Journée des doctorants

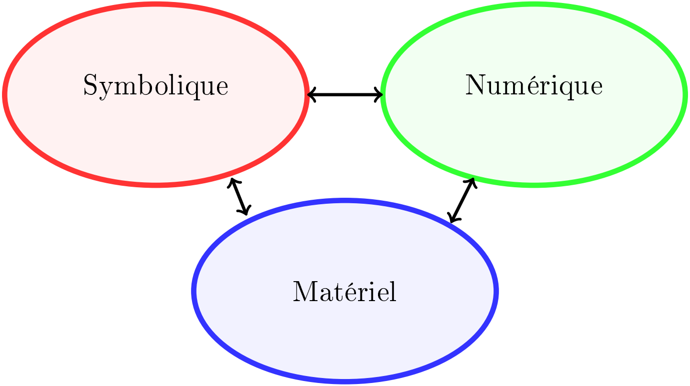
Silviu Filip Sébastien Maulat Stephen Melczer
Vincent Neiger Marie Paindavoine
Antoine Plet Valentina Popescu

Aric Team, LIP, ENS de Lyon, France

June 2015

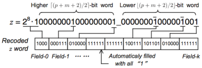
AriC: Arithmetic and Computing

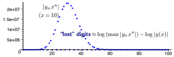
Améliorer le calcul, en termes de performance, d'efficacité et de fiabilité.



AriC: Arithmetic and Computing

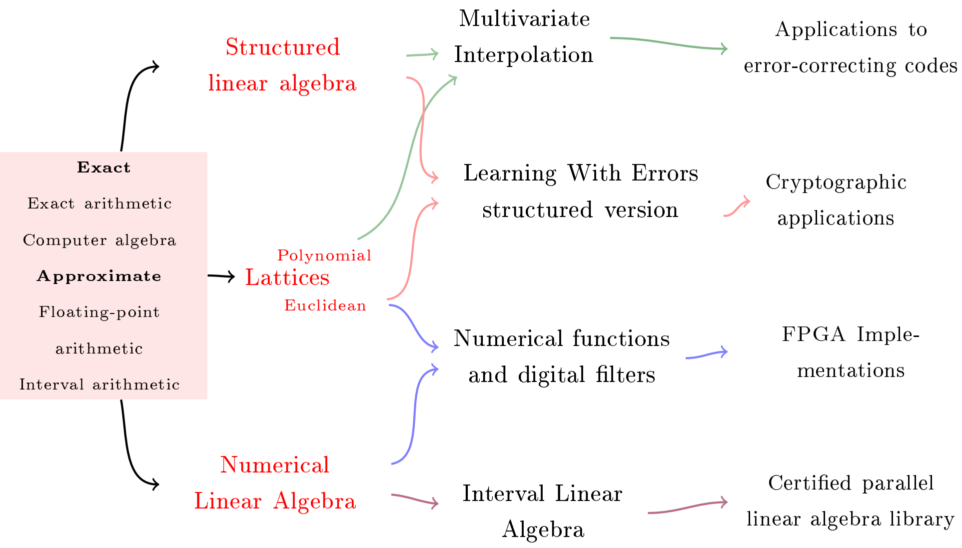
- algorithmes arithmétiques & implantation:
  - virgule flottante;
  - multi-précision;
  - bornes d'erreur fines
- méthodes d'approximation:
  - uniforme/ponctuelle;
  - polynomiale/rationnelle
- calcul certifié et calcul formel:
  - algèbre linéaire à coefficients polynomiaux;
  - récurrences linéaires à coefficients polynomiaux;



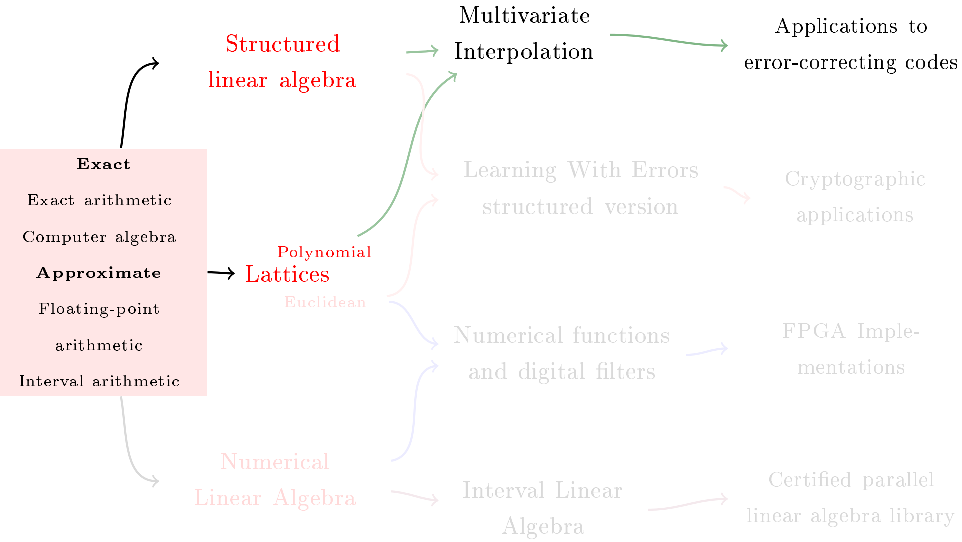


$$\mathrm{Ai}(x) = A - Bx + \frac{A}{6}x^3 - \frac{B}{12}x^4 + \frac{A}{180}x^6 - \frac{B}{504}x^7 + \frac{A}{12960}x^9 - \cdots$$

AriC


Structured linear algebra
Multivariate Interpolation
Applications to error-correcting codes
Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point arithmetic
Interval arithmetic
Polynomial
Lattices
Euclidean
Learning With Errors structured version
Cryptographic applications
Numerical functions and digital filters
FPGA Implementations
Numerical Linear Algebra
Interval Linear Algebra
Certified parallel linear algebra library

AriC


Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point
arithmetic
Interval arithmetic
Structured
linear algebra
Polynomial
Lattices
Euclidean
Numerical
Linear Algebra
Multivariate
Interpolation
Learning With Errors
structured version
Numerical functions
and digital filters
Interval Linear
Algebra
Applications to
error-correcting codes
Cryptographic
applications
FPGA Imple-
mentations
Certified parallel
linear algebra library

AriC — Multivariate interpolation

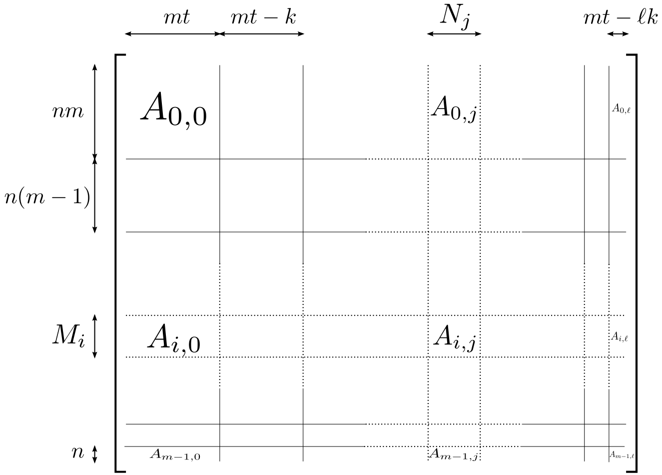
Problem: MULTIVARIATE POLYNOMIAL INTERPOLATION

Applications to:

- Error-correcting codes
  Enable reliable delivery of data over unreliable communication channels
  (add redundancy to the message)
- Crypto: private information retrieval
  look up information in an online database without letting the database servers know (or learn) the query terms or responses.

AriC — Tool: structured matrices

Find a solution of a (scalar) structured linear system,

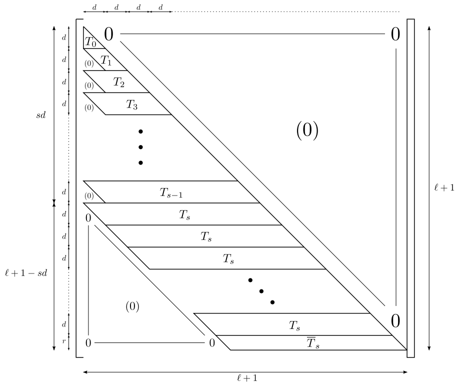


where $A_{i,j}$ is a Toeplitz / Hankel / Vandermonde / ... matrix

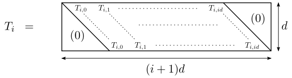
AriC — Tool: polynomial matrices

Find a short vector in the row space of a polynomial matrix
= lattice basis reduction (LLL), over the polynomials

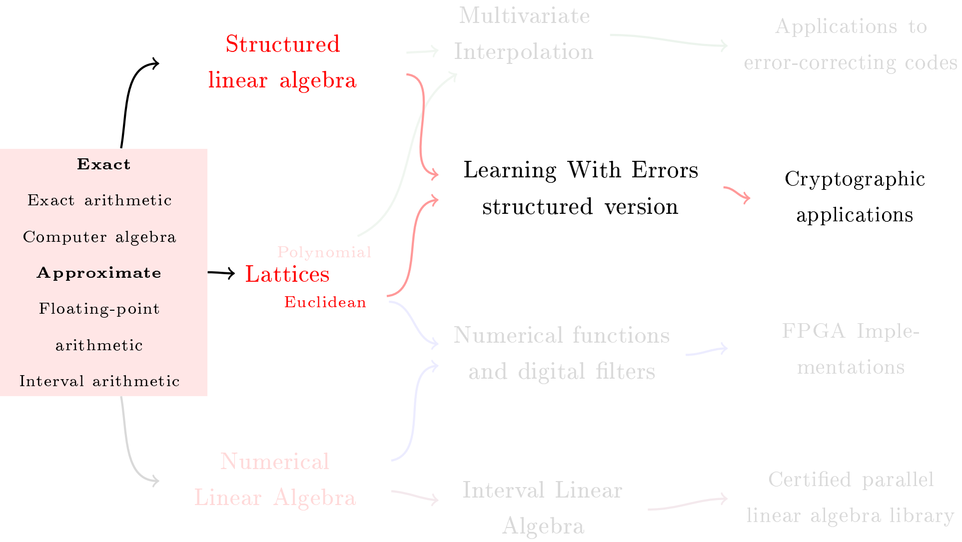
Example matrix (for list-decoding Reed-Solomon codes):



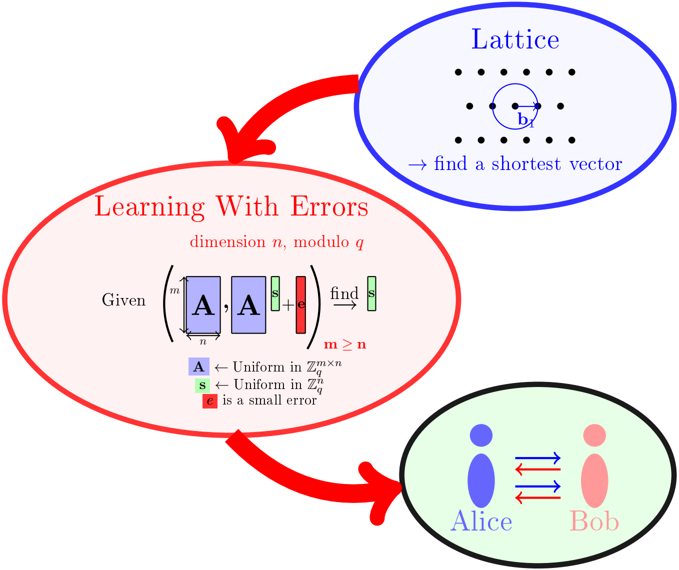
where T_i has a **Toeplitz** structure:



AriC


Structured
linear algebra
Multivariate
Interpolation
Applications to
error-correcting codes
Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point
arithmetic
Interval arithmetic
Polynomial
Lattices
Euclidean
Learning With Errors
structured version
Cryptographic
applications
Numerical functions
and digital filters
FPGA Imple-
mentations
Numerical
Linear Algebra
Interval Linear
Algebra
Certified parallel
linear algebra library

AriC – Lattice-Based Cryptography



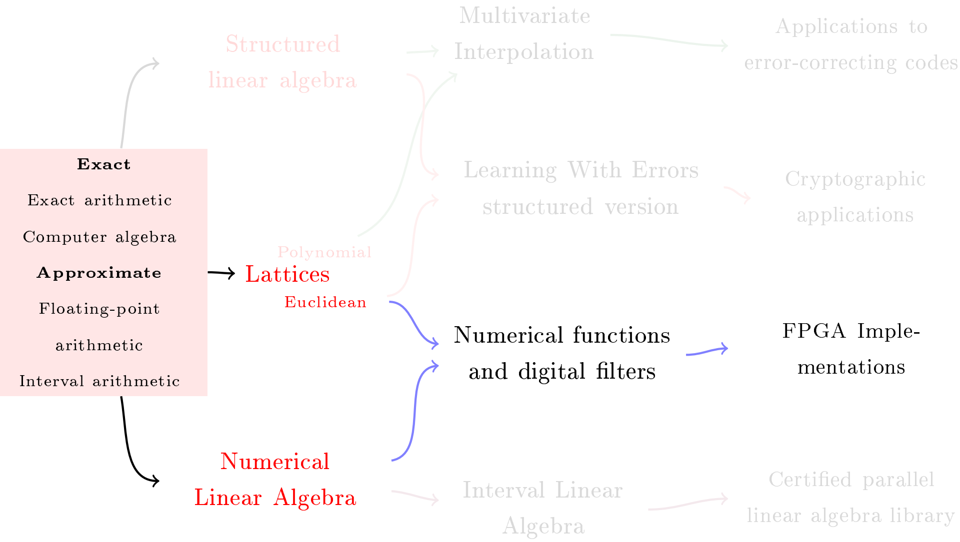
AriC – Lattice-Based Cryptography

- Public Key Encryption
- Identity Based Encryption
- Fully Homomorphic Encryption

- Signature
- Group Signature
- Hash Function

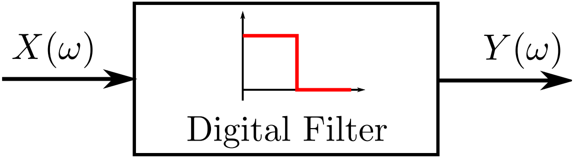
- Cryptographic Multilinear Maps

AriC



Structured linear algebra
Multivariate Interpolation
Applications to error-correcting codes
Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point arithmetic
Interval arithmetic
Learning With Errors structured version
Cryptographic applications
Polynomial
Lattices
Euclidean
Numerical functions and digital filters
FPGA Implementations
Numerical Linear Algebra
Interval Linear Algebra
Certified parallel linear algebra library

AriC – Digital Filter Design



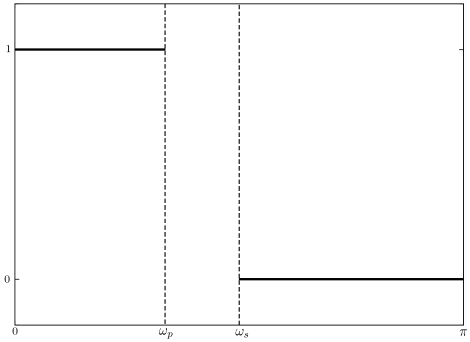
$$Y(\omega) = H_d(\omega) X(\omega), \omega \in [0, \pi]$$

Two types of filters:

- finite impulse response (**FIR**) $\Rightarrow H_d(\omega)$ **polynomial**
- infinite impulse response (**IIR**) $\Rightarrow H_d(\omega)$ **rational function**

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

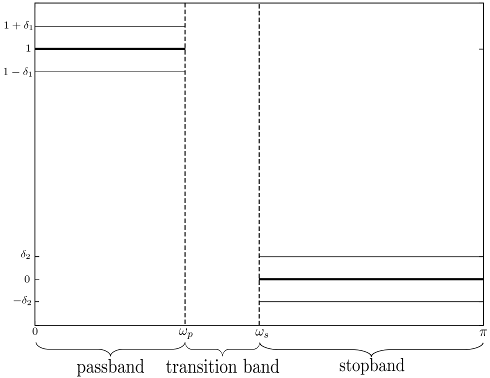
2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

Goal: filter synthesis toolchain for embedded and FPGA targets

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

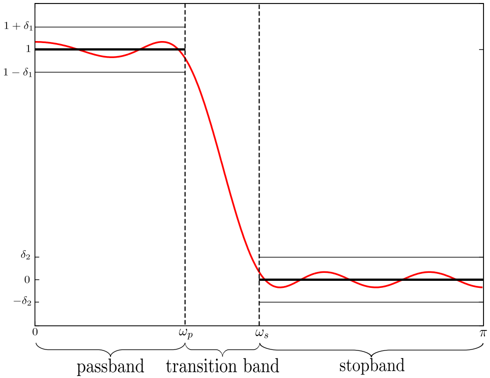
2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

Goal: filter synthesis toolchain for embedded and FPGA targets

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

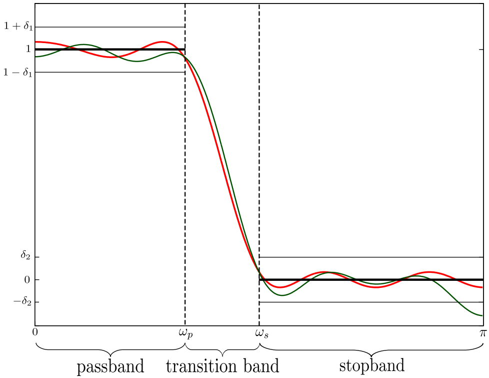
2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

Goal: filter synthesis toolchain for embedded and FPGA targets

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

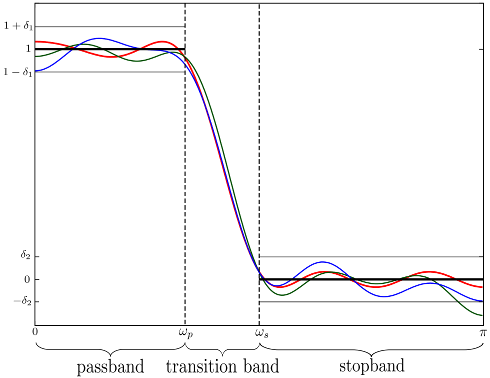
2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

Goal: filter synthesis toolchain for embedded and FPGA targets

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

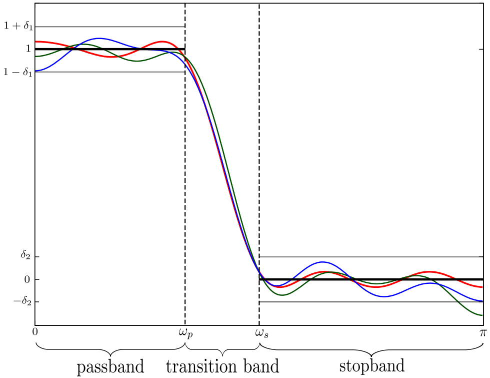
2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

Goal: filter synthesis toolchain for embedded and FPGA targets

AriC – Digital Filter Design

FIR case: $H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$



Steps:

1. Optimal filter computation:

$$H_d(\omega) = \sum_{k=0}^{L} a_k \cos(\omega k)$$

Naive rounding:

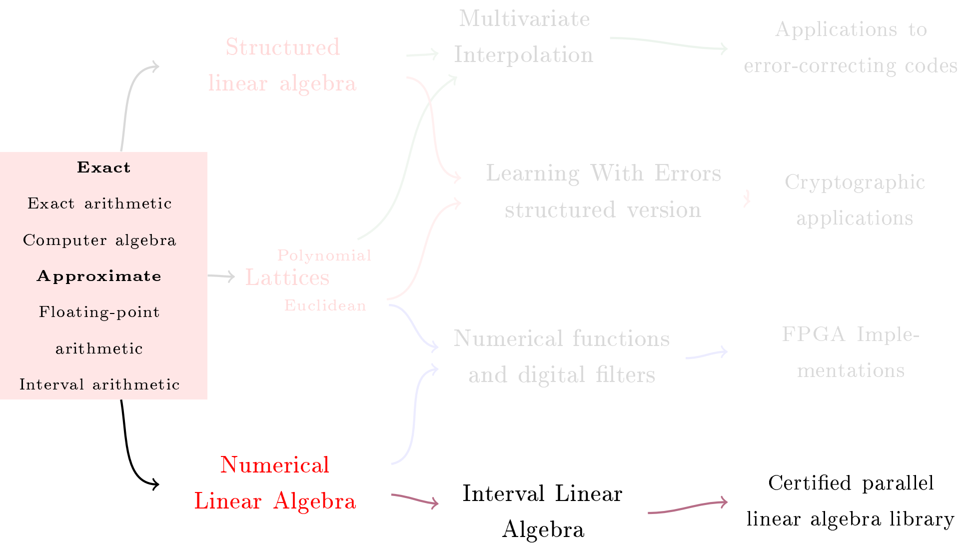
$$\overline{H}_d(\omega) = \sum_{k=0}^{L} \overline{a}_k \cos(\omega k)$$

2. Coefficient quantization:

$$H_d^*(\omega) = \sum_{k=0}^{L} a_k^* \cos(\omega k)$$

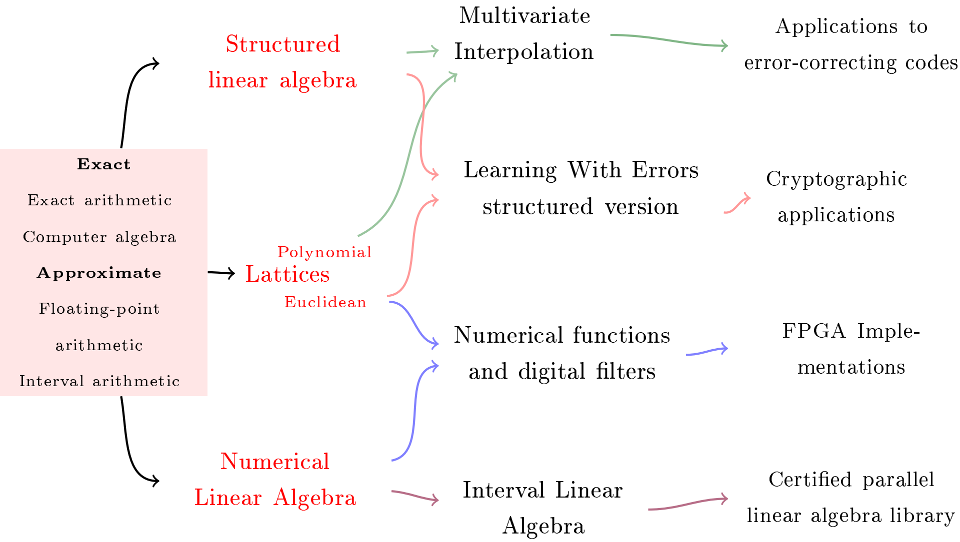
Goal: filter synthesis toolchain for embedded and FPGA targets

AriC



Structured linear algebra
Multivariate Interpolation
Applications to error-correcting codes
Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point arithmetic
Interval arithmetic
Polynomial
Lattices
Euclidean
Learning With Errors structured version
Cryptographic applications
Numerical functions and digital filters
FPGA Implementations
Numerical Linear Algebra
Interval Linear Algebra
Certified parallel linear algebra library

AriC


Exact
Exact arithmetic
Computer algebra
Approximate
Floating-point
arithmetic
Interval arithmetic
Structured
linear algebra
Polynomial
Lattices
Euclidean
Numerical
Linear Algebra
Multivariate
Interpolation
Learning With Errors
structured version
Numerical functions
and digital filters
Interval Linear
Algebra
Applications to
error-correcting codes
Cryptographic
applications
FPGA Imple-
mentations
Certified parallel
linear algebra library