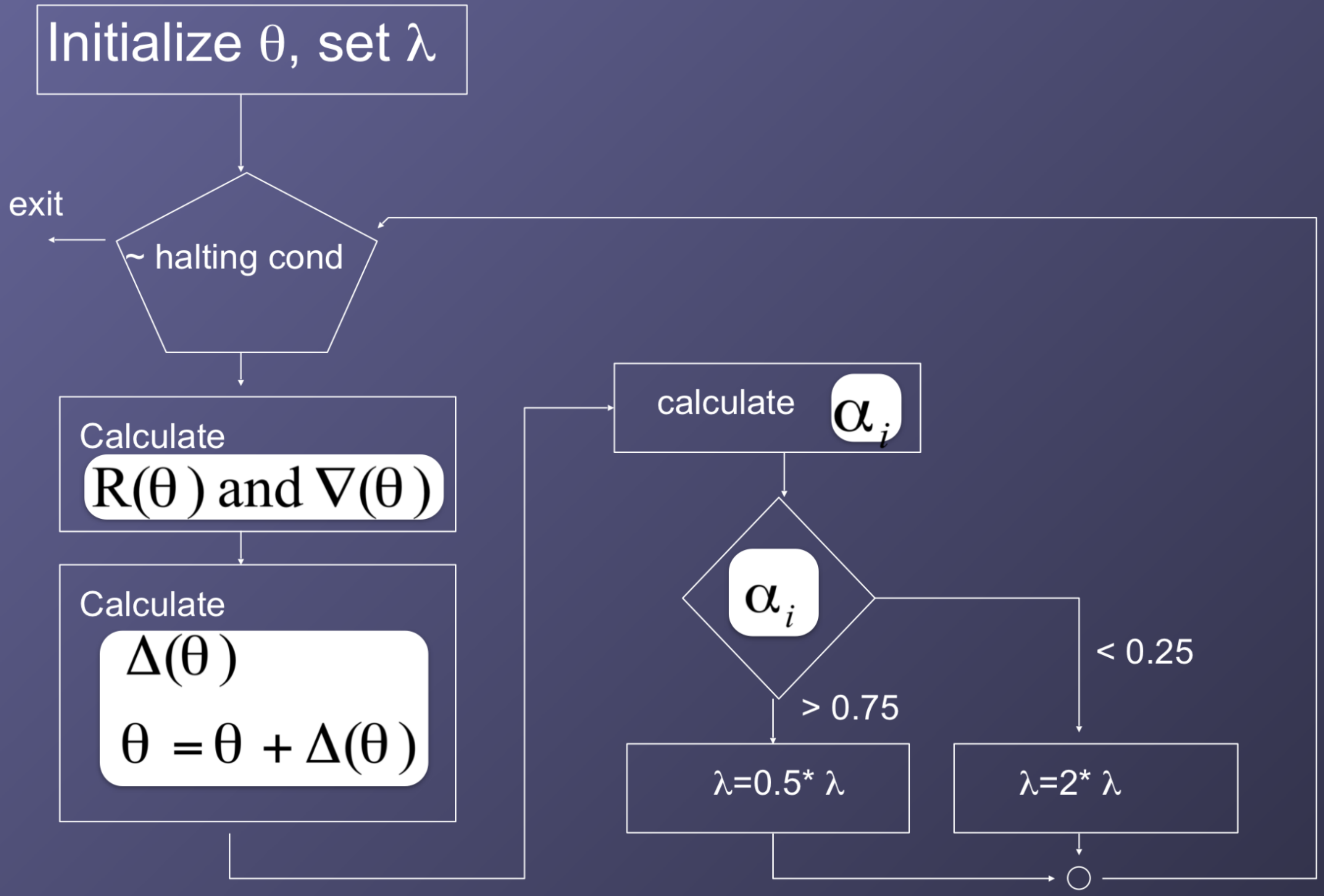
Levenberg-Marquardt Learning of Radial Basis Functions

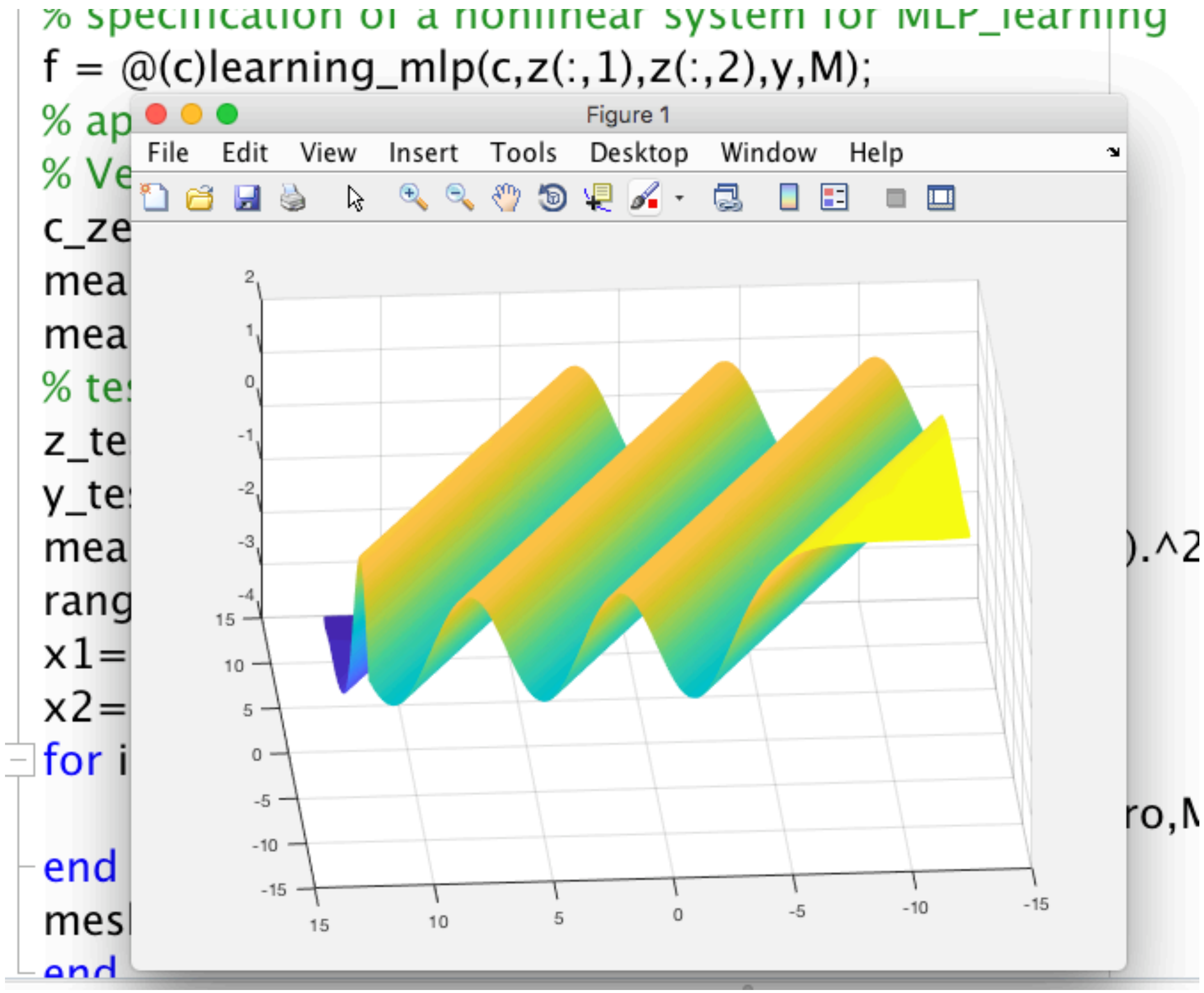

Initialize θ, set λ
exit
~ halting cond
Calculate
R(θ) and ∇(θ)
Calculate
Δ(θ)
θ = θ + Δ(θ)
calculate α_i
α_i
< 0.25
> 0.75
λ=0.5* λ
λ=2* λ

Levenberg-Marquardt method

$\Delta\theta_i$ is determined by

$$(R(\boldsymbol{\theta}_i) + \lambda\mathbf{I})\boldsymbol{\Delta\theta}_i = -\boldsymbol{\nabla}(\boldsymbol{\theta}_i)$$

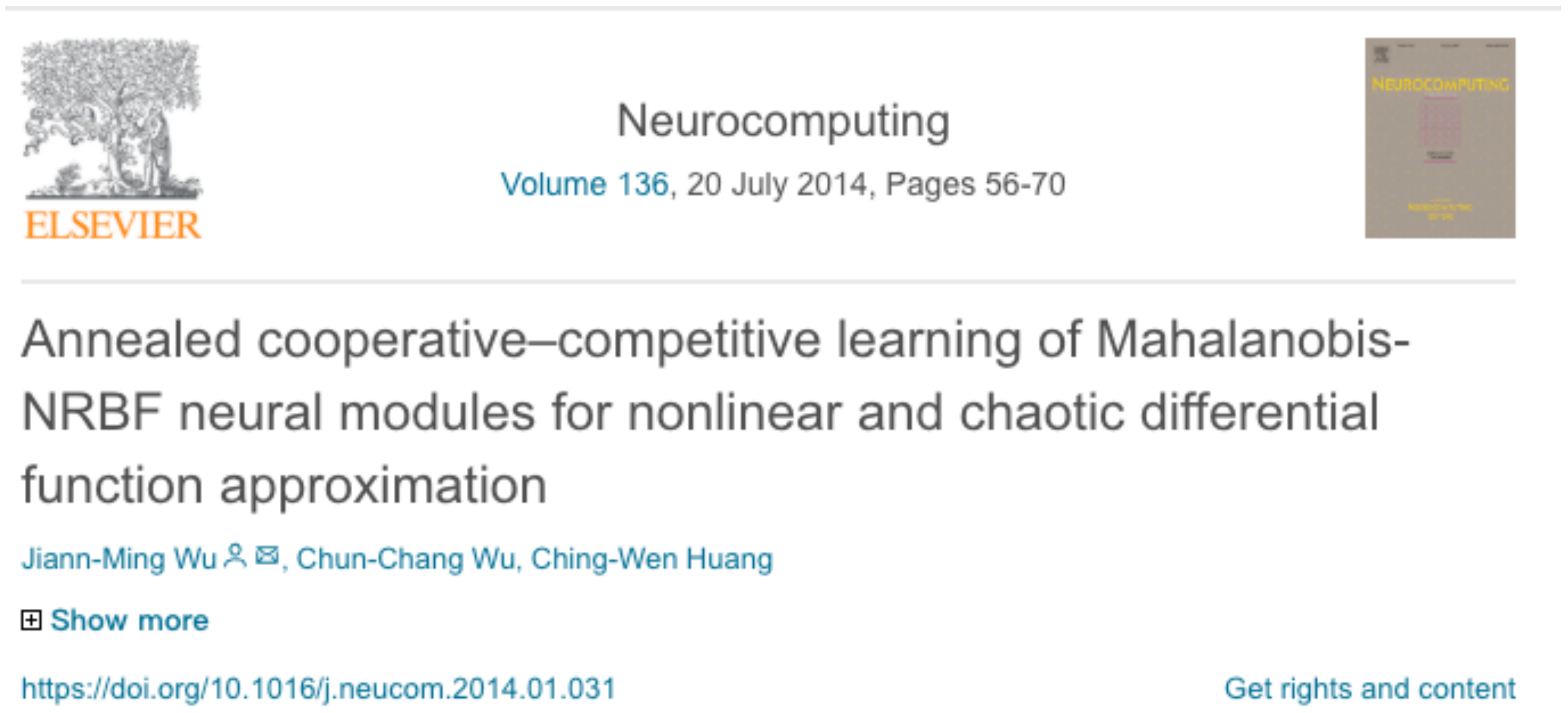
Levenberg-Marquardt learning of MLP



```
% specification of a nonlinear system for MLP_learning
f = @(c)learning_mlp(c,z(:,1),z(:,2),y,M);
% ap
% Ve
c_ze
mea
mea
% te
z_te
y_te
mea                                                    ).^2
rang
x1=
x2=
for i
                                                       ro,M
end
mes
end
```

```
range=4*pi;
x1=-range:0.1:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=g_hat2(x1(i)*ones(length(x2),1),x2',c_zero,M);
end
mesh(x1,x2,C);
```

Radial Basis Functions



Neurocomputing

Volume 136, 20 July 2014, Pages 56-70

Annealed cooperative–competitive learning of Mahalanobis-NRBF neural modules for nonlinear and chaotic differential function approximation

Jiann-Ming Wu, Chun-Chang Wu, Ching-Wen Huang

⊞ Show more

https://doi.org/10.1016/j.neucom.2014.01.031

Get rights and content

```matlab
function D=cross_dis(X,Y)
K=size(Y,1);N=size(X,1);
A=sum(X.^2,2)*ones(1,K);
C=ones(N,1)*sum(Y.^2,2)';
B=X*Y';
D=sqrt(A-2*B+C);
```

$$y(t \mid \theta) = G(\mathbf{x}[t] \mid \theta)$$

$$= w_0 + \sum_{m=1}^{M} w_m \exp\left(-\frac{\left\|\mathbf{x}[t] - \boldsymbol{\mu}_m\right\|^2}{2\sigma_m^2}\right)$$

Network parameter

$$\theta = \{\mathrm{w}_{\mathrm{i}}\}_i \cup \{\boldsymbol{\mu}_i\}_i \cup \{\sigma_i\}_i$$

$\mu_1 \in \mathcal{M}_{2\times 1}$ $\quad \sigma_1 \in R$ $\quad w_1 \in R$

$\vdots \qquad \vdots \qquad \vdots$

$\mu_M \in \mathcal{M}_{2\times 1}$ $\quad \sigma_M \in R$ $\quad w_M \in R$

$w_0 \in R$

$$\theta = [\mu_1^T \; \mu_2^T \; \cdots \; \mu_M^T \; \sigma_1 \cdots \sigma_M \; w_1 \cdots w_M \; w_0]$$

```
function demo_LM_RBF()
syms c   % adaptable parameters in an MLP network
M=40;d=2;N=800
% initialization
c0=rand(1,M*(d+1)+M+1)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
z=rand(N,2)*2*pi-pi;
y=g_sin(z(:,1),z(:,2))+rand(N,1)*0.02-0.01;
size(y)
plot3(z(:,1),z(:,2),y,'.');
h = g_hat2(z(:,1),z(:,2),c0,M);
% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_RBF(c,z(:,1),z(:,2),y,M);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options);
mean((learning_RBF(c_zero,z(:,1),z(:,2),y,M)).^2)
mean((g_hat2(z(:,1),z(:,2),c_zero,M)-y).^2)
% testing phase
z_test=rand(100,2)*2*pi-pi;
y_test=g_sin(z_test(:,1),z_test(:,2));
mean((g_hat2(z_test(:,1),z_test(:,2),c_zero,M)-y_test).^2)
figure
range=2*pi;
x1=-range:0.1:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=g_hat2(x1(i)*ones(length(x2),1),x2',c_zero,M);
end
mesh(x1,x2,C);
end
```

```
% a nonlinear system for MLP_learning

function F = learning_RBF(c,x1,x2,y,M)
A = [x1 x2];
d=2;W=[];
%W=reshape(c(1:M*(d+1)),d+1,M);
for j=1:M
   W=[W c((j-1)*(d)+1:j*(d))'];
end
Y=W';
X=A;
K=size(Y,1);N=size(X,1);
A=sum(X.^2,2)*ones(1,K);
C=ones(N,1)*sum(Y.^2,2)';
B=X*Y';
D=sqrt(A-2*B+C);
sigma=c(M*d+1:M*d+M);
Dsigma=ones(N,1)*sigma;
H=-D./(Dsigma.^2);
V=exp(H);

hh = V*c(M*(d+1)+1:end-1)';
F=hh+ones(length(x1),1)*c(end);
F =F-y;
end


function h = g_sin(x1,x2)
C1=[1 1/2]'; %weight
A = [x1 x2];
h = sin(A*C1); %activation function
end
```