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 1.11.6). The software can be found on Github (`https://github.com/lclaudia/CD-DDA`).

The code is written in JULIA and works on Linux, Mac, and Windows. The underlying C codes are compiled using `cosmocc` from `https://github.com/jart/cosmopolitan`.

LINUX: If you get the error "run-detectors: unable to find an interpreter", you can fix that by running these commands (in bash):
(see `https://github.com/jart/cosmopolitan/blob/master/tool/cosmocc/README.md` for more details).

```
sudo wget -O /usr/bin/ape https://cosmo.zip/pub/cosmos/bin/ape-$(uname -m).elf
sudo chmod +x /usr/bin/ape
sudo sh -c "echo ':APE:M::MZqFpD::/usr/bin/ape:' >/proc/sys/fs/binfmt_misc/register"
sudo sh -c "echo ':APE-jart:M::jartsr::/usr/bin/ape:' >/proc/sys/fs/binfmt_misc/register"
```

WINDOWS: Microsoft might be seeing `run_DDA_ASCII.exe` as a virus and is deleting it. To fix this problem, turn the "Real-time protection" (temporarily) off to execute the codes.

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.

```
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;
MOD_par=[-1 -1 1 a b -c 1];                                     # parameters
# DO NOT FORGET: "chmod +x i_ODE_general_BIG" in linux!
CH_list = 1:3;
DELTA=1;
X = integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,                # integrate system
                              L,DIM,ODEorder,X0,                # function defined in DDAfunctions.jl
                              "",CH_list,DELTA,TRANS);

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 png file and continue? ");
readline()

savefig("Roessler_0.45.png")

b=1;
MOD_par=[-1 -1 1 a b -c 1];                                     # parameters
X = integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,
                              L,DIM,ODEorder,X0,
                              "",CH_list,DELTA,TRANS);

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 png file and continue? ");
readline()

savefig("Roessler_1.png")
```

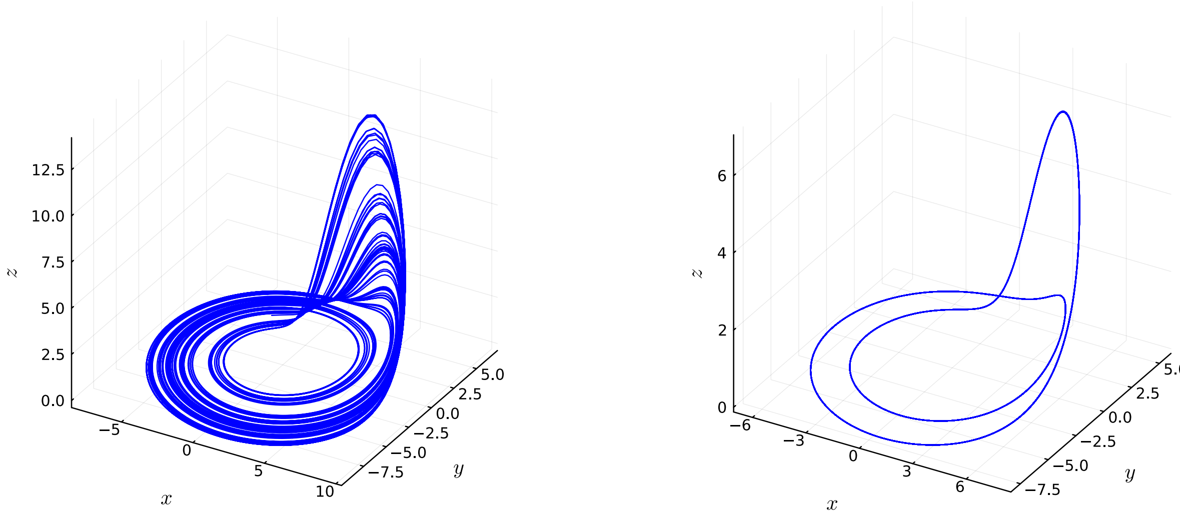

Figure 1: Rössler attractor with $b = 0.45$ (left) and $b = 0.1$ (right)

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

WL=2000;WS=500;WN=500;                                                # assign window parameters
##WL=4000;WS=1000; WN=2000;

DDA_DIR   = "DDA";  dir_exist(DDA_DIR);                               # folders
```

```
DATA_DIR = "DATA"; dir_exist(DATA_DIR);
FIG_DIR  = "FIG";  dir_exist(FIG_DIR);

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:

case	j	Eq. #	variable	n	Eq. #	variable
		from			to	
(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
II=[Int[],I2,I3];

epsilon=0.15;                                                  # coupling strength

MOD_par_add2=repeat([epsilon -epsilon],size(FromTo2,1),1)'[:]';        # MOD_par for coupling part
MOD_par_add3=repeat([epsilon -epsilon],size(FromTo3,1),1)'[:]';        # MOD_par for coupling part

MOD_par_add=[Float64[], MOD_par_add2, MOD_par_add3];
```

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

LL=[WS*(WN-1)+WL+TM+dm;
    WS*WN;
    WS*WN+dm-1];
```

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

```
DELTA=2;                                                       # every second data point
CH_list=1:DIM:DIM*NrSyst;                                      # only x
TRANS=20000;
dt=0.05;

CASE=["i";"ii";"iii"];
for n_CASE=1:length(CASE)
    FN=@sprintf("%s%sCD_DDA_data__WL%d_WS%d_WN%d__case_%s.ascii",
                 DATA_DIR,SL,WL,WS,WN,CASE[n_CASE]);           # data file
    if !isfile(FN)
       X0 = rand(DIM*NrSyst,1);                                # initial conditions
```

```
        if length(II[n_CASE])>0
           M1=[MOD_nr II[n_CASE]]; M2=[MOD_par MOD_par_add[n_CASE]];
        else
           M1=MOD_nr; M2=MOD_par;
        end
        integrate_ODE_general_BIG(M1,M2,                              # encoding of the coupled systems
                                  dt,                                  # step size of num. integration
                                  LL[n_CASE],                          # length
                                  DIM*NrSyst,ODEorder,X0,              # parameters
                                  FN,                                  # output file
                                  CH_list,DELTA,                       # only x; every second point
                                  TRANS);                              # transient
     end
end

global X=Matrix{Any}(undef,1,NrSyst);
for n_CASE=1:length(CASE)
    FN=@sprintf("%s%sCD_DDA_data__WL%d_WS%d_WN%d__case_%s.ascii",
                 DATA_DIR,SL,WL,WS,WN,CASE[n_CASE]);                   # data file
    if n_CASE == 1
       global X=readdlm(FN);
    else
       global X=vcat(X,readdlm(FN));
    end
end

SG = plot(layout = (length(CASE),NrSyst),size=(2100,800));             # make plot of delay embeddings
for n_CASE=1:length(CASE)
    for n_SYST=1:NrSyst
        plot!(SG,subplot=(n_CASE-1)*NrSyst+n_SYST,
              X[((20000:24000) .+ (n_CASE-1)*LL[n_CASE]),      n_SYST],
              X[((20000:24000) .+ (n_CASE-1)*LL[n_CASE]) .- 10,n_SYST],
              legend=false
             )
        end
    end
display(SG)
savefig(SG,@sprintf("%s%sRoessler_7syst_NoNoise.png",DATA_DIR,SL));
```

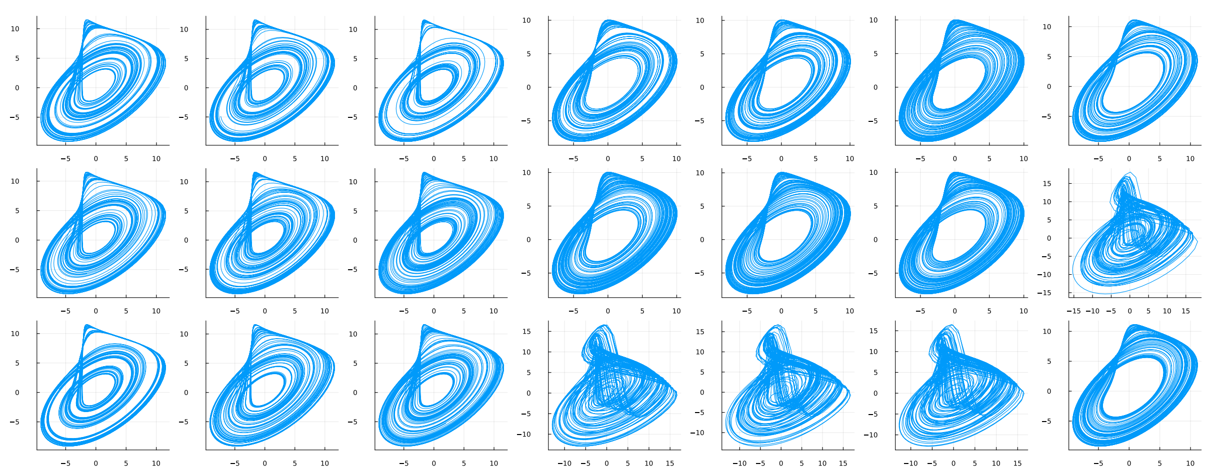

Figure 2: Delay embeddings of the 7 Rössler systems without noise.

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

```
SNR=15;                                                                # signal-to-noise ratio in dB

Y=X .* 1;
```

```
for n_CASE=1:length(CASE)
    for n_SYST=1:NrSyst                                                    # add noise
        range = (1:LL[n_CASE]) .+ (n_CASE-1)*LL[n_CASE];
        Y[range,n_SYST] = add_noise(Y[range,n_SYST],SNR);
    end
end

SG = plot(layout = (length(CASE),NrSyst),size=(2100,800));              # make plot of delay embeddings
for n_CASE=1:length(CASE)
    for n_SYST=1:NrSyst
        plot!(SG,subplot=(n_CASE-1)*NrSyst+n_SYST,
              Y[((20000:24000) .+ (n_CASE-1)*LL[n_CASE]),      n_SYST],
              Y[((20000:24000) .+ (n_CASE-1)*LL[n_CASE]) .- 10,n_SYST],
              legend=false
             )
        end
    end
display(SG)

display(SG)
savefig(SG,@sprintf("%s%sRoessler_7syst_15dB.png",DATA_DIR,SL));



FN=@sprintf("%s%sCD_DDA_data_NoNoise__WL%d_WS%d_WN%d.ascii",DATA_DIR,SL,WL,WS,WN);
writedlm(FN, map(number_to_string, X),' ');                                # save data

FN=@sprintf("%s%sCD_DDA_data_15dB__WL%d_WS%d_WN%d.ascii",DATA_DIR,SL,WL,WS,WN);
writedlm(FN, map(number_to_string, Y),' ');                                # save data
```

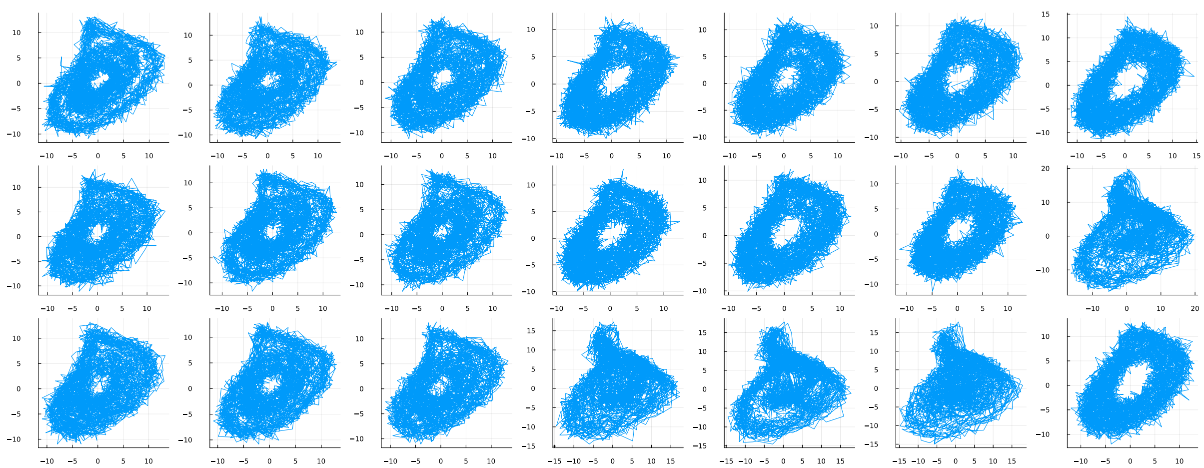

Figure 3: Delay embeddings of the 7 Rössler systems with added white noise.

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]:

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

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.

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.

In the same step, 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.

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. 6. All artifacts are removed.

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

DDA_DIR  = "DDA";  dir_exist(DDA_DIR);                       # folders
DATA_DIR = "DATA";
FIG_DIR  = "FIG";  dir_exist(FIG_DIR);

NOISE=["NoNoise";"15dB"];

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

for n_NOISE = 1:length(NOISE)
```

```
    noise=NOISE[n_NOISE];

    FN_data=@sprintf("%s%sCD_DDA_data_%s__WL%d_WS%d_WN%d.ascii",
                     DATA_DIR,SL,noise,WL,WS,WN);                         # data file
    FN_DDA=@sprintf("%s%s%s__WL%d_WS%d_WN%d.DDA",
                    DDA_DIR,SL,noise,WL,WS,WN);                           # DDA file

    if !isfile(join([FN_DDA,"_ST"]))
       if Sys.iswindows()
          if !isfile("run_DDA_AsciiEdf.exe")
             cp("run_DDA_AsciiEdf","run_DDA_AsciiEdf.exe");
          end

          CMD=".\\run_DDA_AsciiEdf.exe";
       else
          CMD="./run_DDA_AsciiEdf";
       end

       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 1 0";                                      # ST, CT, and CD DDA
       CMD = "$CMD -WL_CT 2 -WS_CT 2";                                    # take pairwise channels for CT and CD
       CMD = "$CMD -CH_list $(join(LL1," "))";                            # list of channel pairs

       if Sys.iswindows()                                                 # run ST, CT, and CD DDA
          run(Cmd(string.(split(CMD, " "))));
       else
          run(`sh -c $CMD`);
       end

       rm(@sprintf("%s.info",FN_DDA));
    end
end
```

Then we plot the results:

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

WL=2000;WS=500;                                                           # window length and shift
global WN=500;

DDA_DIR  = "DDA";  dir_exist(DDA_DIR);                                    # folders
DATA_DIR = "DATA";
FIG_DIR  = "FIG";  dir_exist(FIG_DIR);

NOISE=["NoNoise";"15dB"];

NrSyst=7; DIM=3;                                                          # 7 3D systems
NrCH=NrSyst; CH=collect(1:NrCH);                                          # x-components of 7 systems are channels

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

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

for n_NOISE = 1:length(NOISE)
    noise=NOISE[n_NOISE];

    FN_DDA=@sprintf("%s%s%s__WL%d_WS%d_WN%d.DDA",
                    DDA_DIR,SL,noise,WL,WS,WN);                           # DDA file

    E=fill(NaN,WN,NrSyst,NrSyst,3);                                       # dynamical ergodicity matrix
```

```
    C=fill(NaN,WN,NrSyst,NrSyst,3);                                  # causality matrix

    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
    ST=ST[:,L_AF:L_AF:end];                                          # need only error
    ST=reshape(ST,WN,3,NrSyst);                                      # 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
    for l=1:size(LIST,1)
        ch1=LIST[l,1];ch2=LIST[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

    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
    for l=1:size(LIST,1)
        ch1=LIST[l,1];ch2=LIST[l,2];
        C[:,ch1,ch2,:] = CD[:,:,2,l];
        C[:,ch2,ch1,:] = CD[:,:,1,l];
    end

###   plot results

    l=@layout[a{0.7h} ; b c d];
    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);
    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 png file and continue? ");
```

```
    readline()
    savefig(SG,@sprintf("%s%sE__WL%d_WS%d_WN%d_%s.png",FIG_DIR,SL,WL,WS,WN,noise));

###

    l=@layout[a{0.5h} ; b c d; e f g];
    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 .= c .- minimum(filter(!isnan,c[:]));
    c .= c ./ maximum(filter(!isnan,c[:]));

    N=setdiff(1:NrSyst^2, diagind(C[1,:,:,1]));
    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);
    c .= c .- minimum(filter(!isnan,c[:]));
    c .= c ./ maximum(filter(!isnan,c[:]));

    CG= cgrad([RGB(0.9,0.9,0.9), RGB(0.3,.3,0.3), :magenta, :cyan],
              [0.0, 0.25, 0.2501, 0.635, 1],scale=:linear);

    h = [heatmap!(SG,subplot=k+1,
                  c[:,:,k],
                  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);
    for k=1:3
        q[:,k] .= q[:,k] .- minimum(filter(!isnan,q[:,k]));
        q[:,k] .= q[:,k] ./ maximum(filter(!isnan,q[:,k]));
    end
    q=reshape(c,NrSyst,NrSyst,3);

    GR.setarrowsize(0.5);

    MS = [1,1,1,2,2,2,3];
    colors = [colorant"plum2", colorant"mistyrose1", colorant"lavender"];

    for k=1:3
        A=q[:,:,k];
        A[A .< 0.25] .= 0;
        A[isnan.(A)] .= 0;

        graphplot!(SG,subplot=k+4,A,
                   method=:circular,nodeshape=:circle,
                   names=1:7,
                   markersize=0.15,
                   fontsize=20,
                   linewidth=3,
```

```
                   linealpha=1,
                   markercolor = colors[MS],
                   nodestrokecolor=colors[MS],
                   arrow=arrow(:closed,10),
                   )
    end
    display(SG)

    print("Make png file and continue? ");
    readline()
    savefig(SG,@sprintf("%s%sC__WL%d_WS%d_WN%d_%s.png",FIG_DIR,SL,WL,WS,WN,noise));

###

    CE=C .* E;                                                                  # causality * ergodicity

    l=@layout[a{0.5h} ; b c d; e f g];
    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 .= c .- minimum(filter(!isnan,c[:]));
    c .= c ./ maximum(filter(!isnan,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);
    c .= c .- minimum(filter(!isnan,c[:]));
    c .= c ./ maximum(filter(!isnan,c[:]));

    CG= cgrad([RGB(0.9,0.9,0.9), RGB(0.3,.3,0.3), :magenta, :cyan],
              [0.0, 0.25, 0.2501, 0.635, 1],scale=:linear);

    h = [heatmap!(SG,subplot=k+1,
                   c[:,:,k],
                   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);
    for k=1:3
        q[:,k] .= q[:,k] .- minimum(filter(!isnan,q[:,k]));
        q[:,k] .= q[:,k] ./ maximum(filter(!isnan,q[:,k]));
    end
    q=reshape(c,NrSyst,NrSyst,3);

    for k=1:3
        A=q[:,:,k];
        A[A .< 0.25] .= 0;
        A[isnan.(A)] .= 0;

        graphplot!(SG,subplot=k+4,A,
                   method=:circular,nodeshape=:circle,
                   names=1:7,
                   markersize=0.15,
                   fontsize=20,
```

```
                linewidth=3,
                linealpha=1,
                markercolor = colors[MS],
                nodestrokecolor=colors[MS],
                arrow=arrow(:closed,10),
                init=0
                )
    end
    display(SG)

    print("Make png file and continue? ");
    readline()
    savefig(SG,@sprintf("%s%sCE__WL%d_WS%d_WN%d_%s.png",FIG_DIR,SL,WL,WS,WN,noise));
end
```

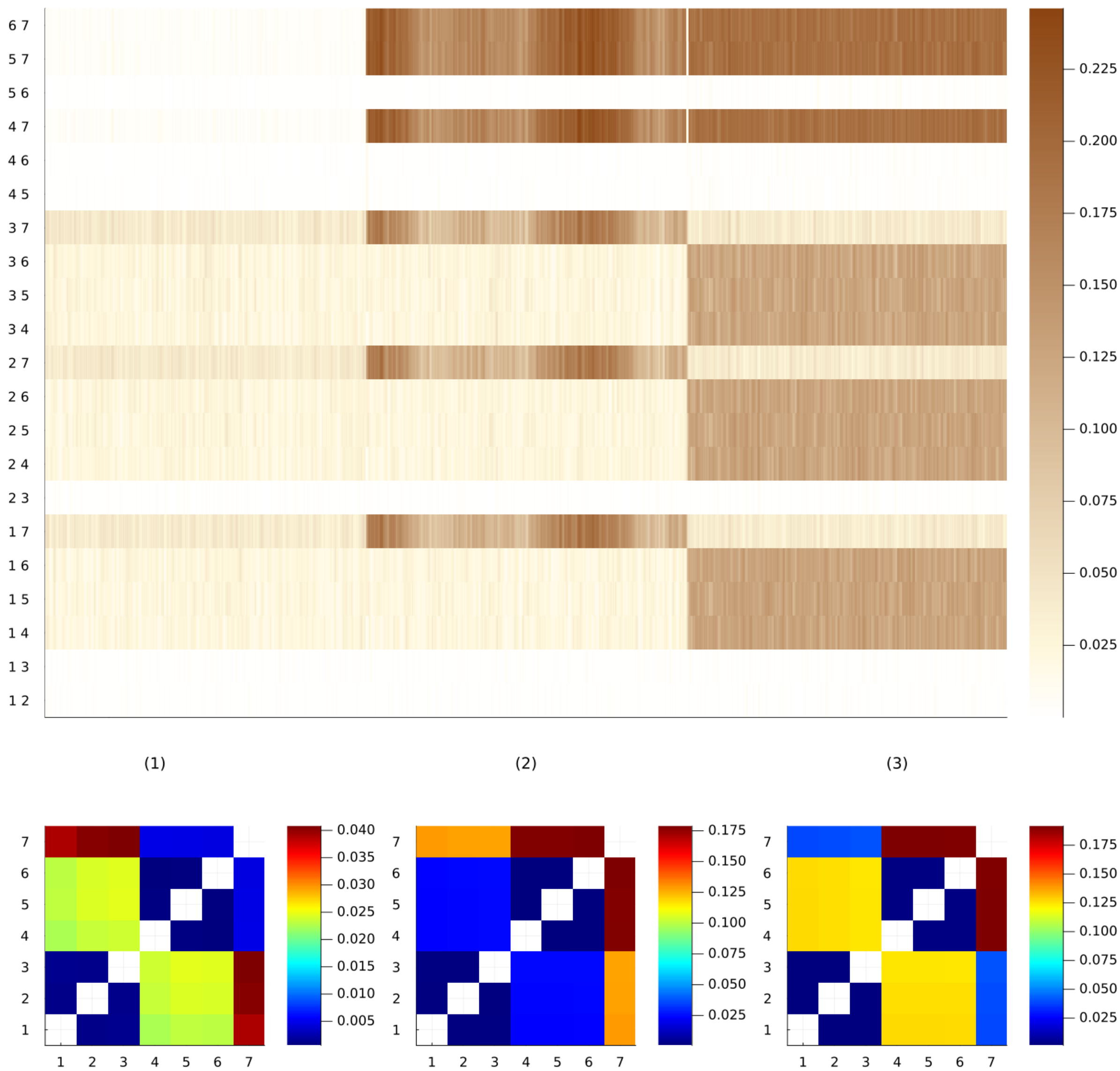

Figure 4: Symmetrical DE-DDA ($\mathcal{E}$) matrices and heatmaps.

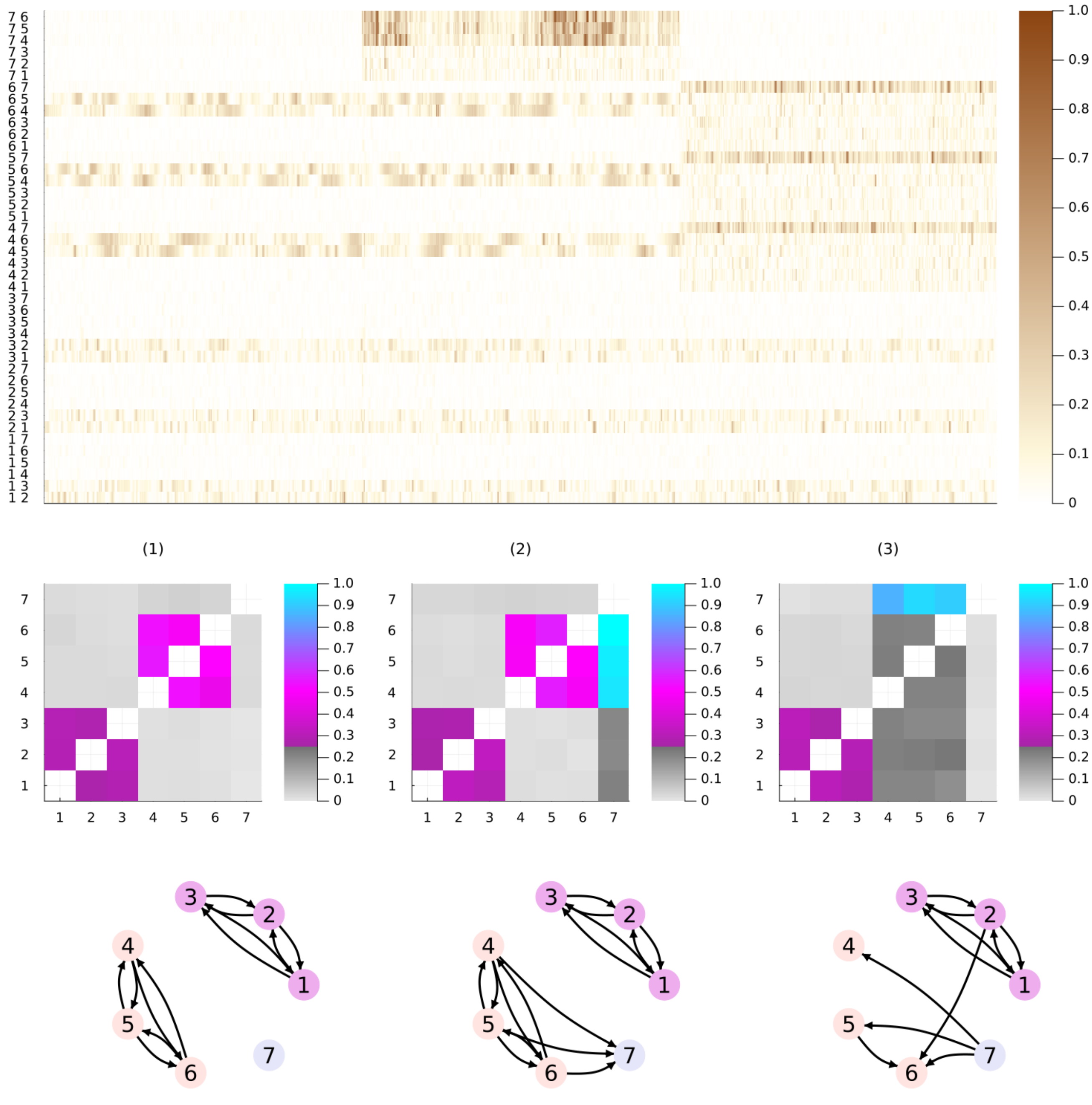

Figure 5: CD-DDA ($\mathcal{C}$)

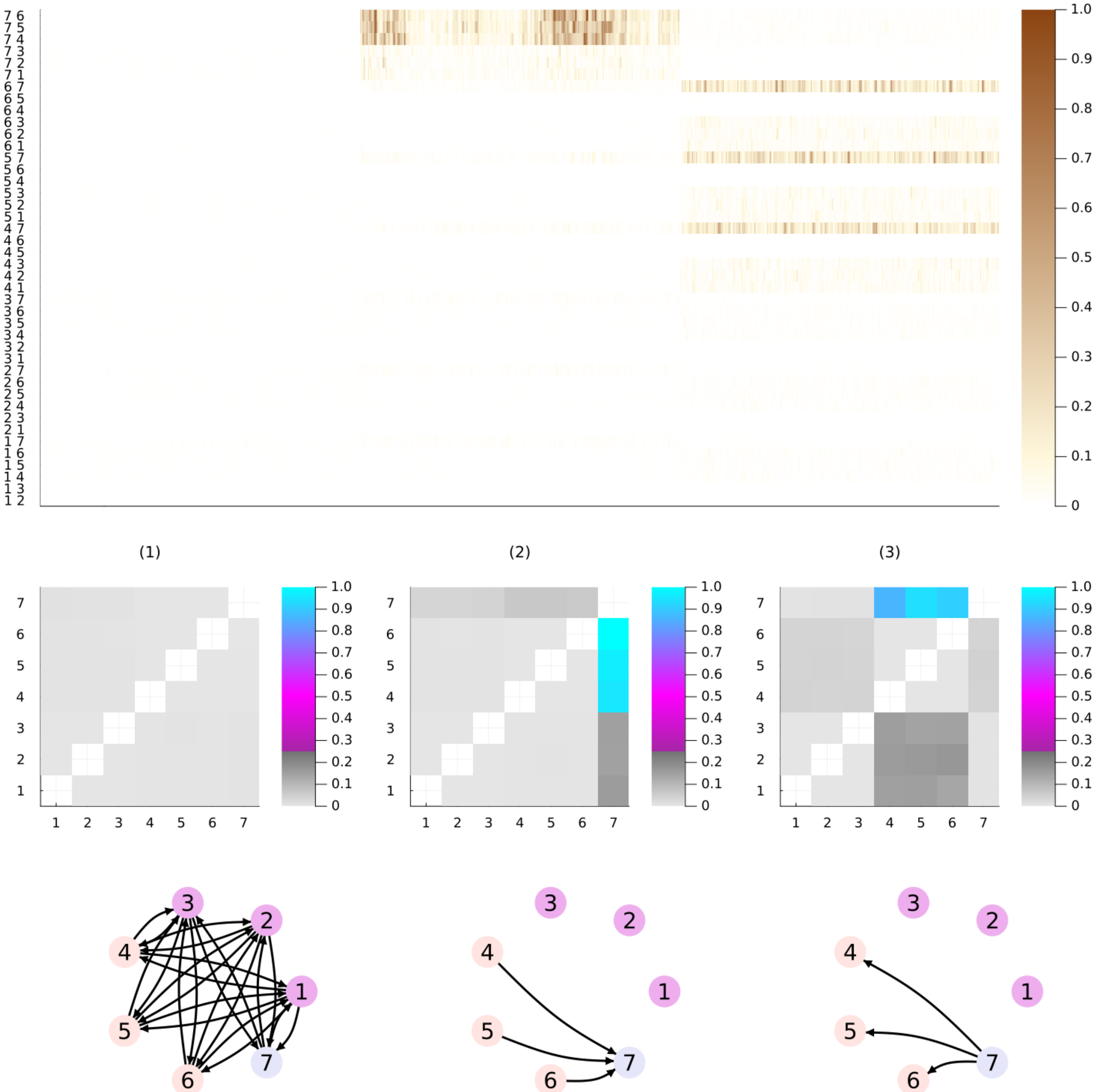

Figure 6: CD-DDA ($\mathcal{C} \star \mathcal{E}$)

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 installing 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")
```

B run_all_in_paper.jl

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

```
#include("julia_first_setup.jl");                                      # packages

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

include("make_data_7_systems.jl");                                      # make data
include("run_DDA_Roessler.jl");                                         # run DDA
include("Roessler_ShowResults.jl");                                     # show results
```

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

if Sys.iswindows()
   SL="\\";
else
   SL="/";
end
```

```
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,ODEorder,X0,FNout,CH_list,DELTA,TRANS=nothing)
  if TRANS===nothing
     TRANS=0;
  end

  if Sys.iswindows()
     if !isfile("i_ODE_general_BIG.exe")
        cp("i_ODE_general_BIG","i_ODE_general_BIG.exe");
     end

     CMD=".\\i_ODE_general_BIG.exe";
  else
     CMD="./i_ODE_general_BIG";
  end

  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 $dt";
  CMD = "$CMD -L $L";
  CMD = "$CMD -DIM $DIM";
  CMD = "$CMD -order $ODEorder";
  if TRANS>0
     CMD = "$CMD -TRANS $TRANS";
  end
  if length(FNout)>0
     CMD = "$CMD -FILE $FNout";
  end
  CMD = "$CMD -DELTA $DELTA";
  CMD = "$CMD -CH_list $(join(CH_list," "))";

  if length(FNout)>0
     if Sys.iswindows()
        run(Cmd(string.(split(CMD, " "))));
     else
        run(`sh -c $CMD`);
     end
  else
     if Sys.iswindows()
       X = read(Cmd(string.(split(CMD, " "))),String);
     else
       X = read(`sh -c $CMD`,String);
     end
     X = split(strip(X), '\n');
     X = hcat([parse.(Float64, split(row)) for row in X]...)';

     return X
  end
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
```