DDA Software

Claudia Lainscsek

This software can be found on Github (`https://github.com/lclaudia/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:
(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 Data for DDA

Before we start with the DDA software we will first generate some simulated data from a 3D nonlinear system of ODEs (ordinary differential equations), namely the Rössler system [9]:

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

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

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

Note, that the equation numbers are (0,1,2) for the three equations. This defines `DIM=3` (the number of equations). There are two "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;                                                            # 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,"",1:3,1,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,"",1:3,1,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")
```

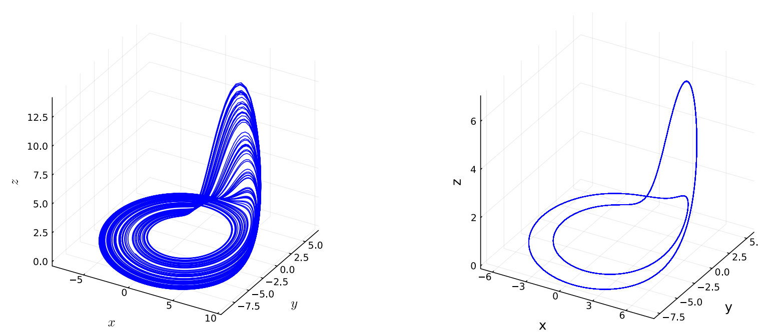


Figure 1: Rössler attractor with $b = 0.45$ (left) and $b = 0.1$ (right)

These are data from a single system

2 Delay Differential Analysis

DDA is a detection/classification framework that combines differential embeddings with linear and nonlinear nonuniform functional delay embeddings [11, 7, 10] to relate the current derivatives of a system to the current and past values of the system variables [1, 6, 3].

More traditional analyses that are often based on spectral features have hundreds of features per data segment and approaches based on artificial neural networks increase the feature space even further. Therefore, such techniques rely on dimensionality reduction techniques to achieve a viable number of features. DDA, on the other hand, achieves a reduced feature space by mapping the data on a "natural" nonlinear basis (inspired by Max Planck's "natural units" [8]) that is selected according to the classification problem. Therefore, DDA is efficient at embedding the meaningful dynamics of the data in a low-dimensional DDA model of only three terms.

2.1 Flavors of DDA

The different versions (flavors) of DDA are summarized in Tab. 1. See the papers in the reference column for more information.

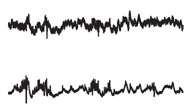
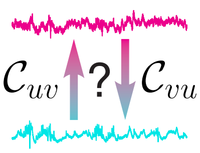
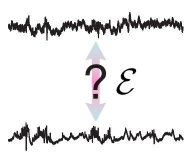
Flavor	Reference	Description
ST-DDA	[5]	$u(t)$  $\dot{\mathbf{u}} = \mathbf{M}_u \ \mathbf{A}_u \Rightarrow \rho_u$
CT-DDA	[4]	$u(t)$, $v(t)$  $\begin{pmatrix}\dot{\mathbf{u}}\\ \dot{\mathbf{v}}\end{pmatrix} = \begin{pmatrix}\mathbf{M}_u\\ \mathbf{M}_v\end{pmatrix} \mathbf{B} \Rightarrow \rho_B$
CD-DDA	[3]	$u(t)$, $v(t)$, $\mathcal{C}_{uv}$? $\mathcal{C}_{vu}$  $\dot{\mathbf{u}} = \mathbf{M}_u \ \mathbf{A}_u \Rightarrow \rho_u$; $\dot{\mathbf{u}} = (\mathbf{M}_u, \mathbf{M}_v) \ \mathbf{D} \Rightarrow \rho_{uv}$; $\dot{\mathbf{v}} = \mathbf{M}_v \ \mathbf{A}_v \Rightarrow \rho_v$; $\dot{\mathbf{v}} = (\mathbf{M}_v, \mathbf{M}_u) \ \mathbf{G} \Rightarrow \rho_{vu}$; $\mathcal{C}_{uv} = \lvert\rho_u - \rho_{uv}\rvert$ $\quad \mathcal{C}_{vu} = \lvert\rho_v - \rho_{vu}\rvert$
DE-DDA	[2]	$u(t)$, $v(t)$, ? $\mathcal{E}$  $\mathcal{E} = \left\lvert \frac{\overline{\rho_u \rho_v}}{\rho_B} - 1 \right\rvert$

Table 1: **The flavors of DDA:** Single-Timeseries DDA (ST-DDA) is the classical variant developed for analyzing single time series. Cross-Timeseries DDA (CT-DDA) determines the overall dynamics of multiple time series simultaneously. Cross-Dynamical DDA (CD-DDA) measures causality between two time series. Dynamical-Ergodicity DDA (DE-DDA)is a combination of ST-DDA and CT-DDA to assess dynamical ergodicity or similarity from data.

2.2 Data

We generate data from the Rössler system in two different dynamical regimes:

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

NrSyst=4;
ROS=[[0  0 2];
     [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);


a123= 0.21;
a456= 0.20;

b1  = 0.2150;
b2  = 0.2020;
b4  = 0.4050;
b5  = 0.3991;

c   = 5.7;

MOD_par=[
         -1 -1 1 a123  b1 -c  1
         -1 -1 1 a123  b2 -c  1
         -1 -1 1 a456  b4 -c  1
         -1 -1 1 a456  b5 -c  1
        ];
MOD_par=reshape(MOD_par',size(ROS,1)*NrSyst)';

TRANS=20000;
dt=0.05;
X0=rand(DIM*NrSyst,1);
CH_list = 1:DIM:DIM*NrSyst;
DELTA=2;
FN_DATA = "ROS_4.ascii";
L=20000;
if !isfile(FN_DATA)
   integrate_ODE_general_BIG(MOD_nr,MOD_par,dt,L,DIM*NrSyst,ODEorder,X0,
                             FN_DATA,CH_list,DELTA,TRANS);
end
```

2.3 Single Timeseries DDA (ST-DDA)

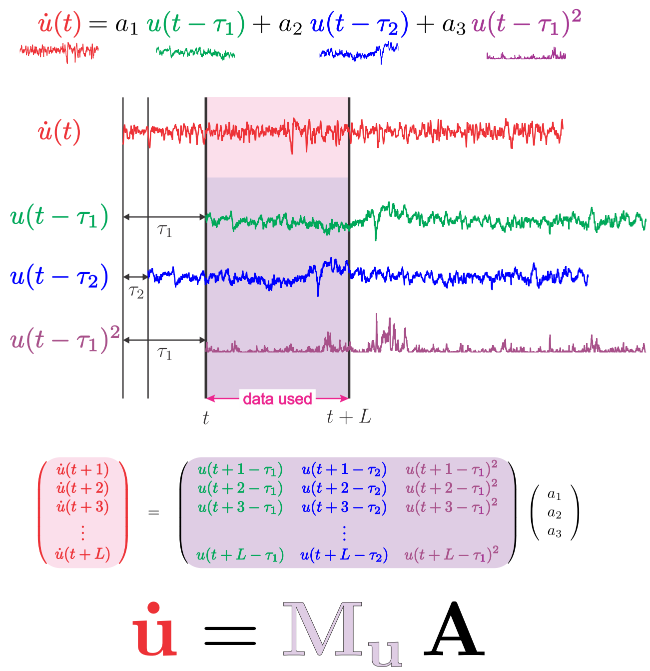


Figure 2: Estimation of the DDA coefficients $A = (a_1, a_2, a_3)$ from data.

We will first generate data from the Rössler system and then apply DDA to these data. To understand the basics of DDA we first code DDA ourselves in Julia before we use the DDA software.

```
###  single time series
Y=readdlm(FN_DATA); Y=Y[:,1];

ST = fill(NaN,WN,4);
for wn=0:WN-1
    anf=wn*WS; ende=anf+WL+TM+2*dm-1;

    data=Y[anf+1:ende+1]; ddata=deriv_all(data,dm); data=data[dm+1:end-dm];

    STD=std(data);
    DATA = (data .- mean(data)) ./ STD; dDATA = ddata / STD;

    M=hcat(DATA[(TM+1:end) .- TAU[1]] ,
           DATA[(TM+1:end) .- TAU[2]] ,
           DATA[(TM+1:end) .- TAU[1]].^3 );
    dDATA=dDATA[TM+1:end];

    ST[wn+1,1:3] = (M \ dDATA);
    ST[wn+1,4]   = sqrt(mean((dDATA .- M*ST[wn+1,1:3]).^2));
end
```

```
###  all four time series
ST = fill(NaN,WN,4,size(Y,2));
for n_Y=1:size(Y,2)
    for wn=0:WN-1
        anf=wn*WS; ende=anf+WL+TM+2*dm-1;

        data=Y[anf+1:ende+1,n_Y]; ddata=deriv_all(data,dm); data=data[dm+1:end-dm];
```

```
        STD=std(data);
        DATA = (data .- mean(data)) ./ STD; dDATA = ddata / STD;

        M=hcat(DATA[(TM+1:end) .- TAU[1]] ,
               DATA[(TM+1:end) .- TAU[2]] ,
               DATA[(TM+1:end) .- TAU[1]].^3 );
        dDATA=dDATA[TM+1:end];

        ST[wn+1,1:3,n_Y] = (M \ dDATA);
        ST[wn+1,4,n_Y]   = sqrt(mean((dDATA .- M*ST[wn+1,1:3,n_Y]).^2));
    end
end
```

2.4 Cross Timeseries DDA (CT-DDA)

```
NrCH=size(Y,2); CH=collect(1:NrCH);
LIST=collect(combinations(CH,2));
LL1=vcat(LIST...)';
LIST=reduce(hcat,LIST)';

CT = fill(NaN,WN,4,size(LIST,1));
for n_LIST=1:size(LIST,1)
    ch1=LIST[n_LIST,1]; ch2=LIST[n_LIST,2];
    for wn=0:WN-1
        anf=wn*WS; ende=anf+WL+TM+2*dm-1;

        data1=Y[anf+1:ende+1,ch1]; ddata1=deriv_all(data1,dm); data1=data1[dm+1:end-dm];
        data2=Y[anf+1:ende+1,ch2]; ddata2=deriv_all(data2,dm); data2=data2[dm+1:end-dm];

        STD=std(data1); DATA1 = (data1 .- mean(data1)) ./ STD; dDATA1 = ddata1 / STD;
        STD=std(data2); DATA2 = (data2 .- mean(data2)) ./ STD; dDATA2 = ddata2 / STD;

        M1=hcat(DATA1[(TM+1:end) .- TAU[1]] ,
                DATA1[(TM+1:end) .- TAU[2]] ,
                DATA1[(TM+1:end) .- TAU[1]].^3 );
        M2=hcat(DATA2[(TM+1:end) .- TAU[1]] ,
                DATA2[(TM+1:end) .- TAU[2]] ,
                DATA2[(TM+1:end) .- TAU[1]].^3 );
        M=vcat(M1,M2); dDATA=vcat(dDATA1[TM+1:end],dDATA2[TM+1:end]);

        CT[wn+1,1:3,n_LIST] = (M \ dDATA);
        CT[wn+1,4,n_LIST]   = sqrt(mean((dDATA .- M*CT[wn+1,1:3,n_LIST]).^2));
    end
end
```

2.5 DDA software run_DDA_AsciiEdf

The same results can be achieved using `run_DDA_AsciiEdf`

```
nr_delays=2;
DDAmodel=[[0 0 1];
          [0 0 2];
          [1 1 1]];
(MODEL, L_AF, DDAorder)=make_MODEL(DDAmodel);

FN_DDA="ROS_4.DDA";

if Sys.iswindows()
   if !isfile("run_DDA_AsciiEdf.exe")
      run(`cp run_DDA_AsciiEdf run_DDA_AsciiEdf.exe`);
```

```
    end
    CMD=".\\run_DDA_AsciiEdf.exe";
else
    CMD="./run_DDA_AsciiEdf";
end
CMD = "$CMD -ASCII";
CMD = "$CMD -MODEL $(join(MODEL," "))"
CMD = "$CMD -TAU $(join(TAU," "))"
CMD = "$CMD -dm $dm -order $DDAorder -nr_tau $nr_delays"
CMD = "$CMD -DATA_FN $FN_DATA -OUT_FN $FN_DDA"
CMD = "$CMD -WL $WL -WS $WS";
CMD = "$CMD -SELECT 1 1 0 0";
CMD = "$CMD -CH_list $(join(LIST'[:]," "))";
CMD = "$CMD -WL_CT 2 -WS_CT 2";
if Sys.iswindows()
    run(Cmd(string.(split(CMD, " "))));
else
    run(`sh -c $CMD`);
end

ST2=readdlm("ROS_4.DDA_ST"); ST2=ST2[:,3:end];
CT2=readdlm("ROS_4.DDA_CT"); CT2=CT2[:,3:end];

mean(reshape(ST,WN,size(ST,2)*size(ST,3)) .- ST2)
mean(reshape(CT,WN,size(CT,2)*size(CT,3)) .- CT2)
```

References

[1] Kremliovsky, M. and Kadtke, J. (1997). Using delay differential equations as dynamical classifiers. *AIP Conference Proceedings*, 411:57.

[2] 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.

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

[4] Lainscsek, C., Sampson, A. L., Kim, R., Thomas, M. L., Man, K., Lainscsek, X., The COGS Investigators, Swerdlow, N. R., Braff, D. L., Sejnowski, T. J., and Light, G. A. (2019b). Nonlinear dynamics underlying sensory processing dysfunction in schizophrenia. *Proceedings of the National Academy of Sciences*, 116(9):3847–3852.

[5] Lainscsek, C., Weyhenmeyer, J., Cash, S. S., and Sejnowski, T. J. (2017). Delay differential analysis of seizures in multichannel electrocorticography data. *Neural computation*, 29(12):3181–3218.

[6] Lainscsek, C., Weyhenmeyer, J., Hernandez, M., Poizner, H., and Sejnowski, T. (2013). Non-linear dynamical classification of short time series of the Rössler system in high noise regimes. *Frontiers in Neurology*, 4(182).

[7] Packard, N. H., Crutchfield, J. P., Farmer, J. D., and Shaw, R. S. (1980). Geometry from a time series. *Phys. Rev. Lett.*, 45:712.

[8] Planck, M. (1900). Über irreversible Strahlungsvorgänge. *Annalen der Physik*, 306(1):69–122.

[9] Rössler, O. E. (1976). An equation for continous chaos. *Physics Letters A*, 57:397.

[10] Sauer, T., Yorke, J. A., and Casdagli, M. (1991). Embedology. *Journal of Statistical Physics*, 65:579.

[11] Takens, F. (1981). Detecting strange attractors in turbulence. In Rand, D. A. and Young, L.-S., editors, *Dynamical Systems and Turbulence, Warwick 1980*, volume 898 of *Lecture Notes in Mathematics*, pages 366–381. Springer Berlin/Heidelberg.