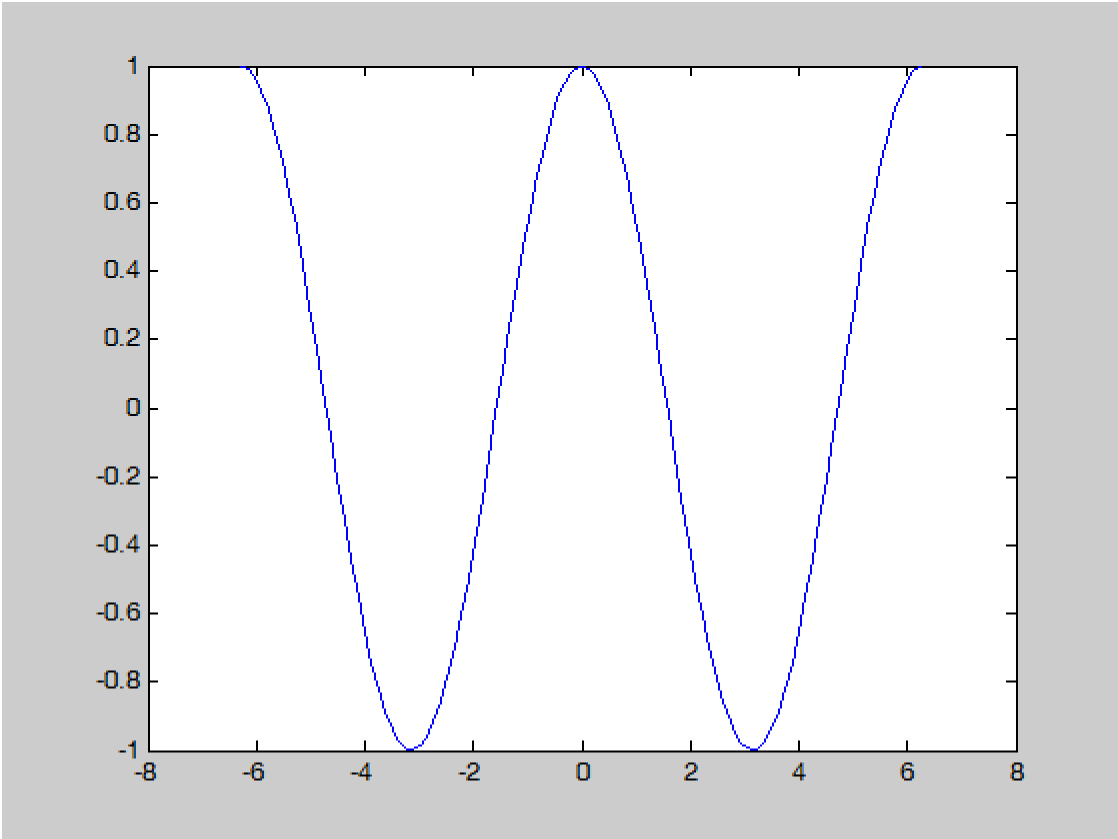
Data driven function approximation

Plot1d

Plot an arbitrary one-dimensional function

```
clear all
fstr=input('input a function: x.^2+cos(x) :','s');
fx=inline(fstr);
range=2*pi;
x=linspace(-range,range);
y=fx(x);
max_y=max(abs(y));
plot(x,y/max_y);
hold on;
```



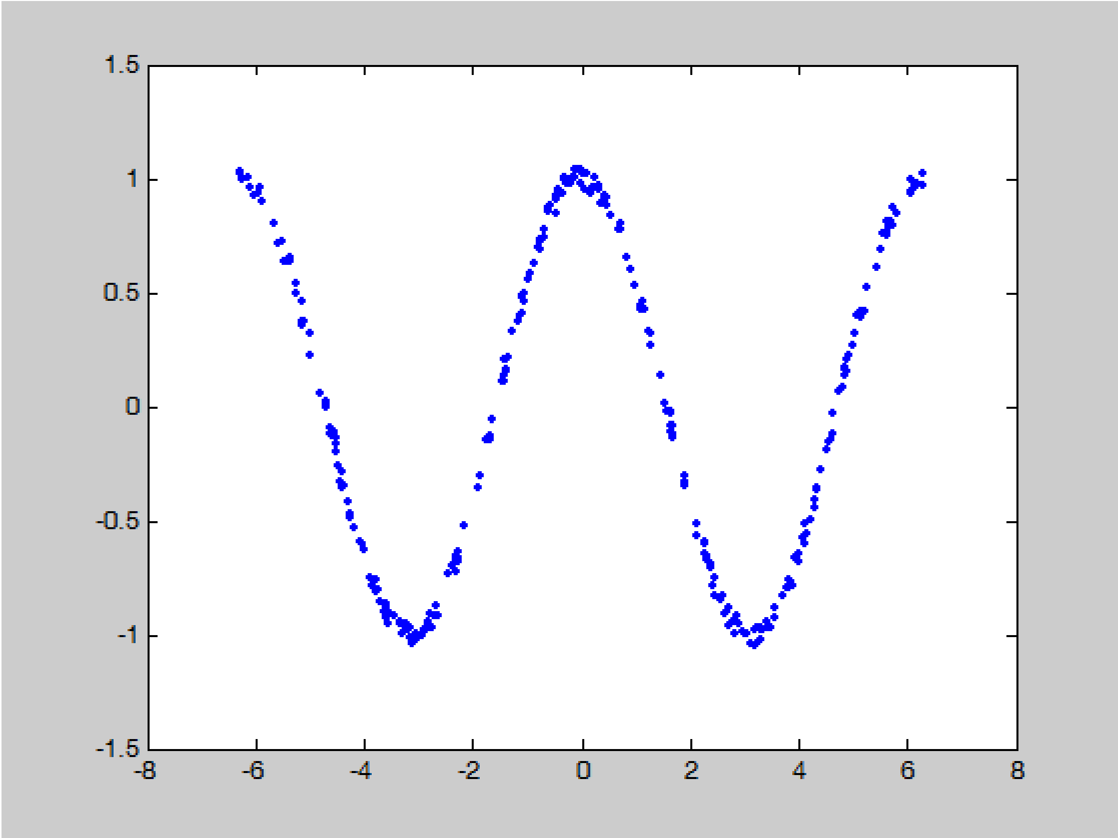
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-8
-6
-4
-2
0
2
4
6
8

Sampling

```
N=input('keyin sample size:');
x=rand(1,N)*2*range-range;
n=rand(1,N)*0.1-0.05;
y=fx(x)/max_y+n;
figure
plot(x,y,'.');
```

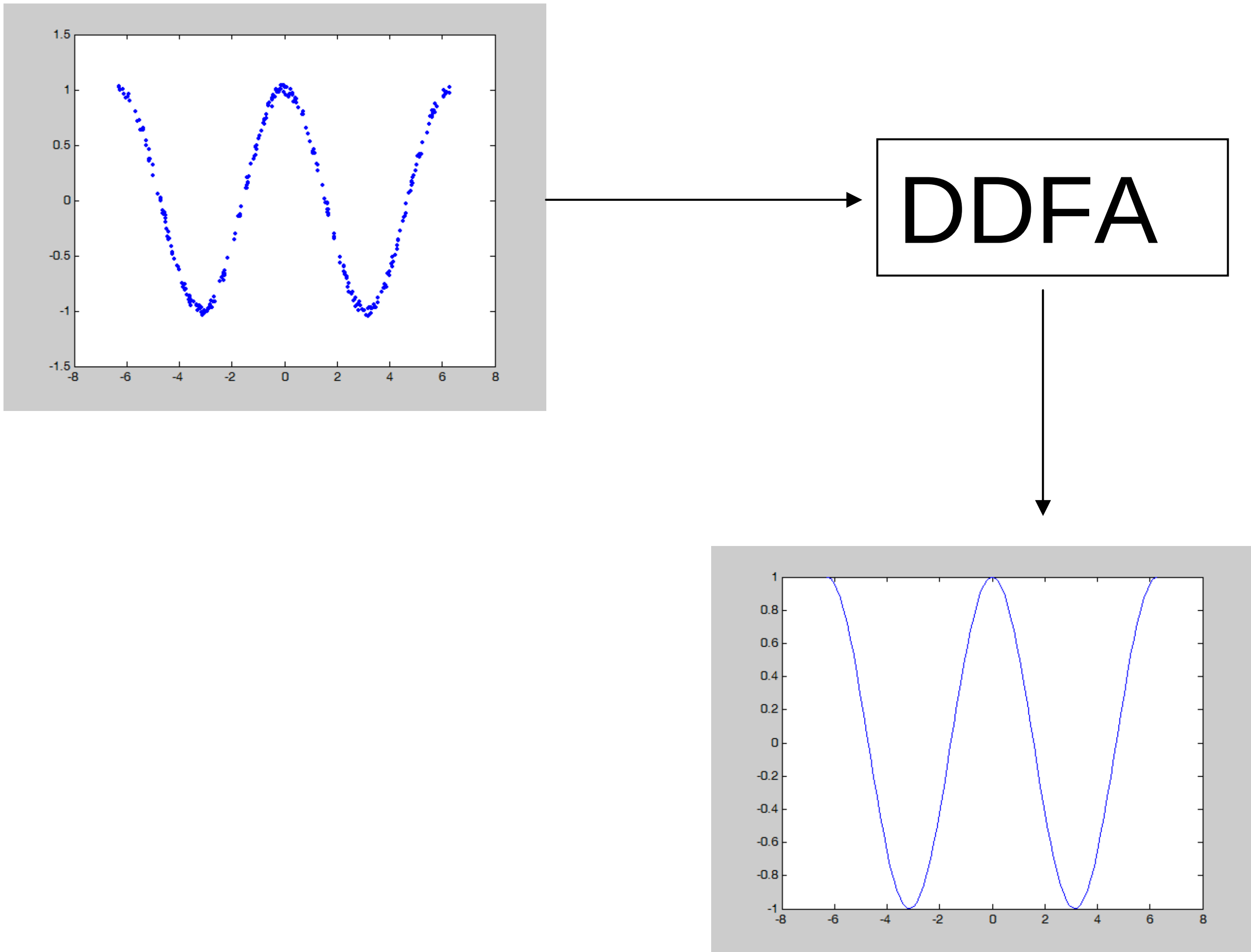
Paired data

Y



X

Function Approximation

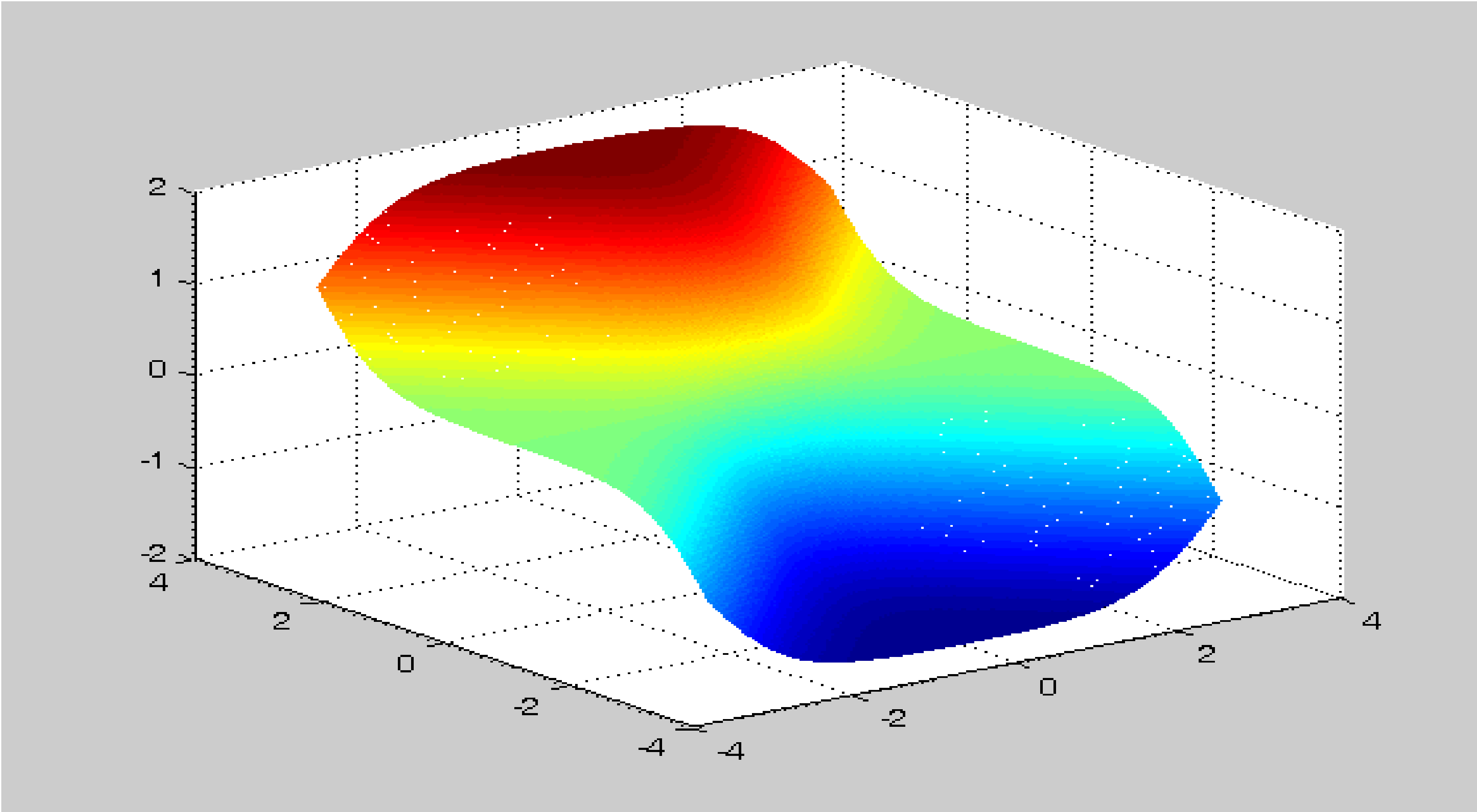


Plot2d

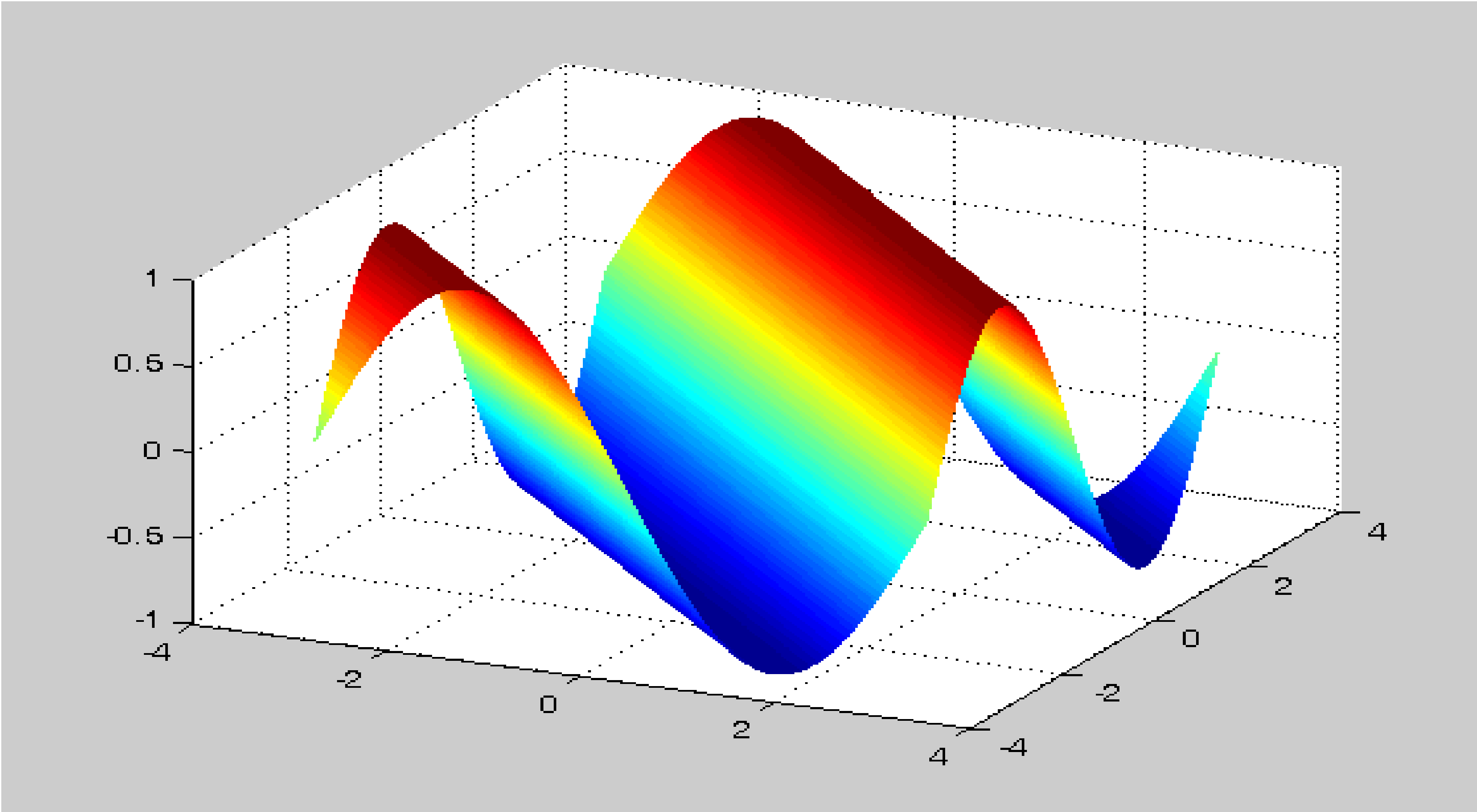
- Plot an arbitrary two-dimensional function

```
fstr=input('input a 2D function: x1.^2+x2.^2+cos(x1) :','s');
fx=inline(fstr);
range=pi;
x1=-range:0.02:range;
x2=x1;
for i=1:length(x1)
    C(i,:)=fx(x1(i),x2);
end
mesh(x1,x2,C);
hold on;
```

tanh(x1+x2)+tanh(x1-x2)



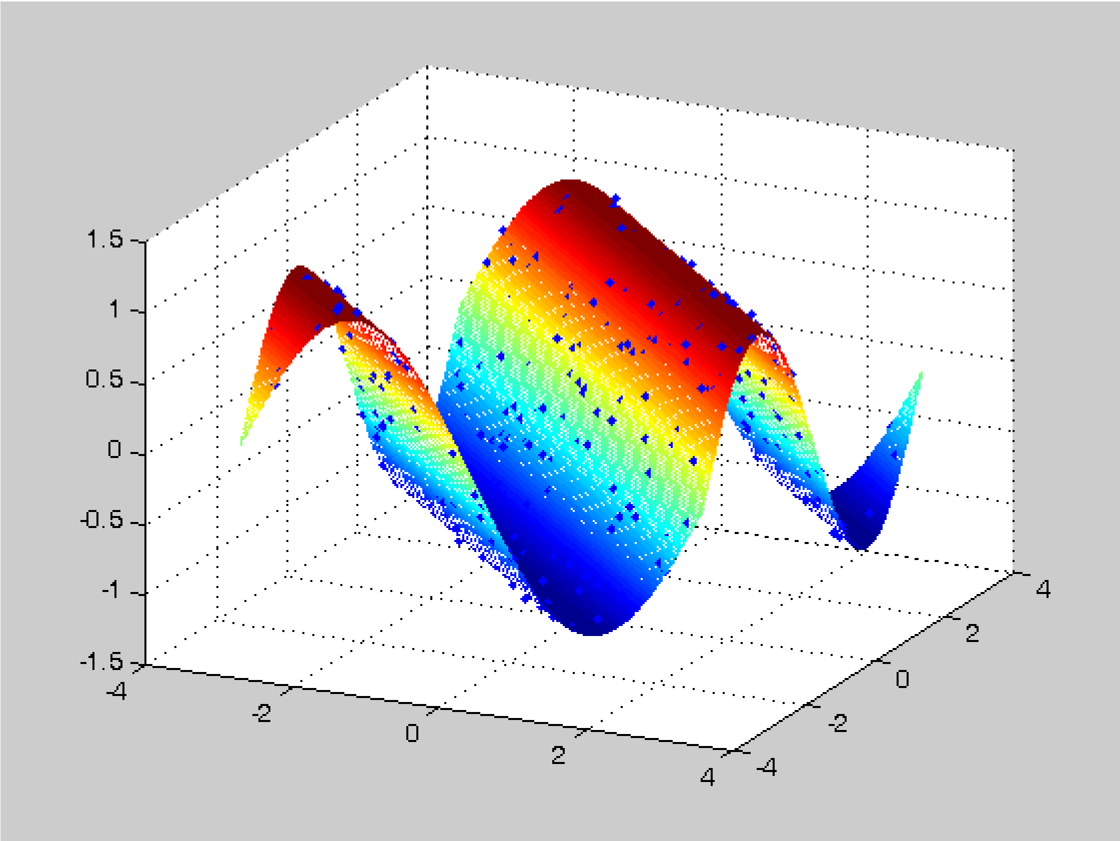
sin(x1+x2)



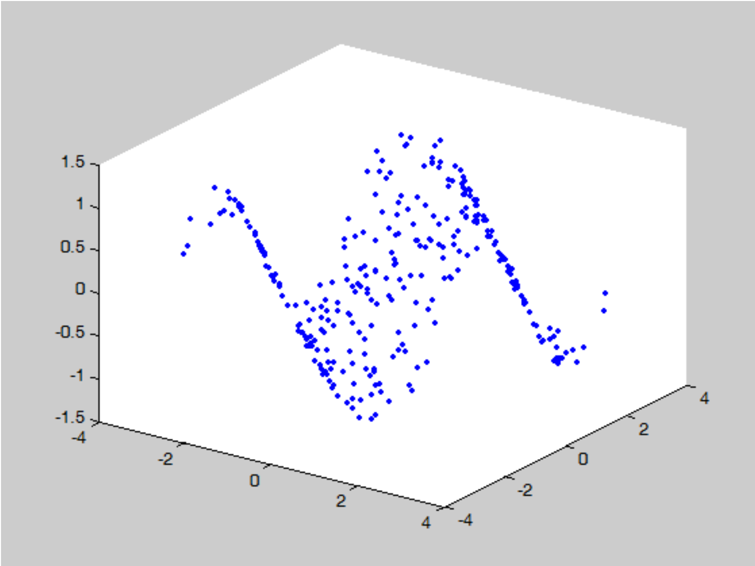
Sampling

```
N=input('keyin sample size:');
x(1,:)=rand(1,N)*2*range-range;
x(2,:)=rand(1,N)*2*range-range;
n=rand(1,N)*0.1-0.05;
y=fx(x(1,:),x(2,:))+n;
plot3(x(2,:),x(1,:),y,'.');
```

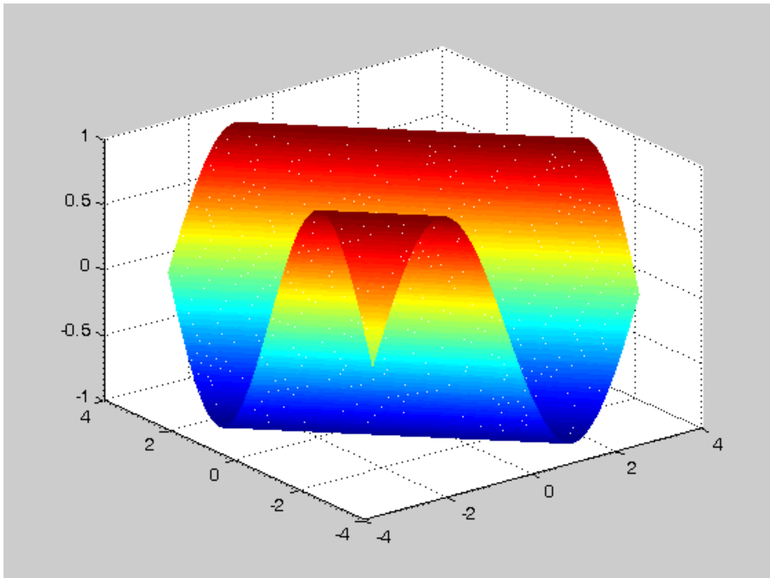
Sampling



Function guessing



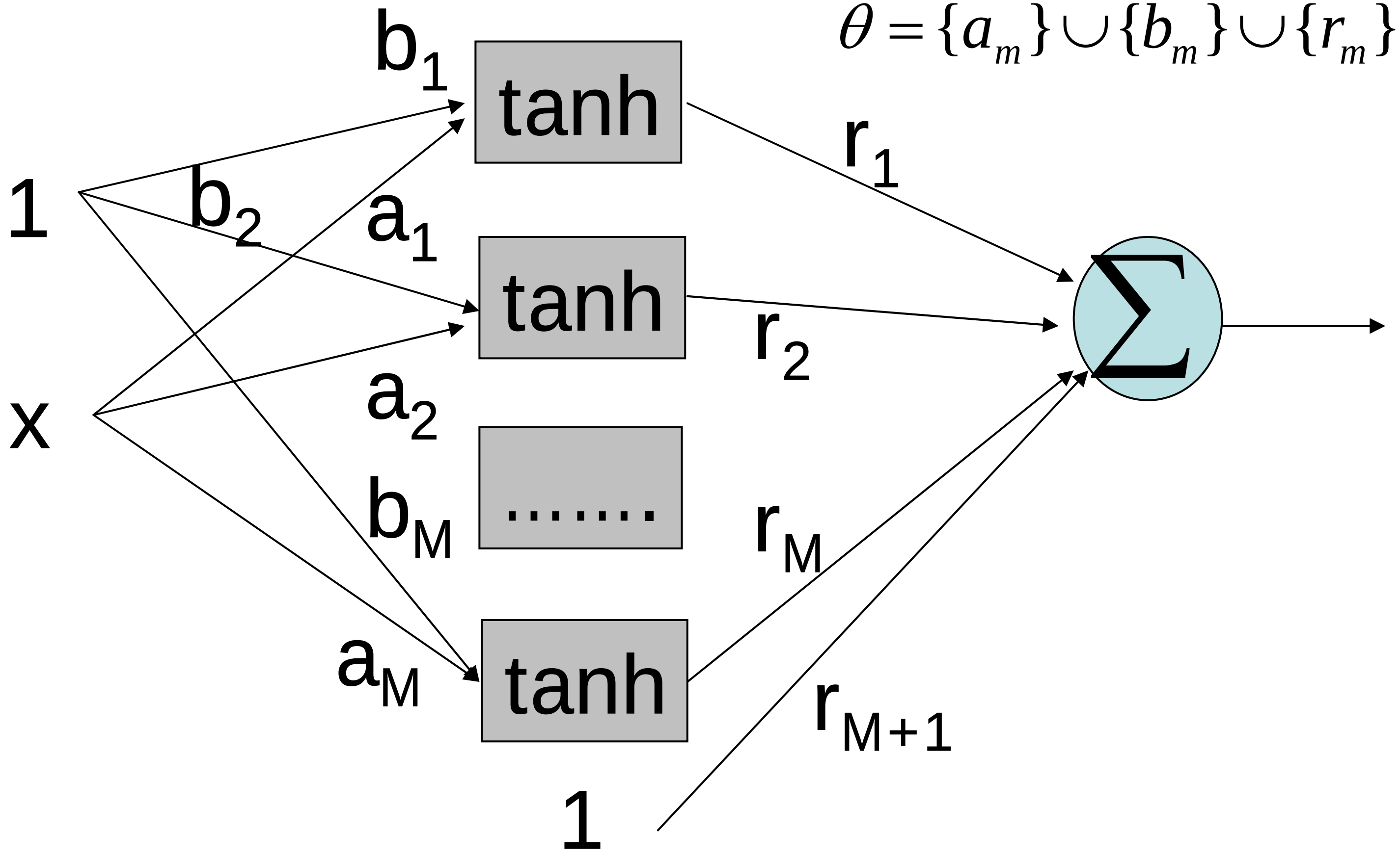
DDFA



Neural Networks

$$f(x_i;\theta) = \sum_{m=1}^{M} r_m \tanh(a_m x + b_m) + r_0$$

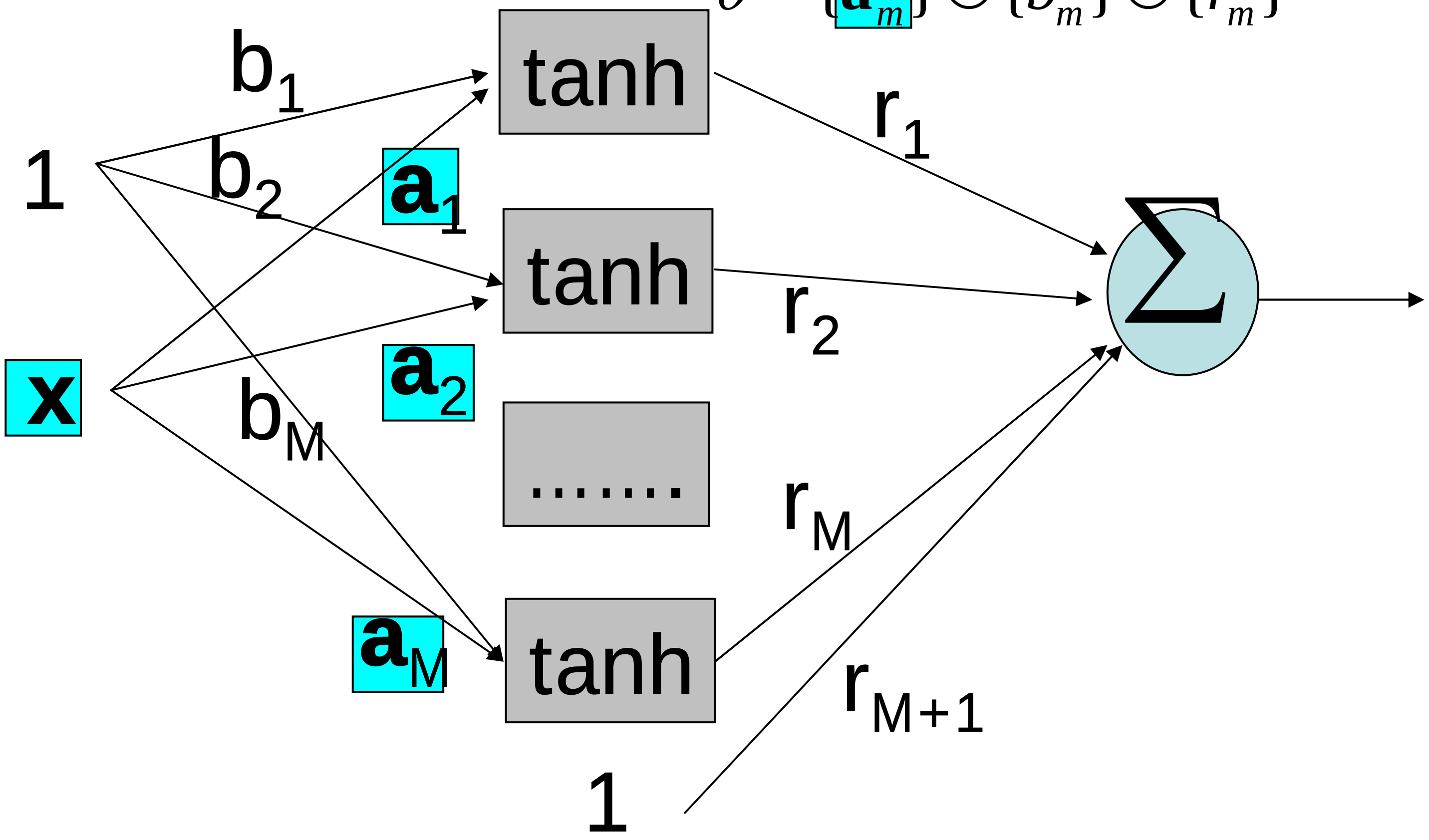
$$\theta = \{a_m\} \cup \{b_m\} \cup \{r_m\}$$



Neural Networks

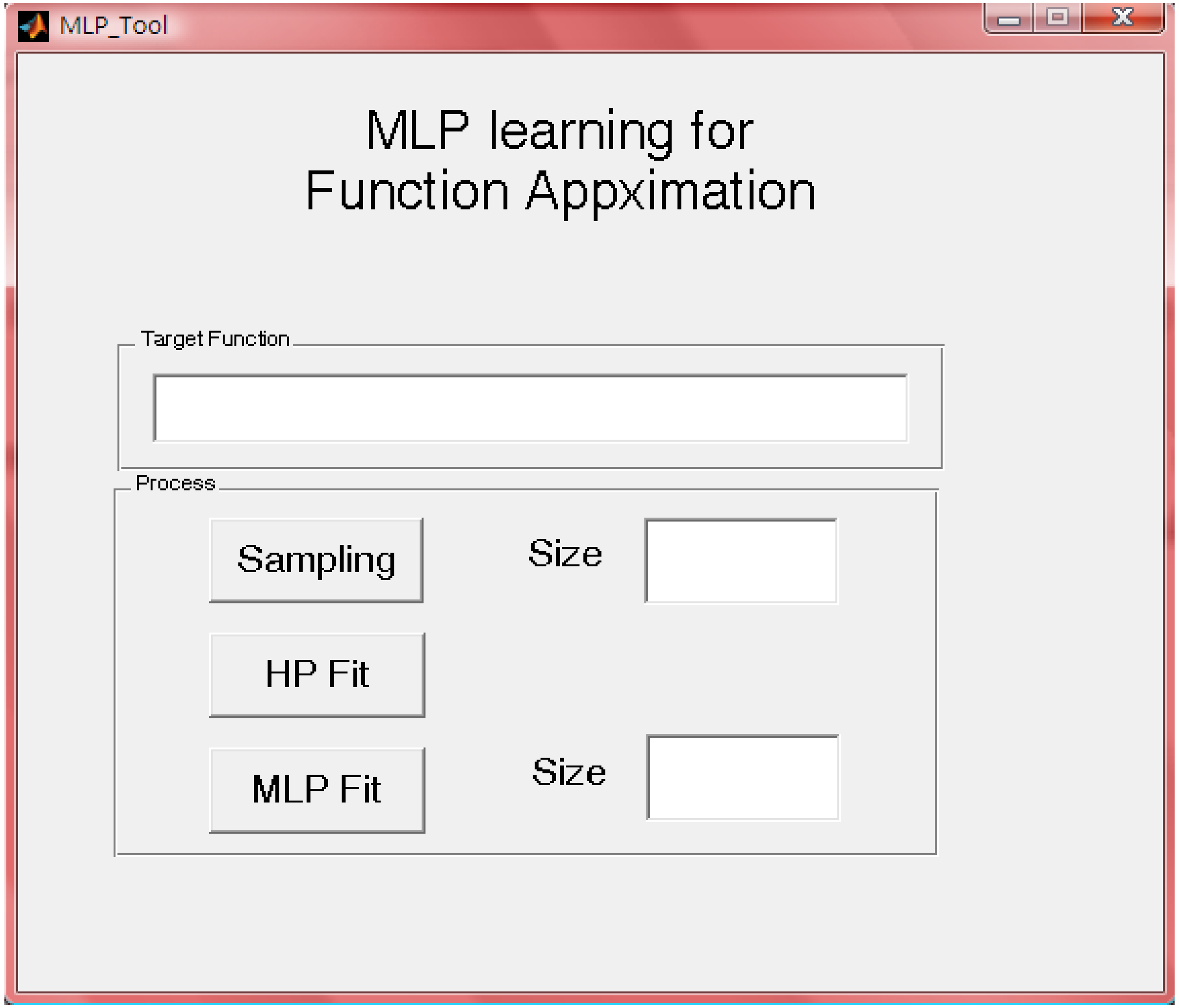
$$f(x_i;\theta) = \sum_{m=1}^{M} r_m \tanh(\mathbf{a}_m^T \mathbf{x} + b_m) + r_0$$

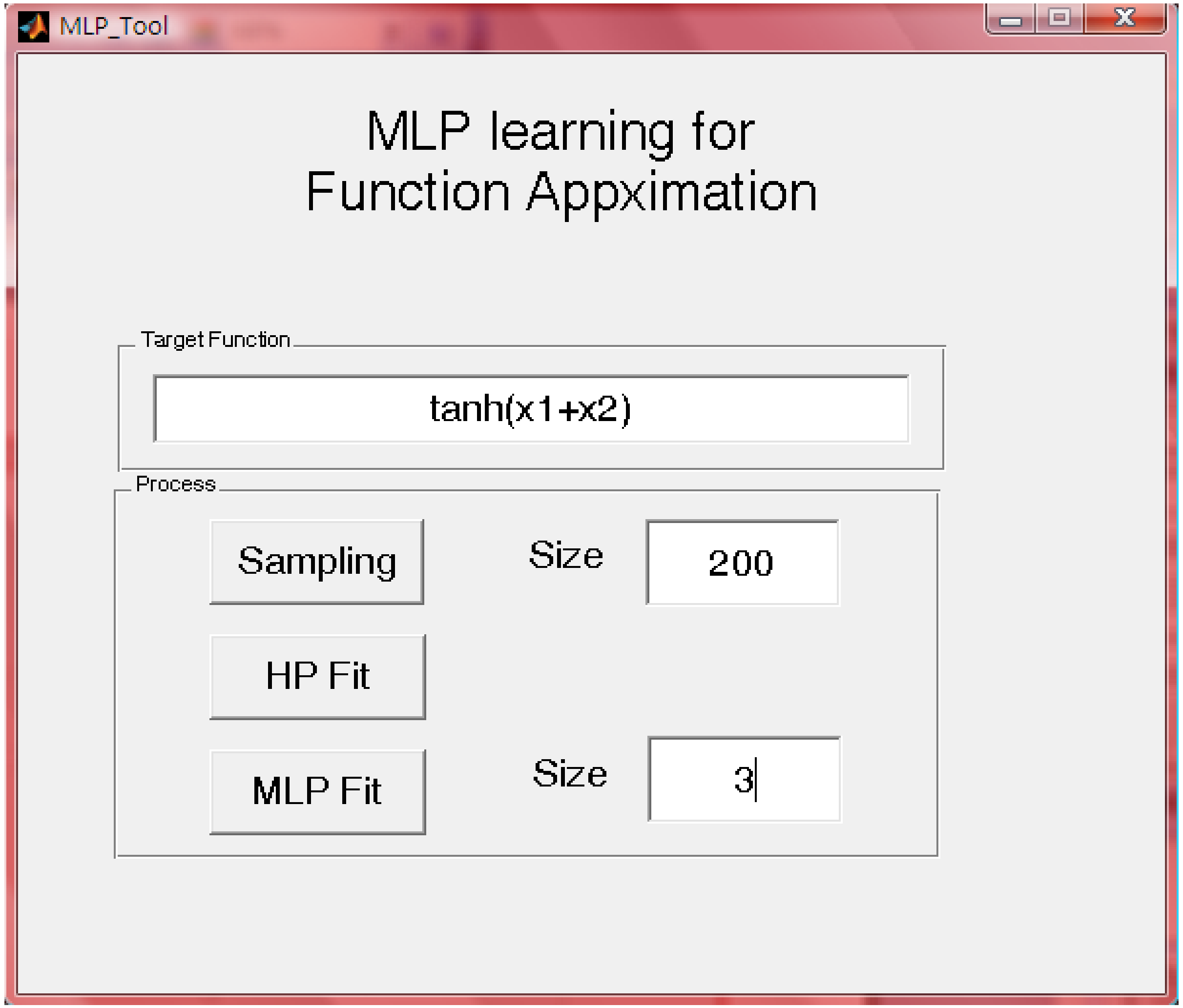
$$\theta = \{\mathbf{a}_m\} \cup \{b_m\} \cup \{r_m\}$$



MLP_Tool.fig

MLP_Tool.m




MLP_Tool
MLP learning for
Function Appximation
Target Function
tanh(x1+x2)
Process
Sampling
Size
200
HP Fit
MLP Fit
Size
3

Mapping

1. Head: function y=eval_MLP2(x,r,a,b)
2. Body:

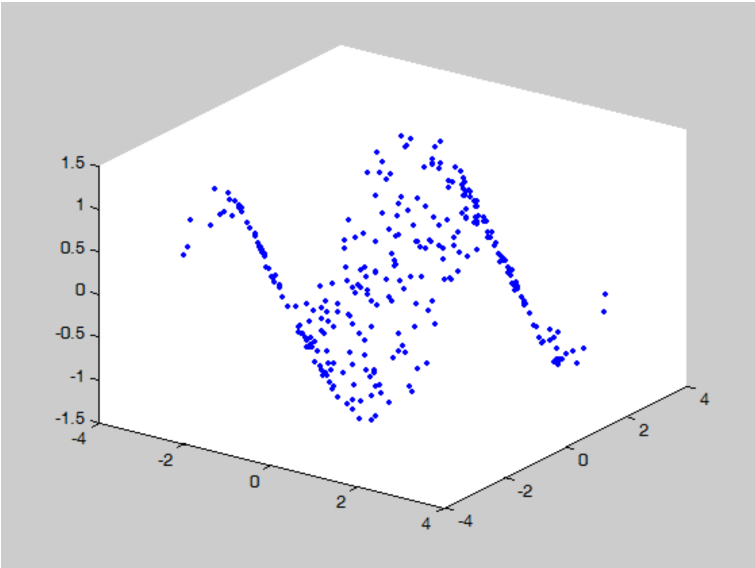
    M=size(a,1)

    y=r(M+1)

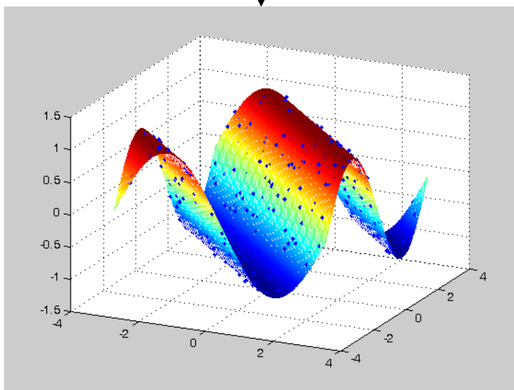
    for m=1:M

        Add r(m)*tanh(a(m,:)*x+b(m)) to y

Function Riddle



DDFA



$$(\mathbf{x}_i, y_i), i = 1, \ldots, n,$$

$$\mathbf{x}_i \in R^d$$

$$E(r, a, b) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i; r, a, b))^2$$

MLP learning

```
[a,b,r]=learn_MLP2(x',y',M);
```

MLP learning

High-dimensional function approximation

The NNSYSID Toolbox

1	eval_MLP2.m	Evaluate MLP function
2	mean_square_error2.m	Calculate E
3	learn_MLP.m	Seek parameters

One-dimensional function approximation

1	eval_MLP.m	Evaluate an MLP function
2	mean_square_error2.m	Calculate E
3	learn_MLP.m	Seek parameters

Example

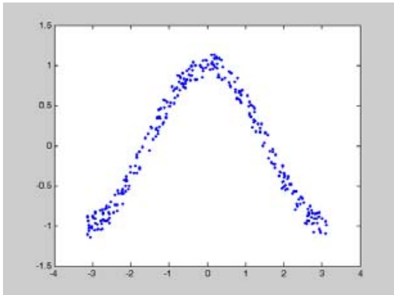
x

Black box
plot1d.m

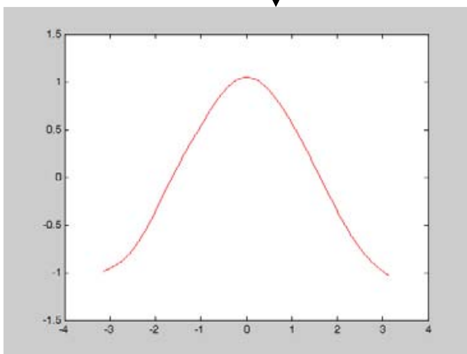
y

observation

Create paired data
sampling.m



DDFA
learning.m



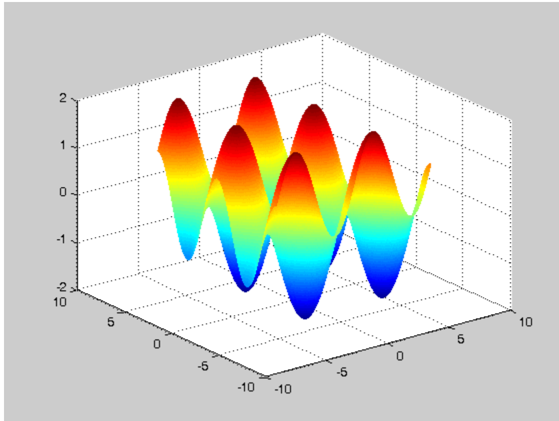
Example

x

Black box
[plot2d.m]

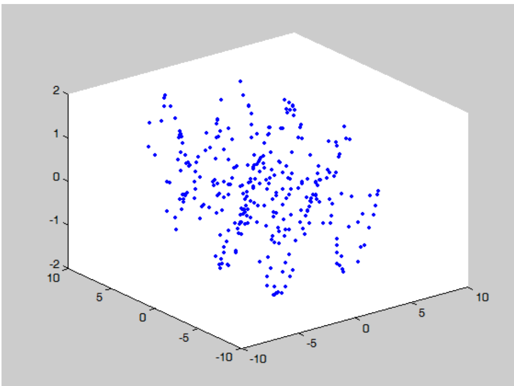
y

observations



y=cos(x1)+sin(x2)

Create paired data
[sampling2.m]



DDFA

M=input('M:');
[a,b,r]=learn_MLP(x',y',M);

E=mean_square_error2(x,y,a,b,r)

Mean Square Error

$$E(r,a,b) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(\mathbf{x}_i; r,a,b)\right)^2$$

Mean square error

- Head: E=mean_square_error2(x,y,a,b,r)
- Body:

E = 0

n = size(x,2)

for i=1:n

a) xi = x(:, i)
b) yi = eval_MLP2(xi,r,a,b)
c) ei=yi-y(i)
d) Add the square of ei to E

E=E/n

DDFA

[demo_fa2d.m]

[fa1d.m]