NB_Method

Notebook author Maria Salem, as supplement to master thesis

Causal analysis framework

"Behind the scenes" of computing predictive asymmetry

NB2

In this notebook, we introduce the theoretical background behind the predictive asymmetry statistic. We use the example of a synthetic system where we already know the causal coupling. By creating two time series X and Y of the system's variables (equivalent to empirical records), we show how predictive asymmetry is estimated to infer the causal relationship between the timeseries.

The notebook walks through all the steps in the predictive asymmetry analysis, with abundant comments and explanatory figures. The aim is to give a general understanding of predictive asymmetry and of what is done in the NBRs. In NB3, the computation of predictive asymmetry is synthesized in one function, so this notebook shows what's "under the hood" of this function.

Outline

1. Create a synthetic system of which we know the causal coupling. We use an autoregressive system of the first order. The stochastic component of the system makes it a representative model of all the complexity that would be in a natural system (in our case, the Pleistocene climate system). **2. Create time series for the variables in the system.** This will be equivalent to the empirical records we have of the system we want to study (in our case, the paleoclimatic variables of global sea level, insolation, pCO2, and dust).

> After synthetic time series are represented, include note on reflections/challenges for using real-world empirical time series - (e.g. resolution, time series continuity). Introduce binned resampling here instead?

3. Binned resampling of time series on a common time grid. (Equivalent to NB1)

- Note: When we create time series data form a synthetic system, we can assign the discrete values (observations) to a regular time grid. Empirical data, however, seldomly give observations on a regular time grid. To get empirical data records on a regular grid it is necessary to a do binned resampling (like we do in NB1). To make the steps in this notebook as similar as possible to the steps in our analysis of empirical data, we will therefore also include binned resampling of the synthetic time series.

Once we have created our time series data, we move on to showing how the analysis is done. (These steps 4-6 are synthesized in the function `function_from_XY_to_normPA` , in NB3):

4. Estimate transfer entropy from one time series to the other.

- use **visitation frequency** to estimate transfer entropy
- include figure of **state space reconstruction of the system, with binning grid** to show what we are talking about.
- The entropy is a trait of probability distribution, and so the number of bins in the state space reconstruction naturally will affect how the entropy is described. We will determine the **binning criterion** ϵ according to the *Palus horizon*, which is the binning optimization given by Palus [...]
- If we have a component of randomness in our system (which we do, to aliken the system to complex natural systems), estimations of transfer entropy will differ slightly with each iteration. We illustrate this with figures; plot showing different iterations of trasfer entropy estimation; plot showing the transfer entropy distribution / confidence interval (realizations from random sequences test) =
- We conclude that transfer entropy does not give unambiguous indications of causal connectivity/directionality. That is why the predictive asymmetry method has been developed.

5. Estimate predictive asymmetry from one parameter to the other

- Plot to show: EXPLAIN - difference between backwards and forwards in time predicition [NEED TO REVIEW THIS].

6. Normalize the predictive asymmetry results,

- In order to have a comparable scale of our results, we have to normalize the predictive asymmetry results.
- The false positive rate, set by *f*, depends on the type of system and the time series length, but is largely on the scale *f = 1* (determined heruristically by Haaga et al. (2020) from a series of experiments with synthetic systems).

The *normalized predictive asymmetry* is the result of interest in the analysis of our empirical data, since it gives a common scale to compare coupling strength, and allows us to draw conclusions (in the same way as a null-hypothesis) based on a significance treshold. In NB3, we have written steps 3-5 into one single function (`function_from_XY_to_normPA`), to optimize the code of the NBRs. This function allows us to compute the normalized predictive asymmetry directly from one time series to the other in one single operation.

Preamble: import the necessary packages'

In [1]:

```
using
UncertainData,  # Data types (uivD) and methods (BinnedResampling) to handle unc
ertain data
CausalityTools, # Built in function ar1_unidir to generate the synthetic system;
Functions to run causality analyses
Distributions,  #  Allows us to define confidence intervals as uniform or normal
ly distributed data
StatsBase,
LaTeXStrings,
Plots; pyplot();

#Test,
#Interpolations,
#Measures,
#DynamicalSystems
```

1. Create a syntehtic system

Disclaimer: First part of the notebook is written using kahaaga example on earthsystemevolution.com [https://www.earthsystemevolution.com/project/uncertaindata/ (https://www.earthsystemevolution.com/project/uncertaindata/)]:

"Consider a case of two unidirectionally coupled first-order autoregressive (AR1) processes."

- (an autoregressive model is a representation of a type of random process; as such, it is used to describe certain time-varying processes, where the state in the next time step is dependant on the previous. One common example is a random walk.)

"Let's take as an example of a system with two variables that are unidirectionally coupled (X drives Y). This does clearly not bear much resemblance to the high complexity we find in natural systems, which are often highly complex. To represent all the interconnectons we would find in a natural system, we include some degree of randomness in our synthetic system" (autoregressive model has a stochastic component).

- (One might argue that a unidirectional system with only one causal coupling cannot in any way validate the use of this method for natural systems with high complexity. This is the reason why we include randomness, representing all the variations arising from the complexity of a natural system.)

In [2]:

```
# Define the system
s = ar1_unidir(c_xy = 0.8)

# ar1_unidir is built into the CausalityTools package
    # representing an auto-regressive system (meaning the next time step is depe
ndant on the previous)
    # of the first order = means that only two variables?
# c_xy defines the coupling strength from x to y


#(...?) parameters = [x-coeff, y coeff, coupling strength, sigma?] # check artic
le
```

Out[2]:

```
2-dimensional discrete dynamical system
 state:       [0.73022, 0.925098]
 e.o.m.:      eom_ar1_unidir
 in-place?    false
 jacobian:    ForwardDiff
 parameters:  [0.90693, 0.40693, 0.8, 0.40662]
```

`ar1_unidir` is built into the CausalityTools package, and defines an auto-regressive system (meaning the next time step is dependant on the previous).

> [What does FIRST ORDER refer to?]. $unidir$ refers to unidirectional forcing from source to target (X → Y). The `c_xy` argument allows you to choose the coupling strength from X to Y. In this case we have given 0.8, stating that the signal in X will determine 80% of the signal in Y [in the next timestep?], and the remaining 20% of the signal comes from the stochastic component of the system.

I found myself asking, *wouldn't it be a closer-to-reality representation of the complexity of natural systems if we chose a high-order system, and only collected two of the variables?* This is the very reason we incorporate a stochastic component in our synthetic system - the randomness is meant to represent the complexity in a natural system.

> Note to self: I want to modulate the coupling strength `c_xy` and see how it affects the predictive asymmetry.

> ?Possible to plot the real thing attractor here**? since we "know" the governing functions of the system? Or only possible to show shadow attractor by embedding?

2. Creating time series (observations) of the system's variables

Now, let's create a 'record of observations' for each of the two system variables changing over time. These time series X and Y will be the equivalent to the empirical data.

With the function `example_uncertain_indexvalue_datasets` , we record N points from the built-in ar1_unidir system, collect the 1^{st} and 2^{nd} array of observations as out time series X and Y, and add some uncertainties to both the indices and the values.

In [3]:

```
?example_uncertain_indexvalue_datasets # Function documentation
```

```
search: example_uncertain_indexvalue_datasets
```

Out[3]:

```
example_uncertain_indexvalue_datasets(system::DynamicalSystems.Discr
eteDynamicalSystem, n::Int, vars; Ttr = 1000,
    d_xval = Uniform(0.01, 0.4), d_yval = Uniform(0.01, 0.5),
    d_xind = Uniform(0.5, 1.5), d_yind = Uniform(0.5, 1.5))
```

Generate a pair of `UncertainIndexValueDataset` s from a discrete dynamical `system` , generated by iterating the system `n` time after a transient run of `Ttr` steps, then gathering the columns at positions `vars` (should be two column indices) as separate time series.

Each of the time series, call them `x` and `y` , are then converted to uncertain values. Specifically, replace `x[i]` and `y[i]` with `UncertainValue(Normal, x[i], rand(d_xval)` and `UncertainValue(Normal, y[i], rand(d_xval)` . Because the time series don't have explicit time indices associated with them, we'll create some time indices as the range `1:tstep:length(x)*tstep` , call them `x_inds` and `y_inds` . The time indices for `x` and `y` are also normally distributed, such that `x_inds[i] = UncertainValue(Normal, i, rand(d_xind)` , and the same for `y_inds` .

Returns a tuple of `UncertainIndexValueDataset` instances, one for `x` and one for `y` .

In [4]:

```
# Create time series of the two parameters in the system
# (this will be the equivalent to our empirical data)

N = 100 # Setting the time series N length to 100 values

X, Y = example_uncertain_indexvalue_datasets(
    s,     # the synthetic system, defined in cell above ## system::DynamicalSyst
ems.DiscreteDynamicalSystem
    N,          # time series length ## n::Int
    (1, 2),     # X first, Y second parameter. ## vars
    tstep = 5, # timestep  ## Ttr??
    d_xval = Uniform(0.1, 0.3), # uniform distribution of x-values within (1σ) r
ange ([0.1,0.3])
                                # Chose values, 1σ ranges between [0.1, 0.3], un
iform = with uniform distribution (all values same likelyhood)
                                # HOW CAN IT BE 1σ if not normal distribution?
    d_yval = Uniform(0.1, 0.3), # ditto for y-values
    d_xind = Uniform(1, 3),     # ditto for x-indices
    d_yind = Uniform(1, 3)      # ditto for y-indices
    )
```

Out[4]:

```
(UncertainIndexValueDataset{UncertainIndexDataset,UncertainValueData
set} containing 100 uncertain values coupled with 100 uncertain indi
ces
, UncertainIndexValueDataset{UncertainIndexDataset,UncertainValueDat
aset} containing 100 uncertain values coupled with 100 uncertain ind
ices
)
```

In [5]:

```
# We check that we were returned X and Y as a tuple, and that they can be separa
ted
X # one UncertainIndexValueDataset (uivD)
Y # another uivD
```

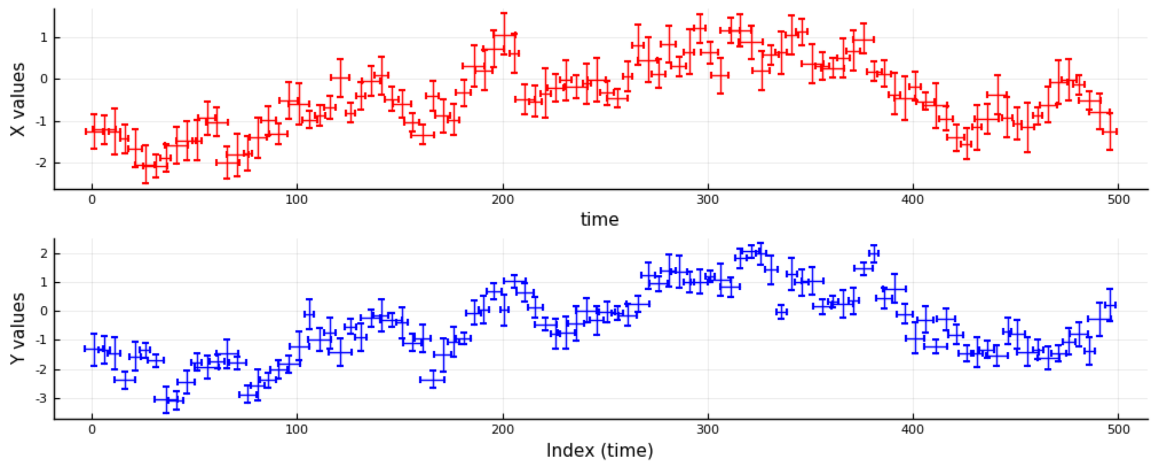
Out[5]:

```
UncertainIndexValueDataset{UncertainIndexDataset,UncertainValueDatas
et} containing 100 uncertain values coupled with 100 uncertain indic
es
```

Plot the time series

In [95]:

```
# Plot the time series of X and Y

qs = [0.025, 0.975] # we want to plot the quantiles of the 95 % confidence inter
val

# plot X time series
plot_X = plot(X,
    ylabel = "X values",
    xlabel = "time",
    #label = "X",
    #ms = 1,
    color = :red,
    qs, # quantiles for the 95% CI on the x-axis?
    qs # quantiles for the 95% CI on the y-axis?
    )

# plot Y time series
plot_Y = plot(Y,
    xlabel = "Index (time)",
    ylabel = "Y values",
    # label = "Y",
    # ms = 1,
    color = :blue,
    qs, qs
    )

# subplot of X and Y
plot(plot_X,
    plot_Y,
    size = (1000,400),
    layout = grid(2,1),
    legend = true,
    link = :x
    )

savefig("../../MASTER_2.0/figurar/4_metode/AR1_timeseries_uivD.pdf")
```



In [ ]:

Not sure if to include the correlation section below?

To what degree is there correlation between these two records? We can quantify this by use of the *Pearson correlation coefficient*, a statistic that measures linear correlation between two variables X and Y. It has a value between [–1, +1], +1 signifying x and y are perfectly correlated, values near 0 meaning no correlation, and -1 being perfectly anticorrelated (antiphase).

In [7]:

```
# define a function to compute the pearson correlation coefficient () of two arr
ays x and y

function pearson_correlation(x,y)
   n = length(x)
   teljar = n*sum(x .* y) - sum(x)*sum(y)
   nemnar = (n * sum(x .^ 2) - sum(x)^2) * (n*sum(y .^ 2) - sum(y)^2)

   teljar / sqrt(nemnar)
end


# pearson_correlation(X,Y) # I guess we can't use this due to data type uivD

# Let's redefine the data type to check the correlations of the records

    # me kan bruke ulike realiseringer av datasettet
    # eller me kan hente ut mean

X.values.values[1].μ # parametric mean of the density function
X_mean = [mean(X.values.values[i]) for i in 1:N] # computing the sample mean (he
re same as the parametric function) using the mean function
Y_mean = [mean(Y.values.values[i]) for i in 1:N]


pearson_correlation(X_mean,Y_mean) # Fairly well correlated
```

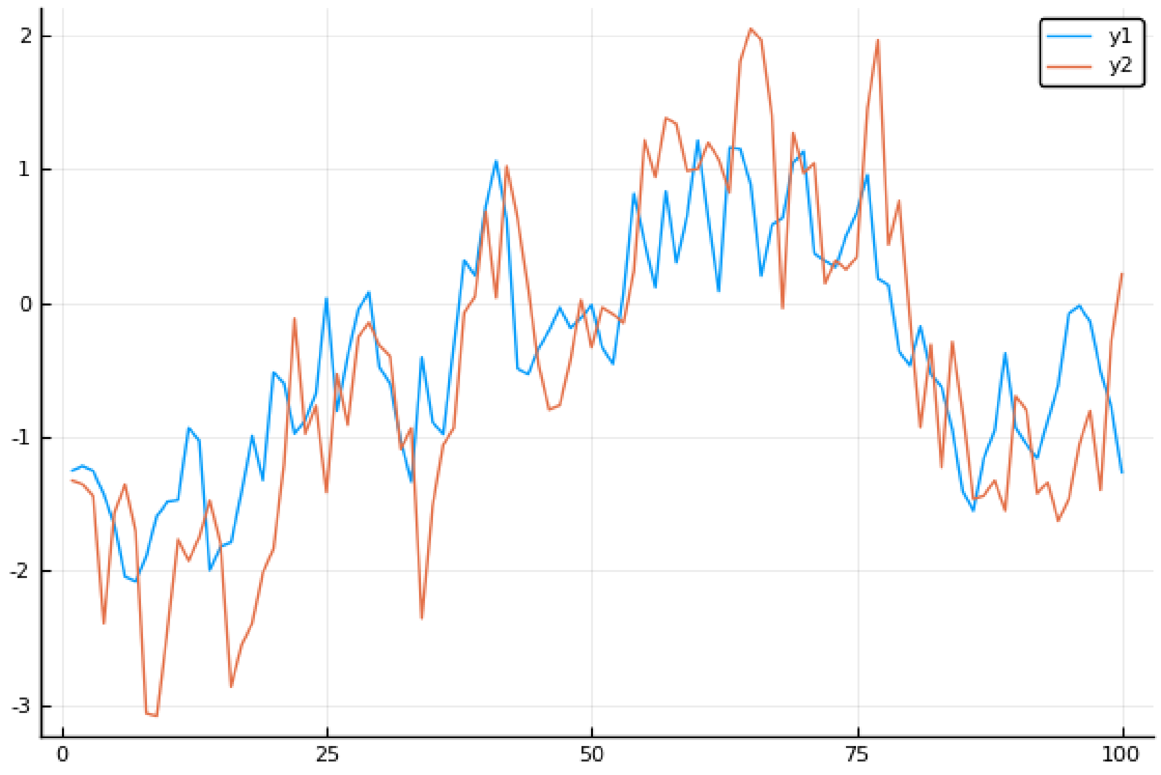
Out[7]:

```
0.797390443416636
```

In [8]:

```
plot(X_mean)
plot!(Y_mean)
```

Out[8]:

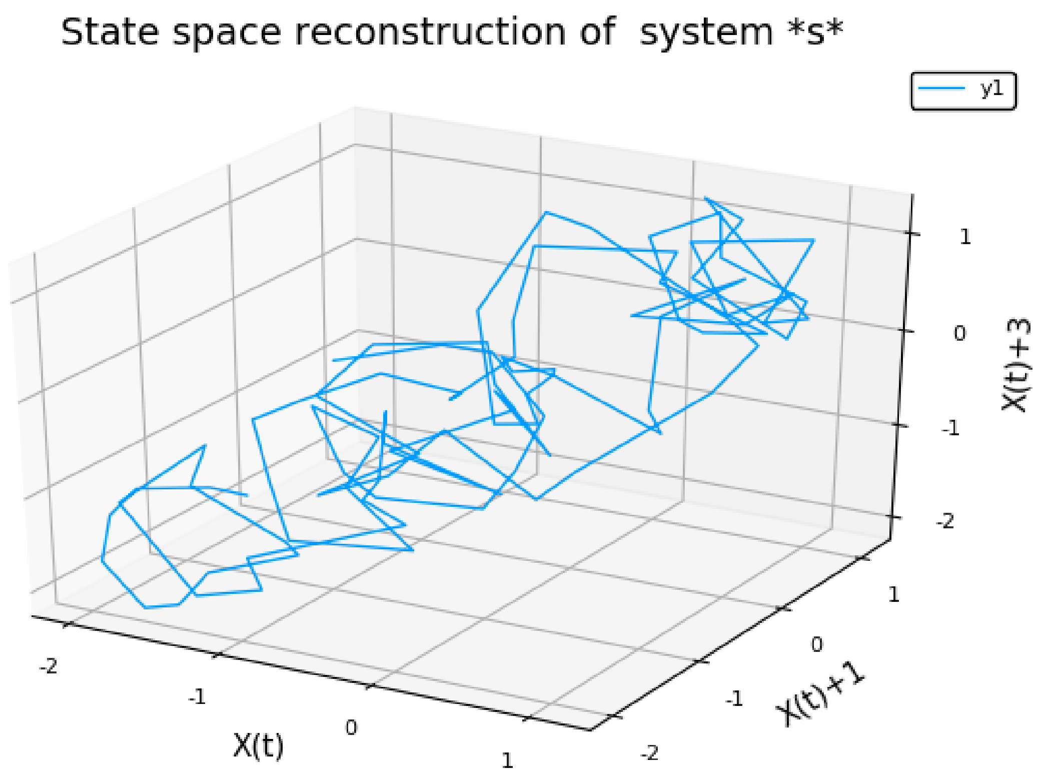


The information of the overarching dynamics defining our system *s* plays out in the time series of each of the system's variables. We can use delay embedding of the time series to make a state space reconstruction of the system's dynamics.

In [12]:

```
# Embedding of time series X by time lags
    # Am I using the terms correctly? Probably not..

length_X = length(X) #100

# pick any discrete number (Integer) as the time step of the time lag
timelag_1 = rand(1:length_X) #61
timelag_2 = rand(1:length_X) #84

# actually, let's pick smaller numbers, so that we don't lose that much of the t
ime series length
timelag_1 = 1
timelag_2 = 3

# make the lagged time series of the same length
     # MEthodError (probably bcs uivD)
X_unlagged = X_mean[(1 + timelag_2) : length_X]
X_lagged1 = X_mean[(1 + timelag_1) : (length_X - timelag_2 + timelag_1)]
X_lagged2 = X_mean[1 : (length_X - timelag_2)]

using Plots; pyplot()

# make a 3D plot of the shadow attractor
shadow_attractor_X = plot(X_unlagged, X_lagged1, X_lagged2,
    title = "State space reconstruction of  system *s*", # stochastic dynamical
 system
    xlabel = "X(t)", ylabel = "X(t)+1", zlabel = "X(t)+3")
```

Out[12]:



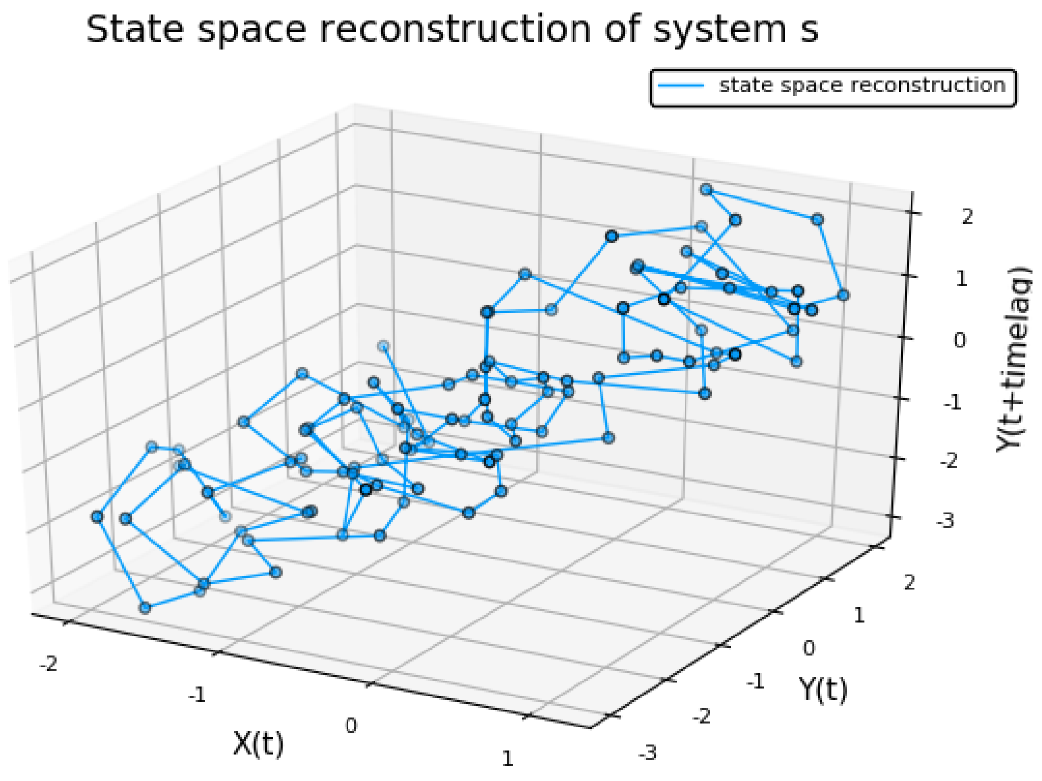
Let's do a 3-dimensional embedding using X, Y and Y + timelag (the parameters needed to calculate transfer entropy)

In [13]:

```
# make the lagged time series of the same length
     # MEthodError (probably bcs uivD)
X_unlagged = X_mean[(1 + timelag_1) : length_X]
Y_unlagged = Y_mean[(1 + timelag_1) : length_X]
Y_lagged = Y_mean[1 : (length_X - timelag_1)]

#using Plots; plotlyJS()
plot(X_unlagged, Y_unlagged, Y_lagged,
    xlabel = "X(t)", ylabel = "Y(t)", zlabel = "Y(t+timelag)",
    title  = "State space reconstruction of system s",
    label = "state space reconstruction", marker = :dotline)
```

```
┌ Warning: Skipped marker arg dotline.
└ @ Plots /Users/maria/.julia/packages/Plots/qZHsp/src/args.jl:760
```

Out[13]:



> , Question if I understood it right: Is the attractor = most probable dim configuration, aka state with the highest entropy?,
>
> > Ikkje bland transfer entropy og information entropy - missing link er kullback liebler divergens
>
> Also, when estimating transfer entropy by the visitation frequency estimator, is it the shadow attractor we are estimating the nearest neighbours of?
>
> > nearest neighbors er i CCM. Shadow attractor òg, men kun deterministiske system.

3. Binning the time series to a common time grid,

using `BinnedResampling`

- Note: When we create time series data form a synthetic system, we can assign the discrete values (observations) to a regular time grid. Empirical data, however, seldomly give observations on a regular time grid. To get empirical data records on a regular time grid it is necessary to a do binned resampling (like we do in NB1). To make the steps in this notebook as similar as possible to the steps in our analysis of empirical data, we will therefore also include binned resampling of the synthetic time series.

In [14]:

```
binsize = 5 # we choose the same (or larger) binsize for the binned resampling a
s the original resolution (tstep = 5)
n_draws = 1000 # How many times we resample within each bin (resampling from the
*probability distribution** of each uncertain index value)

#common_grid = 0+binsize/2 : binsize : 500-binsize/2
time_grid = 0 : binsize : 500

resampling_method = BinnedResampling(time_grid, n_draws)

X_binned = resample(X, resampling_method)
Y_binned = resample(Y, resampling_method)
```

Out[14]:

```
UncertainIndexValueDataset{UncertainIndexDataset,UncertainValueDatas
et} containing 100 uncertain values coupled with 100 uncertain indic
es
```

In [15]:

```
print("X indices are of type " , typeof(X.indices.indices), ".
X_binned indices are of type " , typeof(X_binned.indices.indices), ".")
```

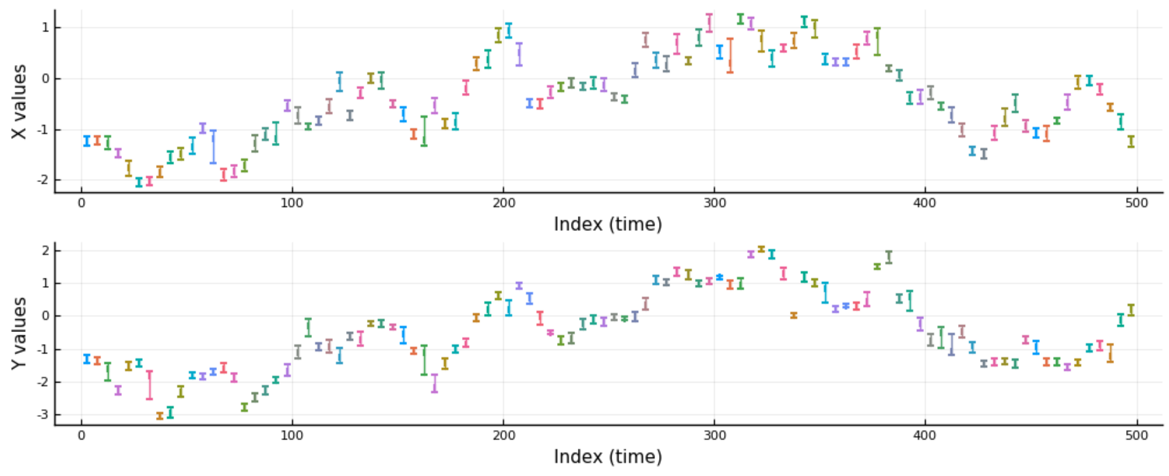
```
X indices are of type Array{UncertainScalarNormallyDistributed{Conti
nuous,Int64,Float64},1}.
X_binned indices are of type Array{CertainValue{Float64},1}.
```

We see that through the BinnedResampling, we have binned the values to a regular time grid, shuffling the uncertainty in the 1st dimension (time, in our case) over to uncertainty in the second dimension (value).

In [16]:

```
# Plot the time series of X and Y

#qs = [0.26,0.84] # ± 1σ = 68 % confidence interval = [26th quantile, 84th quant
ile]
qs = [0.025, 0.975] # ± 2σ = 95$ confidence interval = [2.5th quantile, 97.5th q
uantile]

# plot X_binned
plot_X_binned = plot(X_binned, ylabel = "X values",xlabel = "Index (time)", #col
or = :black,
    #ribbon = (qs, qs)
    ) # quantiles for the 95% CI (x-axis, y-axis)

# plot Y_binned
plot_Y_binned = plot(Y_binned, xlabel = "Index (time)", ylabel = "Y values", #co
lor = :red,
    #qs, qs # If we leave out
    )

# subplot of X and Y
plot(plot_X_binned,
    plot_Y_binned,
    size = (1000,400),
    layout = grid(2,1),
    )
```

Out[16]:



In [17]:

```
length(X_binned) #100
```

Out[17]:

```
100
```

In [18]:

```
binmidpoints = [X_binned.indices[i].value for i in 1:length(X_binned.indices)]
```

Out[18]:

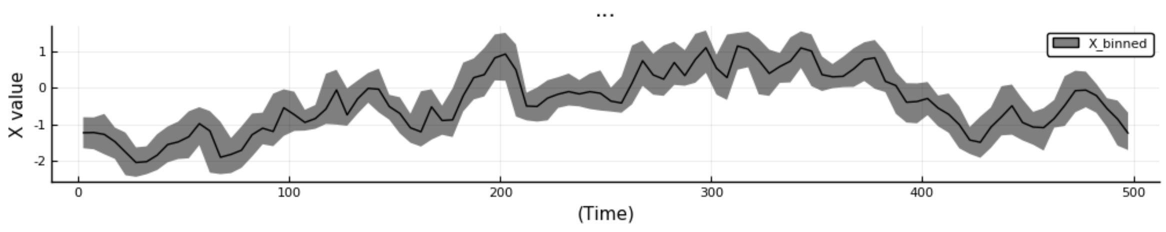
```
100-element Array{Float64,1}:
   2.5
   7.5
  12.5
  17.5
  22.5
  27.5
  32.5
  37.5
  42.5
  47.5
  52.5
  57.5
  62.5
   ⋮
 442.5
 447.5
 452.5
 457.5
 462.5
 467.5
 472.5
 477.5
 482.5
 487.5
 492.5
 497.5
```

> THIS IS WHY I WANTED TO `common_grid = tmin-binsize/2 : tmax+binsize/2`

In [19]:

```
# Plot on the common time grid, showing the 95% confidence envelope

# defining the median in each bin, and the confidence interval we want to use (9
5%)
bin_median_X = quantile.(X_binned.values, 0.5)
#ok
bin_upper_X = quantile.(X_binned.values, 0.975) .- bin_median_X

        # SUPER WIERD:
        # .- bin_median_X gives MethodError: no method matching resample
                                #NOT WHAT I'M TRYING TO DO


bin_lower_X = bin_median_X .- quantile.(X_binned.values, 0.025)

binmidpoints = [X_binned.indices[i].value for i in 1:length(X_binned.indices)]



### Plot
plot_X_binned_ribbon = plot(title = "...",size = (1000, 200), xlabel = "(Time)",
ylabel = "X value")
plot!(binmidpoints, bin_median_X, ribbon = (bin_lower_X, bin_upper_X),label = "X
_binned", color = :black)
```

Out[19]:



Pearson Correlation

In [17]:

```
...
```

```
syntax: invalid identifier name "..."
```

We see there is a correlation between the records X and Y, but can we infer the causal link? If so, what is the strength and directionality of the interaction?

Predictive asymmetry analysis

Now that we have created our time series data and binned them to a common time grid, we move on to showing how the analysis is done. These next steps (4-6) are synthesized in the function `function_from_XY_to_normPA` in NB3, and run under the hood in the NBRs.

5. Estimate predictive asymmetry from one parameter to the other

- Plot to show: EXPLAIN - difference between backwards and forwards in time predicition [NEED TO REVIEW THIS].

6. Normalize the predictive asymmetry results,

- In order to have a comparable scale of our results, we have to normalize the predictive asymmetry results.
- The significance treshold *f = 1* has been heuristically determined in a series [100? 1000?] of synthetic systems by Haaga et al. (2020).

The *normalized predictive asymmetry* is the result of interest in the analysis of our empirical data, since it allows us to make conclusions based on the significance treshold. In NB3, we have written steps 3-5 into one single function (`function_from_XY_to_normPA`), to optimize the code of the NBRs. This function allows us to compute the normalized predictive asymmetry directly from one time series to the other in one single operation.

)

4. Estimate transfer entropy from one time series to the other.

- use visitation frequency to estimate transfer entropy
- make a plot showing the confidence interval, if we have a component of randomness in our system (which we do, to aliken the system to complex natural systems).
- We will see that transfer entropy does not give unambiguous indications of causal connectivity/directionality. That is why the predictive asymmetry method has been developed.

4.1 - Estimate transfer entropy using the visitation frequency estimator

> What transfer entropy is
>
> Estimators of Transfer entropy - we will use the visitation frequency estimator.
>
> - include something of discretization of the shadow attractor?

In [46]:

```
# Defining which test will be used to estimate Transfer Entropy (VisitationFrequ
encyTest)

# Define the parameters of the visitation frequency test
binning = RectangularBinning(4) # dictates how the delay embedding is discretize
d (in other words, the "size" of bins in histogram of visitation frequency)
ηmax = 20                       # Defining the number of prediction lags η
#ηs = -ηmax : ηmax              # prediction lags backwards and forwards in time
ηs = -20 : ηmax                 # prediction lags backwards and forwards in time

# Define the visitation frequency test to estimate transfer entropy
TE_test = VisitationFrequencyTest(binning = binning, ηs = ηs)



# The ``causality()`` function calls to run ``TE-test`` from X to Y
TE_XtoY = causality(X, Y, TE_test)
# ...and from Y to X
TE_YtoX = causality(Y, X, TE_test)

# Output is ηs arrays of TE-values, one for each prediction lag η
```

Out[46]:

```
41-element Array{Float64,1}:
 0.2802904734558753
 0.3074418985230958
 0.3504206862650161
 0.35037018094757055
 0.3040728175546228
 0.26848024436856477
 0.33198453883163204
 0.3033146819556043
 0.3202855472348496
 0.3654069921498202
 0.33046255575013195
 0.32221991132928895
 0.358505996482946
 ⋮
 0.11946012251143578
 0.10463243134780509
 0.2624419505956359
 0.3173653408738142
 0.18098009323448228
 0.11458203550178059
 0.12652921409452667
 0.22264624469089522
 0.12889935494153182
 0.172128926088563
 0.13208778318411873
 0.16093220707582923
```

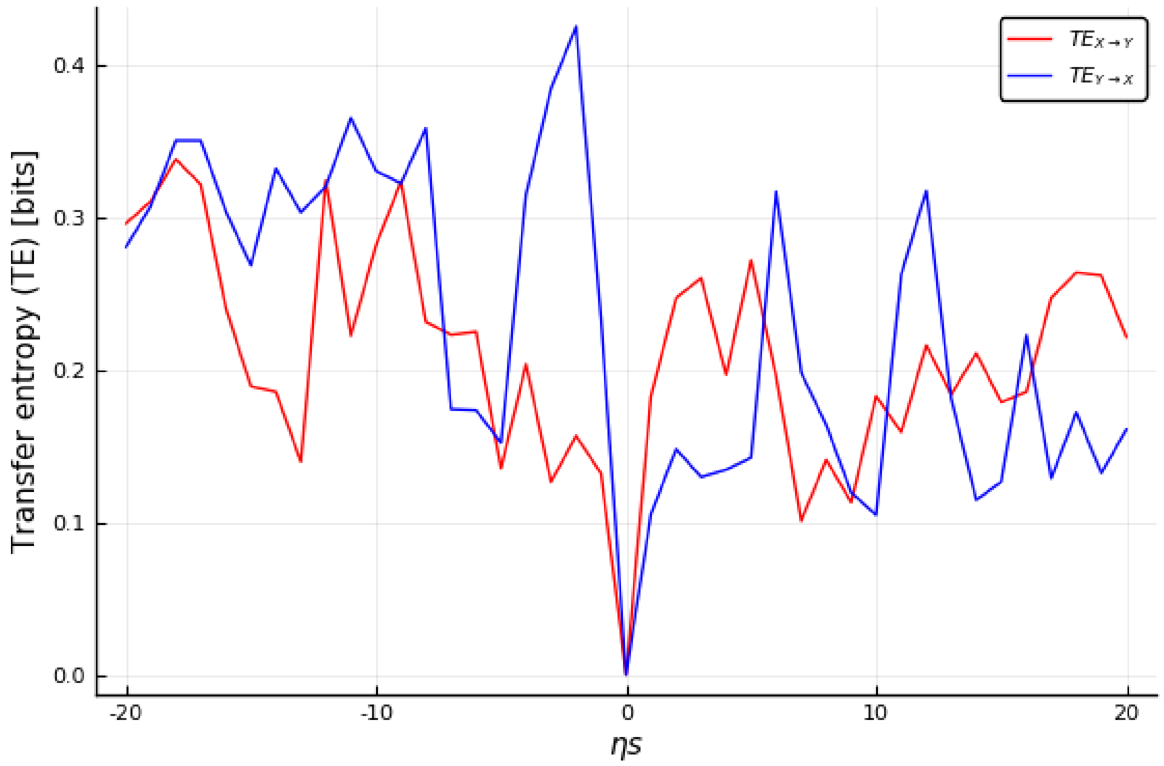
The output is an array of transfer entropy-values, with one value for each prediction lag η

The array of transfer entropy-results is 41 elements long. This is because we estimate transfer entropy for 20 time steps forwards and 20 time steps backwards, plus 0 (naturally, a time step of 0 has no change in transfer entropy, as there is no time to change over). It is the difference between the forwards-in-time and backwards-in-time entropy trends that will define our predictive asymmetry statistic.

In [70]:

```
TE_1 =
plot(ηs, TE_XtoY, xlabel = L"ηs", ylabel = "Transfer entropy (TE) [bits]", label
= L"TE_{X \rightarrow Y}", color = :red)
plot!(ηs, TE_YtoX, xlabel = L"ηs", ylabel = "Transfer entropy (TE) [bits]", labe
l = L"TE_{Y \rightarrow X}", color = :blue)
```

Out[70]:



Note that the arrays of transfer entropy changes with every iteration of the cell above. This is a manifestation of the uncertainty in the dataset (uivD). Let's have a look at , say, 10 different TE-arrays, to get an idea of the extent to which they differ.

In [27]:

```
nreps = 10 # let's plot 10 TE-arrays
TE_results_XtoY = zeros(nreps, length(ηs))
TE_results_YtoX = zeros(nreps, length(ηs))


for i in 1:nreps    # ten iterations of...
    TE_results_XtoY[i, :] = causality(X,Y, TE_test) # ... transfer entropy from
 X to Y
    TE_results_YtoX[i, :] = causality(Y,X, TE_test) # ... transfer entropy from
 Y to X
end

size(TE_results_XtoY) # Dimensions is a 10 * 41 array
print( "Every row is an array of TE-results. ", size(TE_results_XtoY)[2], " elem
ents long array, showing transfer entropy at different timelags η (", ηmax, " ba
ckwards in time and ", ηmax, " forwards in time, plus 0).
")
print("Each of the ", size(TE_results_XtoY)[1], " rows is an iteration of the TE
_test. The transfer entropy estimations vary from iteration to iteration due to
 the uncertainty in the time series data.
")

TE_results_XtoY # Output is 10 arrays of TE-values (each row is an iteration of
 the TE_test)
```

```
Every row is an array of TE-results. 41 elements long array, showing
transfer entropy at different timelags η (20 backwards in time and 20
forwards in time, plus 0).
Each of the 10 rows is an iteration of the TE_test. The transfer ent
ropy estimations vary from iteration to iteration due to the uncerta
inty in the time series data.
```

Out[27]:

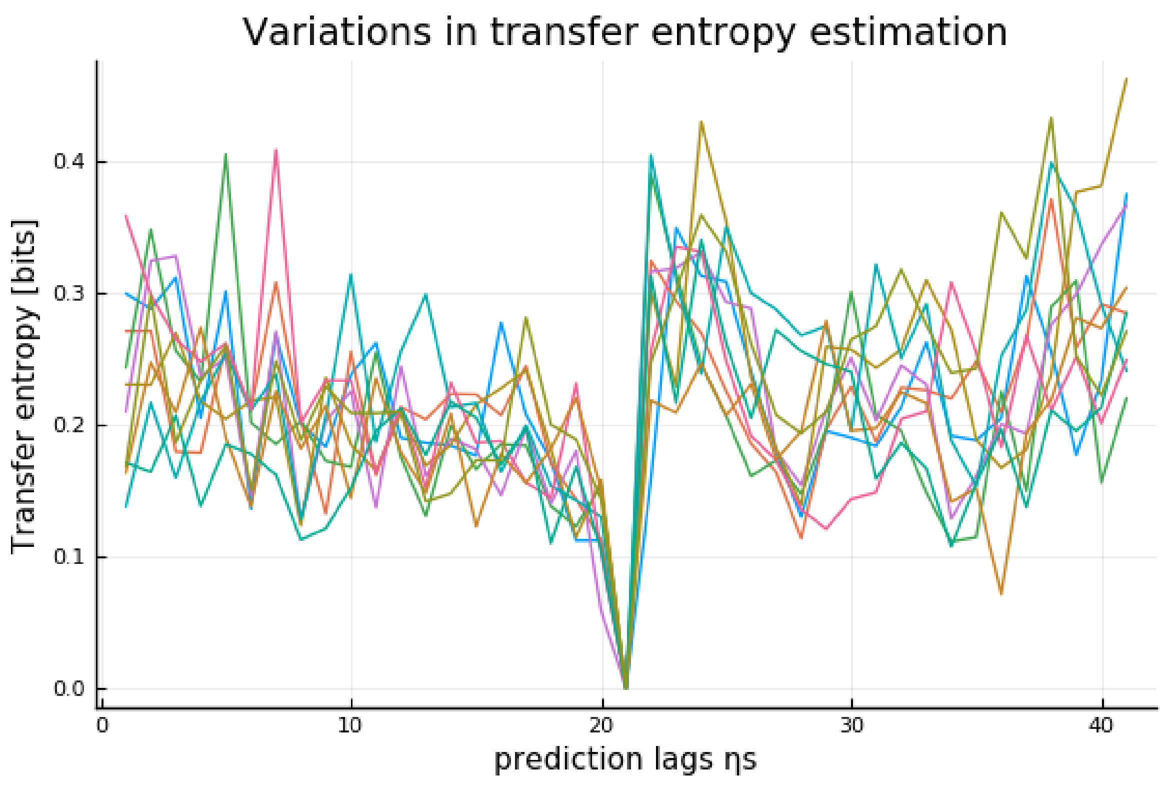
```
10×41 Array{Float64,2}:
 0.299187  0.287496  0.311099  0.204875  …  0.177006  0.23338   0.37
4695
 0.270778  0.2709    0.179386  0.178596     0.258194  0.291077  0.28
4328
 0.242901  0.347565  0.255712  0.232644     0.308818  0.15592   0.21
9894
 0.209449  0.323874  0.327625  0.236479     0.29851   0.336142  0.36
6023
 0.230148  0.229954  0.26937   0.217672     0.375944  0.380617  0.46
1554
 0.137556  0.216479  0.159593  0.217124  …  0.361788  0.291013  0.24
0348
 0.357886  0.298402  0.264678  0.247593     0.251261  0.200739  0.24
9016
 0.163215  0.247006  0.209159  0.273347     0.280691  0.272744  0.30
3813
 0.171189  0.164039  0.206639  0.138221     0.19505   0.213055  0.28
4795
 0.167832  0.297837  0.186314  0.232993     0.251788  0.2217    0.27
0683
```

Let's plot the different arrays of TE-results to get an idea of the extent to which they differ

In [29]:

```
te_results_iterations = plot(title = "Variations in transfer entropy estimation"
, xlabel = "prediction lags ηs", ylabel = "Transfer entropy [bits]") # assigning
a plot object; here we plot the first array of te-results
for i in 1: 10 # for each iteration of TE-results arrays
    plot!(TE_results_XtoY[i, :], label = "")
                                        #"iteration # $i ") # add to plot object: plot all colum
n values in each of the rows
end

te_results_iterations # call plot object to display it
```

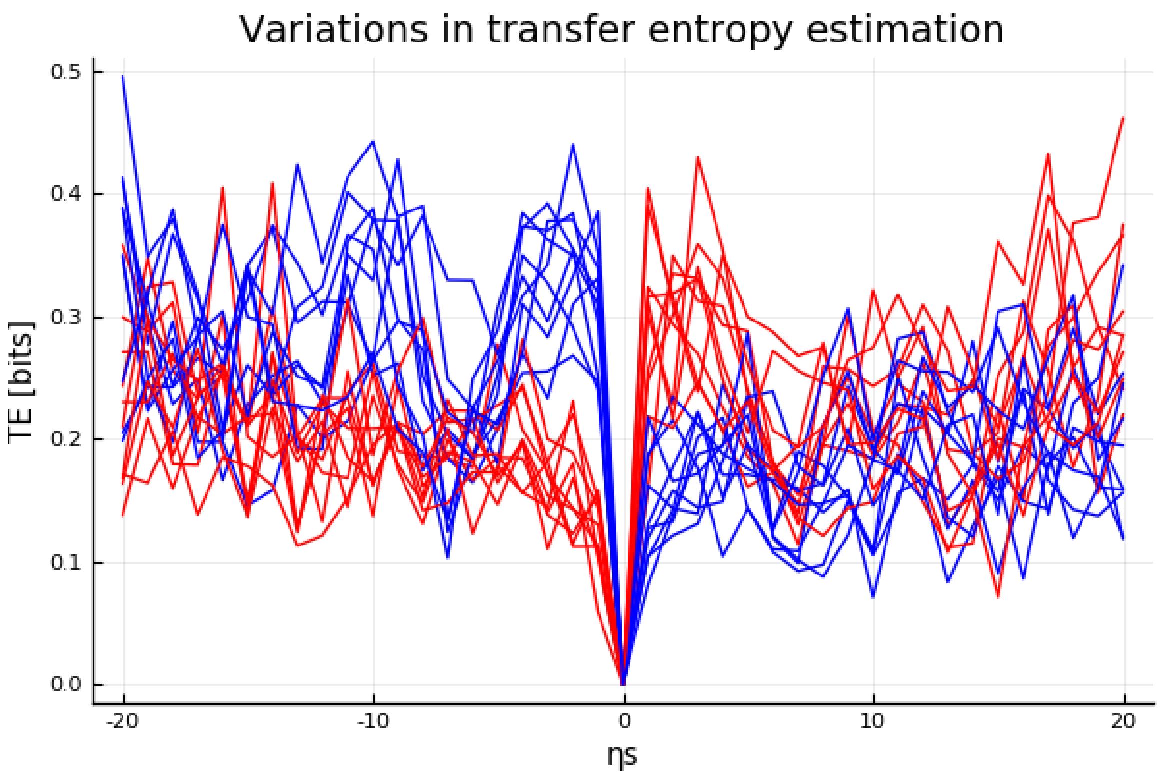
Out[29]:



In [91]:

```
te_results_10i_bidir = plot(title = "Variations in transfer entropy estimation",
    xlabel = "ηs",
    ylabel = "TE [bits]",
    legend = :topright #legend = false
    )
    plot!(color = :green, label = L"X $\rightarrow$ Y" )
    plot!(color = :pink, label =  L"Y $\rightarrow$ X" )
for i in 1: nreps # for each iteration of TE-results arrays
    plot!(ηs,TE_results_XtoY[i, :], color = :red, label = "")#L"X $\rightarrow$
 Y" )
    plot!(ηs,TE_results_YtoX[i, :], color = :blue, label = "" )# L"Y $\rightarro
w$ X" )
end

te_results_10i_bidir # call plot object to display it
```

Out[91]:



Above: **10 iterations of transfer entropy estimation.** The varition in estimation of transfer entropy stems from the stochastic component of the system, inferred by the uncertainty in the time series. Each iteration shows a series of transfer entropy values over 20 prediction lags (ηs) back and forth in time.

5) Use TE as input for the `PredictiveAsymmetryTest()`

Mandatory keywords

- (predictive_test = ''TE_test'')
- in TE_test: ηs = Int, symmetric around zero

> WHAT HAPPENED TO THIS? I've done without the mandatory keyword.
>
> from *earthsystemevolution.com/project/causalitytools/* : Note that `predictive_test` is a *mandatory* keyword. PredictiveAsymmetryTest(predictive_test = test_visitfreq)

In [49]:

```
# Defining our Predictive Asymmetry-test (PA_test) to use TE values gathered fro
m the test above.
PA_test = PredictiveAsymmetryTest(TE_test);

# Run the PA_test from one time series to the other
PA_XtoY = causality(X, Y, PA_test) # Check it there is prediction from X to Y
PA_YtoX = causality(Y, X, PA_test) # Check if there is prediction from Y to X

# Outputs are arrays of 20 TE-values. (???)
# Due to the uncertainty in the time series, these will change with every iterat
ion)
```

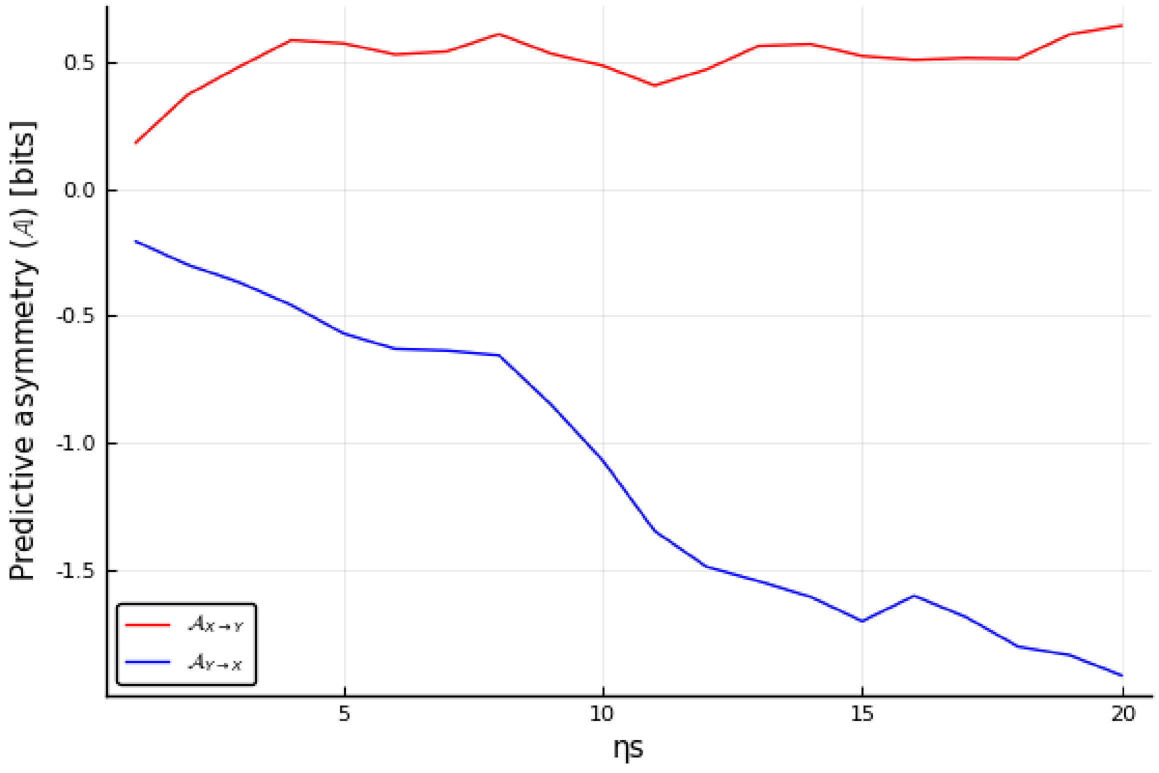
Out[49]:

```
20-element Array{Float64,1}:
 -0.20551385309925418
 -0.2979605856870844
 -0.3676451967779606
 -0.4570914796875494
 -0.5686069851808844
 -0.6297626177733369
 -0.6363513669364345
 -0.654932843799994
 -0.8483647277891944
 -1.0694245112491103
 -1.347739483122381
 -1.4886253345369456
 -1.5454375969118104
 -1.6068166807967241
 -1.7030943643377707
 -1.6030784672843064
 -1.6873069389154267
 -1.8031364313906915
 -1.8374214101917907
 -1.9173152385819074
```

In [74]:

```
# plot results (1 iteration) of predictive asymmetry

PA_1 = plot(title = "",#"Predictive asymmetry between X and Y",
    xlabel = "ηs",
    ylabel = string("Predictive asymmetry ", L"(\mathbb{A})", " [bits]" ))
# from X to Y
plot!(PA_XtoY, color = "red", label = L"\mathcal{A}_{X \rightarrow Y}")
# from Y to X
plot!(PA_YtoX, color = "blue", label = L"\mathcal{A}_{Y \rightarrow X}")
```

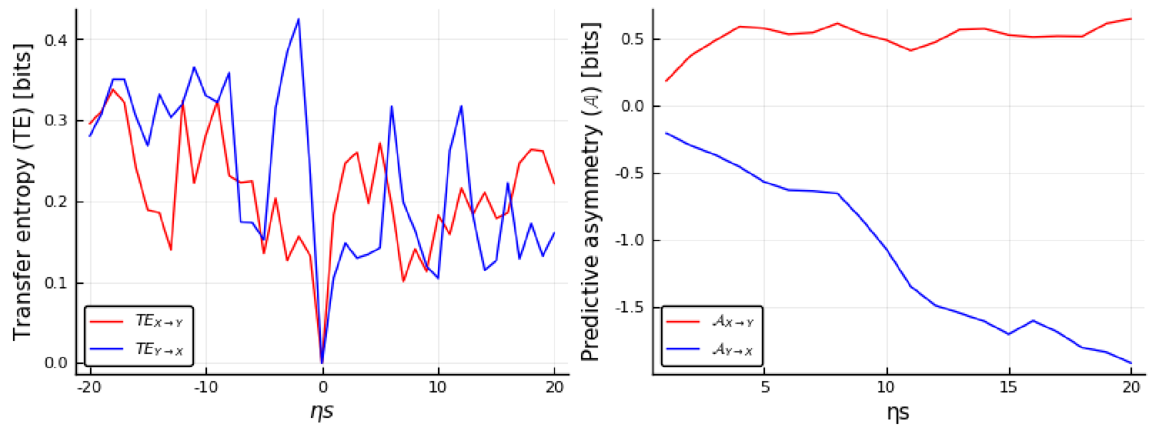
Out[74]:



The `PredictiveAsymmetry` is a numerical estimation of causal connections. It works by calculating the prediction skill from one time series to the other

In [92]:

```
plot(TE_1, PA_1, size = (800,300), legend = :bottomleft)
savefig("../../MASTER_2.0/figurar/4_metode/TE_PA_1")
```

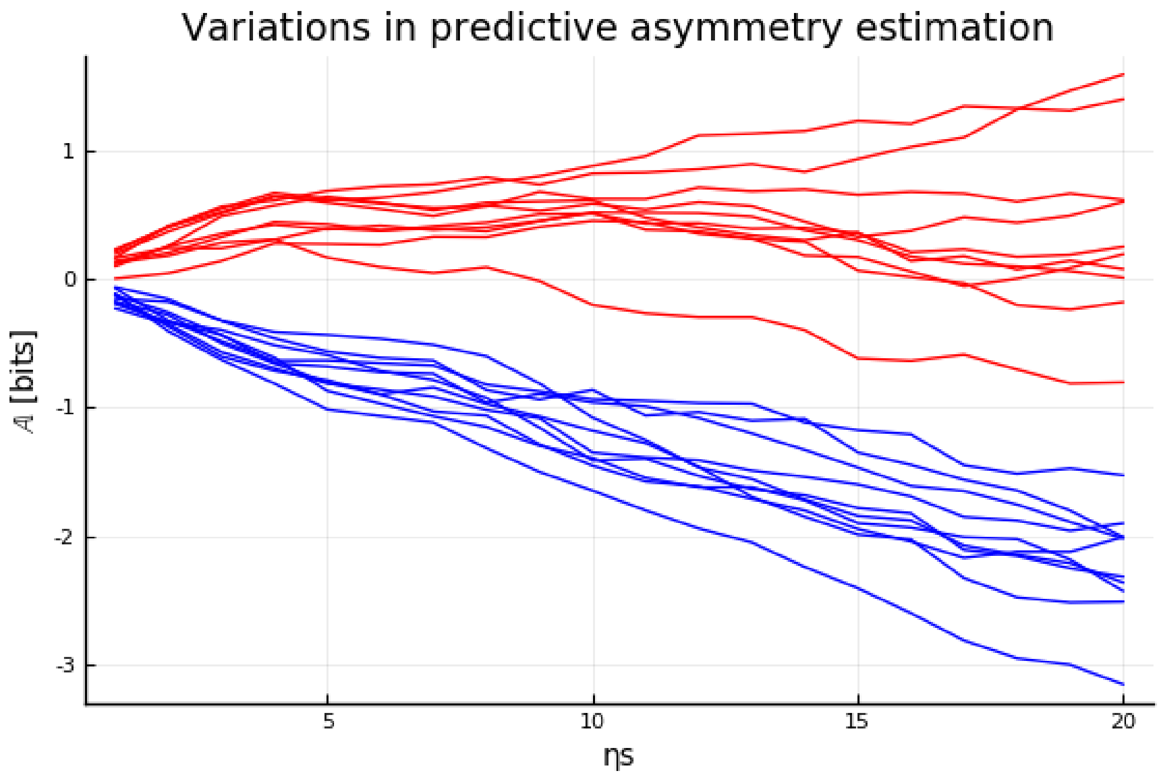


As with transfer entropy, the predictive asymmetry results will vary with each iteration, because of the stochastic component in our system s. Lets look at 10 iterations of the PA test, to see to which extent the results vary.

In [34]:

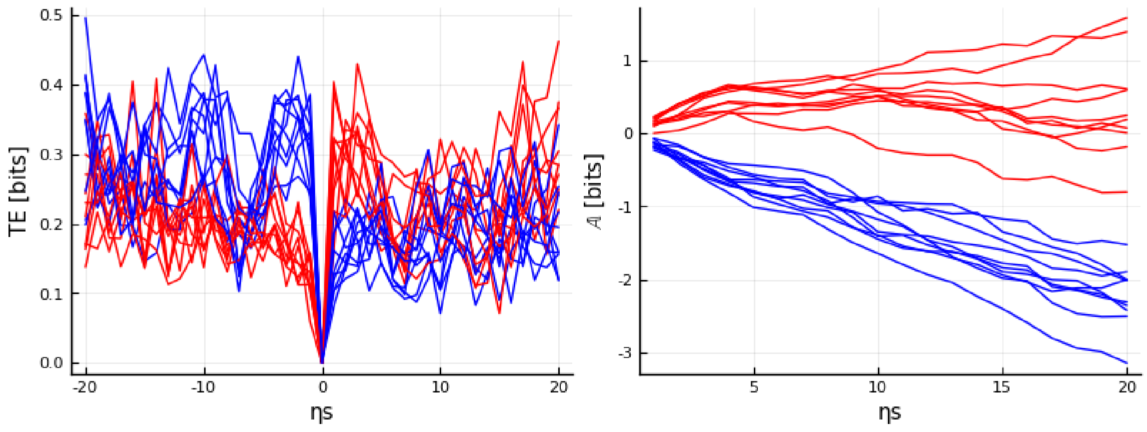
```
# Lets look at 10 iterations of the PA test, to see to which extent the results
 vary.

nreps = 10 # let's run 10 iterations of PA_test
pa_length = ηmax # Each array of PA-results will have 41 elements (± ηmax, plus η
= 0)
PA_results_XtoY = zeros(nreps, pa_length) # create an empty matrix that we can f
ill in the for-loop below
PA_results_YtoX = zeros(nreps, pa_length) # create an empty matrix that we can f
ill in the for-loop below

for i in 1:nreps
    PA_results_XtoY[i, :] = causality(X,Y, PA_test)
    PA_results_YtoX[i, :] = causality(Y,X, PA_test)
end

pa_results_iterations = plot(title = "Variations in predictive asymmetry estimat
ion", xlabel = "ηs", ylabel = L"$\mathbb{A}$ [bits]") # assigning a plot object;
here we plot the first array of pa-results
for i in 1:nreps # for each iteration of pa-results arrays
    plot!(PA_results_XtoY[i, :], color = :red, label = "X --> Y" ) # add to plot
object: for each row, plot all column values
    plot!(PA_results_YtoX[i, :], color = :blue, label = "Y --> X", legend = fals
e )
end

pa_results_iterations # call plot object to display it
```

Out[34]:



In [93]:

```
plot(te_results_10i_bidir, pa_results_iterations, title = "", size = (800,300))
savefig("../../MASTER_2.0/figurar/4_metode/TE_PA_10iterations")
```



This is why we must do many iterations- are we? Random sequences test.

Visualizing confidence

How confident are we in this result? To get a **confidence envelope** on our results, we use a `RandomSequencesTest` . This applies our causality test to multiple independent draws of the datasets, where each draw from a randomly selected consecutive chunk of points.

In [35]:

```
# Defining the random sequences tests

# Let's make n random sequences (chunks) to run the PA-test on
n_chunks = 150     # let's make 150 chunks/sequences/windows

# each chunk /sequence/window should be a bit shorter than the original time ser
ies length, we'll use 70% of full time series length
N = length(X) # NB ON HOW YOU DEFINE N. should be fine?
min_chunk = 0.70*N
max_chunk = 0.95*N

chunks = RandomSequences(n_chunks, min_chunk : max_chunk) # the ensemble of 150
 random sequences

# 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
```

Out[35]:

```
RandomSequencesTest{PredictiveAsymmetryTest{VisitationFrequencyTes
t},RandomSequences}(PredictiveAsymmetryTest{VisitationFrequencyTest}
(predictive_test = VisitationFrequencyTest(k = 1, l = 1, m = 1, n =
1, τ = 1, b = 2, binning_summary_statistic = mean, estimator = Visita
tionFrequency(), binning = RectangularBinning(4), ηs = -20:20)), Rand
omSequences(150, 70.0:1.0:95.0))
```

In [36]:

```
chunks
```

Out[36]:

```
RandomSequences(150, 70.0:1.0:95.0)
```

In [37]:

```
# running the random sequences (rs) test for TE and PA

# ...from X to Y
rsTE_XtoY = causality(X, Y, rsTE_test) # this gives us a family of Transfer Entr
opies (150 values) from time series X to time series Y
rsPA_XtoY = causality(X, Y, rsPA_test) # this gives us a family of Predictive As
ymmetries (150 values)

# ...and from Y to X
rsTE_YtoX = causality(Y, X, rsTE_test)
rsPA_YtoX = causality(Y, X, rsPA_test);
```

```
ArgumentError: `resampling.sequence_length`must be an integer or a c
ollection of integers

Stacktrace:
 [1] causality(::UncertainIndexValueDataset{UncertainIndexDataset,Un
certainValueDataset}, ::UncertainIndexValueDataset{UncertainIndexDat
aset,UncertainValueDataset}, ::RandomSequencesTest{VisitationFrequen
cyTest,RandomSequences}) at /Users/maria/.julia/packages/CausalityTo
ols/MyU5d/src/causalitytests/highlevel_tests/causality_RandomSequenc
esTest.jl:98
 [2] top-level scope at In[37]:1
```

In [38]:

```
#length(chunks)
show(chunks)
```

```
RandomSequences(150, 70.0:1.0:95.0)
```

> HELP: Since Int:1.0:Int, then length(chunk) must be an Int. SO I DON'T GET THE ERROR MESSAGE ABOVE

—

In [39]:

```
# Note:
typeof(rsTE_XtoY) # Array{Array{Float64,1},1}
    # Note: the results are given in the form array of arrays with dimensions ηm
ax by nchunks (20 rows, 150 columns)
    # This is the format in which the object will be sent to normalization funct
ion
```

```
UndefVarError: rsTE_XtoY not defined

Stacktrace:
 [1] top-level scope at In[39]:1
```

Compute median and quantiles for a 95% confidence interval

In [32]:

```
#(delete this cell and keep cells with plots for this notebook)

#= But first, in order to calculate the confidence interval
we need to concatenate the arrays (required format input for the quantile functi
on) =#

#rsTE_XtoY_ = hcat(rsTE_XtoY...) # "unpacked", or concatenated to a single array
is denoted with the suffix _
#rsTE_YtoX_ = hcat(rsTE_YtoX...)
#rsPA_YtoX_ = hcat(rsPA_YtoX...)
#rsPA_XtoY_ = hcat(rsPA_XtoY...);
```

In [56]:

```
# IS THIS CELL NECESSARY?

    # In order to calculate the confidence interval, we need to concatenate the
 arrays (required format input for the quantile function)
    rsTE_XtoY_ = hcat(rsTE_XtoY...) # "unpacked", or concatenated to a single ar
ray is denoted with the suffix _
    rsTE_YtoX_ = hcat(rsTE_YtoX...)



# Compute median and quantiles for a 95% confidence interval of TE
# ... for TE from X to Y
TE_XtoY_median =[quantile(rsTE_XtoY_[i,:], 0.5) for i in 1:ηmax] # median
TE_XtoY_upper = [quantile(rsTE_XtoY_[i,:], 0.975) for i in 1:ηmax] .- TE_XtoY_me
dian # upper quantile
TE_XtoY_lower = TE_XtoY_median .- [quantile(rsTE_XtoY_[i,:], 0.025) for i in 1:η
max] # lower quantile;

# ... for TE from Y to X
TE_YtoX_median =[quantile(rsTE_YtoX_[i,:], 0.5) for i in 1:ηmax] # median
TE_YtoX_upper = [quantile(rsTE_YtoX_[i,:], 0.975) for i in 1:ηmax] .- TE_YtoX_me
dian  # upper quantile
TE_YtoX_lower = TE_YtoX_median .- [quantile(rsTE_YtoX_[i,:], 0.025) for i in 1:η
max] # lower quantile;


# Eg vil lage eit plot for å forstå. ER DET DETTE ME GJER?

plot(title = "Transfer Entropy with 95% confidence interval", # visualisation of
uncertainty range
    xlabel = "prediction lags (ηs)",
    ylabel = "Transfer Entropy",
    size = (800,200))
plot!(ηs, # OR ηmax?
    TE_XtoY_median,
    ribbon = (TE_XtoY_lower, TE_XtoY_upper),
    label = "TE from X to Y",
    color = :red,
    fillalpha = 0.3)
plot!(ηs,
    TE_YtoX_median,
    ribbon = (TE_YtoX_lower, TE_YtoX_upper),
    label = "TE from Y to X",
    color = :blue,
    fillalpha = 0.3)


# CYCLICITY IN TE ?
# burde det ikkje heller vere speilvending rundt 0?
# or should I have plotted only ηmax on the x-axis?
#Hjelp, forstår ikkje
```

```
UndefVarError: rsTE_XtoY not defined

Stacktrace:
 [1] top-level scope at In[56]:1
```

As we can see in the plot above, plotting transfer entropy does not give unambiguous information about cause and effect. That is why the Predictive asymmetry methd has been developed. Let's see how it represents graphically.

In [ ]:

```

```

Compute median and quantiles for a 95% confidence interval of Predictive Asymmetry

In [57]:

```
# Computing median and quantiles for a 95% CI on Predictive Asymmetry

    # In order to calculate the confidence interval, we need to concatenate the
 arrays
    #(required format input for the quantile function)
    rsPA_XtoY_ = hcat(rsPA_XtoY...)# "unpacked", or concatenated to a single arr
ay is denoted with the suffix _
    rsPA_YtoX_ = hcat(rsPA_YtoX...);

# ... for PA from X to Y
PA_XtoY_median = [quantile(rsPA_XtoY_[i,:], 0.5) for i in 1:ηmax] # median
PA_XtoY_upper = [quantile(rsPA_XtoY_[i,:], 0.975) for i in 1:ηmax] .- PA_XtoY_me
dian # upper quantile
PA_XtoY_lower = PA_XtoY_median .- [quantile(rsPA_XtoY_[i,:], 0.025) for i in 1:η
max] # lower quantile;

# ... for PA from Y to X
PA_YtoX_median = [quantile(rsPA_YtoX_[i,:], 0.5) for i in 1:ηmax] # median
PA_YtoX_upper = [quantile(rsPA_YtoX_[i,:], 0.975) for i in 1:ηmax] .- PA_YtoX_me
dian  # upper quantile
PA_YtoX_lower = PA_YtoX_median .- [quantile(rsPA_YtoX_[i,:], 0.025) for i in 1:η
max] # lower quantile;



# Plot PA with the 95% confidence interval

plot_PA_95CI =
plot(title = string(L"$\mathcalbb{A}", " (predictive asymmetry) between X and Y"
),
    xlabel = "ηs (prediction lags)",
    ylabel = string(L"$mathbb{A}", "[bits]"),
    size = (1000,200))
plot!(#ηs,
    PA_XtoY_median,
    label = "from X to Y",
    color = "red",
    ribbon = (PA_XtoY_upper, PA_XtoY_lower),
    fillalpha = 0.3)
plot!(#ηs,
    PA_YtoX_median,
    label = "from Y to X",
    color = "blue",
    ribbon = (PA_YtoX_upper, PA_YtoX_lower),
    fillalpha = 0.3)
```

```
UndefVarError: rsPA_XtoY not defined

Stacktrace:
 [1] top-level scope at In[57]:1
```

Next, we normalize the data, to define a **significance level**

In [58]:

```
# We use the following function to normalize the PA-results

function normalized_mean_predictive_asymmetry(  # input arguments:
            ηmax::Int,                          # number of prediction lags
            te_test_result::AbstractArray,      # random sequences TE-results (bef
ore concatenated)
            pa_test_result::AbstractArray;      # random sequences PA-results (bef
ore concatenated)
            f::Number = 1.0                     # WHAT IS f? Significance level?
            #if nothing else is stated, f=1.0 by default (arguments presented af
ter ; = default arguments)
            )
    TE = hcat(te_test_result...)
    PA = hcat(pa_test_result...)
    PA_hat = vcat([mean(PA[1:i, :] ./ (mean(TE[ηmax+1-i:ηmax+i, :]) * f), dims =
1) for i in 1:ηmax]...)
    return PA_hat                               # WHAT IS PA-hat? Normalized mean
 PA?
end
```

Out[58]:

```
normalized_mean_predictive_asymmetry (generic function with 1 metho
d)
```

In [59]:

```
# Results from normalized mean:

# (NB. Works only BEFORE we concatenate rs-results)
PA_hat_XtoY = normalized_mean_predictive_asymmetry(ηmax, rsTE_XtoY, rsPA_XtoY) #
inputs: rsTE of rsPA
PA_hat_YtoX = normalized_mean_predictive_asymmetry(ηmax, rsTE_YtoX, rsPA_YtoX);
```

```
UndefVarError: rsTE_XtoY not defined

Stacktrace:
 [1] top-level scope at In[59]:1
```

In [60]:

```
# compute the 95% CI quantiles for PA_hat
# BEFORE CONCATENATING

# ... for PA from X to Y
PA_hat_XtoY_median = [quantile(PA_hat_XtoY[i,:], 0.5) for i in 1:ηmax] # median
PA_hat_XtoY_upper = [quantile(PA_hat_XtoY[i,:], 0.975) for i in 1:ηmax] .- PA_ha
t_XtoY_median # upper quantile
PA_hat_XtoY_lower = PA_hat_XtoY_median .- [quantile(PA_hat_XtoY[i,:], 0.025) for
i in 1:ηmax] # lower quantile;

# ... for PA from Y to X
PA_hat_YtoX_median = [quantile(PA_hat_YtoX[i,:], 0.5) for i in 1:ηmax] # median
PA_hat_YtoX_upper = [quantile(PA_hat_YtoX[i,:], 0.975) for i in 1:ηmax] .- PA_ha
t_YtoX_median  # upper quantile
PA_hat_YtoX_lower = PA_hat_YtoX_median .- [quantile(PA_hat_YtoX[i,:], 0.025) for
i in 1:ηmax] # lower quantile;
```

```
UndefVarError: PA_hat_XtoY not defined

Stacktrace:
 [1] (::getfield(Main, Symbol("##26#27")))(::Int64) at ./none:0
 [2] iterate at ./generator.jl:47 [inlined]
 [3] collect(::Base.Generator{UnitRange{Int64},getfield(Main, Symbol
("##26#27"))}) at ./array.jl:606
 [4] top-level scope at In[60]:1
```

In [61]:

```
# bla trash
# this is what we get if we compute the quantiles  after concatenating
    rsTE_XtoY_ = hcat(rsTE_XtoY...)
    rsTE_YtoX_ = hcat(rsTE_YtoX...);

    rsPA_XtoY_ = hcat(rsPA_XtoY...)
    rsPA_YtoX_ = hcat(rsPA_YtoX...);

PA_hat_XtoY_ = hcat(PA_hat_XtoY...)
PA_hat_YtoX_ = hcat(PA_hat_YtoX...)


# ... for PA from X to Y
PA_hat_XtoY_median = [quantile(PA_hat_XtoY_[i,:], 0.5) for i in 1:ηmax] # median
PA_hat_XtoY_upper = [quantile(PA_hat_XtoY_[i,:], 0.975) for i in 1:ηmax] .- PA_h
at_XtoY_median # upper quantile
PA_hat_XtoY_lower = PA_hat_XtoY_median .- [quantile(PA_hat_XtoY_[i,:], 0.025) fo
r i in 1:ηmax] # lower quantile;

# ... for PA from Y to X
PA_hat_YtoX_median = [quantile(PA_hat_YtoX_[i,:], 0.5) for i in 1:ηmax] # median
PA_hat_YtoX_upper = [quantile(PA_hat_YtoX_[i,:], 0.975) for i in 1:ηmax] .- PA_h
at_YtoX_median  # upper quantile
PA_hat_YtoX_lower = PA_hat_YtoX_median .- [quantile(PA_hat_YtoX_[i,:], 0.025) fo
r i in 1:ηmax] # lower quantile;
```

```
UndefVarError: rsTE_XtoY not defined

Stacktrace:
 [1] top-level scope at In[61]:1
```

In [62]:

```
# and plot PA_hat with 95% confidence envelope

plot_PA_hat_95CI =
plot(title = string(L"$\mathcal{A}", " (normalized predictive asymmetry) between
X and Y"),
    xlabel = "ηs (prediction lags)", # CORRECT on the x-axis? I'm not explicitel
y plotting anything.
    ylabel = L"$\mathcal{A}$" #"PA_hat",
    size = (800,400),
    hline([1], line = (:dash, :black)),
    label = "significance level",
    xlim = (0,ηmax))    # If I understood the function right, significance level
is f = 1 by default?
plot!(1:ηmax,
    PA_hat_XtoY_median,
    ribbon = (PA_hat_XtoY_upper, PA_hat_XtoY_lower),
    label = "from X to Y",
    color = "red")
plot!(1:ηmax,
    PA_hat_YtoX_median,
    ribbon = (PA_hat_YtoX_upper, PA_hat_YtoX_lower),
    label = "from X to Y",
    color = "blue")
```

```
syntax: missing comma or ) in argument list
```

# bla # DON'T concatenate before plotting PA_hat_XtoY_median_ = hcat(PA_hat_XtoY_median...) PA_hat_XtoY_upper_ = hcat(PA_hat_XtoY_upper...) PA_hat_XtoY_lower_ = hcat(PA_hat_XtoY_lower...) PA_hat_YtoX_median_ = hcat(PA_hat_YtoX_median...) PA_hat_YtoX_upper_ = hcat(PA_hat_YtoX_upper...) PA_hat_YtoX_lower_ = hcat(PA_hat_YtoX_lower...) # this is what will happen: plot(title = "Normalized mean PA between pCO2 (Bereiter) and GSL (SprattLisiecki)", xlabel = "ηs (prediction lags)", # CORRECT on the x-axis? I'm not explicitely plotting anything. ylabel = "PA_hat", # Should we write normalized PA on y-axis, or how can I write the fancy A? size = (800,400), hline([1,-1], line = (:dash, :black)), label = "significance level") # If I understood the function right, significance level is f = 1 by default? plot!(#ηs, PA_hat_XtoY_median_, ribbon = (PA_hat_XtoY_upper_, PA_hat_XtoY_lower_), label = "from CO2 to GSL", color = "red") plot!(#ηs, PA_hat_YtoX_median_, ribbon = (PA_hat_YtoX_upper_, PA_hat_YtoX_lower_), label = "from GSL to CO2", color = "blue")

In this notebook we have followed and visualized the steps in the analysis in detail. In the next notebook (notebook 4), we will concentrate all the steps of the analysis in a function, so that we can more efficiently run the analysis for all our time series.

"**We find that** the predictive asymmetry $\mathbb{A}$ in the causal direction (X → Y) is positive, whereas the asymmetry in the non-causal direction (Y → X) is negative. We thus recover the expected unidirectional causal relationship X → Y .

Note that because we defined our time series as UncertainIndexValueDataset, the causality() test function returns a distribution of predictive asymmetry values for each prediction lag." (???)

```
In [ ]:
```