5. Extended Euclidean algorithm and interpolation from erroneous data

CS-E4500 Advanced Course on Algorithms
Spring 2019

Petteri Kaski
Department of Computer Science
Aalto University

Lecture schedule

Tue 15 Jan:	1. Polynomials and integers
Tue 22 Jan:	2. The fast Fourier transform and fast multiplication
Tue 29 Jan:	3. Quotient and remainder
Tue 5 Feb:	4. Batch evaluation and interpolation
Tue 12 Feb:	5. Extended Euclidean algorithm and interpolation from erroneous data
Tue 19 Feb:	*Exam week — no lecture*
Tue 27 Feb:	6. Identity testing and probabilistically checkable proofs
Tue 5 Mar:	*Break — no lecture*
Tue 12 Mar:	7. Finite fields
Tue 19 Mar:	8. Factoring polynomials over finite fields
Tue 26 Mar:	9. Factoring integers

CS-E4500 Advanced Course in Algorithms (5 ECTS, III–IV, Spring 2019)

2019	KALENTERI	2019

Tammikuu	Helmikuu	Maaliskuu	Huhtikuu	Toukokuu	Kesäkuu
1 Ti Uudenvuodenpäivä	1 Pe	1 Pe	1 Ma Vk 14 T9	1 Ke Vappu	1 La
2 Ke	2 La	2 La	2 Ti	2 To	2 Su
3 To	3 Su D3	3 Su	3 Ke	3 Pe	3 Ma Vk 23
4 Pe	4 Ma Vk 06 T3	4 Ma Vk 10	4 To	4 La	4 Ti
5 La	5 Ti L4	5 Ti Laskiainen	5 Pe	5 Su	5 Ke
6 Su Loppiainen	6 Ke	6 Ke Break	6 La	6 Ma Vk 19	6 To
7 Ma Vk 02	7 To Q4	7 To	7 Su	7 Ti	7 Pe
8 Ti	8 Pe	8 Pe	8 Ma Vk 15	8 Ke	8 La
9 Ke	9 La	9 La	9 Ti	9 To	9 Su Helluntaipäivä
10 To	10 Su D4	10 Su D6	10 Ke	10 Pe	10 Ma Vk 24
11 Pe	11 Ma Vk 07 T4	11 Ma Vk 11 T6	11 To	11 La	11 Ti
12 La	12 Ti L5	12 Ti L7	12 Pe	12 Su Äitienpäivä	12 Ke
13 Su	13 Ke	13 Ke	13 La	13 Ma Vk 20	13 To
14 Ma Vk 03	14 To Q5	14 To Q7	14 Su Palmusunnuntai	14 Ti	14 Pe
15 Ti L1	15 Pe	15 Pe	15 Ma Vk 16	15 Ke	15 La
16 Ke	16 La	16 La	16 Ti	16 To	16 Su
17 To Q1	17 Su	17 Su D7	17 Ke	17 Pe	17 Ma Vk 25
18 Pe	18 Ma Vk 08	18 Ma Vk 12 T7	18 To	18 La	18 Ti
19 La	19 Ti Exam	19 Ti L8	19 Pe Pitkäperjantai	19 Su Kaatuneiden muistopäivä	19 Ke
20 Su D1	20 Ke	20 Ke Kevätpäiväntasaus	20 La	20 Ma Vk 21	20 To
21 Ma Vk 04 T1	21 To week	21 To Q8	21 Su Pääsiäispäivä	21 Ti	21 Pe Kesäpäivänseisaus
22 Ti L2	22 Pe	22 Pe	22 Ma 2. pääsiäispäivä	22 Ke	22 La Juhannus
23 Ke	23 La	23 La	23 Ti	23 To	23 Su
24 To Q2	24 Su D5	24 Su D8	24 Ke	24 Pe	24 Ma Vk 26
25 Pe	25 Ma Vk 09 T5	25 Ma Vk 13 T8	25 To	25 La	25 Ti
26 La	26 Ti L6	26 Ti L9	26 Pe	26 Su	26 Ke
27 Su D2	27 Ke	27 Ke	27 La	27 Ma Vk 22	27 To
28 Ma Vk 05 T2	28 To Q6	28 To Q9	28 Su	28 Ti	28 Pe
29 Ti L3		29 Pe	29 Ma Vk 18	29 Ke	29 La
30 Ke		30 La	30 Ti	30 To Helatorstai	30 Su
31 To Q3		31 Su Kesäaika alkaa D9		31 Pe	

L = Lecture; hall T5, Tue 12–14

Q = Q & A session; hall T5, Thu 12–14

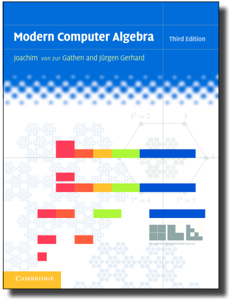
D = Problem set deadline; Sun 20:00

T = Tutorial (model solutions); hall T6, Mon 16–18

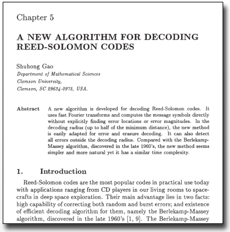
Recap of last week

- Fast **batch evaluation** and **interpolation** of polynomials
- Reduction to fast quotient and remainder
  —divide-and-conquer recursive remaindering along a **subproduct tree**
- **Secret sharing** by randomization

Goal: Near-linear-time toolbox for univariate polynomials



- Multiplication
- Division (quotient and remainder)
- Batch evaluation
- Interpolation
- Extended Euclidean algorithm (gcd) (this week)
- Interpolation from partly erroneous data (this week)



Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-0975, USA.

Abstract A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

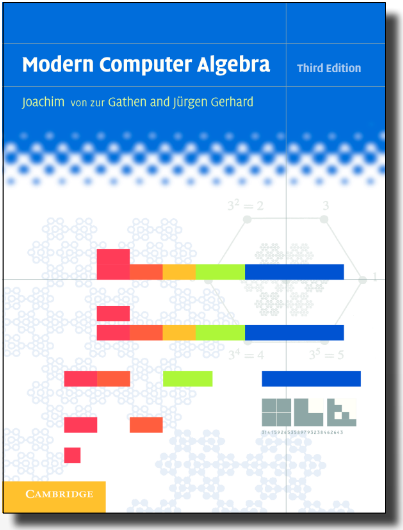
Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Further motivation for this week

- After this week we have completed our work on the near-linear time toolbox for univariate polynomials
- This week is also our first encounter with **uncertainty** in computation
- This week we learn how to cope with uncertainty in the form of **errors in data** by using **error-correcting codes**
- Next week look at **errors in computation** ...

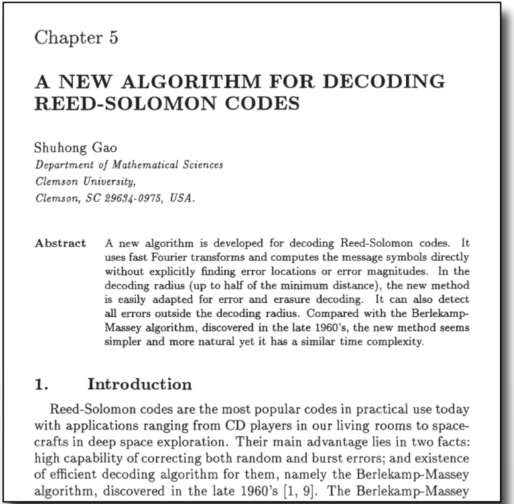
Fast extended Euclidean algorithm (for polynomials)

(von zur Gathen and Gerhard [11],
Section 11.1)



Fast interpolation from partly erroneous data

(Gao [10])



Chapter 5

A NEW ALGORITHM FOR DECODING REED-SOLOMON CODES

Shuhong Gao
Department of Mathematical Sciences
Clemson University,
Clemson, SC 29634-0975, USA.

Abstract A new algorithm is developed for decoding Reed-Solomon codes. It uses fast Fourier transforms and computes the message symbols directly without explicitly finding error locations or error magnitudes. In the decoding radius (up to half of the minimum distance), the new method is easily adapted for error and erasure decoding. It can also detect all errors outside the decoding radius. Compared with the Berlekamp-Massey algorithm, discovered in the late 1960's, the new method seems simpler and more natural yet it has a similar time complexity.

1. Introduction

Reed-Solomon codes are the most popular codes in practical use today with applications ranging from CD players in our living rooms to spacecrafts in deep space exploration. Their main advantage lies in two facts: high capability of correcting both random and burst errors; and existence of efficient decoding algorithm for them, namely the Berlekamp-Massey algorithm, discovered in the late 1960's [1, 9]. The Berlekamp-Massey

Fast extended Euclidean algorithm (for integers)

(Möller [20])

MATHEMATICS OF COMPUTATION
Volume 77, Number 261, January 2008, Pages 589–607
S 0025-5718(07)02017-0
Article electronically published on September 12, 2007

ON SCHÖNHAGE'S ALGORITHM AND SUBQUADRATIC INTEGER GCD COMPUTATION

NIELS MÖLLER

ABSTRACT. We describe a new subquadratic left-to-right GCD algorithm, inspired by Schönhage's algorithm for reduction of binary quadratic forms, and compare it to the first subquadratic GCD algorithm discovered by Knuth and Schönhage, and to the binary recursive GCD algorithm of Stehlé and Zimmermann. The new GCD algorithm runs slightly faster than earlier algorithms, and it is much simpler to implement. The key idea is to use a stop condition for HGCD that is based not on the size of the remainders, but on the size of the next difference. This subtle change is sufficient to eliminate the back-up steps that are necessary in all previous subquadratic left-to-right GCD algorithms. The subquadratic GCD algorithms all have the same asymptotic running time, $O(n(\log n)^2 \log\log n)$.

1. INTRODUCTION

In this paper, we describe four subquadratic GCD algorithms: Schönhage's algorithm from 1971, Stehlé's and Zimmermann's binary recursive GCD, a hitherto unpublished GCD algorithm discovered by Schönhage in 1987, and a novel GCD algorithm that uses similar ideas in a HGCD framework. The algorithms are compared with respect to running time and implementation complexity. The new algorithm is slightly faster than all the earlier algorithms, and much simpler to implement.

The paper is organized as follows: First we review the development of integer GCD algorithms in recent years. Section 2 describes the general structure and flavor of the subquadratic GCD algorithms, the idea of using a half-GCD function, and the resulting asymptotic running time. In Section 3, we briefly describe one variant of Schönhage's 1971 algorithm, and in Section 4, we describe the binary recursive GCD algorithm. The objective of these two sections is to provide sufficient details so that the new algorithm can be compared to earlier algorithms; we define the corresponding half-GCD functions, but we don't provide correctness proofs or detailed analysis.

Section 5 describes a GCD algorithm modeled on Schönhage's algorithm for reduction of binary quadratic forms [8], and in Section 6, this algorithm is reorganized into half-GCD form, resulting in a novel GCD algorithm. Section 7 describes the implementation of the different GCD algorithms, their running times, and code complexity.

Received by the editor November 19, 2005.
2000 *Mathematics Subject Classification.* Primary 11A05, 11Y16.

Key content for Lecture 5

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm for polynomials by **divide and conquer**
  - The two polynomial operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)
- Coping with **errors in data** using **error-correcting codes**
- A family of error-correcting codes (**Reed–Solomon codes**) based on evaluation–interpolation duality for univariate polynomials
  - Key observation: low-degree polynomials have few roots (exercise)
  - Fast **encoding** and **decoding** of Reed–Solomon codes via the fast univariate polynomial toolkit and **Gao's** (2003) **decoder**

Extended Euclidean algorithm (for polynomials)

- Let F be a field and let $f, g \in F[x]$ with $\deg f \geq \deg g \geq 0$
- Traditional extended Euclidean algorithm:
  1. $r_0 \leftarrow f$, $s_0 \leftarrow 1$, $t_0 \leftarrow 0$,
     $r_1 \leftarrow g$, $s_1 \leftarrow 0$, $t_1 \leftarrow 1$
  2. $i \leftarrow 1$,
     while $r_i \neq 0$ **do**
     $q_i \leftarrow r_{i-1} \operatorname{quo} r_i$
     $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
     $s_{i+1} \leftarrow s_{i-1} - q_i s_i$
     $t_{i+1} \leftarrow t_{i-1} - q_i t_i$
     $i \leftarrow i + 1$
  3. $\ell \leftarrow i - 1$
     return ℓ, r_i, s_i, t_i for $i = 0, 1, \ldots, \ell + 1$, and q_i for $i = 1, 2, \ldots, \ell$
- We want a faster algorithm

Example (over $\mathbb{Z}_2[x]$)

- Let $f = x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{Z}_2[x]$ and $g = x^5 + x^4 + 1 \in \mathbb{Z}_2[x]$
- We obtain

i	r_i	s_i	t_i	q_i
0	$x^5 + x^4 + x^3 + x^2 + x + 1$	1	0	
1	$x^5 + x^4 + 1$	0	1	1
2	$x^3 + x^2 + x$	1	1	$x^2 + 1$
3	$x^2 + x + 1$	$x^2 + 1$	x^2	x
4	0	$x^3 + x + 1$	$x^3 + 1$	

- In particular $\ell = 3$ and $r_\ell = x^2 + x + 1$ is a greatest common divisor of $x^5 + x^4 + x^3 + x^2 + x + 1$ and $x^5 + x^4 + 1$

Terminology

- The sequence $q_1, q_2, \ldots, q_\ell$ is the **quotient sequence** produced by the algorithm
- The polynomial r_i is the **remainder** at iteration i
- The polynomials s_i and t_i are the **Bézout coefficients** at iteration i
- The Bézout coefficients satisfy $r_i = s_i r_0 + t_i r_1$

Desiderata for a fast algorithm

- Let F be a field and let $f, g \in F[x]$ with $d \geq \deg f \geq \deg g \geq 0$
- Desired output:
  The quotients $q_1, q_2, \ldots, q_h$ and two consecutive rows r_h, s_h, t_h and $r_{h+1}, s_{h+1}, t_{h+1}$ for a choice of $h = 1, 2, \ldots, \ell$
- Using $O(M(d) \log d)$ operations in F

The degree sequences m_i and n_i

- It will be convenient to work with the following two sequences
- For $i = 1, 2, \ldots, \ell + 1$ let
$$m_i = \deg q_i$$
where, for convenience, we let $m_{\ell+1} = \infty$
- For $i = 0, 1, \ldots, \ell + 1$, let
$$n_i = \deg r_i$$
recalling that $n_{\ell+1} = \deg 0 = -\infty$
- By assumption, we have $\deg r_0 \geq \deg r_1 \geq 0$
- Since we have $r_{i+1} = r_{i-1} - q_i r_i$ and $\deg r_i > \deg r_{i+1}$ for all $i = 1, 2, \ldots, \ell$, it follows that
$$n_{i-1} = n_i + m_i$$

Example (over $\mathbb{Z}_2[x]$)

- Let $f = x^5 + x^4 + x^3 + x^2 + x + 1 \in \mathbb{Z}_2[x]$ and $g = x^5 + x^4 + 1 \in \mathbb{Z}_2[x]$
- We obtain

i	r_i	s_i	t_i	q_i	m_i	n_i
0	$x^5 + x^4 + x^3 + x^2 + x + 1$	1	0			5
1	$x^5 + x^4 + 1$	0	1	1	0	5
2	$x^3 + x^2 + x$	1	1	$x^2 + 1$	2	3
3	$x^2 + x + 1$	$x^2 + 1$	x^2	x	1	2
4	0	$x^3 + x + 1$	$x^3 + 1$		∞	$-\infty$

- In particular $\ell = 3$ and $r_\ell = x^2 + x + 1$ is a greatest common divisor of $x^5 + x^4 + x^3 + x^2 + x + 1$ and $x^5 + x^4 + 1$

The halting threshold $h = h(k)$

- Given a threshold parameter $k = 0, 1, \ldots, n_0$ as input, we want the algorithm to halt at iteration $h = h(k)$ determined by

$$m_1 + m_2 + \ldots + m_h \leq k$$

  and

$$m_1 + m_2 + \ldots + m_h + m_{h+1} > k$$

- In particular, we observe that $0 \leq h \leq \ell$

The halting threshold $h = h(k)$

- Equivalently, since $n_i = n_{i-1} - m_i$ for $i = 1, 2, \ldots, \ell + 1$, we have

  $$n_h \geq n_0 - k$$

  and

  $$n_{h+1} < n_0 - k$$

- That is, the algorithm halts at the unique iteration $h = 0, 1, \ldots, \ell$ when the degree of r_{h+1} for the first time decreases below $n_0 - k$

Truncating a polynomial

- Let

$$f = \varphi_n x^n + \varphi_{n-1} x^{n-1} + \ldots + \varphi_1 x + \varphi_0 \in F[x]$$

with **leading coefficient** $\operatorname{lc} f = \varphi_n \neq 0$

- For $k \in \mathbb{Z}$, define the **truncated polynomial**

$$f \upharpoonright k = \varphi_n x^k + \varphi_{n-1} x^{k-1} + \ldots + \varphi_{n-k+1} x + \varphi_{n-k} \in F[x]$$

where we set $\varphi_i = 0$ for $i < 0$ as necessary

- For $k \geq 0$ we have that $f \upharpoonright k$ is a polynomial of degree k whose coefficients are the $k + 1$ highest coefficients of f
- For $k < 0$ we have $f \upharpoonright k = 0$
- For all $i = 0, 1, \ldots$ we have $(fx^i) \upharpoonright k = f \upharpoonright k$

Example: Truncating a polynomial

- Let us work with the polynomial

$$f = 2 + 9x + 10x^2 + 4x^3 \in \mathbb{Z}_{11}[x]$$

- We obtain the truncations

$$\begin{aligned}
&\vdots \\
f \restriction -2 &= 0 \\
f \restriction -1 &= 0 \\
f \restriction 0 &= 4 \\
f \restriction 1 &= 10 + 4x \\
f \restriction 2 &= 9 + 10x + 4x^2 \\
f \restriction 3 &= 2 + 9x + 10x^2 + 4x^3 \\
f \restriction 4 &= 2x + 9x^2 + 10x^3 + 4x^4 \\
f \restriction 5 &= 2x^2 + 9x^3 + 10x^4 + 4x^5 \\
&\vdots
\end{aligned}$$

Coinciding pairs of polynomials

- Let $f, g, \tilde{f}, \tilde{g} \in F[x] \setminus \{0\}$ with $\deg f \geq \deg g$ and $\deg \tilde{f} \geq \deg \tilde{g}$
- For $k \in \mathbb{Z}$, we say that (f, g) and $(\tilde{f}, \tilde{g})$ **coincide up to** k and write $(f, g) \equiv_k (\tilde{f}, \tilde{g})$ if

$$f \upharpoonright k = \tilde{f} \upharpoonright k$$

$$g \upharpoonright (k - (\deg f - \deg g)) = \tilde{g} \upharpoonright (k - (\deg \tilde{f} - \deg \tilde{g}))$$

- Remark:
  If $(f, g) \equiv_k (\tilde{f}, \tilde{g})$ and $k \geq \deg f - \deg g$, then $\deg f - \deg g = \deg \tilde{f} - \deg \tilde{g}$

Example: Coinciding pairs of polynomials

- The pairs
$$f = 7 + 2x + x^2 + x^3 + 10x^4 + 7x^5 + x^6 + 5x^7 + 9x^8 + 5x^9 + 7x^{10} \in \mathbb{Z}_{11}[x]$$
$$g = 3 + 7x + 4x^2 + 2x^3 + 2x^4 + 6x^5 + 3x^6 + 2x^7 + 4x^8 \in \mathbb{Z}_{11}[x]$$
and
$$\tilde{f} = 1 + 5x + 9x^2 + 5x^3 + 7x^4 \in \mathbb{Z}_{11}[x]$$
$$\tilde{g} = 3 + 2x + 4x^2 \in \mathbb{Z}_{11}[x]$$
coincide up to 4
- Indeed, we have $\deg f = 10$, $\deg g = 8$, $\deg \tilde{f} = 4$, and $\deg \tilde{g} = 2$, with
$$f \upharpoonright 4 = \tilde{f} \upharpoonright 4 = 1 + 5x + 9x^2 + 5x^3 + 7x^4$$
$$g \upharpoonright 2 = \tilde{g} \upharpoonright 2 = 3 + 2x + 4x^2$$

Quotients of coinciding pairs of polynomials

- The following lemma enables us to design a divide-and-conquer extended Euclidean algorithm by truncating the operands to division

Lemma 8 (Sufficiently coinciding pairs of polynomials have identical quotients)

Suppose that $(f, g) \equiv_{2k} (\tilde{f}, \tilde{g})$ *for* $k \in \mathbb{Z}$ *with* $k \geq \deg f - \deg g \geq 0$. *Define* $q, r, \tilde{q}, \tilde{r} \in F[x]$ *by division with quotients and remainders as follows*

$$\begin{aligned} f &= qg + r, \qquad & \deg r &< \deg g\,, \\ \tilde{f} &= \tilde{q}\tilde{g} + \tilde{r}, \qquad & \deg \tilde{r} &< \deg \tilde{g}\,. \end{aligned}$$

Then, $q = \tilde{q}$ *and at least one of the following holds* $(g, r) \equiv_{2(k-\deg q)} (\tilde{g}, \tilde{r})$ *or* $r = 0$ *or* $k - \deg q < \deg g - \deg r$.

Proof.

Exercise □

Example: Quotient of coinciding pairs of polynomials

- The pairs

$$f = 7 + 2x + x^2 + x^3 + 10x^4 + 7x^5 + x^6 + 5x^7 + 9x^8 + 5x^9 + 7x^{10} \in \mathbb{Z}_{11}[x]$$
$$g = 3 + 7x + 4x^2 + 2x^3 + 2x^4 + 6x^5 + 3x^6 + 2x^7 + 4x^8 \in \mathbb{Z}_{11}[x]$$

  and

$$\tilde{f} = 1 + 5x + 9x^2 + 5x^3 + 7x^4 \in \mathbb{Z}_{11}[x]$$
$$\tilde{g} = 3 + 2x + 4x^2 \in \mathbb{Z}_{11}[x]$$

  coincide up to 4, with $4 \geq \deg f - \deg g = 2$

- Accordingly (by Lemma 8), the quotients agree:

$$f \operatorname{quo} g = 9 + 10x + 10x^2$$
$$\tilde{f} \operatorname{quo} \tilde{g} = 9 + 10x + 10x^2$$

Quotient sequences of coinciding pairs of polynomials

- Now let us study what happens in the extended Euclidean algorithm if we execute it for two inputs, (r_0, r_1) and $(\tilde{r}_0, \tilde{r}_1)$, with $\deg r_0 \geq \deg r_1 \geq 0$ and $\deg \tilde{r}_0 \geq \deg \tilde{r}_1 \geq 0$:

$$
\begin{aligned}
r_0 &= q_1 r_1 + r_2, & \tilde{r}_0 &= \tilde{q}_1 \tilde{r}_1 + \tilde{r}_2 \\
r_1 &= q_2 r_2 + r_3, & \tilde{r}_1 &= \tilde{q}_2 \tilde{r}_2 + \tilde{r}_3 \\
&\vdots & &\vdots \\
r_{i-1} &= q_i r_i + r_{i+1}, & \tilde{r}_{i-1} &= \tilde{q}_i \tilde{r}_i + \tilde{r}_{i+1} \\
&\vdots & &\vdots \\
r_{\ell-1} &= q_\ell r_\ell, & \tilde{r}_{\tilde{\ell}-1} &= \tilde{q}_{\tilde{\ell}} \tilde{r}_{\tilde{\ell}}
\end{aligned}
$$

- In particular, our interest is on the case $(r_0, r_1) \equiv_{2k} (\tilde{r}_0, \tilde{r}_1)$ ...

Quotient sequences of coinciding pairs of polynomials

- We can now study the execution on two *coinciding* inputs (r_0, r_1) and $(\tilde{r}_0, \tilde{r}_1)$ with $\deg r_0 \geq \deg r_1 \geq 0$ and $\deg \tilde{r}_0 \geq \deg \tilde{r}_1 \geq 0$ as follows

Lemma 9 (Identical quotient sequences up to the halting threshold)

Let $k \in \mathbb{Z}$ with $(r_0, r_1) \equiv_{2k} (\tilde{r}_0, \tilde{r}_1)$. Then, $h(k) = \tilde{h}(k)$ with $q_i = \tilde{q}_i$ for all $i = 1, 2, \ldots, h(k)$.

Proof sketch.

By induction on i and using Lemma 8 for the induction step, the following holds for all $0 \leq i \leq h(k)$: we have $i \leq \tilde{h}(k)$, $q_i = \tilde{q}_i$, and at least one of the following holds: $i = h(k)$ or $(r_i, r_{i+1}) \equiv_{2(k-\sum_{j=1}^{i} m_j)} (\tilde{r}_i, \tilde{r}_{i+1})$. $\square$

Example: Quotient sequences of coinciding pairs

- Let us run the extended Euclidean algorithm for a pair of polynomials in $\mathbb{Z}_{11}[x]$:

i	q_i	r_i	s_i	t_i
0		$7 + x + 3x^2 + 5x^3 + 9x^4 + 10x^5 + 7x^6$	1	0
1	4	$4 + 10x + 7x^2 + 4x^3 + 7x^4 + 4x^5 + 10x^6$	0	1
2	$4 + 2x$	$2 + 5x + 8x^2 + 3x^4 + 5x^5$	1	7
3	$4 + 10x$	$7 + 8x + 9x^2 + 10x^3 + 6x^4$	$7 + 9x$	$6 + 8x$
4	$2 + 3x$	$7 + 2x + 2x^2 + 2x^3$	$6 + 4x + 9x^2$	$5 + 7x + 8x^2$
5	$10 + 9x$	$4 + 5x + 10x^2$	$6 + 5x + 3x^2 + 6x^3$	$7 + x + 7x^2 + 9x^3$
6	$4 + 8x$	$4x$	$1 + 10x + x^3 + x^4$	$1 + 6x^2 + x^3 + 7x^4$
7	x	4	$2 + x + 2x^3 + 10x^4 + 3x^5$	$3 + 4x + 5x^2 + x^3 + 8x^4 + 10x^5$
8		0	$1 + 8x + 10x^2 + x^3 + 10x^4 + x^5 + 8x^6$	$1 + 8x + 2x^2 + 7x^3 + 6x^4 + 3x^5 + x^6$

- Here is a run on a pair that coincides with the first pair up to length $2k = 4$:

i	q_i	r_i	s_i	t_i
0		$3 + 5x + 9x^2 + 10x^3 + 7x^4$	1	0
1	4	$7 + 4x + 7x^2 + 4x^3 + 10x^4$	0	1
2	$4 + 2x$	$8 + 3x^2 + 5x^3$	1	7
3	$4 + 10x$	$8 + 10x + 6x^2$	$7 + 9x$	$6 + 8x$
4	$6x$	$9 + x$	$6 + 4x + 9x^2$	$5 + 7x + 8x^2$
5	$8 + 7x$	8	$7 + 6x + 9x^2 + x^3$	$6 + 2x^2 + 7x^3$
6		0	$5 + 6x + 5x^2 + 6x^3 + 4x^4$	$1 + 9x + 3x^2 + 7x^3 + 6x^4$

- Observe that the quotient sequences agree up to total degree $\deg q_1 + \deg q_2 + \ldots + \deg q_{h(k)} \leq k$ with $h(k) = 3$

A divide-and-conquer extended Euclidean algorihtm

- We now use Lemma 9 to design a fast divide-and-conquer version of the extended Euclidean algorihtm
- For a given input $(r_0, r_1) \in F[x]^2$ with $\deg r_0 \geq \deg r_1 \geq 0$ and halting parameter $k \geq 0$, the key idea is to truncate the input using the "$\upharpoonright$"-operator and build the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$ using two recursive calls with halting parameter at most $\lfloor k/2 \rfloor$ each
- That is, the idea essentially to use the first recursive call to recover $q_1, q_2, \ldots, q_{h(\lfloor k/2 \rfloor)}$, then compute (as needed) the next quotient $q_{h(\lfloor k/2 \rfloor)+1}$ explicitly, and then make a second recursive call (as needed) to recover the rest of the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$
- With careful implementation, this leads to an algorithm that runs in $O(M(k) \log k)$ operations in F
- Before describing the algorithm in detail, let us recall some further terminology ...

Invariants of the extended Euclidean algorithm

- Recall the matrices

$$R_0 = \begin{bmatrix} s_0 & t_0 \\ s_1 & t_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, \ell$$

  and $R_i = Q_i Q_{i-1} \cdots Q_1 R_0 \in F[x]^{2\times 2}$ for $i = 0, 1, \ldots, \ell$ from the analysis of the traditional extended Euclidean algorithm in Problem Set 1

- We recall that for all $i = 0, 1, \ldots, \ell$ we have $R_i = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$ and $R_i \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}$

- Our algorithm design will be such that on input (r_0, r_1) and k it produces as output (i) the value $h(k)$, (ii) the quotient sequence $q_1, q_2, \ldots, q_{h(k)}$, and (iii) the matrix $R_{h(k)}$ ...

Truncating inputs to the extended Euclidean algorithm

- Let us write $h(k), q_1, q_2, \ldots, q_{h(k)}, R_{h(k)} \leftarrow \mathrm{extgcd}(k, r_0, r_1)$ to indicate that the algorithm produces the output $h(k), q_1, q_2, \ldots, q_{h(k)}, R_{h(k)}$ on input k, r_0, r_1 with $\deg r_0 \geq \deg r_1 \geq 0$
- Lemma 9 now implies that we have

$$\mathrm{extgcd}(k, r_0, r_1) = \mathrm{extgcd}\Big(k, r_0 \upharpoonright 2k, r_1 \upharpoonright (2k - (\deg r_0 - \deg r_1))\Big) \tag{30}$$

- In particular, we can assemble the output recursively so that the input polynomials to each recursive call are truncated in degree to the minimum enabled by (30)
- We are now ready for the detailed pseudocode of the algorithm ...

A divide-and-conquer extended Euclidean algorithm I

- Let F be a field and let $k \in \mathbb{Z}$ and $r_0, r_1 \in F[x]$ with $\deg r_0 \geq \deg r_1$ and $r_0 \neq 0$ be given as input

1. If $k < \deg r_0 - \deg r_1$ holds, then return with output $h(k) \leftarrow 0$ and $R_{h(k)} \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
2. If $k = 0$ and $\deg r_0 = \deg r_1$ hold, then return with output $h(k) \leftarrow 1$, $q_1 = \frac{\operatorname{lc} r_0}{\operatorname{lc} r_1}$, and $R_{h(k)} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -\frac{\operatorname{lc} r_0}{\operatorname{lc} r_1} \end{bmatrix}$
3. Set $k_1 \leftarrow \lfloor k/2 \rfloor$
4. Make the first recursive call $h_1, q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}, R^{(1)} \leftarrow \operatorname{extgcd}\big(k_1, r_0 \upharpoonright 2k_1, r_1 \upharpoonright (2k_1 - (\deg r_0 - \deg r_1))\big)$
5. Compute the matrix-vector product $\begin{bmatrix} \tilde{r}_{h_1} \\ \tilde{r}_{h_1+1} \end{bmatrix} \leftarrow R^{(1)} \begin{bmatrix} r_0 \upharpoonright 2k \\ r_1 \upharpoonright (2k - (\deg r_0 - \deg r_1)) \end{bmatrix}$

A divide-and-conquer extended Euclidean algorithm II

6. If $\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1} > k$ holds, then return with output $h(k) \leftarrow h_1, q_1, q_2, \ldots, q_{h(k)} \leftarrow q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}$, and $R_{h(k)} \leftarrow R^{(1)}$
7. Compute the quotient $q_{h_1+1} \leftarrow \tilde{r}_{h_1} \operatorname{quo} \tilde{r}_{h_1+1}$ and the matrix $Q_{h_1+1} \leftarrow \begin{bmatrix} 0 & 1 \\ 1 & -q_{h_1+1} \end{bmatrix}$
8. Compute the remainder $\tilde{r}_{h_1+2} \leftarrow \tilde{r}_{h_1} - q_{h_1+1}\tilde{r}_{h_1+1}$
9. Set $k_2 \leftarrow k - (\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg q_{h_1+1})$
10. Make the second recursive call
$h_2, q_1^{(2)}, q_2^{(2)}, \ldots, q_{h_2}^{(2)}, R^{(2)} \leftarrow \operatorname{extgcd}\big(k_2, \tilde{r}_{h_1+1} \upharpoonright 2k_1, \tilde{r}_{h_1+2} \upharpoonright (2k_1 - (\deg \tilde{r}_{h_1+1} - \deg \tilde{r}_{h_1+2}))\big)$
11. Return with output $h(k) \leftarrow h_1 + 1 + h_2$,
$q_1, q_2, \ldots, q_{h(k)} \leftarrow q_1^{(1)}, q_2^{(1)}, \ldots, q_{h_1}^{(1)}, q_{h_1+1}, q_1^{(2)}, q_2^{(2)}, \ldots, q_{h_2}^{(2)}$, and
$R_{h(k)} \leftarrow R^{(2)} Q_{h_1+1} R^{(1)}$

Remarks and analysis

- Caveat: In Step 1 we may have $\deg r_1 = -\infty$ (that is, $r_1 = 0$) and in Step 6 we may have $\deg \tilde{r}_{h_1+1} = -\infty$ (that is, $\tilde{r}_{h_1+1} = 0$)
- After Step 1 it holds that $k \geq \deg r_0 - \deg r_1 \geq 0$, after Step 2 it holds that $k \geq 1$ and $\deg r_0 > \deg r_1 \geq 0$; thus, $0 \leq k_1 \leq k - 1$
- After Step 5 we have

$$\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} \leq k_1$$

  and, also recalling that $k_1 = \lfloor k/2 \rfloor$,

$$\deg q_1^{(1)} + \deg q_2^{(1)} + \ldots + \deg q_{h_1}^{(1)} + \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1} \geq k_1 + 1 \geq \lceil k/2 \rceil$$

- Assuming that $\tilde{r}_{h_1+1} \neq 0$, we have $\deg q_{h_1+1} = \deg \tilde{r}_{h_1} - \deg \tilde{r}_{h_1+1}$
- Thus, $k_2 \leq \lfloor k/2 \rfloor \leq k - 1$
- The algorithm runs in $T(k) \leq T(k_1) + T(k_2) + O(M(k)) \leq 2T(\lfloor k/2 \rfloor) + O(M(k))$ operations in F; that is, $T(k) = O(M(k) \log k)$ operations in F

Key content for Lecture 5 (recalled)

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm for polynomials by **divide and conquer**
  - The two polynomial operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)
- Coping with **errors in data** using **error-correcting codes**
- A family of error-correcting codes (**Reed–Solomon codes**) based on evaluation–interpolation duality for univariate polynomials
  - Key observation: low-degree polynomials have few roots (exercise)
  - Fast **encoding** and **decoding** of Reed–Solomon codes via the fast univariate polynomial toolkit and **Gao's** (2003) **decoder**

Number of roots

- Let F be a field
- A **root** of a polynomial $f \in F[x]$ is an element $\xi \in F$ with $f(\xi) = 0$

Theorem 10 (Number of roots)

A nonzero polynomial $f \in F[x]$ of degree at most d has at most d distinct roots.

Proof.

Exercise $\square$

Two distinct polynomials mostly disagree

- Let F be a field
- Let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ be a vector of e distinct elements of F
- Associate with $f \in F[x]$ the vector of evaluations

$$f(\Xi) = (f(\xi_1), f(\xi_2), \ldots, f(\xi_e)) \in F^e$$

Lemma 11 (Bounded agreement of low-degree polynomials)

Let $f_0, f_1 \in F[x]$ be distinct polynomials of degree at most d.
Then, $f_0(\Xi)$ and $f_1(\Xi)$ agree in at most d coordinates.

Proof.

The difference $f_0 - f_1 \neq 0$ is a polynomial of degree at most d and thus has at most d distinct roots $\square$

Reconstructibility from partly erroneous data

- Let $f \in F[x]$ be a polynomial of degree at most d
- Let $e \geq d+1$ and let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ consist of distinct elements

Lemma 12 (Unique reconstructibility)

Suppose that the vectors $\Gamma \in F^e$ and $f(\Xi)$ disagree in at most $(e-d-1)/2$ coordinates. Then, Γ uniquely identifies f

Proof.

Let $f_0, f_1 \in F[x]$ be two polynomials of degree at most d such that $f_0(\Xi)$ and $f_1(\Xi)$ each disagree with Γ in at most $(e-d-1)/2$ coordinates. In total there are e coordinates, so $f_0(\Xi)$ and $f_1(\Xi)$ and Γ must thus all agree in at least $e - 2(e-d-1)/2 = d+1$ coordinates. By Lemma 11 thus $f_0 = f_1$. □

(Furthermore, we can, very inefficiently, recover f from Γ by considering in turn each vector $\tilde{\Gamma} \in F^e$ that disagrees with Γ in at most $(e-d-1)/2$ coordinates: for each such $\tilde{\Gamma}$, interpolate f from $f(\Xi) = \tilde{\Gamma}$, and stop when f has degree at most d.)

Reed–Solomon codes

- Suppose we want to protect a sequence $\Phi = (\varphi_0, \varphi_1, \ldots, \varphi_d) \in F^{d+1}$ of elements of a field F against errors
- We may represent Φ as a polynomial $f = \varphi_0 + \varphi_1 x + \ldots + \varphi_d x^d \in F[x]$ of degree at most d
- Let $e \geq d + 1$ and let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ consist of distinct elements
- Let us use $\Psi = f(\Xi) \in F^e$ as the encoded representation of Φ
- Suppose that $\hat{\Psi}$ disagrees with Ψ in at most $(e - d - 1)/2$ coordinates. Then, Lemma 12 implies that we can recover Φ from $\hat{\Psi}$
- That is, $\hat{\Psi}$ may have up to $\lfloor (e - d - 1)/2 \rfloor$ errors and we can still recover Φ
- Encoding can be done in near-linear-time by fast batch evaluation ...
- ... but how efficiently can we decode in the presence of errors?

Example: Encoding

- Let us work with $e = 8$, $d = 3$, $F = \mathbb{Z}_{11}$, and the evaluation points $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) = (0, 1, 2, 3, 4, 5, 6, 7) \in \mathbb{Z}_{11}^e$
- Suppose we want to protect the data vector $\Phi = (5, 3, 1, 9) \in \mathbb{Z}_{11}^{d+1}$
- We view Φ as the degree-at-most-d polynomial $f = 5 + 3x + x^2 + 9x^3 \in \mathbb{Z}_{11}[x]$
- The encoded representation of Φ is

$$\Psi = f(\Xi) = (f(\xi_1), f(\xi_2), \ldots, f(\xi_e)) = (5, 7, 10, 2, 4, 4, 1, 5) \in \mathbb{Z}_{11}^e$$

Gao's (2003) decoder for Reed–Solomon codes

- Let $f \in F[x]$ be a polynomial of degree at most d
- Let $e \geq d+1$ and let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ consist of distinct elements
- Suppose that the vectors $\Gamma \in F^e$ and $f(\Xi)$ disagree in at most $(e-d-1)/2$ coordinates. Then, Γ uniquely identifies f (Lemma 12)
- Moreover, given Ξ, Γ, d as input, f can be computed in $O(M(e) \log e)$ operations in F (Gao [10])

Gao's decoding algorithm

- Let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ consisting of distinct elements, $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_e) \in F^e$, and $d \in \mathbb{Z}_{\geq 0}$ with $d + 1 \leq e$ be given as input
- Gao's algorithm [10] proceeds as follows:
  1. Using a subproduct tree, construct the polynomial $g_0 = \prod_{i=1}^{e}(x - \xi_i)$
  2. Interpolate the unique polynomial $g_1 \in F[x]$ of degree at most $e - 1$ that satisfies $g_1(\xi_i) = \gamma_i$ for all $i = 1, 2, \ldots, e$
  3. Apply the extended Euclidean algorithm to g_0 and g_1 to produce the consecutive remainders g_h, g_{h+1} with $\deg g_h \geq D$, and $\deg g_{h+1} < D$ for $D = (e + d + 1)/2$. Let $s_{h+1}, t_{h+1} \in F[x]$ be the associated Bézout coefficients with $g_{h+1} = s_{h+1}g_0 + t_{h+1}g_1$
  4. Divide g_{h+1} by t_{h+1} to obtain the quotient $f_1 \in F[x]$ and the remainder $r \in F[x]$ with $g_{h+1} = t_{h+1}f_1 + r$ and $\deg r < \deg t_{h+1}$
  5. Output f_1 as the result of interpolation if both $\deg f_1 \leq d$ and $r = 0$; otherwise assert decoding failure
- It is immediate that the algorithm runs in $O(M(e) \log e)$ operations in F

Example: Decoding I

- Let us work with $e = 8$, $d = 3$, $F = \mathbb{Z}_{11}$, and the evaluation points $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) = (0, 1, 2, 3, 4, 5, 6, 7) \in \mathbb{Z}_{11}^e$
- Suppose we have the vector $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_e) = (5, 7, 1, 2, 9, 4, 1, 5) \in \mathbb{Z}_{11}^e$
- First, we construct the polynomial

$$g_0 = \prod_{i=1}^{e} (x - \xi_i) = 9x + 2x^3 + 4x^4 + 9x^5 + 3x^6 + 5x^7 + x^8$$

- Then, we interpolate the polynomial

$$g_1 = 5 + 7x + 5x^2 + 2x^3 + 10x^4 + 9x^5 + 6x^6 + 7x^7$$

  that satisfies $g_1(\xi_i) = \gamma_i$ for all $i = 1, 2, \ldots, e$

Example: Decoding II

- Next we apply the extended Euclidean algorithm to g_0 and g_1 to produce the consecutive remainders g_h, g_{h+1} with $\deg g_h \geq D$, and $\deg g_{h+1} < D$ for $D = (e + d + 1)/2 = 6$ ...
- For convenience, we display the entire output of the extended Euclidean algorithm (but omitting the first Bézout coefficient sequence):

i	q_i	g_i	t_i
0		$9x + 2x^3 + 4x^4 + 9x^5 + 3x^6 + 5x^7 + x^8$	0
1	$8 + 8x$	$5 + 7x + 5x^2 + 2x^3 + 10x^4 + 9x^5 + 6x^6 + 7x^7$	1
2	$7 + 10x$	$4 + x + 3x^2 + x^3 + 7x^4 + 4x^6$	$3 + 3x$
3	$3 + 3x$	$10 + 4x + 7x^2 + 9x^3 + 6x^4 + 5x^5$	$2 + 4x + 3x^2$
4	$6 + 10x$	$7 + 3x + 3x^2 + 8x^3 + 6x^4$	$8 + 7x + x^2 + 2x^3$
5	$10 + 9x$	$1 + 4x + 3x^2 + 8x^3$	$9 + 3x + 4x^2 + 2x^4$
6	$4 + 10x$	$8 + 9x + 3x^2$	$6 + 6x + 10x^3 + 2x^4 + 4x^5$
7	$5 + 4x$	$2 + 9x$	$7 + 7x + 10x^2 + 4x^3 + 4x^4 + 8x^5 + 4x^6$
8	$10 + x$	9	$4 + 9x + 10x^2 + 5x^3 + 10x^4 + 3x^5 + 3x^6 + 6x^7$
9		0	$x + 10x^3 + 9x^4 + x^5 + 4x^6 + 3x^7 + 5x^8$

- (In a fast implementation we would of course use the divide-and-conquer extended Euclidean algoritm and would not produce the entire sequence of remainders g_i)

Example: Decoding III

- From the extended Euclidean algorithm we obtain that $h = 2$ with

$$g_{h+1} = 10 + 4x + 7x^2 + 9x^3 + 6x^4 + 5x^5$$
$$t_{h+1} = 2 + 4x + 3x^2$$

- Dividing g_{h+1} by t_{h+1} we obtain the quotient

$$f_1 = 5 + 3x + x^2 + 9x^3$$

  and the remainder $r = 0$
- In particular, the decoding is successful, and the reconstructed data vector is $(5, 3, 1, 9) \in \mathbb{Z}_{11}^{d+1}$
- Re-encoding the reconstructed vector as appropriate, we can also observe that the vector Γ has two errors, namely $f(\xi_3) = 10 \neq \gamma_3 = 2$ and $f(\xi_5) = 4 \neq \gamma_5 = 9$

Correctness I

- First, suppose that the algorithm does not assert failure
- Then, $f_1 = g_{h+1}/t_{h+1}$ has degree at most d
- Since $t_{h+1}f_1 = g_{h+1} = s_{h+1}g_0 + t_{h+1}g_1$, we have $s_{h+1}g_0 = t_{h+1}(f_1 - g_1)$ and hence for all $i = 1, 2, \ldots, e$ we have $t_{h+1}(\xi_i) = 0$ or $f_1(\xi_i) = g_1(\xi_i) = \gamma_i$
- Since g_{h+1} is the first remainder with $\deg g_{h+1} < D$ and $\deg g_0 = e$, by the structure of the Bézout coefficients we have $\deg t_{h+1} \leq e - D = (e - d - 1)/2$
- Indeed, from the definition of Bézout coefficients we have $\deg s_{h+1}, \deg t_{h+1} \leq \sum_{i=1}^{h} \deg q_i = \deg g_0 - \deg g_h \leq e - D$ since $\deg g_i + \deg q_i = \deg g_{i-1}$ and $\deg g_h \geq D$
- Since t_{h+1} has at most $\deg t_{h+1}$ roots, we have $f_1(\xi_i) \neq \gamma_i$ for at most $(e - d - 1)/2$ coordinates $i = 1, 2, \ldots, e$
- Thus, f_1 is a valid output for input Ξ, Γ, d

Correctness II

- Next, let $f \in F[x]$ be a polynomial of degree at most d, let $\Xi = (\xi_1, \xi_2, \ldots, \xi_e) \in F^e$ consist of distinct elements, and let $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_e) \in F^e$ be a vector that disagrees with $f(\Xi)$ in at most $(e - d - 1)/2$ coordinates for $d + 1 \leq e$
- By Lemma 12, we know that Γ uniquely determines f
- We show that Gao's algorithm outputs $f_1 = f$ on input Ξ, Γ, d
- Let $B = \{i \in \{1, 2, \ldots, e\} : f(\xi_i) \neq \gamma_i\}$ be the set of "bad" coordinates
- That is, B is the set of coordinates where Γ and $f(\Xi)$ disagree
- By assumption we have $|B| \leq (e - d - 1)/2$
- To understand the operation of the algorithm, let us split the polynomials g_0 and g_1 into parts based on B and $G = \{1, 2, \ldots, e\} \setminus B$ (the "bad" and "good" coordinates)

Correctness III

- Toward this end, let

$$q = \prod_{i \in G} (x - \xi_i) \in F[x], \qquad r_0 = \prod_{i \in B} (x - \xi_i) \in F[x]$$

- It is immediate that $g_0 = qr_0$
- Let $r_1 \in F[x]$ be the unique polynomial of degree at most $(e - d - 1)/2 - 1$ with $r_1(\xi_i) = q(\xi_i)^{-1}(\gamma_i - f(\xi_i)) \neq 0$ for all $i \in B$
- Thus, we have $g_1 = qr_1 + f$
- We have that $\gcd(r_0, r_1) = 1$ since no root of r_0 is a root of r_1 and r_0 factors into a product of degree 1 polynomials
- The following lemma will imply that the algorithm outputs $f_1 = f$; we postpone the proof and give it as Lemma 13

Correctness IV

- **Gao's Lemma.** (Lemma 13 below) Let $c, d, D \in \mathbb{Z}_{\geq 0}$ and let $q, r_0, r_1, f_0, f_1 \in F[x]$ with $\gcd(r_0, r_1) = 1$, $\deg q \geq D \geq c + d + 1$, and $\deg r_i \leq c$, $\deg f_i \leq d$ for $i = 0, 1$. Run the extended Euclidean algorithm on input $g_0 = qr_0 + f_0$ and $g_1 = qr_1 + f_1$ to obtain the remainders g_h and $g_{h+1} = s_{h+1}g_0 + t_{h+1}g_1$ for $s_{h+1}, t_{h+1} \in F[x]$ with $\deg g_h \geq D$ and $\deg g_{h+1} < D$. Then, $s_{h+1} = -\alpha r_1$ and $t_{h+1} = \alpha r_0$ for some $\alpha \in F \setminus \{0\}$
- Take $f_0 = 0, f_1 = f$, $c = |B|$ in the lemma and recall that we have $D = (e + d + 1)/2$
- Thus, $c \leq (e - d - 1)/2$, $\deg q = |G| = e - |B| \geq D \geq c + d + 1$, and the lemma applies to the polynomials $g_0 = qr_0$ and $g_1 = qr_1 + f$ constructed in the algorithm
- Let $g_{h+1}, s_{h+1}, t_{h+1}$ be the output of the lemma (also constructed by the algorithm)
- Because $f_0 = 0$ and $f_1 = f$, we have $g_{h+1} = -\alpha r_1 qr_0 + \alpha r_0(qr_1 + f) = t_{h+1}f$
- In particular, the algorithm outputs $f_1 = f = g_{h+1}/t_{h+1}$ $\square$

Preparation for Gao's Lemma

- Recall the matrices
$$R_0 = \begin{bmatrix} s_0 & t_0 \\ s_1 & t_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \qquad Q_i = \begin{bmatrix} 0 & 1 \\ 1 & -q_i \end{bmatrix} \quad \text{for } i = 1, 2, \ldots, \ell$$
and $R_i = Q_i Q_{i-1} \cdots Q_1 R_0 \in F[x]^{2\times 2}$ for $i = 0, 1, \ldots, \ell$ from the analysis of the traditional extended Euclidean algorithm in Problem Set 1
- We recall that for all $i = 0, 1, \ldots, \ell$ we have $R_i = \begin{bmatrix} s_i & t_i \\ s_{i+1} & t_{i+1} \end{bmatrix}$ and $R_i \begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = \begin{bmatrix} r_i \\ r_{i+1} \end{bmatrix}$
- Since $\det Q_i = -1$ we have $\det R_i = (-1)^i$ and thus $R_i^{-1} = (-1)^i \begin{bmatrix} t_{i+1} & -t_i \\ -s_{i+1} & s_i \end{bmatrix}$
- Since $r_{\ell+1} = 0$, we have $\begin{bmatrix} r_0 \\ r_1 \end{bmatrix} = R_\ell^{-1} \begin{bmatrix} r_\ell \\ 0 \end{bmatrix} = \begin{bmatrix} (-1)^\ell t_{\ell+1} r_\ell \\ (-1)^{\ell+1} s_{\ell+1} r_\ell \end{bmatrix}$
- We conclude that $s_{\ell+1} = (-1)^{\ell+1} r_1 / r_\ell$ and $t_{\ell+1} = (-1)^\ell r_0 / r_\ell$

Gao's Lemma

Lemma 13 (Gao [10])

Let $c, d, D \in \mathbb{Z}_{\geq 0}$ *and let* $q, r_0, r_1, f_0, f_1 \in F[x]$ *with* $\gcd(r_0, r_1) = 1$, $\deg q \geq D \geq c + d + 1$, *and* $\deg r_i \leq c$, $\deg f_i \leq d$ *for* $i = 0, 1$. *Run the extended Euclidean algorithm on input* $g_0 = qr_0 + f_0$ *and* $g_1 = qr_1 + f_1$ *to obtain the remainders* g_h *and* $g_{h+1} = s_{h+1}g_0 + t_{h+1}g_1$ *for* $s_{h+1}, t_{h+1} \in F[x]$ *with* $\deg g_h \geq D$ *and* $\deg g_{h+1} < D$. *Then,* $s_{h+1} = -\alpha r_1$ *and* $t_{h+1} = \alpha r_0$ *for some* $\alpha \in F \setminus \{0\}$

Proof of Gao's Lemma I

- Let $r_0, r_1, \ldots, r_\ell, r_{\ell+1}$ and $q_1, q_2, \ldots, q_\ell$ be the sequences of remainders and quotients in the extended Euclidean algorithm on input r_0, r_1
- Since $\gcd(r_0, r_1) = 1$, we have $r_\ell \in F \setminus \{0\}$ and $r_{\ell+1} = 0$
- Let $s_i, t_i \in F[x]$ for $i = 0, 1, \ldots, \ell + 1$ be the associated sequence of Bézout coefficients
- For all $i = 1, 2, \ldots, \ell$, we have

$$r_{i+1} = r_{i-1} - q_i r_i\,, \quad s_{i+1} = s_{i-1} - q_i s_i\,, \quad t_{i+1} = t_{i-1} - q_i t_i \tag{31}$$

- For all $i = 2, 3, \ldots, \ell + 1$ define $g_i = s_i g_0 + t_i g_1$
- From (31) it follows that $g_{i+1} = g_{i-1} - q_i g_i$ for all $i = 1, 2, \ldots, \ell$
- Let us show that $\deg g_i$ is a monotone decreasing sequence for $i = 1, 2, \ldots, \ell$

Proof of Gao's Lemma II

- We have $r_i = s_i r_0 + t_i r_1$ for all $i = 1, 2, \ldots, \ell + 1$. Furthermore, $\deg s_i \leq c$ and $\deg t_i \leq c$ for all $i = 1, 2, \ldots, \ell + 1$
- Since $g_0 = qr_0 + f_0$, $g_1 = qr_1 + f_1$, and $g_i = s_i g_0 + t_i g_1$, for all $i = 0, 1, \ldots, \ell$ we have $g_i = qr_i + s_i f_0 + t_i f_1$
- Since $\deg(s_i f_0 + t_i f_1) \leq c + d$ and $\deg q \geq D \geq c + d + 1$, we have $\deg g_i = \deg q + \deg r_i \geq D$ for all $i = 0, 1, \ldots, \ell$
- Since $\deg r_i$ is monotone decreasing for $i = 1, 2, \ldots, \ell$, we have that the same holds for $\deg g_i$
- Thus, we have that $g_0, g_1, \ldots, g_\ell$ and $q_1, q_2, \ldots, q_\ell$ form a prefix of the sequence of remainders and quotients in the extended Euclidean algorithm on input g_0, g_1
- Since $\deg r_\ell = 0$, we have $\deg g_\ell = \deg q \geq D$

Proof of Gao's Lemma III

- Since $s_{\ell+1} = (-1)^{\ell+1} r_1/r_\ell$ and $t_{\ell+1} = (-1)^{\ell} r_0/r_\ell$, we have
$$g_{\ell+1} = s_{\ell+1}g_0 + t_{\ell+1}g_1 = (-1)^{\ell}(-f_0 r_1 + f_1 r_0)/r_\ell$$
- Thus, $\deg g_{\ell+1} \le c + d < D$ and it follows that $g_{\ell+1} = g = sg_0 + tg_1$ with $\alpha = (-1)^{\ell}/r_\ell$, $s = -\alpha r_1$, and $t = \alpha r_0$ $\square$

Recap of Lecture 5

- **Extended Euclidean algorithm** for polynomials recalled and expanded
  - The **quotient sequence**, the **Bézout coefficients**, and the **halting threshold**
- Fast extended Euclidean algorithm by **divide and conquer**
  - The two operands **truncated** to a prefix of the highest-degree monomials determine the prefix of the quotient sequence (exercise)
- Coping with **errors in data** using **error-correcting codes**
- A family of error-correcting codes (**Reed–Solomon codes**) based on evaluation–interpolation duality for univariate polynomials
  - Key observation: low-degree polynomials have few roots (exercise)
  - Fast **encoding** and **decoding** of Reed–Solomon codes via the fast univariate polynomial toolkit and **Gao's** (2003) **decoder**

Learning objectives (1/2)

- Terminology and objectives of modern algorithmics, including elements of algebraic, online, and randomised algorithms
- Ways of coping with uncertainty in computation, including error-correction and proofs of correctness
- The art of solving a large problem by reduction to one or more smaller instances of the same or a related problem
- (Linear) independence, dependence, and their abstractions as enablers of efficient algorithms

Learning objectives (2/2)

- Making use of duality
  - Often a problem has a corresponding **dual** problem that is obtainable from the original (the **primal**) problem by means of an easy transformation
  - The primal and dual control each other, enabling an algorithm designer to use the interplay between the two representations
- Relaxation and tradeoffs between objectives and resources as design tools
  - Instead of computing the exact optimum solution at considerable cost, often a less costly but principled approximation suffices
  - Instead of the complete dual, often only a randomly chosen partial dual or other relaxation suffices to arrive at a solution with high probability