Efficient evaluation of numerical functions

Tools and examples

Sylvain Chevillard

Supervisors: Nicolas Brisebarre and Jean-Michel Muller

École normale supérieure de Lyon
Laboratoire de l'informatique du parallélisme - Arenaire team

July, 6th 2009

- People usually enter in my office saying *I would like to approximate a function.*

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1+\cos(x))$, etc.

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1+\cos(x))$, etc.
- Values not exact in general:

$$\exp(2) = 7.38905609893065\ldots$$

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1+\cos(x))$, etc.
- Values not exact in general:

$$\exp(2) = 7.38905609893065\ldots$$

$\hookrightarrow$ approximated values.

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1+\cos(x))$, etc.
- Values not exact in general:

$$\exp(2) = 7.38905609893065\ldots$$

  $\hookrightarrow$ approximated values.
- For example: error beyond the 15th digit.

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1 + \cos(x))$, etc.
- Values not exact in general:

$$\exp(2) = 7.38905609893065\ldots$$

  $\hookrightarrow$ approximated values.
- For example: error beyond the 15th digit.
- Fifteen correct digits?

- People usually enter in my office saying *I would like to approximate a function.*
- For instance: $\sin(x)$, $\exp(x)$, $\exp(1+\cos(x))$, etc.
- Values not exact in general:

$$\exp(2) = 7.38905609893065\ldots$$

  $\hookrightarrow$ approximated values.
- For example: error beyond the 15th digit.
- Fifteen correct digits?

$$0.999999999999999\,91234$$
$$1.000000000000000\,05678$$

- Don't programs for evaluating functions already exist?

- Don't programs for evaluating functions already exist?
- Yes, but we can:
  - improve the efficiency while keeping the accuracy;
  - improve the accuracy while keeping the efficiency.

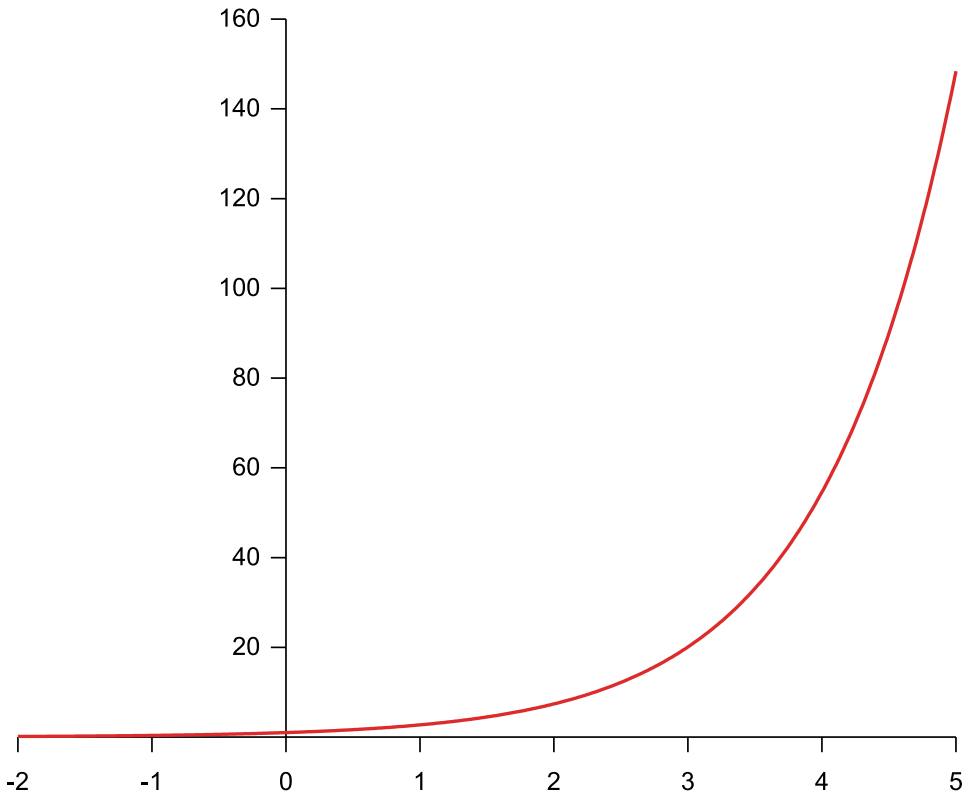
- Don't programs for evaluating functions already exist?
- Yes, but we can:
  - improve the efficiency while keeping the accuracy;
  - improve the accuracy while keeping the efficiency.
- New programs must be designed for new architectures.

- Don't programs for evaluating functions already exist?
- Yes, but we can:
  - improve the efficiency while keeping the accuracy;
  - improve the accuracy while keeping the efficiency.
- New programs must be designed for new architectures.

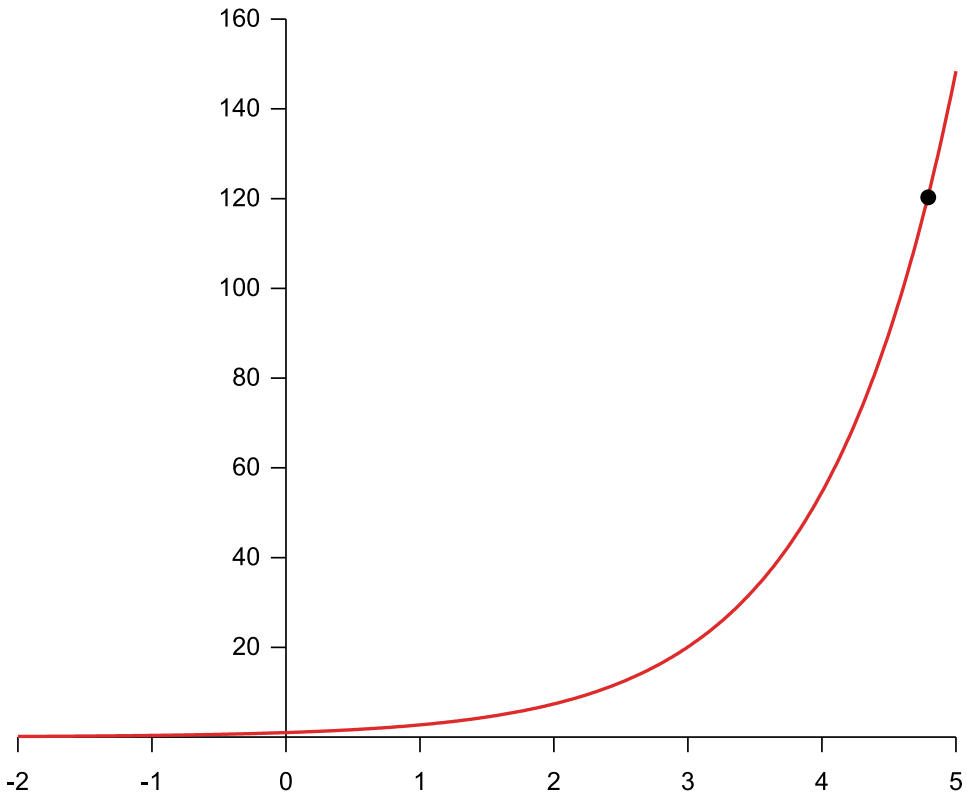
  $\hookrightarrow$ the program must be written quickly.

- Example of function $f(x) = \exp(x)$.

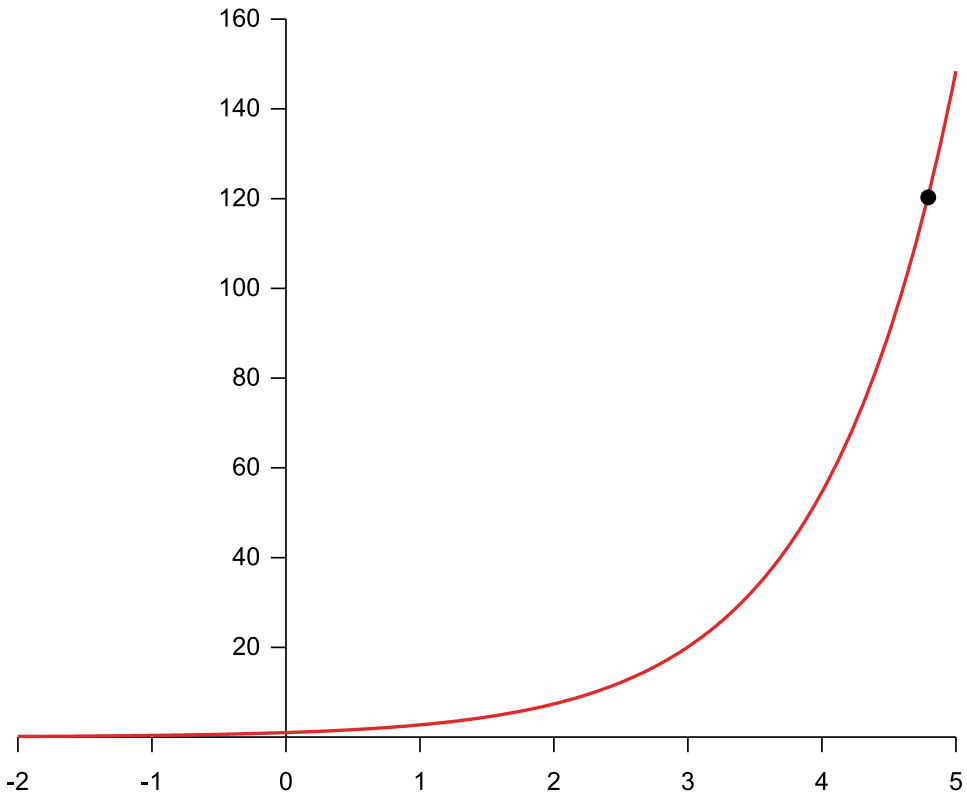
- Example of function $f(x) = \exp(x)$.



- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).

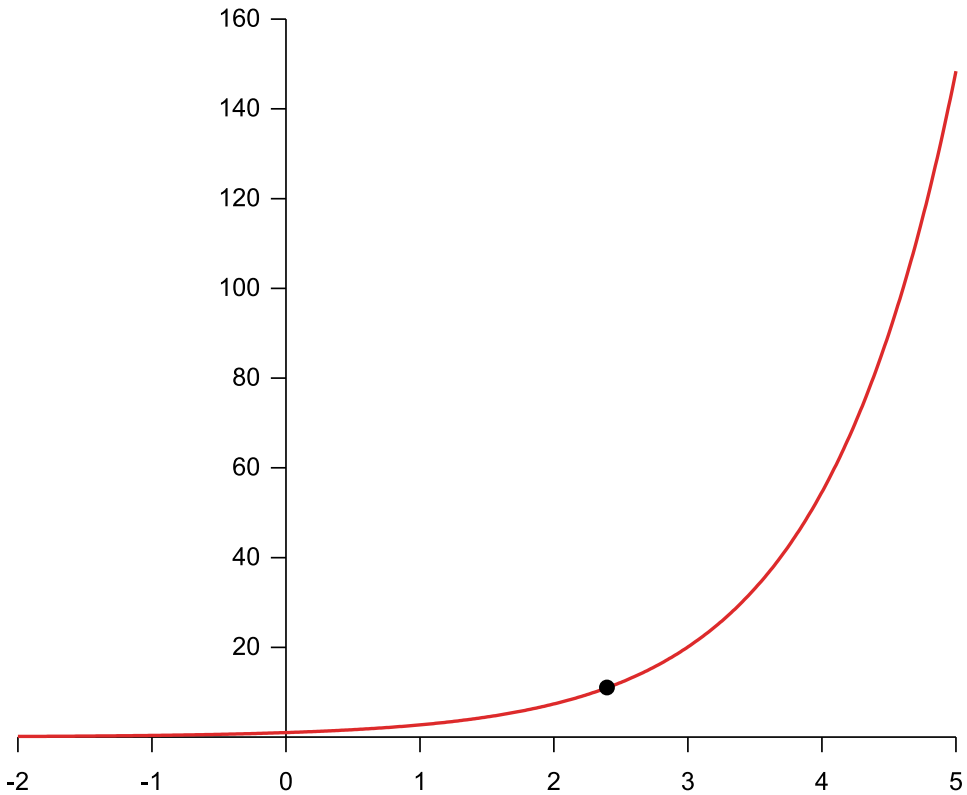


- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



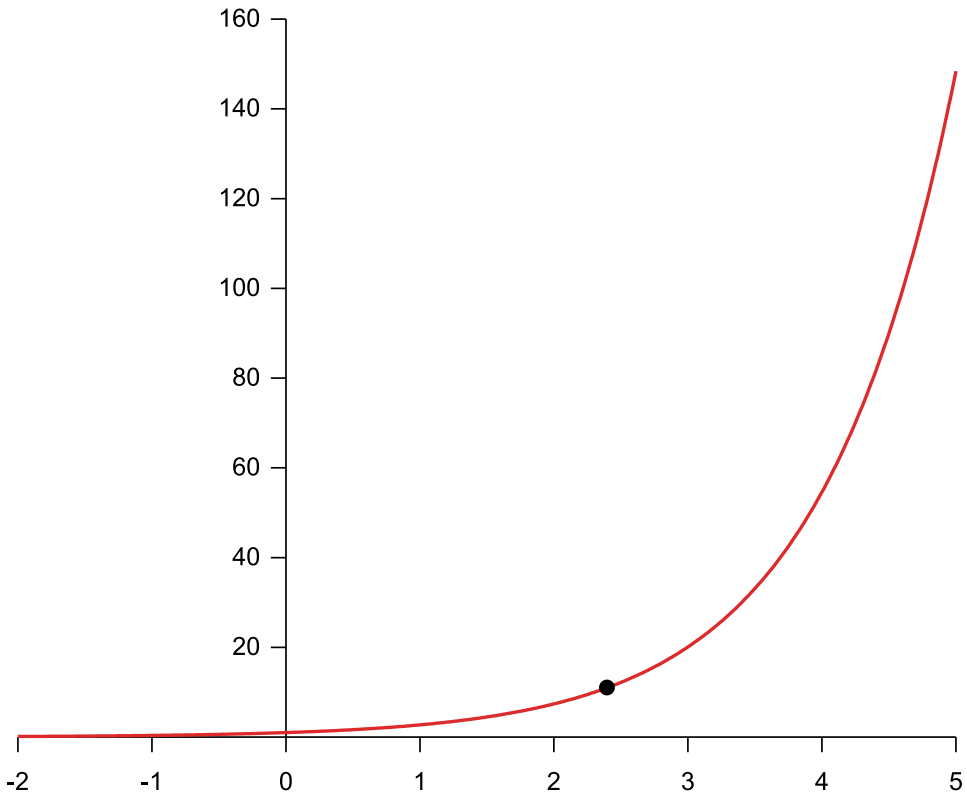
$$\exp(a) = \exp(a/2)^2$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



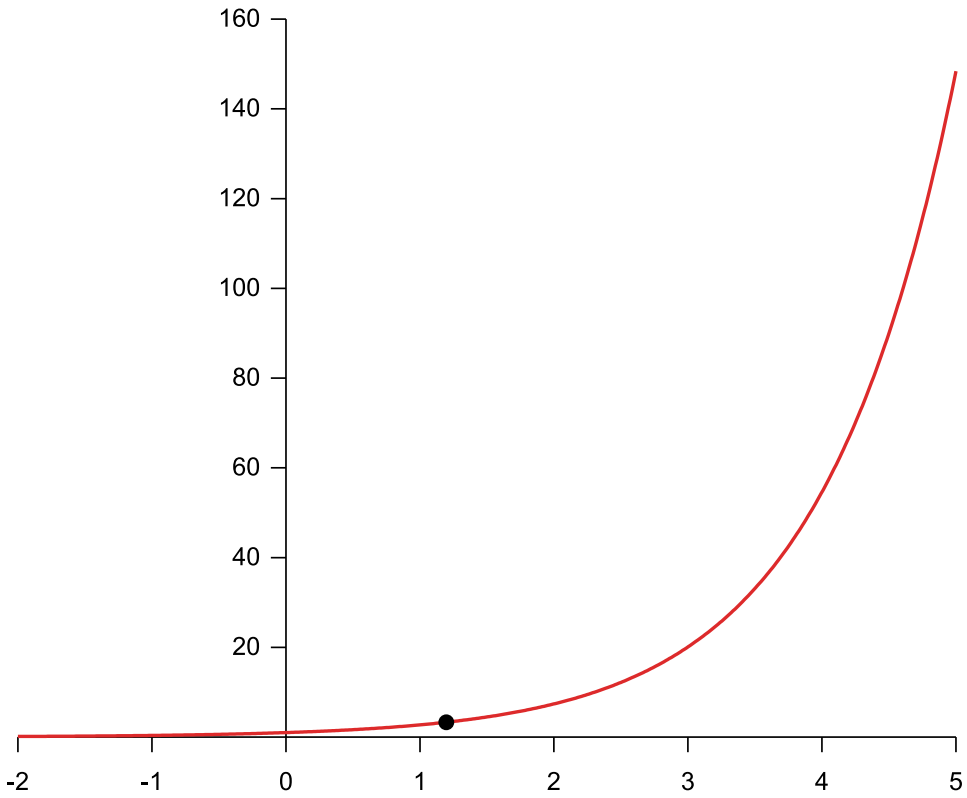
$$\exp(a) = \exp(a/2)^2$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



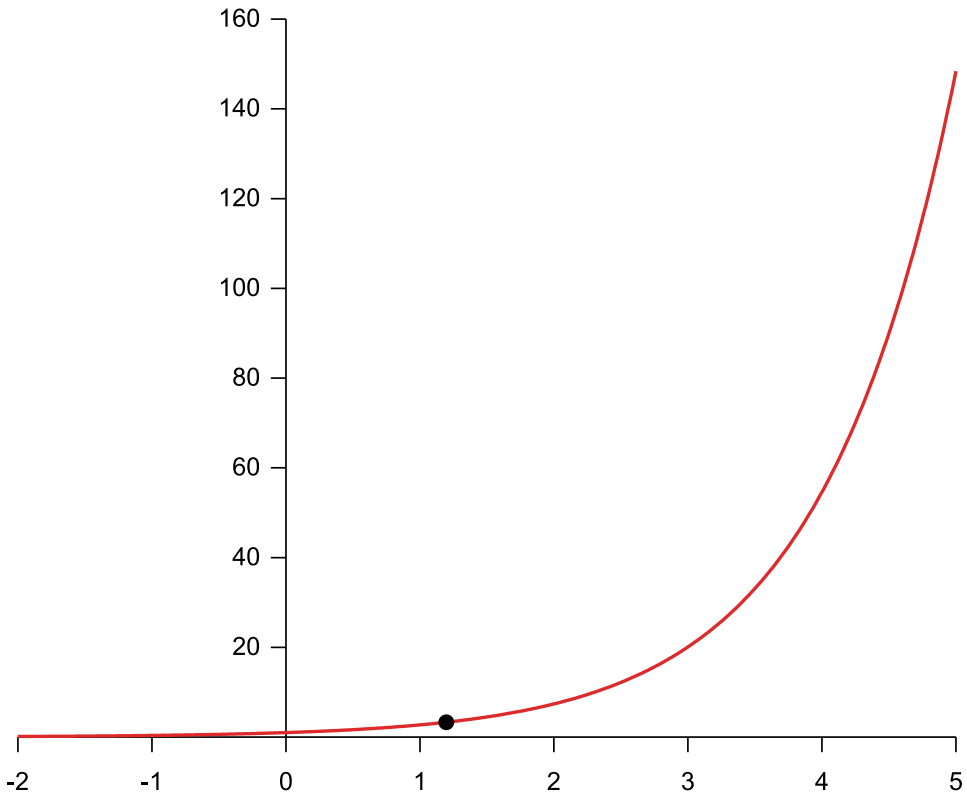
$$\exp(a) = \left(\exp(a/4)^2\right)^2$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



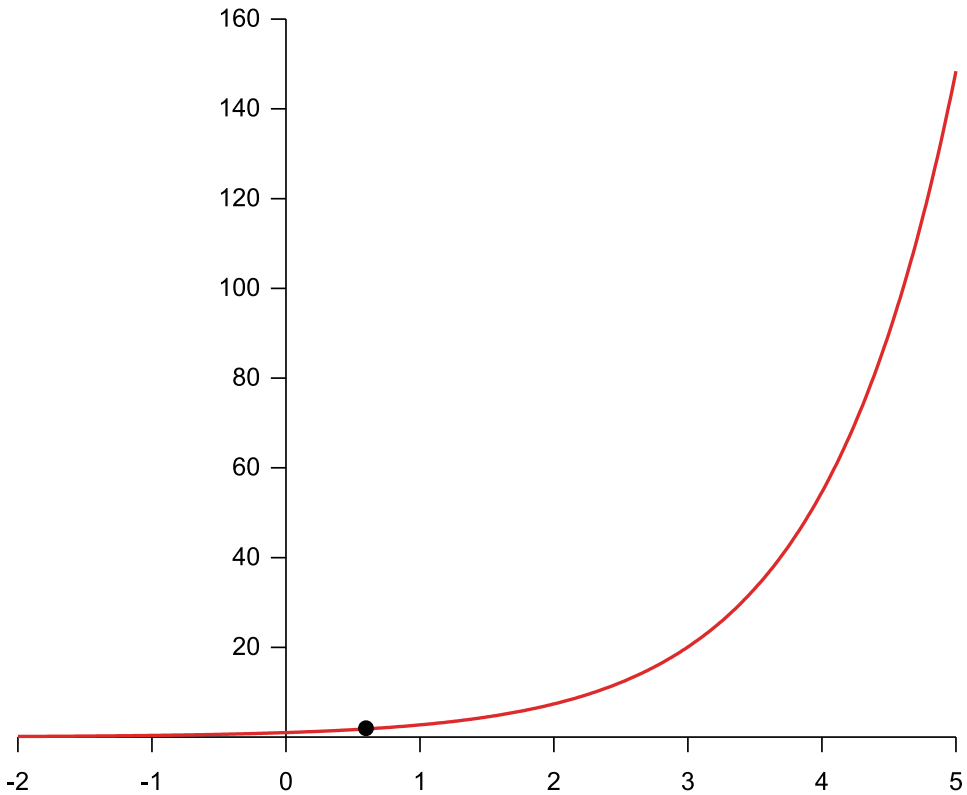
$$\exp(a) = \left(\exp(a/4)^2\right)^2$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



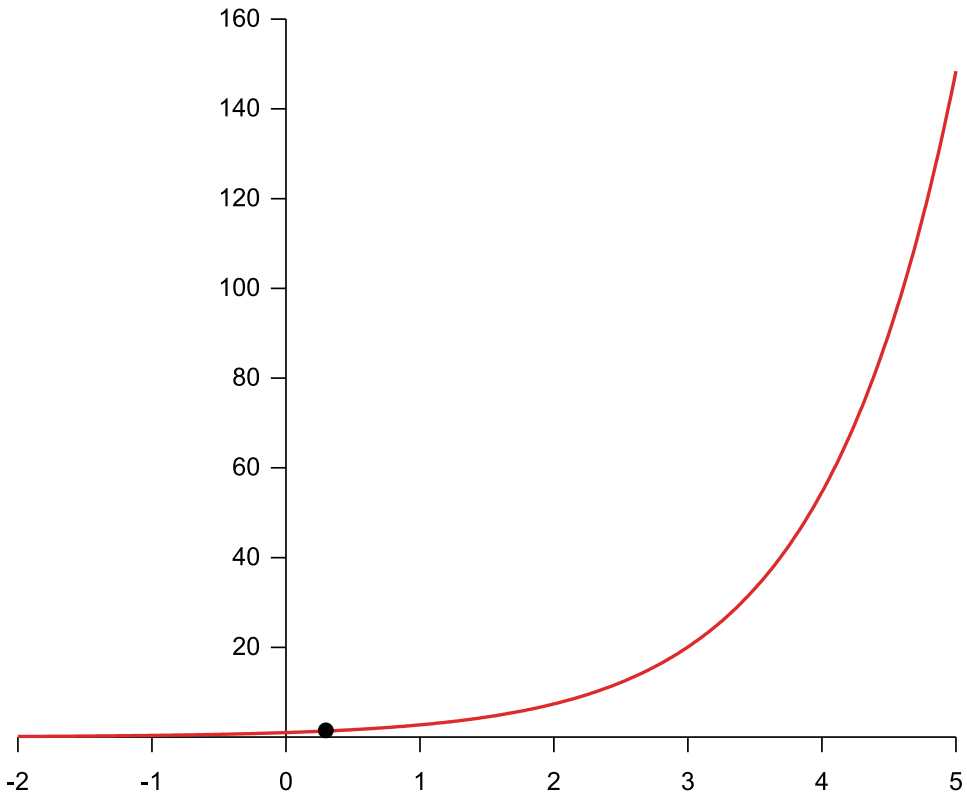
etc.

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).



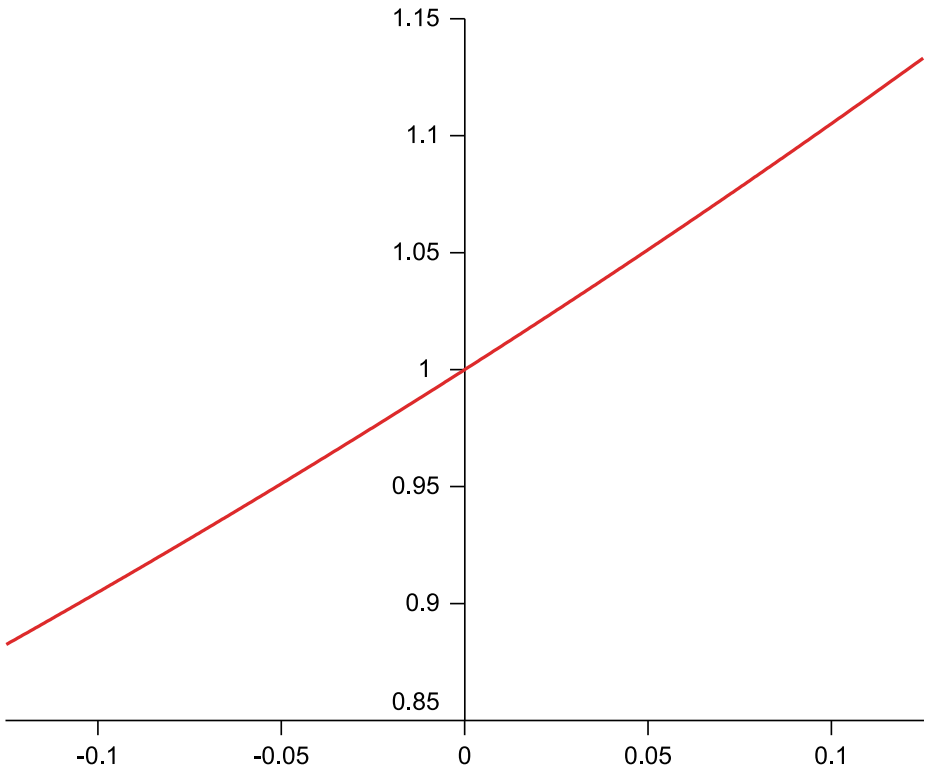
etc.

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).

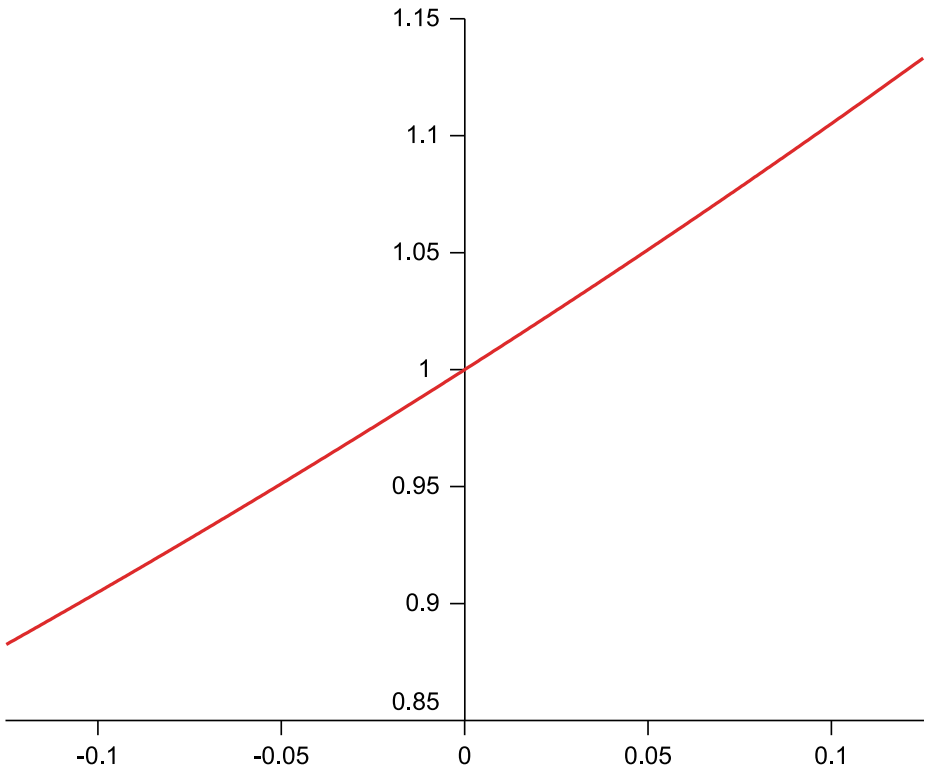


etc.

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).

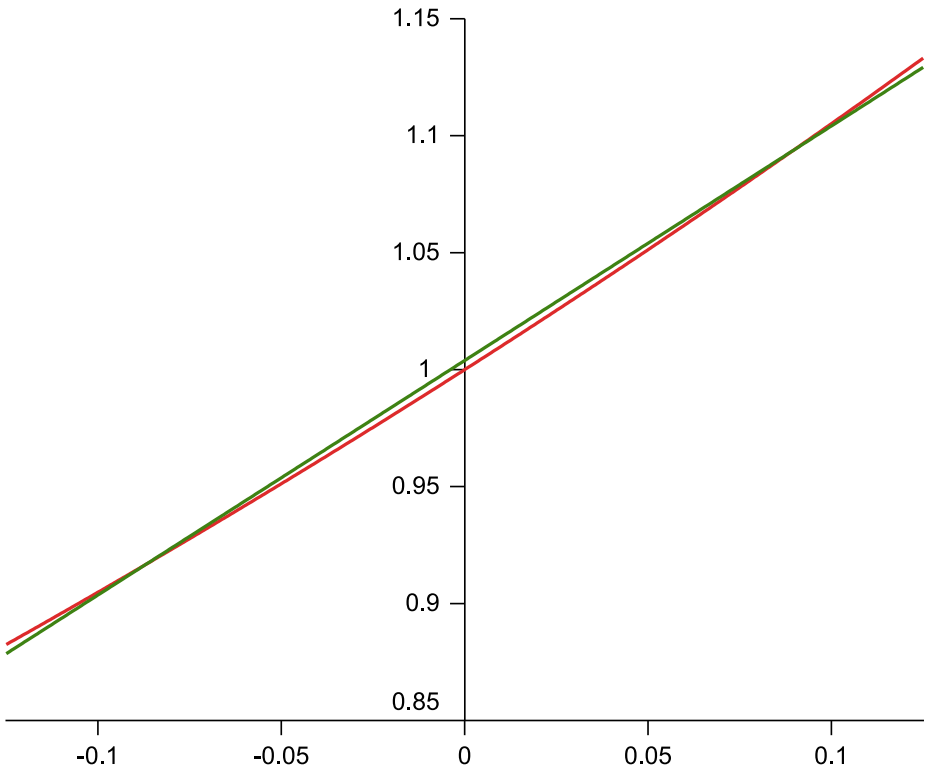


- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.



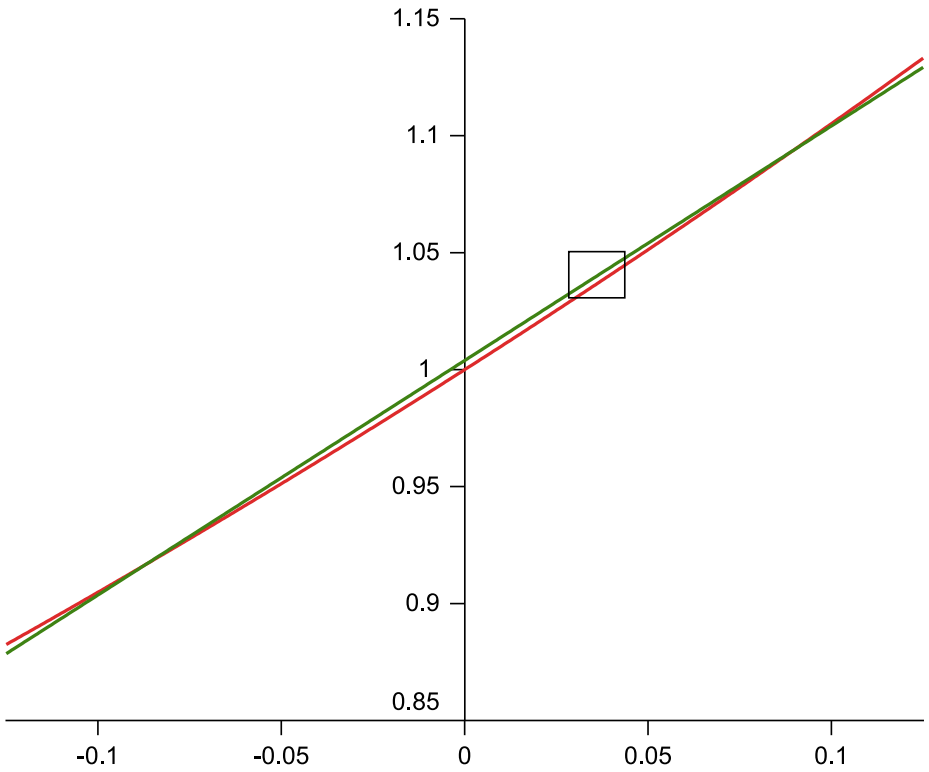
$$p \simeq f$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.

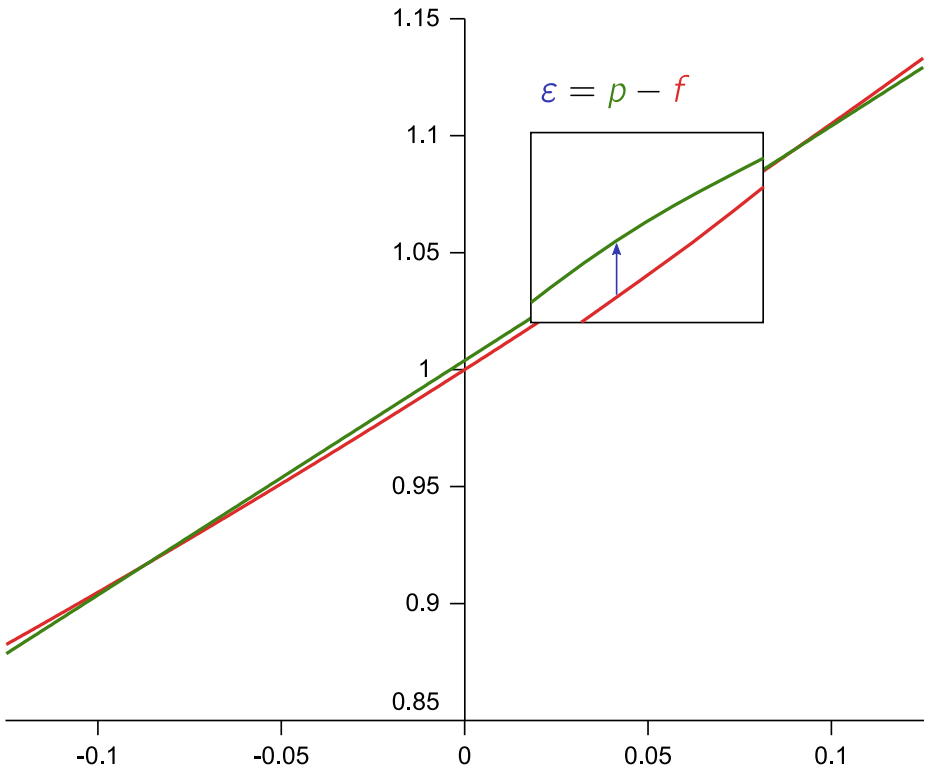


$$p \simeq f$$

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.



- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.



Approximation error:

- absolute:

$$\varepsilon = p - f$$

- relative:

$$\varepsilon = \frac{p - f}{f}$$

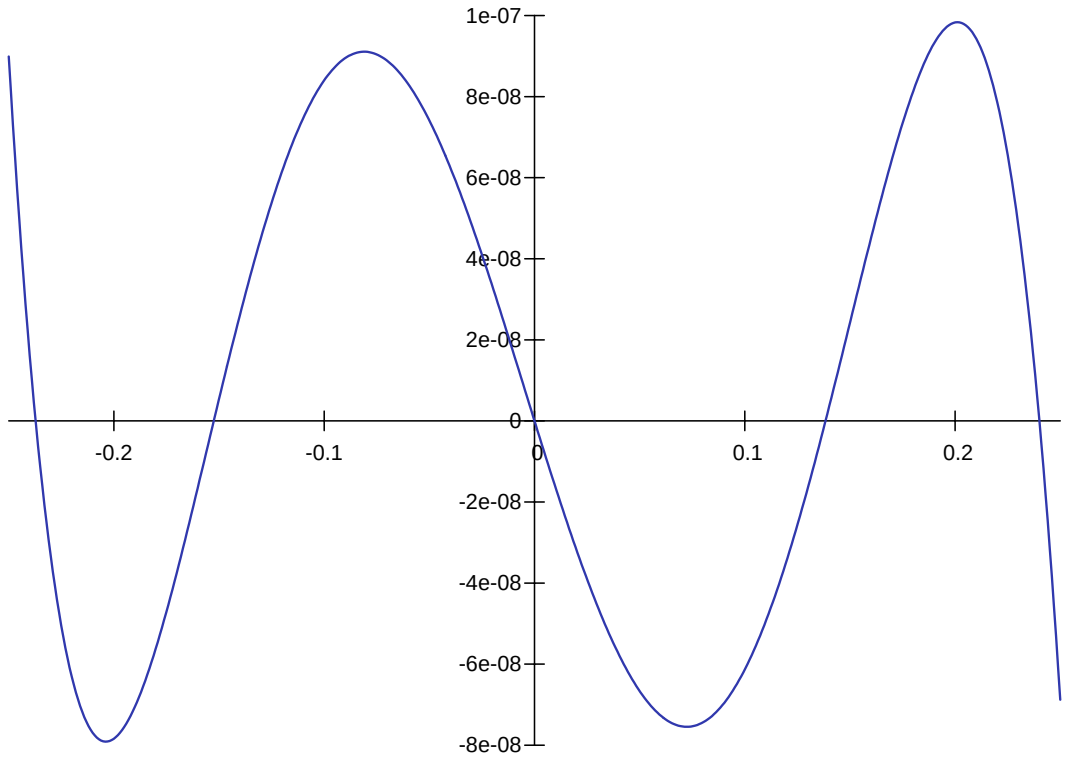
- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.
- Third step: write a program that evaluates the polynomial.

```
#define p_coeff_1h 1.00000000000000000000e+00
#define p_coeff_2h 4.99998331069946289062 50e-01
#define p_coeff_3h 1.66667461395263671875 00e-01
#define p_coeff_4h 4.17752265930175781250 00e-02
#define p_coeff_5h 8.33320617675781250000 00e-03


void p(double *p_resh, double x) {
  volatile float p_t_1_0h;
  volatile float p_t_2_0h;
  volatile float p_t_3_0h;
  volatile float p_t_4_0h;
  volatile float p_t_5_0h;
  volatile float p_t_6_0h;
  volatile float p_t_7_0h;
  volatile float p_t_8_0h;
  volatile float p_t_9_0h;
  volatile float p_t_10_0h;

  p_t_1_0h = p_coeff_5h;
  p_t_2_0h = p_t_1_0h * x;
  p_t_3_0h = p_coeff_4h + p_t_2_0h;
  p_t_4_0h = p_t_3_0h * x;
  p_t_5_0h = p_coeff_3h + p_t_4_0h;
  p_t_6_0h = p_t_5_0h * x;
  p_t_7_0h = p_coeff_2h + p_t_6_0h;
  p_t_8_0h = p_t_7_0h * x;
  p_t_9_0h = p_coeff_1h + p_t_8_0h;
  p_t_10_0h = p_t_9_0h * x;
  *p_resh = p_t_10_0h;
}
```

- Example of function $f(x) = \exp(x)$.
- First step: reduce the range (range reduction).
- Second step: replace the function f by a polynomial p.
- Third step: write a program that evaluates the polynomial.

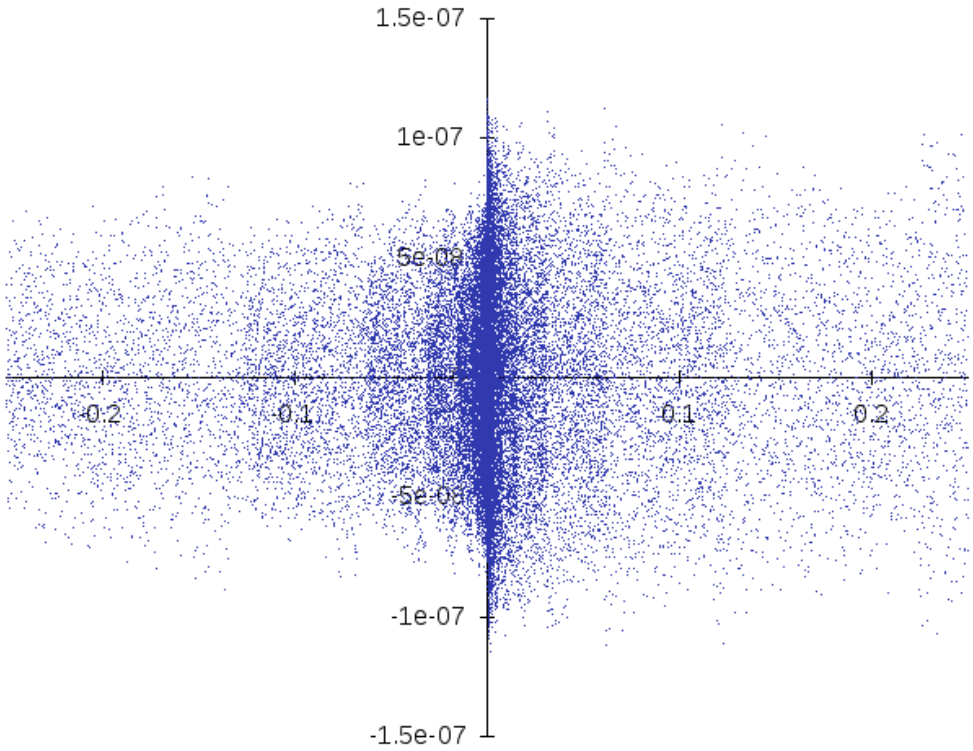
Sources of errors

- Approximation error: $\varepsilon = p - f$.



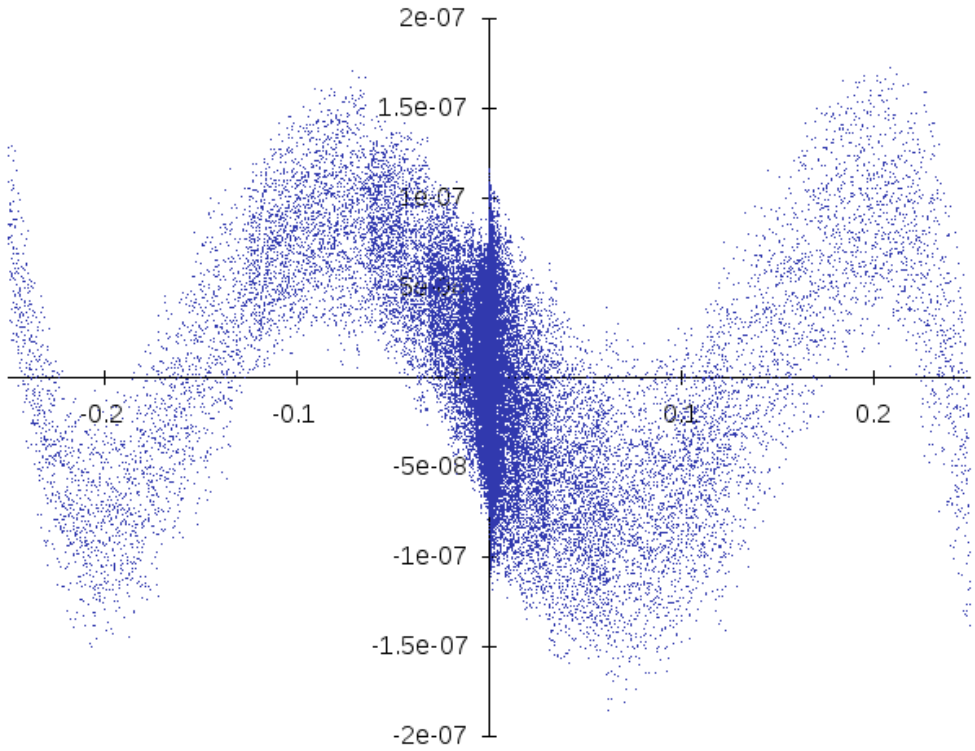
Sources of errors

- Approximation error: $\varepsilon = p - f$.
- Roundoff errors: due to the propagation of errors during the evaluation of p.



Sources of errors

- Approximation error: $\varepsilon = p - f$.
- Roundoff errors: due to the propagation of errors during the evaluation of p.
- The overall error must stay below the 15th digit.



Sources of errors

- Approximation error: $\varepsilon = p - f$.
- Roundoff errors: due to the propagation of errors during the evaluation of p.
- The overall error must stay below the 15th digit.

Sollya

- Sollya: developed with C. Lauter.

  $\hookrightarrow$ making the development of new functions easier.

Sollya

- Sollya: developed with C. Lauter.

  $\hookrightarrow$ making the development of new functions easier.
- Functions, polynomials, roundings, etc. in a safe environment.

Sollya

- Sollya: developed with C. Lauter.

  $\hookrightarrow$ making the development of new functions easier.
- Functions, polynomials, roundings, etc. in a safe environment.
- Now becoming a numerical toolbox.

  $\hookrightarrow$ interesting for anyone who wants guarantees on the quality.

Sollya

- Sollya: developed with C. Lauter.

  $\hookrightarrow$ making the development of new functions easier.
- Functions, polynomials, roundings, etc. in a safe environment.
- Now becoming a numerical toolbox.

  $\hookrightarrow$ interesting for anyone who wants guarantees on the quality.

- Thanks to Sollya, the development of functions has been almost completely automated.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.
- These algorithms are available within Sollya.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.
- These algorithms are available within Sollya.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.

- These algorithms are available within Sollya.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.
- These algorithms are available within Sollya.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.

- These algorithms are available within Sollya.

Main contributions of this thesis

- Implementation of function erf in arbitrary precision:
  - rigorous and effective bounds on sequences/functions;
  - particular constraints imposed by arbitrary precision.
- Algorithms for finding good polynomial/rational approximations:
  - approximation theory (real coefficients);
  - linear programming;
  - euclidean lattices.
- Algorithm for bounding rigorously the approximation error (between p and f):
  - interval arithmetic;
  - automatic computation of bounds on derivatives of a function;
  - root isolation techniques.
- These algorithms are available within Sollya.

- Reminder about approximation theory.

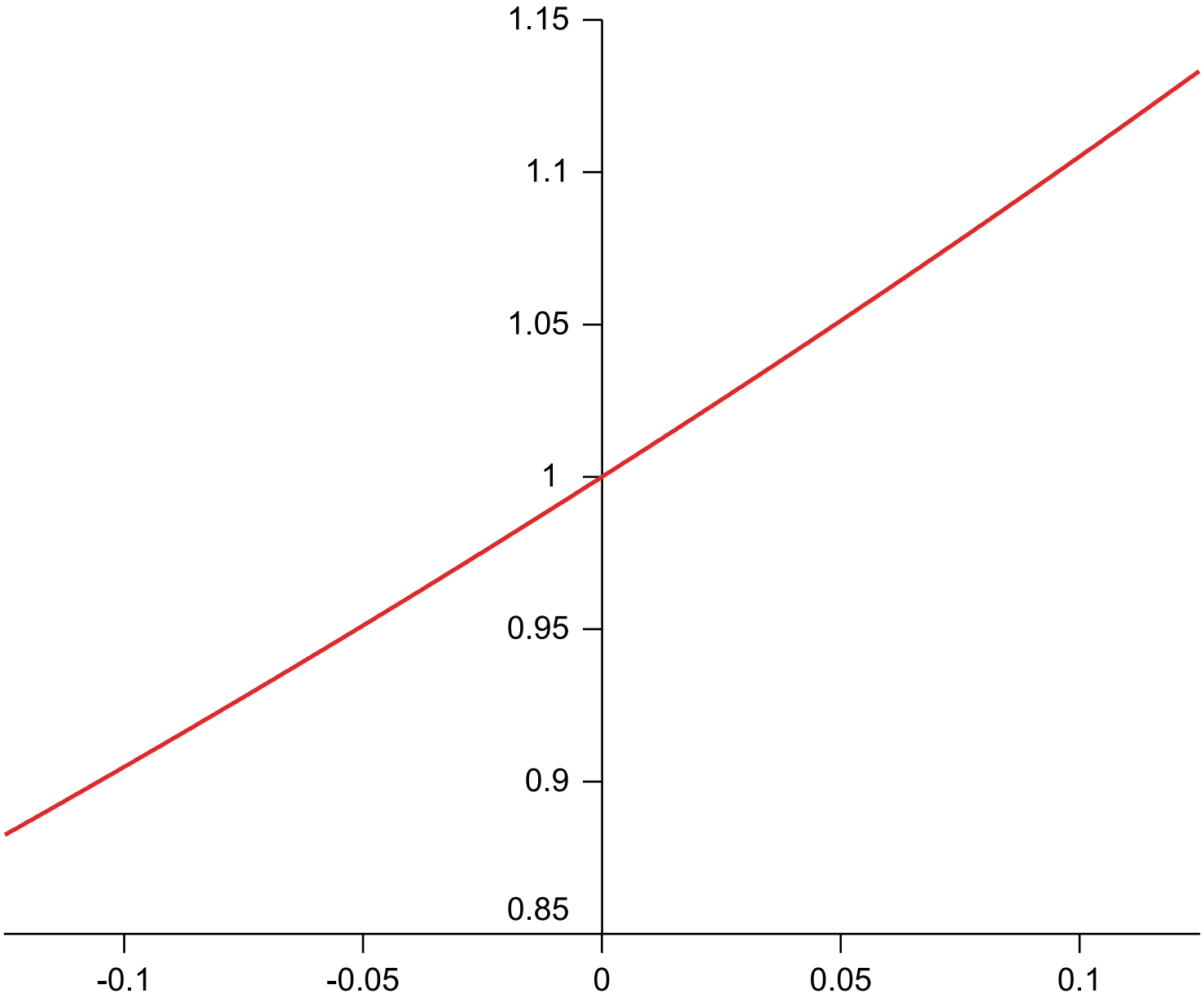
  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

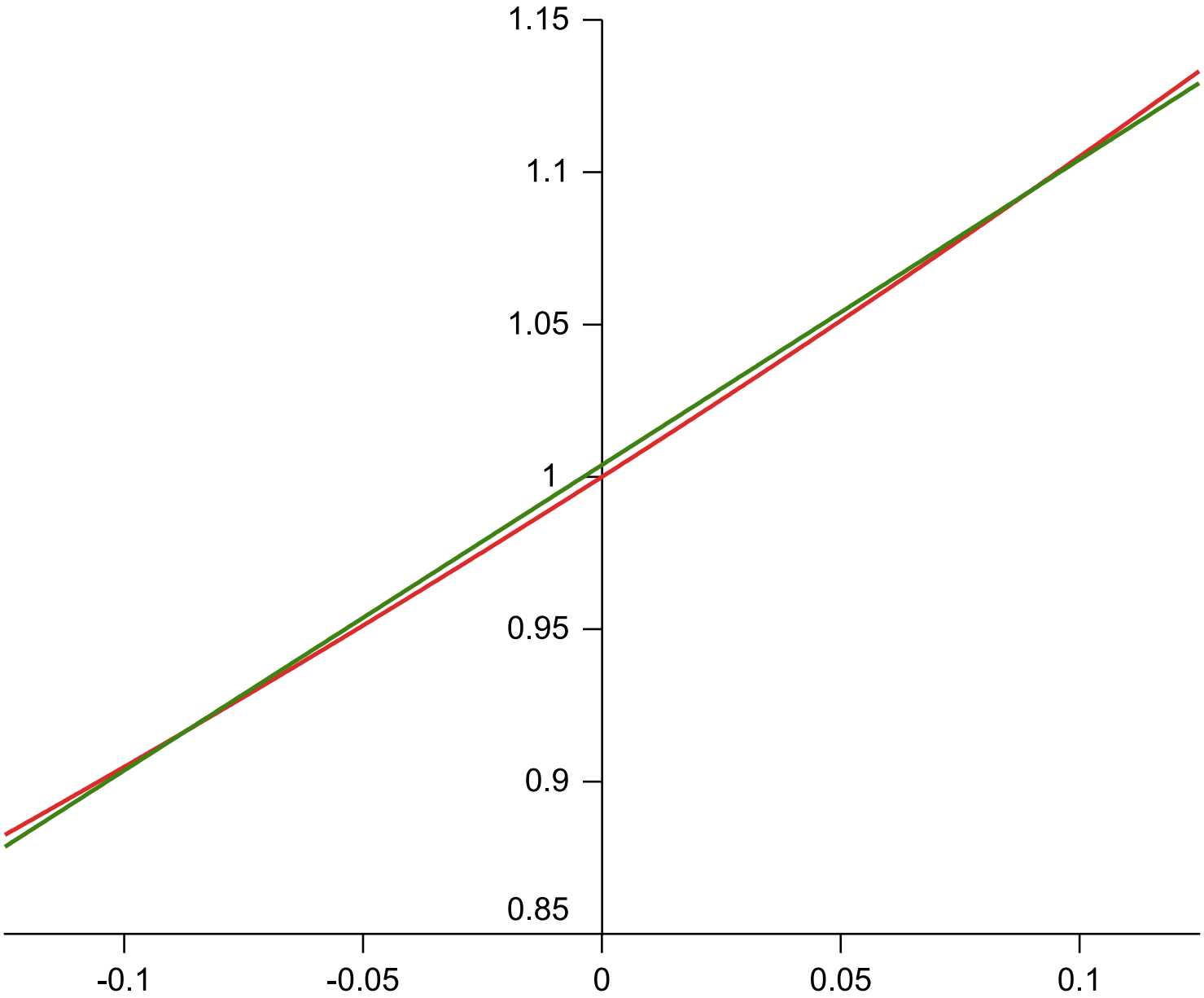
- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

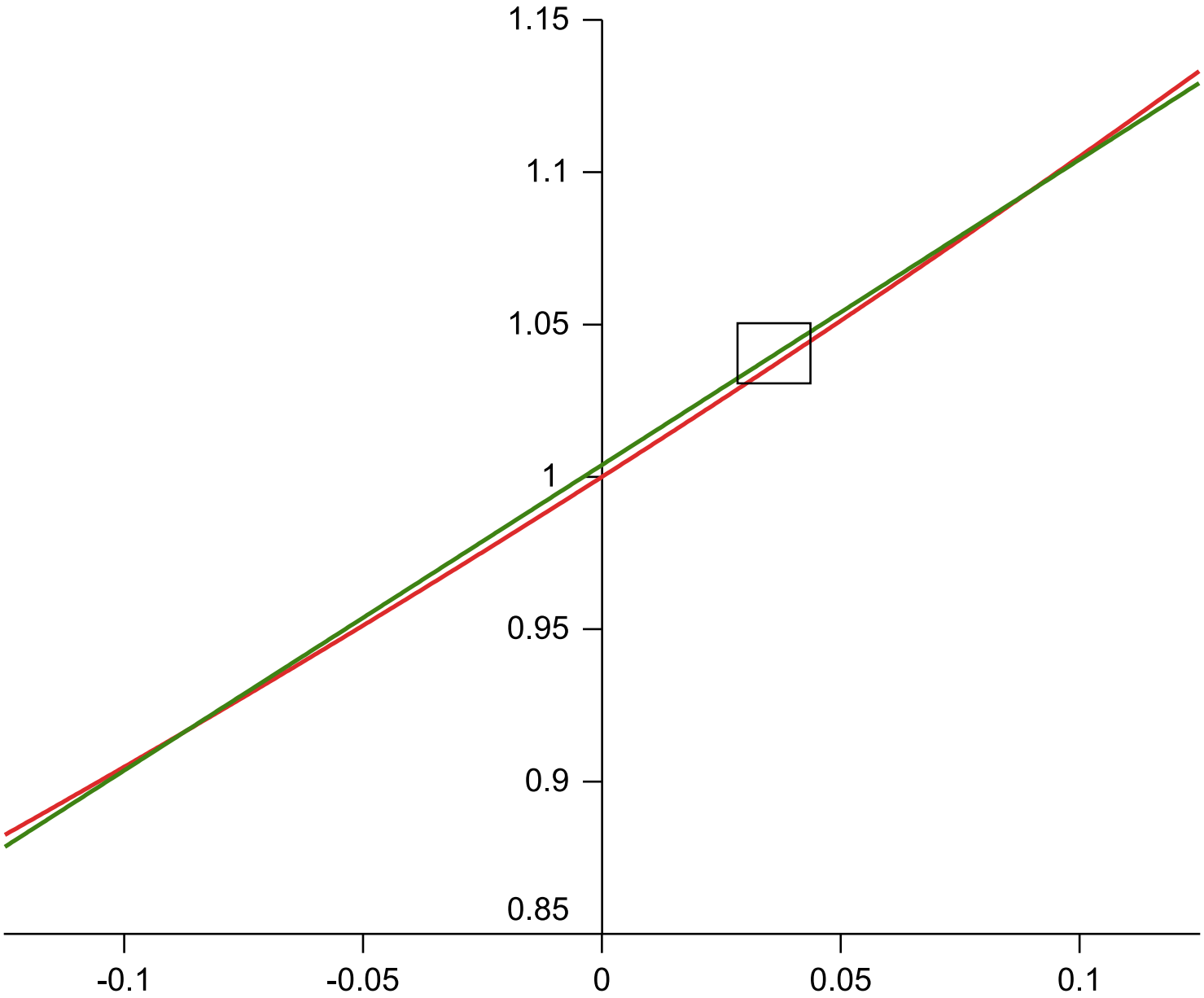
- Reminder about approximation theory.

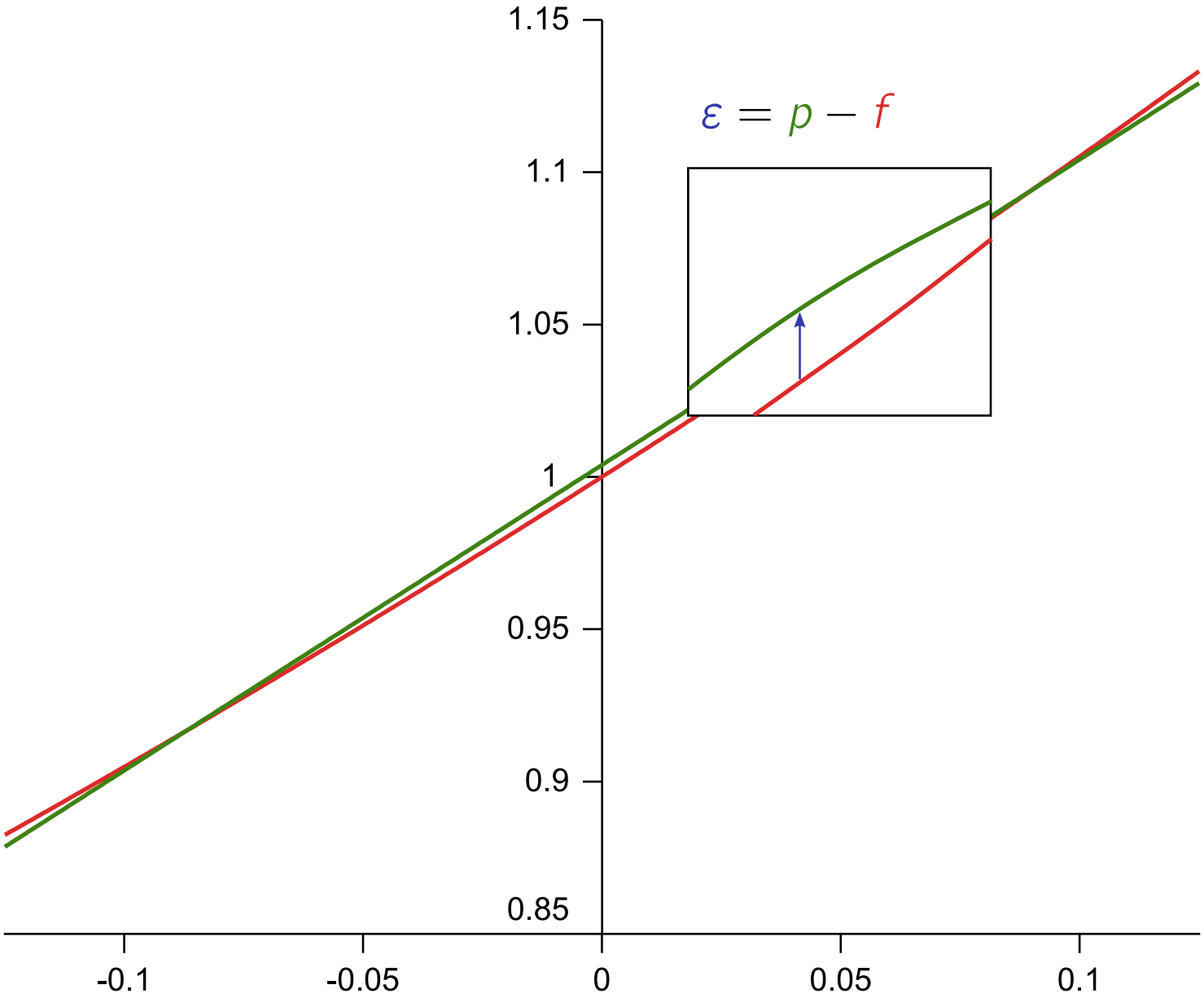
  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

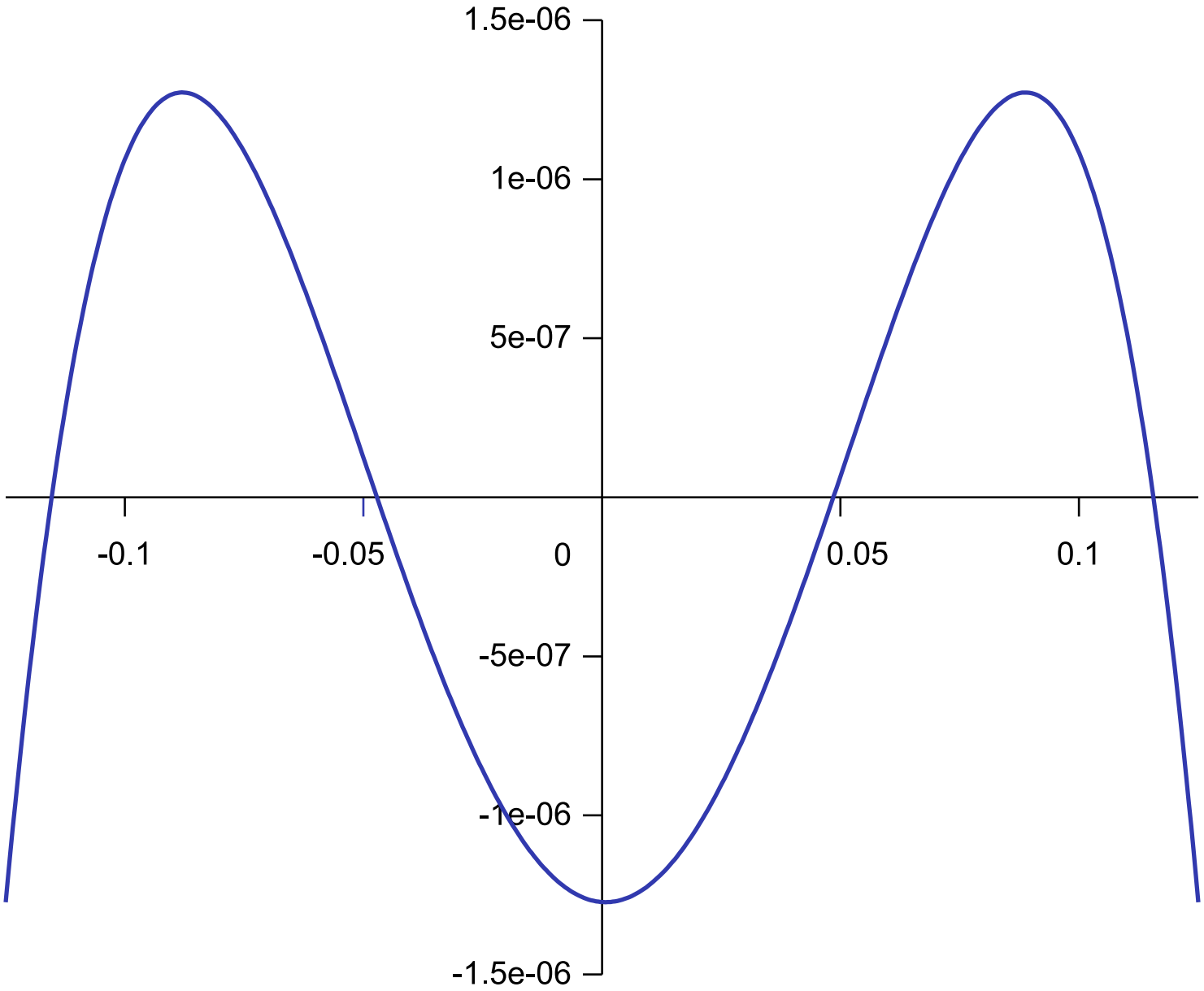
- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.


1.15
1.1
1.05
1
0.95
0.9
0.85
-0.1
-0.05
0
0.05
0.1

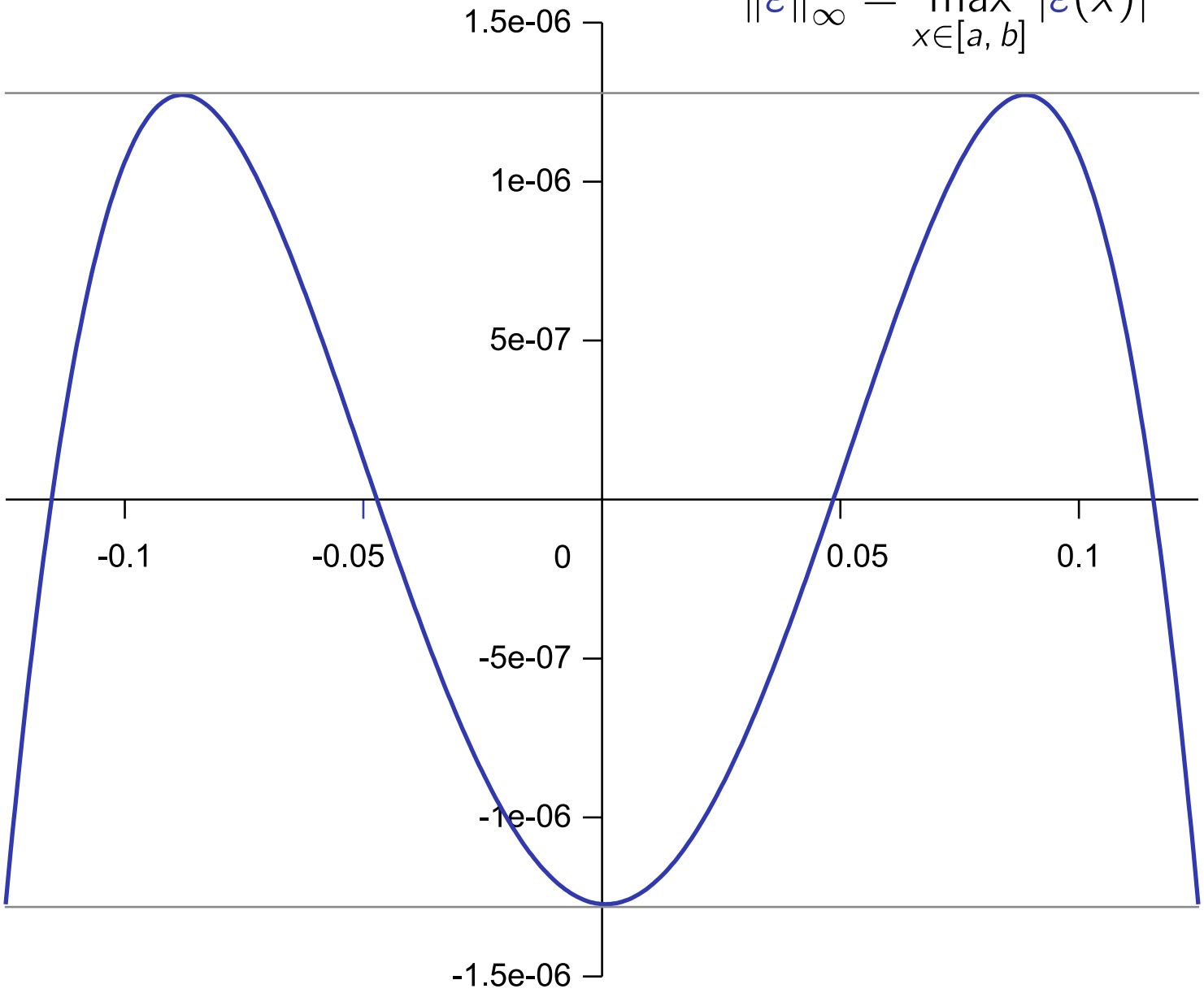

1.15
1.1
1.05
1
0.95
0.9
0.85
-0.1
-0.05
0
0.05
0.1


1.15
1.1
1.05
1
0.95
0.9
0.85
-0.1
-0.05
0
0.05
0.1


1.15
1.1
1.05
1
0.95
0.9
0.85
-0.1
-0.05
0
0.05
0.1
$\varepsilon = p - f$


1.5e-06
1e-06
5e-07
-5e-07
-1e-06
-1.5e-06
-0.1
-0.05
0
0.05
0.1

$$\|\varepsilon\|_\infty = \max_{x \in [a,\, b]} |\varepsilon(x)|$$



Approximation theory

- How to compute a good approximation polynomial?

Approximation theory

- How to compute a good approximation polynomial?

Characterisation theorem (1905)

p a polynomial of degree n, f a continuous function on $[a, b]$.

$$\|p - f\|_\infty \text{ is minimal (i.e. } p \text{ is optimal)}$$
$$\text{if and only if}$$
$$\exists x_1 < \cdots < x_{n+2} \in [a, b],\ \exists \varepsilon > 0,\ \begin{cases} \forall i,\ p(x_i) - f(x_i) = (-1)^i \varepsilon \\ |\varepsilon| = \|p - f\|_\infty . \end{cases}$$

Approximation theory

- How to compute a good approximation polynomial?

Characterisation theorem (1905)

p a polynomial of degree n, f a continuous function on $[a, b]$.

$$\|p - f\|_\infty \text{ is minimal (i.e. } p \text{ is optimal)}$$
$$\text{if and only if}$$
$$\exists x_1 < \cdots < x_{n+2} \in [a, b],\ \exists \varepsilon > 0,\ \begin{cases} \forall i,\ p(x_i) - f(x_i) = (-1)^i \varepsilon \\ |\varepsilon| = \|p - f\|_\infty . \end{cases}$$

- Theorem of la Vallée Poussin (1910).

  $\hookrightarrow$ oscillations and quality of approximation are related.

Approximation theory

- How to compute a good approximation polynomial?

Characterisation theorem (1905)

p a polynomial of degree n, f a continuous function on $[a, b]$.

$$\|p - f\|_\infty \text{ is minimal (i.e. } p \text{ is optimal)}$$
$$\text{if and only if}$$
$$\exists x_1 < \cdots < x_{n+2} \in [a, b],\ \exists \varepsilon > 0,\ \begin{cases} \forall i,\ p(x_i) - f(x_i) = (-1)^i \varepsilon \\ |\varepsilon| = \|p - f\|_\infty . \end{cases}$$

- Theorem of la Vallée Poussin (1910).

  $\hookrightarrow$ oscillations and quality of approximation are related.

- Remez' algorithm (1934).

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134003553928820\ldots\\
a_1 &\simeq 0.36787944121874345820075\ldots\\
a_2 &\simeq 0.18393894381629744544839\ldots\\
a_3 &\simeq -0.00000007647880859004234\ldots\\
a_4 &\simeq 0.13848496316631160839854\ldots\\
a_5 &\simeq 0.15944715191923863665924\ldots
\end{aligned}$$

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134003553928820\ldots\\
a_1 &\simeq 0.36787944121874345820075\ldots\\
a_2 &\simeq 0.18393894381629744544839\ldots\\
a_3 &\simeq -0.00000007647880859004234\ldots\\
a_4 &\simeq 0.13848496316631160839854\ldots\\
a_5 &\simeq 0.15944715191923863665924\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134003553928820\ldots \\
a_1 &\simeq 0.36787944121874345820075\ldots \\
a_2 &\simeq 0.18393894381629744544839\ldots \\
a_3 &\simeq -0.00000007647880859004234\ldots \\
a_4 &\simeq 0.13848496316631160839854\ldots \\
a_5 &\simeq 0.15944715191923863665924\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134003553928820\ldots \\
a_1 &\simeq 0.36787944121874345820075\ldots \\
a_2 &\simeq 0.18393894381629744544839\ldots \\
a_3 &\simeq 0 \\
a_4 &\simeq 0.13848496316631160839854\ldots \\
a_5 &\simeq 0.15944715191923863665924\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134057472430253\ldots\\
a_1 &\simeq 0.36787944115897691446498\ldots\\
a_2 &\simeq 0.18393894243661711831086\ldots\\
a_3 &\simeq 0\\
a_4 &\simeq 0.13848524573630958469899\ldots\\
a_5 &\simeq 0.15943149020444985343712\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134057472430253\ldots \\
a_1 &\simeq 0.36787944115897691446498\ldots \\
a_2 &\simeq 0.18393894243661711831086\ldots \\
a_3 &\simeq 0 \\
a_4 &\simeq 0.13848524573630958469899\ldots \\
a_5 &\simeq 0.15943149020444985343712\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.
- Constrained polynomial error: $1.69\,\mathrm{e}{-10}$.

Sometimes, additional constraints may be interesting.

- **Example:** $e^{\sin(x)-\cos(x^2)}$ on $[-1/16, 1/16]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.36787944134057472430253\ldots \\
a_1 &\simeq 0.36787944115897691446498\ldots \\
a_2 &\simeq 0.18393894243661711831086\ldots \\
a_3 &\simeq 0 \\
a_4 &\simeq 0.13848524573630958469899\ldots \\
a_5 &\simeq 0.15943149020444985343712\ldots
\end{aligned}$$

- Corresponding error: $1.68\,\mathrm{e}{-10}$.
- Constrained polynomial error: $1.69\,\mathrm{e}{-10}$.
- Stiefel's exchange algorithm (1959).

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

Representing real numbers in computers

- `IEEE-754` standard: defines floating-point numbers.

Representing real numbers in computers

- `IEEE-754` standard: defines floating-point numbers.
- A floating-point number x with radix 2 and precision t, is a number of the form:

$$x = 1.b_1 b_2 \dots b_{t-1} \cdot 2^{e'}, \qquad b_i \in \{0, 1\},\ e' \in \mathbb{Z}.$$

Representing real numbers in computers

- `IEEE-754` standard: defines floating-point numbers.
- A floating-point number x with radix 2 and precision t, is a number of the form:

$$x = 1.b_1 b_2 \dots b_{t-1} \cdot 2^{e'}, \qquad b_i \in \{0, 1\},\ e' \in \mathbb{Z}.$$

- Formally, $x = m \cdot 2^e$ where:
  - $m \in \mathbb{Z}$ (with exactly t bits) is its mantissa (or significand);
  - $e \in \mathbb{Z}$ is its exponent.

The problem

- Each coefficient of a polynomial is represented by a floating-point number.

The problem

- Each coefficient of a polynomial is represented by a floating-point number.
- Naive method to obtain a polynomial approximation of f:
  - compute the real minimax $p^\star$;
  - replace each coefficient $a_i^\star$ of $p^\star$ by the nearest floating-point number $\widehat{a_i}$;
  - use $\widehat{p} = \widehat{a_0} + \widehat{a_1}\,x + \cdots + \widehat{a_n}\,x^n$.

The problem

- Each coefficient of a polynomial is represented by a floating-point number.
- Naive method to obtain a polynomial approximation of f:
  - compute the real minimax $p^\star$;
  - replace each coefficient $a_i^\star$ of $p^\star$ by the nearest floating-point number $\widehat{a_i}$;
  - use $\widehat{p} = \widehat{a_0} + \widehat{a_1}\,x + \cdots + \widehat{a_n}\,x^n$.
- Example with $f(x) = \log_2(1 + 2^{-x})$ on $[0; 1]$
  $n = 6$, single precision coefficients (24 bits).

$\|\varepsilon^\star\|$	$\|\widehat{\varepsilon}\|$	$\|\varepsilon_{\text{opt}}\|$
8.3 e−10	119 e−10	10.06 e−10

p_{opt} : best polynomial with floating-point coefficients.

Previous works

- There exist recipes, not published.

Previous works

- There exist recipes, not published.
- D. Kodek (1980) has studied a similar problem in signal processing. Limited to small precisions ($t < 10$).

Previous works

- There exist recipes, not published.
- D. Kodek (1980) has studied a similar problem in signal processing. Limited to small precisions ($t < 10$).
- N. Brisebarre, J.-M. Muller and A. Tisserand (2006) have proposed an approach by linear programming.
  Limited to small degrees ($n < 8$).

Previous works

- There exist recipes, not published.
- D. Kodek (1980) has studied a similar problem in signal processing. Limited to small precisions ($t < 10$).
- N. Brisebarre, J.-M. Muller and A. Tisserand (2006) have proposed an approach by linear programming.
  Limited to small degrees ($n < 8$).
- Typically, we want $t \geq 50$ and $n \geq 10$.

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

Polytope approach

- **Inputs:** degree n, interval $[a, b]$, function f, list of floating-point formats t_i (one per coefficient).
- **Notations:**
  - $\varepsilon^\star = \|p^\star - f\|_\infty$ where $p^\star$ is the real minimax.
  - $\varepsilon_{\text{opt}} = \|p_{\text{opt}} - f\|_\infty$ where p_{opt} is a best polynomial with floating-point coefficients.

Polytope approach

- **Inputs:** degree n, interval $[a, b]$, function f, list of floating-point formats t_i (one per coefficient).
- **Notations:**
  - $\varepsilon^\star = \|p^\star - f\|_\infty$ where $p^\star$ is the real minimax.
  - $\varepsilon_{\text{opt}} = \|p_{\text{opt}} - f\|_\infty$ where p_{opt} is a best polynomial with floating-point coefficients.
- We set a trial error: $\varepsilon_{\text{target}}$.
- We are looking for p such that $\|p - f\|_\infty \leq \varepsilon_{\text{target}}$.

Polytope approach

- **Inputs:** degree n, interval $[a, b]$, function f, list of floating-point formats t_i (one per coefficient).
- **Notations:**
  - $\varepsilon^\star = \|p^\star - f\|_\infty$ where $p^\star$ is the real minimax.
  - $\varepsilon_{\text{opt}} = \|p_{\text{opt}} - f\|_\infty$ where p_{opt} is a best polynomial with floating-point coefficients.
- We set a trial error: $\varepsilon_{\text{target}}$.
- We are looking for p such that $\|p - f\|_\infty \leq \varepsilon_{\text{target}}$.

$$\iff \forall x \in [a, b], \quad f(x) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n} a_i x^i \leq f(x) + \varepsilon_{\text{target}}.$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i \, x_0^i \\ \sum_{i=0}^{n-1} a_i \, x_0^i & \leq & f(x_0) + \varepsilon_{\text{target}} \end{cases}$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_0^i \\ \\ \end{cases}$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

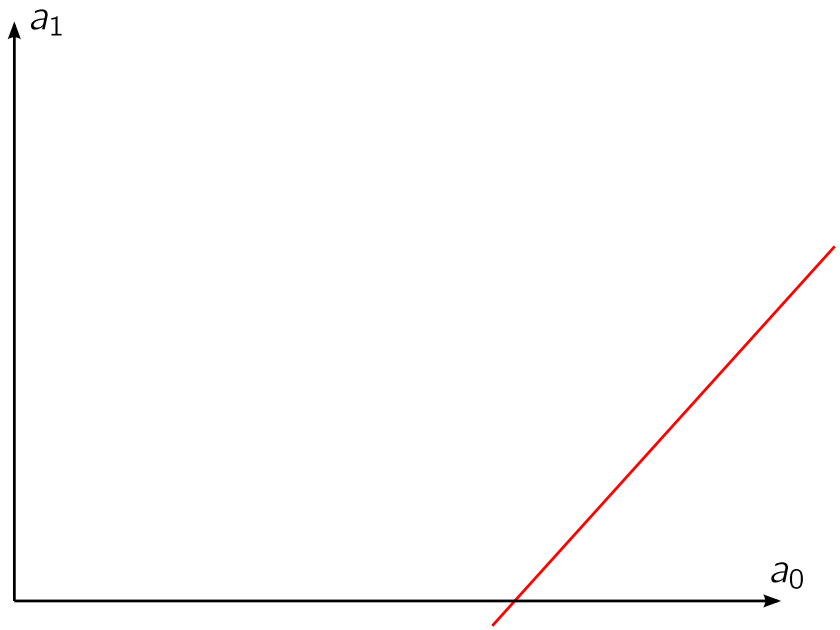
$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} = \sum_{i=0}^{n-1} a_i x_0^i \\ \end{cases}$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} = \sum_{i=0}^{n-1} a_i x_0^i \\ \end{cases}$$

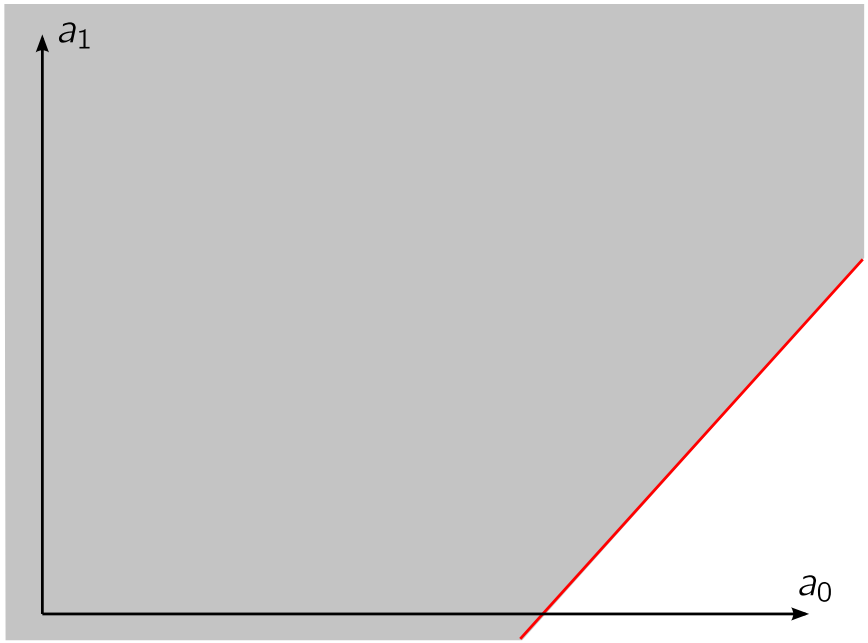


Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_0^i \\ \end{cases}$$

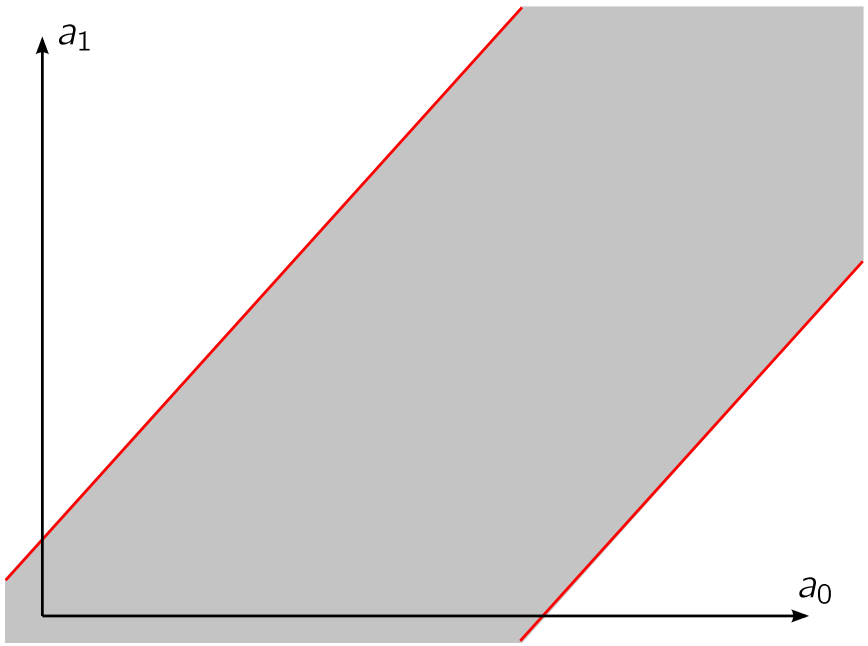


Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0 \in [a, b]$.

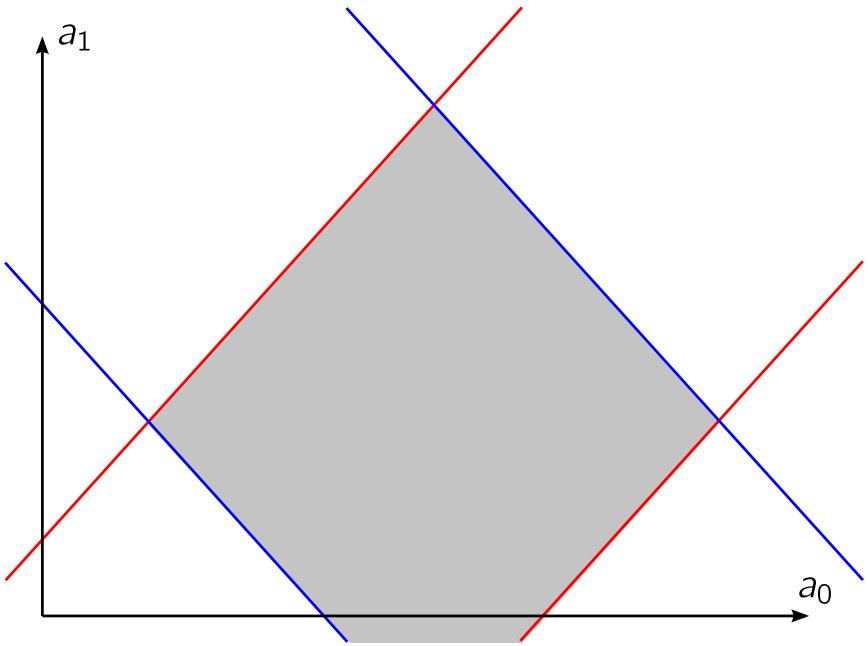
$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i \, x_0^i \\ \sum_{i=0}^{n-1} a_i \, x_0^i & \leq & f(x_0) + \varepsilon_{\text{target}} \end{cases}$$



Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0, x_1 \in [a, b]$.



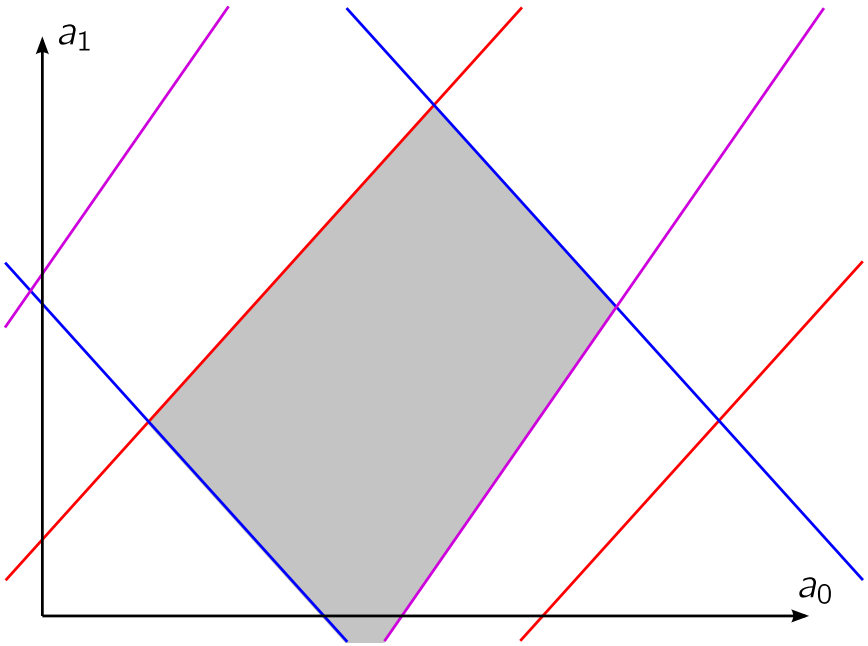
$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i \, x_0^i \\ \sum_{i=0}^{n-1} a_i \, x_0^i & \leq & f(x_0) + \varepsilon_{\text{target}} \end{cases}$$

$$\begin{cases} f(x_1) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i \, x_1^i \\ \sum_{i=0}^{n-1} a_i \, x_1^i & \leq & f(x_1) + \varepsilon_{\text{target}} \end{cases}$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0, x_1, x_2 \in [a, b]$.



$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_0^i \\ \sum_{i=0}^{n-1} a_i x_0^i \leq f(x_0) + \varepsilon_{\text{target}} \end{cases}$$

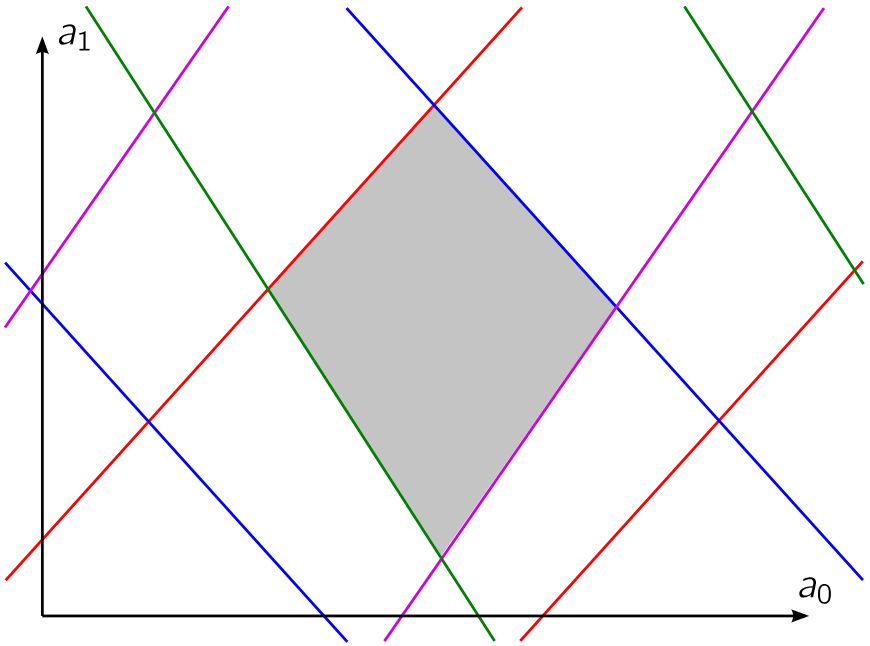
$$\begin{cases} f(x_1) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_1^i \\ \sum_{i=0}^{n-1} a_i x_1^i \leq f(x_1) + \varepsilon_{\text{target}} \end{cases}$$

$$\begin{cases} f(x_2) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_2^i \\ \sum_{i=0}^{n-1} a_i x_2^i \leq f(x_2) + \varepsilon_{\text{target}} \end{cases}$$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0, x_1, x_2, x_3 \in [a, b]$.



$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_0^i \\ \sum_{i=0}^{n-1} a_i x_0^i \leq f(x_0) + \varepsilon_{\text{target}} \end{cases}$$

$$\begin{cases} f(x_1) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_1^i \\ \sum_{i=0}^{n-1} a_i x_1^i \leq f(x_1) + \varepsilon_{\text{target}} \end{cases}$$

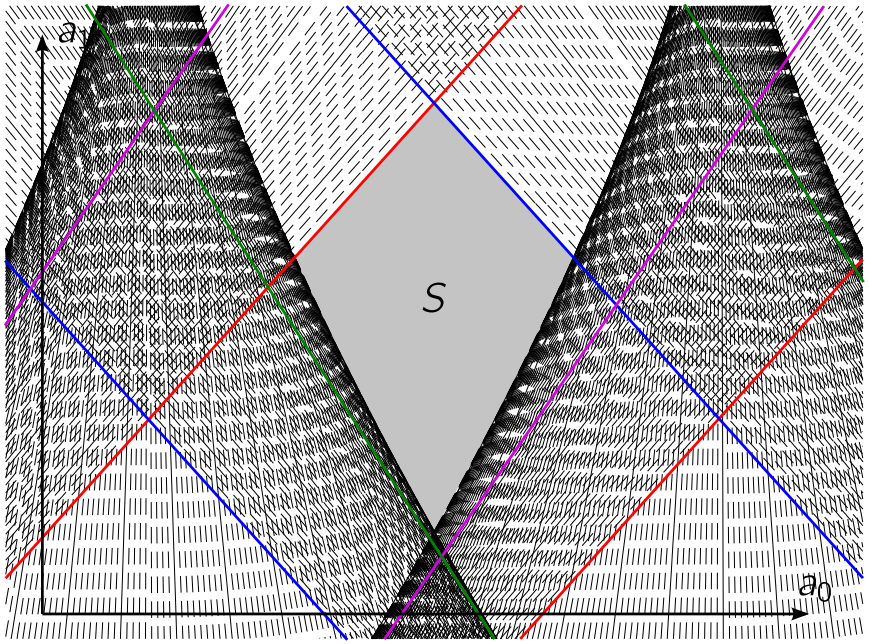
$$\begin{cases} f(x_2) - \varepsilon_{\text{target}} \leq \sum_{i=0}^{n-1} a_i x_2^i \\ \sum_{i=0}^{n-1} a_i x_2^i \leq f(x_2) + \varepsilon_{\text{target}} \end{cases}$$

$\ldots$

Construction of the polytope

$$S = \{(a_0, \ldots, a_n) \in \mathbb{R}^{n+1} \text{ s.t. } \|p - f\|_\infty \leq \varepsilon_{\text{target}}\}.$$

Pick $x_0, x_1, x_2, x_3, \ldots \in [a, b]$.



$$\begin{cases} f(x_0) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i\, x_0^i \\ \sum_{i=0}^{n-1} a_i\, x_0^i & \leq & f(x_0) + \varepsilon_{\text{target}} \end{cases}$$

$$\begin{cases} f(x_1) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i\, x_1^i \\ \sum_{i=0}^{n-1} a_i\, x_1^i & \leq & f(x_1) + \varepsilon_{\text{target}} \end{cases}$$

$$\begin{cases} f(x_2) - \varepsilon_{\text{target}} & \leq & \sum_{i=0}^{n-1} a_i\, x_2^i \\ \sum_{i=0}^{n-1} a_i\, x_2^i & \leq & f(x_2) + \varepsilon_{\text{target}} \end{cases}$$

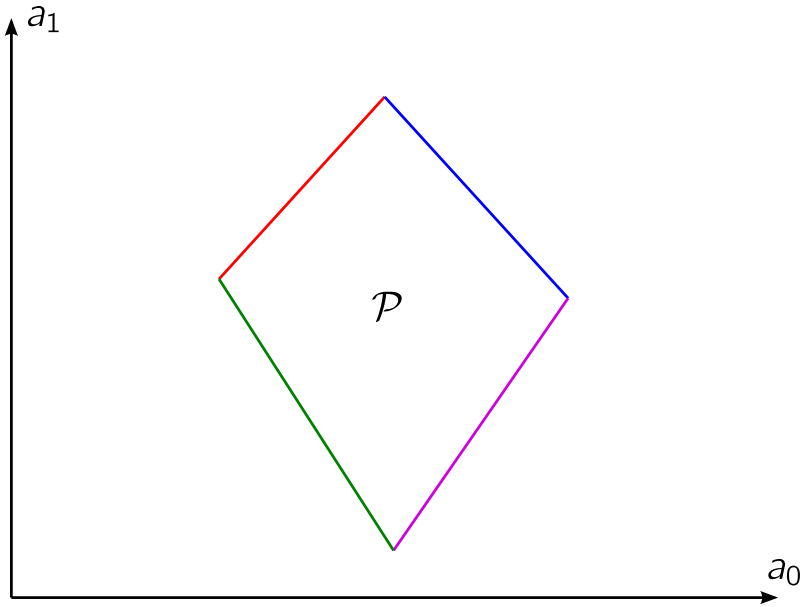
$\ldots$

Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.

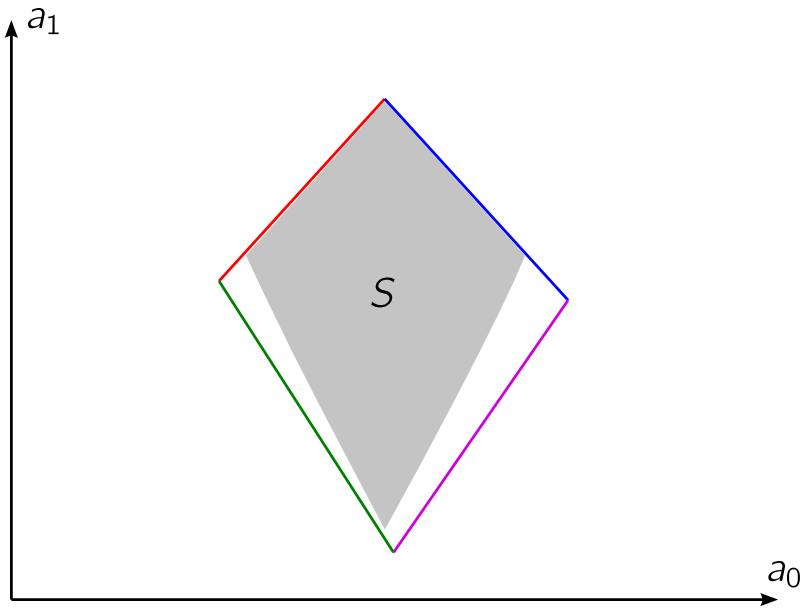
Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.



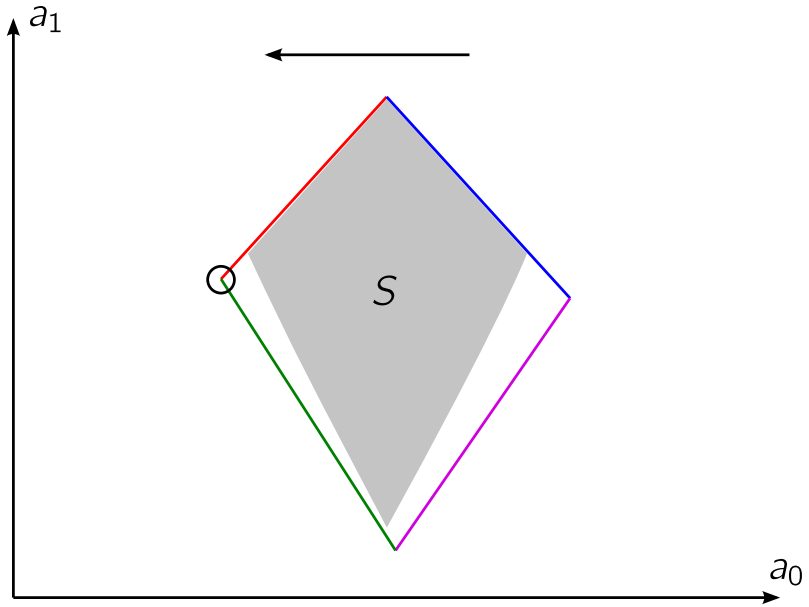
Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.



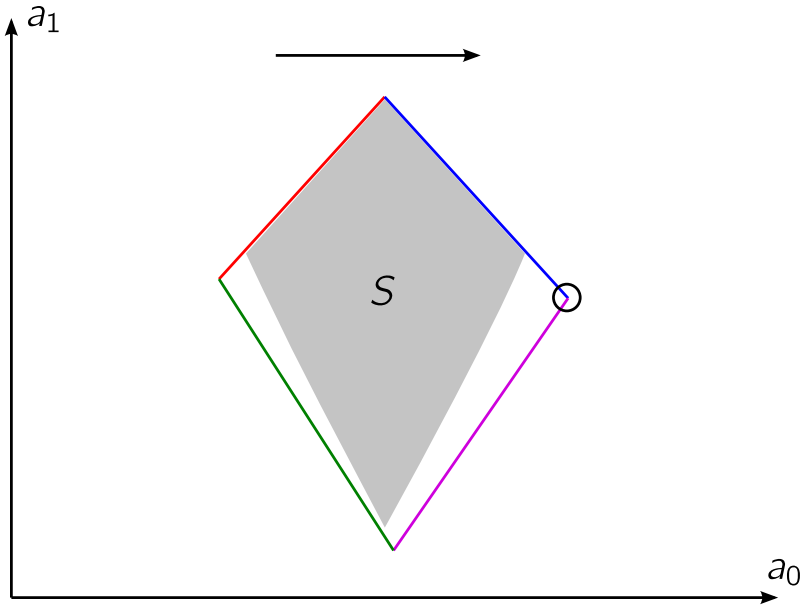
Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.
- Projections are performed by linear programming (simplex).



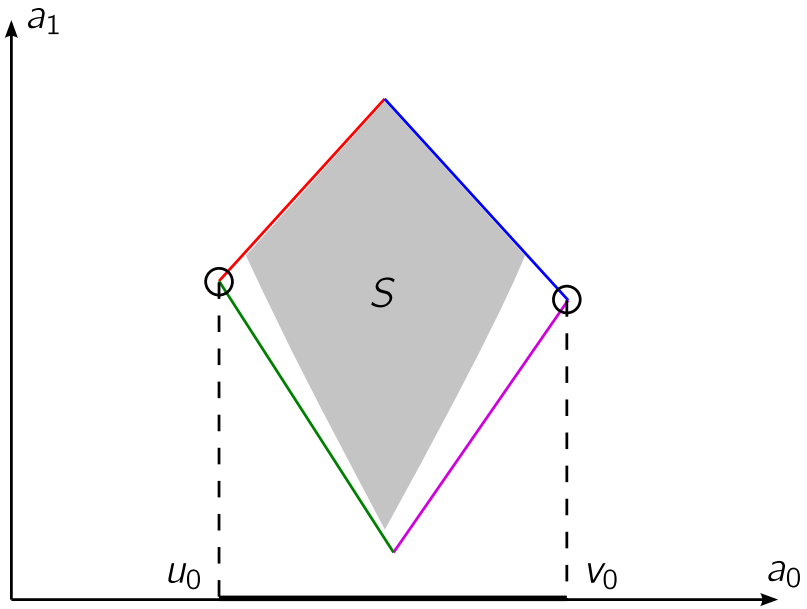
Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.
- Projections are performed by linear programming (simplex).



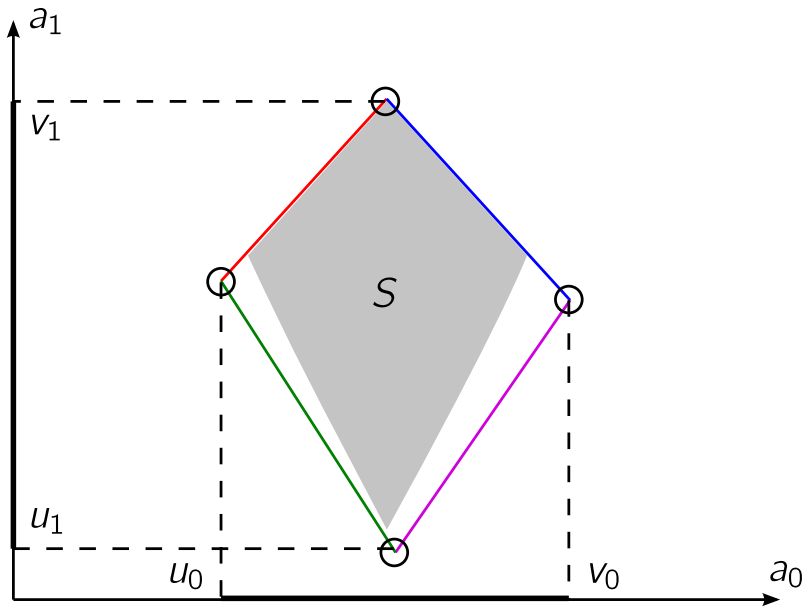
Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.
- Projections are performed by linear programming (simplex).



Construction of the polytope (II)

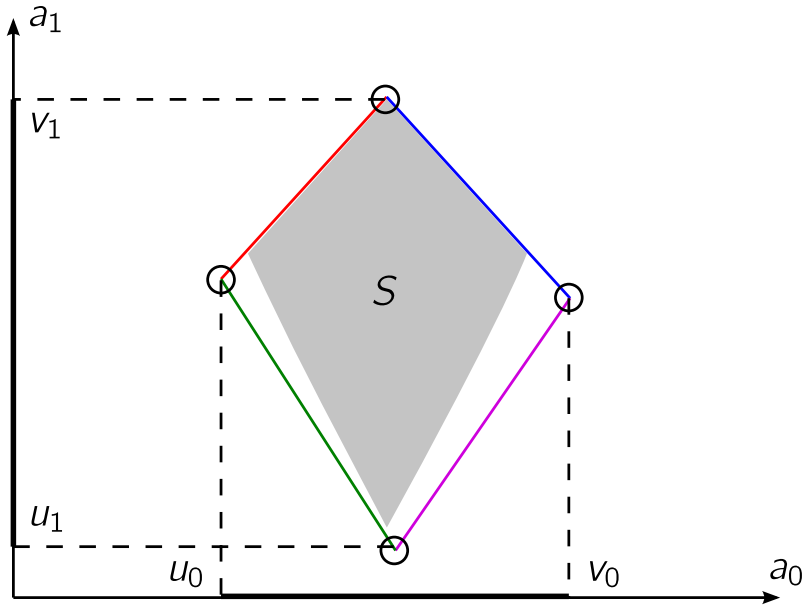
- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.
- Projections are performed by linear programming (simplex).
- They give an enclosure for each coefficient : $a_i \in [u_i, v_i]$.



Construction of the polytope (II)

- We just keep a finite number of points $x_0, \ldots, x_d$.
- The corresponding set of coefficients is a polytope $\mathcal{P}$ of $\mathbb{R}^{n+1}$.
- Projections are performed by linear programming (simplex).
- They give an enclosure for each coefficient : $a_i \in [u_i, v_i]$.

$$\begin{aligned} \|p - f\|_\infty \leq \varepsilon_{\text{target}} &\Rightarrow (a_0, \ldots, a_n) \in S \\ &\Rightarrow (a_0, \ldots, a_n) \in \mathcal{P} \\ &\Rightarrow \forall i,\ a_i \in [u_i, v_i]. \end{aligned}$$



Worked example (beginning)

- Example given by John Harrison (Intel Corp.) when he came to Lyon.

Worked example (beginning)

- Example given by John Harrison (Intel Corp.) when he came to Lyon.
- He asked for a polynomial minimising the absolute error:
  - approximating $f : x \mapsto \frac{2^x - 1}{x}$;
  - on $[a, b] = [-1/16, 1/16]$;
  - with a polynomial of degree 9;

Worked example (beginning)

- Example given by John Harrison (Intel Corp.) when he came to Lyon.
- He asked for a polynomial minimising the absolute error:
  - approximating $f : x \mapsto \frac{2^x - 1}{x}$;
  - on $[a, b] = [-1/16, 1/16]$;
  - with a polynomial of degree 9;
  - coefficient a_0: 129 bits of precision;
  - other coefficients: 64 bits of precision.

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\text{target}} = 400\,\mathrm{e}{-25}$?

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\text{target}} = 400\,\mathrm{e}{-25}$?
  - Polytope constructed with 30 Chebyshev points.

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\text{target}} = 400\,\mathrm{e}{-25}$?
  - Polytope constructed with 30 Chebyshev points.
  - Projection on a_1: the interval $[u_1, v_1]$ does not contain any double extended number!

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\text{target}} = 400\,\mathrm{e}{-25}$?
  - Polytope constructed with 30 Chebyshev points.
  - Projection on a_1: the interval $[u_1, v_1]$ does not contain any double extended number!
  - Hence $\varepsilon_{\text{target}}$ **cannot** be achieved:

$$400\,\mathrm{e}{-25} \leq \varepsilon_{\text{opt}} \leq 4035\,\mathrm{e}{-25}.$$

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\mathrm{target}} = 400\,\mathrm{e}{-25}$?
  - Polytope constructed with 30 Chebyshev points.
  - Projection on a_1: the interval $[u_1, v_1]$ does not contain any double extended number!
  - Hence $\varepsilon_{\mathrm{target}}$ **cannot** be achieved:

$$400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 4035\,\mathrm{e}{-25}.$$

- Generally $[u_i, v_i]$ is so thin that the exponent e_i is fixed.

Worked example (II)

- Minimax (real coefficients): $\varepsilon^\star \simeq 7.9\,\mathrm{e}{-25}$.
- Error of the rounded minimax $\widehat{\varepsilon} \simeq 4035\,\mathrm{e}{-25}$.
- Huge gap! Can we achieve an intermediate error of $\varepsilon_{\text{target}} = 400\,\mathrm{e}{-25}$?
  - Polytope constructed with 30 Chebyshev points.
  - Projection on a_1: the interval $[u_1, v_1]$ does not contain any double extended number!
  - Hence $\varepsilon_{\text{target}}$ **cannot** be achieved:

$$400\,\mathrm{e}{-25} \leq \varepsilon_{\text{opt}} \leq 4035\,\mathrm{e}{-25}.$$

- Generally $[u_i, v_i]$ is so thin that the exponent e_i is fixed.

- Improving the value $4035\,\mathrm{e}{-25}$: we need a fast (possibly heuristic) algorithm.

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

- Reminder about approximation theory.

  $\hookrightarrow$ finds good approximation polynomials with real coefficients.

- Polynomial p with floating-point coefficients.
  - Linear programming.
    $\hookrightarrow$ gets useful informations on the structure of p.
  - Euclidean lattices.
    $\hookrightarrow$ computes a very good polynomial p.

Formalisation of the problem

- Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

Formalisation of the problem

- Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- A simplification: guess the value of each e_i.

  $\hookrightarrow$ heuristic validated by means of projections.

Formalisation of the problem

- Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- A simplification: guess the value of each e_i.

  $\hookrightarrow$ heuristic validated by means of projections.

- Once e_i is guessed, we need to find $m_i \in \mathbb{Z}$ such that

$$\left\| f(x) - \sum_{i=0}^{n} m_i \cdot 2^{e_i} x^i \right\|_\infty$$

  is minimal.

Description of the lattice-based method

Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

Description of the lattice-based method

Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- We assume that p looks like $p^\star$:

Description of the lattice-based method

Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- We assume that p looks like $p^\star$:
  - we choose $n+1$ points $z_0, \cdots, z_n$ in $[a, b]$;

Description of the lattice-based method

Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- We assume that p looks like $p^\star$:
  - we choose $n+1$ points $z_0, \cdots, z_n$ in $[a, b]$;
  - we search $m_0, \cdots, m_n$ such that for all i

$$p(z_i) = m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} z_i + \cdots + m_n \cdot 2^{e_n} z_i^n \simeq p^\star(z_i) \quad .$$

Description of the lattice-based method

Our goal: find p approximating f and with the following form

$$m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} X + \cdots + m_n \cdot 2^{e_n} X^n.$$

- We assume that p looks like $p^\star$:
  - we choose $n+1$ points $z_0, \cdots, z_n$ in $[a, b]$;
  - we search $m_0, \cdots, m_n$ such that for all i

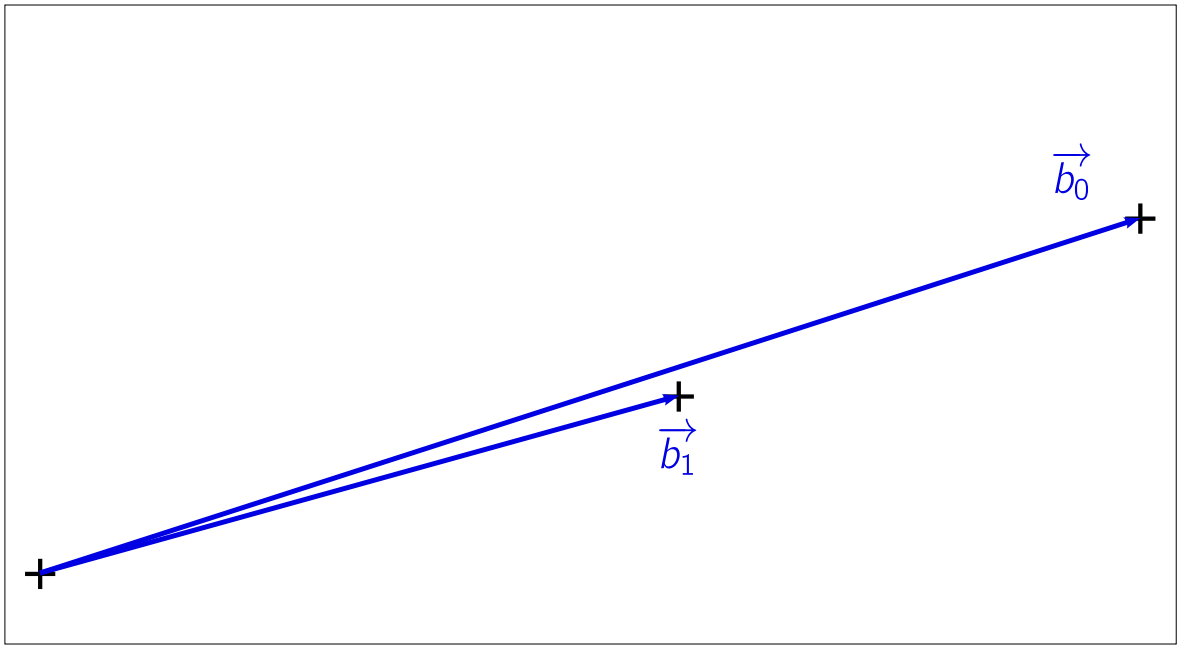
$$p(z_i) = m_0 \cdot 2^{e_0} + m_1 \cdot 2^{e_1} z_i + \cdots + m_n \cdot 2^{e_n} z_i^n \simeq p^\star(z_i) \quad .$$

  - Rewritten with vectors:

$$\underbrace{m_0 \begin{pmatrix} 2^{e_0} \\ 2^{e_0} \\ \vdots \\ 2^{e_0} \end{pmatrix} + \cdots + m_n \begin{pmatrix} 2^{e_n} \cdot z_0^n \\ 2^{e_n} \cdot z_1^n \\ \vdots \\ 2^{e_n} \cdot z_n^n \end{pmatrix}}_{\Gamma \text{ of the form } \mathbb{Z}\overrightarrow{b_0} + \mathbb{Z}\overrightarrow{b_1} + \cdots + \mathbb{Z}\overrightarrow{b_n}} \simeq \underbrace{\begin{pmatrix} p^\star(z_0) \\ p^\star(z_1) \\ \vdots \\ p^\star(z_n) \end{pmatrix}}_{\overrightarrow{v} \in \mathbb{R}^{n+1}} \quad .$$

Notions about lattices

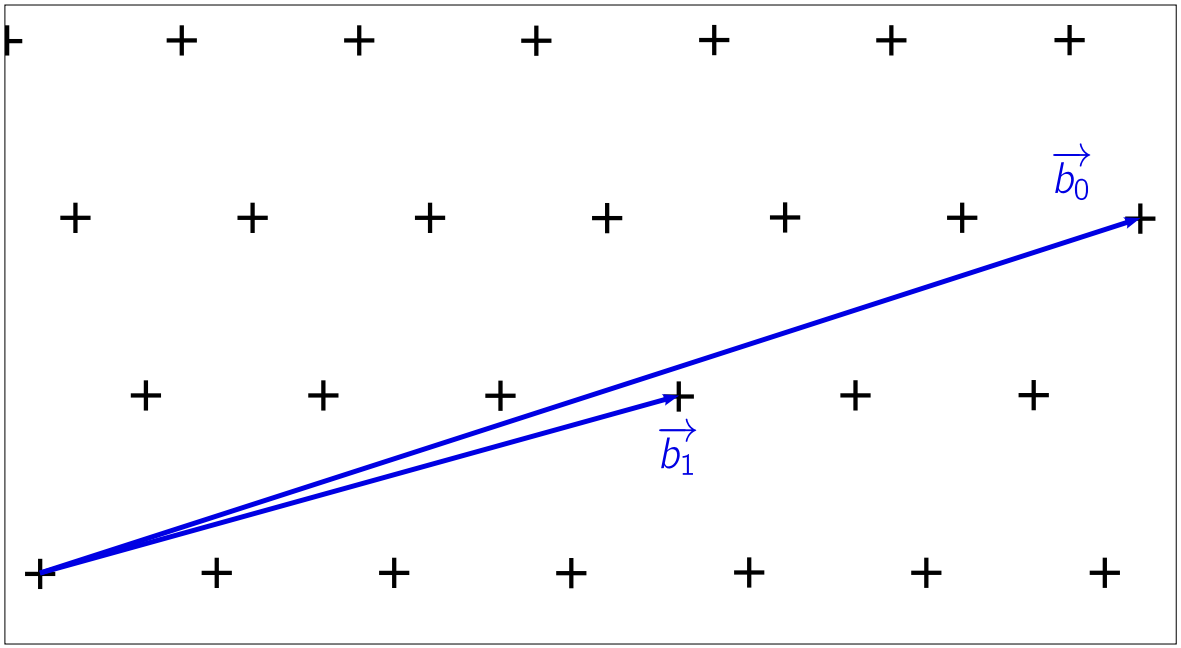
Let $(\overrightarrow{b_0}, \cdots, \overrightarrow{b_n})$ be a basis of a real vector space.



Notions about lattices

Let $(\overrightarrow{b_0}, \cdots, \overrightarrow{b_n})$ be a basis of a real vector space. The set of all integer combinations of the $\overrightarrow{b_i}$ is called a lattice:

$$\Gamma = \mathbb{Z}\overrightarrow{b_0} + \mathbb{Z}\overrightarrow{b_1} + \cdots + \mathbb{Z}\overrightarrow{b_n} \quad .$$

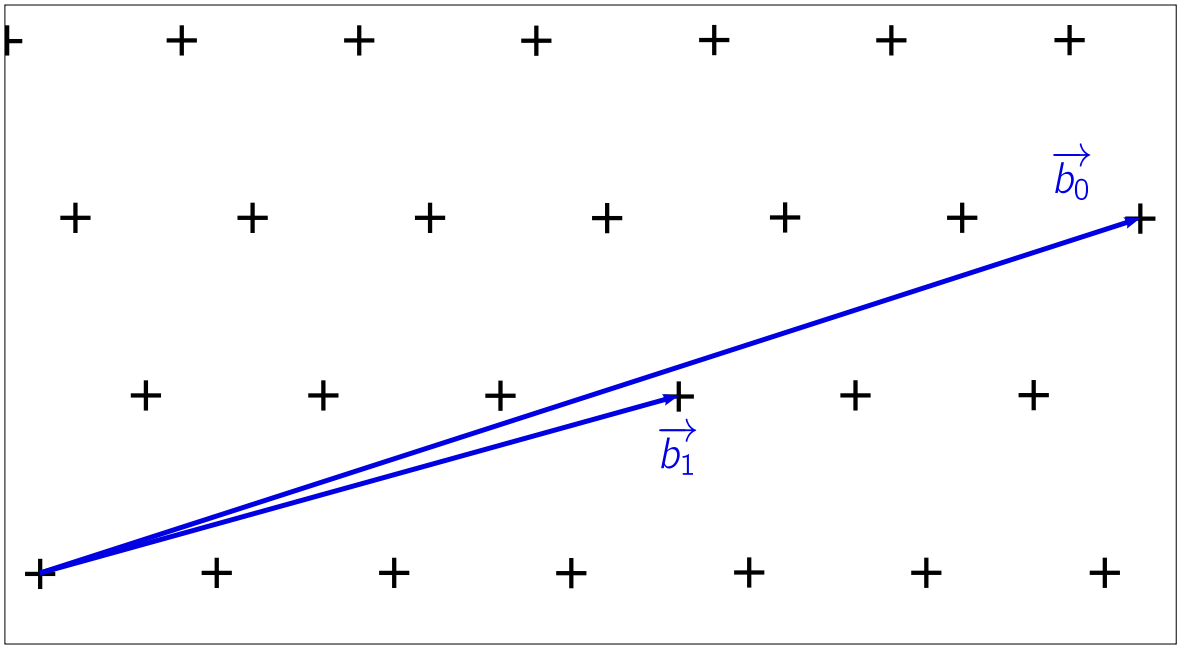


Notions about lattices

Let $(\overrightarrow{b_0}, \cdots, \overrightarrow{b_n})$ be a basis of a real vector space. The set of all integer combinations of the $\overrightarrow{b_i}$ is called a lattice:

$$\Gamma = \mathbb{Z}\overrightarrow{b_0} + \mathbb{Z}\overrightarrow{b_1} + \cdots + \mathbb{Z}\overrightarrow{b_n} \quad .$$

In general, a lattice has infinitely many bases.

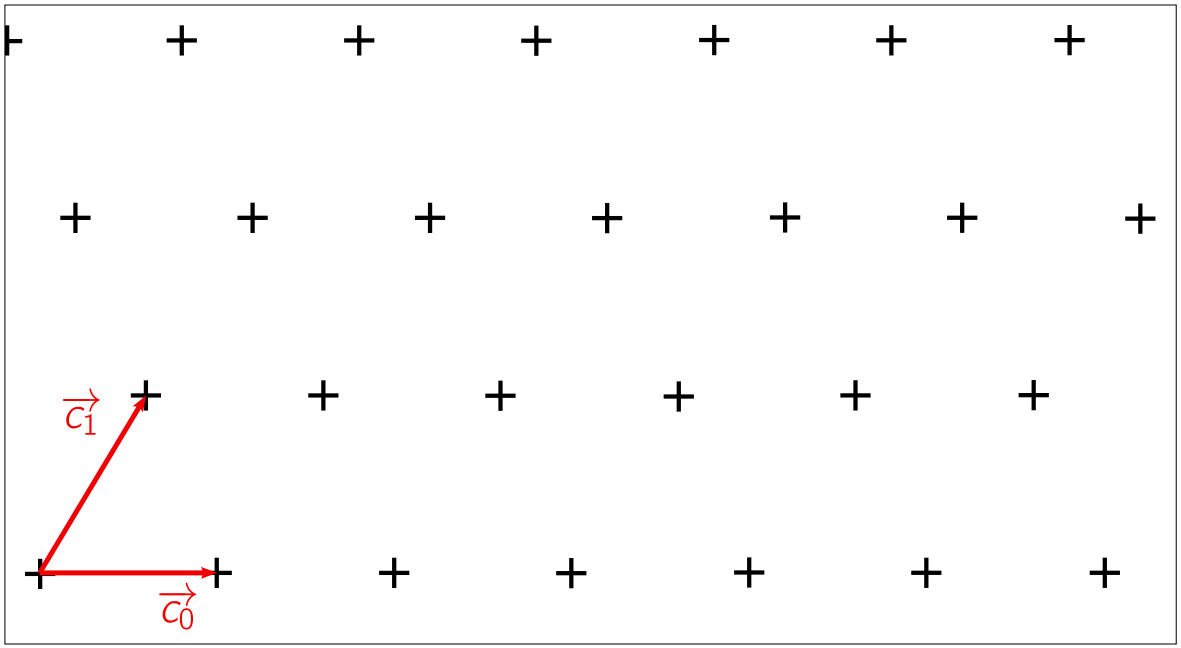


Notions about lattices

Let $(\overrightarrow{b_0}, \cdots, \overrightarrow{b_n})$ be a basis of a real vector space. The set of all integer combinations of the $\overrightarrow{b_i}$ is called a lattice:

$$\Gamma = \mathbb{Z}\overrightarrow{b_0} + \mathbb{Z}\overrightarrow{b_1} + \cdots + \mathbb{Z}\overrightarrow{b_n} \quad .$$

In general, a lattice has infinitely many bases.

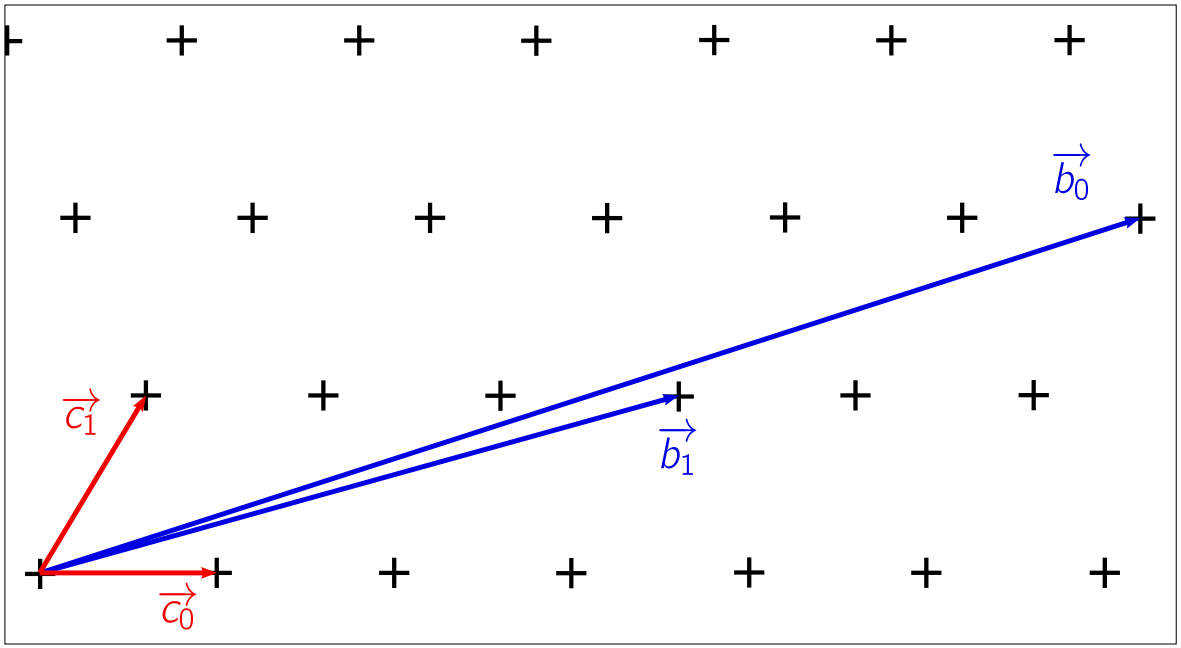


Notions about lattices

Let $(\overrightarrow{b_0}, \cdots, \overrightarrow{b_n})$ be a basis of a real vector space. The set of all integer combinations of the $\overrightarrow{b_i}$ is called a lattice:

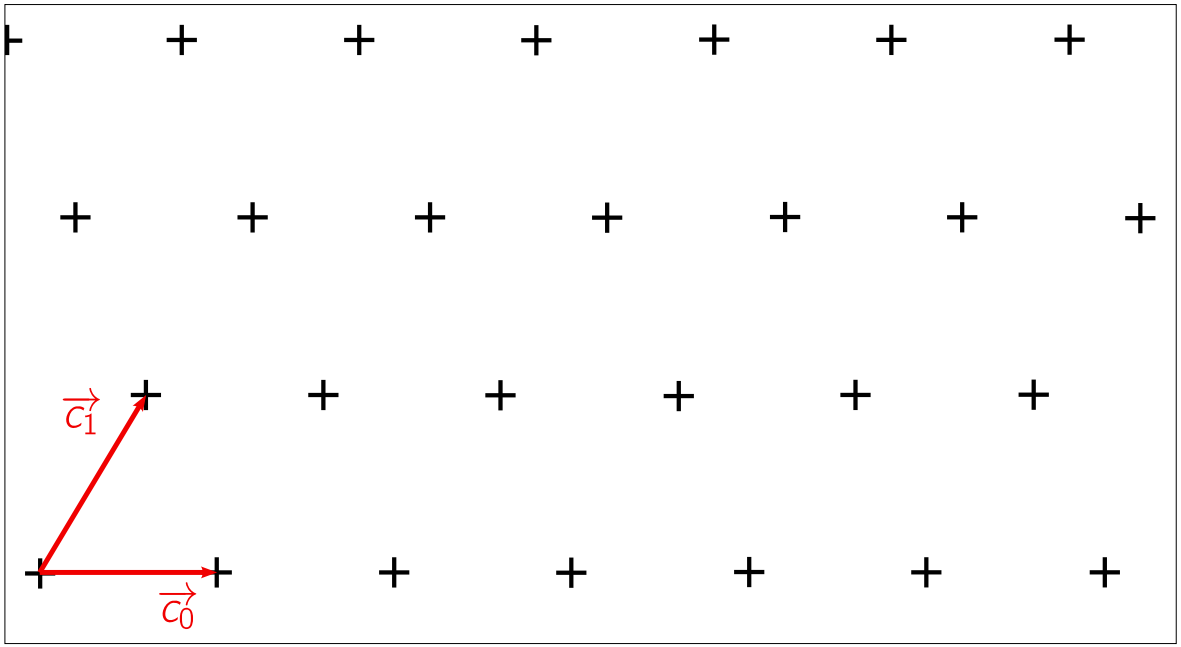
$$\Gamma = \mathbb{Z}\overrightarrow{b_0} + \mathbb{Z}\overrightarrow{b_1} + \cdots + \mathbb{Z}\overrightarrow{b_n} \quad .$$

In general, a lattice has infinitely many bases.



Notions about lattices (2)

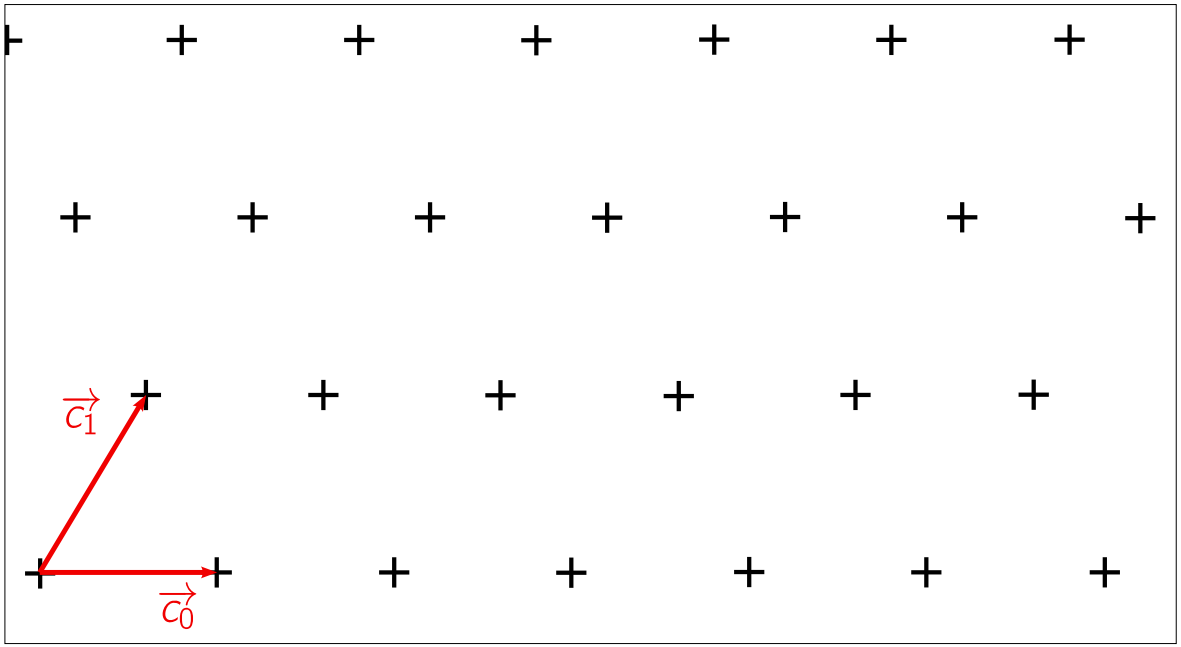
Algorithmic problems:



Notions about lattices (2)

Algorithmic problems:

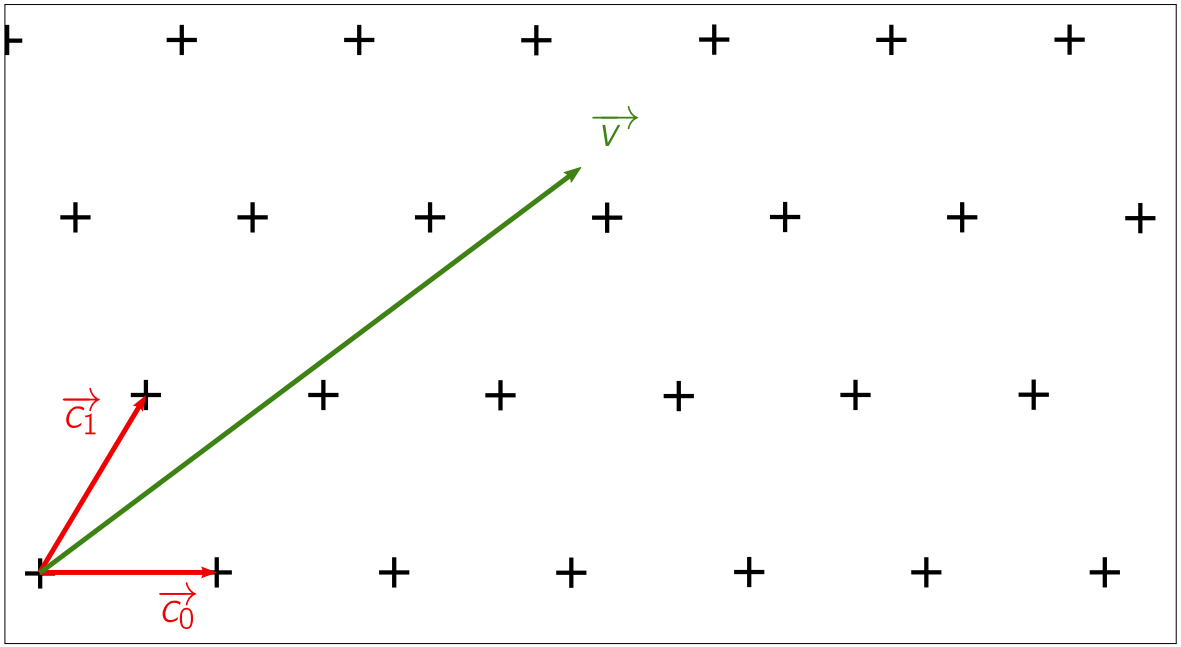
- shortest vector problem (SVP);



Notions about lattices (2)

Algorithmic problems:

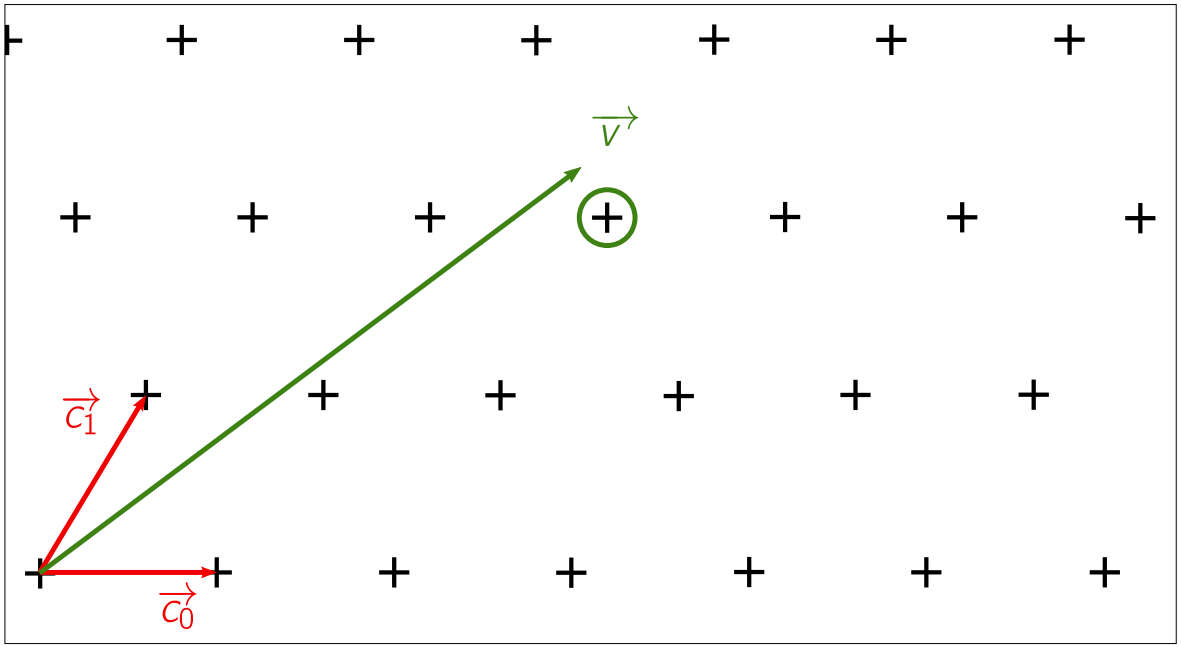
- shortest vector problem (SVP);
- closest vector problem (CVP).



Notions about lattices (2)

Algorithmic problems:

- shortest vector problem (SVP);
- closest vector problem (CVP).

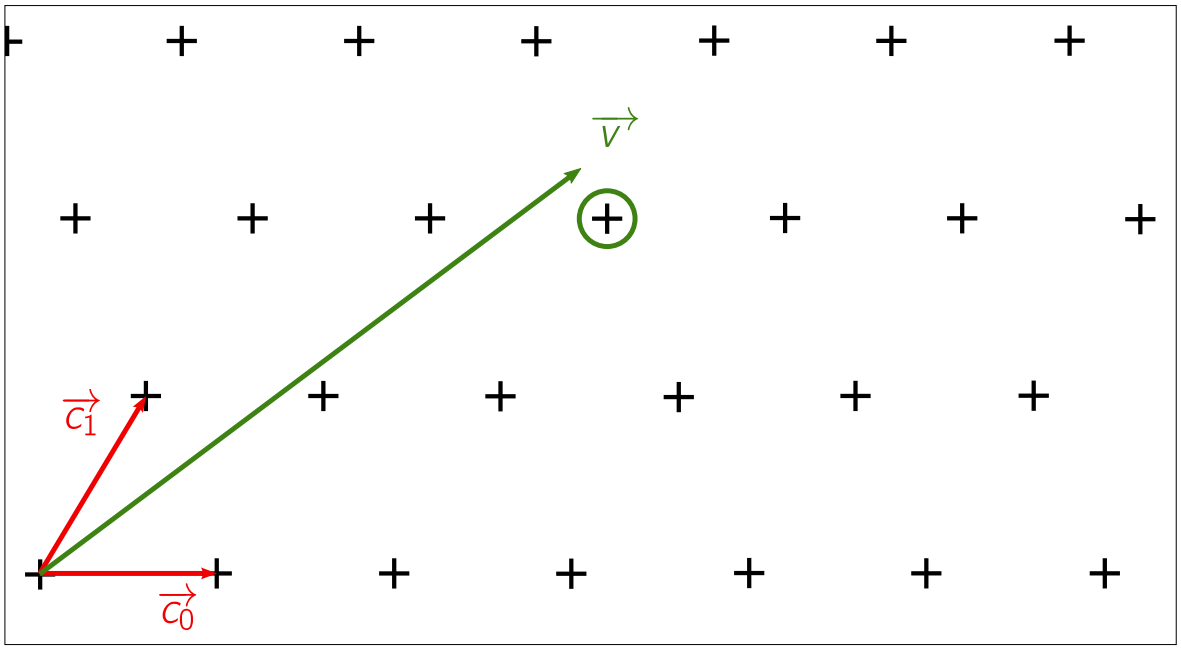


Notions about lattices (2)

Algorithmic problems:

- shortest vector problem (SVP);
- closest vector problem (CVP).

LLL algorithm: Lenstra, Lenstra Jr. and Lovász (1982).

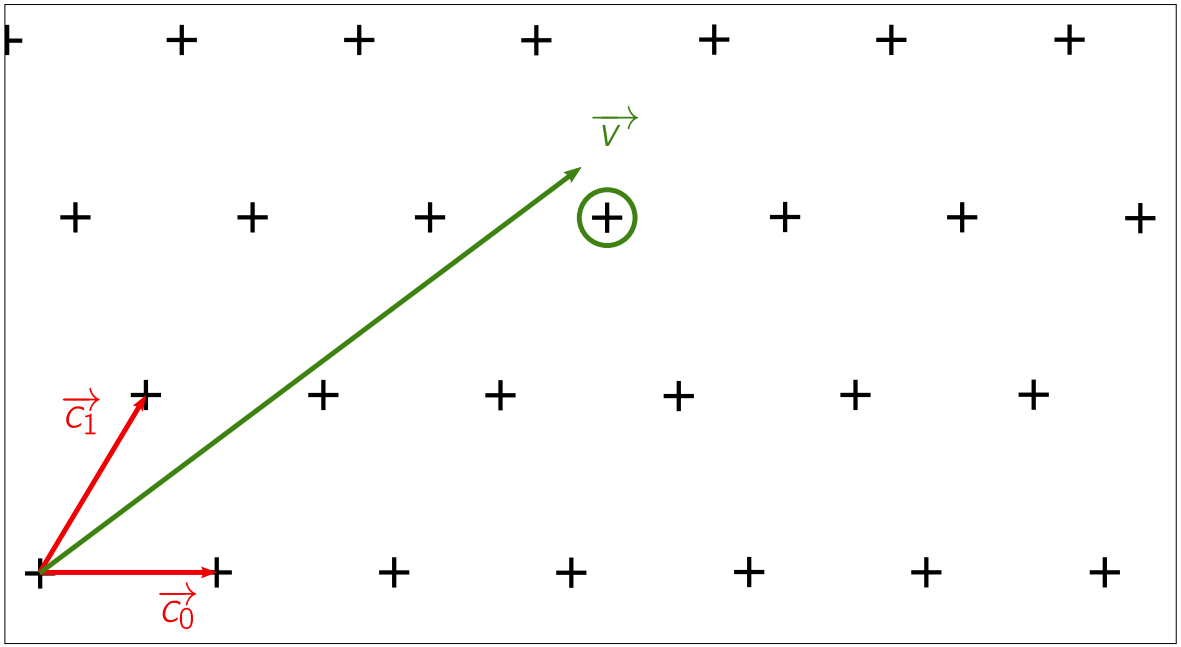


Notions about lattices (2)

Algorithmic problems:

- shortest vector problem (SVP);
- closest vector problem (CVP).

LLL algorithm: finds pretty short vectors in polynomial time.

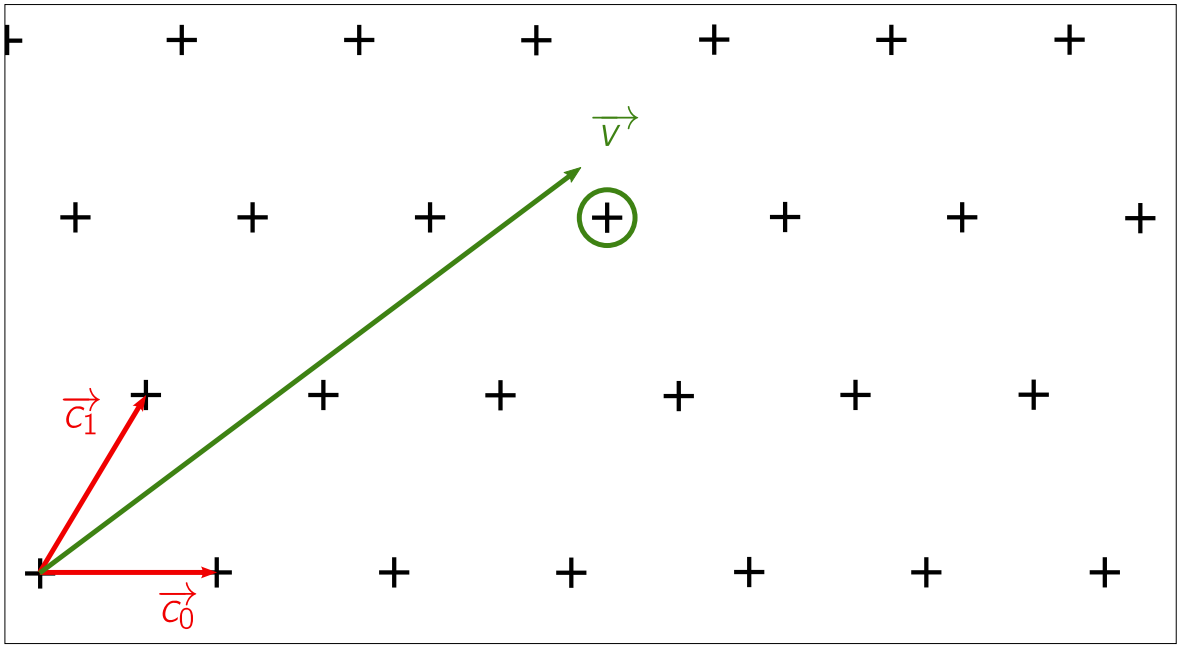


Notions about lattices (2)

Algorithmic problems:

- shortest vector problem (SVP);
- closest vector problem (CVP).

LLL algorithm: used by Babai to solve an approximation of CVP.



A reminder of the formalisation

Remember our formalisation:

$$m_0 \underbrace{\begin{pmatrix} 2^{e_0} \\ 2^{e_0} \\ \vdots \\ 2^{e_0} \end{pmatrix}}_{\overrightarrow{b_0}} + \cdots + m_n \underbrace{\begin{pmatrix} 2^{e_n} \cdot z_0^n \\ 2^{e_n} \cdot z_1^n \\ \vdots \\ 2^{e_n} \cdot z_n^n \end{pmatrix}}_{\overrightarrow{b_n}} \simeq \underbrace{\begin{pmatrix} f(z_0) \\ f(z_1) \\ \vdots \\ f(z_n) \end{pmatrix}}_{\overrightarrow{v}} .$$

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

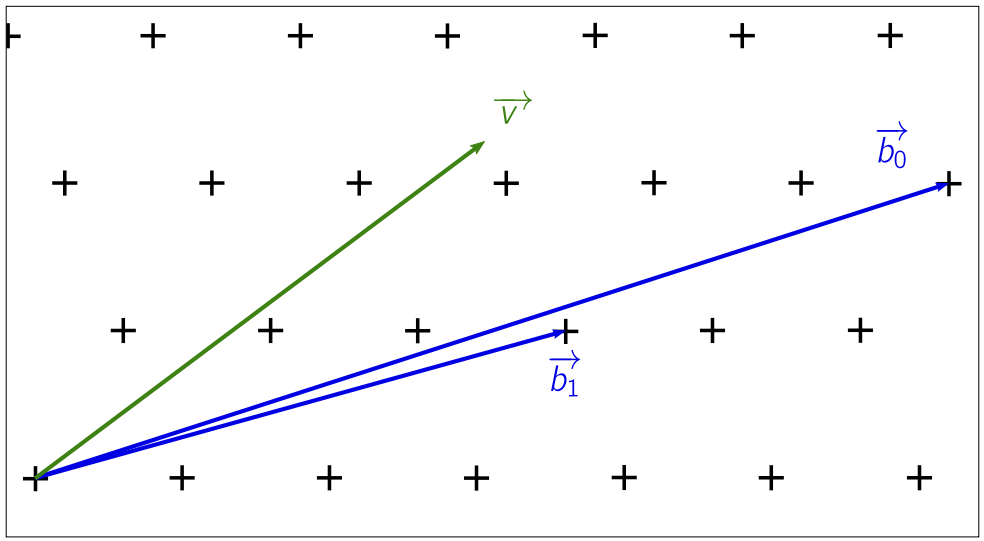
1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
4. Construct the vectors b_j and the vector v.

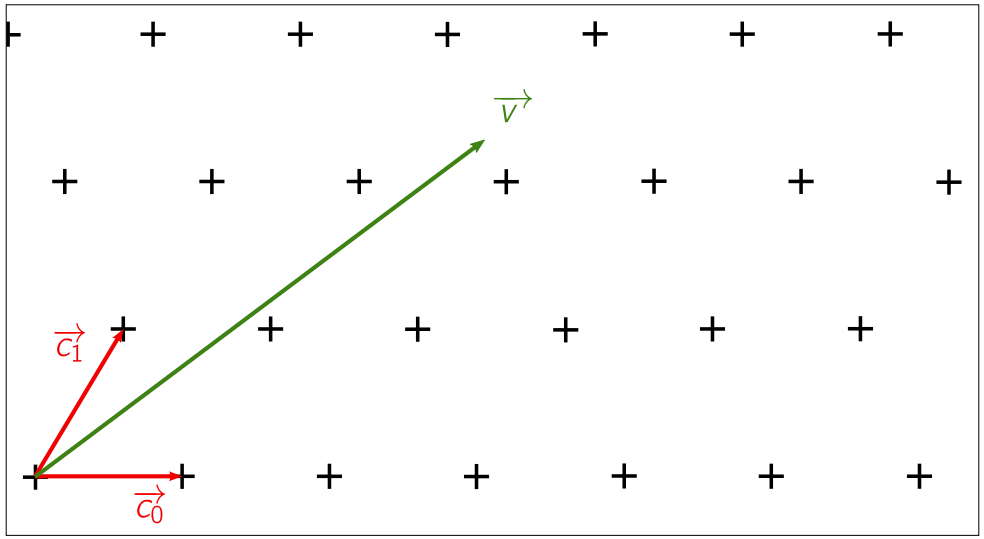


The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
5. LLL-reduce the lattice: we get a basis $(c_0, \ldots, c_n)$.

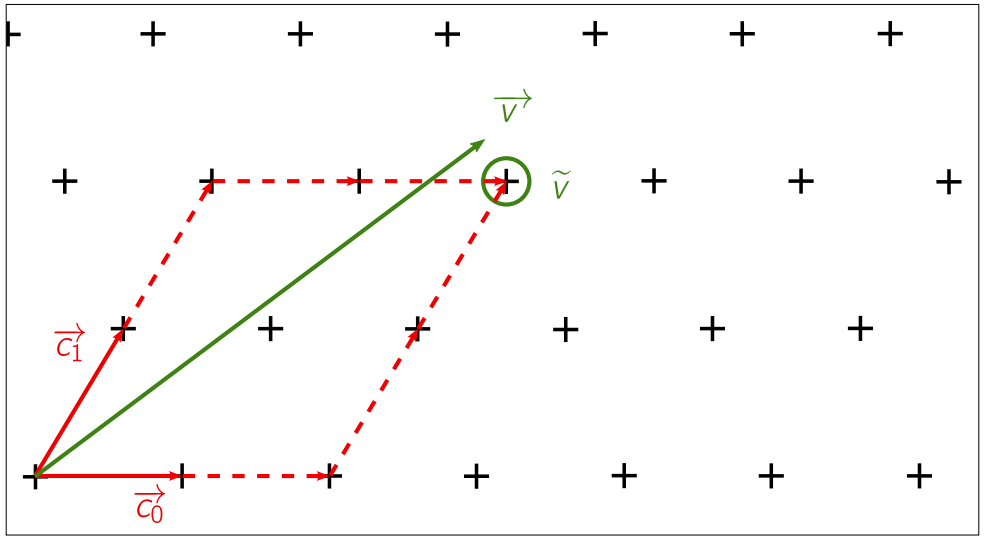


The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
6. Use Babai's algorithm to find a vector $\tilde{v}$ close to v.



The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
4. Construct the vectors b_j and the vector v.
5. LLL-reduce the lattice: we get a basis $(c_0, \ldots, c_n)$.
6. Use Babai's algorithm to find a vector $\tilde{v}$ close to v.

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
4. Construct the vectors b_j and the vector v.
5. LLL-reduce the lattice: we get a basis $(c_0, \ldots, c_n)$.
6. Use Babai's algorithm to find a vector $\tilde{v}$ close to v.
7. Replace each c_j by its expression with the vectors b_k: hence getting the coefficients m_j of $\tilde{v}$ in the basis $(b_0, \ldots, b_n)$.

The algorithm

Input: f, $[a, b]$, n, list of desired floating-point formats t_i.

Output: $p = \sum m_i 2^{e_i} X^i$ (m_i with t_i bits).

1. Compute the real minimax $p^\star = \sum a_i^\star X^i$.
2. Guess e_i (polytope approach).
3. Choose $z_0, \ldots, z_n$ in $[a, b]$.
4. Construct the vectors b_j and the vector v.
5. LLL-reduce the lattice: we get a basis $(c_0, \ldots, c_n)$.
6. Use Babai's algorithm to find a vector $\tilde{v}$ close to v.
7. Replace each c_j by its expression with the vectors b_k: hence getting the coefficients m_j of $\tilde{v}$ in the basis $(b_0, \ldots, b_n)$.

Return: $p = \sum m_i 2^{e_i} X^i$.

Worked example

- How to choose the points?

Worked example

- How to choose the points?
  - We need $n+1$ points.

Worked example

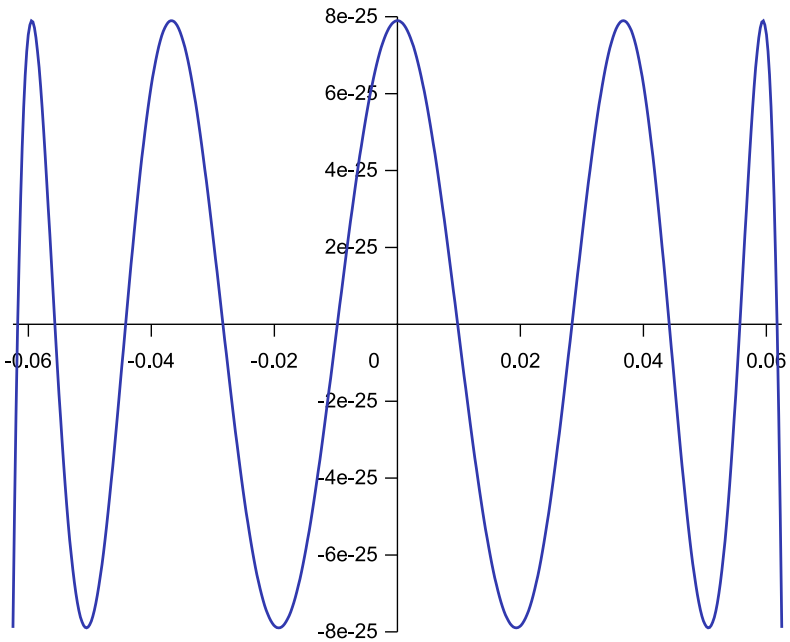
- How to choose the points?

- We need $n+1$ points.
- They should correspond to the interpolation intuition :

$$p(z_i) \simeq p^\star(z_i).$$

Worked example

- How to choose the points?



Graph of the error function $p^\star - f$.

- We need $n+1$ points.
- They should correspond to the interpolation intuition :

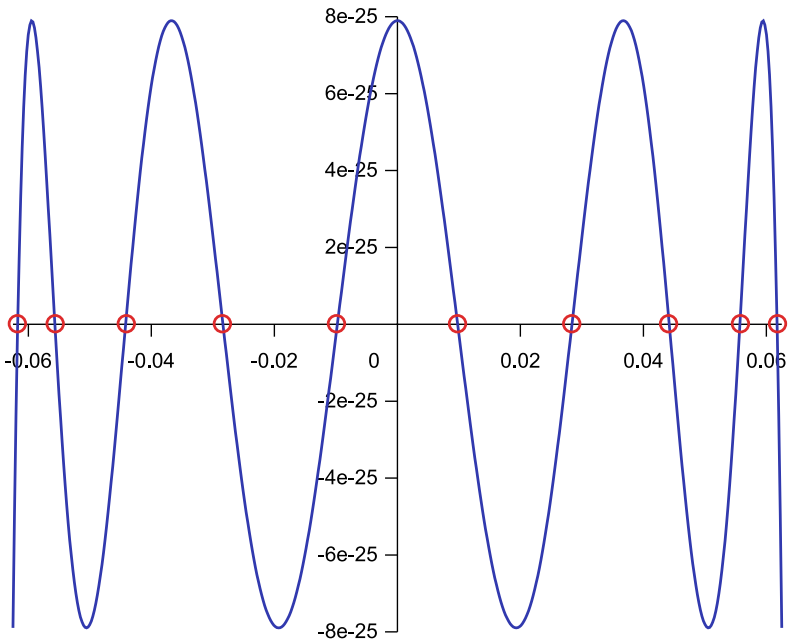
$$p(z_i) \simeq p^\star(z_i).$$

- Idea: if possible take points where

$$p^\star(z_i) = f(z_i).$$

Worked example

- How to choose the points?



Graph of the error function $p^\star - f$.

- We need $n+1$ points.
- They should correspond to the interpolation intuition :

$$p(z_i) \simeq p^\star(z_i).$$

- Idea: if possible take points where

$$p^\star(z_i) = f(z_i).$$

Results

- We get a polynomial p_0 with floating-point coefficients.
- The error of p_0 is $\varepsilon_0 \simeq 532\,\mathrm{e}{-25}$.

Results

- We get a polynomial p_0 with floating-point coefficients.
- The error of p_0 is $\varepsilon_0 \simeq 532\,\mathrm{e}{-25}$.
- Remember that we knew: $400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 4035\,\mathrm{e}{-25}$.

Results

- We get a polynomial p_0 with floating-point coefficients.
- The error of p_0 is $\varepsilon_0 \simeq 532\,\mathrm{e}{-25}$.
- Remember that we knew: $400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 4035\,\mathrm{e}{-25}$.

  $\hookrightarrow$ now we know that

$$400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 532\,\mathrm{e}{-25}.$$

Results

- We get a polynomial p_0 with floating-point coefficients.
- The error of p_0 is $\varepsilon_0 \simeq 532\,\mathrm{e}{-25}$.
- Remember that we knew: $400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 4035\,\mathrm{e}{-25}$.

  $\hookrightarrow$ now we know that

$$400\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq 532\,\mathrm{e}{-25}.$$

- It is even possible to prove that

$$444.02\,\mathrm{e}{-25} \leq \varepsilon_{\mathrm{opt}} \leq \underbrace{444.92\,\mathrm{e}{-25}}_{\text{effectively reached}}.$$

Conclusion

- Several techniques for polynomial approximation:
  - approximation with real coefficients;
  - approximation with floating-point coefficients:
    $\hookrightarrow$ linear programming, euclidean lattices.

Conclusion

- Several techniques for polynomial approximation:
  - approximation with real coefficients;
  - approximation with floating-point coefficients:
    $\hookrightarrow$ linear programming, euclidean lattices.

- Available within Sollya.

Conclusion

- Several techniques for polynomial approximation:
  - approximation with real coefficients;
  - approximation with floating-point coefficients:
    $\hookrightarrow$ linear programming, euclidean lattices.
- Available within Sollya.
- Other topics studied during the thesis:
  - computing automatically a certified bound on $\|p - f\|_\infty$;
  - implementing a function in arbitrary precision.

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.
  - More numerical procedures (interpolation, integration).

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.
  - More numerical procedures (interpolation, integration).
  - Solving differential equations.

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.
  - More numerical procedures (interpolation, integration).
  - Solving differential equations.
  - Linear algebra (inversion of matrices, resolution of linear systems, etc.)

Perspectives

- Apply our techniques to signal processing:

  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.
  - More numerical procedures (interpolation, integration).
  - Solving differential equations.
  - Linear algebra (inversion of matrices, resolution of linear systems, etc.)

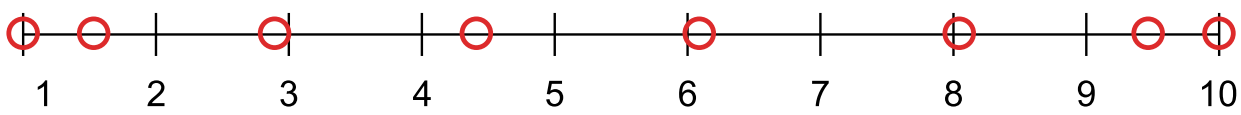
Perspectives

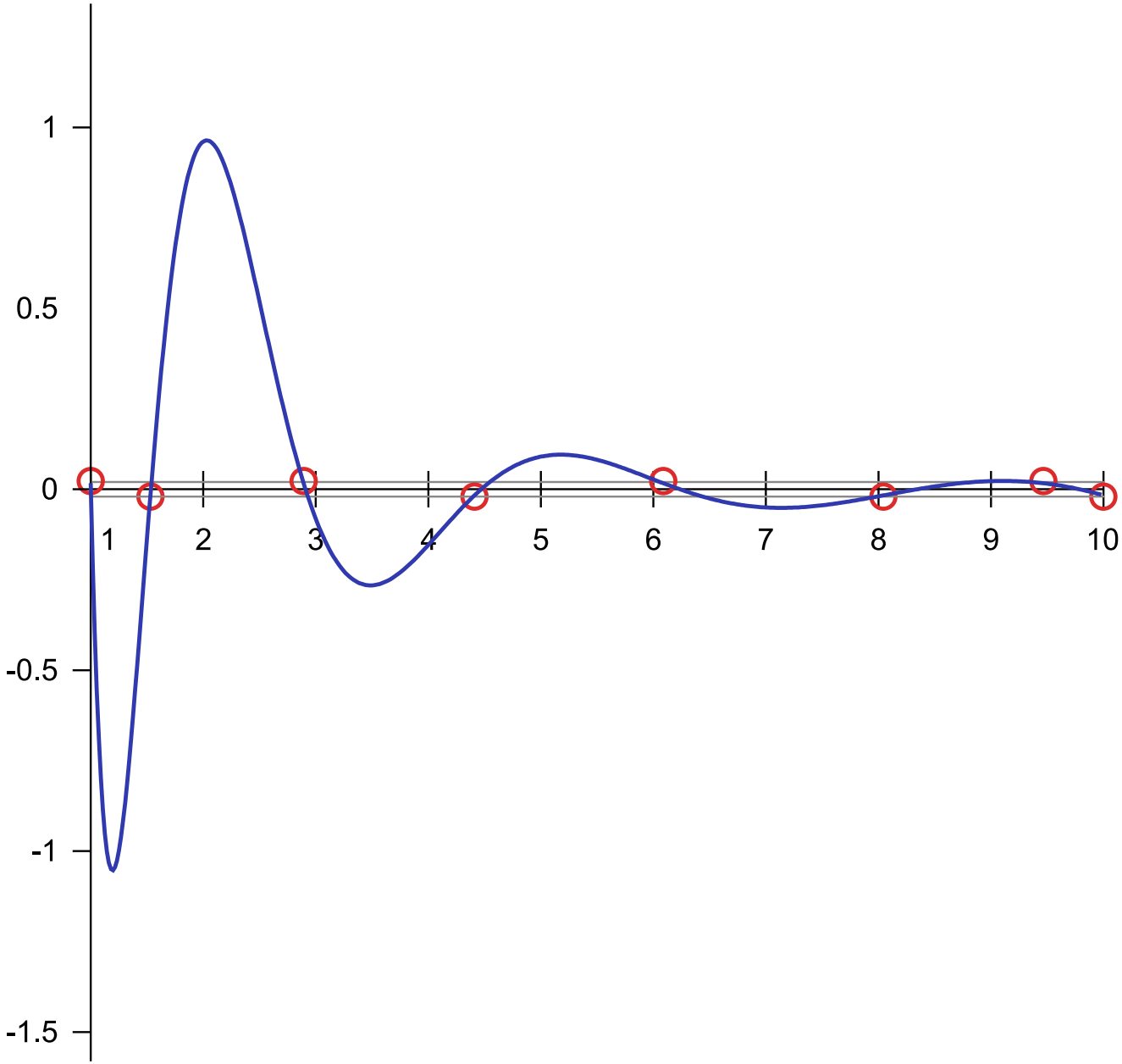
- Apply our techniques to signal processing:

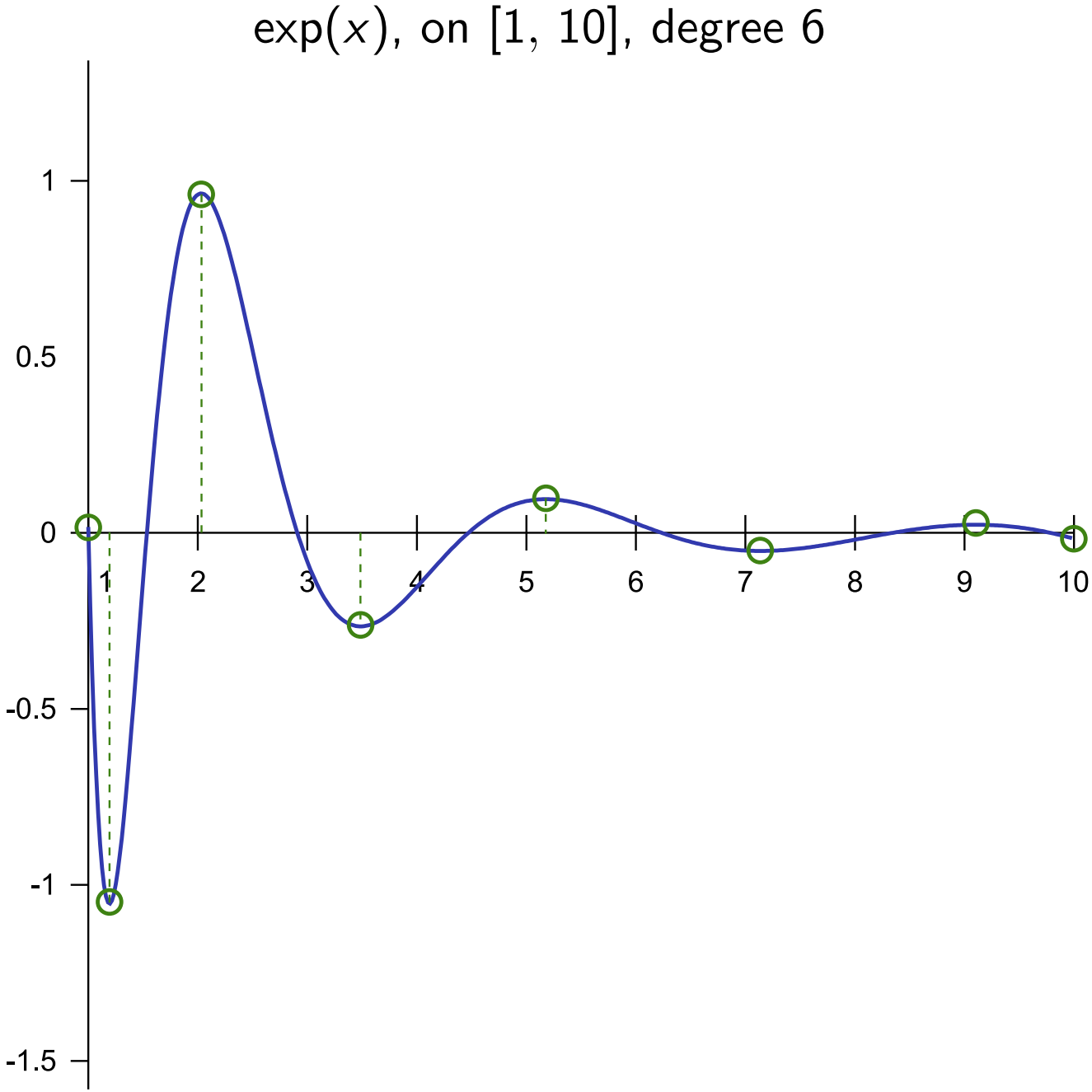
  $\hookrightarrow$ find trigonometric polynomials with floating-point coefficients
- Implementation of functions in arbitrary precision:
  - painful to do manually;
  - goal: automate the implementation;
  - makes it possible to explore new ideas.
- Continue developing Sollya.

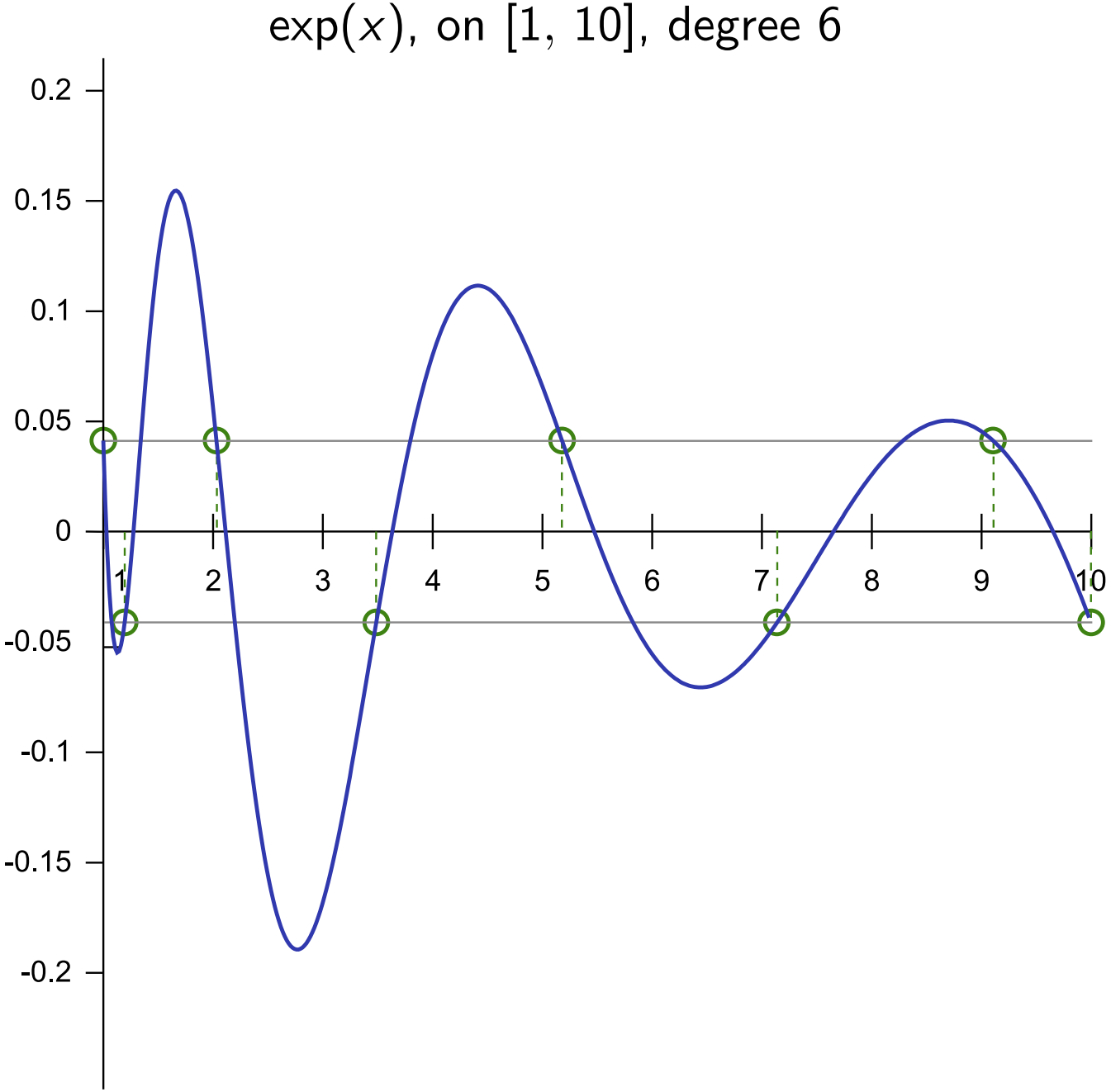
  $\hookrightarrow$ the most important keyword is safety.
  - Support for multivariate functions.
  - More numerical procedures (interpolation, integration).
  - Solving differential equations.
  - Linear algebra (inversion of matrices, resolution of linear systems, etc.)

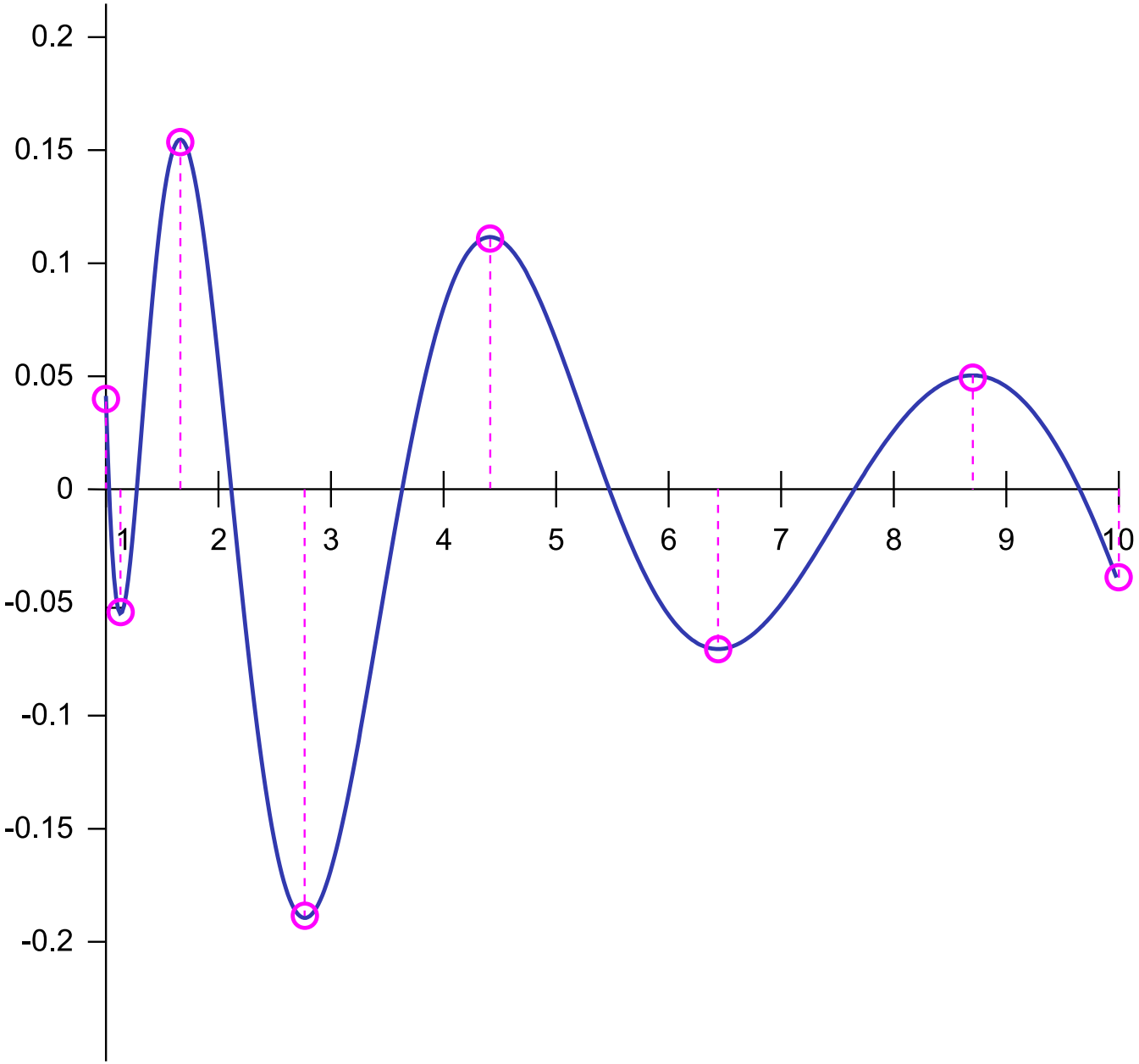
$\exp(x)$, on $[1, 10]$, degree 6

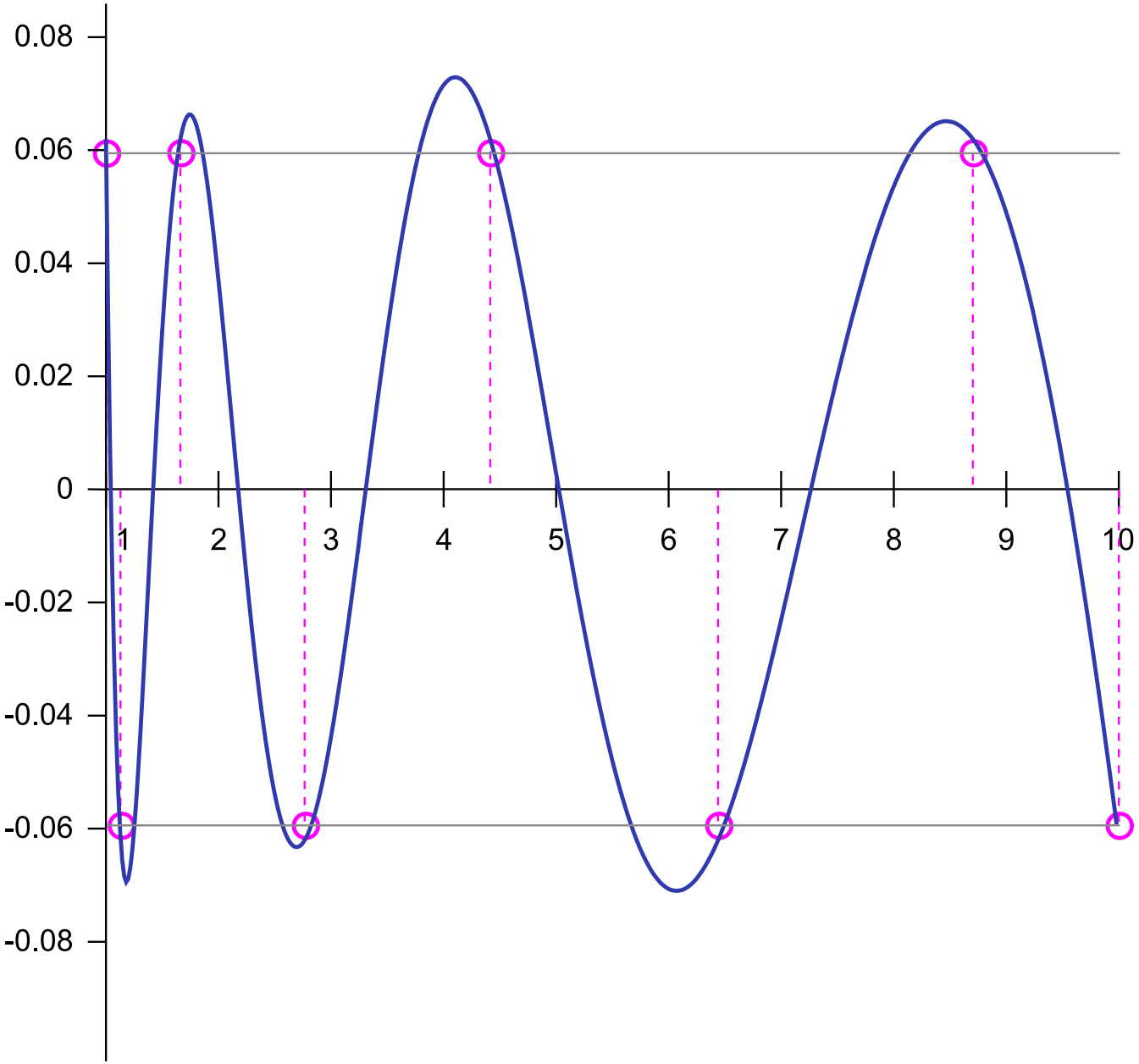


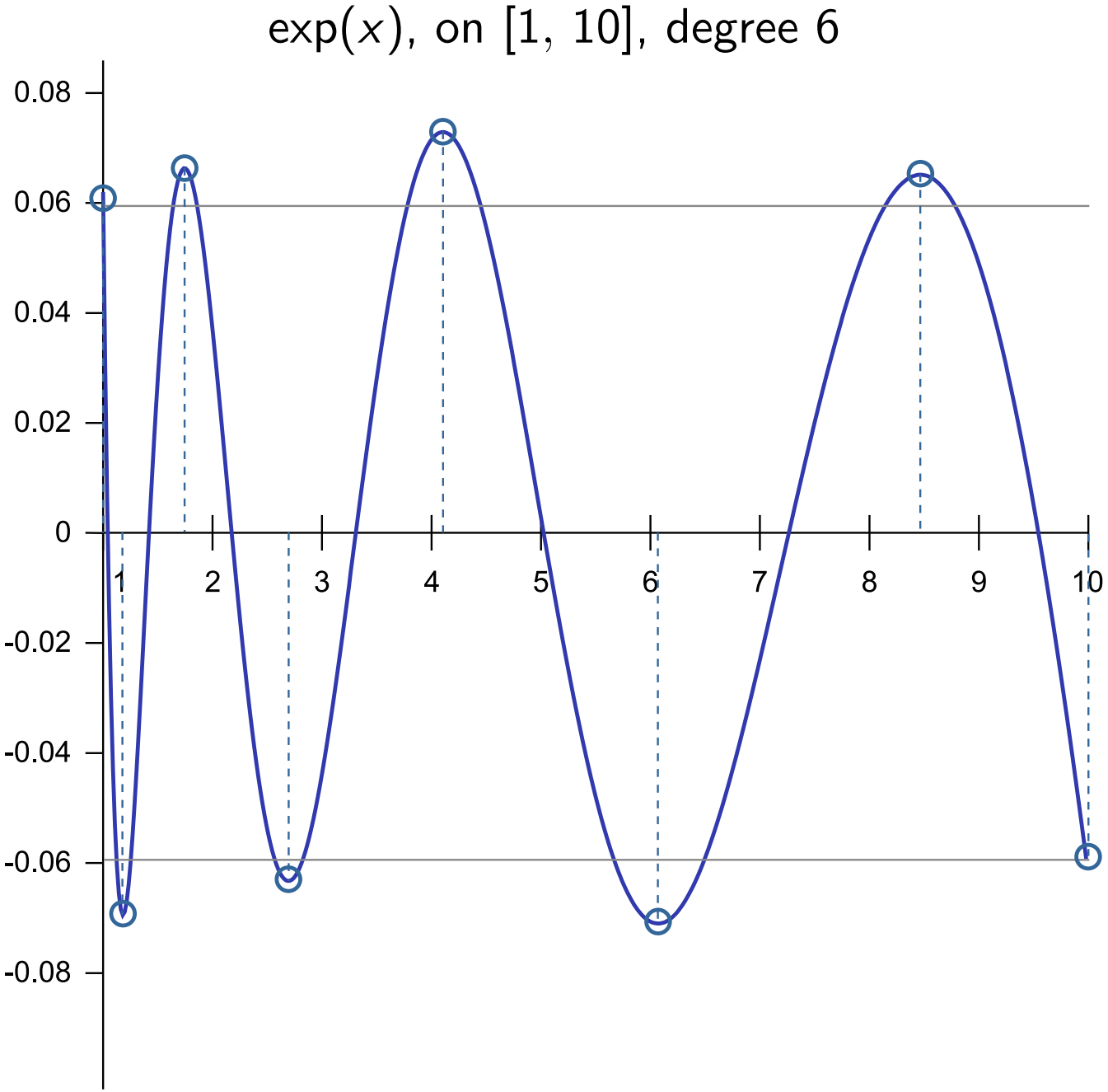

exp(x), on [1, 10], degree 6
1
0.5
0
-0.5
-1
-1.5
1
2
3
4
5
6
7
8
9
10

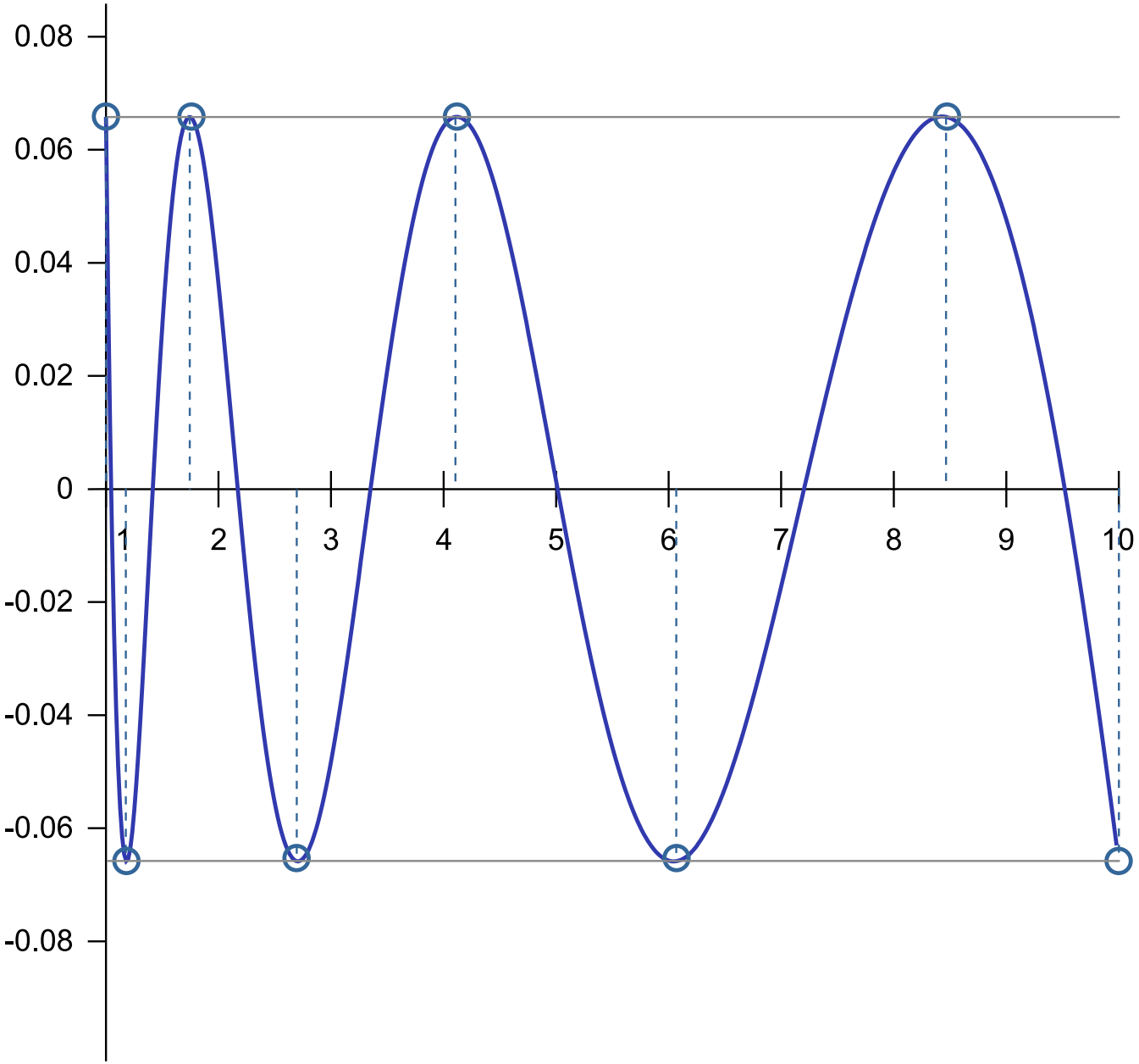

exp(x), on [1, 10], degree 6
1
0.5
0
-0.5
-1
-1.5
1
2
3
4
5
6
7
8
9
10


exp(x), on [1, 10], degree 6
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
1
2
3
4
5
6
7
8
9
10


exp(x), on [1, 10], degree 6
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
1
2
3
4
5
6
7
8
9
10


exp(x), on [1, 10], degree 6
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
1
2
3
4
5
6
7
8
9
10


exp(x), on [1, 10], degree 6
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
1
2
3
4
5
6
7
8
9
10


exp(x), on [1, 10], degree 6
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
-0.08
1
2
3
4
5
6
7
8
9
10

In some circumstances, one may wish additional constraints:

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.99999999904782645291762\ldots \\
a_1 &\simeq 1.00000006000677910949618\ldots \\
a_2 &\simeq 0.50000048748217150868457\ldots \\
a_3 &\simeq 0.16665453887128398679564\ldots \\
a_4 &\simeq 0.04165519344690637013988\ldots \\
a_5 &\simeq 0.00888564825713070913304\ldots
\end{aligned}$$

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.99999999904782645291762\ldots \\
a_1 &\simeq 1.00000006000677910949618\ldots \\
a_2 &\simeq 0.50000048748217150868457\ldots \\
a_3 &\simeq 0.16665453887128398679564\ldots \\
a_4 &\simeq 0.04165519344690637013988\ldots \\
a_5 &\simeq 0.00888564825713070913304\ldots
\end{aligned}$$

- Corresponding error: $2\,\mathrm{e}{-9}$

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 0.99999999904782645291762\ldots \\
a_1 &\simeq 1.00000006000677910949618\ldots \\
a_2 &\simeq 0.50000048748217150868457\ldots \\
a_3 &\simeq 0.16665453887128398679564\ldots \\
a_4 &\simeq 0.04165519344690637013988\ldots \\
a_5 &\simeq 0.00885648257130709133 04\ldots
\end{aligned}$$

- Corresponding error: $2\,\mathrm{e}{-9}$

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$a_0 \simeq 1$$
$$a_1 \simeq 1$$
$$a_2 \simeq 0.5$$
$$a_3 \simeq 0.16665453887128398679564\ldots$$
$$a_4 \simeq 0.041655193446906370139 88\ldots$$
$$a_5 \simeq 0.008885648257130709133 04\ldots$$

- Corresponding error: $2\,\mathrm{e}{-9}$

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 1\\
a_1 &\simeq 1\\
a_2 &\simeq 0.5\\
a_3 &\simeq 0.16665960056981588342415\ldots\\
a_4 &\simeq 0.041669874819269985517 32\ldots\\
a_5 &\simeq 0.008788946223164906868 62\ldots
\end{aligned}$$

- Corresponding error: $2\,\mathrm{e}{-9}$

In some circumstances, one may wish additional constraints:

- **Example 1:** e^x on $[-1/8, 1/4]$, degree 5:

$$p = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + a_4\,x^4 + a_5\,x^5$$

with

$$\begin{aligned}
a_0 &\simeq 1\\
a_1 &\simeq 1\\
a_2 &\simeq 0.5\\
a_3 &\simeq 0.16665960056981588342415\ldots\\
a_4 &\simeq 0.041669874819269985517 32\ldots\\
a_5 &\simeq 0.008788946223164906868 62\ldots
\end{aligned}$$

- Corresponding error: $2\,\mathrm{e}{-9}$
- Constrained polynomial error: $4.5\,\mathrm{e}{-9}$.