ADAPTIVE CAVALIERI SIMPSON ALGORITHM

POLITECNICO
MILANO 1863

DIPARTIMENTO DI MATEMATICA

Project *Algorithm and Parallel Computing*

Professor Danilo Ardagna

Academic Year 2018-2019

Daniele Ceccarelli & Matilde Perego

Overview

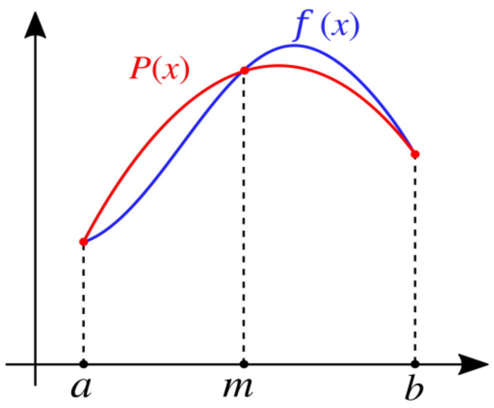
POLITECNICO DI MILANO

- **Introduction**
  - The formulas: Standard, Composite & Adaptive Cavalieri Simpson
  - Adaptive Cavalieri Simpson
    - Definition
    - Intervals
    - Error and tolerance
- **Code**
  - Single
  - MPI
- **Numerical experiments**

Introduction

- **Standard Cavalieri Simpson**

$$I_s(f) = (b-a)[\, f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)]$$



- **Composite Cavalieri Simpson**

$$I_s{}^c(f) = \frac{H}{6}\sum_{k=0}^{N-1}[f(x_k) + 4f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_{k+1})]$$

- **Adaptive Cavalieri Simpson**

Adaptive Cavalieri Simpson formula

POLITECNICO DI MILANO

- Goal: approximate the integral of a function

- Characteristic:
  - **Non-uniform distribution** of integration nodes;
  - **Higher performance**
    → adaptive algorithm guarantees the same accuracy of the composite formula, but with a lower number of quadrature nodes and so with **less evaluation of f**;
  - **Recursivity**
    → if the error exceeds a fixed tolerance, the algorithm divides the interval of integration in two subintervals and applies adaptive Simpson's method to the first one.

Adaptive Intervals

POLITECNICO DI MILANO

The **integration step** is modified **dynamically** in order to satisfy the tolerance control

1. Calculate an **approximation** $I_s(f)$ of $I(f) = \int_\alpha^\beta f(x)dx$

   (First step: α = a, β = b)

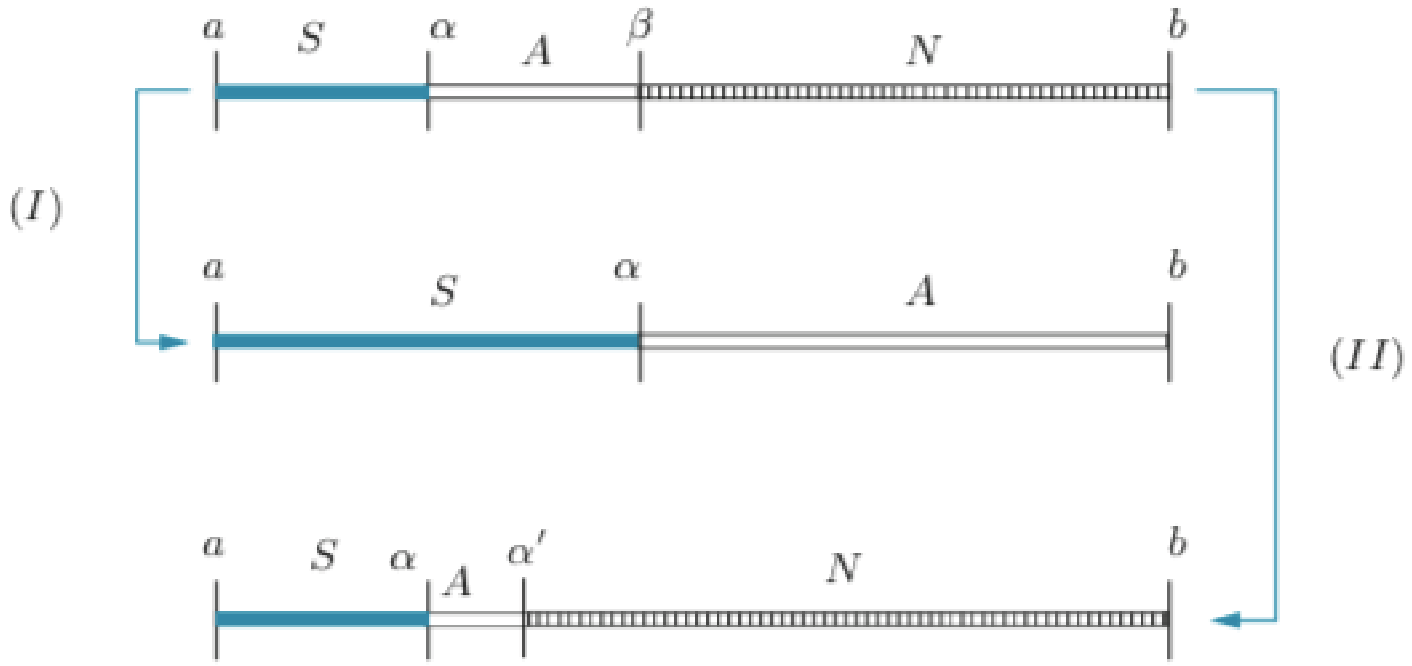
2. Define $\boldsymbol{H = \beta - \alpha}$ and **evaluate the error**:
   - Error < tolerance ✓ → the approximation is ok
   - Error > tolerance ✗ → the interval (α, β) is divided in two subinterval: $H = (\beta - \alpha)/2$

     → evaluation of $I(f) = \int_\alpha^{\alpha+H} f(x)dx$

     → check again the error

3. Consider the interval $(\alpha+H, b)$, **update** $H = (\beta-(\alpha+H))$ and repeat the procedure until the whole interval has been approximated.

Adaptive Intervals



Intervals:

- **A = active**
  → interval on which we are evaluating the integral;
- **S = already evaluated**
  → interval of integration that had already been evaluated;
- **N = still to evaluate**
  → remaining interval on which evaluating the integral.

Error & tolerance

$E_s(f;\alpha,\beta) < \varepsilon\,\frac{\beta-\alpha}{b-a}$ in [α,β] ⊂ [a, b] $\Rightarrow$ then the error on the whole interval respects the tolerance ε.

Using the Cavalieri-Simpson error formula $I(f) - I_s(f) = -\frac{1}{16}\frac{(b-a)^5}{180} f^{(4)}(\xi)$ we obtain:

$$E_s(f;\alpha,\beta) = \int_\alpha^\beta f(x)dx - I_s(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi) < \varepsilon\,\frac{\beta-\alpha}{b-a} \qquad (1)$$

As ξ is unknown we can use the **composite quadrature formula** to estimate the value of $\int_\alpha^\beta f(x)dx$ with a step $H = (\beta-\alpha)/2$ and we get:

$$\int_\alpha^\beta f(x)dx - I_s{}^c(f) = -\frac{(b-a)^5}{46080} f^{(4)}(\eta) \qquad (2)$$

Error & tolerance

Subtracting (1) - (2): $\Delta I = I_s{}^c(f) - I_s(f) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi) + \frac{(b-a)^5}{46080} f^{(4)}(\eta)$

Assuming $f^{(4)}(x)$ is almost constant on the interval $f^{(4)}(\xi) \simeq f^{(4)}(\eta)$ we can evaluate its value and obtain the error estimate:

$$\int_{\alpha}^{\beta} f(x)dx - I_s{}^c(f) \cong \frac{1}{15} \Delta I$$

the integration step H will be accepted only if: $\boxed{\frac{1}{15} |\Delta I| < \frac{\varepsilon}{2} \frac{\beta-\alpha}{b-a}}$

ERROR ESTIMATE

Single code organization

Adaptive_cav_simp.hh

```
private:
  interval_helper I;
  const real tol;
  const real h_min;
  const std::function<real(real)> &f;
  void divide_interval_adapt();
  void update_extremes_adapt();
  bool tol_control(…) const

public:
  adaptive_cav_simp(…)
  real solve();
  bool finish_computation() const
  void print_intervals() const;
```

Interval_helper.hh

```
private:
    real a,b;
    real length;
    real alpha, beta;
    std::vector<real> nodes;
    bool finish = false;
    bool inverted;

public:
    interval_helper(…)
    void divide_interval();
    void update_extremes();
    void change_finish_flag();

    getters & print
```

Cavalieri_simpson.hh

```
private:
real cavalieri_simpson (…);
```

Cavalieri_simpson_5_points.hh

```
private:
real cavalieri_simpson_5_points (…);
```

Classes *single* code

POLITECNICO DI MILANO

- **Adaptive_cav_simp.hh**

  - **Solve ( )** → the real core of the program
    → applies adaptive Cavalieri Simpson formula (*)

  - **Interval_helper object** → helps to manage dynamical interval

  - **Tol_control** → control over the tolerance using the error estimate

  - **Print_intervals ( )** → prints the mesh dynamically created by the algorithm

Classes *single* code

- **Interval_helper.hh**

    - **Divide_interval ( )** → splits the interval dynamically when the error is bigger than the tolerance

    - **Update_extremes ( )** → updates the extremes dynamically when the error respects the tolerance

    - This class contains both the fixed interval extremes and the dynamical ones

Classes *single* code

- **Cavalieri_simpson.hh**
  - Evaluates the integral of the function f between a and b with standard Cavalieri-Simpson formula:

$$I_s(f) = (b-a)[\, f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)]$$

- **Cavalieri_simpson_5_points.hh**
  - Evaluates the integral of the function f between a and b with 5 points composite Cavalieri-Simpson formula

$$I_s{}^c(f) = \frac{H}{6}\sum_{k=0}^{4}[f(x_k) + 4f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_{k+1})]$$

Solve()

POLITECNICO DI MILANO

```
while(I.get_alpha() != b)        //we stop when alpha (begin of dynamical interval) is equal to b (end of fixed interval)
{
  real alpha = I.get_alpha();   //copy of alpha and beta
  real beta = I.get_beta();

  real value1 = numerical::cavalieri_simpson(f, alpha, beta);      //compute the integral with classical cav_simp
  real value2 = numerical::cavalieri_simpson_5_points(f, alpha, beta);    //compute the integral (with 5 points cav_simp)
                                                                                    //for error control

  if(tol_control(value1, value2))       //if the tollerance control is true
  {
    result += value1;                  //sum the partial integral (value1)
    update_extremes_adapt();           //update dynamical extremes
  }
  else if (beta - alpha < h_min)      //if the interval is less than h_min
  {
      std::cout << "Warning, h too small in: ( " << alpha << ", " << beta << " )" << '\n';
      result += value1;                //warning, then as before
      update_extremes_adapt();
  }
  else        //tol_control false but the interval is larger than h_min
  {
    divide_interval_adapt();           //split the interval
  }
}
```

MPI code introduction

POLITECNICO DI MILANO

We decided to implement a code that put together the composite algorithm and the adaptive one of Cavalier Simpson formula for quadrature in order to improve the performances → for this purpose we used **parallel computation**.

How we used MPI?

- For the evaluation of the integrals to control the tolerance;
- We did not divided all the interval a priori, because in presence of singularities we could run into the risk that a rank has to do a higher number of iteration respect the others.

MPI code organization

Adaptive_cav_simp.hh

```
private:
  interval_helper I;
  const real tol;
  const real h_min;
  const std::function<real(real)> &f;
  void divide_interval_adapt();
  void update_extremes_adapt();
  bool tol_control(…) const

public:
  adaptive_cav_simp(…)
  real solve();
  bool finish_computation() const
  void print_intervals()const;
```

Interval_helper.hh

```
private:
    real a,b;
    real length;
    real alpha, beta;
    std::vector<real> nodes;
    bool finish = false;
    bool inverted;

public:
    interval_helper(…)
    void divide_interval();
    void update_extremes();
    void change_finish_flag();

    getters & print
```

Cavalieri_simpson_MPI.hh

```
private:
   real cavalieri_simpson_MPI (…);
```

Cavalieri_simpson_composite.hh

```
private:
real cavalieri_simpson_composite (…);
```

Classes *MPI* code

- **Adaptive_cav_simp.hh**

  - **Solve ( )** → the real core of the program
    → applies adaptive Cavalieri Simpson formula (*)
  - **Interval_helper object** → helps to manage dynamical interval
  - **Tol_control** → control over the tolerance using the error estimate
  - **Print_intervals ( )** → prints the mesh dynamically created by the algorithm
  - **N** → number of subintervals for the composite cavalieri simpson method

Classes *MPI* code

- **Interval_helper.hh**

    - **Divide_interval ( )** → splits the interval dynamically when the error is bigger than the tolerance

    - **Update_extremes ( )** → updates the extremes dynamically when the error respects the tolerance

    - This class contains both the fixed interval extremes and the dynamical ones

Classes *MPI* code

- **Cavalieri_simpson_MPI.hh**

  - Divides the dynamical interval into ***N* subintervals** and send to every rank a portion of length ***local_h*** in order to apply composite Cavalieri Simpson formula with ***local_n*** intervals

  - Manage the case n % size ! = 0 employing a cyclic partition

  - Use MPI_Allreduce to sum all the result computed in the ranks

Classes *MPI* code

- **Cavalieri_simpson_composite.hh**

    - Evaluates the integral of the function f between a and b composite Cavalieri-simpson formula with 2**local_N + 1* points

$$I_s^c(f) = \frac{H}{6}\sum_{k=0}^{N-1}\left[f(x_k) + 4f\left(\frac{x_k + x_{k+1}}{2}\right) + f(x_{k+1})\right]$$

Numerical experiment

- **Which methods should we compare?**

  Our ''competitor'' is of course the **classical composite Cav-Simp** and we compare our two methods with it separately.

- **How can we compare the methods?**
  - Number of **functional evaluations** needed to have a fixed error;
  - for **MPI-comp.** this number is the sum of eval. for all the ranks;
  - We print both the total number and the number of final iteration since the **classical composite Cav-Simp** does not have a tolerance control.

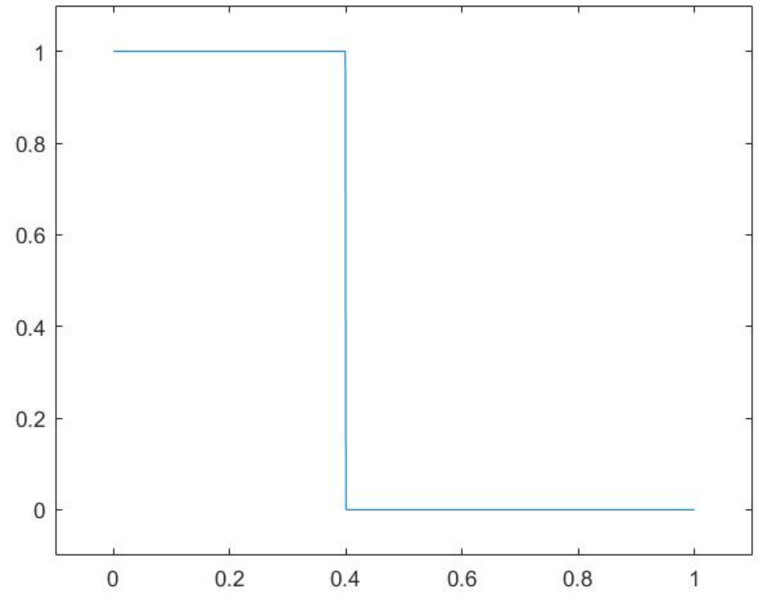
Numerical experiment

	Single Adaptive	Composite
Error 1	$1.8541*10^{-4}$	$1.8315*10^{-4}$
Func. Eval. 1	24(150)	365(16830)
Error 2	$2.2093*10^{-6}$	$2.2092*10^{-6}$
Func. Eval. 2	27(195)	30177 (113846499)

	MPI Comp. Adaptive	Composite
Error 1	$2.9084*10^{-5}$	$2.9036*10^{-5}$
Func. Eval. 1	88(539)	2297(660669)
Error 2	$2.2093*10^{-6}$	$2.2092*10^{-6}$
Func. Eval. 2	99(704)	30177 (113846499)

tol1 = 10^{-4}, h_min1 = 10^{-3}
tol2 = 10^{-5}, h_min2 = 10^{-4}

$$f(x) = \chi_{\{0;0.4\}}(x), \qquad \forall x \in [0,1]$$



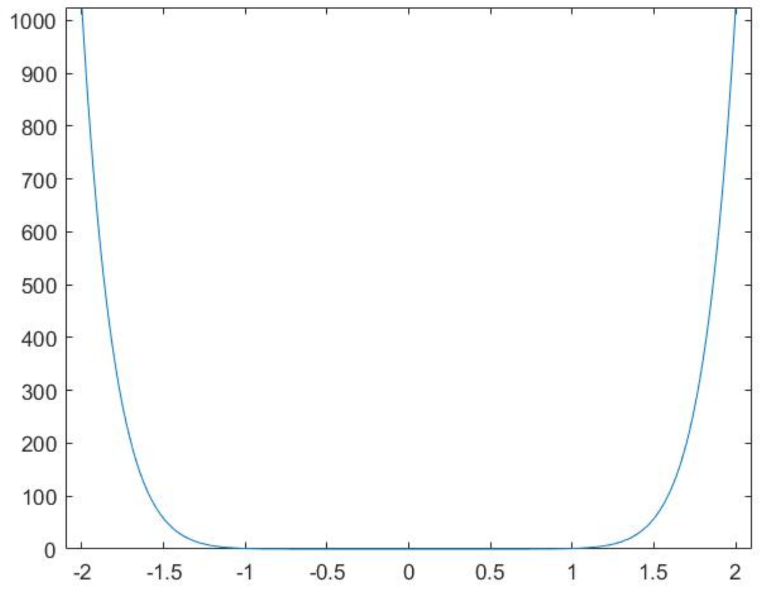
Numerical experiment

	Single Adaptive	Composite
Error 1	$2.5998*10^{-4}$	$2.4956*10^{-4}$
Func. Eval. 1	192(1096)	181(4180)
Error 2	$2.8023*10^{-5}$	$2.7658*10^{-5}$
Func. Eval. 2	342(2244)	313(12397)

	MPI Comp. Adaptive	Composite
Error 1	$2.42571*10^{-4}$	$2.2856*10^{-4}$
Func. Eval. 1	154(550)	185(4365)
Error 2	$2.2737*10^{-5}$	$2.26445*10^{-5}$
Func. Eval. 2	264(1133)	329(13689)

tol1 = 10^{-4}, h_min1 = 10^{-3}
tol2 = 10^{-5}, h_min2 = 10^{-4}

$$f(x) = x^{10}, \qquad \forall x \in [-2,2]$$



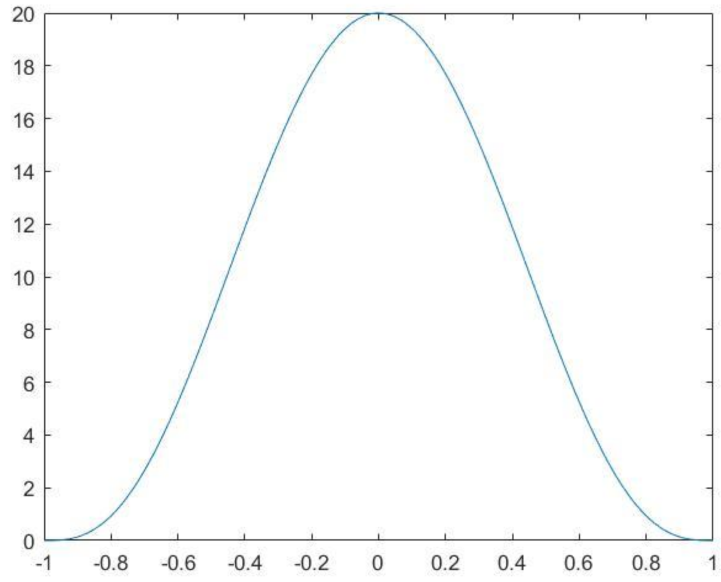
Numerical experiment

	Single Adaptive	Composite
Error 1	$3.1458*10^{-5}$	$2.3280*10^{-5}$
Func. Eval. 1	48(180)	53 (372)
Error 2	$9.7810*10^{-6}$	$7.9696*10^{-6}$
Func. Eval. 2	87(396)	69 (624)

	MPI Comp. Adaptive	Composite
Error 1	$1.4022*10^{-4}$	$1.0105*10^{-4}$
Func. Eval. 1	33(66)	37(184)
Error 2	$4.2033*10^{-6}$	$4.1620*10^{-6}$
Func. Eval. 2	66(165)	81(855)

tol1 = 10^{-4}, h_min1 = 10^{-3}
tol2 = 10^{-5}, h_min2 = 10^{-4}

$$f(x) = 20(1 - x^2)^3, \qquad \forall x \in [-1,1]$$



Adaptive Cavalieri Simpson Algorithm

Speed Up

POLITECNICO DI MILANO

$$\text{Speedup(k cores)} = \frac{\textit{Time adaptive composite serial}}{\textit{Time adaptive composite with k cores}}$$

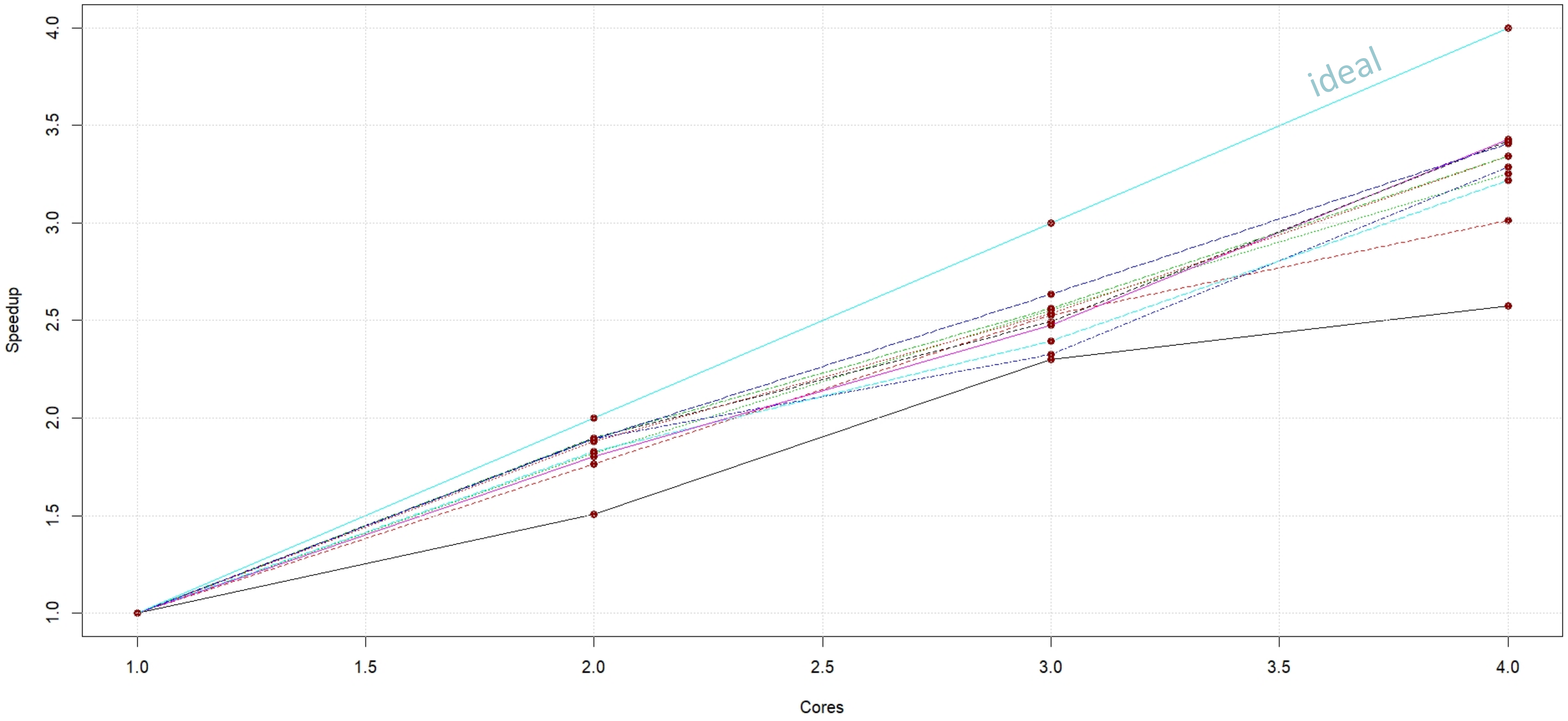
tol = 10^{-7}, h_min = 10^{-6}

$$f(x) = x^{10}, \qquad \forall x \in [-2,2]$$

Cores\N	100	200	300	400	500	600	700	800	900	1000	Ideal
2	1,508575	1,765025	1,818812	1,891797	1,830256	1,800858	1,895594	1,880052	1,897403	1,890893	2
3	2,300389	2,526643	2,551977	2,32655	2,396777	2,477877	2,492347	2,534427	2,560699	2,63699	3
4	2,575355	3,011805	3,251444	3,287034	3,216842	3,428276	3,418942	3,342296	3,34121	3,408207	4

Speed Up

POLITECNICO DI MILANO



Adaptive Cavalieri Simpson Algorithm

References

- Calcolo Scientifico: Esercizi e problemi risolti con MATLAB e Octave, Alfio Quarteroni, Fausto Saleri
- Slide of lessons of the course Algorithm and Parallel Computing of Professor Danilo Ardagna