Matlab gyakorlat III.

Idősor előrejelzés, dinamikus neurális hálózatok, nemellenőrzött tanítás

1. SVM használata idősor előrejelzésre

Az idősor előrejelzés legegyszerűbb módja egy statikus regressziós eljárás alkalmazása megfelelő mintakészlet generálása után. Ez a következőképp történik: a minták bemenetei az idősor egy szakaszát jelölik, a kívánt kimenetei pedig az idősor egy adott értékét jelentik, a szakaszhoz viszonyított állandó eltolással véve. Ez a szakasz jelenti a tanuló rendszer számára az emlékezetet, ez alapján jósolja meg az idősor következő értékét.

Az első példához a Mackey-Glass idősort fogjuk használni, melyről bővebben itt olvashat: http://www.scholarpedia.org/article/Mackey-Glass_equation

Először is vizsgáljuk meg az idősort, és generáljunk belőle mintákat, egy önkényesen megválasztott időablakal.

```
clear all;
close all;
clc;

load('mg.dat','-ascii');
plot(mg);

% Emlékezet hossza.
tw = 10;

% 1D idősorból minta mátrix készítése egy tw+1 széles ablak eltolásával.
samples = zeros(length(mg) - tw, tw + 1);
for i = 1 : tw + 1
    samples(:, i) = mg(i : size(samples ,1) + i - 1);
end
```

Majd egy SVM-nek, regressziós feladatként tanítsuk meg a mintakészletet.

```
% Paraméterek beállítása.
C = 10;
lambda = 1e-6;
epsilon = 0.01;
kerneloption = 0.1;
kernel = 'gaussian';
verbose = 0;
```

```
```

```
% Tanítás.
[xsup,ysup,w,w0] = ...
    svmreg(samples(1:500, 1:end-1), samples(1:500, end), ...
           C, epsilon, kernel, kerneloption, lambda, verbose);

% Szimuláció.
rx = svmval(samples(:, 1 : end - 1), xsup, w, w0, kernel, kerneloption);
plot(mg(tw + 1 : end), 'b'); hold on; plot(rx, 'r'); hold off;


% Hiba számítása (tanító + tesztkészlet együtt).
mse = mean((mg(tw+1:end) - rx).^2);
disp(mse)
```

Az eredmények értékelésénél érdemes a numerikusan számított átlagos négyzetes hibát figyelni, mivel a kirajzolt grafikonon az idősor pár mintával történő elcsúszásait sokkal nehezebb észrevenni.

2. Lipschitz-index számítás

Idősor előrejelzési feladat esetén első lépés a modell kiválasztása. Ehhez első lépésként a fokszámot határozzuk meg. Ehhez tudjuk felhasználni az NNSYSID toolbox *lipschit* függvényét.

A Lipschitz-index definíciója (a „Neurális hálózatok” könyvből):

$$Lq^{(N)} = \left(\prod_{k=1}^{p} \sqrt{N} q^{(N)}(k) \right)^{1/p}$$

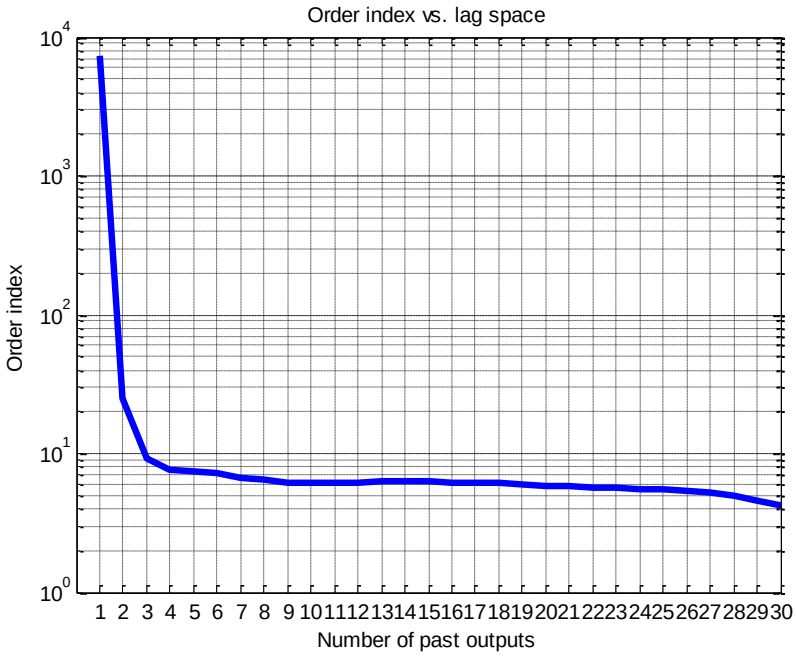
ahol $q^{(N)}(k)$ a

$$q_{ij}^{(N)} = \frac{y(i) - y(j) \vee}{\|x(i) - x(j)\|} i \neq j; i, j = 1,2, \dots,$$

Lipschitz hányadosok közül a *k*-adik legnagyobb érték, *N* a bemeneti változók száma (a regresszorvektor dimenziója), *p* pedig egy alkalmasan megválasztott pozitív szám, rendszerint p = 0,01P ~ 0,02P. (*P* a szokásos jelölésnek megfelelően a tanítópontok száma).

A Lipschitz-index számításához az alábbiakat kell begépelni, ezzel azt mondjuk meg a függvénynek, hogy idősorral dolgozunk, és 1 és 30 között nézze meg a lehetséges fokszámokat. Azt az értéket célszerű választani, ahol a törés található, ez az ábrán 3-nál van.

```
lipschit([],mg,0,1:30);
```



3. SVM idősor előrejelzés, a megfelelő hosszúságú emlékezettel

Futtassuk le most újra a mintagenerálást és tanítást, és hasonlítsuk össze az eredményeket, de immár a Lipschitz-indexből meghatározott időablak-hosszal.

```
% Emlékezet hossza.
tw = 3;

% 1D idõsorból minta mátrix készítése egy tw+1 széles ablak eltolásával.
samples = zeros(length(mg) - tw, tw + 1);
for i = 1 : tw + 1
    samples(:, i) = mg(i : size(samples ,1) + i - 1);
end

% Tanítás.
[xsup,ysup,w,w0] = ...
    svmreg(samples(1:500, 1:end-1), samples(1:500, end), ...
           C, epsilon, kernel, kerneloption, lambda, verbose);

% Szimuláció.
rx2 = svmval(samples(:, 1 : end - 1), xsup, w, w0, kernel, kerneloption);
plot(mg(tw + 1 : end), 'b'); hold on; plot(rx2, 'r');  hold off;

% Hiba számítása (tanító + tesztkészlet együtt).
mse = mean((mg(tw+1:end) - rx2).^2);
disp(mse)
```

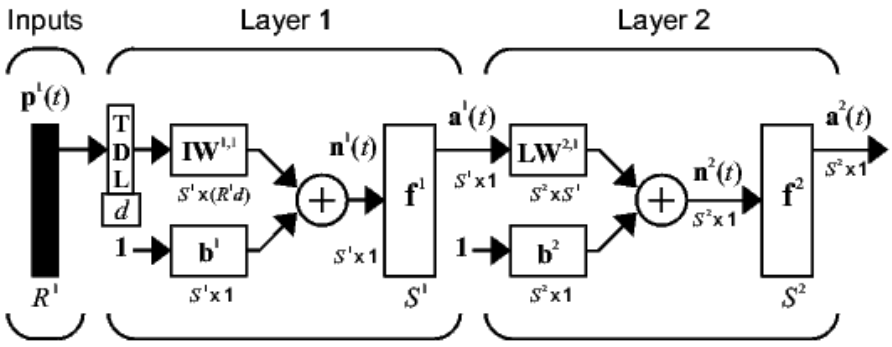
Látható, hogy az így kapott MSE kisebb, mint a *tw*=10 esetben. Ez a túltanulással magyarázható, a nagyobb időablak nem biztos, hogy jobb eredményre vezet. Érdemes lehet még kipróbálni az 1, illetve 2 hosszú időablakokat is, szem előtt tartva, hogy 1 minta alapján hogyan (nem) lehet idősort előrejelezni.

4. Idősor előrejelzés dinamikus neurális hálózattal

A *newfftd* hálóarchitektúrával olyan dinamikus rendszert hozhatunk létre, melynek a bemenetén egy általunk specifikált késleltető lánc van. Először a Mackey-Glass idősorral fogjuk tanítani a hálót.

```
clear all;
close all;
clc;

load('mg.dat');
tw=3;
%a NN toolbox a vektorok sorozatát cella tömbbel jelöli, ezér kell áttérni
%erre, csak az els? 500 pontot használjuk tanításra, mint az SVM esetén
%tettük
y = con2seq(mg(1:500)');
%bemeneti és kimeneti vektor ugyanaz: y, y
%csak a bemenet késleltetve van: [1:3]
%és 5 neuron van az egy rejtett rétegben
net = newfftd(y,y,[1:tw],5);

p = y(tw+1:end);
t = y(tw+1:end);
Pi = y(1:tw);

%a szokásos bemenet, kívánt kimenet pár mellé még fel kell tölteni a
%késleltet? sort is adatokkal, az els? pár minta tanításához. Ez a Pi
net = train(net,p,t,Pi);
```

Itt az [1:3] azt jelenti, hogy a *t*-edik pillanat előrejelzéséhez a *t*-1,*t*-2 és *t*-3 értékeket használjuk fel. Ezért tehetjük meg azt, hogy ugyanazt adjuk bemenetként és várt kimenetként is a hálónak.



A tanítás után le kell kérni az eredményeket és ki kell rajzoltatni azokat.

```
test = con2seq(mg');
%a sim függvényben is a trainhez hasonlóan, inicializálni kell a
%késleltet?láncot
r = sim(net,test(tw+1:end),test(1:tw));
%visszatérés cella tömbr?l vektorra
r = seq2con(r);
r = r{1};
figure();
plot(mg(tw+1:end),'b');hold on;plot(r,'r');hold off;

% Hiba számítása (tanító + tesztkészlet együtt).
mse = mean((mg(tw+1:end) - r').^2);
disp(mse)
```

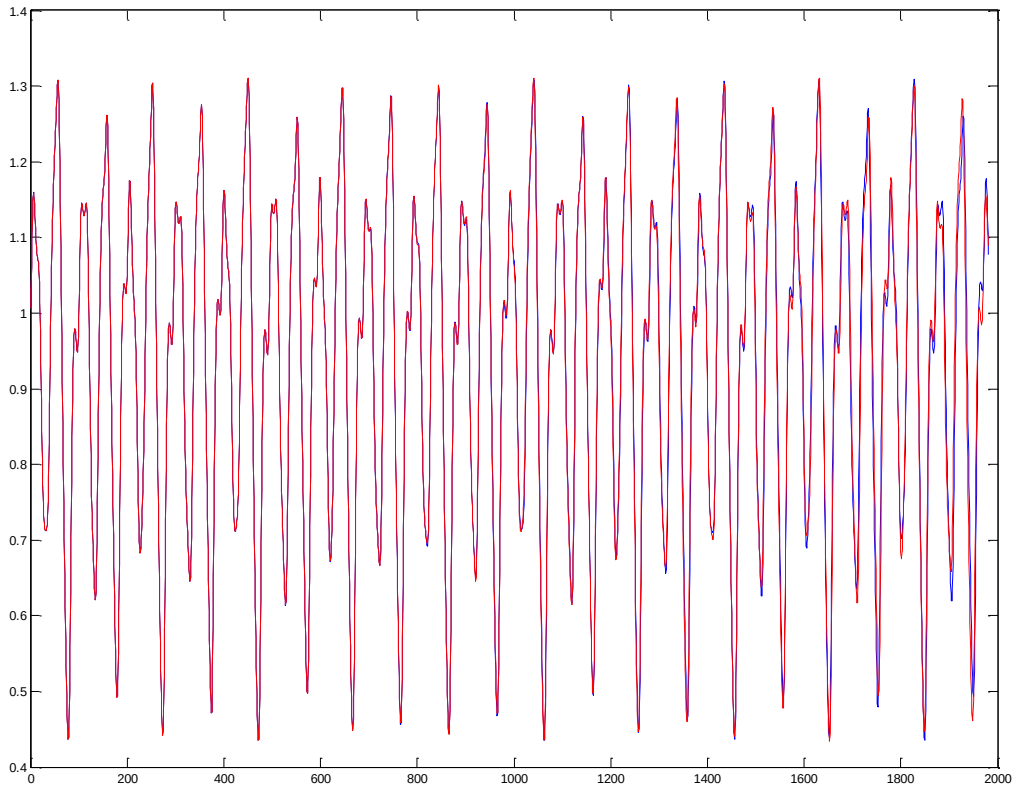
Jól látszik, hogy a háló egylépéses előrejelzés esetén tökéletesen megtanulta a feladatot. Most vizsgáljuk meg úgy is, hogy többlépéses előrejelzést csinálunk, iteratívan felhasználva a már előre jelzett mintákat.

```
clear all;
close all;
clc;

load('mg.dat');
y = con2seq(mg(1:500)');
tw = 20;

net = newfftd(y,y,[1:tw],5);
p = y(tw+1:end);
t = y(tw+1:end);
Pi = y(1:tw);
net.trainParam.epochs = 250;
net = train(net,p,t,Pi);
```

Majd szimuláljuk le a teljes idősorra az előrejelzést. Vegyük észre, hogy csak az első 500 minta alapján tanítottunk, és a teszt során is csak ezt használtuk fel, ez alapján jeleztük előre mind a 2000 értéket. Lényeges különbség viszont, hogy most az időablak hossza 20 minta.

```
%lefuttatjuk az elõrejelzést a tanítókészleten
r = sim(net,p,Pi);
r = seq2con(r);
r = r{1};

%az utolsó értékek maradnak a késleltetõláncban
Pi = p(end-tw+1:end);


%a hosszútávú elõrejelzést tároló tömb
long_term = zeros(1501,1);
```

```
%mintánként végigmegyünk, az éppen elõrejelzett mintát ismertnek tekintve,
%hozzátoldjuk a bemenethez a késleltetõ láncba
for i = 1 : 1501
    %mivel csak egy skalár bemenetre jelzünk elõre, ehhez minden
    %megtalálható a késleltetõláncban, nem kell bemeneti vektor
    act_r = sim(net,{0},Pi);
    act_r = seq2con(act_r);
    act_r = act_r{1};
    %a késleltetõláncba beleshifteljük az új elõrejelzést
    Pi(1:end-1) = Pi(2:end);
    Pi{end} = act_r;

    long_term(i) = act_r;
end
figure();
plot(mg(tw+1:end), 'b'); hold on; plot([r, long_term'],'r');hold off;

% Hiba számítása (tanító + tesztkészlet együtt).
mse = mean((mg(tw+1:end) - [r, long_term']').^2);
disp(mse)
```



5. Dinamikus rendszer modellezése neurális hálózattal

Generáljunk először is egy dinamikus rendszerből mérési adatokat, bemeneti gerjesztést, és az ehhez tartozó kimeneten mért választ, mentsük el a „mért” adatokat egy fájlba.

```
% Generate dynamic system.
u = zeros(1, 1500);
y = zeros(1, 1500);

for n = 3 : length(u) - 1
    if (n <= 500)
        % Use a simple sinus wave for training.
        u(n) = sin(2 * pi * n / 250);
    else
        % Linear combination of sine waves with different periods for
%testing.
        u(n) = 0.1 * sin(2 * pi * n / 1000 * 13) + ...
               0.3 * sin(2 * pi * n / 1000 * 29) + ...
               0.15 * sin(2 * pi * n / 1000 * 81);
    end
    % A "random" nonlinear function of u.
    g_u = 0.5 * (sin(pi * u(n)))^3 - ...
          2 / (u(n)^3 + 2) - ...
          0.1 * cos(4 *  pi * u(n)) + ...
          1.125;
    % Output is the function of the previous u and y;
    y(n + 1) = 0.3 * y(n) + 0.6 * y(n - 1) + g_u;
end

% Display and save results.
plot(u); hold on; plot(y,'r'); hold off;
y = y';
u = u';
save ds.mat y u
```

5.1. NFIR

Először próbáljuk meg előrejelezni a kimenetet pusztán a bemeneti idősor ismerete alapján.

```
% Predict the DS with an NN getting only the input.
clear all
load('ds.mat');

% Memory size.
tw = 20;

% Pick training samples.
input = con2seq(u(1 : 500)');
output = con2seq(y(1 : 500)');
p = input(tw + 1 : end);
t = output(tw + 1 : end);
Pi = input(1 : tw);
d = 1 : tw;
```

```
% Setup network.
net = newfftd(input, output, d, 10);
net.trainFcn = 'trainbr';

% Train network (net, input, output, inital history).
net = train(net, p, t, Pi);

% Eval network.
p = con2seq(u(tw + 1 : end)');
Pi = con2seq(u(1 : tw)');
r = sim(net, p, Pi);
r = seq2con(r);
r = r{1};

% Display result and get error.
figure();
plot(y(tw + 1:end), 'b'); hold on; plot(r, 'r'); hold off;
mse = mean((y(tw + 1 : end) - r').^2);
disp(mse);
```

Látható, hogy azon a részen, ahol nem tanítottunk, és megváltozott a gerjesztő jel, a háló nem tudja jól modellezni a rendszert.

5.2. Idősor előrejelzés

Lássuk, hogy mi történik, ha csak a korábbi kimenetek ismeretében próbáljuk megbecsülni a rendszer kimenetét, mint az idősor előrejelzés esetén.

```
% Predict the DS with an NN getting only the output (as in time series).
clear all
load('ds.mat');

% Memory size.
tw = 20;

% Pick training samples.
input = con2seq(y(1 : 500)');
output = con2seq(y(1 : 500)');
p = input(tw + 1 : end);
t = output(tw + 1 : end);
Pi = input(1 : tw);
d = 1 : tw;

% Setup network.
net = newfftd(input, output, d, 10);
net.trainFcn = 'trainbr';

% Train network (net, input, output, inital history).
net = train(net, p, t, Pi);

% Eval network.
p = con2seq(y(tw + 1 : end)');
Pi = con2seq(y(1 : tw)');
r = sim(net, p, Pi);
```

```
r = seq2con(r);
r = r{1};

% Display result and get error.
figure();
plot(y(tw + 1:end), 'b'); hold on; plot(r, 'r'); hold off;
mse = mean((y(tw + 1 : end) - r').^2);
disp(mse);
```

Láthatóan jóval jobb eredményt értünk el ilyen módon, a rendszer dinamikáját jóval pontosabban tudjuk a kimenet megfigyelésével követni. Viszont a bemenet ismerete nélkül a dinamika változására nem tudunk következtetni.

5.3. NARX

Lássuk, hogyan teljesít egy neurális hálózat, ha mind a korábbi bemenet, mind a korábbi kimenet értékeit is ismerjük.

```
% Predict the DS with an NN getting both input and output.
clear all
load('ds.mat');

% Max memory size.
tw = 2;

% Pick training samples.
input = con2seq(u(1 : 500)');
output = con2seq(y(1 : 500)');
p = input(tw + 1 : end);
t = output(tw + 1 : end);
Pi = [input(1 : tw); output(1 : tw)];

% Input delays.
d1 = 1;

% Output delays.
d2 = 1 : tw;

% Setup network.
narx_net = newnarxsp(p, t, d1, d2, 10);
narx_net.trainFcn = 'trainbr';

% Train network (net, input, output, inital history).
narx_net = train(narx_net, [p; t], t, Pi);

% Eval network.
p = con2seq(u(tw + 1 : end)');
t = con2seq(y(tw + 1 : end)');
Pi = [con2seq(u(1 : tw)'); con2seq(y(1 : tw)')];
r = sim(narx_net, [p; t], Pi);
r = seq2con(r);
r = r{1};
```

```
% Display result and get error.
figure();
plot(y(tw + 1:end), 'b'); hold on; plot(r, 'r'); hold off;
mse = mean((y(tw + 1 : end) - r').^2);
disp(mse);
```

Ezek alapján így egy nagyságrenddel jobb eredményt tudtunk elérni.

Nemellenőrzött tanulás

6. K-means

Először is töltsünk be egy példa adatsort, és vizsgáljuk meg az adatpontokat tartalmazó mátrix méretét.

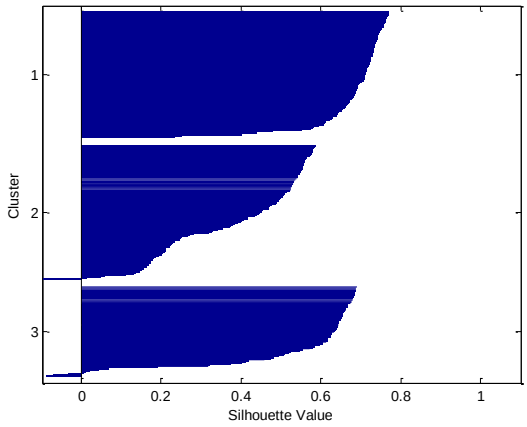
```
load kmeansdata;
size(X)
```

Ezen az 560 pontból álló, 4 dimenziós adathalmazon futtassunk le rajta egy K-means osztályozást, 3 osztályra, háztömb (L1) távolságmértékkel.

```
idx3 = kmeans(X,3,'distance','city');
```

Az *idx3* tömb fogja tartalmazni egyenként az 560 pont osztályát. Az osztályozás minőségének ellenőrzéséhez használjuk a *silhouette* diagram típust, mivel a 4 dimenziós adatot csak nehezen tudnánk 2D-ben ábrázolni. Ez a diagram megadja, hogy az egyes pontok mennyire hasonlóak (egy megadott mérték szerint) a velük egy osztályba sorolt pontokkal, a más osztályba sorolt pontokkal szemben, -1 és 1 közé normálva. Ezeket az értékeket egy függőleges tengely mentén ábrázoljuk, osztályonként növekvő sorrendben. A kicsi vagy negatív érékek azt jelzik, hogy az adott osztály nem különül el megfelelően a többi osztálytól.

```
[silh3,h] = silhouette(X,idx3,'city');
xlabel('Silhouette Value');
ylabel('Cluster');
```



Futtassuk le a programot a következő változtatásokkal:

```
idx4 = kmeans(X,4,'distance','city', 'display','iter');

[silh4,h] = silhouette(X,idx4,'city');
xlabel('Silhouette Value');
ylabel('Cluster');
```

Egyrészt így láthatóak a K-means algoritmus egyes iterációinak paraméterei, másrészt a 4 osztállyal már jóval jobb eredményeket érünk el.

A biztonság kedvéért vizsgáljuk meg az 5 osztályt is.

```
idx5 = kmeans(X,5,'distance','city');

[silh5,h] = silhouette(X,idx5,'city');
xlabel('Silhouette Value');
ylabel('Cluster');
```

Itt újra megjelenik negatív érték is a diagramon. Objektívebb összehasonlításhoz kiszámolhatjuk a 3 diagram értékeinek átlagát:

```
mean(silh3)
mean(silh4)
mean(silh5)
```

Ez alapján is kijelenthető, hogy a 4 osztályba sorolás a legjobb erre az adathalmazra.

7. Főkomponens analízis (PCA)

7.1 Egyszerű adathalmaz

Generáljunk 500 pontot egy kétdimenziós normális eloszlásból, majd alkalmazzuk a főkomponens analízist a ponthalmazra. A Matlabban erre a célra a *pca* függvény szolgál. Alapvetően először zérus

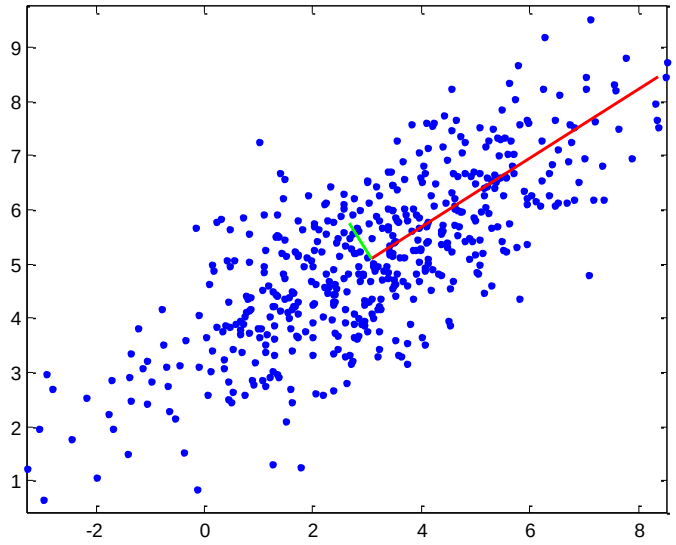
várható értékűvé, és egységnyi szórásúvá transzformálja az adatot, majd (az alapértelmezett beállítás esetén) a szinguláris értékek szerinti felbontást (SVD) végez az adatmátrixból (**X**) számolt kovarianciamátrixon ($\mathbf{X}^T\mathbf{X}$).

```
%vegyünk 500 mintát egy többváltozós normális eloszlásból
X = mvnrnd([3,5],[4, 2; 2, 2],500);

%majd jelenítsük meg a pontokat
plot(X(:, 1), X(:, 2), '.');
axis equal;

%végezzük el a fõkomponens analízist
%a C mátrixba kerülnek a fõkomponensek (sajátvektorok),
%S-be a fõkomponens térben a pontok koordinátái,
%L-be pedig a fõkomponensek mentén a mintahalmaz varianciája (sajátértékek)
[C, S, L] = pcas(X); %saját implementáció, a 2007b verzió PCA-ja eltérő
szignatúrájú
```

A három kimeneti mátrix a következő jelentéssel bír. **C**-be kerülnek a főkomponensek (vagyis a kovarianciamátrix sajátvektorai), sajátértékek szerint csökkenő sorrendben. Az **S** mátrixba kerülnek a pontok főkomponens térben vett koordinátái. Az eredeti térbe történő visszaszámítás a $\mathbf{SC}^T$ kiszámításával lehetséges. Az L vektorba pedig a mintahalmaznak a főkomponensek mentén vett varianciái kerülnek, vagyis a kovarianciamátrix sajátértékei (szintén csökkenő sorrendben).

Végezetül rajzoljuk be az ábrába a főkomponenseket.

```
%rajzoljuk ki a fõkomponenseket
mu = mean(X);
hold on;
line([mu(1), mu(1) + 1 * L(1) * C(1,1)'],[mu(2), mu(2) + 1 * L(1) *
C(2,1)'],'Color','r','LineWidth',2);
line([mu(1), mu(1) + 1 * L(2) * C(1,2)'],[mu(2), mu(2) + 1 * L(2) *
C(2,2)'],'Color','g','LineWidth',2);

hold off;
```

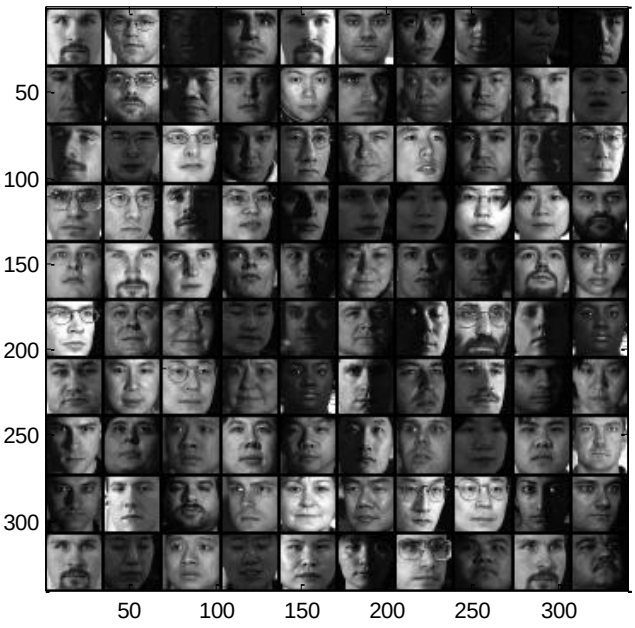


7.2 Eigenfaces

Vizsgáljuk meg a PCA-t egy igazi feladatra. Számos, arcokról készült, gépi tanulási célokra szánt adathalmaz létezik. Az egyik ilyen a Carnegie Mellon University PIE Database, melynek számos előfeldolgozott változata érhető el az interneten. Egy ilyet fogunk most használni, mely 32x32 pixeles, szürkeárnyalatos képeket tartalmaz.

http://vasc.ri.cmu.edu/idb/html/face/

http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html



Először is készítsünk egy függvényt, mellyel könnyedén megjeleníthetünk 100 arcot (lásd: fenti ábra). Minden arcot egy 1024 hosszú vektorként kezelünk, mely sorfolytonosan tartalmazza a kép pixeleit.

displayFace.m :

```
function displayFace(data)
    img=zeros((32+2)*10,(32+2)*10);
    for y=0:9
        for x=0:9
            img(2+y*34:2+y*34+31,2+x*34:2+x*34+31) = ...
                reshape(data(x+y*10+1,:),32,32);
        end
    end
    colormap(gray);
    imagesc(img);
    axis square;
end
```

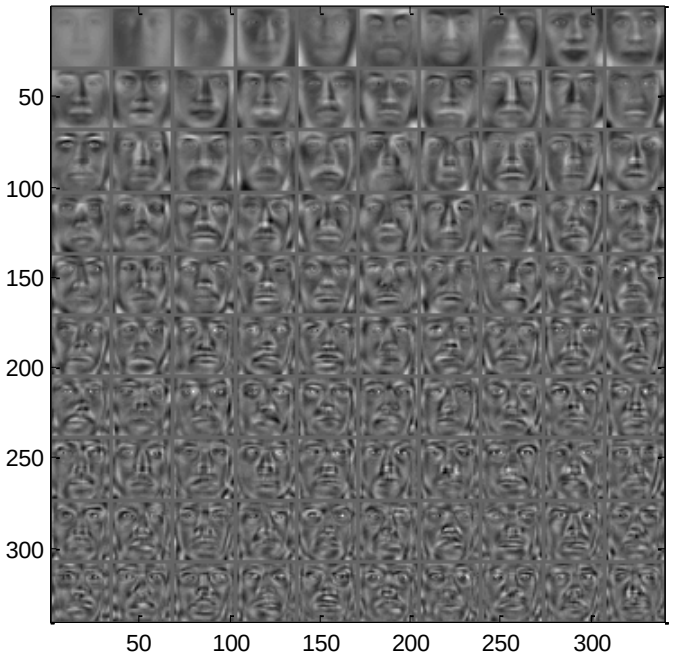
Ezt követően töltsük be az adatokat, majd jelenítsük is meg az első 100 arcot:

```
%PIE database, CMU
load('faces.mat');
%vegyük az elsõ 6000 elemet, memória okok miatt
faces = faces(1:6000,:);
%jelenítsünk meg párat az arcok közül
displayFace(faces);
```

Az előbb megismert módon végezzük el a főkomponens analízist, és jelenítsük meg a főkomponens arcokat:

```
%végezzük el a fõkomponens analízist
[C, S, L] = pcas(faces);
m=mean(faces);

%jelenítsük meg a 100 legnagyobb sajátértékû fõkomponenst
figure;
displayFace(C');
```

Vegyük észre, hogy az átlagos arcot is kiszámoltuk, ennek később lesz szerepe az arcok visszaállításakor, a megjelenítés miatt.



Vizsgáljuk meg, hogy hogyan csökken a variancia az egymás utáni fõkomponensek mentén:

```
%vizsgáljuk meg a sajátértékek csökkenését
figure;
plot(L);
```

Látható, hogy viszonylag kevés komponens mentén nagy a variancia, a többi mentén pedig szinte elhanyagolható. Próbáljuk meg elhagyni a fölöslegesnek tűnő komponenseket, ilyen módon adattömörítést elérve. Rekonstruáljuk az arcokat és hasonlítsuk össze az eredeti képpel.

```
%állítsuk helyre a 100 legnagyobb sajátértékû fõkomponens alapján
%az eredeti képeket
%(1024 dimenzióból 100 dimenzióra tömörítettük az adatot)
n=100;
subplot(1,2,1);
displayFace(faces);
subplot(1,2,2);
displayFace(S(:,1:n)*C(:,1:n)'+ones(6000,1)*m);
```

Vegyük észre, hogy az előzőekben kiszámolt átlagos arcot minden rekonstruált archoz hozzáadjuk. Mivel a PCA számításakor a várható érték zérusra változott minden dimenzió (pixel) mentén, ily módon biztosítani tudjuk, hogy a pixelek egymáshoz képesti fényességét is visszaállítsuk, ami a pixelek képként történő értelmezését teszi lehetővé.