Split and Merge Strategies for Solving Uncertain Equations Using Affine Arithmetic

Oliver Scharf
oliver.scharf@ims.uni-hannover.de

Markus Olbrich
markus.olbrich@ims.uni-hannover.de

Erich Barke
erich.barke@ims.uni-hannover.de

Institute of Microelectronic Systems
Leibniz Universität Hannover, Germany

ABSTRACT

The behaviour of systems is determined by various parameters. Due to several reasons like e. g. manufacturing tolerances these parameters can have some uncertainties. Corner Case and Monte Carlo simulations are well known approaches to handle uncertain systems. They sample the corners and random points of the parameter space, respectively. Both require many runs and do not guarantee the inclusion of the worst case. As alternatives, range based approaches can be used. They model parameter uncertainties as ranges. The simulation outputs are ranges which include all possible results created by the parameter uncertainties. One type of range arithmetic is the affine arithmetic, which allows to maintain linear correlations to avoid over-approximation. An equation solver based on affine arithmetic has been proposed earlier. Unlike many other range based approaches it can solve implicit non-linear equations. This is necessary for analog circuit simulation. For large uncertainties the solver suffers from convergence problems. To overcome these problems it is possible to split the parameter ranges, calculate the solutions separately and merge them again. For higher dimensional systems this leads to excessive runtimes as each parameter is split. To minimize the additional runtime several split and merge strategies are proposed and compared using two analog circuit examples.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Type of simulation—*range arithmetic, parametric*; G.1.0 [**Numerical Analysis**]: General—*affine arithmetic*; G.1.5 [**Numerical Analysis**]: Roots of nonlinear equations—*iterative methods*

General Terms

Algorithms, Performance, Verification

Keywords

split, merge, circuit simulation, implicit equations, uncertain, parametric, non-linear

1. INTRODUCTION

System uncertainties are introduced by different sources, e.g. manufacturing tolerances, ageing or indirect measurement of system parameters. To verify uncertain systems several methods can be used. Well known methods are Monte Carlo or Corner Case simulations. They choose sets of parameters and for every set a nominal simulation is performed as if the parameters were exact. The Monte Carlo method samples the parameter space randomly, whereas the Corner Case method takes samples from the corners of the parameter space. If the relation between the parameters and the system behaviour is non-monotonic the Corner Case approach misses the extreme values. The Monte Carlo method can detect them if enough samples are drawn. However, both methods do not guarantee the inclusion of the worst case and require a lot of simulation runs. As they use single points of the parameter space these methods are called point arithmetic. Alternatively, methods based on range arithmetics like interval or affine arithmetic can be used. They allow to describe a range in the parameter space and calculate the corresponding range in the solution space. Arithmetic operations are defined on ranges so that all values from an input range are mapped to an output range. This guarantees the inclusion of the extremal values mathematically. Interval arithmetic suffers from an effect called error explosion. To avoid this affine arithmetic was proposed [2]. It maintains linear correlations during calculations and reduces the over-approximation in comparison to interval arithmetic. In literature some approaches exist to use range arithmetic for verifying uncertain systems. Methods for uncertain linear systems using affine arithmetic are described in [7], whereas interval methods for non-linear systems are shown in [5].

Equations of analog or mixed-signal circuits can contain arbitrary non-linear functions and can only be described implicitly:

$$f(\vec{x}, \vec{p}) = 0. \tag{1}$$

Parameter deviations can be caused by manufacturing tolerances or external influences like temperature. With decreasing structure sizes in microelectronics the influence of parameter deviations on the behaviour of analog circuits increases. An algorithm to solve implicit equations using affine arithmetic was proposed in [3, 4]. This method is limited in its convergence range. It only converges if the considered parameter deviations and non-linearities are small. To simulate larger circuits an extension of the algorithm was proposed in [6]. It splits the parameter space, calculates the

SIMUTOOLS 2015, August 24-26, Athens, Greece

DOI 10.4108/eai.24-8-2015.2260594

solution for the split range using affine arithmetic like before and merges the obtained solutions. The merged solution solves the original problem. In [6] all parameters are split and a merge is performed after each split. For higher dimensional systems this results in an explosion of the runtime, so a better strategy is needed. In this paper selective split strategies to split only selected parameters are explored. For transient simulations different merge strategies to postpone the merge operation are investigated as well.

2. PRELIMINARIES

Affine arithmetic is an enhancement of interval arithmetic. In contrast to interval arithmetic it preserves linear correlations during the calculation so that over-approximation can be reduced.

Parameters and variables with deviations can be described as affine forms. They are denoted in the following with a hat symbol:

$$\hat{x} = x_0 + \sum_{i \in \mathbb{N}_{\hat{x}}} x_i \varepsilon_i. \tag{2}$$

All deviation symbols ε_i lie in the interval $[-1 \dots 1]$ and correspond to a certain source of uncertainty. In this way the reason for a variation of the circuit's behaviour can be tracked. The set $\mathbb{N}_{\hat{x}}$ contains all indexes of the deviation symbols which make up the affine form $\hat{x}$. Vectors whose components are affine forms are denoted as $\vec{\hat{x}}$.

The affine form is symmetric to the central value x_0. The sum of the absolute values of all x_i denotes the maximal deviation from the central value and is called radius of the affine form:

$$\mathrm{rad}(\hat{x}) = \sum_{i \in \mathbb{N}_{\hat{x}}} |x_i|. \tag{3}$$

With the minimum and maximum of an affine form it can be converted to an interval:

$$\min(\hat{x}) = x_0 - \mathrm{rad}(\hat{x}) \tag{4}$$

$$\max(\hat{x}) = x_0 + \mathrm{rad}(\hat{x}). \tag{5}$$

The arithmetic operations on affine forms are defined in such a way that they return an affine form. All operations on affine forms

$$\hat{f}(\hat{p}) : \mathcal{P}(\mathbb{R}) \to \mathcal{P}(\mathbb{R}) \tag{6}$$

are defined inclusion isotonely concerning the corresponding point-arithmetic function

$$f(p) : \mathbb{R} \to \mathbb{R}. \tag{7}$$

That means the following holds:

$$\forall (q \in \mathbb{R}, \hat{q} \in \mathcal{P}(\mathbb{R}), q \in \hat{q}) : \left(f(q) \in \hat{f}(\hat{q}) \right) \tag{8}$$

$$\forall (\hat{r}, \hat{s} \in \mathcal{P}(\mathbb{R}), \hat{r} \subseteq \hat{s}) : \left(\hat{f}(\hat{r}) \subseteq \hat{f}(\hat{s}) \right). \tag{9}$$

All points within the input range are mapped on points of the output range. Functions can be classified in affine and non-affine functions. The result of an affine function can be exactly described as an affine form. Addition, subtraction and multiplication with a scalar constant belong to this class:

$$c(\hat{x} \pm \hat{y}) = c(x_0 \pm y_0) + \sum_{i \in (\mathbb{N}_{\hat{x}} \cup \mathbb{N}_{\hat{y}})} c(x_i \pm y_i)\varepsilon_i. \tag{10}$$

All other operations are non-affine. To represent their results as an affine form a first order Taylor approximation with a Lagrangian remainder around the central value x_0 is performed:

$$\tilde{f}(\hat{x}) = x_0 + f'(x_0)(\hat{x} - x_0) + \frac{f''(\xi)}{2}(\hat{x} - x_0)^2. \tag{11}$$

The term $x_0 + f'(x_0)(\hat{x} - x_0)$ can be written as an affine form. The approximation error is enclosed by an additional deviation term $y_{n+1}\varepsilon_{n+1}$ which holds the maximal approximation error in $\xi \in [min(\hat{x}), max(\hat{x})]$. So the result of a non-affine form can be written as:

$$\hat{y} = f_{\text{non-affin}}(\hat{x}) = y_0 + \sum_{i \in \mathbb{N}_{\hat{x}}} (y_i \varepsilon_i) + y_{n+1}\varepsilon_{n+1}. \tag{12}$$

The additional deviation symbol ε_{n+1} is uncorrelated to all previously used symbols. With each evaluation of a non-affine function an additional deviation symbol is generated. These will be called NLPD symbols (non-linear partial deviation) in the following. The second derivatives of most elementary functions are monotone. In these cases no search for the extremal values has to be performed to find the maximal approximation error. Functions composed from several elementary functions have non-monotone deviations in general. To avoid a computationally intensive search for the extremal values the optimal minimal inclusion is set aside. Instead, composed functions are decomposed into a sequence of elementary functions which is processed step by step. This results in an output range which overestimates the minimal range but guarantees to include it. This is one reason for over-approximation, the difference between the optimal and the calculated affine solution. Other sources of over-approximation are the solving algorithm itself and the merge operation. This will be discussed below.

An algorithm to solve implicit equations with uncertain parameters using affine arithmetic was described in [3, 4]. In Fig. 1 it is shown together with the extension from [6]. It determines the nominal solution first and then adds deviation symbols. The deviations are calculated from the linearisation of the parameter dependencies. Due to the linearisation error the inclusion is not guaranteed. To maintain inclusion additional uncorrelated deviation symbols are added to each variable. Their size is adjusted by an iterative method. The nominal solution is calculated by the function solveNominal using the Newton-Raphson method. The affine parameters $\vec{\hat{p}}$ are replaced by the nominal central values $\vec{p}_0$. The solutions $\vec{x}_0$ are used as the central values of the affine solutions $\vec{\hat{x}}$. As a first approximation of the deviation symbols x_i the linearised equation system is used:

$$f(x, p) \approx \left.\frac{df}{dx}\right|_{x_0, p_0} \cdot x_{PPD} + \left.\frac{df}{dp}\right|_{x_0, p_0} \cdot p_{PPD} = 0. \tag{13}$$

These deviation symbols have the same indexes as the parameter deviations and describe the same source of uncertainty. They are called PPD symbols (parameter partial deviation).

Due to the linearisation error the solution

$$\vec{\hat{x}} = \vec{x}_0 + \sum_{i \in \mathbb{N}_{\vec{\hat{p}}}} \vec{x}_i \varepsilon_i \tag{14}$$

solveAffine($\vec{p}$)

1: $\vec{x}_0 = \textbf{solveNominal}(f(\vec{x_0}, \vec{p_0}) = 0)$
2: $\vec{x}_{PPD} = \frac{dp}{df}\Big|_{x_0,p_0} - \vec{p}_{PPD} \cdot \frac{df}{dp}\Big|_{x_0,p_0}$
3: $\vec{\hat{x}} = \vec{x}_0 + \vec{x}_{PPD} \cdot \varepsilon_{PPD} + \vec{x}_{EPD} \cdot \varepsilon_{EPD}$
4: **repeat**
5: $\quad \vec{r} = f(\vec{\hat{x}}, \vec{p})$
6: $\quad \vec{t} = \mathbf{M}^{-1} \cdot \vec{x}_{EPD}$
7: $\quad$ **for all** t_i **do**
8: $\quad\quad$ **if** $t_{i,0} > 0$ **then**
9: $\quad\quad\quad s_i = \max(\hat{t}_i)$
10: $\quad\quad$ **else**
11: $\quad\quad\quad s_i = -\min(\hat{t}_i)$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad \vec{x}_{EPD} = \vec{s} \cdot \vec{x}_{EPD}$
15: $\quad \vec{\hat{x}} = \vec{x}_0 + \vec{x}_{PPD} \cdot \varepsilon_{PPD} + \vec{x}_{EPD} \cdot \varepsilon_{EPD}$
16: $\quad$ **if splitNeeded**() **then**
17: $\quad\quad (\vec{p}_i', \vec{p}_i'') = \textbf{splitParameters}(\vec{p})$
18: $\quad\quad \vec{\hat{x}}_i' = \textbf{solveAffine}(\vec{p}_i')$
19: $\quad\quad \vec{\hat{x}}_i'' = \textbf{solveAffine}(\vec{p}_i'')$
20: $\quad\quad \vec{\hat{x}} = \textbf{mergeSolutions}(\vec{\hat{x}}', \vec{\hat{x}}'')$
21: $\quad\quad$ **return** $\vec{\hat{x}}$
22: $\quad$ **end if**
23: **until** $(\forall i : |s_i| \approx 1 \vee x_{EPD,i} \approx 0)$
24: **return** $\vec{\hat{x}}$

Figure 1: Affine solving algorithm with splitting

computed up to now does not preserve inclusion. This is why additional uncorrelated deviation symbols called EPD symbols (enhanced partial deviation) are added to each variable. Their size can only be determined iteratively. According to [3] the start value $x_{EPD} = 0.1 \cdot \text{rad}(\hat{x})$ is chosen. Its value will be improved in the repeat until loop (Lines 4 to 23). With the current estimation $\vec{\hat{x}}$ the residuum of $f(\vec{\hat{x}})$ is calculated. This is used to calculate a refinement of the EPD symbols. These steps are repeated until the scaling factor s_i for every variable $\hat{x}_i$ reaches one or the $x_{EPD,i}$ disappears. This guarantees that the solution

$$\vec{\hat{x}} = \vec{x}_0 + \sum_{i \in \mathbb{N}_{\vec{p}}} (\vec{x}_i \varepsilon_i) + \vec{x}_{EPD} \varepsilon_{EPD} \tag{15}$$

includes the minimal solution area under all combinations of parameter variations in $\vec{p}$.

This algorithm can be used for all simulation types like AC, DC, transient (TR) and reachability analyses. In this paper we focus on DC and transient analysis.

The original algorithm suffers from some convergence problems. If the parameter deviations are too large or the equations are strongly non-linear the algorithm does not converge. To overcome these problems the parameter range can be split if necessary. For each part the affine solution is computed as before. Afterwards the solutions are merged to form an inclusion isotone solution of the original problem.

The split of a single parameter $\hat{p}_i$ into two parts $\hat{p}_i'$ and $\hat{p}_i''$ is performed as follows:

$$\hat{p}_i' = p_0 - \frac{p_i}{2} + \frac{p_i}{2}\varepsilon_i \tag{16}$$

$$\hat{p}_i'' = p_0 + \frac{p_i}{2} + \frac{p_i}{2}\varepsilon_i. \tag{17}$$

However, the split into more than two parts can be performed in a similar way. It cannot be determined a priori if it is necessary to perform a split to achieve convergence. In practice there are some critical operating points, but most operating points can be solved without splitting. Three criteria to automate the decision when to split have been proposed and compared in [6]. A split is performed if the algorithm does not converge in n_{max} steps (Criterion 1). If the considered system is solvable the scaling factor s over the number of iterations falls monotonically and approaches 1 asymptotically. Strongly increasing values (Criterion 2) or local maxima (Criterion 3) can be additional indicators for a non-converging system. After a split the function solveAffine is called recursively with the split parameters $\vec{p}'$ and $\vec{p}''$. The solutions $\vec{\hat{x}}'$ and $\vec{\hat{x}}''$ obtained from these calls are merged. The merged solution $\vec{\hat{x}}$ solves the circuit's equations for both $\vec{p}'$ and $\vec{p}''$ as well as the original set of parameters $\vec{p}$ inclusion isotonely. The merge algorithm used in [6] is shown in Fig. 2. To obtain the results of this paper an improved version of the merge operation shown in Fig. 3 was used. It gives the same result as the original version in Fig. 2 for non-overlapping affine forms, but it avoids over-approximation if the split solutions $\vec{\hat{x}}'$ and $\vec{\hat{x}}''$ have a large overlap or are nearly identical (see Fig. 4). This happens at points with large non-linearities. Nevertheless, the inclusion of the solution is maintained.

3. SPLIT STRATEGIES

One possibility to overcome the limitation of the affine solving algorithm is to split the parameter ranges. The smaller the ranges are the better the algorithm converges. As we have an implicit equation system it is not possible to split in state space as proposed e. g. in [1, 5]. We have to split the parameter range and calculate the state space solution for that split. Splitting all parameters like proposed in [6] leads to a large computational effort for high-dimensional systems as 2^n parts have to be calculated separately. 2^n is the number for splitting n parameter in halves, the number of parts for splitting in more parts is calculated analogically. To avoid this effort we propose to select one parameter to split. The selection is done for every split operation so different parameters can be chosen in subsequent split operations. After n_{max} splits using the selected split strategy were performed, the algorithm falls back to split all parameters to avoid endless loops if the current split strategy selects a parameter which does not improve convergence (see Fig. 5).

mergeSolutions($\vec{\hat{x}}', \vec{\hat{x}}''$)

1: $top = \max(\max(\vec{\hat{x}}'), \max(\vec{\hat{x}}''))$
2: $bottom = \min(\min(\vec{\hat{x}}'), \min(\vec{\hat{x}}''))$
3: $\vec{\hat{x}}.center = \frac{bottom+top}{2}$
4: $\vec{\hat{x}}.deviations = \sum_{i \in \mathbb{N}_{\hat{x}'}} x_i' \varepsilon_i + \sum_{i \in \mathbb{N}_{\hat{x}''}} x_i'' \varepsilon_i$

Figure 2: Merge operation from [6]

mergeSolutions$(\vec{\hat{x}}', \vec{\hat{x}}'')$

$top = \max(\max(\vec{\hat{x}}'), \max(\vec{\hat{x}}''))$
$bottom = \min(\min(\vec{\hat{x}}'), \min(\vec{\hat{x}}''))$
$\vec{\hat{x}}.center = \frac{bottom+top}{2}$
$\vec{\hat{x}}.deviations = \sum_{i \in \mathbb{N}_{\hat{x}'}} x'_i \varepsilon_i + \sum_{i \in \mathbb{N}_{\hat{x}''}} x''_i \varepsilon_i$
$neededRadius = \frac{top-bottom}{2}$
$scalingFactor = \frac{neededRadius}{x.\,\mathrm{rad}()}$
$\vec{\hat{x}}.deviations = scalingFactor \cdot \vec{\hat{x}}.deviations$

Figure 3: Improved merge operation

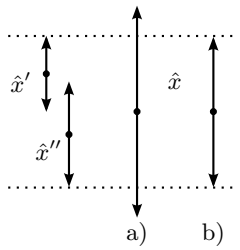

Figure 4: Merge with overlapping range, a) merge (Fig. 2), b) improved merge (Fig. 3)

As the relation between parameters and the result space is non-linear in general, the optimal parameter to split can not be obtained directly. Several strategies are described and compared here. They use the helper function findMax which returns the column index of the largest element $a_{i,j}$ in the matrix A. The strategy "select by deviation" selects the parameter with the largest deviation:

selectByDeviation

$deviationMatrix = [p_1.deviation() \dots p_n.deviation()]$
return findMax$(deviationMatrix)$.

The strategy "select by sensitivity" selects the parameter with the largest sensitivity according to the largest entry in the sensitivity matrix $\frac{df}{dp}$:

selectBySensitivity

return findMax $\left(\frac{df}{dp}\right)$.

Using the strategy "select by sensitivity and deviation" the sensitivity is scaled by the corresponding parameter deviations. The operator $\odot$ means element-wise multiplication. The parameter with the largest entry in the scaled sensitivity matrix A is selected:

selectBySensitivityAndDeviation

$$devMatrix = \begin{bmatrix} p_1.deviation() & \cdots & p_n.deviation() \\ \vdots & \vdots & \vdots \\ p_1.deviation() & \cdots & p_n.deviation() \end{bmatrix}$$

$A = devMatrix \odot \frac{df}{dp}$
return findMax(A).

The algorithms "select by EPD and PPD" and "select by EPD and sensitivity" use the size of the EPD symbol which is adjusted during solving. Both select the equation with the largest EPD symbol. As this equation can depend on multiple parameters two ways of selecting the parameter to be split from the selected equation are considered. The strategy "select by EPD and PPD" selects the parameter with the largest PPD:

selectParameters

$n = n + 1$
if $n < n_{max}$ **then**
    return selectByX()
else
    return All
end if

Figure 5: Selection of parameter to split, selectByX with X ∈ {Deviation, Sensitivity, SensitivityAndDeviation, EPDAndPPD, EPDAndSensitivity}

solveAffine$(solutionTree, parameterTree)$

for all nodes $n \in parameterTree$ **do**
    $n.isSolved = false$
end for
for all leaves $l \in parameterTree$ && $l.isSolved$ **do**
    $\vec{p} = l$
    replaceDerivatives()
⋮
    if splitNeeded() **then**
        $i =$ **selectParameters**()
        $subTree =$ **splitParameters**(l, i)
        addSubTree$(parameterTree, l, subTree)$
        continue
    else
⋮
        $l.isSolved = true$
        $solutionTree(l) = \vec{\hat{x}}$
    end if
end for

Figure 6: Affine solving algorithm using trees to store results and parameters

selectByEPDAndPPD

$EPDMatrix = [x_{1,EPD} \dots x_{n,EPD}]$
$index =$ **findMax**$(EPDMatrix)$
$PPDMatrix = [x_{index,PPD_0} \dots x_{index,PPD_{n_p}}]$
return findMax$(PPDMatrix)$.

In contrast, "select by EPD and sensitivity" selects the parameter with the largest sensitivity:

selectByEPDAndSensitivity

$EPDMatrix = [x_{1,EPD} \dots x_{n,EPD}]$
$index =$ **findMax**$(EPDMatrix)$
$B = \left[\frac{df_{index}}{dp_1} \dots \frac{df_{index}}{dp_{n_p}}\right]$
return findMax(B).

4. MERGE STRATEGIES

The algorithm in Fig. 1 always performs a merge operation after a split. For transient simulations other strategies are possible and are investigated in the following. For this purpose the solveAffine method has been modified. The pa-

1: $prevSolution = \vec{0}$
2: $prevParameter = \vec{\hat{p}}$
3: **for all** $t_n \in [0 \dots t_{\text{end}}]$ **do**
4: $(solutionTree, parameterTree) =$ **solveAffine**$(prevSolution, prevParameter)$
5: $(prevSolution, prevParameter) =$ **mergeTree**$(solutionTree, parameterTree)$
6: $\overrightarrow{solution}.append(prevSolution)$
7: **end for**
8: **plot**$(\overrightarrow{solution})$

Figure 7: Affine transient simulation with merge strategy "Merge each"

rameters and the solutions are stored in trees (see Fig. 9). Each node of the parameter tree represents a combination of parameters. The flag *isSolved* indicates that this combination has been solved successfully. The result for this combination is stored in the corresponding node of the solution tree. The extended solving algorithm in Fig. 6 resets the *isSolved* flag for all nodes (Line 1) of the parameter tree at first. Then it iterates over all leaves not yet solved (Line 4). With the parameter set stored in the current leaf the calculations according to Fig. 1 are performed. The function replaceDerivatives replaces the derivatives with the backward Euler formula. The values of the previous variables are taken from the corresponding node of the solution tree of the previous time step t_{n-1}. If an additional split operation is necessary in the current time step t_n which has not been calculated in the previous time step t_{n-1} the result from the corresponding parent node is taken (see Fig. 10). This introduces over-approximation as the split parameter range gives tighter inclusions in general but maintains the inclusion of the result. Otherwise it would be necessary to go back in time and recalculate all splits additionally used in the current time step for all previous time steps. If the solution step was successful the node is marked as solved and the result is stored in the solution tree (Lines 13 f.). If a split is needed then one or all parameters are selected and a split is performed. The resulting subtree is attached to the current leaf l and the iteration over all leaves of the parameter tree is restarted (Line 9 - 11). After solving the equations with the split parameter a merge operation is performed to get a single affine result for plotting.

For transient simulations there are several options when the merge operation can be performed. It can be performed after each time step. This strategy is called "merge each step" (see Fig. 7). This way only time steps which require splitting according to the used splitting criteria are split. On the other hand, it is necessary to redetermine the required split parameter set for each time step. This requires lots of iterations of the affine solving algorithm as an a priori determination is not possible. As the merge operation introduces additional over-approximation, this method provides a more pessimistic inclusion. The error propagates from time step to time step via the backward Euler formula. The opposed strategy to keep the split parameter for all remaining time steps ("do not merge", Fig. 8) gives tighter inclusion as the corresponding node from the previous solution tree is used for integration. Some compromises between both extremes

1: $prevSolutionTree.root = \vec{0}$
2: $prevParameterTree.root = \vec{\hat{p}}$
3: **for all** $t_n \in [0 \dots t_{\text{end}}]$ **do**
4: $(prevSolutionTree, prevParameterTree) =$ **solveAffine**$(prevSolutionTree, prevParameterTree)$
5: $\overrightarrow{solutionTree}.append(prevSolutionTree)$
6: **end for**
7: $\overrightarrow{solution} =$ **mergeTree**$(\overrightarrow{solutionTree})$
8: **plot**$(\overrightarrow{solution})$

Figure 8: Affine transient simulation with merge strategy "Do not merge"

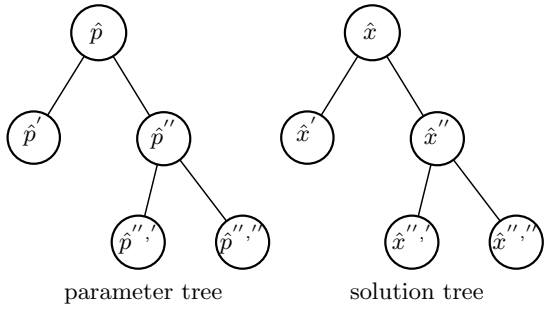

Figure 9: Parameter and solution tree built by algorithm in Fig. 6

are considered, too. The strategy "merge every n^{th} step" keeps the splits from previous steps and only merges after n steps where n can be set by the user. Another strategy tries to solve without splits first and only if this fails it reconsiders the splits from the previous time step (strategy "try root first").

5. RESULTS

The affine solving algorithm and the proposed split and merge strategies were implemented in MATLAB. To compare their performances two exemplary circuits were investigated. The first one is the inverter circuit using a bipolar transistor depicted in Fig. 11. Its parameters and their uncertainties are given in Table 1. It was chosen because of its non-linear behaviour when sweeping the operating ranges of

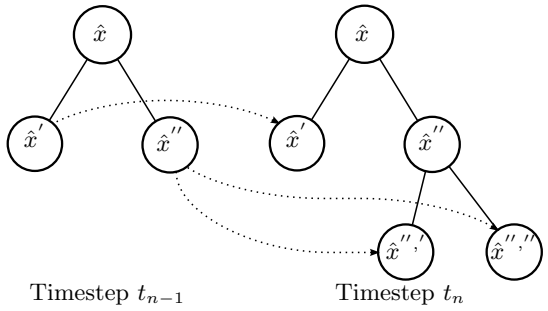

Figure 10: Solution tree for two time steps of a transient simulation, the dotted arrows show the source node for the backward Euler formula

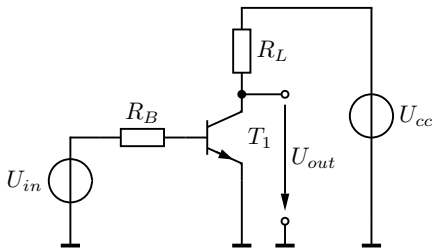

Figure 11: BJT inverter circuit

Table 1: Parameters for BJT inverter in Fig. 11

Symbol	Central value	Deviation
R_B	5 kΩ	±450 Ω
R_L	10 kΩ	±50 Ω

the transistor which is a challenge for the affine solving algorithm. The other circuit is an active bandpass with an ideal op-amp, shown in Fig. 12. It was chosen as it has a larger number of parameters with uncertainties which makes the selection of the right parameter to split more important. The parameters and their uncertainties are given in Table 2. To determine if splitting is necessary a maximum number of 30 iterations (Criterion 1) was used, unless noted otherwise.

The simulation setups used for these examples are summarized in Table 3. The over-approximation and the runtimes for setup S_1 with different numbers of splits (forced independently of split criterion) is shown in Fig. 13. This setup cannot be solved without splitting. The over-estimation decreases with the number of splits but the runtime increases exponentially. The number of splits and with it the runtime needed to solve the system equations increase with the size of the deviations (see Fig. 14). The time needed to solve using the same number of splits and the same size of deviations, can be reduced by using the different split strategies (see Table 4). For comparison the results of splitting a random parameter and splitting all parameters are also given. The results in Fig. 13 and Table 4 show the behaviour of the split strategies by an artificial enforcement of a given number of splits. The runtimes for the different

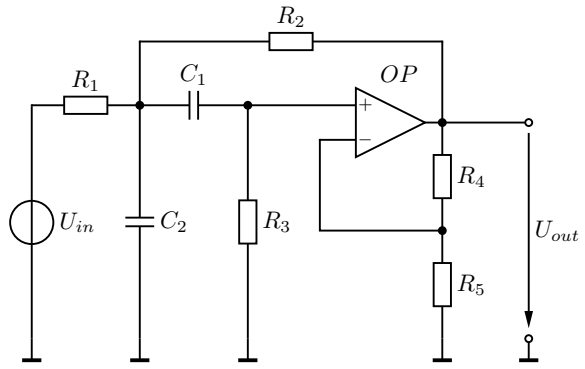

Figure 12: Bandpass circuit

Table 2: Parameters for bandpass circuit in Fig. 12

Symbol	Central value	Deviation
R_1	15.9 kΩ	±1.125 kΩ
R_2	15.9 kΩ	±1.11 kΩ
R_3	31.8 kΩ	±3.01 kΩ
R_4	20 kΩ	±3 kΩ
R_5	20 kΩ	±3 kΩ
C_1	10 nF	±0.2 nF
C_2	10 nF	±0.2 nF

Table 3: Simulation setups used for BJT Inverter (Inv.) in Fig. 11 and Bandpass (BP) in Fig. 12

Name	Circuit	Type	t_{end}/ms	U_{in}/V
S_1	Inv.	DC	-	0.489
S_2	Inv.	TR	0.025	2.5+2.5·sin(2π·0.5 kHz·(t+1.29 ms))
S_3	BP	TR	0.75	$2 \cdot \sin(2\pi \cdot 0.5\,\text{kHz} \cdot (t - 1\,\text{ms})) + 2 \cdot \sin(2\pi \cdot 2.5\,\text{kHz} \cdot (t - 5\,\text{ms}))$

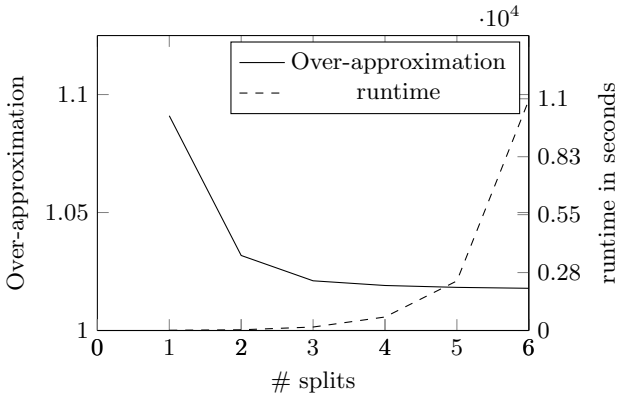

Figure 13: Over-approximation and runtime versus number of splits for S_1, split strategy = All

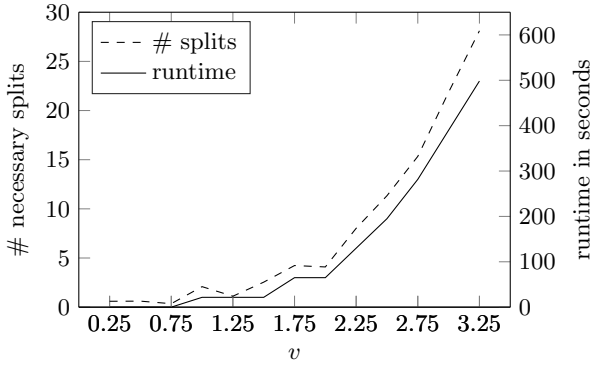

Figure 14: Number of splits and runtime versus deviation for S_1, parameter deviations given in Table 1 are scaled with factor v, split strategy = All

Table 4: Simulation runtime in seconds for different number of splits (forced independently of splitting criterion) and different split strategies for S_1, $\lightning$ means that convergence was not achieved in 30 iterations

	# splits					
split strategy	0	1	2	3	4	5
All	$\lightning$	12.4	33.6	165.2	640.6	2353.5
Deviation	$\lightning$	10.5	18.3	52.0	179.5	691.2
Sensitivity	$\lightning$	29.3	41.3	80.9	216.9	766.3
Sens. and deviation	$\lightning$	20.6	39.8	80.7	204.9	779.5
EPD and PPD	$\lightning$	26.5	57.3	78.9	218.5	781.6
EPD and sensitivity	$\lightning$	26.8	40.0	79.3	204.2	771.9
Random	$\lightning$	34.5	31.0	54.6	204.8	765.1

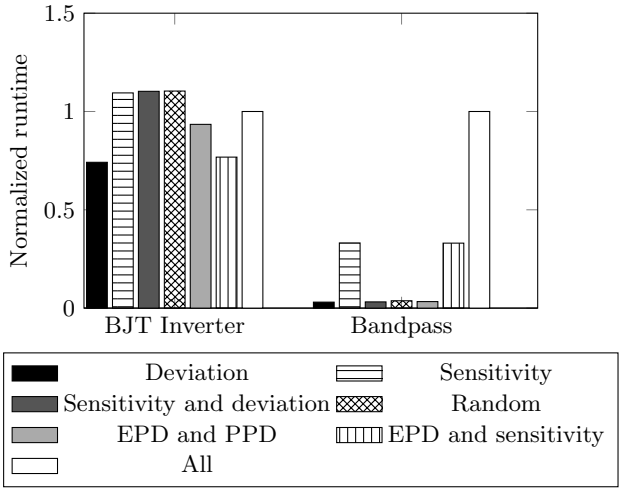

Figure 15: Simulation runtime for different split strategies

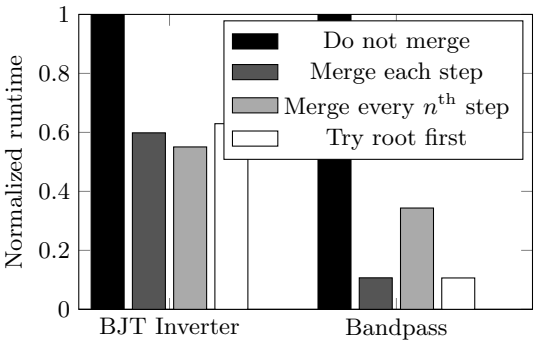

Figure 16: Simulation runtime for different merge strategies

strategies using Criterion 1 to determine if splitting is necessary are shown in Fig. 15. They are evaluated using DC simulations for the inverter (S_1) and transient simulations for the bandpass circuit (S_3) as no splitting was necessary for DC. The runtimes are normalized to the case when all parameters are split. For both circuits the strategy "select by deviation" is the fastest. The benefit is a lot larger for the bandpass (3 % of the runtime necessary) as for the inverter (74 %) as it has more uncertain parameters. For the BJT inverter the strategies "sensitivity", "sensitivity and deviation" and "random" are even slower than splitting all parameters. For the evaluation of the merge strategies transient simulations using the setups S_2 and S_3 were performed. The split strategy was set to split all parameters in each split operation. As a compromise for both circuits $n = 4$ was selected. The runtimes for the different merge strategies are shown in Fig. 16. They are given normalized to the strategy with the largest runtime which was "do not merge" for both circuits. The fastest merge strategy for the inverter was "merge every n^{th} time step" whereas for the bandpass "merge each step" was the fastest. A comparison of the over-approximation for the different merge strategies is shown in Fig. 17. The over-approximation was calculated as mean relative over-approximation m_{OA}:

$$m_{OA} = \frac{1}{n} \sum_{x=0}^{x_n} \frac{d_A(x)}{d_{MC}(x)}. \tag{18}$$

d_A and d_{MC} denote the interval width determined from the affine and the minimum/maximum of 1000 Monte-Carlo runs, respectively. For the bandpass the over-approximation for all strategies is larger than for the BJT inverter as there are more uncertain parameters and more split operations are performed. The "do not merge" strategy causes the smallest over-approximation for both circuits. For the BJT inverter the benefit is small compared to the other three strategies which are almost equal in terms of over-approximation. By choosing the slowest strategy "do not merge" the over-approximation can be reduced most efficiently.

6. CONCLUSION

The convergence range of the original affine solving algorithm is limited to small deviations. If the parameters are split it is possible to perform simulations on circuits with uncertain parameters which are not possible without splitting.

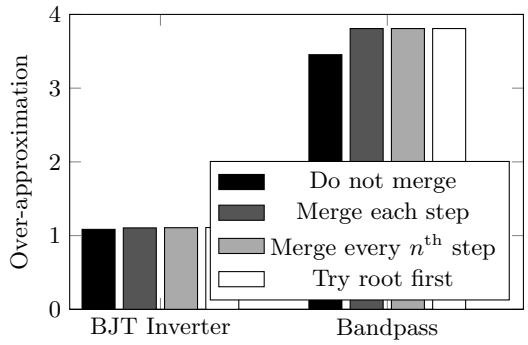

Figure 17: Over-approximation m_{OA} for different merge strategies

Additionally, the over-approximation can be reduced by employing more splits. For higher dimensional systems splitting all parameters leads to excessive runtimes as the number of parts to calculate increases exponentially. To avoid this drawback different strategies for splitting and merging were presented and compared. The split strategies select one parameter to split by different criteria. The best split strategy in terms of runtime is "select by deviation" which selects the parameter with the largest deviation for splitting. This reduces the runtime to 3 % of the time needed to split all parameters for the bandpass example. For transient simulations different merge strategies were proposed. The fastest merge strategy depends on the circuit. For the BJT inverter the strategy "merge every n^{th} step" and for the bandpass "merge each step" was the fastest. Comparing the merge with the split strategies it can be observed that the latter provide higher benefits in terms of runtime. Nevertheless, merge strategies are crucial as they influence the over-approximation via the backward Euler formula and by that the solvability of transient simulations. The merge strategy "do not merge" caused the smallest over-approximation but needs the largest runtime. The other merge strategies are faster but cause larger over-approximation which is almost equal among themselves. By choosing a merge strategy runtime can be traded for over-approximation depending on which effect is more critical in the current application.

7. REFERENCES

[1] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *47th IEEE Conference on Decision and Control*, pages 4042–4048, Dec 2008.

[2] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium Monographs. IMPA, 1997.

[3] D. Grabowski. *Gebietsarithmetische Verfahren zur Simulation analoger Schaltungen mit Parameterunsicherheiten*. PhD thesis, Leibniz Universität Hannover - Institute of Microelectronic Systems, 2009. In German.

[4] D. Grabowski, M. Olbrich, and E. Barke. Analog circuit simulation using range arithmetics. In *Proceedings of the ASP-DAC 2008*, pages 762–767.

[5] I. Krasnochtanova, A. Rauh, M. Kletting, H. Aschemann, E. P. Hofer, and K.-M. Schoop. Interval methods as a simulation tool for the dynamics of biological wastewater treatment processes with parameter uncertainties. *Applied Mathematical Modelling*, 34(3):744 – 762, 2010.

[6] O. Scharf, M. Olbrich, and E. Barke. Lösungsverfahren für nichtlineare implizite Gleichungssysteme unter Verwendung von Affiner Arithmetik und Gebietsaufteilungen. In *ANALOG 2013*. In German.

[7] I. Skalna. Direct method for solving parametric interval linear systems with non-affine dependencies. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 6068 of *Lecture Notes in Computer Science*, pages 485–494. Springer, 2010.