A SPACE MAPPING MODEL FOR MICROSTRIP RECTANGULAR PATCH ANTENNAS

J.W. Bandler, M. H. Bakr and N. Georgieva

SOS-99-17-R

June 1999

A SPACE MAPPING MODEL FOR MICROSTRIP RECTANGULAR PATCH ANTENNAS

J.W. Bandler, M. H. Bakr and N. Georgieva

Simulation Optimization Systems Research Laboratory
Department of Electrical and Computer Engineering
McMaster University, Hamilton, Canada L8S 4K1

Introduction

Printed antennas on dielectric substrates are nowadays used in a number of applications encompassing all kinds of wireless communication services as well as in satellite and radar technology. The single patch radiators are typically very narrow-band. Their far-field pattern has a relatively broad main beam (60-80 degrees) in both, the E-plane and the H-plane. By arranging the single patches in arrays though, one can achieve high directivity and the possibility to electronically control the far-field pattern.

Unfortunately, the analysis and the design of microstrip patch antennas and arrays is a troublesome, slow and expensive process. This is due to the fact, that the current equivalent-circuit and empirical models are not accurate enough, especially for the requirements of the printed circuit board technology, which does not leave room for post-manufacturing tuning. Recently, a variety of full-wave electromagnetic simulators has become available in the software market. Some of them (Sonnet's ***em***™ [1], HP Momentum [2], etc.) are especially well suited for the analysis of layered structures. Software tools of this type can be used for the analysis of single-patch radiators or very small arrays (two to four elements). Even in these cases, contemporary personal computers or workstations of average performance would require days of simulation time. This makes the design process based on full-wave simulations impractical.

An opportunity to make the design based on full-wave simulators a feasible task can be found in the Space Mapping optimization techniques [3, 4]. Space Mapping requires a fast although not very accurate model, which would be used to carry out the bulk of the simulations in the design loop.

This report describes the implementation of the well known transmission-line model of a single rectangular patch radiator. It is an excellent candidate for a coarse model of a single patch when the input impedance of the antenna is of primary importance to the design. The program can be easily expanded to calculate other antenna parameters, such as far-field patterns, beam width, directivity, etc.

General Structure Description

The microstrip patch antenna (MPA) is designed for a substrate of height h=1.27 mm and a dielectric constant of e_r=10.2. The antenna is fed through a 75 O microstrip line whose width is w_0=0.43 mm. The antenna must be designed to have approximately 75 O input impedance at f_0=1.6 GHz. The frequency range of interest is defined between 1.55 GHz and 1.65 GHz. The optimization specification are:

at f=1.6 GHz, Re{Z_{11}}=75 O and Mag{Z_{11}}=75 O
at 1.55 GHz $\leq f \leq$ 1.59 GHz, Re{Z_{11}} $\leq$ 10 O
at 1.61 GHz $\leq f \leq$ 1.65 GHz, Re{Z_{11}} $\leq$ 10 O.

The microstrip patches have generally very high impedance if they are fed directly at the edge [5, 6]. To achieve low input impedance, an inset distance *y* is introduced at the feed point (see Fig. 1). The input impedance is very sensitive to the values of the inset *y*.

The gap between the inset-feed line *d* and the patch introduces parasitic capacitance, which can change the resonant frequency of the antenna and its input impedance. Generally, this capacitance can be compensated by slight changes in the inset distance *y*. That is why *d* is kept constant at half the width of the feed line w_0.

The width *w* affects slightly the dominant mode resonant frequency, too, but this frequency can be adjusted through the length *l*. The patch width is kept constant at a value, which would ensure good radiation efficiency. A practical formula for calculation of the patch width is given in [7].

$$w = \frac{1}{2 f_0 \sqrt{\mu_0 e_0}} \sqrt{\frac{2}{e_r + 1}} \tag{1}$$

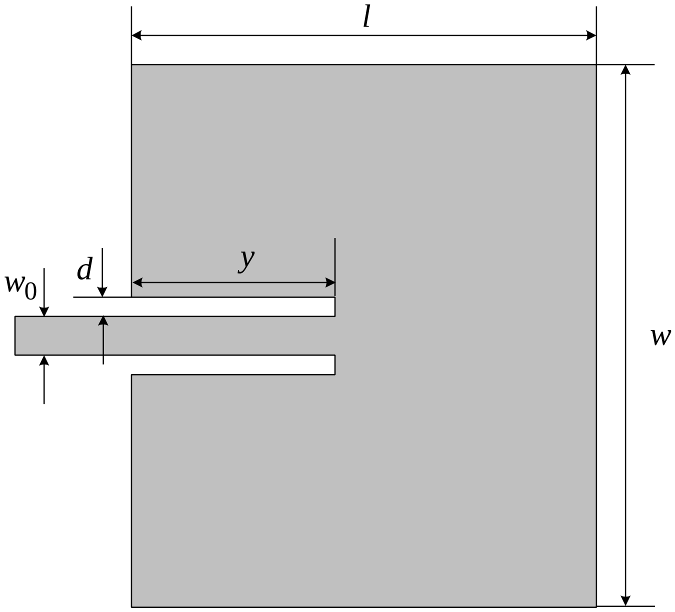


Fig. 1. The microstrip fed patch antenna with a feed-inset distance.

In this design, the patch width is fixed at *w*=39.99 mm, which is close to the value obtained via (1) and is a multiple of the step size in the fine model. Thus, there are two optimization parameters: the patch length *l*, and the inset distance *y*.

The fine model is a Sonnet's ***em***™ simulation. The nominal project's geometry is shown in Fig. 2. The fine model applies very fine discretization steps along both axes, *x* and *y*. This is dictated by the small size of the gap introduced by the feed inset and by the sensitivity of the problem with respect to the inset distance *y* and the patch length *l*. This is well seen in the Geometry Capture Editor Window in Fig. 3.

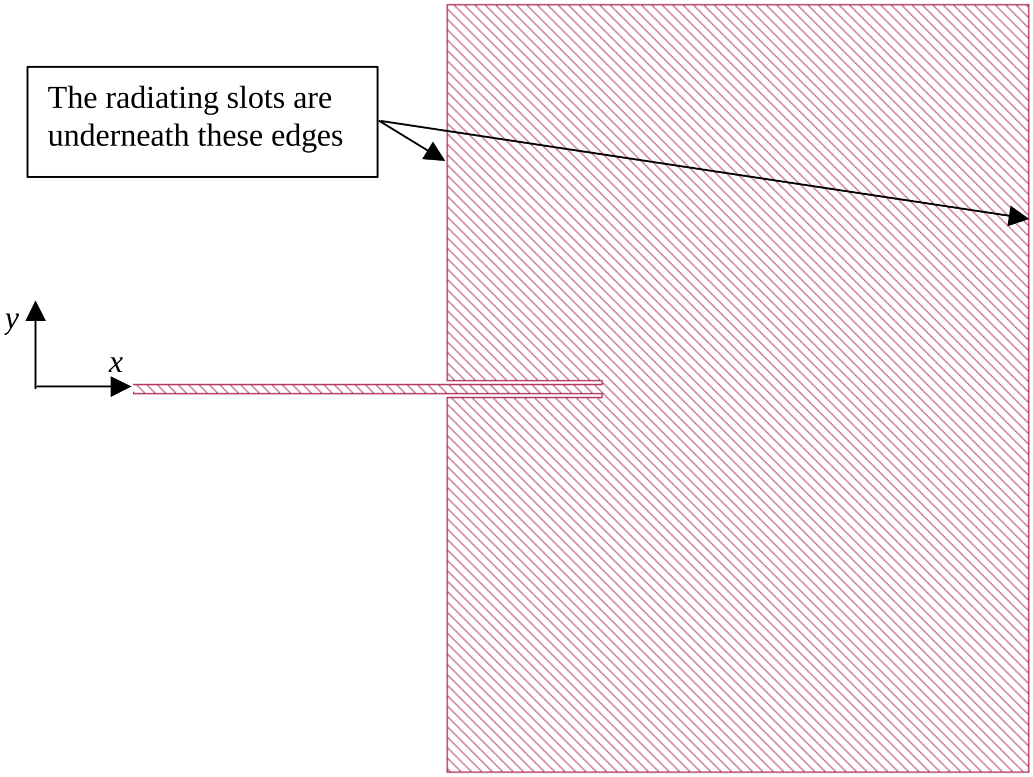
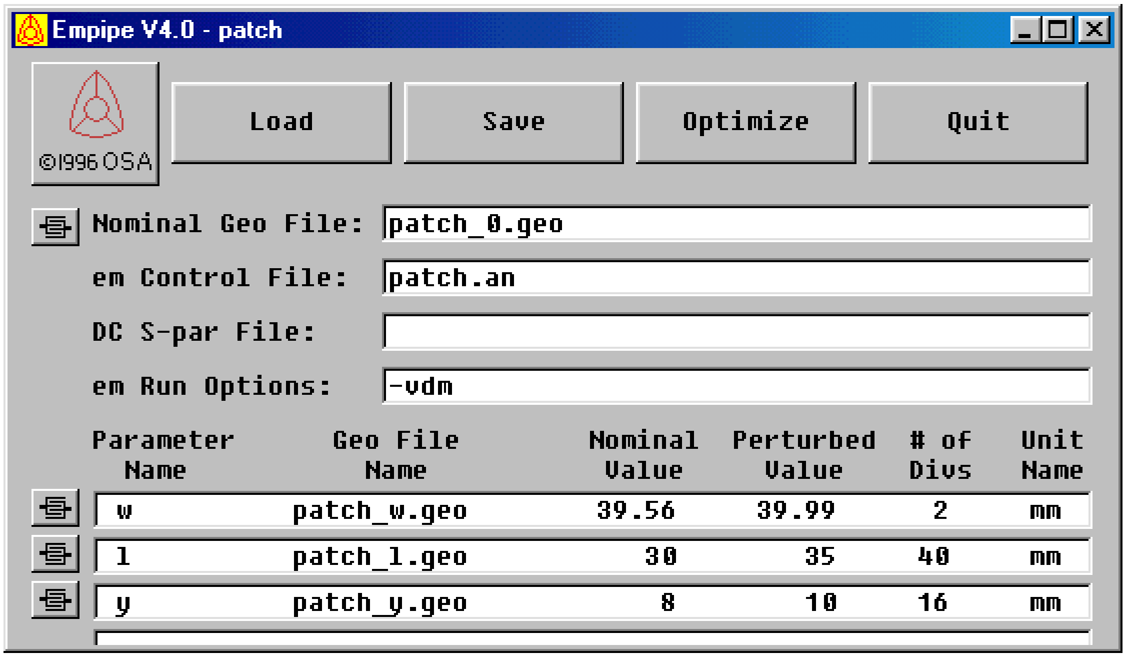


Fig. 2. The nominal geometry of the fine model (Sonnet's ***em*** simulation).



Nominal Geo File: patch_0.geo

em Control File: patch.an

DC S-par File:

em Run Options: -vdm

Parameter Name	Geo File Name	Nominal Value	Perturbed Value	# of Divs	Unit Name
w	patch_w.geo	39.56	39.99	2	mm
l	patch_l.geo	30	35	40	mm
y	patch_y.geo	8	10	16	mm

Fig. 3. Geometry capture settings for the fine model (Sonnet's ***em***™) of the MPA.

The Coarse Model

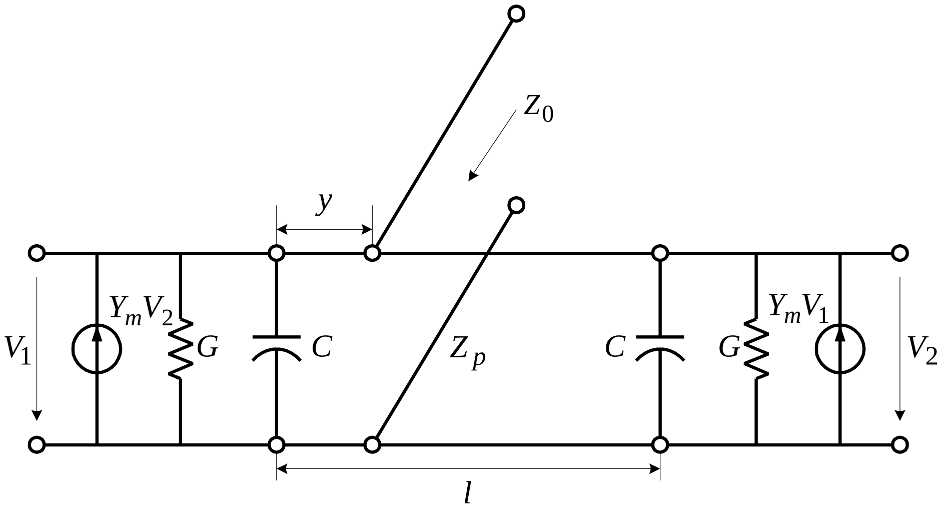


Fig. 4. Transmission-line model of a rectangular MPA with a feed inset.

The MPA can be described via the transmission-line model [6], which provides acceptable accuracy for initial design purposes. Each patch edge is a parallel combination of a shunt capacitor (edge effect) and a conductance (radiation effect). The two voltage controlled current sources represent the electromagnetic coupling between the two radiating slots formed by the front and the back patch edges (see Fig. 2). The edges are connected by a length of transmission line formed by the microstrip patch, which can be viewed as a microstrip line of a characteristic impedance Z_p (see Fig. 4), when it is operating in the dominant quasi-TEM mode. The feed microstrip line is represented by the transmission line of characteristic impedance Z_0˜ 75 O.

The calculation of C, G, V_1 and V_2 requires mathematical functions, which are not directly available in microwave circuit simulators such as OSA90/hope. A child program `pp.c` in C was made to carry out most of the calculations (see the Appendix B). The following computations are performed by `pp.c`.

Effective dielectric constant: of the patch $e^{p}_{r_{eff}}$

of the microstrip line $e^{0}_{r_{eff}}$

In both cases the formula of Hammerstad and Jensen [8] was used:

$$e_e = \frac{e_r+1}{2} + \frac{e_r-1}{2}\left(1+10\frac{h}{w}\right)^{-a\left(\frac{w}{h}\right)b(e_r)}, \qquad (2)$$

where

$$a\left(\frac{w}{h}\right) = 1 + \frac{1}{49}\ln\left\{\frac{\left(\frac{w}{h}\right)^4 + \left(\frac{w}{52h}\right)^2}{\left(\frac{w}{h}\right)^4 + 0.432}\right\} + \frac{1}{18.7}\ln\left\{1+\left(\frac{w}{18.1h}\right)^3\right\}$$

$$b(e_r) = 0.564\left(\frac{e_r - 0.9}{e_r + 3}\right)^{0.053}$$

Characteristic impedance: of the patch Z_p
of the microstrip line Z_0

Hammerstad and Jensen [8] provide an accurate empirical formula for the calculation of the characteristic impedance of a microstrip line.

$$Z_c = \frac{60}{\sqrt{e_e}} \ln\left[F_1\left(\frac{w}{h}\right)\frac{h}{w} + \sqrt{1 + \left(\frac{2h}{w}\right)^2}\right], \qquad (3)$$

where

$$F_1\left(\frac{w}{h}\right) = 6 + (2p - 6)\exp\left[-\left(30.666\frac{h}{w}\right)^{0.7528}\right].$$

The accuracy of the expressions (2) and (3) is better than 0.01% for $w/h \le 1$ and is better than 0.03% for $w/h \le 1000$. These formulas do not take into account the strip thickness. Equivalently accurate expressions for finite strip thickness can be found in [8]. The dispersive correction for the effective dielectric constant and the characteristic impedance are calculated as explained below [8].

The effect of dispersion

$$e_e(f) = e_r - \frac{e_r - e_e}{1 + (f/f_{50})^m}, \qquad (4)$$

where

$$f_{50} = \frac{f_{k,TM_0}}{0.75 + \left(\frac{0.332}{e_r^{1.73}}\right)\frac{w}{h}}; \quad f_{k,TM_0} = \frac{c \cdot \tan^{-1}\left(e_r\sqrt{\frac{e_e - 1}{e_r - e_e}}\right)}{2ph\sqrt{e_r - e_e}};$$

and

$$m = m_0 m_c,$$

$$m_0 = 1 + \frac{1}{1 + \sqrt{w/h}} + 0.32\left(\frac{1}{1 + \sqrt{w/h}}\right)^3,$$

$$m = \begin{cases} 1 + \frac{1.4}{1 + w/h}\left[0.15 - 0.235\exp\left(-\frac{0.45f}{f_{50}}\right)\right] & , w/h \le 0.7 \\ 1 & , w/h > 0.7 \end{cases}$$

Here e_e represents the quasi-static result of equation (2) and c is the speed of light. The dispersion effect on the characteristic impedance can be then calculated as:

$$Z_c(f) = Z_c \frac{e_e(f) - 1}{e_e - 1} \sqrt{\frac{e_e}{e_e(f)}}\,, \tag{5}$$

where Z_c is the quasi-static impedance calculated by (3).

Effective patch length

The edge effect at the front and at the back of the patch introduces an additional extension ?*l* to the patch physical length *l*. An empirical formula for the calculation of ?*l* is provided in [6], which is valid for 0.01 ≤ *w/h* ≤ 100 and e_r ≤ 128.

$$\Delta l = \frac{x_1 x_3 x_5}{x_4} h\,, \tag{6}$$

where

$$x_1 = 0.434907 \frac{e_{r_{eff}}^{0.81} + 0.26}{e_{r_{eff}}^{0.81} - 0.189} \frac{(w/h)^{0.8544} + 0.236}{(w/h)^{0.8544} + 0.87}$$

$$x_2 = 1 + \frac{(w/h)^{0.371}}{2.358 e_r + 1}$$

$$x_3 = 1 + \frac{0.5274 \tan^{-1}[0.084 (w/h)^{1.9413/x_2}]}{e_{r_{eff}}^{0.9236}}$$

$$x_4 = 1 + 0.5274 [6 - 5 e^{0.036(1-e_r)}] \tan^{-1}[0.067 (w/h)^{1.456}]$$

$$x_5 = 1 - 0.218 e^{-7.5 w/h}$$

Here $e_{r_{eff}}$ denotes the effective dielectric constant, which can be calculated without or with the dispersion effect included (see formulas 2 and 4). In the current implementation, the dispersion effect has been taken into account. The end-effect extension ?*l* can be related to the edge capacitance using the equivalence between a capacitive impedance and the impedance of a short (< *l*/4) section of a transmission line of characteristic impedance Z_c:

$$C = \frac{\tan(b \Delta l)}{w Z_c} \tag{7}$$

Here *b* =2*p*/ l_g, where $l_g = l_0 / \sqrt{e_{r_{eff}}}$ (l_0 is the free-space wavelength).

Radiation conductance

The radiation conductance of a single slot represents the power loss due to radiation. In general it is defined as:

$$G = \frac{2 P_{rad}}{|V|^2}\,, \tag{8}$$

where *V* is the voltage across the slot. Using the cavity model (see [6] or [7]) with its dominant mode one can obtain the radiated power of each slot as:

$$P_{rad} = \frac{|V|^2}{2ph_0}\int_0^p \Im(q)dq \ , \tag{9}$$

where

$$\Im(q) = \left[\frac{\sin\left(\frac{k_0 w}{2}\cos q\right)}{\cos q}\right]^2 \sin^3 q$$

$$?_0 = \sqrt{\mu_0 / e_0}\ .$$

Finally,

$$G = \frac{1}{120p^2}\int_0^p \Im(q)dq\ . \tag{10}$$

Mutual admittance

The mutual admittance of two radiators is generally defined in terms of the mutual radiated power. If the edges are excited by equal voltages (as they usually are), their mutual admittance Y_m is defined in [9]:

$$Y_m = \frac{1}{|V|^2}\iint \vec{E}_1 \times \vec{H}_2^* d\vec{s} \tag{11}$$

Here $\vec{E}_1$ is the electric field at one edge, $\vec{H}_2^*$ is the complex conjugate of the magnetic field at the other edge, and, $d\vec{s}$ is the surface element of a large hemisphere surrounding the patch structure. The real part of Y_m can be obtained from (11) as:

$$G_m = \frac{1}{120p^2}\int_0^p \Im_m(q)dq\ , \tag{12}$$

where

$$\Im_m(q) = \left[\frac{\sin\left(\frac{k_0 w}{2}\cos q\right)}{\cos q}\right]^2 J_0(k_0 l \sin q)\sin^3 q\ .$$

In (12), J_0 denotes the Bessel function of the first kind of order zero. Equation (12) has been reduced to a closed form expression [10], which is a good approximation to (12):

$$G_m = G\left[J_0(k_0 l) + \frac{s^2}{24 - s^2} J_2(k_0 l) \right] \tag{13}$$

Here G is the edge self-conductance (see equation 10), J_2 is the Bessel function of the first kind of order two, and $s = k_0 \Delta l$ (Δl being the end-effect extension, see formula 6).

The imaginary part of (11) can be also reduced to a simpler expression [10]:

$$B_m = \frac{p}{2} \frac{Y_0(k_0 l) + [s^2/(24 - s^2)]Y_2(k_0 l)}{\ln(s/2) + C^e - 1.5 + (s^2/12)/(24 - s^2)} \left(1 - e^{-0.21 \cdot k_0 w}\right) wC\,, \tag{14}$$

where

C^e =0.577216… is Euler's constant,
C is the capacitance of the egde (7),
Y_0 and Y_2 are Bessel functions of the second kind of order zero and two, respectively.

Bessel functions of the first and the second kind of integer order

These subroutines are a necessary accessory for the calculation of the antenna parameters. They are a direct translation from FORTRAN77 code available at

http://www.netlib.org/liblist.html

Note: The subroutines suggested in the *Numerical Recipes* books related to the computation of Bessel functions of the first and the second kind of integer order are not to be recommended.

The equivalent circuit shown in Fig. 4 is built in OSA90/hope™ [11]. The respective netlist file is given in Appendix A.

Verification and Comparisons Regarding the Coarse Model

Effective dielectric constant and characteristic impedance

Formulas (4) and (5), which propose an empirical model of the effective dielectric constant and the characteristic impedance, are implemented in `pp.c`. To verify the relevant subroutines, a comparison was made between this empirical model (Hammerstad et al. [8]) and the model used by R. Sainati in [6], which is actually the empirical model of Bahl et al. [5]. The results for the effective dielectric constant and the characteristic impedance of the feed line are given in Fig. 5 and in Fig. 6, respectively.

The effective dielectric constant and the characteristic impedance of the patch itself, as calculated by Hammerstad's and by Bahl's models, have been compared, too. The comparison in terms of $e_{r_{eff}}$ is shown in Fig. 7. The characteristic impedance comparison is given in Fig. 8. It is obvious from Figures 5 through 8 that both models are very close.

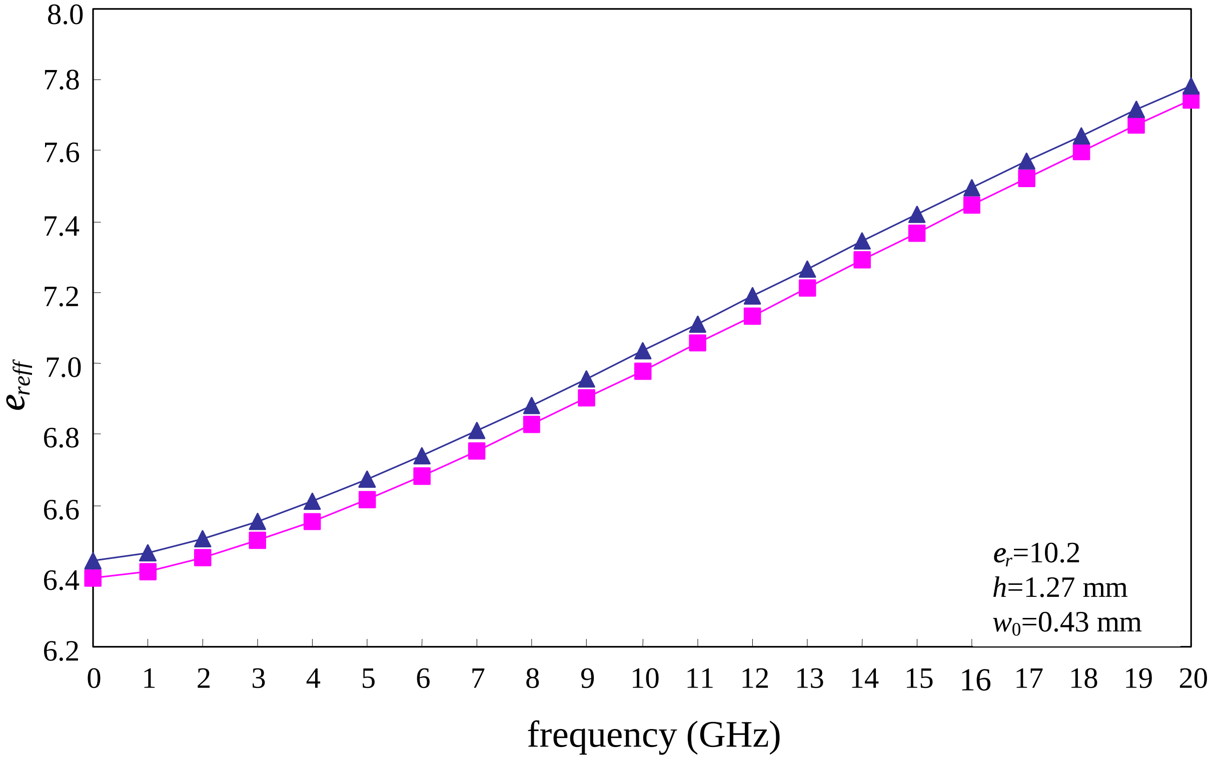


Fig. 5. Effective dielectric constant $e_{r_{eff}}$ of the microstrip feed line of the MPA: comparison between Hammerstad's (-■-) and Bahl's (-▲-) models.

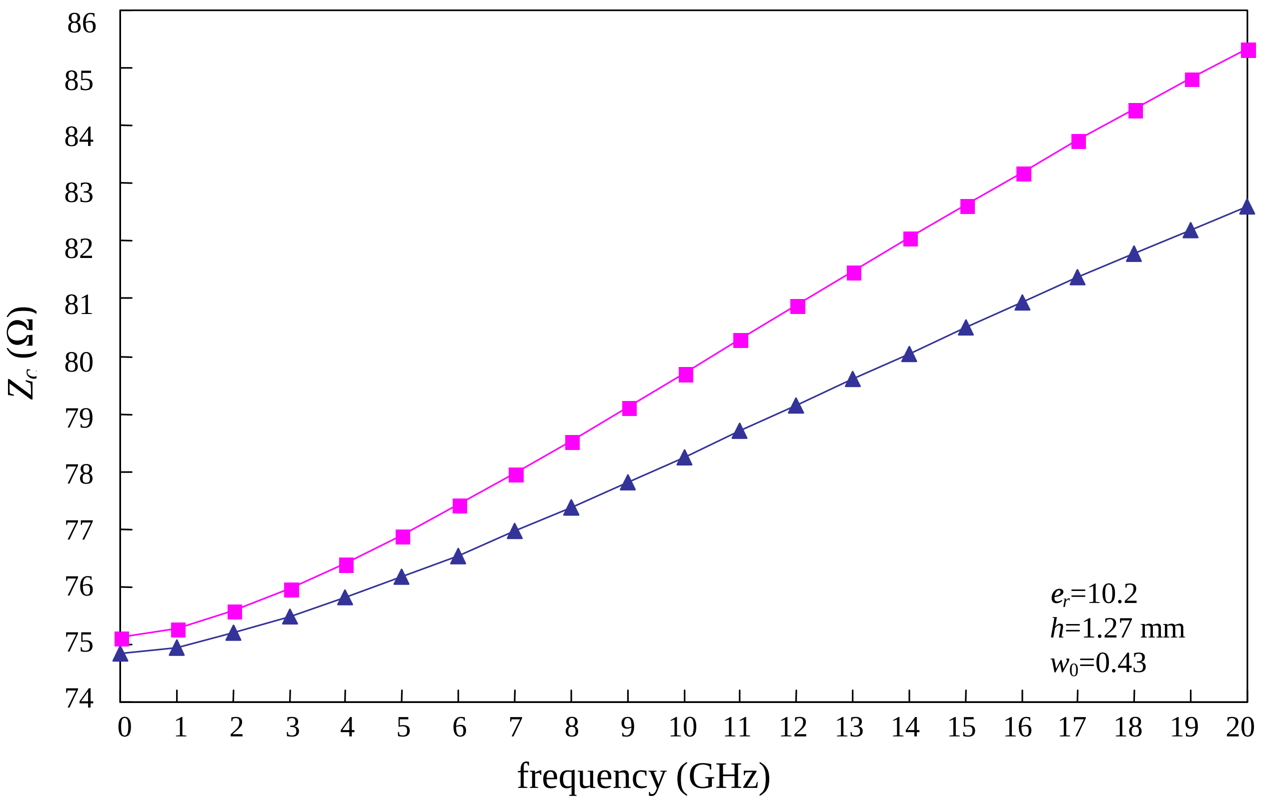


Fig. 6. Characteristic impedance Z_c of the microstrip feed line of the MPA: comparison between Hammerstad's (-■-) and Bahl's (-▲-) models.

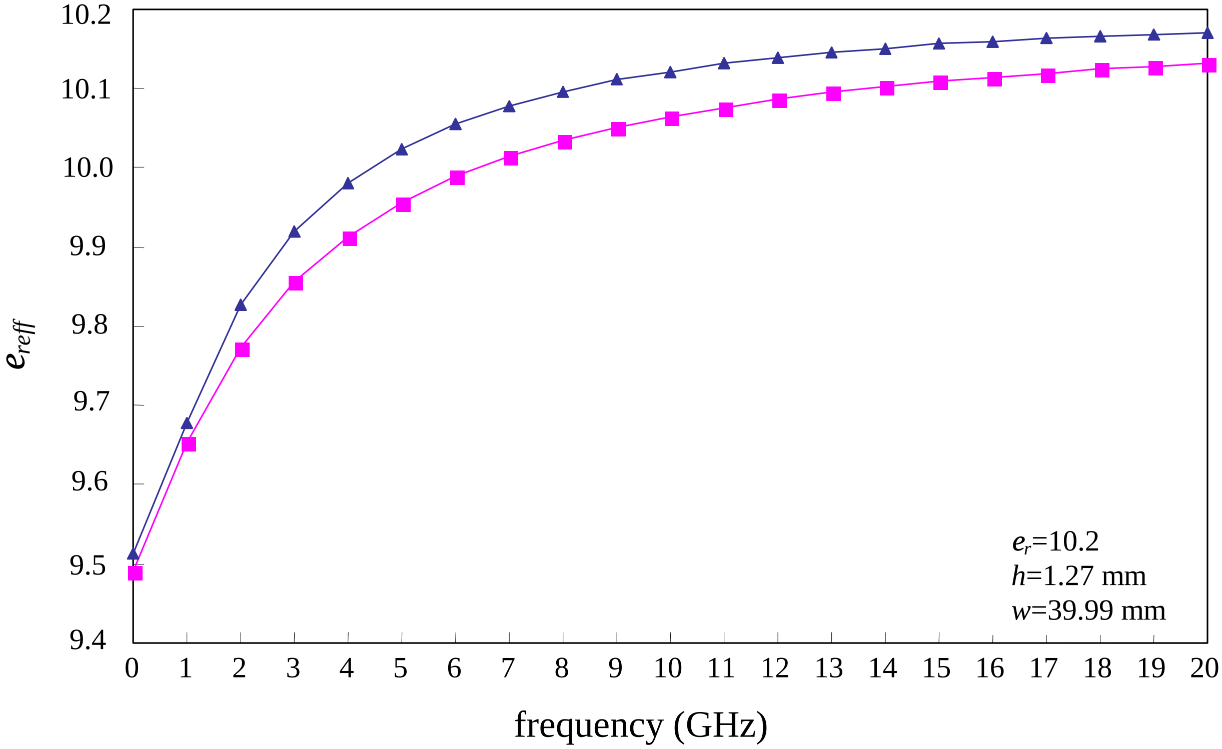


Fig. 7. Effective dielectric constant $\varepsilon_{r_{eff}}^{p}$ of the MPA: comparison between Hammerstad's (■) and Bahl's (▲) models.

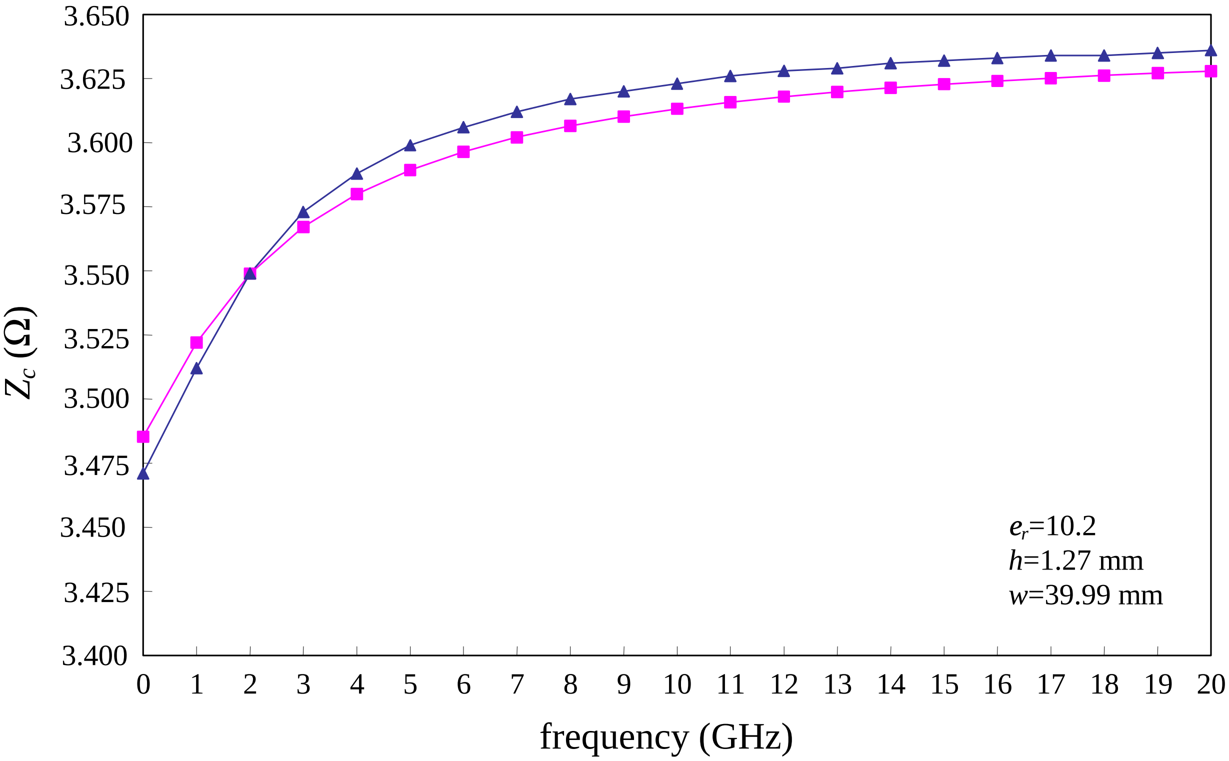


Fig. 8. Chracteristic impedance Z_c^p of the MPA: comparison between Hammerstad's (■) and Bahl's (▲) models.

End-effect extension Δl and end-effect capacitance C

The values of $e_{r_{eff}}$ and Z_c are used to calculate the end-effect extension Δ*l* (see expression 6) and the end-effect capacitance *C* (see formula 7). (6) is the model proposed in [12] and will be referred to as Jansen's model, which is recommended by Robert Sainati in [6]. Our current implementation uses Jansen's model of the end-effect in conjunction with Hammerstad's formulas for $e_{r_{eff}}$ and Z_c. Hammerstad also proposed a simple model for the end-effect extension Δ*l* [13]:

$$\Delta l = 0.412 \cdot h \cdot \frac{(e_{r_{eff}} + 0.3)\left(\frac{w}{h} + 0.264\right)}{(e_{r_{eff}} - 0.258)\left(\frac{w}{h} + 0.8\right)} \tag{15}$$

The above model (15) is compared with Jansen's model (6). *Both models* use the parameters $e_{r_{eff}}$ and Z_c as calculated via Hammerstad's expressions (2) through (5). The dispersive open-end capacitance *C* for a patch of width *w*=39.99 mm is given in Fig. 7 as calculated by Jansen's and by Hammerstad's formulas.

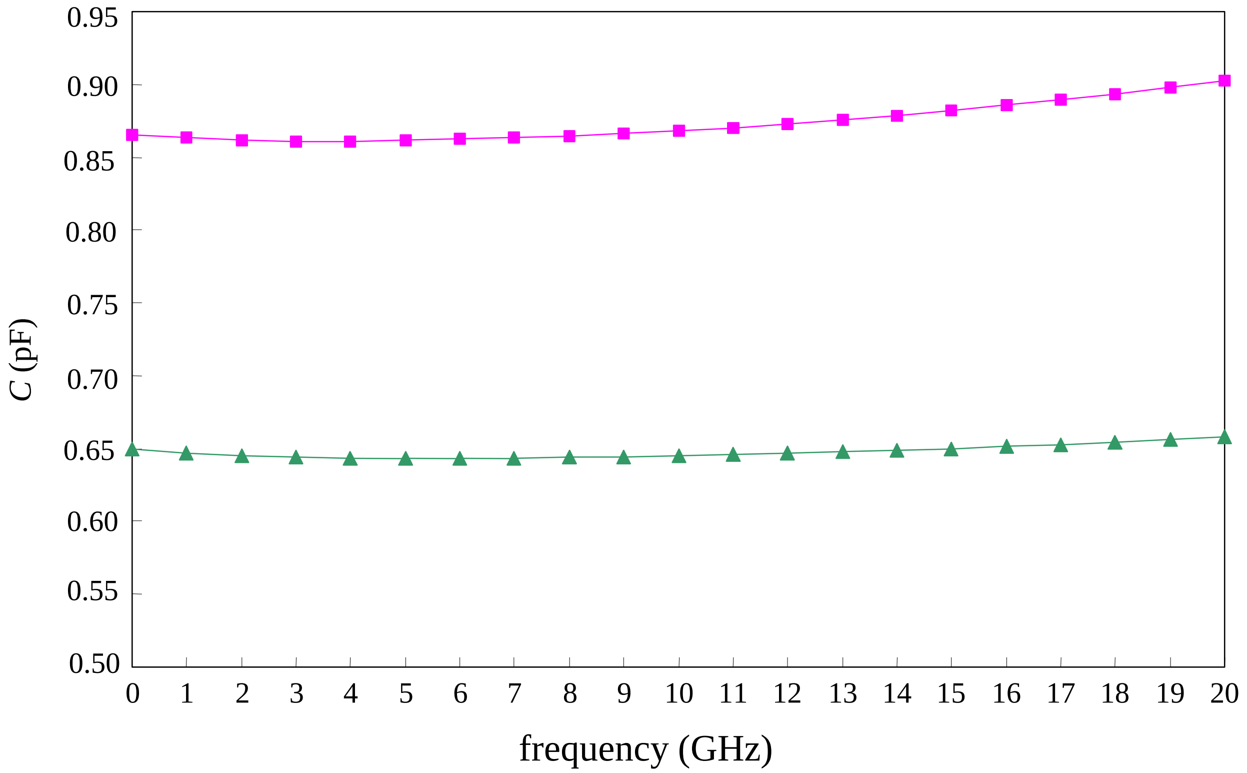


Fig. 9. Open-end capacitance C at the radiating edges of the MPA: comparison between Hammerstad's (■) and Jansen's (▲) models.

Microstrip-fed patch model

The initial design was done using the program `Patch9.exe`, which is a free executable supplied with [6] (source-code is not available). As explained in [6], `Patch9.exe` is based on the transmission-line model. The differences between `pp.c` and `Patch9.exe` should be only in:

1. `Patch9.exe` uses Bahl's model of a microstrip line ($e_{r_{eff}}$ and Z_c), while `pp.c` uses Hammerstad's model (formulas 2 through 5). This would results in slight differences in the calculation of the end-effect capacitance C.

2. `Patch9.exe` uses closed-form approximations to the single-slot radiated power integral (9) and to the mutual radiated power integral (11) while `pp.c` calculates those integrals numerically.

It can be expected, therefore, that `Patch9.exe` and `pp.c` (`pp.exe`) would produce different output. Figs. 10 and 11 provide a comparison among the input patch impedance as resulting from three models: `Patch9.exe`, `pp.exe` and Sonnet's ***em***. The comparison is made at the initial point of the coarse model (the `Patch9.exe` optimal design).

Note: `Patch9.exe` has a limited design capability: it can optimize only for the patch length l through an unknown optimization procedure.

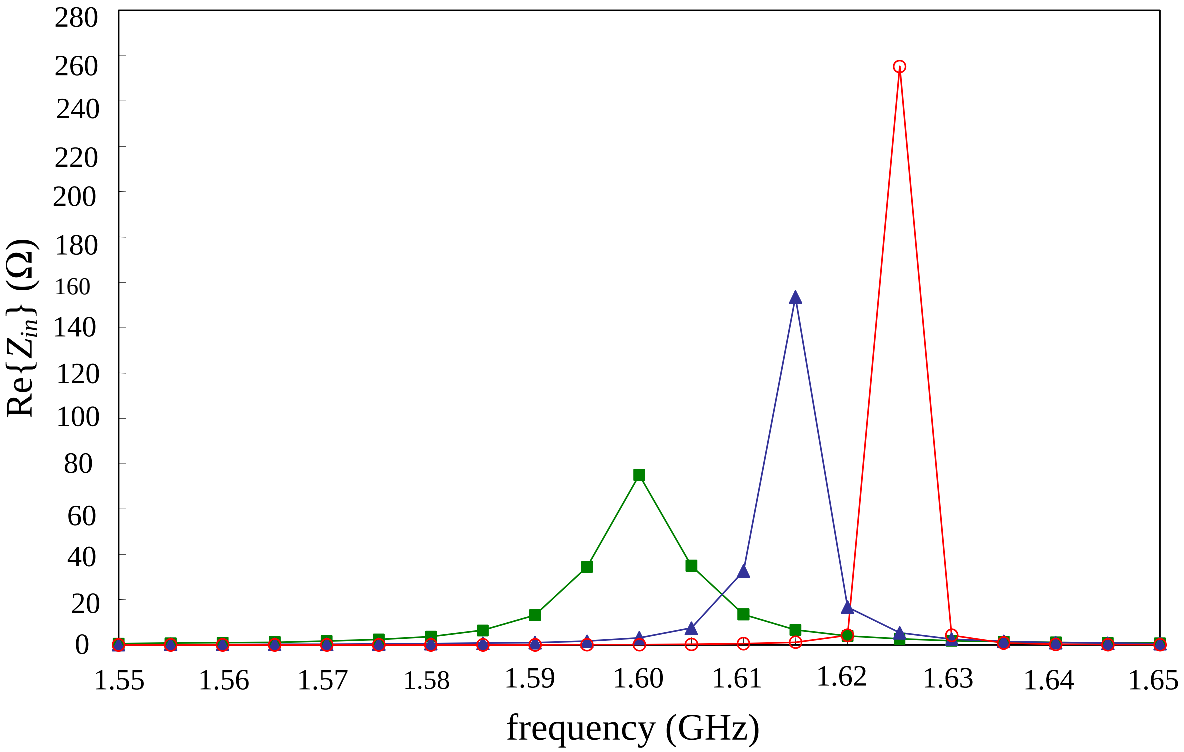


Fig. 10. Real part of the input impedance of the MPA: comparison among the fine model (Sonnet's ***em***) (–○–), the transmission-line model of R. Sainati `Patch9.exe (-■-),` and the transmission-line model implemented in `pp.c` (–▲–).

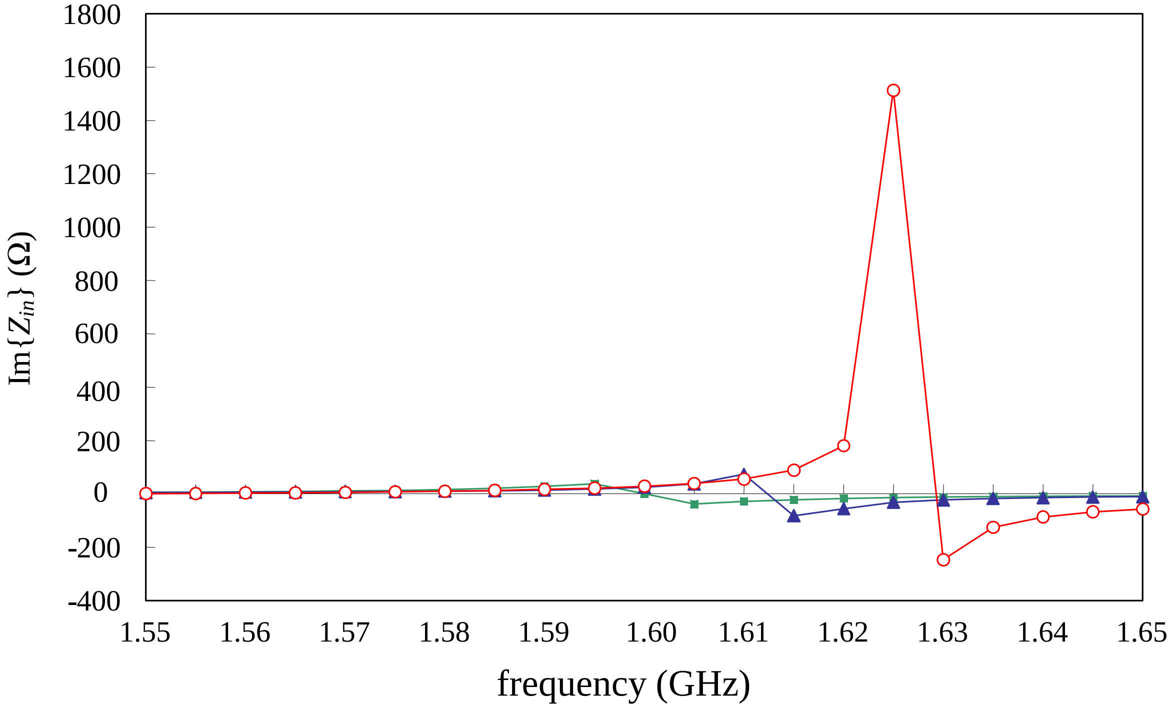


Fig.11. Imaginary part of the input impedance of the MPA: comparison between the fine model (Sonnet's ***em***) (─○─), the transmission-line model of R. Sainati `Patch9.exe` (─■─), and the transmission-line model implemented in `pp.c` (─▲─).

Optimization of the Coarse Model of the MPA

As mentioned before, the initial design was done using the program `Patch9.exe`. These values were used as the starting point for the optimization of the coarse model (`pp.c`). Here is the output file of `Patch9.exe`:

```
PATCH9.V50       05-28-1999          09:43:41

Substrate height (cm) =  0.1270         Line thickness (cm) = 0.0000
Relative dielectric constant = 10.200     Loss tangent = 0.0000
Patch width (cm) =   3.9990             Patch length (cm) =   2.8910
Feed type = microstrip line
Feed point inset (cm) =   1.0090
Feed line width (cm) =  0.043
Resonant frequency (GHz) =   1.600000
Resonant impedance =    74.995 j    -0.0119 ohms

FREQUENCY         INPUT IMPEDANCE
 (GHz)                (ohms)
1.550         0.589  J     6.424
1.555         0.732  J     7.184
1.560         0.931  J     8.126
1.565         1.221  J     9.323
1.570         1.666  J    10.892
1.575         2.395  J    13.035
1.580         3.705  J    16.110
1.585         6.392  J    20.807
1.585         6.392  J    20.807
1.590        13.085  J    28.338
1.595        34.497  J    37.262
1.600        74.995  J    -0.012
1.605        34.909  J   -37.505
1.610        13.493  J   -28.895
1.615         6.715  J   -21.492
1.620         3.964  J   -16.850
1.625         2.608  J   -13.802
1.630         1.847  J   -11.676
1.635         1.378  J   -10.116
1.640         1.069  J    -8.926
1.645         0.855  J    -7.989
1.650         0.701  J    -7.232
```

The starting and final values of the optimization variables are given in Table I.

TABLE I
OPTIMIZATION VARIABLES: INITIAL AND OPTIMIZED VALUES

Parameter	Initial value (mm)	Optimal value (mm)
l	28.91	29.1730
y	10.09	11.9568

The real part of the input impedance before and after the optimization is shown in Fig. 12 and in Fig. 13, respectively. The minimax objective-function error is given in the iteration report in Fig. 14.

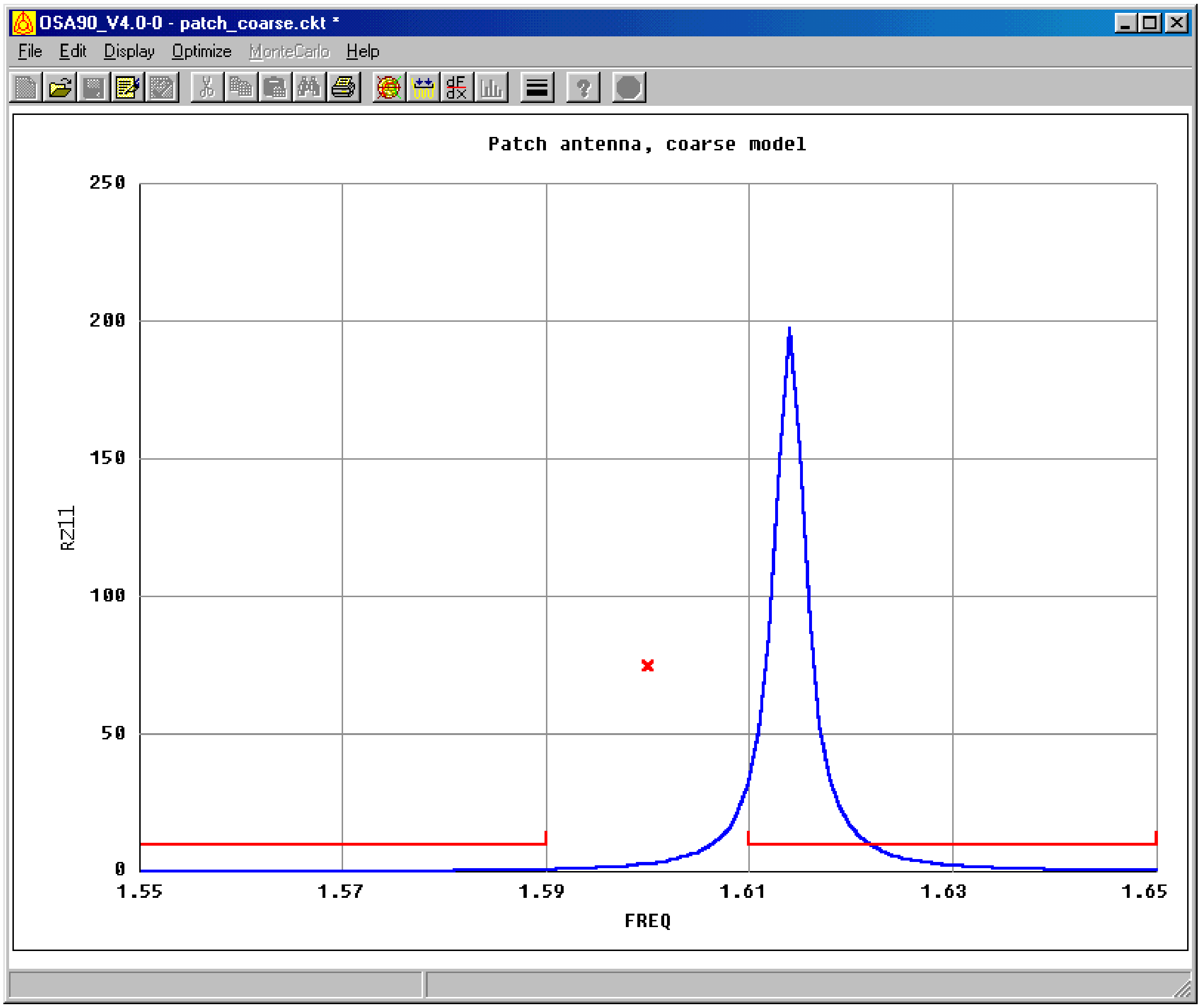


Fig. 12. The real part of the patch input impedance (coarse model) before optimization.

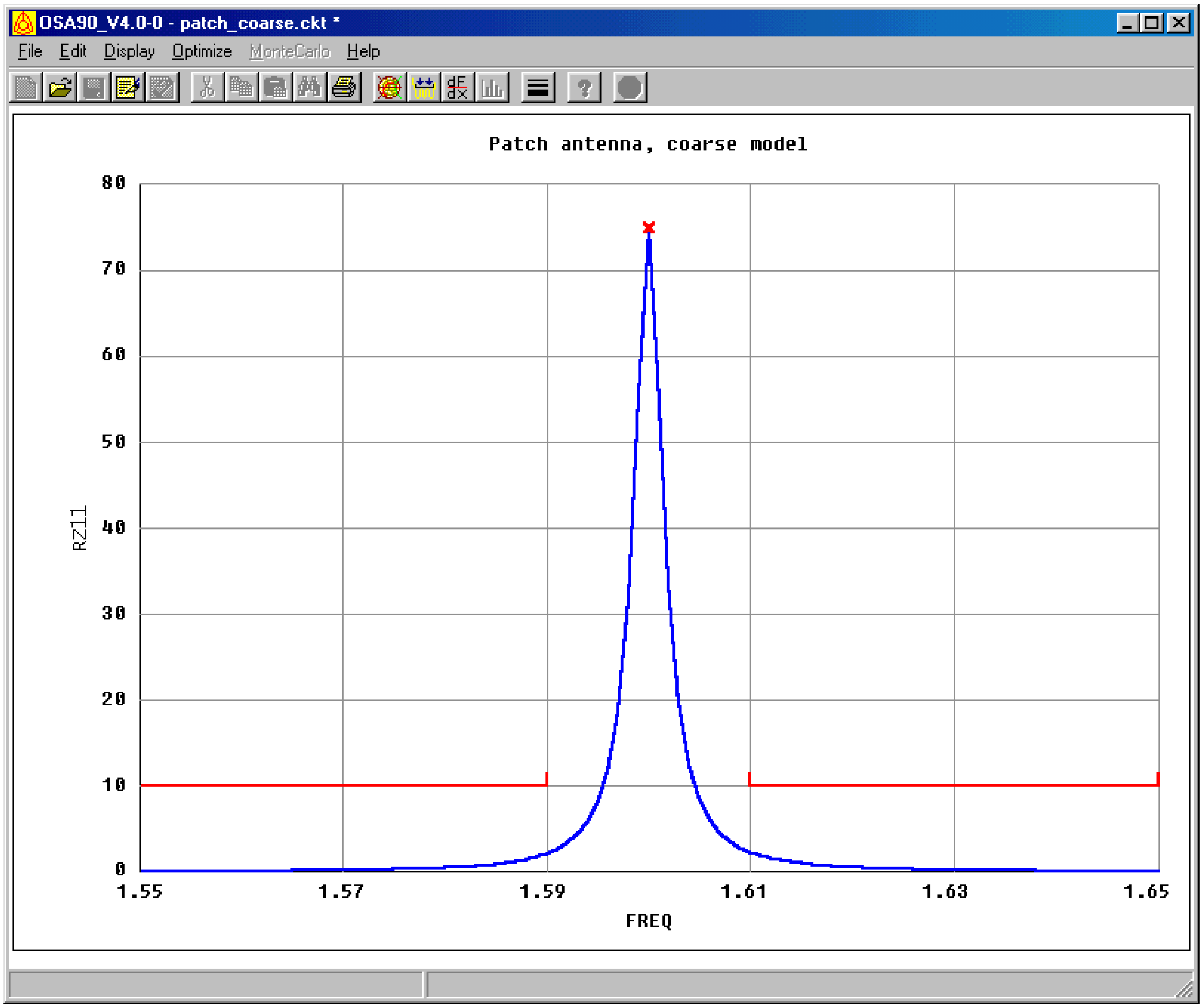
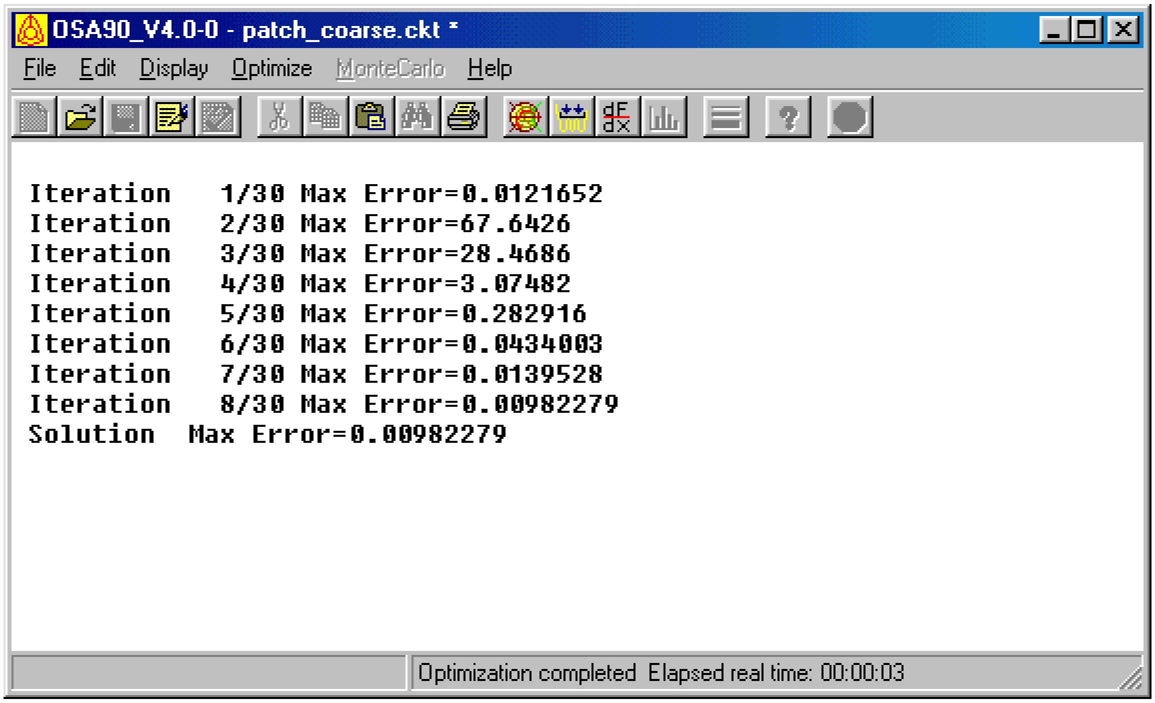


Fig. 13. The real part of the patch input impedance (coarse model) after optimization.


OSA90_V4.0-0 - patch_coarse.ckt *
File Edit Display Optimize MonteCarlo Help

```
Iteration   1/30 Max Error=0.0121652
Iteration   2/30 Max Error=67.6426
Iteration   3/30 Max Error=28.4686
Iteration   4/30 Max Error=3.07482
Iteration   5/30 Max Error=0.282916
Iteration   6/30 Max Error=0.0434003
Iteration   7/30 Max Error=0.0139528
Iteration   8/30 Max Error=0.00982279
Solution  Max Error=0.00982279
```

Optimization completed Elapsed real time: 00:00:03

Fig. 14. The minimax objective-function error after the optimization of the coarse model of the MPA is completed

References

[1] ***em***$^{\text{TM}}$, *Sonnet*$^{®}$ *User's Manual* Release 5.1, Volume 1, Sonnet Software, Inc., 1020 Seventh North Street, Suite 210, Liverpool, NY 13088, October 1997.

[2] HP Advanced Design System, Version 1.1, Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.A.

[3] J.W. Bandler, R.M. Biernacki, S.H. Chen, P.A. Grobelny and R.H. Hemmers, "Space mapping technique for electromagnetic optimization," *IEEE Trans. Microwave Theory Tech.*, vol. 42, 1994, pp. 2536-2544.

[4] M.H. Bakr, J.W. Bandler, R.M. Biernacki, S.H. Chen and K. Madsen, "A trust region aggressive space mapping algorithm for EM optimization," *IEEE Trans. Microwave Theory Tech.*, vol. 46, 1998, pp. 2412-2425.

[5] I. J. Bahl and P. Bhartia, *Microstrip Antennas*. Dedham, MA: Artech House 1980.

[6] Robert A. Sainati, *CAD of Microstrip Antennas*. Norwood, MA: Artech House, 1996.

[7] Constantine A. Balanis, *Antenna Theory*, 2nd ed. New York, NY: John Wiley & Sons, 1997.

[8] E. Hammerstad and O. Jensen, "Accurate models for microstrip computer-aided design," *IEEE MTT-S Int. Microwave Symposium Digest*, 1980, pp. 407-409.

[9] A. G. Derneryd, "A theoretical investigation of the rectangular microstrip antenna element," *IEEE Trans. on Antennas and Propagation*, vol. 26, pp. 532-535, 1978.

[10] H. Pues and A. Van de Capelle, "Accurate transmission line model for the rectangular microstrip antenna," *IEE Proc. Part H (Microwaves, Optics, and Acoustics)*, vol. 131, pp. 334-340, 1984.

[11] *OSA90/hope*$^{\text{TM}}$ Version 4.0, formerly Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, now HP EEsof Division, Hewlett-Packard Company, 1400 Fountaingrove Parkway, Santa Rosa, CA 95403-1799.

[12] M. Kirschning, R. H. Jansen and N. H. L. Koster, "Accurate model for open end effect of microstrip lines," *Elec. Letts.*, vol. 17, pp. 123-125, 1981.

[13] E. O. Hammerstad, "Equations for microstrip circuit design," *Proc. 5th European Microwave Conference*, 1975, pp. 268-272.

Appendix A

The Coarse Model of the MPA: OSA90 Netlist File

```
!  Wed May 19 14:36:14 1999. Minimax Optimizer. 5 Iterations. 00:00:00 CPU.
Model
   W= 39.99;
   L= ?29.6135?;
   yo=?12.1774?;

   er=10.2;
   h= 1.27;
   wo=0.43;

Datapipe: SIM File="./pp.out"
               N_INPUT=5 INPUT=(W, L, er, h, freq)
               N_OUTPUT=7  OUTPUT=(Zp, eref_p, Rr, Ce, magYm, angYm, delta_L);

  inset = yo + delta_L;
  Le = L – yo + delta_L;

  CAP 1 0 C=(Ce * 1nF);
  RES 1 0 R=(Rr * 1oh);
  VCCS 2 0 1 0 M=(magYm * 1/oh) A=(angYm * 1rad);
  CAP 2 0 C=(Ce * 1nF);
  RES 2 0 R=(Rr * 1oh);
  VCCS 1 0 2 0 M=(magYm * 1/oh) A=(angYm * 1rad);

  TRL 1 3  Z=(Zp * 1oh) L=(inset * 1mm) K=eref_p;
  TRL 2 3  Z=(Zp * 1oh) L=(Le * 1mm) K=eref_p;
  PORT 3 0 R=75;

CIRCUIT;

  MZ = sqrt(RZ11*RZ11 + IZ11*IZ11);

end
Sweep
  ac: freq: from 1.55GHz to 1.65Ghz step=0.01Ghz
      RZ11 IZ11 MZ
          {Xsweep Title="Patch antenna, coarse model"
           Y=RZ11.green Y_title="RZ11"
           SPEC=(from 1.55 to 1.59, < 10).red &
                (at 1.6, =75).red &
                (from 1.61 to 1.65, < 10).red};
end
Specification
  ac: freq: 1.60GHz RZ11=75 MZ=75 w=1;
  ac: freq: from 1.55GHz to 1.59GHz step=0.01GHz RZ11 <= 10;
  ac: freq: from 1.61GHz to 1.65GHz step=0.01GHz RZ11 <= 10;
end
control
  Optimizer=minimax;
end
Report
Xos_opt=[$L$
         $yo$];
end
```

Appendix B

pp.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "ippcv2.h"

double jy0();
double jy1();
double besjy();
double G1();
double G12();
double f1();
double f2();
double eps_eff();
double chr_imp();
double delta_L();

#define  NX        5   /* number of inputs */
#define  NY        7   /* number of outputs */
#define  K_FREQ    4   /* which input is the frequency */

/**********************************************************************/
/* input vector x[5]:           output vector y[7]:                   */
/*                                                                    */
/* x[0] = W, mm (patch width)   y[0] = z_p, ohm (patch TL impedance)  */
/* x[1] = L, mm (patch length)  y[1] = eref_p (patch eff. diel. cons) */
/* x[2] = er, (sub diel. const) y[2] = rr, ohm (eq. rad. resistance)  */
/* x[3] = h, mm (sub height)    y[3] = cap, nF (edge capacitance)     */
/* x[4] = freq, GHz             y[4] = ymag, S (mutual slot cond.,mag)*/
/*                              y[5] = yang, rad (mutual susc., angle)*/
/*                              y[6] = dl, mm (fringing length)       */
/**********************************************************************/
main ()
{
  float x[NX], y[NY];
  char *message;
  int nx, ny, error;
  double W, L, er, h, freq;
  double z_p, eref_p, rr, cap, ymag, yang, dl;
  double gr, s, phi, sp, sw, y0, y2, gm, bm;
  double lp, k0, lambda0;
  double vl = 2.99792458e+02, pi = 3.14159265358979323846264 3;

  for (;;)
    {
      error = 0;

      pipe_initialize2();
      pipe_read2(&nx, sizeof(int), 1);
      pipe_read2(&ny, sizeof(int), 1);
      pipe_read2(x, sizeof(float), nx);

      if (nx != NX || ny != NY)
      {
        message = "Incorrect Number of Input/Output";
        error = strlen(message) + 1;
      }
      else
      {
        W = (double)x[0];
        L = (double)x[1];
        er = (double)x[2];
        h = (double)x[3];
        freq = (double)x[4];

        eref_p = eps_eff(er,h,W,freq);
        z_p = chr_imp(er,h,W,freq,eref_p);
        lambda0 = vl / freq;
        k0 = (2.0*pi) / lambda0;
        dl = delta_L(W,h,er,eref_p);
        lp = L+dl;
        gr=G1(W,k0,0,180);
        rr = 1 / gr;
        gm=G12(W,k0,lp,0,180);

        phi = 2.0 * pi * sqrt(eref_p) * (dl/lambda0);
        cap = tan(phi) / (2.0*pi*z_p*freq);

        s = k0 * dl;
        sp = k0 * L;
        sw = k0 * W;
        phi = s * s;
        phi = phi / (24.0-phi);
        y0 = besjy(1,0,sp);
        y2 = besjy(1,2,sp);
        bm = (pi/2.0) * (y0 + (phi * y2));
        bm = bm / (log(s/2.0) + 0.577216 - 1.5 + (phi/12.0));
        bm = bm * (1.0 - exp(-0.21*sw));
        bm = bm * 2.0 * pi * freq * cap;

        ymag = sqrt(gm*gm +bm*bm);
        yang = atan2(bm,gm);

        y[0] = (float)z_p;
        y[1] = (float)eref_p;
        y[2] = (float)rr;
        y[3] = (float)cap;
        y[4] = (float)ymag;
        y[5] = (float)yang;
        y[6] = (float)dl;
      }

      pipe_write2(&error, sizeof(int), 1);
      if (!error)
      {
        pipe_write2(&y, sizeof(float), NY);
      }
      else
      pipe_write2(message, 1, error);
    }
}
/* end MAIN */
/**********************************************************************/
double delta_L(width,height,eps_r,eps_eff)
double width,height,eps_r,eps_eff;
{
  double xi1,xi2,xi3,xi4,xi5,ere,wh,whp, result;
  wh = width / height;
  ere = pow(eps_eff, 0.81);
  whp = pow(wh, 0.8544);
  xi1 = 0.434907 * (ere+0.26) / (ere-0.189);
  xi1 = xi1 * (whp+0.236) / (whp+0.87);

  whp = pow(wh, 0.371);
  xi2 = 1.0 + (whp / ((2.358*eps_r)+1.0));

  ere = pow(eps_eff, 0.9236);
  whp = 0.084 * pow(wh, (1.9413/xi2));
  xi3 = 1.0 + ((0.5274 * atan(whp)) / ere);

  ere = 0.036 * (1-eps_r);
  whp = 0.067 * pow(wh, 1.456);
  xi4 = 1.0 + (0.5274 * atan(whp) * (6.0-5.0*exp(ere)));

  xi5 = 1.0 - (0.218 * exp(-7.5*wh));

  result = xi1 * xi3 * xi5 / xi4 * height;
  return(result);
}
/* end of delta_L */
/*********************************************************************/
/* numerical integration of far field power: equivalent conductance */
double G1(width,k0,ll,ul)
double width,k0;
int ll,ul;

{
  double pi = 3.14159265358979323846264 3;
  double theta,res,xi,g1v;
  int ll1,i;

  theta=pi/180.0;
  res=0.0;
  ll1=ll+1;
  for (i = ll1; i <= ul; i++)
    {
      xi=i*pi/180.0;
      res=res+f1(xi,width,k0)*sin(xi)*theta;
    };
  g1v=res/(120.0*pi*pi);
  return(g1v);
}
/* end of G1 */
/*********************************************************************/
/* numerical integration of far field mutual power: eq. mutual conductance */
double G12(width,k0,length,ll,ul)
double width,k0,length;
int ll,ul;
{
  double pi = 3.14159265358979323846264 3;
  double theta,res,xi,g12v;
  int ll1,i;

  theta=pi/180.0;
  res=0.0;
  ll1=ll+1;
  for (i = ll1; i <= ul; i++)
    {
      xi=i*pi/180.0;
      res=res+f2(xi,width,k0,length)*sin(xi)*theta;
    }
  g12v=res/(120.0*pi*pi);
  return(g12v);
}
/* end of G12 */
/**********************************************************************/
/* radiated field of a single slot */
double f1(theta,width,k0)
double theta,width,k0;
{
  double tol=1.0e-6;
  double argu,sinc,f1v;

  argu=(k0*width/2.0)*cos(theta);
  if (fabs(argu-0.0) < tol)
    sinc=1.0;
  else
    sinc=sin(argu)/argu;

  f1v=((k0*width)/2.0)*sinc*sin(theta);
  return(f1v*f1v);
}
/* end of f1 */
/**********************************************************************/
/* integrand of mutual coupling integral */
double f2(theta,width,k0,length)
double theta,width,k0,length;
{
  double tol = 1.0e-6;
  double argu, x, sinc, j0, f2v;

  argu = (k0*width/2.0)*cos(theta);
  if (fabs(argu-0.0) < tol)
    sinc = 1.0;
  else
    sinc = sin(argu) / argu;

  f2v = ((k0*width)/2.0)*sinc*sin(theta);
  x = k0*length*sin(theta);
  j0 = besjy(0,0,x);
  f2v=f2v*f2v*j0;
  return(f2v);
}
/* end of f2 */
/**********************************************************************/
/* calculating the effective dielectric constant */
double eps_eff(er,h,wo,freq)
double er,h,wo,freq;
{
  double vl=2.99792458,pi=3.14159265358979323846264 3;
  double wh,a,b,m,ee,eef,f50;

  wh=wo/h;b=wh*wh*wh*wh;
  a=(wh/52.0)*(wh/52.0);
  b=(b+a)/(b+0.432);
  a=1.0+(1.0/49.0)*log(b)+(1.0/18.7)*log(1.0+pow((wh/18.1),3.0));

  b=(er-0.9)/(er+3.0);
  b=0.564*pow(b,0.053);
  ee=0.5*(er+1.0)+0.5*(er-1.0)*pow((1.0+(10.0/wh)),(-a*b));
/* dispersion effect */
  a=1.0/(1.0+sqrt(wh));
  a=1.0+a+0.32*pow(a,3.0);
  if (ee != er)
    {
      m=er*sqrt((ee-1.0)/(er-ee));
      m=atan(m);
      m=((50.0*vl/pi)*m)/(h*sqrt(er-ee));
      f50=m/(0.75+(0.75-(0.332/pow(er,1.73)))*wh);
      if (wh <= 0.7)
      b=1.0+(1.4/(1.+wh))*(0.15-0.235*exp(-0.45*freq/f50));
      else
      b=1.0;
      m=a*b;
      eef=er-((er-ee)/(1.+pow((freq/f50),m))); }
  else
    eef=ee;
  return(eef);
} /* end of eps_eff */
/**********************************************************************/
/* calculating the characteristic impedance */
double chr_imp(er,h,wo,freq,eref)
double er,h,wo,freq,eref;
{
  double zo,wh,a,b,m,ee,f1;
  double pi=3.141592653589793238462643;

  wh=wo/h;b=wh*wh*wh*wh;
  a=(wh/52.)*(wh/52.);
  b=(b+a)/(b+0.432);
  a=1.0+(1.0/49.0)*log(b)+(1.0/18.7)*log(1.0+pow((wh/18.1),3.0));

  b=(er-0.9)/(er+3.0);
  b=0.564*pow(b,0.053);

  ee=0.5*(er+1.0)+0.5*(er-1.0)*pow((1.0+(10./wh)),(-a*b));

  f1=6.0+(2.0*pi-6.)*exp(-pow((30.666/wh),0.7528));
  zo=(60.0/sqrt(ee))*log(f1/wh+sqrt(1.0+(2.0/wh)*(2.0/wh)));
/* dispersion */
  zo=zo*((eref-1.0)/(ee-1.0))*sqrt(ee/eref);
  return(zo);
} /* end of chr_imp */
/**********************************************************************/
/* calculates Bessel functions J or Y of integer order */
/* kind=0 => J; kind=1 => Y */
double jy0();
double jy1();

double besjy(kind,order,arg)
int kind,order;
double arg;
{
  int j;
  double bes,by,bym,byp,tox;
  double two=2.0e0;

  if (order == 0)
    {
      bes=jy0(arg,kind);
      goto end_location;
    };
  if (order == 1)
    {
      bes=jy1(arg,kind);
      goto end_location;
    };
  /* order > 1 */
  tox = two/arg;
  by = jy1(arg,kind);
  bym = jy0(arg,kind);
  for (j = 1; j <= (order-1); j++)
    {
      byp = ((double)j * tox * by) - bym;
      bym = by;
      by = byp;
    };
  bes = by;
 end_location: ;
  return(bes);
}
/* end of besjy */
/*------------------------------------------------------------------*/
double jy0(arg,mode)
double arg;
int mode;
{
int i;
double ax,down,prod,resj,result,r0,r1,up,w,wsq,xden,xnum,
       xy,z,zsq;
/*------------------------------------------------------------------*/
/*  Mathematical constants                                          */
/*    CONS = ln(.5) + Euler's gamma                                 */
/*------------------------------------------------------------------*/
double zero=0.0e0,one=1.0e0,three=3.0e0,four=4.0e0,eight=8.0e0,
       five5=5.5e0,sixty4=64.0e0,oneov8=0.125e0,p17=1.716e-1,
       two56=256.0e0,cons=-1.1593151565841244881e-1,
       pi2=6.3661977236758134308e-1,twopi=6.2831853071795864769e0,
       twopi1=6.28125e0,twopi2=1.9353071795864769253E-3;
/*-------------------------------------------------------------------*/
/*  Machine-dependent constants                                      */
/*-------------------------------------------------------------------*/
double xmax=1.07e+09,xsmall=9.31e-10,xinf=1.7e+38;
/*-------------------------------------------------------------------*/
/*  Zeroes of Bessel functions                                       */
/*-------------------------------------------------------------------*/
double xj0=2.4048255576957727686e+0,xj1=5.5200781102863106496e+0,
       xy0=8.9357696627916752158e-1,xy1=3.9576784193148578684e+0,
       xy2=7.0860510603017726976e+0,
       xj01=616.0e+0,xj02=-1.4244423042272313784e-03,
       xj11=1413.0e+0,xj12=5.4686028631064959660e-04,
       xy01=228.0e+0,xy02=2.9519662791675215849e-03,
       xy11=1013.0e+0,xy12=6.4716931485786837568e-04,
       xy21=1814.0e+0,xy22=1.1356030177269762362e-04;

double PLG[4], QLG[4], PJ0[7], QJ0[5], PJ1[8], QJ1[7], PY0[6], QY0[5],
       PY1[7], QY1[6], PY2[8], QY2[7], P0[6], Q0[5], P1[6], Q1[5];

/*-------------------------------------------------------------------*/
/*  Coefficients for rational approximation to ln(x/a)               */
```

```
/*-------------------------------------------------------------------*/
PLG[0] = -2.4562334077563243311e+01; PLG[1] = 2.3642701335621505212e+02;
PLG[2] = -5.4989956895857911039e+02; PLG[3] = 3.5687548468071500413e+02;
QLG[0] = -3.5553900764052419184e+01; QLG[1] = 1.9400230218539473193e+02;
QLG[2] = -3.3442903192607538956e+02; QLG[3] = 1.7843774234035750207e+02;
/*-------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                       */
/*  J0(X) / (X**2 - XJ0**2),  XSMALL  <  |X|  <=  4.0                */
/*-------------------------------------------------------------------*/
PJ0[0] = 6.6302997904833794242e+06; PJ0[1] =-6.2140700423540120665e+08;
PJ0[2] = 2.7282507878605942706e+10; PJ0[3] =-4.1298668500990866786e+11;
PJ0[4] =-1.2117036164593528341e-01; PJ0[5] = 1.0344222815443188943e+02;
PJ0[6] =-3.6629814655107086448e+04;
QJ0[0] = 4.5612696224219938200e+05; QJ0[1] = 1.3985097372263433271e+08;
QJ0[2] = 2.6328198300859648632e+10; QJ0[3] = 2.3883787996332290397e+12;
QJ0[4] = 9.3614022392337710626e+02;
/*-------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                       */
/*  J0(X) / (X**2 - XJ1**2),  4.0  <  |X|  <=  8.0                   */
/*-------------------------------------------------------------------*/
PJ1[0] = 4.4176707025325087628e+03; PJ1[1] = 1.1725046279757103576e+04;
PJ1[2] = 1.0341910641583726701e+04; PJ1[3] =-7.2879702464464618998e+03;
PJ1[4] =-1.2254078161378989535e+04; PJ1[5] =-1.8319397969392084011e+03;
PJ1[6] = 4.8591703355916499363e+01; PJ1[7] = 7.4321196680624245801e+02;
QJ1[0] = 3.3307310774649071172e+02; QJ1[1] =-2.9458766545509337327e+03;
QJ1[2] = 1.8680990008359188352e+04; QJ1[3] =-8.4055062591169562211e+04;
QJ1[4] = 2.4599102262586308984e+05; QJ1[5] =-3.5783478026152301072e+05;
QJ1[6] =-2.5258076240801555057e+01;
/*-------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                       */
/*  (Y0(X) - 2 LN(X/XY0) J0(X)) / (X**2 - XY0**2),                   */
/*        XSMALL  <  |X|  <=  3.0                                    */
/*-------------------------------------------------------------------*/
PY0[0] = 1.0102532948020907590e+04; PY0[1] =-2.1287548474401797963e+06;
PY0[2] = 2.0422274357376619816e+08; PY0[3] =-8.3716255451260504098e+09;
PY0[4] = 1.0723538782003176831e+11; PY0[5] =-1.8402381979244993524e+01;
QY0[0] = 6.6475986689240190091e+02; QY0[1] = 2.3889393209447253406e+05;
QY0[2] = 5.5662956624278251596e+07; QY0[3] = 8.1617187777290363573e+09;
QY0[4] = 5.8873865738997033405e+11;
/*-------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                       */
/*    (Y0(X) - 2 LN(X/XY1) J0(X)) / (X**2 - XY1**2),                 */
/*        3.0  <  |X|  <=  5.5                                       */
/*-------------------------------------------------------------------*/
PY1[0] =-1.4566865832663635920e+04; PY1[1] = 4.6905288611678631510e+06;
PY1[2] =-6.9590439394619619534e+08; PY1[3] = 4.3600098638603061642e+10;
PY1[4] =-5.5107435206722644429e+11; PY1[5] =-2.2213976967566192242e+13;
PY1[6] = 1.7427031242901594547e+01;
QY1[0] = 8.3030857612070288823e+02; QY1[1] = 4.0669982352539552018e+05;
QY1[2] = 1.3960202770986831075e+08; QY1[3] = 3.4015103849971240096e+10;
QY1[4] = 5.4266824419412347550e+12; QY1[5] = 4.3386146580707264428e+14;
/*-------------------------------------------------------------------*/
/* Coefficients for rational approximation of                        */
/*   (Y0(X) - 2 LN(X/XY2) J0(X)) / (X**2 - XY2**2),                  */
/*       5.5  <  |X|  <=  8.0                                        */
/*-------------------------------------------------------------------*/
PY2[0] = 2.1363534169313901632e+04; PY2[1] =-1.0085539923498211426e+07;
PY2[2] = 2.1958827170518100757e+09; PY2[3] =-1.9363051266772083678e+11;
PY2[4] =-1.2829912364088687306e+11; PY2[5] = 6.7016641869173237784e+14;
PY2[6] =-8.0728726905150210443e+15; PY2[7] =-1.7439661319197499338e+01;
QY2[0] = 8.7903362168128450017e+02; QY2[1] = 5.3924739209768057030e+05;
```

```
QY2[2] = 2.4727219475672302327e+08; QY2[3] = 8.6926121104209825246e+10;
QY2[4] = 2.2598377924042897629e+13; QY2[5] = 3.9272425569640309819e+15;
QY2[6] = 3.4563724628846457519e+17;
/*-------------------------------------------------------------------*/
/* Coefficients for Hart,s approximation,  |X| > 8.0                 */
/*-------------------------------------------------------------------*/
P0[0] = 3.4806486443249270347e+03; P0[1] = 2.1170523380864944322e+04;
P0[2] = 4.1345386639580765797e+04; P0[3] = 2.2779090197304684302e+04;
P0[4] = 8.8961548424210455236e-01; P0[5] = 1.5376201909008354296e+02;
Q0[0] = 3.5028735138235608207e+03; Q0[1] = 2.1215350561880115730e+04;
Q0[2] = 4.1370412495510416640e+04; Q0[3] = 2.2779090197304684318e+04;
Q0[4] = 1.5711159858080893649e+02;
P1[0] =-2.2300261666214198472e+01; P1[1] =-1.1183429920482737611e+02;
P1[2] =-1.8591953644342993800e+02; P1[3] =-8.9226600200800094098e+01;
P1[4] =-8.8033303048680751817e-03; P1[5] =-1.2441026745835638459e+00;
Q1[0] = 1.4887231232283756582e+03; Q1[1] = 7.2642780169211018836e+03;
Q1[2] = 1.1951131543434613647e+04; Q1[3] = 5.7105024128512061905e+03;
Q1[4] = 9.0593769594993125859e+01;
/*------------------------------------------------------------------*/
/* Check for error conditions                                       */
/*------------------------------------------------------------------*/
 ax = fabs(arg);
 if ((mode == 1) && (arg <= zero))
   {
     result = -xinf;
     goto end_location;
   }
 else
   {
     if (ax > xmax)
       {
       result = zero;
       goto end_location;
       };
   };
 if (ax > eight)
   goto location_800;
 if (ax <= xsmall)
   {
     if (mode == 0)
       result = one;
     else
       result = pi2 * (log(ax) + cons);
     goto end_location;
   };
/*-------------------------------------------------------------------*/
/* Calculate J0 for appropriate interval, preserving                 */
/*    accuracy near the zero of J0                                   */
/*-------------------------------------------------------------------*/
 zsq = ax * ax;
 if (ax <= four)
   {
     xnum = (PJ0[4] * zsq + PJ0[5]) * zsq + PJ0[6];
     xden = zsq + QJ0[4];
     for (i = 0; i <= 3; i++)
       {
       xnum = xnum * zsq + PJ0[i];
       xden = xden * zsq + QJ0[i];
       };
     prod = ((ax - xj01/two56) - xj02) * (ax + xj0);
   }
 else
   {
     wsq = one - (zsq / sixty4);
     xnum = PJ1[6] * wsq + PJ1[7];
     xden = wsq + QJ1[6];
     for (i = 0; i <= 5; i++)
       {
       xnum = xnum * wsq + PJ1[i];
       xden = xden * wsq + QJ1[i];
       }
     prod = (ax + xj1) * ((ax - xj11/two56) - xj12);
   };
 result = prod * xnum / xden;
 if (mode == 0)
   goto end_location;
/*-------------------------------------------------------------------*/
/* Calculate Y0.  First find  RESJ = pi/2 ln(x/xn) J0(x),            */
/*   where xn is a zero of Y0                                        */
/*-------------------------------------------------------------------*/
 if (ax <= three)
   {
     up = (ax - xy01/two56) - xy02;
     xy = xy0;
   }
 else
   {
     if (ax <= five5)
       {
       up = (ax - xy11/two56) - xy12;
       xy = xy1;
       }
     else
       {
       up = (ax - xy21/two56) - xy22;
       xy = xy2;
       };
   };
 down = ax + xy;
 if (fabs(up) < (p17*down))
   {
     w = up/down;
     wsq = w*w;
     xnum = PLG[0];
     xden = wsq + QLG[0];
     for (i=1; i <=3; i++)
       {
       xnum = xnum*wsq + PLG[i];
       xden = xden*wsq + QLG[i];
       };
     resj = pi2 * result * w * xnum/xden;
   }
 else
   {
     resj = pi2 * result * log(ax/xy);
   };
/*------------------------------------------------------------------*/
/* Now calculate Y0 for appropriate interval, preserving            */
/*    accuracy near the zero of Y0                                  */
/*------------------------------------------------------------------*/
 if (ax <= three)
   {
     xnum = PY0[5] * zsq + PY0[0];
     xden = zsq + QY0[0];
     for (i=1; i <= 4; i++)
       {
       xnum = xnum * zsq + PY0[i];
       xden = xden * zsq + QY0[i];
       }
   }
 else
   {
     if (ax <= five5)
       {
       xnum = PY1[6] * zsq + PY1[0];
       xden = zsq + QY1[0];
       for (i=1; i <= 5; i++)
         {
           xnum = xnum * zsq + PY1[i];
           xden = xden * zsq + QY1[i];
         }
       }
     else
       {
       xnum = PY2[7] * zsq + PY2[0];
       xden = zsq + QY2[0];
       for (i=1; i <= 6; i++)
         {
           xnum = xnum * zsq + PY2[i];
           xden = xden * zsq + QY2[i];
         };
       };
   };
 result = resj + up * down * xnum / xden;
 goto end_location;
/*-------------------------------------------------------------------*/
/* Calculate J0 or Y0 for |ARG|  >  8.0                              */
/*-------------------------------------------------------------------*/
 location_800: ;
 z = eight / ax;
 w = ax / twopi;
 w = floor(w) + oneov8;
 w = (ax - w * twopi1) - w * twopi2;
 zsq = z * z;
 xnum = P0[4] * zsq + P0[5];
 xden = zsq + Q0[4];
 up = P1[4] * zsq + P1[5];
 down = zsq + Q1[4];
 for (i = 0; i <= 3; i++)
   {
     xnum = xnum * zsq + P0[i];
     xden = xden * zsq + Q0[i];
     up = up * zsq + P1[i];
     down = down * zsq + Q1[i];
   };
 r0 = xnum / xden;
 r1 = up / down;
 if (mode == 0)
   result = sqrt(pi2/ax) * (r0*cos(w) - z*r1*sin(w));
 else
   result = sqrt(pi2/ax) * (r0*sin(w) + z*r1*cos(w));

 end_location: ;
 return(result);

}
/*----------end of jy0 ---------------------------------------------*/
double jy1(arg,mode)
double arg;
int mode;
{
  int i;
  double PJ0[7],PJ1[8],PLG[4],PY0[7],PY1[9],P0[6],P1[6],
         QJ0[5],QJ1[7],QLG[4],QY0[6],QY1[8],Q0[6],Q1[6];
  double ax,down,prod,resj,result,r0,r1,up,w,wsq,xden,xnum,xy,z,zsq;
/*---------------------------------------------------------------------*/
/*  Mathematical constants                                             */
/*---------------------------------------------------------------------*/
  double eight=8.0e0, four=4.0e0, half=0.5e0, throv8=0.375e0,
         pi2=6.3661977236758134308e-1, p17=1.716e-1,
         twopi=6.2831853071795864769e+0, zero=0.0e0,
         twopi1=6.28125e0, twopi2=1.9353071795864769253e-03,
         two56=256.0e+0, rtpi2=7.9788456080286535588e-1;
/*---------------------------------------------------------------------*/
/*  Machine-dependent constants                                        */
/*---------------------------------------------------------------------*/
  double xmax=1.07e+09, xsmall=9.31e-10, xinf=1.7e+38;
/*---------------------------------------------------------------------*/
/*  Zeroes of Bessel functions                                         */
/*---------------------------------------------------------------------*/
  double xj0=3.8317059702075123156e+0, xj1=7.0155866698156187535e+0,
         xy0=2.1971413260310170351e+0, xy1=5.4296810407941351328e+0,
         xj01=981.0e+0, xj02=-3.2527979248768438556e-04,
         xj11=1796.0e+0, xj12=-3.8330184381246462950e-05,
         xy01=562.0e+0, xy02=1.8288260310170351490e-03,
         xy11=1390.0e+0, xy12=-6.4592058648672279948e-06;
/*---------------------------------------------------------------------*/
/*  Coefficients for rational approximation to ln(x/a)                 */
/*---------------------------------------------------------------------*/
  PLG[0] =-2.4562334077563243311e+01; PLG[1] = 2.3642701335621505212e+02;
  PLG[2] =-5.4989956895857911039e+02; PLG[3] = 3.5687548468071500413e+02;
  QLG[0] =-3.5553900764052419184e+01; QLG[1] = 1.9400230218539473193e+02;
  QLG[2] =-3.3442903192607538956e+02; QLG[3] = 1.7843774234035750207e+02;
/*---------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                         */
/*  J1(X) / (X * (X**2 - XJ0**2)),  XSMALL  <  |X|  <=  4.0            */
/*---------------------------------------------------------------------*/
  PJ0[0] = 9.8062904098958257677e+05; PJ0[1] =-1.1548696764841276794e+08;
  PJ0[2] = 6.6781041261492395835e+09; PJ0[3] =-1.4258509801366645672e+11;
  PJ0[4] =-4.4615792982775076130e+03; PJ0[5] = 1.0650724020080236441e+01;
  PJ0[6] =-1.0767857011487300348e-02;
  QJ0[0] = 5.9117614494174794095e+05; QJ0[1] = 2.0228375140097033958e+08;
  QJ0[2] = 4.2091902282580133541e+10; QJ0[3] = 4.1868604460820175290e+12;
  QJ0[4] = 1.0742272239517380498e+03;
/*---------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                         */
/*  J1(X) / (X * (X**2 - XJ1**2)),  4.0  <  |X|  <=  8.0               */
/*---------------------------------------------------------------------*/
  PJ1[0] = 4.6179191852758252280e+00; PJ1[1] =-7.1329006872560947377e+03;
  PJ1[2] = 4.5039658105749078904e+06; PJ1[3] =-1.4437717718363239107e+09;
  PJ1[4] = 2.3569285397217157313e+11; PJ1[5] =-1.6324168293282543629e+13;
  PJ1[6] = 1.1357022719979468624e+14; PJ1[7] = 1.0051899717115285432e+15;
  QJ1[0] = 1.1267125065029138050e+06; QJ1[1] = 6.4872502899596389593e+08;
  QJ1[2] = 2.7622777286244082666e+11; QJ1[3] = 8.4899346165481429307e+13;
  QJ1[4] = 1.7128800897135812012e+16; QJ1[5] = 1.7253905888447681194e+18;
  QJ1[6] = 1.3886978985861357615e+03;
/*---------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                         */
/*    (Y1(X) - 2 LN(X/XY0) J1(X)) / (X**2 - XY0**2),                   */
/*        XSMALL  <  |X|  <=  4.0                                      */
/*---------------------------------------------------------------------*/
  PY0[0] = 2.2157953222280260820e+05; PY0[1] =-5.9157479997408395984e+07;
  PY0[2] = 7.2144548214502560419e+09; PY0[3] =-3.7595974497819597599e+11;
  PY0[4] = 5.4708611716525426053e+12; PY0[5] = 4.0535726612579544093e+13;
  PY0[6] =-3.1714424660046133456e+02;
  QY0[0] = 8.2079908168393867438e+02; QY0[1] = 3.8136470753052572164e+05;
  QY0[2] = 1.2250435122182963220e+08; QY0[3] = 2.7800352738690585613e+10;
  QY0[4] = 4.1272286200406461981e+12; QY0[5] = 3.0737873921079286084e+14;
/*---------------------------------------------------------------------*/
/*  Coefficients for rational approximation of                         */
/*    (Y1(X) - 2 LN(X/XY1) J1(X)) / (X**2 - XY1**2),                   */
/*        4.0  <  |X|  <=  8.0                                         */
/*---------------------------------------------------------------------*/
  PY1[0] = 1.9153806858264202986e+06; PY1[1] =-1.1957961912070617006e+09;
  PY1[2] = 3.7453673962438488783e+11; PY1[3] =-5.9530713129741981618e+13;
  PY1[4] = 4.0686275289804744814e+15; PY1[5] =-2.3638408497043134724e+16;
  PY1[6] =-5.6808094574724204577e+18; PY1[7] = 1.1514276357909013326e+19;
  PY1[8] =-1.2337180442012953128e+03;
  QY1[0] = 1.2855164849321609336e+03; QY1[1] = 1.0453748201934079734e+06;
  QY1[2] = 6.3550318087088919566e+08; QY1[3] = 3.0221766852960403645e+11;
  QY1[4] = 1.1187010065856971027e+14; QY1[5] = 3.0837179548112881950e+16;
  QY1[6] = 5.6968198822857178911e+18; QY1[7] = 5.3321844313316185697e+20;
/*---------------------------------------------------------------------*/
/*  Coefficients for Hart,s approximation,  |X| > 8.0                  */
/*---------------------------------------------------------------------*/
  P0[0] =-1.0982405543459346727e+05; P0[1] =-1.5235293511811373833e+06;
  P0[2] =-6.6033732483649391093e+06; P0[3] =-9.9422465050776411957e+06;
  P0[4] =-4.4357578167941278571e+06; P0[5] =-1.6116166443246101165e+03;
  Q0[0] =-1.0726385991103820119e+05; Q0[1] =-1.5118095066341608816e+06;
  Q0[2] =-6.5853394797230870728e+06; Q0[3] =-9.9341243899345856590e+06;
  Q0[4] =-4.4357578167941278568e+06; Q0[5] =-1.4550094401904961825e+03;
  P1[0] = 1.7063754290207680021e+03; P1[1] = 1.8494262873223866797e+04;
  P1[2] = 6.6178836581270835179e+04; P1[3] = 8.5145160675335701966e+04;
  P1[4] = 3.3220913409857223519e+04; P1[5] = 3.5265133846636032186e+01;
  Q1[0] = 3.7890229745772202641e+04; Q1[1] = 4.0029443582266975117e+05;
  Q1[2] = 1.4194606696037208929e+06; Q1[3] = 1.8194580422439972989e+06;
  Q1[4] = 7.0871281941028743574e+05; Q1[5] = 8.6383677696049909675e+02;
/*-------------------------------------------------------------------*/
/*  Check for error conditions                                       */
/*-------------------------------------------------------------------*/
  ax = fabs(arg);
  if ((mode == 1) && ((arg <= zero) || ((arg < half) && (ax*xinf < pi2))))
    {
      result = -xinf;
      goto end_location;
    }
  else
    {
      if (ax > xmax)
      {
        result = zero;
        goto end_location;
      };
    };
  if (ax > eight)
```

```
    goto location_800;
  else
    {
      if (ax <= xsmall)
      {
        if (mode == 0)
          result = arg * half;
        else
          result = -pi2 / ax;
        goto end_location;
      };
    };
/*------------------------------------------------------------------*/
/*  Calculate J1 for appropriate interval, preserving               */
/*     accuracy near the zero of J1                                 */
/*------------------------------------------------------------------*/
  zsq = ax * ax;
  if (ax <= four)
    {
      xnum = (PJ0[6] * zsq + PJ0[5]) * zsq + PJ0[4];
      xden = zsq + QJ0[4];
      for (i = 0; i <= 3; i++)
      {
        xnum = xnum * zsq + PJ0[i];
        xden = xden * zsq + QJ0[i];
      };
      prod = arg * ((ax - xj01/two56) - xj02) * (ax + xj0);
    }
  else
    {
      xnum = PJ1[0];
      xden = (zsq + QJ1[6]) * zsq + QJ1[0];
      for (i = 1; i <= 5; i++)
      {
        xnum = xnum * zsq + PJ1[i];
        xden = xden * zsq + QJ1[i];
      };
      xnum = xnum * (ax - eight) * (ax + eight) + PJ1[6];
      xnum = xnum * (ax - four) * (ax + four) + PJ1[7];
      prod = arg * ((ax - xj11/two56) - xj12) * (ax + xj1);
    };
  result = prod * (xnum / xden);
  if (mode == 0)
    goto end_location;
/*------------------------------------------------------------------*/
/*  Calculate Y1.  First find  RESJ = pi/2 ln(x/xn) J1(x),           */
/*    where xn is a zero of Y1                                      */
/*------------------------------------------------------------------*/
  if (ax <= four)
    {
      up = (ax-xy01/two56)-xy02;
      xy = xy0;
    }
  else
    {
      up = (ax-xy11/two56)-xy12;
      xy = xy1;
    };
  down = ax + xy;
  if (fabs(up) < (p17*down))
    {
      w = up/down;
      wsq = w*w;
      xnum = PLG[0];
      xden = wsq + QLG[0];
      for (i = 1; i <= 3; i++)
      {
        xnum = xnum * wsq + PLG[i];
        xden = xden * wsq + QLG[i];
      };
      resj = pi2 * result * w * xnum/xden;
    }
  else
    resj = pi2 * result * log(ax/xy);
/*------------------------------------------------------------------*/
/*  Now calculate Y1 for appropriate interval, preserving           */
/*     accuracy near the zero of Y1                                 */
/*------------------------------------------------------------------*/
  if (ax <= four)
    {
      xnum = PY0[6] * zsq + PY0[0];
      xden = zsq + QY0[0];
      for (i = 1; i <= 5; i++)
      {
        xnum = xnum * zsq + PY0[i];
        xden = xden * zsq + QY0[i];
      };
    }
  else
    {
      xnum = PY1[8] * zsq + PY1[0];
      xden = zsq + QY1[0];
      for (i = 1; i <= 7; i++)
      {
        xnum = xnum * zsq + PY1[i];
        xden = xden * zsq + QY1[i];
      };
    };
  result = resj + (up*down/ax) * xnum / xden;
  goto end_location;
/*------------------------------------------------------------------*/
/*  Calculate J1 or Y1 for |ARG|  >  8.0                            */
/*------------------------------------------------------------------*/
 location_800: ;
  z = eight / ax;
  w = ax / twopi;
  w = floor(w) + throv8;
  w = (ax - w * twopi1) - w * twopi2;
  zsq = z * z;
  xnum = P0[5];
  xden = zsq + Q0[5];
  up = P1[5];
  down = zsq + Q1[5];
  for (i = 0; i <= 4; i++)
    {
      xnum = xnum * zsq + P0[i];
      xden = xden * zsq + Q0[i];
      up = up * zsq + P1[i];
      down = down * zsq + Q1[i];
    };
  r0 = xnum / xden;
  r1 = up / down;
  if (mode == 0)
    result = (rtpi2/sqrt(ax)) * (r0*cos(w) - z*r1*sin(w));
  else
    result = (rtpi2/sqrt(ax)) * (r0*sin(w) + z*r1*cos(w));
  if ((mode == 0) && (arg < zero))
    result = -result;

 end_location: ;
  return(result);
}
/*----------end of jy1 ---------------------------------------------*/
/*----------end of pp.c --------------------------------------------*/
```