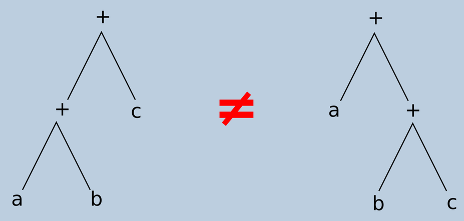
Numerical Accuracy Improvement of programs: Principles and Experiments

Nasrine Damouche [1], Matthieu Martel [1] & Alexandre Chapoutot [2]

[1]**University of Perpignan** (Laboratoire LAMPS), 52 av Paul Alduy, 66860 PERPIGNAN Cedex 9
[2]**Paris-Saclay University** (U2IS, ENSTA ParisTech,), 828 bd des Maréchaux, 91762 Palaiseau cedex France
nasrine.damouche@univ-perp.fr, matthieu.martel@univ-perp.fr, alexandre.chapoutot@ensta-paristech.fr

Problem

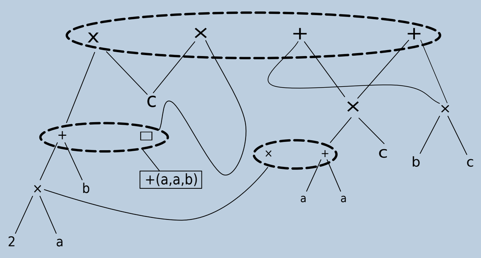
In general, the correctness of numerical computations of programs [4] based on floating-point arithmetic is not intuitive [1] and developers expect to compute an accurate result without guaranty.



To solve this problem, we proceed by automatic source to source transformation of programs to improve their numerical accuracy.

Transforming Expressions

For automatic transformation of single arithmetic expressions, several techniques have already been proposed [6]. Among them, the APEG (Abstract Program Expression Graph) introduce a new intermediary representation that manipulates in a single data structure a large set of equivalent arithmetic expressions [5]. Basically, APEG do not duplicate the common parts of the syntactic trees of mathematically equivalent expressions. The APEG corresponding to the expression $e = (a + a) + b * c$. An expression is built from an APEG by selecting one operator in each dotted box (see figure above) and one arbitrary parsing for each box (e.g. $\boxed{+(a,a,b)}$).



Transforming Commands

The transformation uses an environment δ which maps identifiers to formal expression, a black list β of identifiers and a target variable μ to be optimized. Programs are assumed to be given SSA form and Φ-nodes are associated to conditionals and loops.

Assignments $c \equiv id = e$

- If the following conditions are satisfied ⇒ remove the assignment from the program and memorize $e[id \mapsto e]$ in δ.
  - the variables of e do not exist in δ,
  - $v \notin \beta$ and $v \neq \nu$,
- Otherwise, we inline the variables saved in δ in the concerned expression e and we transform the resulting expression.

Sequences $c \equiv c_1; c_2$

- If c_1 or c_2 is nop ⇒ rewrite the other member of the sequence,
- Otherwise, rewrite both members of the sequence.

Conditionals $c \equiv \text{if}_\Phi\ e$ then c_1 else c_2

- If e is true ⇒ rewrite c_1,
- If e is false ⇒ rewrite c_2,
- If e is statically unknown ⇒ rewrite both branches of the conditionals,
- If the variables of e have been removed from the program ⇒ re-insert them into the source code and then rewrite it (the variables of e are inserted in the black list β).

While Loops $c \equiv \text{while}_\Phi\ e$ do c

- Rewrite the body of the loop,
- re-inject the variables discarded into the program and then rewrite it (again the concerned variables are added to β).

Experiments and Results

To illustrate the improvement of numerical accuracy, we have taken an example from robotics. It computes the position of a two wheeled robot by odometry [3]. Note that s_l and s_r are the instantaneous rotation speeds of the left and right wheels, c is the circumference of the wheels and l the length of its axle. The left Figure represents the initial program of the odometry while the right one gives the transformed code.

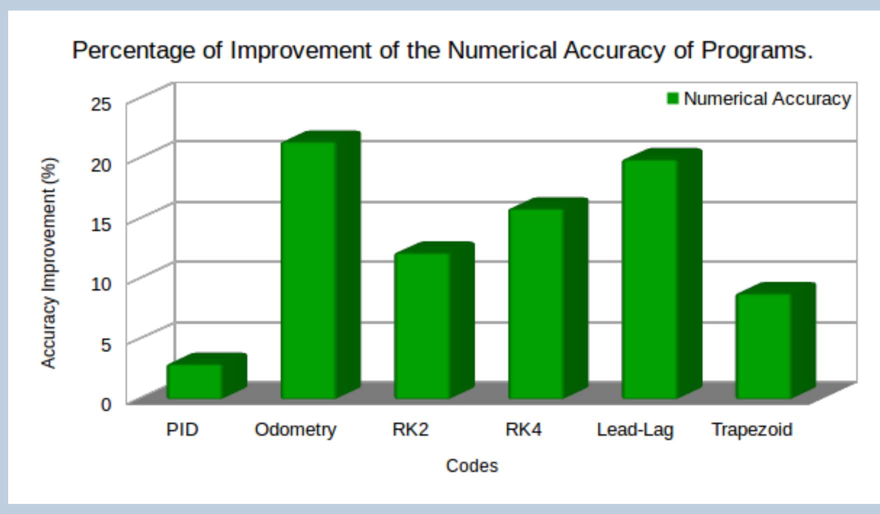
```
sl = [0.52,0.53];
sr = 0.785398163397; c = 12.34;
theta = t = x = y = 0.0 ; inv_l = 0.1 ;
while (t < 100.0 ) do {
  delta_dl = (c * sl);
  delta_dr = (c * sr);
  delta_d = ((delta_dl + delta_dr) * 0.5);
  delta_theta = ((delta_dr - delta_dl) * inv_l);
  arg = (theta + (delta_theta * 0.5));
  cos = (1.0 - ((arg * arg) * 0.5))
       + ((((arg * arg)* arg)* arg) / 24.0);
  x = (x + (delta_d * cos));
  sin = (arg - (((arg * arg)* arg)/6.0))
       + (((((arg * arg)* arg)* arg)* arg)/120.0);
  y = (y + (delta_d * sin));
  theta = (theta + delta_theta);
  t = (t + 0.1);}
```

Listing of the initial Odometry program.

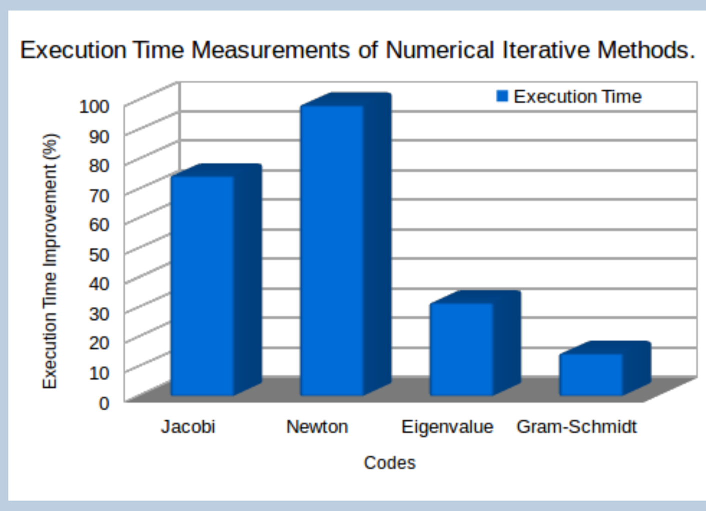
```
sl = [0.52,0.53]; theta = t = x = y = 0.0;
while (t < 100.0) do {
 TMP_6 = (0.1 * (0.5 * (9.691813336318980 - (12.34 * sl))));
 TMP_23 = ((theta + (((9.691813336318980 - (sl * 12.34))
          * 0.1) * 0.5)) * (theta + ((( 9 691813336318980
          - (sl * 12.34)) * 0.1) * 0.5)));
 TMP_25 = ((theta + TMP_6) * (theta + TMP_6)) * (theta
          + (((9.691813336318980 - (sl * 12.34)) * 0.1) * 0.5));
 TMP_26 = (theta + TMP_6) ;
 x = ((0.5 * (((1.0 - (TMP_23 * 0.5)) + ((TMP_25 * TMP_26)
   / 24.0)) * ((12.34 * sl) + 9.691813336318980))) + x);
 TMP_27 = ((TMP_26 * TMP_26) * (theta
          + (((9.691813336318980 - (sl * 12.34)) * 0.1) * 0.5)));
 TMP_29 = (((TMP_26 * TMP_26) * TMP_26) * (theta
          + (((9.691813336318980 - (sl * 12.34)) * 0.1) * 0.5)));
 y = (((9.691813336318980 + (12.34 * sl)) * (((TMP_26
   - (TMP_27 / 6.0)) + ((TMP_29 * TMP_26) / 120.0)) * 0.5)) + y);
 theta = (theta + (0.1 * (9.691813336318980 - (12.34 * sl))));
 t = t + 0.1;}
```

Listing of the transformed Odometry program.

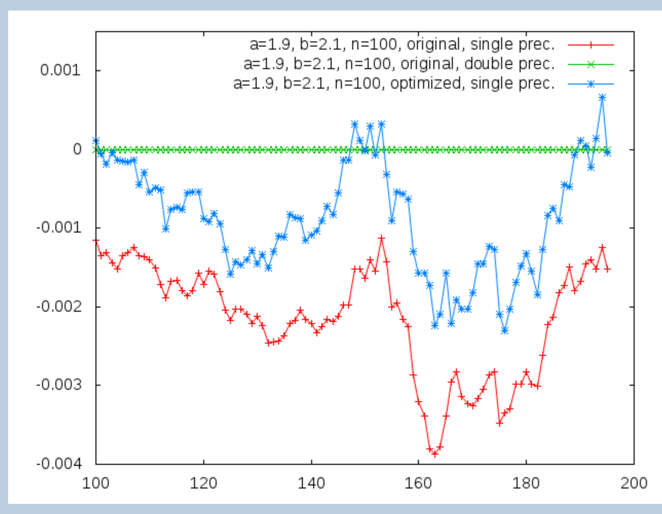
For the demonstration of its efficiency, we evaluate through our tool a set of programs in the domains of robotics, avionics, mathematics, etc. The experimental results show the efficiency of our approach to optimize the numerical accuracy. We compare the initial and the new error of each of programs (see Figure (a)). We have also demonstrated [2] that improving the numerical accuracy accelerates the convergence time of numerical iterative algorithms (see Figure (b)). Finally, we have compared initial codes working in double precision with optimized codes in simple precision (see figure (c)).



(a)



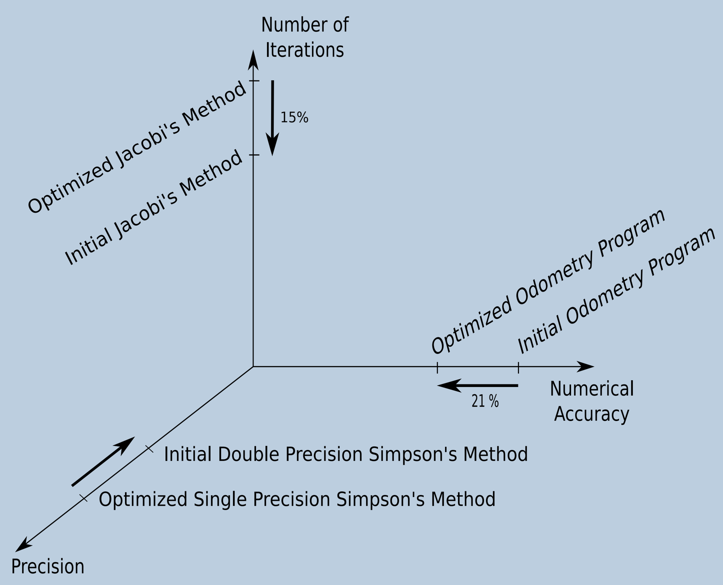
(b)



(c)

Conclusion and Perspectives

- Improve numerical accuracy of programs by up to 20%,
- Accelerate convergence of numerical iterative methods up to 15%,
- Transformed programs are close to programs with double precision by up to 99%.
- Extend this work to deal with other or more complicated structures of programs such that functions and arrays.
- Validate that programs transformed by our tool are equivalent to the original programs using the Coq proof assistant.
- Optimize the numerical accuracy of parallel programs.



References

[1] ANSI/IEEE. *IEEE Standard for Binary Floating-point Arithmetic*, std 754-2008 edition, 2008.

[2] N. Damouche, M. Martel, and A. Chapoutot. Impact of accuracy optimization on the convergence of numerical iterative methods. In Moreno Falaschi, editor, *LOPSTR'15*, volume LNCS 9527 of *LNCS*. Springer, 2015.

[3] N. Damouche, M. Martel, and A. Chapoutot. Intra-procedural optimization of the numerical accuracy of programs. In M.Núñez and M. Güdemann, editors, *FMICS'15*, volume 9128 of *LNCS*, pages 31–46. Springer, 2015.

[4] E. Darulova and V. Kuncak. Sound compilation of reals. In *POPL*, 2014.

[5] A. Ioualalen and M. Martel. A new abstract domain for the representation of mathematically equivalent expressions. In *SAS'12*, volume 7460 of *LNCS*, pages 75–93. Springer, 2012.

[6] J. R. Wilcox P. Panchekha, A. Sanchez-Stern and Z. Tatlock. Automatically improving accuracy for floating point expressions. In *PLDI*, 2015.