University of Tehran
College of Engineering
School of Electrical and Computer Engineering

Communication Circuits

Computer Assignment 3

Soroush Mesforush
810198472

Masoud Tahmasbi
810198429

Tir 02

Contents

Abstract

In this project we aim to estimate the parameters of a patch microstrip antenna with the aid of its pattern, to do this we have designed and trained a neural network accordingly and obtained the desired results respectively.

1 Dataset creation

We've learned from the **Intelligent Systems** and **Neural Networks** courses that to design a proper neural network, we must train it on an appropriate and carefully chosen dataset. In our first step, we aim to create such a dataset, to do so we use **patchMicrostrip** given the following parameters.

- **Frequency**
- **Patch Length**
- **Patch Width**
- **Patch Height**

Given these parameters, we are able to determine the pattern for the patch microstrip antenna via matlab and save them as our dataset, it is important to note that we have saved our parameters in a separate **.csv** file.

1.1 Dataset generation MATLAB code

We encountered many obstacles whilst designing the code to create a proper dataset, in the end we were able to design an algorithm consisting of generating the data based on the degrees of freedom we have. Firstly we generated the preliminary values needed for each parameter, then we created arrays that change size on each run of the code to dynamically change our degrees of freedom then we go on to create the antennas and their respective patterns via nested for loops.

```
clc
clear all
N = 3;
n = 6;
cnt = 15;
%Freq
minfreq = 2e9;
maxfreq = 2.4e9;
coeffreq = (maxfreq - minfreq)./16;
%Patch Length
minplen = 0.3;
maxplen = 0.5;
coefplen = (maxplen - minplen)./16;
%Patch Width
minpwid = 0.15;
maxpwid = 0.3;
coefpwid = (maxpwid - minpwid)./16;
%Patch Height
minpheight = 0.03;
maxpheight = 0.06;
coefpheight = (maxpheight - minpheight)./16;

frequncies = linspace(minfreq + (cnt-1)*coeffreq,minfreq + (cnt)*coeffreq,N)
  ;%*10^9; %mm
patch_length = linspace(minplen + (cnt-1)*coefplen,minplen + (cnt)*coefplen,N);
   %mm
patch_width = linspace(minpwid + (cnt-1)*coefpwid,minpwid + (cnt)*coefpwid,N);
  %mm
patch_height = linspace(minpheight + (cnt-1)*coefpheight,minpheight + (cnt)*
  coefpheight,N); %mm
pat = zeros(360/n,N^4);
parameters = zeros(4,N^4);
i = 0;
for i1 = 1:N
  for i2=1:N
    for i3=1:N
      for i4=1:N
        i = i + 1;
        pm = patchMicrostrip('Length',patch_length(i1), 'Width',patch_width(i2)
  ,                    ...
        'Height',patch_height(i3));
        [pat(:,i),~,~] = pattern (pm, frequncies(i4), 0, 1:n:360);
        parameters(:,i) = [patch_length(i1),patch_width(i2),patch_height(i3),
  frequncies(i4)];
          disp(i)
      end
    end
  end
end
```


Figure 1: Pattern Generation

1.2 Saving Dataset to .csv files

To save the dataset to **.csv** files, we payed unfading attention to the fact that we must save the parameters and pattern output in separate **.csv** files.

```
% Sample data (16,000 columns)
numColumns = N^4;
numRows = 360/n;


% Create column headers
headers = cell(1, numColumns);
for i = 1:numColumns
  headers{i} = sprintf('Column%d', i);
end

% Specify the filename
filename = 'pattern15.csv';

% Open the file for writing
fileID = fopen(filename, 'w');

% Write the column headers
fprintf(fileID, '%s,', headers{1:end-1});
fprintf(fileID, '%s\n', headers{end});

% Write the data
for i = 1:numRows
  fprintf(fileID, '%f,', pat(i, 1:end-1));
  fprintf(fileID, '%f\n', pat(i, end));
end

% Close the file
fclose(fileID);

disp('Data has been written to the CSV file.');


```

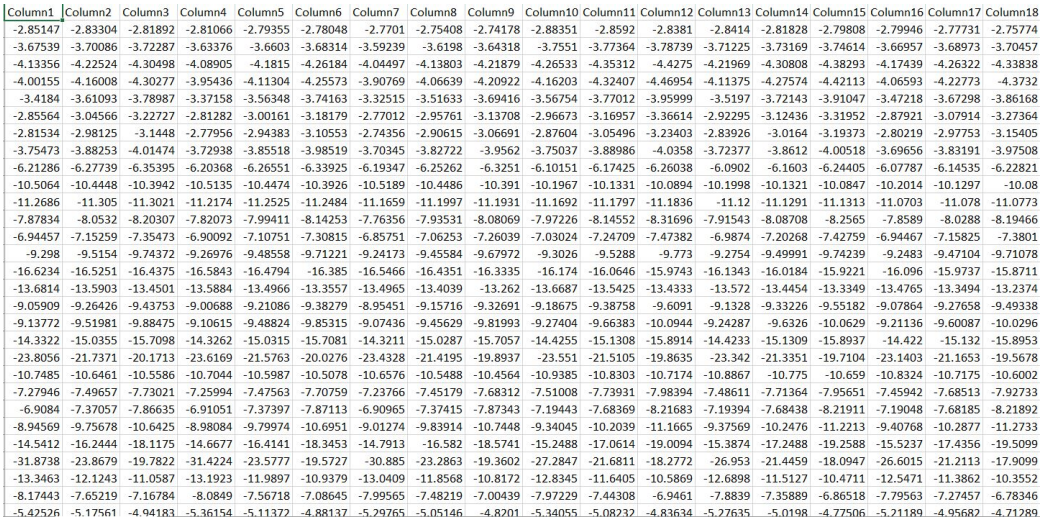
Figure 2: Code to save patterns

```
numColumns = N^4;
numRows = 4;


% Create column headers
headers = cell(1, numColumns);
for i = 1:numColumns
  headers{i} = sprintf('Column%d', i);
end

% Specify the filename
filename = 'parameters15.csv';

% Open the file for writing
fileID = fopen(filename, 'w');

% Write the column headers
fprintf(fileID, '%s,', headers{1:end-1});
fprintf(fileID, '%s\n', headers{end});

% Write the data
for i = 1:numRows
  fprintf(fileID, '%f,', parameters(i, 1:end-1));
  fprintf(fileID, '%f\n', parameters(i, end));
end

% Close the file
fclose(fileID);

disp('Data has been written to the CSV file.');


```

Figure 3: Code to save parameters

After taking these steps with much difficulty are dataset is complete and ready, in the next phase we shall design the appropriate neural network.

Before doing so we shall include a photo of the dataset to create the needed mindset for the rest of this project.



Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16	Column17	Column18
-2.85147	-2.83304	-2.81892	-2.81066	-2.79355	-2.78048	-2.7701	-2.75408	-2.74178	-2.88351	-2.8592	-2.8381	-2.8414	-2.81828	-2.79808	-2.79946	-2.77731	-2.75774
-3.67539	-3.70086	-3.72287	-3.63376	-3.6603	-3.68314	-3.59239	-3.6198	-3.64318	-3.7551	-3.77364	-3.78739	-3.71225	-3.73169	-3.74614	-3.66957	-3.68973	-3.70457
-4.13356	-4.22524	-4.30498	-4.08905	-4.1815	-4.26184	-4.04497	-4.13803	-4.21879	-4.26533	-4.35312	-4.4275	-4.21969	-4.30808	-4.38293	-4.17439	-4.26322	-4.33838
-4.00155	-4.16008	-4.30277	-3.95436	-4.11304	-4.25573	-3.90769	-4.06639	-4.20922	-4.16203	-4.32407	-4.46954	-4.11375	-4.27574	-4.42113	-4.06593	-4.22773	-4.3732
-3.4184	-3.61093	-3.78987	-3.37158	-3.56348	-3.74163	-3.32515	-3.51633	-3.69416	-3.56754	-3.77012	-3.95999	-3.5197	-3.72143	-3.91047	-3.47218	-3.67298	-3.86168
-2.85564	-3.04566	-3.22727	-2.81282	-3.00161	-3.18179	-2.77012	-2.95761	-3.13708	-2.96673	-3.16957	-3.36614	-2.92295	-3.12436	-3.31952	-2.87921	-3.07914	-3.27364
-2.81534	-2.98125	-3.1448	-2.77956	-2.94383	-3.10553	-2.74356	-2.90615	-3.06691	-2.87604	-3.05496	-3.23403	-2.83926	-3.0164	-3.19373	-2.80219	-2.97753	-3.15405
-3.75473	-3.88253	-4.01474	-3.72938	-3.85518	-3.98519	-3.70345	-3.82722	-3.9562	-3.75037	-3.88986	-4.0358	-3.72377	-3.8612	-4.00518	-3.69656	-3.83191	-3.97508
-6.21286	-6.27739	-6.35395	-6.20368	-6.26551	-6.33925	-6.19347	-6.25262	-6.3251	-6.10151	-6.17425	-6.26038	-6.0902	-6.1603	-6.24405	-6.07787	-6.14535	-6.22821
-10.5064	-10.4448	-10.3942	-10.5135	-10.4474	-10.3926	-10.5189	-10.4486	-10.391	-10.1967	-10.1331	-10.0894	-10.1998	-10.1321	-10.0847	-10.2014	-10.1297	-10.08
-11.2686	-11.305	-11.3021	-11.2174	-11.2525	-11.2484	-11.1659	-11.1997	-11.1931	-11.1692	-11.1797	-11.1836	-11.12	-11.1291	-11.1313	-11.0703	-11.078	-11.0773
-7.87834	-8.0532	-8.20307	-7.82073	-7.99411	-8.14253	-7.76356	-7.93531	-8.08069	-7.97226	-8.14552	-8.31696	-7.91543	-8.08708	-8.2565	-7.8589	-8.0288	-8.19466
-6.94457	-7.15259	-7.35473	-6.90092	-7.10751	-7.30815	-6.85751	-7.06253	-7.26039	-7.03024	-7.24709	-7.47382	-6.9874	-7.20268	-7.42759	-6.94467	-7.15825	-7.3801
-9.298	-9.5154	-9.74372	-9.26976	-9.48558	-9.71221	-9.24173	-9.45584	-9.67972	-9.3026	-9.5288	-9.773	-9.2754	-9.49991	-9.74239	-9.2483	-9.47104	-9.71078
-16.6234	-16.5251	-16.4375	-16.5843	-16.4794	-16.385	-16.5466	-16.4351	-16.3335	-16.174	-16.0646	-15.9743	-16.1343	-16.0184	-15.9221	-16.096	-15.9737	-15.8711
-13.6814	-13.5903	-13.4501	-13.5884	-13.4966	-13.3557	-13.4965	-13.4039	-13.262	-13.6687	-13.5425	-13.4333	-13.572	-13.4454	-13.3349	-13.4765	-13.3494	-13.2374
-9.05909	-9.26426	-9.43753	-9.00688	-9.21086	-9.38279	-8.95451	-9.15716	-9.32691	-9.18675	-9.38758	-9.6091	-9.1328	-9.33226	-9.55182	-9.07864	-9.27658	-9.49338
-9.13772	-9.51981	-9.88475	-9.10615	-9.48824	-9.85315	-9.07436	-9.45629	-9.81993	-9.27404	-9.66383	-10.0944	-9.24287	-9.6326	-10.0629	-9.21136	-9.60087	-10.0296
-14.3322	-15.0355	-15.7098	-14.3262	-15.0315	-15.7081	-14.3211	-15.0287	-15.7057	-14.4255	-15.1308	-15.8914	-14.4233	-15.1309	-15.8937	-14.422	-15.132	-15.8953
-23.8056	-21.7371	-20.1713	-23.6169	-21.5763	-20.0276	-23.4328	-21.4195	-19.8937	-23.551	-21.5105	-19.8635	-23.342	-21.3351	-19.7104	-23.1403	-21.1653	-19.5678
-10.7485	-10.6461	-10.5586	-10.7044	-10.5987	-10.5078	-10.6576	-10.5488	-10.4564	-10.9385	-10.8303	-10.7174	-10.8867	-10.775	-10.659	-10.8324	-10.7175	-10.6002
-7.27946	-7.49657	-7.73021	-7.25994	-7.47563	-7.70759	-7.23766	-7.45179	-7.68312	-7.51008	-7.73931	-7.98394	-7.48611	-7.71364	-7.95651	-7.45942	-7.68513	-7.92733
-6.9084	-7.37057	-7.86635	-6.91051	-7.37397	-7.87113	-6.90965	-7.37415	-7.87343	-7.19443	-7.68369	-8.21683	-7.19394	-7.68438	-8.21911	-7.19048	-7.68185	-8.21892
-8.94569	-9.75678	-10.6425	-8.98084	-9.79974	-10.6951	-9.01274	-9.83914	-10.7448	-9.34045	-10.2039	-11.1665	-9.37569	-10.2476	-11.2213	-9.40768	-10.2877	-11.2733
-14.5412	-16.2444	-18.1175	-14.6677	-16.4141	-18.3453	-14.7913	-16.582	-18.5741	-15.2488	-17.0614	-19.0094	-15.3874	-17.2488	-19.2588	-15.5237	-17.4356	-19.5099
-31.8738	-23.8679	-19.7822	-31.4224	-23.5777	-19.5727	-30.885	-23.2863	-19.3602	-27.2847	-21.6811	-18.2772	-26.953	-21.4459	-18.0947	-26.6015	-21.2113	-17.9099
-13.3463	-12.1243	-11.0587	-13.1923	-11.9897	-10.9379	-13.0409	-11.8568	-10.8172	-12.8345	-11.6405	-10.5869	-12.6898	-11.5127	-10.4711	-12.5471	-11.3862	-10.3552
-8.17443	-7.65219	-7.16784	-8.0849	-7.56718	-7.08645	-7.99565	-7.48219	-7.00439	-7.97229	-7.44308	-6.9461	-7.8839	-7.35889	-6.86518	-7.79563	-7.27457	-6.78346
-5.42526	-5.17561	-4.94183	-5.36154	-5.11372	-4.88137	-5.29765	-5.05146	-4.8201	-5.34055	-5.08232	-4.83634	-5.27635	-5.0198	-4.77506	-5.21189	-4.95682	-4.71289

Figure 4: Pattern Dataset

2 Neural Network Design

Here we design the neural network to predict the parameters given the dataset step by step as follows.

```
from keras.models import Sequential
from keras.layers import Dense,Conv2D,MaxPooling2D,Dropout,Activation,Input,
    Flatten
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras import layers
import random
import pandas as pd
import numpy as np

```

Figure 5: Needed Libraries

```
parameters = pd.read_csv('ParametersDataset.csv')
pattern = pd.read_csv('PatternDataset.csv')

```

Figure 6: Importing the needed .csv files

```
parameters.iloc[3] = parameters.iloc[3]/1e8
X = pattern.to_numpy().T
y = parameters.to_numpy().T
y[0]

```

Figure 7: Preprocessing

Here we perform the needed preprocessing, we normalize the frequency to achieve an appropriate loss in training.

```
 train_idx = np.random.randint(1296, size=1290)
test_idx = np.random.randint(1296, size=6)
X_train = X[train_idx,:]
y_train = y[train_idx,:]
X_test = X[test_idx,:]
y_test = y[test_idx,:]
X_test
X_train.shape, y_train.shape, X_test.shape, y_test.shape

```

Figure 8: Splitting train and test data

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(60,)))
model.add(layers.Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(layers.Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(layers.Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(layers.Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(4))
optimizer = Adam(lr=0.005)
model.compile(optimizer=optimizer, loss='mean_squared_error')

```

Figure 9: Neural Network design

Here we have designed the required neural network to solve the regression problem needed to estimate our wanted parameters, we have used the appropriate activation functions and layers such as dense layers to create the fully connected network and dropout layers to prevent overfitting, next we go on to fit the model.

```
model.fit(X_train, y_train, epochs=200, batch_size=64, verbose=1)

```

Figure 10: Fitting the model

A portion of the results of training is as follows.

```
Epoch 175/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0107
Epoch 176/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0121
Epoch 177/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0112
Epoch 178/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0121
Epoch 179/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0124
Epoch 180/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0110
Epoch 181/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0111
Epoch 182/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0112
Epoch 183/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0101
Epoch 184/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0097
Epoch 185/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0110
Epoch 186/200
21/21 [==============================] - 0s 7ms/step - loss: 0.0108
Epoch 187/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0105
Epoch 188/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0107
Epoch 189/200
21/21 [==============================] - 0s 7ms/step - loss: 0.0107
Epoch 190/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0098
Epoch 191/200
21/21 [==============================] - 0s 7ms/step - loss: 0.0091
Epoch 192/200
21/21 [==============================] - 0s 7ms/step - loss: 0.0115
Epoch 193/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0168
Epoch 194/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0101
Epoch 195/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0092
Epoch 196/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0099
Epoch 197/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0113
Epoch 198/200
21/21 [==============================] - 0s 9ms/step - loss: 0.0108
Epoch 199/200
21/21 [==============================] - 0s 7ms/step - loss: 0.0094
Epoch 200/200
21/21 [==============================] - 0s 8ms/step - loss: 0.0104
<keras.callbacks.History at 0x7ff7fc1dfb20>

```

Now we shall compare the real values and predictions of our neural network for some values, we shall plot them via MATLAB in due course.

```
1/1 [==============================] - 0s 116ms/step
array([[ 0.4267951 ,  0.24109367,  0.05186379, 22.80456   ],
[ 0.46956602,  0.27102467,  0.05491298, 23.542824  ],
[ 0.33851427,  0.17129165,  0.03029491, 20.821371  ],
[ 0.33394715,  0.15886563,  0.03298518, 20.515186  ],
[ 0.33708516,  0.16705135,  0.0295681 , 20.743458  ],
[ 0.33793047,  0.16802919,  0.02939084, 20.752104  ]],
dtype=float32)

```

Figure 12: Prediction

```
array([[ 0.425   ,  0.253125,  0.050625, 22.5     ],
[ 0.46875 ,  0.276562,  0.054375, 23.25    ],
[ 0.3375  ,  0.178125,  0.035625, 20.75    ],
[ 0.30625 ,  0.159375,  0.031875, 20.25    ],
[ 0.33125 ,  0.16875 ,  0.03375 , 20.75    ],
[ 0.325   ,  0.178125,  0.034687, 20.75    ]])
dtype=float32)

```

Figure 13: Real values

As we can see, our neural network performs admirably, now we shall include the MATLAB plots.

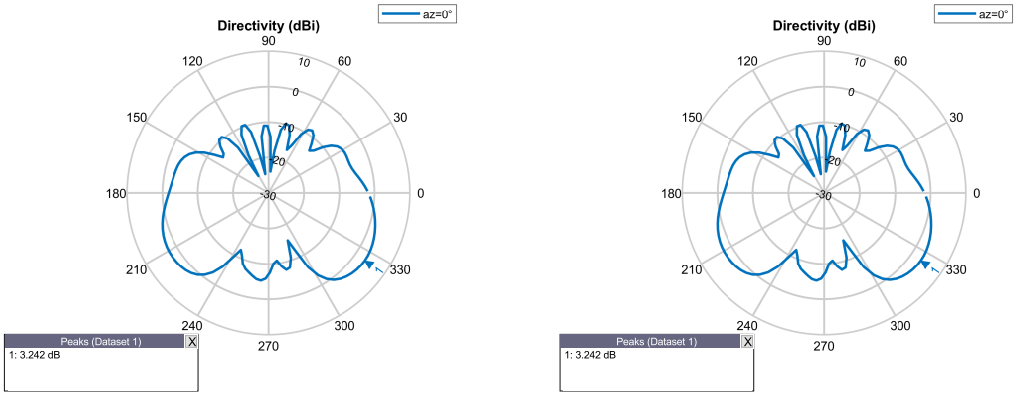


Figure 14: Comparison of Predicted and real pattern

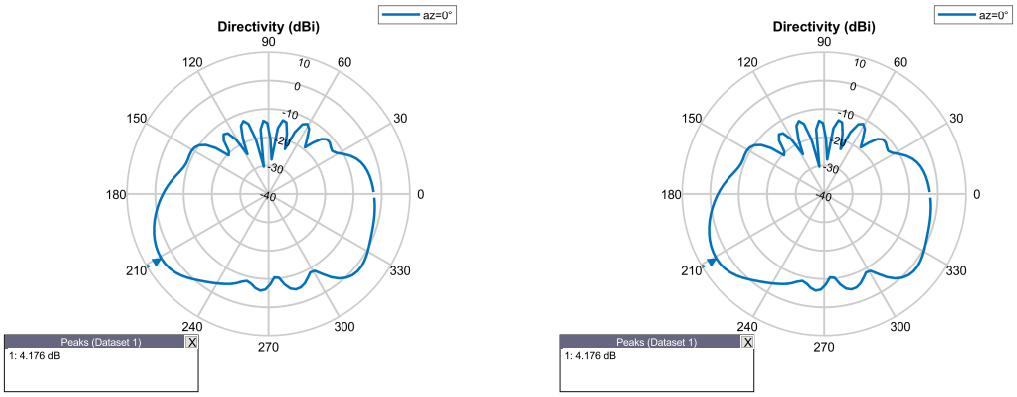


Figure 15: Comparison of Predicted and real pattern

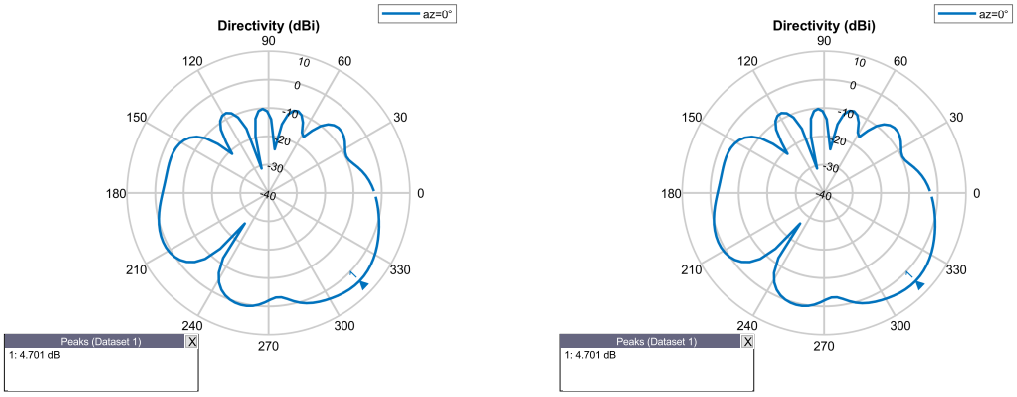


Figure 16: Comparison of Predicted and real pattern

As we can see our neural network has worked with impeccable accuracy and the predicted pattern is very accurate and similar to the real one. So we have been able to design a neural network which predicts a microstrip patch antenna's parameters given a dataset of patterns.

References

[1] Jalil Rashed-Mohassel, *Antenna theory lecture notes, Spring 02*

[2] Ahmad Kalhor, *Neural Networks Lecture Notes, Fall 01*

[3] Reshad Hosseini, *Intelligent Systems Lecture Notes, Fall 01*