Introduction to ADerrors

Alessandro Conigli

IFT UAM-CSIC

December 13th -17th Tor Vergata

Analysis of Monte Carlo data

- The main challenge in MC data analysis is to assess the statistical and systematic errors of derived observables
- The strong autocorrelations typical of MC data make error estimation difficult

Resampling methods

- Bootstrap and jack-knife
- Blocks of data are averaged in bins
- Correlations is reduced
- Power-like decrease of remaining autocorrelation

Γ-method

- Estimate slow decaying modes of the MC chain in the error estimates
- Correlations is reduced
- Exponentially-fast decrease of remaining autocorrelation
- Propagate errors using AD

The Γ- method

- In general, some primary observables A_i^α are measured in several MC simulations

$$\alpha \rightarrow ensemble \qquad i = 1,\ldots,N_{obs}^\alpha \rightarrow observables\ measured\ in\ \alpha$$

- In real applications only a finite set of MC measurements for each primary observable

$$a_i^\alpha(t) \qquad t = 1,\ldots,N_\alpha \qquad t \rightarrow MC\ time$$

- As estimate for A_i^α we use the MC average

$$\bar{a}_i^\alpha = \frac{1}{N_\alpha}\sum_{t=1}^{N_\alpha} a_i^\alpha(t) \qquad \delta_i^\alpha(t) = a_i^\alpha(t) - \bar{a}_i^\alpha$$

- Typically what we want is a function F of the primary observables. We can estimate F by:

$$F \equiv f(A_i^\alpha) \rightarrow \bar{F} = f(\bar{a}_i^\alpha)$$

The Γ- method

- In order to compute the error of $\bar{F} = f(\bar{a}_i^\alpha)$ we use linear propagation

$$f(A_i^\alpha + \epsilon_i^\alpha) = f(A_i^\alpha) + \epsilon_i^\alpha \frac{\partial f}{\partial A_i^\alpha}\bigg|_{\bar{a}_i^\alpha} + O(\epsilon_i^2)$$

- Moreover, we need the autocorrelation function of the primary observables from the MC chain

$$\Gamma_{ij}^{\alpha\beta}(t) = \frac{\delta_{\alpha\beta}}{N_\alpha - t} \sum_{t'=1}^{N_\alpha - t} \delta_i^\alpha(t+t')\delta_j^\alpha(t') \qquad \delta_i^\alpha(t) = a_i^\alpha(t) - \bar{a}_i^\alpha$$

- Finally the error estimates for F is given in terms of the autocorrelation functions

$$\rho_F^\alpha(t) = \frac{\Gamma_F^\alpha(t)}{\Gamma_F^\alpha(0)}, \qquad \Gamma_F^\alpha(t) = \sum_{ij} \frac{\partial f}{\partial A_i^\alpha}\bigg|_{\bar{a}_i^\alpha} \frac{\partial f}{\partial A_j^\alpha}\bigg|_{\bar{a}_j^\alpha} \Gamma_{ij}^{\alpha\alpha}(t)$$

The Γ- method

- The autocorrelation functions are then used to define the per-ensemble variances

$$(\sigma_F^\alpha)^2 = \Gamma_F^\alpha(0), \qquad \tau_{int}^\alpha(F) = \frac{1}{2} + \sum_{t=1}^{\infty} \frac{\Gamma_F^\alpha(t)}{\Gamma_F^\alpha(0)}$$

Integrated autocorrelation time

- Since different ensembles are statistically uncorrelated, we can combine them in quadrature

$$(\delta\bar{F})^2 = \sum_\alpha \frac{(\sigma_F^\alpha)^2}{N_\alpha} 2\tau_{int}^\alpha(F)$$

Final error estimate

- Since each ensemble is treated independently, we can estimate the sources of error as

$$R_\alpha(F) = \frac{(\sigma_F^\alpha)^2 2\tau_{int}^\alpha(F)}{N_\alpha(\delta\bar{F})^2}$$

The Γ- method

- A crucial step is to perform the truncation of the infinite sum in $\tau^{\alpha}_{int}(F)$

$$\tau^{\alpha}_{int}(F) = \frac{1}{2} + \sum_{t=1}^{W^{\alpha}_{F}} \frac{\Gamma^{\alpha}_{F}(t)}{\Gamma^{\alpha}_{F}(0)}$$

- Ideally W^{α}_{F} has to be large compared to the exponential autocorrelation time τ^{α}_{exp}

Truncation error: $O(e^{-W^{\alpha}/\tau^{\alpha}_{exp}})$

- In practice W^{α}_{F} is chosen to minimise the sum of the systematic error of the truncation and the statistical error

- The procedure requires an estimate of τ^{α}_{exp}. Here it is assumed that

$$\tau^{\alpha}_{exp} \approx S_{\tau}\tau^{\alpha}_{int}$$

where S_{τ} is a parameter tuned by inspecting the data.

The Γ- method

- In many practical cases τ^{α}_{exp} and $\tau^{\alpha}_{int}(F)$ are very different. In these cases the summation window W^{α}_{F} cannot be taken much larger than τ^{α}_{exp} , or the errors are underestimated.

- A solution would be to improve the estimate for $\tau^{\alpha}_{int}(F)$:

  ★ First the autocorrelation function is summed up to W^{α}_{F}

  ★ For $t > W^{\alpha}_{F}$ the autocorrelation function is assumed to be the slowest mode

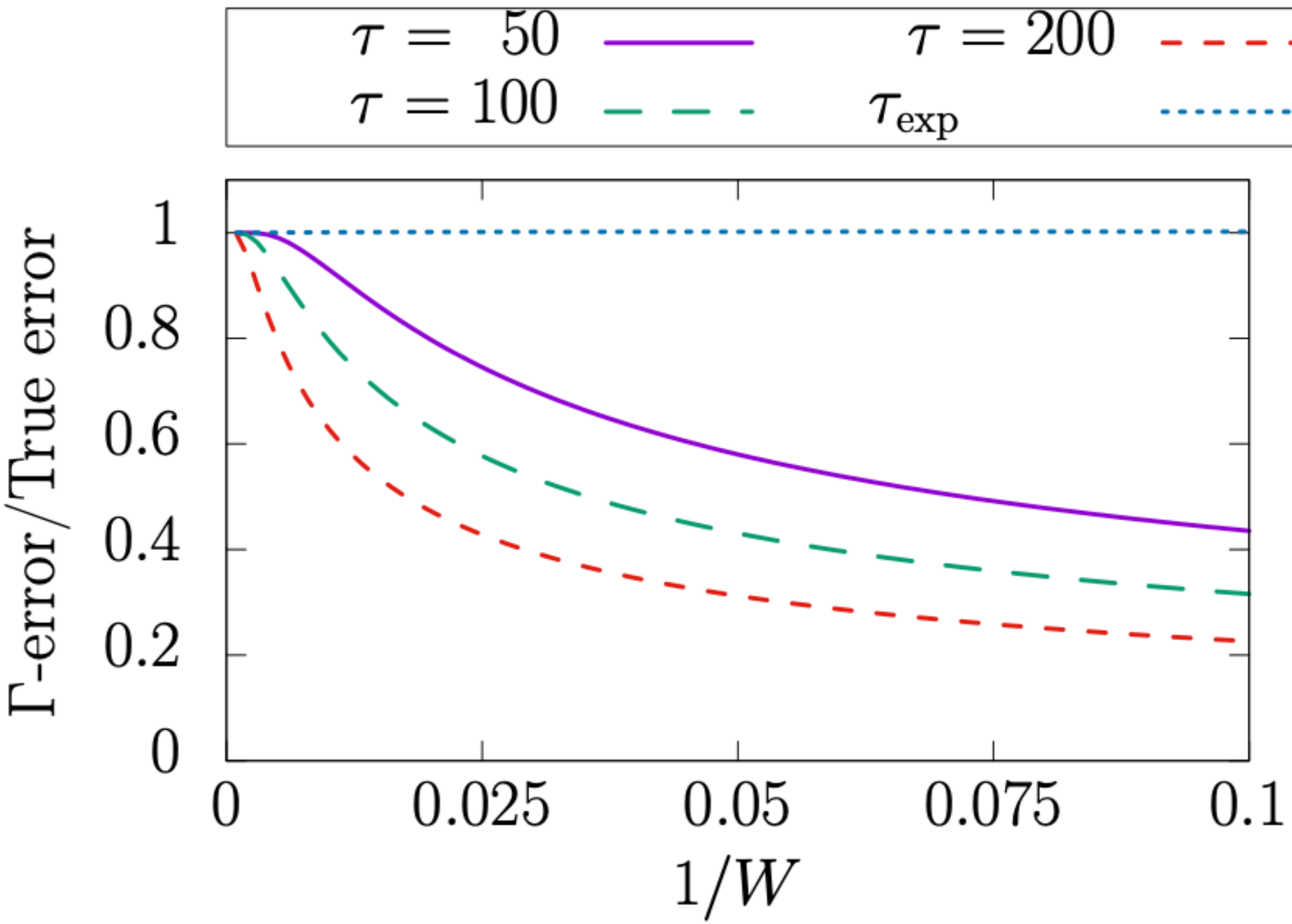
$$\rho^{\alpha}_{F}(t) \sim \exp(-|t|/\tau_{exp})$$

  and it is explicitly added to the computation of $\tau^{\alpha}_{int}(F)$

$$\tau^{\alpha}_{int}(F) = \frac{1}{2} + \sum_{t=1}^{W^{\alpha}_{F}} \frac{\Gamma^{\alpha}_{F}(t)}{\Gamma^{\alpha}_{F}(0)} + \tau^{\alpha}_{exp}\rho^{\alpha}_{F}(W^{\alpha}_{F}+1)$$

Integrated autocorrelation time

The Γ- method



[A. Ramos, arXiv:1809.01289]

Automatic Differentiation

- Computation of derivatives with arbitrary functions, free from systematic and round-off errors
- Any function are just a series of fundamental operations and the evaluation of few intrinsic hard-coded functions
- Imagine a simple composition of functions

$$y = f_3(f_2(f_1(x)))$$

- The chain rule gives for the derivatives

$$\frac{dy}{dx} = \frac{dy}{dz_2}\frac{dz_2}{dz_1}\frac{dz_1}{dx} \qquad where \qquad z_1 = f_1(x), \quad z_2 = f_2(z_1), \quad z_3 = f_3(z_2)$$

- Two modes to perform AD
  - forward accumulation
  - backward accumulation

AD applied to MC data: errors in fit parameters

- In non linear least squared we are interested in minimising the function

$$\chi^2(p_i, d_a), \qquad p_i \ (i = 1,\ldots,N_{par}), \quad d_a \ (a = 1,\ldots,N_{data})$$

where typically the explicit form of the χ^2 is

$$\chi^2(p_i, d_a) = \sum_{a=1}^{N_{data}} \left(\frac{f(x_a, p_i) - d_a}{\sigma(y_a)} \right)^2$$

- The result of the fit is some parameters $\bar{p}_i$ that make $\chi^2(\bar{p}_i, \bar{d}_a)$ minimum for some fixed values of the data $\bar{d}_a$

- To estimate the errors we shift the data $d_a \to \bar{d}_a + \delta d_a$ and we ask how much the parameters will change. At leading order we have:

$$\chi^2(p_i, \bar{d}_a + \delta d_a) = \chi^2(p_i, \bar{d}_a) + \partial_a \chi^2 \Big|_{(p_i, \bar{d}_a)} \delta d_a, \qquad \partial_a \equiv \partial / \partial d_a$$

AD applied to MC data: errors in fit parameters

- After the shift we have a new minimum in $p_i \to \bar{p}_i + \delta p_i$, and by minimising and expanding with respect to $\bar{p}_i$ we get:

$$\partial_j \partial_i \chi^2 \Big|_{(\bar{p}_i, \bar{d}_a)} \delta p_j + \partial_i \partial_a \chi^2 \Big|_{(\bar{p}_i, \bar{d}_a)} \delta d_a = 0, \qquad \partial_i \equiv \partial / \partial p_i$$

- Defining the Hessian matrix at the minimum $H_{ij} = \partial_j \partial_i \chi^2 \Big|_{(\bar{p}_i, \bar{d}_a)}$, we get the derivatives of the fit parameters with respect to the data:

$$\frac{\delta p_i}{\delta d_a} = - \sum_{j=1}^{N_{param}} (H^{-1})_{ij} \partial_j \partial_a \chi^2 \Big|_{(\bar{p}_i, \bar{d}_a)}$$

- The iterative procedure is performed just once!
  Error propagation is performed later by evaluating the derivatives of the χ^2 function.

ADerrors.jl

Installation

- The software is not in the Julia general registry, still we can install it directly from the GItHub using the Julia package manager `Pkg`

- `ADerrors.jl` also depends on `BDIO.jl` and it should be installed beforehand

Code (Julia)

```
julia> import Pkg
julia> Pkg.add("https://gitlab.ift.uam-csic.es/alberto/bdio.jl")
julia> Pkg.add("add https://gitlab.ift.uam-csic.es/alberto/aderrors.jl")
```

ADerrors.uwreal

- At the core of the `ADerrors.jl` package there is the `uwreal` data type

```
uwreal(x::Float64)
uwreal([value::Float64, error::Float64], mcid)
uwreal(data::Vector{Float64}, mcid[, replica::Vector{Int64}])
```

- Input is a `Float64`: the variable is treated as a real number with zero error
- Input is a 2-element `Vector{Float64}`: the variable is understood as value $\pm$ error
- Input is a `Vector{Float64}` with length > 4: the data is understood as a consecutive measurements of an observable in a MC simulation

★ In the last two cases and ensemble ID is required as input. Data with the same ID are considered correlated

ADerrors.uwreal

- Data can contain measurements in several replica, independent simulations with the same physical and algorithmic parameters.

```
using ADerrors # hide
# 1000 measurements in three replica of lengths
# 500, 100 and 400
a = uwreal(rand(1000), "Ensemble with three replica", [500, 100, 400])
```

- Gaps in the measurements: what if an observable is not measured in every configuration?

```
using ADerrors # hide
# Observable measured on the odd configurations
# 1, 3, 5, ..., 999 on an emsemble of length 1000
a = uwreal(rand(500), "Observable with gaps", collect(1:2:999), 1000)

# Observable measured on the first 900 configurations
# on the same emsemble
b = uwreal(rand(900), "Observable with gaps", collect(1:900), 1000)
```

ADerrors.uwerr

- The function uwerr performs error analysis on a given observable

Code (Julia)

```
julia> x = uwreal([12.31,0.23], "Experiment A") # x 12.31(23) from experiment A
12.31 (Error not available... maybe run uwerr)

julia> y = uwreal([4.22,0.12], "Experiment B") # y=4.22(12) from experiment B
4.22 (Error not available... maybe run uwerr)

julia> z = x + y
16.53 (Error not available... maybe run uwerr)
julia> uwerr(z) # Determine total error in z
julia> println(z) # sqrt(0.23^2+0.12^2) = 0.25... (exp. A and B uncorrelated)
16.53 +/- 0.25942243542145693

julia> details(z)
16.53 +/- 0.25942243542145693
 ## Number of error sources: 2
 ## Number of MC ids       : 0
 ## Contribution to error  :                Ensemble  [%]     [MC length]
  #                                 Experiment A  78.60            -
  #                                 Experiment B  21.40            -
```

ADerrors.uwerr : optimal window

- Error in data coming from MC ensemble is determined by summing the Γ for each ID

```
uwerr(a::uwreal[, wpm::Dict{Int64,Vector{Float64}}])
uwerr(a::uwreal[, wpm::Dict{String,Vector{Float64}}])
```

- By default the summation window is determined with a parameter $S_\tau = 4$ [U. Wolff proposal]

★ `wpm[1]`: the autocorrelation function is summed up to `t=round(wpm[1])`

★ `wpm[2]`: the summation window is determined using U. Wolff proposal $S_\tau =$ `wpm[2]`

★ `wpm[3]`: the $\Gamma(t)$ is summed up to a point where $\delta\Gamma(t) \geq$ `wpm[3]*signal`

★ `wpm[4]`: tells `ADerrors` to add a tail to the error with $\tau_{exp} =$ `wpm[4]`

- Keep in mind!
  Negative values of `wpm[1:4]` are ignored and only one value of `wpm[1:3]` has to be positive

Error propagation in iterative algorithms: preliminaries

- ADerrors can deal with both uncorrelated and correlated fit

$$\chi^2(p_i, d_a), \qquad p_i \ (i = 1,\ldots,N_{par}), \quad d_a \ (a = 1,\ldots,N_{data})$$

Uncorrelated fits

$$\chi^2(p,d) = \sum_a [d_a - f_a(p)] W_a [d_a - f_a(p)]$$

$$\chi^2_{exp}(p,d) = [C_{ab}(\delta_{ab} W_a - P_{ab})]$$

Correlated fits

$$\chi^2(p,d) = \sum_a [d_a - f_a(p)] W_{ab} [d_b - f_b(p)]$$

$$\chi^2_{exp}(p,d) = [C_{ab}(W_{ab} - P_{ab})]$$

where $C_{ab} = cov(d_a, d_b)$ and $P_{ab} = (\partial_i \partial_a \chi^2)(H^{-1})_{ij}(\partial_j \partial_b \chi^2)$

- The χ^2_{exp} is thought to measure the true number of d.o.f. in a fit
- As a general rule, $\chi^2/\chi^2_{exp} \approx 1$ is a good measure of the quality of the fit in both cases

ADerrors.fit_error

- ADerrors.jl is agnostic about how you minimise the χ^2 function
- Once the central values of the fit parameters have been estimated (using LsqFit.jl, LeastSquaresOptim.jl) then we can propagate the errors with fit_error

```
fit_error(chisq::Function, xp::Vector{Float64}, data::Vector{uwreal}[, wpm];
                W = Vector{Float64}(), chi_exp = true)
```

★ chisq: assumed to have the previously discussed form, i.e. quadratic in the data

★ xp: values of the fit parameters at the minimum of the χ^2

★ data: vector of uwreal whose fluctuations enter in the evaluation of chisq

★ wpm: criteria to select the summation window in uwerr

★ W: weights that enter in the evaluation of chisq. If a vector is passed the matrix is assumed diagonal (uncorrelated fits). If nothing is passed, W is assumed diagonal with entries given by the inverse of the errors squared of the data

THANK YOU!