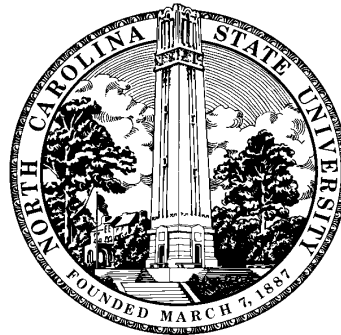
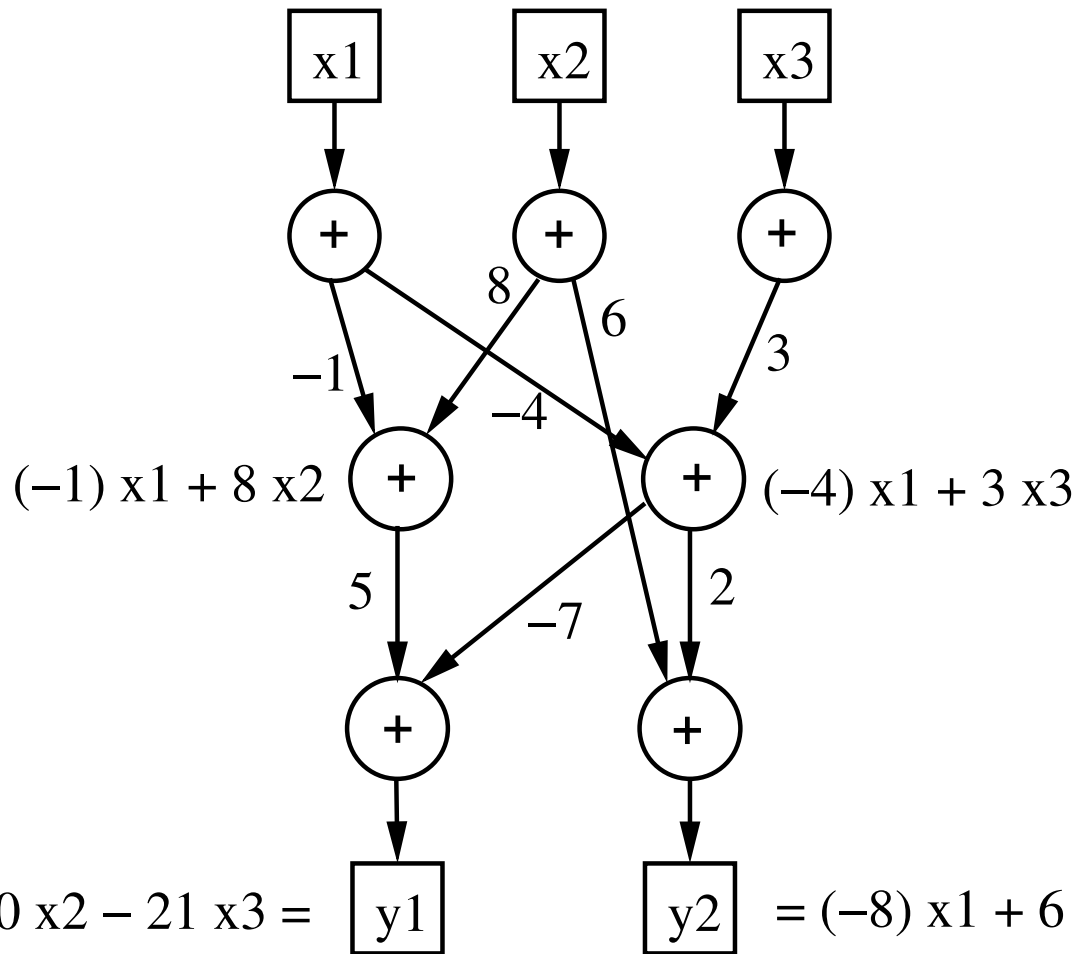


Tellegen's principle and the synthesis of algorithms

Erich Kaltofen
North Carolina State University
www.kaltofen.us

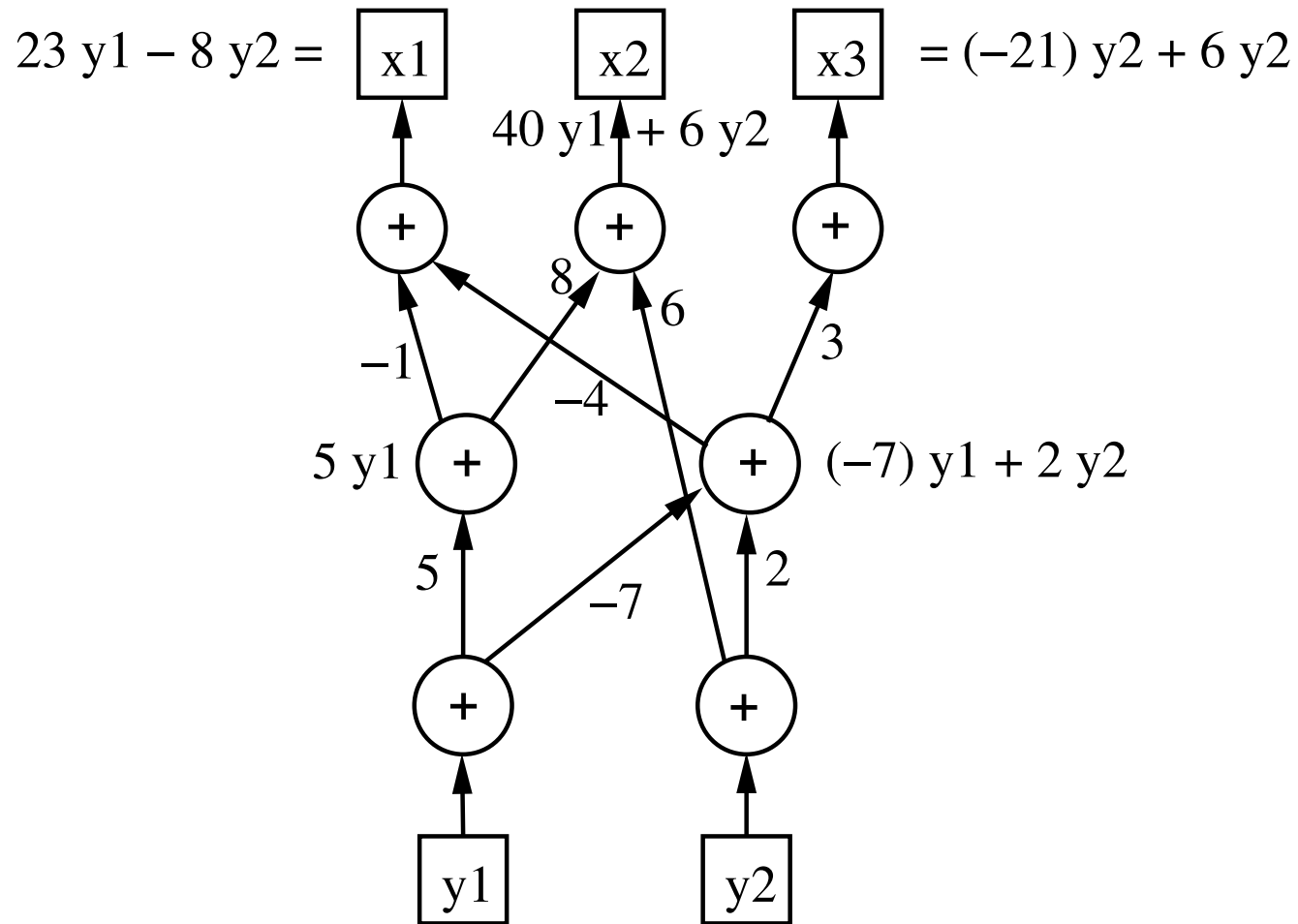


Tellegen's principle (Bordewijk's theorem)



$$\begin{bmatrix} 23 & 40 & -21 \\ -8 & 6 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Tellegen's principle (Bordewijk's theorem)



$$[y_1 \ y_2] \cdot \begin{bmatrix} 23 & 40 & -21 \\ -8 & 6 & 6 \end{bmatrix} = [x_1 \ x_2 \ x_3]$$

A first application: weighted power sums

Input: $x_1, \dots, x_n, y_1, \dots, y_n$ (weights)

Output: $b_i = x_1^i y_1 + \dots + x_n^i y_n$ for all $i = 0, 1, \dots, n - 1$

$$\begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \\ \vdots & & \vdots \\ x_1^{n-1} & \dots & x_n^{n-1} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

A first application: weighted power sums

Input: $x_1, \dots, x_n, y_1, \dots, y_n$ (weights)

Output: $b_i = x_1^i y_1 + \dots + x_n^i y_n$ for all $i = 0, 1, \dots, n-1$

$$\begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \\ \vdots & & \vdots \\ x_1^{n-1} & \dots & x_n^{n-1} \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

= transposed multipoint polynomial evaluation
(transposed Vandermonde-matrix times a vector)

$$\begin{bmatrix} c_0 & c_1 & \dots & c_{n-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_n \\ \vdots & & \vdots \\ x_1^{n-1} & \dots & x_n^{n-1} \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$

where $f(X) = c_0 + c_1 X + \dots + c_{n-1} X^{n-1}$.

Needed in our 1989 sparse polynomial multiplication algorithm
[see also J. van der Hoeven, Proc. ISSAC 2004]
and Shoup's polynomial factorization algorithm.

Direct solution: $V^{Tr} \cdot V = [\sum_k x_k^{i+j}]_{i,j} = H$ is Hankel, so

$$(V^{Tr} \cdot V)V^{-1}y = H(V^{-1}y) = b.$$

Uses interpolation $V^{-1}y$,
plain power sums via Newton's identities,
and polynomial multiplication.

Multipoint evaluation

Let $n = 2^m$:

Compute

$$\begin{aligned} 1: & (X - x_1)(X - x_2), (X - x_3)(X - x_4), \dots \\ 2: & (X - x_1) \cdots (X - x_4), (X - x_5) \cdots (X - x_8), \dots \\ & \vdots \\ m-1: & (X - x_1) \cdots (X - x_{n/2}), (X - x_{n/2+1}) \cdots (X - x_n) \end{aligned}$$

then

$$\begin{aligned} 1: & f(X) \bmod (X - x_1) \cdots (X - x_{n/2}), \\ & f(X) \bmod (X - x_{n/2+1}) \cdots (X - x_n) \\ & \vdots \\ m-2: & f(X) \bmod (X - x_1) \cdots (X - x_4), \\ & f(X) \bmod (X - x_5) \cdots (X - x_8), \dots \\ m-1: & f(X) \bmod (X - x_1)(X - x_2), f(X) \bmod (X - x_3)(X - x_4), \dots \\ m: & f(x_1) = f(X) \bmod (X - x_1), f(x_2) = f(X) \bmod (X - x_2), \dots \end{aligned}$$

Complexity is parameterized by polynomial multiplication/division algorithm:

$O(M(n) \log n)$ arithmetic operations

$M(n) = O(n^2)$ classical for small n

$M(n) = O(n^{1.59})$ [Karatsuba]

$M(n) = O(n(\log n)(\log \log n))$ [Cantor and Kaltofen]

$M(n) = O(n(\log n))$ with primitive roots

Complexity is parameterized by polynomial multiplication/division algorithm:

$O(M(n) \log n)$ arithmetic operations

$M(n) = O(n^2)$ classical for small n

$M(n) = O(n^{1.59})$ [Karatsuba]

$M(n) = O(n(\log n)(\log \log n))$ [Cantor and Kaltofen]

$M(n) = O(n(\log n))$ with primitive roots

Practically efficiently reverse all those algorithms [Bostan, Lecerf and Schost, Proc. ISSAC 2003]

Ostrowski, Wolin, and Borisow (1971) automatic differentiation (reverse mode)

Consider straight-line program

$$\begin{aligned}v_j &\leftarrow x_j, & 1 \leq j \leq n, \\v_i &\leftarrow v_{L_i} \circ_i v_{R_i}, & n+1 \leq i \leq n+s.\end{aligned}$$

Compute all $\partial_{x_j}(v_{n+s})$ by unravelling from the back:

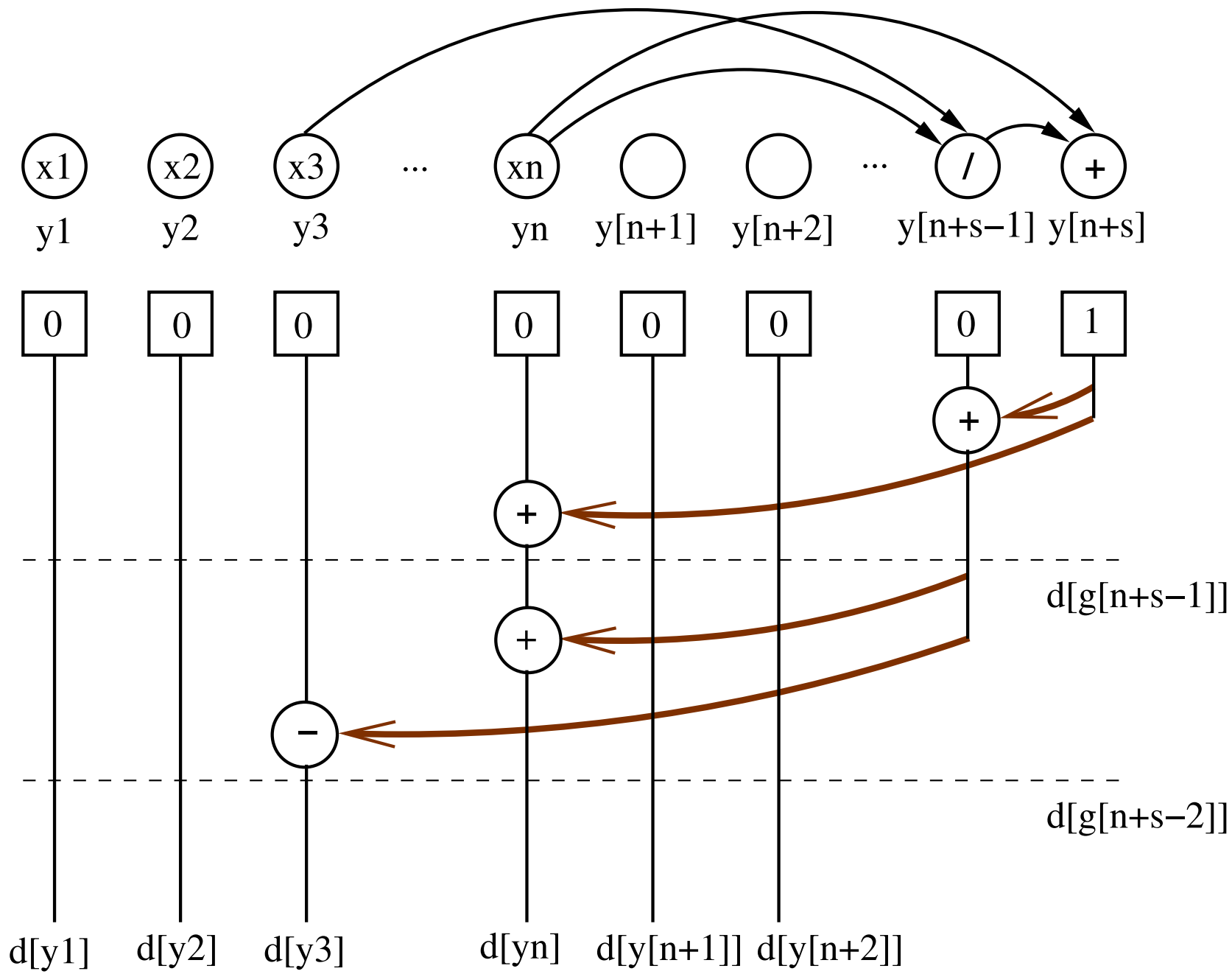
$$g_{i-1}(y_1, \dots, y_{i-1}) := g_i(y_1, \dots, y_{i-1}, h_i(y_{L_i}, y_{R_i})), i = n+s, \dots, n+1.$$

By chain rule for $j \neq L_i, j \neq R_i$

$$(\partial_{y_j} g_{i-1})(y_1, \dots, y_{i-1}) = (\partial_{y_j} g_i)(y_1, \dots, y_{i-1}, h_i(y_{L_i}, y_{R_i}))$$

and for $j = L_i$ or $j = R_i$

$$\begin{aligned}(\partial_{y_j} g_{i-1})(y_1, \dots, y_{i-1}) &= (\partial_{y_j} g_i)(y_1, \dots, y_{i-1}, h_i(y_{L_i}, y_{R_i})) \\ &\quad + (\partial_{y_i} g_i)(y_1, \dots, y_{i-1}, h_i(y_{L_i}, y_{R_i})) \cdot (\partial_{y_j} h_i)(y_{L_i}, y_{R_i}).\end{aligned}$$



Ostrowski et al. '71: Size (= sequential time) of circuit for $\partial f / \partial x_i$
 $\leq 4s$

Kaltofen & Singer '91: Depth (= parallel time) of circuit for $\partial f / \partial x_i$
 $= O(\text{depth of circuit for } f)$.

Transformation is numerically stable.

Inverted transposition principle by automatic differentiation

The problems $A^{-1} \cdot b$ and $(A^{Tr})^{-1} \cdot b$, given $A \in \mathbb{F}^{n \times n}$ non-singular and $b \in \mathbb{F}^n$, have the same asymptotic circuit complexity: Let

$$f(x_1, \dots, x_n) = ([x_1 \ \dots \ x_n] \cdot (A^{Tr})^{-1}) \cdot b \in \mathbb{F}[x_1, \dots, x_n].$$

Then

$$\begin{bmatrix} \partial_{x_1} f \\ \vdots \\ \partial_{x_n} f \end{bmatrix} = (A^{Tr})^{-1} b.$$

Note: Transposition principle may not apply due to divisions.

Inverted transposition principle by automatic differentiation

The problems $A^{-1} \cdot b$ and $(A^{Tr})^{-1} \cdot b$, given $A \in \mathbb{F}^{n \times n}$ non-singular and $b \in \mathbb{F}^n$, have the same asymptotic circuit complexity: Let

$$f(x_1, \dots, x_n) = ([x_1 \ \dots \ x_n] \cdot (A^{Tr})^{-1}) \cdot b \in \mathbb{F}[x_1, \dots, x_n].$$

Then

$$\begin{bmatrix} \partial_{x_1} f \\ \vdots \\ \partial_{x_n} f \end{bmatrix} = (A^{Tr})^{-1} b.$$

Note: Transposition principle may not apply due to divisions.

Used for:

- $(\text{Vandermonde}^{Tr})^{-1} \cdot b$ for sparse polynomial interpolation (Kaltofen and Lakshman '88).
- Constant improvements to interpolation via the transposed algorithm [Bostan, Lecerf, Schost '03].

Reduction: Matrix Inverse \preceq Determinant (Baur, Strassen '83)

Consider a circuit for the determinant,

$$f(a_{1,1}, \dots, a_{n,n}) = \det \left(\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \right).$$

Then

$$(-1)^{i+j} \frac{\partial f}{\partial a_{j,i}} = \det(A) (A^{-1})_{i,j}.$$

\implies Circuit for partials computes adjoint matrix. Used for:

- Division-free computation of adjoint of A in $O(n^{2.69})$ arithmetic operations (Kaltofen and Villard '01/'04).

Villard 2003: automatic differentiation does not preserve bit complexity

$x^T y c$ where x, y are vectors with constant entries,
 c a large constant

takes $O(n + \log |c|)$ bit operations,
 yc takes $O(n \log |c|)$ bit operations

My ECCAD'98 open problem 6

Let $\sigma \in \mathbb{F}[\alpha, \beta]/(f, g)$ where $f(\alpha, \beta) = 0$ and $g(\beta) = 0$.

E.g., $\sigma = \sqrt{1 + \sqrt{2}} - \sqrt{2} = \alpha - \beta$, $f = \alpha^2 - \beta - 1$, and $g = \beta^2 - 2$.

Task: Compute the minimum polynomial $h(\sigma) = 0$:

$$h(x) = x^m - c_{m-1}x^{m-1} - \dots - c_0 \in \mathbb{F}[x], \quad m \leq \deg(f) \cdot \deg(g)$$

The coefficient vectors $\overrightarrow{\sigma^i}$ of $\sigma^i \bmod (f(\alpha, \beta), g(\beta))$ satisfy

$$\forall j \geq 0: \overrightarrow{\sigma^{m+j}} = c_{m-1} \overrightarrow{\sigma^{m-1+j}} + \dots + c_0 \overrightarrow{\sigma^j}$$

Any non-trivial linear projection $\mathcal{L}(\overrightarrow{\sigma^i})$ preserves the linear recursion because h is irreducible.

Power Projections = Transposed Modular Polyn Composition

Linear projections of powers

$$\left[\mathcal{L}(\vec{\sigma}^0) \quad \mathcal{L}(\vec{\sigma}^1) \quad \mathcal{L}(\vec{\sigma}^2) \quad \dots \right] = [u_0 \quad u_1 \quad \dots \quad u_{n-1}] \cdot \underbrace{\left[\vec{\sigma}^0 \mid \vec{\sigma}^1 \mid \vec{\sigma}^2 \mid \dots \right]}_A$$

Modular polynomial composition

$$w(z) = w_0 + w_1 z + w_2 z^2 + \dots \longmapsto w(\sigma) \bmod (f(\alpha, \beta), g(\beta))$$

$$\vec{w}(\sigma) = \underbrace{\left[\vec{\sigma}^0 \mid \vec{\sigma}^1 \mid \vec{\sigma}^2 \mid \dots \right]}_A \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{bmatrix}$$

By Tellegen's Principle [1960] the problems can be solved equally fast

Transposed Modular Polynomial Multiplication in NTL

1. $T_1 \leftarrow \text{FFT}^{-1}(\text{RED}_k(g))$
2. $T_2 \leftarrow T_1 \cdot S_2$
3. $v \leftarrow -\text{CRT}_{0\dots n-2}(\text{FFT}(T_2))$
4. $T_2 \leftarrow \text{FFT}^{-1}(\text{RED}_{k+1}(x^{n-1} \cdot v))$
5. $T_2 \leftarrow T_2 \cdot S_3$
6. $T_1 \leftarrow T_1 \cdot S_4$
7. Replace T_1 by the 2^{k+1} -point residue table whose j -th column ($0 \leq j < 2^{k+1}$) is 0 if j is odd, and is column number $j/2$ of T_1 if j is even.
8. $T_2 \leftarrow T_2 + T_1$
9. $u \leftarrow \text{CRT}_{0\dots n-1}(\text{FFT}(T_2))$

“we offer no other proof of correctness other than the validity of this transformation technique (and the fact that it does indeed work in practice)” [Shoup 1994]

Open Problem 1

*With inputs $A \in \mathbb{F}^{m \times n}$ and $y \in \mathbb{F}^n$ you are given **an algorithm** for $A \cdot y$ that uses $T(m, n)$ arithmetic field operations and $S(m, n)$ auxiliary space.*

*Show how to construct **an algorithm** for $A^T \cdot z$ where $z \in \mathbb{F}^m$ that uses $O(T(m, n))$ time and $O(S(m, n))$ space.*

Your construction must be applicable to practical problems.

Pebble game by Gilbert et al.

When computing $v_i \leftarrow v_{L_i} \circ_i v_{R_i}$, values of v_{L_i} and v_{R_i} must be stored in memory or recomputed.

—→ not clear how to use the same number of “pebbles” for reverse program.

Use inplace operations: $x += y$;

Pebble for one operand can be used for result, which is reversible.

Not clear how to reverse the address computation (space uniformity of reverse circuit)