Daniele Biasini
Alexandru Obada

Progetto di un array lineare di patch rettangolari per base station WiMax

Progetto di fine corso

Prof. Gaetano Marrocco
Università degli Studi di Roma Tor Vergata
Facoltà di Ingegneria delle Tecnologie di Internet
Tecnologie Elettromagnetiche per Sistemi Wireless
Gennaio 2013

Indice

Elenco delle figure

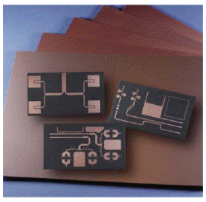
Elenco delle tabelle

Sommario

In questo lavoro è stato progettato un array di antenne a patch secondo alcune specifiche assegnate dal docente. Utilizzando queste specifiche si ottiene un'antenna compatibile con la tecnologia *WiMAX*.
Di seguito sono elencate le specifiche di progetto:

- $f_0 = 5.8GHz$
- $B > 2\%$
- $G_{max} = 16dB$
- $R = -20dB$
- $BW_{-3dB} \approx 10°$

Figura 1: Duroid 5880.



1 Modellazione del singolo patch

1.1 Dimensionamento del patch con il modello di *Carver*

Il primo passo verso la progettazione dell'antenna è il dimensionamento del singolo patch, per svolgere questi calcoli, ed anche i successivi, sono stati sviluppati alcuni script in *Python*, ciò ha portato alcuni vantaggi come la possibilità di automatizzare le operazioni e di avere valori teorici molto precisi. Il primo passo da compiere è la scelta del dielettrico, è stato scelto un *Duroid 5880* (Fig. 1), con costante dielettrica $\epsilon_r = 2.2$ e angolo di perdita $tan\theta = 0.0009$.

Il passo successivo è il calcolo dell'altezza massima del dielettrico che garantisce la trascurabilità dei modi d'onda superficiali presenti nel patch a causa della discontinuità dielettrica. L'altezza scelta per il substrato dielettrico deve essere inferiore od uguale a questo valore.

$$h_{max} = \frac{0.3c}{2\pi f_0 \sqrt{\epsilon_r}} = 1.66503477015mm \tag{1}$$

Si calcola quindi la larghezza del patch:

$$W = \frac{c}{2f_0}\sqrt{\frac{2}{\epsilon_r + 1}} = 20.4457607338mm \tag{2}$$

```
# Velocita' della luce [m/s]
c = 3*10**8
# Imedenza del cavo coassiale [Ohm]
r = 50
# Costante dielettrica del Duroid 5880
epsilon_r = 2.2
# Frequenza di risonanza [Hz]
f = 5.8*10**9
# Altezza massima del substrato [mm]
h_max = (0.3*c)/(2*cmath.pi*f*cmath.sqrt(epsilon_r)).real
# Larghezza del patch [mm]
w = (c/(2*f)*cmath.sqrt(2/(epsilon_r+1))).real
```

A questo punto è possibile utilizzare h_{max}(1) e W(2) per ottenere le altre dimensioni del patch.
Lo script sviluppato contiene i parametri del *Duroid 5880*, e, inserito un valore per lo spessore h del substrato inferiore a quello calcolato (1), calcola automaticamente la lunghezza del patch e del punto di alimentazione.

```
h = get_h(h_max)
```

```
def get_h(h_m):
    while(True):
        h = float(input("introdurre lo spessore del substrato (minore di" + str(h_m*1000)+" mm) \n"))
        if h > 0 and h < h_m*1000:
            return h/1000
```

La lunghezza del patch può essere ottenuta mediante la formula successiva:

$$L = \frac{c}{2f_0\sqrt{\epsilon_r}} - 2\Delta L = 16.8984183063mm \tag{3}$$

dove:

- Estensione del campo di Fringe : $\Delta L = 0.412h\frac{(\epsilon_{eff}+0.3)(\frac{W}{h}+0.264)}{(\epsilon_{eff}-0.258)(\frac{W}{h}+0.8)}$
- Costante dielettrica efficace: $\epsilon_{eff} = \frac{\epsilon_r+1}{2} + \frac{\epsilon_r-1}{2}(1+\frac{h}{W})^{-1}$
- Lunghezza d'onda: $\lambda = \frac{\lambda_0}{\sqrt{\epsilon_{eff}}}$

```
lunghezza = getLength(w,h,c,f,epsilon_r)
```

```
def getLength(w, h , c , f , epsilon_r):
    # Costante dielettrica efficace
    epsilon_eff = ((epsilon_r+1) + (epsilon_r-1)*(1+12*h/w)**(-0.5))/2
    delta_l = h * 0.412 * ( epsilon_eff + 0.3 )*( w/h + 0.264 )/((epsilon_eff - 0.258 )*( w/h + 0.8 ))
    l = c/( 2 * f * cmath.sqrt(epsilon_eff) ) - 2 * delta_l
    return l.real
```

Infine si calcola il punto di alimentazione:

$$l_a = \frac{1}{\beta}arccos\left(\sqrt{\frac{R_{in}}{R}}\right) = 5.32528400943mm \tag{4}$$

dove:

- Costante di fase: $\beta = \frac{2\pi\sqrt{\epsilon_r}}{\lambda_0}$
- Impedenza d'ingresso desiderata: $R_{in} = 50\Omega$
- Resistenza di radiazione: $R_r = \frac{1}{2G_s} = \frac{60\lambda_0}{W}\left[1 - \frac{1}{24}\left(\frac{2\pi h}{\lambda_0}\right)^2\right]^{-1}$

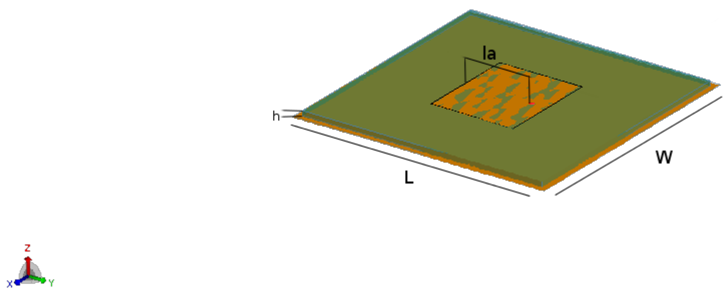
```
l_alimentazione = getFeedPoint(f , h , w , r , c , epsilon_r )
```

```
def getFeedPoint(f , h , w , r , c , epsilon_r):
    lambda_0 = c/f
    r_r = 60*lambda_0/w * (1 - 1/24*(2*cmath.pi*h/lambda_0)**2)**(-1)
    l = lambda_0 / (2*cmath.pi * cmath.sqrt(epsilon_r))*cmath.acos(cmath.sqrt(r/r_r))
    return l.real
```

L'output dello script mostrato in frammenti nel paragrafo è un file di testo con tutte le informazioni utili per poter realizzare il singolo patch con *FEKO*.

```
h = 1.0 mm
la = 5.32528400943 mm
W = 20.4457607338 mm
h_max = 1.66503477015 mm
L = 16.8984183063 mm
```

Figura 2: Patch realizzato in FEKO con le relative dimensioni.



1.2 Creazione del modello del patch con FEKO

Modello teorico

Una volta ottenute le dimensioni del patch è possibile realizzarne un modello tridimensionale attraverso l'uso del CAD elettrmagnetico *FEKO*, nella figura 2 è mostrato il patch realizzato.

Con l'utilizzo di *POSTFEKO* è possibile rielaborare i dati precedenti per ottenere informazioni riguardo al patch, si è scelto di rappresentare il coefficiente di riflessione e l'impedenza di ingresso.
Nella figura 3 è mostrata l'attenuazione del coefficiente di riflessione, come evidenziato nella figura, si può notare che c'è un minimo in $5.81167GHz$ a $-8.83847dB$, inoltre alla frequenza di risonanza $f_0 = 5.8GHz$ l'attenuazione è di $-8.82411dB$.
I valori conseguiti non rispettano le specifiche di progetto, inoltre non è neanche possibile calcolare la banda percentuale a $-15dB$.

La figura 4 mostra i valori dell'impedenza d'ingresso alla frequenza di risonanza: la parte reale dell'impedenza è di $\Re[Z] = 51.1\Omega$, la parte immaginaria è di $\Im[Z] = -38.3\Omega$. La parte reale andrebbe bene ma la parte immaginaria ad un valore così alto porterebbe a ingenti perdite di potenza.

Ottimizzazione del modello teorico

I risultati ottenuti con i modelli teorici mostrano come non solo non siano soddisfatte le specifiche di progetto ma anche come, volendo utilizzare tale modello, si ottengano bassissime prestazioni.
Per ottimizzare il modello teorico è necessario variare i valori dati in input a *POSTFEKO*: le dimensioni del patch. Riuscendo quindi a cambiare opportunamente W, L, la e h è possibile adempiere alle specifiche di progetto, le quali riferendosi al momento al singolo patch, si traducono nell'ottenere un'*attenuazione del coefficiente di riflessione inferiore a 15 dB*, portare la *parte reale dell'impedenza di ingresso ad un valore prossimo allo zero* e nell'avere *una banda percentuale superiore al 2%*.
La realizzazione di ognuna delle specifiche non è indipendente l'una dall'altra, spesso, infatti, il soddisfacimento di una specifica porta al peggioramento di un'altra. Quindi anche la variazione di alcune dimensioni del patch porta a variazioni, anche molto

Figura 3: Coefficiente di riflessione relativo alla configurazione teorica del patch.

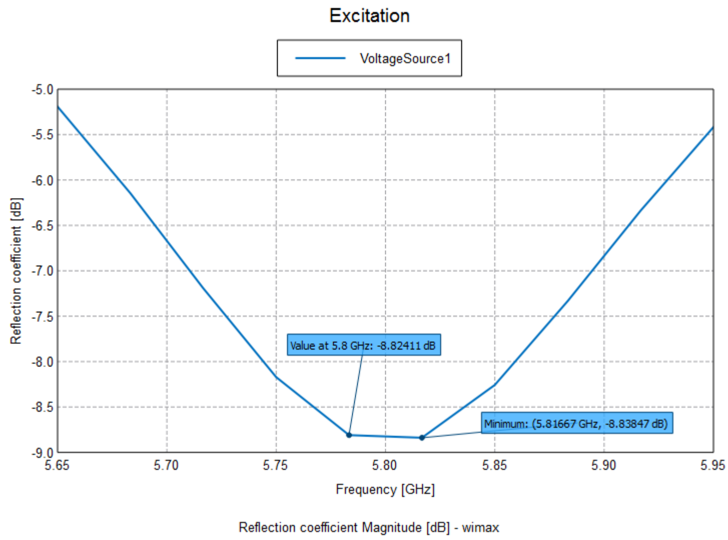


Figura 4: Parte reale e parte immaginaria dell'impedenza relative alla configurazione teorica del patch.

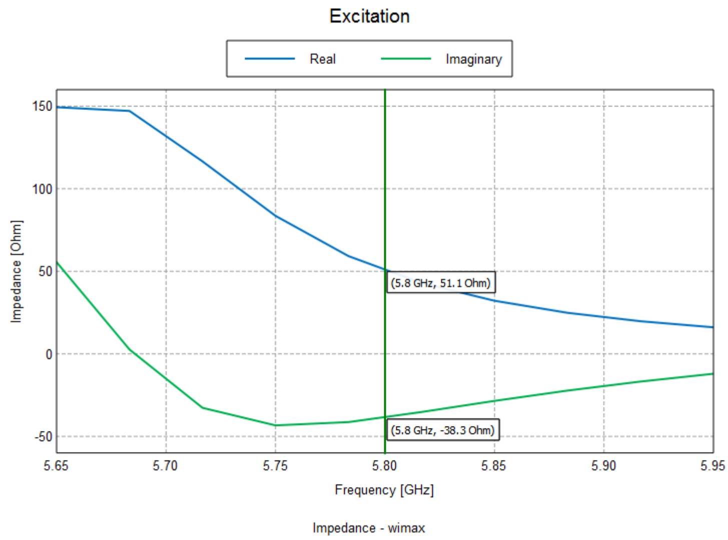
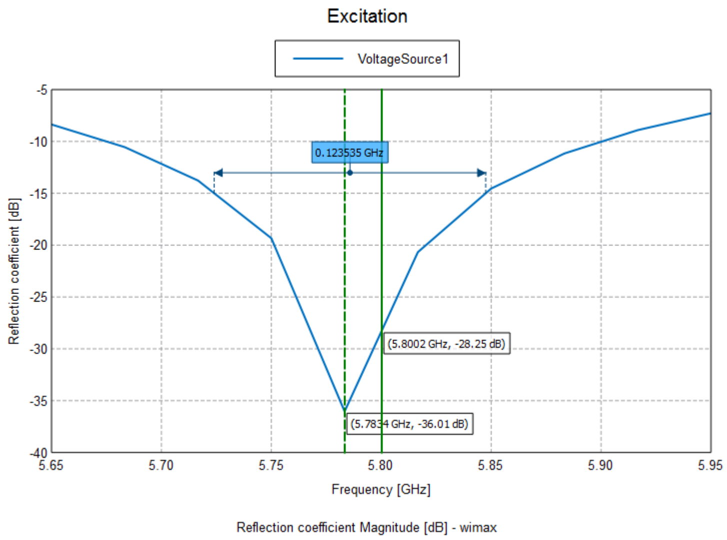


Figura 5: Coefficiente di riflessione adattato alle specifiche di progetto.



diverse, dei tre obiettivi da raggiungere.
Dopo numerose prove, sono stati scelte le seguenti dimensioni (tabella 1) che porteranno ai risultati mostrati nelle figure 5, 6, 7.

Tabella 1: Confronto tra i valori teorici e quelli ottimizzati.

Valori	$h(m)$	$L(m)$	$l_a(m)$	$W(m)$
Teorici	1	16.8984183063	5.32528400943	20.4457607338
Ottimizzati	1.6	16.5	4.3	20.5

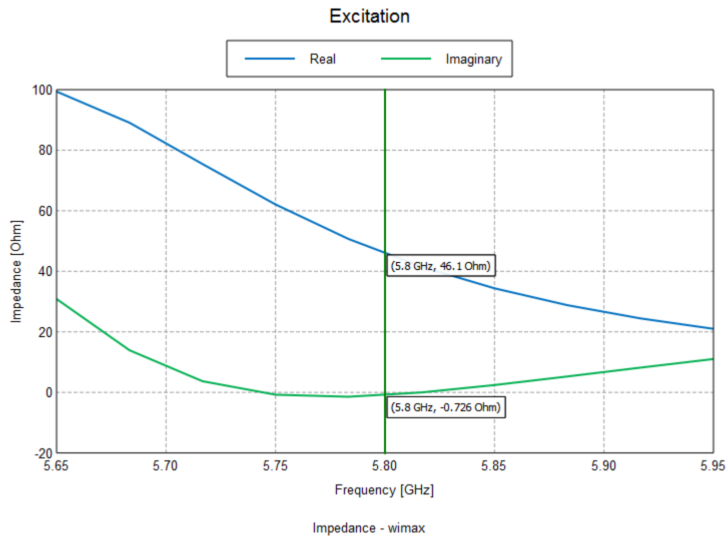
La figura 5 mostra i valori dell'attenuazione del coefficiente di riflessione, si ha un minimo alla frequenza di $5.78GHz$ pari a $-36.01dB$, alla frequenza di risonanza, invece, si ha un valora inferiore, $-28.2dB$, che comunque soddisfa le specifiche di progetto.
Altro dato molto importante è il soddisfacimento del requisito sulla banda percentuale, nella figura è evidenziata la banda a $-15dB$, $0.123535GHz$. La banda percentuale si ottiene dalla formula:

$$B_{\%} = \frac{\Delta f}{f_0} = 2.13\% \tag{5}$$

Il valore supera di 0.13% quello richiesto.

Nella figura 6 sono rappresentati i valori della parte reale e della parte immaginaria dell'impedenza di ingresso, un confronto con i valori precedenti mostra come la parte reale sia di poco inferiore alla precedente 46.1Ω ma la parte immaginaria sia prossima allo zero -0.726Ω; molto importante è quest'ultimo valore, dimostra come siano quasi nulle le perdite di potenza dovute all'impedenza d'ingresso.

Figura 6: Parte reale e parte immaginaria dell'impedenza adattate alle specifiche di progetto.



La figura 7, infine, mostra il diagramma polare del guadagno in dB, sul taglio $E(\phi = \frac{\pi}{2})$ nella direzione di *broadside*, $\theta = \frac{\pi}{2}$. Il guadagno è pari a $7.26dB$.

2 Progettazione dell'array di patch

Una volta soddisfatte le specifiche per il singolo patch rettangolare, è necessario cercare di soddisfare anche le restanti, relative all'array.
Per fare questo è possibile agire su due fattori relativi all'array: il numero degli elementi che formano l'array e la spaziatura tra gli elementi. Per calcolare il numero di elementi che formano l'array ci si deve riferire alla specifica del guadagno massimo dell'array, la relazione che lo lega con il guadagno del singolo patch permette di ottenere il numero di elementi che formano l'array considerando un'illuminazione uniforme.

$$G_{max}^{(array)} = (K+1)G_{max}^{(patch)}$$

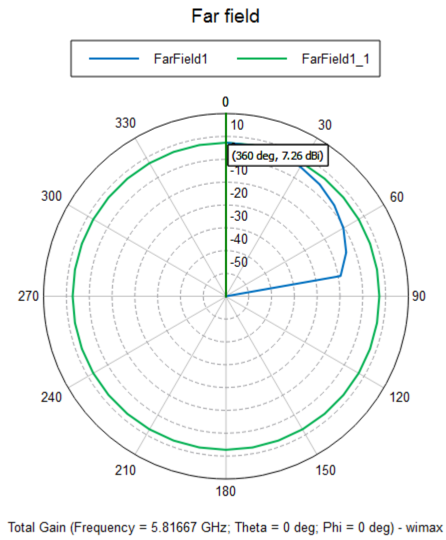
$$\Rightarrow M = K + 1 = \left\lceil \frac{G_{max}^{(array)}}{G_{max}^{(patch)}} \right\rceil = \lceil 7.71 \rceil = 8 \tag{6}$$

dove:

- $G_{max}^{(array)}$ è dato dalla specifica di progetto: $16dB$
- $G_{max}^{(patch)}$ è quello ottenuto nella figura 7: $7.26dB$

Il numero di elementi trovato nell'equazione 6 rappresenta il numero *minimo* di elementi che possono formare l'array a illuminazione uniforme; in questo lavoro, però, si

Figura 7: Diagramma polare del guadagno in scala logaritmica.



studia la progettazione di un array a illuminazione simmetrica, pertanto la scelta del numero *minimo* di elementi si opera attraverso le seguenti equazioni:

$$N = \left\lceil \frac{M-1}{2} \right\rceil = 4 \Rightarrow M = 2N + 1 = 9 \tag{7}$$

Il numero di elementi *minimo* trovato nella (7) soddisfa il requisito sul guadagno massimo dell'array, quest'ultimo passaggio è stato necessario al fine di poter applicare la sintesi di Čebyšëv. In questo modo sarà possibile calcolare il numero ottimo di elementi che formano l'array e la loro distanza.
Restano quindi da soddisfare gli ultimi due requisiti che riguardano *il rapporto tra l'ampiezza massima del lobo principale e quella del lobo secondario* e *l'angolo a metà potenza (BW)*.

2.1 Sintesi di Čebyšëv

La sintesi di Čebyšëvè stata automatizzata attraverso un programma in Python, di cui è allegato il codice.

Il primo script contiene le funzioni che calcolano i parametri di Čebyšëv, i coefficienti di alimentazione e il fattore d'array e eseguono la sintesi dando in output tutti i risultati.

```
import math
from pylab import*

# the function takes in input the number of array elements "n" and
# the attenuation of the secondary lobes compared to the main lobe
```

```
def chebyParam( n , r ):
    n = float(n)
    r = float(r)
    iterator = math.cosh(1/n * math.acosh(r))
    a = (( iterator - 1 ) / 2 ).real
    b = (( iterator + 1 ) / 2 ).real
    #returns the list of chebyshev's parameters
    return [ a , b ]

#chebyshev's parameters and optimized distance value btw array elements
def chebyParamOptimized(n , r , k0 ):
    n = float(n)
    r = float(r)
    iterator = math.cosh(1/n * math.acosh(r))
    a = (( iterator - 1 ) / 2 ).real
    b = (( iterator + 1 ) / 2 ).real
    optimized_dist = ( 2*math.pi - math.acos((1 - a)/b)) / k0
    output = open("valori_a_b.txt",'a')
    output.write(" a = " + str(a) + " b = " + str(b) + "  N = " + str( n*2+1) + "\n")
    output.close()
    return [ a , b , optimized_dist]

# the function takes in input the chebyshev's parameters and returns the
# excitation coefficients

def excitCoeff( n , a, b):
    if n not in range(2,5):
        raise Exception("The number isn't in the range [ 2 , 4 ]")
    # 5 elements array
    def five_elem():
        k0 = 2*a**2 + b**2 - 1
        k1 = 4*a*b/2
        k2 = b**2/2
        return [ k0 , k1 , k2 ]
    #7 elements array
    def seven_elem():
        k0 = 4*a**3 + 6*a*b**2 - 3*a
        k1 = ( 12*a**2*b +3*b**3 - 3*b )/2
        k2 = ( 6*a*b**2 )/2
        k3 = b**3/2
        return [ k0 , k1 , k2 , k3 ]
    #9 elements array
    def nine_elem():
        k0 = -8*a**2 + 8*a**4 - 4*b**2 + 3*b**4 + 24*a**2*b**2 +1
        k1 = (24*a*b**3 + 32*a**3*b - 16*a*b)/2
        k2 = (4*b**4 - 4*b**2 + 24*a**2*b**2)/2
        k3 = 8*a*b**3/2
        k4 = b**4/2
        return [ k0 , k1 , k2 , k3 , k4 ]

    feed_coeff = { 2 : five_elem , 3 : seven_elem , 4 : nine_elem }
    return feed_coeff[n]()

#the function takes in input the excitation coefficients, the distance between array elements,
#the angle of orientation
#returns the array factor

def arrayFactor( coeff , dist , angle , k0 ):
    u = k0 * dist * cos((angle*pi)/180)
    f = zeros(len(angle))
    for item in range(0,len(angle)):
        f[item] = coeff[0]
        for item2 in range(1,len(coeff)):
```

```
            f[item] = f[item] + 2*coeff[item2]*cos(item2*u[item])
    return f


# Chebyshev sysnthesis

def chebyshevSynthesis( f0 , r , angle , gain , n ):

    gain = 10**(array(gain)/10)
    c = float(3*(10**8))
    #wave length
    r = 10**(r/20)
    lambda_0 = c/float(f0)
    k0 = 2*math.pi/lambda_0
    psi = 90 - np.array(angle)
    m = int(ceil(( n-1 )/2))
    params = chebyParamOptimized( m , r , k0)
    excitation_coeff = excitCoeff( m, params[0] , params[1] )
    array_factor = np.array(arrayFactor( excitation_coeff, params[2] , psi , k0))
    #square absolute value of the array factor
    abs_array_factor = abs(np.array(array_factor))**2

    # the sum of the absolute values of the excitation parameters
    sum_coeff = sum( abs(array(excitation_coeff))**2)*2 - abs(excitation_coeff[1])**2

    array_factor_gain = abs_array_factor/float(sum_coeff)
    #the gain of the antenna
    system_gain = array_factor_gain * gain
    db_system_gain = 10*log10(system_gain)


    max_gain = max(db_system_gain)

    db_3_gain = 0
    teta_3_db = -1

    for  item in range(1, len(angle)):
        if db_system_gain[item] == db_3_gain :
            teta_3_db = angle[item]
            break
        elif db_system_gain[item] < db_3_gain:
            teta_3_db = angle[item - 1]
            break


    if teta_3_db == -1 :
        print(" Impossible to calculate the beamwidth ")
    print(teta_3_db)
    bw = 2*teta_3_db

    return [ array_factor , array_factor_gain , system_gain , max_gain , bw ]


def chebySynthesisDistance( f0 , r , angle , gain , n , dist ):
    c = float(3*10**8)
    r = 10**(r/20)
    lambda_0 = c/float(f0)
    k0 = 2*math.pi/lambda_0
    psi = 90 - array(angle)
    m = int(ceil((n-1)/2))
    params = chebyParam( m , r )
    excitation_coeff = excitCoeff( m , params[0] , params[1] )
    array_factor = arrayFactor(excitation_coeff , dist , psi , k0 )
```

```
    abs_array_factor = abs(array_factor)**2
    sum_coeff = sum(abs(array(excitation_coeff))**2)*2 - abs(excitation_coeff[1])**2
    array_factor_gain = abs_array_factor/float(sum_coeff)
    system_gain = array_factor_gain*gain
    db_system_gain = 10*log10(system_gain)
    max_gain = max(system_gain)
    db_max_gain = max(db_system_gain)
    db_3_gain = db_max_gain - 3
    print(db_system_gain)
    teta_3_db = -1
    for  item in range(0, len(angle)):
        if db_system_gain[item] == db_3_gain:
            teta_3_db = angle[item]
            break
        elif db_system_gain[item] < db_3_gain:
            teta_3_db = angle[item - 1]
            break

    if teta_3_db == -1 :
        print(" Impossible to calculate the beamwidth ")

    bw = 2*teta_3_db
    return [ array_factor , array_factor_gain , system_gain , max_gain , bw ]

def plot_function(axes,values , names ):
    m = values[0]
    g=[]
    for item in axes:
        g.append(-1*item)
    g = g[::-1]
    axes = g + axes
    values = array(list(values[::-1])+list(values))
    plot(axes,values)
    ylim(-60,35)
    xlim(-90,90)
    annotate(str(m)[0:5], xy=(5, m+6), xytext=(5, m+6),bbox=dict(boxstyle="larrow", fc="w"), rotation = 35)
    grid(True)

    ylabel(names[0])
    xlabel(names[1])
    title(names[2])
```

Il seguente script genera i grafici del fattore d'array e del guadagno totale dell'antenna.

```
import antenna_package
import math
from pylab import*

r = float(20)
f0 = 5.8*10**9
file_gain = open("gainTotal.txt",'r')
file_angles = open("gain_angoli.txt",'r')

gain = []
angles = []
while True:
    line = file_gain.readline()
    if not line:
        break
    gain.append(float(line))
file_gain.close()

```

```
while True:
    line = file_angles.readline()
    if not line:
        break
    angles.append(float(line))
file_angles.close()

for item in gain:
    item = 10**item/10

n = [ 5,7,9 ]
array_factors = zeros((len(n),len(gain)))
array_factors_gain = zeros((len(n),len(gain)))
system_gain = zeros((len(n),len(gain)))
max_gain = zeros(len(n))
beam_width = zeros(len(n))
output = open("beamwidth.txt",'wa')
for item in range(0,len(n)):
    synthesis_result = antenna_package.chebyshevSynthesis( f0, r , angles , gain, n[item] )
    array_factors[item] = synthesis_result[0]
    array_factors_gain[item] = synthesis_result[1]
    system_gain[item] = synthesis_result[2]
    max_gain[item] = synthesis_result[3]
    beam_width[item] = synthesis_result[4]

for item in range(0,len(n)):
    output.write("N = " + str(n[item]) + " bw = " + str(beam_width[item]) + "\n")
output.close()
max_gain_db = 10*log10(max_gain)
array_factors_abs = abs(array_factors)
array_factors_abs_db = 20*log10(array_factors_abs)
array_factors_gain_db = 10*log10(array_factors_gain)
system_gain_db = 10*log10(system_gain)

g = []
for item in angles:
    g.append(-1*item)
g=g[::-1]
axes = g + angles
a = list(array_factors_abs_db[2][::-1]) + list(array_factors_abs_db[2])

for item in range(0,len(n)):
    names_array_factors = [ "db","angle","Array factor absolute value for " + str(n[item]) + " elements"]
    figure()
    antenna_package.plot_function(angles,array_factors_abs_db[item],names_array_factors)
    show()
    names_system_gain = ["db","angle"," Gain of the " + str(n[item]) + " elements array patch antenna"]
    figure()
    antenna_package.plot_function(angles,system_gain_db[item],names_system_gain)
    show()
```

L'ultimo script genera i grafico dell'andamento del beamwidth in funzione della spaziatura tra i patch.

```
import antenna_package
import math
from pylab import*

n = 5
c = 3.0*10**8
r = 20.0
f0 = 5.8*10**9
lambda_0 = c/f0
```

```python

file_gain = open("gainTotal.txt",'r')
file_angles = open("gain_angoli.txt",'r')

gain = []
angles = []
while True:
    line = file_gain.readline()
    if not line:
        break
    gain.append(float(line))
file_gain.close()

while True:
    line = file_angles.readline()
    if not line:
        break
    angles.append(float(line))
file_angles.close()

for item in range(0,len(gain)):
    gain[item] = 10**(gain[item]/10)

#distances between array elements
distances = arange(lambda_0/2,lambda_0,0.01)
array_factor = zeros((len(distances),len(angles)))
array_factor_gain = zeros((len(distances),len(angles)))
system_gain = zeros((len(distances),len(angles)))
max_gain = zeros(len(distances))
beam_width = zeros(len(distances))

for item in range(0,len(distances)):
    synthesis_results =  antenna_package.chebySynthesisDistance(f0, r , angles, gain, n, distances[item])
    synthesis_results[0]
    array_factor[item] = synthesis_results[0]
    array_factor_gain[item] = synthesis_results[1]
    system_gain[item] = synthesis_results[2]
    max_gain[item] = synthesis_results[3]
    beam_width[item] = synthesis_results[4]
print(len(beam_width))
print(len(distances))
plot(distances,beam_width)
grid(True)
xlabel( "distance between pathces" )
ylabel(" beamwidth")
title(" Beamwidth variation for " + str(n) + " elements array antenna " )
show()
```

2.2 Risultati

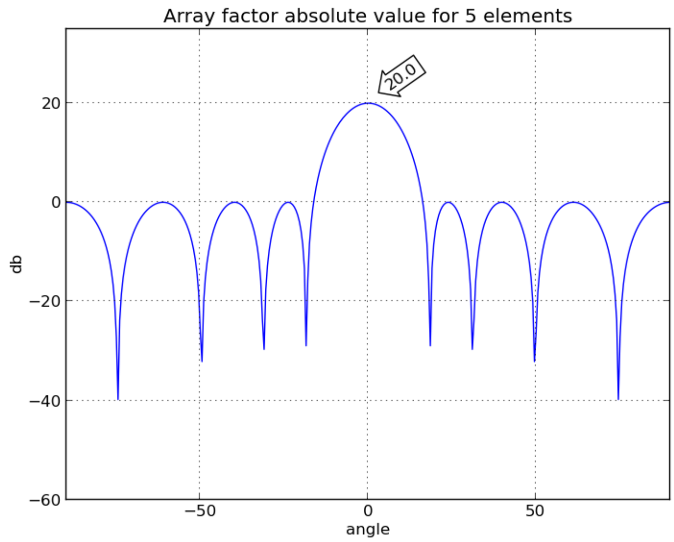
I risultati della sintesi di Čebyšëvsono contenuti in grafici che rappresentano il guadagno del fattore di array, il guadagno totale del patch e dell'array e l'angolo a metà potenza. Per ognuna di queste tre tipologie è stato considerato un numero di elementi dell'array variabile: $M = 5, 7, 9$.
Si è inoltre considerata la spaziatura ottima tra gli elementi, utilizzando i valori ottenuti dalla sintesi di Čebyšëv(Tab. 2) mediante la formula:

$$d_{ottima} = \frac{2\pi - arcos(\frac{1-a}{b})}{k_0} \tag{8}$$

Tabella 2: Valori di a e b ottenuti dalla sintesi di Čebyšëv.

	$M = 5$	$M = 7$	$M = 9$
a	0.672603939956	0.270214865098	0.146645950261
b	1.67260393996	1.2702148651	1.14664595026

Figura 8: Modulo del fattore di array per M = 5.



Nei grafici che rappresentano il modulo del fattore di array (Fig. 8, 9, 10) e il guadagno totale (Fig. 11, 12, 13) si è considerata una spaziatura d fissa, calcolata per $M = 5, 7, 9$.

Nelle figure 8, 9, 10 è mostrato il modulo del fattore di array per $M = 5, 7, 9$. Si può vedere chiaramente come all'aumentare di M aumentino il numero di lobi secondari, ma altrettanto evidente è la differenza, *in dB*, tra il lobo principale e il lobo secondario. In ogni grafico, quindi per ogni M, si può vedere come sia soddisfatta la specifica su R, la quale era richiesta essere di $-20dB$, tale, infatti, è la differenza in ampiezza tra il lobo principale e quelli secondari, che sono invece fermi a $0dB$.

Per quanto riguarda il guadagno totale del patch e dell'array (Fig. 11, 12, 13), si può vedere come all'aumentare di M diminuisca la larghezza dei lobi (in particolare di quello principale) e aumenti il numero di lobi secondari. Molto importante è anche l'aumento del guadagno che arriva ad un valore prossimo a $16dB$ (come previsto dalle specifiche) per $M = 9$, avendolo impostato nella relazione (6).

Infine è stata fatta variare la distanza tra gli elementi dell'array (tra $\frac{\lambda}{2}$ e λ) ed

Figura 9: Modulo del fattore di array per M = 7.

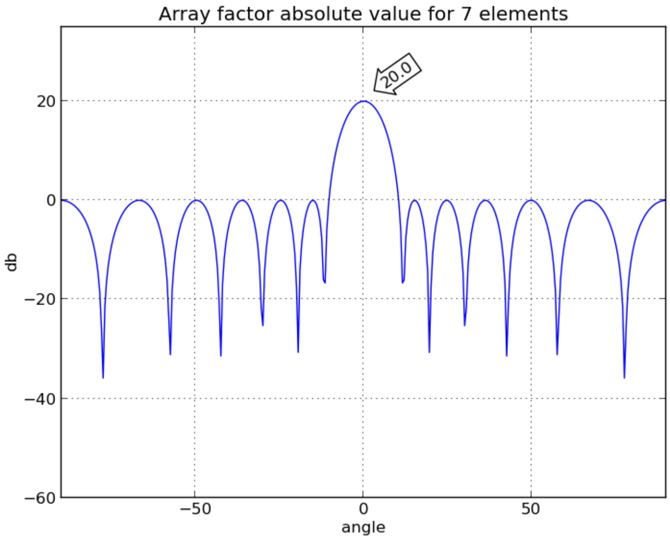


Figura 10: Modulo del fattore di array per M = 9.

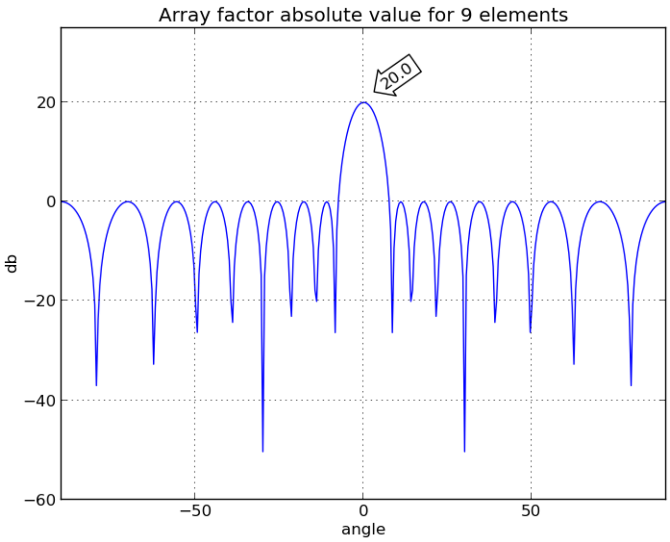


Figura 11: Guadagno totale per M = 5.

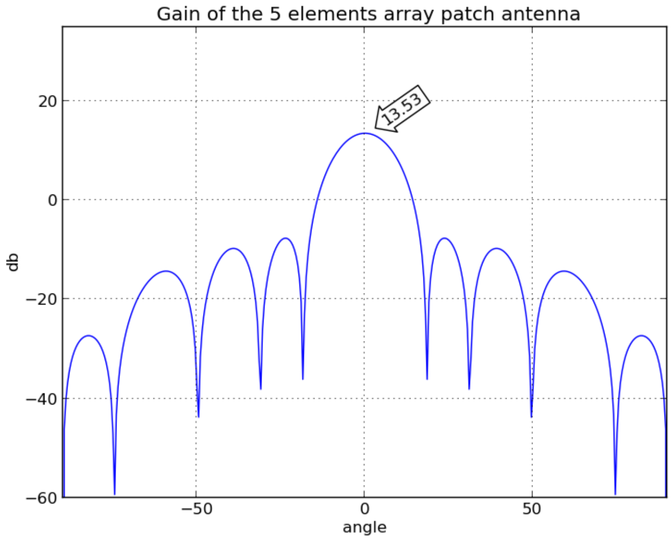


Figura 12: Guadagno totale per M = 7.

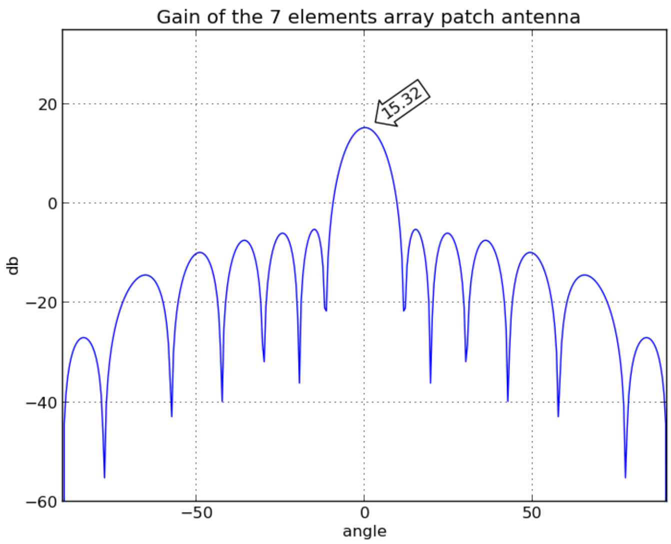
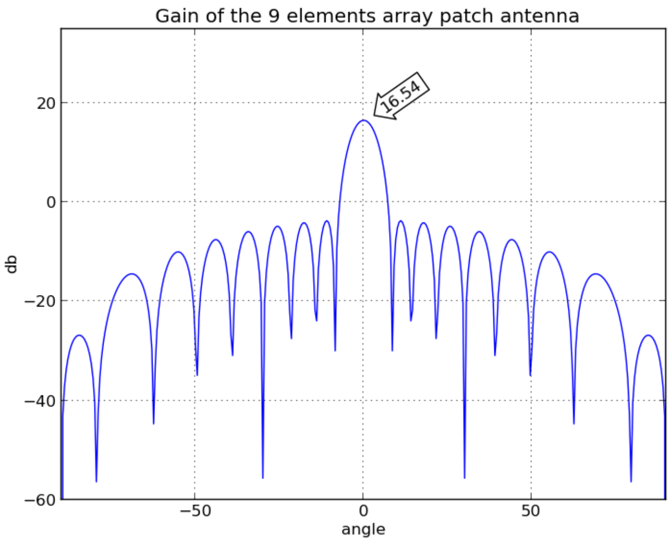


Figura 13: Guadagno totale per M = 9.



è stato rappresentato il beamwidth; in ognuno dei tre grafici è stata considerata una diversa cardinalità degli elementi degli array, $M = 5$ (Fig. 14), 7 (Fig. 15), 9 (Fig. 16).
Si può notare chiaramente come il *beamwidth* diminuisca all'aumentare della distanza tra gli elementi. Per capire se sia soddisfatta la specifica sull'angolo a metà potenza è necessario controllare il valore di 10° del *beamwidth*: nel caso di $M = 5$ nel range considerato di d non è possibile trovare il *beamwidth* richiesto, al contrario che nelle altre due configurazioni. In particolare si può notare come nel caso di $M = 9$ la scelta di d sia diversa da quella calcolata con la (8) ma nonostante questo, si può scegliere il relativo valore a $B = 10°$, in quanto le specifiche su R e su G, restano soddisfatte perché indipendenti dalla distanza d.

Nella tabella 3 sono riassunti i risultati ottenuti nelle tre tipologie di grafico rappresentate nelle pagine precedenti, in funzione della cardinalità degli elementi dell'array.

Tabella 3: Risultati al variare di M.

	$M = 5$	$M = 7$	$M = 9$
Distanza ottima (m)	0.0404149389796	0.043831493932	0.0457028645254
Guadagno max (dB)	13.53	15.32	16.54
Beamwidth	14.0°	9.0°	7.0°

Figura 14: Angolo a metà potenza per M = 5.

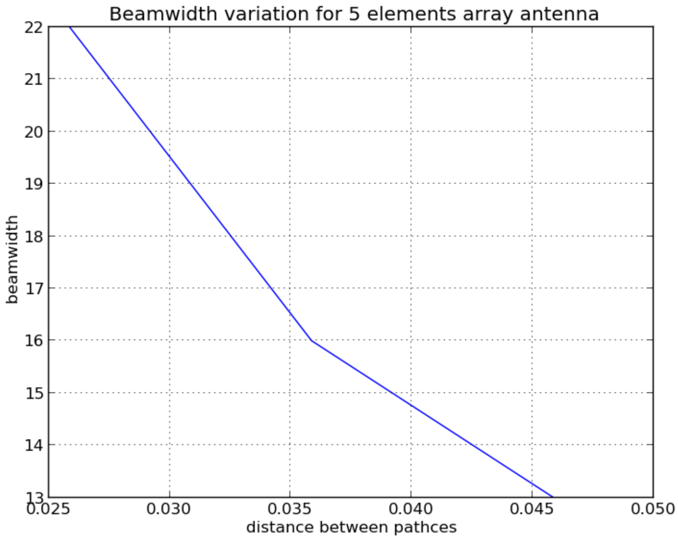


Figura 15: Angolo a metà potenza per M = 7.

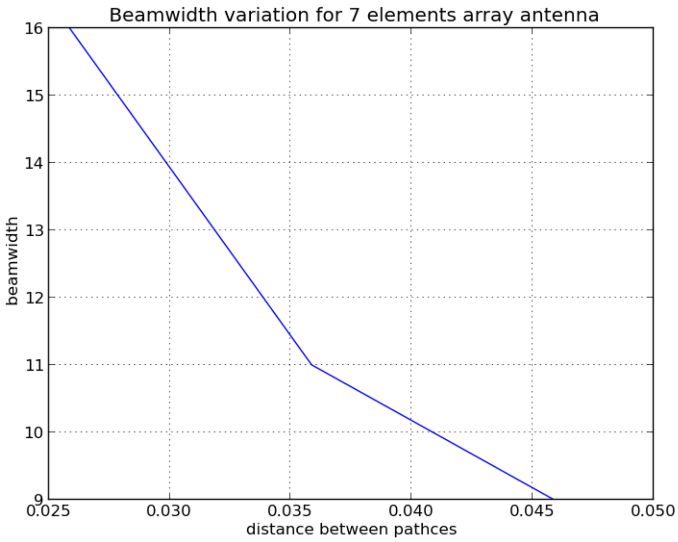
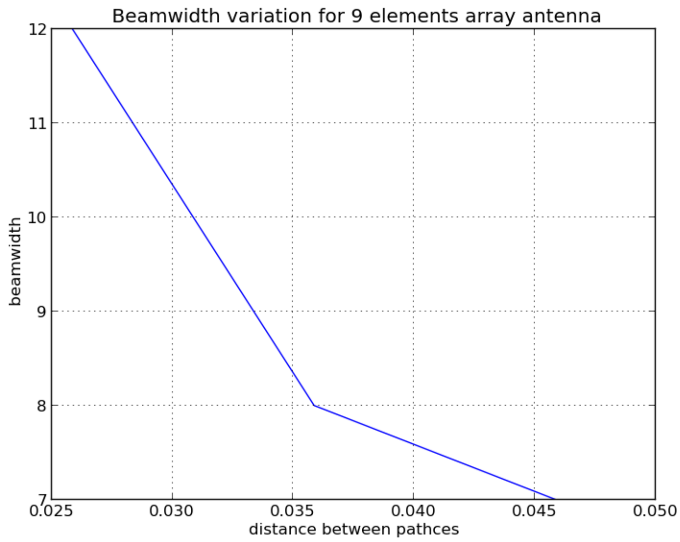


Figura 16: Angolo a metà potenza per M = 9.



3 Conclusioni

Lo studio effettuato nelle pagine precedenti mostra come la scelta ideale sia quella di $M = 9$ e $d = 0.031m$, questa scelta deriva dai risultati mostrati nei grafici 10, 13, 16.
Il primo mostra infatti come la specifica sulla differenza in dB tra il lobo principale e quello secondario, R, sia soddisfatta.
La seconda evidenzia il guadagno massimo del patch e dell'array, $16.54dB$, come da specifica; mentre nel terzo si vede chiaramente come la scelta di $0.031m$ per la distanza tra gli elementi dell'array sia ideale per ottenere un angolo a metà potenza pari a $10°$.

Un'ulteriore scelta potrebbe essere $M = 7$ con la distanza ottima calcolata nella (8), $d = 0.043831493932$, questa scelta soddisferebbe sicuramente i requisiti su R e B, ma il guadagno sarebbe di circa $0.7dB$ inferiore a quello richiesto.