# Traitements Numériques

# pour les Systèmes Embarqués

## 3) Optimisation de la précision des commandes

Matthieu Martel

UPVD - LAMPS

matthieu.martel@univ-perp.fr.fr

# Introduction

Floating-point computations sensitive to how formulas are written

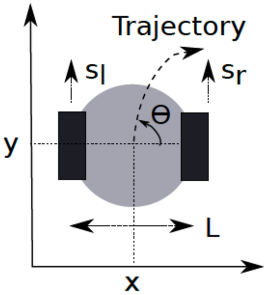
F.P. arithmetic not intuitive. Ex: in OCaml 4.01 (Ubuntu, Core I5):

```
# let f x = x ** 2.0 -. 2.0 *. x +. 1.0 ;;
# let g x = (x -. 1.0) *. (x -. 1.0) ;;
# f 0.99;;
- : float = 9.99999999999889866e-05
# g 0.99;;
- : float = 0.000100000000000000181
```

**Goal:** optimize accuracy almost like compilers do for exec. time

Main difference: ranges are given for the inputs of the programs

# Example: Odometry




$$\theta(t+1) = \theta(t) + \Delta\theta(t)$$

$$\Delta d(t) = \frac{\Delta d_r(t) + \Delta d_l(t)}{2}$$

$$\Delta\theta(t) = \frac{\Delta d_r(t) - \Delta d_l(t)}{L}$$

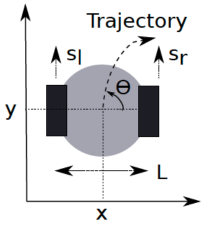
$$x(t+1) = x(t) + \Delta d(t+1) \times \cos\left(\theta(t) + \frac{\Delta\theta(t+1)}{2}\right)$$

$$\Delta d_l(t) = s_l(t) \times C \qquad \Delta d_r(t) = s_r(t) \times C$$

# Example: Odometry

## Source program:

```
sl = [0.52,0.53] ; sr = 0.785398163397 ;
theta = 0.0 ; t = 0.0 ; x = 0.0 ;
y = 0.0 ; inv_l = 0.1 ; c = 12.34 ;
while (t < 1.5) 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)) ;
  theta = (theta + delta_theta) ;
  t = (t + 0.1)
}
```

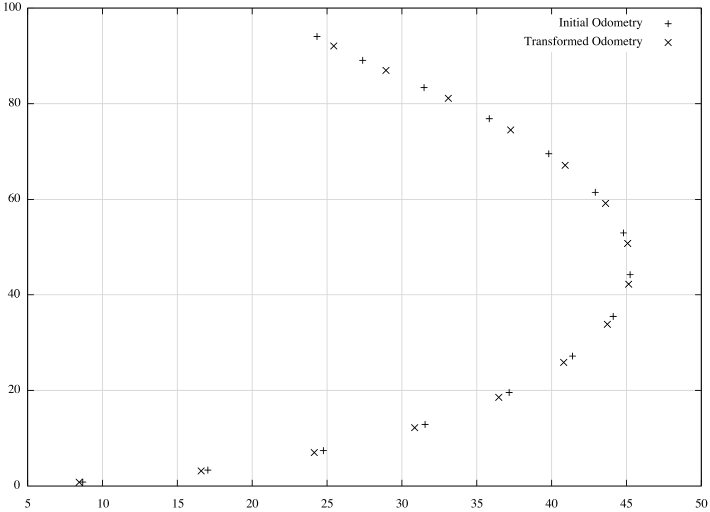


# Example: Odometry

## Optimized program:

```
sl = [0.52,0.53] ; sr = 0.785398163397 ;
theta = 0.0 ; y = 0.0 ; x = 0.0 ; t = 0.0 ;
while (t < 1.5) do {
   x = (x + ((0.5 * ((1.0 - ((theta + (((9.69181333632
     - (sl * 12.34)) * 0.1) * 0.5)) * (0.5 * ((0.5 * (0.1
     * (9.69181333632 - (sl * 12.34))))+ theta))))
     + (((((theta + (((9.69181333632- (sl * 12.34)) * 0.1)
     * 0.5)) * (theta + (((9.69181333632 - (sl * 12.34))
     * 0.1)* 0.5))) * (theta + (((9.69181333632 - (sl * 12.34))
     * 0.1) * 0.5)))* (theta + (((9.69181333632 - (sl * 12.34))
     * 0.1) * 0.5))) / 24.0)))*(9.69181333632 + (sl * 12.34))));
   theta = (theta + (0.1 * (9.69181333632 - (sl * 12.34)))) ;
   t = (t + 0.1)
}
```

# Example: Odometry



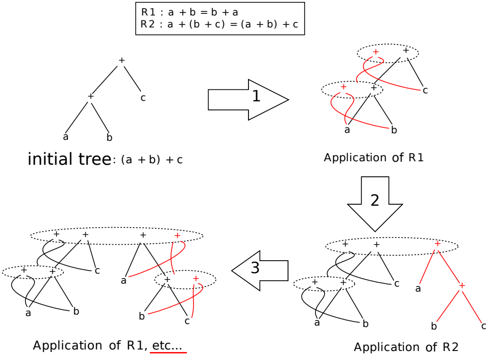
# Equivalent Program Expansion Graphs

Introduced for the *phase ordering problem*

Based on rewriting rules

Notion of equivalence class

[R. Tate, M. Stepp, Z. Tatlock and S. Lerner, *Equality Saturation: A New Approach to Optimization*, POPL'09]



# From EPEGs to APEGs

$(2n-1)!!$

EPEGs are exponential in size and possibly infinite (ex : $a = 1 \times a$)

APEG = Abstract Program Equivalence Graphs

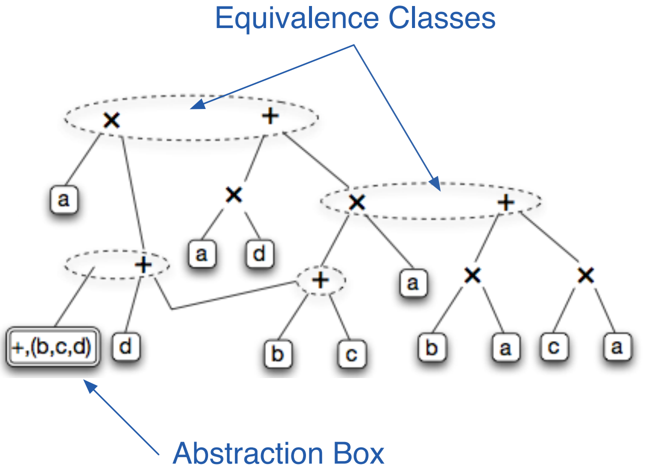
Represent many equivalent expressions in polynomial size

APEGs contain equivalence classes like EPEGs

APEGs contain abstraction boxes: $\boxed{*(e_1, \ldots, e_n)}$ ($*$ associative)

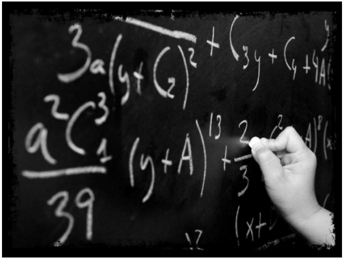
$\boxed{*(e_1, \ldots, e_n)}$ represents all the parsings of $e_1 * \ldots * e_n$

# Example of APEG



# Expression Transformation

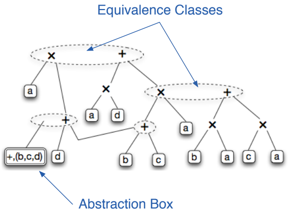
1 - **Construction of an APEG**



2 - Extraction of an expression optimizing the accuracy

# Insertion of New Nodes

In polynomial size :




Distribute products (largest factors)

Factorize every product (largest factors)

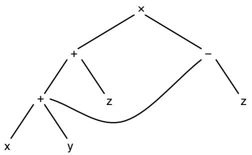
Propagate subtractions

Create a box for the left/right hand side of the operator

Create a box for the englobing homogeneous expression

Merges nested boxes

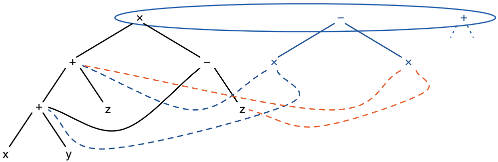
# Example (1/4)




$$((x + y) + z) \times ((x + y) - z)$$

# Example (2/4)

## Distribute:



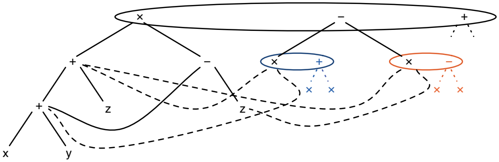

$$((x+y)+z)\times((x+y)-z)$$

$$((x+y)+z)\times(x+y)-((x+y)+z)\times z$$

$$(x+y)\times((x+y)-z)+z\times((x+y)-z)$$

# Example (3/4)

## Distribute:

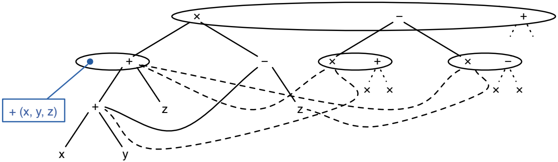



$$((x+y)+z) \times ((x+y)-z)$$
$$((x+y)+z) \times (x+y) - ((x+y)+z) \times z$$
$$(x+y) \times ((x+y)-z) + z \times ((x+y)-z)$$

$$(((x+y)+z) \times x + ((x+y)+z) \times y) - ((x+y)+z) \times z$$
$$(((x+y) \times (x+y))((x+y) \times z)) - ((x+y)+z) \times z$$
$$((x+y)+z) \times (x+y) - ((x+y) \times z + z \times z)$$
$$(((x+y)+z) \times x + ((x+y)+z) \times y)((x+y) \times z + z \times z)$$
$$(((x+y) \times (x+y))((x+y) \times z)) - ((x+y) \times z + z \times z)$$

# Example (4/4)

Box introduction: $\boxed{+(\text{x,y,z})} = \{(x+y)+z,\ x+(y+z),\ (x+z)+y\}$




$\boxed{+(\text{x,y,z})} \times ((x+y)-z)$

$\boxed{+(\text{x,y,z})} \times (x+y) - ((x+y)+z) \times z$

$(x+y) \times ((x+y)-z) + z \times ((x+y)-z)$

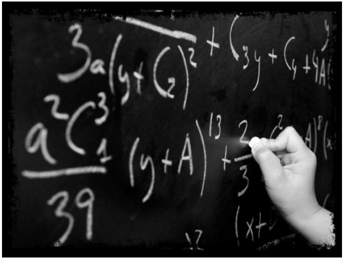
$(\boxed{+(\text{x,y,z})} \times x + \boxed{+(\text{x,y,z})} \times y) - \boxed{+(\text{x,y,z})} \times z$

$(((x+y)\times(x+y))((x+y)\times z)) - \boxed{+(\text{x,y,z})} \times z$

$\boxed{+(\text{x,y,z})} \times (x+y) - ((x+y) \times z + z \times z)$

$(\boxed{+(\text{x,y,z})} \times x + \boxed{+(\text{x,y,z})} \times y) - ((x+y) \times z + z \times z)$

# Expression Transformation



1 - Construction of an APEG

2 - **Extraction of an expression optimizing the accuracy**

# Formula Synthesis: The Case of Boxes

An abstraction box represents $(2n-1)!!$ expressions

Greedy heuristic - Complexity: $O(n^2)$

At each step, select terms $a$ and $b$ such that $\downarrow(a * b)$ is minimal

Example:

$$\boxed{+(\text{a,b,c,d,e})} \rightarrow \boxed{+(\text{a,c,e,}\boxed{+(\text{b,d})})} \rightarrow \boxed{+(\text{e,}\boxed{+(\text{a,c})},\boxed{+(\text{b,d})})}$$

$$\rightarrow \boxed{+(\boxed{+(\text{e,}\boxed{+(\text{a,c})})},\boxed{+(\text{b,d})})}$$

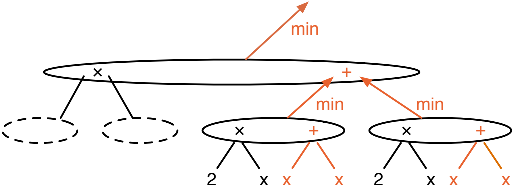
We synthetize $(e + (a + c)) + (b + d)$

# Formula Synthesis: Equivalence Classes

**Simplest approach:**

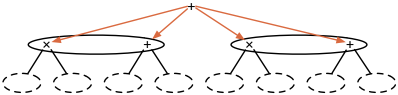
For each class, select the operation which yields the smallest error

Complexity: $O(n)$



# Formula Synthesis: Improvement

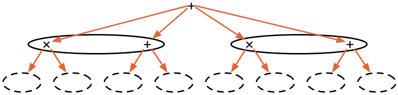
Minimize the error for one operator using the classes below




We compute the errors on:

$(a+a)+(b+b)$ $(a+a)+(2\times b)$ $(2\times a)+(b+b)$ $(2\times a)+(2\times b)$

**Generalization:** Consider all the classes up to $k$ levels



# Error Bound Computation

## Elementary operations: (current rounding mode $= \circ_\sim$)

$$
\begin{array}{cccc}
 & [x_1] & , & [\epsilon_1] \\
+ & [x_2] & , & [\epsilon_2] \\
\hline
= & [x_1] \oplus_\sim [x_2] & , & [\epsilon_1] \oplus_\leftrightarrow [\epsilon_2] \\
 & & & \oplus_\leftrightarrow \\
 & & & [\pm \frac{1}{2} ulp([x_1] +_\sim [x_2])]
\end{array}
$$

$$
\begin{array}{cccc}
 & [x_1] & , & [\epsilon_1] \\
\times & [x_2] & , & [\epsilon_2] \\
\hline
= & [x_1] \otimes_\sim [x_2] & , & [\epsilon_1] \otimes_\leftrightarrow [x_2] \\
 & & & \oplus_\leftrightarrow \\
 & & & [\epsilon_2] \otimes_\leftrightarrow [x_1] \\
 & & & \oplus_\leftrightarrow \\
 & & & [\epsilon_1] \otimes_\leftrightarrow [\epsilon_2] \\
 & & & \oplus_\leftrightarrow \\
 & & & [\pm \frac{1}{2} ulp([x_1] \otimes_\sim [x_2])]
\end{array}
$$

```
-] 0.1
      float64:  [9.999999999999999E-2,1.000000000000001E-1]
        error:  [-6.938893903907228E-18,6.938893903907229E-18]
-] 0.2
      float64:  [1.999999999999999E-1,2.000000000000001E-1]
        error:  [-1.387778780781445E-17,1.387778780781446E-17]
-] 0.1+0.2
      float64:  [2.999999999999999E-1, 3.000000000000001E-1]

        error:  [-4.857225732735059E-17, 4.857225732735060E-17]
```

# Overview

Simple imperative language

$$
\begin{aligned}
\mathsf{Cmd} \ni c \quad ::= \quad & \mathtt{x} \ := e \mid c \ ; \ c \mid \mathtt{END} \\
& \mid \mathtt{if}\ b\ \mathtt{then}\ c\ \mathtt{else}\ c \\
& \mid \mathtt{while}\ b\ \mathtt{do}\ c
\end{aligned}
$$

Programs terminate by `END`

Single static assignments

Same variables written in both branches of a conditionnal

# Transformation Rules

$$\mathsf{FEnv} \;:\; \mathsf{Id} \to \mathsf{Cmd} \qquad \mathsf{VEnv} \;:\; \mathsf{Id} \to \mathsf{Val}^\sharp$$

$$\begin{array}{ccccccccccc} \mathcal{T} & : & \mathsf{Cmd} & \times & \mathsf{FEnv} & \times & \wp(\mathsf{Id}) & \times & \mathsf{VEnv} & \to & \mathsf{Cmd} \\ & & (c & , & \varrho & , & \lambda & , & \varsigma) & \mapsto & c_{opt} \end{array}$$

$\lambda$ set of identifiers whose accuracy has to be optimized

## Transformation Rules: Assignments

$$\mathcal{T}\big(\mathtt{x} := e\ ;\ c, \varrho, \lambda, \varsigma\big) = \mathcal{T}\big(c, \varrho[\mathtt{x} \mapsto e], \lambda, \varsigma\big)$$

$$\mathcal{T}\big(\mathtt{END}, \varrho, \lambda, \varsigma\big) = \mathtt{REIFY}(\varrho, \lambda, \varsigma)$$

$$\begin{array}{rclcl} \mathtt{REIFY}(\varrho, \lambda) & : & \mathsf{FEnv} \times \wp(\mathsf{Id}) \times \mathsf{VEnv} & \to & \mathsf{Cmd} \\ & & (\varrho, \lambda, \varsigma) & \mapsto & c \end{array}$$

$$\mathtt{REIFY}(\mathtt{x}, \varrho, \varsigma) = \mathtt{x}\ := \mathcal{R}(\mathtt{INLINE}(e, \varrho), \varsigma)$$

$\mathcal{R}(e, \varsigma)$ optimizes the expression $e$ for the environment $\varsigma$

# Transformation Rules: Conditionals

$$\mathcal{T}\big(\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ ; } c, \varrho, \lambda, \varsigma\big) =$$

$$\left|\begin{array}{l}
\texttt{REIFY}(\varrho, \texttt{VAR}(b), \varsigma) \texttt{ ;} \\
\texttt{if } b \texttt{ then } c_1' \texttt{ else } c_2' \texttt{ ;} \\
\mathcal{T}(c, \varrho, \lambda \setminus (\texttt{VAR}(b) \cup \lambda'), \varsigma) \\
\\
\textit{where}: \\
c_1' = \mathcal{T}(c_1 \texttt{ ; END}, \varrho, \lambda', \varsigma), \\
c_2' = \mathcal{T}(c_2 \texttt{ ; END}, \varrho, \lambda', \varsigma), \\
\lambda' = \texttt{WRITE}(c_1, c_2) \cap (\lambda \cup \texttt{READ}(c))
\end{array}\right.$$

# Transformation Rules: Loops

$$\mathcal{T}\big(\texttt{while}\ b\ \texttt{do}\ c_1\ ;\ c, \varrho, \lambda, \varsigma\big) =$$

$$\left|\begin{array}{l}
\texttt{REIFY}(\varrho, \texttt{VAR}(b), \varsigma)\ ; \\
\texttt{while}\ b\ \texttt{do}\ c_1'; \\
\mathcal{T}(c, \varrho, \lambda \setminus (\texttt{VAR}(b) \cup \lambda'), \varsigma) \\
\\
\textit{where}: \\
c_1' = \mathcal{T}(c_1\ ;\ \texttt{END}, \varrho, \lambda', \varsigma), \\
\lambda' = \texttt{WRITE}(c_1) \cap (\lambda \cup \texttt{READ}(c))
\end{array}\right.$$

# Transformation Rules: Functions

$$\mathcal{T}\Big(\texttt{x} \ := f(e_1, \ldots, e_n) \ ; \ c, \varrho, \lambda, \varsigma\Big) =$$

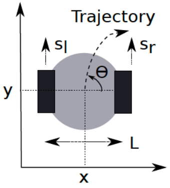
$$\left| \begin{array}{l} \texttt{REIFY}(\varrho, \texttt{VAR}(e_1) \cup \ldots \cup \texttt{VAR}(e_n), \varsigma) \ ; \\ \texttt{x} \ := f(\mathcal{R}(e_1, \varsigma), \ldots, \mathcal{R}(e_n, \varsigma)) \ ; \\ \mathcal{T}(c, \varrho, \lambda \setminus \{\texttt{x}_1, \ldots, \texttt{x}_\texttt{n}\}, \varsigma) \end{array} \right.$$

$$\mathcal{T}\Big(f(x_1, \ldots, x_n) \ \{c \ ; \ \texttt{return} \ x \ \}, \varrho, \lambda, \varsigma\Big) =$$

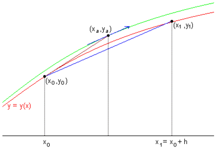
$$f(x_1, \ldots, x_n) \ \{\mathcal{T}(c, \bot, \{x\}, \varsigma); \ \texttt{return} \ x \ \}$$

# Case Studies

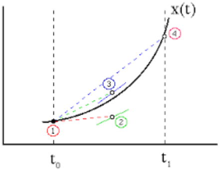
## Odometry



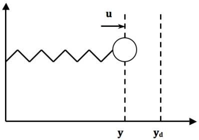

## Runge-Kutta 2



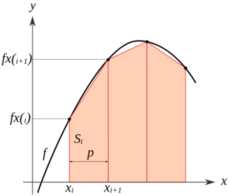

## Runge-Kutta 4



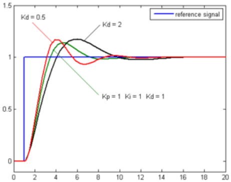

## Spring-Mass System



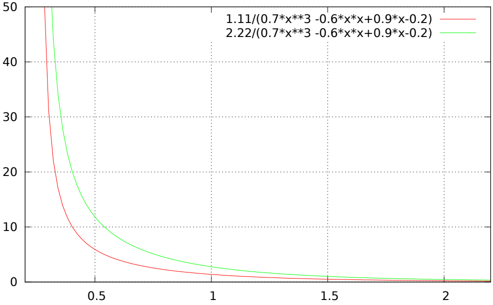

## Trapeze Method




## PID



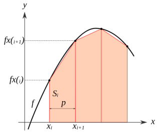
# Trapeze Method




$$I = \int_a^b f(x)dx$$

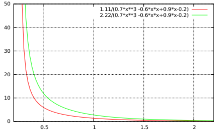
$$g(x) = \frac{u}{0.7x^3 - 0.6x^2 + 0.9x - 0.2} \qquad u \in [1.11, 2.22]$$

# Trapeze Method

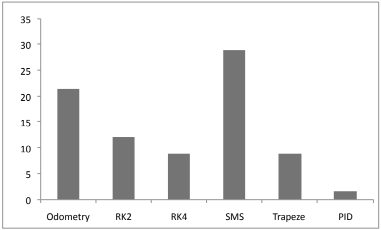



```
u = [1.11, 2.22]; a = 0.25;
b = 5000.0; n = 25.0; r = 0.0;
xa = 0.25; h = ((b - a) / n) ;
while (xa < 5000.0) do {
  xb = (xa + h) ;
  if (xb>5000.0) then xb=5000.0 ;
  gxa=(u/((((((0.7*xa)*xa)*xa)
    -((0.6*xa)*xa))+(0.9*xa))-0.2));
  gxb=(u/((((((0.7*xb)*xb)*xb)
    -((0.6*xb)*xb))+(0.9*xb))-0.2));
  r = (r+(((gxb+gxa)*0.5)*h)) ;
  xa = (xa + h)
}
```

```
u = [1.11, 2.22] xa = 0.25; r = 0.0;
while (xa < 5000.)  do {
  xb = (xa + 199.99) ;
  if (xb>5000.0) then xb=5000.0 ;
  r=(r+(99.995*((u/(((0.9*xa)
    -0.2)+(((xa*(xa*xa))*0.7)
    -((xa*xa)*0.6))))
    +(u/(((0.9*(xa+199.99))-0.2)
    +(((199.99+xa)*((199.99+xa)
    *((199.99+xa)*0.7)))-
    ((199.99+xa)*
    ((199.99+xa)*0.6))))))));
  xa = (xa + 199.99)
}
```



# Experimental Results





| Code | Initial Error | New Error |
|---|---|---|
| Odom | 0.111405189793203 e-10 | 0.881575893315158 e-11 |
| RK2 | 0.750448486755706 e-7 | 0.658915054553695 e-7 |
| RK4 | 0.201827996912328 e1 | 0.183745378465205 e1 |
| Spring | 0.395917695956325 e-13 | 0.281441385725130 e-13 |
| Trapeze | 0.536291684923368 e-9 | 0.488971110442931 e-9 |
| PID | 0.453945103062736 e-14 | 0.446837288725632 e-14 |

# Jacobi's Method

System of $n$ equations: $\mathbf{Ax} = \mathbf{b}$

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(k)}}{a_{ii}} \qquad \textit{stop when} \qquad |x_i^{(k+1)} - x_i^k| < \epsilon$$

$$\begin{pmatrix} 0.62 & 0.1 & 0.2 & -0.3 \\ 0.3 & 0.602 & -0.1 & 0.2 \\ 0.2 & -0.3 & 0.6006 & 0.1 \\ -0.1 & 0.2 & 0.3 & 0.601 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1.0/2.0 \\ 1.0/3.0 \\ 1.0/4.0 \\ 1.0/5.0 \end{pmatrix}$$

*Sufficient condition for convergence* $|a_{ii}| > \sum_{j=1, j\neq i}^{j=4} |a_{ij}|$

# Jacobi's Method

## Original program

$e = 1.0;\ eps = 10^{e-16};\ a_{11} = 0.61;\ a_{22} = 0.602;\ a_{33} = 0.6006;\ a_{44} = 0.601;$

$b_1 = 0.5;\ b_2 = 1.0/3.0;\ b_3 = 0.25;\ b_4 = 1.0/5.0;$

$while\ (e > eps)\ \{$

$\quad x_{n1} = (b_1/a_{11}) - (0.1/a_{11}) * x_2 - (0.2/a_{11}) * x_3 + (0.3/a_{11}) * x_4;$

$\quad x_{n2} = (b_2/a_{22}) - (0.3/a_{22}) * x_1 + (0.1/a_{22}) * x_3 - (0.2/a_{22}) * x_4;$

$\quad x_{n3} = (b_3/a_{33}) - (0.2/a_{33}) * x_1 + (0.3/a_{33}) * x_2 - (0.1/a_{33}) * x_4;$

$\quad x_{n4} = (b_4/a_{44}) + (0.1/a_{44}) * x_1 - (0.2/a_{44}) * x_2 - (0.3/a_{44}) * x_3;$

$\quad e = x_{n1} - x_1;$

$\quad x_1 = x_{n1};$

$\quad x_2 = x_{n2};$

$\quad x_3 = x_{n3};$

$\quad x_4 = x_{n4};$

$\}$

# Jacobi's Method

Accuracy optimized up to 44.5%

## Transformed program

$e = 1.0;\ \ eps = 10^{e-16};$
$while\ (e > eps)\ \{$
$\quad TMP_1 = (0.553709856035437 - (x_1 * 0.498338870431894));$
$\quad TMP_2 = (0.166112956810631 * x_3);$
$\quad TMP_6 = (0.333000333000333 * x_1);$
$\quad x_{n1} = (((0.819672131147541 - (0.163934426229508 * ((TMP_1 + TMP_2) - (0.332225913621263 * x_4)))) - (0.327868852459016 * (((0.416250416250416 - TMP_6) + (0.4995004995005 * x_2)) - (0.166500166500167 * x_4)))) + (0.491803278688525 * (((0.332778702163062 + (0.166389351081531 * x_1)) - (0.332778702163062 * x_2)) - (0.499168053244592 * x_3))));$
$\quad x_{n2} = (((0.553709856035437 - (0.498338870431894 * x_{n1})) + (0.166112956810631 * (((0.416250416250416 - TMP_6) + (0.4995004995005 * x_2)) - (0.166500166500167 * x_4)))) - (0.332225913621263 * (((0.332778702163062 + (0.166389351081531 * x_1)) - (0.332778702163062 * x_2)) - (0.499168053244592 * x_3))));$
$\quad x_{n3} = (((0.416250416250416 - (0.333000333000333 * x_{n1})) + (0.4995004995005 * x_{n2})) - (0.166500166500167 * (((0.332778702163062 + (0.166389351081531 * x_1)) - (0.332778702163062 * x_2)) - (0.499168053244592 * x_3))));$
$\quad x_{n4} = (((0.332778702163062 + (0.166389351081531 * x_{n1})) - (0.332778702163062 * x_{n2})) - (0.499168053244592 * x_{n3}));$
$\quad e = (x_{n4} - x_4);$
$\quad x_1 = x_{n1};$

# Jacobi's Method

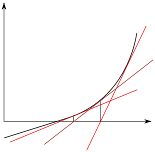
## Experimental Results

| $x_i$ | Initial it nbr | Optimized it nbr | Difference | Percentage |
|---|---|---|---|---|
| $x_1$ | 1891 | 1628 | 263 | 14.0 |
| $x_2$ | 2068 | 1702 | 366 | 17.3 |
| $x_3$ | 2019 | 1702 | 317 | 15.7 |
| $x_4$ | 1953 | 1628 | 325 | 16.7 |

Number of iterations of Jacobi's method needed before and after optimization to compute $x_i$, $1 \leq i \leq 4$

# Newton-Raphson's Method

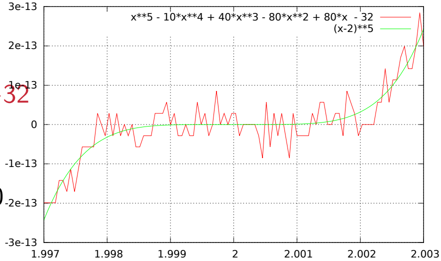
Newton's method converges quickly in $\mathbb{R}$
Problems due to floating-float errors



$$f(x) = (x-2)^5$$

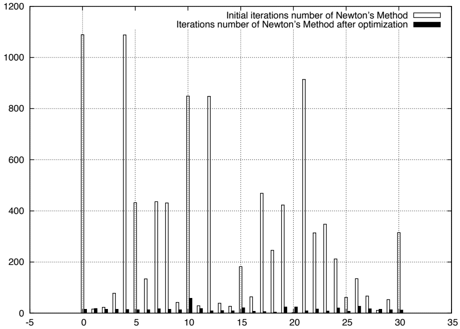
$$f(x) = x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 32$$

$$f'(x) = 5x^4 - 40x^3 + 120x^2 - 160x + 80$$



# Newton-Raphson's Method

## Experimental Results

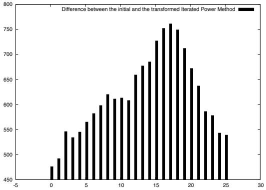



Number of iterations of the Newton-Raphson's Method before and after optimization for initial values ranging from 0 to 3 (30 runs with a step of 0.1)

## Iterated Power Method

Computes the largest eigenvalue

$$\mathbf{A} = \begin{pmatrix} \mathtt{d} & 0.01 & 0.01 & 0.01 \\ 0.01 & \mathtt{d} & 0.01 & 0.01 \\ 0.01 & 0.01 & \mathtt{d} & 0.01 \\ 0.01 & 0.01 & 0.01 & \mathtt{d} \end{pmatrix}$$

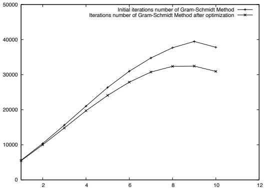



Difference between numbers of iterations of initial and optimized Iterated Power Method (tests done for $\mathtt{d} \in [175, 200]$ with a step of 1)

## Orthogonalization of a set of vectors

Numerical problems for small vectors

$$\left\{ \mathbf{q}_1 = \begin{pmatrix} 1/7\mathtt{n} \\ 0 \\ 0 \end{pmatrix}, \mathbf{q}_2 = \begin{pmatrix} 0 \\ 1/25\mathtt{n} \\ 0 \end{pmatrix}, \mathbf{q}_3 = \begin{pmatrix} 1/2592 \\ 1/2601 \\ 1/2583 \end{pmatrix} \right\}$$




Num. of it. of initial and optimized iterative GS for the family $(Q_n)_n$ of vectors, $1 \le n \le 10$.

# Performance Analysis

## Complementary results: speedups

| | Original Code Execution Time in $s$ | Optimized Code Execution Time in $s$ | Percentage Improvement | Mean on $n$ Runs |
|---|---|---|---|---|
| Jacobi | $1.49 \cdot 10^{-4}$ | $0.38 \cdot 10^{-4}$ | 74.5% | $10^4$ |
| Newton | $1.34 \cdot 10^{-3}$ | $0.02 \cdot 10^{-3}$ | 98.4% | $10^4$ |
| Eigenvalue | $4.50 \cdot 10^{-2}$ | $3.07 \cdot 10^{-2}$ | 31.6% | $10^3$ |
| Gram-Schmidt | $1.99 \cdot 10^{-1}$ | $1.70 \cdot 10^{-1}$ | 14.5% | $10^2$ |

Execution time measurements of programs

# Performance Analysis

| Method | $\sharp$ of $\pm$ per it Original Code | $\sharp$ of $\pm$ per it Optimized Code | Total $\sharp$ of $\pm$ Original Code | Total $\sharp$ of $\pm$ opt Optimized Code | Percentage of Improvement |
|---|---|---|---|---|---|
| Jacobi | 13 | 15 | 25389 | 24420 | 3.81 |
| Newton-Raphson | 11 | 11 | 3465 | 132 | 96.19 |
| Eigenvalue | 15 | 15 | 694080 | 685995 | 1.16 |
| Gram-Schmidt | 21 | 19 | 791364 | 715996 | 9.52 |
| Method | $\sharp$ of $\times$ per it Original Code | $\sharp$ of $\times$ per it Optimized Code | Total $\sharp$ of $\times$ Original Code | Total $\sharp$ of $\times$ opt Optimized Code | Percentage of Improvement |
| Jacobi | 28 | 14 | 54684 | 22792 | 58.32 |
| Newton-Raphson | 27 | 26 | 8505 | 312 | 96.33 |
| Eigenvalue | 19 | 19 | 879168 | 868927 | 1.16 |
| Gram-Schmidt | 22 | 20 | 712316 | 647560 | 9.09 |

Number of floating-point operations needed by the programs to converge

# Perspectives


Inter-procedural transformation

Lustre version

Generate correctness certificates

Improve static analysis

Estimate improvements on runs


numalis
français
english
Accueil
Le portail du calcul
Vos besoins, nos solutions
Actus du réseau
Contact
16e CONCOURS NATIONAL
Numalis est lauréat du concours "i-Lab"
12/05/2014
Le prix "i-Lab" est le Concours Nationale d'Aide à la Création d'Entreprise de Technologies Innovantes organisé par BPI France. Il a récompensé Numalis dans la catégorie de sociétés en "Émergence".
Valider vos calculs
Modélisation mathématique ?
Approximation de calcul ?
Fiabilité des résultats ?
Ne laissez plus les erreurs de calculs de vos ordinateurs altérer vos décisions ou les leurs, Numalis vous fournit des solutions logicielles et une expertise unique pour la validation numérique de tous vos programmes.
Optimiser leur précision
Précision de calcul ?
Performances ?
Respect des certifications ?
La précision et les performance ne sont pas antagonistes, Numalis vous fournit le premier outil permettant d'obtenir la meilleure implémentation de vos calculs pour tous vos programmes.
Pour la finance
Gestion d'actifs, assurance, trading...
La fiabilité de vos outils conditionne
Pour la simulation géophysique
De la prospection à l'analyse jusqu'à l'exploitation, optimisez vos


[N. Damouche, M. Martel & A. Chapoutot, *Impact of Accuracy Optimization on the Convergence of Numerical Iterative M* ic Programming and Semantics Program Transformations (LO

[N. Damouche, M. Martel & poutot, *I* *ural Optimization of the Numerical Accuracy of Programs*, Formal Met ustrial Critical Systems (FMICS) , LNCS, 2015]

[N. Damouche, M. Martel & A. Chapoutot *ation of a PID Controller for Numerical Accuracy*, Numerical Software (NSV), ENTCS, 2014]

[E. Goubault, *Static analysis by abstr etation of numerical programs and systems, and FLUCTUAT*. Static Ana posium, LNCS 7935, 2013]

[M. Martel, *Accurate Evaluation of A Expressions (Invited Talk)*, 4th Int. Workshop on Numerical and Symbolic Abstract Domains (NSAD), ENTCS, 2012]

[A. Ioualalen & M. Martel, *A New Ab omain for the Representation of Mathematically Equivalent Expression* Analysis Symposium, LNCS 7460, 2012]

[R. Tate, M. Stepp, Z. Tatlock, & S. Lerner, *Equality saturation: A new approach to optimization*. Logical Methods in Comp. Sci., 7(1), 2011]