Code study on

Ikeda, Determinant, and Locally linear embedding

A report for Code Study

- Problem statement
- Materials and Data: illustrate input and output
- Mathematics and Methods
- Numerical results
- Conclusions

Step 1. Generate 2001 Ikeda data, x(1:2001) and y(1:2001)
Step2. Set z = [x(1:2000), y(1:2000)], target =x(2:2001).
Learning net_x for approximating $F_x(\mathbf{x}_n \mid \theta)$. Set net_x to net
Step3. Set z = [x(1:2000), y(1:2000)], target =y(2:2001).
Learning net_y for approximating $F_y(\mathbf{x}_n \mid \theta)$. Set net_y to net
Step4. Apply net_x and net_y to predict Ikeda data, x(2001:3000) and y(2001:3000)

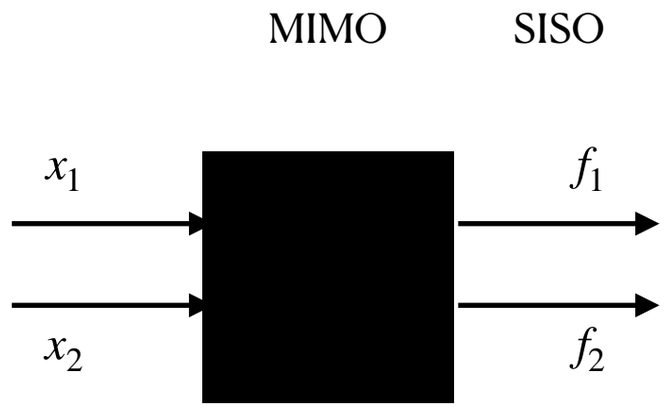
Our first project : learning Ikea data and predicting future data

Imagenet Deep learning of convolutional neural networks for image recognition

First order recursive function : generating Ikeda Map

The Ikeda map[52] is characterized by a nonlinear system,

$$f(x_1, x_2) = \begin{cases} f_1(x_1, x_2) \\ \\ f_2(x_1, x_2) \end{cases}$$



where coordinate functions are given by

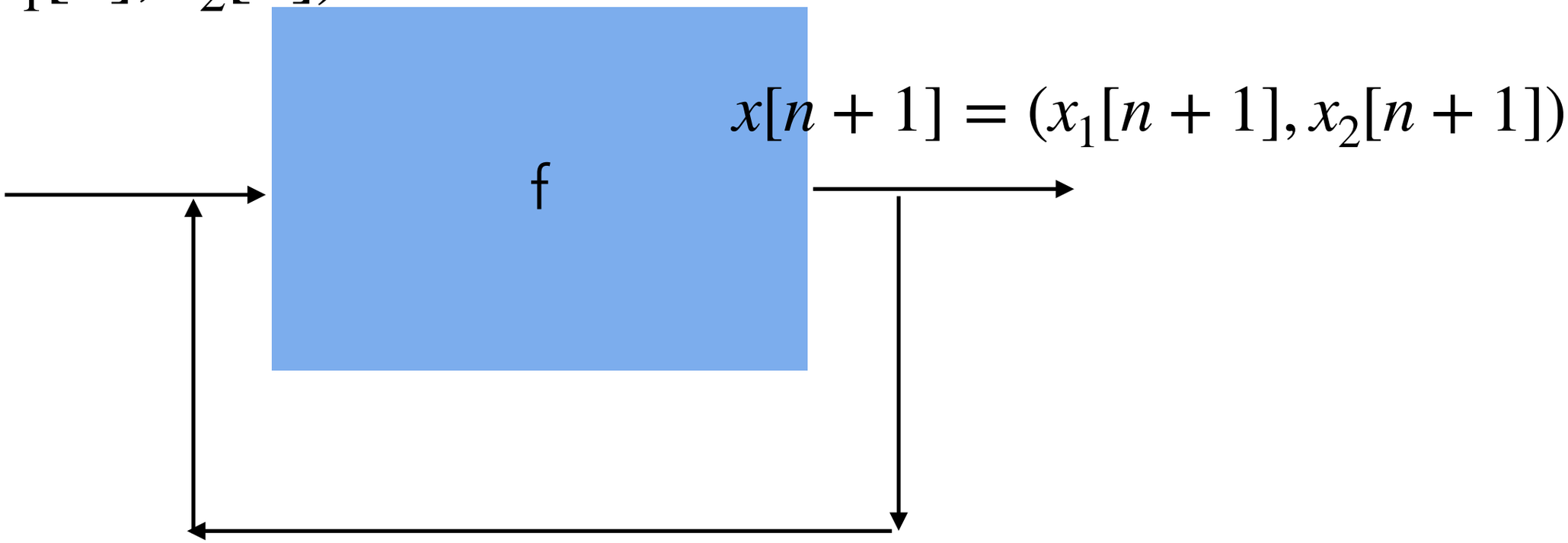
$$\begin{aligned} f_1(x_1, x_2) &= R + C_2(x_1 \cos\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right) - x_2 \sin\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right)), \\ \\ f_2(x_1, x_2) &= C_2(x_1 \sin\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right) + x_2 \cos\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right)), \end{aligned}$$

The parameters are given by $R = 1, C_1 = 0.4, C_2 = 0.9$, and $C_3 = 6$.

Let $\mathbf{x}[n] = (x_1[n], x_2[n])$ and $\mathbf{x}[0]$ denote a random initial condition. The Ikeda time series is generated by the first-order recursive function,

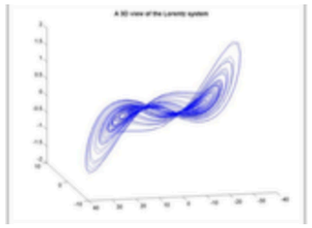
$$\mathbf{x}[n+1] = f(\mathbf{x}[n]). \tag{1}$$

$$x[n] = (x_1[n], x_2[n])$$



$$x[n+1] = (x_1[n+1], x_2[n+1])$$

Alexandros Leontitsis (2021). Chaotic Systems Toolbox (https://www.mathworks.com/matlabcentral/fileexchange/1597-chaotic-systems-toolbox), MATLAB Central File Exchange. Retrieved October 2, 2021.



Chaotic Systems Toolbox

version 1.0.0.0 (26.3 KB) by Alexandros Leontitsis

Analysis of chaotic systems.

This toolbox contains a set of functions which can be used to simulate some of the most known chaotic systems, such as:

- The Henon map
- The Ikeda map
- The Logistic map
- The quadratic map
- The Lorentz flow
- The Mackey-Glass flow
- The Rossler flow

$$
\begin{aligned}
f_1(x_1, x_2) &= R + C_2(x_1 \cos\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right) - x_2 \sin\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right)), \\
f_2(x_1, x_2) &= C_2(x_1 \sin\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right) + x_2 \cos\left(C_1 - C_3/(1 + x_1^2 + x_2^2)\right)),
\end{aligned}
$$

$$t = 0.4 - \frac{6}{1 + x_0^2 + y_0^2}$$

$$x_1 = 1 + \mu(x_0 cos(t) - y_0 sin(t))$$

$$y_1 = \mu(x_0 sin(t) + y_0 cos(t))$$

$$t = 0.4 - \frac{6}{1 + x_0^2 + y_0^2}$$

$$x_1 = 1 + \mu(x_0 cos(t) - y_0 sin(t))$$

$$y_1 = \mu(x_0 sin(t) + y_0 cos(t))$$

parfor

for i=2:n

$$t = 0.4 - \frac{6}{1 + x_{i-1}^2 + y_{i-1}^2}$$

$$x_i = 1 + \mu(x_{i-1} cos(t) - y_{i-1} sin(t))$$

$$y_i = \mu(x_{i-1} sin(t) + y_{i-1} cos(t))$$

$$t = 0.4 - \frac{6}{1 + x_0^2 + y_0^2}$$

$$x_1 = 1 + \mu(x_0 cos(t) - y_0 sin(t))$$

$$y_1 = \mu(x_0 sin(t) + y_0 cos(t))$$

for i=2:n

```
% Initialize
t=0.4-6/(1+x0^2+y0^2);
x(1,1)=1+mu*(x0*cos(t)-y0*sin(t));
y(1,1)=mu*(x0*sin(t)+y0*cos(t));

% Simulate
for i=2:n
    t=0.4-6/(1+x(i-1,1)^2+y(i-1,1)^2);
    x(i,1)=1+mu*(x(i-1,1)*cos(t)-y(i-1,1)*sin(t));
    y(i,1)=mu*(x(i-1,1)*sin(t)+y(i-1,1)*cos(t));
end
```

for i=2:n

$$t = 0.4 - \frac{6}{1 + x_{i-1}^2 + y_{i-1}^2}$$

$$x_i = 1 + \mu(x_{i-1}cos(t) - y_{i-1}sin(t))$$

$$y_i = \mu(x_{i-1}sin(t) + y_{i-1}cos(t))$$

```
% Initialize
t=0.4-6/(1+x0^2+y0^2);
x(1,1)=1+mu*(x0*cos(t)-y0*sin(t));
y(1,1)=mu*(x0*sin(t)+y0*cos(t));

% Simulate
for i=2:n
    t=0.4-6/(1+x(i-1,1)^2+y(i-1,1)^2);
    x(i,1)=1+mu*(x(i-1,1)*cos(t)-y(i-1,1)*sin(t));
    y(i,1)=mu*(x(i-1,1)*sin(t)+y(i-1,1)*cos(t));
end
```

sers ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2022-I NA數值分析 ▸ Chaotic Systems Toolbox

ditor - /Users/apple/Desktop/Jiann-Ming Wu/2022-I NA數值分析/Chaotic Systems Toolbox/ikeda.m

demo_fsolve_CNN2.m | demo_LM_CNN_regressions.m | learning_CNN.m | g_hat_CNN.m | ikeda.m | Contents.m

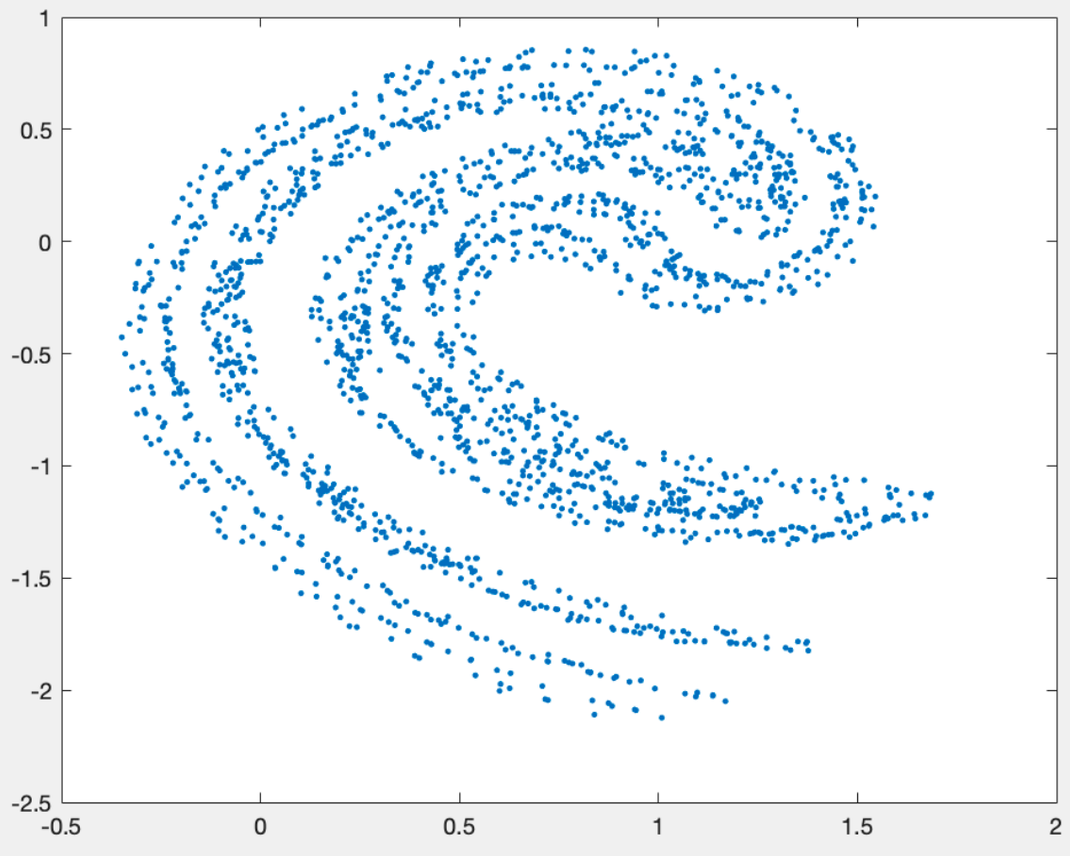
```
function [x,y]=ikeda(n,level,mu,x0,y0)
%Syntax: [x,y]=ikeda(n,level,mu,x0,y0)
%_____________________________________
%
% Simulation of the Ikeda map.
%    x'=1+mu(xcos(t)-y
%    y'=mu(xsin(t)+yco
%
% x and y are the simu
% n is the number of t
% level is the noise s                                    o
%   the noise-free tim
% mu is the parameter.
% x0 is the initial va
% y0 is the initial va
%
%
% Reference:
```

mmand Window

```
>> n=2000;[x,y]=ikeda(n);
>> plot(x,y,'.')
>>
```

Figure 1

File Edit View Insert Tools Desktop Window Help



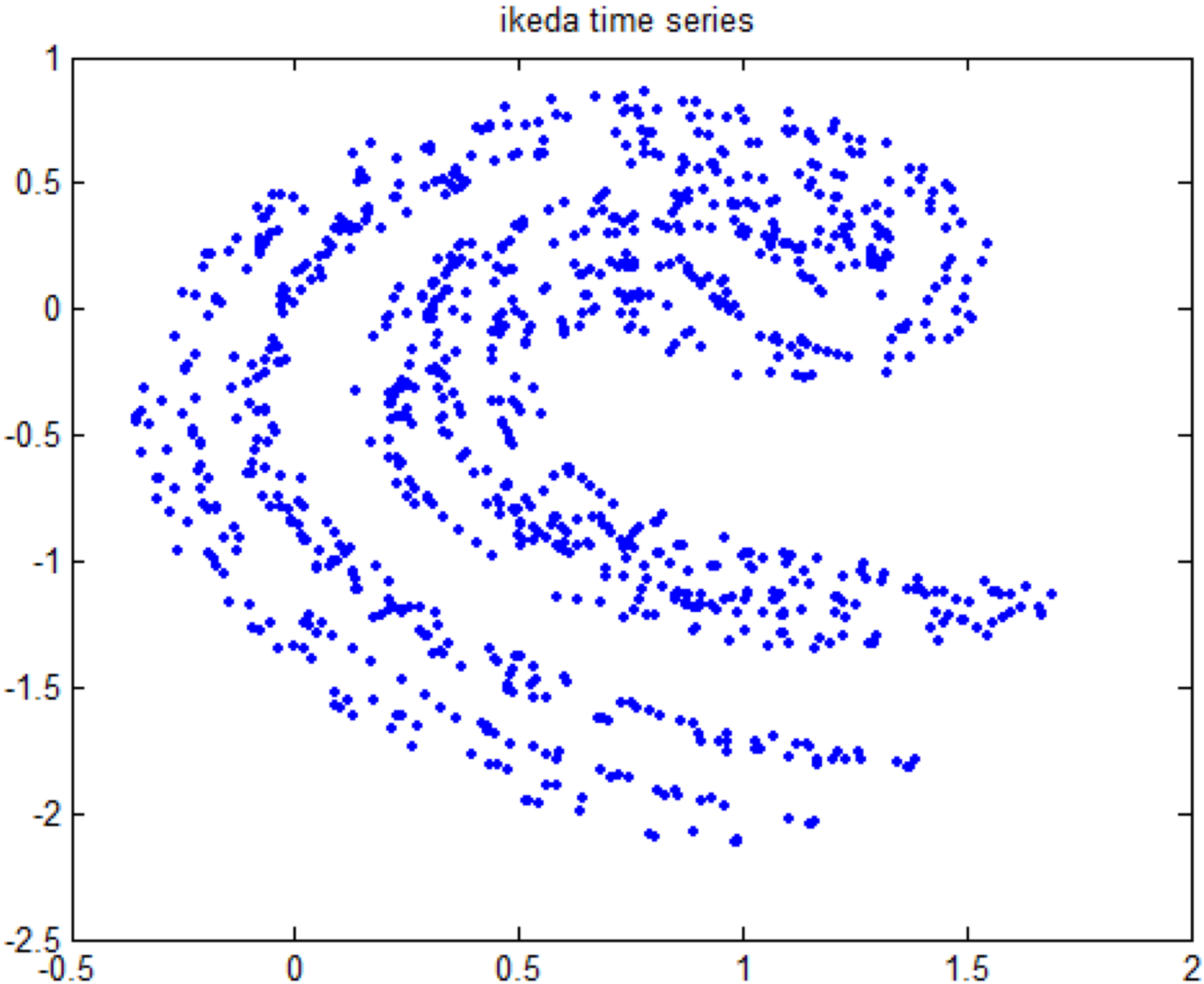
$$t = 0.4 - \frac{6}{1 + x_0^2 + y_0^2}$$

```
%
% Simulation of the Ikeda map.
%    x'=1+mu(xcos(t)-ysin(t)
%    y'=mu(xsin(t)+ycos(t))
%
```

$$x_1 = 1 + \mu(x_0 cos(t) - y_0 sin(t))$$

$$y_1 = 1 + \mu(x_0 sin(t) - y_0 cos(t))$$

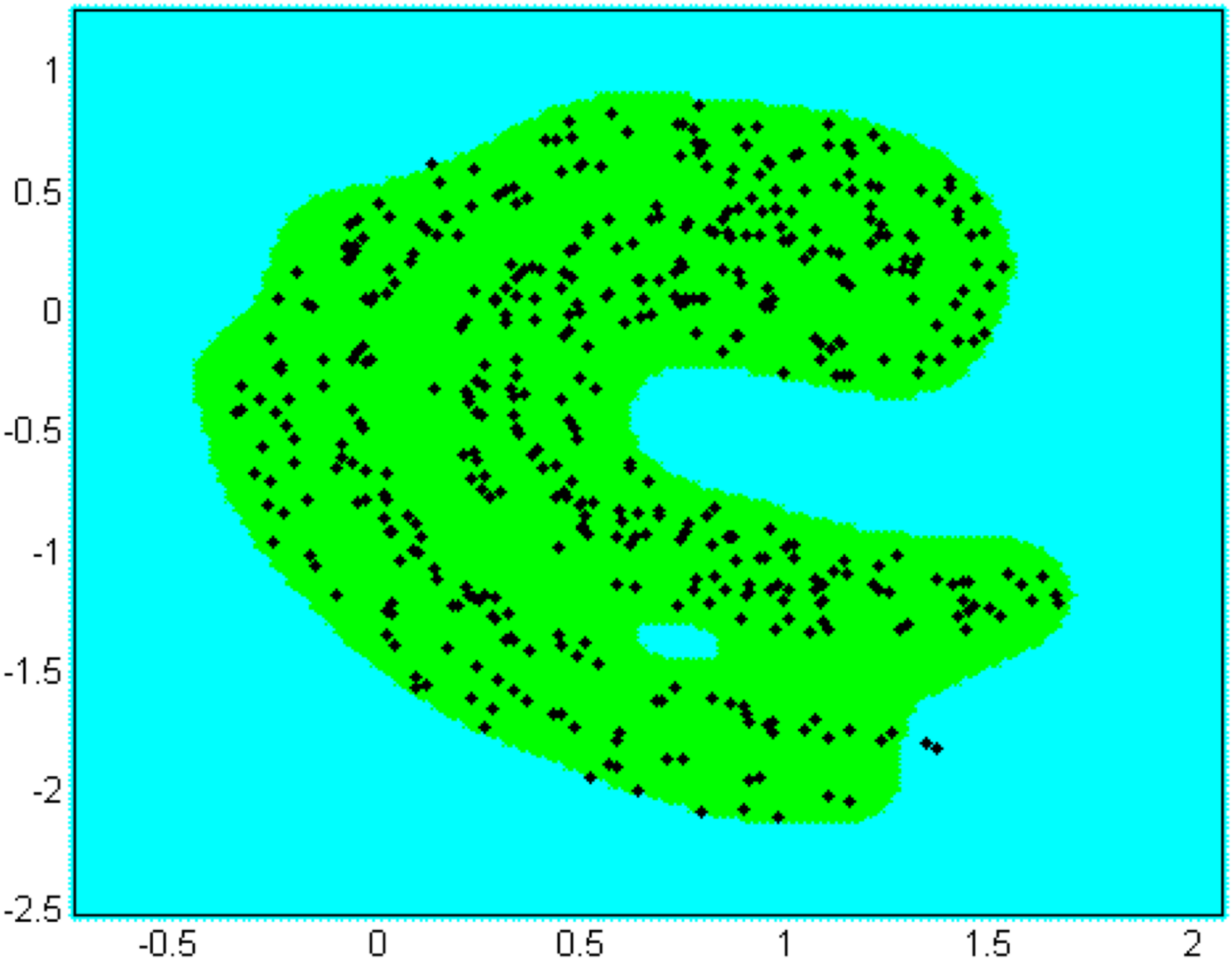
```
% Initialize
t=0.4-6/(1+x0^2+y0^2);
x(1,1)=1+mu*(x0*cos(t)-y0*sin(t));
y(1,1)=mu*(x0*sin(t)+y0*cos(t));

% Simulate
for i=2:n
    t=0.4-6/(1+x(i-1,1)^2+y(i-1,1)^2);
    x(i,1)=1+mu*(x(i-1,1)*cos(t)-y(i-1,1)*sin(t));
    y(i,1)=mu*(x(i-1,1)*sin(t)+y(i-1,1)*cos(t));
end

% Add normal white noise
x=x+randn(n,1)*level*std(x);
y=y+randn(n,1)*level*std(y);
```


ikeda time series
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-0.5
0
0.5
1
1.5
2

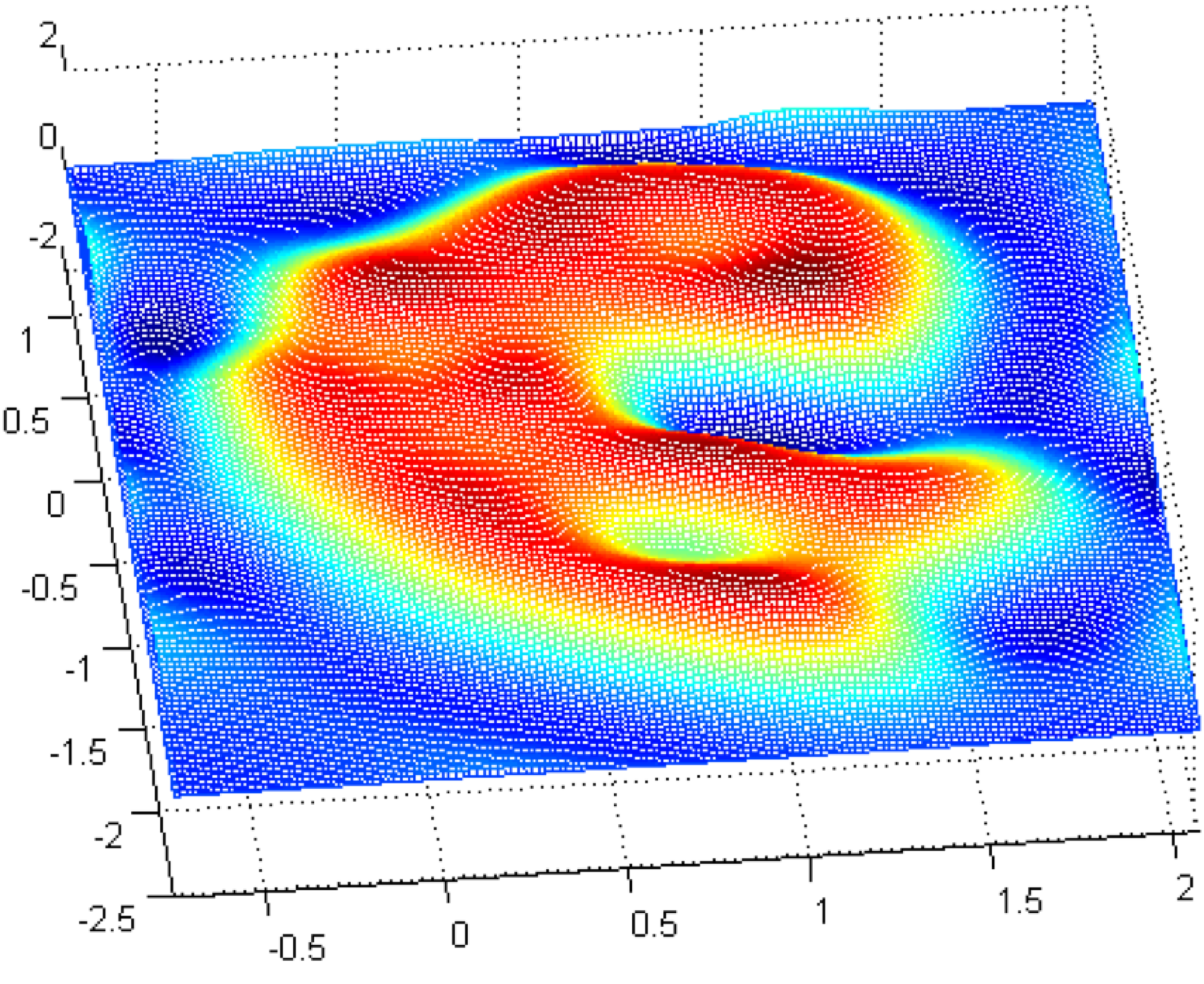
Learning neural functions for prediction

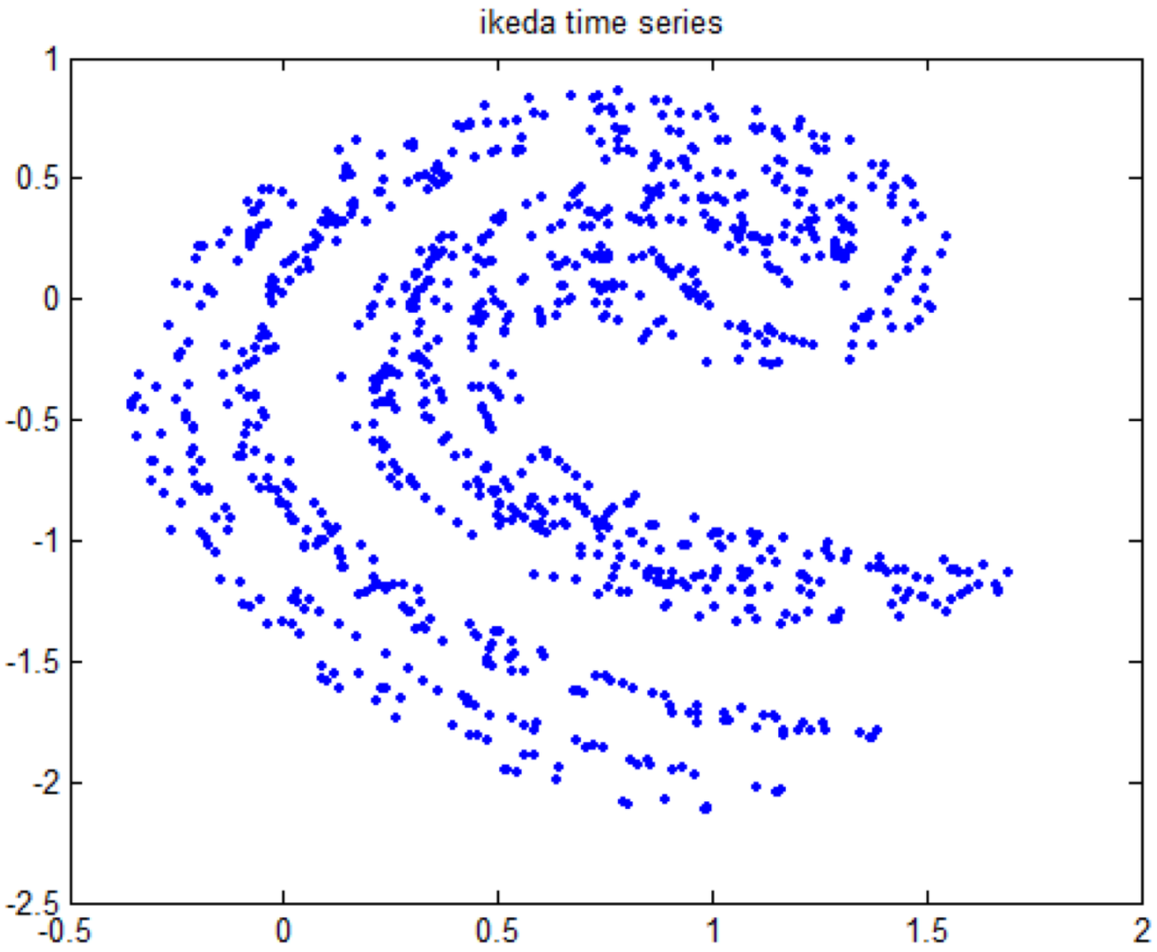
Given n=2000 points generated by Ikeda map, learn mapping from x_n to x_{n+1} and predict n=2001~3000

Density support



Support indictor function





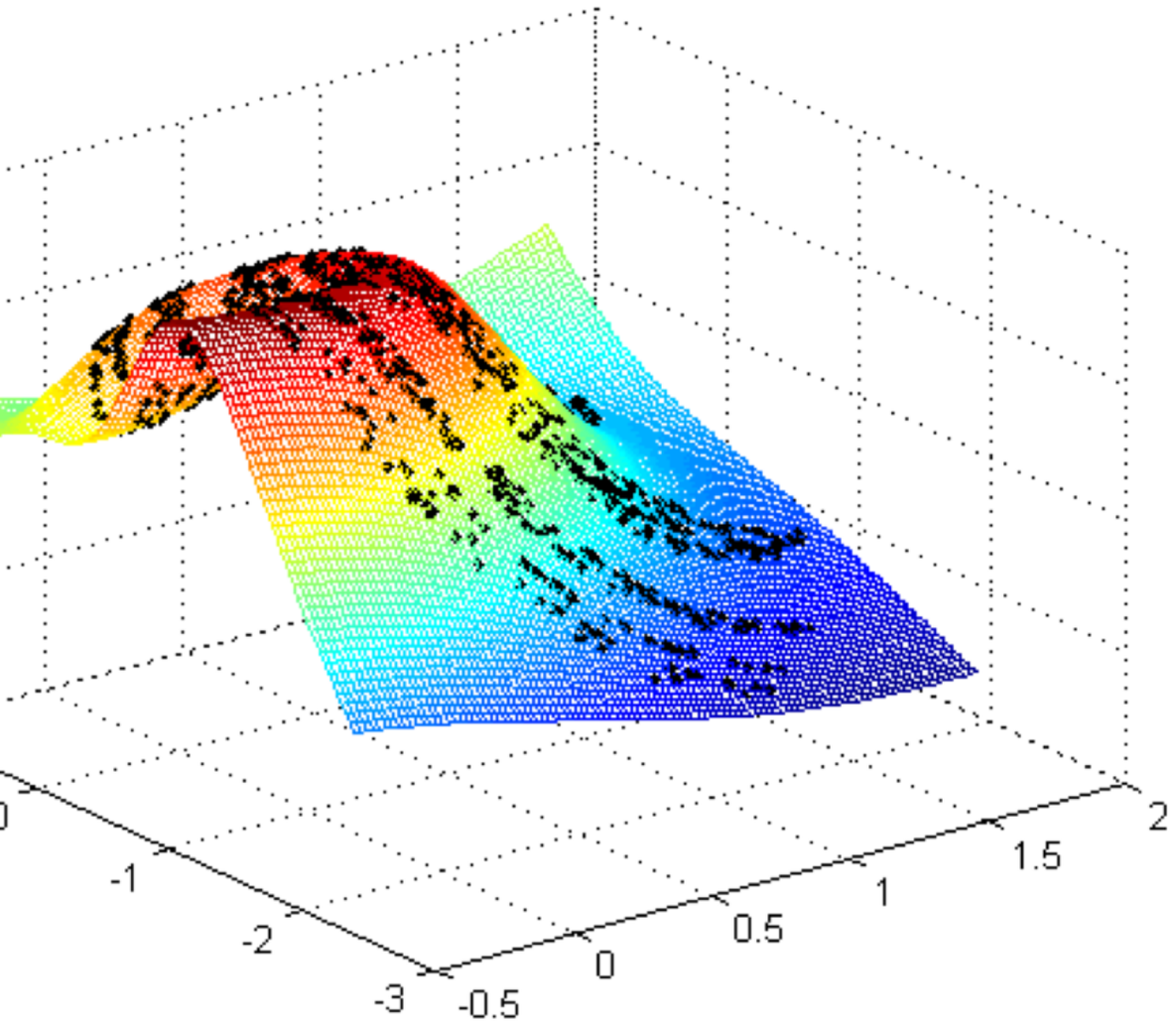
$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \cdots \rightarrow \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} \rightarrow \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Given time series, learning is to optimize adaptive parameters for approximating the underlying map

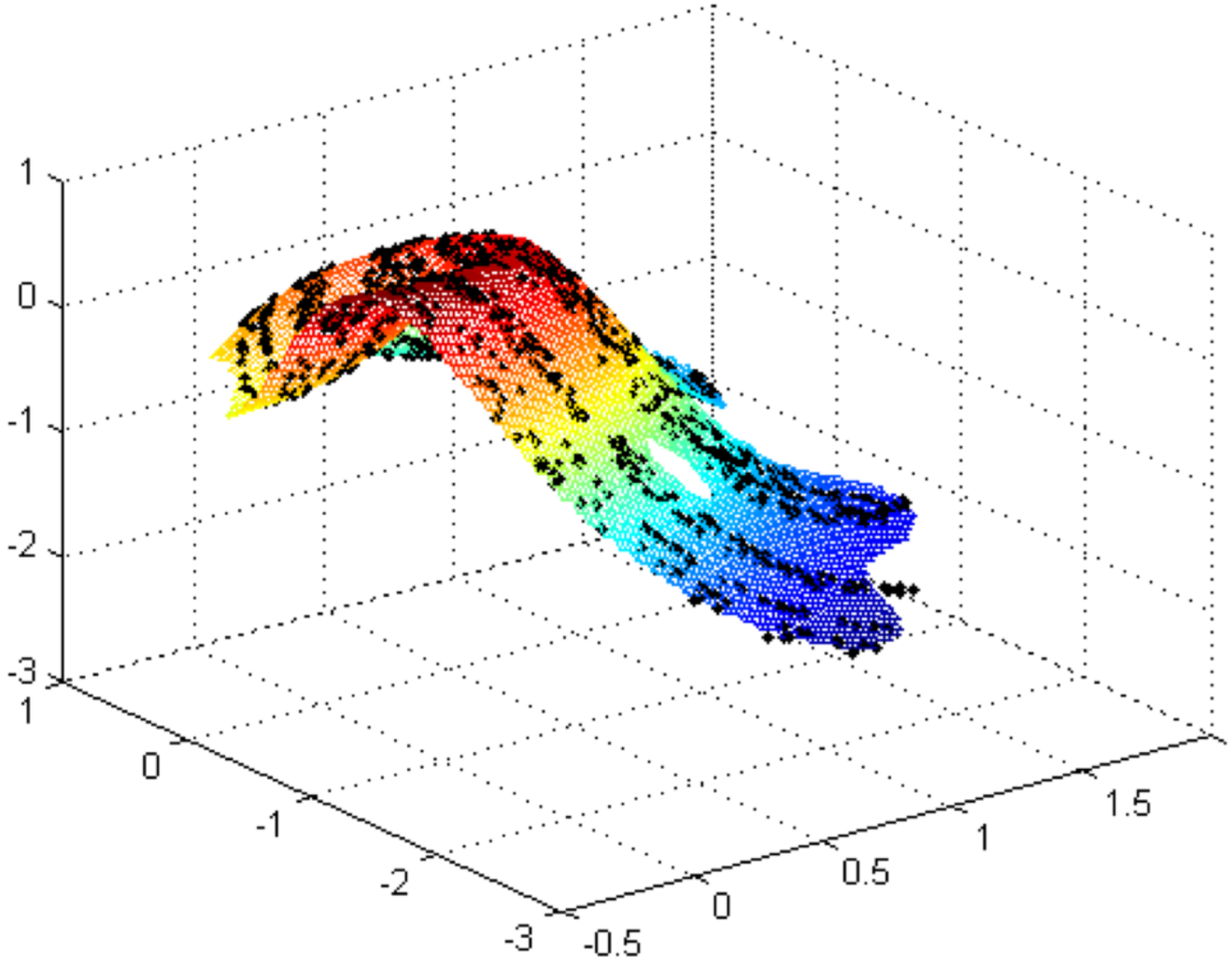
$$\hat{x} = F(x, y \mid \theta_1)$$

$$\hat{y} = F(x, y \mid \theta_2)$$

F is a neural function and θ denotes adaptive parameters



$f_2(x_1, x_2)$



$F_2(x_1, x_2 \mid \theta_2)$

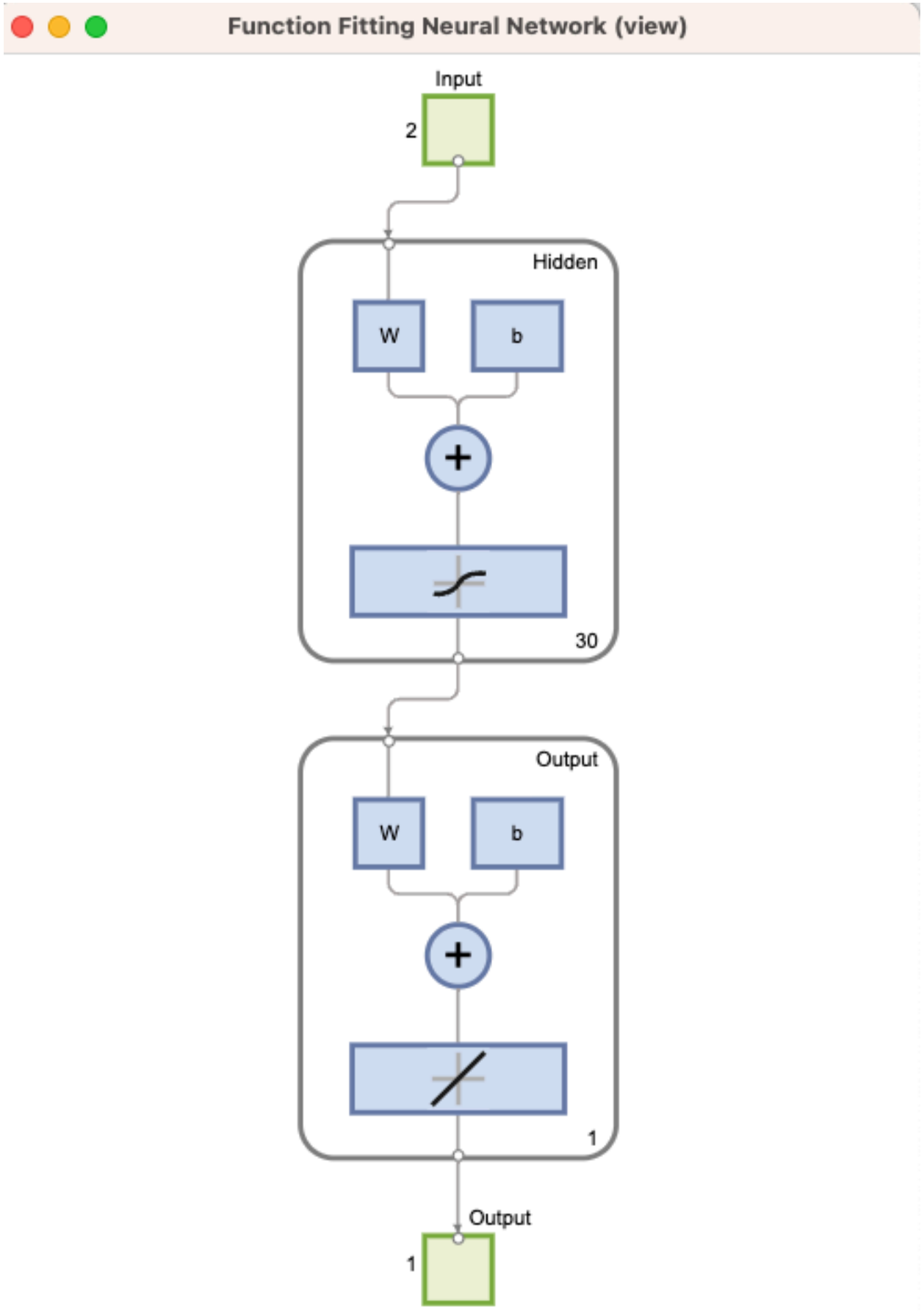
MSE : Training error : 8.5348e-008

Testing error : 6.3944e-007

Prediction

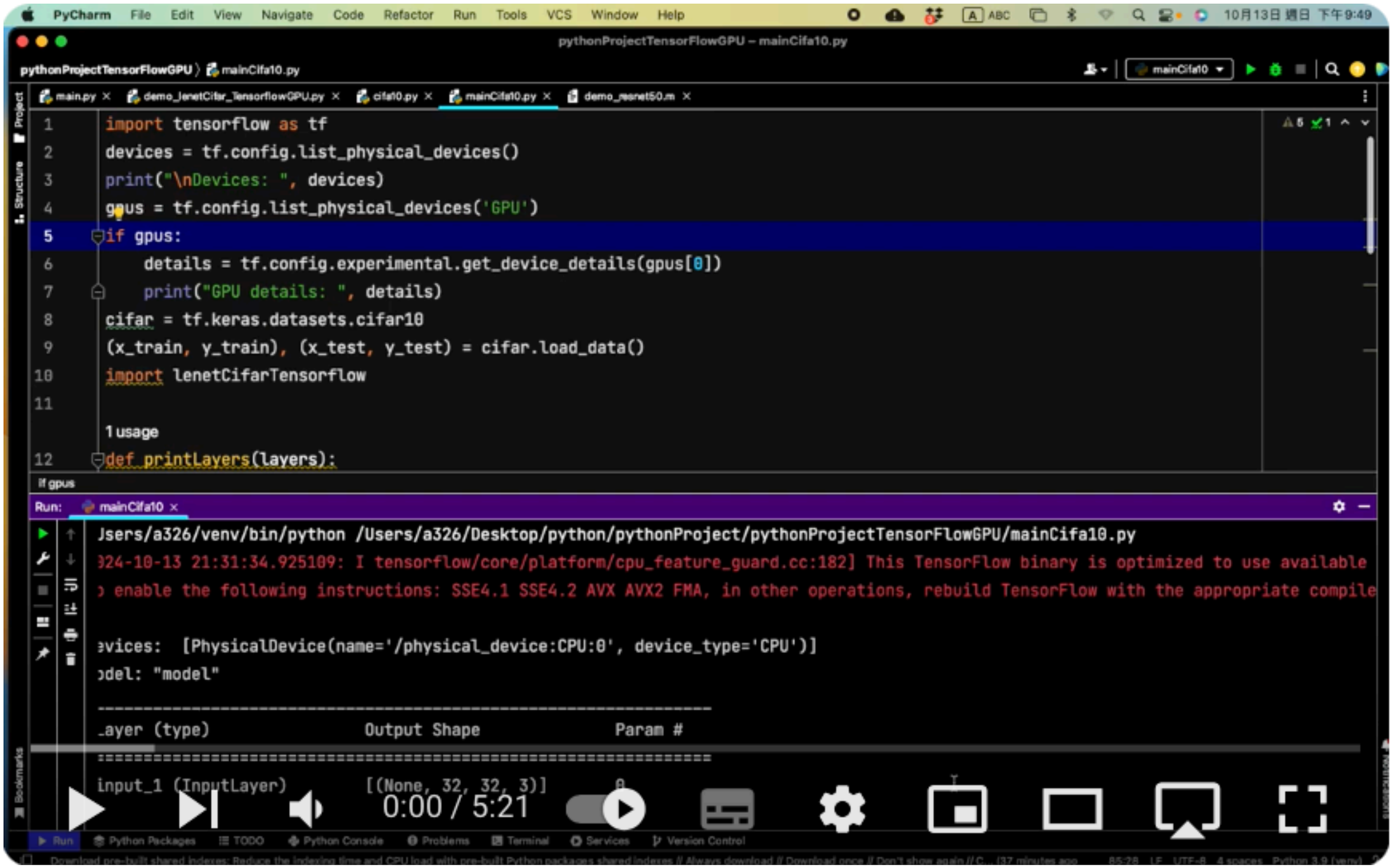
- $z_n = [x_n, y_n]$
- Find $x_{n+1} = f(z_n \mid \theta_x)$
- Find $y_{n+1} = f(z_n \mid \theta_y)$

```
>> z = [x(1:2000) y(1:2000)];
>> target = x(2:2001);
>> demo_iketa_x

performance =

    1.8161e-05
```


Function Fitting Neural Network (view)
Input
2
Hidden
W
b
+
30
Output
W
b
+
1
Output
1

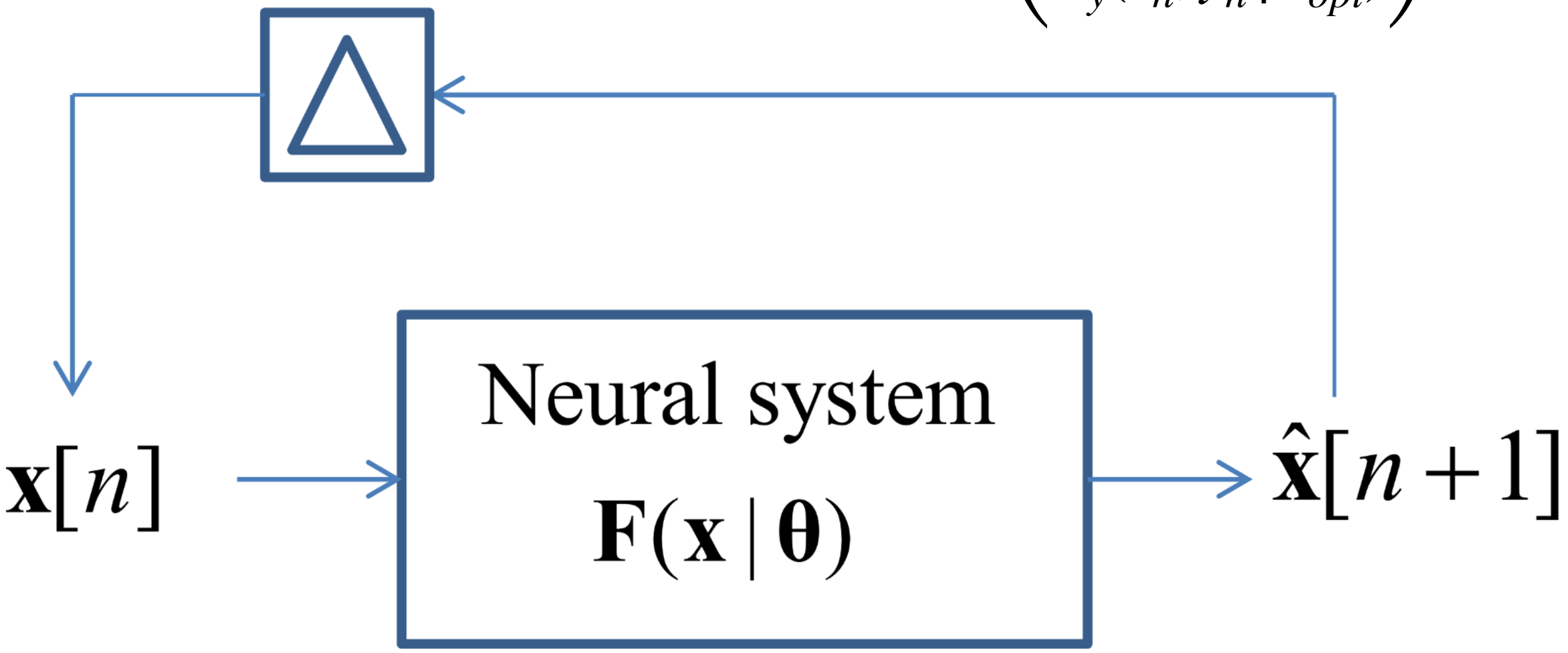
```
x = z';
t = target’;
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 30;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)
```

demo_iketa_x


PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help
10月13日 週日 下午9:49
pythonProjectTensorFlowGPU – mainCifa10.py
pythonProjectTensorFlowGPU 〉 mainCifa10.py
mainCifa10
main.py
demo_lenetCifar_TensorflowGPU.py
cifa10.py
mainCifa10.py
demo_resnet50.m
Project
Structure
1 import tensorflow as tf
2 devices = tf.config.list_physical_devices()
3 print("\nDevices: ", devices)
4 gpus = tf.config.list_physical_devices('GPU')
5 if gpus:
6 details = tf.config.experimental.get_device_details(gpus[0])
7 print("GPU details: ", details)
8 cifar = tf.keras.datasets.cifar10
9 (x_train, y_train), (x_test, y_test) = cifar.load_data()
10 import lenetCifarTensorflow
11
1 usage
12 def printLayers(layers):
If gpus
Run: mainCifa10
Jsers/a326/venv/bin/python /Users/a326/Desktop/python/pythonProject/pythonProjectTensorFlowGPU/mainCifa10.py
)24-10-13 21:31:34.925109: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available
) enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compile
evices: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
odel: "model"
_ayer (type) Output Shape Param #
input_1 (InputLayer) [(None, 32, 32, 3)] 0
0:00 / 5:21
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services
Version Control
Bookmarks

$$\mathbf{x}[n] = \begin{pmatrix} x_n \\ y_n \end{pmatrix},$$

$$\mathbf{x}[n+1] = \begin{pmatrix} F_x(x_n, y_n \,|\, \theta_{opt}) \\ F_y(x_n, y_n \,|\, \theta_{opt}) \end{pmatrix}$$



$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \rightarrow \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \dots \rightarrow \begin{pmatrix} x_{n-1} \\ y_{n-1} \end{pmatrix} \rightarrow \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Learning previous data and Prediction of future

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} \rightarrow \begin{pmatrix} \hat{x}_{n+1} \\ \hat{y}_{n+1} \end{pmatrix} \dots \rightarrow \begin{pmatrix} \hat{x}_{2n-1} \\ \hat{y}_{2n-1} \end{pmatrix} \rightarrow \begin{pmatrix} \hat{x}_{2n} \\ \hat{y}_{2n} \end{pmatrix}$$

Step 1. Generate 2001 Ikeda data, x(1:2001) and y(1:2001)
Step2. Set z = [x(1:2000), y(1:2000)], target =x(2:2001).
Learning net_x for approximating $F_x(\mathbf{x}_n \mid \theta)$. Set net_x to net
Step3. Set z = [x(1:2000), y(1:2000)], target =y(2:2001).
Learning net_y for approximating $F_y(\mathbf{x}_n \mid \theta)$. Set net_y to net
Step4. Apply net_x and net_y to predict Ikeda data, x(2001:3000) and y(2001:3000)

Our first project : learning Ikea data and predicting future data

Lorentz Time series

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2022-I NA數值分析 ▸ Chaotic Systems Toolbox

Current Folder

Name

- AAFT.m
- Contents.m
- derivs.m
- gencorint.m
- henon.m
- IAAFT.m
- ikeda.m
- Knearest.m
- logistic.m
- lorentz.m
- mackeyglass.m
- noiseest.m
- noisergeo.m
- noiserSchreiber.m
- phaseran.m
- phasespace.m
- quadratic.m
- radnearest.m
- rossler.m
- shuffle.m
- SSA.m
- SSAeye.m
- SSAforeiter.m
- SSAinv.m
- vr.m

lorentz.m (Function)

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2022-I NA數值分析/Chaotic Systems Toolbox/lorentz.m

area.m | ikeda.m | Contents.m | Knearest.m | lorentz.m

```
1  function [x,y,z]=lorentz(n,level,s,r,b,x0,y0,z0,h)
2  %Syntax: [x,y,z]=lorentz(n,level,s,r,b,x0,y0,z0,h)
3  %_____
4  %
5  % Simulation of the Lorent
6  %    dx/dt=s*(y-x)
7  %    dy/dt=r*x-y-xz
8  %    dz/dt=x*y-b*z;
9  %
10 % x, y, and z are the simu
11 % n is the number of the s
12 % level is the noise stand
13 %   noise-free time series
14 % s, r, b, and s are the p
15 % x0 is the initial value
16 % y0 is the initial value
17 % z0 is the initial value
18 % h is the step size.
```

Command Window

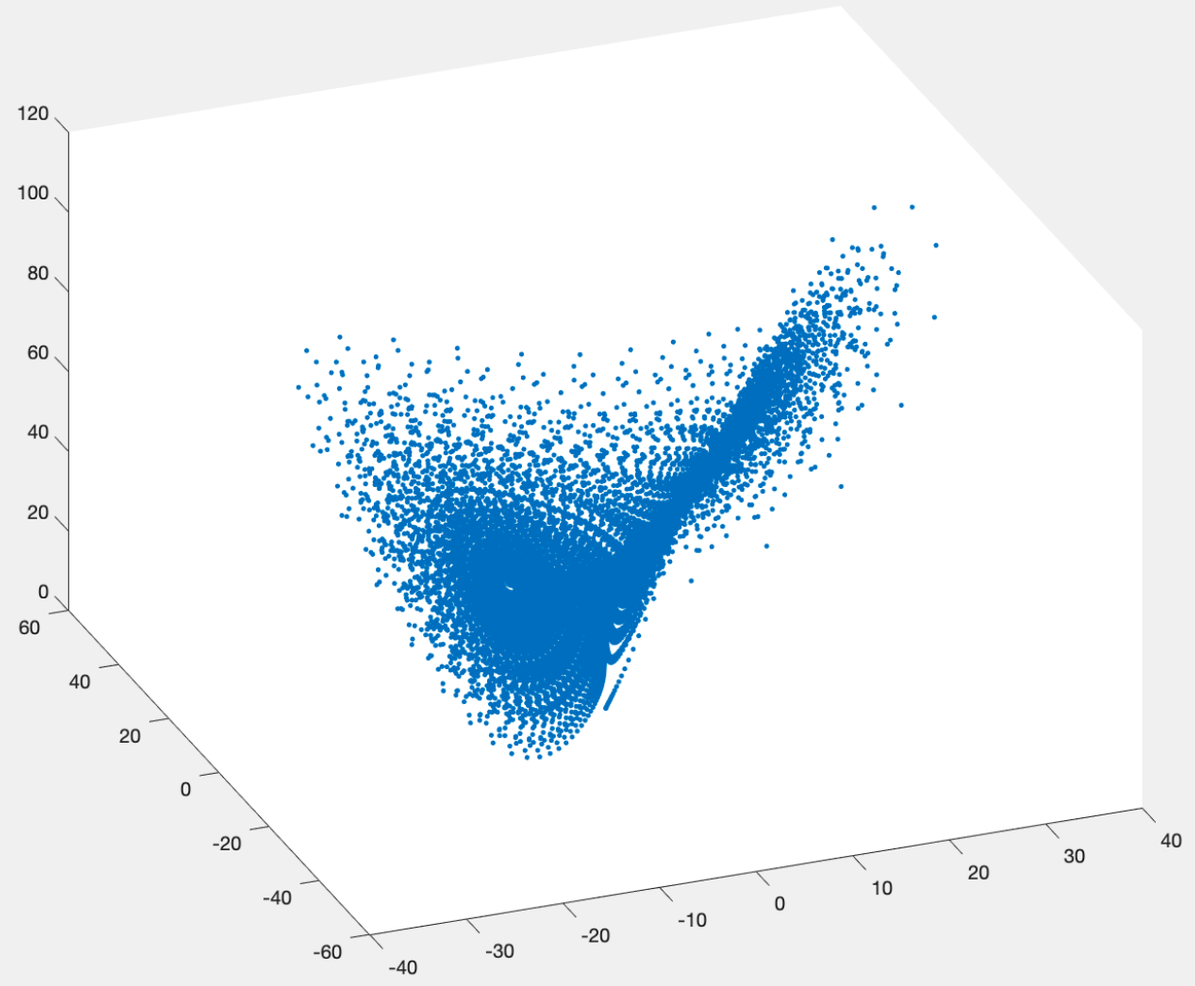
```
>> [x,y,z]=lorentz(20000);
>> plot3(x,y,z,'.')
>>
```

Workspace

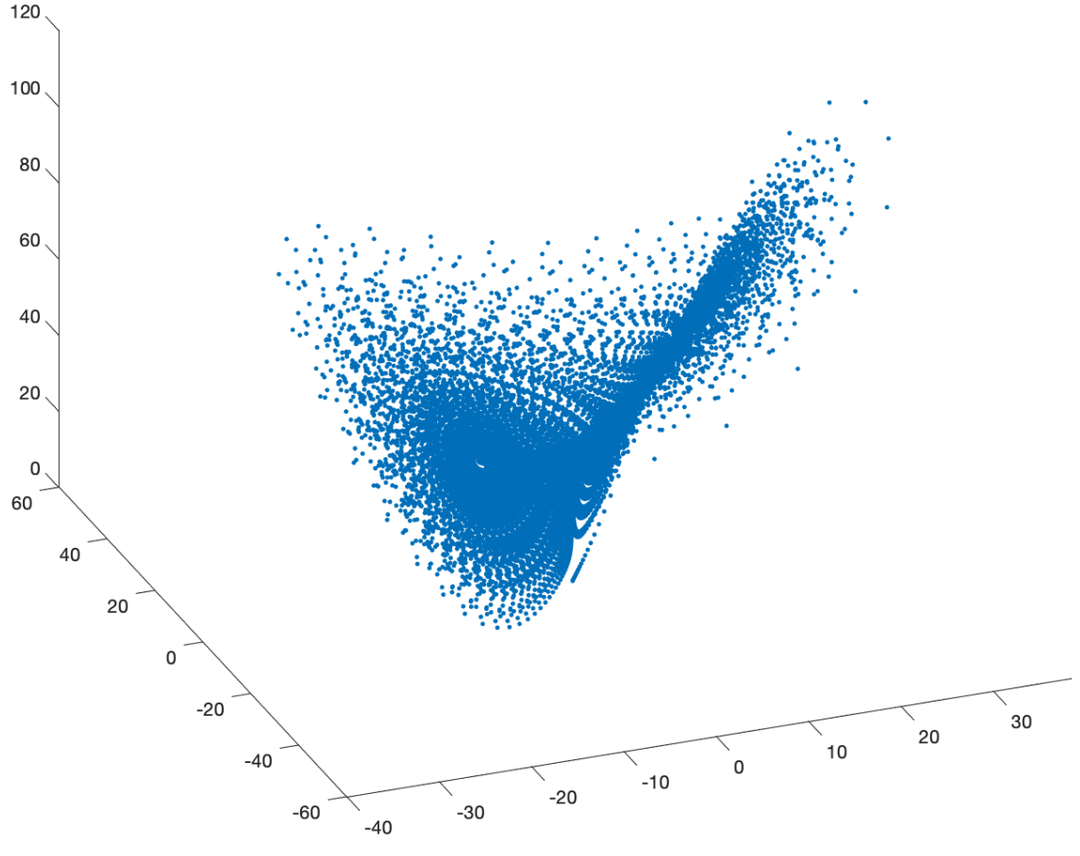
Name	Value
x	20000
y	20000

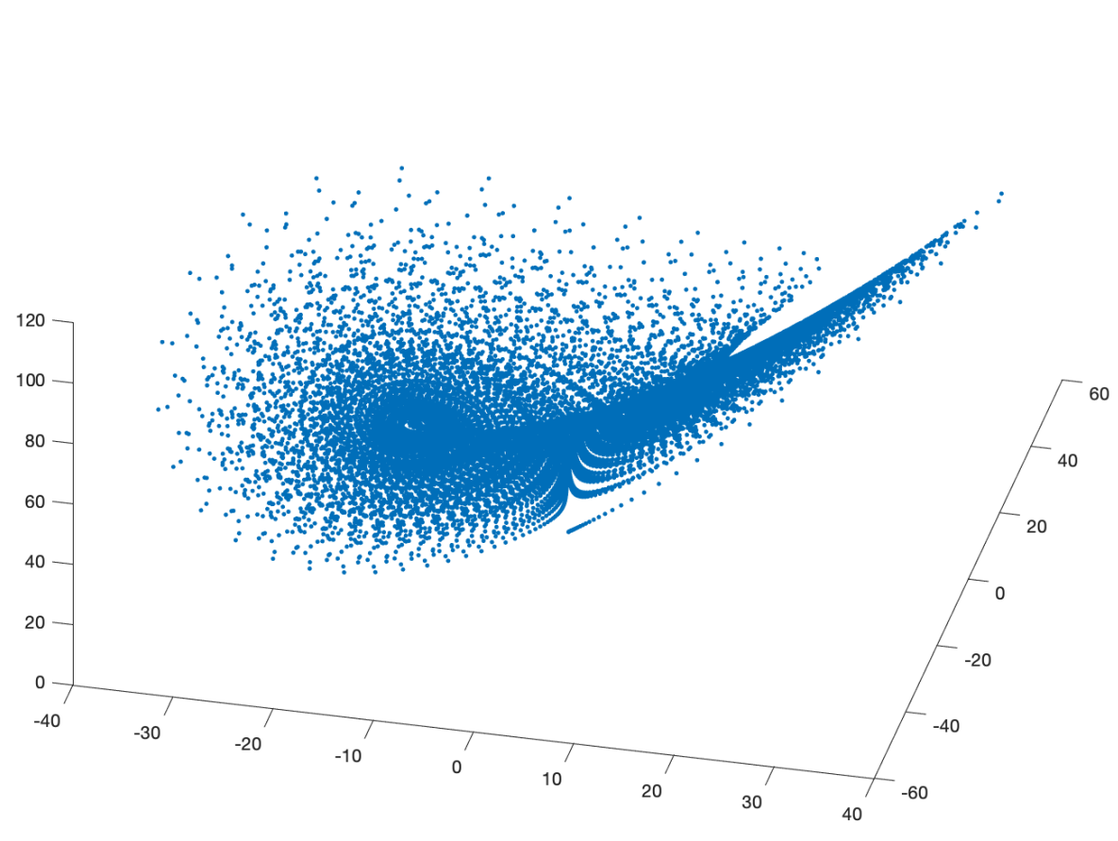
Figure 1

File Edit View Insert Tools Desktop Window Help



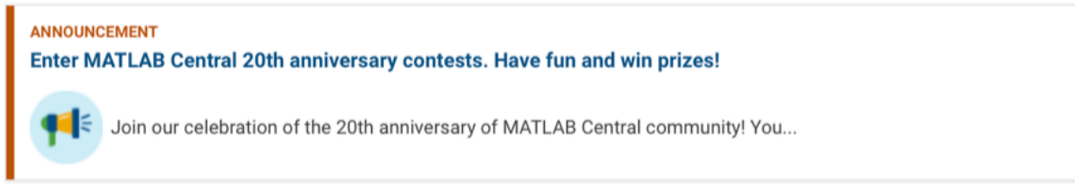
Density support Lorentz time series approximation and prediction

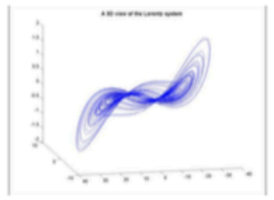




Project : Ikeda data generation and approximation for prediction

Step-wise procedure



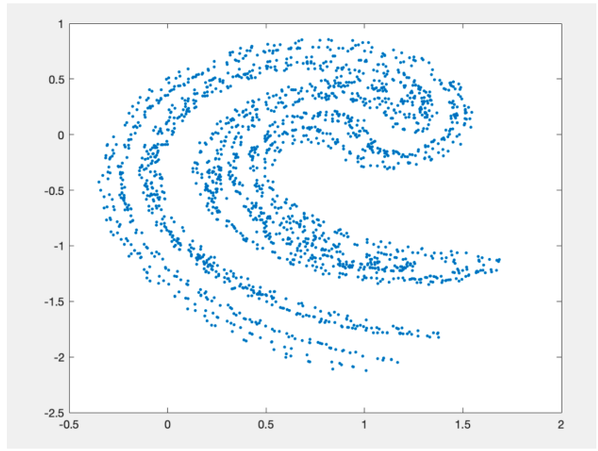


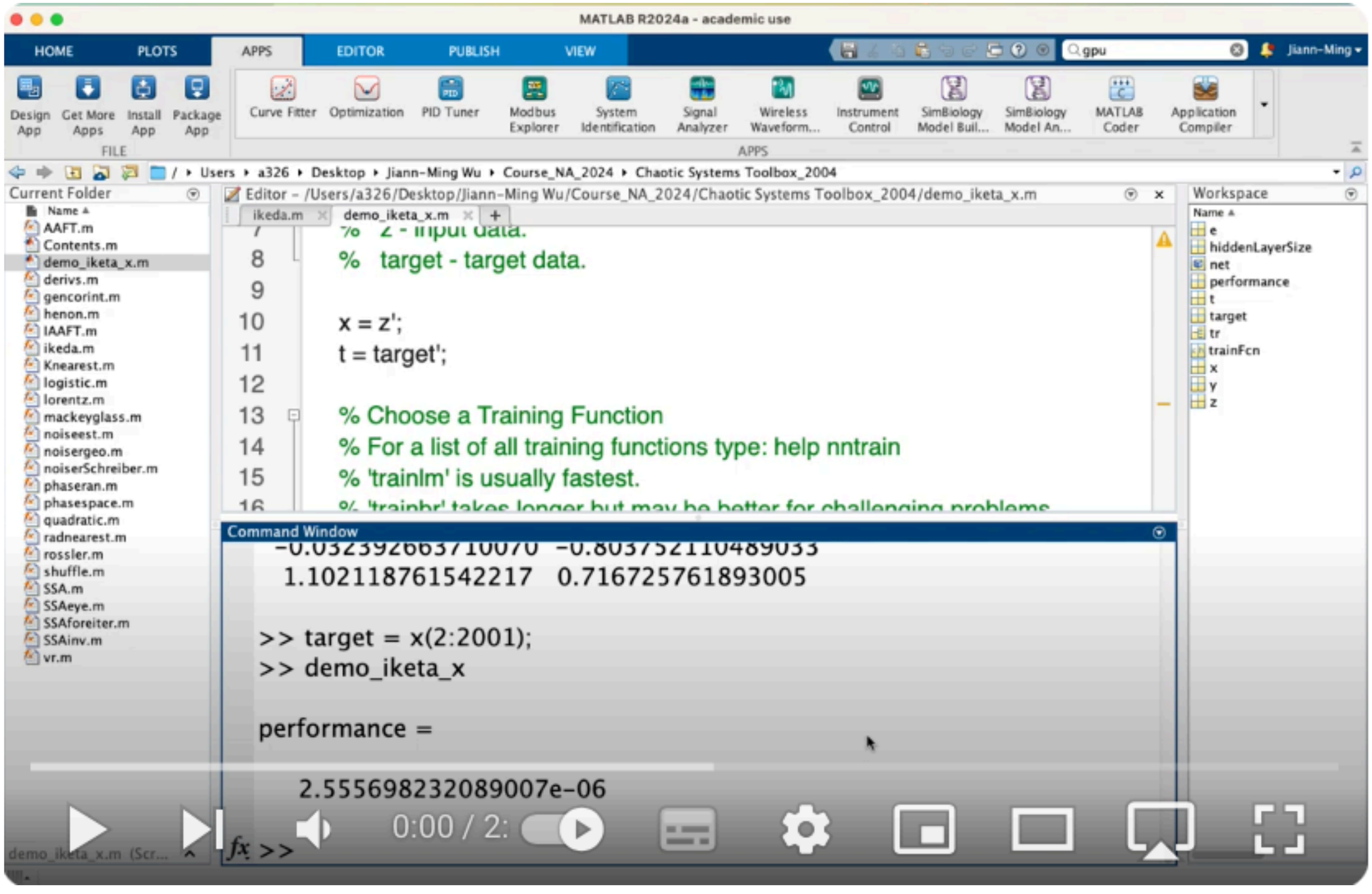
Chaotic Systems Toolbox

version 1.0.0.0 (26.3 KB) by Alexandros Leontitsis

Analysis of chaotic systems.

1. Download chaotic system toolbox
2. Generate Ikeda data for n=3000 and plot all data (page 14)
3. Install and test neural fit toolbox
4. Prepare training and testing data for approximation (page 22)




MATLAB R2024a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
gpu
Jiann-Ming
Design App
Get More Apps
Install App
Package App
FILE
Curve Fitter
Optimization
PID Tuner
Modbus Explorer
System Identification
Signal Analyzer
Wireless Waveform...
Instrument Control
SimBiology Model Buil...
SimBiology Model An...
MATLAB Coder
Application Compiler
APPS
/ ▸ Users ▸ a326 ▸ Desktop ▸ Jiann-Ming Wu ▸ Course_NA_2024 ▸ Chaotic Systems Toolbox_2004
Current Folder
Name
AAFT.m
Contents.m
demo_iketa_x.m
derivs.m
gencorint.m
henon.m
IAAFT.m
ikeda.m
Knearest.m
logistic.m
lorentz.m
mackeyglass.m
noiseest.m
noisergeo.m
noiserSchreiber.m
phaseran.m
phasespace.m
quadratic.m
radnearest.m
rossler.m
shuffle.m
SSA.m
SSAeye.m
SSAforeiter.m
SSAinv.m
vr.m
Editor - /Users/a326/Desktop/Jiann-Ming Wu/Course_NA_2024/Chaotic Systems Toolbox_2004/demo_iketa_x.m
ikeda.m
demo_iketa_x.m
% z - input data.
% target - target data.
x = z';
t = target';
% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems
Workspace
Name
e
hiddenLayerSize
net
performance
t
target
tr
trainFcn
x
y
z
Command Window
-0.032392663710070 -0.803752110489033
1.102118761542217 0.716725761893005
>> target = x(2:2001);
>> demo_iketa_x
performance =
2.555698232089007e-06
fx >>
0:00 / 2:
demo_iketa_x.m (Scr...

- Train a neural function for predicting x value and export the neural function F_x
- Train a neural function for predicting y value and export the neural function F_y
- Apply exported F_x and F_y to generate Ikeda data 2001-3000, and plot the generated data

Report due to 10.29

- Title
- Abstract
- Author
- Problem statement
- Methods and data
- Numerical simulations
- Conclusions
- References

My determinat I

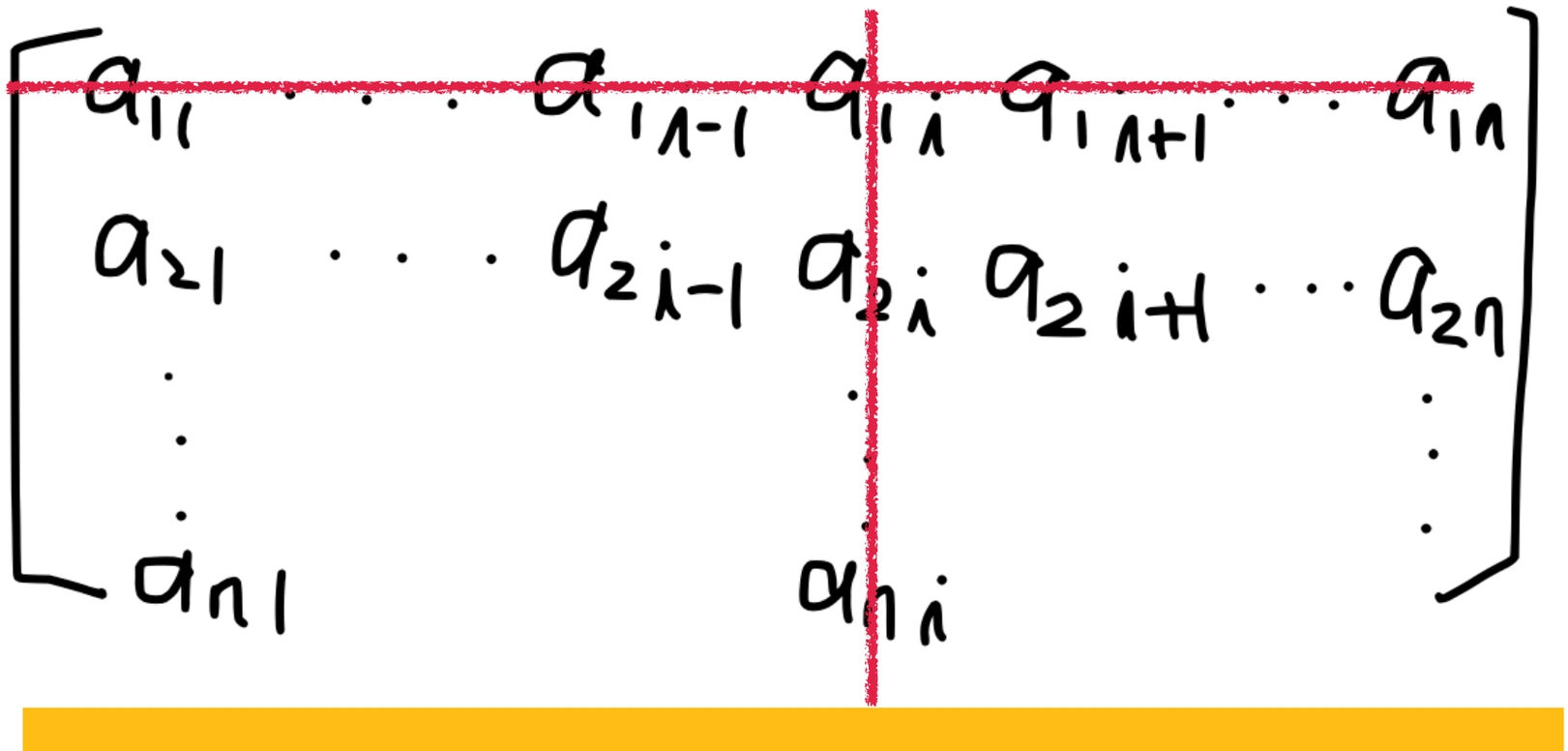
Matlab det

Recurrent relation of Laplacian expansion

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$det(A) = ?$$

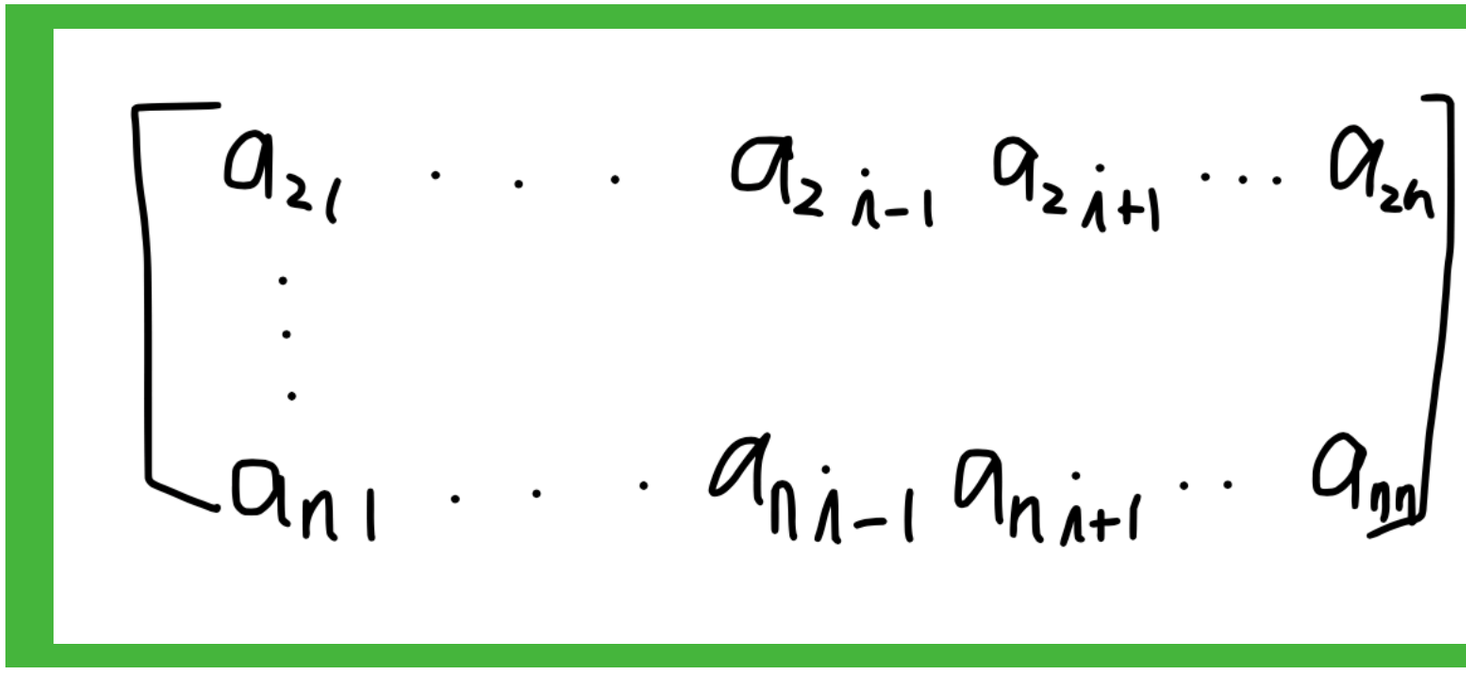
n=100



$n \times n$

Deleting row 1 and column i

$$A_{1i} =$$



$(n-1) \times (n-1)$

$$A_{1i} =$$

$$\begin{bmatrix} a_{21} & \cdots & a_{2\,i-1} & a_{2\,i+1} & \cdots & a_{2n} \\ \vdots & & & & & \\ a_{n1} & \cdots & a_{n\,i-1} & a_{n\,i+1} & \cdots & a_{nn} \end{bmatrix}$$

$(n-1) \times (n-1)$

1. A subtask that finds determinant of sub-matrix A_{1i}.
2. n subtasks that find determinants of A_{1i} for all i.
3. Combine answers of n subtasks to determine det(A)

I = 1…n

```
S = randperm(99);
num_A = S(1:52);
x = num_A(randi(52));
fprintf("x is %d\n",x)
num_B = sort(num_A);

L = 1;
R = length(num_B);
count = 1;
display(num_B);
fprintf("init:          L : %2d, R : %2d\n",L,R)
k = 0;
while L <= R
    [redacted]
    fprintf("loop:%d, M : %2d, L : %2d, R : %2d, \n",count,M,L,R)
    count = count + 1;
end
fprintf("The number at %2d is %2d\n",k,num_B(k))
```

Recurrent relation of Laplacian expansion

- det(A) is decomposed to n sub-tasks
- Each calculates determinant of an (n-1)-by-(n-1) matrix
- The problem size is reduced from n to n-1

$$det(A) = \sum_{i=1}^{n} (-1)^{i+1} a_{1i} \det(A_{1i})$$

$n \times n$

```
>> A = [ 2 3 1;4 1 2;1 2 1]

A =

     2     3     1
     4     1     2
     1     2     1
```

```
>> A1 = [A(2:n,1:i-1) A(2:n,i+1:n)]
```

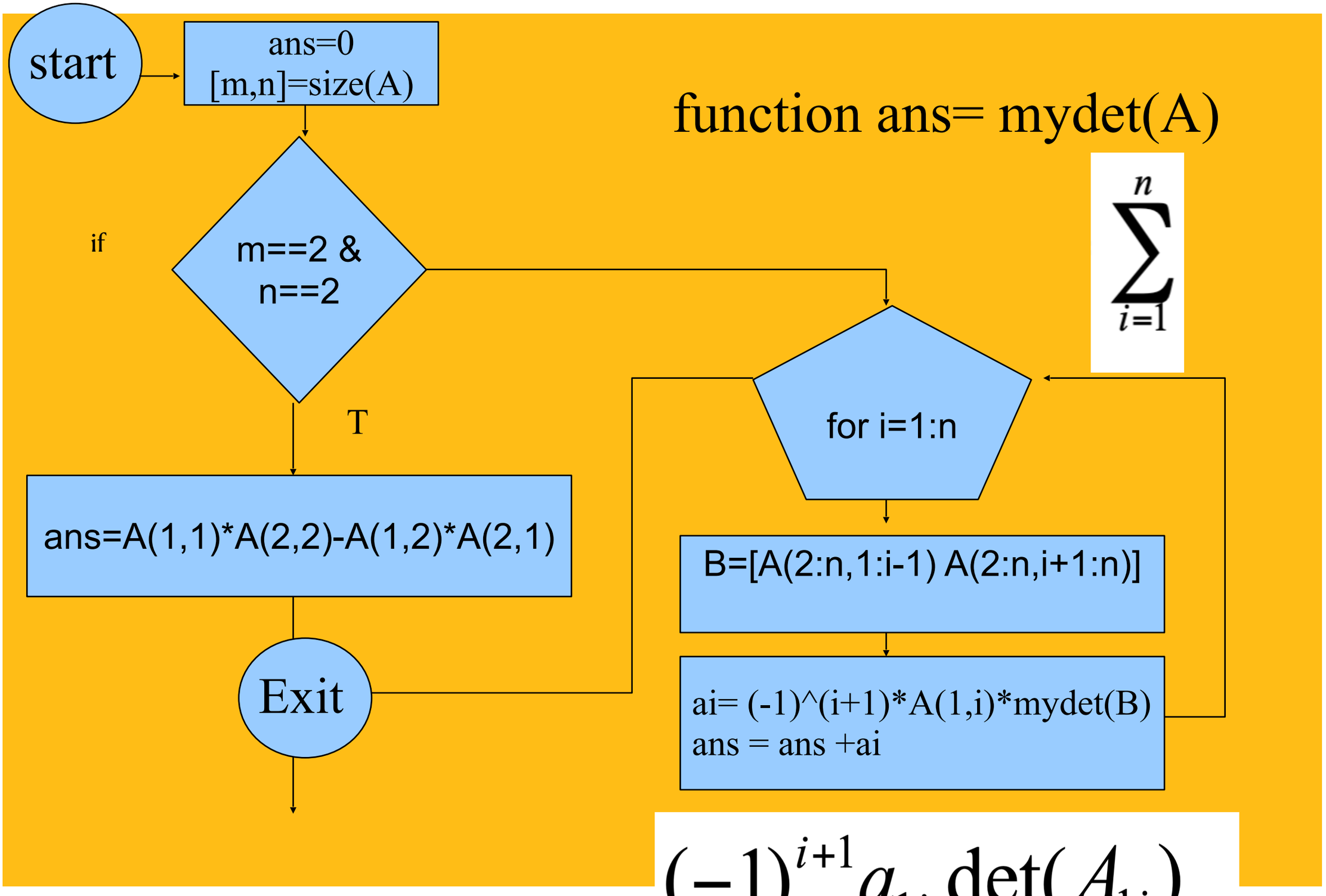
```
>> i = 1;

A1 =

     1     2
     2     1
```

```
>> i = 2;

A2 =

     4     2
     1     1
```

```
A3 =

     4     1
     1     2
```

```
>> A(1,1)*det(A1) -A(1,2)* det(A2) +A(1,3)* det(A3)
```

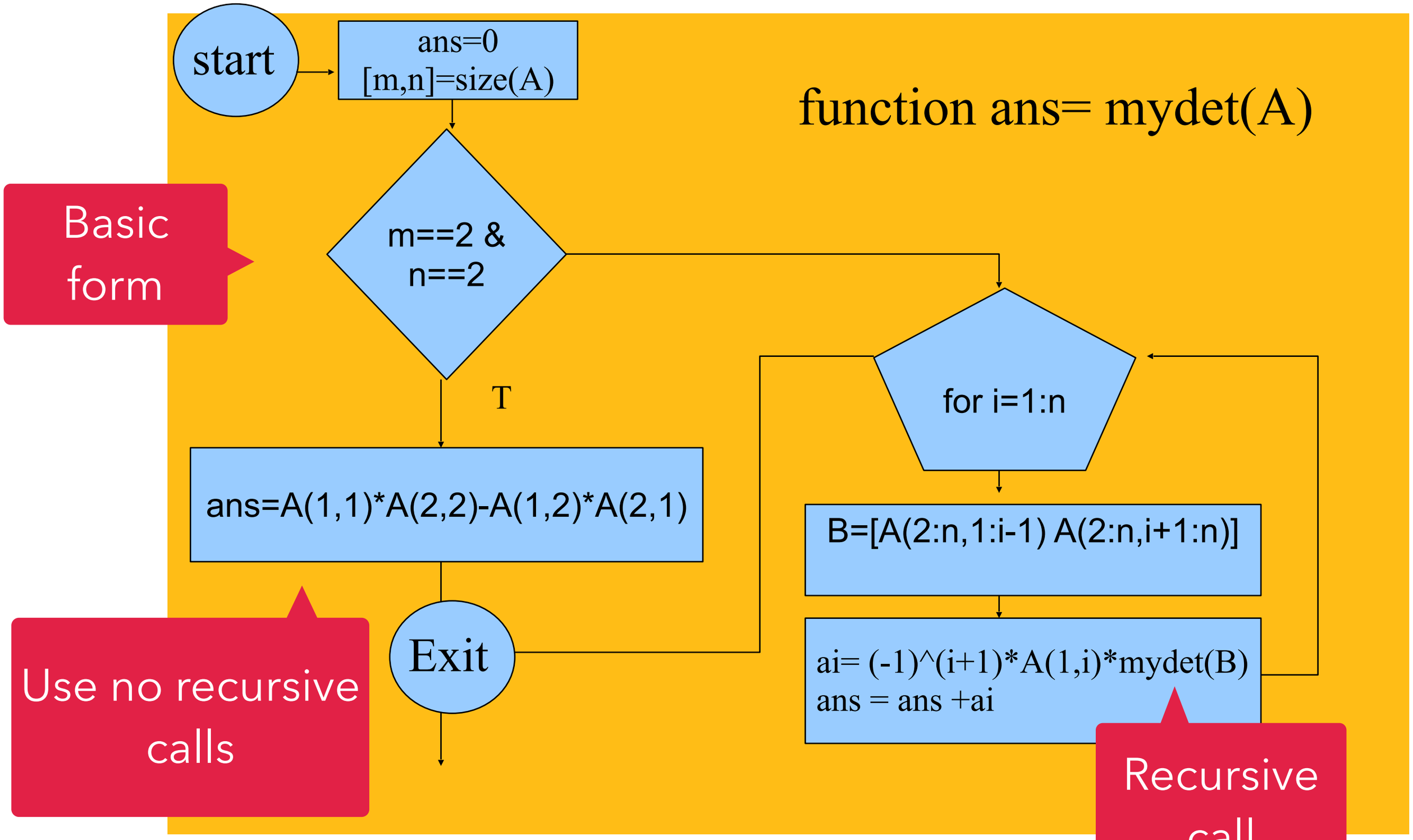
Recursive programming based on Laplacian expansion



$$\sum_{i=1}^{n}$$

$$(-1)^{i+1} a_{1i} \det(A_{1i})$$

Recursive programming based on Laplacian expansion



Recursive call in my_det is a statement that calls my_det for calculating determinant of a sub-matrix.

How to implement the flow chart? Drawback? Advantages?

Implementation of det

Laplacian exapansion

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2023-I NA數值分析 ▸

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/demo_mydetNew.m

mydetNew.m | demo_mydetNew.m | +

```
1   function demo_mydetNew()
2       A = randi(10,5,5)
3       ans = mydetNew(A);
4       ansMatlab = det(A);
5       display(ans)
6       display(abs(ans - ansMatlab))
```

- Computational complexity, O(n!), non-polynomial
- Time consuming
- Memory consuming
- If n >10, it results in intolerant computing time to evaluate determinant by recursive programming.
- An improvement by Bareiss's standard fraction free Gaussian elimination

n: problem size

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$$

PDF	$\det(2\pi\boldsymbol{\Sigma})^{-\frac{1}{2}}\ \exp(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})),$ exists only when $\boldsymbol{\Sigma}$ is positive-definite

$$Pr(x) \propto exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

$$\int Pr(x)dx = 1$$

Bareiss's standard fraction free Gaussian elimination

My Determinant II

Bareiss's standard fraction free Gaussian elimination

$$
\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}, \quad \text{for } 1 \le k < n, k < i, j \le m.
\end{aligned}
$$

It is well known that

$$
A_{i,j}^{(k)} = \begin{vmatrix} A_{1,1} & \cdots & A_{1,k} & A_{1,j} \\ \vdots & & \vdots & \vdots \\ A_{k,1} & \cdots & A_{k,k} & A_{k,j} \\ A_{i,1} & \cdots & A_{i,k} & A_{i,j} \end{vmatrix}.
$$

Thus when $m = n$, $\det(A) = A_{n,n}^{(n-1)}$, and when $A = \begin{pmatrix} M & b \\ I & 0 \end{pmatrix}$, for square

[N,M]=size(A);
a=zeros(N,N,N);

for k=1:N
for i=k+1:N
for j=k+1:N

for $1 \leq k < n, k < i, j \leq m.$

$$
\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \leq i \leq n, 1 \leq j \leq m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}},
\end{aligned}
$$

for k=1:N
for i=k+1:N
for j=k+1:N

$$
\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}},
\end{aligned}
$$

end
end
end

if k==1
a(i,j,k)=A(k,k)*A(i,j)-A(i,k)*A(k,j);
end

$$
\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}},
\end{aligned}
$$

if k==2
a(i,j,k)=(a(k,k,1)* a(i,j,1)-a(i,k,1)* a(k,j,1))/A(k-1,k-1);
end

if k>2
a(i,j,k)=(a(k,k,k-1)* a(i,j,k-1)-a(i,k,k-1)* a(k,j,k-1))/a(k-1,k-1,k-2);
end

```
[N,M]=size(A);
a=zeros(N,N,N);
for k=1:N
for i=k+1:N
for j=k+1:N
    if k==1
        a(i,j,k)=A(k,k)*A(i,j)-A(i,k)*A(k,j);
    end
    if k==2
       a(i,j,k)=(a(k,k,1)* a(i,j,1)-a(i,k,1)* a(k,j,1))/A(k-1,k-1);
    end
    if k>2
        a(i,j,k)=(a(k,k,k-1)* a(i,j,k-1)-a(i,k,k-1)* a(k,j,k-1))/a(k-1,k-1,k-2);
    end
end
end
end
a(N,N,N-1)
```

Bareiss' standard fraction free Gaussian elimination (Bareiss, 1968).

$$
\begin{aligned}
A_{0,0}^{(-1)} &= 1, \\
A_{i,j}^{(0)} &= A_{i,j}, \text{ for } 1 \le i \le n, 1 \le j \le m, \\
A_{i,j}^{(k)} &= \frac{A_{k,k}^{(k-1)} A_{i,j}^{(k-1)} - A_{i,k}^{(k-1)} A_{k,j}^{(k-1)}}{A_{k-1,k-1}^{(k-2)}}, \text{ for } 1 \le k < n, k < i, j \le m.
\end{aligned}
$$

It is well known that

$$
A_{i,j}^{(k)} = \begin{vmatrix} A_{1,1} & \cdots & A_{1,k} & A_{1,j} \\ \vdots & & \vdots & \vdots \\ A_{k,1} & \cdots & A_{k,k} & A_{k,j} \\ A_{i,1} & \cdots & A_{i,k} & A_{i,j} \end{vmatrix}.
$$

Thus when $m = n$, $\det(A) = A_{n,n}^{(n-1)}$, and when $A = \begin{pmatrix} M & b \\ I & 0 \end{pmatrix}$, for square

How to develop a novel algorithm for calculating determinant based on the two previous methods for further improvement?

Nonlinear Dimensionality Reduction

Dimensionality reduction : Locally Linear Embedding LLE

Laurens van der Maaten

Research scientist in machine learning and computer vision.

Email

Facebook

Github

Matlab Toolbox for Dimensionality Reduction

The Matlab Toolbox for Dimensionality Reduction contains Matlab implementations of 34 techniques for dimensionality reduction and metric learning. A large number of implementations was developed from scratch, whereas other implementations are improved versions of software that was already available on the Web. The implementations in the toolbox are conservative in their use of memory. The toolbox is available for download here.

Please note I am no longer actively maintaining this toolbox. Your mileage may vary!

Currently, the Matlab Toolbox for Dimensionality Reduction contains the following techniques:

1. Principal Component Analysis (PCA)
2. Probabilistic PCA
3. Factor Analysis (FA)
4. Classical multidimensional scaling (MDS)

```
% step 1. add all sub-directories
clear all
dataname="swiss";
[X,labels,t]=generate_data(dataname,5000,0);
plot3(X(:,1),X(:,2),X(:,3),'.')
no_dims = 2;
k = 12;
[mappedX, mapping] = lle(X, no_dims, k);
figure
plot(mappedX(:,1),mappedX(:,2),'.')
```

Number of neighbors

HOME | PLOTS | APPS | EDITOR | PUBLISH | VIEW

Search Documentation

New | Open | Save | Find Files | Compare | Print — FILE
Go To | Find — NAVIGATE
Insert | Comment | Indent — EDIT
Breakpoints — BREAKPOINTS
Run | Run and Advance | Run Section | Advance | Run and Time — RUN

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019_2020_2021 ▸ drtoolbox ▸

Current Folder

Name
- test_toolbox.m
- reconstruct_data.m
- Readme.txt
- prewhiten.m
- out_of_sample_est.m
- out_of_sample.m
- mexall.m
- intrinsic_dim.m
- generate_data.m
- drgui.m
- Contents.m
- compute_mapping.m
- techniques
- my_example
- gui

Editor - /Users/apple/Desktop/Jiann-Ming Wu/code2019_2020_2021/drtoolbox/my_example/demo_swiss_LLE.m

+8 | demo_sudoku.m | sudokuEngine.m | demo_newton2024.m | ikeda.m | demo_iketa_x.m | isomap.m | demo_swiss_LLE.m

```
1   % step 1. add all sub-directories
2
3   clear all
4   dataname="swiss";
5   [X,labels,t]=generate_data(dataname,5000,0);
6   plot3(X(:,1),X(:,2),X(:,3),'.')
7   no_dims = 2;
8   k = 12;
```

Command Window

```
>> addpath('techniques')
>>
```

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019_2020_2021 ▸ drtoolbox ▸

Current Folder

Name ▲

- gui
- my_example
- techniques
- compute_mapping.m
- Contents.m
- drgui.m
- generate_data.m
- intrinsic_dim.m
- mexall.m
- out_of_sample.m
- out_of_sample_e
- prewhiten.m
- Readme.txt
- reconstruct_data.
- test_toolbox.m

Editor - /Users/apple/Desktop/Jiann-Ming Wu/code2019_2020_

demo_swiss_LLE.m

```
1   % step 1. add all sub-directories
2   clear all
3   dataname="swiss";
4   [X,labels,t]=generate_data(dataname,
5   plot3(X(:,1),X(:,2),X(:,3),'.')
```

Figure 1

File Edit View Insert Tools Desktop Window Help

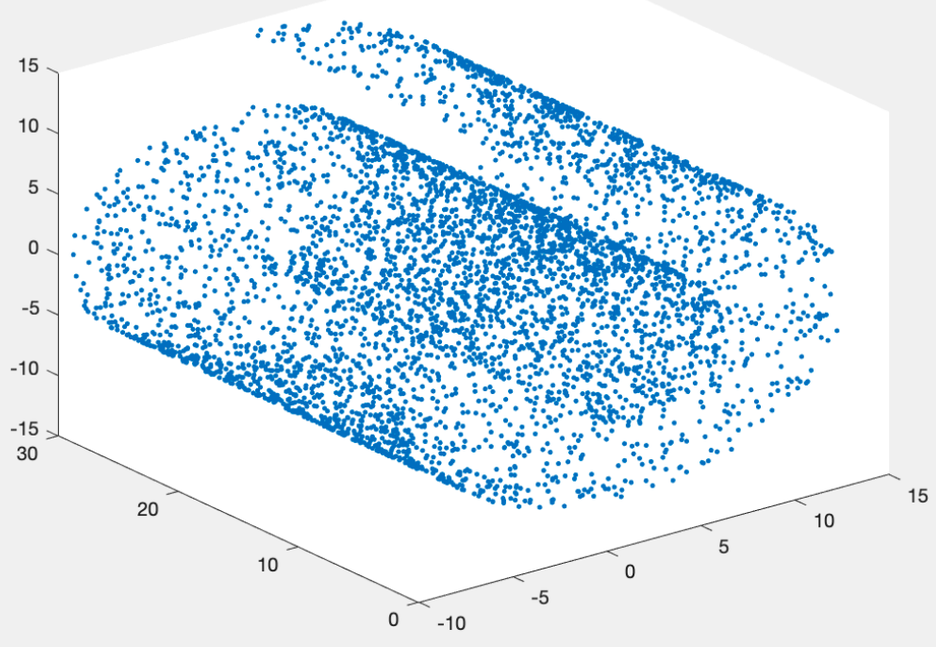
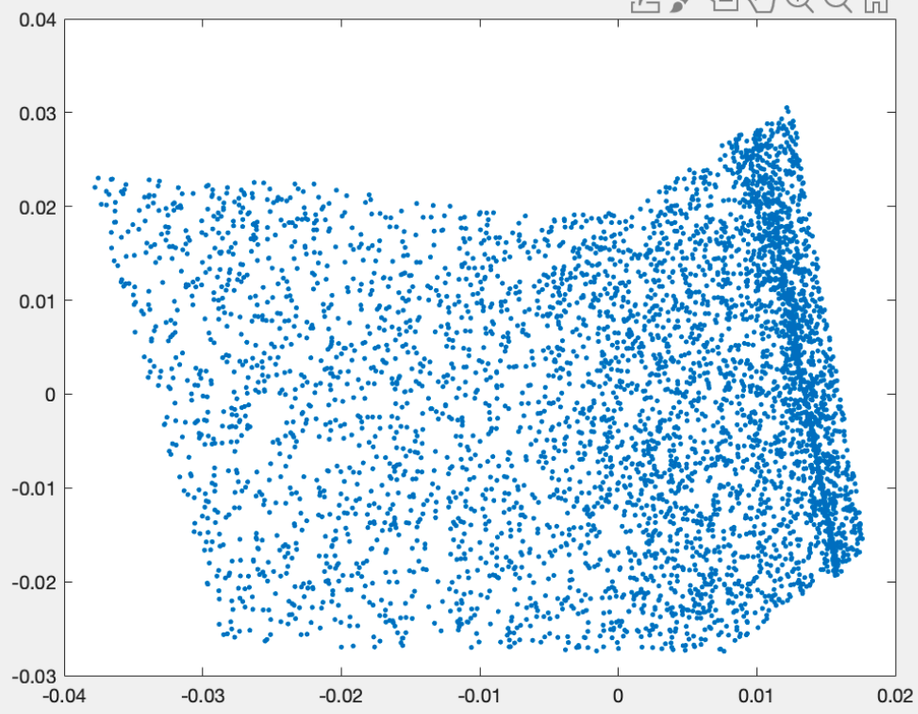


Figure 2

File Edit View Insert Tools Desktop Window Help

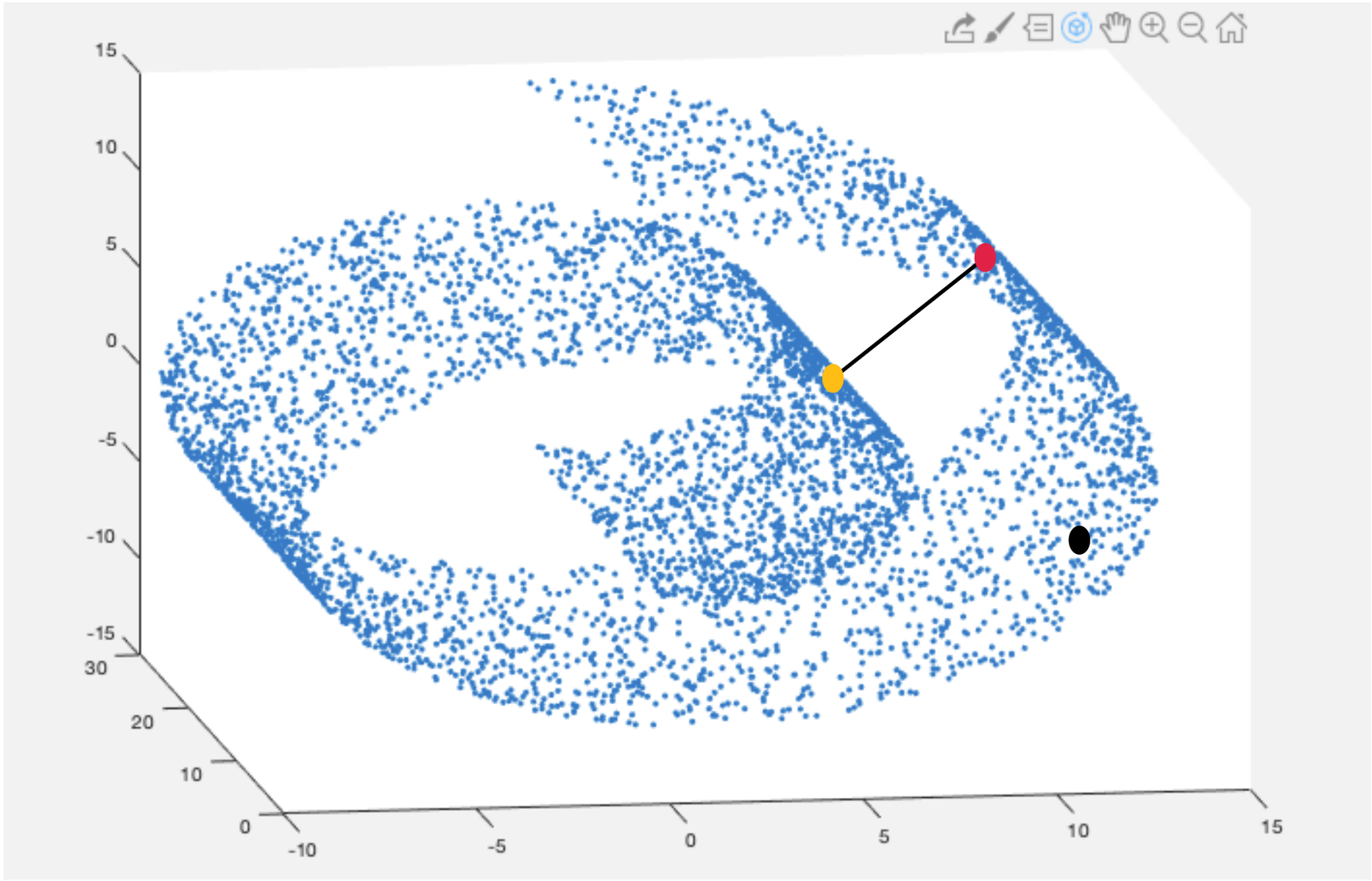


Value
"swiss
12
5000x
5000x
1x1 s
2
5000x
5000x

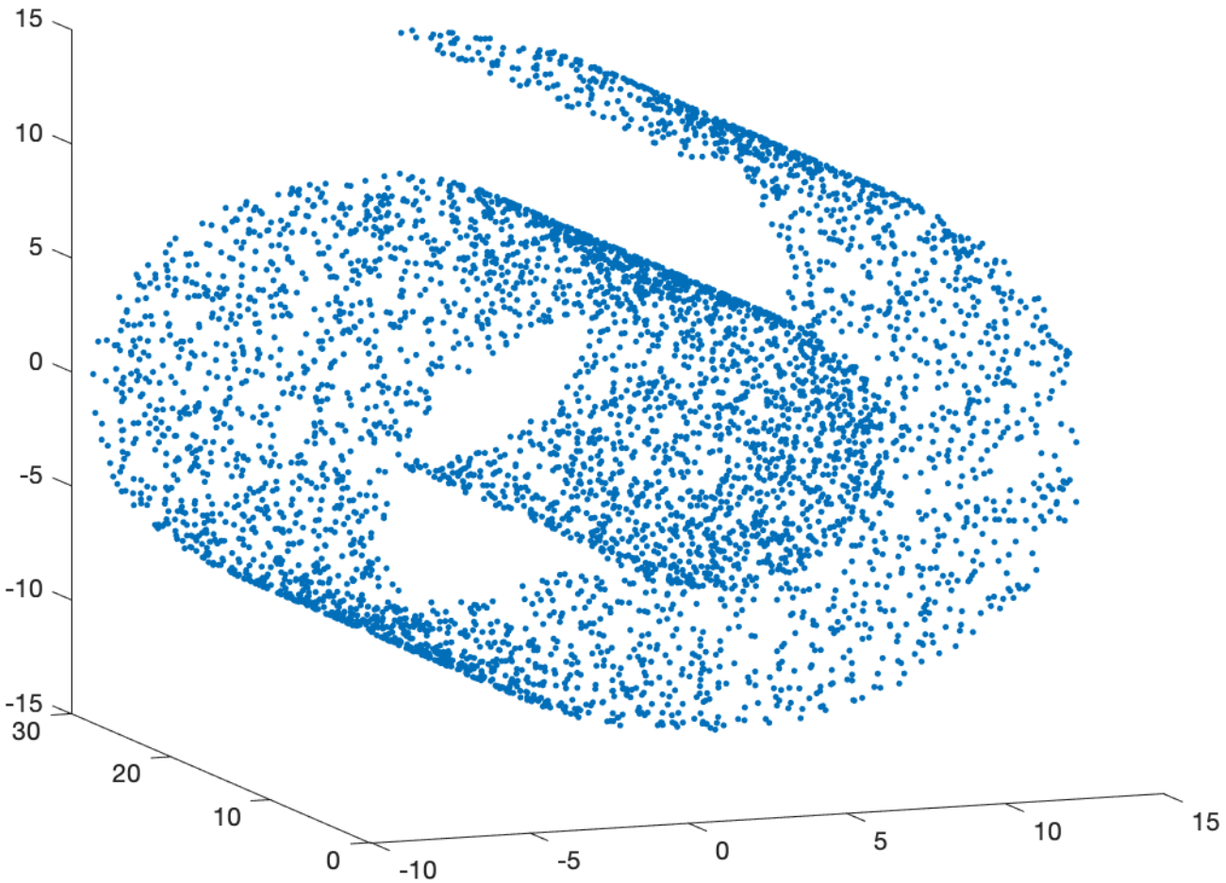
ims,
.')
ate.
1253)
01)

fx >>

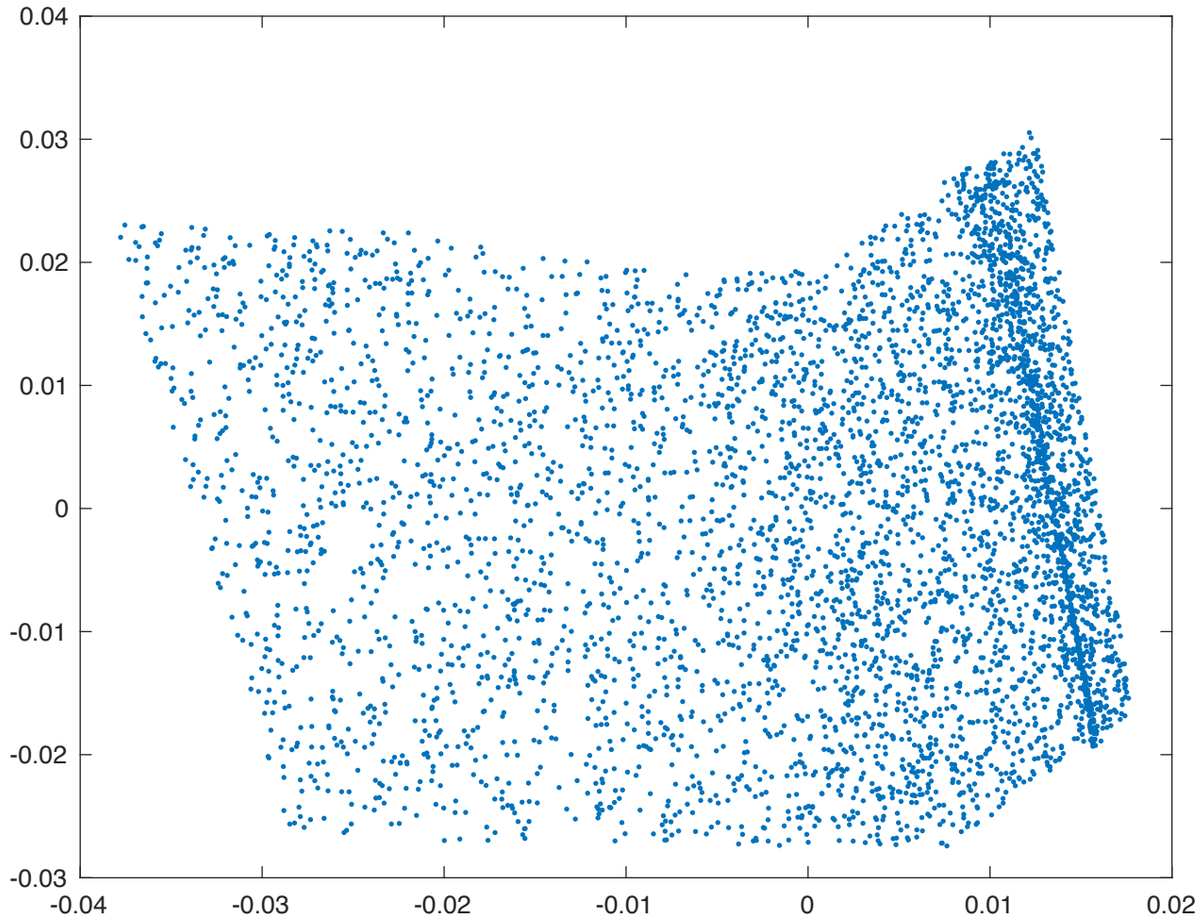
Details


15
10
5
0
-5
-10
-15
30
20
10
0
-10
-5
0
5
10
15

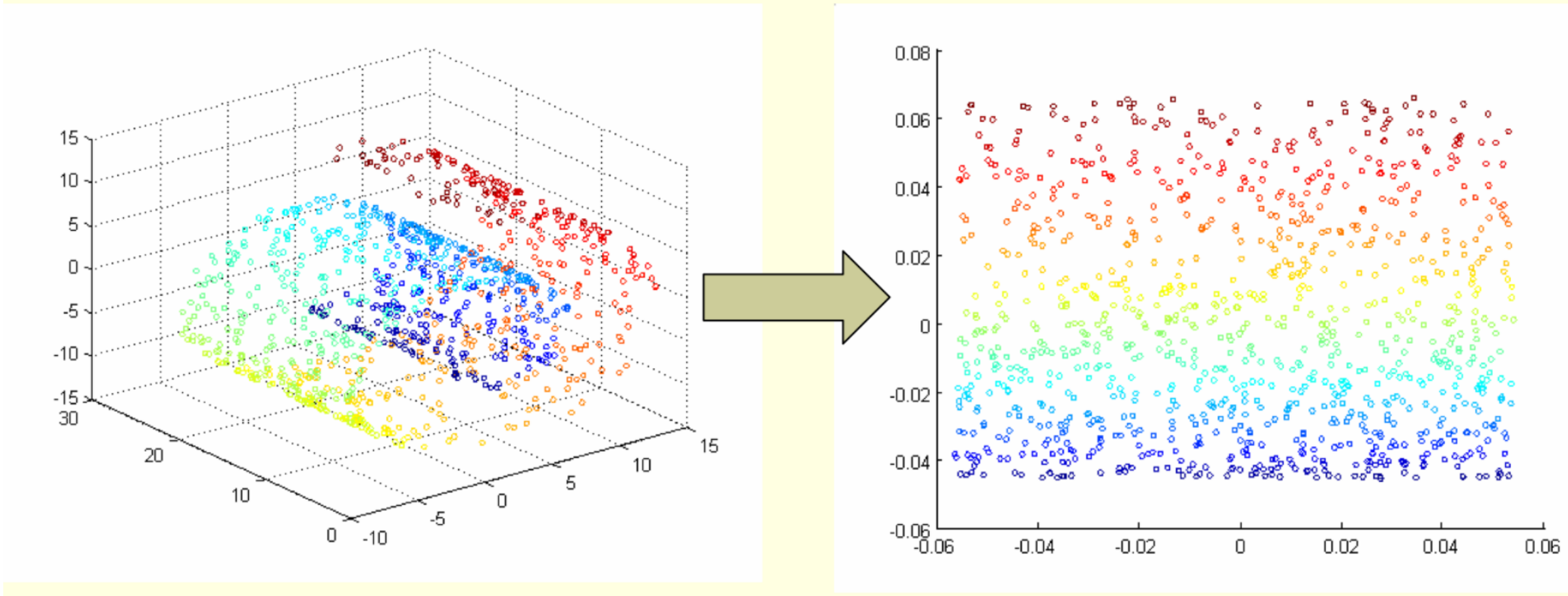
3D data



An embedded 2D manifold



Discover embedding



A Global Geometric Framework for Nonlinear Dimensionality Reduction

Joshua B. Tenenbaum,[1]* Vin de Silva,[2] John C. Langford[3]

Scientists working with large volumes of high-dimensional data, such as global climate patterns, stellar spectra, or human gene distributions, regularly confront the problem of dimensionality reduction: finding meaningful low-dimensional structures hidden in their high-dimensional observations. The human brain confronts the same problem in everyday perception, extracting from its high-dimensional sensory inputs—30,000 auditory nerve fibers or 10^6 optic nerve fibers—a manageably small number of perceptually relevant features. Here we describe an approach to solving dimensionality reduction problems that uses easily measured local metric information to learn the underlying global geometry of a data set. Unlike classical techniques such as principal component analysis (PCA) and multidimensional scaling (MDS), our approach is capable of discovering the nonlinear degrees of freedom that underlie complex natural observations, such as human handwriting or images of a face under different viewing conditions. In contrast to previous algorithms for nonlinear dimensionality reduction, ours efficiently computes a globally optimal solution, and, for an important class of data manifolds, is guaranteed to converge asymptotically to the true structure.

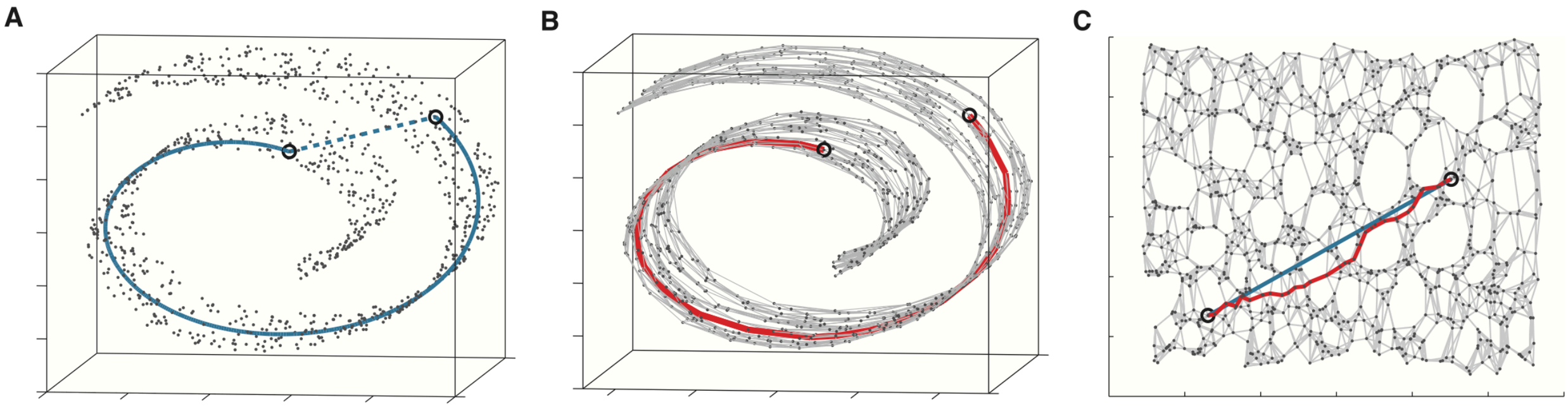
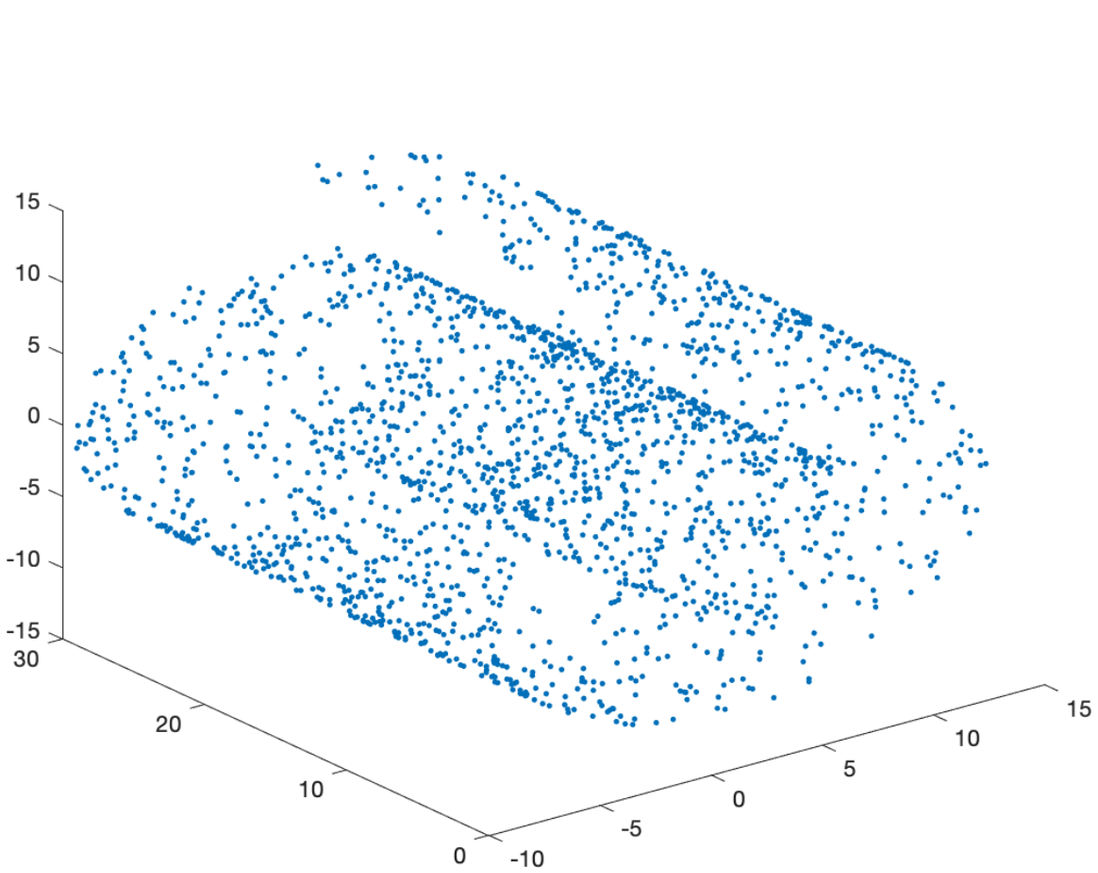
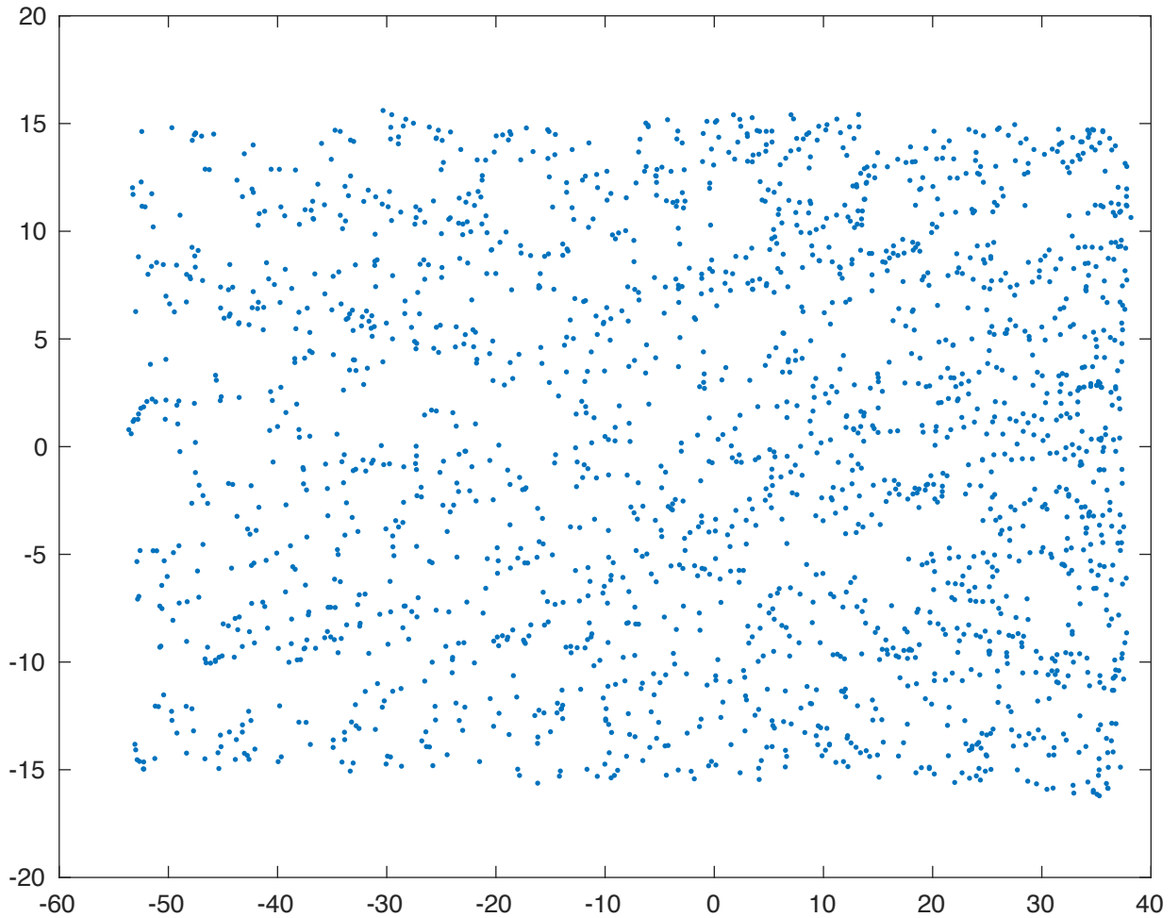


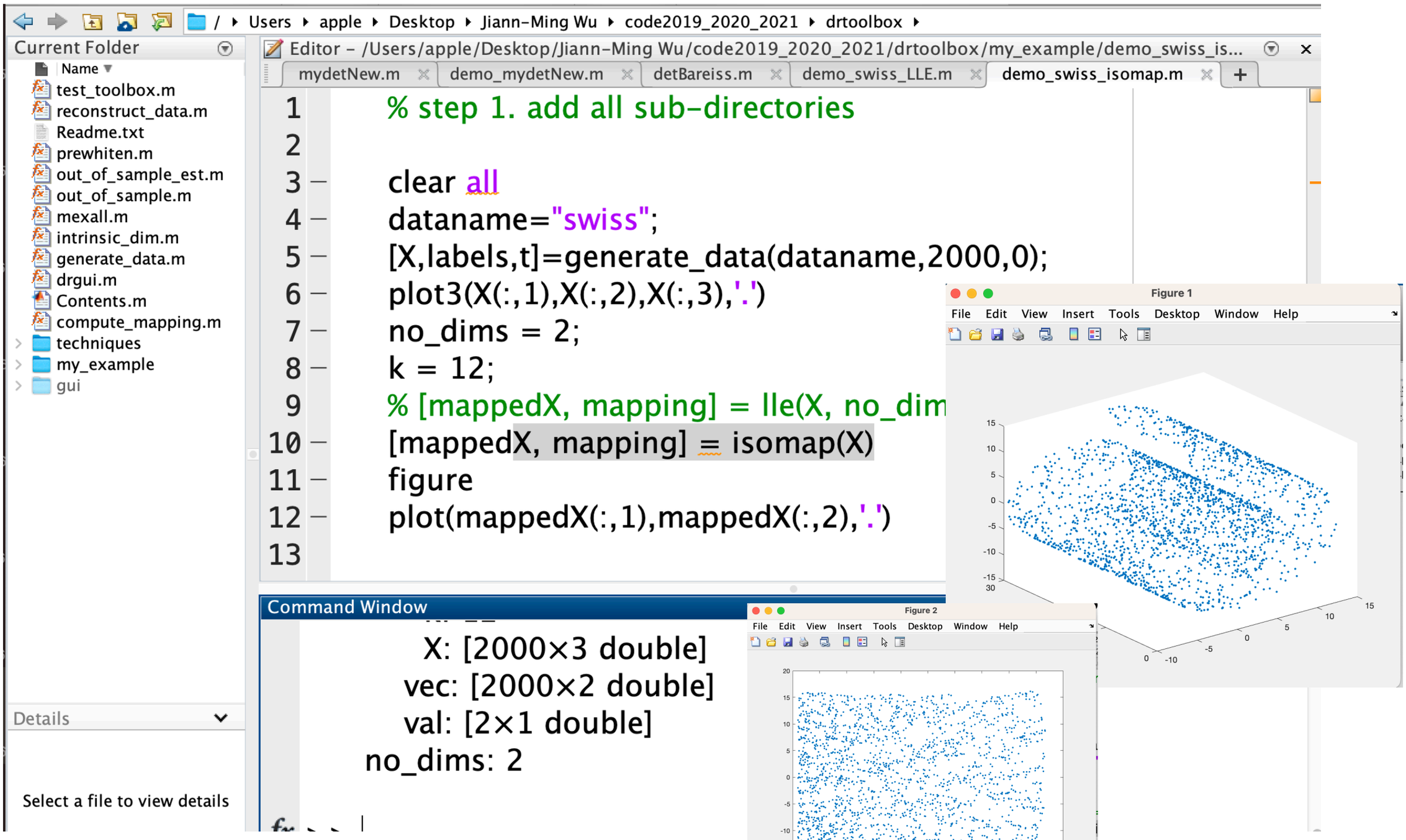
Fig. 3. The "Swiss roll" data set, illustrating how Isomap exploits geodesic paths for nonlinear dimensionality reduction. (**A**) For two arbitrary points (circled) on a nonlinear manifold, their Euclidean distance in the high-dimensional input space (length of dashed line) may not accurately reflect their intrinsic similarity, as measured by geodesic distance along the low-dimensional manifold (length of solid curve). (**B**) The neighborhood graph G constructed in step one of Isomap (with $K = 7$ and $N = 1000$ data points) allows an approximation (red segments) to the true geodesic path to be computed efficiently in step two, as the shortest path in G. (**C**) The two-dimensional embedding recovered by Isomap in step three, which best preserves the shortest path distances in the neighborhood graph (overlaid). Straight lines in the embedding (blue) now represent simpler and cleaner approximations to the true geodesic paths than do the corresponding graph paths (red).


15
10
5
0
-5
-10
-15
30
20
10
0
-10
-5
0
5
10
15


20
15
10
5
0
-5
-10
-15
-20
-60
-50
-40
-30
-20
-10
0
10
20
30
40

ISOMap

Matlab demo function



Three steps

are detailed in Table 1. The first step determines which points are neighbors on the manifold M, based on the distances $d_X(i,j)$ between pairs of points i,j in the input space

1 Construct neighborhood graph

Define the graph G over all data points by connecting points i and j if [as measured by $d_X(i,j)$] they are closer than ϵ (ϵ-Isomap), or if i is one of the K nearest neighbors of j (K-Isomap). Set edge lengths equal to $d_X(i,j)$.

X. Two simple methods are to connect each point to all points within some fixed radius ϵ, or to all of its K nearest neighbors (*15*). These neighborhood relations are represented as a weighted graph G over the data points, with edges of weight $d_X(i,j)$ between neighboring points (Fig. 3B).

9.30

In its second step, Isomap estimates the geodesic distances $d_M(i,j)$ between all pairs of points on the manifold M by computing their shortest path distances $d_G(i,j)$ in the graph G. One simple algorithm (*16*) for finding shortest paths is given in Table 1.

2 Compute shortest paths

Initialize $d_G(i,j) = d_X(i,j)$ if i,j are linked by an edge; $d_G(i,j) = \infty$ otherwise. Then for each value of $k = 1, 2, \ldots, N$ in turn, replace all entries $d_G(i,j)$ by $\min\{d_G(i,j), d_G(i,k) + d_G(k,j)\}$. The matrix of final values $D_G = \{d_G(i,j)\}$ will contain the shortest path distances between all pairs of points in G (*16*, *19*).

The final step applies classical MDS to the matrix of graph distances $D_G = \{d_G(i,j)\}$, constructing an embedding of the data in a d-dimensional Euclidean space Y that best preserves the manifold's estimated intrinsic geometry (Fig. 3C). The coordinate vectors $\mathbf{y}_i$ for points in Y are chosen to minimize the cost function

$$E = \|\tau(D_G) - \tau(D_Y)\|_{L^2} \qquad (1)$$

where D_Y denotes the matrix of Euclidean distances $\{d_Y(i,j) = \|\mathbf{y}_i - \mathbf{y}_j\|\}$ and $\|A\|_{L^2}$ the L^2 matrix norm $\sqrt{\sum_{i,j} A_{ij}^2}$. The τ operator

3 Construct d-dimensional embedding

Let λ_p be the p-th eigenvalue (in decreasing order) of the matrix $\tau(D_G)$ (*17*), and v_p^i be the i-th component of the p-th eigenvector. Then set the p-th component of the d-dimensional coordinate vector $\mathbf{y}_i$ equal to $\sqrt{\lambda_p} v_p^i$.

---

17. The operator τ is defined by $\tau(D) = -HSH/2$, where S is the matrix of squared distances $\{S_{ij} = D_{ij}^2\}$, and H is the "centering matrix" $\{H_{ij} = \delta_{ij} - 1/N\}$ (*13*).

Science paper

- Nonlinear dimensionality Reduction by locally linear embedding

Nonlinear Dimensionality Reduction by Locally Linear Embedding

Sam T. Roweis[1] and Lawrence K. Saul[2]

Many areas of science depend on exploratory data analysis and visualization. The need to analyze large amounts of multivariate data raises the fundamental problem of dimensionality reduction: how to discover compact representations of high-dimensional data. Here, we introduce locally linear embedding (LLE), an unsupervised learning algorithm that computes low-dimensional, neighborhood-preserving embeddings of high-dimensional inputs. Unlike clustering methods for local dimensionality reduction, LLE maps its inputs into a single global coordinate system of lower dimensionality, and its optimizations do not involve local minima. By exploiting the local symmetries of linear reconstructions, LLE is able to learn the global structure of nonlinear manifolds, such as those generated by images of faces or documents of text.

How do we judge similarity? Our mental representations of the world are formed by processing large numbers of sensory inputs—including, for example, the pixel intensities of images, the power spectra of sounds, and the joint angles of articulated bodies. While complex stimuli of this form can be represented by points in a high-dimensional vector space, they typically have a much more compact description. Coherent structure in the world leads to strong correlations between inputs (such as between neighboring pixels in images), generating observations that lie on or close to a smooth low-dimensional manifold. To compare and classify such observations—in effect, to reason about the world—depends crucially on modeling the nonlinear geometry of these low-dimensional manifolds.

Scientists interested in exploratory analysis or visualization of multivariate data (*1*) face a similar problem in dimensionality reduction. The problem, as illustrated in Fig. 1, involves mapping high-dimensional inputs into a low-dimensional "description" space with as many coordinates as observed modes of variability. Previous approaches to this problem, based on multidimensional scaling (MDS) (*2*), have computed embeddings that attempt to preserve pairwise distances [or generalized disparities (*3*)] between data points; these distances are measured along straight lines or, in more sophisticated usages of MDS such as Isomap (*4*), along shortest paths confined to the manifold of observed inputs. Here, we take a different approach, called locally linear embedding (LLE), that eliminates the need to estimate pairwise distances between widely separated data points. Unlike previous methods, LLE recovers global nonlinear structure from locally linear fits.

The LLE algorithm, summarized in Fig. 2, is based on simple geometric intuitions. Suppose the data consist of N real-valued vectors $\vec{X}_i$, each of dimensionality D, sampled from some underlying manifold. Provided there is sufficient data (such that the manifold is well-sampled), we expect each data point and its neighbors to lie on or close to a locally linear patch of the manifold. We characterize the local geometry of these patches by linear coefficients that reconstruct each data point from its neighbors. Reconstruction errors are measured by the cost function

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \Sigma_j W_{ij} \vec{X}_j \right|^2 \qquad (1)$$

which adds up the squared distances between all the data points and their reconstructions. The weights W_{ij} summarize the contribution of the jth data point to the ith reconstruction. To compute the weights W_{ij}, we minimize the cost

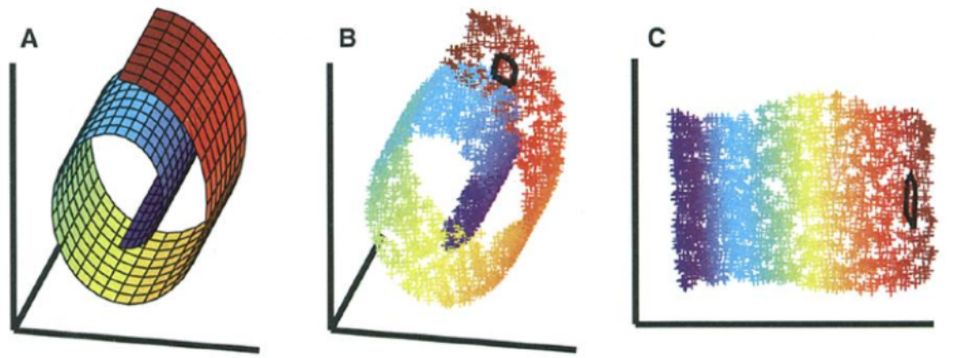


Fig. 1. The problem of nonlinear dimensionality reduction, as illustrated (*10*) for three-dimensional data (B) sampled from a two-dimensional manifold (**A**). An unsupervised learning algorithm must discover the global internal coordinates of the manifold without signals that explicitly indicate how the data should be embedded in two dimensions. The color coding illustrates the neighborhood-preserving mapping discovered by LLE; black outlines in (**B**) and (**C**) show the neighborhood of a single point. Unlike LLE, projections of the data by principal component analysis (PCA) (*28*) or classical MDS (*2*) map faraway data points to nearby points in the plane, failing to identify the underlying structure of the manifold. Note that mixture models for local dimensionality reduction (*29*), which cluster the data and perform PCA within each cluster, do not address the problem considered here: namely, how to map high-dimensional data into a single global coordinate system of lower dimensionality.

[1]Gatsby Computational Neuroscience Unit, University College London, 17 Queen Square, London WC1N 3AR, UK. [2]AT&T Lab—Research, 180 Park Avenue, Florham Park, NJ 07932, USA.

E-mail: roweis@gatsby.ucl.ac.uk (S.T.R.); lsaul@research.att.com (L.K.S.)

$$|x|^2 = x^T x$$

- To compute the N x N weight matrix W we want to minimize the following cost function:

$$X_i \approx \sum_j W_{ij} X_j$$

Objective function

$$\varepsilon(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2$$

where $W_{ij} = 0$ if X_j is not one of the K nearest neighbors of X_i and where the rows of W sum to 1

Constraints

$$\sum_j W_{ij} = 1 \longrightarrow$$

$$\boldsymbol{W} = \begin{pmatrix} .5 & .2 & .3 \ldots & 0 & 0 & 0 \\ \ldots \text{ W sparse} & & & & & \\ \ldots & & & & & \\ \ldots & & & & & \\ & & & & & \end{pmatrix} \begin{matrix} \text{N} \\ \\ \\ \\ \\ \text{N} \end{matrix}$$

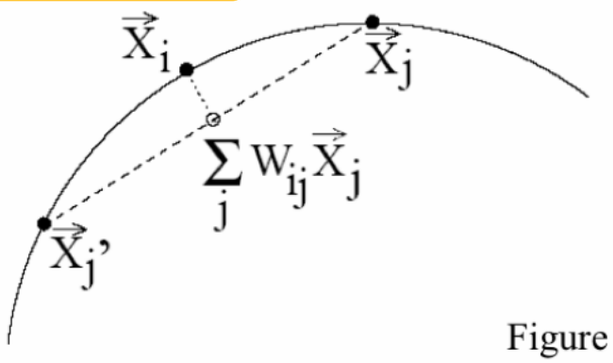


Figure from Roweis and Saul, 2003

Solving for one row of W

- Consider a particular data point $X_i = z$ with K nearest neighbors $X_j = n_j$ and reconstruction weights $W_{ij} = w_j$ that sum to one. Then,

$$\varepsilon = \left| z - \sum_j w_j n_j \right|^2$$

$$= \left| \sum_j w_j (z - n_j) \right|^2 \qquad \text{since } \sum_j w_j = 1$$

$$= \sum_j \sum_k w_j w_k C_{jk} \qquad \text{where } C_{jk} = (z - n_j) \cdot (z - n_k),$$

the local covariance matrix

Method 1 provided in paper

7. Fits: The constrained weights that best reconstruct each data point from its neighbors can be computed in closed form. Consider a particular data point $\vec{x}$ with neighbors $\vec{\eta}_j$ and sum-to-one reconstruction weights w_j. The reconstruction error $\left|\vec{x} - \sum_{j=1}^{K} w_j \vec{\eta}_j\right|^2$ is minimized in three steps. First, evaluate inner products between neighbors to compute the neighborhood correlation matrix, $C_{jk} = \vec{\eta}_j \cdot \vec{\eta}_k$, and its matrix inverse, C^{-1}. Second, compute the Lagrange multiplier, $\lambda = \alpha/\beta$, that enforces the sum-to-one constraint, where $\alpha = 1 - \sum_{jk} C_{jk}^{-1}(\vec{x} \cdot \vec{\eta}_k)$ and $\beta = \sum_{jk} C_{jk}^{-1}$. Third, compute the reconstruction weights: $w_j = \sum_k C_{jk}^{-1}(\vec{x} \cdot \vec{\eta}_k + \lambda)$. If the correlation matrix C is nearly singular, it can be conditioned (before inversion) by adding a small multiple of the identity matrix. This amounts to penalizing large weights that exploit correlations beyond some level of precision in the data sampling process.

	1	2		j	5000
1		1			
i				1	
5000					

NB(i,j)=1, if x_j is a neighbor of x_i

```
% Construct reconstruction weight matrix
W = zeros(max_k, n);
for i=1:n
    nbhd = neighborhood(:,i);
    nbhd = nbhd(nbhd ~= 0);
    kt = numel(nbhd);
    z = bsxfun(@minus, X(:,nbhd), X(:,i));
    C = z' * z;
    C = C + eye(kt, kt) * tol * trace(C);
    wi = C \ ones(kt, 1);
    wi = wi / sum(wi);
    W(:,i) = [wi; nan(max_k - kt, 1)];
end
```

$$X_i \approx \sum_j W_{ij} X_j$$

Codes employed by lle

$z_j = x_j - x_i$, where $x_j \in NB(x_i$

$$C = \sum_{j \in NB(i)} z_j^T z_j$$

$$(C + \lambda I)w = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

1. Find K nearest neighbors of each vector, X_i, in R^D as measured by Euclidean distance.

2. Compute the weights W_{ij} that best reconstruct X_i from its neighbors.

$$X_i \approx \sum_j W_{ij} X_j$$

3. Compute vectors Y_i in R^d reconstructed by the weights W_{ij}. Solve for all Y_i simultaneously.

$$Y_i \approx \sum_j W_{ij} Y_j$$

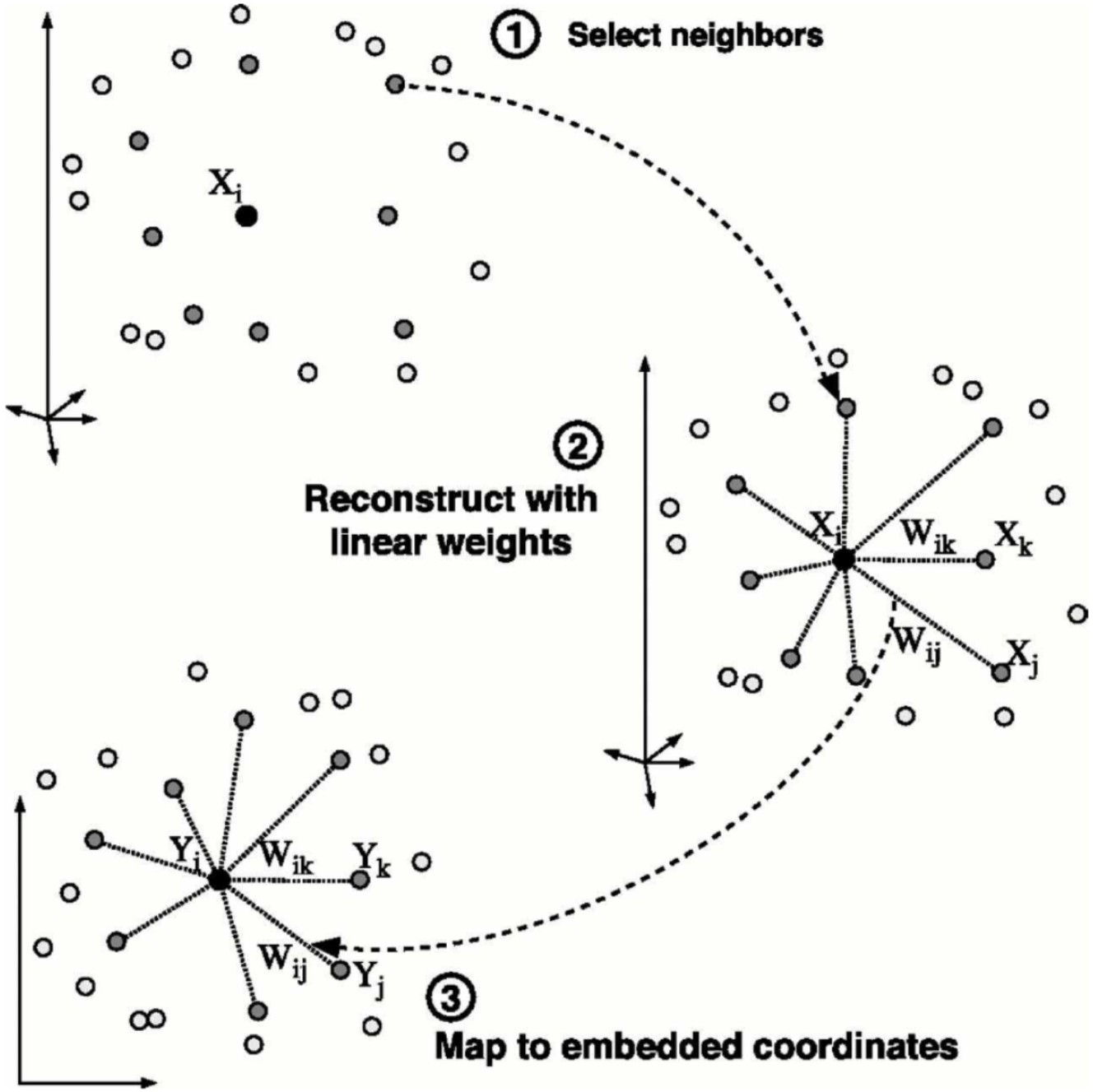


Figure from Roweis and Saul, 2001

Single constraint [edit]

For the case of only one constraint and only two choice variables (as exemplified in Figure 1), consider the optimization problem

$$\begin{aligned} &\text{maximize } f(x, y) \\ &\text{subject to: } g(x, y) = 0 \end{aligned}$$

(Sometimes an additive constant is shown separately rather than being included in g, in which case the constraint is written $g(x, y) = c$, as in Figure 1.) We assume that both f and g have continuous first partial derivatives. We introduce a new variable (λ) called a **Lagrange multiplier** (or **Lagrange undetermined multiplier**) and study the **Lagrange function** (or **Lagrangian** or **Lagrangian expression**) defined by

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y),$$

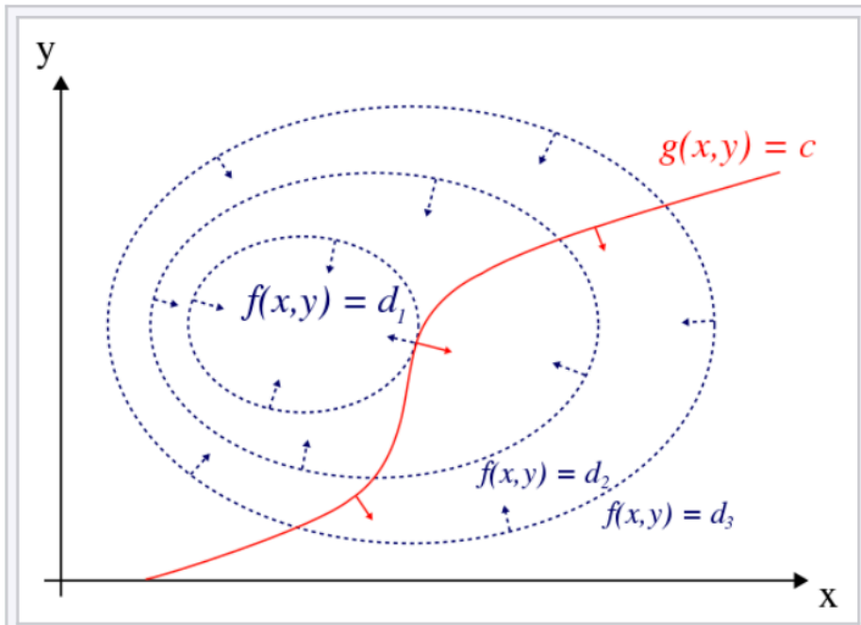


Figure 1: The red curve shows the constraint

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y),$$

and solve

$$\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = 0.$$

Note that this amounts to solving three equations in three unknowns. This is the method of Lagrange multipliers.

Note that $\nabla_\lambda \mathcal{L}(x, y, \lambda) = 0$ implies $g(x, y) = 0$, as the partial derivative of $\mathcal{L}$ with respect to λ is $-g(x, y)$, which clearly is zero if and only if $g(x, y) = 0$.

To summarize

$$\nabla_{x,y,\lambda} \mathcal{L}(x, y, \lambda) = 0 \iff \begin{cases} \nabla_{x,y} f(x, y) = \lambda \nabla_{x,y} g(x, y) \\ g(x, y) = 0 \end{cases}$$

The method generalizes readily to functions on n variables

$$\nabla_{x_1,\ldots,x_n,\lambda} \mathcal{L}(x_1, \ldots, x_n, \lambda) = 0$$

which amounts to solving $n + 1$ equations in $n + 1$ unknowns.

$$E(w) = \sum_j \sum_k w_j C_{jk} w_k - \lambda(\sum_j w_j - 1)$$

$$\frac{\partial E}{\partial w_i} = 2 \sum_j w_j C_{ji} - \lambda = 0, \text{for all} \quad i$$

$$\frac{\partial E}{\partial \lambda} = \sum_j w_j - 1 = 0$$

$$E(w) = \sum_j \sum_k w_j C_{jk} w_k + \lambda(\sum_j w_j - 1)$$

$$\sum_j w_j C_{ji} - \frac{\lambda}{2} = 0, \text{for all} \quad i$$

$$\sum_j w_j = 1$$

solving $n + 1$ equations in $n + 1$ unknowns.

$$\sum_j w_j C_{ji} - \frac{\lambda}{2} = 0, \text{for all} \quad i$$

Replace $\frac{\lambda}{2}$ with λ

$$\sum_j w_j = 1$$

$$\begin{bmatrix} C_{11} & \dots & C_{1j} & \dots & C_{1n} & -1 \\ & \dots & & \dots & & \\ C_{i1} & \dots & C_{ij} & \dots & C_{in} & -1 \\ & \dots & & \dots & & \\ C_{n1} & \dots & C_{nj} & \dots & C_{nn} & -1 \\ 1 & \dots & 1 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \dots \\ w_i \\ \dots \\ w_n \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ \dots \\ 0 \\ \dots \\ 0 \\ 1 \end{bmatrix}$$

Implementation is not exactly based on the Lagrange multiplier

/ ▸ Users ▸ apple ▸ Desktop ▸ Jiann-Ming Wu ▸ code2019_2020_2021 ▸ drtoolbox ▸

Editor - /Users/apple/Desktop/Jiann-Ming Wu/code2019_2020_2021/drtoolbox/techn

demo_swiss_LLE.m | lle.m | +

```
74        % Construct reconstruction weight matrix
75        W = zeros(max_k, n);
76        for i=1:n
77            nbhd = neighborhood(:,i);
78            nbhd = nbhd(nbhd ~= 0);
79            kt = numel(nbhd);
80            z = bsxfun(@minus, X(:,nbhd), X(:,i));
81            C = z' * z;
82            C = C + eye(kt, kt) * tol * trace(C);
83            wi = C \ ones(kt, 1);
84            wi = wi / sum(wi);
85            W(:,i) = [wi; nan(max_k - kt, 1)];
86        end
87
88        % Now that we have the reconstruction weights matrix,
89        % sparse cost matrix M = (I-W)'*(I-W)
```

```
                                % Shift point to origin
                                % Compute local covarian
                                % Regularization of cova
                                % Solve linear system
                                                  sum is
```

trace(C) is the sum of the diagonal elements of C

Command Window

```
K>> sum(W(:,1))

ans =

    1.0000
```

Regulariza
-tion terms

$$E(w) = \frac{1}{2}\sum_j \sum_k w_j C_{jk} w_k + \frac{\lambda}{2}(\sum_j w_j - 1)^2$$

$$E(w) = \frac{1}{2}\sum_j \sum_k w_j C_{jk} w_k + \frac{\lambda}{2}(\sum_j w_j)^2 - \lambda \sum_j w_j + \frac{\lambda}{2}$$

$$\frac{dE}{dw} = (C + \lambda I)w = \begin{bmatrix} 1 \\ 1 \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$$

Computing Embedded Vectors Y_i

- Now that we have our weight matrix W, we would like to compute each of our embedding vectors Y_i. Minimize the following cost functions for fixed weights W_{ij}

$$\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

$$Y_i \approx \sum_j W_{ij} Y_j$$

- To make the problem well posed we add two constraints: (1) centered at the origin and (2) unit covariance:

$$\sum_i Y_i = 0 \qquad \frac{1}{N} \sum_i Y_i Y_i^T = I$$

- The first constraint removes the degree of freedom that Y be translated by a constant amount. The second expresses an assumption that reconstruction errors for different coordinates in the embedding space should be measured on the same scale.

Solving for matrix Y

- Let Y be the matrix that contains Y_i as each of it's columns

$$\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2$$

$$= \left| (I - W) Y \right|^2$$

$$= \mathrm{Y}^{\mathrm{T}} \mathrm{M} \mathrm{Y}$$

Unitary condition : $\sum_j w_j = 1$

Where $M = (I - W)^T (I - W)$ is N x N

- Using Lagrange multipliers and setting the derivative to zero gives

$$(M - \Lambda) Y^T = 0$$

- Λ here is the diagonal Lagrange multiplier matrix. This is an eigenvalue problem where all eigenvectors of M are solutions. The eigenvectors with the smallest eigenvalues minimize our cost. We discard the first (i.e. smallest) eigenvector which corresponds to the mean of Y to enforce constraint (1). The next d eigenvectors then give the Y that minimizes our cost subject to the constraints (see K. Fan for more information on the proof).

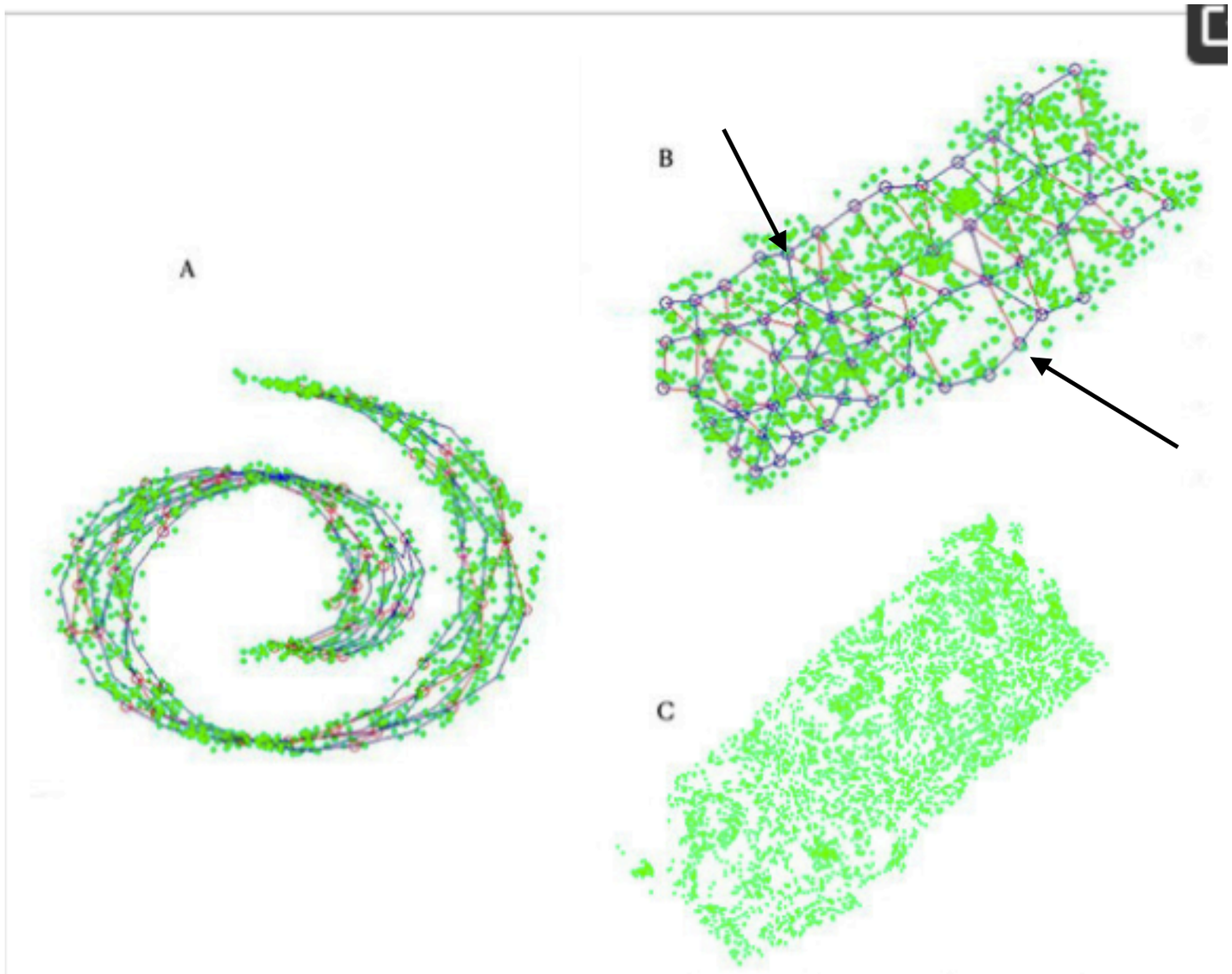
```
% Now that we have the reconstruction weights matrix, we define the
% sparse cost matrix M = (I-W)'*(I-W).
M = sparse(1:n, 1:n, ones(1, n), n, n, 4 * max_k * n);
for i=1:n
    w = W(:,i);
    j = neighborhood(:,i);
    indices = find(j ~= 0 & ~isnan(w));
    j = j(indices);
    w = w(indices);
    M(i, j) = M(i, j) - w';
    M(j, i) = M(j, i) - w;
    M(j, j) = M(j, j) + w * w';
end
```

Codes employed by lle

9. The embedding vectors $\vec{Y}_i$ are found by minimizing the cost function $\Phi(Y) = \Sigma_i |\vec{Y}_i - \Sigma_j W_{ij}\vec{Y}_j|^2$ over $\vec{Y}_i$ with fixed weights W_{ij}. This optimization is performed subject to constraints that make the problem well posed. It is clear that the coordinates $\vec{Y}_i$ can be translated by a constant displacement without affecting the cost, $\Phi(Y)$. We remove this degree of freedom by requiring the coordinates to be centered on the origin: $\Sigma_i \vec{Y}_i = \vec{0}$. Also, to avoid degenerate solutions, we constrain the embedding vectors to have unit covariance, with outer products that satisfy $\frac{1}{N}\Sigma_i \vec{Y}_i \otimes \vec{Y}_i = I$, where I is the $d \times d$ identity matrix. Now the cost defines a quadratic form, $\Phi(Y) = \Sigma_{ij} M_{ij}(\vec{Y}_i \cdot \vec{Y}_j)$, involving inner products of the embedding vectors and the symmetric $N \times N$ matrix

$$M_{ij} = \delta_{ij} - W_{ij} - W_{ji} + \sum_k W_{ki}W_{kj} \qquad (3)$$

where δ_{ij} is 1 if $i = j$ and 0 otherwise. The optimal embedding, up to a global rotation of the embedding space, is found by computing the bottom $d + 1$ eigenvectors of this matrix (*24*). The bottom eigenvector of this matrix, which we discard, is the unit vector with all equal components; it represents a free translation mode of eigenvalue zero. (Discarding it enforces the constraint that the embeddings have zero mean.) The remaining d eigenvectors form the d embedding coordinates found by LLE. Note that the matrix M can be stored and manipulated as the sparse matrix $(I - W)^T(I - W)$, giving substantial computational savings for large values of N. Moreover, its bottom $d + 1$ eigenvectors (those corresponding to its smallest $d + 1$ eigenvalues) can be found efficiently without performing a full matrix diagonalization (*25*).


A
B
C