A R Adaptive Multiple Importance Sampling (ARAMIS)

Luca Pozzi, University of California, Berkeley, U.S.A.
Antonietta Mira, University of Lugano, Lugano, Switzerland

Modified: November 9th, 2012, Compiled: January 21, 2013

Abstract

ARAMIS is an R package that runs the AMIS [1] algorithm. The main features of ARAMIS are parallelization and customization.
ARAMIS exploits the massively parallel structure of AMIS to improve the performance of the algorithm as it was implemented in the original paper. As a result simulation time is reduced by orders of magnitudes.

As for customization, the potential of the R language is fully exploited by ARAMIS which allows the user to taylor the software to the model which results from his or her own research setting. Target and proposal kernel can be easily specified by the user. Some working examples contained in the manual explain how this can be efficiently and easily done.

As a consequence of the flexibility and efficiency of the package, even fairly complicated problems can be accommodated, e.g. sampling from an Extreme Value (EV) Copula distribution with a mixture of EV distributions as the proposal kernel. The latter is an interesting and useful example of how the user can specify some "real-world" combination of target/proposal and it is added, in the manual, to the two working examples detailed in [1].

1 Introduction

ARAMIS is an R package that runs AMIS [1], an Adaptive Multiple Importance Sampler that uses a set of particles that evolve over an artificial time dimension. The particles are generated from an importance distribution and, properly weighted, provide a representation of (a sample from) the target distribution, as typically done in Importance Sampling (IS). What distinguishes AMIS from the other IS algorithms is that old (i.e. generated at earlier times) particles are exploited to adaptively design the importance distribution in the same spirit of Adaptive Markov Chain Monte Carlo (AMCMC) where the proposal distribution is "adapted" instead. The main difference from classic AMCMC, is that in this class time is a natural parameter (i.e. the evolution time of the stochastic process generated by the Metropolis-Hastings algorithm) and to each sample (that form the simulated path of the "Markov" chain), there is no associated importance weight. We can thus define AMIS as an hybrid between AMCMC and IS: exploiting the best from both strategies resulting in an efficient sampler with good performance also in high dimensions as proved in [1].

The main features of ARAMIS are parallelization and customization.

Parallel implementation: by its nature AMIS shows a embarrassingly parallel structure which is exploited by ARAMIS to improve the performance of the algorithm and reduce simulation time by orders of magnitudes relative to its original implementation in [1]. Following a Map-Reduce scheme the computations are sent to the independent computing units and then the results are collected and merged together. Each core takes care of an approximately equal set of particles evaluating the importance weights and merging them together to compute the new IS estimates at each iteration of the algorithm. Thus, the reduction in computational tim is directly proportional to the number of particles used in the underlying Importance Sampler.
The musketeers slogan "all for one andone for all" can be re-interpreted in light of the parallel features or ARAMIS that make it an hard to beat software to sample from complicated and high dimensional target distributions.

Customization: the potential of the R language is fully exploited by ARAMIS which allows the user to taylor the software to the model which results from his or her own research setting. Target and proposal kernel can be easily specified by the user.

As a consequence of the flexibility of the package, even fairly complicated problems can be accommodated, e.g. sampling from a Copula distribution with a mixture of extreme value distribution as the proposal kernel. The latter is an interesting and useful example of how the user can specify some "real-world" combination of target/proposal and it is added, in the manual, to the two working examples detailed in [1]. Very little ad-hoc code is needed to sample an extreme value copula distribution: package "copula" provides a way to evaluate the target distribution and package "evd" offers useful tools for extreme value distributions which are easily plugged in the ARAMIS function. This is explained extensively in Section 5

2 Adaptive Multiple Importance Sampling (AMIS)

Given a density π known up to a normalizing constant, and a function h, interest is in computing

$$\Pi(h) = \int h(x)\pi(x)\mu(dx) = \frac{\int h(x)\tilde{\pi}(x)\mu(dx)}{\int \tilde{\pi}(x)\mu(dx)}$$

when $\int h(x)\tilde{\pi}(x)\mu(dx)$ is intractable. As an alternative to Metropolis-Hastings type algorithms, to estimate $\Pi(h)$ one can rely on Importance Sampling strategies. An iid sample $x_1, \ldots, x_N \sim Q$ is generated from an importance distribution Q, s.t. $Q(dx) = q(x)\mu(dx)$ and $\Pi(h)$ is estimated by

$$\hat{\Pi}^{IS}_{Q,N}(h) = N^{-1} \sum_{i=1}^{N} h(x_i)\{\pi/q\}(x_i)$$

The IS estimator relies on alternative representation of $\Pi(h)$:

$$\Pi(h) = \int h(x)\{\pi/\mathbf{q}\}(\mathbf{x})q(x)\mu(dx)$$

IS can be generalized to encompass much more adaptive/local schemes than thought previously. Adaptivity means learning from experience, i.e., designing new IS distributions based on the performances of earlier ones. The intuition of the AMIS algorithm is, indeed, to use previous sample(s) to learn about π and construct cleaver importance distributions q in the same spirit as adaptive MCMC where cleaver proposal distributions are designed based on the past history of simulated stochastic process.

Following is a description of the AMIS algorithm as it appears in [1]. The algorithm starts with an initialization phase whose rationale is to provide a good scaling for the initial importance distribution, Q_0.

Step 0 - Initialization : (N_0 particles, $t = 0$)

- generate a single set of N_0 iid uniforms in $\mathcal{U}^{\otimes p}_{[0,1]}$;
- with scaled logistic transformation go back to $\Re^p$;
- compute IS weights as function of the scale, τ, of the logistic distribution;
- use ESS(w) to find a "good" proposal initial scaling (as an alternative, use ESS + tempering);

This initial step is computationally very cheap and automatic.

Step 1 - Adaptation : (N_1 particles, $t = 1 \cdots T_1$)
The rationale of this step is global adaptation of the parameters of the importance distribution. This can be achieved either by plain adaptation (Step 1) or by combining adaptation with a variance reduction technique introduced to prevent IS weights impoverishment by [3] (Step 1*).

A potential problem when implementing Step 1 is high variability of the importance weights. This can be solved implementing, instead, Step 1*.

Step 1* - Adaptation plus Variance reduction. The rationale of this step is twofold: Ensure that all particles are on the same "weighting scale" and can thus be easily and efficiently combined to get final estimator; Ensure variance reduction thanks to weights stabilization.

Step 2 - Clustering : (N_2 particles, $t = T_1 + 1 \cdots T_2$)
At iteration $t = T_1 + 1, \cdots, T_2$, for $1 \leq i \leq N_2$:

1. Run **Rao–Blackwellised clustering algorithm** based on (possibly trimmed by resample) past particles using a mixture of Gaussian distributions to cluster. The number of components, G, is chosen by BIC.
2. Estimate the mean, $\hat{\mu}_g$, and the covariance $\hat{\Sigma}_g$, on each cluster g.
3. Simulate new sample $x_1^{t+1}, \ldots, x_n^{t+1}$ from p-variate mixture (G components) of $\mathcal{T}_{(3)}(\hat{\mu}_g, \hat{\Sigma}_g)$.
4. For $i = 1, \ldots, N$ compute weights ω_i^{T+1} of particle x_i^{T+1} using **Deterministic Mixture** idea.
5. For $1 \leq l \leq T$, actualize past importance weights.

The final AMIS estimator recycles all $[N_0] + [N_1 \times T_1] + [N_2 \times (T_2 - T_1)]$ generated particles with the corresponding importance weights. This is done to avoid waste of particles and to combine **global** (step 1) and **local** adaptation (step 2).

The resulting AMIS algorithm is user friendly, provides an unbiased estimator of integrals with respect to the target and is highly efficient requiring no tuning, allowing for automatic local and global adaptation of the importance distribution. The algorithm is multi-purpose, interruptible and requires no burn-in (unlike AMCMC) and, as simulation proceeds, updates both the weights and the parameters of the mixture importance distribution.

3 *ARAMIS* Basics: Inputs, Outputs and Help of ARAMIS

The structure of AMIS naturally leads to require the following inputs (to be specified by the user), for ARAMIS:

Inputs (to be specified by the user):

- the size of the population both in the initialization, global and local adaptation phase
- the length of the simulation in the global and local adaptation phase
- the target distribution
- the kernel used in the importance distribution (whose parameters are adapted as the simulation proceeds): the default kernel is a non-central T-distribution with 3 degrees of freedom (as suggested in [1]).

Outputs Upon running ARAMIS the following output is obtained:

- generated samples and corresponding importance weights
- effective sample size (a measure of the algorithm performance introduced in Section 4.1 of [2]))
- mean and variance covariance matrix of the target distribution estimated by importance sampling

Help an on-line help is available with three examples worked out in details to guide the user: two are taken form the original paper [1]) (a banana-shaped and a mixture distribution target), while the last one is an original example where the target is a multivariate Extreme Value (EV) distribution obtained by a Copula construction.

4 First Steps with *ARAMIS*

Let's start by loading the package

```
> library(ARAMIS)
```

As a first example let's run an Adaptive Importance Sampling (i.e. the scheme in which the weights don't get actualized), on a "Banana"-shaped target distribution which is defined as follows:

- start with a p-dimensional Gaussian

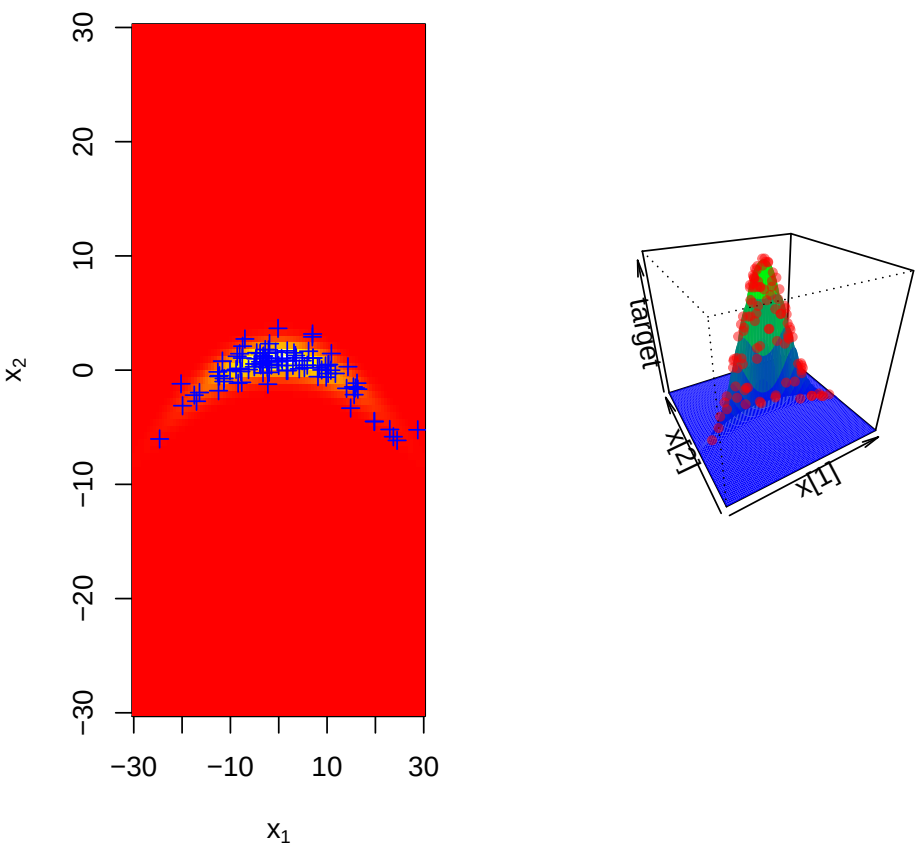
$$(x_1, ..., x_p) \sim N_p(\mathbf{0},\ diag(100, 1, ..., 1))$$

- change the second coordinate to

$$y_2 = x_2 + bx_1^2 - 100b$$

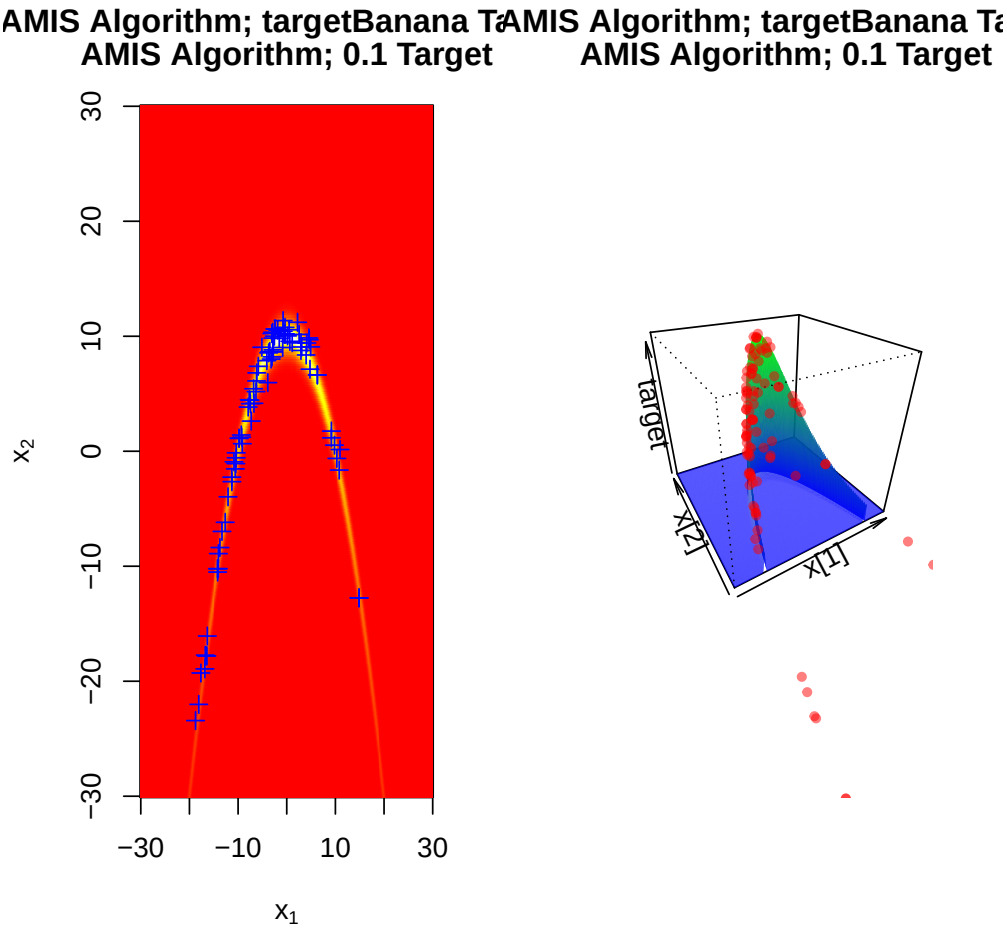
for some "Banana"-ness coeficient b which is set by default to $b = 0.03$.

```
> ais <- AIS(N=1000,
+                 niter=10,
+                 p=2,
+                 target= targetBanana(),
+                 initialize=uniInit(),
+                 mixture= mclustMix())
```



Let's now try the AMIS algorithm on a even stronger shaped Banana target ($b = 0.1$, which implies a much harder target)

```
> amis <- AMIS(N=1000,
+                        niter=10,
+                        p=2,
+                        target=targetBanana(b=0.1),
+                        initialize= amisInit(maxit=5000),
+                         mixture=mclustMix())
```



The `ISO` object returned by the function contains the sample from the target distribution and some further useful information.

```
> showClass("ISO")

Class "ISO" [package "ARAMIS"]

Slots:

Name:          IS        Prop        Mean         Var        Perp         ESS
Class:      matrix     numeric      matrix       array     numeric     numeric

Name:         seed       envir        call        args
Class:     integer environment        call        list
```

```
> summary(amis)

Call:
AMIS(N = 1000, niter = 10, p = 2, target = targetBanana(b = 0.1),
    initialize = amisInit(maxit = 5000), mixture = mclustMix())

Global ESS: 763.7322
Maximum Perplexity: 0.8668435
```

`ISO` objects allow the use of familiar syntax

```
> amis$ESS

[1] 763.7322

> ais[["ESS"]]

[1] 9009.858
```

but have also other useful methods, like `plot`, `mean` and `var`: let's use them for a simple sanity check:

```
      x1       x2
125.2236 448.2367

          x1        x2
x1 125.38778  18.07847
x2  18.07847 448.82438
```

5 How to use *ARAMIS* for your Research

The purpose of this section is to illustrate the functionality of the package on a real world example and to briefly show how the algorithm can be personalized to respond to your needs.

The algorithm is written in a general form so that the user can specify his favorite combination of target distribution, component of the proposal distribution and clustering algorithm, the way to really exploit all the flexibility is to use the technique called function closure.

To illustrate the use of function closures let's try the algorithm on a hard-to-tackle-problem. Let's use as a target the following Galambos copula

```
> # Galambos Copula
> # a : location parameter GEV
> # b : scale parameter GEV
> # s : shape parameter GEV
> # theta : copula's parameter
>
> galambosCop <- function(a=0,b=1,s=0,theta=1.5,copula=galambosCopula){
+
+         require("copula")
+         require("evd")
+
+         Cop <- copula(theta)
+
+         function(xx){
+
+                 U <- pgev(xx,loc=a, scale=b,shape=s)
+                 # numerical stability can be a issue...
+                 U[U>1-sqrt(.Machine$double.eps)] <- 1-sqrt(.Machine$double.eps)
```

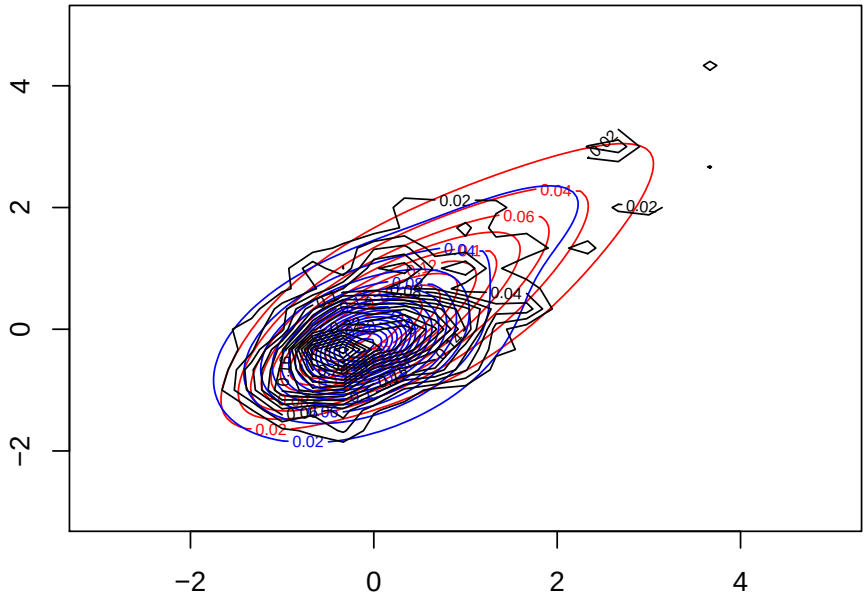
```
+                 U[U<sqrt(.Machine$double.eps)] <- sqrt(.Machine$double.eps)
+
+                 if(length(dim(xx)>1)){
+                         trgt <- log(dCopula(U,Cop))+
+                                    rowSums(apply(xx,2,function(x)
+                                              dgev(x,loc=a, scale=b,shape = s,log=TRUE)))
+                         }else{
+                                 trgt <- log(dCopula(U,Cop))+
+                                             sum(dgev(xx,loc=a, scale=b,shape=s,log=TRUE))
+                                 }
+
+                 trgt
+
+                 }
+         }
```

This is what the function closure consists into: to build an environment around the function that contains all the parameters the function needs but that are not being passed as an argument. This allows us to make the list of arguments completely general, in the example `a`, `b`, `s`, etc... are enclosed in scope of the function

```
> toughCop <- galambosCop()
> ls(envir=environment(toughCop))

[1] "Cop"    "a"      "b"      "copula" "s"      "theta"

> get("theta",envir=environment(toughCop))

[1] 1.5
```

We can now run `AMIS` and see from the plot how this is fully captured by the algorithm.

```
> resCop <- AMIS(N=c(500,1000,500),
+                             niter=c(10,50),
+                             p=2,
+                            target=toughCop,
+                             initialize= amisInit(maxit=5000),
+                             mixture=mclustMix())
> plot(resCop,whichOne=3L, xlim=c(-3,5),ylim=c(-3,5),N=17000)
>
```



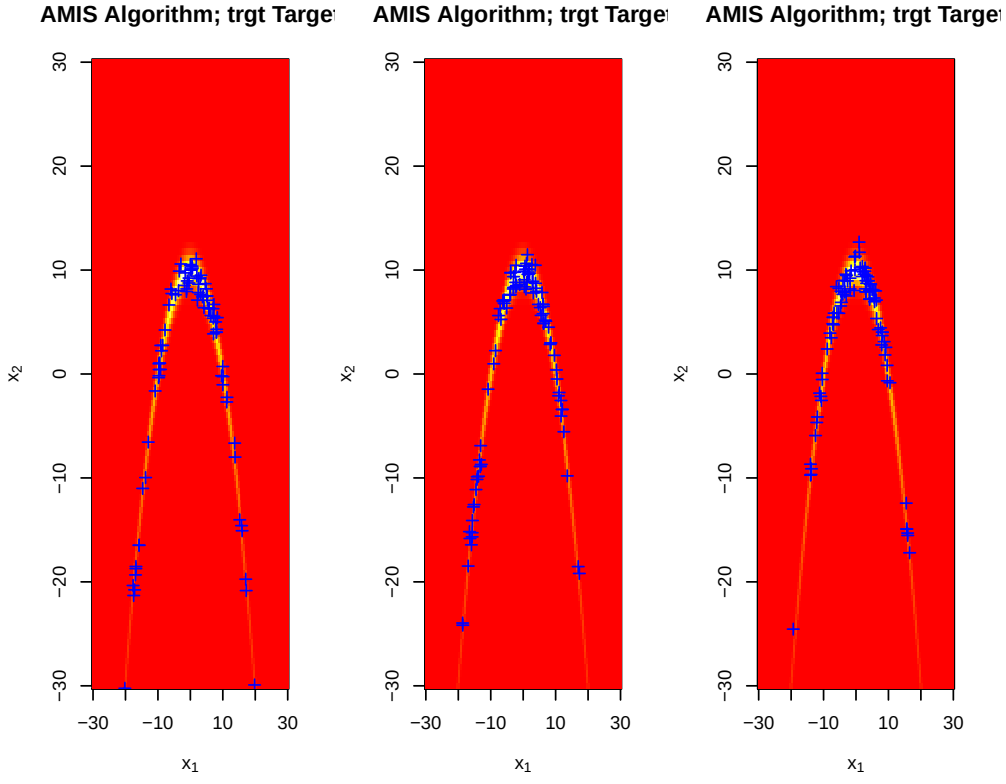
6 Parallelization

The evaluation of a particle is at each iteration completely independent from the others. `ARAMIS` exploits this feature to speed up computations considerably by equally sharing the burden of simulating the particles and computing the weights between the cores, then collecting the result and computing the summary statistics of the Importance Sample, in a Map-Reduce framework.

In practice the load gets evenly divided between the available cores and then the results merged for computing the mean and variance, for each iteration.

```
> trgt <- targetBanana(b=0.1)
> mxtr <- mclustMix()
> system.time(amis <- AMIS(N=1000,niter=10,p=2,target=trgt,mixture=mxtr))

   user  system elapsed
 42.212   0.671  44.611

> invisible(gc())
> system.time(amisMC <- AMIS(N=1000,niter=10,p=2,target=trgt,mixture=mxtr,parallel="multicore",nCores=2)

   user  system elapsed
 42.072   6.132  46.948

> invisible(gc())
> system.time(amisSN <- AMIS(N=1000,niter=10,p=2,target=trgt,mixture=mxtr,parallel="snow",nCores=2))

   user  system elapsed
 26.987   0.470  37.244
```

```
> summary(amis)

Call:
AMIS(N = 1000, niter = 10, p = 2, target = trgt, mixture = mxtr)

Global ESS: 823.0445
Maximum Perplexity: 0.8766368

> summary(amisMC)

Call:
AMIS(N = 1000, niter = 10, p = 2, target = trgt, mixture = mxtr,
    parallel = "multicore", nCores = 2)

Global ESS: 921.6402
Maximum Perplexity: 0.9197068

> summary(amisSN)

Call:
AMIS(N = 1000, niter = 10, p = 2, target = trgt, mixture = mxtr,
    parallel = "snow", nCores = 2)

Global ESS: 783.0711
Maximum Perplexity: 0.8846833

> par(mfrow=c(1,3))
> plot(amis,whichOne=1L)
> plot(amisMC,whichOne=1L)
> plot(amisSN,whichOne=1L)
>
```



Acknowledgments

We thank J.M. Marin for providing the code to run the AMIS algorithm on the examples of the paper [1] and M. Garieri for helping with many computational issues and frustrations.

References

[1] Cornuet, J.M. and Marin, J.M. and Mira, A. and Robert,C. *Adaptive Multiple Importance Sampling*. Scandinavian Journal of Statistics (2012).

[2] Liu, J. and Chen, R. *Blind Deconvolution via Sequential Imputations*. Journal of the American Statistical Association (1995).

[3] Owen, A. and Zhou, Y. *Safe and Effective Importance Sampling*. Journal of the American Statistical Association (2000).

[4] Gareth, O. and Roberts,C. and. Rosenthal, J.S.. *Examples of Adaptive MCMC*. Journal of Computational and Graphical Statistics (2009).

[5] Tierney L. and Mira A. *Some Adaptive Monte Carlo Methods for Bayesian Inference*. Statistics in Medicine (1999).

SessionInfo

- R Under development (unstable) (2013-01-16 r61667), `x86_64-apple-darwin9.8.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: ARAMIS 1.0.1, LearnBayes 2.12, MASS 7.3-23, copula 0.999-5, evd 2.3-0, mclust 4.0
- Loaded via a namespace (and not attached): ADGofTest 0.3, gsl 1.9-9, mvtnorm 0.9-9994, pspline 1.0-14, stabledist 0.6-5, stats4 3.0.0, tools 3.0.0