Graded Homework 2 Foundations of Computational Math 2 Spring 2021

The solutions are due by 11:59PM on Monday 22 February 2021

Programming Exercise

Objectives

- To review the basic primitive ideas of the Discrete Fourier Transformation.
- To understand and implement basic data structure for the Discrete Fourier Transformation and Fast Fourier Transformation.
- To empirically explore the basic properties of the DFT and reconstruction error.
- To empirically demonstrate the computational time advantage of the FFT.

Review of Basic Forms

The discrete Fourier transform, denoted DFT, is a complex linear transformation on a vector in $\mathbb{C}^n$ and can be defined in terms of a matrix $F_n \in \mathbb{C}^{n\times n}$ given by

$$F_n = \frac{1}{\sqrt{n}}\bar{\Phi},$$

where $\theta = 2\pi/n$, $\omega = e^{i\theta}$,

$$\Phi = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix}. \tag{1}$$

and $\overline{\Phi}$ denotes the matrix formed by taking the elementwise complex conjugate. (Other versions with different scaling and permuted columns corresponding to alternate orderings of the basis functions used are possible, as we have discussed.)

Recall that the matrix, F_n^{-1}, defining the inverse DFT, denoted IDFT, can be found from the fact that F is a unitary matrix, i.e., $\frac{1}{n}\Phi^H\Phi = I \in \mathbb{C}^{n\times n}$ and by noticing $\Phi^H = \bar{\Phi}$ and therefore

$$F_n^{-1} = \frac{1}{\sqrt{n}}\Phi = F_n^H. \tag{2}$$

Additionally, since F_n and F_n^H are unitary matrices, they preserve the vector that preserves vector 2-norm, $\|F_n x\|_2 = \|x\|_2$ and $\|F_n^H x\|_2 = \|x\|_2$.

When $n = 2^k, k \in \mathbb{N}$ is power of 2, one version of fast Fourier transformation, FFT, can be derived by the factorization of F_n. Let $n = 8$, for example, we have For $n = 8$

$$F_8 f^{(8)} = \frac{1}{\sqrt{2}} \begin{bmatrix} I_4 & \Omega_4 \\ I_4 & -\Omega_4 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} I_2 & \Omega_2 & 0 & 0 \\ I_2 & -\Omega_2 & 0 & 0 \\ 0 & 0 & I_2 & \Omega_2 \\ 0 & 0 & I_2 & -\Omega_2 \end{bmatrix} \begin{bmatrix} F_2 & 0 & 0 & 0 \\ 0 & F_2 & 0 & 0 \\ 0 & 0 & F_2 & 0 \\ 0 & 0 & 0 & F_2 \end{bmatrix} P_8 f^{(8)}$$

where $\theta_n = 2\pi/n$, $\omega_n = e^{i\theta_n}$, $\mu_n = \bar{\omega}_n$, $F_2 \in \mathbb{C}^{2\times 2}$ is the DFT of vectors of length 2, $\Omega_2 = diag(1, \mu_4)$, $\Omega_4 = diag(1, \mu_8, \mu_8^2, \mu_8^3)$ and P_8 is a permutation matrix defining the bit-reversal permutation.

In general, when $n = 2^k$, we can write down the factorization as

$$F_n = (A_1 A_2 \cdots A_{k-1}) D_n P_n = \left(\prod_{i=1}^{k-1} A_i \right) D_n P_n \tag{3}$$

where each A_i is scaled by $1/\sqrt{2}$ and has the block structure using I and Ω of the appropriate dimensions, P_n is the bit-reversal permutation matrix and $D_n = diag(F_2, \cdots, F_2)$ is a block diagonal matrix with $n/2$, 2×2 DFT matrices.

Codes

Implement the following algorithms:

- Discrete Fourier Transformation (DFT).
- Inverse Discrete Fourier Transformation (IDFT).
- Fast Fourier Transformation (FFT).
- Inverse Fast Fourier Transformation (IFFT).

Comments

- Each algorithm should have an input/output structure that allows for the use of complex numbers. Therefore, you are free to use a complex library or implement your own.
- Storage and computational complexity for DFT should be $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ respectively, i.e. you should not explicitly form F_n, $\bar{\Phi}$ or Φ in (1) as matrices.
- The DFT and IDFT code should be implemented in the same routine such that a simple change to a single parameter can define which transform is performed and there should be no restriction on n, i.e., do not assume $n = 2^k$.

- Storage and computational FFT should be $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ respectively. You may use either the factored form or the recusive form.
- The FFT and IFFT code should be implemented in the same routine such that a simple change to a single parameter can define which transform is performed and you should assume $n = 2^k$ for some k.

DFT/FFT Tasks

1. Describe how you achieved the storage and computational complexity required. Make sure to include the efficient production of the elements of the sequence of the matrices Ω_j. Indicate any dynamic storage that is used, e.g., stack and heap storage allocated (and possibly deallocated) during runtime. (This can be especially important if you use recursive subroutine calls.)
2. Given $F_n = \frac{1}{\sqrt{n}}\bar{\Phi}$ and $F_n^H = \frac{1}{\sqrt{n}}\Phi$, empirically validate the listed properties for both the DFT/IDFT and FFT/IFFT pairs. **Note that since you do not form the matrices explicitly this empirical validation must be done based on output vectors observed given particular input vectors plus in some cases additional computation. You should used appropriate deterministic and random choices of input vectors; means, variances and histograms to summarize your evidence and to support your arguments.** The properties are:
   - (isometry) Preservation of norms e.g. $||F_n x||_2 = ||x||_2$.
   - (unitary) $F_n^H F_n = F_n F_n^H = I$.
   - (reconstruction) Given a randomly generated $x \in \mathbb{C}^n$ show $IDFT(DFT(x)) = x$ and investigate aliasing with the IDFT/DFT pair or the IFFT/FFT pair.
3. Write a routine that allows for an efficient comparison of the computational time and complexity of the DFT versus the FFT. You can consider the ratio of the times in your presentation and the ratio of the computational rates, i.e., operations per unit time. You should use a range of n values that make the clear case of the computational benefits of the FFT.
4. Let $C_n \in \mathbb{C}^{n \times n}$ be a circulant matrix of order $n = 2^k$ defined by its first row $a \in \mathbb{C}^n$, $a = \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \end{pmatrix}$. For example, for $n = 4$ and using the first row as the parameters we have

$$C_4 = \begin{pmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_3 & \alpha_0 & \alpha_1 & \alpha_2 \\ \alpha_2 & \alpha_3 & \alpha_0 & \alpha_1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \alpha_0 \end{pmatrix}$$

   Use the DFT/IDFT or FFT/IFFT pair to implement a routine that:

(a) finds all n eigenvalues, λ_j, $0 \leq j \leq n-1$, of C_n;

(b) computes the matrix 2-norm $\|C_n\|_2$; (The matrix 2-norm of any matrix, M, is equal to $\sqrt{\gamma}$ where γ is the eigenvalue of $M^H M$ with the largest magnitude.);

(c) solves the linear system $C_n x = b$. (Note that you must be able to generate non-singular circulant matrices or have your routine check if the matrix is sufficiently nonsingular.)

Other Test Problems

After you have submitted your solutions make an appointment with the TA Tejas Natu. He will ask you to demonstrate your code on some test problems as a final evaluation of your codes correctness.

You will be asked to perform DFT/FFT on samples generated from complex-valued functions on $[0, 2\pi]$, with different sampling sizes. He will provide subroutines implementing the function. The template zip file posted on the class website contains more information on how to evaluate those functions and some example code. (Read "README.txt" first). The questions you are asked will include such things as reconstruction error and the distribution of frequency components.

Some Additional Instructions

Time a routine

To time a routine in c++, include the header

#include <time.h>

Create 2 timestamps (clock_t type) and record the clock before and after the routine you want to time. Then the difference over clocks_per_second gives you the time. Use wall-clock time for your analysis.

Note that in order to make sure the routine lasts long enough to amortize the overhead of timing, you may generate the data for the DFT/FFT routines, then time a loop that executes the routine multiple times. Also note you should execute the routine once outside and just before the timed loop to reduce the misleading effects of virtual memory. See the example in the posted template.

Remote connection to department machines

All are expected to time your DFT/FFT routine on the same department machine, compute2. Here are the instructions on how to connect to the machine via ssh. You are assumed to know basic command-line operation on terminal on Mac and Linux or CMD on Windows.

1. Open terminal/CMD.

2. Connect to department's main server.

   ssh your-math-account@henri.math.fsu.edu

   Your password to math account will be asked here. (You may be able to connect directly compute2.)

3. Connect to particular machine, e.g., connect2 below.

   ssh compute2

   Your math password will be asked again and once you are on compute2, you can use command-line operations to compile and exectute code.

Written Exercises

Problem 2.2

If $A \in \mathbb{C}^{n_1 \times n_2}$ and $B \in \mathbb{C}^{n_3 \times n_4}$ then the Kronecker product

$$M = A \otimes B \in \mathbb{C}^{n_1 n_3 \times n_2 n_4}$$

is defined in terms of blocks $M_{ij} \in \mathbb{C}^{n_3 \times n_4}$ for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$ where

$$M_{ij} = \alpha_{ij} B.$$

The Kronecker product is useful for expressing many structured matrix expressions, e.g., the Cooley-Tukey FFT/IFFT.

Let $A \in \mathbb{C}^{m \times m}$, $B \in \mathbb{C}^{n \times n}$, $x \in \mathbb{C}^{mn}$, and $y \in \mathbb{C}^{mn}$.

2.2.a. Describe an algorithm to evaluate the matrix vector product

$$y = (A \otimes B)x$$

i.e., given A, B, x determine y.

2.2.b. What is the complexity of the algorithm?

2.2.c. How does the complexity of the algorithm compare to the standard matrix-vector product computation, $y = Mx$, that ignores the structure of M.

Problem 2.3

The factored form of the Cooley-Tukey FFT

$$F_n = (A_1 A_2 \cdots A_{k-1})\, D_n P_n = \left(\prod_{i=1}^{k-1} A_i \right) D_n P_n, \tag{4}$$

where each A_i is scaled by $1/\sqrt{2}$ and has the block structure using I and Ω of the appropriate dimensions, P_n is the bit-reversal permutation matrix and $D_n = diag(F_2, \cdots, F_2)$ is a block diagonal matrix with $n/2$, 2×2 DFT matrices, was derived in the class notes by using the basic properties of the n roots of unity and writing a polynomial in the monomial basis in terms of the sum of the polynomials involving the even and odd power terms.

Given the relationship between $\omega_n = e^{i\theta_n}$ and $\mu_n = \bar{\omega}_n$, with $\theta_n = 2\pi/n$, the same proof can be repeated with μ_n replaced by ω_n to derive the IFFT as the factorization

$$F_n^H = \left(\overline{A}_1 \overline{A}_2 \cdots \overline{A}_{k-1}\right) \overline{D}_n P_n = \left(\prod_{i=1}^{k-1} \overline{A}_i \right) \overline{D}_n P_n, \tag{5}$$

where $\overline{M}$ replaces elements with their complex conjugates. This is equivalent to the factored form of the FFT with μ replaced by ω.

Recall the basic properties of the matrices F and F^H:

$$F = (F)^T, \quad F^H = \left(F^H\right)^T$$

$$F^H F = I = F F^H \rightarrow F^H = F^{-1}.$$

Show that these properties can be used to derive (5) directly from (4).

Problem 2.4

Consider an $n \times n$ nonsingular upper triangular matrix where all nonzero elements in row i are equal to α_i, e.g., for $n = 4$

$$U_4 = \begin{pmatrix} \alpha_1 & \alpha_1 & \alpha_1 & \alpha_1 \\ & \alpha_2 & \alpha_2 & \alpha_2 \\ & & \alpha_3 & \alpha_3 \\ & & & \alpha_4 \end{pmatrix}$$

Assume that $\alpha_i \neq 0$ and show that the system $U_n x = b$ can be solved in significantly fewer than $n^2 + O(n)$ computations. Give your complexity result in the form $Cn^k + O(n^{k-1})$ where C is a constant independent of n and $k > 0$.

Problem 2.5

Define the elementary matrix $N(y, k) = I - y e_k^T \in \mathbb{R}^{n \times n}$, where $1 \leq k \leq n$ is an integer, $y \in \mathbb{R}^n$ and $e_k \in \mathbb{R}^n$ is the k-th standard basis vector. $N(y, k)$ is a Gauss-Jordan transform if it is defined by requiring $N(y, k)v = e_k$ for a given vector $v \in \mathbb{R}^n$.

Perform all of the basic analyses on the Gauss-Jordan transform that were performed on the Gauss transform and elementary unit lower triangular matrices, i.e., existence, inverse, etc., and use the results to show that the Gauss-Jordan algorithm that computes A^{-1} and x from a nonsingular matrix $A \in \mathbb{R}^{n \times n}$ and vector $b \in \mathbb{R}^n$ can be expressed in terms of Gauss-Jordan transforms. Identify the condition for each Gauss-Jordan transformation to exist on each step of the elimination, i.e., determine when the algorithm fails in terms of the transformation's construction.