

BAM user manual and installation guide v. 2025.7.27

July 27, 2025

For information about this software, contact Nick Sahinidis at niksah@minlp.com.

Contents

1	Introduction	1
1.1	Licensing and software requirements	3
1.2	Installation	3
2	Algorithms implemented	3
3	Running BAM	4
4	Example input file	5
5	Input file grammar	5
6	BAM data and options specification statements	6
6.1	Required parameters	6
6.2	Optional vector parameters	7
6.3	Optional data specifications	7
7	BAM output	9
7.1	BAM screen output	9
8	Termination conditions and error messages	10
9	Bibliography	12

1 Introduction

The purpose of BAM (Branch-And-Model) is to solve optimization problems for which the objective function or constraints are not algebraically available. BAM can be used to optimize a model defined via a black-box simulator or laboratory experiment. The solver is designed especially for problems for which the simulator is expensive to run and is therefore imperative to minimize the number of calls to the black box during the course of the algorithm. BAM has been developed

having in mind the optimization of complex simulation models, optimization with laboratory data, and hyperparameter tuning for complex machine learning models.

The class of optimization problems addressed by BAM is often referred to as derivative-free optimization, black-box optimization, simulation optimization, and data-driven optimization. Optimization algorithms for this class of problems are classified into local search and global search algorithms. Local search algorithms improve a current trial solution in its neighborhood using direct search or model-based methods. Direct search methods propose sample points for evaluation that form a particular geometry. These algorithms include Nelder-Mead and generalized pattern search. Model-based search methods use evaluated points to build a typically linear or quadratic surrogate model and propose sample points that optimize the model. Some examples are trust-region methods and the implicit filtering method. Global-search methods incorporate global exploration steps to escape from local optima. These algorithms include Bayesian optimization, genetic algorithms, and algorithms based on domain partitioning.

Inspired by the success of branch-and-bound algorithms for algebraic optimization, BAM implements a domain partitioning algorithm for data-driven optimization. The algorithm relies on a box-subdivision algorithm that is guaranteed to converge to a global minimum, even for problems involving discontinuities, nonconvexities, and integer or categorical variables. BAM's unique subdivision scheme allows a flexible partition of the search space and exploits previously evaluated points. The algorithm approximates the function of interest using a collection of local surrogate models around each evaluated point. The implementation relies on the use of Automated Learning of Algebraic MOdels (ALAMO) to generate simple and accurate algebraic models. ALAMO's ability to produce sparse models from a large collection of potential basis functions allows the construction of more complex surrogate models than quadratic ones if necessary. With the model-based search, BAM exploits the local trends of the objective function using the information from the evaluated points and usually results in fast convergence during solution refinement. The algorithm relies on the leading global optimization solver BARON to solve its local surrogate models to global optimality to identify the most promising regions for further exploitation at every iteration. Compared to other data-driven algorithms, BAM performs a more effective search, especially for problems with higher dimensions.

BAM can:

- solve optimization problems defined via a simulation or experimental black-box system
- use previously collected data to initialize the search
- call a user-specified function (simulator) to collect measurements
- enforce variable bounds

The problems addressed by the software have long been studied in the fields of optimization, statistics, design of experiments, and machine learning. Whereas existing techniques from this literature are routinely used to fit data to local or global models and use these models to guide the search, the main challenges addressed by the BAM software are in determining where to run the simulations or experiments, what models to fit, how to optimize these models, when to use local refinement or global exploration, and how to perform all these tasks in an entirely automated manner without user intervention.

1.1 Licensing and software requirements

The code is available for download at <http://minlp.com/bam>. The same URL provides information about licensing the software.

1.2 Installation

Install BAM and the BAM license file in any directory of your choice and add it to your path. On Windows, BAM's installer will take care of these steps for you. On Linux and OSX systems, unzip the BAM download and place the files it contains in a directory in your path. Users should not open the license file in an editor as this may invalidate the license in certain operating systems. For all operating systems, make sure that BAM and the BAM license file are readable by all intended users on your machine.

For a silent (non-interactive) installation on Windows, download the installer and run it as follows from the command line:

```
bam-windows64.exe /SILENT
```

If an installation log file is additionally desired, it can be generated by the command:

```
bam-windows64.exe /SILENT /LOG=filename
```

where `filename` is the desired name for the log file. In both cases, Windows will still request permission to run the executable before the silent installer is launched.

2 Algorithms implemented

The BAM algorithm starts with optional user-specified previously evaluated or proposed starting points, or a space-filling LHS initialization strategy. Each iteration gradually includes additional evaluated points found by model-based search and volume-based search. There are four main steps in each iteration:

1. BAM's unique box subdivision scheme is utilized to partition the search space into a set of boxes, with each box corresponding to the area of influence of each previously evaluated point. The subdivision scheme makes use of all previously evaluated points and each box contains exactly one evaluated point. The algorithm performs a dense search using the proposed subdivision scheme. Dense sampling is the essential pillar for guaranteeing the convergence of the algorithm.
2. After the partitioning of the search domain, BAM selects the most promising boxes after excluding boxes for which a Lipschitzian argument guarantees suboptimality (no Lipschitz constant is assumed by the algorithm). Boxes that pass this test are used for local surrogate model construction. This step avoids unnecessary model building and prevents delays by avoiding searching around non-global basins.

3. BAM builds a collection of local surrogate models around the evaluated point in each selected box. Models are fitted using only information from other nearby evaluated points, requiring no further points to be evaluated. Multiple local surrogate models allow the behavior of each surrogate function to be simple compared to utilizing a single surrogate function for the entire feasible domain. In a small neighborhood, the function is assumed to behave independently for each variable. We use ALAMO for model construction, an algorithm that is proven to construct simple and accurate models by selecting a small subset of basis functions from a potentially large group. Building surrogate models exploits the local trends of the objective function using the information from the evaluated points and results in demonstrated faster convergence.
4. Based on the sizes of boxes and the objective values obtained by minimizing the local surrogate models using BARON, BAM identifies a list of candidate points for evaluation at each iteration. The newly sampled points are added to the collection utilized for the box subdivision at the next iteration.

The bibliography at the end of this document offers more details of the methodology implemented in BAM and demonstrates the advantages of this methodology in comparison to currently utilized approaches, including deterministic and stochastic search algorithms. BAM relies in a unique way in advanced machine learning software (ALAMO) and advanced optimization software (BARON). Both software are embedded in the BAM distribution. For information on the algorithms utilized by ALAMO and BARON, see <https://minlp.com>.

3 Running BAM

Users can utilize BAM only from the command line. BAM reads model data and algorithmic options from a text file in a simple format. Even though it is not required, it is strongly recommended that all BAM input files have the extension ‘.bam.’ If the input file is named ‘test.bam’ and the BAM executable is named ‘bam,’ issuing the command

```
bam test
```

or

```
bam test.bam
```

results in BAM parsing test.bam and solving the problem. In addition to screen displays, BAM can also provide results in the listing file ‘test.lst’ that is generated during the run. The .lst file is always stored in the execute directory, even when the .bam file is in a different path. During execution, BAM creates and utilizes a directory for storing various work files. When calling BAM, the user may optionally include a second command line argument in order to specify BAM’s working directory:

```
bam test.bam myscratchdir
```

where ‘myscratchdir’ denotes the name of BAM’s scratch directory. If this argument is not specified, BAM will create and utilize a directory named ‘test.bamsr’ in the execute directory. If the scratch directory exists, it is erased in the beginning of the run. At the end of the run, BAM will delete its scratch directory.

4 Example input file

The following file is referred to as ‘e1.bam’ and pertains to optimizing the Six-Hump Camel function—often called camel6—a classic benchmark in nonlinear optimization, especially for testing global optimization algorithms. The function is defined for two variables, x_1 and x_2 , as $f(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2$. The input domain we consider is $x_1 \in [-3, 3]$ and $x_2 \in [-1.5, 1.5]$. The function has two global minima at approximately $(-0.0898, 0.7126)$ and $(0.0898, -0.7126)$, both with a function value of approximately -1.0316 .

Following the comment line that begins with an exclamation mark, the file specifies that we are dealing with a problem with two variables ($nvars = 2$). The minimum and maximum allowed values of the variables are then specified. An initial sampling data set is specified and is comprised of a single data point ($ndata = 1$); the starting point we specify is the origin. The user provides a simulator named camel6.exe that reads the values of x_1 and x_2 (one variable per line) from a file named input.in, calculates the function, and returns the function value in a file named output.out. We are willing to invest no more than 80 function calls ($maxevals$) to optimize this problem.

```
! camel6 is a classic NLP benchmark
nvars      2
xmin    -3 -1.5
xmax     3 1.5
ndata     1
BEGIN_DATA
  0 0
END_DATA
dataprovder /usr/local/testlibs/dfo/all/camel6.exe
datain input.in
dataout output.out
maxevals 80
```

Many additional examples of BAM input files can be found at <https://minlp.com/black-box-optimization>

5 Input file grammar

The following rules should be followed when preparing an BAM input file:

- The name of the input file should include its exact path location if the file is not present in the execute directory.

- The name of the input file should not exceed 1000 characters in length.
- The input is not case sensitive.
- Most options are entered one per line, in the form of ‘keyword’ followed by ‘value’. Certain vector options are entered in multiple lines, starting with ‘BEGIN_<keyword>’, followed by the vector input, followed by ‘END_<keyword>’.
- Certain options must appear first in the input file. This requirement is discussed explicitly in option descriptions provided below.
- With the exception of arguments involving paths, character-valued options should not contain spaces.
- Blank lines, white space, and lines beginning with *, #, % or ! are skipped. Inline comments that are preceded by #, % or ! are permitted in any line that contains alphanumeric options. Blocks of comment lines are allowed using ‘BEGIN_COMMENT’, followed by the block of comment lines, followed by ‘END_COMMENT’; these comment blocks are entirely ignored by BAM.

6 BAM data and options specification statements

6.1 Required parameters

The following parameters must be specified in the input file.

Parameter	Description
NVARS	Number of model input variables. NVARS must be a positive integer and defines the number of optimization variables. NVARS must be provided before providing variable bounds and starting points.
DATAPROVIDER	Complete path of executable that will provide function values at requested points. The DATAPROVIDER executable must be capable of reading file DATAIN and writing file DATAOUT. DATAIN is provided by BAM and contains the values of the variables, one per line, where a function evaluation is requested. In DATAOUT, the DATAPROVIDER must return a single line with the value of the function at the evaluated point. BAM will execute the DATAPROVIDER in a scratch directory it generates during its run; hence, the DATAPROVIDER should not rely on any relative paths in order to access other programs or files.

6.2 Optional vector parameters

The following parameters may be specified in the input file only after NVARs has already been specified.

Parameter	Description
XMIN	Row vector specifying minimum values for each of the input variables. This should contain exactly NVARs entries that are space delimited.
XMAX	Row vector specifying maximum values for each of the input variables. This should contain exactly NVARs entries that are space delimited.
XISINT	Row vector of 0/1 flags that specify which input variables, if any, BAM should treat as integers. For integer inputs, BAM's sampling will be restricted to integer values.

XMIN and XMAX define the domain of the optimization problem. If left unspecified, XMIN and XMAX will be initialized by BAM using BAM's MAXBOUND parameter described below. If left unspecified, all values of XISINT are initialized to zero.

6.3 Optional data specifications

This section describes optional parameters pertaining to the particular problem being solved.

Option	Description	Default
NDATA	Number of unevaluated starting points specified by the user. NDATA must be a nonnegative integer.	0
NEVALDATA	Number of evaluated starting points provided by the user. NEVALDATA must be a nonnegative integer.	0
BATCH	A 0–1 indicator. If 1, BAM will propose evaluation points and terminate the run.	0
DATAIN	Name of input file for the DATAPROVIDER. BAM generates this file.	input.txt
DATAOUT	Name of output file for the DATAPROVIDER. BAM expects the DATAPROVIDER to provide this file after each call.	output.txt
MAXEVALS	Maximum number of new function evaluations permitted during this run. MAXEVALS must be at least as large as NDATA.	2500
MAXTIME	Maximum total execution time allowed in seconds. This time includes all steps of the algorithm, including time to read problem, preprocess data, solve optimization subproblems and machine learning subproblems, and print results.	∞

MAXITER	Maximum number of algorithmic iterations permitted during this run. A value of -1 implies no limit on the number of algorithmic iterations (MAXEVALS and MAXTIME limits apply).	-1
MAXPROFAILS	Maximum number of DATAPROVIDER failures to tolerate before terminating. A value of -1 implies no limit on the number of DATAPROVIDER failures.	-1
MAXBOUND	Maximum absolute value allowed for variable bounds. If lower bounds for variables are left unspecified, they will be set equal to $-\text{MAXBOUND}$. If upper bounds for variables are left unspecified, they will be set equal to MAXBOUND . If variable bounds are specified through XMIN and XMAX, their values will be adjusted to enforce MAXBOUND .	10000
SAMPLER	Technique to be used for space filling initial sampling. Possible values are 1 through 7, with the following meaning: <ol style="list-style-type: none"> 1. Latin hypercube sampling, 2. Sobol sampling, 3. Faure sampling, 4. Halton sampling, 5. Hammersley sampling, 6. Niederreiter2 sampling, 7. random sampling. 	2
PRESET	A value indicating that the DATAPROVIDER was not able to compute a specific requested point. This value must be carefully chosen to be an otherwise not realizable value for the function at hand.	-111111.
PRINT_TO_FILE	A 0–1 indicator. Output is directed to the listing file if this option is set to 1; if set to 0, no output is sent to the listing file.	1
PRINT_TO_SCREEN	A 0–1 indicator. Output is directed to the screen if this option is set to 1; if set to 0, no output is sent to the screen.	1
PREVALS	In addition to the listing file, BAM optionally prints a file containing function evaluations. Each line of this file contains an evaluation number, followed by current CPU time, the values of the problem variables, the corresponding objective function value, and the best objective function value found during the search. This file is printed only if PREVALS is positive, in which case printing occurs every PREVALS function calls.	1

The parser is not case sensitive. For example, nvars, nVARS, nVars, and NVARS are equivalent. If the parameter NDATA is set, then a data section must follow subsequently in the input file

with precisely NDATA rows, one for each starting point (NVARs space separate values for all problem variables) specified in the following form:

```
BEGIN_DATA
```

```
:
```

```
END_DATA
```

If the parameter NEVALDATA is set, then a data section must follow subsequently in the input file with precisely NDATA rows, one for each previously evaluated point (NVARs space separate values for all problem variables, followed by space, followed by the corresponding objective function value) specified in the following form:

```
BEGIN_EVALDATA
```

```
:
```

```
END_EVALDATA
```

7 BAM output

7.1 BAM screen output

The screen output below is obtained for problem e1.bam.

```
*****
BAM version 2025.7.27. Built: WIN-64 Sun Jul 27 00:48:34 EDT 2025
Running on machine PONTIOS
```

```
If you use this software, please cite:
Ma, K., L. M. Rios, A. Bhosekar, N. V. Sahinidis and S. Rajagopalan,
Branch-and-Model: A derivative-free global optimization algorithm,
Computational Optimization and Applications, 85, 337-367, 2023.
```

```
BAM is powered by the ALAMO and BARON software from http://www.minlp.com/
*****
Licensee: Nick Sahinidis at The Optimization Firm, LLC, niksah@minlp.com.
*****
Reading input data
Checking input consistency and initializing data structures
```

```
Call Time      Best function value  Merit function
```

1	0.310000E-01	0.00000	0.00000
2	0.660000E-01	0.00000	124.650
3	0.100000	0.00000	0.562500E-01
4	0.126000	0.00000	0.562500E-01
5	0.158000	0.00000	1.44272
6	0.177000	0.00000	13.5541
7	0.205000	0.00000	2.14585
8	0.236000	0.00000	8.35093
9	0.264000	0.00000	2.78074
10	0.284000	0.00000	3.00344
11	0.315000	0.00000	38.4445
12	0.331000	0.00000	0.315905
13	0.364000	0.00000	3.00344
14	0.394625	0.00000	2.78074
15	0.425625	0.00000	0.315905
16	0.456625	0.00000	38.4445
17	0.479625	0.00000	19.7398
18	0.506625	0.00000	9.58945
19	0.536625	0.00000	2.11318
20	0.567625	0.00000	0.523732
21	0.589625	0.00000	3.25023
22	0.621625	0.00000	65.7231
23	0.646625	0.00000	1.06178
24	0.678625	0.00000	0.588561
25	0.709625	0.00000	7.07076
26	0.731625	0.00000	1.87340
27	0.757625	0.00000	18.5692
28	0.788625	0.00000	3.26407
29	0.820625	0.00000	2.24336
30	0.852625	0.00000	2.66223

The software first reports the version, platform, and compilation date of the executable, followed by credits. Then, after reading the input data, a consistency check is run on the problem data and, if passed, the data structures are initialized. Subsequently, information is provided for each function evaluation: the current cumulative CPU time, the current best known objective function value for the problem, and the value of the objective function computed in the most recent call to the DATAPROVIDER. At the end of the run, BAM provides a termination code and explanatory message.

8 Termination conditions and error messages

Errors in the input file are reported on the screen and/or the listing file in the form of “warnings” and “severe errors.” BAM attempts to continue execution despite warnings. If the errors are severe, the program execution is stopped and, if relevant, the line of the input file where the fatal error occurred is displayed. The input file should be checked even if the warnings are not severe,

as the problem might have been parsed in a way other than it was intended to be. Detailed error messages are provided in that case.

BAM may terminate with any of the following termination codes, all of which are self-explanatory:

1. Licensing error. A valid license is required in order to run this software.
2. BAM must be called with one or two command line arguments.
3. BAM input file name must be no longer than 1000 characters.
4. BAM input file not found.
5. BAM listing file cannot be opened.
6. Unable to open trace file.
7. Insufficient memory to allocate data structures.
8. BAM input file cannot be opened.
9. Keyword too long in input file.
10. No keyword may be specified more than once.
11. Number of variables (NVARs) must be specified before specifying XMIN values.
12. Number of variables (NVARs) must be specified before specifying XMAX values.
13. Number of variables (NVARs) must be specified before specifying XISINT.
14. END_DATA missing or incomplete DATA section.
15. Number of data points (NDATA) must be specified before the DATA section of the input file.
16. Number of variables (NVARs) must be specified before the DATA section of the input file.
17. Keyword not recognized in input file.
18. Only one DATA section is allowed.
19. Input data file missing required keyword(s).
20. At least one of DATAPROVIDER and BATCH must be specified.
21. Error while attempting to access the BAM scratch directory.
22. Error while attempting to access the BAM execution directory.
23. XMAX-XMIN for all variables must be positive.
24. XDATA must be in the range [XMIN, XMAX].
25. XEVALDATA must be in the range [XMIN, XMAX].

26. Premature end of input file.
27. Each line of the input file must contain no more than 10000 characters. Longer data records may be split into multiple lines using & at the end of a line to signify continuation of the record in the next line.
28. Syntax error in input file.
29. Inline comments must be preceded by ! or #.
30. Solver reached limit on function calls.
31. Input value in error in input file.
32. Error while attempting to write the input file for the data provider.
33. Error while attempting to read the output file of the data provider.
34. Error while attempting to access the data provider.
35. Error while trying to copy file to disk.
36. Error while attempting to write file to disk.
37. Error while attempting to read file from disk.
38. Error while trying to run ALAMO.
39. Error while trying to run BARON.
40. Too many iterations without progress.
41. Maximum CPU time (MAXTIME) exceeded.
42. Numerical difficulties. Please report to niksah@minlp.com.
43. NEVALDATA is nonzero but XEVALDATA were not provided.
44. NDATA is nonzero but XDATA were not provided.
45. Run interrupted by user.

9 Bibliography

The following is a partial list of BAM-related publications that describe the algorithms implemented in the software, the theory behind them, and some related applications.

1. S. Amaran, N. V. Sahinidis, B. Sharda and S. J. Bury. Simulation optimization: A review of algorithms and applications. *Annals of Operational Research*, 240, 351-380, 2016.
2. A. Cozad, N. V. Sahinidis, and D. C. Miller. Learning surrogate models for simulation-based optimization. *AIChE Journal*, 60, 2211-2227, 2014.

3. A. Cozad, N. V. Sahinidis, and D. C. Miller. A combined first-principles and data-driven approach to model building. *Computers & Chemical Engineering*, 73, 116–127, 2015.
4. K. Ma, L. M. Rios, A. Bhosekar, N. V. Sahinidis and S. Rajagopalan. Branch-and-Model: A derivative-free global optimization algorithm. *Computational Optimization and Applications*, 85, 337-367, 2023.
5. K. Ma, L. M. Rios, H. Zheng, N. V. Sahinidis and S. Rajagopalan. Model-and-Search: A derivative-free local optimization algorithm. *Computational Optimization And Applications*,
<https://doi.org/10.1007/s10589-025-00686-9>.
6. N. Ploshkas and N. V. Sahinidis. Review and comparison of algorithms and software for mixed-integer derivative-free optimization. *Journal of Global Optimization*, 82, 433-462, 2022.
7. L. M. Rios and N. V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56, 1247-1293, 2013.
8. Y. Zhang and N. V. Sahinidis. Solving continuous and discrete nonlinear programs with BARON. *Computational Optimization and Applications*,
<https://doi.org/10.1007/s10589-024-00633-0>.