# Package: malt (via r-universe)

October 18, 2024

**Title** Metropolis Adjusted Langevin Trajectories

**Version** 0.9

**Description** Implements the sampling algorithm: Metropolis Adjusted Langevin Trajectories (Riou-Durand and Vogrinc 2022). Details available at: <arXiv:2202.13230>. The MALT algorithm is a robust extension of Hamiltonian Monte Carlo, for which robustness of tuning is enabled by a positive choice of damping parameter (a.k.a friction).

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.2

**URL** https://github.com/lrioudurand/malt

**BugReports** https://github.com/lrioudurand/malt/issues

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**Imports** stats, coda

**Repository** https://lrioudurand.r-universe.dev

**RemoteUrl** https://github.com/lrioudurand/malt

**RemoteRef** HEAD

**RemoteSha** cb291ee5c4933d8cd4c87cc4268cfc25733d6e70

## Contents

## ess_summary *ess_summary*

### Description

computes effective sample sizes of the chain

### Usage

```
ess_summary(output)
```

### Arguments

| | |
|---|---|
| output | an output of the malt function |

### Value

a list of effective sample sizes when estimating the first and second moments

### Examples

```
d=50
sigma=((d:1)/d)^(1/2)
init=rnorm(d)*sigma
U=function(x){sum(0.5*x^2/sigma^2)}
grad=function(x){x/sigma^2}
n=10^4
g=1.5
h=0.20
L=8
output=malt(init,U,grad,n,g,h,L)
ess_summary(output)
```

## malt *malt*

### Description

Implements the sampling algorithm: Metropolis Adjusted Langevin Trajectories (MALT) as described in Riou-Durand and Vogrinc (2022).

### Usage

```
malt(init, U, grad, n, g, h, L, warm = FALSE)
```

## Arguments

| | |
|---|---|
| `init` | Real vector. Initial values for the sampling algorithm. |
| `U` | A potential function to return the log-density of the distribution to be sampled from, up to an additive constant. It should input a real vector of the same length as `init` and output a scalar. |
| `grad` | A function to return the gradient of the potential. It should input and output a real vector of the same length as `init`. |
| `n` | The number of samples to be generated. Positive integer. |
| `g` | The friction, a.k.a damping parameter. Non-negative real number. The choice g=0 boils down to Hamiltonian Monte Carlo. |
| `h` | The time step. Positive real number. |
| `L` | The number of steps per trajectory. Positive integer. The choice L=1 boils down to the Metropolis Adjusted Langevin Algorithm. |
| `warm` | Should the chain be warmed up? Logical. If TRUE, the samples are generated after a warm-up phase of n successive trajectories. The first half of the warm-up phase is composed by unadjusted trajectories. |

## Details

Generates approximate samples from a distribution with density

$$\Pi(x) \propto e^{-U(x)}$$

A Markov chain is generated by drawing successive Langevin trajectories. Each trajectory starts from a fresh Gaussian velocity and is faced with an accept-reject test known as Metropolis adjustment. The Hamiltonian Monte Carlo (HMC) algorithm is recovered as a particular case when the damping parameter is set to zero. A positive choice of damping can ensure robustness of tuning, see Riou-Durand and Vogrinc (2022) for further details.

## Value

Returns a list with the following objects:

| | |
|---|---|
| `samples` | a matrix whose rows are the samples generated. |
| `draw` | a vector corresponding to the last draw of the chain. |
| `accept` | the acceptance rate of the chain. An acceptance rate close to zero/one indicates that the time step chosen is respectively too large/small. Optimally, the time step should be tuned to obtain an acceptance rate slightly above 65%. |
| `param` | the input parameters of the malt algorithm. |

## References

Riou-Durand and Vogrinc (2022). Available at: https://arxiv.org/abs/2202.13230.

## See Also

`trajectory`, which draws a Langevin trajectory and computes the numerical error along the path.

## Examples

```
d=50
sigma=((d:1)/d)^(1/2)
init=rnorm(d)*sigma
U=function(x){sum(0.5*x^2/sigma^2)}
grad=function(x){x/sigma^2}
n=10^4
g=1.5
h=0.20
L=8
malt(init,U,grad,n,g,h,L)
```

---

`print.malt` *print.malt*

---

## Description

print.malt

## Usage

```
## S3 method for class 'malt'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

## Arguments

| | |
|---|---|
| `x` | an output of the malt function |
| `digits` | the number of significative digits |
| `...` | other parameters |

---

`trajectory` *trajectory*

---

## Description

Draws a Langevin trajectory starting from a Gaussian velocity and computes its numerical error. with target density

$$\Pi(x) \propto e^{-U(x)}$$

Given a potential function $U$ and its gradient evaluation, draws a trajectory and computes its numerical error. The trajectory drawn corresponds to the proposal in the sampling algorithm: Metropolis Adjusted Langevin Trajectories (Riou-Durand and Vogrinc 2022). Details available at: https://arxiv.org/abs/2202.13230.

## Usage

```
trajectory(init, U, grad, g, h, L)
```

## Arguments

| | |
|---|---|
| `init` | Real vector. Initial values for the Langevin trajectory. |
| `U` | A potential function to return the log-density of the distribution to be sampled from, up to an additive constant. It should input a real vector of the same length as `init` and output a scalar. |
| `grad` | A function to return the gradient of the potential. It should input and output a real vector of the same length as `init`. |
| `g` | Non-negative real number. The friction, a.k.a damping parameter. The choice g=0 boils down to Hamiltonian Monte Carlo. |
| `h` | Positive real number. The time step. |
| `L` | Positive integer. The number of steps per trajectory. The choice L=1 boils down to the Metropolis Adjusted Langevin Algorithm. |

## Value

Returns a list with the following objects:

| | |
|---|---|
| `path` | a matrix whose rows are the successive steps of the trajectory. |
| `draw` | a vector containing the output of the trajectory. |
| `num_error` | a vector containing the cumulative numerical errors along the path. The numerical error is measured by the energy difference incurred by the leapfrog integrator. |

## Examples

```
d=50
sigma=((d:1)/d)^(1/2)
init=rnorm(d)*sigma
U=function(x){sum(0.5*x^2/sigma^2)}
grad=function(x){x/sigma^2}
g=1.5
h=0.20
L=8
trajectory(init,U,grad,g,h,L)
```

# Index