CIOPT : A PROGRAM FOR OPTIMIZATION OF THE FREQUENCY RESPONSE OF LINEAR CIRCUITS

J.M. MIRO SANS and P. PALA SCHONWALDER

Department of Signal Theory and Communications
E. Enginyers Telecomunicació. P.O. Box 30.002 08080 Barcelona SPAIN

ABSTRACT

An interactive personal-computer program to optimize the frequency response of linear lumped circuits (CiOpt) is presented.

CiOpt has proved to be an efficient tool in improving designs where the inclusion of more accurate device models distorts the desired frequency response and also in device modelling. The output of CiOpt are the element values which best match the obtained and the desired frequency response.

The optimization algorithms used (Fletcher-Powell and Newton's method, which, respectively, make use of approximate and exact second order sensitivities) are fully described in DFD form.

Analysis is carried out by obtaining the symbolic form of the transfer function $H(s)$.

This paper includes an application example.

INTRODUCTION

Optimization has proven to be an important stage in circuit design. Application fields where optimization techniques are used are nominal circuit design, design centering, worst-case design, device modelling, fault analysis, post production tuning...

In this paper we describe a computer program (CiOpt), developed to optimize the frequency response of linear lumped circuits, that is able to effectively help the designer in nominal circuit design and device modelling. Starting with a suitable circuit structure and initial element values, CiOpt computes the values of the elements chosen to be trimmed that best match the obtained and the desired frequency response.

DESCRIPTION

A graphical interface allows the user to enter the circuit description (Fig. 4). Circuits may contain all types of circuit elements (R, L, C, controlled sources). Ideal and finite GB OpAmp models along with different macromodels for BJT and FET are also available.

As a part of the description process, the user may select appropriate frequency and impedance scaling values in order to minimize the inherent errors of any numerical procedure.

After this steps, a file containing a symbolic form of the equations resulting from the Modified Nodal Approach [1] is generated.

After the circuit is described, the symbolic transfer function $H(s)$ is obtained by interpolation on the unity circle [6].

As a first step in the optimization process, the user may choose to get a plot of the magnitude of the frequency response over a specified range of frequencies. At this point, markers allow graphical input of samples of the desired response -along with their relative weights (Fig. 5). This data will be used to compute the error function defined as

$$E = \frac{1}{2} \sum_{i=1}^{Ns} q_i \left(A_i - |H(j\omega_i)| \right)^2 \qquad (1)$$

where q_i is the weight of sample i, A_i is the desired and $|H(j\omega_i)|$ the real value of the gain at the frequency ω_i. Ns is the number of samples.

Before starting optimization the designer may select up to 16 elements to be trimmed simultaniously.

CiOpt uses the following convergence criterion:

$$\left| \frac{\Delta x_k}{x_k} \right| < E_v, \quad \left| \frac{\partial E(\mathbf{x})}{\partial x_k} \right| < E_g, \; k = 1, 2, \ldots, n$$

where n is the number of elements beingtrimmed. The parameters E_g and E_v may also be changed by the user. Default values are 10^{-4}.

Once the optimization procedure is started, at each optimization step a window displays information about objective function value, element values and first-order derivatives.

To increase flexibility, the values of the variable elements may be restarted or modified manually at any moment. It is also possible (and often necessary) to modify the objective function after a number of iterations, adding / deleting samples or changing their relative weight.

After convergence is achieved -or optimization is stopped manually- the new transfer function is obtained and a graphical superposition of the previous and the optimized response is showed, allowing to view the achieved improvement. It is also always possible to view the present element values.

OPTIMIZATION PROCEDURES

Since using the transpose system method [6] it is not too expensive to find exact first and second derivatives, we can use first and second-order multidimensional optimization algorithms [4] without dramatically increasing the computational cost.

In CiOpt, we have implemented different optimization routines. A Quasi-Newton algorithm (Fletcher-Powell) and a modification of Newton's method.

Linear search

It is well-known that almost all multidimensional optimization algorithms have to face the one-dimensional minimizing problem. Stated more precisely, our problem is to find the minimum of a multidimensional function, when moving along a search direction $\mathbf{s}$ in the variable space.

At each iteration step we will know, at least, the objective function value and the gradient $\mathbf{g}$. This means that at each point and for a given search direction $\mathbf{s}$ we have information about the function value (FO) and its slope: the directional derivative (DDO) computed as $DDO = \mathbf{s}^T \mathbf{g}$.

As we will see next, there are multidimensional optimization methods that only provide information on the direction of the minimum. In this case it is necessary to have a method to find the magnitude of the first step to be taken in that search direction. Ciopt implements the following idea [2]

$$\hat{\lambda} = \frac{-0.2\,E(0)}{\mathbf{s}^T \mathbf{g}}$$ for the first step and

$$\hat{\lambda} = \frac{-2\,(E_{i-1}-E_i)}{\mathbf{s}^T\,\mathbf{g}} \qquad (2)$$

for successive iterations.

Once a first step λ_1 is taken, we compute the new values of objective function (*F*1), gradient and directional derivative (*DD*1). With this new information, we are able to decide (fig. 1) if the step was [2]:

* Good enough : if the objective function has decreased and the magnitude of the slope has decreased by a factor greater than 10.

* Too short. In this case, we linearly extrapolate to zero the slope of the function -assuming implicitly quadratic behaviour. The increase beyond the step just taken, $\hat{\lambda}-\lambda_1$ is:

$$\hat{\lambda} - \lambda_1 = \lambda_1\, z \quad , \quad \text{where} \quad z = \frac{DD1}{DD0-DD1} \qquad (3)$$

This extrapolation factor is limited to 4 to avoid excessively long steps.

* Too long. This means that the minimum lies between the two points. As we have four data items (*F*0, *DD*0, *F*1 and *DD*1) we are able to fit a cubic function and find analytically the minimum between the data points. The cubic interpolation step can be shown to be:

$$z = 1 - \frac{DD1 + w - v}{2w + DD1-DD0} \qquad (4)$$

where

$$v = 3\,\frac{F0-F1}{\hat{\lambda}_1} + DD1 + DD0 \quad \text{and} \quad w = \sqrt{v^2 - DD0\cdot DD1}$$

This algorithm is described in DFD form in fig. 1.

Please note that the description of all algorithms has been done to allow easy understanding of the substantial ideas. The real coding has been done slightly different in order to minimize computation time.

The Fletcher-Powell algorithm

This algorithm belongs to the class of quasi-Newton methods. It is known [4] that for quadratic functions

$$f(\mathbf{x}) = a + \mathbf{g}^T\mathbf{x} + 0.5\mathbf{x}^T\mathbf{H}\,\mathbf{x} \qquad (5)$$

the step **s** that takes us to the global minimum when starting at an arbitrary point is obtained solving:

$$\mathbf{Hs} = -\mathbf{g} \qquad (6)$$

This expression turns out to be also a good approximation of any function if we are in the neighbourhood of a minimum. In this case **H** is the -positive definite- Hessian matrix.

There are several methods which avoid the explicit computation of second order derivatives (necessary to build the Hessian matrix) and the resolution of (6). The idea behind this methods is to generate approximations of $\mathbf{H}^{-1}$ based on the knowledge of the successive gradient vectors. The Fletcher-Powell update formula is [4]:

$$\mathbf{H}_{i+1}^{-1} = \mathbf{H}_i^{-1} + \lambda_i^{*}\,\frac{\mathbf{s}_i\,\mathbf{s}_i^T}{\mathbf{g}_i^T\,\mathbf{H}_i^{-1}\mathbf{g}_i} - \frac{\mathbf{H}_i^{-1}\mathbf{q}_i\mathbf{q}_i^T\,\mathbf{H}_i^{-1}}{\mathbf{q}_i^T\,\mathbf{H}_i^{-1}\mathbf{q}_i} \qquad (7)$$

where $\mathbf{q}_i = \mathbf{g}_i - \mathbf{g}_{i-1}$.

The complete algorithm is described in figure 2. Initially we start with $\mathbf{H}^{-1} = \mathbf{I}$. It follows that the first search direction used is steepest descent. All subsequent directions are tested to be descent directions. If not, the whole procedure is restarted by taking $\mathbf{H}^{-1} = \mathbf{I}$. Note that *DeltaF* is computed to obtain the first step for the linear search -according to (2)- since the computed vector **s** is used as a search direction, instead of being used as the real step.

Tests performed on Rosenbrock's banana function show that restarting $\mathbf{H}^{-1} = \mathbf{I}$ every 3*n* iterations, where *n* is the number of parameters that are optimized simultaniously, improves the overall performance.

Modified Newton method (Levenberg-Marquard)

When optimizing the frequency response of linear circuits, it is possible to evaluate exact second order sensitivities by using the transpose system method without dramatically increasing the computation time [5], [6]. We have implemented a procedure (HEval), which returns objective function value, gradient vector and Hessian matrix for a given parameter value.

With this exact values of first and second order sensitivities, the step to be taken **s** is computed -following the idea of the Levenberg-Marquard method- solving

$$(\mathbf{H} + v\mathbf{I})\ \mathbf{s} = -\mathbf{g} \qquad (8)$$

where *v* is a positive real parameter. Clearly if *v*=0, equation (8) reduces to Newton's method and taking *v*>>1 we obtain an infinitesimal steepest descent step. When selecting an itermediate value, we obtain an interpolating vector between these extreme steps [3].

This method avoids the problems related to non positive definite Hessian matrix. In such cases, the search directions obtained by applying Newton's method may be ascent directions. When this happens, it is necessary to compute a new descent direction (the easiest one is steepest descent). If this situation appears repeatedly, the whole procedure converges as slow as a simple steepest descent strategy. By selecting an appropriate value of *v* we can force the eigenvalues of (**H** + *v***I**) to be positive, avoiding the outlined difficulties.

In our implementation, the directional derivative of the objective function along a computed direction s is computed. If it turns to be positive, the value of *v* is repeatedly multiplied by *DeltaV* until a descent direction is obtained from (8). In this way a descent direction which is nearest to the -theoretically best- Newton step is obtained. At each iteration where (8) already gives a descent direction, *v* is divided by the factor *DeltaV*. As we get closer to the minimum, where the Hessian becomes positive definite, the value of *v* tends to zero as it is repeatedly divided by *DeltaV*, and applying (8) the optimum Newton step is taken.

PROGRAM HIGHLIGHTS

CiOpt has been written in Turbo-Pascal (V5.0), and runs on any IBM-PC or compatible with CGA, EGA or VGA graphics card. A Math coprocessor may optionally be used to improve accuracy and speed.

Program features are :

- User-friendly interface. (Graphical inputs, pull-down menus)
- All circuit elements are allowed. Up to 64 circuit elements may be entered.
- Ideal and finite-GB OpAmp-models along with different macromodels for BJT and FET are also available.
- Maximum circuit order is 20.
- 16 elements may be trimmed simultaniously.
- Continuous information (including element values and gradient) during progress.

- High resolution plots, allowing graphical superposition.
- 32 objective function values may be entered by positioning markers on the plots.
- Symbolic form of the transfer function H(s) is obtained.
- Flexible optimization (two optimization algorithms, user-definable convergence criterion, modifiable objective function, element values may be changed manually), with trial-and-error features.

EXAMPLE

An active low-pass fifth order Chebychew filter with 40 dB gain, 0.5 dB ripple and 80 KHz cutoff frequency is designed using well-known methods. Simulation including the finite GB OpAmp model (GB=1MHz, Ao=100 dB) shows very important distortion in the passband (Fig. 5).

Selecting four elements to be trimmed, CiOpt reconstructed the desired response in 14 iterations (Fig. 6). This took 3 minutes on a 80286 machine with coprocessor.

A comparison between the implemented Levenberg-Marquard method and classical Newton's method is shown in figure 7. The evolution of Fletcher-Powell's method is also shown.

CONCLUSION

An efficient program for optimization of the frequency response of linear lumped circuits (CiOpt) has been described. Special emphasis has been put in the description of the implemented algorithms. An example has showed one of the application fields of CiOpt. A comparison between algorithms has showed that the classical Newton method may converge very slowly while the Levenberg-Marquard strategy yields much better performance.

REFERENCES

[1] CHUNG-WEN HO, RUEHLI, A.E., BRENNAN, P.A. "The Modified Nodal Approach to Network Analysis". IEEE Trans. Circ. Sist., Vol. CAS-22, No.6, June 1975

[2] CUTHBERT, T.R. "Circuit Design Using Personal Computers". Wiley-Interscience, 1983

[3] CUTHBERT, D.R. "Optimization Using Personal Computers". Wiley-Interscience, 1987.

[4] RAO, S.S. "Optimization : Theory and Applications. Wiley-Eastern, 1978

[5] REDON, T., VASILESCU, G. "Second- and Third-Order Sensitivities of Microwave Circuits". Electronics Letters, VOL 25, pp 607-609, 1989.

[6] VLACH, J., SINGHAL, K. "Computer Methods for Circuit Analysis and Design. Van Nostrand, 1983

```
OPTIMIZATION ALGORITHMS
 \
  MOD
  |\
  | # Procedure Eval returns function value and gradient
  | # Procedure HEval returns objective function value
  |   its gradient and exact Hessian
  PRC
   \
    LinearSearch
    Fletcher_Powell
    Newton_L-M


LinearSearch
|\
| RES---------------------PAR
|  \                       \
|   # Best step : Lambda    # Initial function value : F0
|   # Minimum function      # Initial slope          : DD0
|     value      : F1       # Search direction       : s
|   # DecrF,Decrs :flags    # Initial step           : Lambda
ALG                         # Element values         : vo
 \
  LOC
  |\
  | # TestedPoint : Record to keep info on tested points
  | # Lambda0     : Auxiliar variable
  | # p           : Iteration counter
  |
  # Initialization of local variables
  *
  |\
  | # Increase iteration number               : p:=p+1
  | # Evaluate increment in elements     : DeltaX :=Lambda s
  | # Evaluate new element values             : vo:=vo+DeltaX
  | # Function & gradient for new values : F1,g1 := Eval(vo)
  | # Save Lambda and F1 in TestedPoint[p]
  | # Compute Directional Derivative          : DD1:=s g1
  | |
  | # Function value has decreased ?
  | |\
  | | DecrF := F1 < F0
  | |
  | # Has the slope magnitude decreased by a factor of 10 ?
  | |\
  | | DecrS := abs( DD1 / DD0 ) < 0.1
  | |
  | ( # Stay in loop if progress is not good enough and
  |   # the number of iterations is less than the maximum
  | |\
  | | not(DecrF and DecrS) and (p<Max_Iteration)
  | ?
  | |\
  | | o-------------------------------o
  | | ( # Extrapolation is needed ?    ( Else
  | | |\                               # Cubic interpolation
  | | | DecrF and not DecrS and DD1<0   \
  | | |                                  # Restore
  | | # Compute z according to (3)         element values
  | | # Continue from this point         # Compute z (4)
  | |  \
  | |   F0,DD0  := F1,DD1
  | |   Lambda0 := Lambda0 + Lambda
  | |
  | Lambda := Lambda • z
  |
  Lambda:=Lambda0 + Lambda
  ?
   \
    o-----------------------------------------------o
    ( # if loop was exited without good progress    (else
     \                                               nul
      # Search in TestedPoint and return best value
      # of Lambda and corresponding function value
```

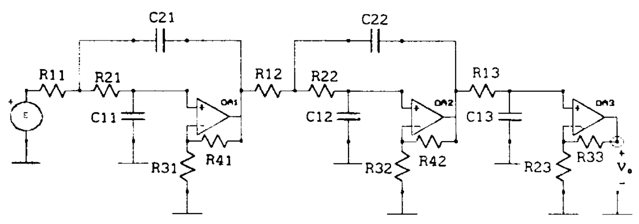
Fig. 1 Linear Search Strategy

```
Fletcher_Powell
|\
| RES----------------------------PAR
|  \                               \
ALG # Element values : vo       # Problem description
 \
  LOC
  |\
  | # Approximation of $H^{-1}$ : H_
  | # Numerator of (2) : DeltaF
  | # Iteration counter : nIter
  |
  F0,go := Eval(vo)
  H_ := I
  DeltaF := 0.1*F0
  nIter :=0
  *
   \
    # Increase iteration counter      : nIter:=nIter+1
    # Compute search direction        : s := -H_go
    # Compute Directional Derivative  : DDO := $s^T$go
    ?
    |\
    | o------------------------------o
    | ( # Function decreases        ( Else
    |  |\                           # Restart
    |  | DDO < 0                     \
    |  |                              H_:=I
    |  # Find best step along s       DeltaF:=0.1*F0
    |  |\
    |  | Lambda:=-2*DeltaF/DDO
    |  | Lambda,F1:=LinearSearch(F0,DDO,s,Lambda)
    |  |
    |  # New values : vo = vo + Lambda s
    |  # Compute     : DeltaF:=F0-F1
    |  # Update H_ according to (7)
    |
    # Perform convergence test
    ( # Stay in loop while not Converged
    # Every 3n iterations restart :
     \
      H_ := I
      DeltaF := 0.1*F0
```

Fig. 2 Fletcher-Powell Algorithm

```
Newton_L-M
|\
| RES-------------------------PAR
|  \                            \
|   # Element values : vo       # Problem
ALG                               description
 \                              # v,deltaV
  LOC
  |\
  | # Iteration counter : nIter
  | # Search direction  : s
  | # Best step         : Lambda
  |
  nIter:=0
  *
   \
    # Increase counter : nIter:=nIter + 1
    F0,go,H := HEval(vo)
    # Perform convergence test
    ( # Stay in loop while not converged
    # Compute H := H + v I
    # Solve for s : H s = -go
    # Directional Derivative : DDO := $s^T$go
    ?
    |\
    | o-------------------------o
    | ( # Increase along s      ( Else
    |  |\                        v:=v/deltaV
    |  | DDO > 0
    |  *
    |   \
    |    H:=H+(deltaV-1)I
    |    v:=v*deltaV
    |    # Solve for s : H s = -go
    |    # Compute : DDO := $s^T$go
    |    ( DDO > 0
    |
    Lambda:=LineSearch(F0,DDO,s,1)
    vo:=vo + Lambda s
```

Fig. 3 Levenberg-Marquardt Algorithm



R11 = 38.883 KΩ	R12 = 25.600 KΩ	R13 = 45.756 KΩ
R21 = 1.742 KΩ	R22 = 5.749 KΩ	R23 = 61.009 KΩ
R31 = 50.782 KΩ	R32 = 39.187 KΩ	R33 = 183.027 KΩ
R41 = 203.128 KΩ	R42 = 156.748 KΩ	C13 = 120 pF
C11 = 470 pF	C12 = 470 pF	
C21 = 120 pF	C22 = 120 pF	

Fig. 4 Fifth-Order Lowpass Filter. Initial element values

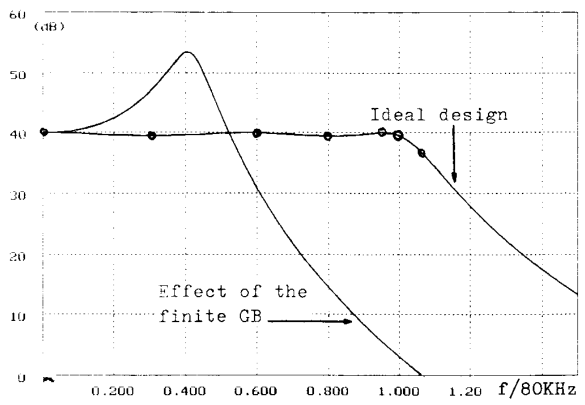


Fig. 5 Finite GB OpAmps distort response. Markers are placed on the desired response. All weights are set to 1.

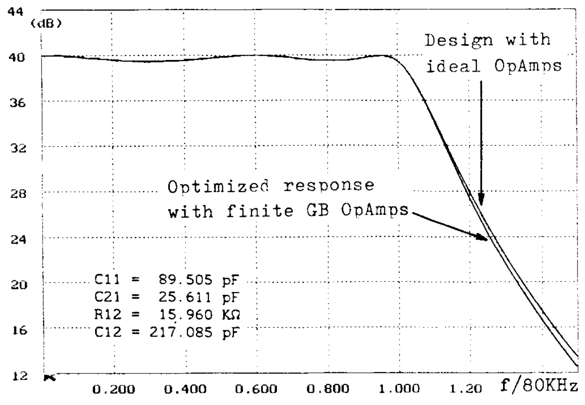


Fig. 6 After 14 iterations the response has been improved allowing 4 elements to change

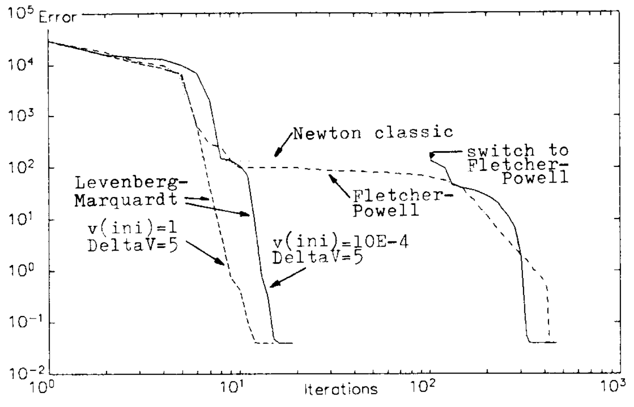


Fig. 7 A comparison between methods