The MCIntegrator Fortran Module

Francesco Calcavecchia

August 7, 2015

The `MCIntegrator` Fortran module contains simple tools for computing numerical integrals in `NDIM` dimensions with the Monte Carlo technique, exploiting the M(RT)^2 algorithm for sampling from the provided sampling function. Included, there is a module called `estimators`, which can be used to estimate the mean and standard deviation of a set of uncorrelated or correlated data.

More specifically the integration is performed by sampling coordinates `X` from a given probability density function `g(X)`, and summing up an *observable* `f(X)`:

$$\int d\mathtt{X}\, \mathtt{f(X)} = \sum_{\mathtt{X}_i \text{ sampled from } \mathtt{g(X)}} \mathtt{f(X}_i\mathtt{)} \tag{1}$$

Both modules do not have any dependence.

1 Declaration of the module

First of all one has to declare the `MCIntegrator` module, using the instruction

```
USE mcintegrator
```

which has to be inserted just before the `IMPLICIT NONE` command in any `PROGRAM`, `SUBROUTINE`, `FUNCTION`, or `MODULE`.

2 Integrator's declaration

In the following we will use the variable name `imc` for labeling a `MCIntegrator` object. An integrator `imc` must be declared as

```
TYPE(MCI) :: imc
```

3 Integrator's initialization

The first step is to initialize the integrator, specifying the number of dimensions `NDIM`

```
CALL imc%initialize(NDIM= )
```

`NDIM` is of type `INTEGER(KIND=4)`.

The second mandatory step is to specify the observable:

```
CALL imc%setObservable(OBSERVABLE= )
```

The argument requested is a function (with an explicit interface) that should respect the following interface:

```
FUNCTION observable(x)
   IMPLICIT NONE
   REAL(KIND=8) :: observable
   REAL(KIND=8), INTENT(IN) :: x(:)
END FUNCTION observable
```

4 Optional settings

The following settings are said to be optional because there is a default value, but they actually could be essential for your calculation.

4.1 Integral domain

```
CALL imc%setIRange(IRANGE= )
```

The argument `IRANGE` is an array of type `REAL(KIND=8)` with shape `(1:2,1:NDIM)`. For example, in the one-dimensional case, to integrate between `La` and `Lb`, one has to set `IRANGE(1,1)=La` and `IRANGE(2,1)=Lb`. It is assumed that `La` $<$ `Lb`.

When the integral domain is set, the initial coordinates and the $\mathrm{M(RT)}^2$ step are set accordingly. Specifically, the coordinates are set to be in the middle of the integration volume, whereas the step is set to be half of the integration sides for each direction.

4.2 Initial coordinates

As the reader should now, the Markov chain is built starting from an initial point. By default this is assumed to be in the middle of the integration space, however it might be convenient to set it manually in certain specific situations. This can be done with the command

```
CALL imc%setX(X0= )
```

where `X0` is an array of type `REAL(KIND=8)` with shape `(1:NDIM)`.

4.3 M(RT)^2 step

The initial M(RT)^2 step is set equal to 0.1 by default for every direction.

If the user does not provide a sampling function, it is important that he/she provides a reasonable step.

However, if a sampling function has been provided, it is not essential to provide this parameter, because before proceeding with the integration, `MCIntegrator` adjust the step in order to obtain an acceptance close to the target one (by default 50%, see subsection 6.5). In any case, setting by hand a reasonable value can result in a tuning speed-up.

Use

```
CALL imc%setMRT2Step(STEP= )
```

where `STEP` is of type `REAL(KIND=8)` and must have shape `(1:NDIM)`.

4.4 Acceptance rate

In the context of the M(RT)^2 algorithm, the acceptance rate is one of the most important parameters to control. Before proceeding with the integration, `MCIntegrator` automatically adjust the M(RT)^2 step in order to obtain an acceptance rate close to the provided target one (by default 50%). A target acceptance rate of 50% provides very good performance in almost all cases. We remark that if the user does not specify a sampling function, the acceptance rate will always be 100%, independently of the M(RT)^2 step.

For setting the target acceptance rate use

```
CALL imc%setTargetAcceptanceRate(TARGETACCRATE= )
```

where `TARGETACCRATE` is a `REAL(KIND=8)`.

4.5 Sampling function

Whenever it is possible, it is convenient to sample from a probability density function as similar as possible to the observable function. We remark that one of the properties of a probability density function is the normalization. the user is responsible for providing a correct sampling function.

To set a sampling function, use

```
CALL imc%setSamplingFunction(SAMPLING_FUNCTION= )
```

where `SAMPLING_FUNCTION` must fulfill the same interface as the observable (see Section 3).

5 Integration

Once that all the settings are done, one can obtain the result by invoking

```
CALL imc%integrate(NMC= , AVERAGE= , ERROR= )
```

where `NMC` is a `INTEGER(KIND=8)` and must be provided as input, whereas `AVERAGE` and `ERROR` are of type `REAL(KIND=8)` and are provided as output. `NMC` is the number of sampled points: The larger its value, the more accurate will be the result, according to the well known $1/\sqrt{\texttt{NMC}}$ rule. `AVERAGE` will contain the resulting numeric estimation of the integral with an estimated standard deviation equal to `ERROR`.

6 Getting additional information

It is possible to obtain some additional information about the integration.

6.1 Dimensionality

It is possible to get the number of dimensions with

```
imc%getNDim()
```

6.2 Integral domain

The `IRANGE` described in subsection 4.1 can be obtained back by calling

```
imc%getIRange()
```

6.3 Coordinates

The actual coordinates can be obtained by

```
imc%getX()
```

Before the integration this call will give back the initial coordinates, while after an integration this function will provide their last value.

6.4 M(RT)^2 step

One can in any moment access to the value of the M(RT)^2 step using

```
imc%getMRT2Step()
```

If a sampling function is provided, after an integration, its value will be changed automatically in order to obtain the an acceptance rate as close as possible to the target.

6.5 Acceptance rate

The target acceptance rate can be checked by calling

```
imc%getTargetAcceptanceRate()
```

The actual acceptance rate resulting from an integration is obtainable with

```
imc%getAcceptanceRate()
```

7 Free the memory

After computing the integral, one can free the allocated memory embedded with the `MCIntegrator` by using the instruction

```
CALL imc%terminate()
```

8 Estimation of the average and its standard deviation

Given an array of `REAL(KIND=8)`, one can use the module `estimators` to extract its average value and standard deviation. To do so one has to first declare the module

```
USE estimators
```

and then use one of the three available functions:

- `uncorrelated_estimator(N= , X= )`
  where `N` is an `INTEGER(KIND=8)`, and `X` an array of `REAL(KIND=8)` of size `N`. The function will return an array of `REAL(KIND=8)` with size 2, where the first element will be the average value, and the second one will be the standard deviation. This function can be used when data are not correlated;

- `uncorrelated_estimator(N= , X= )`
  input and output are structures as in the previous function, but in this case data are considered to be correlated, and the blocking technique is used to correctly estimate the standard deviation;

- `block_estimator(N= , X= , NBLOCKS= )`
  where `NBLOCKS` is an `INTEGER(KIND=4)`. This function computes average and standard deviation dividing the data into `NBLOCKS` blocks and computing their average, and then use this new data to compute their standard deviation as if they were uncorrelated.