Computing Riemann–Roch spaces for Algebraic Geometry codes

Elena Berardini

joint work with S. Abelard (Thales), A. Couvreur (Inria), G. Lecerf (LIX)

Télécom Paris, Institut polytechnique de Paris, France

Cyber–Crypto Seminar
12 October 2021

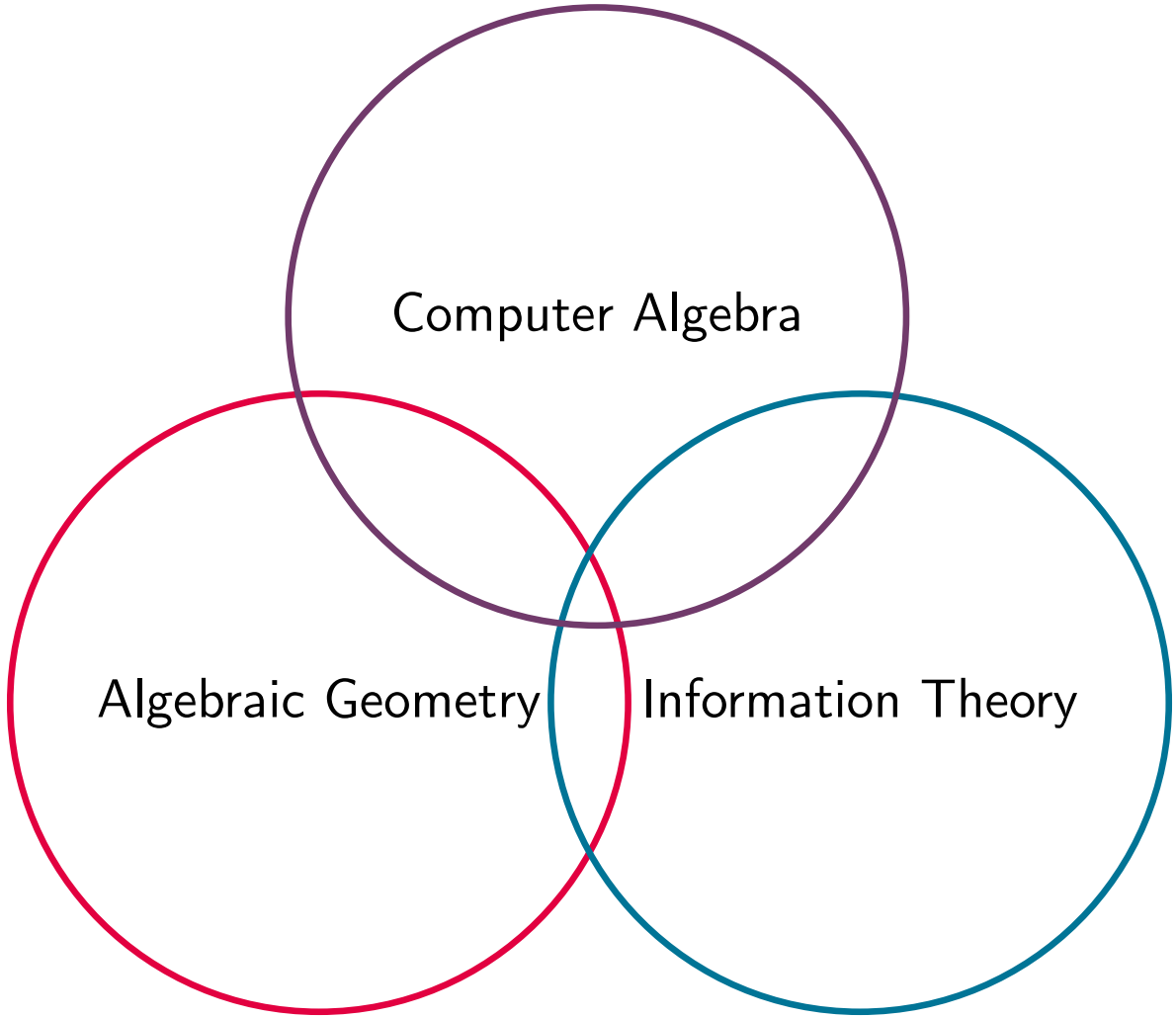

Computer Algebra
Algebraic Geometry
Information Theory

Table of Contents

What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage

What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage

A $\mathbb{F}_q$-vector subspace of $\mathbb{F}_q^n$ (linear codes).

Three parameters:

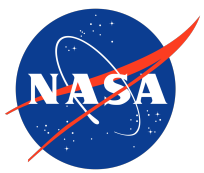
- **n**, the length
- **k**, the dimension
- **d**, the minimum distance

Rate of transmission: k/n
Detects up to $d-1$ errors
Corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors

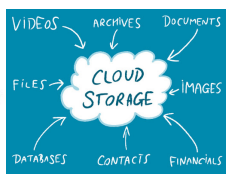
What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage







A $\mathbb{F}_q$-vector subspace of $\mathbb{F}_q^n$ (linear codes).

Three parameters:

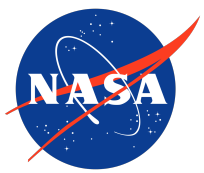
- **n**, the length
- **k**, the dimension
- **d**, the minimum distance

Rate of transmission: k/n
Detects up to $d-1$ errors
Corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors

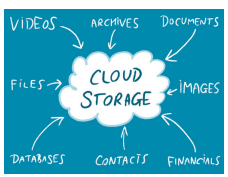
What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage







GOAL: to encode as much data as possible and to detect and correct as many errors as possible!

A $\mathbb{F}_q$-vector subspace of $\mathbb{F}_q^n$ (linear codes).

Three parameters:

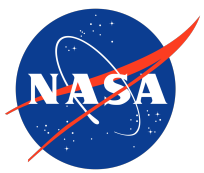
- **n**, the length
- **k**, the dimension
- **d**, the minimum distance

Rate of transmission: k/n
Detects up to $d-1$ errors
Corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors

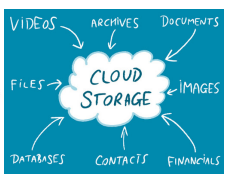
What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage







GOAL: to encode as much data as possible and to detect and correct as many errors as possible!

A $\mathbb{F}_q$-vector subspace of $\mathbb{F}_q^n$ (linear codes).

Three parameters:

- **n**, the length
- **k**, the dimension
- **d**, the minimum distance

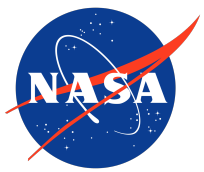
Rate of transmission: k/n
Detects up to $d-1$ errors
Corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors

GOAL: to have **k** and **d** as big as possible!

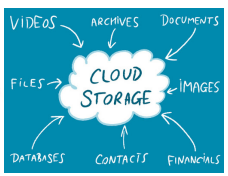
What is an error correcting code?

A tool for transmitting and storing data.

Main feature: detection and correction of the errors that can occur during transmission/storage







GOAL: to encode as much data as possible and to detect and correct as many errors as possible!

A $\mathbb{F}_q$-vector subspace of $\mathbb{F}_q^n$ (linear codes).

Three parameters:

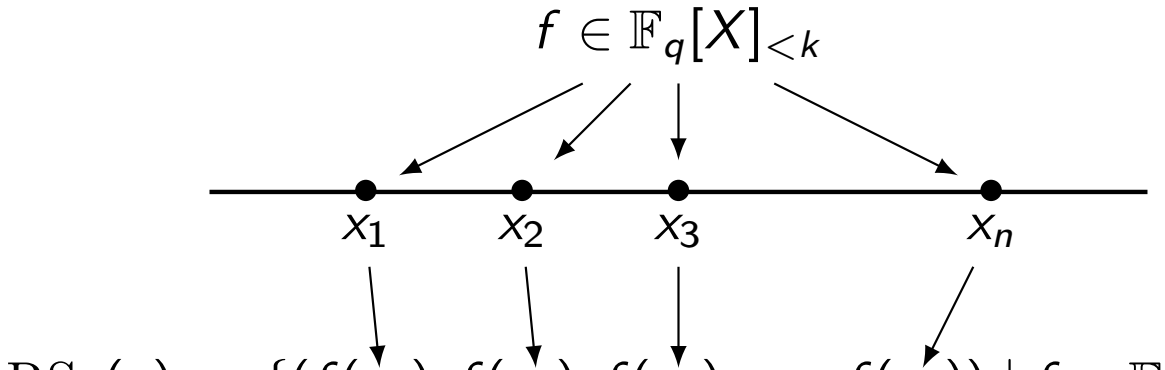
- **n**, the length
- **k**, the dimension
- **d**, the minimum distance

Rate of transmission: k/n
Detects up to $d-1$ errors
Corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors

GOAL: to have **k** and **d** as big as possible!

Singleton Bound: $k + d \leqslant n + 1$
$\rightsquigarrow$ tradeoff between redundancy and capacity of errors-correction

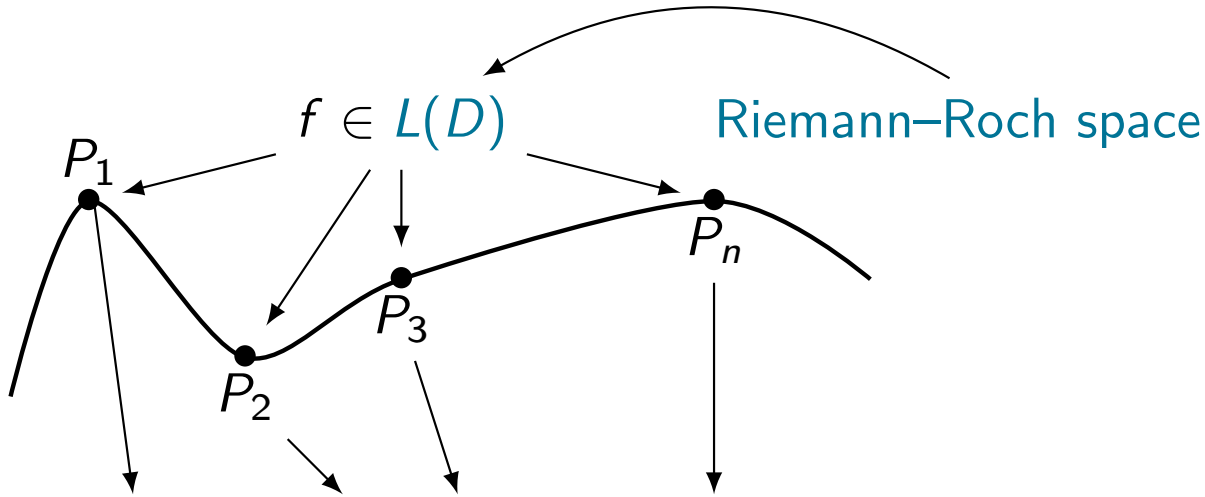
Evaluation codes: from Reed–Solomon codes...



$$\mathrm{RS}_k(\mathbf{x}) := \{(f(x_1), f(x_2), f(x_3), \ldots, f(x_n)) \mid f \in \mathbb{F}_q[X]_{<k}\}$$

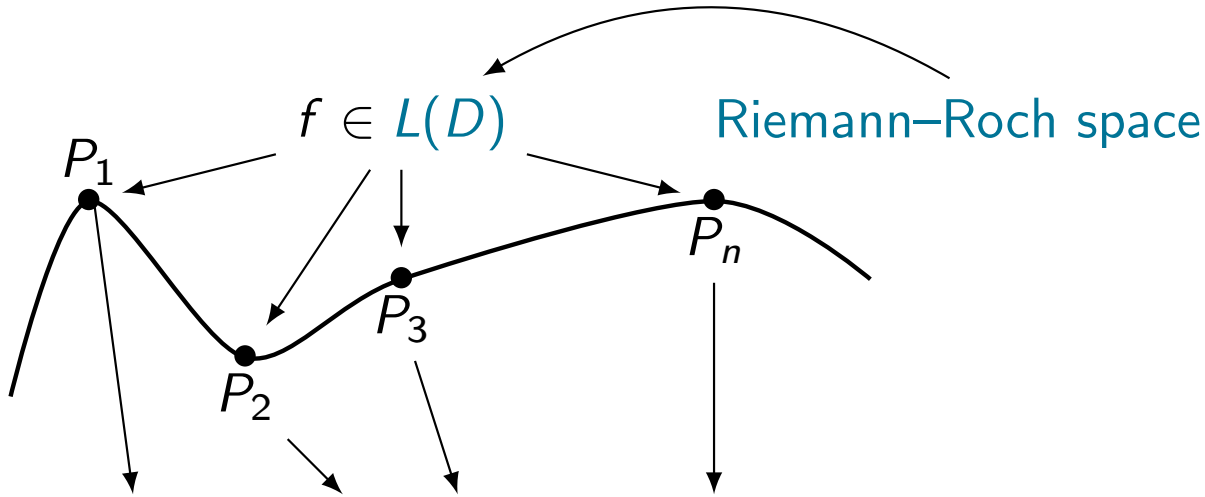
- ✓ Optimal parameters: $k + d = n + 1$ (MDS codes)
- ✓ Efficient decoding algorithm (Berlekamp, 1968)
- ✓ Operations on data
- ⚠ Drawback: require $n \leq q$

...to Algebraic Geometry codes



$$\mathcal{C}((P_i)_i, D) := \{(f(P_1), f(P_2), f(P_3), \ldots, f(P_n)) \mid f \in L(D)\}$$

...to Algebraic Geometry codes



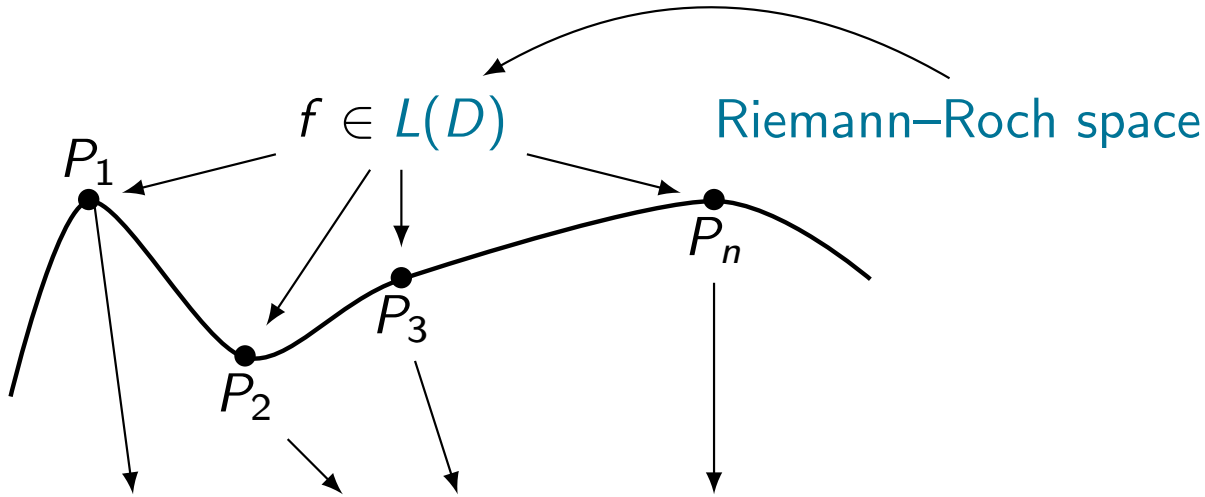
$$\mathcal{C}((P_i)_i, D) := \{(f(P_1), f(P_2), f(P_3), \ldots, f(P_n)) \mid f \in L(D)\}$$

Proposition

The parameters $[n, k, d]$ of AG codes from curves satisfy

$$k \geq \deg D + 1 - g \qquad d \geq n - \deg D.$$

...to Algebraic Geometry codes



$$\mathcal{C}((P_i)_i, D) := \{(f(P_1), f(P_2), f(P_3), \ldots, f(P_n)) \mid f \in L(D)\}$$

Proposition

The parameters $[n, k, d]$ of AG codes from curves satisfy

$$k \geq \deg D + 1 - g \qquad d \geq n - \deg D.$$

AG codes satisfy $n + 1 - g \leq k + d \leq n + 1$

$\rightsquigarrow$ AG codes from curves lie at distance g from optimality

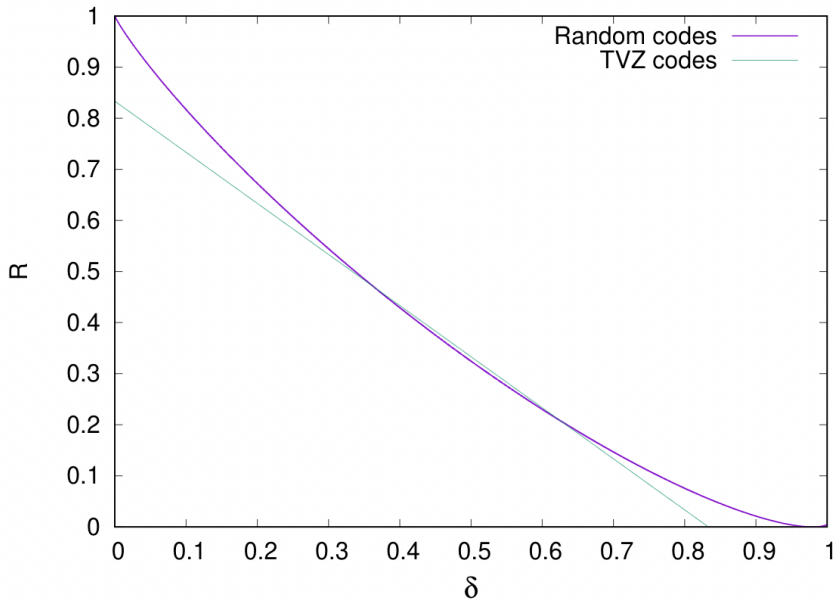
A brief history of algebraic geometry codes

1981: Goppa introduces AG codes from smooth algebraic curves

A brief history of algebraic geometry codes

1981: Goppa introduces AG codes from smooth algebraic curves

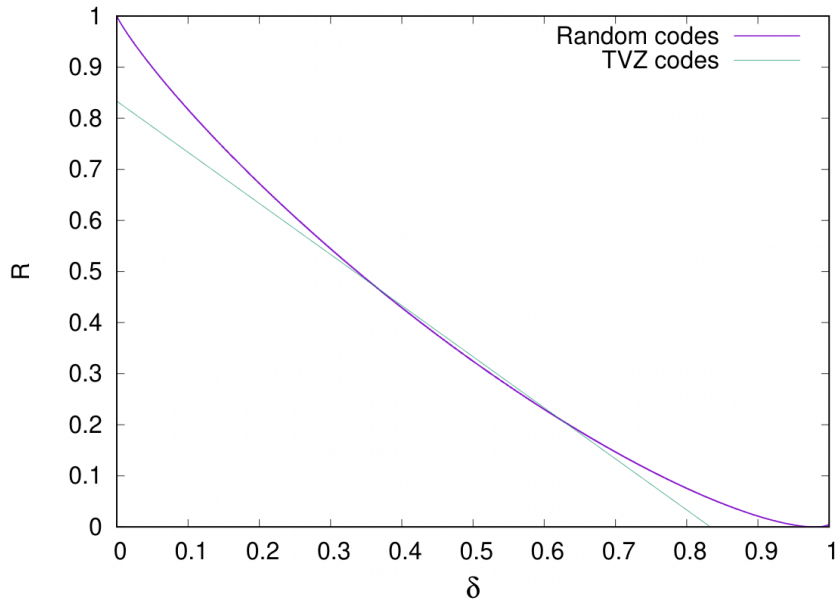
1982: Tsfasman, Vlăduț and Zink use AG codes for beating the Gilbert–Varshamov bound



A brief history of algebraic geometry codes

1981: Goppa introduces AG codes from smooth algebraic curves

1982: Tsfasman, Vlăduț and Zink use AG codes for beating the Gilbert–Varshamov bound



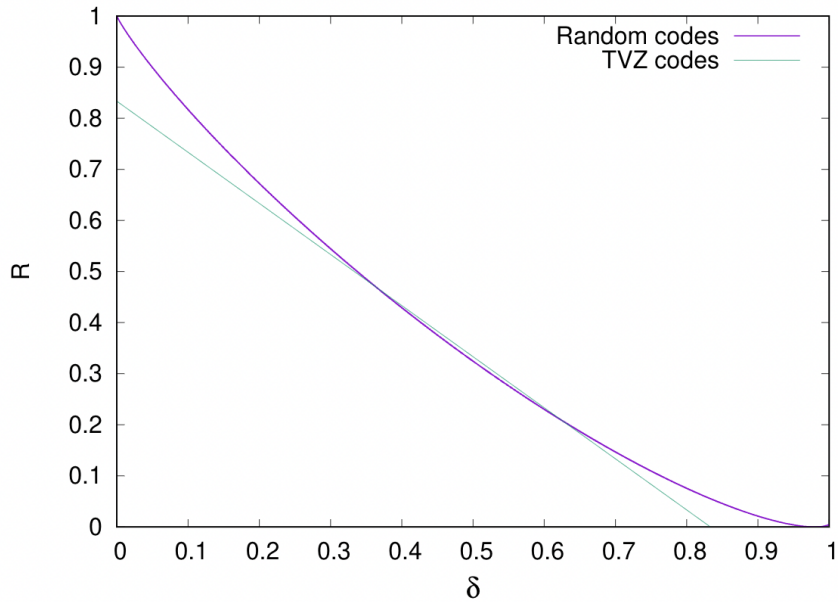
XXc: different families of curves are studied to obtain optimal codes

↪ only curves whose Riemann–Roch spaces are already known are used

A brief history of algebraic geometry codes

1981: Goppa introduces AG codes from smooth algebraic curves

1982: Tsfasman, Vlăduț and Zink use AG codes for beating the Gilbert–Varshamov bound



XXc: different families of curves are studied to obtain optimal codes

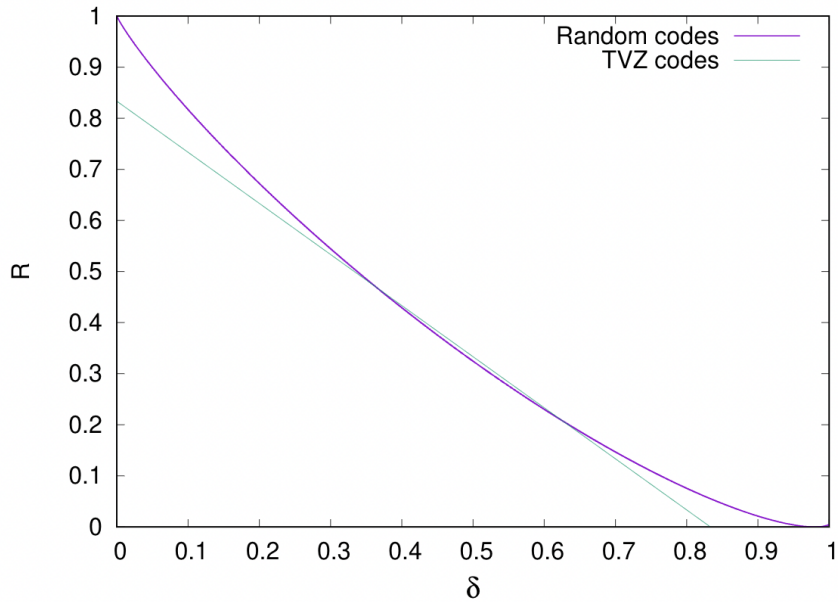
↪ only curves whose Riemann–Roch spaces are already known are used

XXIc: AG codes are used in new area of information theory

A brief history of algebraic geometry codes

1981: Goppa introduces AG codes from smooth algebraic curves

1982: Tsfasman, Vlăduț and Zink use AG codes for beating the Gilbert–Varshamov bound



XXc: different families of curves are studied to obtain optimal codes

↪ only curves whose Riemann–Roch spaces are already known are used

XXIc: AG codes are used in new area of information theory ...let's see how!

Cloud storage

- data is distributed over various servers
- in case of server(s) failure data needs to be reconstructed
- need to limit the bandwidth

Cloud storage

- data is distributed over various servers
- in case of server(s) failure data needs to be reconstructed
- need to limit the bandwidth

Model: one symbol per server

Cloud storage

- data is distributed over various servers
- in case of server(s) failure data needs to be reconstructed
- need to limit the bandwidth

Model: one symbol per server

How do we reconstruct one symbol from a small set of other symbols?

Cloud storage

- data is distributed over various servers
- in case of server(s) failure data needs to be reconstructed
- need to limit the bandwidth

Model: one symbol per server

How do we reconstruct one symbol from a small set of other symbols?

⚠Reed–Solomon codes $\rightsquigarrow$ reconstructing one symbol requires k symbols

Locally Recoverable Codes (LRCs)

Definition

A Locally Recoverable Code with locality ℓ is a code of length n such that for every $i \in \{1, \ldots, n\}$ there exists at least one subset $J_i \in \{1, \ldots, n\}$ not containing i with $\#J_i = \ell$ and such that the coordinate c_i can be recovered from the coordinates c_j for $j \in J_i$.

Locally Recoverable Codes (LRCs)

Definition

A Locally Recoverable Code with locality ℓ is a code of length n such that for every $i \in \{1, \ldots, n\}$ there exists at least one subset $J_i \in \{1, \ldots, n\}$ not containing i with $\#J_i = \ell$ and such that the coordinate c_i can be recovered from the coordinates c_j for $j \in J_i$.

Singleton-type boud: $k + d \leq n - \lceil \frac{k}{\ell} \rceil + 2$

Locally Recoverable Codes (LRCs)

Definition

A Locally Recoverable Code with locality ℓ is a code of length n such that for every $i \in \{1, \ldots, n\}$ there exists at least one subset $J_i \in \{1, \ldots, n\}$ not containing i with $\#J_i = \ell$ and such that the coordinate c_i can be recovered from the coordinates c_j for $j \in J_i$.

Singleton-type boud: $k + d \leq n - \lceil \frac{k}{\ell} \rceil + 2$

AG codes enter the game:

- Barg, Tamo and Vlăduț[1] proposed constructions of LRC using curves that are optimal (parameters reach the Singleton-type bound)
- extension of this approach to more curves and surfaces[2]
- optimal exemples of LRC from (fibered) surfaces[3]

[1] IEEE Transactions on Information Theory, 2017
[2] Barg et al, Algebraic geometry for coding theory and cryptography, 2017
[3] Salgado, Varilly-Alvarado, Voloch, IEEE Transactions on Information Theory, 2021

Locally Recoverable Codes (LRCs)

Definition

A Locally Recoverable Code with locality ℓ is a code of length n such that for every $i \in \{1, \ldots, n\}$ there exists at least one subset $J_i \in \{1, \ldots, n\}$ not containing i with $\#J_i = \ell$ and such that the coordinate c_i can be recovered from the coordinates c_j for $j \in J_i$.

Singleton-type boud: $k + d \leq n - \lceil \frac{k}{\ell} \rceil + 2$

AG codes enter the game:

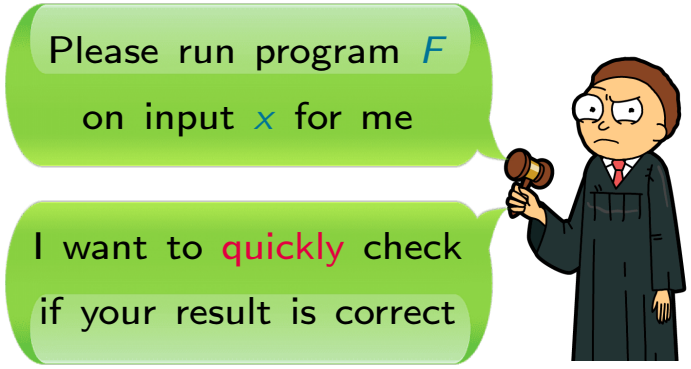
- Barg, Tamo and Vlăduț[1] proposed constructions of LRC using curves that are optimal (parameters reach the Singleton-type bound)
- extension of this approach to more curves and surfaces[2]
- optimal exemples of LRC from (fibered) surfaces[3]

⚠Yes, we can construct AG codes from surfaces...but this is another story!

[1] IEEE Transactions on Information Theory, 2017
[2] Barg et al, Algebraic geometry for coding theory and cryptography, 2017
[3] Salgado, Varilly-Alvarado, Voloch, IEEE Transactions on Information Theory, 2021

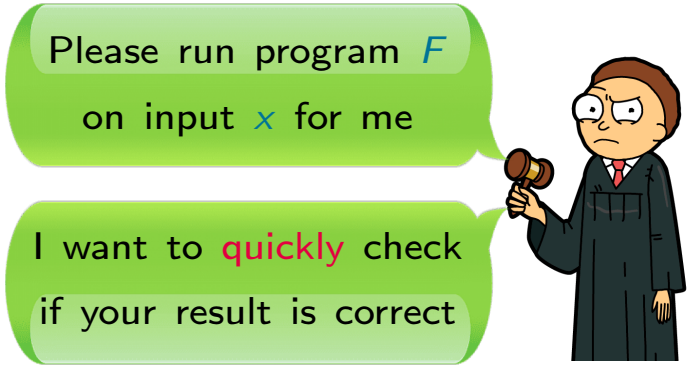
Verifiable Computing

Powerful Prover
(e.g. a server)



Weak Verifier
(e.g. a client)

Verifiable Computing



Powerful Prover
(e.g. a server)

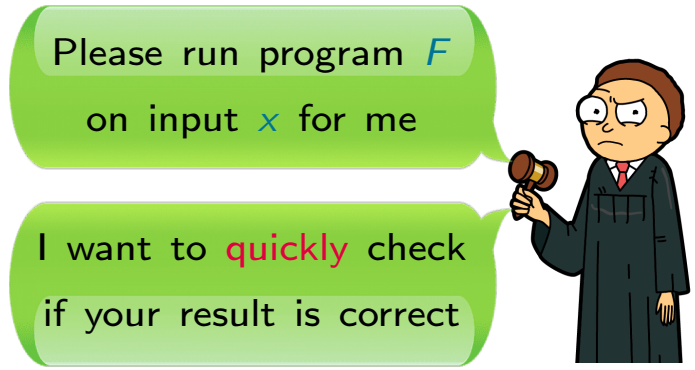
outputs result y and
proof of correctness π

y, π →

Weak Verifier
(e.g. a client)

checks validity of π
for statement $y = F(x)$

Verifiable Computing



Powerful Prover
(e.g. a server)

outputs result y and
proof of correctness π

$\xrightarrow{y,\pi}$

Weak Verifier
(e.g. a client)

checks validity of π
for statement $y = F(x)$

$\vdots$

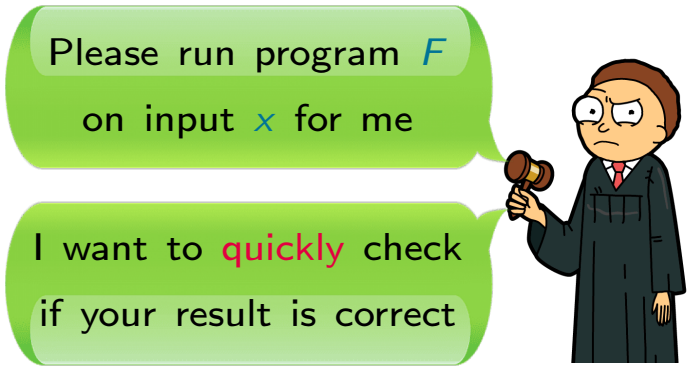
Proximity to a code C

Provers produces a word

- $c \in C$ if the statement $y = F(x)$ holds,
- $\tilde{c}$ is very far from C otherwise.

Applications: cryptocurrencies, blockchain...

Verifiable Computing



Powerful Prover
(e.g. a server)

outputs result y and
proof of correctness π

$\xrightarrow{y,\pi}$

Weak Verifier
(e.g. a client)

checks validity of π
for statement $y = F(x)$

Proximity to a code C

Provers produces a word

- $c \in C$ if the statement $y = F(x)$ holds,
- $\tilde{c}$ is very far from C otherwise.

Applications: cryptocurrencies, blockchain...

Which codes can be used? AG codes seem a good option[4]

[4] S. Bordage and J. Nardi, preprint, 2020

McEliece cryptosystem (1978)

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

Blocks his message into vectors m_i of length k

Randomly constructs e a n-vector of weight t

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

Blocks his message into vectors m_i of length k

Randomly constructs e a n-vector of weight t

Computes

$y_i = m_i \bar{G} + e$

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

Blocks his message into vectors m_i of length k
Randomly constructs e a n-vector of weight t

Receives y_i

Computes $y_i = m_i \bar{G} + e$

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

Blocks his message into vectors m_i of length k
Randomly constructs e a n-vector of weight t

Receives y_i

Computes $y_i = m_i\bar{G} + e$

Computes $y_i P^{-1} = (m_i\bar{G} + e)P^{-1}$
$= m_i SG + eP^{-1} = m_i SG + e'$
Applies $\mathcal{A}$ to retrieve $m_i SG$

$$m_i = m_i SG \times G^{-1}S^{-1}$$

McEliece cryptosystem (1978)

- G, matrix of a $[n, k, 2t + 1]$-code
- $\mathcal{A}$, decoding algorithm
- S, a $k \times k$ matrix
- P, a $n \times n$ matrix

Computes $\bar{G} = SGP$

PubKey $= (\bar{G}, t)$, SecKey$= (G, P, S, \mathcal{A})$

Blocks his message into vectors m_i of length k
Randomly constructs e a n-vector of weight t

Receives y_i

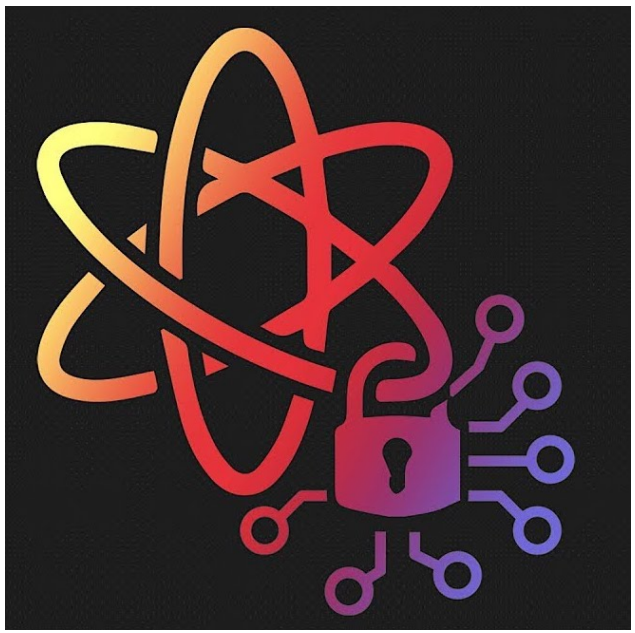
Computes $y_i = m_i\bar{G} + e$

Computes $y_i P^{-1} = (m_i\bar{G} + e)P^{-1}$
$= m_i SG + eP^{-1} = m_i SG + e'$
Applies $\mathcal{A}$ to retrieve $m_i SG$

$$m_i \quad = \quad m_i SG \times G^{-1}S^{-1}$$

McEliece cryptosystem for post-quantum cryptography

- security relies on
  - computational hardness of decoding a random code
  - computational hardness of distinguishing a structured code from a random code
- ✓ Post-quantum
- ⚠ Requires huge key sizes



Classic McEliece[5], a cryptosystem using binary AG codes, is at the third round of NIST's Post-Quantum Cryptography Standardization Project.

[5] Berstein et al, NIST submission, 2017

Riemann–Roch spaces: AG codes and beyond

Explicit construction of AG codes for

- Locally Recoverable Codes
- Verifiable Computing
- McEliece cryptosystem

$\rightsquigarrow$ need of explicit computation of Riemann–Roch spaces

Riemann–Roch spaces: AG codes and beyond

Explicit construction of AG codes for

- Locally Recoverable Codes
- Verifiable Computing
- McEliece cryptosystem

$\rightsquigarrow$ need of explicit computation of Riemann–Roch spaces

This can be also useful for...

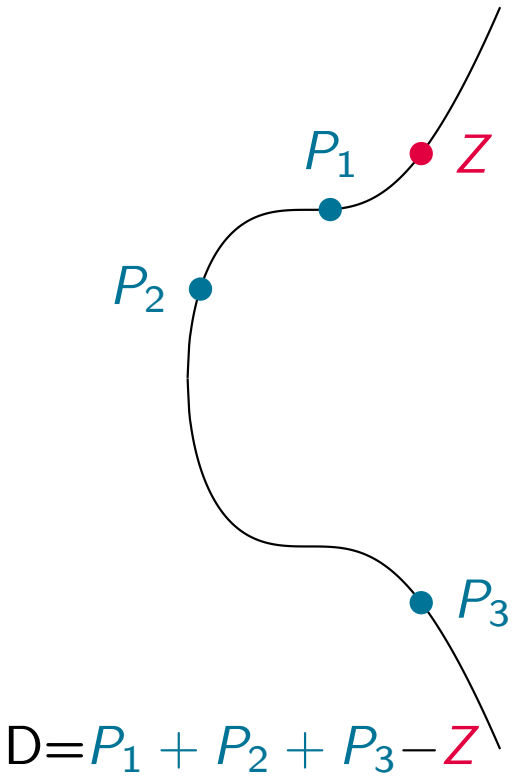
- Group operations on Jacobians of curves[6]
- Symbolic integration[7]

[6] K. Khuri-Makdisi, Mathematics of Computations, 2007

[7] J.H. Davenport, Intern. Symp. on Symbolic and Algebraic Manipulation, 1979

Riemann–Roch space

Divisor on a curve $\mathcal{C}$: $D = \sum_{P \in \mathcal{C}} n_P P$



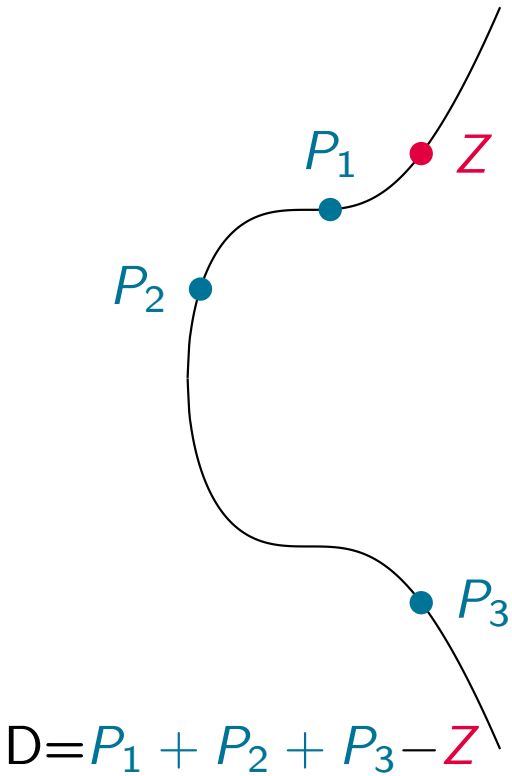
The **Riemann–Roch space** $L(D)$ is the space of all functions $\frac{G}{H} \in \mathbb{K}(\mathcal{C})$ s. t.:

- if $n_P < 0$ then P must be a zero of G (of multiplicity $\geqslant -n_P$)
- if $n_P > 0$ then P can be a zero of H (of multiplicity $\leqslant n_P$)
- G/H has not other poles outside the points P with $n_P > 0$

Here: Z must be a zero of G, the P_i's can be zeros of H

Riemann–Roch space

Divisor on a curve $\mathcal{C}$: $D = \sum_{P \in \mathcal{C}} n_P P$



The **Riemann–Roch space** $L(D)$ is the space of all functions $\frac{G}{H} \in \mathbb{K}(\mathcal{C})$ s. t.:

- if $n_P < 0$ then P must be a zero of G (of multiplicity $\geqslant -n_P$)
- if $n_P > 0$ then P can be a zero of H (of multiplicity $\leqslant n_P$)
- G/H has not other poles outside the points P with $n_P > 0$

Here: Z must be a zero of G, the P_i's can be zeros of H

Riemann–Roch theorem $\rightsquigarrow$ dimension of $L(D) = \deg D + 1 - g$

where the degree of a divisor is $\deg D = \sum_P n_P$

Toy example

Take $\mathcal{C} = \mathbb{P}^1$, $P = [0:1]$ and $Q = [1:1]$. Set $D = P - Q$, then

$$f \in L(D) \iff \begin{cases} \text{f has a zero of order at least 1 at } Q \\ \text{f can have a pole of order at most 1 at } P \\ \text{f has no other poles outside } P \end{cases}$$

Toy example

Take $\mathcal{C} = \mathbb{P}^1$, $P = [0:1]$ and $Q = [1:1]$. Set $D = P - Q$, then

$$f \in L(D) \iff \begin{cases} \text{f has a zero of order at least 1 at } Q \\ \text{f can have a pole of order at most 1 at } P \\ \text{f has no other poles outside } P \end{cases}$$

$$f = \frac{X-1}{X} \text{ is a solution}$$

$$g = 0, \deg D = 0 \xrightarrow[\text{Theorem}]{\text{Riemann–Roch}} \dim L(D) = \deg D + 1 - g = 1$$

$$\rightarrow \text{f generates the solutions space}$$

Toy example

Take $\mathcal{C} = \mathbb{P}^1$, $P = [0:1]$ and $Q = [1:1]$. Set $D = P - Q$, then

$$f \in L(D) \iff \begin{cases} \text{f has a zero of order at least 1 at } Q \\ \text{f can have a pole of order at most 1 at } P \\ \text{f has no other poles outside } P \end{cases}$$

$$f = \frac{X-1}{X} \text{ is a solution}$$

$$g = 0, \deg D = 0 \xrightarrow[\text{Theorem}]{\text{Riemann–Roch}} \dim L(D) = \deg D + 1 - g = 1$$

$$\rightarrow \text{f generates the solutions space}$$

⚠no explicit method to compute a basis of $L(D)$
How do we handle the problem in general?

Riemann-Roch problem: state of the art

Geometric methods:
(Brill–Noether theory ∼1874)
- Goppa, Le Brigand–Risler (80's)
- Huang–Ierardi (90's)
- Khuri-Makdisi (2007)
- Le Gluher–Spaenlehauer (2018)
- Abelard–Couvreur–Lecerf (2020)

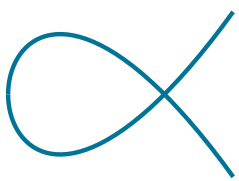
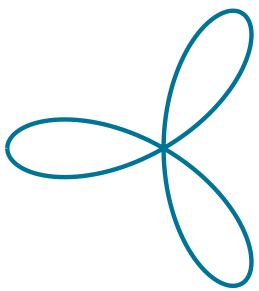
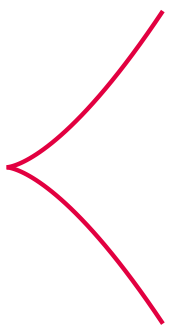
Arithmetic methods:
(Ideals in function fields)
- Hensel–Landberg (1902)
- Coates (1970)
- Davenport (1981)
- Hess (2001)

Riemann-Roch problem: state of the art

Geometric methods:
(Brill–Noether theory ∼1874)
- Goppa, Le Brigand–Risler (80's)
- Huang–Ierardi (90's)
- Khuri-Makdisi (2007)
- Le Gluher–Spaenlehauer (2018)
- Abelard–Couvreur–Lecerf (2020)

Arithmetic methods:
(Ideals in function fields)
- Hensel–Landberg (1902)
- Coates (1970)
- Davenport (1981)
- Hess (2001)

Nodal/ordinary curves:	Las Vegas algorithm computing $L(D)$ in $\tilde{\mathcal{O}}((\delta^2 + \deg D)^{\frac{\omega+1}{2}})$ field operations[8]
Non-ordinary curves:	⚠ no explicit complexity exponent

[8] here $2 \leqslant \omega \leqslant 3$ is a feasible exponent for linear algebra ($\omega = 2.373$)

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H) P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H) P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Description of $L(D)$ for $\mathcal{C} : F(X, Y, Z) = 0$ a plane projective curve.

Non-zero elements are of the form $\frac{G_i}{H}$ where

- *H satisfies $(H) \geqslant D$*
- *H passes through all the singular points of $\mathcal{C}$ with ad hoc multiplicities*
- *$\deg G_i = \deg H$, G_i coprime with F and $(G_i) \geqslant (H) - D$*

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H) P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Description of $L(D)$ for $\mathcal{C} : F(X, Y, Z) = 0$ a plane projective curve.

Non-zero elements are of the form $\frac{G_i}{H}$ where

- *H satisfies $(H) \geqslant D$*
- *H passes through all the singular points of $\mathcal{C}$ with ad hoc multiplicities*
- $\deg G_i = \deg H$, *G_i coprime with F and $(G_i) \geqslant (H) - D$*

How do we handle singular points?

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H)P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Description of $L(D)$ for $\mathcal{C} : F(X, Y, Z) = 0$ a plane projective curve.

Non-zero elements are of the form $\frac{G_i}{H}$ where

- *H satisfies $(H) \geqslant D$*
- *H satisfies $(H) \geqslant \mathcal{A}$ (we say that "H is adjoint to the curve")*
- *$\deg G_i = \deg H$, G_i coprime with F and $(G_i) \geqslant (H) - D$*

How do we handle singular points?

$\rightsquigarrow$ the adjoint divisor $\mathcal{A}$ "encodes" the singular points of $\mathcal{C}$ with their multiplicities

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H)P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Description of $L(D)$ for $\mathcal{C} : F(X, Y, Z) = 0$ a plane projective curve.

Non-zero elements are of the form $\frac{G_i}{H}$ where

- *H satisfies $(H) \geqslant D$*
- *H satisfies $(H) \geqslant \mathcal{A}$*
- $\deg G_i = \deg H$, *G_i coprime with F and $(G_i) \geqslant (H) - D$*

How do we handle singular points?

$\rightsquigarrow$ the adjoint divisor $\mathcal{A}$ "encodes" the singular points of $\mathcal{C}$ with their multiplicities

How do we handle divisors?

Brill–Noether in a nutshell

Brill–Noether method $\rightsquigarrow$ NSC on H and G such that $G/H \in L(D)$

Notation:

- $(H) = \sum_{P \in \mathcal{C}} \mathrm{ord}_P(H) P$ (zeros of H with multiplicity)
- $D \geqslant D'$ means $D - D' = \sum n_P P$ with $n_P \geqslant 0$ for every P

Description of $L(D)$ for $\mathcal{C} : F(X, Y, Z) = 0$ a plane projective curve.

Non-zero elements are of the form $\frac{G_i}{H}$ where

- *H satisfies $(H) \geqslant D$*
- *H satisfies $(H) \geqslant \mathcal{A}$*
- $\deg G_i = \deg H$, *G_i coprime with F and $(G_i) \geqslant (H) - D$*

How do we handle singular points?

$\rightsquigarrow$ the adjoint divisor $\mathcal{A}$ "encodes" the singular points of $\mathcal{C}$ with their multiplicities

How do we handle divisors?

series expansions of multi-set representations $((P_i)_i, m_i)$ $\rightsquigarrow$ routines on divisors have negligible cost

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$

Step 2: Compute a common denominator H

Step 3: Compute $(H) - D$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$

Step 2: Compute a common denominator H

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$

Step 2: Compute a common denominator H

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

The adjoint condition via Puiseux expansions

Let $F \in \mathbb{K}[x, y]$ be absolutely irreducible, monic in y and of degree d_y in y. The roots of $F \in \mathbb{K}((x))[y]$ in $\cup_{e \geqslant 1} \overline{\mathbb{K}}((x^{1/e}))$ are its Puiseux expansions $\varphi_0, \ldots, \varphi_{d_y - 1}$, so that F writes

$$F = \prod_{i=1}^{d_y - 1} (y - \varphi_i) = \prod_{i=1}^{d_y - 1} (y - \sum_{j=n}^{\infty} \beta_{i,j} x^{j/e_i}).$$

Toy example: $F = y^2 - x^3 \rightsquigarrow F = (y - x^{3/2})(y + x^{3/2})$

Fix φ_0 of degree e_0 and let ζ be a primitive e_0-th root of unity. Then for $0 \leqslant k < e_0$ we can construct other e_0 PE by replacing x^{1/e_0} by $\zeta^k x^{1/e_0}$. These PE are all equivalent and represented by one

Rational Puiseux Expansion: a pair $(X(t), Y(t)) = (\gamma t^e, \sum_{j=n}^{\infty} \beta_j t^j)$

Toy example (continue): $\rightsquigarrow (X(t), Y(t)) = (t^2, t^3)$

⚠RPE are often defined over an extension of $\mathbb{K}$.
It is an algorithmic question of taking minimal extension of fields.

The adjoint divisor

The adjoint divisor is

$$\mathcal{A} = \sum_{P \in \mathrm{Sing}(\mathcal{C})} - \sum_{\mathcal{P}|P} \mathrm{val}_{\mathcal{P}}\left(\frac{dx}{F_y}\right)\mathcal{P}$$

$$\xrightarrow[\text{Puiseux expansions}]{\text{Using Rational}} \mathrm{val}_{\mathcal{P}}\left(\frac{dx}{F_y}\right) = \mathrm{val}_t\left(\frac{et^{e-1}}{F_y(X(t), Y(t), 1)}\right)$$

Example

Consider $\mathcal{C} : y^2 - x^3 = 0$, $(0,0)$ *is the (only, non-ordinary) singular point*

$$(X(t), Y(t)) = (t^2, t^3) \rightsquigarrow \mathrm{val}_{\mathcal{P}}\left(\frac{dx}{F_y}\right) = \mathrm{val}_t\left(\frac{2t}{2t^3}\right) = -2$$

Computation: algorithms for Puiseux expansions of germs of curves[9]

$\rightsquigarrow \mathcal{A}$ computed with an expected number of $\tilde{O}(\delta^3)$ field operations

[9] A. Poteaux and M. Weimann, Annales Herni Lebesgue, 2021

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$ ✓ $\leftarrow \tilde{O}(\delta^3)$

Step 2: Compute H

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$ ✓ $\leftarrow \tilde{O}(\delta^3)$

Step 2: Compute H

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Finding a denominator in practice

Straightforward linear solving

Let $d = \deg H$.

$$\text{Condition } (H) \geqslant \mathcal{A} + D_+$$

$\rightsquigarrow$ linear system with $\deg \mathcal{A} + \deg D \sim \delta^2 + \deg D$ equations

$\rightsquigarrow$ Gaussian elimination costs

$$\tilde{O}((d\delta + \delta^2 + \deg D)^\omega)$$

Finding a denominator in practice

Straightforward linear solving

Let $d = \deg H$.

$$\text{Condition } (H) \geqslant \mathcal{A} + D_+$$

$\rightsquigarrow$ linear system with $\deg \mathcal{A} + \deg D \sim \delta^2 + \deg D$ equations

$\rightsquigarrow$ Gaussian elimination costs

$$\tilde{O}((d\delta + \delta^2 + \deg D)^\omega)$$

How big is d?

We proved that $d = \left\lceil \frac{(\delta-1)(\delta-2)+\deg D}{\delta} \right\rceil$ is enough

$\rightsquigarrow \tilde{O}((\delta^2 + \deg D)^\omega)$ field operations[10]

[10] again $2 \leqslant \omega \leqslant 3$ is a feasible exponent for linear algebra ($\omega = 2.373$)

Second method: structured linear algebra

$$\mathrm{val}_t(H(X(t), Y(t), 1) \geqslant \mathrm{val}_t\left(\frac{et^{e-1}}{F_y(X(t), Y(t), 1)}\right)$$

$\rightsquigarrow$ space of polynomials $H(x, y)$ satisfying these conditions is a $\mathbb{K}[x]$-module

$\rightsquigarrow$ computing a basis[11] costs $\tilde{O}((\delta^2 + \deg D)^\omega)$

Same complexity exponent but...

Benefits:

- bases with smaller representation size in general
- better complexity bound for algebraically closed fields
- possibility of future improvements

[11] C.-P. Jeannerod, V. Neiger, É. Schost and G. Villard, Journal of Symbolic Computation, 2017

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$ ✓ $\leftarrow \tilde{O}(\delta^3)$

Step 2: Compute H ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^\omega)$

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i (similar to Step 2)

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Sketch of the algorithm

Input

$\mathcal{C} : F(X, Y, Z) = 0$ *a plane projective curve,* D *a smooth divisor.*

Step 1: Compute the adjoint divisor $\mathcal{A}$ ✓ $\leftarrow \tilde{O}(\delta^3)$

Step 2: Compute H ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^\omega)$

Step 3: Compute $(H) - D$ ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^2)$

Step 4: Compute numerators G_i ✓ $\leftarrow \tilde{O}((\delta^2 + \deg D)^\omega)$

Output

A basis of the Riemann–Roch space $L(D)$ *in terms of* H *and the* G_i*.*

Theorem (Abelard, B., Couvreur, Lecerf)

Las Vegas algorithm computing $L(D)$ *in* $\tilde{\mathcal{O}}((\delta^2 + \deg D)^\omega)$ *field operations*[12]

[12] with $2 \leqslant \omega \leqslant 3$ is a feasible exponent for linear algebra ($\omega = 2.373$)

Future questions about R–R spaces and AG codes

Computing Riemann–Roch spaces of curves.

- ◇ Implementation including fast structured linear algebra.
- ◇ Computing Riemann–Roch spaces of non-ordinary curves in "small" positive characteristic (in progress with A. Couvreur and G. Lecerf)
- ◇ Improving the complexity in the non-ordinary case (sub–quadratic?)

Future questions about R–R spaces and AG codes

Computing Riemann–Roch spaces of curves.

- ◇ Implementation including fast structured linear algebra.
- ◇ Computing Riemann–Roch spaces of non-ordinary curves in "small" positive characteristic (in progress with A. Couvreur and G. Lecerf)
- ◇ Improving the complexity in the non-ordinary case (sub–quadratic?)

AG codes in higher dimension.

- ◇ Computing Riemann–Roch spaces of surfaces
  $\rightsquigarrow$ explicit construction of (good) AG codes from surfaces.

Future questions about R–R spaces and AG codes

Computing Riemann–Roch spaces of curves.

- ◇ Implementation including fast structured linear algebra.
- ◇ Computing Riemann–Roch spaces of non-ordinary curves in "small" positive characteristic (in progress with A. Couvreur and G. Lecerf)
- ◇ Improving the complexity in the non-ordinary case (sub–quadratic?)

AG codes in higher dimension.

- ◇ Computing Riemann–Roch spaces of surfaces
  $\rightsquigarrow$ explicit construction of (good) AG codes from surfaces.

Rank metric codes.

- ◇ Can we use curves and/or Riemann–Roch spaces to construct good codes in the rank metric?

Future questions about R–R spaces and AG codes

Computing Riemann–Roch spaces of curves.

- ◇ Implementation including fast structured linear algebra.
- ◇ Computing Riemann–Roch spaces of non-ordinary curves in "small" positive characteristic (in progress with A. Couvreur and G. Lecerf)
- ◇ Improving the complexity in the non-ordinary case (sub–quadratic?)

AG codes in higher dimension.

- ◇ Computing Riemann–Roch spaces of surfaces
  $\rightsquigarrow$ explicit construction of (good) AG codes from surfaces.

Rank metric codes.

- ◇ Can we use curves and/or Riemann–Roch spaces to construct good codes in the rank metric?

Thank you for your attention!

Questions?
elena.berardini@telecom-paris.fr