

Algorithms and Data Structures Fast Fourier Transform

4th and 7th October, 2011

ADS: lects 4 & 5 – slide 1 – 4th and 7th October, 2011

Strassen for general side length n

In Lecture 3, we saw Strassen's algorithm, which multiplies two square $n \times n$ matrices using $\Theta(n^{\lg(7)})$ arithmetic operations when the dimension n is a power-of-2.

Show how to adapt Strassen's algorithm to achieve the exact same Θ expression in the case when n is not a power of 2.

CLASS discussion

ADS: lects 4 & 5 – slide 3 – 4th and 7th October, 2011

Tutorials

- ▶ Start in week 3 (*today*, for those with a Tuesday group);
- ▶ Thanks to all who completed the doodle poll:
<http://doodle.com/wibqm3s2zs7mm4pa>
- ▶ I *hope* we will have an allocation on the ITO database before noon today. If *not* here are some rules:
 - ▶ If you ticked Wednesday as one of your options, then you *definitely* will be allocated to Wednesday.
 - ▶ If you ticked Friday *only* and no other slot, then you *definitely* will be allocated to Friday.
 - ▶ If you ticked Tuesday, or both Tuesday *and* Friday, or *all three possibilities*, then you are in one of the Tuesday groups. Check the course webpage in advance of today's tutorial:
<http://www.inf.ed.ac.uk/teaching/courses/ads/> *If there is no allocation to a specific room yet, come to AT3.03 and we will split you for this week.*

ADS: lects 4 & 5 – slide 2 – 4th and 7th October, 2011

Complex numbers

Any polynomial $p(x)$ of degree d ought to have d roots. (i.e., $p(x) = 0$ should have d solutions.)

But the equation

$$x^2 + 1 = 0 \quad (*)$$

has no solutions at all if we restrict our attention to real numbers.

Introduce a special symbol i to stand for a solution to $(*)$. Then $i^2 = -1$ and $(*)$ has the required two solutions, i and $-i$.

Adding i allows all polynomial equations to be solved! Indeed a polynomial of degree d has d roots (taking account of multiplicities). This is the *Fundamental Theorem of Algebra*.

ADS: lects 4 & 5 – slide 4 – 4th and 7th October, 2011

Roots of Unity

In particular,

$$x^n = 1$$

has n solutions in the complex numbers. They may be written

$$1, \omega_n, \omega_n^2, \dots, \omega_n^{n-1}$$

where ω_n is the **principal n th root of unity**:

$$\omega_n = \cos(2\pi/n) + i \sin(2\pi/n), \quad (\dagger).$$

Convention: from now on ω_n denotes the principal n th root of unity given by (\dagger) .

Note: $e^{iu} = \cos u + i \sin u$ so $\omega_n = e^{2\pi i/n}$.

ADS: lects 4 & 5 – slide 5 – 4th and 7th October, 2011

The Discrete Fourier Transform (DFT)

Instance A sequence of n complex numbers

$$a_0, a_1, a_2, \dots, a_{n-1}.$$

Output The sequence of n complex numbers

$$A(1), A(\omega_n), A(\omega_n^2), \dots, A(\omega_n^{n-1})$$

obtained by evaluating the polynomial

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

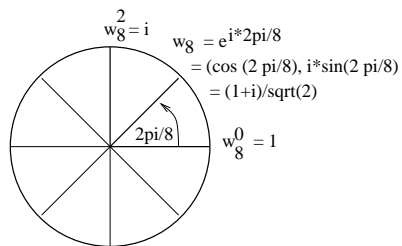
at the n th roots of unity.

The DFT is a *fingerprint* of size n of a polynomial.

CLASS QUESTION: It's not the only fingerprint (why?)

ADS: lects 4 & 5 – slide 7 – 4th and 7th October, 2011

8th Roots of Unity



“Wheel” representation of 8th roots-of-unity (complex plane)).
Same wheel structure for any n (then ω_n found at angle $2\pi/n$).

ADS: lects 4 & 5 – slide 6 – 4th and 7th October, 2011

Motivation for algorithms for DFT/Inverse DFT

Direct. Signal processing: mapping between time and frequency domains.

Indirect. Subroutine in numerous applications, e.g., multiplying polynomials or large integers, cyclic string matching, etc.

It is important, therefore to find the fastest method. There is an obvious $\Theta(n^2)$ algorithm. Can we do better?

YES! Really cool algorithm (Fast Fourier Transform (FFT)) runs in $O(n \lg n)$ time. Published by Cooley & Tukey in 1965 - basics known by Gauss in 1805!

Used in *every* Digital Signal Processing application. Probably the most important algorithm of today. In Lecture 5, we'll show how to apply FFT to get polynomial mult. in $O(n \lg n)$ (not the main application, but cute!).

ADS: lects 4 & 5 – slide 8 – 4th and 7th October, 2011

Divide-and-Conquer: "attempt" 1

We are interested in evaluating:

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}.$$

Assume n even. Put

$$\begin{aligned} A_{\text{small}}(y) &= a_0 + a_1y + \dots + a_{n/2-1}y^{n/2-1}, \\ A_{\text{big}}(y) &= a_{n/2} + a_{n/2+1}y + \dots + a_{n-1}y^{n/2-1} \end{aligned}$$

so that

$$A(x) = A_{\text{small}}(x) + x^{n/2}A_{\text{big}}(x).$$

To evaluate $A(x)$ at the n th roots of unity, we need to evaluate $A_{\text{small}}(y)$ and $A_{\text{big}}(y)$ at the n th roots of unity.

But we are making no progress in reducing the n .

Need relationship between degree of poly. and number of roots

Try again ...

ADS: lects 4 & 5 – slide 9 – 4th and 7th October, 2011

Key Facts

Assuming n is even:

- ▶ $\omega_n^2 = (e^{\frac{2\pi i}{n}})^2 = e^{\frac{2\pi i}{n/2}} = \omega_{n/2}$, and
- ▶ $\omega_n^{n/2} = (e^{\frac{2\pi i}{n}})^{n/2} = e^{\pi i} = -1$.

Thus we have the following relationships between ω_n and $\omega_{n/2}$:

$$\begin{array}{cccccccc} 1 & \omega_n^2 & \dots & \omega_n^{n-2} & \omega_n^n & \omega_n^{n+2} & \dots & \omega_n^{2(n-1)} \\ \parallel & \parallel & \dots & \parallel & \parallel & \parallel & \dots & \parallel \\ 1 & \omega_{n/2} & \dots & \omega_{n/2}^{n/2-1} & 1 & \omega_{n/2} & \dots & \omega_{n/2}^{n/2-1} \end{array}$$

ADS: lects 4 & 5 – slide 11 – 4th and 7th October, 2011

Divide-and-Conquer: attempt 2

We are interested in evaluating:

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1}.$$

Assume n a power of 2. Put

$$\begin{aligned} A_{\text{even}}(y) &= a_0 + a_2y + \dots + a_{n-2}y^{n/2-1}, \\ A_{\text{odd}}(y) &= a_1 + a_3y + \dots + a_{n-1}y^{n/2-1}, \end{aligned}$$

so that

$$A(x) = A_{\text{even}}(x^2) + x A_{\text{odd}}(x^2).$$

To evaluate $A(x)$ at the n th roots of unity, we need to evaluate $A_{\text{even}}(y)$ and $A_{\text{odd}}(y)$ at the points $1, \omega_n^2, \omega_n^4, \dots, \omega_n^{2(n-1)}$.

We'll show now that these are DFTs. (wrt $n/2$)

ADS: lects 4 & 5 – slide 10 – 4th and 7th October, 2011

Key Facts (cont'd)

$$A(1) = A_{\text{even}}(1) + 1 \cdot A_{\text{odd}}(1)$$

$$A(\omega_n) = A_{\text{even}}(\omega_{n/2}) + \omega_n A_{\text{odd}}(\omega_{n/2})$$

$$A(\omega_n^2) = A_{\text{even}}(\omega_{n/2}^2) + \omega_n^2 A_{\text{odd}}(\omega_{n/2}^2)$$

⋮

$$A(\omega_n^{n/2-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) + \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1})$$

ADS: lects 4 & 5 – slide 12 – 4th and 7th October, 2011

Key Facts (cont'd)

$$A(\omega_n^{n/2}) = A_{\text{even}}(1) - 1 \cdot A_{\text{odd}}(1)$$

$$A(\omega_n^{n/2+1}) = A_{\text{even}}(\omega_{n/2}) - \omega_n A_{\text{odd}}(\omega_{n/2})$$

⋮

$$A(\omega_n^{n-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) - \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1})$$

We use this (reln. with $j < n/2$ cases) on lines 7.8. of the Alg.

ADS: lects 4 & 5 – slide 13 – 4th and 7th October, 2011

Implementation

Algorithm FFT_n($\langle a_0, \dots, a_{n-1} \rangle$)

1. **if** $n = 1$ **then return** $\langle a_0 \rangle$
2. $\omega_n \leftarrow e^{2\pi i/n}$
3. $\omega \leftarrow 1$
4. $\langle y_0^{\text{even}}, \dots, y_{n/2-1}^{\text{even}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_0, a_2, \dots, a_{n-2} \rangle)$
5. $\langle y_0^{\text{odd}}, \dots, y_{n/2-1}^{\text{odd}} \rangle \leftarrow \text{FFT}_{n/2}(\langle a_1, a_3, \dots, a_{n-1} \rangle)$
6. **for** $k \leftarrow 0$ **to** $n/2 - 1$ **do**
7. $y_k \leftarrow y_k^{\text{even}} + \omega y_k^{\text{odd}}$
8. $y_{k+n/2} \leftarrow y_k^{\text{even}} - \omega y_k^{\text{odd}}$
9. $\omega \leftarrow \omega \omega_n$
10. **return** $\langle y_0, \dots, y_{n-1} \rangle$

Algorithm assumes that n is a power of 2. If it is not, input sequence is padded by 0s (CLASS discussion).

ADS: lects 4 & 5 – slide 15 – 4th and 7th October, 2011

The Fast Fourier Transform (FFT)

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1},$$

assume n is a power of 2. Compute

$$A(1), A(\omega_n), A(\omega_n^2), \dots, A(\omega_n^{n-1}), \quad (*)$$

as follows:

1. If $n = 1$ then $A(x)$ is a constant so task is trivial. Otherwise split A into A_{even} and A_{odd} .
2. By making two recursive calls compute the values of $A_{\text{even}}(y)$ and $A_{\text{odd}}(y)$ at the $(n/2)$ points $1, \omega_{n/2}, \omega_{n/2}^2, \dots, \omega_{n/2}^{n/2-1}$.
3. Compute the values (*) by using the equation

$$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2).$$

ADS: lects 4 & 5 – slide 14 – 4th and 7th October, 2011

Analysis

$T(n)$ worst-case running time of FFT.

Lines 1–3: $\Theta(1)$

Lines 4–5: $\Theta(1) + 2T(n/2)$

Loop, 6–9: $\Theta(n)$

Line 10: $\Theta(1)$

Yields the following recurrence:

$$T(n) = 2T(n/2) + \Theta(n).$$

Solution:

$$T(n) = \Theta(n \cdot \lg(n)).$$

ADS: lects 4 & 5 – slide 16 – 4th and 7th October, 2011

The Discrete Fourier Transform

Recall

- ▶ The DFT maps a tuple $\langle a_0, \dots, a_{n-1} \rangle$ to the tuple $\langle y_0, \dots, y_{n-1} \rangle$ defined by

$$y_j = \sum_{k=0}^{n-1} a_k \omega_n^{jk},$$

where $\omega_n = e^{2\pi i/n}$ is the principal n th root of unity.

- ▶ Thus for every n (power of 2) we may view DFT_n as mapping $\mathbb{C}^n \rightarrow \mathbb{C}^n$, where \mathbb{C} denote the complex numbers.
- ▶ FFT (the Fast Fourier Transform) is an algorithm computing DFT_n in time

$$\Theta(n \lg(n)).$$

ADS: lects 4 & 5 – slide 17 – 4th and 7th October, 2011

An alternative view on the DFT

DFT_n is the linear mapping described by the matrix

$$V_n = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \dots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}.$$

That is, we have

$$V_n \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

We will NOT *actually perform* the naïve matrix mult. (we will do *much* better: $O(n \lg n)$)

ADS: lects 4 & 5 – slide 19 – 4th and 7th October, 2011

The inverse DFT

$$\begin{aligned} \text{DFT}_n : \mathbb{C}^n &\rightarrow \mathbb{C}^n \\ \langle a_0, \dots, a_{n-1} \rangle &\mapsto \langle y_0, \dots, y_{n-1} \rangle \end{aligned}$$

Question

Can we go back from $\langle y_0, \dots, y_{n-1} \rangle$ to $\langle a_0, \dots, a_{n-1} \rangle$?

More precisely:

1. Is DFT_n invertible, that is, is it one-to-one and onto?
2. If the answer to (1) is 'yes', can we compute DFT_n^{-1} efficiently?

ADS: lects 4 & 5 – slide 18 – 4th and 7th October, 2011

Inverse of DFT

Claim: V_n is a *van-der-Monde* matrix and thus invertible.

Proof: Define the following “Inverse” matrix:

$$V_n^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \dots & \omega_n^{-2(n-1)} \\ 1 & \omega_n^{-3} & \omega_n^{-6} & \dots & \omega_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}.$$

ADS: lects 4 & 5 – slide 20 – 4th and 7th October, 2011

Inverse of DFT (proof)

Verification: We must check that $V_n V_n^{-1} = I_n$:

Want ll -th entry = 1 $\forall l$, and lj -th entry = 0 $\forall l, j$ with $l \neq j$.

Expanding ...

$$\begin{aligned} (V_n V_n^{-1})_{lj} &= \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{\ell k} \omega_n^{-kj} \\ &= \frac{1}{n} \sum_{k=0}^{n-1} \omega_n^{(\ell-j)k}, \\ &= \begin{cases} 1 & \text{if } \ell = j \text{ (because } \omega_n^{\ell-j} = 1) \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

$(V_n V_n^{-1})_{lj} = 0$ case uses the fact that for all $r \neq 0$ ($r = (\ell - j)$)

$$\text{we have } \sum_{k=0}^{n-1} \omega_n^{rk} = 0.$$

ADS: lects 4 & 5 – slide 21 – 4th and 7th October, 2011

Inverse DFT (efficiently)

ω_n^{-1} is an n th root of unity (though not the principal one). Note that

$$(\omega_n^{-1})^j = 1/\omega_n^j = \omega_n^n/\omega_n^j = \omega_n^{n-j},$$

for every $0 \leq j < n$.

Inverse FFT

- Compute $\text{DFT}_n \langle y_0, \dots, y_{n-1} \rangle$ (*deliberately* using DFT_n , not inverse), to obtain the result $\langle d_0, \dots, d_{n-1} \rangle$.
- Flip the sequence d_1, d_2, \dots, d_{n-1} in this result (keeping d_0 fixed), then divide every term by n .

$$a_i = \begin{cases} \frac{d_0}{n} & \text{if } i = 0 \\ \frac{d_{n-i}}{n} & \text{if } 1 \leq i \leq n-1 \end{cases}$$

Worst-case running time is $\Theta(n \lg(n))$.

ADS: lects 4 & 5 – slide 23 – 4th and 7th October, 2011

Inverse of DFT

We have shown DFT_n is invertible with

$$\text{DFT}_n^{-1} : \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} \mapsto V_n^{-1} \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}.$$

Problem

If we were to apply $V_n^{-1} \langle y_0, \dots, y_{n-1} \rangle$ directly in order to recover $\langle a_0, \dots, a_{n-1} \rangle$, the evaluation of $V_n^{-1} \langle y_0, \dots, y_{n-1} \rangle$ would take $\Theta(n^2)$ time!!!

Solution

Take another look back at the V_n^{-1} matrix, and see that it is *more-or-less* a “flipped-over” DFT!

ADS: lects 4 & 5 – slide 22 – 4th and 7th October, 2011

Our Application! Multiplication of Polynomials

$$\begin{aligned} \text{Input: } p(x) &= a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} \\ q(x) &= b_0 + b_1x + b_2x^2 + \dots + b_{m-1}x^{m-1}. \end{aligned}$$

Required output:

$$\begin{aligned} p(x)q(x) &= (a_0b_0) \\ &\quad + (a_0b_1 + a_1b_0)x \\ &\quad + (a_0b_2 + a_1b_1 + a_2b_0)x^2 \\ &\quad \vdots \\ &\quad + (a_{n-2}b_{m-1} + a_{n-1}b_{m-2})x^{n+m-3} \\ &\quad + (a_{n-1}b_{m-1})x^{n+m-2} \end{aligned}$$

Naive method uses $\Theta(nm)$ arithmetic operations

CAN WE DO BETTER?

ADS: lects 4 & 5 – slide 24 – 4th and 7th October, 2011

Interpolation

Theorem

Let $\alpha_0, \dots, \alpha_{n-1} \in \mathbb{C}$ pairwise distinct and $y_0, \dots, y_{n-1} \in \mathbb{C}$.
Then there exists exactly one polynomial $p(X)$ of degree at most $n - 1$ such that for $0 \leq k \leq n - 1$

$$p(\alpha_k) = y_k.$$

- ▶ The sequence

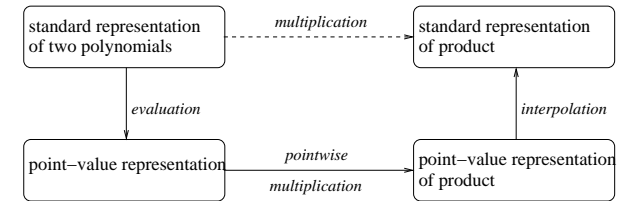
$$\langle (\alpha_0, y_0), \dots, (\alpha_{n-1}, y_{n-1}) \rangle$$

is called a point-value representation of the polynomial p .

- ▶ The process of computing a polynomial from a point-value representation is called **interpolation**.

ADS: lects 4 & 5 – slide 25 – 4th and 7th October, 2011

Multiplication of polynomials (cont'd)



we take the solid-arrow route, using 3 steps, to achieve performance $\Theta(n \lg(n))$.

ADS: lects 4 & 5 – slide 27 – 4th and 7th October, 2011

Multiplication of polynomials (cont'd)

Observation

Suppose we have two polynomials $p(X)$ (of degree $n - 1$) and $q(X)$ (of degree $m - 1$). Assume $\max\{m, n\} = n$. If $\langle (\alpha_0, y_0), \dots, (\alpha_{n+m-2}, y_{n+m-2}) \rangle$ and $\langle (\alpha_0, z_0), \dots, (\alpha_{n+m-2}, z_{n+m-2}) \rangle$ are point-value representations $p(X)$ and $q(X)$ respectively (evaluated at exactly the same points), then

$$\langle (\alpha_0, y_0 z_0), \dots, (\alpha_{n+m-2}, y_{n+m-2} z_{n+m-2}) \rangle$$

is a point-value representation of $p(X)q(X)$ (with enough points to allow us to recover $pq(X)$ by interpolation).

ADS: lects 4 & 5 – slide 26 – 4th and 7th October, 2011

Multiplication of polynomials (cont'd)

Key idea

Let n' be the smallest power of 2 such that $n' \geq n + m - 1$. Use the n' -th roots of unity as the evaluation points: $\alpha_0 = 1, \alpha_1 = \omega_{n'}, \alpha_2 = \omega_{n'}^2, \dots, \alpha_{n'-1} = \omega_{n'}^{n'-1}$.

Then

- ▶ evaluation \equiv DFT, and
- ▶ interpolation \equiv inverse DFT

Overall running time is

$$\begin{aligned} & \Theta(n' \log n') = \Theta(n \log n) && \text{(FFT)} \\ + & \Theta(n') = \Theta(n) && \text{(pointwise multiplication)} \\ + & \Theta(n' \log n') = \Theta(n \log n) && \text{(inverse FFT)} \\ \hline = & \Theta(n \log n) \end{aligned}$$

ADS: lects 4 & 5 – slide 28 – 4th and 7th October, 2011

Digression: Fourier transforms

- ▶ $f : \mathbb{C} \rightarrow \mathbb{C}$ well-behaved. (continuous) **Fourier transform** of F of f defined by

$$F(y) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i y x} dx.$$

Inversion

$$f(x) = \int_{-\infty}^{\infty} F(y) e^{2\pi i y x} dy.$$

- ▶ **Fourier series** for periodic f (period 1 here)

$$f(x) = \sum_{k=-\infty}^{\infty} F_k e^{2\pi i k x}.$$

Idea: Represent f in terms of “basic periodic functions”

$$e^{2\pi i k x} = \cos(2\pi k x) + i \sin(2\pi k x).$$

k is the **frequency**.

- ▶ **DFT** can be viewed as finite approximation.

ADS: lects 4 & 5 – slide 29 – 4th and 7th October, 2011

Example (cont'd)

DFT

y_0	y_1	y_2	y_3	y_4	y_5	\cdots	y_{12}	y_{13}	y_{14}	y_{15}
0	8	0	$-2i$	0	0	\cdots	0	$2i$	0	8

Inverse DFT

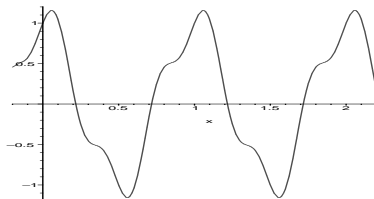
$$\begin{aligned} a_k &= \frac{1}{16} \sum_{j=0}^{15} y_j \omega_{16}^{-jk} \\ &= \frac{1}{16} \left(8\omega_{16}^{-k} - 2i\omega_{16}^{-3k} + 2i\omega_{16}^{-13k} + 8\omega_{16}^{-15k} \right) \\ &= \frac{1}{2} (\omega_{16}^{-k} + \omega_{16}^{-15k}) + \frac{1}{8} i (\omega_{16}^{-13k} - \omega_{16}^{-3k}) \\ &= \frac{1}{2} \cdot 2 \cdot \cos(2\pi k/16) + \frac{1}{8} i \cdot (-2i) \sin(3 \cdot 2\pi k/16) \end{aligned}$$

(Using $\omega_{16}^{-\ell k} = \cos(-2\pi \ell k/16) + i \sin(-2\pi \ell k/16)$ and $\cos(x) = \cos(-x)$, $\sin(x) = -\sin(-x)$.)

ADS: lects 4 & 5 – slide 31 – 4th and 7th October, 2011

Application: Signal Processing

Example: $f(x) = \cos(2\pi x) + \frac{1}{4} \sin(3 \cdot 2\pi x)$



Discretise ($n = 16$ sample points a_0, \dots, a_{15})

$$a_k = f(2\pi k/n) = \cos(2\pi k/n) + \frac{1}{4} \sin(6\pi k/n).$$

ADS: lects 4 & 5 – slide 30 – 4th and 7th October, 2011

Reading Assignment

Fast Fourier Transform, by M. Cryan, notes handed out today.

[CLRS] Section 30.2 (pp. 830-838). *Section 32.2 (pp. 783-791) in [CLR].*

Problems

- Exercise 30.2-2, p. 838 of [CLRS]. *Ex. 32.2-2, p. 790 of [CLR].*
- Let $f(x) = 3 \cos(2x)$. For $0 \leq k \leq 3$, let $a_k = f(2\pi k/4)$. Compute the DFT of $\langle a_0, \dots, a_3 \rangle$.
Do the same for $f(x) = 5 \sin(x)$.
- Exercise 30.2-3, p. 838 of [CLRS]. *Ex. 32.2-3, p. 791 in [CLR].*
- Exercise 30.2-7, p. 838 of [CLRS]. *Ex. 32.2-7, p. 791 in [CLR].*

ADS: lects 4 & 5 – slide 32 – 4th and 7th October, 2011