Efficient Implementation of a Gibbs Sampler for Multivariate Truncated Gaussian and of the Gaussian Adaptation Algorithm

Patrick Plattner, ETH Zürich, plpatric@student.ethz.ch

June 24, 2011

Abstract

Executing the Gibbs sampler or Gaussian Adaptation in Matlab can be a time consuming task. The aim is to reduce this execution time by reimplementing these two algorithms in Fortran.

Contents

1 Definition of the Project

This project is a "Research in Computer Science" project, course number 263-0600-00L, conducted within the Master of Computer Science curriculum at ETH Zurich. The objective of the project is concerned with efficient reimplementation (and possible extensions) of two algorithms in MEX/FORTRAN based on existing MATLAB code. First, an efficient Gibbs sampler for truncated multivariate Gaussian for arbitrary linear constraints has to be reimplemented. Second, the original Gaussian Adaptation algorithm has to be reimplemented. Slight modifications to the algorithm that reduce the complexity of the algorithm from $O(N^3)$ to $O(N^2)$ are conceivable.

2 Gibbs Sampler for Multivariate Truncated Gaussian

The reimplementation of the Gibbs sampler is based on the following literature and files.

- Efficient Gibbs Sampling of Truncated Multivariate Normal with Application to Constrained Linear Regression by Rodriguez, Davis and Scharf [3]
- Matlab mvntrunc.m by Müller [5]

2.1 Algorithm

The Gibbs sampler consists of the following steps. The definition of the used variables can be found in the appendix A.

1. If the covariance matrix C is given it gets decomposed into the eigenvalues and eigenvectors $C = B\Lambda B^T$ where Λ is a diagonal matrix.
2. The matrix T gets calculated such it holds $TCT^T = I$ where I is the identity matrix. This results in $T = \frac{1}{\sqrt{\Lambda}} \cdot B^T$. With this matrix each dimension of the transformed $x \sim \mathcal{N}(\mu, \sigma C)$ can be sampled independently $z = Tx \sim \mathcal{N}(T\mu, \sigma I)$. With T constructed this way it's inverse is $T^{-1} = \sqrt{\Lambda} \cdot B$.
3. The matrix D gets calculated such the constraints hold in the transformed space $Dz \leq b \Leftrightarrow Ax \leq b$ where $z = Tx$. This results in $D = AT^{-1}$.
4. For each dimension of z a new value is sampled.
   (a) The lower and upper bounds are calculated.
   (b) The new value is sampled from a one dimensional truncated Gaussian (2.1.1) $z(i) \sim \mathcal{N}(lb, ub, (T\mu)_i, \sigma)$.
5. The newly sampled z gets transformed back $x = T^{-1}z$.
6. Both, the x and z samples can be returned.

2.1.1 1-D Truncated Gaussian Sampler

To sample from a Truncated Gaussian the following function are used.

$$\begin{aligned} \mathrm{erf}(x) &= \tfrac{2}{\sqrt{\pi}} \textstyle\int_0^x e^{-t^2} dt\,, \\ \Phi(x) &= \tfrac{2}{\sqrt{2\pi}} \textstyle\int_0^x e^{\frac{-t^2}{2}} dt\,, \\ \Phi(x) &= \tfrac{1}{2}\mathrm{erf}(\tfrac{x}{\sqrt{2}})\,, \\ \Phi^{-1} &= \sqrt{2}\mathrm{er}f^{-1}(2x-1)\,, \end{aligned}$$

Where Φ is the cumulative Gaussian and erf is the Error function [4] [9].

First with the Error function the lower and upper bounds (ya, yb) get calculated from the cumulative Gaussian distribution. Then a uniform random value $r \sim U(0,1)$ gets scaled into [ya, yb] and then calculated back to the Gaussian distribution with the inverse Error function. The result is a random variable with the distribution $z \sim \mathcal{N}(lb, ub, \mu, \sigma)$. The Error function is used because the Φ function is not implemented and there are the same up to scaling factors.

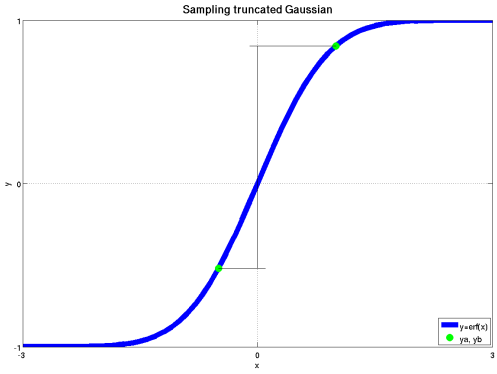

Figure 1: Visualization of the 1-D truncated Gaussian sampler

2.1.2 Statistics

The following figures show the statistical analysis of the Gibbs sampler.

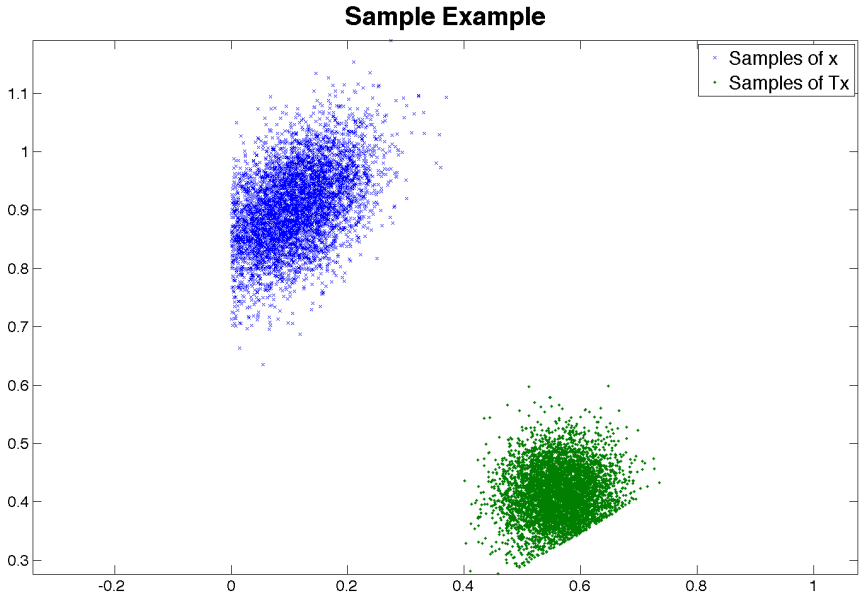

Figure 2: Sampling a truncated Gaussian

Figure 2 shows an example of 5000 samples with the constraint $x - axis > 0$. In blue (x) the resulting samples and in green (.) the transformed and dimensionally independent samples.

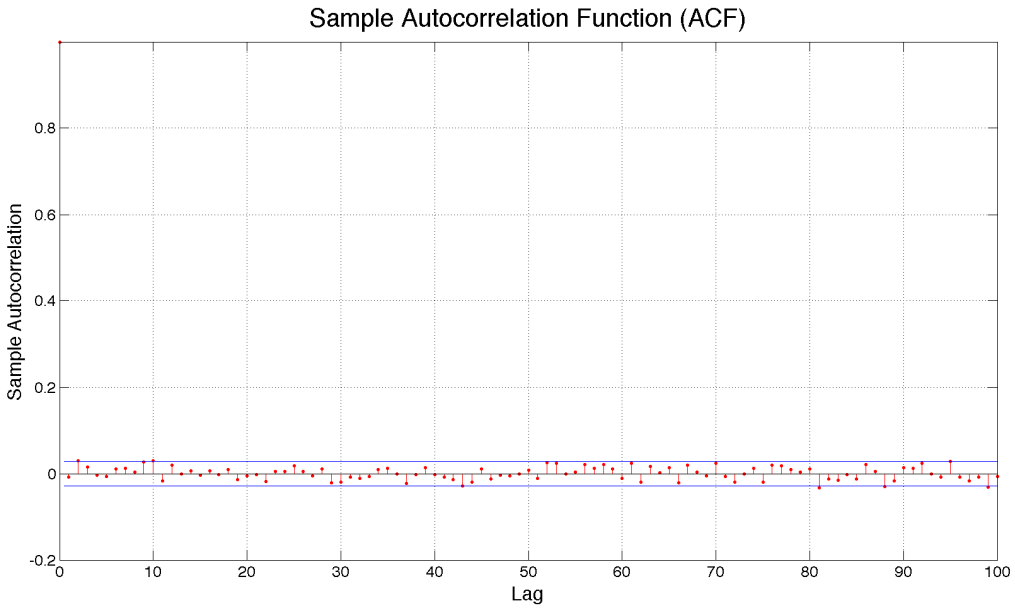

Figure 3: Auto correlation of the Gibbs sampler

Figure 3 shows the auto correlation of the drawn samples.

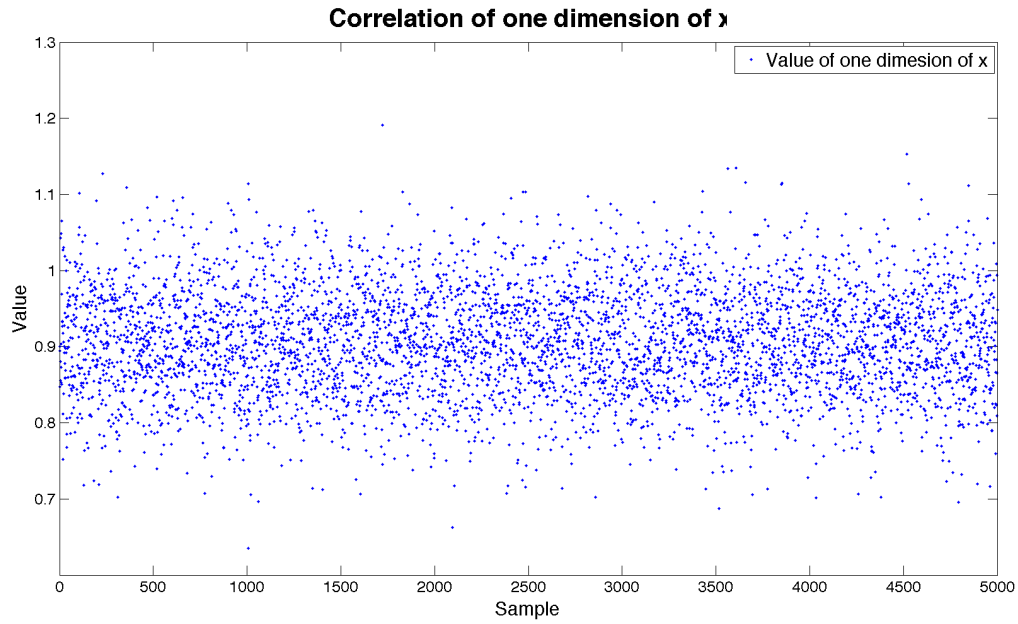

Figure 4: Correlation of the Gibbs sampler

Figure 4 plots the value of one dimension of the samples to visually show that there is no correlation between the samples.

2.2 Performance

To evaluate the performance of the reimplemented Gibbs sampler the execution time of both implementations are compared.

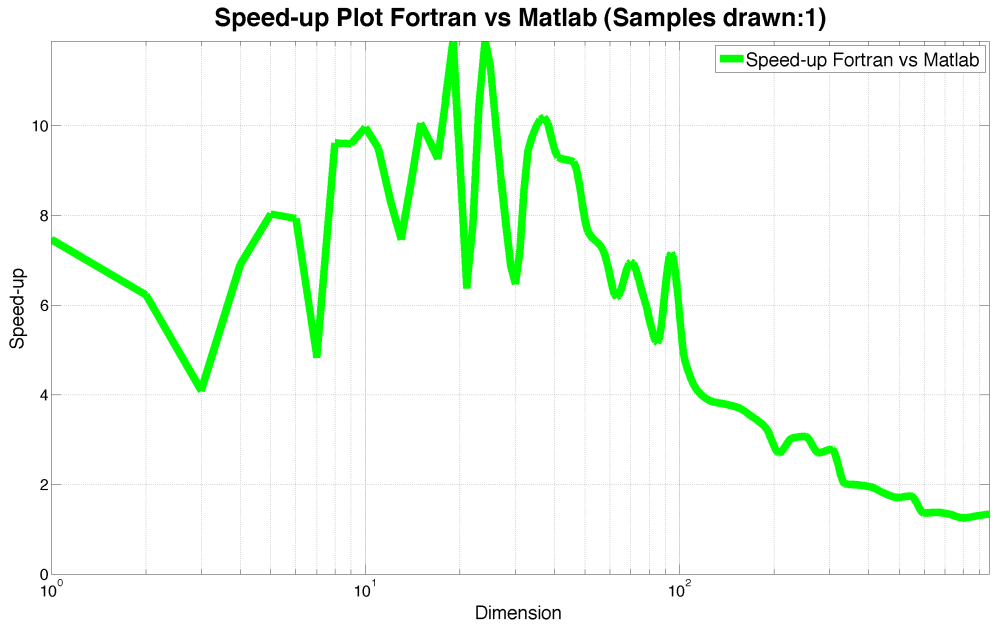

Figure 5: Speed-up of one sample on a quad core mac with 3GHz each

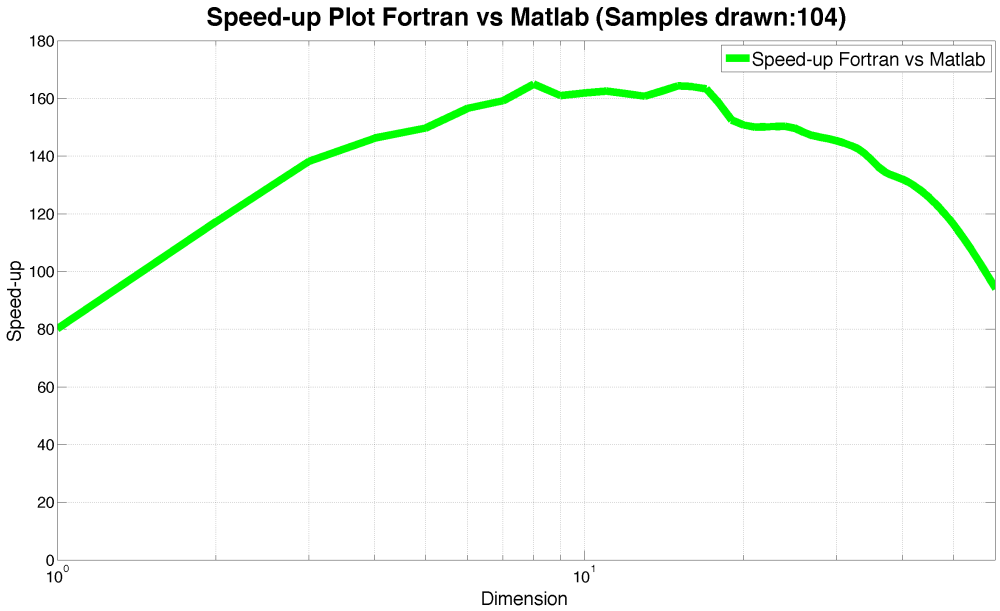

Figure 6: Speed-up of 104 samples on a quad core mac with 3GHz each

Figures 5 and 6 show the speed-up of the Fortran sampler versus the one implemented in Matlab. With one sample up to 100 dimensions the speed-up stays over 4 with a some fluctuations. After that the speed-up drops but falls never under 1.5. The more samples are drawn the greater the speed-up gets. With 104 samples the Fortran sampler is over 80 times faster.

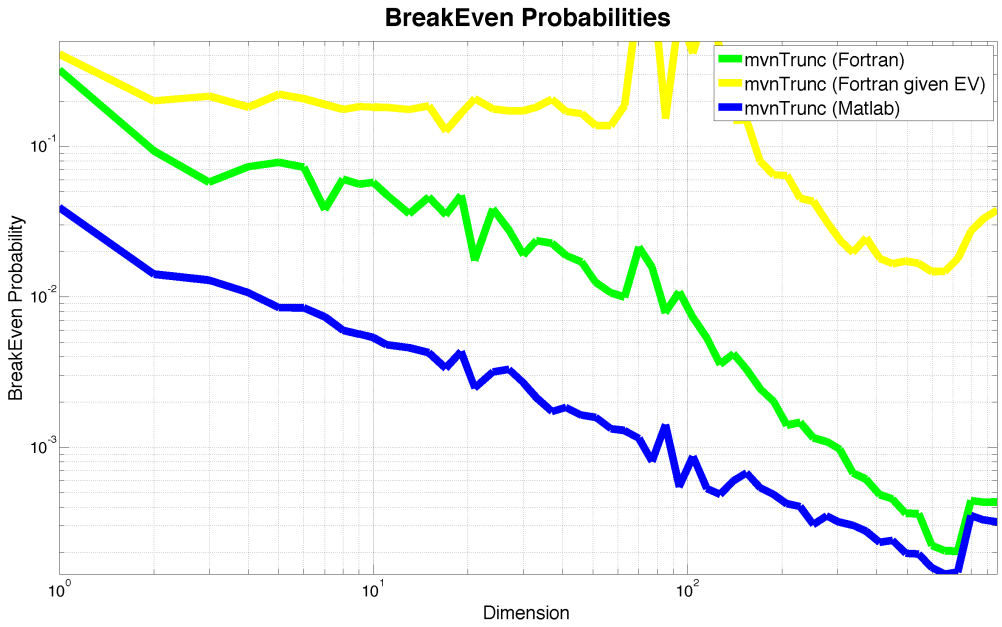

Figure 7: Break-even probabilities (loglog plot) on a quad core mac with 3GHz each

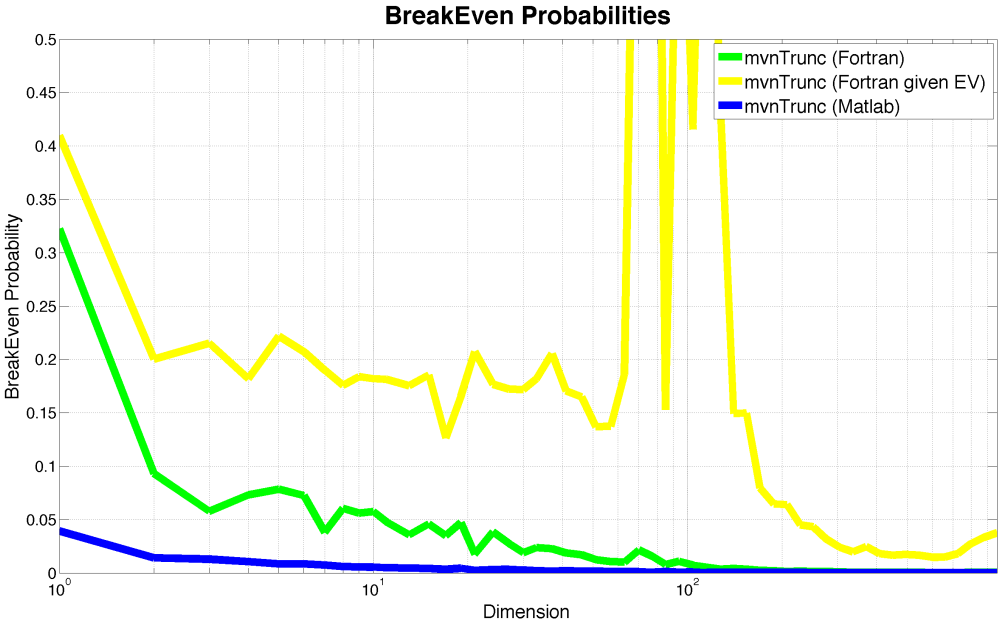

Figure 8: Break-even probabilities (semilogx plot) on a quad core mac with 3GHz each

Figures 7 and 8 show the break-even Probabilities of the Fortran and Matlab Gibbs sampler against the Rejection Sampler. The break-even probability indicates at which acceptance probability the additional effort of the Gibbs sampler in comparison to the Rejection Sampler starts to pay off.

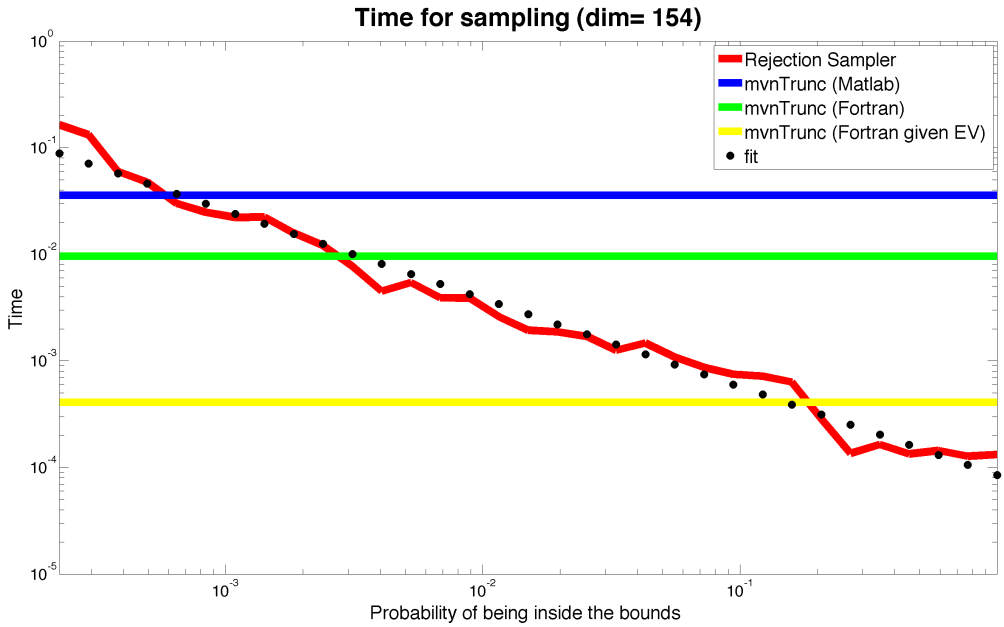

Figure 9: Execution time (s) of the three sampler on a quad core mac with 3GHz each

Figure 9 shows the execution time of the Fortran and Matlab Gibbs sampler and the Rejection Sampler at a given dimension and varying acceptance probability of the Rejection Sampler.

2.2.1 Data

The performance measurement data is located in the source folder [7] under "93_SamplerEval/". The [label] flag is used to distinguish between different test runs.

- mvnTruncPerformance.m: generates the performance measurement and saves the data into mat files.
  - plots/[label]_time.mat: stores the accumulated execution time per configuration.
  - plots/[label]_ks.mat: stores how many times each configuration has been run.
- mvnTruncPlotSpeedUp.m: uses the mat files to generate the speed-up plots.
  - plots/[label]_SpeedUp.png
- mvnTruncPlotBreakEven.m: uses the mat files to generate the break-even plots.
  - plots/[label]_BreakEvenLogLog.png
  - plots/[label]_BreakEvenSemiLog.png
  - plots/[label]_dim[*]_time.png

2.3 Usage

To show how the reimplemented Gibbs sampler is used, examples are given and the function parameters are described. The definition of the parameters and return values is located in the appendix A.

2.3.1 From Matlab

The Gibbs sampler can be used in the following four ways:

- With given covariance matrix C and the samples X as result.

```
[X] = mvntruncFortran( mu, sigma, C, A, b, ...
        numSamples, burnin, thinInterval )
```

- With given eigen-decomposition B and Λ of the covariance matrix and the samples X as result.

```
[X] = mvntruncFortran( mu, sigma, B, diag(Lambda), A, b, ...
         numSamples, burnin, thinInterval )
```

- With given covariance matrix C and the samples X and Z as result.

```
[X Z] = mvntruncFortran( mu, sigma, C, A, b, ...
           numSamples, burnin, thinInterval )
```

- With given eigen-decomposition B and Λ of the covariance matrix and the samples X and Z as result.

```
[X Z] = mvntruncFortran( mu, sigma, B, diag(Lambda), A, b, ...
           numSamples, burnin, thinInterval )
```

2.3.2 From Fortran

In Fortran, the following two subroutines can be used. Where the parameter Z is optional. The module "userdefined" (4.1) is needed.

- With given covariance matrix C

```
subroutine mvnTruncCov(mu, sigma, C, A, b,
     +   numSamples, burnin, thinInterval, X, Z)
         use userdefined
         implicit none

       type(vector), intent(in) :: mu
       real*8, intent(in) :: sigma
       type(matrix), intent(in) :: C, A
       type(vector), intent(in) :: b
       mwSize, intent(in) :: numSamples, burnin, thinInterval
       type(matrix), intent(inout) :: X
       type(matrix), intent(inout), optional :: Z
end subroutine
```

- With given eigen-decomposition B and Λ of the covariance matrix.

```
subroutine mvnTruncEigCov(mu, sigma, B, Lambda, A, b,
       +   numSamples, burnin, thinInterval, X, Z)
          use userdefined
          implicit none

       type(vector), intent(in) :: mu
       real*8, intent(in) :: sigma
       type(matrix), intent(in) :: B
       type(vector), intent(in) :: Lambda
       type(matrix), intent(in) :: A
       type(vector), intent(in) :: b
       mwSize, intent(in) :: numSamples, burnin, thinInterval
       type(matrix), intent(inout) :: X
       type(matrix), intent(inout), optional :: Z
end subroutine
```

2.4 Files

The Gibbs sampler is coded in the following Fortran files which are located in the source folder [7] under "03_sampler/".

- mvntruncFortran.F: implements the mex interface from Matlab and parses the given Matlab data structures into Fortran data structures.
- mvnTrunc.F: implements the Gibbs sampler in Fortran.
- randNTrunc.F: implements the one dimensional truncated Gaussian Sampler.

3 Gaussian Adaptation

The reimplementation of the Gaussian Adaptation algorithm is based on the following literature and files.

- Gaussian Adaptation revisited, Müller, Sbalzarini [2]
- Gaussian Adaptation black-box optimization, Müller, Sbalzarini [1]
- Matlab gaussAdapt.m by Müller [6]

3.1 Algorithm

The Gaussian Adaptation consists of the following steps.

1. Initialize the variables used in the algorithm and the figures for plotting in Matlab.
2. Iterate until the one of the strop criteria is met.
   (a) Sample $x \sim \mathcal{N}(\mu, \sigma C)$
      i. in the constraint case with the Gibbs sampler (2).
      ii. in the unconstraint case with $x = \mu + r \cdot \sqrt{\Lambda} \cdot B \cdot \text{randn}(n, 1)$
   (b) Evaluate the objective function $f = \text{fitfun}(x)$
   (c) Check for the sample to accept and adapt the parameters
   (d) Plot the current state if demanded
   (e) Check the restart conditions
   (f) Save the current values if demanded
3. Return the solution

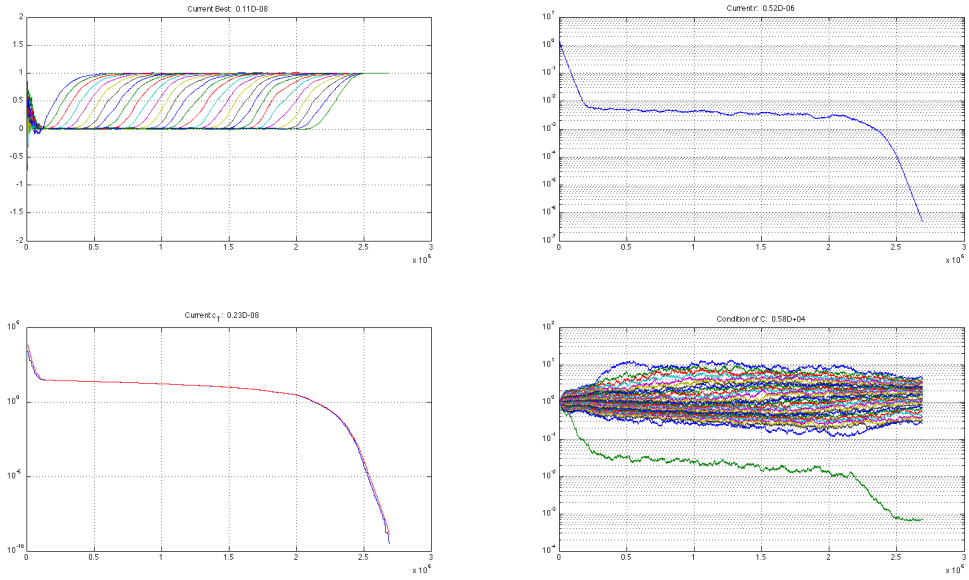

Figure 10: Example of a Gaussian Adaptation run showing mean, search radius, fitness and covariance

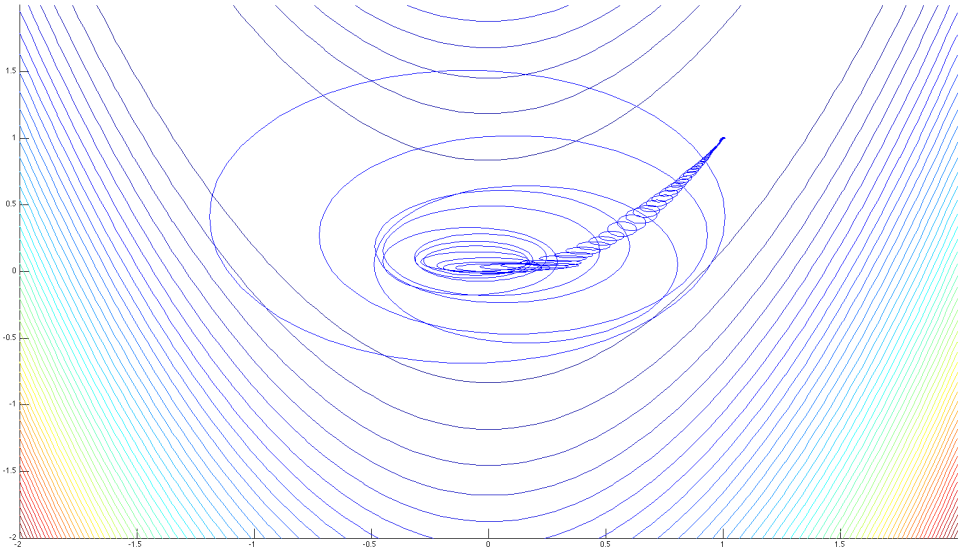

Figure 11: Example of a Gaussian Adaptation run showing the confidence ellipses of the covariance matrix

3.1.1 Difference to the original Algorithm

The Gibbs sampler (2) needs the eigen-decomposition of the covariance matrix, so the Cholesky of the covariance matrix is not used and therefore not calculated.

3.2 Performance

To evaluate the performance the two implementations are compared on different functions and dimensions.

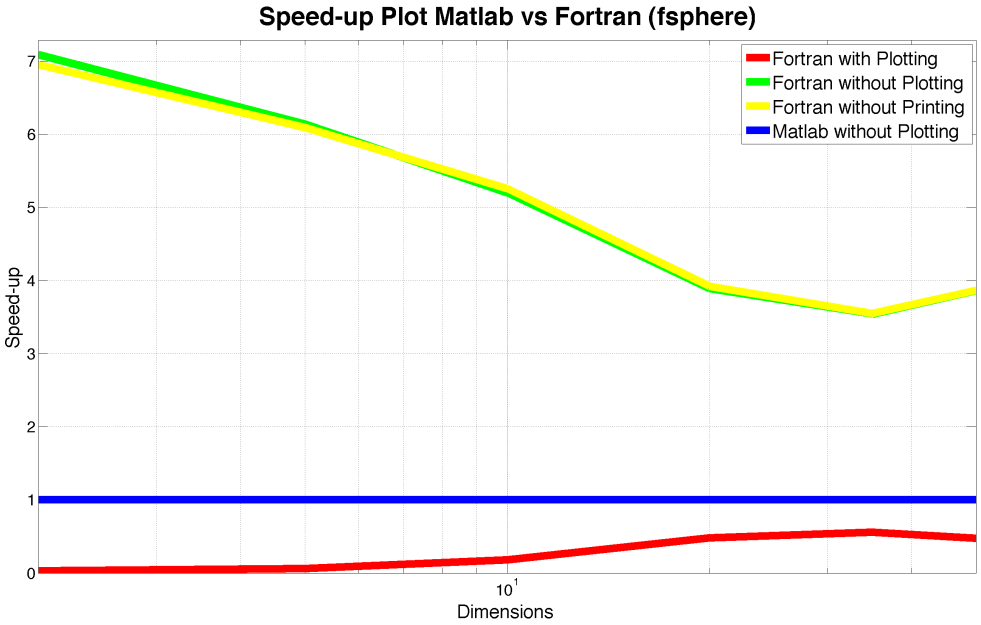

Figure 12: Speed-up on the sphere function on a quad core mac with 3GHz each

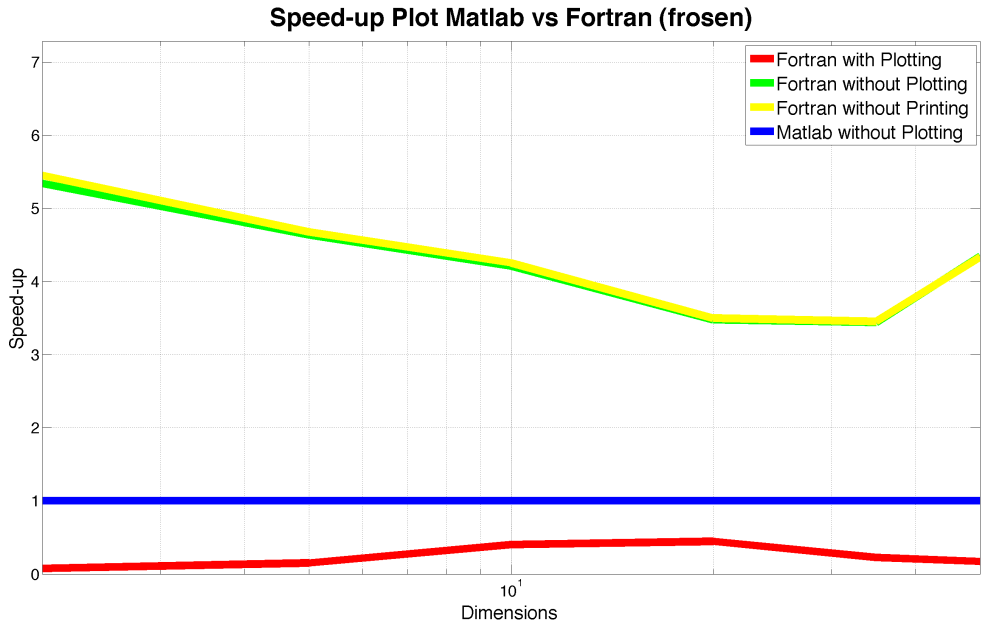

Figure 13: Speed-up on the Rosenbrock function on a quad core mac with 3GHz each

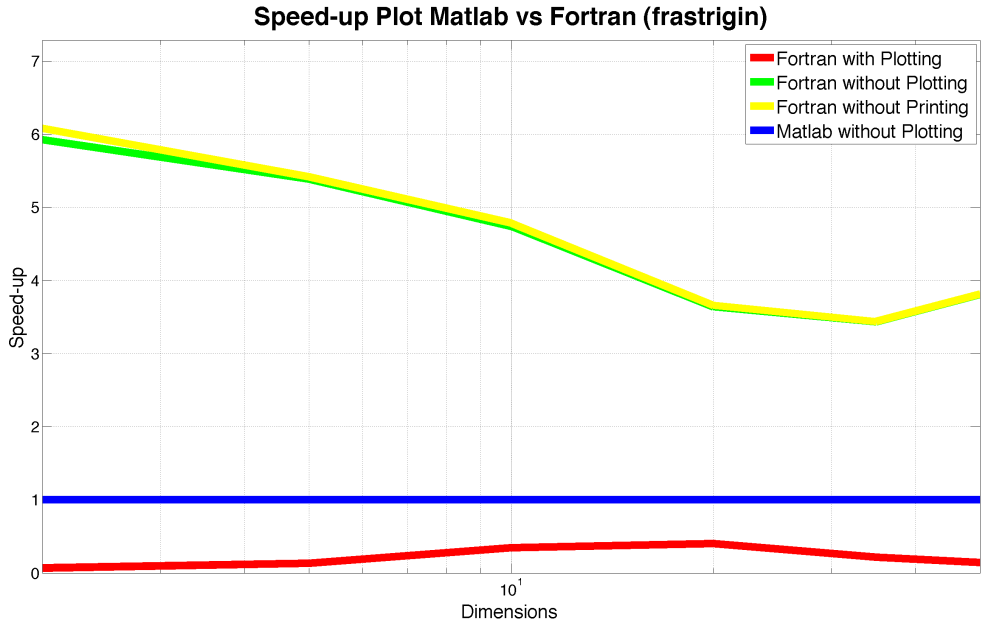

Figure 14: Speed-up on the Rastrigin function on a quad core mac with 3GHz each

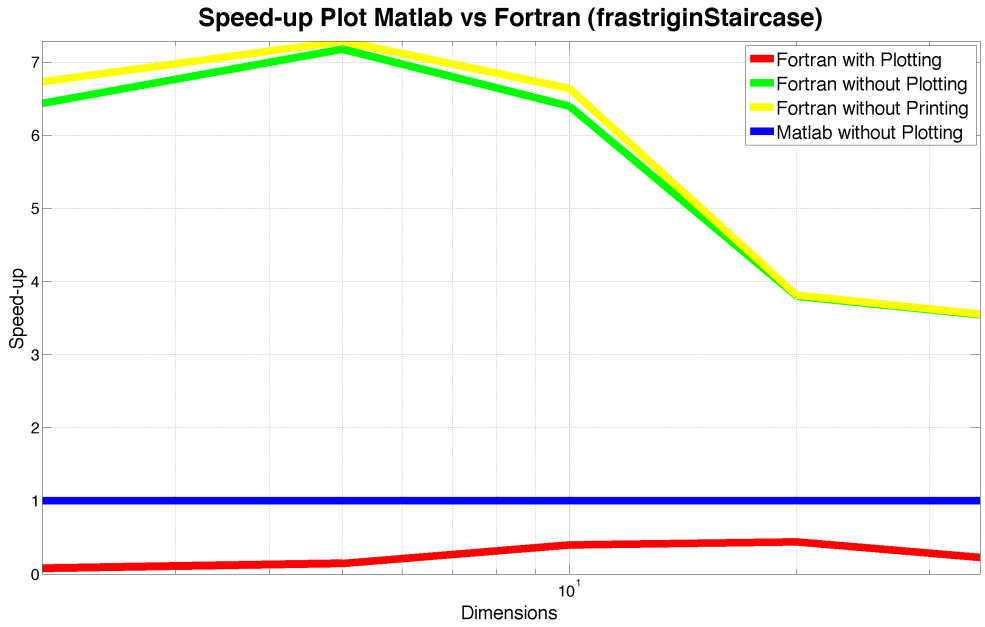

Figure 15: Speed-up on the Rastrigin Staircase function on a quad core mac with 3GHz each

Figures 12, 13, 14 and 15 show speed-up plots of various functions. The Rastrigin Staircase function is introduced in the master thesis of Daniel Zünd [10] who will later use the reimplemented Gaussian Adaptation in his thesis.

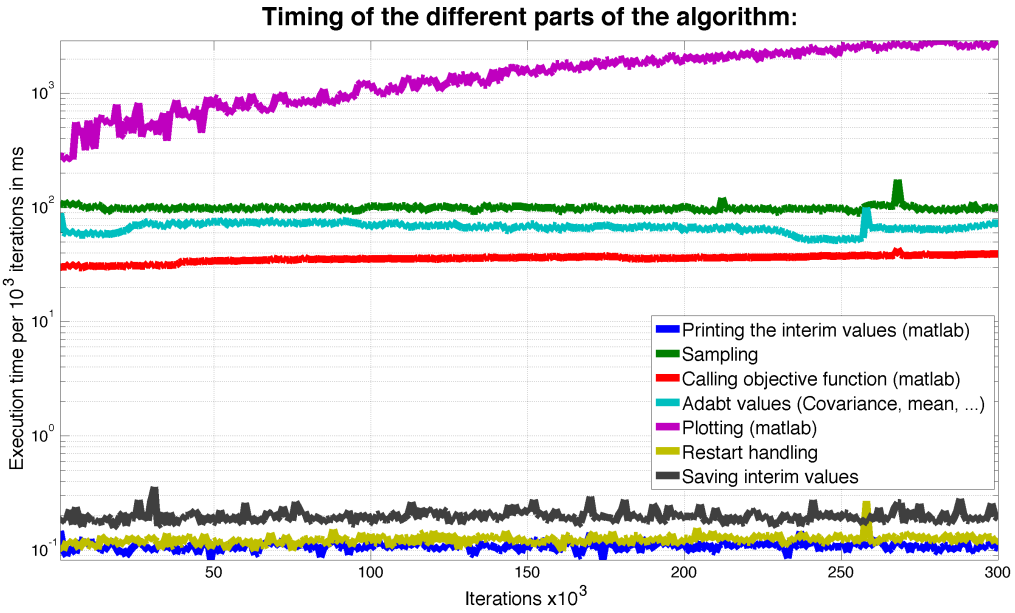

Figure 16: Timing of the different parts of the algorithm on a quad core mac with 3GHz each

Figure 16 shows how much time is spent on the different parts of the algorithm. As seen, the most time is used by plotting and displaying in Matlab what can be turned off if not needed.

3.2.1 Data

The performance measurement data is located in the source folder [7] under "94_GaAEval/". The [label] flag is used to distinguish between different test runs.

- gaussAdaptPerformance.m: generates the performance measurement and saves the data into mat files.
  - plots/[label]_time.mat: stores the accumulated execution time per configuration.
  - plots/[label]_ks.mat: stores how many times each configuration has been run.
- gaussAdaptPlotSpeedUp.m: uses the mat files to generate the speed-up plots.
  - plots/[label]_[functions]_[Bounded|Unbounded]_SpeedUp.png
- gaussAdaptPlotTime.m: uses the mat files to generate the execution time plots.
  - plots/[label]_[functions]_[Bounded|Unbounded]_SpeedUp.png

3.3 Usage

To show how the reimplemented Gaussian Adaptation algorithm is used, an example is given. It can only be used from Matlab, because it supports only Matlab functions to be used as objective function.

3.3.1 From Matlab

For a description of all the possible parameters go to section "module gaAModule" (4.2).

```
N = 10;
xStart = (rand(N,1)-0.5)*4;
inOpts.A = [eye(N); -eye(N)];
inOpts.b = ones(2*N, 1)*2;
inOpts.StopFitness = 1e-9;
inOpts.r = 4/exp(1);
inOpts.Restart = true;
inOpts.VerboseModulo = 1000;
inOpts.Saving = true;
inOpts.SavingModulo = 10;
inOpts.MaxIter = N*1e4;
inOpts.Display = true;
inOpts.Plotting = true;
inOpts.Printing = true;
inOpts.valP = 1/exp(1);
inOpts.LBounds = [-2 -2];
inOpts.UBounds = [2 2];
inOpts.c_T = 1;
inOpts.Mode = 1;
[ xMin fMin countEval out ] = gaussAdaptFortran(@function, xStart, inOpts)
```

The function has the following parameters.

- @function: Matlab function pointer of the objective function.
- xStart: starting point
- inOpts: options for Gaussian Adaptation (4.2).

The following four variables can be returned.

- xMin: is the position of the found minimum.
- fMin: is the value of the found minimum.
- countEval: is the number of iteration used.
- out: is a structure containing all the values collected during the iterations.

3.4 Files

The Gaussian Adaptation Algorithm is coded in the following Fortran files which are located in the source folder [7] under "04_GaA/".

- gaussAdaptFortran.F: implements the mex interface from Matlab and parses the given Matlab data structures into Fortran data structures.
- gaussAdapt.F: implements the Gaussian Adaptation algorithm in Fortran.
- gaAModule.F: contains the options data structure used for Gaussian Adaption.
- drawErrorEllipse.F: draw an ellipse for plotting in Matlab form a given covariance matrix and a radius.

4 Additional Modules

To make the data and configuration handling easier in Fortran, two modules have been created. One containing a matrix and a vector data type to capsulize both, the data and size information in one variable. Also a module for all the options possible for the Gaussian Adaptation algorithm has been created.

4.1 Module userdefined

To use the userdefined module it has to be included in subroutine definition. It is located in the Source Folder [7] under "10_includes/userdefined.F".

```
subroutine someName(...)
use userdefined
...
```

4.1.1 Type matrix

The matrix contains two fields of type mwSize (m, n) which hold the size for the y- and x-dimension. The type mwSize is aligned with the Matlab functions header definition (mex..., mx...). Further it contains an allocatable array (data) of type real*8 to hold the values of the matrix.

- An example to create a matrix and fill it with values from a Matlab matrix.

```
type(matrix) :: X
X%m = 10
X%n = 10
allocate( X%data(X%m, X%n), stat = alloc_status )
call mxCopyPtrToReal8( mxGetPr( X_pr ), X%data, X%m*X%n )
```

- An example to use a matrix with LAPACK [8] .

```
call DSYEV( 'V', 'L', X%n, X%data, X%m, v%data,
     +   work%data, work%m, info)
```

4.1.2 Type vector

The vector contains one field of type mwSize (m) which holds the length of the vector. The type mwSize is aligned with the Matlab functions header definition (mex..., mx...). Further it contains an allocatable array (data) of type real*8 to hold the values of the vector.

- An example to create a vector and fill it with values from a Matlab vector.

```
type(vector) :: v, work
v%m = 10
allocate( v%data(v%m), stat = alloc_status )
call mxCopyPtrToReal8( mxGetPr( v_pr ), v%data, v%m )
...
```

- An example to use vectors with LAPACK [8] .

```
call DSYEV( 'V', 'L', X%n, X%data, X%m, v%data,
     +   work%data, work%m, info)
```

4.1.3 Type locmat

The type locmat is similar to the matrix (4.1.1) with the difference that the array (data) is of the type logical.

4.1.4 Constants

Because Fortran doesn't have built in constants like NaN, Inf, eps and PI, these constants have been defined in this module.

4.2 Module gaAModule

This module contains a structure with all the configurable variables for the Gaussian Adaptation algorithm. It is subdivided into 5 sections (Stop criteria, Bounds, Output, Algorithm, Mode) and is located in the Source Folder [7] under "04_GaA/gaAModule.F".

- An example how this options structure can be used in Fortran.

```
subroutine someName(...)
use gaAModule
...
type(gaAOpts) :: opts
```

- The following table describes all the options which can be used in the algorithm. To pass them from Matlab create a Matlab structure which contains a subset of the configurable variables but without the subdivided sections. For example opts.StopFitness = 1e-9.

Variable	description
opts%stopCrit%StopFitness	Gaussian Adaptation stops when the value of the objective functions drops below this value
opts%stopCrit%MaxIter	Maximal number of iterations
opts%stopCrit%TolX	Gaussian Adaptation stops (and restarts) if the history of x drops below this value
opts%stopCrit%TolFun	Gaussian Adaptation stops (and restarts) if the the history of f drops below this value
opts%stopCrit%TolR	Gaussian Adaptation stops (and restarts) if the search radius drops below this value
opts%stopCrit%TolCon	Gaussian Adaptation stops (and restarts) if difference between the threshold and the min function value is smaller than this value
opts%stopCrit%Restart	If true restarts are allowed
opts%bounds%Active	If true the there are constraints (Ax<=b)
opts%bounds%A	Matrix for the constraints (Ax<=b)
opts%bounds%b	Vector for the constraints (Ax<=b)
opts%output%Display	If true the contour of the function gets displayed
opts%output%Plotting	If true the progress of the Gaussian Adaptation gets plotted
opts%output%Printing	If true the status gets printed
opts%output%Saving	If true the values used in Gaussian Adaptation get returned
opts%output%VerboseModulo	Defines how verbose the output (Display, Plotting and Printing) is
opts%output%SavingModulo	Defines how verbose the values are saved for return
opts%output%LBounds	Defines the lower bounds for the "Display" (two dimensional vector)
opts%output%UBounds	Defines the upper bounds for the "Display" (two dimensional vector)
opts%output%length	Defines the resolution of the "Display"
opts%algorithm%ValP	Hitting probability
opts%algorithm%r	Initial step size
opts%algorithm%MaxR	Maximal allowed step size
opts%algorithm%MaxCond	Maximal allowed condition of the covariance matrix
opts%algorithm%N_mu	Mean adaptation weight
opts%algorithm%N_C	Covariance adaptation weight
opts%algorithm%N_T	Threshold adaptation weight
opts%algorithm%inc_T	Factor for N_T increase at restart
opts%algorithm%beta	Step size increase/decrease factor
opts%algorithm%ss	Expansion of the step size on success
opts%algorithm%sf	Contraction of the step size on fail
opts%algorithm%c_T	Initial threshold
opts%algorithm%FunArgs	Additional arguments for the objective function
opts%mode%Design	Design centering (opts.mode=0)
opts%mode%Optimization	Optimization (opts.mode=1)
opts%mode%Sampling	MCMC sampling (opts.mode=2)

Table 1: Gaussian Adaptation options

A Definitions

- n: dimension of the space
- $x \in \mathbb{R}^n$: position vector
- $z \in \mathbb{R}^n$: transformed position vector
- $\mu \in \mathbb{R}^n$: means of the Gaussian distribution

- $\sigma \in \mathbb{R}^n$: sigma of the Gaussian distribution
- $C \in \mathbb{R}^{nxn}$: covariance matrix
- $B \in \mathbb{R}^{nxn}$: eigenvectors of the covariance matrix
- $\Lambda \in \mathbb{R}^{nxn}$: eigenvalues of the covariance matrix
- $T \in \mathbb{R}^{nxn}$: transformation matrix
- $lb \in \mathbb{R}$: lower bound
- $ub \in \mathbb{R}$: upper bound
- $f \in \mathbb{R}$: value of the objective function evaluation
- d: number of constraints
- $A \in \mathbb{R}^{dxn}$: constraint matrix
- $D \in \mathbb{R}^{dxn}$: transformed constraint matrix
- $b \in \mathbb{R}^d$: constraint vector
- numSamples (s): specifies how many accepted samples are drawn
- burnin: specifies how many samples are rejected before the first sample is accepted
- thinInterval: specifies what is the number of the next sample that is accepted (every x-th sample is taken)
- $X \in \mathbb{R}^{nxs}$: matrix containing the drawn samples
- $Z \in \mathbb{R}^{nxs}$: matrix containing the transformed samples

B Additional Files

To simplify the development, some files have been created.

- Source folder [7] under "03_sampler/" and "04_GaA/".
  - compile.m: Contains the command to compile the implemented algorithms into a mex executable.
  - testMvn.m, testGaA.m: Contains a test run for the algorithms.
- Source folder [7] under "10_includes/".
  - Print.F: contains subroutines to print the data structures defined in the module userdefined (4.1).
- Source folder [7] under "91_mexOpts/".
  - mexoptsMac.sh, mexoptsLinux.sh: are the Matlab mex configurations used to compile the algorithms.
- Source folder [7] under "99_Documentation/".
  - This folder contains all the files for this document.

References

[1] Ivo F. Sbalzarini Christian L. Müller. Gaussian adaptation as a unifying framework for continuous black-box optimization an adaptive monte carlo sampling. CCIB, Barcelona, Spain, 07 2010. Pages 1-8.

[2] Ivo F. Sbalzarini Christian L. Müller. *Gaussian Adaptation revisited - an entropic view on Covariance Matrix Adaptation*, pages 432–441. Springer, volume i edition, 2010.

[3] Richard A. Davis Gabriel Rodriguez-Yam and Louis L. Scharf. Efficient gibbs sampling of truncated multivariate normal with application to constrained linear regression. 2004.

[4] Intel. Erfinv. `http://software.intel.com/sites/products/documentation/hpc/compilerpro/en-us/cpp/win/mkl/refman/vml/functn_vErfInv.html`.

[5] Christian Müller Lorenz. mvntrunc.m. Given by file, 2011.

[6] Christian. L. Müller and Ivo. F. Sbalzarini. Matlab code for gaussian adaptation. `http://www.mosaic.ethz.ch/Downloads/GaA/GaA.zip`, 2010.

[7] Patrick Plattner. Source folder, research in computer science. researchProject.zip, 06 2011. Contains all the file created and collected during the project.

[8] Berkeley; Univ. of Colorado Denver; Univ. of Tennessee; Univ. of California and NAG Ltd. Linear algebra package. `http://www.netlib.org/lapack/`.

[9] Wikipedia. Error function. `http://en.wikipedia.org/wiki/Error_function`.

[10] Daniel Zünd. Unifying zeroth- and first-order variable metric algorithms for continuous non-convex optimization. Master's thesis, ETH Zürich, 2011. in progress.