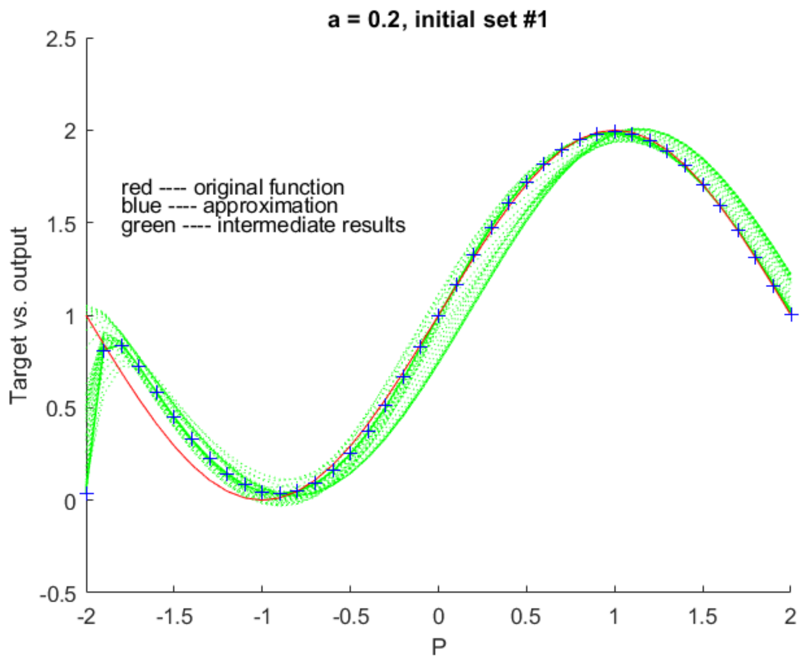
Κεφάλαιο 3

Πρόβλημα 3

3.1 $S^1 = 2$



Εικόνα 3.1: $S^1 = 2, a = 0.2$, *Initial set 1*

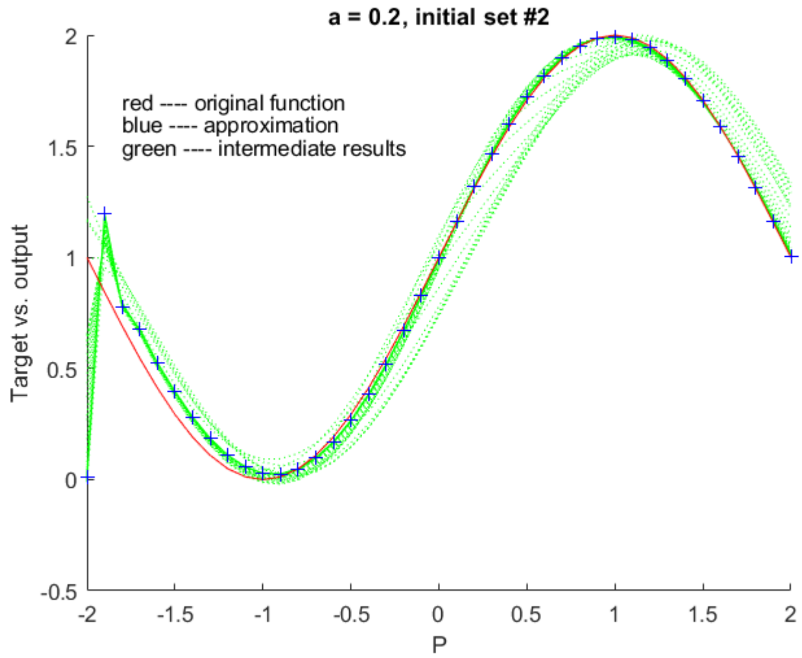
3.2 $S^1 = 10$

The code was modified and tested for $S^1 = 10$ and led us to the following conclusions .

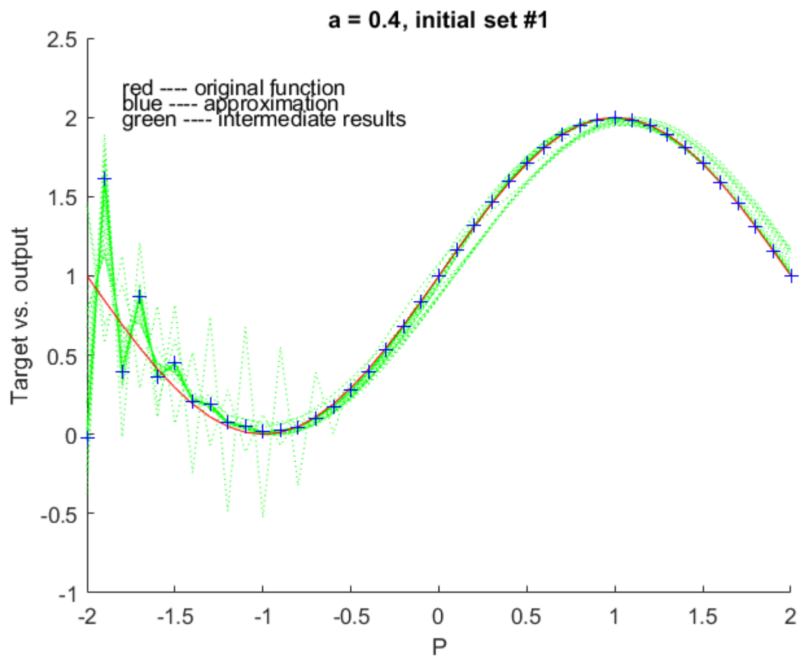
3.3 Discuss the convergence properties of the algorithm as the learning rate changes

1. The higher the learning rate is, the faster the iteration process converges

2. When $a > 0.6$, the solution becomes unstable. e.g.
   When $a = 0.8$ is used, the converging process is very unstable, and takes a long time



Εικόνα 3.2: $S^1 = 2, a = 0.2$, *Initial set 2*

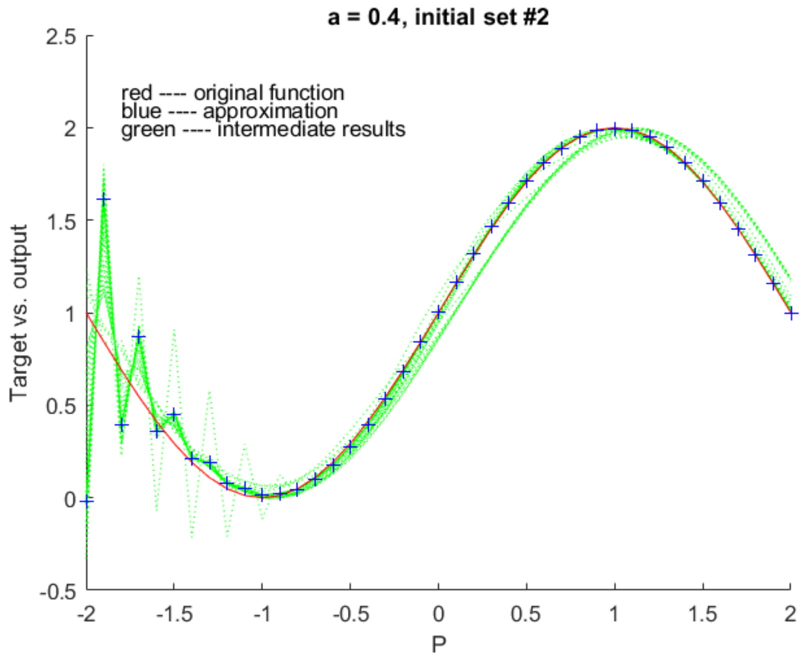


Εικόνα 3.3: $S^1 = 2, a = 0.4$, *Initial set 1*

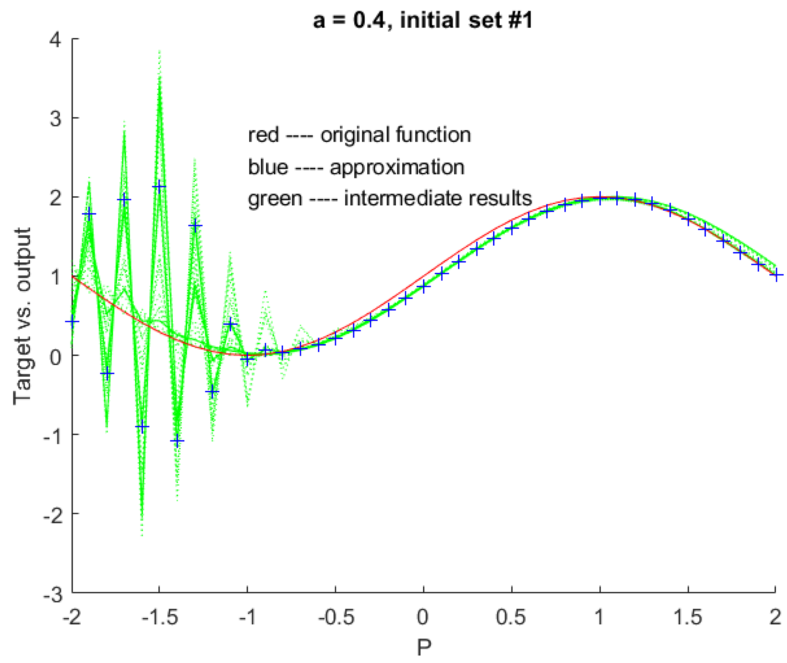
3. The initial weights and biases affects the converging speed and the final results

4. The momentum backpropagation variance would probably be applied successfuly in this model

3.4 MATLAB Code



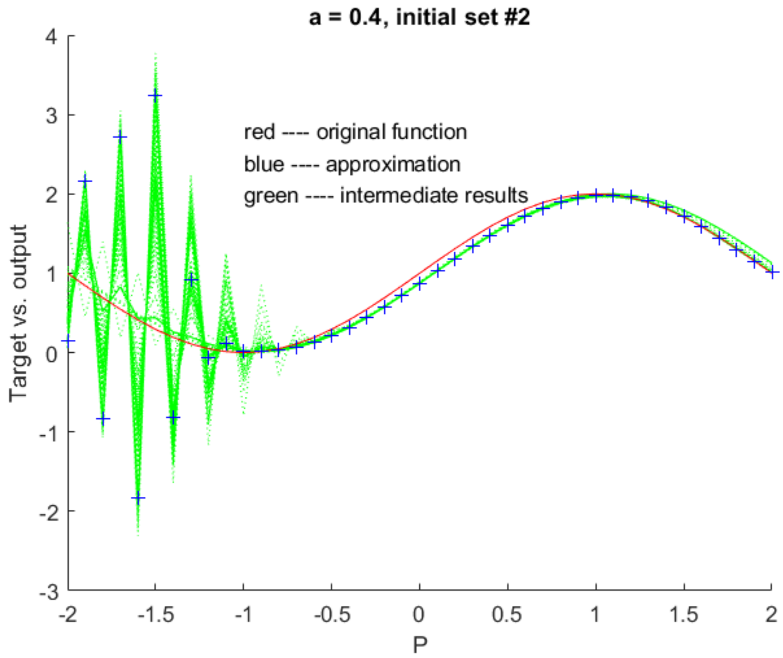
Εικόνα 3.4: $S^1 = 2, a = 0.4$, *Initial set 2*



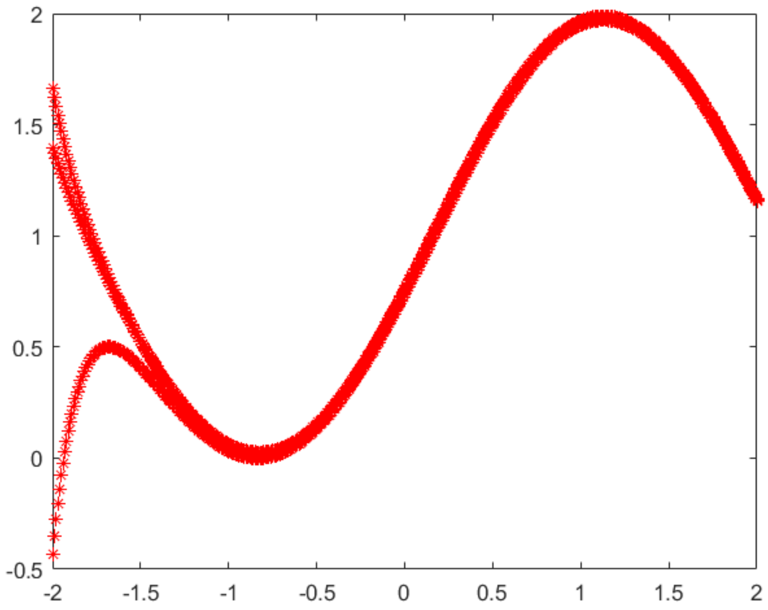
Εικόνα 3.5: $S^1 = 2, a = 0.6$, *Initial set 1*

```
clear
%Initialize data
W1 = rand(2,1) - 0.5;
W2 = rand(1,2) - 0.5;
b1 = rand(2,1) - 0.5;
b2 = rand - 0.5;
a1 = zeros(2,1);

```



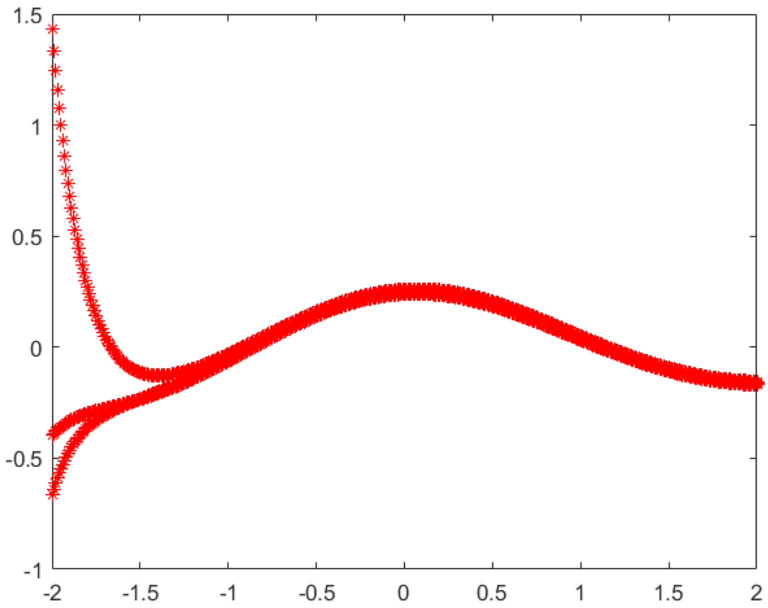
Εικόνα 3.6: $S^1 = 2, a = 0.6$, *Initial set 2*



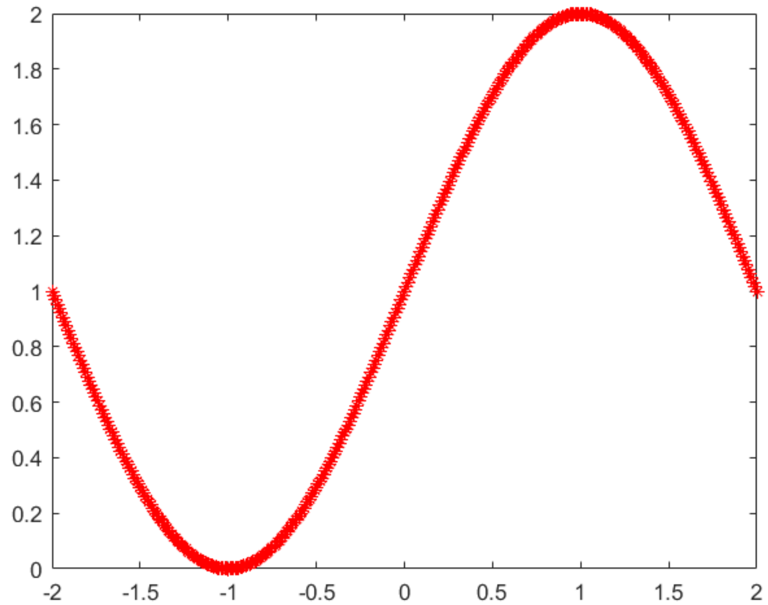
Εικόνα 3.7: $S^1 = 10$, *approximation*

```
%Output the initial set
W1_0 = W1
b1_0 = b1
W2_0 = W2
b2_0 = b2

alfa = 0.6;     %learning rate
tol = 0.1;      %tol: tolerance
```



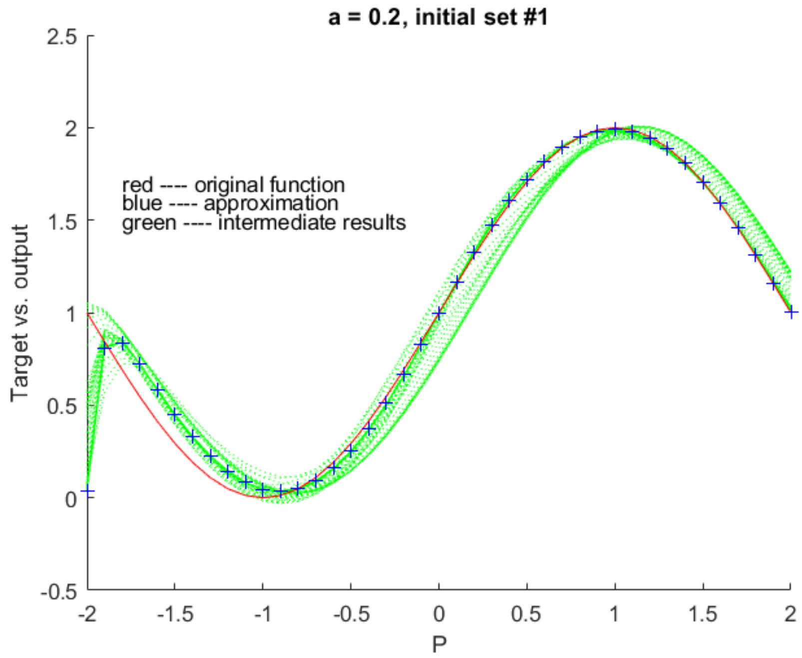
Εικόνα 3.8: $S^1 = 10$, *error*



Εικόνα 3.9: $S^1 = 10$, *realfunctionvalue*

```
mse = 1;        %mse: mean square error
iter = 0;

figure;
while (mse > tol && iter < 1000)
   mse = 0;
   i = 0;
   iter = iter + 1;
```



Εικόνα 3.10: $S^1 = 2, a = 0.2$, *Initial set 1*

```
  for P = -2 : .1 :2
     i = i + 1;
     T = 1 + sin(pi*P/2);
     a1 = logsig(W1*P + b1);
   a2 = purelin(W2*a1 + b2);
     mse = mse + (T - a2)^2;
     A(i) = a2;

   dlogsig = [(1 - a1(1))* a1(1) 0;0 (1 - a1(2))* a1(2)];
   s2 = -2 * (T - a2);
   s1 = dlogsig * W2' * s2;

   W2 = W2 - alfa * s2 * a1';
   W1 = W1 - alfa * s1 * P;
   b2 = b2 - alfa * s2;
     b1 = b1 - alfa * s1;
  end
  P = -2 : .1 : 2;
  if (mod(iter,10) == 0)
     plot(P,A,'g:')
  end
  hold on;
end

%Display in graph
P = -2 : .1 : 2;
T = 1 + sin(pi*P/2);
%figure;
plot(P,T,'r-',P,A,'b+')
title('a = 0.4, initial set #2');
```

```
text(-1.0,2.8,'red ---- original function');
text(-1.0,2.4,'blue ---- approximation');
text(-1.0,2.0,'green ---- intermediate results');
xlabel('P'), ylabel('Target vs. output');
W1
b1
W2
b2
iter
```

```
clf; clc; clear;

% Init min/max
xMax = 0.5;
xMin = -xMax;
difference = xMax-xMin;

% Init w1, b1, w2, b2

W1(:, 1) = xMin + rand(1, 10)*difference; % For the first layer (logsig)
B1(:, 1) = xMin + rand(1, 10)*difference; % The new values of the weights, biases will be added to each
     column

W2(1, :) =(xMin+rand(1,10)*difference)'; % For the second layer (purelin)
B2(1) = xMin + rand(1, 1)*(difference); % The new values of the weights, biases will be added to each row

i = 1;
learningRate = 0.01;

while (1)
  for p = -2:0.01:2

    % FEED-FORWARD PHASE

    product = (W1(:, i).* p + B1(:, i)); % Compute activation for 1st layer
    A1(:, i) = 1./(1+exp(-product));
    A2(i) = W2(i, :)*A1(:, i) + B2(1); % Compute activation for 2nd layer

    % COMPUTE ERROR THROUGH THE COST FUNCTION

    error = 1 + sin(p * (pi/2)) - A2(i);

    % BACKPROPAGATION PREPARATION

    % Compute derivative for 1st activation, compute sensitivities
    derivative =[];
    for row = 1:10
      element = (1-A1(row,i))*A1(row,i);
      derivative = [derivative element];
    end
    R = diag(derivative);

    s2 = -2 * 1 * error;
```

```
    s1 = R * W2(i, :)' *s2;

    % UPDATE WEIGHTS AND BIASES

    W1(:, i+1) = W1(:, i) - learningRate * s1 * p;
    B1(:, i+1) = B1(:, i) - learningRate * s1;

    W2(i+1, :) = W2(i, :) - learningRate * s2 * A1(:,i)';
    B2(i+1) = B2(i) - learningRate * s2;

    figure(1);
    plot(p, error,'*b');
    hold on;

    figure(2);
    plot(p, A2(i),'*g');
    hold on;

    figure(3)
    plot(p,1 + sin(p*(pi/2)),'*r');
    hold on;

    i = i + 1;

  end

  if (abs(error) < 0.1 || i>1000)
    return % End if absolute error is smaller than the predefined threshold
  end

end
```