MATH 603 - Final Assignment

Aiden Taylor

University of Calgary
Department of Mathematics & Statistics
Calgary, AB, Canada

December 5th, 2024

The Problems

1. Write a computer program to implement the Fast Fourier Transform (FFT).
2. Using the FFT, write a computer program to solve numerically the initial-value problem (IVP) for the heat equation,

$$\begin{cases} u_t = u_{xx} & (t, x) \in [0, T] \times [0, L] \\ u(0, x) = u_0(x) & x \in [0, L] \end{cases}.$$

Outline

Continuous Fourier Transform

The Fourier Transform of some function $f(x)$ is given as

$$F(\omega) = \hat{f}(\omega) = \mathcal{F}[f](\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-i\omega x}dx,$$

where the function can be recovered as

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega x}d\omega.$$

Discretization

Given $n \in \mathbb{N}$,

$$\begin{cases} \omega_m = 2\pi m/n, & m = 0, 1, \ldots, n-1, \\ x_k = x_0 + k\Delta x, & k = 0, 1, \ldots, n-1, \end{cases}$$

where $x_0 = 0$ and $\Delta x = L/(n-1)$.

DFT and IDFT

Letting $f_k = f(x_k)$, the DFT and Inverse DFT (IDFT) are given as

$$f_m^{\#} = \sum_{k=0}^{n-1} f_k e^{-i\omega_m k}, \quad m = 0, 1, \ldots, n-1,$$

and

$$f_k = \frac{1}{n} \sum_{m=0}^{n-1} f_m^{\#} e^{i\omega_m k}, \quad k = 0, 1, \ldots, n-1.$$

Matrix Form

Letting $\xi = e^{i2\pi/n}$, the DFT and IDFT can be written as

$$\begin{bmatrix} f_0^{\#} \\ f_1^{\#} \\ f_2^{\#} \\ \vdots \\ f_{n-1}^{\#} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \xi^{-1} & \xi^{-2} & \dots & \xi^{-(n-1)} \\ 1 & \xi^{-2} & \xi^{-4} & \dots & \xi^{-2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \xi^{-(n-1)} & \xi^{-2(n-1)} & \dots & \xi^{-(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{bmatrix},$$

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{n-1} \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \xi^{1} & \xi^{2} & \dots & \xi^{(n-1)} \\ 1 & \xi^{2} & \xi^{4} & \dots & \xi^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \xi^{(n-1)} & \xi^{2(n-1)} & \dots & \xi^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} f_0^{\#} \\ f_1^{\#} \\ f_2^{\#} \\ \vdots \\ f_{n-1}^{\#} \end{bmatrix},$$

where the computational complexity for each is $\mathcal{O}(n^2)$.

Outline

Problem 1

- Discrete Fourier Transform
- Fast Fourier Transform

Problem 2

- Solving Numerically
- Results

Redundancies in the DFT

If $n = 4$, then

$$\begin{cases} f_0^{\#} &= f_0\xi^0 + f_1\xi^0 + f_2\xi^0 + f_3\xi^0 \\ f_1^{\#} &= f_0\xi^0 + f_1\xi^{-1} + f_2\xi^{-2} + f_3\xi^{-3} \\ f_2^{\#} &= f_0\xi^0 + f_1\xi^{-2} + f_2\xi^{-4} + f_3\xi^{-6} \\ f_3^{\#} &= f_0\xi^0 + f_1\xi^{-3} + f_2\xi^{-6} + f_3\xi^{-9}. \end{cases}$$

Redundancies in the DFT

Noticing that $\xi^0 = \xi^{-4} = 1$, $\xi^{-2} = \xi^{-6} = -1$, $\xi^{-1} = \xi^{-9} = -i$, and $\xi^{-3} = i$, then

$$\begin{cases} f_0^{\#} & = (f_0 + f_1) + \xi^0 (f_2 + f_3) \\ f_1^{\#} & = (f_0 - f_1) + \xi^{-1} (f_2 - f_3) \\ f_2^{\#} & = (f_0 + f_1) + \xi^{-2} (f_2 + f_3) \\ f_3^{\#} & = (f_0 - f_1) + \xi^{-3} (f_2 - f_3). \end{cases}$$

Cooley-Tukey Algorithm

If $n = 2^\ell$ where $\ell \in \mathbb{Z}^+$, then

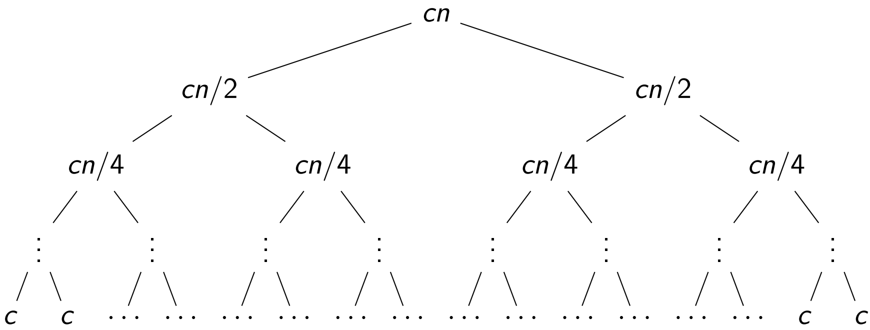
$$f_m^\# = \sum_{k=0}^{n-1} f_k \xi^{-mk} = \sum_{k=0}^{\frac{n}{2}-1} f_{2k} \xi^{-m(2k)} + \sum_{k=0}^{\frac{n}{2}-1} f_{2k+1} \xi^{-m(2k+1)}$$

$$= \sum_{k=0}^{\frac{n}{2}-1} f_{2k} \xi^{-2mk} + \xi^{-m} \sum_{k=0}^{\frac{n}{2}-1} f_{2k+1} \xi^{-2mk},$$

for $m = 0, 1, \ldots, n-1$.

Computation Complexity

The computational complexity of the Cooley-Tukey algorithm is $\mathcal{O}(n \log_2 n)$, which can be illustrated by the following binary tree,



Height of the Binary Tree

At the bottom level we know that we should only need

$$cn/2^i = \mathcal{O}(1) = k$$

floating point operations for each node, and taking the logarithm of both sides gives us

$$\begin{aligned}
&\log_2 (cn/2^i) = \log_2 k \\
&\Rightarrow \log_2 cn - \log_2 2^i = \log_2 k \\
&\Rightarrow \log_2 n + \log_2 c - i \log_2 2 = \log_2 k \\
&\Rightarrow \log_2 n + \log_2 c - \log_2 k = i,
\end{aligned}$$

which results in a height of $\mathcal{O}(\log_2 n)$ and an over all complexity of $\mathcal{O}(n \log_2 n)$.

The Algorithm

Algorithm 1 Fast Fourier Transform

Procedure FFT(f, $f^{\#}$, n, ξ)
if $n \leftarrow 1$ **then**
 $f^{\#}[0] \leftarrow f[0]$
else
 $f_e[k] \leftarrow$ empty array of size $\frac{n}{2}$
 $f_o[k] \leftarrow$ empty array of size $\frac{n}{2}$
 for k from 0 to $\frac{n}{2} - 1$ **do**
  $f_e[k] \leftarrow f[2k]$
  $f_o[k] \leftarrow f[2k+1]$
 end for
 $f_e^{\#}[k] \leftarrow$ empty array of size $\frac{n}{2}$
 $f_o^{\#}[k] \leftarrow$ empty array of size $\frac{n}{2}$
 FFT(f_e, $f_e^{\#}$, $\frac{n}{2}$, ξ^2)
 FFT(f_o, $f_o^{\#}$, $\frac{n}{2}$, ξ^2)
 for k from 0 to $n - 1$ **do**
  $f[k] \leftarrow f_e^{\#}[k \mod \frac{n}{2}] + \xi^k f_o^{\#}[k \mod \frac{n}{2}]$
 end for
end if
End Procedure

Outline

The IVP

Revisting Problem 2, our goal is to solve numerically the IVP,

$$\begin{cases} u_t = u_{xx} & (t, x) \in [0, T] \times [0, L] \\ u(0, x) = u_0(x) & x \in [0, L] \end{cases},$$

using the FFT.

Fourier Transform of the Heat Equation

The Fourier Transform of u and its derivatives are

- $u(t,x) \overset{\mathcal{F}}{\Longrightarrow} \hat{u}(t,\omega)$
- $u_t \overset{\mathcal{F}}{\Longrightarrow} \frac{d}{dt}\hat{u}(t,\omega)$
- $u_{xx} \overset{\mathcal{F}}{\Longrightarrow} -\omega^2\hat{u}(t,\omega)$

which gives us

$$\frac{d}{dt}\hat{u}(t,\omega) = -\omega^2\hat{u}(t,\omega),$$

which has the solution

$$\hat{u}(t,\omega) = \hat{u}_0 e^{-\omega^2 t}.$$

Discretization

Choosing $n = 2^\ell$, where $\ell \in \mathbb{Z}^+$, and taking

$$\begin{cases} \omega_m = 2\pi m/n, & m = 0, 1, \ldots, n-1, \\ x_k = x_0 + k\Delta x, & k = 0, 1, \ldots, n-1, \end{cases}$$

where $x_0 = 0$ and $\Delta x = L/(n-1)$, gives us

$$\begin{bmatrix} \frac{d}{dt} u^{\#}(t, \omega_0) \\ \vdots \\ \frac{d}{dt} u^{\#}(t, \omega_{n-1}) \end{bmatrix} = \begin{bmatrix} -\omega_0^2 u^{\#}(t, \omega_0) \\ \vdots \\ -\omega_{n-1}^2 u^{\#}(t, \omega_{n-1}) \end{bmatrix},$$

and

$$\begin{bmatrix} u^{\#}(t, \omega_0) \\ \vdots \\ u^{\#}(t, \omega_{n-1}) \end{bmatrix} = \begin{bmatrix} u_0^{\#}(t, \omega_0) e^{-\omega_0^2 t} \\ \vdots \\ u_0^{\#}(t, \omega_{n-1}) e^{-\omega_{n-1}^2 t} \end{bmatrix}.$$

Outline

Example IVP

Consider,

$$\begin{cases} u_t = u_{xx} & (t,x) \in [0,100] \times [0,1] \\ u(0,x) = u_0(x) & x \in [0,1] \end{cases}$$

where

$$u_0 = \begin{cases} 1 & x \in [0.3725, 0.6235] \\ 0 & \text{otherwise} \end{cases}$$

and $n = 256$.

Initial-value Function

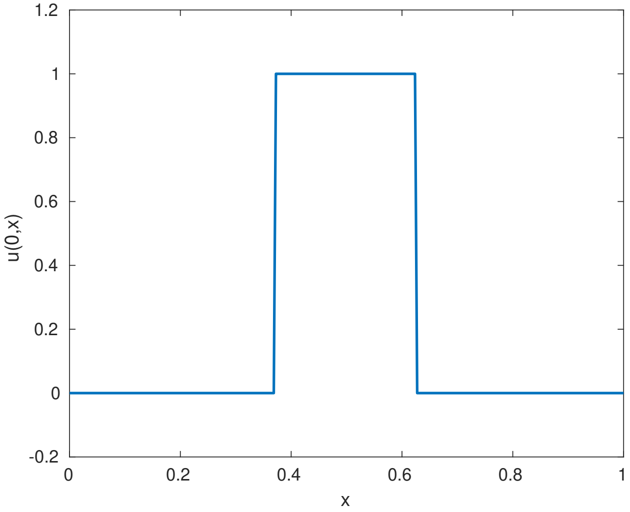


Figure: Plot of $u_0(x)$.

2-dimensional Plot

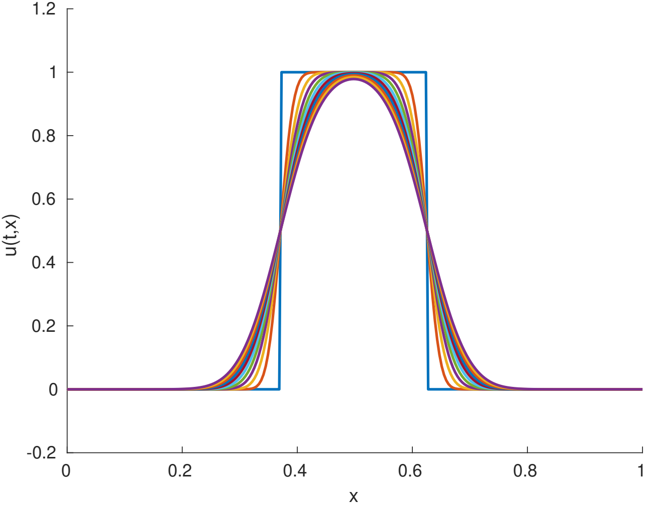


Figure: 2-dimensional plot of the numerical solution to the IVP.

3-dimensional Plot

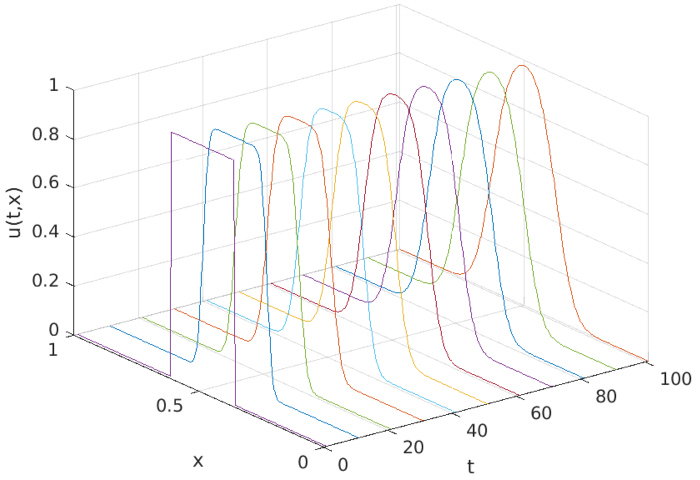


Figure: 3-dimensional plot of the numerical solution to the IVP.

https://github.com/Aidenwjt/MATH-603

Thank you!