Automatic differentiation for error analysis

Alberto Ramos <alberto.ramos@ific.uv.es>

References:

- U. Wolff, "Monte Carlo errors with less errors". Comput.Phys.Commun. 156 (2004) 143-153.
- F. Virotta, "Critical slowing down and error analysis of lattice QCD simulations." PhD thesis.
- Stefan Schaefer, Rainer Sommer, Francesco Virotta, "Critical slowing down and error analysis in lattice QCD simulations". Nucl.Phys.B 845 (2011) 93-119.
- A. Ramos, "Automatic differentiation for error analysis of Monte Carlo data". Comput.Phys.Commun. 238 (2019) 19-35.
- M. Bruno, R. Sommer, In preparation.

Software: https://igit.ific.uv.es/alramos/aderrors.jl

Data analysis in Lattice QCD

From simulations to the proton mass

- Discretize space-time in a lattice of spacing a
- Use Monte Carlo techniques to generate "configurations" (representatives of the QCD vacuum)
- "Measure" correlation functions in these configurations

$$aM_p, af_\pi, af_k, af_+^{B\to D}(q^2), aM_\pi, \dots$$

- Detailed analysis to extrapolate results to the physical world:

$$a \to 0, L \to \infty, m_{\rm q} \to m_{\rm q}^{\rm phys}, \dots$$

- Our result (i.e. $M_p = 943(17)\text{MeV}, V_{\rm cb} = 0.039(3), \dots$) is a (complicated) function of the "measured" quantities in **several** ensembles (values of $a, L, m_{\rm q}$)

$$M_p = F(aM_p, af_\pi, af_k, aM_\pi, \dots)$$

- This talk: techniques to determine uncertainty in derived observables

Data analysis in Lattice QCD

From simulations to the proton mass

- Discretize space-time in a lattice of spacing a
- Use Monte Carlo techniques to generate "configurations" (representatives of the QCD vacuum)
- "Measure" correlation functions in these configurations primary observables

$$aM_p, af_\pi, af_k, af_+^{B\to D}(q^2), aM_\pi, \dots$$

- Detailed analysis to extrapolate results to the physical world:

$$a \to 0, L \to \infty, m_{\rm q} \to m_{\rm q}^{\rm phys}, \dots$$

- Our result (i.e. $M_p = 943(17)\text{MeV}, V_{\rm cb} = 0.039(3), \dots$) is a (complicated) function of the "measured" quantities in **several** ensembles (values of $a, L, m_{\rm q}$)

$$M_p = F(aM_p, af_\pi, af_k, aM_\pi, \dots) \qquad \text{(derived observables)}$$

- This talk: techniques to determine uncertainty in derived observables

Linear error propagation

Seems trivial...

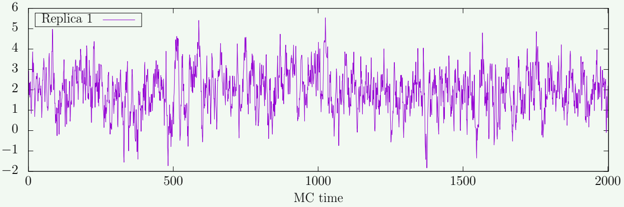
$$Z = F(X) \quad \Longrightarrow \quad \delta Z = \frac{\mathrm{d}F}{\mathrm{d}X}\delta X$$
$$X = 1.0 \pm 0.1 \quad \Longrightarrow \quad Z = \sin(X) = 0.841 \pm 0.054$$

But we need to be careful with

- Correlations between data
- What happens when F is an iterative algorithm

$$F = \text{Fit data } (x, y) \text{ to model } f$$

- Final error... Where does it come from?
- What to do when data is not $X = 1.0 \pm 0.1$, but



Welcome to ADerrors.jl

- Exact linear error propagation, even in iterative algorithms. Thanks to Automatic diferentiation (`ForwardDiff.jl`)
- Handles data from any number of ensembles (i.e. simulations with different parameters/data from different sources).

Code (Julia)

```
        julia> import Pkg
        (v1.X) pkg> add https://igit.ific.uv.es/alramos/bdio.jl
        (v1.X) pkg> add https://igit.ific.uv.es/alramos/aderrors.jl
```

A calculator with uncertainties

Code (Julia)

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

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

julia> z = x + y
16.53 (Error not available... maybe run uwerr)
julia> uwerr(z)

julia> println(z) # sqrt(0.23^2 + 0.12^2) = 0.25942243542145693
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          -

julia> zero_error = sin(z) - ( sin(x)*cos(y) + cos(x)*sin(y) )
-6.661338147750939e-16 (Error not available... maybe run uwerr)
julia> uwerr(zero_error)

julia> println(zero_error) # ADerrors keeps correlations!
-6.661338147750939e-16 +/- 1.7281005654554058e-16
```

ADerrors.jl ALLOWS TO INPUT CORRELATED DATA

Code (Julia)

```
julia> avg = [16.26, 0.12, -0.0038];
julia> Mcov = [0.478071 -0.176116 0.0135305
               -0.176116 0.0696489 -0.00554431
               0.0135305 -0.00554431 0.000454180];

julia> p = cobs(avg, Mcov, "Correlated data")
3-element Array{uwreal,1}:
 16.26 (Error not available... maybe run uwerr)
 0.12 (Error not available... maybe run uwerr)
 -0.0038 (Error not available... maybe run uwerr)

julia> uwerr.(p);
julia> p
3-element Array{uwreal,1}:
 16.26 +/- 0.6914267857119798
 0.12 +/- 0.2639107803785211
 -0.0038 +/- 0.021311499243366245
julia> cov(p)
3×3 Array{Float64,2}:
  0.478071   -0.176116     0.0135305
 -0.176116    0.0696489   -0.00554431
  0.0135305  -0.00554431   0.00045418

julia> z = p[1] + p[2] + sin(p[3]);  # Correlations are propagated
julia> uwerr(z)
julia> z
16.37620000914533 +/- 0.4603415450741195
```

Works with iterative algorithms: Root of $f(x) = a\cos(b\sin(x)) - x$

Code (Julia)

```
julia> a = uwreal([1.34, 0.12], "Data 01") # a = 1.34 +/- 0.12
julia> b = uwreal([1.34, 0.12], "Data 02") # b = 1.34 +/- 0.12

julia> x0 = uwreal([0.5,0.5], "Initial position") # x0 = 0.5 +/- 0.5
julia> while true # This is just newton method
           val = a*cos(b*sin(x0)) - x0
           der = -a*b*sin(b*sin(x0))*cos(x0) - 1.0
           x1  = x0 - val/der
           if (abs(value(x0) - value(x1))<1.0E-10)
               break
           else
               x0 = x1
           end
       end

julia> uwerr(x1)

julia> details(x1)
0.7838003744331717 +/- 0.056992589665847124
 ## Number of error sources: 3
 ## Number of MC ids       : 0
 ## Contribution to error  :                Ensemble  [%]     [MC length]
  #                                          Data 02   63.25           -
  #                                          Data 01   36.75           -
  #                                 Initial position    0.00           -
```

Awesome... but I do not want to re-code Levenberg-Marquardt

AD allows to propagate errors to fit parameters

- Find p_i ($i = 1, \dots, N_{\text{parm}}$) that minimize

$$\chi^2(p_i; d_a)\,, \qquad p_i\,(i = 1, \dots, N_{\text{parm}})\,, \quad d_a\,(a = 1, \dots, N_{\text{data}})\,.$$

  with d_a some MC data.
- χ^2 is minimum at $\bar{p}_i$ for the central values of the data $\bar{d}_a$
- How much changes the values of the parameters ($\bar{p}_i \to \bar{p}_i + \delta p_i$) that minimize χ^2 when the data is shifted $\bar{d}_a \to \bar{d}_a + \delta d_a$?

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

  with

$$H_{ij} = \partial_j \partial_i \chi^2 \Big|_{(\bar{p}_i; \bar{d}_a)}\,,$$

- Error propagation only requires derivatives of χ^2 at the minima (we do not care how you arrived there!)

Exact error propagation in fits

Code (Julia)

```
...
julia> lm, csq = FFfit_funcs(fp, f0, 4, 4, nlatt, FLAGp); # Definition of fit functions
julia> uwdt
9.5     1.05004 +/- 0.039482274503883384
11.6     1.17597 +/- 0.04580916938779833
8.5     1.00648 +/- 0.012263563919187603
...
julia> r = optimize(xx -> lm(xx, value.(uwdt)), zeros(9), LevenbergMarquardt())
 * Algorithm: LevenbergMarquardt
 * Minimizer: [0.03562259277485376,-0.2733271920349287,1.6897944291757172,-89.38349668726151,0.0
 * Sum of squares at Minimum: 4.901151

julia> fitp, cexp = fit_error(csq, r.minimizer, uwdt) # Error propagation with ADerrors
julia> uwerr.(fitp) # The fit parameters are normal uwreal variables
julia> print("Vcb: ") # The last fit parameter is Vcb
julia> details(fitp[end])
Vcb: 0.04032176066923957 +/- 0.0021953643904483967
 ## Number of error sources: 24
 ## Number of MC ids       : 0
 ## Contribution to error  :                Ensemble  [%]     [MC length]
  #                                 BaBar 00000001  31.84           -
  #                                 BaBar 00000002  23.33           -
  #                             BaBar 10b 00000001  16.30           -
  #                             BaBar 10b 00000002  11.80           -
  #                                 BaBar 00000003  10.13           -
  #                    Lattice form factors 00000003   2.54           -
...
```

Handling Monte Carlo data

- Generate a random walk in the interval $[-1, 1]$

Code (Julia)

```julia
julia> # Generate some correlated data
       eta  = randn(10000);
julia> x    = Vector{Float64}(undef, 10000);
julia> x[1] = 0.0;
julia> for i in 2:10000 # This is just a random walk in [-1,1]
           x[i] = x[i-1] + 0.2*eta[i]
           if abs(x[i]) > 1.0
               x[i] = x[i-1]
           end
       end
```

- Input a Monte Carlo history as uwreal. Correlations handled automatically

Code (Julia)

```julia
julia> xp2 = uwreal(x.^2, "Random walk ensemble in [-1,1]")
0.3533602504472119 (Error not available... maybe run uwerr)

julia> xp4 = uwreal(x.^4, "Random walk ensemble in [-1,1]")
0.2197572322981959 (Error not available... maybe run uwerr)

julia> cov([xp2, xp4])
2×2 Array{Float64,2}:
 8.12723e-5  6.57753e-5
 6.57753e-5  5.28194e-5
```

Handling Monte Carlo data (and mix with variables with error)

- You can operate with MC data as with any other `uwreal`

Code (Julia)

```
julia> Z = uwreal([1.23, 0.067], "Normalization factor")
1.23 (Error not available... maybe run uwerr)

julia> res = Z*xp2^2/xp4
0.6606310543026856 (Error not available... maybe run uwerr)

julia> uwerr(res)

julia> details(res)
0.6606310543026856 +/- 0.038157302195505186
 ## Number of error sources: 2
 ## Number of MC ids       : 1
 ## Contribution to error  :               Ensemble  [%]     [MC length]
  #                           Normalization factor  88.94           -
  #                   Random walk ensemble in [-1,1]  11.06   10000
```

Scales to large projects

- Example from a recent project (determination of α_s)

Code (Julia)

```
julia> details(lambda)
  Lambda: 392.0531757233569 +/- 10.640559075729831
   ## Number of error sources: 210
   ## Number of MC ids       : 163
   ## Contribution to error  :                 Ensemble  [%]    [MC length]
    #                                                -2  48.72          -
    #                                          85000480   4.21   4120,600
    #                                          90000480   4.11   4760,820,820,820
    #                                          95000480   3.86   4620,1540,980,1080,1060
    #                                          90000240   3.44   14500
    #                                          95000240   3.27   16000
    #                                         100000480   3.22   5540,860,2260,2420
    #                                                 1   2.81          -
    #                                         105000480   2.49   2040,1700,1700,900,2220,1840,1080,
    #                                         110000480   2.23   580,600,600,580,8780,1940,1680,168
    #                                          85000240   1.81   14500
    #                                         100000240   1.81   19100
    #                                          80000480   1.35   3380,280
    #                                          47284245   1.10   2438,2449
    #                                         110000240   1.04   21800
    #                                         110000161   0.86          -
    #                                          80000240   0.83   12900
    #                                         105000240   0.78   18800
    #                                          46172205   0.76   2531,2528
    #                                          49104325   0.71   995,996,996,993,996
...
```

Conclusions

Error propagation with automatic diferentiation

- **Robust:** If central values are correctly computed, errors will be correctly propagated.
- **Faster** and more **accurate** than alternatives based on resampling (i.e. jackknife, bootstrap)
- Correlations taken care of automatically
- **Exact** Error propagation in iterative algorithms: errors in fit paramers
- Bookkeeping of the contribution of each source to the error of a variable
- Autocorrelations of Monte Carlo data handled robustly

If you want to understand the theoretical ideas

A. Ramos, "Automatic differentiation for error analysis of Monte Carlo data". Comput.Phys.Commun. 238 (2019) 19-35.

Free implementation available

https://igit.ific.uv.es/alramos/aderrors.jl
(also fortran implementation available if interested)