` problem closely follows the reference given in `asa_usr.c`, and was rewritten from code given to the author in 1992. Other researchers have sent the author different code for this system, and all results agree within round-off errors.

However, note that the actual problem stated in the reference in `asa_usr.c` is harder, requiring the finding of an optimal point and not an optimal region. The code for that problem is given in `asa_usr.c` when `ASA_TEST_POINT` is set to `TRUE` (having the effect of setting `COST_FILE` to `FALSE` in `asa_usr_asa.h`). The http://www.ingber.com/asa_examples.txt file illustrates how that global minimum can be attained.

9.1.26. MY_TEMPLATE=TRUE

When `MY_TEMPLATE` is set to `TRUE` (the default), locations in `asa_usr.c` and `asa_usr_asa.h` become active sites for your own code. Searching `asa_usr.c` for “`MY_TEMPLATE`” provides a guide for additional code to add for your own system. For example, just above the occurrence of the guides for `MY_TEMPLATE_cost` 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.

9.1.27. USER_INITIAL_COST_TEMP=FALSE

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

9.1.28. RATIO_TEMPERATURE_SCALES=FALSE

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

9.1.29. USER_INITIAL_PARAMETERS_TEMPS=FALSE

Setting USER_INITIAL_PARAMETERS_TEMPS to TRUE permits you to specify the initial parameter temperatures in the User_Parameter_Temperature [] array. 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.

9.1.30. 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 the User_Delta_Parameter [] array in the user module as discussed below.

9.1.31. QUENCH_PARAMETERS=FALSE

This DEFINE_OPTIONS 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.

Once you decide you can quench, there are many more alternative algorithms you might wish to choose for your system, e.g., creating a hybrid global–local adaptive quenching search algorithm, e.g., using USER_REANNEAL_PARAMETERS described below. Note that just using the quenching OPTIONS provided with ASA can be quite powerful, as demonstrated in the http://www.ingber.com/asa_examples.txt file.

Setting QUENCH_PARAMETERS to TRUE can be extremely useful in very large parameter dimensions; see the ASA–NOTES file under the section on Quenching. 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

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

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

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

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 focused 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.

If QUENCH_PARAMETERS is TRUE, then User_Quench_Param_Scale [] must be defined as described below.

If OPTIONS_FILE_DATA, QUENCH_COST, and QUENCH_PARAMETERS are TRUE, then *User_Quench_Cost_Scale and User_Quench_Param_Scale [] all are read in from asa_opt. If RECUR_OPTIONS_FILE_DATA, QUENCH_COST, and QUENCH_PARAMETERS are TRUE, then *User_Quench_Cost_Scale and User_Quench_Param_Scale [] all are read in from asa_opt_recur.

9.1.32. 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 OPTIONS does not affect the annealing proof of ASA, and so this may be 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, USER_ACCEPT_ASYMP_EXP, USER_ACCEPT_THRESHOLD, or USER_ACCEPTANCE_TEST.

If QUENCH_COST is TRUE, then User_Quench_Cost_Scale [0] must be defined as described below.

If OPTIONS_FILE_DATA and QUENCH_COST are TRUE, then User_Quench_Cost_Scale [] is read in from asa_opt. If RECUR_OPTIONS_FILE_DATA and QUENCH_COST are TRUE, then *User_Quench_Cost_Scale is read in from asa_opt_recur.

Similarly as noted above for QUENCH_PARAMETERS, setting QUENCH_COST to TRUE can be extremely useful in very large parameter dimensions; see the ASA-NOTES file under the section on Quenching.

9.1.33. QUENCH_PARAMETERS_SCALE=TRUE

When QUENCH_PARAMETERS is TRUE, if QUENCH_PARAMETERS_SCALE is TRUE, then the temperature scales and the temperature indexes are affected by User_Quench_Param_Scale []. This can have the effects of User_Quench_Param_Scale [] appear contrary, as the effects on the temperatures from the temperature scales and the temperature indexes can have opposing effects. However, these defaults are perhaps most intuitive when the User_Quench_Param_Scale [] are on the order of the parameter dimension.

When QUENCH_PARAMETERS is TRUE, if QUENCH_PARAMETERS_SCALE is FALSE, only the temperature indexes are affected by User_Quench_Param_Scale []. The same effect could be managed by raising Temperature_Anneal_Scale to the appropriate power, but this may not be as convenient.

9.1.34. QUENCH_COST_SCALE=TRUE

When QUENCH_COST is TRUE, if QUENCH_COST_SCALE is TRUE, then the temperature scale and the temperature index are affected by User_Quench_Cost_Scale [0]. This can have the effects

of `User_Quench_Cost_Scale [0]` appear contrary, as the effects on the temperature from the temperature scale and the temperature index can have opposing effects. However, these defaults are perhaps most intuitive when `User_Quench_Cost_Scale [0]` is on the order of the parameter dimension.

When `QUENCH_COST` is `TRUE`, if `QUENCH_COST_SCALE` is `FALSE`, only the temperature index is affected by `User_Quench_Cost_Scale [0]`. The same effect could be managed by raising `Temperature_Anneal_Scale` to the appropriate power, but this may not be as convenient.

9.1.35. `ASA_TEMPLATE=FALSE`

There are several templates that come with the ASA code. To permit use of these `OPTIONS` without having to delete these extra tests, these templates are wrapped with `ASA_TEMPLATE`'s. To use your own cost function, you likely will only have to write `cost_function ()` in `asa_usr_cst.c`, and use the `asa_opt` file. If you wish to add more `OPTIONS` or code, you may need to write relevant portions of `cost_function ()` and `initialize_parameters ()` in `asa_usr.c` and `asa_usr.h`.

The ASA-Makefile has several examples of `DEFINE_OPTIONS` that will generate test examples using special `ASA_TEMPLATE`'s set to `TRUE`. These are `{ASA_TEMPLATE_LIB, ASA_TEMPLATE_ASA_OUT_PID, ASA_TEMPLATE_MULTIPLE, ASA_TEMPLATE_SELF_OPT, ASA_TEMPLATE_SAMPLE, ASA_TEMPLATE_QUEUE, ASA_TEMPLATE_PARALLEL, ASA_TEMPLATE_SAVE}`; the sets of Pre-Compile `OPTIONS` these use are defined in `asa_usr_asa.h`.

Lines marked off by `ASA_TEMPLATE`, with no additional suffix, are for specific examples only. `ASA_TEMPLATE`, with no suffix, should not be set to `TRUE`, else all groups of these examples will be brought into the code, likely not what is wanted.

9.1.36. `OPTIONAL_DATA_DBL=FALSE`

It can be useful to return/pass additional information to the user module from/through the `asa` module. When `OPTIONAL_DATA_DBL` is set to `TRUE`, an additional Program Option pointer, `*Asa_Data_Dbl`, and its dimension, `Asa_Data_Dim_Dbl`, are available in `USER_DEFINES` `*USER_OPTIONS` to gather such data.

In the `ASA_TEMPLATE_SELF_OPT` example provided (see the set of `DEFINE_OPTIONS` used in `asa_usr_asa.h`), `OPTIONAL_DATA_DBL` 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 `asa_usr.c`. Here, `Asa_Data_Dbl [0]` is used as a flag to print out `Asa_Data_Dbl [1]` in `asa_usr.c`, set to `*best_number_generated_saved` in `asa.c`.

9.1.37. `OPTIONAL_DATA_INT=FALSE`

It can be useful to return/pass additional integer information to the user module from/through the `asa` module. When `OPTIONAL_DATA_INT` is set to `TRUE`, an additional Program Option pointer, `*Asa_Data_Int`, and its dimension, `Asa_Data_Dim_Int`, are available in `USER_DEFINES` `*USER_OPTIONS` to gather such data.

9.1.38. `OPTIONAL_DATA_PTR=FALSE`

It can be useful to return/pass additional array or structure information to the user module from/through the `asa` module (possibly containing other structures, e.g., useful when `SELF_OPTIMIZE` is `TRUE`). When `OPTIONAL_DATA_PTR` is set to `TRUE`, an additional Program Option pointer, `*Asa_Data_Ptr`, and its dimension, `Asa_Data_Dim_Ptr`, are available in `USER_DEFINES` `*USER_OPTIONS` to gather such data. The type of `*Asa_Data_Dim_Ptr` is a pre-compile `OPTIONS` set by `OPTIONAL_PTR_TYPE`. See examples under `Asa_Data_Dim_Ptr` and `Asa_Data_Ptr`.

If `OPTIONAL_DATA_PTR` is being used for `RECUR_USER_OPTIONS` as well as for `USER_OPTIONS`, you need not create (or free) additional memory in `recur_cost_function()` for `Asa_Data_Dim_Ptr` and `Asa_Data_Ptr` to be passed to the inner `cost_function()`, but rather link pointers to those in `RECUR_USER_OPTIONS`. In `asa_usr.c`, there are guidelines to set “`#if TRUE`” to “`#if FALSE`” at these points of the code. This is the proper technique to use if `ASA_SAVE`, `ASA_SAVE_OPT`, or

ASA_SAVE_BACKUP is set to TRUE (since data is saved by asa() depending on the level of recursion)..

If ASA_SAVE, ASA_SAVE_OPT, and ASA_SAVE_BACKUP are not set to TRUE, then multiple levels of recursion can each have their own defined information indexed to different elements of the array of structures of size Asa_Data_Dim_Ptr.

The http://www.ingber.com/asa_examples.txt file contains some guidance of the use of OPTIONAL_DATA_PTR and Asa_Data_Ptr.

9.1.39. OPTIONAL_PTR_TYPE=USER_TYPE

When OPTIONAL_DATA_PTR is set to TRUE, the type of *Asa_Data_Ptr is a pre-compile OPTIONS set by OPTIONAL_PTR_TYPE, e.g., changing the label USER_TYPE in asa_usr_asa.h. Be sure to place any non-standard types, like your own typedef struct, before the #define OPTIONAL_PTR_TYPE at the top of asa_usr_asa.h, e.g., under #if MY_TEMPLATE (since OPTIONAL_PTR_TYPE is tested below in asa_usr_asa.h). See the discussion under Asa_Data_Ptr.

9.1.40. 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 asa_usr.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 asa_usr.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;

9.1.41. USER_ACCEPT_ASYMP_EXP=FALSE

When USER_ACCEPT_ASYMP_EXP is TRUE, an asymptotic form of the exponential function as an alternative to the Boltzmann function becomes available for the acceptance test. A parameter OPTIONS->Asymp_Exp_Param becomes available, with a default of 1.0 in asa_usr.c giving the standard Boltzmann function. If you require a more moderate acceptance test, then negative Asymp_Exp_Param may be helpful.

9.1.42. USER_ACCEPT_THRESHOLD=FALSE

When USER_ACCEPT_THRESHOLD is TRUE, a simple alternative to the Boltzmann function becomes available for the acceptance test, simply defining the probability of acceptance to be 1 if

$$C(p_{k+1}) - C(p_k) \leq T_{\text{cost}} ,$$

and 0 otherwise.

9.1.43. USER_ACCEPTANCE_TEST=FALSE

If the Pre-Compile DEFINE_OPTIONS USER_ACCEPTANCE_TEST is set to TRUE, the Boltzmann test probability function used in the acceptance criteria in asa.c can be changed, adaptively if desired, in asa_usr.c in user_acceptance_test () (and in recur_user_acceptance_test () if SELF_OPTIMIZE is TRUE). The names of these functions are set to the relevant pointer in asa_usr.c, and can be changed if desired, i.e.,

If both USER_ACCEPTANCE_TEST and USER_ACCEPT_ASYMP_EXP are set to TRUE, then the default OPTIONS->Asymp_Exp_Param = 1 can be used in asa_usr.c to duplicate the Boltzmann test in asa.c, e.g., as a template to further develop a new acceptance test.

USER_OPTIONS->Acceptance_Test = user_acceptance_test;

RECUR_USER_OPTIONS->Acceptance_Test = recur_user_acceptance_test;

When USER_ACCEPTANCE_TEST is TRUE, then any random numbers needed for the acceptance criteria are generated in the user module instead of in the asa module.

When `USER_ACCEPTANCE_TEST` is `TRUE`, additional `OPTIONS` are available to modify the acceptance criteria, either after the cost function is calculated or during its calculation:

```
USER_OPTIONS->User_Acceptance_Flag
USER_OPTIONS->Cost_Acceptance_Flag
USER_OPTIONS->Last_Cost
USER_OPTIONS->Cost_Temp_Curr
USER_OPTIONS->Cost_Temp_Init
USER_OPTIONS->Cost_Temp_Scale
USER_OPTIONS->Prob_Bias
USER_OPTIONS->Random_Seed
```

Failing the acceptance test is not equivalent to dropping generated states from consideration for testing with the acceptance criteria, e.g., if they fail some regional constraints. `asa()` is designed so that `User_Acceptance_Flag` is set to `TRUE` prior to calling the cost function whenever acceptance tests need not be performed, i.e., when using the cost function to generate initial conditions, when being used to calculate derivatives, or when samples are being generated to calculate the cost temperature; otherwise it is set to `FALSE`. The value of `Cost_Acceptance_Flag` always is set to `FALSE` before entering the cost function.

When entering the acceptance function, if `Cost_Acceptance_Flag` is `TRUE`, then the value of `USER_OPTIONS->User_Acceptance_Flag` (assuming `*valid_state_generated_flag` is `TRUE`) calculated in `user_cost_function()` determines the value of the acceptance test. Otherwise, `USER_OPTIONS->Acceptance_Test()` is called to calculate the value of `USER_OPTIONS->User_Acceptance_Flag`. Note that if the cost function is used to calculate the acceptance criteria, and it is acceptable (e.g., also `*valid_state_generated_flag` is `TRUE`), then both `USER_OPTIONS->User_Acceptance_Flag` and `USER_OPTIONS->Cost_Acceptance_Flag` must be set to `TRUE`.

For example, this can be useful if during the calculation of the cost function, without having to proceed to the final evaluation, it becomes clear that the acceptance criteria will not be passed. This might occur if the cost function is increasing during its calculation and an acceptance test is carried out using the uniform random number calculated at the top of the cost function. The partially evaluated cost function can be compared to the `Last_Cost`, using the Boltzmann criteria or whatever criteria is established in `USER_OPTIONS->user_acceptance_test()`. Then it is clear that the acceptance criteria will not be met (of course after checking that any constraints are met and setting `*valid_state_generated_flag` to `TRUE` if so), then `USER_OPTIONS->User_Acceptance_Flag` can be set to or left at `FALSE`, and then proceed to return to `asa()`. However, other information registered in the acceptance function still should be calculated, e.g., updating indices, information used for `ASA_SAMPLE` and `ASA_PARALLEL`, etc.

9.1.44. `USER_GENERATING_FUNCTION=FALSE`

The ASA generating probability function in `asa.c` can be changed if the user sets the Pre-Compile `DEFINE_OPTIONS USER_GENERATING_FUNCTION` to `TRUE`; then this function can be changed, adaptively if desired, in `asa_usr.c` in `user_generating_distrib()` (and in `recur_user_generating_distrib()` if `SELF_OPTIMIZE` is `TRUE`) by setting `USER_GENERATING_FUNCTION` to `TRUE`. The names of these functions are set to the relevant pointer in `asa_usr.c`, and can be changed if desired, i.e.,

```
USER_OPTIONS->Generating_Distrib = user_generating_distrib;
```

```
RECUR_USER_OPTIONS->Generating_Distrib = recur_user_generating_distrib;
```

The parameters passed to these functions are further described below.

Several parameters additional to those required for the ASA distribution are passed to make it easier to install other common distributions. Note that range checks take place at multiple stages of search, so be sure your chosen ranges can take this into account.

9.1.45. `USER_REANNEAL_COST=FALSE`

In `asa.c` reannealing of the cost temperature is determined by the algorithm described above in the section Reannealing Temperature Schedule.

If the user sets the Pre-Compile `DEFINE_OPTIONS USER_REANNEAL_COST` to `TRUE`, while `Reanneal_Cost` is not 0 or -1, then the function controlling the new reannealed cost temperature can be controlled, adaptively if desired using `USER_OPTIONS`, in `asa_usr.c` in `user_reanneal_cost ()`, and in `recur_user_reanneal_cost ()` if `SELF_OPTIMIZE` is `TRUE`. The names of these functions are set to the relevant pointer in `asa_usr.c`, and can be changed if desired, i.e.,

`USER_OPTIONS->Reanneal_Cost_Function = user_reanneal_cost;`

`RECUR_USER_OPTIONS->Reanneal_Cost_Function = recur_user_reanneal_cost;`

In these functions, the variables `*current_cost_temperature`, `*initial_cost_temperature`, and the best and last saved cost function can be altered, and the returned integer value of `TRUE` or `FALSE` determines whether to use the best saved cost function as the current cost temperature.

Since these functions can be called every value of `Acceptance_Frequency_Modulus`, `Generated_Frequency_Modulus`, or when the ratio of accepted to generated points is less than `Accepted_To_Generated_Ratio`, this opportunity also can be used to adaptively change other `OPTIONS`. This can be very useful for systems where the scales of the acceptance criteria do not simply correlate the cost temperature with the current best value of the cost function.

For example, this function could be used when the last saved cost function is so close to zero that the effect would be to set the `*initial_cost_temperature` to that value, but the best value for the cost function is known to be less than zero. (An alternative moving average example is given in `asa_usr.c`.) Other alternatives are to use `USER_REANNEAL_COST` with default `FALSE` and `Reanneal_Cost > 1` or `< -1`, as described below.

9.1.46. `USER_REANNEAL_PARAMETERS=FALSE`

In `asa.h`, the macro

```
#define \
FUNCTION_REANNEAL_PARAMS(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_PARAMETERS` is `FALSE`.

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

`USER_OPTIONS->Reanneal_Params_Function = user_reanneal_params;`

`RECUR_USER_OPTIONS->Reanneal_Params_Function = recur_user_reanneal_params;`

Since `FUNCTION_REANNEAL_PARAMS ()` can be called every value of `Acceptance_Frequency_Modulus`, `Generated_Frequency_Modulus`, or when the ratio of accepted to generated points is less than `Accepted_To_Generated_Ratio`, this opportunity also can be used to adaptively change other `OPTIONS`. For example, if the `QUENCH_PARAMETERS OPTIONS` is set to `TRUE`, as discussed above, it may useful to create a hybrid global-local adaptive quenching search algorithm.

9.1.47. `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.

9.1.48. `REANNEAL_SCALE=10.0`

The reannealing scale used when `MAXIMUM_REANNEAL_INDEX` is exceeded.

9.1.49. 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. Four OPTIONS become available to monitor the sampling: 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[|:#|+]. A sample () function in asa_usr.c illustrates the use of these tags.

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. However, the selected OPTIONS still must yield good convergence if the optimal region is to be properly sampled.

9.1.50. ADAPTIVE_OPTIONS=FALSE

When ADAPTIVE_OPTIONS is set to TRUE, it is possible to change relevant OPTIONS in the USER_DEFINES struct in asa_usr_asa.h during run time, by including an asa_adaptive_options file. This can be very useful to efficiently determine how to tune long runs.

The function call to adaptive_options(USER_OPTIONS) should be placed at the top of your cost function. Some examples of the format of lines in the comma-delimited asa_adaptive_options file are given just above the adaptive_options function in asa_usr.c.

9.1.51. ASA_QUEUE=FALSE

When ASA_QUEUE is set to TRUE, a first-in first-out (FIFO) queue, of size USER_OPTIONS->Queue_Size, is used to collect generated states. When a new state is generated, its parameters are tested, within specified resolutions of USER_OPTIONS->Queue_Resolution [] (the absolute values of each of the differences between the parameters of the current generated state and those in the queue). If a previous state is already represented, then the stored values of the cost function and the cost flag are returned, instead of calling the cost function again. Note that the size of the array required to store the queued parameters is Queue_Size times the number of parameters, and this can consume a lot of CPU time as well as storage, so this OPTIONS is only useful for cost functions that are themselves very costly to evaluate. Setting ASA_TEMPLATE_QUEUE to TRUE will run an example using the ASA_TEST problem.

The ASA_QUEUE DEFINE_OPTIONS also can be used to coarse-grain a fit, by setting high values of Queue_Resolution []. Note the difference between the operations of this DEFINE_OPTIONS and ASA_RESOLUTION.

If ASA_QUEUE is TRUE and ASA_RESOLUTION is FALSE, machine precision is used for type double variables, the queue is created and subsequent variables are tested against this queue. If ASA_RESOLUTION and ASA_QUEUE are both TRUE, then the Coarse_Resolution [] array is used for Queue_Resolution [], ASA_RESOLUTION is enforced from the very first call to the cost function, and the queue is created using these coarse variables.

The default in asa.c for the FIFO queue uses a simple search among stored parameter values, under the assumption that for most complex systems for which ASA_QUEUE=TRUE is useful, the bottleneck is

in the evaluation of the cost functions. If you think this is not true for you, and you need to conserve CPU time in using lists, the http://www.ingber.com/asa_contrib.txt file gives code that uses doubly-linked and hashed lists.

If `ASA_QUEUE` and `ASA_PRINT_MORE` are TRUE then, whenever a queued cost function is used, this is recorded in `asa_out`.

9.1.52. `ASA_RESOLUTION=FALSE`

When `ASA_RESOLUTION` is set to TRUE, parameters are resolved to a user-defined resolution set in `USER_OPTIONS->Coarse_Resolution []`, i.e., within plus or minus the values of `Coarse_Resolution []`. This is performed as soon as candidate values are generated, for each parameter for which `Coarse_Resolution []` is greater than zero. Note the difference between the operations of this `OPTIONS` and `ASA_QUEUE`.

If `ASA_QUEUE` is TRUE and `ASA_RESOLUTION` is FALSE, machine precision is used for type double variables, the queue is created and subsequent variables are tested against this queue. If `ASA_RESOLUTION` and `ASA_QUEUE` are both TRUE, then the `Coarse_Resolution []` array is used for `Queue_Resolution []`, `ASA_RESOLUTION` is enforced from the very first call to the cost function, and the queue is created using these coarse variables.

When `USER_OPTIONS->Coarse_Resolution []` is > 0 and `parameter_type []` is > 0 (specifying an integer parameter), `ASA_RESOLUTION` takes precedence over `parameter_type []` when calculating new generated parameters.

9.1.53. `ASA_FUZZY=FALSE`

When `ASA_FUZZY` is set to TRUE, a modified fuzzy-logic algorithm originally developed by Hime Junior performs an adaptive tuning of selected `ASA OPTIONS`. `ASA_FUZZY` has several adaptive parameters defined in `asa_usr_asa.h`, and some fixed `#define` parameters in the several subroutines at the bottom of `asa_usr.c`. The default functions are set to tune `QUENCH_COST` and `QUENCH_PARAMETERS` parameters, so these are tested to be TRUE in the default `asa_usr_asa.h`. If you change any defaults, pay attention to sections labeled `ASA_FUZZY` at top and bottom sections of `asa_usr.c`, and in `asa_usr.h`.

Caution: Often `ASA_FUZZY` will turn on `QUENCHING > 1`, violating the proof of ASA. For many systems, this speeding up of the sampling process can be a welcome efficiency, but in some systems global minima may be missed, e.g., in systems with multiple scales.

9.1.54. `FITLOC=FALSE`

When `FITLOC` is set to TRUE, three subroutines become active to perform a local fit after leaving `asa ()`. This can be useful to shunt `asa ()` to a local code after the region of the global fit is known with some confidence, which many times is an efficient procedure.

Any robust quasi-linear optimization code may work well for this purpose. To illustrate this procedure, the user module contains `fitloc ()` which sets up the calls to `simplex ()`. `simplex ()` calls `calcf ()` which calls `cost_function ()`, and adds `USER_OPTIONS->Penalty` whenever `simplex ()` asks for parameters out of ranges of the parameters or whenever a constraint in `cost_function ()` is violated.

`ASA` parameters, the `OPTIONS`, are raised to a high level of view for direct control by the user. However, most optimization codes have their own parameters that may not be apparent to the user. For example, `fitloc ()` calls `simplex ()` which contains parameters such as `{tol1, tol2, no_progress, alpha, beta1, beta2, gamma, delta, iters}`. Many problems will require tuning of these parameters to achieve good results just from this `simplex ()` algorithm.

`USER_OPTIONS->Fit_Local` is passed to `cost_function ()`. This provides additional flexibility in deciding when to shunt `asa ()` over to `fitloc ()`, e.g., during multiple or recursive optimizations. `USER_OPTIONS->Iter_Max` determines the maximum iterations of the `cost_function ()` by `simplex ()`. `USER_OPTIONS->Penalty` determines how to weight violation of constraints, exceeding boundaries, etc.

9.1.55. FITLOC_ROUND=TRUE

If FITLOC is set to TRUE and FITLOC_ROUND is TRUE, then each time parameters are passed to or between the local routines, simplex (), calcf (), and fitloc (), they are first processed by rounding integers or respecting rounding according to ASA_RESOLUTION constraints prior to any further calculations. I.e., all values of a parameter within a given resolution are considered to be equivalent for calculating the cost function.

9.1.56. FITLOC_PRINT=TRUE

When FITLOC is set to TRUE, if FITLOC_PRINT is TRUE, then intermediate calculations will be printed out from fitloc () and simplex () in the user module.

9.1.57. MULTI_MIN=FALSE

When MULTI_MIN is set to TRUE, the lowest USER_OPTIONS->Multi_Number values of the cost function, determined to be the best-generated during the sampling process, of the cost function and their parameters are saved. These can be read out just after asa () returns after its fit. The pre-compile number USER_OPTIONS->Multi_Number and OPTIONS *Multi_Cost, **Multi_Params, *Multi_Grid, and Multi_Specify become available. In asa_usr.c, memory for the arrays USER_OPTIONS->Multi_Cost [USER_OPTIONS->Multi_Number][*parameter_dimension], USER_OPTIONS->Multi_Params [USER_OPTIONS->Multi_Number][*parameter_dimension], and USER_OPTIONS->Multi_Grid [*parameter_dimension] are set. Multi_Grid values must be set by the user, but may be overridden as explained below under USER_OPTIONS->Multi_Grid.

If OPTIONS->Curvature_0 is FALSE, all USER_OPTIONS->Multi_Number tangents and curvatures are calculated. This can be useful for some calculations requiring the shapes of the local minima.

This procedure selects local minima that statistically have maintained some quasi-stability during sampling. Note that this procedure does not guarantee that the USER_OPTIONS->Multi_Number lowest sampled values of the cost function will be saved, only those that were selected to be the best-generated during the sampling process.

If OPTIONS->Multi_Specify is set to 0, the selection of best-generated states includes all sampled instances of the cost functions. If OPTIONS->Multi_Specify is set to 1, the selection of best-generated states is constrained to include only those with different values of the cost function.

The http://www.ingber.com/asa_examples.txt file contains some guidance of the use of MULTI_MIN (and Multi_[] OPTIONS).

9.1.58. ASA_PARALLEL=FALSE

The parallelization procedure employed here does *not* destroy the sampling properties of ASA. When ASA_PARALLEL is set to TRUE, parallel blocks of generated states are calculated of number equal to the minimum of USER_OPTIONS->Gener_Block and USER_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 USER_OPTIONS->Gener_Mov_Avr, of the previous numbers of USER_OPTIONS->Gener_Block of generated states required to find a new best accepted state. If and when USER_OPTIONS->Gener_Mov_Avr is set to 0, then USER_OPTIONS->Gener_Block is not changed thereafter.

Each block of generated states is sorted to permit the lowest cost functions to pass first through the acceptance test.

There are hooks in asa.c to spawn off multiple processors. Parallel code should be inserted in asa.c between the lines:

```
/* *** ENTER CODE TO SPAWN OFF PARALLEL GENERATED STATES *** */
```

```
...
/* *** EXIT CODE SPAWNING OFF PARALLEL GENERATED STATES *** */
```

The ASA_TEMPLATE_PARALLEL example given in asa_usr.c illustrates how the run would proceed. Note that since the random number generator is called differently, generating some extra states as described above, the results are not identical to the serial ASA_TEST calculation.

9.1.59. FDLIBM_POW=FALSE

When FDLIBM_POW is set to TRUE, a user-defined function s_pow () is used instead of pow (). This may be desirable on some machines when a speed-up can be realized. Some code in http://www.ingber.com/asa_contrib.txt should first be tested with the standard ASA_TEST OPTIONS to see if the resulting asa_out file agrees with the asa_test_asa file.

9.1.60. FDLIBM_LOG=FALSE

When FDLIBM_LOG is set to TRUE, a user-defined function s_log () is used instead of log (). This may be desirable on some machines when a speed-up can be realized. Some code in http://www.ingber.com/asa_contrib.txt should first be tested with the standard ASA_TEST OPTIONS to see if the resulting asa_out file agrees with the asa_test_asa file.

9.1.61. FDLIBM_EXP=FALSE

When FDLIBM_EXP is set to TRUE, a user-defined function s_exp () is used instead of exp (). This may be desirable on some machines when a speed-up can be realized. Some code in http://www.ingber.com/asa_contrib.txt should first be tested with the standard ASA_TEST OPTIONS to see if the resulting asa_out file agrees with the asa_test_asa file.

9.2. Printing DEFINE_OPTIONS

9.2.1. USER_OUT=\"asa_usr_out\"

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

9.2.2. INCL_STDOUT=TRUE

Some compilers on some systems under some levels of optimization will not compile if “stdout” is present in the code. All instances of “stdout” and “printf” in the user and the asa modules can be commented out by setting INCL_STDOUT to FALSE. Note that this also will suppress some output from such OPTIONS as ASA_PIPE, TIME_CALC, etc.

The use of INCL_STDOUT set to FALSE is recommended for creating a DLL as described in the DLL ASA-Makefile sub-Section of Section 7.

9.2.3. ASA_PRINT=TRUE

Setting this to FALSE will suppress all printing within asa.

9.2.4. 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 ASA-Makefile.) If ASA_OUT=\"STDOUT\" then ASA will print to stdout. See the discussion of the use of ASA_TEMPLATE_ASA_OUT_PID in the section USER_ASA_OUT below to obtain multiple output files numbered according to the system pid.

9.2.5. 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 asa_usr.c, if USER_OPTIONS->Asa_Out_File = "STDOUT";, then ASA will print to stdout.

In the ASA_TEMPLATE_MULTIPLE example provided (see the set of DEFINE_OPTIONS used in asa_usr_asa.h), USER_ASA_OUT is used to generate multiple files of separate ASA runs. (If QUENCH_PARAMETERS and/or QUENCH_COST is set to TRUE, then this example will separate runs with different quenching values.)

In the ASA_TEMPLATE_ASA_OUT_PID example provided (see the set of DEFINE_OPTIONS used in asa_usr_asa.h), USER_ASA_OUT is used to generate ASA_OUT files of the form asa_out_x and asa_usr_out_x, where x is the system pid. This can be useful for a series of runs just changing parameters in asa_opt, getting different output files without recompiling. Depending on your system, you may have to change the include file and the prototype of getpid () in asa_usr.h under ASA_TEMPLATE_ASA_OUT_PID, and possibly the int declaration of pid_int in asa_usr.c.

9.2.6. 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 ASA-Makefile or on your compilation command lines.

9.2.7. 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.

The section above, Use of Documentation for Tuning, emphasizes the importance of using ASA_PRINT_MORE set to TRUE to help determine optimal tuning of ASA on specific problems.

9.2.8. G_FIELD=12 & G_PRECISION=7

The field width and precision of doubles is specified in asa.c as G_FIELD.G_PRECISION, e.g., as %gG_FIELD.G_PRECISION or %g-G_FIELD.G_PRECISION. These two Printing DEFINE_OPTIONS are available to change the default of 12.7.

9.2.9. ASA_SAVE=FALSE

When ASA_SAVE is set to TRUE, asa saves enough information in file asa_save after each newly best accepted state, to restart from the point entering the main annealing loop, continue thereafter from the best accepted state in asa_save. Of course, this use of I/O takes CPU resources, and can appreciably slow down your runs. When SYSTEM_CALL is set to TRUE, for extra protection, e.g., in case the run aborts during a write of asa_save, each time a file asa_save is written, it also is copied to a new file asa_save.old.

In order to store the whole block of random numbers used at any time, the number USER_OPTIONS->Random_Array_Dim and array USER_OPTIONS->Random_Array are required. These may be changed by the user in asa_usr.c for different random number generators and shuffling algorithms. The default is to use SHUFFLE defined in asa_usr.h for Random_Array_Dim in the default random number generator in asa_usr.c, and the pointer Random_Array is set to the pointer of the static array random_array at the top of asa_usr.c.

Just restart the run by executing asa_run. When ASA_SAVE is set to TRUE, the existence of file asa_save is used to determine whether a new run or a rerun is to proceed. Therefore, be sure your ASA

directory does not have any old `asa_save` file present if a new run is to start.

The `asa_opt` file is included just after `asa_save` files are read into the code. Therefore, any new C code you wish to have override information read in from `asa_save` can be simply added to the bottom of `asa_opt`. Be sure you write the names of these variables as they are used in the `asa.c` file, which can differ from their counterparts in `asa_usr.c` file. Some example are given at the end of `asa_opt` before the `#endif` statement. Each time you add new information to be compiled, be sure to enforce a new recompile of `asa.c` and `asa_run`. In most cases this can be done simply by removing `asa.o` before using a `make` or recompiling the executable. However, see `ASA_SAVE_OPT` for changes that may be made without any recompilation.

When ASA is run at several levels of recursion, if `USER_OPTIONS->Asa_Recursive_Level` is properly incremented from 0 at the innermost shell, the outermost shell at level `n` will create files `asa_save_{n}`.

9.2.10. ASA_SAVE_OPT=FALSE

When `ASA_SAVE_OPT` is set to `TRUE`, when `asa` is restarted, if the file `asa_opt_save` is present in the same directory as `asa_opt`, then new values of ASA parameters and `OPTIONS` are read in after initializing to the point of the last writing of `asa_save`.

No recompilation of the code is necessary, and only warnings are issued if `asa_save_opt` is not present. The file `asa_save_opt` should be created as an exact copy of `asa_opt` before changes in values of parameters and `OPTIONS` are made. When `ASA_SAVE_OPT` is `TRUE`, `ASA_SAVE` is automatically set to `TRUE` in `asa_usr_asa.h`.

9.2.11. ASA_SAVE_BACKUP=FALSE

When `ASA_SAVE_BACKUP` is set to `TRUE`, `asa` saves enough information after each newly best accepted state, creating a file `asa_save.{N_Accepted}`, to enable the user to restart from any previous best accepted state when that `asa_save.{best_state}` is copied to `asa_save`.

When used with `ASA_PIPE` and/or `ASA_PIPE_FILE`, `ASA_SAVE_BACKUP` permits the user to interactively tune the optimization process without having to start new runs. Read the above `ASA_SAVE` section on the use of the `asa_opt` file to modify code before reading in the `asa_save` file.

When `ASA_SAVE_BACKUP` is `TRUE`, `ASA_SAVE` is automatically set to `TRUE` in `asa_usr_asa.h`.

When ASA is run at several levels of recursion, if `USER_OPTIONS->Asa_Recursive_Level` is properly incremented from 0 at the innermost shell, the outermost shell at level `n` will create files `asa_save_{n}.{N_Accepted}`.

9.2.12. ASA_PIPE=FALSE

When `ASA_PIPE` is set to `TRUE`, `asa` prints to `STDOUT` lines of data after calls to the cost function, which can be used to update databases or graphs in real time. This information is {number of valid generated states, number of accepted states, best cost function, best parameter values, current cost temperature, current parameter temperatures, last cost function}.

9.2.13. ASA_PIPE_FILE=FALSE

When `ASA_PIPE_FILE` is set to `TRUE`, `asa` prints to `asa_pipe` lines of data that can be used to examine run data. This can be used complementary to `ASA_PIPE`.

9.2.14. ASA_EXIT_ANYTIME=FALSE

When `ASA_EXIT_ANYTIME` is set to `TRUE`, the first call to `asa()` creates the file `asa_exit_anytime`. To permit `IMMEDIATE_EXIT` before calculating the next generated state, simply delete the file `asa_exit_anytime`, and `asa()` will exit with code `IMMEDIATE_EXIT`. This can be useful on long runs.

9.2.15. SYSTEM_CALL=TRUE

When SYSTEM_CALL is set to FALSE, asa avoids popen () commands. This is useful on machines that do not permit these commands. For example, when ASA_SAVE is set to TRUE, asa uses a popen call in asa.c, to copy asa_save to asa_save.old. This also is required to use ASA_SAVE_BACKUP set to TRUE.

9.3. 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_Ratio;
    double Temperature_Anneal_Scale;
#ifdef USER_INITIAL_COST_TEMP
    double *User_Cost_Temperature;
#endif

    int Include_Integer_Parameters;
    int User_Initial_Parameters;
    ALLOC_INT Sequential_Parameters;
    double Initial_Parameter_Temperature;
#ifdef RATIO_TEMPERATURE_SCALES
    double *User_Temperature_Ratio;
#endif
#ifdef USER_INITIAL_PARAMETERS_TEMPS
    double *User_Parameter_Temperature;
#endif

    int Acceptance_Frequency_Modulus;
    int Generated_Frequency_Modulus;
    int Reanneal_Cost;
    int Reanneal_Parameters;

    double Delta_X;
#ifdef DELTA_PARAMETERS
    double *User_Delta_Parameter;
#endif

    int User_Tangents;
    int Curvature_0;

#ifdef QUENCH_PARAMETERS
    double *User_Quench_Param_Scale;
#endif
#ifdef QUENCH_COST
    double *User_Quench_Cost_Scale;
#endif
}
```

```

    LONG_INT N_Accepted;
    LONG_INT N_Generated;
    int Locate_Cost;
    int Immediate_Exit;

    double *Best_Cost;
    double *Best_Parameters;
    double *Last_Cost;
    double *Last_Parameters;

#ifdef OPTIONAL_DATA_DBL
    ALLOC_INT Asa_Data_Dim_Dbl;
    double *Asa_Data_Dbl;
#endif
#ifdef OPTIONAL_DATA_INT
    ALLOC_INT Asa_Data_Dim_Int;
    double *Asa_Data_Int;
#endif
#ifdef OPTIONAL_DATA_PTR
    ALLOC_INT Asa_Data_Dim_Ptr;
    OPTIONAL_PTR_TYPE *Asa_Data_Ptr;
#endif
#ifdef USER_ASA_OUT
    char *Asa_Out_File;
#endif
#ifdef USER_COST_SCHEDULE
    double ( *Cost_Schedule ) ();
#endif
#ifdef USER_ACCEPT_ASYMP_EXP
    double Asymp_Exp_Param;
#endif
#ifdef USER_ACCEPTANCE_TEST
    void ( *Acceptance_Test ) ();
    int User_Acceptance_Flag;
    int Cost_Acceptance_Flag;
    double Last_Cost;
    double Cost_Temp_Curr;
    double Cost_Temp_Init;
    double Cost_Temp_Scale;
    double Prob_Bias;
    LONG_INT *Random_Seed;
#endif
#ifdef USER_GENERATING_FUNCTION
    double ( *Generating_Distrib ) ();
#endif
#ifdef USER_REANNEAL_COST
    int ( *Reanneal_Cost_Function ) ();
#endif
#ifdef USER_REANNEAL_PARAMETERS
    double ( *Reanneal_Params_Function ) ();
#endif
#ifdef ASA_SAMPLE
    double Bias_Acceptance;
    double *Bias_Generated;

```

```

        double Average_Weights;
        double Limit_Weights;
    #endif
    #if ASA_QUEUE
        ALLOC_INT Queue_Size;
        double *Queue_Resolution;
    #endif
    #if ASA_RESOLUTION
        double *Coarse_Resolution;
    #endif
    #if FITLOC
        int Fit_Local;
        int Iter_Max;
        double Penalty;
    #endif
    #if MULTI_MIN
        double *Multi_Cost;
        double **Multi_Params;
        double *Multi_Grid;
        int Multi_Specify;
    #endif
    #if ASA_PARALLEL
        int Gener_Mov_Avr;
        LONG_INT Gener_Block;
        LONG_INT Gener_Block_Max;
    #endif
    #if ASA_SAVE
        ALLOC_INT Random_Array_Dim;
        LONG_INT *Random_Array;
    #endif
        int Asa_Recursive_Level;
    }
USER_DEFINES;

```

Note that two ways are maintained for passing the Program Options. Check the comments in the ASA-NOTES file. It may be necessary to change some of the options for some systems. Read the http://www.ingber.com/asa_examples.txt 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) asa_usr.c file

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 asa_usr.c. In the user module, the Program Options are stored in the structure USER_DEFINES *USER_OPTIONS (and in USER_DEFINES *RECUR_USER_OPTIONS if SELF_OPTIMIZE is TRUE).

(B) asa_opt file

It likely is most efficient to use a separate data file avoiding repeated compilations of the code, to test various combinations of Program Options, e.g., using the file asa_opt when OPTIONS_FILE and OPTIONS_FILE_DATA are set to TRUE in the ASA-Makefile or on your compilation command lines.

In the asa module (which can be called recursively) the structure is called USER_DEFINES *OPTIONS. For the rest of this file, where no confusion can reasonably arise, the Program Options will be referred to as USER_DEFINES *OPTIONS.

9.3.1. 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.

9.3.2. 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.

9.3.3. 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 `asa_usr.c`. For all calls \geq the limit of the number of calls to the cost function, terminate by setting `OPTIONS->Limit_Invalid_Generated_States = 0` and setting `*cost_flag = FALSE`. (Note that a quick exit also can be achieved using `OPTIONS->Immediate_Exit`.)

9.3.4. 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 `Acceptance_Frequency_Modulus` or `Generated_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.

9.3.5. OPTIONS->Cost_Precision[1.0E-18]

This sets the precision required of the cost function if exiting because of reaching `Maximum_Cost_Repeat`, which is effective as long as `Maximum_Cost_Repeat > 0`.

9.3.6. OPTIONS->Maximum_Cost_Repeat[5]

The maximum number of times that the cost function repeats itself, within limits set by `Cost_Precision`, before quitting. This test is performed only when `Acceptance_Frequency_Modulus` or `Generated_Frequency_Modulus` is invoked, or when the ratio of accepted to generated points is less than `Accepted_To_Generated_Ratio`, in order to help prevent exiting prematurely in a local minimum. If `Maximum_Cost_Repeat` is 0, this test is bypassed.

9.3.7. OPTIONS->Number_Cost_Samples[5]

When `Number_Cost_Samples > 0`, the initial cost temperature is calculated as the average of the absolute values of `Number_Cost_Samples` sampled cost functions.

When `Number_Cost_Samples < -1`, the initial cost temperature is calculated as the deviation over a sample of `-Number_Cost_Samples` number of cost functions, i.e., the square-root of the difference of the second moment and the square of the first moment, normalized by the ratio of `-Number_Cost_Samples` to `-Number_Cost_Samples - 1`.

When `ASA_SAVE` is set to `TRUE`, `Number_Cost_Samples` is set to 1 after the initial run since all the required information for subsequent runs already has been collected.

If the calculated initial cost temperature is not set using `USER_INITIAL_COST_TEMP`, and if it is essentially zero (using a test with `SMALL_FLOAT`), then it is reset to $e = 2.718$. If `ASA_PRINT` is `TRUE` a warning is printed.

See Reanneal_Cost for similar treatment of the reannealed cost temperature.

9.3.8. 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 ASA-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 `OPTIONS->User_Temperature_Ratio []`. 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.

The ASA-NOTES contain a section giving a little more explanation on the use of `Temperature_Ratio_Scale`.

9.3.9. OPTIONS->Cost_Parameter_Scale_Ratio[1.0]

This is the ratio of cost:parameter temperature annealing scales. As explained in the ASA papers, and as outlined in the ASA-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`,

$$c_{\text{cost}} = c \text{ Cost_Parameter_Scale_Ratio} .$$

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

9.3.10. 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 ASA-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`.

9.3.11. OPTIONS->User_Cost_Temperature

If USER_INITIAL_COST_TEMP is TRUE, a pointer, OPTIONS->User_Cost_Temperature, is used to adaptively initialize the cost temperature. If this choice is elected, then User_Cost_Temperature [] must be initialized.

9.3.12. OPTIONS->Include_Integer_Parameters[FALSE]

If Include_Integer_Parameters is TRUE, include integer parameters in derivative and reannealing calculations, except those with INTEGER_TYPE (2). 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 ASA_RESOLUTION to TRUE, as discussed further below.

9.3.13. 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.

9.3.14. 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.

9.3.15. OPTIONS->Initial_Parameter_Temperature[1.0]

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

9.3.16. 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.

9.3.17. 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.

9.3.18. OPTIONS->Acceptance_Frequency_Modulus[100]

The frequency of testing for periodic testing and reannealing, dependent on the number of accepted states. If Acceptance_Frequency_Modulus is set to 0, then this test is not performed.

9.3.19. OPTIONS->Generated_Frequency_Modulus[10000]

The frequency of testing for periodic testing and reannealing, dependent on the number of generated states. If Generated_Frequency_Modulus is set to 0, then this test is not performed.

9.3.20. OPTIONS->Reanneal_Cost[1]

A value of Reanneal_Cost set to FALSE=0 bypasses reannealing of the cost temperature. This might be done for systems where such reannealing is not useful. Note that the use of USER_REANNEAL_COST permits users to define their own cost temperature reannealing algorithm when Reanneal_Cost is not 0 or -1.

A value of Reanneal_Cost = 1 permits the default reannealing of the cost temperature to be part of the fitting process, correlating the cost temperature with the current last and best values of the cost function as described above.

If Reanneal_Cost > 1, then the reannealed initial cost temperature is calculated as the deviation over a sample of -Reanneal_Cost number of cost functions, i.e., the square-root of the difference of the second moment and the square of the first moment, normalized by the ratio of Reanneal_Cost to Reanneal_Cost - 1. For example, if the initial cost temperature is reannealed to a larger value, this increases the effective index of the current cost temperature, effectively slowing down the rate of decrease of future current cost temperatures as this index is increased for each acceptance test.

If Reanneal_Cost < -1, then the cost index is reset to 1, and the initial and current cost temperatures are calculated as the deviation over a sample of -Reanneal_Cost number of cost functions, i.e., the square-root of the difference of the second moment and the square of the first moment, normalized by the ratio of -Reanneal_Cost to -Reanneal_Cost - 1. This often gives rise to fluctuating current cost temperatures, sometimes diminishing the value of the acceptance test. However, for some systems that have different behavior at different scales, this can be a very useful OPTIONS.

The algorithms with Reanneal_Cost > 1 or < -1 typically require more calls to the cost function than the default of Reanneal_Cost = 1. This typically is even more so when Reanneal_Cost < -1 than when Reanneal_Cost > 1 due to the resetting of the current cost temperature as well as the initial cost temperature. Because of the fat tail of the parameter distributions, quite often relatively large values of the cost function will be included in the periodic sampling. However, of course the parameter temperatures continue to diminish, focusing the fit towards the global optimal value.

Note that Number_Cost_Samples can be used similarly for calculating the initial cost temperature.

9.3.21. OPTIONS->Reanneal_Parameters[TRUE]

This permits reannealing of the parameter temperatures 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.

9.3.22. OPTIONS->Delta_X[0.001]

The fractional increment of parameters used to take numerical derivatives when calculating tangents for reannealing, for each parameter chosen to be reannealed. This is overridden when DELTA_PARAMETERS is set to TRUE. If Delta_X is set to 0, then no tangents are calculated.

Note, that for second-derivative off-diagonal curvature calculations, the algorithm used may 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. Note that the user may set User_Tangents to TRUE, as discussed below, to choose any other algorithm to calculate derivatives or other indicators to be used for reannealing.

When calculating derivatives, an invalid cost function, i.e., returning *cost_flag = FALSE from the cost function, will exit asa() with *exit_code INVALID_COST_FUNCTION_DERIV. Also, when calculating derivatives, no extra test is performed to check that parameters are within their lower and upper bounds (since meshes for derivatives may exceed these bounds). If this is not desired, then within the cost function, within a test for USER_OPTIONS->Locate_Cost == 3, a decision may be made

whether to return this *exit_code.

9.3.23. 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 each parameter chosen to be reannealed. 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. No tangents are calculated for any parameter if User_Delta_Parameter[] is 0.

9.3.24. 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 asa_usr.c and asa_usr_cst.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_SAMPLE example.

9.3.25. 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 Acceptance_Frequency_Modulus, Generated_Frequency_Modulus, or Accepted_To_Generated_Ratio, then set OPTIONS->Curvature_0 to -1.

9.3.26. 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, and values defined for each dimension. 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.

Note that you can use this control quite differently, to slow down the annealing process by setting OPTIONS->User_Quench_Param_Scale [] to values less than 1. This can be useful in problems where the global optimal point is at a quite different scale from other local optima, masking its presence.

If OPTIONS_FILE_DATA, QUENCH_COST, and QUENCH_PARAMETERS are TRUE, then *User_Quench_Cost_Scale and User_Quench_Param_Scale [] all are read in from asa_opt. If RECUR_OPTIONS_FILE_DATA, QUENCH_COST, and QUENCH_PARAMETERS are TRUE, then *User_Quench_Cost_Scale and User_Quench_Param_Scale [] all are read in from asa_opt_recur.

9.3.27. 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. 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.

If `OPTIONS_FILE_DATA` and `QUENCH_COST` are `TRUE`, then `*User_Quench_Cost_Scale` is read in from `asa_opt`. If `RECUR_OPTIONS_FILE_DATA` and `QUENCH_COST` are `TRUE`, then `*User_Quench_Cost_Scale` is read in from `asa_opt_recur`.

9.3.28. OPTIONS->N_Accepted

`N_Accepted` contains the current number of points saved by the acceptance criteria. This can be used to monitor the fit. On exiting from `asa()` `N_Accepted` contains the value of `*best_number_accepted_saved` in `asa.c`. Note that the value of `N_Accepted` typically will be less in the cost function than in `ASA_OUT`, as the value of the returned cost must be tested back in `asa()` to see if `N_Accepted` should be incremented.

9.3.29. OPTIONS->N_Generated

`N_Generated` contains the current number of generated states. This can be used to monitor the fit. On exiting from `asa()` `N_Generated` contains the value of `*best_number_generated_saved` in `asa.c`. Note that the value of `N_Generated` typically will be less in the cost function than in `ASA_OUT`, as this value is only incremented upon returning to `asa()` if some tests are passed.

9.3.30. OPTIONS->Locate_Cost

`Locate_Cost` is a flag set in `asa()`, telling at what point the cost function is being called. This can be useful for determining when to perform tests while in your cost function. When `ASA_PRINT` is `TRUE`, the value is printed out upon exiting `asa()`. Note that there are several possible values that be reasonable, depending on from where the final exit was called.

`Locate_Cost = 0`. The cost function is being used for the initial cost temperature.

`Locate_Cost = 1`. The cost function is being used for the initial cost value.

`Locate_Cost = 2`. The cost function is being used for a new generated state.

`Locate_Cost = 12`. The cost function is being used for a new generated state just after a new best state was achieved.

`Locate_Cost = 3`. The cost function is being used for the cost derivatives to reanneal the parameters.

`Locate_Cost = 4`. The cost function is being used for reannealing the cost temperature.

`Locate_Cost = 5`. The cost function is being used for the exiting of `asa()` to calculate final curvatures.

`Locate_Cost = -1`. Exited main loop of `asa()` because of user-defined constraints in `OPTIONS`, e.g., `Acceptance_Frequency_Modulus`, `Generated_Frequency_Modulus`, or `Accepted_To_Generated_Ratio`.

When calculating derivatives, an invalid cost function, i.e., returning `*cost_flag = FALSE` from the cost function, will exit `asa()` with `*exit_code INVALID_COST_FUNCTION_DERIV`. Also, when calculating derivatives, no extra test is performed to check that parameters are within their lower and upper bounds (since meshes for derivatives may exceed these bounds). If this is not desired, then within the cost function, within a test for `USER_OPTIONS->Locate_Cost == 3`, a decision may be made whether to return this `*exit_code`.

9.3.31. OPTIONS->Immediate_Exit[FALSE]

`OPTIONS->Immediate_Exit` is initialized to `FALSE` when entering `asa()`. At any time during the fit, except while the call to `cost_function()` from `asa()` is being used simply to calculate derivatives or the cost temperature, if the user sets `Immediate_Exit` to `TRUE`, then just after bookkeeping is performed by the acceptance test, `asa()` will exit with code `IMMEDIATE_EXIT`. All closing calculations of current

cost_tangents [] and cost_curvature [] are bypassed.

9.3.32. OPTIONS->Best_Cost

In asa_usr_asa.h, the OPTIONS *Best_Cost is a pointer to the value of the cost function of the saved best state calculated in asa (). E.g., together with *Best_Parameters and Locate_Cost, these OPTIONS can aid several adaptive features of ASA, e.g., automating the diminishing of ranges each time a new best state is achieved, as illustrated in the ASA_TEMPLATE just after the comment MY_TEMPLATE_diminishing_ranges in asa_usr.c. This OPTIONS is to be used read-only by the cost function, as calculated in asa (); do not change it in the user module unless you so wish to modify the sampling.

9.3.33. OPTIONS->Best_Parameters

In asa_usr_asa.h, the OPTIONS *Best_Parameters is a pointer to the values of the parameters of the saved best state calculated in asa (). E.g., together with *Best_Cost and Locate_Cost, these OPTIONS can aid several adaptive features of ASA, e.g., automating the diminishing of ranges each time a new best state is achieved, as illustrated in the ASA_TEMPLATE just after the comment MY_TEMPLATE_diminishing_ranges in asa_usr.c. This OPTIONS is to be used read-only by the cost function, as calculated in asa (); do not change it in the user module unless you so wish to modify the sampling.

9.3.34. OPTIONS->Last_Cost

In asa_usr_asa.h, the OPTIONS *Last_Cost is a pointer to the value of the cost function of the last saved state calculated in asa (). This can be compared to *Best_Cost as a measure of fluctuations among local minima. An example of use in a user-defined acceptance test is in asa_usr.c when USER_ACCEPTANCE_TEST is TRUE. This OPTIONS is to be used read-only by the cost function, as calculated in asa (); do not change it in the user module unless you so wish to modify the sampling.

9.3.35. OPTIONS->Last_Parameters

In asa_usr_asa.h, the OPTIONS *Last_Parameters is a pointer to the values of the parameters of the last saved state calculated in asa (). This can be compared to *Best_Parameters as a measure of fluctuations among local minima. This OPTIONS is to be used read-only by the cost function, as calculated in asa (); do not change it in the user module unless you so wish to modify the sampling.

9.3.36. OPTIONS->Asa_Data_Dim_Dbl

If the Pre-Compile Option OPTIONAL_DATA_DBL [FALSE] is set to TRUE, an additional Program Option, OPTIONS->Asa_Data_Dim_Dbl, becomes available to define the dimension of OPTIONS->Asa_Data_Dbl [].

9.3.37. OPTIONS->Asa_Data_Dbl

If the Pre-Compile Option OPTIONAL_DATA_DBL [FALSE] is set to TRUE, an additional Program Option pointer, OPTIONS->Asa_Data_Dbl, becomes available 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 asa_usr.c when SELF_OPTIMIZE is TRUE.

9.3.38. OPTIONS->Asa_Data_Dim_Int

If the Pre-Compile Option OPTIONAL_DATA_INT [FALSE] is set to TRUE, an additional Program Option, OPTIONS->Asa_Data_Dim_Int, becomes available to define the dimension of OPTIONS->Asa_Data_Int [].

9.3.39. OPTIONS->Asa_Data_Int

If the Pre-Compile Option `OPTIONAL_DATA_INT [FALSE]` is set to `TRUE`, an additional Program Option pointer, `OPTIONS->Asa_Data_Int`, becomes available to return additional integer 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.

9.3.40. OPTIONS->Asa_Data_Dim_Ptr

If the Pre-Compile Option `OPTIONAL_DATA_PTR [FALSE]` is set to `TRUE`, an additional Program Option, `OPTIONS->Asa_Data_Dim_Ptr`, becomes available to define the dimension of `OPTIONS->Asa_Data_Ptr`.

For example, a value of `Asa_Data_Dim_Ptr = 2` might be used to set different entries in data arrays at two levels of recursion. See the discussion under `OPTIONAL_DATA_PTR` and `Asa_Data_Ptr` for use in multiple recursion.

9.3.41. OPTIONS->Asa_Data_Ptr

If the Pre-Compile Option `OPTIONAL_DATA_PTR [FALSE]` is set to `TRUE`, an additional Program Option pointer, `OPTIONS->Asa_Data_Ptr`, becomes available to define an array, of type `OPTIONAL_PTR_TYPE` defined by the user, which can be used to pass arbitrary arrays or structures 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.

For example, struct `DATA` might contain an array `data[10]` to be used in the `cost_function`. `Asa_Data_Dim_Ptr` might have a value 2. Set `#define OPTIONAL_PTR_TYPE DATA`. Then, `data[3]` in struct `Asa_Data_Ptr[1]` could be set and accessed as `USER_OPTIONS->Asa_Data_Ptr[1].data[3]` in the `cost function`.

For example, your main program that calls `asa_main()` would have developed a struct `SelectedType *SelectedPointer`, and you can call `asa_main(SelectedPointer, ...)`. In `asa_usr_asa.h`, you would have

```
#define OPTIONAL_PTR_TYPE SelectedType
```

In `asa_usr.c` (and `asa_usr.h`) you would develop `int asa_main(OPTIONAL_PTR_TYPE *OptionalPointer, ...)` and, close to the appropriate `ASA_TEMPLATE`, you would set

```
USER_OPTIONS->Asa_Data_Ptr = OptionalPointer;
```

if `Asa_Data_Dim_Ptr = 1` were being passed to `ASA`; otherwise, if `Asa_Data_Dim_Ptr > 1`, as illustrated in the `ASA_TEMPLATE` in `asa_usr.c`, memory must be allocated (and then freed) for `Asa_Data_Ptr`, and each dimension must be properly aligned with the structs being passed from the main program.

Note that the typedef of the `SelectedType` struct (whether or not you explicitly use the term typedef), developed in your main program, cannot be passed by its name to `asa_main()`, and so it must be declared as well in the `asa` module. In `asa_usr_asa.h`, the `SelectedType` struct can be included with the use of an `#include SelectedStruct.h` common to your main program and `asa_usr_asa.h`.

As noted in `asa_usr.c`, before exiting `asa_main()`, instead of freeing `Asa_Data_Ptr`, if memory has been allocated outside `ASA`, e.g., by the use of `ASA_LIB`, use

```
USER_OPTIONS->Asa_Data_Ptr = NULL;
```

The http://www.ingber.com/asa_examples.txt file contains some guidance of the use of `OPTIONAL_DATA_PTR` and `Asa_Data_Ptr`.

9.3.42. 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.

9.3.43. 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 asa_usr.c, and to recur_user_cost_schedule () if SELF_OPTIMIZE is set to TRUE.

9.3.44. OPTIONS->Asymp_Exp_Param

When USER_ACCEPT_ASYMP_EXP [FALSE] is TRUE, an asymptotic form of the exponential function as an alternative to the Boltzmann function becomes available for the acceptance test. A parameter OPTIONS->Asymp_Exp_Param becomes available, with a default of 1.0 in asa_usr.c giving the standard Boltzmann function. The asymptotic approximation to the exp function used for the acceptance distribution is

$$\exp(-x) \rightarrow [1 - (1 - q)x]^{1/(1-q)}.$$

If you require a more moderate acceptance test, then negative Asymp_Exp_Param may be helpful.

9.3.45. OPTIONS->Acceptance_Test

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then (*Acceptance_Test) () is created as a pointer to the function user_acceptance_test () in asa_usr.c, and to recur_user_acceptance_test () if SELF_OPTIMIZE is set to TRUE.

9.3.46. OPTIONS->User_Acceptance_Flag

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then User_Acceptance_Flag is created. In asa (), User_Acceptance_Flag is set to TRUE prior to calling the cost function whenever acceptance tests need not be performed, i.e., when using the cost function to generate initial conditions, when being used to calculate derivatives, or when samples are being generated to calculate the cost temperature; otherwise it is set to FALSE. If User_Acceptance_Flag is returned from the cost function as FALSE, then it is assumed that the cost function will fail the acceptance criteria, but other data is still collected in the acceptance function. When entering the acceptance test in asa, a test is done to see if the acceptance test has already been determined by the cost function; if not, then OPTIONS->Acceptance_Test () is called to calculate the acceptance test to determine the resulting value of User_Acceptance_Flag.

9.3.47. OPTIONS->Cost_Acceptance_Flag

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then Cost_Acceptance_Flag is created. In asa (), Cost_Acceptance_Flag is set to a default of FALSE before entering the cost function. If both Cost_Acceptance_Flag and User_Acceptance_Flag are returned from the cost function as TRUE, then it is assumed that the cost function has decided that the acceptance criteria is passed, and other data is collected in the acceptance function.

9.3.48. OPTIONS->Cost_Temp_Curr

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then Cost_Temp_Curr is available to user_cost_function and/or to OPTIONS->Acceptance_Test to calculate the acceptance criteria.

9.3.49. OPTIONS->Cost_Temp_Init

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then Cost_Temp_Init is available to user_cost_function and/or to OPTIONS->Acceptance_Test to calculate the acceptance criteria.

9.3.50. OPTIONS->Cost_Temp_Scale

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then Cost_Temp_Scale is available to user_cost_function and/or to OPTIONS->Acceptance_Test to calculate the acceptance criteria.

9.3.51. OPTIONS->Prob_Bias

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then Prob_Bias is returned by the user module to the asa module. This usually is the Boltzmann test term which is compared with a uniform random number to determine acceptance, and its value can be required for other OPTIONS such as ASA_SAMPLE.

9.3.52. OPTIONS->Random_Seed

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then in asa() OPTIONS->Random_Seed is set to the address of the random seed throughout asa, to synchronize the random number generator with the rest of the run, e.g., permitting *Random_Seed to be used in user_cost_function().

9.3.53. OPTIONS->Generating_Distrib

If USER_GENERATING_FUNCTION [FALSE] is set to TRUE, then (*Generating_Distrib) () is created as a pointer to the function user_generating_distrib () in asa_usr.c, and to recur_user_generating_distrib () if SELF_OPTIMIZE is set to TRUE. The parameters passed to these functions are further described below.

9.3.54. OPTIONS->Reanneal_Cost_Function

If USER_REANNEAL_COST [FALSE] is set to TRUE, then (*Reanneal_Cost_Function) () is created as a pointer to the function user_reanneal_cost () in asa_usr.c, and to recur_user_reanneal_cost () if SELF_OPTIMIZE is set to TRUE.

9.3.55. OPTIONS->Reanneal_Params_Function

If USER_REANNEAL_PARAMETERS [FALSE] is set to TRUE, then (*Reanneal_Params_Function) () is created as a pointer to the function user_reanneal_params () in asa_usr.c, and to recur_user_reanneal_params () if SELF_OPTIMIZE is set to TRUE.

9.3.56. OPTIONS->Bias_Acceptance

If ASA_SAMPLE is TRUE, this is the bias of the current state from the Boltzmann acceptance test described above, taken to be the minimum of one and the Boltzmann factor if the new point is accepted, and one minus this number if it is rejected.

9.3.57. 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. Memory for OPTIONS->Bias_Generated [] must be created in the user module.

9.3.58. 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.

9.3.59. 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.

9.3.60. OPTIONS->Queue_Size

If ASA_QUEUE is set to TRUE, Queue_Size is a limit on the size of the FIFO queue used to store generated states. This array must be defined in the user module.

After Queue_Size has been set, and memory created in asa (), it may be changed adaptively to any number less than this. If Queue_Size is 0, then no queue is used.

9.3.61. OPTIONS->Queue_Resolution

If ASA_QUEUE is set to TRUE, Queue_Resolution is a pointer to an array of resolutions used to compare the currently generated parameters to those in the queue. This array must have space allocated and be defined in the user module. See the discussions on ASA_QUEUE and ASA_RESOLUTION on the differences in operations of these two OPTIONS.

The ASA_QUEUE OPTIONS also can be used to coarse-grain a fit, by setting high values of Queue_Resolution [].

9.3.62. OPTIONS->Coarse_Resolution

If ASA_RESOLUTION is set to TRUE, Coarse_Resolution is a pointer to an array of resolutions used to resolve the values of generated parameters. This array must have space allocated and be defined in the user module. See the discussions on ASA_QUEUE and ASA_RESOLUTION on the differences in operations of these two OPTIONS.

9.3.63. OPTIONS->Fit_Local

If FITLOC is TRUE, OPTIONS->Fit_Local is passed to cost_function (). This provides additional flexibility in deciding when to shunt asa () over to fitloc (), e.g., during multiple or recursive optimizations. As used in asa_usr.c, a value of Fit_Local set ≥ 1 is required to enter the local code.

If Fit_Local is set to 2, any better fit found by the local code better than asa () is ignored if that local fit is achieved by violating the ranges of the parameters. This additional test is much stricter than that imposed by OPTIONS->Penalty,

9.3.64. OPTIONS->Iter_Max

When FITLOC is TRUE, OPTIONS->Iter_Max determines the maximum iterations of the cost_function () by simplex ().

9.3.65. OPTIONS->Penalty

If FITLOC is TRUE, OPTIONS->Penalty determines how to weight violation of constraints, exceeding boundaries, etc.

9.3.66. Multi_Number

If MULTI_MIN is set to TRUE, then OPTIONS->Multi_Number, the number of best-generated states of the lowest sampled values of the cost function, must be set by the user. Of course this can be changed adaptively.

Note that OPTIONS->Multi_Cost and OPTIONS->Multi_Params are seeded with the highest cost function state just before entering the main annealing loop. Therefore, if OPTIONS->Multi_Number is larger than the actual number of accepted best states (new lowest values), or if a set of states are identical within the precision reported, then there will be a set of duplicate states returned which should be filtered via a unique macro.

9.3.67. OPTIONS->Multi_Cost

If MULTI_MIN is TRUE, OPTIONS->Multi_Cost save the lowest OPTIONS->Multi_Number values of the cost function.

9.3.68. OPTIONS->Multi_Params

If MULTI_MIN is TRUE, OPTIONS->Multi_Params save the parameters of the lowest OPTIONS->Multi_Number values of the cost function.

9.3.69. OPTIONS->Multi_Grid

If MULTI_MIN is TRUE, OPTIONS->Multi_Grid must be set by the user to define the resolution permitted to distinguish among parameter values of the best-generated states. However, this is overridden in asa.c, to ensure that Multi_Grid is greater or equal to EPS_DOUBLE and to OPTIONS->Coarse_Resolution if ASA_RESOLUTION is TRUE.

9.3.70. OPTIONS->Multi_Specify

If MULTI_MIN is TRUE, and if OPTIONS->Multi_Specify is set to 0, the selection of best-generated states includes all sampled instances of the cost functions. If Multi_Specify is set to 1, the selection of best-generated states is constrained to include only those with different values of the cost function.

If Multi_Specify is set to 0, and multiple best cost functions are found, the parameters of the final cost function reported will be the parameters of the last best cost function. All new best cost functions will be reported, but if ASA_PRINT_MORE is TRUE only new lower best cost functions will have additional print out.

The final values in asa.c of *best_number_generated_saved and *best_number_accepted_saved at the best cost function will be those of the first instance of the final best value. If Multi_Specify is set to 0, the parameters corresponding to this final value may not be among the set of final Multi_Params if there are more best cost functions than Multi_Number.

9.3.71. 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.

9.3.72. OPTIONS->Gener_Block

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

9.3.73. 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.

9.3.74. OPTIONS->Random_Array_Dim

When ASA_SAVE is set to TRUE, OPTIONS->Random_Array_Dim defines the dimension of the array used to hold shuffled random numbers used by the random number generator defined in asa_usr.c. The default is to use SHUFFLE defined in asa_usr.h for Random_Array_Dim.

9.3.75. OPTIONS->Random_Array

When ASA_SAVE is set to TRUE, OPTIONS->Random_Array holds the shuffled random numbers used by the random number generator defined in asa_usr.c. The default is to set the pointer of the static array random_array at the top of asa_usr.c.

9.3.76. OPTIONS->Asa_Recursive_Level

When using ASA recursively, it often is useful to be able to keep track of the level of recursion.

If ASA_SAVE is set to TRUE, set Asa_Recursive_Level to be 0 at the most inner shell and increment at each successive outer shell. Then, ASA_SAVE will take effect at the most outer recursive shell.

10. User Module

This module includes asa_usr.c, asa_usr.h, asa_usr_cst.c, and asa_usr_asa.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.

10.1. int main(int argc, char **argv)

In main (), set up your initializations and calling statements to asa. The files asa_usr.c, asa_usr.h, and asa_usr_cst.c provide a sample () function, 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.

main () returns 0 for a normal exit, -1 if there was a calloc allocation error in asa.c, or -2 if there was a calloc allocation error in asa_usr.c.

```
10.2. int asa_main(
#if ASA_TEMPLATE_LIB
    double *main_cost_value,
    double *main_cost_parameters,
    int *main_exit_code
#endif
)
```

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 asa_usr.c is closed off, and a different main ()/asa_main () procedure in asa_usr.c is used.

asa_main () returns 0 for a normal exit, -1 if there was a calloc allocation error in asa.c, or -2 if there was a calloc allocation error in asa_usr.c.

If you require parameters to be passed by asa_main () back to your main program, e.g., cost_value, *cost_parameters, *exit_code, etc., then these can be added as *main_cost_value, *main_cost_parameters, *main_exit_code, with memory allocated in your own main (), etc. Such use is illustrated by ASA_TEMPLATE_LIB.

At the end of the asa_usr.c, part of ASA_TEMPLATE_LIB is an example of a main () program that could call asa_main ().

randflt () calls resettable_randflt () each time to actually implement the RNG. This is to provide the capability of getting the same runs if the same multiple calls to asa () are made, e.g., when using ASA_LIB set to TRUE. To enforce this, asa_main () should call resettable_randflt (rand_seed, 1) at the beginning of each run.

```

10.3. int 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,
#if OPTIONS_FILE_DATA
    FILE *ptr_options,
#endif
    USER_DEFINES * USER_OPTIONS)

```

Before calling `asa`, the user must allocate storage and initialize some of the passed parameters. A `sample ()` function 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 `asa_usr.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 `asa_usr.h` instead of in `main ()` in `asa_usr.c`, but since the `asa` module only passes local parameters to facilitate recursive use, this approach is taken here as well.

`initialize_parameters ()` returns 0 for a normal exit or -2 if there was a `calloc` allocation.

```

10.4. int 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,
#if OPTIONS_FILE_DATA
    FILE *recur_ptr_options,
#endif
    USER_DEFINES * RECUR_USER_OPTIONS)

```

This procedure is used only if `SELF_OPTIMIZE` is `TRUE`, and is constructed similar to `initialize_parameters ()`. `recur_initialize_parameters ()` returns 0 for a normal exit or -2 if there was a `calloc` allocation.

```

10.5. double cost_function(
    double *x,
    double *parameter_lower_bound,
    double *parameter_upper_bound,
    double *cost_tangents,
    double *cost_curvature,
    ALLOC_INT *parameter_dimension,
    int *parameter_int_real,
    int *cost_flag,
    int *exit_code,
    USER_DEFINES *USER_OPTIONS)

```

10.5.1. cost_function

You can give any name to `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 `cost_function`.

10.5.2. *x

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

10.5.3. double *parameter_lower_bound

See discussion below on `parameter_upper_bound`.

10.5.4. double *parameter_upper_bound

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 for sampling, although it still will be printed out with other parameters if `OPTIONS` permit. Whenever this occurs, within `asa()` the effective dimension defining the size of the search space is reduced by the number of parameters so removed.

For example, if your parameter constraints are correlated in subsets, you can implement the following in your `cost_function()` in `asa_usr.c` or `asa_usr_cst.c`. Immediately upon entering `cost_function()` after receiving a full set of new parameters from `asa()`, check all correlated subsets of points. If some correlated subset is not valid, for all parameters that do satisfy your constraints, save `parameter_lower_bound []` and `parameter_upper_bound []` in some temporary arrays, set `parameter_lower_bound []` equal to `parameter_upper_bound []`, `*cost_flag` equal to `FALSE`, and return to `asa()`. Have `asa()` keep recalculating the new subsets of points until all subsets are valid. Then, reset `parameter_lower_bound []` and `parameter_upper_bound []` from the temporary arrays, and continue on with the rest of `cost_function()`. You may wish to perform this with the Quenching `OPTIONS` turned on, so that you also can accordingly adaptively modify the annealing rates using the new effective number of active parameters.

10.5.5. double *cost_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 the default implementation of ASA, the tangents are used to determine the relative reannealing among parameters.

10.5.6. double *cost_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, e.g., as a "covariance matrix" for the fitted parameters.

When the `DEFINE_OPTIONS Curvature_0` option is set to `TRUE` the curvature calculations are bypassed. This can be useful for very large spaces.

10.5.7. ALLOC_INT *parameter_dimension

An integer containing the dimensionality of the state space is passed next. 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`.

Setting the lower bound equal to the upper bound will remove a parameter from consideration. Whenever this occurs, within `asa()` the effective dimension defining the size of the search space is reduced by the number of parameters so removed.

10.5.8. int *parameter_int_real

This integer array is passed next. 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. (Derivatives can be taken with `INTEGER_TYPE` (1) only if `OPTIONS->Include_Integer_Parameters` is set to `TRUE`.) For example, this can be useful to exclude discontinuous functions from being reannealed. Note that the values of the parameters and their ranges are always passed as doubles in the code, but their values will be integral for those parameters which are defined as `INTEGER_TYPE` or `INTEGER_NO_REANNEAL`.

If a system parameter is discrete, but not a simple set of sequential integers, then it may be necessary to define a transformation within the cost function in terms of a new parameter which is a set of sequential integers. Also note that `ASA_RESOLUTION` may be used in some such cases, which overrides the use of `parameter_int_real`. Then, this new parameter, instead of the original discrete parameter, can be passed between `asa()` and the `cost_function()`. The (approximate) range of the transformed parameter must be reflected in the values assigned to `parameter_lower_bound []` and `parameter_upper_bound []`, as discussed above. Of course, this transformation may be supplemented by constraints that can be enforced using the `*cost_flag` in the user module, as discussed below.

10.5.9. *cost_flag

`cost_flag` is the address of an integer. In `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 as 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 `*cost_flag` enters with a value of `FALSE`, then the user is expected to calculate the `cost_tangents []` array before exiting that particular evaluation of the cost function. An example is provided with the `ASA_TEMPLATE_SAMPLE` example.

10.5.10. int *exit_code

The address of this integer is passed to `asa`. On return it contains the code for the reason `asa` exited. When `ASA_PRINT` is `TRUE`, the value is printed out.

`NORMAL_EXIT` = 0. Given the criteria set largely by the `DEFINE_OPTIONS`, the search has run its normal course, e.g., when `Limit_Acceptances` or `Limit_Generated` is reached.

`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.

IMMEDIATE_EXIT = 5. The user has set OPTIONS->Immediate_Exit to TRUE, or deleted file asa_exit_anytime when ASA_EXIT_ANYTIME is TRUE.

INVALID_USER_INPUT = 7. The user has introduced invalid input. When entering asa (), a function asa_test_asa_options () checks out many user-defined parameters and OPTIONS, and prints out invalid OPTIONS when ASA_PRINT is set to TRUE.

INVALID_COST_FUNCTION = 8. The user has returned a value of the cost function to asa () which is not a valid number, e.g., not between -MAX_DOUBLE and MAX_DOUBLE. Or, the user has returned a value of a parameter no longer within its proper range (excluding cases where the user has set the lower bound equal to the upper bound to remove a parameter from consideration).

INVALID_COST_FUNCTION_DERIV = 9. While calculating numerical cost derivatives, a value of the cost function was returned which is not a valid number, e.g., not between -MAX_DOUBLE and MAX_DOUBLE. Or, while calculating numerical cost derivatives, a value of a parameter no longer within its proper range (excluding cases where the user has set the lower bound equal to the upper bound to remove a parameter from consideration) was set. In such cases, review the bounds of parameters and the OPTIONS used to determine how derivatives are calculated and used.

When calculating derivatives, an invalid cost function, i.e., returning *cost_flag = FALSE from the cost function, will exit asa() with *exit_code INVALID_COST_FUNCTION_DERIV. Also, when calculating derivatives, no extra test is performed to check that parameters are within their lower and upper bounds (since meshes for derivatives may exceed these bounds). If this is not desired, then within the cost function, within a test for USER_OPTIONS->Locate_Cost == 3, a decision may be made whether to return this *exit_code.

CALLOC_FAILED = -1. Calloc memory allocation has failed in asa.c. This error will call Exit_ASA(), the location will be printed to stdout, and asa () will return the double -1 to the calling program. In asa_usr.c, if asa () returns this *exit_code a warning will be printed both to stdout and to USER_OUT. Note that if a calloc memory allocation fails in asa_usr.c, this error will call Exit_USER() to print the location to stdout and then return -2.

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.

10.5.11. 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.

**10.6. 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 ().

**10.7. double user_random_generator(
 LONG_INT *rand_seed)**

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, including an alternative RNG, are provided with the code in the user module.

The random seed, first defined in the user module, is passed to asa (), where it can be reset. This can be useful for some parallelization algorithms.

Most random number generators should be “warmed-up” by calling a set of dummy random numbers. Here, randflt () does this when it is first called, or when it is fed a negative random seed (which can be a useful flag for asa_seed () below).

randflt () calls resettable_randflt () each time to actually implement the RNG. This is to provide the capability of getting the same runs if the same multiple calls to asa () are made, e.g., when using ASA_LIB set to TRUE. To enforce this, asa_main () should call resettable_randflt (rand_seed, 1) at the beginning of each run.

**10.8. LONG_INT asa_seed(
 LONG_INT seed)**

When ASA_LIB is set to TRUE asa_seed () becomes available to set the initial random seed. This can be useful for repeated calls to asa_main () described above. If an absolute value of seed greater than 0 is given to a call of asa_seed, then seed is used as the initial random seed; otherwise asa_main () will by default create its own random seed, and the user need not be concerned with any call to asa_seed (). An example of use is given when ASA_TEMPLATE_LIB is set to true.

If the value of seed is less than 0, this signals the default randflt () to initialize the array of seeds to be shuffled. This is especially useful when using repeated calls to asa_main () when ASA_LIB is TRUE.

**10.9. double user_cost_schedule(
 double test_temperature,
 const void *OPTIONS_TMP)**

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.

**10.10. double recur_user_cost_schedule(
double test_temperature,
const void *OPTIONS_TMP)**

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.

**10.11. void user_acceptance_test(
double *uniform_test,
double current_cost,
double *parameter_lower_bound,
double *parameter_upper_bound,
ALLOC_INT *number_parameters,
const void *OPTIONS_TMP)**

If USER_ACCEPTANCE_TEST [FALSE] is set to TRUE, then this function must determine the acceptance test, e.g., as an alternate to the Boltzmann test. USER_ACCEPT_ASYNC_EXP [FALSE] is an example of a class of such modifications.

**10.12. void recur_user_acceptance_test(
double *uniform_test,
double recur_current_cost,
double *recur_parameter_lower_bound,
double *recur_parameter_upper_bound,
ALLOC_INT *recur_number_parameters,
const void *OPTIONS_TMP)**

If USER_ACCEPTANCE_TEST [FALSE] and SELF_OPTIMIZE [FALSE] both are set to TRUE, then this function must determine the acceptance test, e.g., as an alternate to the Boltzmann test.

**10.13. double user_generating_distrib(
LONG_INT *seed,
ALLOC_INT *number_parameters,
ALLOC_INT index_v,
double temperature_v,
double init_param_temp_v,
double temp_scale_params_v,
double parameter_v,
double parameter_range_v,
double *last_saved_parameter,
const void *OPTIONS_TMP)**

If USER_GENERATING_FUNCTION [FALSE] is set to TRUE, then this function (referred by USER_OPTIONS->Generating_Distrib in the user module) must define the probability distribution (or whatever algorithm is required) to use for generating new states, e.g., as an alternate to the ASA distribution.

Even mild modifications to the ASA distribution can be useful, e.g., slowing down the annealing schedule by taking a fractional root of the current temperature.

The passed array last_saved_parameter [] contains all the last saved parameters, which are sometimes required for algorithms requiring decisions based on all current parameters. This permits the use of the ASA code for heuristic algorithms that may violate its sampling proof, but nevertheless are useful to process some complex systems within a SA framework.

**10.14. double recur_user_generating_distrib(
LONG_INT *seed,
ALLOC_INT *number_parameters,
ALLOC_INT index_v,
double temperature_v,
double init_param_temp_v,
double temp_scale_params_v,
double parameter_v,
double parameter_range_v,
double *last_saved_parameter,
const void *OPTIONS_TMP)**

If USER_GENERATING_FUNCTION [FALSE] and SELF_OPTIMIZE [FALSE] both are set to TRUE, then this function (referred by RECUR_USER_OPTIONS->Generating_Distrib in the user module) must define the probability distribution (or whatever algorithm is required) to use for generating new states, e.g., as an alternate to the ASA distribution.

The passed array last_saved_parameter [] contains all the last saved parameters, which are sometimes required for algorithms requiring decisions based on all current parameters. This permits the use of the ASA code for heuristic algorithms that may violate its sampling proof, but nevertheless are useful to process some complex systems within a SA framework.

**10.15. int user_reanneal_cost(
double *cost_best,
double *cost_last,
double *initial_cost_temperature,
const void *OPTIONS_TMP)**

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

**10.16. int recur_user_reanneal_cost(
double *cost_best,
double *cost_last,
double *initial_cost_temperature,
const void *OPTIONS_TMP)**

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

**10.17. double user_reanneal_params(
double current_temp,
double tangent,
double max_tangent,
const void *OPTIONS_TMP)**

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

**10.18. double recur_user_reanneal_params(
double current_temp,
double tangent,
double max_tangent,
const void *OPTIONS_TMP)**

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

```

10.19. final_cost = asa(
    cost_function,
    randflt,
    rand_seed,
    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 `asa_usr.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`.

```

10.20. double asa(
    double (*user_cost_function) (
        double *, double *, double *, double *, double *,
        ALLOC_INT *, int *, int *, int *, USER_DEFINES *),
    double (*user_random_generator) (LONG_INT *),
    LONG_INT *rand_seed,
    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_usr_asa.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 ()`.

10.20.1. double (*user_cost_function) ()

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

10.20.2. double (*user_random_generator) ()

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

10.20.3. LONG_INT *rand_seed

A pointer to the random seed, defined in the user module, must be passed next.

10.20.4. 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 a 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.

10.21. 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 using `PRINT_FREQUENCY` and `RECUR_PRINT_FREQUENCY` in `asa_usr.c`. See the `ASA-NOTES` file for changes in these routines that may be required on some particular systems.

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

When `ASA_TEMPLATE_SAMPLE` is 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.

10.23. void adaptive_options(USER_DEFINES * USER_OPTIONS)

When `ADAPTIVE_OPTIONS` is set to `TRUE`, it is possible to change relevant `OPTIONS` in the `USER_DEFINES` struct in `asa_usr_asa.h` during run time, by including an `asa_adaptive_options` file. This can be very useful to efficiently determine how to tune long runs.

The function call to `adaptive_options(USER_OPTIONS)` should be placed at the top of your cost function. Some examples of the format of lines in the comma-delimited `asa_adaptive_options` file are given just above the `adaptive_options` function in `asa_usr.c`.

11. Bug Reports and Help With ASA

Please read this `ASA-README.[]` file and the `ASA-NOTES` file before seeking help or reporting bugs.

I 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 `ASA-NOTES` file will contain updates for some standard machines. I welcome your bug reports and constructive critiques regarding this code.

Without seeing any specific output from your system, of course I can't say anything specific. While I cannot promise that I can spend the time to join your quest to the very end to insure you get the global optimal point for your system, I can state that I will at least get back to you after seeing some print out. Many times, it is useful to add `ASA_PRINT_MORE=TRUE` to your compile `DEFINE_OPTIONS` to get more info, and enclose relevant portions your `ASA_OUT` file with your query. (See the section `Use of Documentation for Tuning` above.)

My policy, as stated in the `ASA-NOTES` file, is to keep all my help as well as my commercial work with ASA on a confidential basis. I do not divulge any names of people or information about (legal!) projects unless I am given specific permission to do so or unless the work is published. This policy promotes feedback on ASA which benefits all users as well as those individuals seeking help.

"Flames" will be rapidly quenched.

12. References

- [1] L. Ingber, "Adaptive Simulated Annealing (ASA)," Global optimization C-code, Caltech Alumni Association, Pasadena, CA (1993). URL <http://www.ingber.com/#ASA-CODE>.
- [2] L. Ingber, "Very fast simulated re-annealing," *Mathematical Computer Modelling* **12**, pp. 967-973 (1989). URL http://www.ingber.com/asa89_vfsr.pdf.
- [3] L. Ingber, "Statistical mechanics of combat and extensions" in *Toward a Science of Command, Control, and Communications*, ed. C. Jones, pp. 117-149, American Institute of Aeronautics and Astronautics, Washington, D.C. (1993). ISBN 1-56347-068-3. URL http://www.ingber.com/combata93_c3sci.pdf.
- [4] L. Ingber and B. Rosen, "Very Fast Simulated Reannealing (VFSR)," Global optimization C-code, University of Texas, San Antonio, TX (1992). URL <ftp://ringer.cs.utsa.edu/pub/rosen/vfsr.tar.gz>.
- [5] L. Ingber, "Simulated annealing: Practice versus theory," *Mathematical Computer Modelling* **18**, pp. 29-57 (1993). URL http://www.ingber.com/asa93_sapvt.pdf.
- [6] L. Ingber, "Generic mesoscopic neural networks based on statistical mechanics of neocortical interactions," *Physical Review A* **45**, pp. R2183-R2186 (1992). URL http://www.ingber.com/smni92_mnn.pdf.
- [7] M. Wofsey, "Technology: Shortcut tests validity of complicated formulas," *The Wall Street Journal* **CCXXII**, p. B1 (24 September 1993).
- [8] L. Ingber, "Adaptive simulated annealing (ASA): Lessons learned," *Control and Cybernetics* **25**, pp. 33-54 (1996). This was an invited paper to a special issue of the Polish journal Control and Cybernetics on "Simulated Annealing Applied to Combinatorial Optimization." URL http://www.ingber.com/asa96_lessons.pdf.
- [9] L. Ingber, "Data mining and knowledge discovery via statistical mechanics in nonlinear stochastic systems," *Mathl. Computer Modelling* **27**, pp. 9-31 (1998). URL http://www.ingber.com/path98_datamining.pdf.
- [10] L. Ingber, "Adaptive Simulated Annealing (ASA) and Path-Integral (PATHINT) Algorithms: Generic Tools for Complex Systems," ASA-PATHINT Lecture Plates, Lester Ingber Research (2001). Invited talk U Calgary, Canada, April 2001. URL http://www.ingber.com/asa01_lecture.pdf.
- [11] L. Ingber, "Statistical mechanical aids to calculating term structure models," *Physical Review A* **42**, pp. 7057-7064 (1990). URL http://www.ingber.com/markets90_interest.pdf.
- [12] L. Ingber, "Statistical mechanics of nonlinear nonequilibrium financial markets: Applications to optimized trading," *Mathematical Computer Modelling* **23**, pp. 101-121 (1996). URL http://www.ingber.com/markets96_trading.pdf.
- [13] L. Ingber, "Canonical momenta indicators of financial markets and neocortical EEG" in *Progress in Neural Information Processing*, ed. S.-I. Amari, L. Xu, I. King, and K.-S. Leung, pp. 777-784, Springer, New York (1996). Invited paper to the 1996 International Conference on Neural Information Processing (ICONIP'96), Hong Kong, 24-27 September 1996. ISBN 981 3083-05-0. URL http://www.ingber.com/markets96_momenta.pdf.
- [14] L. Ingber and R.P. Mondescu, "Optimization of Trading Physics Models of Markets," *IEEE Trans. Neural Networks* **12**(4), pp. 776-790 (2001). Invited paper for special issue on Neural Networks in Financial Engineering. URL http://www.ingber.com/markets01_optim_trading.pdf.
- [15] L. Ingber and R.P. Mondescu, "Automated internet trading based on optimized physics models of markets" in *Intelligent Internet-Based Information Processing Systems*, ed. R.J. Howlett, N.S. Ichalkaranje, L.C. Jain, and G. Tonfoni, pp. 305-356, World Scientific, Singapore (2003). Invited paper. URL http://www.ingber.com/markets03_automated.pdf.
- [16] A.F. Atiya, A.G. Parlos, and L. Ingber, "A reinforcement learning method based on adaptive simulated annealing" in *Proceedings International Midwest Symposium on Circuits and Systems (MWCAS)*, December 2003, IEEE CAS, Cairo, Egypt (2003). URL http://www.ingber.com/asa03_reinforce.pdf.

- [17] L. Ingber, "Statistical Mechanics of Financial Markets (SMFM): Applications to Trading Indicators and Options," SMFM Lecture Plates, Lester Ingber Research (2001). Invited talk U Calgary, Canada, April 2001. Invited talk U Florida, Gainesville, April 2002. Invited talk Tulane U, New Orleans, January 2003. URL http://www.ingber.com/markets01_lecture.pdf.
- [18] L. Ingber, "Statistical mechanics of neocortical interactions: A scaling paradigm applied to electroencephalography," *Physical Review A* **44**, pp. 4017-4060 (1991). URL http://www.ingber.com/smni91_eeg.pdf.
- [19] L. Ingber, "Statistical mechanics of neocortical interactions: Canonical momenta indicators of EEG," *Physical Review E* **55**, pp. 4578-4593 (1997). URL http://www.ingber.com/smni97_cmi.pdf.
- [20] L. Ingber, "Statistical mechanics of neocortical interactions: Training and testing canonical momenta indicators of EEG," *Mathl. Computer Modelling* **27**, pp. 33-64 (1998). URL http://www.ingber.com/smni98_cmi_test.pdf.
- [21] L. Ingber, "Statistical Mechanics of Neocortical Interactions (SMNI): Multiple Scales of Short-Term Memory and EEG Phenomena," SMNI Lecture Plates, Lester Ingber Research (2001). Invited talk U Calgary, Canada, April 2001. URL http://www.ingber.com/smni01_lecture.pdf.
- [22] L. Ingber, "Statistical mechanics of neocortical interactions (SMNI): Testing theories with multiple imaging data," *NeuroQuantology Journal* (2008). Invited paper. URL http://www.ingber.com/smni08_tt.pdf.
- [23] L. Ingber, "Statistical mechanics of neocortical interactions: Nonlinear columnar electroencephalography," *NeuroQuantology Journal* **7**(4), pp. 500-529 (2009). URL http://www.ingber.com/smni09_nonlin_column_eeg.pdf.
- [24] L. Ingber, H. Fujio, and M.F. Wehner, "Mathematical comparison of combat computer models to exercise data," *Mathematical Computer Modelling* **15**(1), pp. 65-90 (1991). URL http://www.ingber.com/combat91_data.pdf.
- [25] L. Ingber and D.D. Sworder, "Statistical mechanics of combat with human factors," *Mathematical Computer Modelling* **15**(11), pp. 99-127 (1991). URL http://www.ingber.com/combat91_human.pdf.
- [26] L. Ingber, "Statistical mechanics of combat and extensions" in *Toward a Science of Command, Control, and Communications*, ed. C. Jones, pp. 117-149, American Institute of Aeronautics and Astronautics, Washington, D.C. (1993). ISBN 1-56347-068-3. URL http://www.ingber.com/combat93_c3sci.pdf.
- [27] M. Bowman and L. Ingber, "Canonical momenta of nonlinear combat" in *Proceedings of the 1997 Simulation Multi-Conference, 6-10 April 1997, Atlanta, GA*, Society for Computer Simulation, San Diego, CA (1997). URL http://www.ingber.com/combat97_cmi.pdf.
- [28] L. Ingber, "Statistical Mechanics of Combat (SMC): Mathematical Comparison of Computer Models to Exercise Data," SMC Lecture Plates, Lester Ingber Research (2001). URL http://www.ingber.com/combat01_lecture.pdf.
- [29] L. Ingber, "Real Options for Project Schedules (ROPS)," Report 2007:ROPS, Lester Ingber Research, Ashland, OR (2007). URL http://www.ingber.com/markets07_rops.pdf.
- [30] L. Ingber, "Real Options for Project Schedules (ROPS)," *International Journal of Science, Technology & Management* (2010).
- [31] L. Ingber, "Trading in Risk Dimensions (TRD)," Report 2005:TRD, Lester Ingber Research, Ashland, OR (2005). URL http://www.ingber.com/markets05_trd.pdf.
- [32] L. Ingber, "Trading in Risk Dimensions" in *The Handbook of Trading: Strategies for Navigating and Profiting from Currency, Bond, and Stock Markets*, ed. G.N. Gregoriou, pp. 287-300, McGraw-Hill, New York (2010).
- [33] L. Ingber, "Ideas by Statistical Mechanics (ISM)," Report 2006:ISM, Lester Ingber Research, Ashland, OR (2006). URL http://www.ingber.com/smni06_ism.pdf.

- [34] L. Ingber, “Ideas by Statistical Mechanics (ISM),” *J Integrated Systems Design and Process Science* **11**(3), pp. 22-45 (2007). Special Issue: Biologically Inspired Computing.
- [35] L. Ingber, “AI and Ideas by Statistical Mechanics (ISM)” in *Encyclopedia of Artificial Intelligence*, ed. J.R. Rabuñal, J. Dorado, and A.P. Pazos, pp. 58-64, Information Science Reference, New York (2008). ISBN 978-1-59904-849-9.
- [36] L. Ingber, “Adaptive Simulated Annealing” in *Stochastic global optimization and its applications with fuzzy adaptive simulated annealing*, ed. H.A. Oliveira, Jr., A. Petraglia, L. Ingber, M.A.S. Machado, and M.R. Petraglia, pp. 33-61, Springer, New York (2012). Invited Paper. URL http://www.ingber.com/asall_options.pdf.
- [*] Code and reprints can be retrieved via WWW from <http://www.ingber.com/>.