Slides for a Course on the Analysis and Design of Algorithms

Chapter 7: Digital Representation of Signals

Stephen J. Hegner
Department of Computing Science
Umeå University
Sweden
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

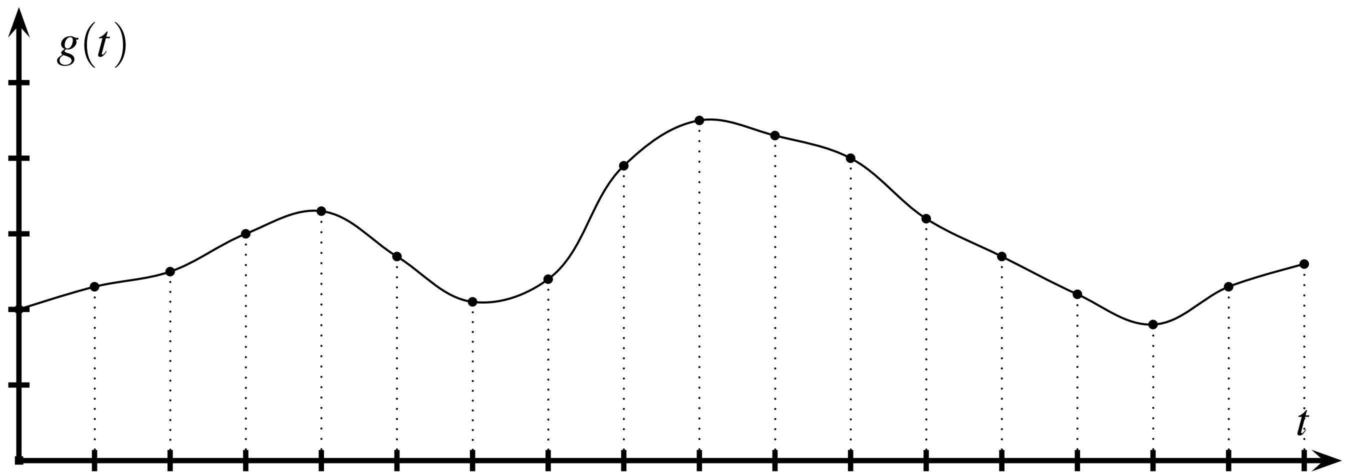
7. Digital Representation of Signals

7.1 Basic Concepts

7.1.1 Background and motivation

- In modern electronic systems, analog signals are often represented by a set of *samples*.
- This is termed *digitization* of the signal.

Question: Under which conditions can the signal be reconstructed from these samples, and how?



- Such a reconstruction is (obviously) impossible in general, but may be achieved under the certain assumptions regarding the properties of the original signal.
- The key assumption is that the original signal may be represented as a linear combination of *basis signals*.

General form:

$$g(t) = \sum_{k=0}^{p} a_k \cdot f_k(t)$$

in which:

- The a_k's are constants.
- f_k is the k^{th} *basis signal*.

Example: polynomial representation

$$f_k(t) = t^k$$
$$g(t) = \sum_{k=0}^{p} a_k \cdot t^k$$

Example: sinusoidal representation

- In this case, the basis signals are of the form

$$f_k(t) = e^{2\pi i \cdot f \cdot t}$$

- $i = \sqrt{-1}$ is the "imaginary" unit of the complex numbers.
- f is a "base frequency" — the signal to be represented is assumed to be periodic of this frequency.
- In effect, this amounts to a representation of the form

$$g(t) = \sum_{k=0}^{p} (a_k \cdot \cos(2\pi f \cdot k \cdot t) + b_k \cdot \sin(2\pi f \cdot k \cdot t))$$

- Each of these representations, as well as methods of translation between them, will now be investigated.

7.2 Polynomial Representation

7.2.1 Fact – unique curve fitting with polynomials *Given a sequence $\langle (x_1, y_1), \ldots (x_n, y_n) \rangle$ of points with $x_i < x_{i+1}$ for $1 \leq i \leq n-1$, there is a unique polynomial of degree at most $n-1$ which passes through each of these points.*

PROOF: First, uniqueness is established. Let p_1 and p_2 each be polynomials of degree at most $n-1$ with the further property that

$$p_j(x_i) = y_i$$

for $j \in \{1, 2\}$ and $i \in \{1, 2, \ldots, n\}$. Then

$$p_2(x_i) - p_1(x_i) = 0$$

for $i \in \{1, 2, \ldots, n\}$. However, $p_2 - p_1$ is also a polynomial of degree at most $n-1$, with roots $\{x_1, x_2, \ldots, x_n\}$. If $p_2 - p_1$ is not the zero polynomial, then by the fundamental theorem of algebra (1.8.3), each $(x - x_i)$ must be a factor of $p_2(x) - p_1(x)$, so

$$(p_2 - p_1)(x) = (x - x_1) \cdot (x - x_2) \cdot \ldots \cdot (x - x_n) \cdot q(x)$$

for some nonzero polynomial $q(x)$. Since the above polynomial is of degree at least n, yet $p_2 - p_1$ is of degree at most $n-1$, it follows that $p_2 - p_1 = 0$; *i.e.*, $p_1 = p_2$.

To establish existence, it suffices to observe that the following polynomial has the desired properties.

$$p(x) := \sum_{i=1}^{n} \left(\prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} \right) \cdot y_i$$

□

7.2.2 LaGrange interpolation

The formula

$$p(x) := \sum_{i=1}^{n} \left(\prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} \right) \cdot y_i \qquad (*)$$

introduced in the proof of 7.2.1 is called the *LaGrange interpolation* of $\langle (x_1, y_1), \ldots (x_n, y_n) \rangle$.

- Each summand of this formula involves $\Theta(n^2)$ multiplications to expand the numerator. [To expand the factored polynomial into an unfactored one requires $\Theta(n^2)$ multiplications of the form $x_j \cdot x_{j'}$.]
- Each denominator is a number. There is a one-time amortized cost of $\Theta(n^2)$ to compute all factors of the form $(x_i - x_j)$, and a time of $\Theta(n)$ for each denominator to perform the associated multiplications.
- Thus, each summand requires $\Theta(n^2)$ time, and so a naïve algorithm for computing the LaGrange interpolation of a polynomial of degree n requires $\Theta(n^3)$ time.
- There are two ways to improve upon this bound.

7.2.3 LaGrange interpolation via division

- In the formula * of 7.2.2, note that two distinct summands differ in only one factor in the numerator, and one in the denominator.
- The idea is to take advantage of this similarity and avoid repeated multiplications.
- The problem is that each factor is missing in at least one product.
- The solution proceeds in four steps:

  (i) Compute the large product of the form

  $$\prod_{j=1}^{n}(x - x_j)$$

  just once as an expanded polynomial of the form:

  $$q(x) = a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0$$

  (ii) Divide this polynomial by $(x - x_i)$ to obtain

  $$q_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} (x - x_j)$$

  (iii) Compute all numbers of the form $x_i - x_j$ for $i \neq j$.

  (iv) Using the result of (iii), compute the numbers

  $$c_i = 1/(\prod_{\substack{j=1 \\ j \neq i}}^{n} (x_i - x_j))$$

  one for each i.

(v) The final result is then

$$p(x) = \sum_{i=1}^{n} c_i \cdot q_i(x)$$

- Steps (i), (iii), and (iv) are each easily seen to be realizable in time $O(n^2)$.
- It is also possible to perform a single division of the form

$$q(x)/(x - x_j)$$

  in time $\Theta(n)$, as shown below.
- Therefore the entire operation may be performed in time $\Theta(n^2)$.

7.2.4 Lemma – polynomial division *Let $r(x)$ be a polynomial in x of degree $n \geq 1$, and let $s(x)$ be a polynomial of degree one which is a factor of $r(x)$. There exists a $\Theta(n)$ worst-case time algorithm for computing $r(x)/s(x)$.*

PROOF OUTLINE: It is easiest to see how things work by running through an example. One simply performs long division, as one would with numbers. Each of the n steps of the division takes constant time.

$$\begin{array}{r|l}
 & x^2 - 3x + 2 \\ \hline
x - 3 & x^3 - 6x^2 + 11x - 6 \\
 & \underline{x^3 - 3x^2} \\
 & \quad - 3x^2 + 11x \\
 & \quad \underline{- 3x^2 + 9x} \\
 & \qquad\qquad 2x - 6 \\
 & \qquad\qquad \underline{2x - 6}
\end{array}$$

□

7.2.5 LaGrange interpolation via stepwise interpolation

- Let $S = \langle (x_1, y_1), \ldots, (x_n, y_n) \rangle$ be a nonempty sequence of n pairs of real numbers, let $m \le n$, and let p be a polynomial of degree $m-1$ with the property that

  $$p(x_i) = y_i$$

  for $1 \le i \le m-1$. Define the polynomial

  $$\mathsf{RFit}(p, S, m)(x) := (y_m - p(x_m)) \cdot \prod_{j=1}^{m-1} \frac{(x - x_j)}{(x_m - x_j)} + p(x) \qquad (*)$$

7.2.6 Lemma *Conditions as in 7.2.5 above, the polynomial* $\mathsf{RFit}(p, S, m)$ *satisfies*

$$\mathsf{RFit}(p, S, m)(x_i) = y_i$$

for all i, $1 \le i \le m$. *In particular,* $\mathsf{RFit}(p, S, n)$ *passes through every point of S.*

PROOF: For $1 \le i \le m-1$, *i.e.*, for $x = x_i$, the term $(x - x_j)$ with $x_j = x_i$ in the large product will be zero, and so the entire left summand is zero. The entire value will therefore be the right summand, which has value $p(x_i)$.

For $i = m$, *i.e.*, for $x = x_m$, the large $(m-1)$-fold product evaluates to one, and so the value is $(y_m - p(x_m)) \cdot 1 + p(x_m) = y_m$. □

7.2.7 Newtonian interpolation

- The realization of an algorithm which uses the formula (*) of 7.2.5 is straightforward in principle.
- The induction is “primed” by choosing the initial polyomial to be $p(x) = y_1$.

- However, to understand the complexity, it is necessary to identify two data types and three critical operations.

  Arbitrary polynomials: The data type Poly consists of all polynomials in a single variable, which is usually taken to be x.

  Polynomials of degree one: The data type Poly1 consists of just those polynomials in Poly which have degree one and lead coefficient one; *i.e.*, those polynomials which are of the form $x+a$, with $a \in \mathbb{R}$.

  Polynomial evaluation: The operation

  $$\mathsf{PolyEval} : \mathsf{Poly} \times \mathbb{R} \to \mathbb{R}$$

  evaluates a polynomial at a real number, returning a real number.

  Polynomial addition: The operation

  $$\mathsf{PolyAdd} : \mathsf{Poly} \times \mathsf{Poly} \to \mathsf{Poly}$$

  adds two polynomials together, returning a polynomial.

  Limited polynomial multiplication: The operation

  $$\mathsf{PolyMult1} : \mathsf{Poly} \times \mathsf{Poly1} \to \mathsf{Poly}$$

  takes an arbitrary polynomial and polynomial of degree one, and returns their product.

  Scalar polynomial multiplication: The operation

  $$\mathsf{PolyMultReal} : \mathsf{Poly} \times \mathbb{R} \to \mathsf{Poly}$$

  takes an arbitrary polynomial and a real number, and returns the result of multiplying each coefficient of the polynomial by the real number.

- As the first step in building an algorithm for polynomial interpolation based upon the rules of 7.2.5, define $\mathsf{RFitAux}(S,i)$ to be the expansion (multiplied out) of the following intermediate polynomial.
$$(x-x_1)\cdot(x-x_2)\cdot\ldots\cdot(x-x_i)$$
- Note that
$$\mathsf{RFitAux}(S,i)=\mathsf{PolyMult1}(\mathsf{RFitAux}(S,i-1),(x-x_i))$$
- Next, note that $\mathsf{RFit}(p,S,m)$ may be expressed somewhat informally by the following formula.

$$\mathsf{RFit}(p,S,m)=\\ (y_m-p(x_m))\cdot\frac{\mathsf{RFitAux}(S,m-1)}{\mathsf{PolyEval}(\mathsf{RFitAux}(S,m-1),x_m)}+\mathsf{RFit}(p,S,m-1)$$

- To make things precise for an algorithm, all of the polynomial operations must be expressed explicitly.

$$\mathsf{RFit}(p,S,m)=\\ \mathsf{PolyAdd}\Bigg(\mathsf{PolyMultReal}\Bigg(\mathsf{RFitAux}(S,m-1),\frac{y_m-\mathsf{PolyEval}(p,x_m)}{\mathsf{PolyEval}(\mathsf{RFitAux}(S,m-1),x_m)}\Bigg),\\ \mathsf{RFit}(p,S,m-1)\Bigg) \qquad (NI)$$

- This equation represents the basis of *Newtonian interpolation*.
- It is applied iteratively to obtain the interpolating polynomial.
- The complexity will now be examined.

7.2.8 Horner's rule

- The function PolyEval is best realized by *Horner's rule*.
- The idea is easily illustrated via example.
- Consider the polynomial $2x^3+3x^2+5x+1$.

$$\begin{aligned} 2x^3+3x^2+5x+1 &= x\cdot(2x^2+3x+5)+1 \\ &= x\cdot(x\cdot(2x+3)+5)+1 \\ &= x\cdot(x\cdot(x\cdot(x\cdot(2)+3)+5)+1 \end{aligned}$$

- To evaluate this polynomial at $x=4$, use an "inside-out" approach on the last representation in the sequence:
  - Evaluate $(2) = 2$;
  - Evaluate $x\cdot(2)+3 = 11$;
  - Evaluate $x\cdot(11)+5 = 49$;
  - Evaluate $x\cdot(49)+1 = 197$.
- Pseudocode for this algorithm is as follows:

function PolyEval$(p:$ polynomial, $a:$ real$):$ real;
  /* Function to evaluate $p(a)$ */
  /* Note: $\mathit{coefficient}(\sum(a_i\cdot x_i),i)=a_i$ */
  $s \leftarrow \mathit{coefficient}(p,\mathit{degree}(p))$;
  for $i \leftarrow \mathit{degree}(p)-1$ **downto** 0 **do**
    $s \leftarrow s*a+\mathit{coefficient}(p,i)$;
  return s;

7.2.9 The complexity of Newtonian interpolation

- Horner's rule (7.2.8) runs in time $\Theta(n)$, with n the degree of the polynomial.
- The function PolyAdd simply adds coefficients together, power-by-power, and so may easily be realized to run in time $\Theta(n)$, with n the maximum of the degrees of the two polynomials.
- The function PolyMult1 may be realized with $2 \cdot n$ multiplications, where n is the degree of the larger polynomial. Thus, it may be realized in time $\Theta(n)$.
- The function PolyMultReal may be realized in time $\Theta(n)$, with n the degree of the polynomial, since it just multiplies each coefficient of the polynomial by a constant.
- It follows that one iteration of the assignment (NI) of 7.2.7 will run in time $\Theta(n)$.
- Since this computation must be iterated for m from 2 to n, it follows that the entire process of Newtonian interpolation requires $\Theta(n^2)$ time.

7.2.10 The Vandermonde matrix

- The matrix equation

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \cdots x_n^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

  characterizes a polynomial

$$p(x) = \sum_{i=1}^{n} a_i \cdot x_i$$

  which passes through the points in $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

- Note that the square matrix in the middle has the form

$$\begin{pmatrix} C^0 & C^1 & C^2 & \cdots & C^{n-1} \end{pmatrix} \qquad \text{with} \qquad C = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

  and C^i meaning that each member of C is raised to the i^{th} power.

- Such a structure is called a *Vandermonde matrix*, and has determinant

$$\prod_{1 \leq j < k \leq n} (x_k - x_j)$$

  and so is invertible (since the x_i's are all distinct).

- Solving systems of linear equations takes $O(n^3)$ time, and so this is not the best approach for computing an interpolating polynomial.

- The Vandermonde-matrix perspective will be useful in the study of sinusoidal representations, however.

7.3 Sinusoidal Representation

7.3.1 Complex numbers and roots of unity

- Following standard mathematical notation, the symbol i will be used in the remainder of this chapter to denote $\sqrt{-1}$.

Note: Electrical engineers usually use j instead, since in their domain i has already been conscripted to denote current.

- The symbol e is also reserved for the rest of this section to denote the basis of the natural logarithms: $\sum_{i=1}^{\infty} 1/k!$

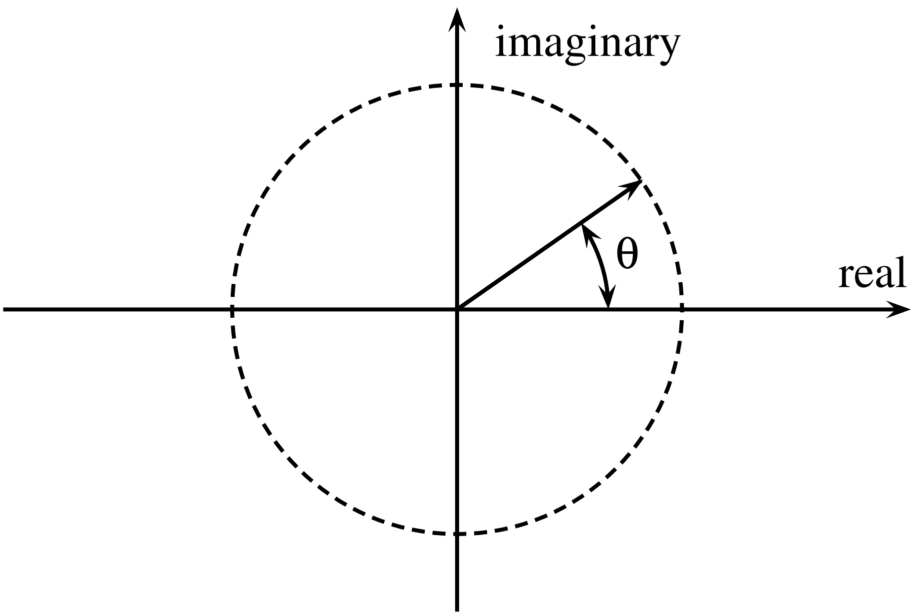
- The identity

$$e^{i\theta} = \cos(\theta) + i \cdot \sin(\theta)$$

is well known.

- From it and the Pythagorean equation, it follows that for any real number θ, $e^{i\theta}$ has magnitude one.

- In the complex plane, $e^{i\theta}$ may be visualized as a unit vector at angle θ (in radians) from the real axis:



7.3.2 Roots of unity

(a) A complex number c with the property that $c^n = 1$ is called an n^{th} *root of unity*.

(b) A complex n^{th} root of unity c for which $c^k \neq 1$ for $0 < k < n$ is called a *principal* (or *primitive*) n^{th} root of unity.

7.3.3 Properties of principal n^{th} roots of unity

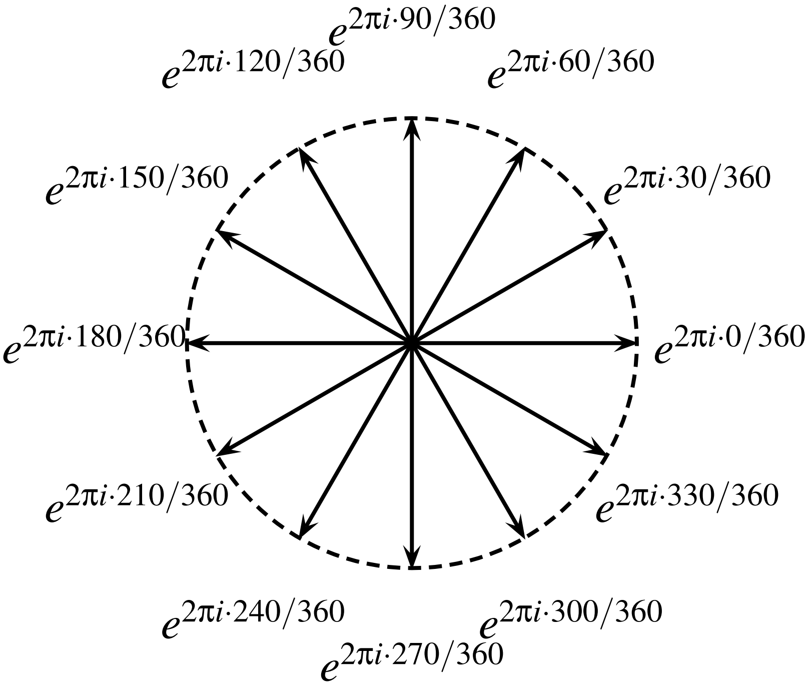
(a) *If k and n are relatively prime positive integers with $k < n$, then $e^{2\pi ik/n}$ is a principal n^{th} root of unity.*

(b) *For any principal n^{th} root of unity ρ,*

$$\sum_{k=0}^{n-1} \rho^k = \sum_{k=1}^{n} \rho^k = 0.$$

PROOF OUTLINE:

- Part (a) is immediate from the fact that $e^{2\pi ik/n^m} = 1$ iff $(2\pi ik/n) \cdot m$ is a multiple of 2π, which is true iff $(m \cdot k/n)$ is an integer.
- Part (b) is most easily seen by visualizing the vector addition of the summands. The diagram below shows this situation for $n = 12$.



□

7.3.4 Notation for principal n^{th} roots of unity

- $\rho_n = e^{2\pi i/n}$
- $\rho_{-n} = e^{2\pi i(n-1)/n}$

7.4 The Discrete Fourier Transform

7.4.1 The problem of polynomial multiplication

- In signal- and image-processing applications, the operation of multiplying two polynomials arises frequently.
- This operation is often termed *convolution*, and is denoted by $*$.
- More precisely, let

$$p_1(x) = \sum_{k=0}^{n_1-1} a_k \cdot x^k \qquad\qquad p_2(x) = \sum_{k=0}^{n_2-1} b_k \cdot x^k$$

- Then

$$p_1 * p_2 = \sum_{k=0}^{n_1+n_2-2} \sum_{r+s=k} a_r \cdot b_s \cdot x^{r+s}$$

- The "naïve" algorithm for computing the convolution of two polynomials requires time $\Theta(n_1 \cdot n_2)$, or $\Theta(n^2)$ if both polynomials are of degree n.
- Using the discrete Fourier transform plus a divide-and-conquer strategy, it will be shown that this time may be reduced to $\Theta(n \cdot \log(n))$.

7.4.2 An overview of the discrete Fourier transform

- Operationally, the discrete Fourier transform may be viewed as a mapping from polynomials to polynomials.
- If
$$p = \sum_{k=0}^{n-1} a_k \cdot x^k$$
and $m \geq n$, then the *discrete Fourier transform* of p of degree m
$$\mathfrak{F}_m(p) = \sum_{k=0}^{m-1} \mathsf{F}_m(p,k) \cdot x^k$$
is a complex polynomial of degree at most $n-1$.
- The coefficients $\mathsf{F}_m(p,k)$ are in general complex numbers, even though all of the a_k's are real.
- This transform has the following desirable properties:
  1. It is invertible; that is, there is a mapping $\mathfrak{F}_m^{-1}$ (called the inverse discrete Fourier transform) with the property that
  $$\mathfrak{F}_m^{-1}(\mathfrak{F}_m(p)) = \mathfrak{F}_m(\mathfrak{F}_m^{-1}(p)) = p$$
  2. The convolution $p * q$ of two polynomials corresponds to pointwise multiplication of the corresponding coefficients of the discrete Fourier transform. If
  $$p_1(x) = \sum_{k=0}^{n-1} a_k \cdot x^k \qquad p_2(x) = \sum_{k=0}^{n-1} b_k \cdot x^k$$
  then
  $$\mathfrak{F}_m(p_1 * p_2) = \sum_{k=0}^{2n-2} \mathsf{F}_m(p_1,k) \cdot \mathsf{F}_m(p_2,k) \cdot x^k$$

- To compute $p * q$, the solid path indicated in the following diagram is followed:

$$
\begin{tikzcd}
(p,q) \arrow[r, dashed] \arrow[d, "\Theta(n \cdot \log(n))"'] & p * q = \mathfrak{F}^{-1}_{2n-2}(\mathfrak{F}_{2n-2}(p) \odot \mathfrak{F}_{2n-2}(p)) \\
(\mathfrak{F}_{2n-2}(p), \mathfrak{F}_{2n-2}(q)) \arrow[r, "\Theta(n)"'] & \mathfrak{F}_{2n-2}(p) \odot \mathfrak{F}_{2n-2}(q) \arrow[u, "\Theta(n \cdot \log(n))"']
\end{tikzcd}
$$

- In the above $\odot$ denotes pointwise multiplication.
- It will now be shown in detail how these computations are realized using an implementation of the discrete Fourier transform known as the fast Fourier transform.

7.4.3 The discrete Fourier transform

- Let

$$p(x) = \sum_{k=0}^{n-1} a_k \cdot x^k$$

be a polynomial of degree at most $n-1$, and let $m \geq n$.

(a) The m^{th}-degree *discrete Fourier transform* (*DFT* for short) of p is given by

$$\mathfrak{F}_m(p) = \sum_{k=0}^{m-1} \mathsf{F}_m(p,k) \cdot x^k$$

with

$$\mathsf{F}_m(p,k) = p(\rho_m^k) = p(e^{2\pi ik/m})$$

- Thus,

$$\mathsf{F}_m(p,k) = \sum_{\ell=0}^{n-1} a_\ell \cdot e^{2\pi ik\ell/m} = \sum_{\ell=0}^{n-1} a_\ell \cdot \rho_m^{k\ell}$$

7.4.4 Notation

- The polynomial p, as given above, may be represented by the sequence

$$\langle a_0, a_1, a_2, \ldots, a_{n-1} \rangle$$

- The m^{th}-degree DFT may then be viewed as the sequence

$$\langle \mathsf{F}_m(p,0), \mathsf{F}_m(p,1), \mathsf{F}_m(p,2), \ldots, \mathsf{F}_m(p,m-1) \rangle$$

- This notation will be used frequently in that which follows.

7.4.5 Matrix representation

- There is a convenient matrix representation of the DFT:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \rho_m & \rho_m^2 & \rho_m^3 & \cdots & \rho_m^{m-1} \\ 1 & \rho_m^2 & \rho_m^4 & \rho_m^6 & \cdots & \rho_m^{2(m-1)} \\ 1 & \rho_m^3 & \rho_m^6 & \rho_m^9 & \cdots & \rho_m^{3(m-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \rho_m^{m-1} & \rho_m^{2(m-1)} & \rho_m^{3(m-1)} & \cdots & \rho_m^{(m-1)^2} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} \mathsf{F}_m(p,0) \\ \mathsf{F}_m(p,1) \\ \mathsf{F}_m(p,2) \\ \mathsf{F}_m(p,3) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \mathsf{F}_m(p,m-1) \end{pmatrix}$$

- This representation leads naturally to to the construction of the inverse DFT.

7.4.6 Lemma *The above square matrix has the following inverse:*

$$\frac{1}{m} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \rho_{-m} & \rho_{-m}^2 & \rho_{-m}^3 & \cdots & \rho_{-m}^{m-1} \\ 1 & \rho_{-m}^2 & \rho_{-m}^4 & \rho_{-m}^6 & \cdots & \rho_{-m}^{2(m-1)} \\ 1 & \rho_{-m}^3 & \rho_{-m}^6 & \rho_{-m}^9 & \cdots & \rho_{-m}^{3(m-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \rho_{-m}^{m-1} & \rho_{-m}^{2(m-1)} & \rho_{-m}^{3(m-1)} & \cdots & \rho_{-m}^{(m-1)^2} \end{pmatrix} \quad \square$$

7.4.7 The inverse discrete Fourier transform

- Let

$$p(x) = \sum_{k=0}^{n-1} a_k \cdot x^k$$

be a polynomial of degree at most $n-1$, and let $m \geq n$.

(a) The m^{th}-degree *inverse discrete Fourier transform* (*inverse DFT* for short) of p is given by

$$\mathfrak{F}_m^{-1}(p) = \sum_{k=0}^{m-1} \mathsf{F}_m^{-1}(p,k) \cdot x^k$$

with

$$\mathsf{F}_m^{-1}(p,k) = \frac{1}{m} \cdot p(\rho_{-m}^{k}) = \frac{1}{m} \cdot p(e^{-2\pi ik/m})$$

- Thus,

$$\mathsf{F}_m^{-1}(p,k) = \frac{1}{m} \cdot \sum_{\ell=0}^{n-1} a_\ell \cdot e^{-2\pi ik\ell/m} = \frac{1}{m} \cdot \sum_{\ell=0}^{n-1} a_\ell \cdot \rho_{-m}^{k\ell}$$

7.4.8 Notation

- In analogy to 7.4.4, if the polynomial p may be represented by the sequence

$$\langle a_0, a_1, a_2, \ldots, a_{n-1} \rangle$$

then the m^{th}-degree inverse DFT may be viewed as the sequence

$$\langle \mathsf{F}_m^{-1}(p,0), \mathsf{F}_m^{-1}(p,1), \mathsf{F}_m^{-1}(p,2), \ldots, \mathsf{F}_m^{-1}(p,m-1) \rangle$$

- This notation will be used frequently in that which follows.

7.4.9 Matrix representation of the inverse DFT

- Using 7.4.6, and in analogy to 7.4.5, there is a convenient matrix representation of the inverse DFT:

$$\frac{1}{m}\cdot\begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \rho_{-m} & \rho_{-m}^2 & \rho_{-m}^3 & \cdots & \rho_{-m}^{m-1} \\ 1 & \rho_{-m}^2 & \rho_{-m}^4 & \rho_{-m}^6 & \cdots & \rho_{-m}^{2(m-1)} \\ 1 & \rho_{-m}^3 & \rho_{-m}^6 & \rho_{-m}^9 & \cdots & \rho_{-m}^{3(m-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \rho_{-m}^{m-1} & \rho_{-m}^{2(m-1)} & \rho_{-m}^{3(m-1)} & \cdots & \rho_{-m}^{(m-1)^2} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} \mathsf{F}_m^{-1}(p,0) \\ \mathsf{F}_m^{-1}(p,1) \\ \mathsf{F}_m^{-1}(p,2) \\ \mathsf{F}_m^{-1}(p,3) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \mathsf{F}_m^{-1}(p,m-1) \end{pmatrix}$$

7.4.10 Proposition – the inverse DFT is really an inverse

- *For any polynomial p of degree at most $n-1$, and any $m \geq n$,*

$$\mathfrak{F}_m^{-1}(\mathfrak{F}_m(p)) = \mathfrak{F}_m(\mathfrak{F}_m^{-1}(p)) = p$$

PROOF: This follows immediately from the matrix representations of 7.4.5, 7.4.6 and 7.4.9. □

7.4.11 Further notation

(a) The symbol $\odot$ will be used to denote pointwise multiplication. Thus, if

$$p_1(x) = \sum_{k=0}^{n-1} a_k \cdot x^k \qquad p_2(x) = \sum_{k=0}^{n-1} b_k \cdot x^k$$

Then

$$p_1 \odot p_2 = \sum_{k=0}^{n-2} a_k \cdot b_k \cdot x^k$$

(b) For any complex number α and any positive integer n, define the matrix

$$\mathsf{M}(\alpha, n) = \begin{pmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{m-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \cdots & \alpha^{2(m-1)} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \cdots & \alpha^{3(m-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{m-1} & \alpha^{2(m-1)} & \alpha^{3(m-1)} & \cdots & \alpha^{(m-1)^2} \end{pmatrix}$$

- Note that the matrix which represents the m^{th} degree DFT is

  $$\mathsf{M}(\rho_m, m)$$

  while that which represents the m^{th}-degree inverse DFT is

  $$\frac{1}{m} \cdot \mathsf{M}(\rho_{-m}, m)$$

- The problem of showing that

$$\mathfrak{F}_m(p * q) = \mathfrak{F}_m(p) \odot \mathfrak{F}_m(q)$$

will now be addressed.

7.4.12 Proposition *Let*

$$p_1(x) = \sum_{k=0}^{n-1} a_k \cdot x^k \qquad\qquad p_2(x) = \sum_{k=0}^{n-1} b_k \cdot x^k$$

be two polynomials of degree at most $n-1$, and let

$$q = \sum_{k=0}^{n-2} c_k \cdot x_k := p_1 * p_2 = \sum_{k=0}^{n_1+n_2-2} \sum_{r+s=k} a_r \cdot b_s \cdot x^{r+s}$$

Then

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ c_{2n\text{-}2} \end{pmatrix} = \begin{pmatrix} a_0 & 0 & 0 & 0 & \cdots & 0 & a_{n\text{-}1} & a_{n\text{-}2} & a_{n\text{-}3} & \cdots & a_1 \\ a_1 & a_0 & 0 & 0 & \cdots & 0 & 0 & a_{n\text{-}1} & a_{n\text{-}2} & \cdots & a_2 \\ a_2 & a_1 & a_0 & 0 & \cdots & 0 & 0 & 0 & a_{n\text{-}1} & \cdots & a_2 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{n\text{-}2} & a_{n\text{-}3} & a_{n\text{-}4} & a_{n\text{-}5} & \cdots & 0 & 0 & 0 & \cdots & 0 & a_{n\text{-}1} \\ a_{n\text{-}1} & a_{n\text{-}2} & a_{n\text{-}3} & a_{n\text{-}4} & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & a_{n\text{-}1} & a_{n\text{-}2} & a_{n\text{-}3} & \cdots & a_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & a_{n\text{-}1} & a_{n\text{-}2} & \cdots & a_1 & a_0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & a_{n\text{-}1} & \cdots & a_2 & a_1 & a_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{n\text{-}1} & a_{n\text{-}2} & a_{n\text{-}3} & \cdots & a_1 & a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_{n\text{-}1} \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix}$$

PROOF: Straightforward verification. □

7.4.13 Some useful matrix and vector notation

Let

$$V = \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{\ell-1} \\ v_\ell \end{pmatrix}$$

Define

$$\mathsf{RotDn}(V) = \begin{pmatrix} v_\ell \\ v_0 \\ v_1 \\ \vdots \\ v_{\ell-2} \\ v_{\ell-1} \end{pmatrix}$$

and

$$\mathsf{Circ}(V) = \begin{pmatrix} v_0 & v_\ell & v_{\ell-1} & \cdots v_1 & \\ v_1 & v_0 & v_\ell & \cdots v_2 & \\ v_2 & v_1 & v_0 & \cdots v_3 & \\ \vdots & & & & \\ \vdots & & & & \\ \vdots & & & & \\ \vdots & & & & \\ \vdots & & & & \\ v_{\ell-1} & v_{\ell-2} & v_{\ell-3} & \cdots & v_\ell \\ v_\ell & v_{\ell-1} & v_{\ell-2} & \cdots & v_0 \end{pmatrix}$$

$$= \begin{pmatrix} v & \mathsf{RotDn}(V) & \mathsf{RotDn}^2(V) & \cdots & \mathsf{RotDn}^{\ell-1}(V) \end{pmatrix}$$

Now let $m \geq n$, and for

$$p_1(x) = \sum_{k=0}^{n-1} a_k \cdot x^k \qquad\qquad p_2(x) = \sum_{k=0}^{n-1} b_k \cdot x^k$$

define the following two vectors of m rows each, noting in particular that the last $m-n$ entries of each are 0.

$$\mathcal{V}(p_1,m) = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \qquad\qquad \mathcal{V}(p_2,m) = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

7.4.14 Observation *The matrix equation of 7.4.12 may be rewritten as*

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{2n-2} \end{pmatrix} = \mathsf{Circ}(\mathcal{V}(p_1, 2n-1) \cdot \mathcal{V}(p_2, 2n-1)$$

□

7.4.15 Lemma *Let $p = \sum_{k=0}^{n}$ be a polynomial of degree at most $n-1$. Then, for any integer $m \geq n$,*

$$\mathsf{M}(\rho_m, m) \cdot \mathsf{Circ}(\mathcal{V}(p,m)) \cdot \frac{1}{m} \cdot \mathsf{M}(\rho_{-m}, m))$$

is the diagonal matrix

$$\mathfrak{F}_m^{\Delta}(p) := \begin{pmatrix} \mathsf{F}_m(p,0) & 0 & 0 & 0 & \cdots & 0 \\ 0 & \mathsf{F}_m(p,1) & 0 & 0 & \cdots & 0 \\ 0 & 0 & \mathsf{F}_m(p,2) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \mathsf{F}_m(p,m-1) \end{pmatrix}$$

PROOF: The $(\ell, j)^{th}$ entry of $\mathsf{M}(\rho_m, m) \cdot \mathsf{Circ}(\mathcal{V}(p,m))$ is

$$\begin{aligned} & \sum_{k=0}^{m-1} \rho_m^{\ell \cdot k} \cdot \bar{a}_{(m+k-j)\mathsf{mod}(m)} \qquad \text{with } \bar{a}_k = \begin{cases} a_k & \text{if } k \leq n-1 \\ 0 & \text{if } k \geq n \end{cases} \\ = \ & \rho_m^{\ell \cdot j} \cdot \sum_{k=0}^{m-1} \rho_m^{\ell \cdot (k-j)} \cdot \bar{a}_{(m+k-j)\mathsf{mod}(m)} \\ = \ & \rho_m^{\ell \cdot j} \cdot \sum_{k=0}^{m-1} \rho_m^{\ell \cdot (k-j)\mathsf{mod}(m)} \cdot \bar{a}_{(m+k-j)\mathsf{mod}(m)} \\ = \ & \rho_m^{\ell \cdot j} \cdot \sum_{k=0}^{m-1} \rho_m^{\ell \cdot (m+k-j)\mathsf{mod}(m)} \cdot \bar{a}_{(m+k-j)\mathsf{mod}(m)} \\ = \ & \rho_m^{\ell \cdot j} \cdot \sum_{k=0}^{m-1} \rho_m^{\ell \cdot k} \cdot \bar{a}_k \ = \ \rho_m^{\ell \cdot j} \cdot \sum_{k=0}^{m-1} \rho_m^{\ell \cdot k} \cdot a_k \ = \ \rho_m^{\ell \cdot j} \cdot \mathsf{F}_m(p, \ell) \end{aligned}$$

Thus,

$$\mathsf{M}(\rho_m, m) \cdot \mathsf{Circ}(\mathcal{V}(p,m)) = \mathfrak{F}_m^{\Delta}(p) \cdot \mathsf{M}(\rho_m, m)$$

Since $1/m \cdot \mathsf{M}(\rho_{-m}, m)$ is the inverse of $\mathsf{M}(\rho_m, m)$, the result follows.
□

- Finally, it is possible to prove the main characterization theorem.

7.4.16 Theorem – translation between convolution and pointwise multiplication *Let*

$$p_1(x) = \sum_{k=0}^{n-1} a_k \cdot x^k \qquad p_2(x) = \sum_{k=0}^{n-1} b_k \cdot x^k$$

be two polynomials of degree at most $n-1$*. Then*

$$p_1 * p_2 = \mathfrak{F}_{2n-1}^{-1}(\mathfrak{F}_{2n-1}(\mathcal{V}(p_1, 2n-1)) \odot \mathfrak{F}_{2n-1}(\mathcal{V}(p_2, 2n-1)))$$

PROOF: Denote the coefficients of $p_1 * p_2$ as follows:

$$p * q = \sum_{k=0}^{2n-2} c_k \cdot x^k$$

The following further notation is also useful.

$$C := \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} \qquad \mathfrak{F}_{2n-1}^{\updownarrow}(p_2) := \begin{pmatrix} \mathsf{F}_{2n-1}(p_2, 0) \\ \mathsf{F}_{2n-1}(p_2, 1) \\ \vdots \\ \mathsf{F}_{2n-1}(p_2, 2n-2) \end{pmatrix}$$

$$H = \mathsf{M}(\rho_{2n-1}, 2n-1) \qquad H^{-1} = \frac{1}{m} \cdot \mathsf{M}(\rho_{-2n+1}, 2n-1)$$

Now

$$
\begin{aligned}
C &= \mathsf{Circ}(\mathcal{V}(p_1, 2n-1)) \cdot \mathcal{V}(p_2, 2n-1) && \text{[by 7.4.14]} \\
&= (H^{-1} \cdot H) \cdot \mathsf{Circ}(\mathcal{V}(p_1, 2n-1)) \cdot (H^{-1} \cdot H) \cdot \mathcal{V}(p_2, 2n-1) \\
&= H^{-1} \cdot (H \cdot \mathsf{Circ}(\mathcal{V}(p_1, 2n-1)) \cdot H^{-1}) \cdot (H \cdot \mathcal{V}(p_2, 2n-1)) \\
&= H^{-1} \cdot \mathfrak{F}^{\Delta}_{2n-1} \cdot (H \cdot \mathcal{V}(p_2, 2n-1)) && \text{[by 7.4.15]} \\
&= H^{-1} \cdot \mathfrak{F}^{\Delta}_{2n-1} \cdot \mathfrak{F}^{\updownarrow}(p_2) && \text{[Just the FFT!]} \\
&= H^{-1} \cdot \begin{pmatrix} \mathsf{F}_{2n-1}(p_1, 0) \cdot \mathsf{F}_{2n-1}(p_2, 0) \\ \mathsf{F}_{2n-1}(p_1, 1) \cdot \mathsf{F}_{2n-1}(p_2, 1) \\ \vdots \\ \mathsf{F}_{2n-1}(p_1, 2n-2) \cdot \mathsf{F}_{2n-1}(p_2, 2n-2) \end{pmatrix} \\
&= \mathfrak{F}^{-1}_{2n-1}(\mathfrak{F}_{2n-1}(\mathcal{V}(p_1, 2n-1)) \odot \mathfrak{F}_{2n-1}(\mathcal{V}(p_2, 2n-1)))
\end{aligned}
$$

□

7.5 The Fast Fourier Transform

7.5.1 Motivation

- For the DFT to be useful in the implementation of polynomial multiplication, it is necessary to have a fast algorithm for computing both the DFT and its inverse.
- The "naïve" algorithm (matrix multiplication) is $\Theta(n^2)$, with $n-1$ the degree of each polynomial.
- This is the same complexity as that of direct polynomial multiplication, and offers no particular advantage for solving such problems.
- The *fast Fourier transform* (*FFT*) is a divide-and-conquer solution for the computation of the DFT and its inverse, and runs in time $\Theta(n \cdot \log(n))$.
- The idea is as follows:

$$p = \sum_{k=0}^{n-1} a_k \cdot x^k \quad = \quad \text{a polynomial}$$

  Define:

$$p_{\mathsf{even}}(x) = \sum_{\substack{k=0 \\ k \text{ even}}}^{n-1} a_k \cdot x^{k/2} \qquad p_{\mathsf{odd}}(x) = \sum_{\substack{k=0 \\ k \text{ odd}}}^{n-1} a_k \cdot x^{(k-1)/2}$$

- Note that

$$p(x) = p_{\mathsf{even}}(x^2) + p_{\mathsf{odd}}(x^2) * x$$

- Use a divide-and-conquer algorithm to compute the DFT's of $p_{\mathsf{even}}(x)$ and $p_{\mathsf{odd}}(x)$, and then to combine the results.

7.5.2 Proposition *Let $p = \sum_{k=0}^{n-1} a_k \cdot x^k$ be a polynomial of degree at most $n-1$, and let m be an even number with $m \geq n$. Then:*

(a) *For all k, $0 < k < m/2$,*

$$\mathsf{F}_m(p,k) = \mathsf{F}_{m/2}(p_{\mathsf{even}},k) + \mathsf{F}_{m/2}(p_{\mathsf{odd}},k) \cdot \rho_m^k$$

(b) *For all k, $m/2 \leq k < m$,*

$$\mathsf{F}_m(p,k) = \mathsf{F}_{m/2}(p_{\mathsf{even}},k) - \mathsf{F}_{m/2}(p_{\mathsf{odd}},k) \cdot \rho_m^k$$

PROOF: First let $k < m/2$. Then

$$\begin{aligned}
\mathsf{F}_{m/2}(p_{\mathsf{even}},k) &= \sum_{\substack{\ell=0 \\ \ell \text{ even}}}^{n-1} a_\ell \cdot \rho_{m/2}^{k\ell/2} = \sum_{\substack{\ell=0 \\ \ell \text{ even}}}^{n-1} a_\ell \cdot \rho_m^{k\ell} \\
\mathsf{F}_{m/2}(p_{\mathsf{odd}},k) &= \sum_{\substack{\ell=0 \\ \ell \text{ odd}}}^{n-1} a_\ell \cdot \rho_{m/2}^{k(\ell-1)/2} = \sum_{\substack{\ell=0 \\ \ell \text{ odd}}}^{n-1} a_\ell \cdot \rho_m^{k(\ell-1)}
\end{aligned}$$

since $\rho_{m/2} = \rho_m^2$. This establishes part (a), since

$$\begin{aligned}
\mathsf{F}_m(p,k) = \sum_{\ell=0}^{n-1} a_\ell \cdot \rho_m^{k\ell} &= \sum_{\substack{\ell=0 \\ \ell \text{ even}}}^{n-1} a_\ell \cdot \rho_m^{k\ell} + \sum_{\substack{\ell=0 \\ \ell \text{ odd}}}^{n-1} a_\ell \cdot \rho_m^{k(\ell-1)} \cdot \rho_m^k \\
&= \mathsf{F}_{m/2}(p_{\mathsf{even}},k) + \mathsf{F}_{m/2}(p_{\mathsf{odd}},k) \cdot \rho_m^k
\end{aligned}$$

The key to establishing part (b) is to note that for $m/2 \leq k < m$,

$$\rho_m^{k-m/2} = \rho_m^k \cdot \rho_m^{-m/2} = \begin{cases} \rho_m^k & \text{if } k \text{ is even} \\ -\rho_m^k & \text{if } k \text{ is odd} \end{cases}$$

since

$$\rho_m^{-m/2} = \begin{cases} 1 & \text{if } m \text{ is even} \\ -1 & \text{if } m \text{ is odd} \end{cases}$$

Thus

$$\begin{aligned}
\mathsf{F}_m(p,k) &= \sum_{\ell=0}^{n-1} a_\ell \cdot \rho_m^{k\ell} = \sum_{\ell=0}^{n-1} a_\ell \cdot \rho_m^{\ell(k-m/2))} \cdot \rho_m^{\ell m/2} \\
&= \sum_{\substack{\ell=0 \\ \ell \text{ even}}}^{n-1} a_\ell \cdot \rho_m^{\ell(k-m/2)} + \sum_{\substack{\ell=0 \\ \ell \text{ odd}}}^{n-1} a_\ell \cdot \rho_m^{\ell(k-m/2)} \\
&= \mathsf{F}_{m/2}(p_{\mathsf{even}},k) - \mathsf{F}_{m/2}(p_{\mathsf{odd}},k) \cdot \rho_m^k
\end{aligned}$$

□

7.5.3 Remarks

- Note that m must be even for the above result to hold, since m/2 must be an integer.
- Since this result will be applied recursively, m must in fact be a power of two.
- This is not a significant restriction, since m can always be chosen to be the smallest power of two which is larger than the sum of the degrees of the two polynomials.

7.5.4 The pseudocode for the FFT

- In the algorithm below, it is assumed that the input polynomial is represented as a vector, as described in 7.4.4 and 7.4.8.
- The intermediate variables *fft_even* and *fft_odd* are also of this type.
- Note that this algorithm must work with complex numbers, even if the coefficients of the input polynomial are real.
- The complexity is clearly $\Theta(n \cdot \log(n))$, with n the size of the input vector.
- This is easily seen by writing down the recurrence relation, which is similar to that for mergesort.

function $\mathit{FFT}(a : \text{ vector}) : \text{ vector};$
  $n \leftarrow \mathit{length}(a)$
  if $(n = 1)$
    then **return** a
    else
      $\langle\ \mathit{fft\_even} \leftarrow \mathit{FFT}(\mathit{even\_part}(a));$
        $\mathit{fft\_odd} \leftarrow \mathit{FFT}(\mathit{odd\_part}(a));$
      $\rangle$
  $\mathit{shift} \leftarrow 1;$
  for $s \leftarrow 0$ **to** $n/2 - 1$ **do**
    $\langle\ \mathit{result}[s] \leftarrow \mathit{fft\_even}[s] + \mathit{fft\_odd}[s] * \mathit{shift};$
      $\mathit{result}[s + n/2] \leftarrow \mathit{fft\_even}[s] - \mathit{fft\_odd}[s] * \mathit{shift};$
      $\mathit{shift} \leftarrow \mathit{shift} * e^{2\pi i/n};$
    $\rangle$
  return $\mathit{result};$

7.5.5 Further notes on the FFT

- $\Theta(n \cdot \log(n))$ algorithms which significantly better constant multipliers are possible.
- Chips and chipsets which perform FFT's are widely used.
  - ➢ Such chips/chipsets are characterized by:
    - ○ number of points (n in the algorithm)
    - ○ resolution per point
  - ➢ The cutting edge: ST/Philips STV0300:
    - ○ Intended use: digital television.
    - ○ Up to 8192 points (2^{13}).
    - ○ Resolution per point is 2×12 bits (24 bits total for a complex value) for up to 2048 points and 2×10 bits for up to 8192 points.
    - ○ Processing time for an 8192-point FFT: 410 μsec.
    - ○ Projected cost in bulk: less than $50.
  - ➢ Typical stock components: 64 to 256 points, at resolutions per point of up to 2×18.
- The FFT algorithm makes it easy to combine such chips for larger n.
- Multidimensional DFT's (and hence FFT's) are important in applications such as image processing.