NB3: Making our Toolbox

Written by Maria Salem, in supplement to master thesis

In this notebook we synthesize the ingredients we need to run our analyses in the results notebooks (NBRs):

1. We import the libraries we will use in th enext notebooks (NBRs).
2. We load in the .jld2 files with the wrangled data from notebook 1.
3. We reformulate the code (methods detailed in notebook 2) into functions.

When including this notebook in the NBRs we will thus have all the tools required to run our analyses efficiently.

---

1. Import libraries

Importing the necessary libraries for the next notebooks.

In [1]:

```
# loading in the necessary packages
using UncertainData, CausalityTools, DynamicalSystems, Distributions, RecipesBas
e,StatsBase, LaTeXStrings, JLD2,Plots; pyplot()
```

---

2. Import data

Recall the binned resampled time series data, as prepared in NB1. This way we can carry on the arrays in the next notebooks, where we will run the analyses.

In [2]:

```
#### d18O / sea level / ice volume proxies
@load "../WrangledDataFiles/Binned_ts_fullength/LR04.jld2" # LR04 - d18O stack
@load "../WrangledDataFiles/Binned_ts_fullength/SprattLisiecki.jld2" # Spratt&Li
siecki - global sea level stack, spanning last 800 kyrs (post-MPT)
@load "../WrangledDataFiles/Binned_ts_fullength/Elderfield.jld2" # Elderfield -
 global sea level record, spanning last 1.5 Myrs (pre- syn- & post-MPT)
@load "../WrangledDataFiles/Binned_ts_fullength/Grant.jld2" # Grant - Red Sea RS
L record, spanning last 500 kyrs (post-MPT)
@load "../WrangledDataFiles/Binned_ts_fullength/Rohling.jld2" # Rohling - Medite
rranean RSL record, spanning last 5.3 Myrs (pre- syn- & post-MPT)

#### insolation time series
@load "../WrangledDataFiles/La2004.jld2" # La2004 - numerical solution for north
ern hemisphere summer insolation, last 5 Ma computed using AnalySeries (Paillar
d, 1994). (pre- syn- & post-MPT).

#### pCO2 records/proxies
@load "../WrangledDataFiles/Binned_ts_fullength/Bereiter_nointp.jld2" # Bereiter
- post-MPT pCO2 record - with age model uncertainty
print("Bereiter BinnedResampled without interpolation")
@load "../WrangledDataFiles/Binned_ts_fullength/Bereiter_noageuncEDC_intp.jld2"
# Bereiter - post-MPT pCO2 record - without age model uncertainty (for analysis
 with Lambert EDC dust record). NB. this time series containts interpolated valu
es.
@load "../WrangledDataFiles/Binned_ts_fullength/Chalk.jld2" # Chalk - syn-MPT pC
O2 record (time interval 1.088-1.242 Ma BP, updated cut to 1.092-1.240 Ma BP)

#### dust records
@load "../WrangledDataFiles/Binned_ts_fullength/Lambert.jld2" # Lambert - post-M
PT Antarctic dust record (spanning last 800 kyrs)
@load "../WrangledDataFiles/Binned_ts_fullength/MartinezGarcia.jld2" # Martinez-
García - 4 Ma Southern Ocean Fe dust record (pre- syn- & post-MPT)

;
```

```
Bereiter BinnedResampled without interpolation
```

Define the first and last time point in each record. These will be used in the NBRs to determine the common grid.

In [4]:

```
# Define the first and last time point in each record

##### d18O record

# LR04
LR04 = LR04_binned_fullength_fullageunc
# assign to tmin_LR04 the first time value in LR04 (that is, the LR04 record sta
rts at ``tmin_LR04`` years BP)
tmin_LR04 = LR04.indices[1].value  # the binned LR04 record starts at -802 kyrs
 BP (binmidpoint)
tmax_LR04 = LR04.indices[end].value # -0.3 kyrs BP ...last age value in the LR04
record (binmidpoint)
print("The LR04 d18O record spans from ", tmin_LR04, " to ", tmax_LR04, " kyrs B
P.")

##### insolation time series

# La2004 (Ins)
La2004 = La2004_insol65N_fullength
#= recall, insolation is not an uncertain index value dataset type.
We therefore use two arrays (time index and insolation value) =#
tmin_La2004 = La2004_t_fullength[1] # 5000
tmax_La2004 = La2004_t_fullength[end] # 0 kyrs BP (present day)
print("
The La2004 insolation time series (Ins) spans from ", tmin_La2004, " to ", tmax_
La2004, " kyrs BP.")

##### GSL records

# SprattLisiecki (SpraSL)
SL = SL_binned_fullength_ageunc
tmin_SL = SL.indices[1].value # -797.0 kyrs BP
tmax_SL = SL.indices[end].value # -1.0 kyrs BP
print("
Spratt Lisiecki  GSL stack (SpraSL) spans from ", tmin_SL, " to ", tmax_SL, " ky
rs BP.")

# Elderfield (EldSL)
E = E_binned_fullength_ageunc
tmin_E = E.indices[1].value # -1574 kyrs BP
tmax_E = E.indices[end].value # -8 kyrs BP
print("
The Elderfield GSL record (EldSL) spans from ", tmin_E, " to ", tmax_E, " kyrs B
P.")

# Grant (GraSL)
G = G_binned_fullength
tmin_G = G.indices[1].value # -491 kyrs BP
tmax_G = G.indices[end].value # -1 kyrs BP
print("
The Grant sea level record (GraSL) spans from ", tmin_G, " to ", tmax_G, " kyrs
 BP.")

# Rohling (RohSL)
R = R_binned_full
tmin_R = R.indices[1].value # -5329 kyrs BP
tmax_R = R.indices[end].value # -1.0 kyrs BP
print("
The Rohling sea level record (RohSL) spans from ", tmin_R, " to ", tmax_R, " kyr
```

```
s BP.")



##### pCO2 records

# Bereiter (BerCO2)
B = B_binned_fullength_ageunc
tmin_B = B.indices[1].value # -802 kyrs BP
tmax_B = B.indices[end].value # -3 kyrs BP
print("
The Bereiter pCO2 record (BerCO2) spans from ", tmin_B, " to ", tmax_B, " kyrs B
P.")

# Chalk (ChaCO2)
C = C_binned
tmin_C = C.indices[1].value # - 1242 kyrs BP
tmax_C = C.indices[end].value # -1088 kyrs BP
print("
The Chalk pCO2 record (ChaCO2) spans from ", tmin_C, " to ", tmax_C, " kyrs BP."
)

##### dust records

# Lambert (IceDust)
L = L_binned_full
tmin_L = L.indices[1].value # -799 kyrs BP
tmax_L = L.indices[end].value # - 2 kyrs BP
print("
The Lambert dust record (IceDust) spans from ", tmin_L, " to ", tmax_L, " kyrs B
P.")

# Martinez-García (MarFe)
MG = MG_binned_fullength
tmin_MG = MG.indices[1].value # -3999 kyrs BP (4 Myrs)
tmax_MG = MG.indices[end].value # -2 kyrs BP
print("
The Martinez-Garcia Fe flux record (MarFe) spans from ", tmin_MG, " to ", tmax_M
G, " kyrs BP.")
MG_hr = MG_binned_postmpt_hr500
tmin_MG_hr = MG_hr.indices[1].value # -3999 kyrs BP (4 Myrs)
tmax_MG_hr = MG_hr.indices[end].value # -2 kyrs BP
print("
The higher resolution part of the MG record spans from ", tmin_MG_hr, " to ", tm
ax_MG_hr, " kyrs BP.")
```

```
The LR04 d18O record spans from -5320.0 to 0.0 kyrs BP.
The La2004 insolation time series (Ins) spans from -5000.0 to -0.0 k
yrs BP.
Spratt Lisiecki  GSL stack (SpraSL) spans from -797.0 to -1.0 kyrs B
P.
The Elderfield GSL record (EldSL) spans from -1574.0 to -8.0 kyrs B
P.
The Grant sea level record (GraSL) spans from -491.0 to -1.0 kyrs B
P.
The Rohling sea level record (RohSL) spans from -5329.0 to -1.0 kyrs
BP.
The Bereiter pCO2 record (BerCO2) spans from -803.0 to -2.0 kyrs BP.
The Chalk pCO2 record (ChaCO2) spans from -1240.0 to -1092.0 kyrs B
P.
The Lambert dust record (IceDust) spans from -799.0 to -13.0 kyrs B
P.
The Martinez-Garcia Fe flux record (MarFe) spans from -3999.0 to -2.
0 kyrs BP.
The higher resolution part of the MG record spans from -800.0 to -0.
5 kyrs BP.
```

3. Writing the functions to be used in the NBRs for computing the analyses.

3.1 `cut_timeinterval_from_uivD()`

For our method to work, the time series must by synchronous and of equal length. In every NBR, we select a time window for analysis. Here we define the function to cut the binned uivDs at that selected time interval.

We select the time series' common time array as following: all time values greater (younger) than, or equal to, the common grid's starting midpoint `tmin` , and all values smaller (older) than, or equal to, the common grid's ending midpoint `tmax` .

In [6]:

```
# We define a function to cut the uncertain index value datasets

function cut_timeinterval_from_uivD(
    # Input Arguments

        ts::UncertainIndexValueDataset, # the time series to be cut
        tmin::Any,                      # start of the time interval to keep
        tmax::Any,                      # end of the time interval to keep
        bmp_cg::StepRange{Int64,Int64}  # bin midpoints of the common grid
        )


    # Method

    # select time interval
        #= Since the ``=`` (EQUAL TO operator) does not work for the type Uncert
ainIndexValue, we
        instead use ``>`` a value a littlebit below tmin, and ``<``a value a lit
tlebit above tmax =#
    ts_cut = UncertainIndexValueDataset(
    ts.indices[(ts.indices .> tmin-0.001) .& (ts.indices .< tmax+0.001)], # retu
rns the time array (certain values if BinnedResampled)
    ts.values[(ts.indices .> tmin-0.001) .& (ts.indices .< tmax+0.001)]   # retu
rns the associated uncertain values array
    )


    # control that the time interval has been correctly selected
    # by checking if the bin midpoints (bmp) are the same for the time series an
d the common grid

    bmp_ts =[ts_cut.indices[i].value for i in 1:length(ts_cut.indices)] # bin mi
dpoints of the cut time series

    if bmp_ts == bmp_cg
    print("The record is now on the common time grid")
    else
    print("Something's wrong")
    end



    return ts_cut


end
```

Out[6]:

```
cut_timeinterval_from_uivD (generic function with 1 method)
```

In [7]:

```
# testing the function above
X_test = LR04
tmin_test = -400
tmax_test = -100
binsize_test = 1
testgrid_bmp = tmin_test : binsize_test : tmax_test
commongrid_test = tmin_test - binsize_test/2 : binsize_test : tmax_test + binsiz
e_test/2

X_test_cut = cut_timeinterval_from_uivD(X_test, tmin_test, tmax_test, testgrid_b
mp)
# Yay, the function works!

Y_test = B_binned_fullength_ageunc
Y_test_cut = cut_timeinterval_from_uivD(Y_test, tmin_test, tmax_test, testgrid_b
mp)
```

```
The record is now on the common time gridThe record is now on the co
mmon time grid
```

Out[7]:

```
UncertainIndexValueDataset{UncertainIndexDataset,UncertainValueDatas
et} containing 301 uncertain values coupled with 301 uncertain indic
es
```

3.2 `computePredictiveAsymmetries()`

Define a function to compute the transfer entropies and predictive asymmetries (defining the code from NB2 as functions).

The raw cell below is a **recap of the code steps shown in NB2**. We will use these steps as methods in the function we define to compute the predictive asymmetry from time series.

This is a recap to recall the steps done in notebook 2. We will be used as input to create a function "from A to Z" #### defining the parameters needed for estimation of TE and PA binning = RectangularBinning(4) # coarse-grained grid for analysis ηmax = 20 # Defining the number of prediction lags η ηs = -ηmax : ηmax # prediction lags #### Defining which test will be used to estimate Transfer Entropy (VisitationFrequencyTest) TE_test = VisitationFrequencyTest(binning = binning, ηs = ηs) PA_test = PredictiveAsymmetryTest(TE_test); # PA_XtoY = causality(X, Y, PA_test) # Check if X improves prediction of Y # PA_YtoX = causality(Y, X, PA_test) # Check if Y improves prediction of X #### Defining the random sequences tests (to estimate level of confidence for TE and PA) n_chunks = 150 # number of random sequences (chunks) to run the PA-test on N = length(bin_midpoints) # length of our original time series chunks = RandomSequences(n_chunks, N-30 : N-1) # all 150 chunks have length near the true time series length print(chunks) #### defining the random sequences test rsTE_test = RandomSequencesTest(TE_test, chunks) # This function will calculate Transfer Entropy-values for the 150 chunks defined above rsPA_test = RandomSequencesTest(PA_test, chunks) # This function will calculate Predictive Asymmetry-values for the 150 chunks defined above

Define a function for computing the normalized predictive asymmetry.

We here make a function that summarized all the steps explicitly shown in notebook 2: from the time series binned on a common grid, to the output results of TE and $\mathbb{A}$ needed to calculate the $\mathcal{A}$ from one time series to the other (see main text). Note: we include the step to normalize the predictive asymmetry, so that we can set a universal significance treshold.

INPUT arguments are the X and Y time series binned on a common time grid `X`, `Y`, as well as `timestep` (the prediction lag \eta).** For time series with associated uncertainties, the uncertainty is carried on as KDE in an `UncertainIndexValueDataset`. For time series without any associated uncertainty, the time series sent to analysis will simply be an `array` of values, binned to the common grid.

- **OUTPUT** are arrays of transfer entropy and predictive asymmetry computed for the array of \etas, of a family of iterations (random sequences)**.

Saving the results in a .jld2 file. The computation of transfer entropy, predictive asymmetry and the random sequence tests are computationally heavy operations. Saving the results in a file spares us from having to compute these each time we re-open the notebook. We also save the binned time series (`X` and `Y`), and the amount of prediction lag used (`\etamax`) so that we can remake the plots.

In [10]:

```
function computePredictiveAsymmetries(
# INPUT ARGUMENTS:
    X::Any,                     # source time series
    Y::Any;                     # target time series
# optional kwargs:
    timestep::Number            # prediction lag η
        = 1,                        # default
    ηmax::Int                   # number of prediction lags,
        = Int(20/timestep),         # default is 20 kyrs worth of prediction la
gs
    N::Number = length(X),
    ϵ::Int                      # determines the number of bins in the gridding
 of state space (affects probability distribution of transfer entropy estimatio
n)
        = Int(round(length(X)^(1/(3+1)) )),  # Default binning is set by the Pal
us horizon
    rs::RandomSequences         # defines the number of random sequences and the
ir length (used for quantifying confidence intervals)
        = RandomSequences(150, ceil(Int, 0.7*length(X)) : (length(X)-1) ), # def
ault = minimum 70% of total length of the time series)
    filepath::String,           # assign a filepath for where to save the result
s
    )


# METHODS

    #N = length(X)                                  # full length of time series
 (length of common grid)
    #eD = 3                                         # embedding dimension (defaul
t = 3 in the visitation frequency test)
    #ϵ = Int(round( N^(1/(eD+1)) ))                 # The `Palus` horison, optima
lisation of binsize ϵ as function of time series length
    binning = RectangularBinning(ϵ)                # binning in Visitation Freque
ncy test (our TE-test). Default is 4
    #ηmax::Int64 = Int(10/timestep)                 # Defining the number of predi
ction lags η, dependant of time series resolution (binsize)
    ηs = -ηmax : ηmax                              # prediction lags from -10 to 1
0 (e.g. captures lags between driver/response of up to 10 timesteps (10 kyrs))

    TE_test = VisitationFrequencyTest(binning = binning, ηs = ηs) # Estimate Tra
nsfer Entropy by VisitationFrequencyTest

    PA_test = PredictiveAsymmetryTest(TE_test)                    # Estimating p
redictive asymmetry from the results obtained of transfer entropy

    # Defining the random sequences tests (will be used for quantifying confiden
ce)
    rs = RandomSequences(150, ceil(Int, 0.7*N) : (N-1)) # defining 150 random se
quences of little below full length of time series (here we set as minimum 70% o
f total length of the time series)
    rsTE_test = RandomSequencesTest(TE_test, rs)        # Calculate Transfer Ent
ropy-values for the 150 random sequences
    rsPA_test = RandomSequencesTest(PA_test, rs)        # Calculate Predictive A
symmetry-values for the 150 random sequences

    # running the random sequences test to get a family of results:

        # X as source and Y as target time series
```

```
    rsTE_XtoY = causality(X, Y, rsTE_test) # Estimate transfer entropy from X to
Y for every random sequence
    rsPA_XtoY = causality(X, Y, rsPA_test) # Check if X improves prediction of Y
for every random sequence

    rsTE_YtoX = causality(Y, X, rsTE_test) # Estimate transfer entropy from X to
Y for every random sequence
    rsPA_YtoX = causality(Y, X, rsPA_test) # Check if X improves prediction of Y
for every random sequence


    @save filepath X Y ηmax rsTE_XtoY rsPA_XtoY rsTE_YtoX rsPA_YtoX  # Save all
 the elements needed to plot the results.
    return print("Results are saved in the .jld2 file. ")

end
```

Out[10]:

```
computePredictiveAsymmetries (generic function with 1 method)
```

In [11]:

```
#test that the function works

@time computePredictiveAsymmetries(X_test_cut, Y_test_cut, timestep = 1, ηmax =
20,
    filepath = "../../results_ePalus_ns20/test_LRO4_BerCO2.jld2")
```

```
Results are saved in the .jld2 file.  40.039355 seconds (163.32 M al
locations: 11.044 GiB, 14.22% gc time)
```

3.3 `normalizePredictiveAsymmetry()`

Make a function to normalize the predictive asymmetry

In [12]:

```
# MAKE A FUNCTION TO NORMALIZE THE PREDICTIVE ASYMMETRY


function normalizePredictiveAsymmetry(

#input arguments

    rsTE::Array{Array{Float64,1},1}, # arrays of transfer entropies for a family
of random sequences
    rsPA::Array{Array{Float64,1},1}; # arrays of predictive asymmetries for a fa
mily of random sequences
    ηmax::Int64, # number of prediction lags
    f::Number, # false positive rate
    )

# method

    # concatenate the results to get them in the right format to be input in the
following step
    TE = hcat(rsTE...)
    PA = hcat(rsPA...)

    # normalize the predictive asymmetry results
    normPA = vcat([mean(PA[1:i, :] ./ (mean(TE[ηmax+1-i:ηmax+i, :]) * f), dims =
1) for i in 1:ηmax]...)

    return normPA
end
```

Out[12]:

```
normalizePredictiveAsymmetry (generic function with 1 method)
```

Test the `normalizePredictiveAsymmetry` function

In [13]:

```
@load "../../results_ePalus_ns20/test_LRO4_BerCO2.jld2"
normPA_XtoY = normalizePredictiveAsymmetry(rsTE_XtoY, rsPA_XtoY, ηmax = ηmax, f
= 1)
normPA_YtoX = normalizePredictiveAsymmetry(rsTE_YtoX, rsPA_YtoX, ηmax = ηmax, f
= 1)
```

Out[13]:

```
20×150 Array{Float64,2}:
 1.65736  2.1011   1.52212  1.06004  0.66377   …  0.491908  0.61201
1.26502
 1.84086  1.8123   1.36969  1.39749  0.953341     0.791655  0.700089
1.16981
 2.37167  2.05368  1.6033   1.60017  1.16194      1.14752   0.850134
1.28406
 2.96161  2.3204   1.92958  1.78864  1.27411      1.41345   1.04302
1.44289
 3.45293  2.51857  2.2674   1.96914  1.43911      1.6742    1.06187
1.56951
 3.93301  2.71388  2.61173  2.11029  1.61651   …  1.86136   1.04605
1.69016
 4.41     2.84337  2.93592  2.24254  1.79543      1.97452   1.02232
1.71465
 4.82674  2.94482  3.22854  2.3652   1.96425      2.07019   0.979834
1.76552
 5.24049  3.05267  3.51601  2.49521  2.13814      2.16244   0.995427
1.79557
 5.63151  3.14855  3.77537  2.6071   2.29299      2.25078   1.03519
1.84143
 5.9952   3.26236  4.04321  2.71514  2.43355   …  2.37718   1.09922
1.89698
 6.36551  3.3653   4.31304  2.82011  2.59735      2.49161   1.18585
1.94862
 6.70877  3.48223  4.55539  2.91732  2.77261      2.58837   1.25033
2.00983
 7.05048  3.56975  4.80445  3.01976  2.95582      2.66323   1.3095
2.07734
 7.36651  3.64397  5.02142  3.10108  3.14978      2.72164   1.34962
2.16554
 7.65365  3.71049  5.20801  3.17844  3.34858   …  2.74665   1.35167
2.22781
 7.9181   3.73555  5.36769  3.24777  3.5475       2.75128   1.34468
2.28551
 8.1703   3.76426  5.54555  3.30773  3.73658      2.75488   1.33479
2.33156
 8.36931  3.78688  5.70053  3.35362  3.91982      2.72757   1.30463
2.36949
 8.54671  3.80413  5.83553  3.38434  4.07849      2.68198   1.24775
2.38942
```

In [14]:

```
size(normPA_XtoY) # 150 arrays, showing predictive asymmetries integrated over t
he 20 ηs
```

Out[14]:

```
(20, 150)
```

3.4 `quantiles_normPA()` (disused)

Make a function to calculate arrays of quantiles to define a confidence interval of predictive asymmetry. The confidence ascribed is based on to which extent the results from the random sequences differ.

In [15]:

```
function quantiles_normPA(
# input arguments
    normPA::Array{Float64,2}; # concatenated array of normalized predictive asym
metries over ηmax predicion lags -  the 150 iterations (random sequences) is bas
is for our confidence intervals
    upperquantile::Float64 = 0.975, # upper quantile (default for the 95% confid
ence interval)
    lowerquantile::Float64 = 0.025, # lower quantile
    ηmax::Int                       # Number of time steps/prediction lags we wi
ll calculate the quantiles for
    )

# methods
# calculate the quantiles for the 95% confidence interval
normPA_median = [quantile(normPA[i,:], 0.5) for i in 1:ηmax]                # cal
culate the median normPA of the 150 random sequences iterations of PA, one for e
ach prediction lag
normPA_upper = [quantile(normPA[i,:], upperquantile) for i in 1:ηmax] .- normPA_
median  # calculate the upper quantile
normPA_lower = normPA_median .- [quantile(normPA[i,:], lowerquantile) for i in 1
:ηmax]   # calculate the lower quantile

return normPA_median, normPA_upper, normPA_lower
end
```

Out[15]:

```
quantiles_normPA (generic function with 1 method)
```

test the `quantiles_normPA` function

In [16]:

```
# Calculate the quantiles to plot the normalized predictive asymmetry median and
the 95% confidence interval
normPA_XtoY_median, normPA_XtoY_upper, normPA_XtoY_lower = quantiles_normPA(norm
PA_XtoY, upperquantile = 0.975, lowerquantile = 0.025, ηmax = ηmax)
normPA_YtoX_median, normPA_YtoX_upper, normPA_YtoX_lower = quantiles_normPA(norm
PA_YtoX, upperquantile = 0.975, lowerquantile = 0.025, ηmax = ηmax)
```

Out[16]:

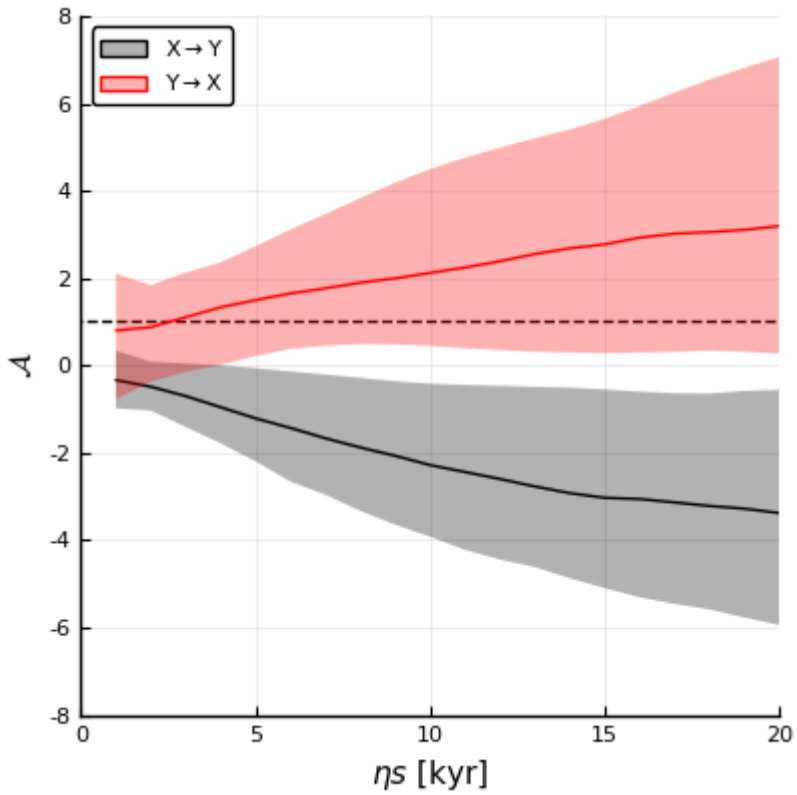
```
([0.7986570645871822, 0.8773275394320009, 1.1042813249023955, 1.3313
436660147735, 1.4946812551033577, 1.6478180857550693, 1.763969003013
1603, 1.8941421416785937, 1.9928330834971444, 2.1170406895839715, 2.
241065517571085, 2.383707574287336, 2.549302592625722, 2.67927473427
99594, 2.7701428466903564, 2.9239491966076567, 3.0166153344888897,
3.051675031327764, 3.102152886538459, 3.1908651514712356], [1.303878
003030131, 0.9586066191647292, 1.0211039628974286, 1.027919694689989
3, 1.2313435161724346, 1.4612858036316982, 1.703573035010574, 1.9323
486586783147, 2.1894600639705404, 2.3755544602283463, 2.513420286682
8477, 2.60003055529509, 2.649656929342491, 2.7185373084297093, 2.873
2154187332433, 3.0073719770668825, 3.2251082279715284, 3.48830576593
86516, 3.706935641806983, 3.8648925931559597], [1.5591306891842136,
1.2479363432598167, 1.2474900705045961, 1.3158057341157354, 1.279535
4704503283, 1.2680380850680133, 1.3170160474168808, 1.40396960342643
86, 1.5109806214435282, 1.6712938635549257, 1.845110149003006, 2.031
399771854259, 2.2356052847631522, 2.3814353043253407, 2.491245165250
458, 2.6265483132343475, 2.70617742809129, 2.708874804254888, 2.7871
74169077116, 2.919755250940006])
```

In [18]:

```
# defining the results plot

test_pAA =
plot(xlims = (0, ηmax), xticks = (0 : 5 : ηmax),
    ylims = (-8,8), yticks = (-8:2:8),
    legend = :topleft,
    xlabel =  string(L"ηs"," [kyr]"),
    ylabel =  L"\mathcal{A}",
    size = (400,400),
    grid = true,
    hline([1], line = (:dash, :black)),
    label =  ""
    )
plot!(normPA_XtoY_median,
    ribbon = (normPA_XtoY_lower, normPA_XtoY_upper),
    label =  string("X", L"\rightarrow", "Y"),
    fillalpha = 0.3, legend = :topleft, color = :black
    )
plot!(normPA_YtoX_median,
    ribbon = (normPA_YtoX_lower, normPA_YtoX_upper),
    label =  string("Y", L"\rightarrow", "X"),
    fillalpha = 0.3, legend = :topleft, color = :red)
```

Out[18]:



Summary

1. The `computePredictiveAsymmetry function` will return predictive asymmetry ($\mathcal{A}$) from time series X to time series Y, and vice versa. Both the transfer entropy (TE) and the predictive asymmetry ($\mathbb{A}$) is computed for 150 random sequences of nearly full time series length. This allows us to quantify our confidence in the results. We will visualize the 95% confidence interval with an envelope on the plots. Since running the function is computationally heavy, the results are saved in a .jld2 file.
2. The significance level is set in the normalization function `normalizePredictiveAsymmetry` .

Way forward in the next notebooks (NBRs)

We will include this notebook to run the analyses and produce results in the following notebooks (NBRs). The data and functions will thereby be available to compute the results.

- Each NBR is defined by the time interval we run analysis over. The selection of time interval is made based on which time intervals we wish to compare the dynamics of (pre-, syn- and post-MPT), as well as the temporal overlap of the records we have available.

Getting the records on a common time grid. To avoid unnecessary iterations of the computationally heavy `BinnedResampling` function, we set the records to a common time grid for analysis in a two-step process: First, the records were binned to regular time grids with a common time step of 1 kyr (and additionally on 500 and 125 yr, for records where we had higher resolution data available). Next, in the NBRs, the common time interval is selected.

1. Cut the binned time series to the common time interval.
2. Compute a family of transfer entropy and predictive asymmetry results, using the function `computePredictiveAsymmetry` .
3. Normalize the predictive asymmetry results, to get a comparable scale.
4. Plot the results.