Significance Analysis for Numerical Models

Jan Riehme, Uwe Naumann
LuFG Informatik 12, RWTH Aachen University
Software and Tools for Computational Engineering
52074 Aachen, Germany
Email: [riehme,naumann]@stce.rwth-aachen.de

***Abstract*—We report first results of a significance analysis developed within the FET-open project SCoRPiO[1]. SCoRPiO aims to introduce result significance to the hardware development process in order to reduce the safeguard power consumption by permitting a controlled level of imprecision into computations and data.**

An important aspect of the project is to formally introduce computational significance as an algorithmic property and expose reliability and energy to the level of the programming model and algorithm. Based on that, SCoRPiO seeks to devise techniques that facilitate automatic characterization of code and data significance using compile-time or runtime analysis.

As a part of SCoRPiO techniques for significance analysis are developed: Based on the source code of algorithms and user-provided significance information, the analysis should identify parts of the algorithms that can be evaluated with less accuracy and, hence, higher energy efficiency.

We describe in this paper the initial version of our significance model. At the moment we start applying it to larger, more realistic codes. First results already initiated the development of refinement strategies for the analysis. Therefore we do present ideas and work in progress instead of a finished theory.

I. Introduction

The SCoRPiO project observes that part of the problem of energy inefficiency is that all computations are treated as equally important, despite the fact that only a subset of these computations might suffice to achieve an acceptable QoS. SCoRPiO seeks to devise techniques that facilitate automatic characterization of code and data significance, using compile- or run-time analysis. In other words, SCoRPiO will invent methods and algorithms to assess the significance of a set of computations and data structures. An important step in the direction of automatically (or semi-automatically) detecting component criticality is to formally define significance of different code fragments, at different times during the execution life of applications.

To our knowledge no rigorous theoretical approach to quantify, or even define significance of numerical models and codes exists so far. Nevertheless, numerous Monte Carlo based significance tests are applied already during different stages of real-world system design.

The significance analysis introduced in SCoRPiO is part of the development phase of an algorithm, that may potentially be executed on an unreliable processing core. Thus, significance analysis can be considered as a compile time technique, where a high computational effort by the compiler can be tolerated. In the context of this project we develop a concept of significance for numerical models based on Algorithmic Differentiation and interval evaluation of `C` and `C++` codes. We investigate also numerical approaches that are highly parallel, yet very costly and may require the use of massively parallel systems (recursive intervall splitting, sampling, Monte Carlo methods, exhaustive search).

The rest of the document is organized as follows: In section II we provide an overview of the suggested significance analysis. A more rigorous definition of significance is developed in section III. In section IV we apply the proposed significance analysis to some examples.

II. Methods for Significance Analysis

In this section we provide an overview of the significance analysis developed in SCoRPiO by discussing possible approaches in general, whereas in section III we develop the significance analysis approach in more detail.

Our literature research did not arose any previous work in significance analysis for numerical models that surpassed discussions of error propagation. Therefore all methods to tackle significance discussed in this section are new contributions that utilizes well established techniques (interval arithmetic, Algorithmic Differentiation, Monte Carlo, Sampling) in a new context.

A. Goal of Significance Analysis

Based on a given implementation of an algorithm and user provided significance and error tolerance information, parts of the given *code* that are *insignificant* under *certain conditions* have to be identified. Moreover, these, potentially dynamic, conditions need to be identified as well.

The user has to specify ranges of input variables, and specify minimum significance bounds or maximum error bounds of the results, depending on the chosen significance criterion.

An identified insignificant computation has a series of appealing properties:

- It can potentially be executed on unreliable hardware as a whole.
- It can be computed with lower precision.

[1] http://www.scorpio-project.eu/outline/

- It can be replaced by a fixed representative of the output range of the code.
- It can be replaced by a less computationally expensive computation reusing previously computed results.

To develop a robust significance analysis, we investigate the methods described in the following sections.

In this section we assume that the application code implements a mathematical function $f : I\!R \longrightarrow I\!R$, that computes a scalar output $y \in I\!R$ by evaluating $y = f(x)$ (e.g. running the given code) for a given scalar input $x \in I\!R$.

B. Computation in interval arithmetic (IA)

For an input interval $[x] = [\underline{x}, \overline{x}] = \{x \in I\!R | \underline{x} \leq x \leq \overline{x}\}$ with lower bound $\underline{x} \in I\!R$ and upper bound $\overline{x} \in I\!R$, interval arithmetic (IA) [1], [2] allows to compute enclosures $f[x]$ that contain all possible function values of $f(x)$ for $x \in [x]$, that is $f[x] \supseteq \{f(x) | x \in [x]\}$ over the input interval $[x]$.

Evaluations in interval arithmetic might suffer from the so called *wrapping effect* under certain conditions, resulting in a large overestimation of $[y] = f[x]$. Affine arithmetic [3] promises to reduce the wrapping effect in many cases.

For interval evaluations, insignificance can be defined by the inequality

$$w([y]) < \epsilon, \tag{1}$$

i.e. input variable x is called **insignificant** for y if the width of the output interval $[y] = f[x]$ is smaller then the user provided significance bound ϵ.

If the output interval $[y] = f[x]$ for the input interval $[x]$ is "acceptable" in the sense of inequality (1), then all **values** $x \in [x]$ **from input interval** $[x]$ **are insignificant**. This knowledge can be exploited by the following modifications to the original evaluation of function f:

- **Unreliable hardware**: Using unreliable hardware to compute an actual input $\check{x}$ does not harm the result $f(\check{x})$ as long as $\check{x} \in [x]$. Than the reliable (e.g. standard) evaluation of $f(\check{x})$ will give a reasonable result $f(\check{x}) \in [y]$.
- **Lower precision**: Computing the actual input $\check{x}$ with a lower precision does not harm the result $f(\check{x})$ as long as $\check{x} \in [x]$. Than the evaluation of $f(\check{x})$ in higher (e.g. standard) precision will give a reasonable result : $f(\check{x}) \in [y]$.
- **Replace by a constant**: If the actual input value $\check{x}$ is in the interval $[x]$ (i.e. $\check{x} \in [x]$), the computation of $f(\check{x})$ can be entirely omitted by having f return a constant value $\check{y} \in [y]$ (e.g. the midpoint $m[y] = \frac{\overline{y} - \underline{y}}{2}$ of $[y]$).

If the significance analysis for a given input interval $[x]$ does not succeed in identifying insignificant computations, sub-intervals of $[x]$ might be searched by heuristic sampling.

If the number of inputs increases, the sub-interval approach becomes a combinatorial problem (as any high dimensional forward analysis such as Finite Differences, Tangent-Linear models, Monte Carlo). Still, since the evaluation of different sub-intervals can be performed independently, such high dimensional problems can be tackled by exploiting parallelism.

Note that with pure interval arithmetic, significance estimation for intermediate program variables v (contained in the code of the implementation of $f(x)$) is possible only by an additional sampling over the (possibly large number) of intermediate program variables.

The following section introduces a concept that allows to estimate significance of intermediate program variables directly with interval valued adjoints, based on algorithmic differentiation (AD) [4], [5] and interval arithmetic.

C. Interval Derivatives by Algorithmic Differentiation

This subsection outlines our favorite approach for automatic significance detection. See [6] for details.[2]

The template class library `dco/c++` (Derivative Code by Overloading in `C++`) [7], [8], [9] is a `C++` implementation of tangent-linear and adjoint Algorithmic Differentiation (AD) [4], [5], developed at the institute Software and Tools for Computational Engineering (STCE) at RWTH Aachen University. For numerical codes implementing $y = f(x)$, `dco/c++` exploits overloading of operators and intrinsic functions to compute derivatives $\nabla_x y = \nabla_x f(x)$ of outputs y with respect to input x.

For the purpose of significance analysis `dco/c++` templates were specialized with an interval base type (`dco/scorpio`), that allows to apply AD to interval functions [10]. Thus interval enclosures of $f[x]$ and its first order derivative $\nabla_{[x]}[y] = \nabla_{[x]} f[x]$, that is the derivative of the function result $[y]$ with respect to the input variable $[x]$, can be computed:

$$\nabla_{[x]}[y] = \nabla_{[x]} f[x] \supseteq \{\nabla_x f(\hat{x}) | \hat{x} \in [x]\}. \tag{2}$$

If `dco/scorpio` computes an interval derivative $\nabla_{[x]}[y]$ of the function result $[y] = f[x]$ with respect to input $[x]$ without overestimation (equality in (2)), we have

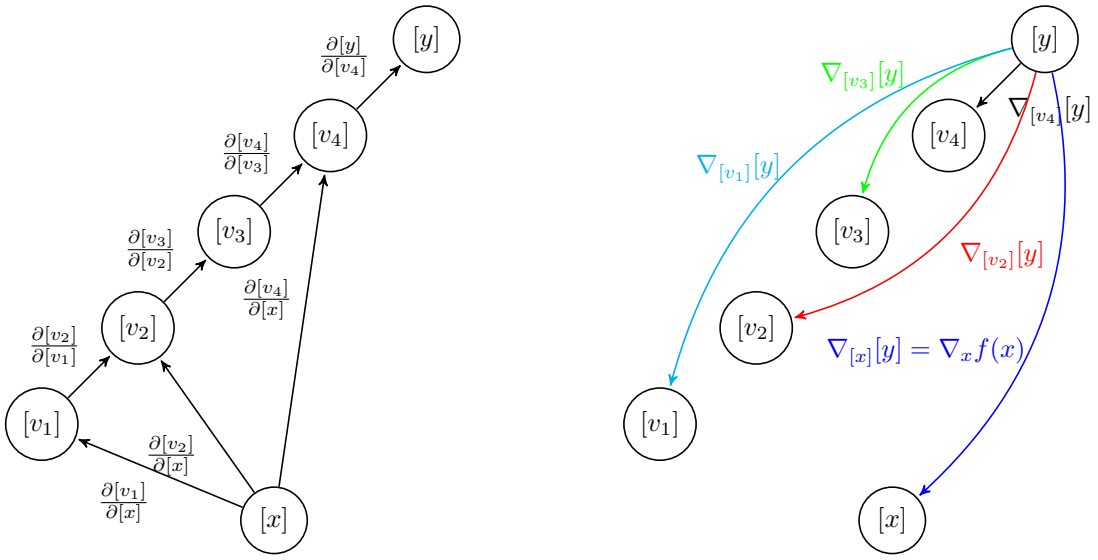
$$\nabla_{[x]}[y] = \left[\min_{\hat{x} \in [x]} \{\nabla_{\hat{x}} f(\hat{x})\}, \max_{\hat{x} \in [x]} \{\nabla_{\hat{x}} f(\hat{x})\} \right]. \tag{3}$$

In other words, the steepest downward and upward slopes of f in the interval $[x]$ form the bounds of the interval derivative. With overestimation only the following inequalities hold:

$$\underline{\nabla_{[x]}[y]} \leq \min_{\hat{x} \in [x]} \{\nabla_{\hat{x}} f(\hat{x})\}, \ \overline{\nabla_{[x]}[y]} \geq \max_{\hat{x} \in [x]} \{\nabla_{\hat{x}} f(\hat{x})\}. \tag{4}$$

Figure 1(a) provides a schematic example of an interval evaluation process for a function $f(x)$, using the computational graph. From input x four intermediate variables are used before the final output value y is computed. Every node in the computational graph represents an atomic

[2] Available under www.scorpio-project.eu.



(a) Computational graph of $f(x)$ with local partials

(b) Adjoints available after evaluating $\nabla_x f(x)$

Figure 1. Computational graph and adjoint propagation

operation (+, −,⋆, /) or intrinsic function evaluation (`sin`, `cos`, `exp`, etc.). Algorithmic Differentiation annotates the edges towards a vertex v in the computational graph with local partial derivatives of the operation represented by v with respect to its arguments (vertices with edges pointing towards vertex v).

Derivatives by means of adjoint mode AD are computed by propagating derivatives through the annotated computational graph, incorporating the local derivatives on the edges: By *adjoint models* derivatives (adjoints) of outputs are propagated towards adjoints of the inputs.

Once the interval derivative $\nabla_{[x]}[y] = \nabla_{[x]} f[x]$ has been evaluated for an adjoint model build by `dco/scorpio`, interval derivatives $\nabla_{[v]}[y]$ of the function result with respect to *all* intermediate program variables are also available due to the internal structure of `dco/c++` (figure 1(b)). Therefore, we obtain an estimation of the extremal slopes of all intermediate program variables. In contrast to pure interval arithmetic, described in section II-B and sampling based methods discussed shortly in section II-D, this approach allows an efficient significance analysis for all intermediate program variables.

Insignificance of the intermediate program variable v based on interval valued adjoints can be defined by the following inequality:

$$w([v] \cdot \nabla_{[v]}[y]) < \epsilon, \tag{5}$$

i.e. v is called insignificant for y if the width of $\nabla_{[v]}[y]$ scaled over the interval $[v]$ is smaller then the user provided significance bound. Note that inequality (5) can be applied to input variables too.

Insignificant variables might be treated as described in section II-B. For example, insignificant variables could be replaced by a constant, or could be computed on unreliable, but less power consuming hardware. In addition, since with interval valued adjoints all intermediate program variables can be tested for significance too, the following new modification to the evaluation of function f arises:

- **Replace by a previous result**: If an insignificant program variable v is evaluated repeatedly (i.e. the same memory location is overwritten with new values), a previously computed value $\hat{v}$ can be reused for $\check{v}$ if additionally holds $[\hat{v}] \subseteq [\check{v}]$ for the interval evaluation. If so, the computation of a new value $\check{v}$ for v can be entirely omitted.

Note that the reused value $\hat{v}$ will not be *replaced by a constant* after the significance analysis: Different input values $\bar{x}, \tilde{x} \in [x]$ might lead still to different intermediate values of $\hat{v}$ and $\check{v}$. Consequently, one can expect to introduce less error to the evaluation of function f by the modification *Replace by a previous result* than by *Replace by a constant*.

In section III we introduce a number of different criteria based on interval valued adjoints that will be considered later too.

Overestimation of $[y] = f([x])$ and thus of $\nabla_{[x]}[y]$ due to the *wrapping effect* in interval evaluations might be addressed by affine arithmetic [3].

Again, if the result of the significance analysis for the given input interval $[x]$ are not satisfactory, heuristics can be used to search for improved significance results in sub-intervals of $[x]$ by sampling.

For an increasing number of inputs, the sub-interval approach becomes a combinatorial problem again. Since the

significance analysis can be considered as a compile time technique which can be done in parallel for several sub-intervals, this can be tolerated.

Limitations of Interval valued Derivatives: Note that differentiability is defined for **differentiable** functions only:

- Integer - based programs need to be checked carefully to determine if integer arithmetic can be replaced by floating point arithmetic.
- For some application kernels within SCoRPiO, integer data are converted into floating point data for non-integer calculations. The result is then converted back to integer by rounding, flooring, or ceiling, which are discontinuous operations. Replacing integer arithmetic with floating point arithmetic preserves these discontinuities. We have to investigate in detail how these discontinuous operations can be evaluated in differentiated interval arithmetic.

D. Monte Carlo Simulation and Exhaustive Search

Monte Carlo methods (MC) do not require differentiable functions, however they can not handle intermediate variables efficiently. In any case, such methods can serve as a backup option for cases where the substitution of integer arithmetic by floating point arithmetic is not possible (leads to invalid results), and also as a verification tool.

A simple framework for MC - simulations was created and used to compute enclosures $[y] = f[x]$ and $\nabla_{[x]}[y]$ so that the derivative based criterion (5) can be applied too.

If the number of inputs increases, Monte Carlo methods and exhaustive search become combinatorial problems as well, without limiting the applicability of the methods, since they are applied at compile-time and can be accelerated by exploiting parallelism.

III. Definition of Significance

This section introduces significance of numerical models as a mathematical concept [6], and the significance model developed for SCoRPiO (that is work in progress).

Definition 1 (Significance for a scalar valued function): Let $f : \mathbf{X} \longrightarrow Y : y = f(\mathbf{x}), \mathbf{X} \subseteq I\!R^n, y \subseteq I\!R$, be a differentiable scalar valued function, $[\mathbf{x}] \in [I\!R]^n$ be an input interval vector, and $[y] = f[\mathbf{x}] \in [I\!R]$ the image of the interval function $f[\mathbf{x}]$ for $[\mathbf{x}]$. Moreover, let $\epsilon \in I\!R$ be an user-defined significance bound.

Let $f_{(1)}([\mathbf{x}], [y], [y]_{(1)})$ be an interval evaluation of a first order adjoint model of f obtained by AD.

By setting the initial adjoint $[y]_{(1)}$ to the point interval $[1]$, an evaluation of

$$f_{(1)}([\mathbf{x}], [y], [1]) \tag{6}$$

yields

$$[\mathbf{x}]_{(1)} = \nabla_{[\mathbf{x}]}[y] = \nabla_{[\mathbf{x}]} f[\mathbf{x}], \tag{7}$$

that is the interval gradient $\nabla_{[x]}[y]$ of $f[\mathbf{x}]$ at the evaluation interval $[\mathbf{x}]$.

Table I
ALTERNATIVE SIGNIFICANCE CRITERIA

Criteria	Comment
$w([v] \cdot \nabla_{[v]}[y])$	width of $\nabla_{[v]}[y]$ scaled over $[v]$
$w(\nabla_{[v]}[y])$	width of $\nabla_{[v]}[y]$, measures variation of $\nabla_{[v]}[y]$ of over $[v]$
$\lvert\nabla_{[v]}[y]\rvert \cdot \frac{w[y]}{w[v]}$	max absolute rate of change scaled with ratio of output width to input width

A scalar input interval $[x_i]$,$1 \leq i \leq n$, is called **significant** for the scalar output interval $[y]$ if

$$w([x_i] \cdot (\nabla_{[\mathbf{x}]} f[\mathbf{x}])_i) = w([x_i] \cdot [\mathbf{x}]_{(1),i}) > \epsilon. \tag{8}$$

The complete input interval vector $[\mathbf{x}]$ is called **significant** for the scalar output y if

$$\max(w([\mathbf{x}] \bullet \nabla_{[\mathbf{x}]} f[\mathbf{x}])) = \max(w([\mathbf{x}] \bullet [\mathbf{x}]_{(1)})) > \epsilon, \tag{9}$$

where $a \bullet b$ denote the element-wise multiplication of two vectors $a, b \in [I\!R]^n$.

Example 1 (Simple example): Let $y = f[\mathbf{x}] = \frac{x}{1000}$, $[x] = [1, 3]$.

Interval evaluation : $y = f[x] = [\frac{1}{1000}, \frac{3}{1000}]$
Interval Gradient : $\nabla_{[x]} f[x] = 0.001$
Significance test:
$$w([x] \cdot \nabla_{[x]} f[x]) = w([1, 3] \cdot 0.001) = 0.002 > \epsilon$$
Conclusion: If $\epsilon \leq 0.002$, argument $x \in [1, 3]$ of function f **is significant**, if $\epsilon > 0.002$, argument $x \in [1, 3]$ of function f **is not significant**.

Note 1 (What is an input variable?): After the adjoint propagation phase the interval valued adjoint model created by `dco/scorpio` contains the derivative $\nabla_{[v_j]}[y]$ of output $[y]$ with respect to *all* intermediate program variables v_j, $j = 1, \ldots, k$, in the internal data structures of `dco/scorpio`. [3]

Thus the significance definition can be applied to arbitrary program variables v_j,$j = 1, \ldots, k$, appearing in the implementation of a differentiable scalar valued function $f : \mathbf{X} \longrightarrow Y : y = f(\mathbf{x}), \mathbf{X} \subseteq I\!R^n$, $y \subseteq I\!R$, for a given input interval vector $[\mathbf{x}] \in [I\!R]^n$:

An intermediate program variable v_j, $1 \leq j \leq k$, is called **significant** for the scalar output interval $[y]$ if

$$w([v_j] \cdot \nabla_{[v_j]}[y]) = w([v_j] \cdot [v_{j,(1)}]) > \epsilon. \tag{10}$$

Note 2 (How to check significance?): Table I contains alternative significance criteria for program variables v, that can be used as replacements for (8) and (10).

Note 3 (Bound ϵ): The meaning of the given bound ϵ depends on the underlying application and the chosen significance criterion. Nevertheless the bound might be more complex than just a number.

[3]Compiler generated intermediates are not contained in the internal structure.

IV. Examples

This section discusses the application of the significance concept to various examples. In all cases, the inputs to the analysis are the following:

- A program (C / C++ source code) that computes a function $y = f(x)$ with intermediate variables $v1, v2, \ldots$.
- Input ranges of input variable.
- A significance/error bound ϵ.

The output of the analysis is a list of program variables that are insignificant given the specified input ranges, and under the user-specified significance/error bound. If variables are overwritten in the code, the analysis associates (and reports) an instance number with each overwrite of the variable. Therefore, the significance of each variable can vary in different appearances of the variable in the code, either spatially (for example different places in the code), or temporally (for example across different loop iterations).

A. Simple Example

```
v1 = log( x1 );   v2 = v1 + x2;   y  = v2;
```

Significance criterion: $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$ with $\epsilon = 1$

Results for ranges $[x1] = [1, 2], [x2] = [1, 20]$

	$[v]$	$\nabla_{[v]}[y]$	$[v] \cdot \nabla_{[v]}[y]$
$[x1]$	[1, 2]	[0.5, 1]	[0.5, 2]
$[x2]$	[1, 20]	[1, 1]	[1, 20]
$[v1]$	[0, 0.693]	[1, 1]	[0, 0.693]
$[y]$	[1, 20.7]	[1, 1]	[1, 20.7]

Interpretation Intermediate $v1$ turns out to be insignificant over the complete input ranges of $x1$ and $x2$, since $w([v1] \cdot \nabla_{[v1]}[y]) < \epsilon$. Using the midpoint $m[v1] = .3465$ of $v1$ as constant initializer for $v1$, the code can be simplified for the complete input range of $x1$ and $x2$ to:

```
v1 = .3465;   y  = v1 + x2;
```

An interval evaluation of the modified code for ranges $[x1] = [1, 2], [x2] = [1, 20]$ gives $[y] = [1.35, 20.3]$, which is a sub-interval of the output range of the original code.

B. (In-) Significant Sub-Intervals

```
v1 = x * x;      v2 = v1 * x;    v3 = 1/v2;
v4 = sqrt(v3);   y  = v4 * x ;
```

Significance criterion: $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$ with $\epsilon = 0.04$

Results for range $[x] = [1, 36000]$

	$[v]$	$[v] \cdot \nabla_{[v]}[y]$
$[x]$	[1, 3.6e+04]	[-1.47e+25, 3.5e+04]
$[v1]$	[1, 1.3e+09]	[-4.91e+24, -2.72e-28]
$[v2]$	[1, 4.67e+13]	[-4.91e+24, -2.72e-28]
$[v3]$	[2.14e-14, 1]	[1.17e-14, 1.15e+11]
$[v4]$	[1.46e-07, 1]	[1.53e-07, 3.5e+04]
$[y]$	[1.46e-07, 3.6e+04]	[1.53e-07, 3.5e+04]

Interpretation For the input range $[1, 36000]$, all intermediate and the input variable are significant and have to be evaluated without error. This is because $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$ for all $v \in \{v1, v2, v3, v4\}$.

Results for range $[x] = [10000, 36000]$

	$[v]$	$[v] \cdot \nabla_{[v]}[y]$
$[x]$	[1e+04, 3.6e+04]	[-17.2, 0.036]
$[v1]$	[1e+08, 1.3e+09]	[-5.74, -2.3e-06]
$[v2]$	[1e+12, 4.67e+13]	[-5.74, -2.3e-06]
$[v3]$	[2.14e-14, 1e-12]	[0.000107, 0.123]
$[v4]$	[1.46e-07, 1e-06]	[0.00146, 0.036]
$[y]$	[0.00146, 0.036]	[0.00146, 0.036]

Interpretation
For $[x] = [10000, 36000]$ intermediate variable $v4$ turns out to be insignificant, since $w([v] \cdot \nabla_{[v4]}[y]) = 0.034536 < \epsilon$, which is a sub-interval of the original output range.
Opting to replace code fragments with a constant, a modified code might be:

```
ASSERT(  x < 10000 ) {
  v1 = x * x;      v2 = v1 * x;   v3 = 1/v2; }
  v4 = sqrt(v3);   y  = v4 * x ;
ASSERT(  x > 10000 ) {
  v4 = sqrt(5.11e-13);   y  = v4 * x ;       }
```

Evaluating the modified code for $[x] = [10000, 36000]$ in interval arithmetic results to $[y] = [0.00715, 0.0257]$, which is a sub-interval of the output range of the original code.

C. A slightly more complex example

```
v1 = x1 * x2;   v2 = sin(x1);   v3 = exp(v2);
v4 = v3 * v1;   v5 = log(v4);
y  = v5 / 10 + x2 / 100;
```

Significance criterion: $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$ with $\epsilon = 2.5$
Results for ranges $[x1] = [1, 36000], [x2] = [1, 36000]$

	$[v]$	$[v] \cdot \nabla_{[v]}[y]$
$[x1]$	[1, 3.6e+04]	[-3.45e+13, 3.45e+13]
$[x2]$	[1, 3.6e+04]	[0.01, 9.58e+08]
$[v1]$	[1, 1.3e+09]	[1.04e-11, 9.58e+08]
$[v2]$	[-1, 1]	[-9.58e+08, 9.58e+08]
$[v3]$	[0.368, 2.72]	[1.04e-11, 9.58e+08]
$[v4]$	[0.368, 3.52e+09]	[1.04e-11, 9.58e+08]
$[v5]$	[-1, 22]	[-0.1, 2.2]
$[y]$	[-0.09, 362]	[-0.09, 362]

Interpretation Intermediate $v5$ turns out to be insignificant over the complete input ranges of $x1$ and $x2$, since $w(\nabla_{[v5]}[y] \cdot [v5]) < \epsilon$. Using the midpoint $m(v5) = 10.5$ of $v5$ as constant initializer for $v5$, the code can be simplified for the complete input range of $x1$ and $x2$ to:

```
v5 = 10.5;   y = v5 / 10 + x2 / 100;
```

Evaluating the modified code for ranges $[x1] = [1, 36000]$, $[x2] = [1, 36000]$ in interval arithmetic results to $[y] = [1.06, 361]$, a sub-interval of the original output range.

D. Re-use Previous Results for Overwritten Variables

```
v1 = x1*x2;   v2{0} = sin(x1);   v3 = exp(v2);
v4 = v3*v1;   v2{1} = log(v4);
y  = v2/10 + x2/100;
```

Significance criterion: $w([v] \cdot \nabla_{[v]}[y]) > \epsilon$ with $\epsilon = 2.5$
Results for ranges $[x1] = [1, 36000], [x2] = [1, 36000]$

	$[v]$	$[v] \cdot \nabla_{[v]}[y]$
$[x1]$	[1, 3.6e+04]	[-3.45e+13, 3.45e+13]
$[x2]$	[1, 3.6e+04]	[0.01, 9.58e+08]
$[v1]$	[1, 1.3e+09]	[1.04e-11, 9.58e+08]
$[v2^{\{0\}}]$	[1.0e-11, 9.6e+08]	[-9.58e+08, 9.58e+08]
$[v3]$	[0.368, 2.72]	[1.04e-11, 9.58e+08]
$[v4]$	[0.368, 3.52e+09]	[1.04e-11, 9.58e+08]
$[v2^{\{1\}}]$	[-1, 22]	[-0.1, 2.2]
$[y]$	[-0.09, 362]	[-0.09, 362]

Interpretation The second value $[v2^{\{1\}}]$ of intermediate $v2$ turns out to be insignificant over the complete input ranges of $x1$ and $x2$, since $w([v2^{\{1\}}] \cdot \nabla_{[v2^{\{1\}}]}[y]) < \epsilon$. Since $[v2^{\{0\}}] \subseteq [v2^{\{1\}}]$, the first value $[v2^{\{0\}}]$ of intermediate $v2$ can be used instead. Thus intermediates $v3$, $v4$, and $[v2^{\{1\}}]$ does not need to be computed. The code can be simplified for the complete input range of $x1$ and $x2$ to:

```
v1=x1*x2;   v2=sin(x1);   y=v2/10+x2/100;
```

The modified code computes for $[x1] = [1, 36000]$, $[x2] = [1, 36000]$ in interval arithmetic $[y] = [-0.09, 360]$ (which is a sub-interval of the output range of the original code):

	$[v]$	$[v] \cdot \nabla_{[v]}[y]$
$[y]$	[-0.09, 362]	[-0.09, 362]
$[x1]$	[1, 3.6e+04]	[-3.6e+03, 3.6e+03]
$[x2]$	[1, 3.6e+04]	[0.01, 360]
$[v1]$	[1, 1.3e+09]	[0, 0]
$[v2^{\{0\}}]$	[-1, 1] [-0.1, 0.1]	
$[y]$	[-0.09, 360]	[-0.09, 360]

E. Real world test cases

We applied our prototype implementation of the adjoint based significance analysis to some more complex test cases such as a mid/small size CFD code and a financial application (European call option pricing, solving either a partial differential equation or via Monte Carlo simulation). Unfortunately, (iterative) solvers involved accelerated the wrapping effect resulting in unbounded interval values of program variables v_j destroying any insignificance. First experiments with a *recursive interval splitting algorithm* for variables v_j that exceed the given significance bound, look promising. A semi-automatic parallel analysis based in that approach is under development.

V. Summary and Outlook

We have introduced a formal definition of significance, as well as significance criteria based on interval valued adjoints. At this point we are extending our prototype implementation of the adjoint based significance analysis to handle larger codes by parallel recursive interval splitting as described in section IV-E. By doing that, new insight into the nature of the associated significance and error bounds will be gained for more general test cases.

The influence of overestimation by interval evaluations should be investigated by computing alternative enclosures based on affine arithmetic [3], slopes [11], or Monte Carlo simulations for selected kernels.
Moreover, for integer codes, such as the multimedia kernels, the role of discontinuities has to be investigated by comparing the results attained by AD with those produced from sampling or Monte Carlo methods.

Acknowledgement

The authors wish to thank all SCoRPiO project partners for fruitful discussions, especially the project coordinator Nikolaos Belas.

References

[1] R. E. Moore, *Interval Analysis*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1966.

[2] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, 1st ed. Society for Industrial and Applied Mathematics, 1 2009. [Online]. Available: http://amazon.com/o/ASIN/0898716691/

[3] L. H. de Figueiredo and J. Stolfi, "Affine arithmetic: Concepts and applications," *Numerical Algorithms*, vol. 37, no. 1-4, pp. 147–158, 2004. [Online]. Available: http://dx.doi.org/10.1023/B:NUMA.0000049462.70970.b6

[4] U. Naumann, *The Art of Differentiating Computer Programs. An Introduction to Algorithmic Differentiation.*, ser. Software, Environments, and Tools. SIAM, 2011.

[5] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed. SIAM, 2008.

[6] J. Riehme and U. Naumann, "D1.1: Significance Based Computing Modeling," RWTH Aachen, Tech. Rep., June 2014. [Online]. Available: www.scorpio-project.eu/wp-content/uploads/2014/07/Scorpio_D1.1.pdf

[7] Software and Tools for Scientific Engineering, RWTH Aachen University, Germany, "Derivative Code by Overloading in C++ (`dco/c++`)," http://www.stce.rwth-aachen.de/software/dco_cpp.html.

[8] J. Lotz, K. Leppkes, and U. Naumann, "`dco/c++` - Derivative Code by Overloading in C++," RWTH Aachen, Tech. Rep. AIB-2011-06, May 2011. [Online]. Available: http://aib.informatik.rwth-aachen.de/2011/2011-06.ps.gz

[9] J. Lotz, U. Naumann, and J. Ungermann, "Hierarchical algorithmic differentiation: A case study," in *Recent Advances in Algorithmic Differetiation*. Springer, 2012, pp. 187–196.

[10] L. Rall and G. Corliss, "Automatic differentiation: point and interval automatic differentiation: Point and interval," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Springer US, 2009, pp. 165–170. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-74759-0_28

[11] H. Muñoz and R. B. Kearfott, "Slope intervals, generalized gradients, semigradients, slant derivatives, and csets," *Reliable Computing*, vol. 4, no. 3, pp. 163–193, 2004.