Proceedings of the 5th WSEAS/IASME Int. Conf. on SYSTEMS THEORY and SCIENTIFIC COMPUTATION, Malta, September 15-17, 2005 (pp220-225)

Advanced Algorithms for Analysis and Optimization of Electronic Circuits

JOSEF DOBEŠ
Dept. of Radio Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague
Technická 2, 16627 Praha 6
Czech Republic

dobes@feld.cvut.cz   http : //www.cvut.cz

Abstract: The analysis and optimization of electronic circuits are some of the main applications for the art of computer programming. It is mainly caused by problematic convergence of a certain class of the circuits and by extreme differences among the values of the circuit parameters. In the paper, a very flexible algorithm for analyzing the electronic circuits is described with a novel reliable method for suppressing the divergence. Moreover, an efficient algorithm for optimizing the electronic circuits is also characterized with a new method for ensuring the numerical stability using a reliable normalization of the system matrix. The new features of the analysis and optimization algorithms are tested using practical tasks of electronics.

Key-Words: Modeling, circuit analysis and optimization, robust algorithms, convergence, numerical instability.

1 Introduction

For analyzing the electronic circuits in the static domain, the Newton method is mostly used in the programs of the PSpice type. Similarly, the Gear method is mostly used for analyzing the circuits in the time domain. However, the Gear method needs the Newton one for solving the circuit equations in one time step. Therefore, the convergence properties of the algorithm for solving the nonlinear algebraic equations have fundamental influence to the robustness of the program. In the following section, a more flexible algorithm for solving the system of algebraic-differential equations is defined with a very reliable method for suppressing possible divergence.

For optimizing the electronic circuits, the Levenberg-Marquardt algorithm is frequently used. However, this algorithm should be modified due to potential numerical instability that is caused by enormous differences among the magnitudes of the circuit parameters. That instability is emphasized by multiplying the Jacobian matrices in the Levenberg-Marquardt algorithm. In the third section, a normalization procedure is described, which is used for the vector in the right side and for the matrix product in the left side of the method equation. The normalized Levenberg-Marquardt method is characterized by the enhanced robustness.

2 The Algorithm for the Analysis

The system of algebraic-differential equations of a circuit is generally defined in an implicit form

$$\boldsymbol{f}\left(\boldsymbol{x}(t), \dot{\boldsymbol{x}}(t), t\right) = \boldsymbol{0}. \qquad (1)$$

Let us assume now that the first n steps of a numerical integration of (1) have finished. To make equations simpler, let us mark $\boldsymbol{x}(t_n)$ by $\boldsymbol{x}_n$, and define backward scaled differences by the recurrent formulae

$$\begin{aligned} &\delta^{(0)}\boldsymbol{x}_n = \boldsymbol{x}_n,\\ &\delta^{(i)}\boldsymbol{x}_n = \delta^{(i-1)}\boldsymbol{x}_n - \alpha_n^{(i-1)}\delta^{(i-1)}\boldsymbol{x}_{n-1},\\ &\qquad\qquad i = 1, \dots, k_n + 2, \end{aligned} \qquad (2)$$

where k_n is the order of a polynomial interpolation used in the last integration step and

$$\begin{aligned} &\alpha_n^{(0)} = 1,\\ &\alpha_n^{(i)} = \alpha_n^{(i-1)} \frac{t_n - t_{n-i}}{t_{n-1} - t_{n-1-i}},\ i = 1, \dots, k_n + 1. \end{aligned} \qquad (3)$$

The *predictor* of the variables for the next chosen time (i.e., for t_{n+1}) marked by $\boldsymbol{x}_{n+1}^{(0)}$ is determined by the extrapolation using the backward scaled differences (2) in the *explicit* form [1]

$$\boldsymbol{x}_{n+1}^{(0)} = \sum_{i=0}^{k_{n+1}} \alpha_{n+1}^{(i)} \delta^{(i)} \boldsymbol{x}_n. \qquad (4)$$

By differentiating (4) with respect to t_{n+1}, the predictor of derivatives with respect to time can also be expressed

$$\dot{\boldsymbol{x}}^{(0)}_{n+1} = \sum_{i=0}^{k_{n+1}} \beta^{(i)}_{n+1} \delta^{(i)} \boldsymbol{x}_n, \tag{5}$$

where the β multipliers can simply be derived from the recurrent form (3) in terms of the α ones

$$\begin{aligned} &\beta^{(0)}_n = 0, \\ &\beta^{(i)}_n = \frac{\alpha^{(i-1)}_n + (t_n - t_{n-i})\beta^{(i-1)}_n}{t_{n-1} - t_{n-1-i}}, \ i = 1, \dots, k_n. \end{aligned} \tag{6}$$

(Using (6) in (5) needs replacing the subscript n by $n+1$, of course.)

The *corrector* of the variables $\boldsymbol{x}_{n+1} := \boldsymbol{x}^{(j_{\max n})}_{n+1}$ for t_{n+1} is determined using the modified Newton iterations (the symbol x marks an element of the vector $\boldsymbol{x}$)

$$\begin{aligned} &\left[\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)^{(j)}_{n+1} + \left(\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{x}}}\right)^{(j)}_{n+1} \left(\frac{\mathrm{d}\dot{x}}{\mathrm{d}x}\right)_{n+1} \right] \Delta\boldsymbol{x}^{(j)}_{n+1} = \\ &\left[\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)^{(j)}_{n+1} + \gamma_{n+1} \left(\frac{\partial \boldsymbol{f}}{\partial \dot{\boldsymbol{x}}}\right)^{(j)}_{n+1} \right] \Delta\boldsymbol{x}^{(j)}_{n+1} = \\ &\qquad - \boldsymbol{f}^{(j)}_{n+1}, \ j = 0, \dots, j_{\max n} < \texttt{MAXIT} \end{aligned} \tag{7}$$

(`MAXIT` is maximum allowable number of iterations in one integration step), i.e., by repeated solving the linear system (7) with applying the *implicit* form of the derivatives approximation [2]

$$\begin{aligned} \dot{\boldsymbol{x}}^{(j)}_{n+1} &= \lim_{t_{n+2} \to t_{n+1}} \frac{\boldsymbol{x}^{(j)}_{n+2} - \boldsymbol{x}_{n+1}}{t_{n+2} - t_{n+1}} \\ &= \sum_{i=1}^{k_{n+1}} \frac{1}{t_{n+1} - t_{n+1-i}} \delta^{(i)} \boldsymbol{x}^{(j)}_{n+1} \\ \Rightarrow \gamma_{n+1} &= \sum_{i=1}^{k_{n+1}} \frac{1}{t_{n+1} - t_{n+1-i}}, \end{aligned} \tag{8}$$

that gives a standard formula $\gamma_{n+1} = 1/(t_{n+1} - t_n) = 1/\Delta t_{n+1}$ if the first order (i.e., Euler) method is used.

After resolving the linear system (7), the vectors $\boldsymbol{x}^{(\dots)}_{n+1}$ and $\dot{\boldsymbol{x}}^{(\dots)}_{n+1}$ are updated in the standard way

$$\boldsymbol{x}^{(j+1)}_{n+1} = \boldsymbol{x}^{(j)}_{n+1} + \Delta\boldsymbol{x}^{(j)}_{n+1}, \tag{9a}$$

$$\dot{\boldsymbol{x}}^{(j+1)}_{n+1} = \dot{\boldsymbol{x}}^{(j)}_{n+1} + \gamma_{n+1} \Delta\boldsymbol{x}^{(j)}_{n+1}. \tag{9b}$$

However, if an indication of divergence is detected during the iterations, then the logarithmic damping[1] (again, the symbol x marks an element of the vector $\boldsymbol{x}$)

$$\Delta x^{(j)}_{n+1} := \operatorname{sgn}\left(\Delta x^{(j)}_{n+1}\right) \left|x^{(j)}_{n+1}\right| \ln\left(1 + \frac{\left|\Delta x^{(j)}_{n+1}\right|}{\left|x^{(j)}_{n+1}\right|}\right) \tag{10}$$

can be used[2] for all the elements of the vector $\Delta\boldsymbol{x}^{(j)}_{n+1}$ before updating to the subsequent values by (9).

The logarithmic damping of divergence is a valuable tool ensuring the numerical stability. However, it is often insufficient from the point of view convergence, especially in the static domain (although a modification for the dynamic domain has also been implemented). Therefore, a novel way has been developed for suppressing possible divergence of the static variant of (7)

$$\left(\frac{\partial \boldsymbol{f}_0}{\partial \boldsymbol{x}_0}\right)^{(j)} \Delta\boldsymbol{x}^{(j)}_0 = -\boldsymbol{f}^{(j)}_0, \ \boldsymbol{x}^{(j+1)}_0 = \boldsymbol{x}^{(j)}_0 + \Delta\boldsymbol{x}^{(j)}_0, \quad j = 0, \dots, j_{\max 0}. \tag{11}$$

It uses the modification (m marks the dimension of $\boldsymbol{x}_0$)

```
if $j = 0$ then
    $\boldsymbol{x}' := \boldsymbol{x}^{(0)}_0$,
    $\Delta\boldsymbol{x}' := \Delta\boldsymbol{x}^{(0)}_0$,
    $\boldsymbol{f}' := \boldsymbol{f}^{(0)}_0$,
    iteration is accepted,
else
    if $\frac{1}{m} \sum_{i=1}^{m} \frac{|f^{(j)}_{0_i}|}{|f'_i| + \texttt{FNULL}} < 1$ then
        $\boldsymbol{x}' := \boldsymbol{x}^{(j)}_0$,
        $\Delta\boldsymbol{x}' := \Delta\boldsymbol{x}^{(j)}_0$,                    (12)
        $\boldsymbol{f}' := \boldsymbol{f}^{(j)}_0$,
        iteration is accepted,
    else
        $\Delta\boldsymbol{x}' := \frac{\Delta\boldsymbol{x}'}{2}$,
        $\boldsymbol{x}^{(j)}_0 := \boldsymbol{x}'$,
        $\Delta\boldsymbol{x}^{(j)}_0 := \Delta\boldsymbol{x}'$,
        iteration is rejected,
```

[1] An idea of the logarithmic damping is based on the Maclaurin series $\ln(1+x) = x - x^2/2 + x^3/3 - + \cdots \approx x$ for $x \to 0$.

[2] More precisely, $|x^{(j)}_{n+1}| + \texttt{NULL}$ is used instead of $|x^{(j)}_{n+1}|$ to avoid possible zero division (`NULL` is another algorithm controlling parameter).

which checks the residual value of $\boldsymbol{f}_0^{(j)}$ after each iteration. The basic idea of handling the differences $\Delta\boldsymbol{x}_0^{(j)}$ according to (12) is related to the fundamental property of the Newton-Raphson method. If the average value of the residues does not decrease then the difference is halved and the iteration is repeated. The halving is applied until the average residual value does not decrease. It is sure that the occurrence of the decreasing residue will be found—the program does not even contain a check for a possible infinite loop. As a result, only such $\Delta\boldsymbol{x}_0^{(j)}$ is used for updating the vector $\boldsymbol{x}_0^{(j)}$ that ensures the decrease of the average value of the residues.

The parameter `FNULL` prevents possible division with zero; $\boldsymbol{x}'$, $\Delta\boldsymbol{x}'$, and $\boldsymbol{f}'$ are the auxiliary vectors of the algorithm.

Using the Newton-Raphson method with the controlling procedure (12) leads to a very reliable convergence. However, the number of iterations could be large in that case.

After finishing the iterations (7), a new length of the integration step Δt_{n+2} and a new order of the polynomial interpolation k_{n+2} are to be chosen after converging the corrector (7) and (9)—the quality of the two procedures is very important for the overall efficiency of the algorithm.

Firstly, an estimation of the interpolation error in the last step must be determined. In general, the *absolute* truncation error for the k_{n+1} order caused by the derivatives approximation (8) may be written as

$$e_{n+1} = \frac{\Delta t_{n+1}}{t_{n+1} - t_{n-k_{n+1}}}\, \delta^{(k_{n+1}+1)} x_{n+1} \qquad (13)$$

for any element x_{n+1} of the vector $\boldsymbol{x}_{n+1}$. The relation (13) for the element e_{n+1} of the absolute truncation error vector $\boldsymbol{e}_{n+1}$ must be modified in the following way:

- the absolute errors are replaced by the relative ones (to be comparable with one another and with an algorithm parameter),
- the fraction $\Delta t_{n+1}/(t_{n+1} - t_{n-k_{n+1}})$ is omitted (to give a preference to simpler and more stable lower interpolation orders).

Consequently, the *relative* truncation error acquires the simple form [2][3]

$$\begin{aligned} \varepsilon_{n+1} &= \max_{\forall x_{n+1} \in \boldsymbol{x}_{n+1}} \frac{\left|\delta^{(k_{n+1}+1)} x_{n+1}\right|}{|x_{n+1}|} \\ &= \max_{\forall x_{n+1} \in \boldsymbol{x}_{n+1}} \frac{\left|x_{n+1} - x_{n+1}^{(0)}\right|}{|x_{n+1}|}. \end{aligned} \qquad (14)$$

Therefore, the truncation error can simply be checked using the difference between the corrector and the predictor—the step may be rejected and halved even after the *first* iteration of the corrector if the truncation error seems too big.

Secondly, the new step and order are determined by means of the error (14). Generally, the truncation errors of the i^{th} interpolation order can be determined by the formulae (consider a component of $\boldsymbol{e}$ with the greatest error)

$$\begin{aligned} e_{n+1}^{(i)} &= \text{const.}_{n+1}^{(i)} \left(\frac{\mathrm{d}^{i+1}x}{\mathrm{d}t^{i+1}}\right)_{n+1} \Delta t_{n+1}^{i+1}, \\ e_{n+2}^{(i)} &= \text{const.}_{n+2}^{(i)} \left(\frac{\mathrm{d}^{i+1}x}{\mathrm{d}t^{i+1}}\right)_{n+2} \Delta t_{n+2}^{i+1}. \end{aligned}$$

The new step estimate is based on the assumption of a similarity of neighboring steps

$$\begin{gathered} \text{const.}_{n+1}^{(i)} \approx \text{const.}_{n+2}^{(i)}, \\ \left(\frac{\mathrm{d}^{i+1}x}{\mathrm{d}t^{i+1}}\right)_{n+1} \approx \left(\frac{\mathrm{d}^{i+1}x}{\mathrm{d}t^{i+1}}\right)_{n+2}, \end{gathered}$$

which gives the relation

$$\frac{e_{n+2}^{(i)}}{e_{n+1}^{(i)}} \approx \left(\frac{\Delta t_{n+2}}{\Delta t_{n+1}}\right)^{i+1} \triangleq \frac{\varepsilon_{n+2}^{(i)}}{\varepsilon_{n+1}^{(i)}}.$$

The truncation error in the following integration step should be equal to ε—prescribed relative truncation tolerance, i.e., (Δt_{n+1} is already known when $\Delta t_{n+2}^{(i)}$ should be compared for all possible i)

$$\Delta t_{n+2}^{(i)} = \Delta t_{n+1} \sqrt[i+1]{\frac{\varepsilon}{\varepsilon_{n+1}^{(i)}}}, \; i = 1, \ldots, k_{n+1}+1, \qquad (15)$$

where all the possible truncation relative errors $\varepsilon_{n+1}^{(i)}$ are computed directly by the $\delta^{(i+1)} x_{n+1}$ (that is why the differences (2) are defined up to the $k_n + 2$ order—so that the order of the polynomial interpolation can sequentially increase). However, the step increase is limited due to the stability conditions, especially for higher

[3]Similarly as that in the logarithmic damping, $|x_{n+1}^{(j)}| + \texttt{NULL}$ is used instead of $|x_{n+1}^{(j)}|$ to avoid possible zero division.

orders of interpolation—see the valuable stability comparisons of the basic implicit integration methods in [2]. Thus, the relation (15) must be modified by a semiempirical factor

$$\Delta t_{n+2}^{(i)} = \begin{cases} \Delta t_{n+1} \sqrt[i+1]{\dfrac{\varepsilon}{\varepsilon_{n+1}^{(i)}}} & \text{for } \dfrac{\varepsilon}{\varepsilon_{n+1}^{(i)}} < 4, \\ \Delta t_{n+1} \sqrt[i+1]{4} & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, k_{n+1} + 1, \quad (16)$$

where the factor $\sqrt[i+1]{4}$ may theoretically be derived under special circumstances only; however, it has been proven by thousands practical analyses, as well. In conclusion, the new k_{n+2} order ($k_{n+2} \in \{1, \ldots, k_{n+1}+1\}$) is chosen, whose step determined by (16) is the longest.

To summarize, the algorithm defined above has the following advantages:

- it is also convenient for analyses of microwave (i.e., fast) devices—let us consider that the very short time steps in (3) are divided by one another (they are *not* multiplied as those in standard interpolation schemes which may cause underflow errors for higher orders);
- the algorithm is more flexible than the Gear method implemented in PSpice with respect to quick step and order alterations—the order of the interpolation may change in every step, for instance;
- the algorithm is convenient for the enhancement towards a time domain sensitivity analysis—due to its efficiency, the results are obtained in a reasonable time.

3 The Algorithm for the Optimization

Let us assume that some two circuit outputs are to be monitored in three points as seen in Fig. 1. The circles mark user-specified requirements for the outputs and the squares mark values of the outputs obtained after an analysis. The algorithm seeks to minimize the sum of squares of differences between them

$$S(x_1, \ldots, x_n) = \sum_{k=1}^{m} R_k^2 (x_1, \ldots, x_n), \; n \leqq m, \quad (17)$$

where the unknown optimized parameters of a circuit are marked by $x_1, \ldots, x_n$, and $R_k, \; k = 1, \ldots, m$ are the differences.

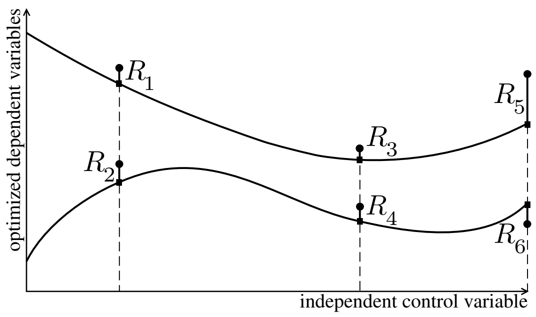

Figure 1: A diagram of a typical optimization task.

An extreme of the function of n variables (17) can be found in the standard way, i.e.,

$$\nabla S = \sum_{k=1}^{m} 2 R_k \nabla R_k = \mathbf{0}. \quad (18)$$

After a standard derivation [3], the generalized least-squares procedure is obtained applying (18)

$$\boldsymbol{J}^{\mathrm{t}} \boldsymbol{J} \, \Delta \boldsymbol{x}^{(l)} = -\boldsymbol{J}^{\mathrm{t}} \, \boldsymbol{r}, \; \boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} + \Delta \boldsymbol{x}^{(l)}, \qquad l = 1, \ldots, l_{\max}, \quad (19)$$

where l is the iteration index and

$$r_k = R_k \left[\boldsymbol{x}^{(l)} \right], \; \frac{\partial r_k}{\partial x_i} = \frac{\partial R_k}{\partial x_i} \left[\boldsymbol{x}^{(l)} \right],$$

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial r_1}{\partial x_1} & \cdots & \dfrac{\partial r_1}{\partial x_n} \\ \vdots & & \vdots \\ \dfrac{\partial r_m}{\partial x_1} & \cdots & \dfrac{\partial r_m}{\partial x_n} \end{bmatrix}, \qquad k = 1, \ldots m, \; i = 1, \ldots, n.$$

The generalized least-squares procedure is very fast, but sometimes insufficiently stable. For this reason, the method is combined with the gradient one

$$\Delta \boldsymbol{x}^{(l)} = -2 \, \boldsymbol{J}^{\mathrm{t}} \boldsymbol{r}, \; l = 1, \ldots, l_{\max}$$

to the Levenberg-Marquardt modification of (19)

$$\left[\boldsymbol{J}^{\mathrm{t}} \boldsymbol{J} + \lambda^{(l)} \mathbf{1} \right] \Delta \boldsymbol{x}^{(l)} = -\boldsymbol{J}^{\mathrm{t}} \boldsymbol{r}, \; \boldsymbol{x}^{(l+1)} = \boldsymbol{x}^{(l)} + \Delta \boldsymbol{x}^{(l)}, \qquad l = 1, \ldots, l_{\max}, \quad (20)$$

where $\mathbf{1}$ is unit matrix and $\lambda^{(l)}$ is a scalar iteration-dependent factor. There are many ways to optimally determine that factor for each iteration—the most sophisticated ones use an estimation based on eigenvalues of the Jacobian in (20) [4]. However, simpler empirical ways are mostly also successful [3]. The program also contains a version of the empirical methods (however, a way based on the eigenvalues is also possible) which seeks to minimize the $\lambda^{(l)}$ factor sequentially (i.e., to make the generalized least-squares method more influential at the end of the process, which is natural):

$$\begin{aligned}\lambda^{(1)} &= 1,\\ \lambda^{(l+1)} &= \frac{\lambda^{(l)}}{5}.\end{aligned} \tag{21}$$

However, this monotone decay must be interrupted (and therefore the gradient method must be sometimes made more influential) when the method seems to diverge:

$$\text{if } l > 1 \,\wedge\, S^{(l)} \geqq \min_{j=1}^{l-1} S^{(j)} \text{ then}$$
$$\boldsymbol{x}^{(l)} := \boldsymbol{x}^{(l-1)},\ \lambda^{(l)} := \lambda^{(l)} 5^2,$$

where the first multiplication by 5 compensates the division by 5 in (21) and the second multiplication by 5 increases that scalar factor.

Unfortunately, the method described above is insufficient for the majority class of the circuit optimization problems. Thus, an improved method has been implemented to our software equipment.

The improvement consists in the following steps:

- The differences defined in (19) *must* be normalized;
- These differences should also be weighted;
- The Jacobian $\boldsymbol{J}$ in (20) *must* be normalized too;
- The Jacobian can quickly be evaluated by sensitivities;
- Evaluating the Jacobian is not necessary in each iteration;
- Possible divergence of iterations (20) can be damped.

3.1 Normalization of the System of Equations

The models of circuit elements contain values of extreme orders (tiny ones together with the huge ones). For such systems, the standard optimization algorithms are unstable. Therefore, a normalization of differences is included to the algorithm as a new feature (together with their weighting, of course)

$$R'_k\left[\boldsymbol{x}^{(l)}\right] \triangleq w_k \frac{y_k^{(\text{output})}\left[\boldsymbol{x}^{(l)}\right] - y_k^{(\text{input})}}{\left|y_k^{(\text{input})}\right| + y_k^{(\text{null})}},$$
$$k = 1, \ldots, m, \tag{22}$$

where the superscipts (input) and (output) mark measured and optimized values if the optimization is used for the identification purposes. However, many numerical experiments have proved that a normalization of the Jacobian is also necessary:

$$\frac{\partial R'_k\left[\boldsymbol{x}^{(l)}\right]}{\partial x_i} := w_k \frac{\partial y_k^{(\text{output})}\left[\boldsymbol{x}^{(l)}\right]}{\partial x_i} \frac{x_i^{(\max)} - x_i^{(\min)}}{\left|y_k^{(\text{input})}\right| + y_k^{(\text{null})}},$$
$$k = 1, \ldots, m,\ i = 1, \ldots, n, \tag{23}$$

where $\partial y_k^{(\text{output})} \big/ \partial x_i$ is a result of *sensitivity* analysis.

The equation (22) is a definition. However, the equation (23) represents an assignment. Therefore, a solution of the system (20) must be modified by the assignment

$$\Delta x_i^{(l)} := \Delta x_i^{(l)} \left[x_i^{(\max)} - x_i^{(\min)}\right],\ i = 1, \ldots, n$$

after each iteration, where $x_i^{(\min)}$ and $x_i^{(\max)}$ represent minimum and maximum allowable values, respectively—they are specified by the user.

4 Testing the Novel Algorithm Features

4.1 Analysis

Consider the power operational amplifier in Fig. 2. The standard Newton-Raphson method is not able to determine the operating point due to powerful negative feedbacks in the circuit. However, if the modification (12) is used, the operating point is found without any problem using only 174 iterations.

4.2 Optimization

As known, the identification of the models of semiconductor devices can be considered demanding test of the optimization. For this reason, the identification of the model parameters for the original Czech MOSFET KF521 has been performed with the results shown in Fig. 3. The root mean square (rms) error of the identification was only $4.06\ \%$, and maximum absolute value deviation $(\delta_{\max})$ was only $14.5\ \%$—both numbers indicate successful optimization.

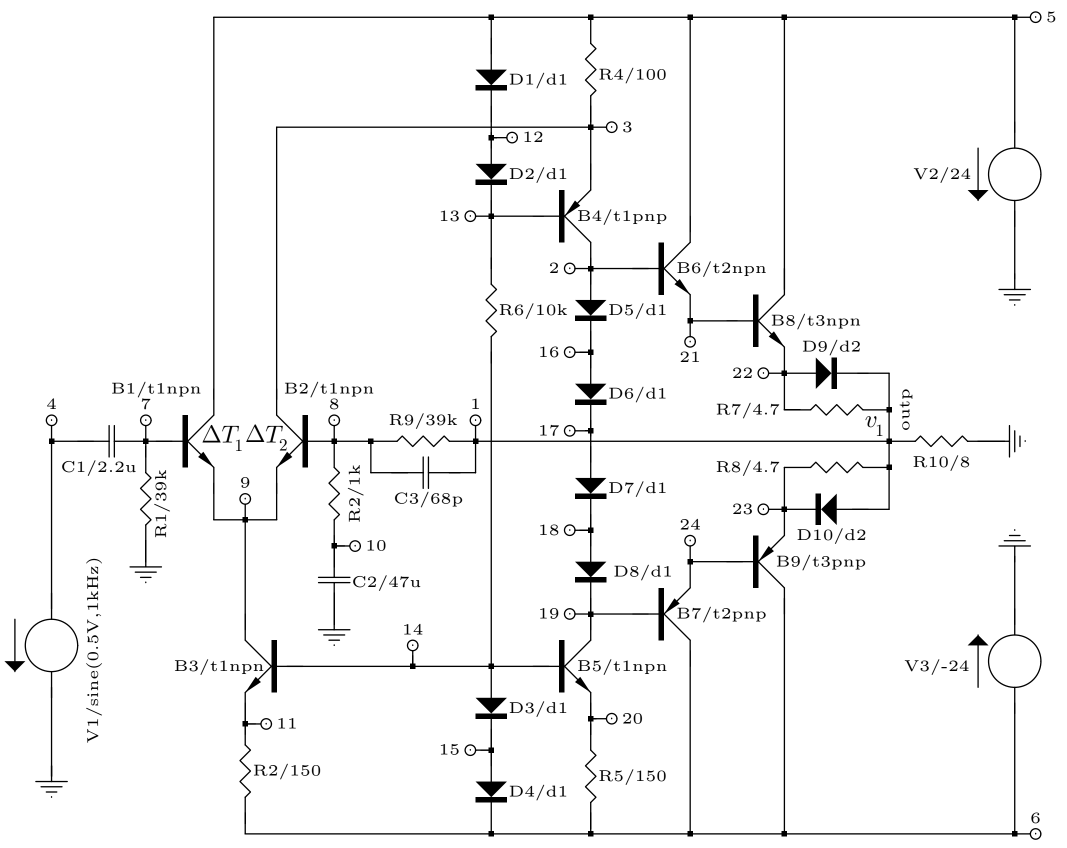

Figure 2: Power operational amplifier as an example of suppressing the divergence by means of the novel method.

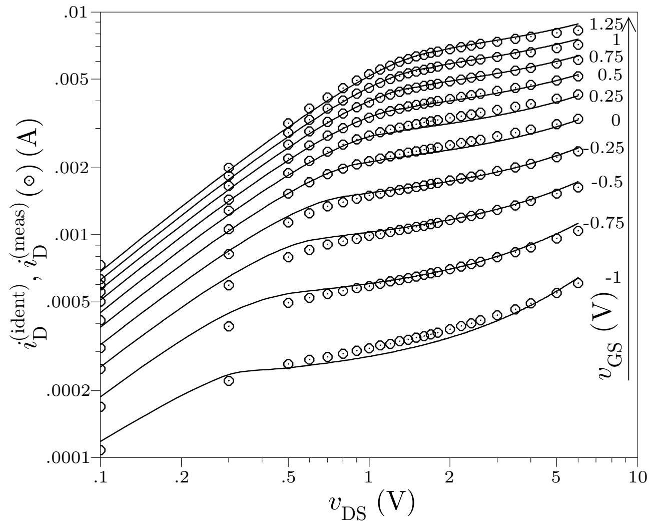

Figure 3: Forward DC characteristics of the N-MOSFET KF521.

5 Conclusion

A flexible analysis algorithm has been presented for solving the system of algebraic-differential equations with a novel robust method for suppressing divergence, which enables analyzing strong feedback systems.

An optimization algorithm has also been presented which is convenient for the robust identifications of complicated tasks. The algorithm has been improved using the normalization, which enables performing the tasks with large differences among optimized variables.

Acknowledgement This paper has been supported by the grant of the European Commission TARGET (6th Framework Programme of EU), by the Grant Agency of the Czech Republic, grant No. 102/05/0277, and by the Czech Technical University Research Project MSM 6840770014.

References

[1] Brenan K.E., Campbell S.L., and Petzold L.R. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. SIAM, Philadelphia, 1996.

[2] Petrenko A.I., Vlasov A.I., and Timtschenko A.P. *Tabular Methods of Computer-Aided Modeling (In Russian)*. Higher School, Kiyv, 1977.

[3] Fletcher R. *Practical Methods of Optimization*. John Wiley & Sons, 1978.

[4] Finschi L. An implementation of the Levenberg-Marquardt algorithm. *Eidgenőssische Technische Hochschule Zűrich*, 1996.