Politecnico di Torino

Guiding Electromagnetic Systems

Design of a stepped impedance LPF

Final Project Report

Master degree in Electronics Engineering

Referents: Prof. Paola Pirinoli

Casalino	Andrea	242094
Gualco	Andrea	240039
Jamal	Muhammad Usman	231741
Inglese	Pietro	243243

July 28, 2017

Contents

CHAPTER 1

Specifications

1.1 LPF specifications

The goal of the project is to design and realize a low-pass filter. Table 1.1 shows all the filter specifications. Since the required behavior is 0.5 dB ripple, it is necessary to use Chebyshev filter.

Filter type	low-pass
Response type	Equal ripple (0.5 dB)
$\boldsymbol{f_0}$ [GHz]	2.4
$\boldsymbol{R_0}$ [Ω]	50
Insertion loss [dB] @ $\boldsymbol{f}$ = [4.8 GHz]	> -30

Table 1.1: LPF specs

On the contrary, table 1.1 shows all the microstrip and physical specifications which will be used to realize the LPF.

$\boldsymbol{\epsilon_r}$	4
h [mm]	2
Electric length transmission lines [°]	45
Loss tangent [rad]	0

Table 1.2: Microstrip specs

1.2 Requests

- Design the prototype filter and apply the required frequency and impedance transformations considering “T” and “Pi” network;
- Design the filter using *stepped impedance* technique. Consider to insert also two segments of line at input and output of the microstrip for the connection of the device to the connectors;
- Simulate the frequency response of the filter;
- Print the “gerber” file of both the top and the bottom layers of the microstrip circuit;
- Measure the frequency response of the filter (S_{11} and S_{21}).

CHAPTER 2

Design and Simulation

2.1 Matlab design

To design the LPF it has been written a Matlab script (see chapter 5 for the code) using the formulæ seen during the lectures.
The resulting number of poles of the LPF is $N = 7$.

2.1.1 Lumped elements

Therefore the LPF Prototype Filter coefficients are:

$g_1 = 1.7373$	$g_2 = 1.2582$	$g_3 = 2.6383$	$g_4 = 1.3443$
$g_5 = 2.6383$	$g_6 = 1.2582$	$g_7 = 1.7373$	$g_8 = 1$

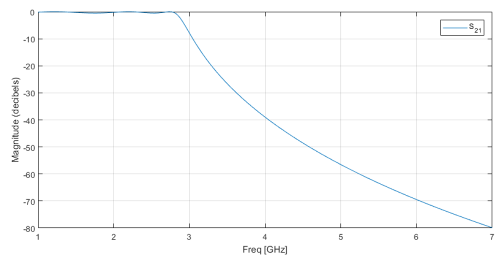
For the "T" network the values of the inductors and capacitors are:

$L_1 = 4.9199$ nH	$C_2 = 1.4253$ pF	$L_3 = 7.4715$ nH	$C_4 = 1.5228$ pF
$L_5 = 7.4715$ nH	$C_6 = 1.4253$ pF	$L_7 = 4.9199$ nH	

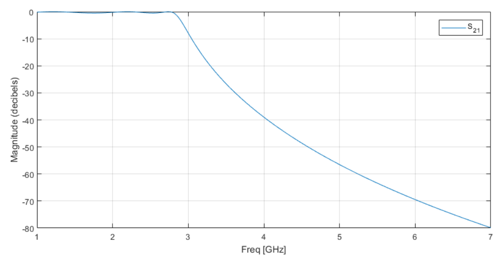
For the "Pi" network the values of the inductors and capacitors are:

$C_1 = 1.9680$ pF	$L_2 = 3.5631$ nH	$C_3 = 2.9886$ pF	$L_4 = 3.8070$ nH
$C_5 = 2.9886$ pF	$L_6 = 3.5631$ nH	$C_7 = 1.9680$ pF	

Below are reported the 2 lumped elements frequency responses. As expected, those are identical.



(a) T-network frequency response



(b) Pi-network frequency response

2.1.2 Comparison between lumped elements and stepped impedance

After designing the LPF using lumped elements, the same filter has been designed using microstrips, in particular *stepped impedance* technology.
It must be taken into account that the lumped elements filter will be more precise than the one

designed using *stepped impedance*. Below are shown the comparisons between lumped elements and microstrip implementations on both T-network and Pi-network. It has been decided to use the "T" one because the behaviour is closer to the lumped elements circuit, as expected.

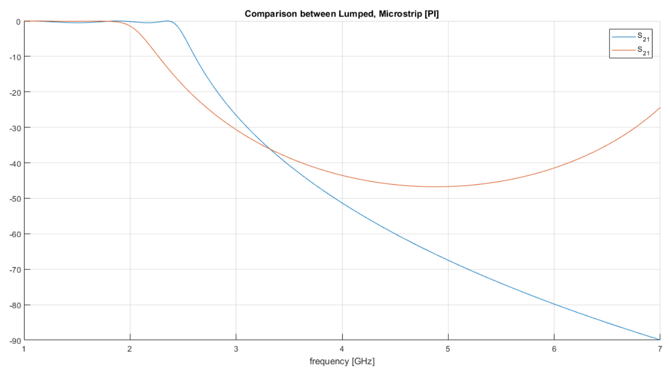


Figure 2.2: PI-network lumped (blue) and stepped impedance (orange)

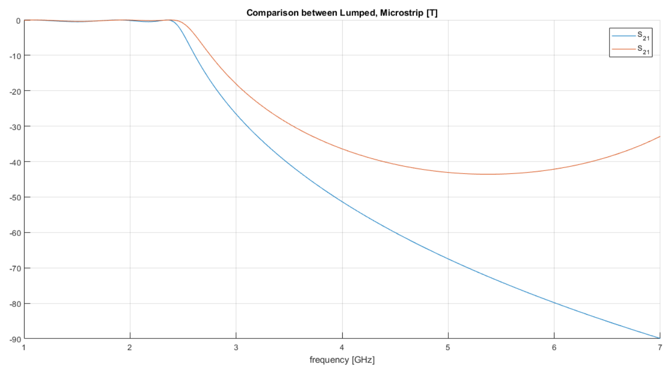


Figure 2.3: T-network lumped (blue) and stepped impedance (orange)

2.1.3 Stepped impedance

In order to create a functioning project on AWR, the length and width of the circuit have to be modeled. As written in the previous section the best choice is the T-network, therefore the circuit has been designed starting from it. The final dimensions and impedances of the filter are:

Line	**Length [mm]**	**Width [mm]**	**Impedance [Ω]**
line 1	8.0	0.7	110.60
line 2	7.4	8.3	31.211
line 3	8.2	0.2	167.96
line 4	7.4	9.0	29.212
line 5	8.2	0.2	167.96
line 6	7.4	8.3	31.211
line 7	8.0	0.7	110.60

Table 2.1: Stepped impedance LPF

2.1.4 Comments

Theoretically, using a LPF with $N = 5$ to meet the specifications.
In practice this is not possible: it is better to use a filter with a higher number of poles in order to be sure that the final product could meet the specifications.

2.2 AWR design

The next step to realize the layout of the filter has been done using AWR.
The filter has been realized using the data in Table 2.1 and the microstrip was designed using the data in Table 1.2.
The final obtained layout is shown in Figure 2.4.

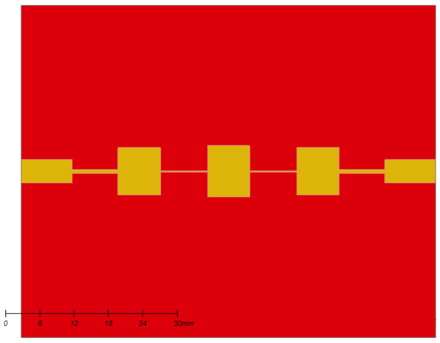


Figure 2.4: Layout of the filter

The correctness was also verified through the graphs (Figure 2.5) provided by AWR.

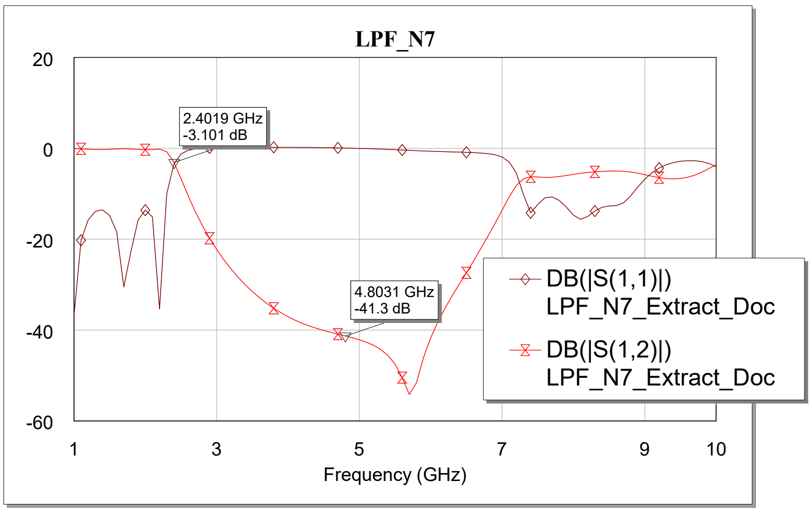


Figure 2.5: Behavior of the filter designed with AWR

CHAPTER 3

Realization

Once designed and verified the correctness of the filter behavior, the design has been sent to the producer.
The final specifications of the filter sent to realization are:

Filter type	LPF Chebyshev
Response type	Equal ripple (0.5 dB)
$\boldsymbol{f_0}$ [GHz]	2.4019
Insertion loss [**dB**] @ $\boldsymbol{f}$ = [4.8031 GHz]	-41.3

Table 3.1: LPF final specifications

Regarding the geometry of the filter, the values are the ones from the table 2.1.

CHAPTER 4

Measurement on real device

Once the filter has been realized, it was necessary to take the measurements in order to test and verify the design phase correctness.

4.1 Filter verification

The filter parameters has been measured using the instrumentation of the LED5 laboratory, which is also able to show scattering parameters. In the Table 4.1 are shown the measured values.

Frequency [GHz]	**Scattering**	**Attenuation [dB]**	**Scattering**	**Attenuation [dB]**
2.07	S_{11}	3	S_{21}	3
2.4	S_{11}	0.4	S_{21}	10.426
4.8	S_{11}	0.1	S_{21}	50.121

Table 4.1: Measurement on the real filter

4.2 Comments

As it can be seen from table 4.1, the final designed filter partially meets the specifications of table 1.1.
The bigger issue is the different cut-off frequency: this means that the filter will start to attenuate at lower frequencies.
The attenuation at 2.8 GHz is also bigger than expected.
Everything looks shifted on the left with respect to the performed design (hence at lower frequency).
This is probably related to the fact that the ϵ_r used to perform the calculations is an estimation of the one used in the manufacturing process (the used material has a different relative dielectric constant from the one used in the design phase).
It has been tried to find an ϵ_r capable to justify the final results by inverting the formulæ used in the design script.
The approximate value found is:

$$\epsilon_r = 7.38$$

In chapter 5, it is reported the matlab script used to calculate the approximated value of ϵ_r.

CHAPTER 5

Attachments

Matlab script to design the filter in lumped elements and stepped impedance technology

```
clear all; clc; close all;
%% ---------------------------------------------------------------------------------------------
%
%   LPF STEPPED IMPEDANCE
%
%% data (insert data here) ---------------------------------------------------------------------
% filter
f0 = 2.41e9;                      % f0 central frequency
type = ".5db";                    % type of the filter
                                  % "flat" = Butterworth
                                  % "3dB"  = Chebyshev 3dB ripple
                                  % ".5dB" = Chebyshev 0.5dB ripple
frange = (1e9 : 1e7 : 7e9);       % frequncy range
IL = 30;                          % insertion loss
f = 4.8e9;                        % frequency of insertion loss
r0 = 50;
Npresent = "yes";                 % if N is given type "yes" otherwise "no"
N = 7;                            % if no comment this line microstrip
vph = physconst('LightSpeed');    % default
epsr = 4;    h = 2e-3;            %[mm]
measureh = "mm";                  % if d is in inches type "in" otherwise "mm"
fixed = "length";                 % what value is fixed
len = 1/8;                        % write the lenght between quotes

%% Find the order filter -----------------------------------------------------------------------
Omega = abs(f / f0);
if Npresent == "no"
  switch type
    case "flat"
      logOmega = log10(10 ^ (IL / 10) - 1) / log10(Omega);  % changing base of the logarithm
      N = 0.5 * logOmega;                                   % find N
      N = ceil(N);                                          % round N to the upper integer
    case "3db"
      epsilon = sqrt(10 ^ ((3) / 10) - 1);
      N = (acosh(sqrt((IL ^ 2 - 1) / (epsilon ^ 2)))) / (acosh(Omega));
      N = ceil(N);
    case ".5db"
      epsilon = sqrt(10 ^ ((0.5) / 10) - 1);
      N = (acosh(sqrt((IL ^ 2 - 1) / (epsilon ^ 2)))) / (acosh(Omega));
      N = ceil(N);
  end
end
%% Find the prototype coefficients -------------------------------------------------------------
switch type
  %% Butterworth filter
  case "flat"
    switch N
      case 1
        g = [2.0000 1.0000];
      case 2
```

```
          g = [1.4142 1.4142 1.0000];
        case 3
          g = [1.0000 2.0000 1.0000 1.0000];
        case 4
          g = [0.7654 1.8478 1.8478 0.7654 1.0000];
        case 5
          g = [0.6180 1.6180 2.0000 1.6180 0.6180 1.0000];
        case 6
          g = [0.5176 1.4142 1.9318 1.9318 1.4142 0.5176 1.0000];
        case 7
          g = [0.4450 1.2470 1.8019 2.0000 1.8019 1.2470 0.4450 1.0000];
        case 8
          g = [0.3902 1.1111 1.6629 1.9615 1.9615 1.6629 1.1111 0.3902 1.0000];
        case 9
          g = [0.3473 1.0000 1.5321 1.8794 2.0000 1.8794 1.5321 1.0000 0.3473 1.0000];
        case 10
          g = [0.3129 0.9080 1.4142 1.7820 1.9754 1.9754 1.7820 1.4142 0.9080 0.3129 1.0000];
      end
    %% Chebyshev 3dB ripple
    case "3db"
      switch N
        case 1
          g = [1.9953 1.0000];
        case 2
          g = [3.1013 0.5339 5.8095];
        case 3
          g = [3.3487 0.7117 3.3487 1.0000];
        case 4
          g = [3.4389 0.7483 4.3471 0.5920 5.8095];
        case 5
          g = [3.4817 0.7618 4.5381 0.7618 3.4817 1.0000];
        case 6
          g = [3.5045 0.7685 4.6061 0.7929 4.4641 0.6033 5.8095];
        case 7
          g = [3.5181 0.7723 4.6386 0.8039 4.6386 0.7723 3.5181 1.0000];
        case 8
          g = [3.5277 0.7745 4.6575 0.8089 4.6990 0.8018 4.4990 0.6073 5.8095];
        case 9
          g = [3.5340 0.7760 4.6692 0.8118 4.7272 0.8118 4.6692 0.7760 3.5340 1.0000];
        case 10
          g = [3.5384 0.7771 4.6768 0.8136 4.7425 0.8164 4.7260 0.8051 4.5142 0.6091 5.8095];
      end
    %% Chebyshev .5dB ripple
    case ".5db"
      switch N
        case 1
          g = [0.6986 1.0000];
        case 2
          g = [1.4029 0.7071 1.9841];
        case 3
          g = [1.5963 1.0967 1.5963 1.0000];
        case 4
          g = [1.6703 1.1926 2.3661 0.8419 1.9841];
        case 5
          g = [1.7058 1.2296 2.5408 1.2296 1.7058 1.0000];
        case 6
          g = [1.7254 1.2479 2.6064 1.3137 2.4758 0.8696 1.9841];
        case 7
          g = [1.7373 1.2582 2.6383 1.3443 2.6383 1.2582 1.7373 1.0000];
        case 8
          g = [1.7451 1.2647 2.6564 1.3590 2.6964 1.3389 2.5093 0.8796 1.9841];
        case 9
          g = [1.7504 1.2690 2.6678 1.3673 2.7939 1.3673 2.6678 1.2690 1.7504 1.0000];
        case 10
          g = [1.7543 1.2721 2.6754 1.3725 2.7392 1.3806 2.7231 1.3485 2.5239 0.8842 1.9841];
      end
      %%
end % end switch type
%% Value of L and C (lumped elements) ---------------------------------------------------------
% T network
for i = 1:N
  if mod(i, 2) == 1 % i is odd (dispari)
    Lt(i) = (g(i) * r0) / (2 * pi * f0);
  else % i is even (pari)
```

```
    Ct(i) = g(i) / (r0 * 2 * pi * f0);
  end
end
% delete zero elements in the vector
Lt(Lt == 0) = [];
Ct(Ct == 0) = [];
% graph frequency respose of T network
LPF_T = rfckt.lclowpasstee('L',Lt,'C',Ct);
analysis_LPF_T = analyze(LPF_T, frange);

% Pi network
for i = 1:N
  if mod(i, 2) == 1 % i is odd (dispari)
    Cpi(i) = g(i) / (r0 * 2 * pi * f0);
  else % i is even (pari)
    Lpi(i) = (g(i) * r0) / (2 * pi * f0);
  end
end
% delete zero elements in the vector
Lpi(Lpi == 0) = [];
Cpi(Cpi == 0) = [];
% analyse the frequency response of Pi network
LPF_PI = rfckt.lclowpasspi('L',Lpi,'C',Cpi);
analysis_LPF_PI  = analyze(LPF_PI, frange);

%% Stepped impedance T ----------------------------------------------------------------
theta = 2 * pi * len; %length microstrip is fixed
for i = 1:N
  if mod(i, 2) == 1 % i is odd
    Zinf(i) = (g(i) * r0) / (theta); %inductor
    A = Zinf(i)*sqrt((epsr+1)/2)/60+((epsr-1)/(epsr+1))*(0.23+0.11/epsr);
    W_over_h(i) = 8*exp(A)/(exp(2*A)-2);
  else % i is even (pari)
    Zinf(i) = (r0 * theta) / (g(i)); %capacitor
    B = 377*pi/(2*Zinf(i)*sqrt(epsr)) ;
    W_over_h(i) = (2/pi)*(B-1-log(2*B-1)+((epsr-1)/(2*epsr))*(log(B-1)+0.39-(0.61/epsr)));
  end
   Width(i)  = W_over_h(i)* h ;
   epseff(i) = (epsr+1)/2+(epsr-1)/(2*sqrt(1+12/W_over_h(i)));
   lenght(i) = vph * len * 1 / (f0 * sqrt(epseff(i)));
end
Width,  lenght

% display frequency response of the stepped impedance filter
for i=1:N
  micro(i)=rfckt.microstrip('height',h, 'LineLength',lenght(i), 'Width',Width(i), 'EpsilonR'
      ,epseff(i));
end
switch N
  case 1
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1)});
  case 2
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2)});
  case 3
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3)});
  case 4
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)});
  case 5
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5)});
  case 6
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5), micro(6)});
  case 7
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5), micro(6), micro(7)});
  case 8
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5), micro(6), micro(7), micro(8)});
  case 9
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5), micro(6), micro(7), micro(8), micro(9)});
  case 10
    Cascade_microstrip_T = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4),
        micro(5), micro(6), micro(7), micro(8), micro(9), micro(10)});
end
analisys_Microstrip_T = analyze(Cascade_microstrip_T, frange);
%% plots
figure, hold on
plot(LPF_T,'S21')
plot(Cascade_microstrip_T,'S21')
title('Comparison between Lumped, Microstrip [T]'),  xlabel('frequency [GHz]')

%% Stepped impedance PI ————————————————————————————————————————
if 1==1
  theta = 2 * pi * len; %length microstrip is fixed
  for i = 1:N
    if mod(i, 2) == 1 % i is odd
      Zinf(i) = (r0 * theta) / (g(i)); %capacitor
      A = Zinf(i)*sqrt((epsr+1)/2)/60+((epsr-1)/(epsr+1))*(0.23+0.11/epsr);
      W_over_h(i) = 8*exp(A)/(exp(2*A)-2);
    else % i is even (pari)
      Zinf(i) = (g(i) * r0) / (theta); %inductor
      B = 377*pi/(2*Zinf(i)*sqrt(epsr)) ;
      W_over_h(i) = (2/pi)*(B-1-log(2*B-1)+((epsr-1)/(2*epsr))*(log(B-1)+0.39-(0.61/epsr)));
    end
    Width(i)  = W_over_h(i)* h ;
    epseff(i) = (epsr+1)/2+(epsr-1)/(2*sqrt(1+12/W_over_h(i)));
  end
  W_over_h,  Width, epseff
  % necessary to use txline here to have acorrect result
  % here i did approx the epsilon_eff of the W/h calc with epsilon_r
  for i = 1:N
    l(i) = vph * len * 1 / (f0 * sqrt(epseff(i)));
  end
  % display frequency response of the stepped impedance filter
  for i=1:N
    micro(i) = rfckt.microstrip('height', h, 'LineLength',lenght(i), 'Width',Width(i), '
        EpsilonR', epseff(i));
  end
  switch N
    case 1
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1)});
    case 2
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2)});
    case 3
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3)});
    case 4
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          });
    case 5
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5)});
    case 6
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5), micro(6)});
    case 7
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5), micro(6), micro(7)});
    case 8
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5), micro(6), micro(7), micro(8)});
    case 9
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5), micro(6), micro(7), micro(8), micro(9)});
    case 10
      Cascade_microstrip_PI  = rfckt.cascade('Ckts', {micro(1), micro(2), micro(3), micro(4)
          , micro(5), micro(6), micro(7), micro(8), micro(9), micro(10)});
  end
  analisys_Microstrip_PI = analyze(Cascade_microstrip_PI, frange);
  %% plots
  figure, hold on
  plot(LPF_PI,'S21')
  plot(Cascade_microstrip_PI,'S21')
  title('Comparison between Lumped, Microstrip [PI]'),  xlabel('frequency [GHz]')
end
```

Matlab script to find an approximate value of the real relative dielectric constant ϵ_r

```
clc, close all, clear all;
%% script to find the value of the relative dielectric constant ---------------------------
LightSpeed = physconst('LightSpeed');
f_c    = 2.07e9; % measured cut-off frequency
h      = 2e-3;
L(1)   = 8.00;          W(1)  = 0.70  ;
L(2)   = 7.40;          W(2)  = 8.30  ;
L(3)   = 8.20;          W(3)  = 0.20  ;
L(4)   = 7.40;          W(4)  = 9.00  ;
L      = L.*1e-3;       W     = W.*1e-3;
alpha = sqrt(1+12.*h./W);
epsilon_eff_V = (LightSpeed./(8*f_c*L)).^2;
epsilon_r_V    = (2.*alpha.*epsilon_eff_V - alpha+1)./(alpha+1);

epsilon_eff = mean(epsilon_eff_V)
epsilon_r    = mean(epsilon_r_V)
```