NM-DDA Software

Claudia Lainscsek

This software reproduces the Rössler part of “Network-motif delay differential analysis of brain activity during seizures", Chaos 33(12):123136; 2023 [3]. The original software in the paper was written in Matlab. This has been changed to Julia (tested using version 1.10.2). The software runs on Linux and can be found on Github (`https://github.com/lclaudia/CD-DDA`).

1 Single Rössler system

Before introducing coupled Rössler systems the code to integrate a single system is presented. The equations for the Rössler system [5] are

$$\begin{aligned}\dot{u}_1 &= -u_2 - u_3 \\ \dot{u}_2 &= u_1 + a\,u_2 \\ \dot{u}_3 &= b - c\,u_3 + u_1 u_3\end{aligned} \tag{1}$$

with $a = 0.2$ and $c = 5.7$ and $\delta t = 0.05$. This system can be encoded as

system	equation #	variable		coefficients
$\dot{u}_1 = \color{red}{-u_2} - u_3$	0	0	2	-1
$\dot{u}_1 = -u_2 \color{red}{- u_3}$	0	0	3	-1
$\dot{u}_2 = \color{red}{u_1} + a\,u_2$	1	0	1	1
$\dot{u}_2 = u_1 + \color{red}{a\,u_2}$	1	0	2	a
$\dot{u}_3 = \color{red}{b} - c\,u_3 + u_1\,u_3$	2	0	0	b
$\dot{u}_3 = b \color{red}{- c\,u_3} + u_1\,u_3$	2	0	3	$-c$
$\dot{u}_3 = b - c\,u_3 + \color{red}{u_1\,u_3}$	2	1	3	1

Note, that the equation numbers are (0,1,2) for the three equations. This defines `DIM=3`. There are 2 “variable” columns which define the order of nonlinearity `ODEorder=2`. The numbers in the two columns are 1 for u_1, 2 for u_2, and 3 for u_3. A line with only zeros denotes a constant term. All other entries are filled with zeros.

This encoding can be used to numerically integrate the Rössler system. The plots are shown in Fig. 1.

```julia
include("DDAfunctions.jl");                                        # set of Julia functions

NrSyst=1;                                                          # 1 single system
ROS=[[0  0 2];                                                     # single Roessler system
     [0  0 3];
     [1  0 1];
     [1  0 2];
     [2  0 0];
     [2  0 3];
     [2  1 3]
    ];
 (MOD_nr,DIM,ODEorder,P) = make_MOD_nr(ROS,NrSyst);                 # encoding of the Roessler system
                                                                    # function defined in DDAfunctions.jl
a=.2; c=5.7;
dt=.05; X0=rand(DIM,1);                                             # choice of parameters
L=10000; TRANS=5000;                                                # integration length and transient

b=0.45;                                                             # chaotic attractor
MOD_par=[-1 -1 1 a b -c 1];                                         # parameters

X = integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,
                              L,DIM,ODEorder,X0,TRANS);             # integrate system
```

```
                                                                       # function defined in DDAfunctions.jl
plot(X[:,1],X[:,2],X[:,3],                                             # plot the attractor
     color=:blue,legend=false,
     xlabel=L"x",ylabel=L"y",zlabel=L"z")
plot!(size=(500,500))

display(current());
print("Make pdf file and continue? ");
readline()

savefig("Roessler_0.45.pdf")

b=1;                                                                   # periodic attractor
MOD_par=[-1 -1 1 a b -c 1];                                            # parameters
X = integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,
                              L,DIM,ODEorder,X0,TRANS);                # integrate system
                                                                       # function defined in DDAfunctions.jl
plot(X[:,1],X[:,2],X[:,3],                                             # plot the attractor
     color=:blue,legend=false,
     xlabel=L"x",ylabel=L"y",zlabel=L"z")
plot!(size=(500,500))

display(current());
print("Make pdf file and continue? ");
readline()

savefig("Roessler_1.pdf")
```

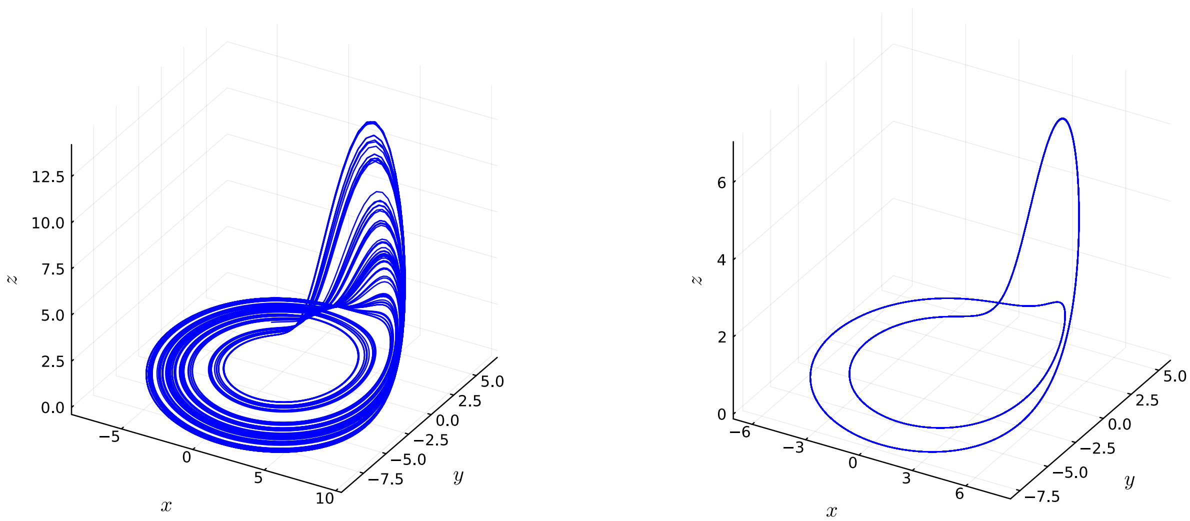

Figure 1: Rössler attractor with $b = 0.45$ (left) and $b = 0.1$ (right)

2 Coupled Rössler systems

We couple Rössler systems using diffusive coupling as introduced in Paluš and Vejmelka [4] and consider here seven (coupled) Rössler systems

$$\begin{aligned}
\dot{u}_{1,n} &= -u_{2,n} - u_{3,n} + \sum_j \epsilon(u_{1,n} - u_{1,j}) \\
\dot{u}_{2,n} &= u_{1,n} + a_n\, u_{2,n} \\
\dot{u}_{3,n} &= b_n + c_n u_{3,n} + u_{1,n} u_{3,n}
\end{aligned} \tag{2}$$

with $n = 1, 2, \ldots, 7$ and x_j is the u_1-component of another system j. The values for a_n, b_n, and c_n are listed in Tab. 1. ϵ is either 0 or 0.15 depending on which systems are coupled.

We have 7 three-dimensional systems and therefore 21 variables. In the code we want to number them $x_1, x_2, \ldots x_{21}$ and therefore need to make the following change in variables: $u_{1,n} \to x_{3n-2}$, $u_{2,n} \to x_{3n-1}$, $u_{3,n} \to x_{3n}$. In general, for a N-dimensional system we would have $u_{k,n} \to x_{Nn-(N-k)}$ This change of variables changes system (2) to

$$\begin{aligned}
\dot{x}_{3n-2} &= -x_{3n-1} - x_{3n} + \sum_j \epsilon(x_{3n-2} - x_{3j-2}) \\
\dot{x}_{3n-1} &= x_{3n-2} + a_n\, x_{3n-1} \\
\dot{x}_{3n} &= b_n + c_n x_{3n} + x_{3n-2} x_{3n}
\end{aligned} \tag{3}$$

In the code we first encode the 7 systems without the coupling part:

```
include("DDAfunctions.jl");                                    # set of Julia functions

NrSyst=7;                                                      # 7 coupled systems
ROS=[[0  0 2];                                                 # single Roessler system
     [0  0 3];
     [1  0 1];
     [1  0 2];
     [2  0 0];
     [2  0 3];
     [2  1 3]
    ];
(MOD_nr,DIM,ODEorder,P) = make_MOD_nr(ROS,NrSyst);             # encoding of the 7 Roessler systems
                                                               # function defined in DDAfunctions.jl
```

Table 1: Parameters of the seven Rössler systems

#	a_n	b_n	c_n
1	0.21	0.21505	-4.5
2	0.21	0.20201	-4.5
3	0.21	0.20411	-4.5
4	0.20	0.40503	-4.5
5	0.20	0.39905	-4.5
6	0.20	0.41000	-4.5
7	0.18	0.50000	-6.8

```
a123=0.21;                                                             # model parameters
a456=0.20;
a7  =0.18;
b1 = 0.2150;
b2 = 0.2020;
b3 = 0.2041;
b4 = 0.4050;
b5 = 0.3991;
b6 = 0.4100;
b7 = 0.5000;
c =5.7;
c7=6.8;
MOD_par=[
         -1 -1 1 a123  b1 -c  1
         -1 -1 1 a123  b2 -c  1
         -1 -1 1 a123  b3 -c  1
         -1 -1 1 a456  b4 -c  1
         -1 -1 1 a456  b5 -c  1
         -1 -1 1 a456  b6 -c  1
         -1 -1 1 a7    b7 -c7 1
        ];
MOD_par=reshape(MOD_par',size(ROS,1)*NrSyst)';
```

The numerical coupling experiment is done in three segments: (i) seven uncoupled systems, (ii) systems #(4,5,6)→#7 with $\epsilon = 0.15$, and (iii) #7→#(4,5,6) with $\epsilon = 0.15$.

The encoding for the couplings (ii) and (iii) are done in the following way:

	from			to		
case	j	Eq. #	variable	n	Eq. #	variable
(ii)	4	0	0 1	7	0	0 1
	5	0	0 1	7	0	0 1
	6	0	0 1	7	0	0 1
(iii)	7	0	0 1	4	0	0 1
	7	0	0 1	5	0	0 1
	7	0	0 1	6	0	0 1

```
FromTo2=[[4 0  0 1   7 0  0 1];                                        # from 4th system 1st Eq. variable 1
                                                                       #   to 7th system 1st Eq. variable 1
         [5 0  0 1   7 0  0 1];
         [6 0  0 1   7 0  0 1]];

FromTo3=[[7 0  0 1   4 0  0 1];
         [7 0  0 1   5 0  0 1];
         [7 0  0 1   6 0  0 1]];

I2=make_MOD_nr_Coupling(FromTo2,DIM,P);                                # MOD_nr part for coupling; case (ii)
I3=make_MOD_nr_Coupling(FromTo3,DIM,P);                                # MOD_nr part for coupling; case (iii)
                                                                       # function defined in DDAfunctions.jl

epsilon=0.15;                                                          # coupling strength

MOD_par_add_23=[epsilon -epsilon epsilon -epsilon epsilon -epsilon];   # MOD_par for coupling part
```

We want to have for each of the three cases the same number of sliding windows in the DDA part. We therefore need to adjust the integration length according to the DDA parameters. For data of length `L`, the maximal delay `TM`, the number of data points for numerical integration `dm`, a window length `WL`, and a window shift `WS` the window number we loose `dm + TM` data points at the beginning of the time series and dm data points at the end. The number of windows WN of the DDA output is then `WN = 1 + floor((L-WL-TM-2*dm)/WS)`.

For anticipated 500 windows we then can compute the data lengths for the three cases.

```
TAU=[32 9]; TM=maximum(TAU); dm=4;                                     # DDA parameters
WL=2000;WS=500;                                                        # window length and window shift for DDA
WN=500;                                                                # assign window number for each case
L1=WS*(WN-1)+WL+TM+dm;                                                 # ajust integration length to have
L2=WS*WN;                                                              # equal number of windows for each case
L3=WS*WN+dm;
```

The seven Rössler systems are integrated with a step size of 0.05 and down-sampled by a factor of two.

```
TRANS=20000;
dt=0.05;
X0 = rand(DIM*NrSyst,1);                                               # initial conditions

X=integrate_ODE_general_BIG(MOD_nr,MOD_par,                            # encoding of the 7 systems, case (i)
                            dt,                                        # step size of num. integration
                            L1*2,                                      # length * 2
                            DIM*NrSyst,ODEorder,X0,                    # parameters
                            TRANS);                                    # transient

X0=X[end,:];                                                           # initial conditions for (ii)
Y=integrate_ODE_general_BIG([MOD_nr I2],[MOD_par MOD_par_add_23],      # encoding of the coupled systems (ii)
                            dt,                                        # step size of num. integration
                            L2*2,                                      # length * 2
                            DIM*NrSyst,ODEorder,X0,                    # parameters
                            0);                                        # no transient

X=[X; Y];                                                              # concatenate (i) and (ii)
Y = nothing; GC.gc();                                                  # clear variable Y
X0=X[end,:];                                                           # initial conditions for (iii)
Y=integrate_ODE_general_BIG([MOD_nr I3],[MOD_par MOD_par_add_23],      # encoding of the coupled systems (iii)
                            dt,                                        # step size of num. integration
                            L3*2,                                      # length * 2
                            DIM*NrSyst,ODEorder,X0,                    # parameters
                            0);                                        # no transient

X=[X;Y];                                                               # concatenate (i), (ii), and (iii)
Y = nothing; GC.gc();                                                  # clear variable Y

X = X[1:2:end,1:3:end];                                                # take every second data point
                                                                       #   and only u_{1,n}

SG = plot(layout = (3,7),size=(2100,800));                             # make plot of delay embeddings
for k1=1:3
    for k2=1:7
        plot!(SG,subplot=(k1-1)*7+k2,
              X[((20000:24000) .+ (k1-1)*L2),k2], X[((20000:24000) .+ (k1-1)*L2) .- 10,k2], legend=false)
        end
    end
display(SG)

dir_exist("PDFs");
savefig("PDFs/Roessler_7syst_NoNoise.pdf")

dir_exist("CD_DDA");
FN=@sprintf("CD_DDA/CD_DDA_data_NoNoise__WL%d_WS%d_WN%d.ascii",
             WL,WS,WN);                                                # noise free data file
writedlm(FN, map(number_to_string, X),' ');
```

We add white noise with a signal-to-noise ratio of 15dB to the data.

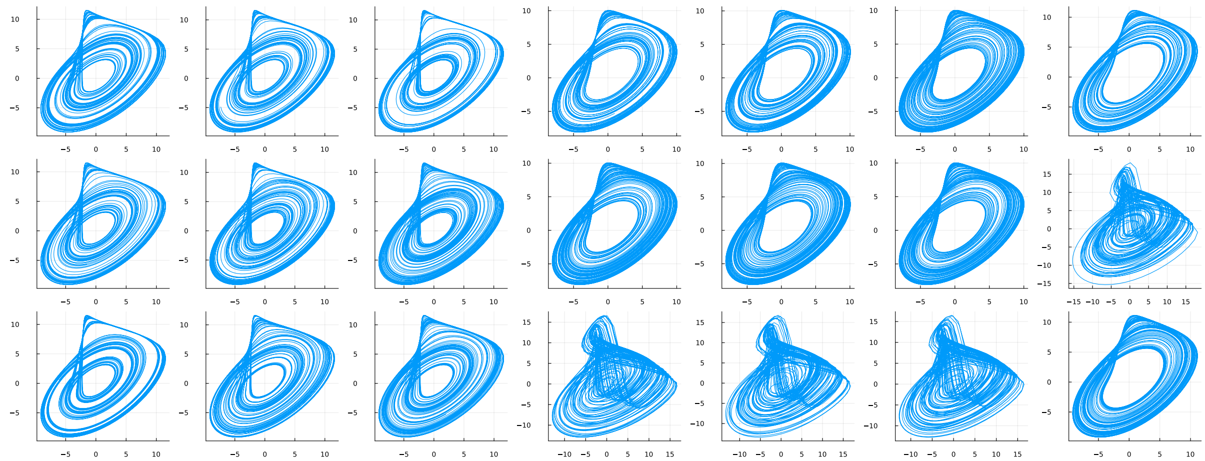

Figure 2: Delay embeddings of the 7 Rössler systems without noise.

```
SNR=15;                                                   # signal-to-noise ratio in dB
for k=1:size(X,2)                                         # add noise
    X[:,k]=add_noise(X[:,k],SNR);
end

SG = plot(layout = (3,7),size=(2100,800));                # make plot of delay embeddings
for k1=1:3
    for k2=1:7
        plot!(SG,subplot=(k1-1)*7+k2,
              X[((20000:24000) .+ (k1-1)*L2),k2], X[((20000:24000) .+ (k1-1)*L2) .- 10,k2], legend=false)
        end
    end
display(SG)
savefig("PDFs/Roessler_7syst_15dB.pdf")

FN=@sprintf("CD_DDA/CD_DDA_data_15dB__WL%d_WS%d_WN%d.ascii",WL,WS,WN);
writedlm(FN, map(number_to_string, X),' ');               # save data
X = nothing; GC.gc();
```

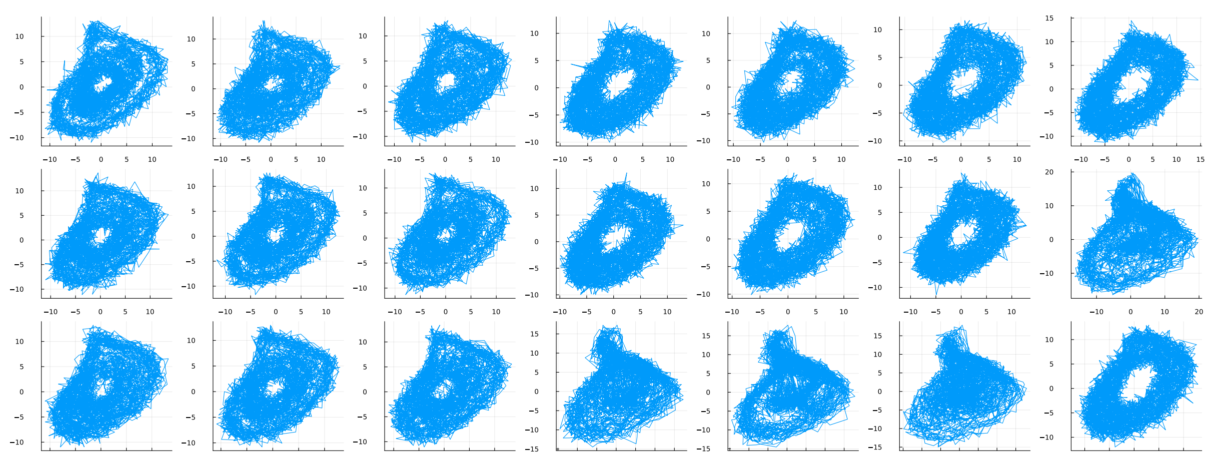

Figure 3: Delay embeddings of the 7 Rössler systems with added white noise.

3 DDA

For the DDA part we choose a window length of 2000 data points and a window shift of 500 data points. We use the same model and delays as in [2]:

$$\dot{v} \;\; = \;\; a_1\, v_1 + a_2\, v_2 + a_3\, v_1^3 \tag{4}$$

with $v_j = v(t - \tau_j)$, $\tau_1 = 32\ \delta t$, $\tau_2 = 9\ \delta t$, and $\delta t = 0.025$.

The encoding of Eq. (4) is as follows:

DDA	variable		
$\dot{v} = a_1\, {\color{red}v_1} + a_2\, v_2 + a_3\, v_1^3$	0	0	1
$\dot{v} = a_1\, v_1 + a_2\, {\color{red}v_2} + a_3\, v_1^3$	0	0	2
$\dot{v} = a_1\, v_1 + a_2\, v_2 + a_3\, {\color{red}v_1^3}$	1	1	1

We compute DE-DDA ($\mathcal{E}$) as explained in [1] for all pairwise combinations of the seven x_n components of the seven Rössler systems in Eq. (2). The lower the value of $\mathcal{E}$ the more dynamically similar the data are.

```
#include("DDAfunctions.jl");                                      # set of Julia functions
#
#WL=2000;WS=500;WN=500;                                           # window length and shift
##WL=4000;WS=1000;WN=2000;

NrSyst=7; DIM=3;                                                  # 7 3D systems

NrCH=NrSyst; CH=collect(1:NrCH);                                  # x-components of 7 systems are channels

LIST=collect(combinations(CH,2));                                 # pairwise combinations of channels
LL1=vcat(LIST...)';
LIST=reduce(hcat,LIST)';

nr_delays=2; dm=4;                                                # DDA parameters
                                                                  # encoding of DDA model
                                                                  # \dot{v} =
DDAmodel=[[0 0 1];                                                #    a_1 v_1 +
          [0 0 2];                                                #    a_2 v_2 +
          [1 1 1]];                                               #    a_3 v_1^3
(MODEL, L_AF, DDAorder)=make_MODEL(DDAmodel);                     # DDA model encoding for DDA code

TAU=[32 9]; TM=maximum(TAU);                                      # delays

FN_data=@sprintf("CD_DDA/CD_DDA_data_NoNoise__WL%d_WS%d_WN%d.ascii",
                  WL,WS,WN);                                      # noise free data file
FN_DDA=@sprintf("CD_DDA/CD_DDA_data_NoNoise__WL%d_WS%d_WN%d.DDA",
                 WL,WS,WN);                                       # DDA file

if !isfile(join([FN_DDA,"_ST"]))
   CMD = "./run_DDA_ASCII -ASCII";                                # DDA executable
   CMD = "$CMD -MODEL $(join(MODEL," "))";                        # model
   CMD = "$CMD -TAU $(join(TAU," "))";                            # delays
   CMD = "$CMD -dm $dm -order $DDAorder -nr_tau $nr_delays";      # DDA parameters
   CMD = "$CMD -DATA_FN $FN_data -OUT_FN $FN_DDA";                # input and output files
   CMD = "$CMD -WL $WL -WS $WS";                                  # window length and shift
   CMD = "$CMD -SELECT 1 1 0 0";                                  # ST and CT DDA
   CMD = "$CMD -WL_CT 2 -WS_CT 2";                                # take 2 channels for CT DDA
   CMD = "$CMD -CT_CH_list $(join(LL1," "))";                     # list of channel pairs for CT DDA

   run(Cmd(string.(split(CMD, " "))));                            # run ST and CT DDA
end

ST=readdlm(join([FN_DDA,"_ST"]));                                 # read ST DDA output file
T=ST[:,1:2]; ST=ST[:,3:end];                                      # first 2 numbers in each line are
                                                                  #    start and end of window
```

```
WN=Int(size(T,1)/3);                                                  # window number = number of lines
ST=ST[:,L_AF:L_AF:end];                                               # need only error
ST=reshape(ST,WN,3,7);                                                # reshape matrix: 3 cases, 7 systems

CT=readdlm(join([FN_DDA,"_CT"]));                                     # read CT DDA output file
CT=CT[:,3:end];                                                       # first 2 numbers in each line are
                                                                      #    start and end of window
CT=CT[:,L_AF:L_AF:end];                                               # need only error
CT=reshape(CT,WN,3,size(LIST,1));                                     # reshape matrix: 3 cases,
                                                                      #    length(LIST) combinations

E=fill(NaN,WN,NrSyst,NrSyst,3);                                       # dynamical ergodicity matrix
for l=1:size(LIST,1)
    ch1=LIST1[l,1];ch2=LIST1[l,2];
    E[:,ch1,ch2,:] = abs.( dropdims(mean(ST[:,:,[ch1,ch2]],dims=3),dims=3) ./ CT[:,:,l] .- 1 );
    E[:,ch2,ch1,:] = E[:,ch1,ch2,:];
end

l=@layout[a{0.7h} ; b c d];                                           # plot results
SG = plot(layout = l,size=(1500,1500));
CG= cgrad([:white, RGB(1,0.97,0.86), RGB(0.55,0.27,0.07)],
          [0,0.1],scale=:linear);

e=reshape(E,size(E,1),NrSyst^2,3);
e=permutedims(e,[1,3,2]);
e=reshape(e,WN*3,NrSyst^2)';

N=tril(reshape(1:NrSyst^2,NrSyst,NrSyst),-1)[:];
N=filter(x -> x != 0, N);
S=[join(string.(x), " ") for x in eachrow(LIST)];

heatmap!(SG,subplot=1,
         e[N,:],
         c=CG,
         xtickfont=font(12), ytickfont=font(12),
         colorbar = true,
         yticks=(1:21,S),
         xticks=(100," ")
         )

e=dropdims(mean(E[20:end,:,:,:],dims=1),dims=1);                      # mean over windows

for k=1:3
        heatmap!(SG,
            subplot = k+1,
            e[:,:,k],
            c=:jet,
            colorbar = true,
            xtickfont=font(12), ytickfont=font(12),
            xlims=(0.5, 7.5), ylims=(0.5, 7.5),
            title=@sprintf("(%d)",k),
            aspect_ratio = :equal
            )
end
display(SG)

print("Make pdf file and continue? ");
readline()

savefig(@sprintf("PDFs/CD_DDA_E__WL%d_WS%d_WN%d.pdf",WL,WS,WN));      # save figure as pdf
```

The Symmetrical DE-DDA ($\mathcal{E}$) matrix is shown in Fig. 4. Systems (1,2,3) and (4,5,6) have similar parameters (see Tab. 1). The combination of those systems ($\mathcal{E}_{1,2}$,$\mathcal{E}_{1,3}$, and $\mathcal{E}_{2,3}$) and ($\mathcal{E}_{4,5}$,$\mathcal{E}_{4,6}$, and $\mathcal{E}_{5,6}$) therefore have the lowest numbers which corresponds to highest dynamical similarity.

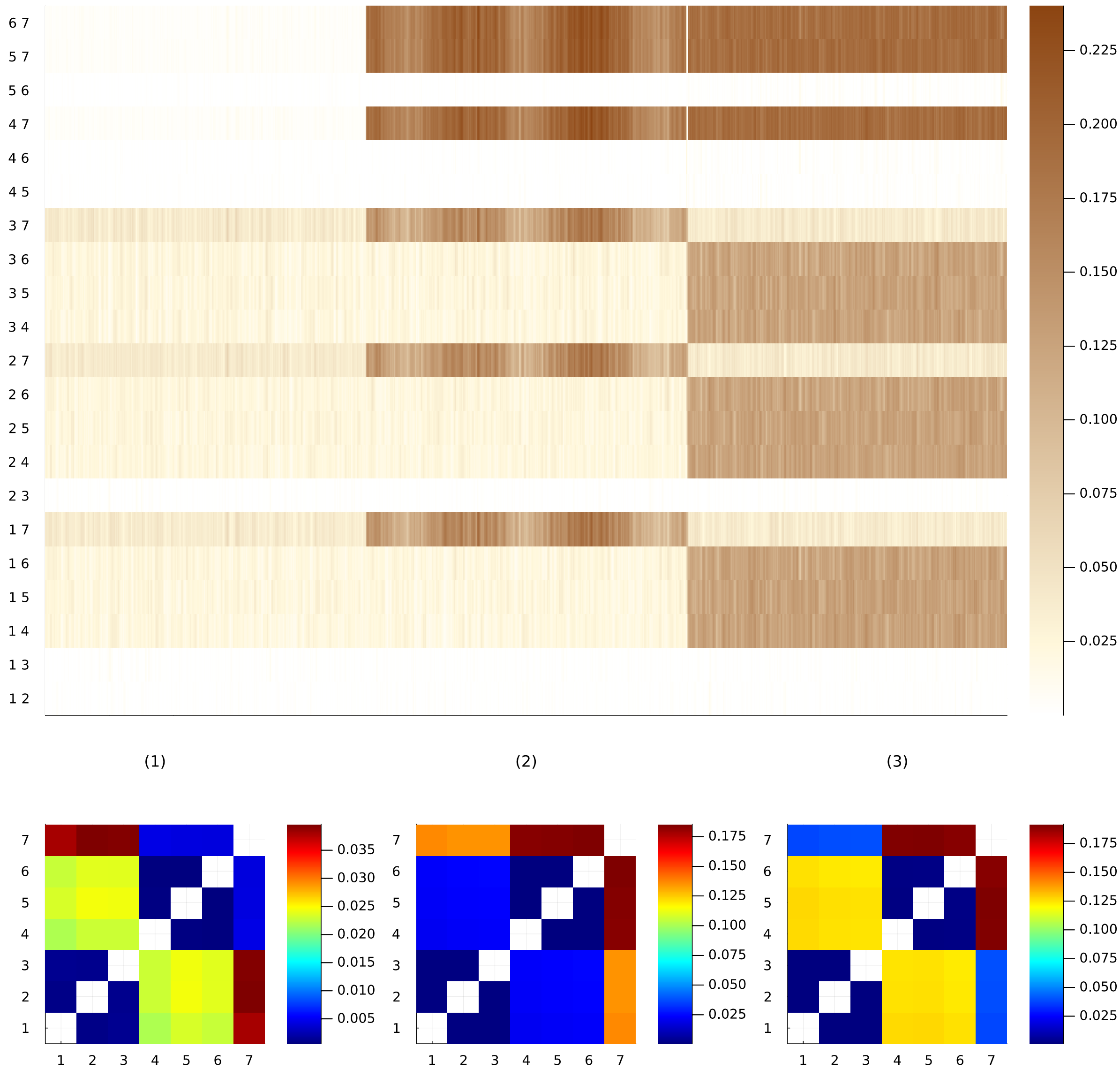

Figure 4: Symmetrical DE-DDA ($\mathcal{E}$) matrices and heatmaps.

Next, we compute the CD-DDA causality matrix of all pairwise combinations of the 7 systems. The results are shown in Fig. 5. In case (i) there are 7 uncoupled systems and no causality. The magenta boxes indicate causality in all directions between systems (1,2,3) and between systems (4,5,6). Those are artifacts because the systems have similar parameters. For the other two cases there are also additional causality artifacts.

```
if !isfile(join([FN_DDA,"_CD_DDA_ST"]))
   CMD = "./run_DDA_ASCII -ASCII";                                   # DDA executable
   CMD = "$CMD -MODEL $(join(MODEL," "))";                           # model
   CMD = "$CMD -TAU $(join(TAU," "))";                               # delays
   CMD = "$CMD -dm $dm -order $DDAorder -nr_tau $nr_delays";         # DDA parameters
   CMD = "$CMD -DATA_FN $FN_data -OUT_FN $FN_DDA";                   # input and output files
   CMD = "$CMD -WL $WL -WS $WS";                                     # window length and shift
   CMD = "$CMD -SELECT 0 0 1 0";                                     # run CD-DDA
   CMD = "$CMD -PAIRS $(join(LL1," "))";                             # pairs for CD-DDA
```

```
   run(Cmd(string.(split(CMD, " "))));                                   # run CD-DDA
end

CD=readdlm(join([FN_DDA,"_CD_DDA_ST"]));                                 # read CD-DDA output file
CD=CD[:,3:end];                                                          # first 2 numbers in each line are
                                                                         #    start and end of window
CD=reshape(CD,WN,3,2,size(LIST,1));                                      # reshape matrix: 3 cases,
                                                                         #    length(LIST) combinations

C=fill(NaN,WN,NrSyst,NrSyst,3);                                          # causality matrix
for l=1:size(LIST,1)
    ch1=LIST1[l,1];ch2=LIST1[l,2];
    C[:,ch1,ch2,:] = CD[:,:,2,l];
    C[:,ch2,ch1,:] = CD[:,:,1,l];
end
C[isnan.(C)].=0;

l=@layout[a{0.5h} ; b c d; e f g];                                       # plot results
SG = plot(layout = l,size=(1500,1500));
CG= cgrad([:white, RGB(1,0.97,0.86), RGB(0.55,0.27,0.07)],
          [0,0.1],scale=:linear);

c=reshape(C,size(C,1),NrSyst^2,3);
c=permutedims(c,[1,3,2]);
c=reshape(c,WN*3,NrSyst^2)';
c[isnan.(c)].=0;
c .= c .- minimum(c[:]);                                                 # normalize to 0 and 1
c .= c ./ maximum(c[:]);

N=reshape(1:NrSyst^2,NrSyst,NrSyst);
k=[N[i, i] for i in 1:NrSyst];
N=(1:NrSyst^2)[setdiff(1:NrSyst^2,k)];
S=collect(permutations(CH,2));
S=reduce(hcat,S)';
S=[join(string.(x), " ") for x in eachrow(S)];

heatmap!(SG,subplot=1,
         c[N,:],
         c=CG,
         xtickfont=font(12), ytickfont=font(12),
         colorbar = true,
         yticks=(1:42,S),
         xticks=(100," ")
         )

c=dropdims(mean(C[50:end,:,:,:],dims=1),dims=1);                         # mean over windows
c[isnan.(c)].=0;
c .= c .- minimum(c[:]);                                                 # normalize to 0 and 1
c .= c ./ maximum(c[:]);

CG= cgrad([RGB(0.9,0.9,0.9), RGB(0.3,.3,0.3), :magenta, :cyan],         # define the color map
          [0.0, 0.25, 0.2501, 0.635, 1],scale=:linear);                  #   lower quarter in grays

h = [heatmap!(SG,subplot=k+1,
              c[:,:,k],                                                  # heatmaps
              c = CG, clim=(0,1),
              colorbar = true,
              title=@sprintf("(%d)",k),
              xtickfont=font(12), ytickfont=font(12),
              xlims=(0.5, 7.5), ylims=(0.5, 7.5),
              aspect_ratio = :equal
            )
     for k in 1:3];

q=reshape(c,NrSyst*NrSyst,3);                                            # normalize each case to 0 and 1
for k=1:3
    q[:,k] .= q[:,k] .- minimum(q[:,k]);
    q[:,k] .= q[:,k] ./ maximum(q[:,k]);
end
q=reshape(c,NrSyst,NrSyst,3);
q[q .< 0.25] .= 0;                                                       # disregard lower quarter in graphs
```

```
[graphplot!(SG,subplot=k+4,
            DiGraph(q[:,:,k]),                                       # make digraphs
            markersize = .2,
            names = 1:NrSyst,
            fontsize = 12,
            linecolor = :darkgrey,
            method=:chorddiagram,
            fillcolor=:white
           )
     for k in 1:3];

display(SG)

print("Make pdf file and continue? ");
readline()

savefig(@sprintf("PDFs/CD_DDA_C__WL%d_WS%d_WN%d.pdf",WL,WS,WN))
```

To remove the causality artifacts we normalize CD-DDA causality $\mathcal{C}$ with dynamical ergodicity $\mathcal{E}$ by multiplying them. The results are shown in Fig. 7. All artifacts are removed.

```
CC=C.*E;                                                             # causality * ergodicity

l=@layout[a{0.5h} ; b c d; e f g];                                   # plot results
SG = plot(layout = l,size=(1500,1500));
CG= cgrad([:white, RGB(1,0.97,0.86), RGB(0.55,0.27,0.07)],
          [0,0.1],scale=:linear);

c=reshape(CE,size(CE,1),NrSyst^2,3);
c=permutedims(c,[1,3,2]);
c=reshape(c,WN*3,NrSyst^2)';
c[isnan.(c)].=0;
c .= c .- minimum(c[:]);                                             # normalize to 0 and 1
c .= c ./ maximum(c[:]);

heatmap!(SG,subplot=1,
         c[N,:],
         c=CG,
         xtickfont=font(12), ytickfont=font(12),
         colorbar = true,
         yticks=(1:42,S),
         xticks=(100," ")
         )

c=dropdims(mean(CE[50:end,:,:,:],dims=1),dims=1);                    # mean over windows
c[isnan.(c)].=0;
c .= c .- minimum(c[:]);                                             # normalize to 0 and 1
c .= c ./ maximum(c[:]);

CG= cgrad([RGB(0.9,0.9,0.9), RGB(0.3,.3,0.3), :magenta, :cyan],       # define the color map
          [0.0, 0.25, 0.2501, 0.635, 1],scale=:linear);              #   lower quarter in grays

h = [heatmap!(SG,subplot=k+1,
              c[:,:,k],                                              # heatmaps
              c = CG, clim=(0,1),
              colorbar = true,
              title=@sprintf("(%d)",k),
              xtickfont=font(12), ytickfont=font(12),
              xlims=(0.5, 7.5), ylims=(0.5, 7.5),
              aspect_ratio = :equal
            )
     for k in 1:3];

q=reshape(c,NrSyst*NrSyst,3);                                        # normalize each case to 0 and 1
for k=1:3
    q[:,k] .= q[:,k] .- minimum(q[:,k]);
    q[:,k] .= q[:,k] ./ maximum(q[:,k]);
```

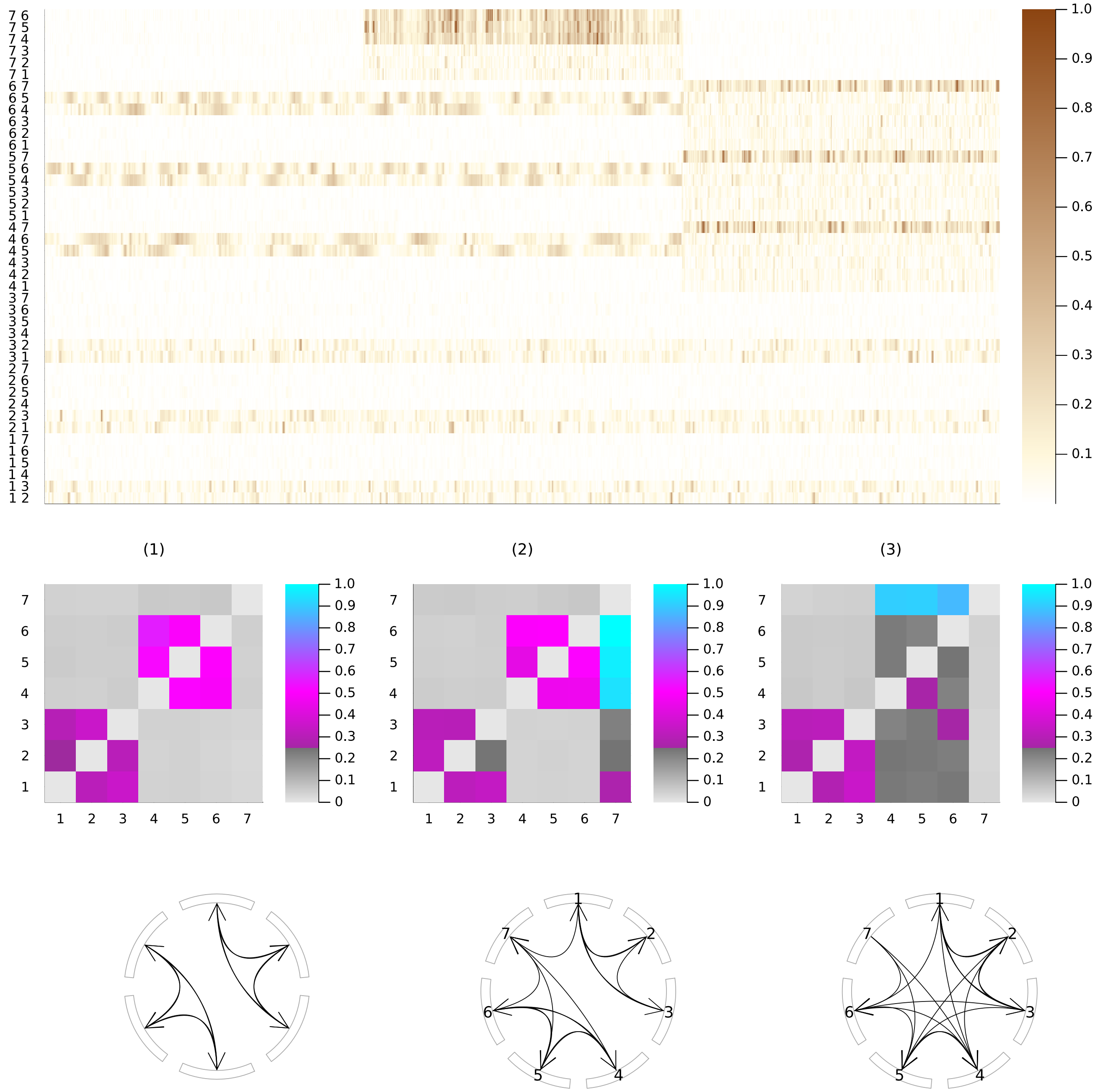

Figure 5: CD-DDA ($\mathcal{C}$)

```
end
q=reshape(c,NrSyst,NrSyst,3);
q[q .< 0.25] .= 0;                                    # disregard lower quarter in graphs

[graphplot!(SG,subplot=k+4,
            DiGraph(q[:,:,k]),                        # make digraphs
            markersize = .2,
            names = 1:NrSyst,
            fontsize = 12,
            linecolor = :darkgrey,
            method=:chorddiagram,
            fillcolor=:white
```

```
            )
      for k in 1:3];

display(SG)

print("Make pdf file and continue? ");
readline()

savefig(@sprintf("PDFs/CD_DDA_CE__WL%d_WS%d_WN%d.pdf",WL,WS,WN))
```

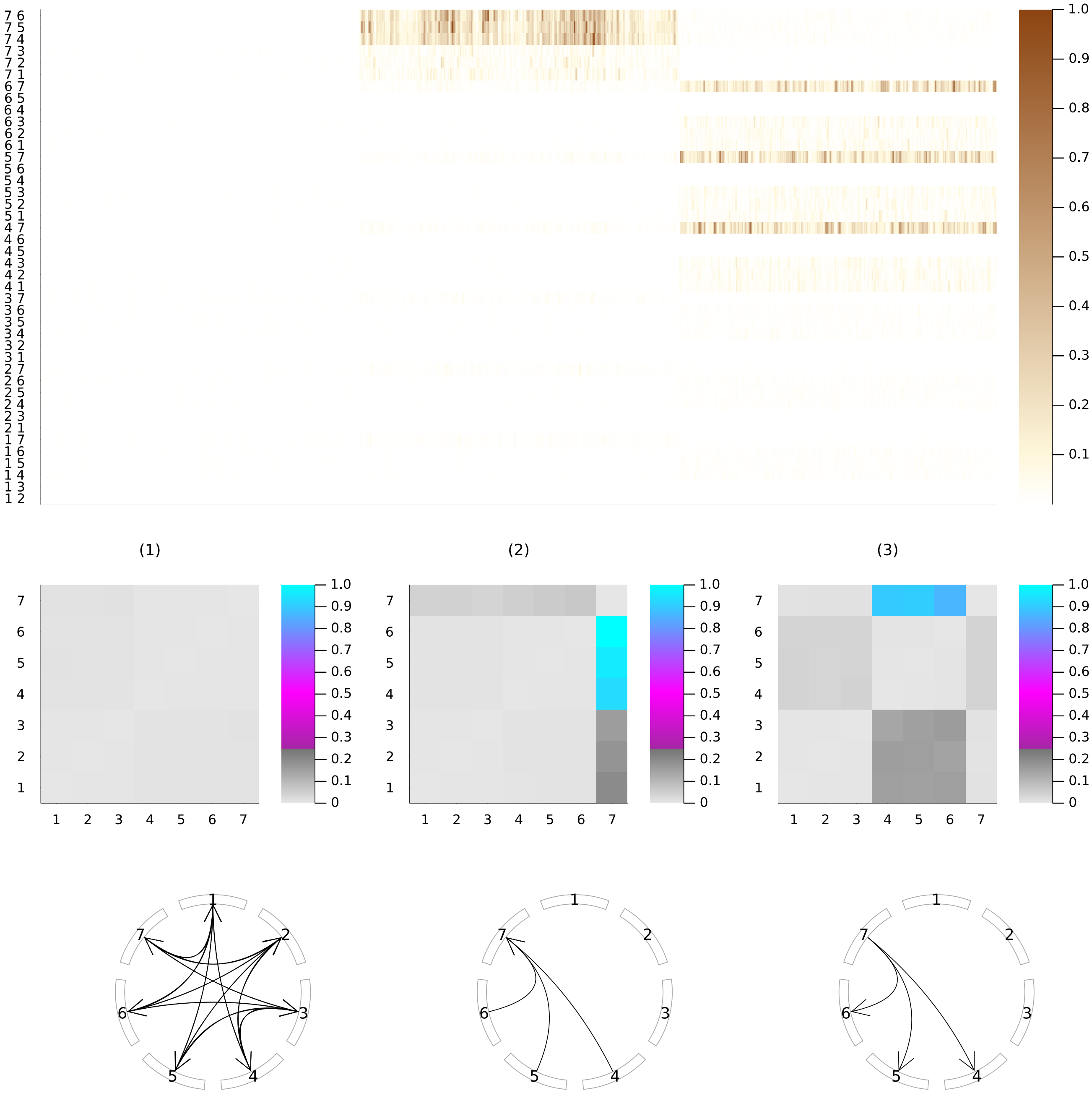

Figure 6: CD-DDA ($\mathcal{C} \star \mathcal{E}$)

Next we compute the singular values and make the plots

```
CE[isnan.(CE)] .= 0;
CE=CE./maximum(CE[:]);
CE=permutedims(CE,[1,4,2,3]);
CE=reshape(CE,WN*3,NrCH,NrCH);

WLsvd=100; WSsvd=1;
WNsvd=Int(1+floor((size(CE,1)-WLsvd)/(WSsvd)));

UU=fill(NaN,WNsvd,NrCH^2); SS=fill(NaN,WNsvd);
for wn=1:WNsvd
    (u,s,v)=svd(reshape(CE[(1:WLsvd) .+ (wn-1)*WSsvd,:,:],WLsvd,NrCH^2)');
    UU[wn,:]=u[:,1];
    SS[wn]=s[1,1];
end

t=collect(1:WNsvd) .+ WLsvd;

l=@layout[a{0.7h} ; b];                                         # plot results
SG = plot(layout = l,size=(1000,1000));
CG= cgrad([:white, RGB(1,0.97,0.86), RGB(0.55,0.27,0.07)],
          [0,0.1],scale=:linear);

heatmap!(SG,subplot=1,
         UU[:,N]',
         c=CG,
         xtickfont=font(12), ytickfont=font(12),
         colorbar = false,
         yticks=(1:length(S),S),
         xticks=(100," ")
         )

plot!(SG,subplot=2,
      t,SS,
      label=L"{\cal C} \star {\cal E}",
      xticks=(WN:WN:2*WN),
      xtickfont=font(12), ytickfont=font(12),
      legendfontsize=18)

display(SG)

print("Make pdf file and continue? ");
readline()

savefig(@sprintf("PDFs/CD_DDA_SVD__WL%d_WS%d_WN%d.pdf",WL,WS,WN));
```

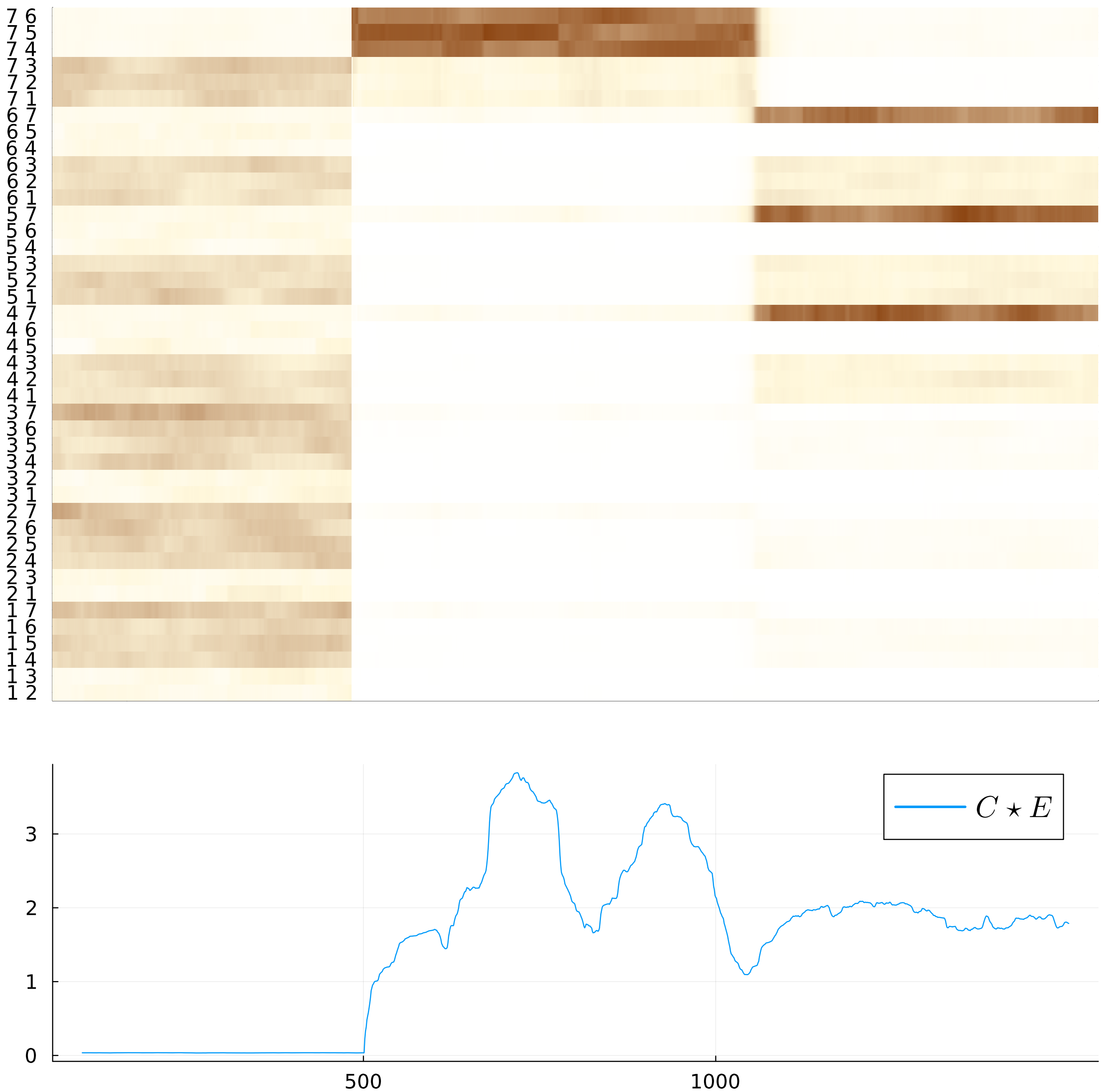

Figure 7: SVD of $(\mathcal{C} \star \mathcal{E})$

References

[1] Lainscsek, C., Cash, S. S., Sejnowski, T. J., and Kurths, J. (2021). Dynamical ergodicity DDA reveals causal structure in time series. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(10):103108.

[2] Lainscsek, C., Gonzalez, C. E., Sampson, A. L., Cash, S. S., and Sejnowski, T. J. (2019). Causality detection in cortical seizure dynamics using cross-dynamical delay differential analysis. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(10):101103.

[3] Lainscsek, C., Salami, P., Carvalho, V. R., Mendes, E. M. A. M., Fan, M., Cash, S. S., and Sejnowski, T. J. (2023). Network-motif delay differential analysis of brain activity during seizures. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(12):123136.

[4] Palus, M. and Vejmelka, M. (2007). Directionality of coupling from bivariate time series: How to avoid false causalities and missed connections. *Phys. Rev. E*, 75.

[5] Rössler, O. E. (1976). Different types of chaos in two simple differential equations. *Z. Naturforsch.*, 31a:1664.

A julia_first_setup.jl

This Julia script installs all packages and detect the OS after instaling Julia.

```
import Pkg; Pkg.add("Combinatorics")
import Pkg; Pkg.add("DataFrames")
import Pkg; Pkg.add("LinearAlgebra")
import Pkg; Pkg.add("Printf")
import Pkg; Pkg.add("Random")
import Pkg; Pkg.add("JLD2")
import Pkg; Pkg.add("Statistics")
import Pkg; Pkg.add("DelimitedFiles")
import Pkg; Pkg.add("Plots")
import Pkg; Pkg.add("StatsBase")

import Pkg; Pkg.add("LaTeXStrings")
import Pkg; Pkg.add("Graphs")
import Pkg; Pkg.add("GraphRecipes")
import Pkg; Pkg.add("Colors")

import Pkg; Pkg.add("MAT")



if Sys.islinux()

   run(`cp i_ODE_general_BIG.linux64 i_ODE_general_BIG`);
   run(`chmod +x i_ODE_general_BIG`);

   run(`cp run_DDA_ASCII.linux64 run_DDA_ASCII`);
   run(`chmod +x run_DDA_ASCII`);

end


if Sys.isapple()
   unm=readchomp(`uname -m`);

   if unm == "arm64"
      run(`cp i_ODE_general_BIG.arm64 i_ODE_general_BIG`);
      run(`cp run_DDA_ASCII.arm64 run_DDA_ASCII`);
   end

   if unm == "x86_64"
      run(`cp i_ODE_general_BIG.x86_64 i_ODE_general_BIG`);
      run(`cp run_DDA_ASCII.x86_64 run_DDA_ASCII`);
   end

   run(`chmod +x i_ODE_general_BIG`);
   run(`chmod +x run_DDA_ASCII`);
end
```

B run_all_in_paper.jl

This Julia scripts does all the Rössler computations and makes the plots.

```
include("DDAfunctions.jl");                                    # set of Julia functions

WL=2000;WS=500;WN=500;                                         # assign window parameters
#WL=4000;WS=1000; WN=2000;                                     # other window settings to try

FN=@sprintf("CD_DDA/CD_DDA_data_NoNoise__WL%d_WS%d_WN%d.ascii",
             WL,WS,WN);                                        # noise free data file
```

```
if !isfile(FN)
   include("make_data_7_systems.jl");
end

include("run_DDA_NoNoise.jl");

include("run_DDA_15dB.jl");
```

C DDAfunctions.jl

```
using DataFrames
using Combinatorics
using LinearAlgebra
using Printf
using Random
using JLD2
using Statistics
using DelimitedFiles
using StatsBase
using Plots
using LaTeXStrings
using Graphs
using GraphRecipes
using Colors
```

```
function dir_exist(DIR)
    if !isdir(DIR)
        mkdir(DIR)
    end
end
```

```
function number_to_string(n::Number)
   return @sprintf("%.15f", n);
end
```

```
function integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,L,DIM,order,X0,TRANS=nothing)

  if TRANS===nothing
     TRANS=0;
  end

  CMD="./i_ODE_general_BIG";
  MOD_NR = join(MOD_nr, " ");
  CMD = "$CMD -MODEL $MOD_NR";
  MOD_PAR = join(MOD_par, " ");
  CMD = "$CMD -PAR $MOD_PAR";
  ANF=join(X0," ");
  CMD = "$CMD -ANF $ANF";
  CMD = "$CMD -dt $(string(dt))";
  CMD = "$CMD -L $(string(L))";
  CMD = "$CMD -DIM $(string(DIM))";
  CMD = "$CMD -order $(string(order))";
  if TRANS>0
     CMD = "$CMD -TRANS $(string(TRANS))";
  end
```

```
  X=readchomp(Cmd(string.(split(CMD, " "))));
  X = split(strip(X), '\n');
  X = hcat([parse.(Float64, split(row)) for row in X]...)';

  return X
end
```

```
function index(DIM, ORDER)
    B = ones(DIM^ORDER, ORDER)

    if DIM > 1
        for i = 2:(DIM^ORDER)
            if B[i-1, ORDER] < DIM
                B[i, ORDER] = B[i-1, ORDER] + 1
            end

            for i_DIM = 1:ORDER-1
                if round((i/DIM^i_DIM - floor(i/DIM^i_DIM))*DIM^i_DIM) == 1
                    if B[i-DIM^i_DIM, ORDER-i_DIM] < DIM
                        for j = 0:DIM^i_DIM-1
                            B[i+j, ORDER-i_DIM] = B[i+j-DIM^i_DIM, ORDER-i_DIM] + 1
                        end
                    end
                end
            end
        end

        i_BB = 1
        BB = Vector{Int}[]
        for i = 1:size(B,1)
            jn = 1
            for j = 2:ORDER
                if B[i, j] >= B[i, j-1]
                    jn += 1
                end
            end
            if jn == ORDER
                push!(BB, B[i, :])
                i_BB += 1
            end
        end
    else
        println("DIM=1!!!")
    end

    return hcat(BB...)
end
```

```
function monomial_list(nr_delays, order)
    # monomials
    P = index(nr_delays+1, order)'
    P = P .- ones(Int64,size(P))

    P = P[2:size(P,1),:];

    return P
end
```

```
function make_MODEL(SYST)
   order=size(SYST,2);
   nr_delays=2;
```

```
   P=monomial_list(nr_delays,order);

   MODEL=fill(0,size(SYST,1))';
   for i=1:size(SYST,1)
       II=SYST[i,:]';

       MODEL[i] = findall( sum(abs.(repeat(II,size(P,1),1)-P),dims=2)' .== 0 )[1][2];
   end
   L_AF=length(MODEL)+1;

   return MODEL, L_AF, order
end
```

```
function make_MOD_nr(SYST,NrSyst)
   DIM=length(unique(SYST[:,1]));
   order=size(SYST,2)-1;

   P=[[0 0]; monomial_list(DIM*NrSyst,order)];

   MOD_nr=fill(0,size(SYST,1)*NrSyst,2);
   for n=1:NrSyst
       for i=1:size(SYST,1)
           II=SYST[i,2:end]';
           II[II .> 0] .+= DIM*(n-1);

           Nr=i+size(SYST,1)*(n-1);
           MOD_nr[Nr,2]=findall( sum(abs.(repeat(II,size(P,1),1)-P),dims=2)' .== 0 )[1][2] - 1;
           MOD_nr[Nr,1]=SYST[i,1]+DIM*(n-1);
       end
       #P[MOD_nr[1:size(SYST,1),2].+1,1:2]
   end
   MOD_nr=reshape(MOD_nr',size(SYST,1)*NrSyst*2)';

   return MOD_nr,DIM,order,P
end
```

```
function make_MOD_nr_Coupling(FromTo,DIM,P)
   order=size(P,2);
   II=fill(0,size(FromTo,1),4);
   for j=1:size(II,1)
       n1=FromTo[j,1]; k1=FromTo[j,2]+1; range1=3:3+order-1;
       n2=FromTo[j,1+range1[end]]; k2=FromTo[j,2+range1[end]]+1; range2=range1 .+ range1[end];

       JJ=FromTo[j,range1]'; JJ[JJ .> 0] .+= DIM * (n1-1);
       II[j,4] = findall( sum(abs.(repeat(JJ,size(P,1),1)-P),dims=2)' .== 0 )[1][2] - 1;

       JJ=FromTo[j,range2]'; JJ[JJ .> 0] .+= DIM * (n2-1);
       II[j,2] = findall( sum(abs.(repeat(JJ,size(P,1),1)-P),dims=2)' .== 0 )[1][2] - 1;

       II[j,1] = DIM*n2-(DIM-k2)-1;
       II[j,3] = DIM*n2-(DIM-k1)-1;
   end
   II=reshape(II',length(II))';

   return II
end
```

```
function add_noise(s,SNR)
   # check the length of the noise free signal
   N = length(s);
```

```
    # n is the noise realization, make it zero mean and unit variance
    n = randn(N);
    n .= (n.-mean(n))./std(n);
    # c is given  from SNR = 10*log10( var(s)/var(c*n) )
    c= sqrt( var(s)*10^-(SNR/10) );

    s_out = (s+c.*n);

    return s_out
end
```