Generation of Pathological Cases for Rounding Errors

Sylvie Boldo David Hamelin Thibault Hilaire Pierre-Yves Piriou

Laboratoire
Méthodes
Formelles

Outline of the talk

- Introduction & motivation
- Tools
  - Gappa
  - Δ-debugging
- Contributions
  - Our algorithm for finding pathological cases
  - Demonstration
  - Benchmark
  - Call for example
- Conclusion

Introduction

- There are many analysis tools, available to obtain a bound on floating-point errors (**Gappa**, Herbie, FPTaylor, Fluctuat, Rosa).

Introduction

- There are many analysis tools, available to obtain a bound on floating-point errors (**Gappa**, Herbie, FPTaylor, Fluctuat, Rosa).
- When those tools give us acceptables upper bounds on the rounding error, all is well...

Introduction

- There are many analysis tools, available to obtain a bound on floating-point errors (**Gappa**, Herbie, FPTaylor, Fluctuat, Rosa).
- When those tools give us acceptables upper bounds on the rounding error, all is well... **But what if the bound is too big ?**

Introduction

- There are many analysis tools, available to obtain a bound on floating-point errors (**Gappa**, Herbie, FPTaylor, Fluctuat, Rosa).
- When those tools give us acceptables upper bounds on the rounding error, all is well... **But what if the bound is too big ?**

$\rightarrow$ **There exists a tighter upper bound**

- The tool isn't "smart enough" to prove it.
- A manual proof may be required.

Introduction

- There are many analysis tools, available to obtain a bound on floating-point errors (**Gappa**, Herbie, FPTaylor, Fluctuat, Rosa).
- When those tools give us acceptables upper bounds on the rounding error, all is well... **But what if the bound is too big ?**

→ **There exists a tighter upper bound**

- The tool isn't "smart enough" to prove it.
- A manual proof may be required.

→ **The given bound is tight**

- How can we convince ourselves and engineers that the upper-bound is practical ?

Introduction

Generation of a pathological case

→ There exists a tighter upper bound

- The tool isn't "smart enough" to prove it.
- A manual proof may be required.

→ The given bound is tight

- How can we convince ourselves and engineers that the upper-bound is practical ?

Motivation

- We are given a function performing only floating-point computations.
- This program has a rounding error $E(\vec{x})$ (either absolute or relative).
- An analysis tool give us a bound $\|E\|$. We suspect that this bound is tight, or close to being tight.

Motivation

- We are given a function performing only floating-point computations.
- This program has a rounding error $E(\vec{x})$ (either absolute or relative).
- An analysis tool give us a bound $\|E\|$. We suspect that this bound is tight, or close to being tight.

We want to find $\vec{c}$ so that $E(\vec{c}) \approx \|E\|$

Outline of the talk

- Introduction & motivation
- Tools
  - Gappa
  - Δ-debugging
- Contributions
  - Our algorithm for finding pathological cases
  - Demonstration
  - Benchmark
  - Call for example
- Conclusion

Génération Automatique de Preuves de Propriétés Arithmétiques

- Interval arithmetic analysis tool made by Guillaume Melquiond.
- Can be used as a solver to find a bound :

$$c \in [-0.3, -0.1] \wedge (2 \cdot a \in [3, 4] \Rightarrow b + c \in [1.4, 2]) \wedge a - c \in [1.9, 2.05] \Rightarrow b + 1 \in ?$$

Génération Automatique de Preuves de Propriétés Arithmétiques

- Interval arithmetic analysis tool made by Guillaume Melquiond.
- Can be used as a solver to find a bound :

$$c \in [-0.3, -0.1] \wedge (2 \cdot a \in [3, 4] \Rightarrow b + c \in [1.4, 2]) \wedge a - c \in [1.9, 2.05] \Rightarrow b + 1 \in \,?$$

$$b + 1 \in [2.5, 3.3]$$

Génération Automatique de Preuves de Propriétés Arithmétiques

- Interval arithmetic analysis tool made by Guillaume Melquiond.
- Can be used as a solver to find a bound :

$$c \in [-0.3, -0.1] \wedge (2 \cdot a \in [3,4] \Rightarrow b + c \in [1.4, 2]) \wedge a - c \in [1.9, 2.05] \Rightarrow b + 1 \in ?$$

$$b + 1 \in [2.5, 3.3]$$

- Contains rounding functions for fixed and floating-point arithmetic.
- Can also be used as a *verifier* :

Génération Automatique de Preuves de Propriétés Arithmétiques

- Interval arithmetic analysis tool made by Guillaume Melquiond.
- Can be used as a solver to find a bound :

$$c \in [-0.3, -0.1] \land (2 \cdot a \in [3,4] \Rightarrow b + c \in [1.4, 2]) \land a - c \in [1.9, 2.05] \Rightarrow b + 1 \in ?$$

$$b + 1 \in [2.5, 3.3]$$

- Contains rounding functions for fixed and floating-point arithmetic.
- Can also be used as a *verifier* :

$$c \in [-0.3, -0.1] \land (2 \cdot a \in [3,4] \Rightarrow b + c \in [1, 2]) \land a - c \in [1.9, 2.05] \Rightarrow b + 1 \in [0, 5]$$ ✅

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);

x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;

{
  u in [0, 1]
->
  |xR - x| in ?
}
```

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);

x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;

{
  u in [0, 1]
->
  |xR - x| in ?
}
```

⟸ **We define an alias for our rounding-mode**

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);
```

⟸ **We define an alias for our rounding-mode**

⟸ **We keep a “rounded” version of each real variable**

```
x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;

{
  u in [0, 1]
->
  |xR - x| in ?
}
```

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);
```
⟸ **We define an alias for our rounding-mode**

⟸ **We keep a “rounded” version of each real variable**

```
x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;
```
⟸ **Rounded computation use rounded variables**

```
{
  u in [0, 1]
->
  |xR - x| in ?
}
```

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);

x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;

{
  u in [0, 1]
->
  |xR - x| in ?
}
```

⟸ We define an alias for our rounding-mode

⟸ We keep a "rounded" version of each real variable

⟸ Rounded computation use rounded variables

⟸ Gappa answers: $[0, 0.00260417]$

A full-size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);
```

⟸ **We define an alias for our rounding-mode**

⟸ **We keep a "rounded" version of each real variable**

```
x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;
```

⟸ **Rounded computation use rounded variables**

```
{
  u in [0, 1]
->
  |xR - x| in [0,0.002] ✅
}
```

⟸ **We can verify a tighter-bound if we specify it**

A full size Gappa example

```
@rnd = float<ieee_32,ne>;
uR = rnd(u);

x = -((u * u) * u) / 6;
xR rnd= -((uR * uR) * uR) / 6;

{
  u in [0.125, 0.125]
->
  |xR - x| in ?
}
```

- `@rnd = float<ieee_32,ne>;` $\Longleftarrow$ **We define an alias for our rounding-mode**
- `uR = rnd(u);` $\Longleftarrow$ **We keep a “rounded” version of each real variable**
- `xR rnd= -((uR * uR) * uR) / 6;` $\Longleftarrow$ **Rounded computation use rounded variables**
- `u in [0.125, 0.125]` $\Longleftarrow$ **We can restrict a variable to a single value**
- `|xR - x| in ?` $\Longleftarrow$ **Gappa answers:** $[9.70128 \cdot 10^{-12}, 9.70128 \cdot 10^{-12}]$

This allows us to test the error of certain input values.
The goal is to find a value with a large rounding error.

Outline of the talk

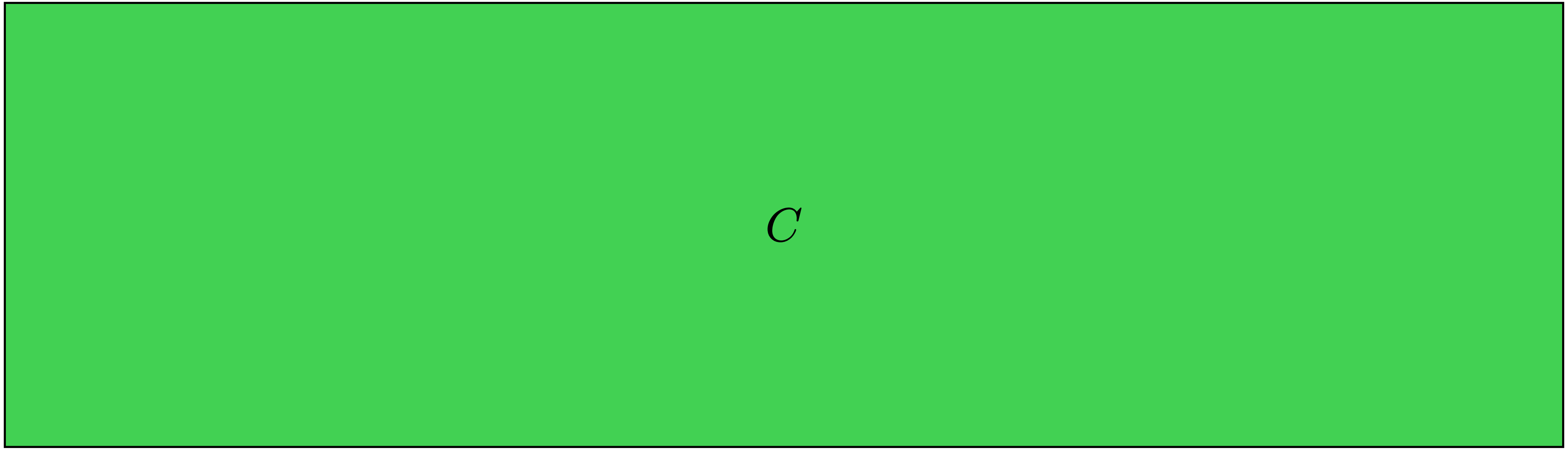
- Introduction & motivation
- Tools
  - Gappa
  - Δ-debugging
- Contributions
  - Our algorithm for finding pathological cases
  - Demonstration
  - Benchmark
  - Call for example
- Conclusion

Δ-debugging

- Δ-debugging is a family of algorithm to find a minimum set $c \subset C$ satisfying a monotonous property $P : \mathcal{P}(C) \to \mathbb{B}$.

Δ-debugging

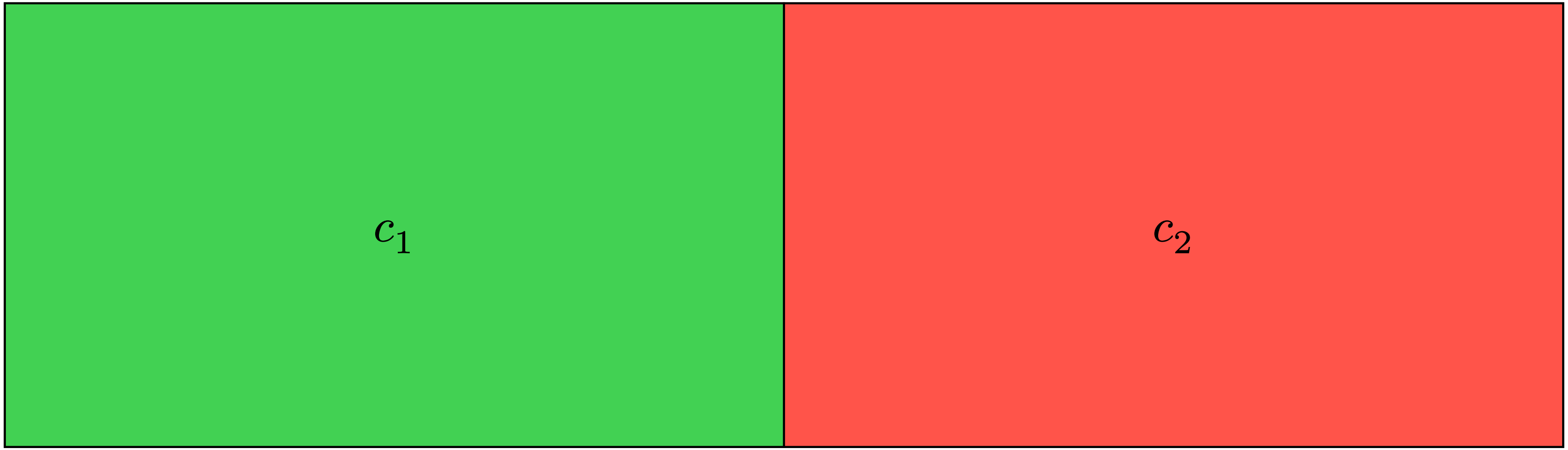
- Δ-debugging is a family of algorithm to find a minimum set $c \subset C$ satisfying a monotonous property $P : \mathcal{P}(C) \to \mathbb{B}$.



The property is true with the *full configuration* C.

Δ-debugging

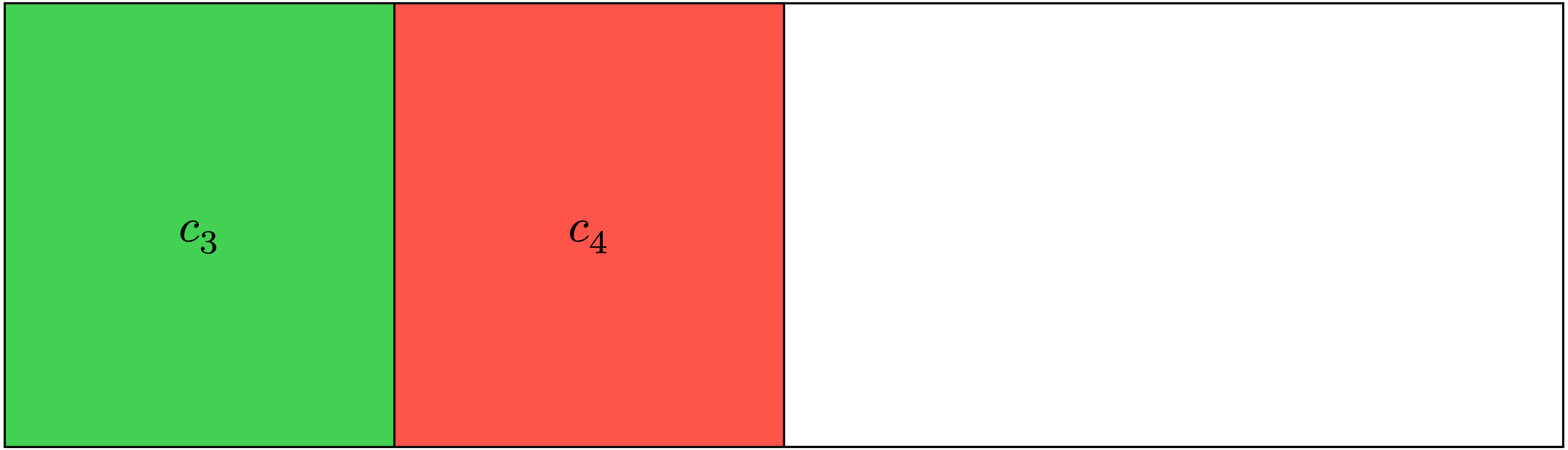
- Δ-debugging is a family of algorithm to find a minimum set $c \subset C$ satisfying a monotonous property $P : \mathcal{P}(C) \to \mathbb{B}$.



We create partition $C = c_1 \uplus c_2$.

Δ-debugging

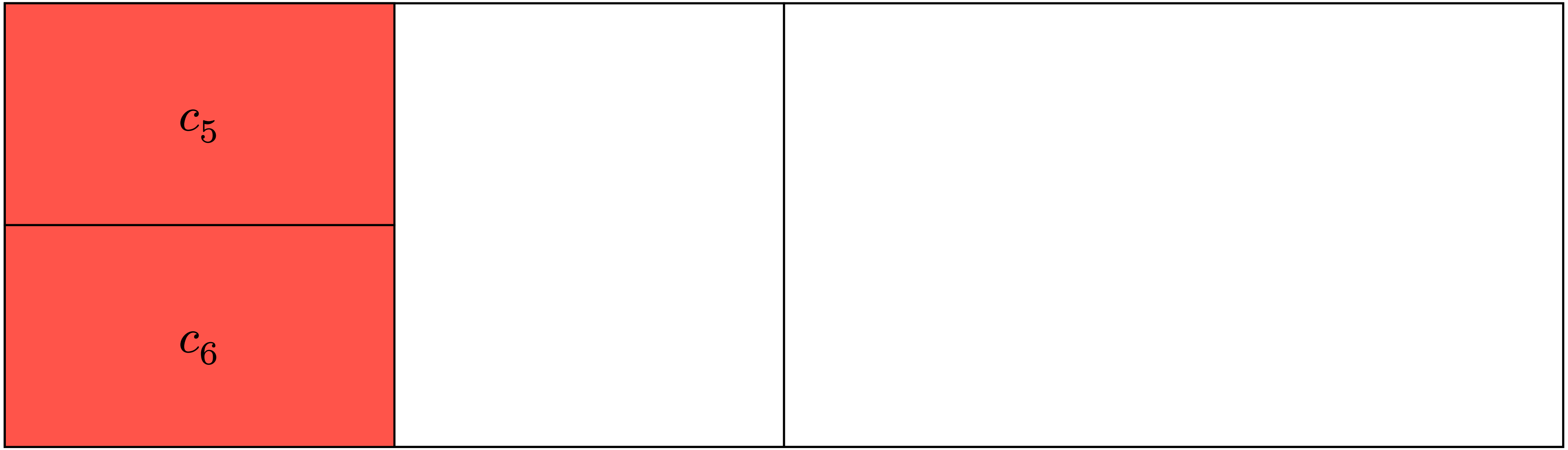
- Δ-debugging is a family of algorithm to find a minimum set $c \subset C$ satisfying a monotonous property $P : \mathcal{P}(C) \rightarrow \mathbb{B}$.



c_3 satisfies the property, so we continue the search there.

Δ-debugging

- Δ-debugging is a family of algorithm to find a minimum set $c \subset C$ satisfying a monotonous property $P : \mathcal{P}(C) \to \mathbb{B}$.



Neither c_5 or c_6 satisfy P. P needs elements from both c_5 and c_6.

Δ-debugging applications

- Initially developped to **find out a bug** in the GNU Debugger :
  - $C =$ the set of all line changes since the last release,
  - $P(c)$ is true if there the patches in c cause a bug.

  → We get a small set of line required to cause the bug.

- Also used to **decrease the format size of variables of an FP program:**
  - $C = \text{set of all variables} \times \{\text{float}, \text{double}, \text{long double}\}$.
  - $P(c)$ is true if the program is accurate enough.

→ We get a compliant small format for each variable.

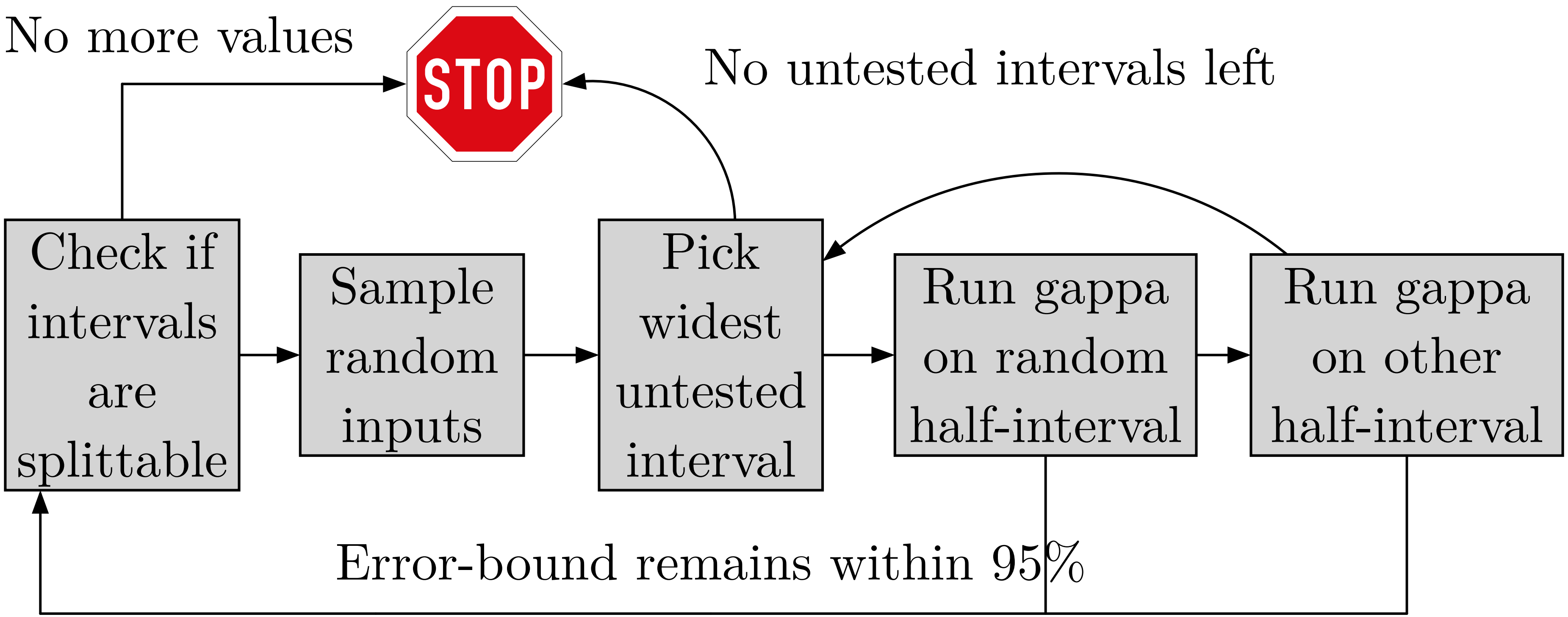
Outline of the talk

- Introduction & motivation
- Tools
  - Gappa
  - Δ-debugging
- Contributions
  - Our algorithm for finding pathological cases
  - Demonstration
  - Benchmark
  - Call for example
- Conclusion

Δ-debugging & Gappa

- Our contribution : using Δ-debugging along with Gappa to generate pathological cases.
- The idea is to reduce the size of our input intervals while still keeping the error-bound high.
- Every time we halve the size of one of the intervals, we randomly test some values, and if the error is larger, we record it.

The algorithm in a nutshell



Algorithm & Demonstration

We will demonstrate the algorithm on an example taken from the paper:

The Classical Relative Error Bounds for Computing $\sqrt{a^2+b^2}$ and $c/\sqrt{a^2+b^2}$ in Binary Floating-Point Arithmetic are Asymptotically Optimal

Claude-Pierre Jeannerod[*], Jean-Michel Muller[†], and Antoine Plet[‡]

[*]Inria, [†]CNRS, [‡]ENS de Lyon
Université de Lyon – Laboratoire LIP (CNRS, Inria, ENS de Lyon, UCBL), Lyon, France

Algorithm & Demonstration

We will demonstrate the algorithm on an example taken from the paper:

T … r

Con … nary

Flo … ally

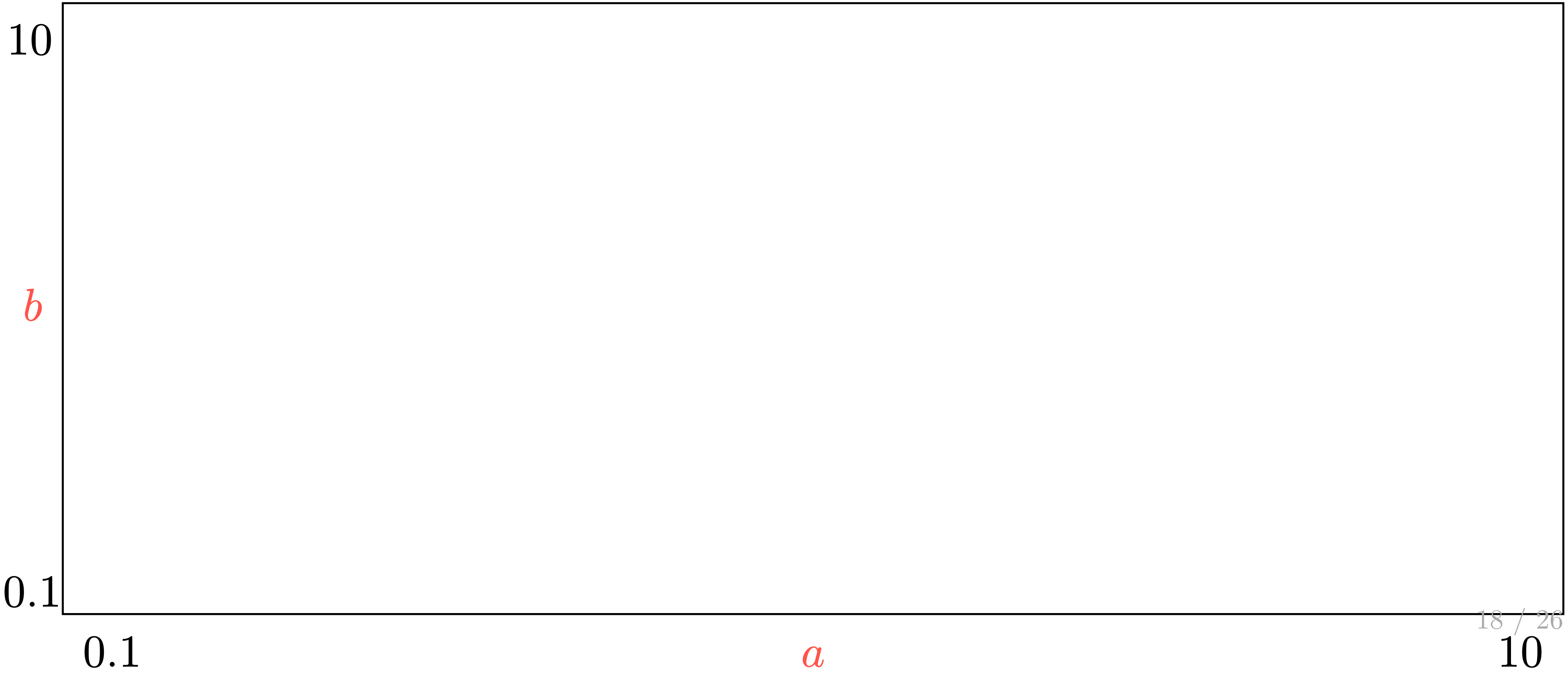
```
@rnd = float<ieee_64, ne>;
aR = rnd(a); bR = rnd(b);
o = sqrt(a*a + b*b); oR rnd= sqrt(aR*aR + bR*bR);
{
  a in [0.01, 10] /\ b in [0.01, 10]
->
  oR -/ o in ?
}
```

Claude-Pierre Jeannerod*, Jean-Michel Muller†, and Antoine Plet‡

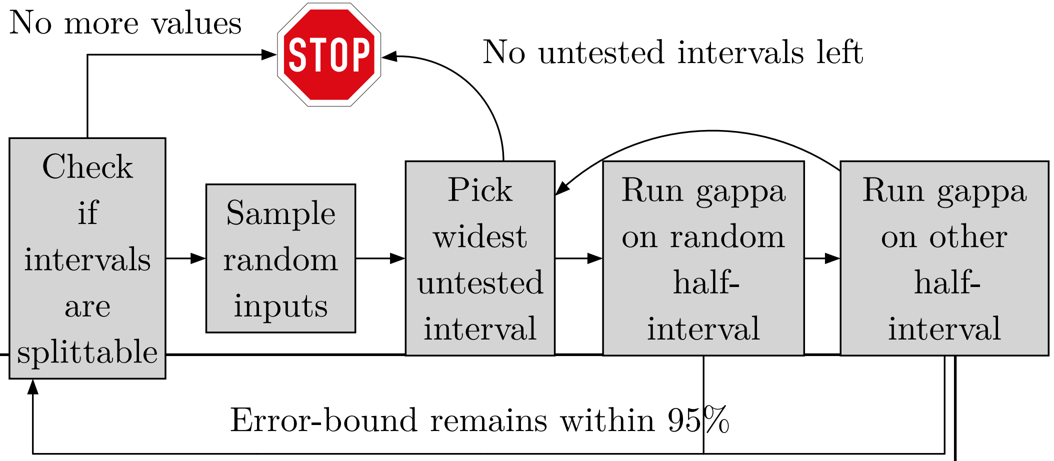
*Inria, †CNRS, ‡ENS de Lyon
Université de Lyon – Laboratoire LIP (CNRS, Inria, ENS de Lyon, UCBL), Lyon, France

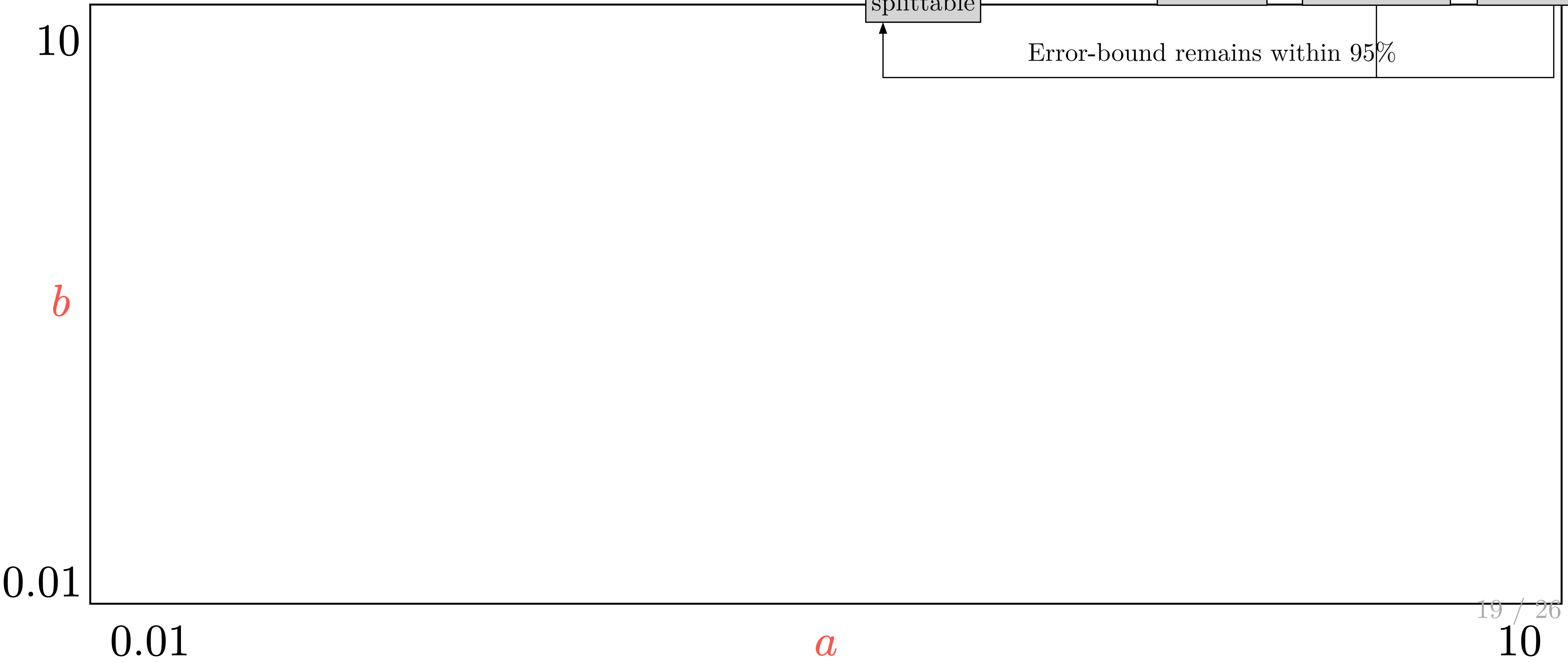
Visualizing the algorithm

We will divide the search-space of possible solutions as square :

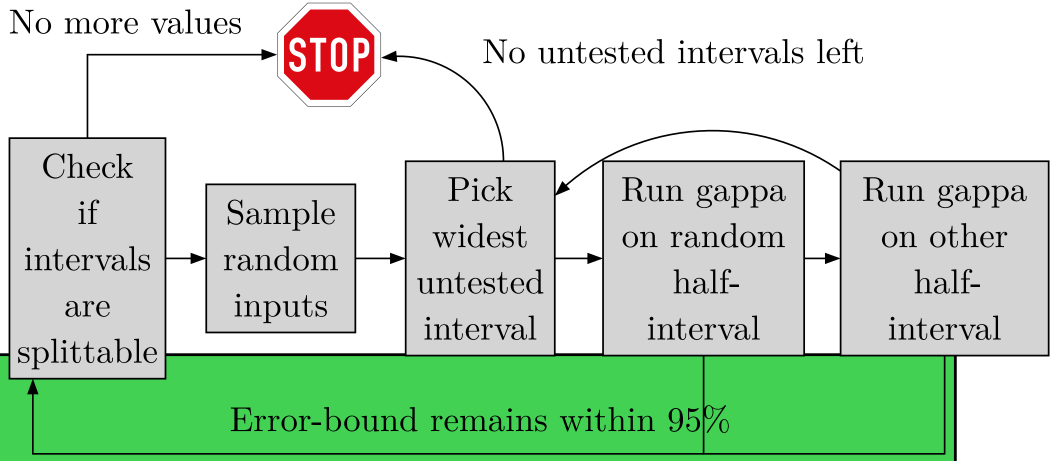


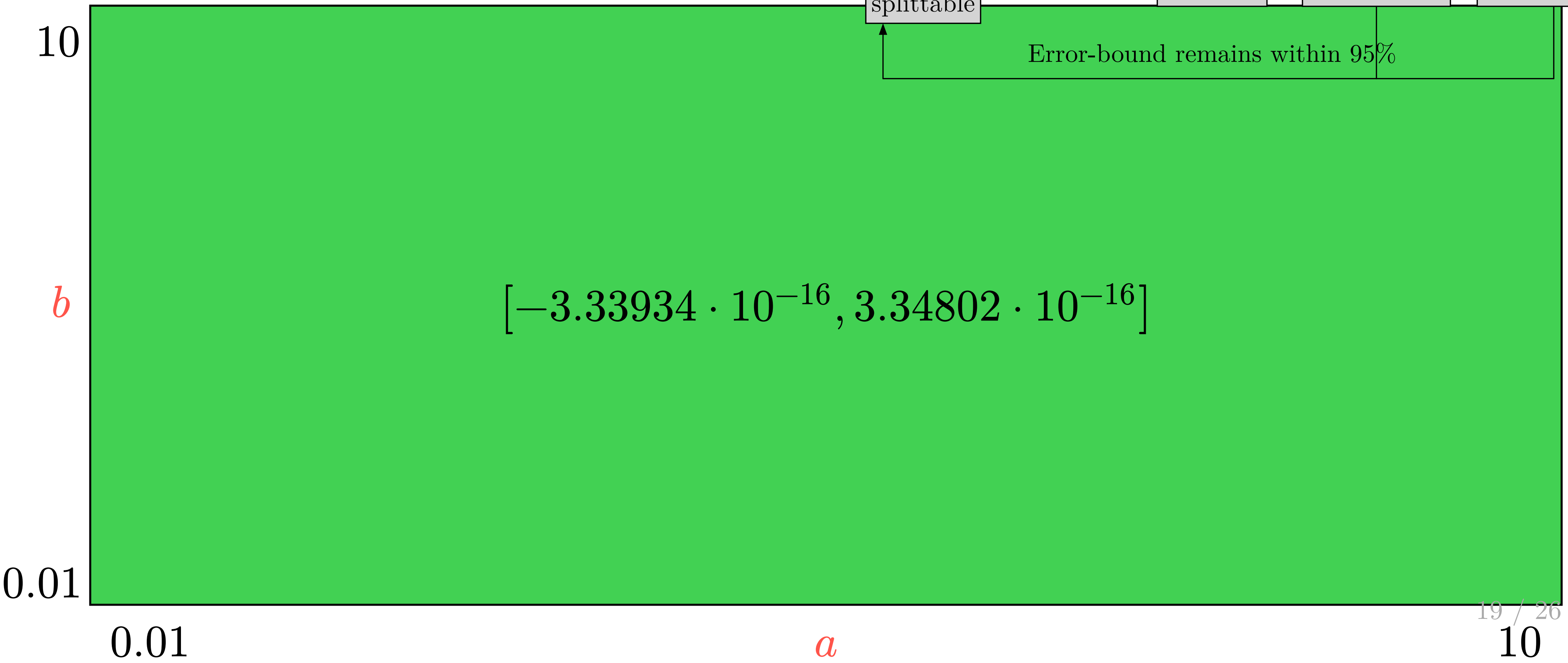
Visualizing the algorithm



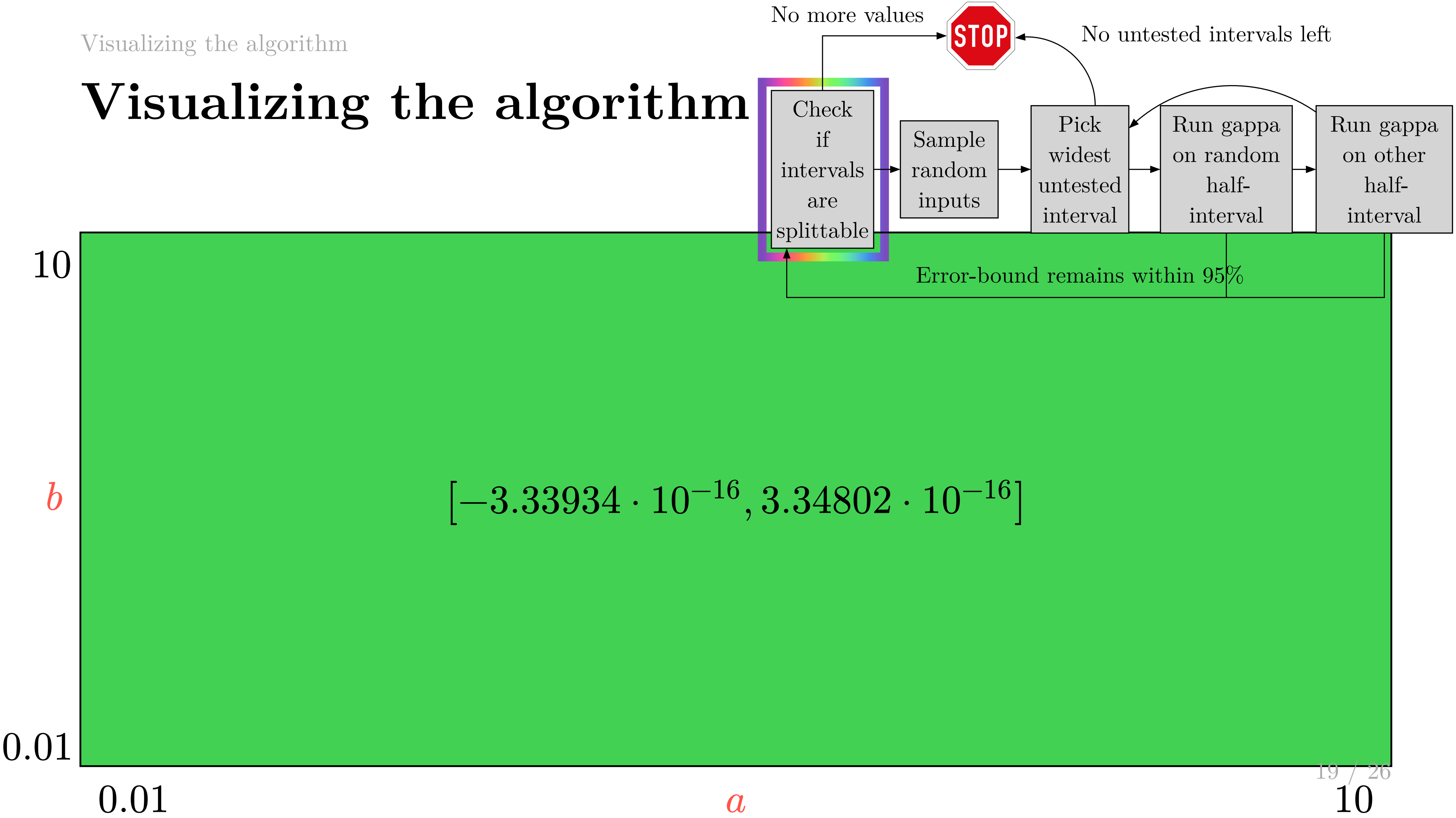


Visualizing the algorithm

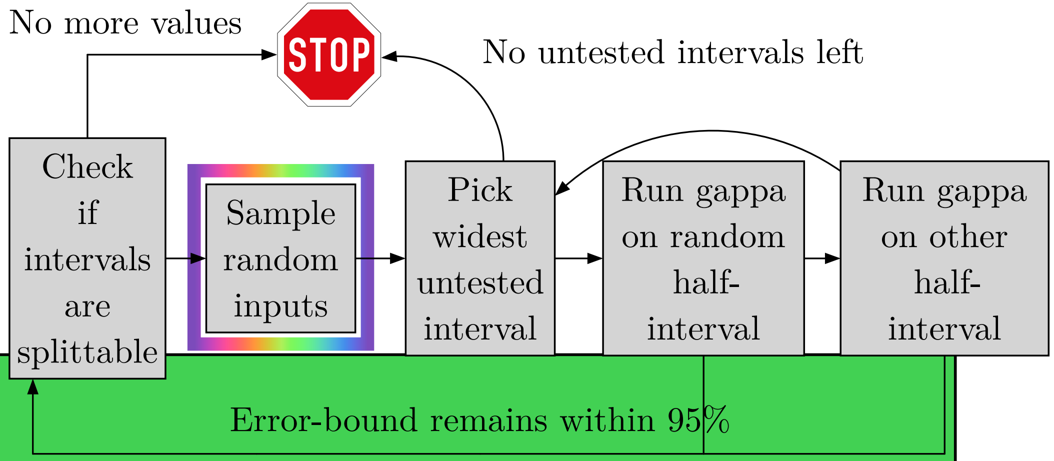


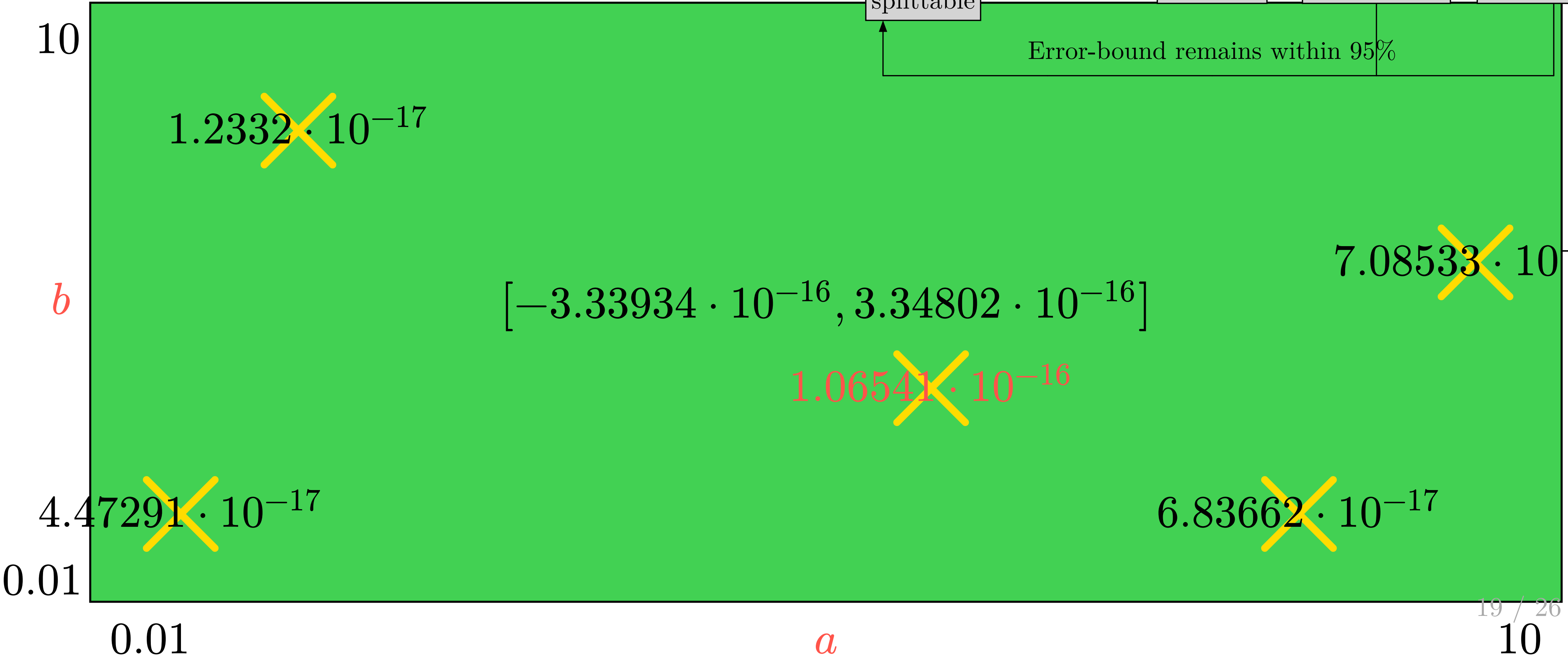


Visualizing the algorithm

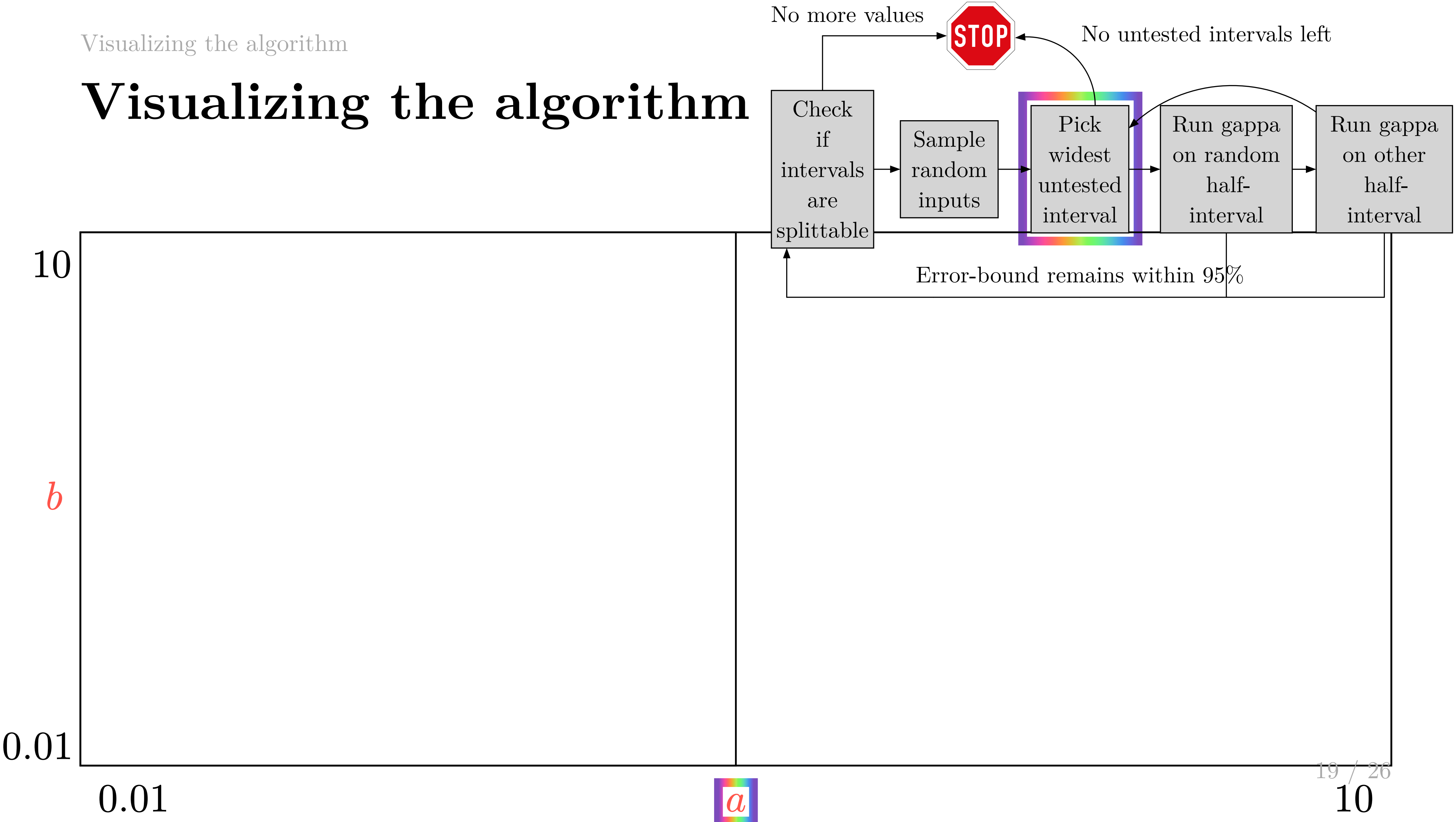


Visualizing the algorithm

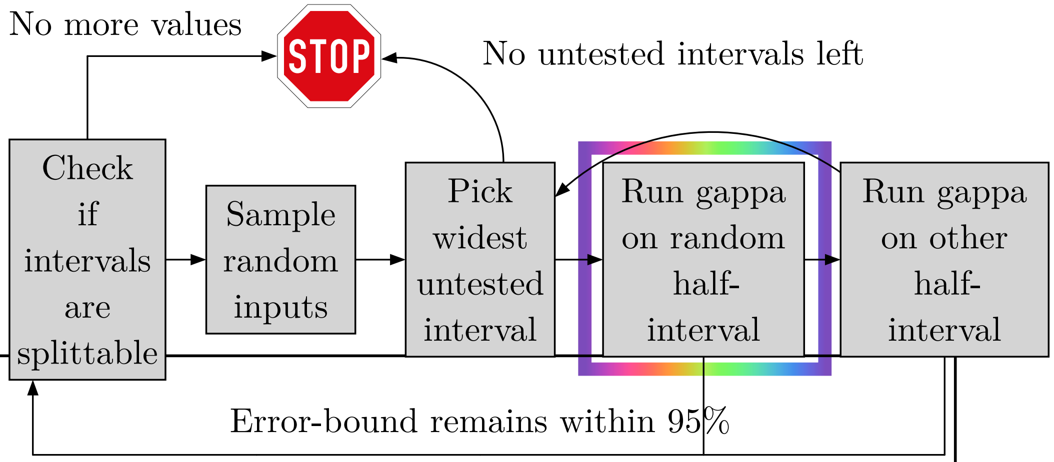


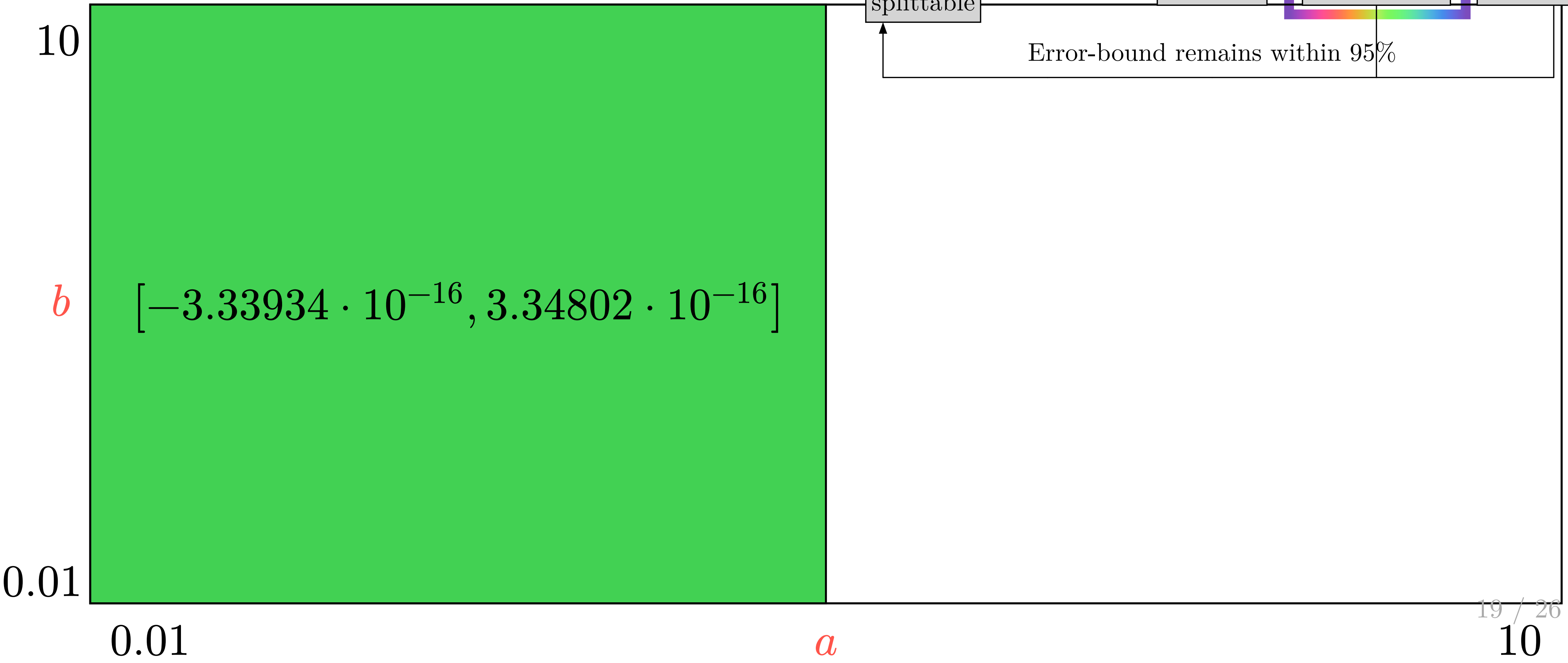


Visualizing the algorithm

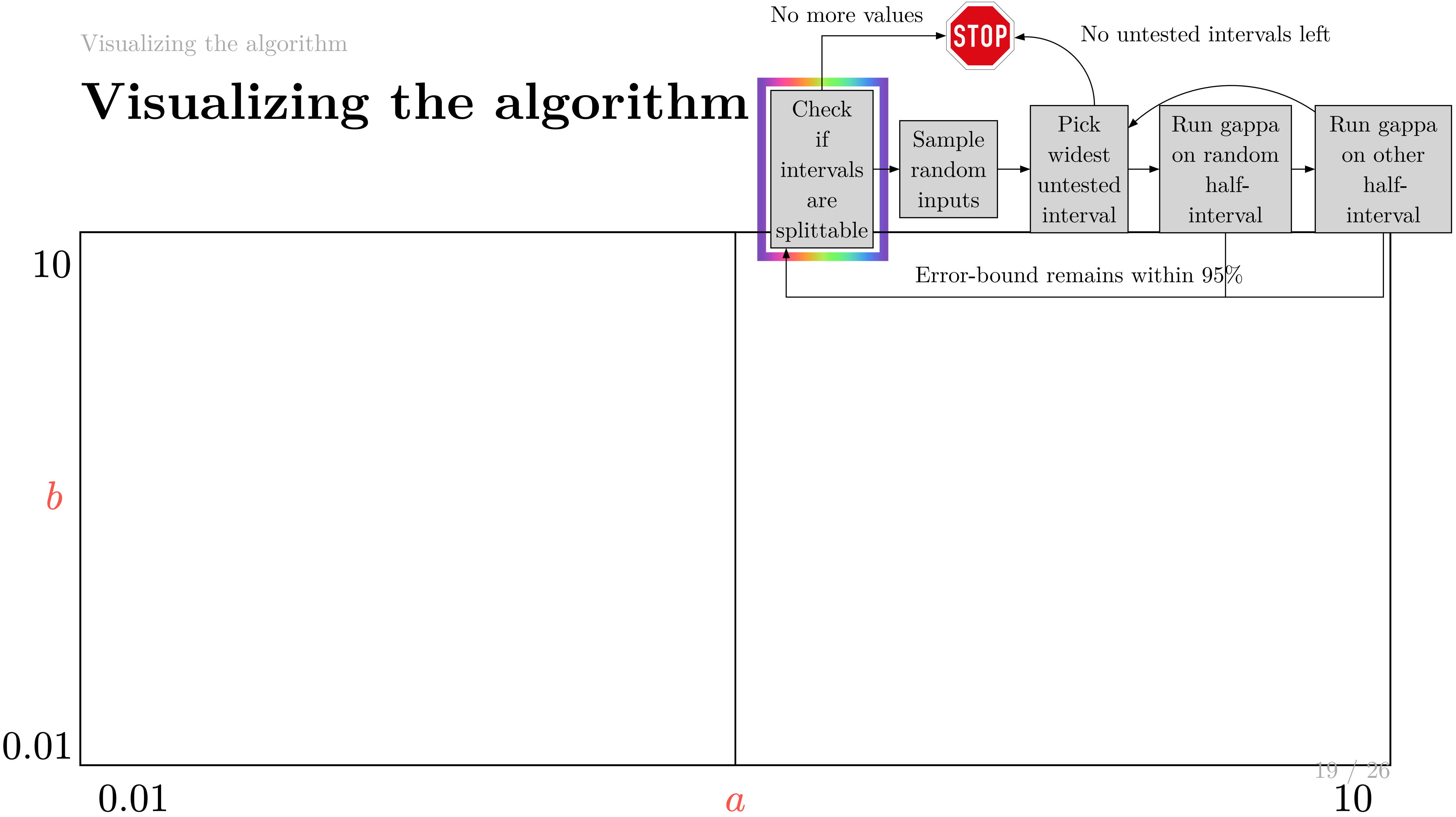


Visualizing the algorithm

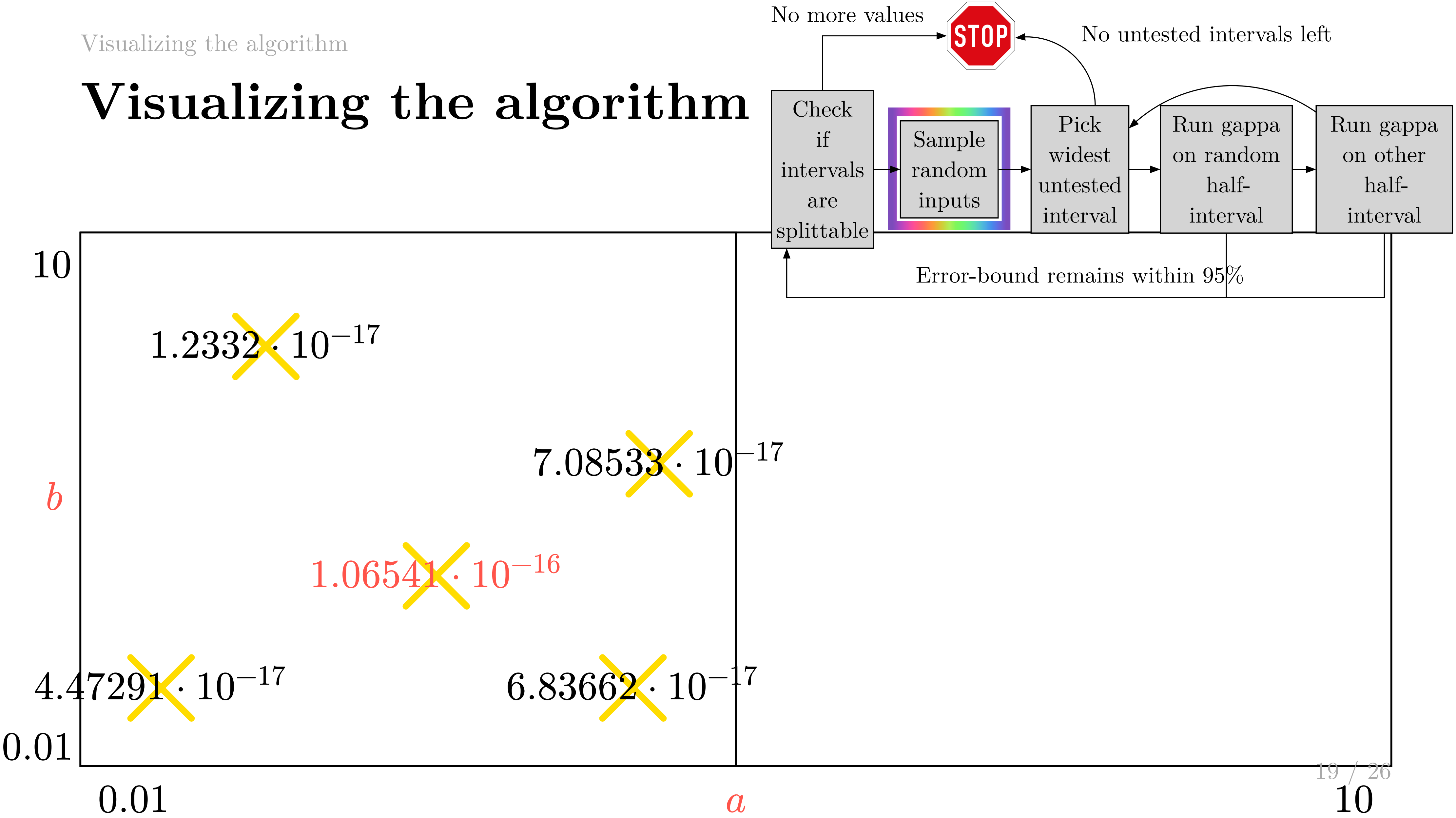




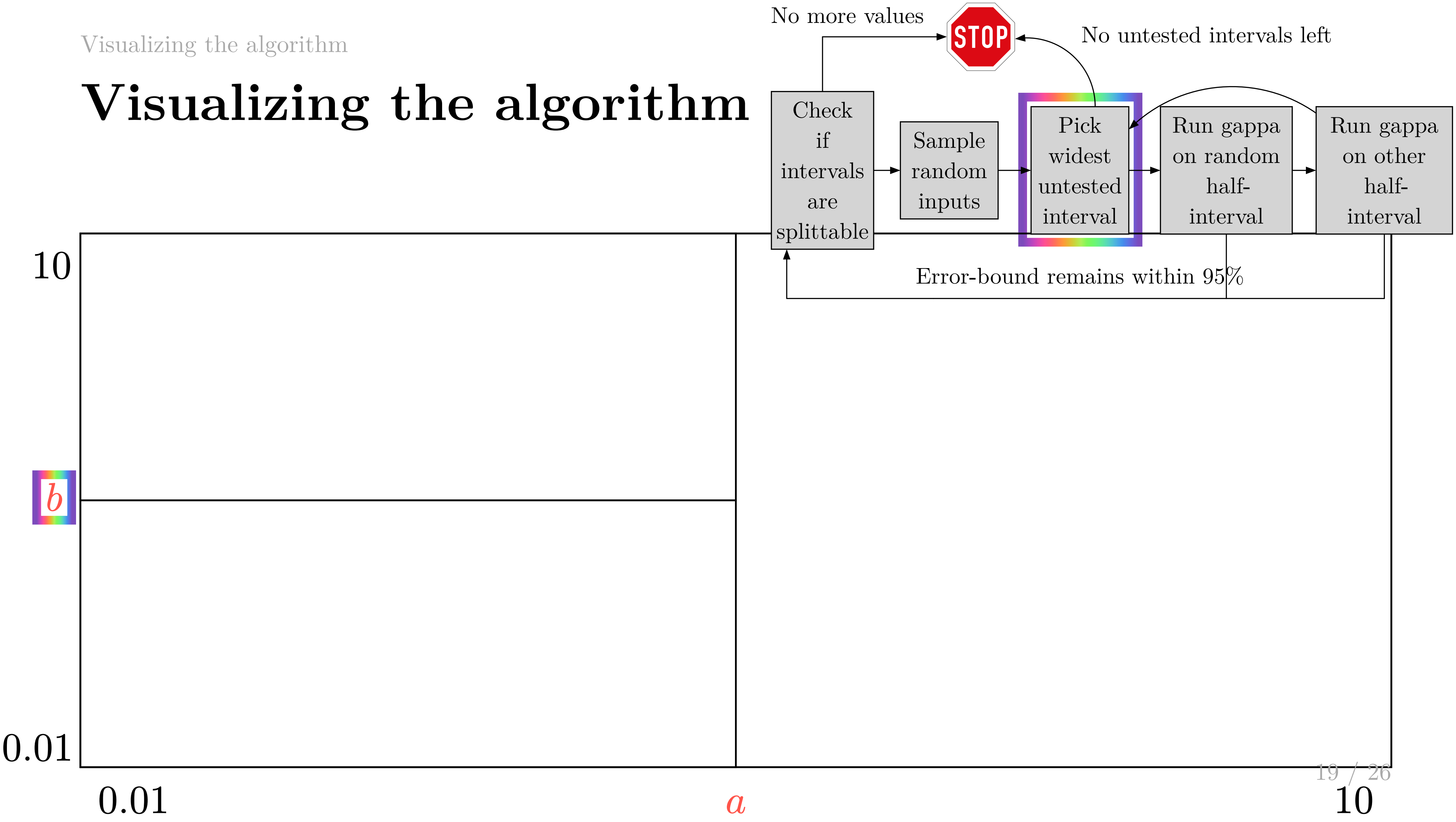
Visualizing the algorithm



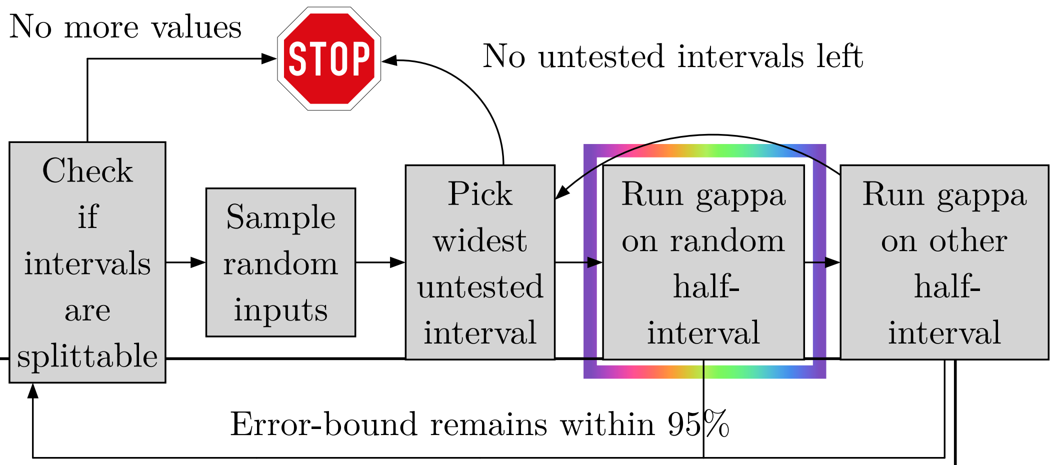
Visualizing the algorithm

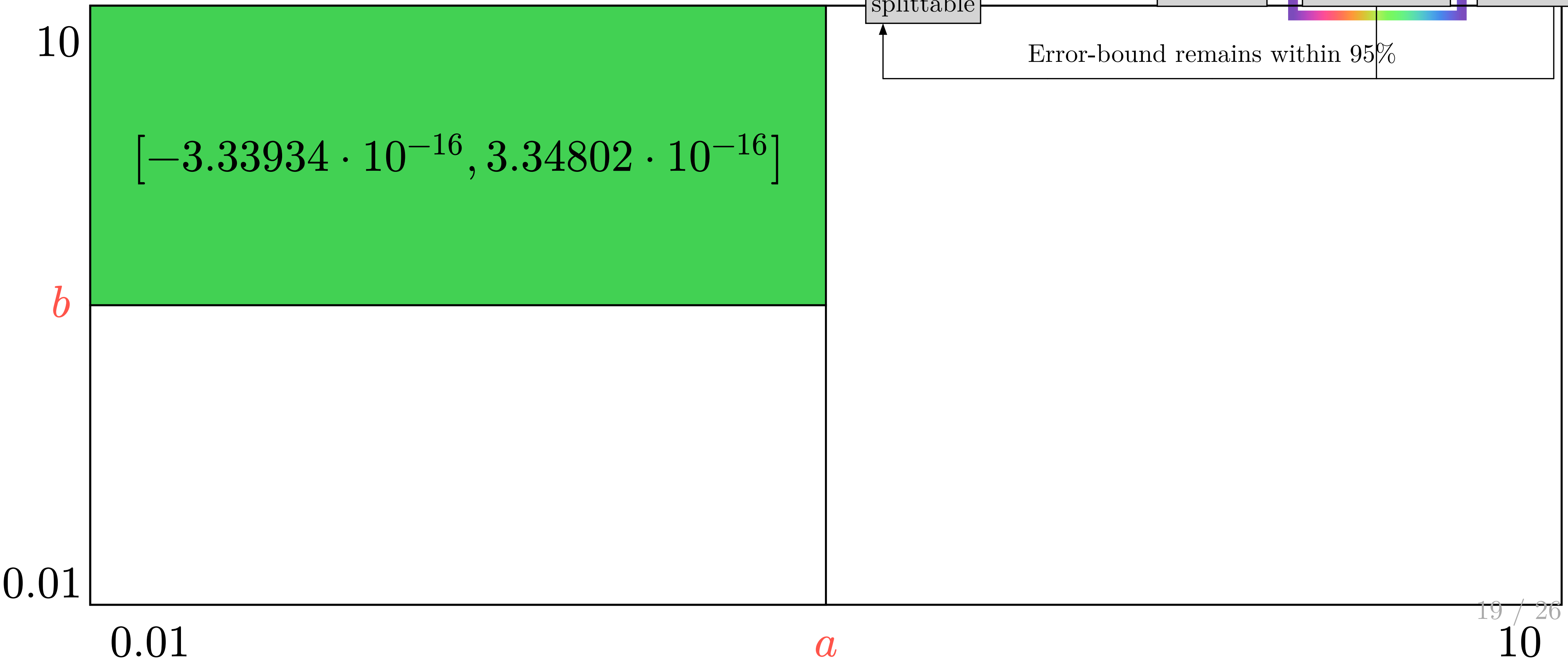


Visualizing the algorithm



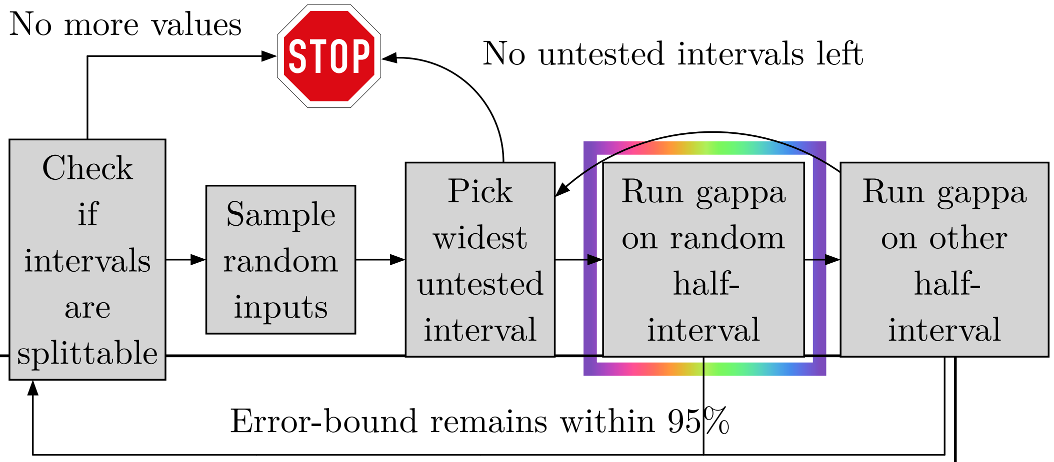
Visualizing the algorithm

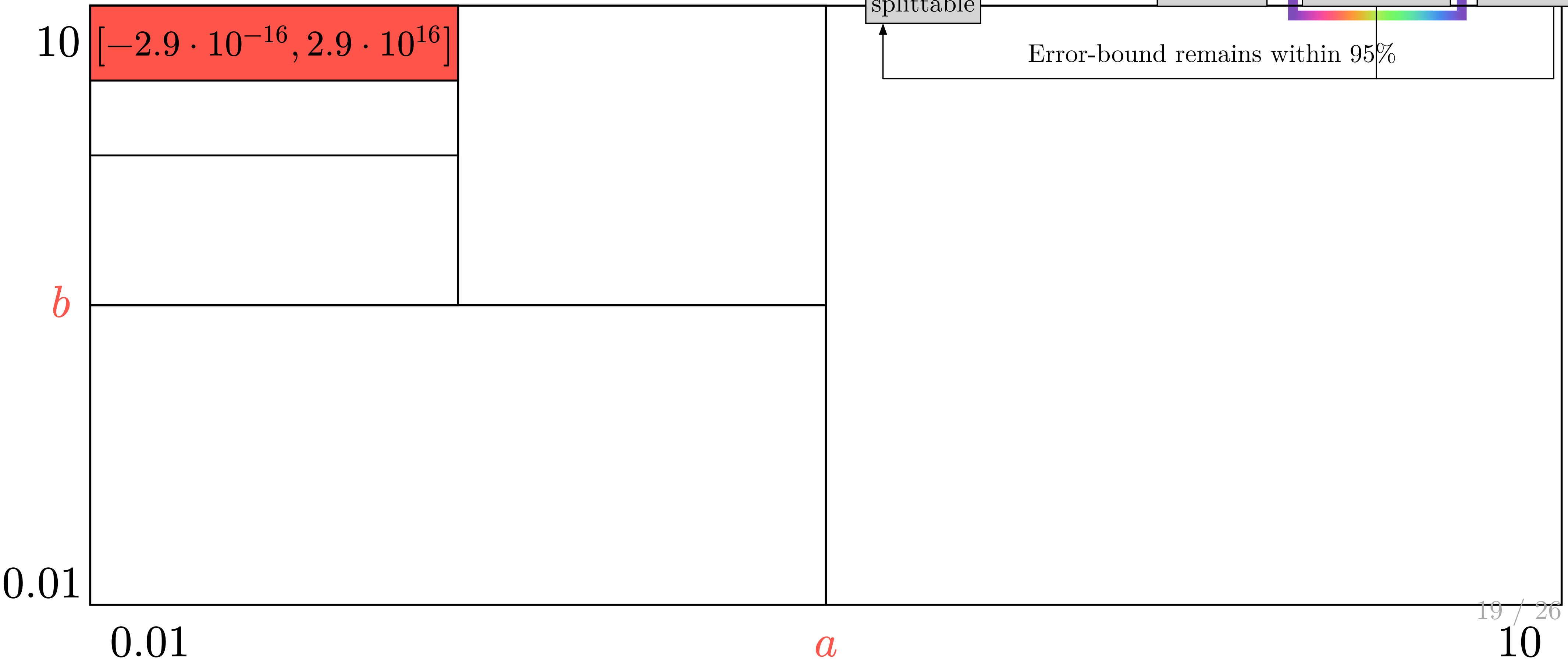




Visualizing the algorithm

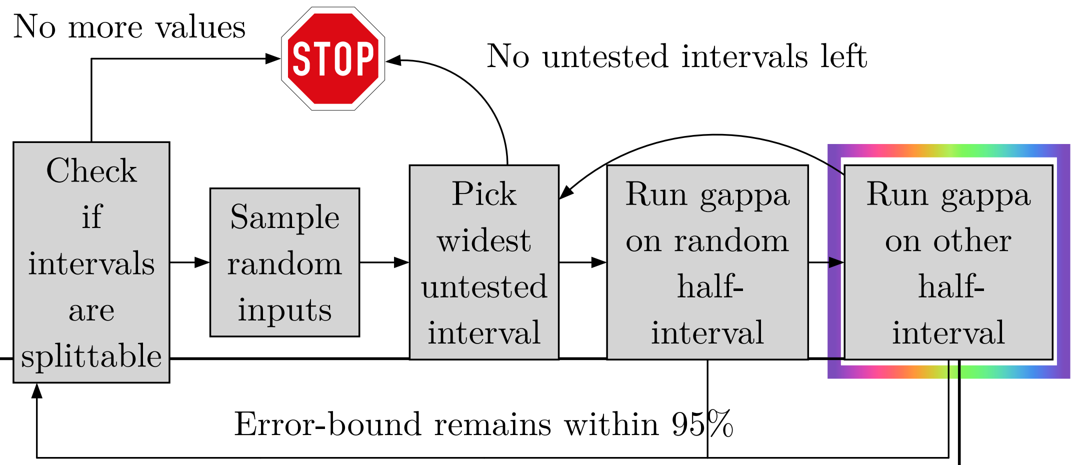
$a \in [0.01, 2.5], b \in [7.5000025, 10.0]$

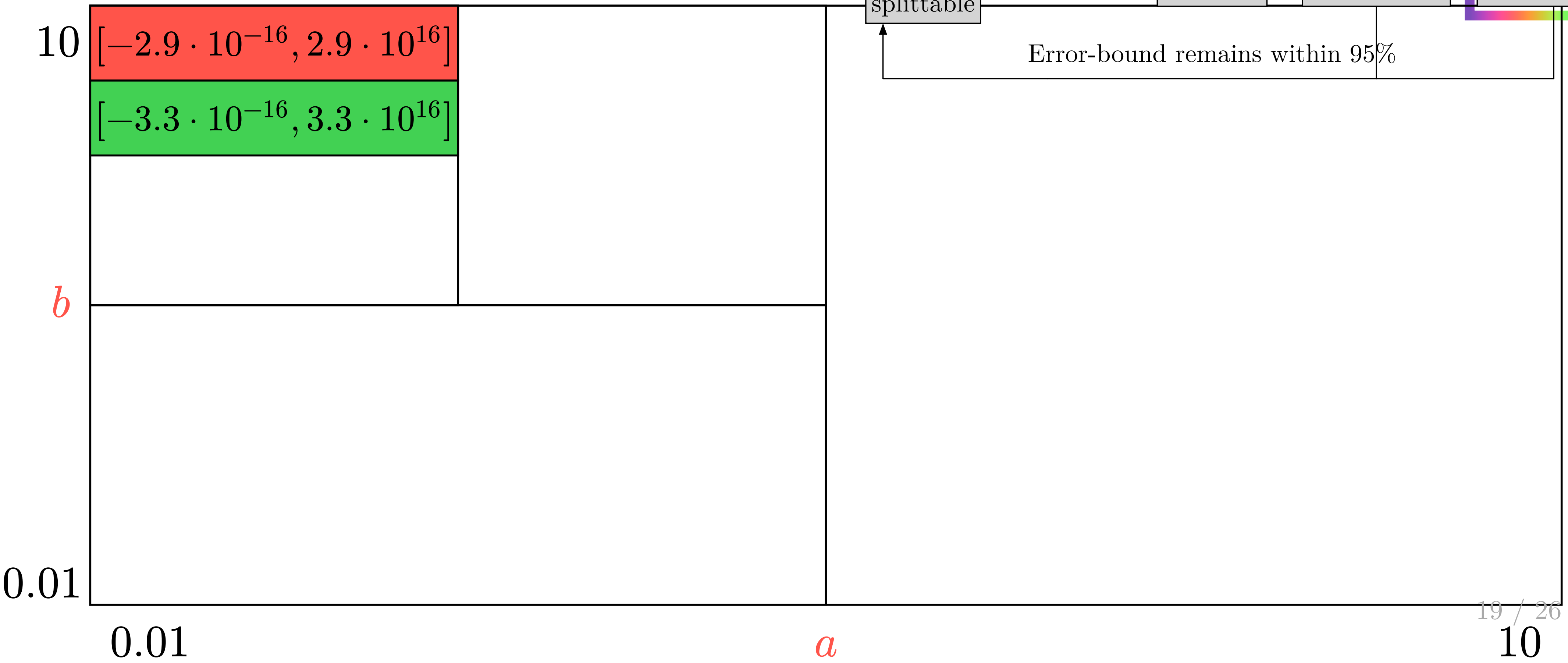




Visualizing the algorithm

$a \in [0.01, 2.5], b \in [7.5000025, 10.0]$





Results :

- The best pathological case had an error of $1.92033 \cdot 10^{-16}$.

Results :

- The best pathological case had an error of $1.92033 \cdot 10^{-16}$.
- The worst, pen-and-paper proven case has an error of $2.22044 \cdot 10^{-16}$.

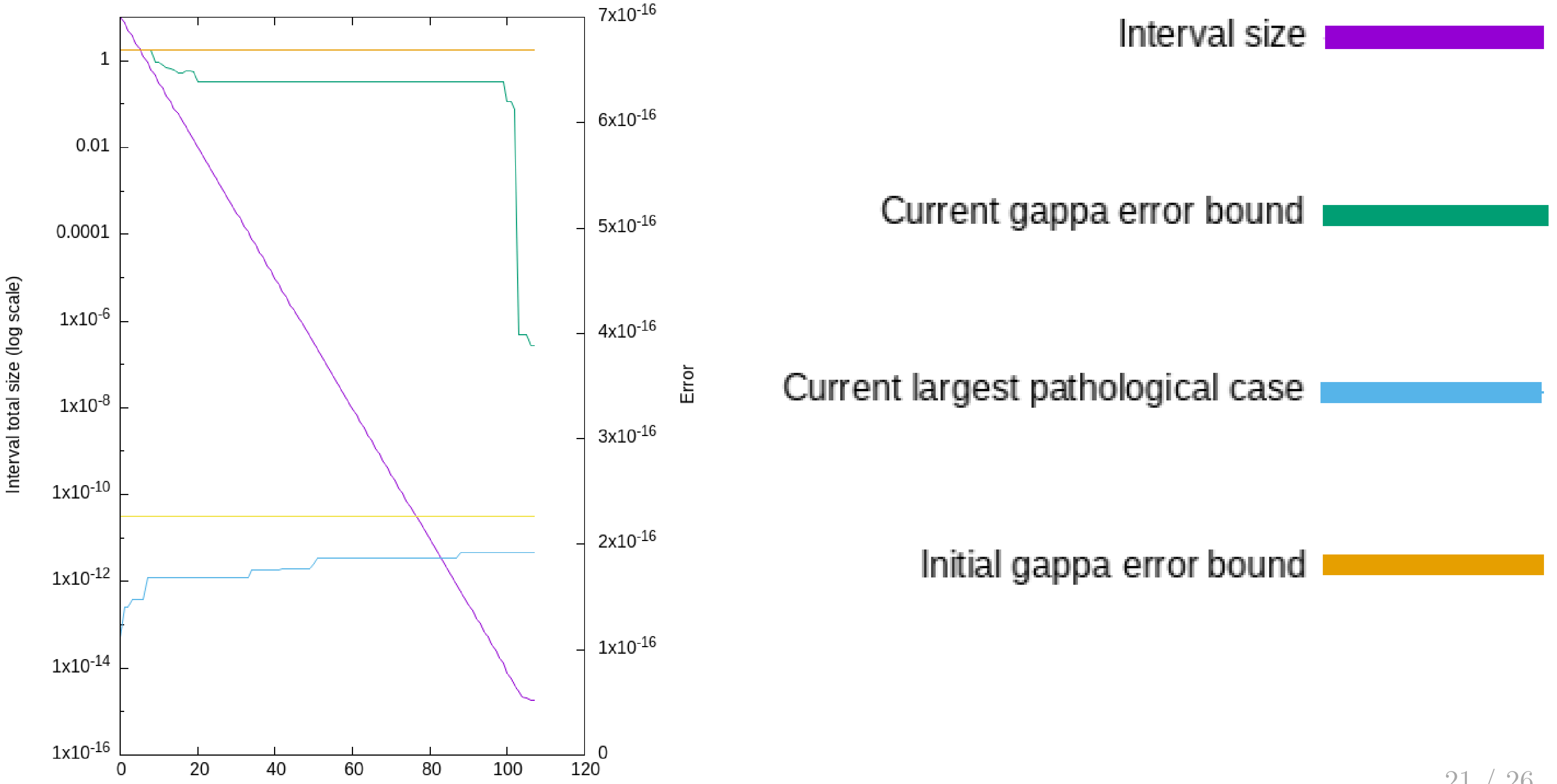
Results :

- The best pathological case had an error of $1.92033 \cdot 10^{-16}$.
- The worst, pen-and-paper proven case has an error of $2.22044 \cdot 10^{-16}$.
- Our error is 85% close !

Results :

- The best pathological case had an error of $1.92033 \cdot 10^{-16}$.
- The worst, pen-and-paper proven case has an error of $2.22044 \cdot 10^{-16}$.
- Our error is 85% close !
- Variance of 3% (tested 100 times).

Plotting the search



Benchmarks

- 48 Benchmarks from *FPBench* (all those compliant with Gappa).
- 3 functions from *Round-off error and exceptional behavior analysis of explicit Runge-Kutta methods* (Boldo, Faissole & Chapoutot).
- 2 functions from *Bounding the Round-Off Error of the Upwind Scheme for Advection* (Ben Salem-Knapp, Boldo & Weens).
- 1 function from *The Classical Relative Error Bounds for Computing $\sqrt{(a^2 + b^2)}$ and $c/\sqrt{(a^2 + b^2)}$ in Binary Floating-Point Arithmetic are Asymptotically Optimal* (Jeannerod, Muller, Plet).
- 1 example from Gappa's manual (we will add the others).

Benchmark results : Timings

- 26 / 55 are below 1 minute
- Several hours ($\max \approx 8$h) when there are many inputs

$\rightarrow$ Very high variance.

- Large overhead of invoking Gappa multiple times on slightly different intervals.

Benchmark results : Quality

- 10 / 55 are within a factor **2** of Gappa's bound
- 44 / 55 are within a factor **10** of Gappa's bound
- 46 / 55 are within a factor **15** of Gappa's bound

This validates the order of magnitude of the bound in more than **80%** of the benchmark.

Call for example !

We are interested in loopless floating-point programs using only additions, subtractions, divisions & square-roots.

Conclusion and perspectives

- This is **work-in-progress**. We need to tune the algorithm, add examples, analyze the results, compare with the state-of-the art, and so on.
- A perspective is a deep **integration in Gappa**, allowing us to reuse the theorems.
- Testing different **heuristics**, in particular those used in state-of-the-art paper.
- Adding more examples related to **fixed-point** arithmetic.