Imperial College London

Computational Physics: Fourier Transforms

Blackett Laboratory

16 October 2018

Today: Outline

- Introduction
- Review of Fourier Transforms (very briefly)
- Discrete Fourier Transforms
- Sampling and Aliasing
- Fast Fourier Transforms
- An Introduction to Pseudocode

Fourier Series and Transforms

Introduction

Analytically:

- As you know: represent "any" function as a sum of others
- Sinusoidal functions which form a complete basis set
- Transform into the "frequency domain"
- The coefficients of the sum carry the information
- In the limit of infinitely "close" coefficients, transform into functions
- Appears directly in numerous places in physics

Fourier Series and Transforms

Introduction

In a computer:

- Sample the original function at regular, discrete points
- Perhaps the result of solving a differential equation
- Or it could be data from an experiment

- Perform a **Discrete Fourier Transform (DFT)**
- DFTs can also be used to as a computational tool, e.g., to solve differential equations
- Many applications:
  - ...
- The "Fast Fourier Transform" is a particularly efficient implementation

Fourier Series and Transforms

Recap and notation

- Continuous Fourier Transform of $f(t)$:

$$\begin{aligned} \tilde{f}(\omega) &= \int_{-\infty}^{+\infty} e^{i\omega t} f(t)\,\mathrm{d}t = \mathcal{F}(f(t)), \\ f(t) &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-i\omega t}\tilde{f}(\omega)\,\mathrm{d}\omega = \mathcal{F}^{-1}(\tilde{f}(\omega)), \end{aligned}$$

  for time t and angular frequency $\omega = 2\pi\nu$ (where ν is frequency)
- Time t and ω are reciprocal 'coordinates' or variables
- $\tilde{f}(\omega)$ is the (angular) frequency spectrum of the function $f(t)$
- The "$\sim$" has nothing to do with the same symbol used elsewhere in the course!
- An expansion over plane waves $\exp(-i\omega t)$ which constitute an orthonormal basis
- $\tilde{f}(\omega)$ are the "coefficients" of each component wave of angular frequency ω

Fourier Series and Transforms

Recap: Derivatives

FTs can be very useful in manipulating differential equations

- Spatial or time derivatives become algebraic operations in Fourier space

$$\frac{\mathrm{d}}{\mathrm{d}x} u(x) = v(x)$$

- Taking the FT of this equation yields

$$-ik\tilde{u}(k) = \tilde{v}(k)$$

- The proof is not complicated:

$$\frac{\mathrm{d}}{\mathrm{d}x} \left(\frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ikx} \tilde{u}(k) \, \mathrm{d}k \right) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-ikx} \tilde{v}(k) \, \mathrm{d}k$$

  and so on...

This can be generalised to higher derivatives and higher dimensions

Discrete Fourier Transforms

Introduction

A Continuous Periodic Function

The Fourier Series

The Complex Fourier Series

The Fourier Transform

A Discrete Periodic Function

Discrete Fourier Transforms

Introduction

DFTs are the equivalent of complex Fourier series, which are defined on a finite length domain, but for a **discretely sampled function** rather than a continuous function

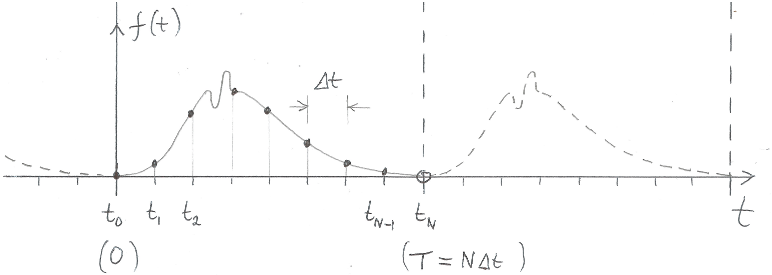
- Using time and angular frequency here
- **Time domain**—We assume the function is sampled by N equally-
- A time domain of length $T = N\Delta t$

$$f_n \equiv f(t_n) \text{ with } n = 0, 1, 2, ..., N-1 \text{ and } \; t_n = n\,\Delta t$$

- The function $f(t)$ is ***assumed to be periodically extended*** beyond the domain $0 \leq t \leq T$ so that $f(t + mT) = f(t)$ for integer m
- For the sampled function, periodicity means $f_N = f_0$; hence we choose to discard f_N in our sequence of samples

Discrete Fourier Transforms

Introduction



Discrete Fourier Transforms

Introduction

- **Frequency domain**—discrete spectrum
- **Longest wave that can fit exactly into the time domain has an angular frequency $\omega_{min} =$**
- **Shortest wave that can be *recognised* by the grid and fits periodically into the time domain has duration**

  so that $\omega_{max} =$
- This maximum frequency is known as the **Nyquist frequency**

$$\omega_{max} = \frac{\pi}{\Delta t} = \Delta\omega \; \frac{N}{2}$$

- These frequencies define the discrete, finite, angular frequency grid on which $\tilde{f}$ is sampled

$$\tilde{f}_p \approx \frac{1}{\Delta t}\tilde{f}(\omega_p) \text{ with } p = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2} \text{ and } \; \omega_p = p\,\Delta\omega = p\frac{2\pi}{N\,\Delta t}$$

Discrete Fourier Transforms

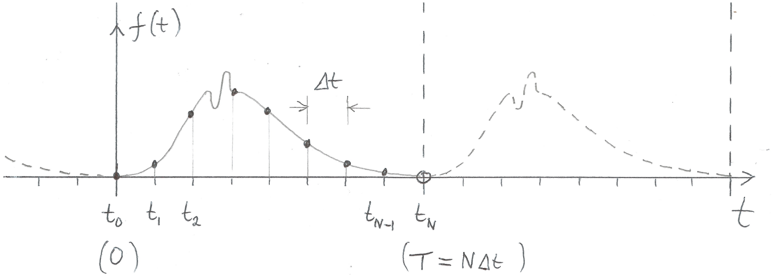
Introduction

- Each discrete frequency corresponds to a wave that fits exactly an integer number of times "p" into the finite domain
- N degrees of freedom ($\tilde{f}_{-N/2} \equiv \tilde{f}_{N/2}$)
- **Why "$\approx$" instead of "$=$" in "$\tilde{f}_p \approx \frac{1}{\Delta t}\tilde{f}(\omega_p)$"?**

- "Aliasing" can cause the DFT to be distorted
  - will come back to this
- Will see in a minute where the Δt comes from

Discrete Fourier Transforms

Introduction



Discrete Fourier Transforms

Definition

The DFT—the analogue of a continuous FT, but with the continuous integral $\int \ldots dt$ replaced by a finite sum $\sum \ldots \Delta t$:

$$\tilde{f}_p = \sum_{n=0}^{N-1} f_n e^{i\omega_p t_n} = \sum_{n=0}^{N-1} f_n e^{i2\pi pn/N},$$

where $\omega_p t_n = \left(\frac{2\pi p}{N\Delta t}\right)(n\Delta t)$ has been used for the second form

- The backwards transform is similarly defined as:

$$f_n = \frac{1}{N} \sum_{p=-N/2+1}^{N/2} \tilde{f}_p e^{-i2\pi pn/N}$$

- The different normalisation factor of $1/N$ accounts for the discrete sampling (from $dt \to \Delta t$ and $d\omega \to \Delta\omega = 2\pi/(N\Delta t)$)
- When going from the FT to the DFT, a factor of Δt is absorbed into $\tilde{f}_p$, so $\tilde{f}_p \approx \tilde{f}(\omega_p)/\Delta t$ Limit of continuously-defined points ($N \to \infty$), and limit of infinite range

Discrete Fourier Transforms

- The backwards transform

$$f_n = \frac{1}{N} \sum_{p=-N/2+1}^{N/2} \tilde{f}_p e^{-i2\pi pn/N}$$

is often written as

$$f_n = \frac{1}{N} \sum_{p=0}^{N-1} \tilde{f}_p \, e^{-i\,2\pi pn/N}$$

  - Aliasing also explains how this is possible
- This is a lossless transform (up to rounding errors etc.)

Discrete Fourier Transforms

Code Snippets

```python
# Set up basic step sizes
N = 100
deltat = 1.0/N # increment from 0 to 1

# the DFT function of p
def ftilde(p):
    sum = 0.0j
    for n in range(N):
        sum += f(n*deltat)*np.exp(1j*2.0*pi*p*n/N)
    return sum

# the inverse DFT
def finverse(n):
    sum = 0.0j
    for p in range(0, N):
        sum += ftilde(p)*np.exp(-1j*2.0*pi*p*n/N)
    return sum/N

# ”Simple” function
def f(x):
    val = -100*x**2+100*x**4 # simple
    # val = -100*x**2+100*x**4+10*sin(x*100) # high-freq
    if(x>0.4 and x<0.6):
        return val*10*abs(x-0.5) # minor blip in the middle
        #return val
    return val

# plot f(t)
tarray = np.zeros(N)
farray = np.zeros(N)
for i in range(0, N):
    tarray[i] = i*deltat
    farray[i] = f(i*deltat)
plt.clf()
plt.plot(tarray,farray,’.-’,label=”Discrete f(t)”,alpha=0.2)
plt.legend()
```

```python
# plot f(x) at high resolution
largeN = N *20
finedeltat = deltat / 20.0
tarray2 = np.zeros(largeN)
farray2 = np.zeros(largeN)
for i in range(0, largeN):
    tarray2[i] = i*finedeltat
    farray2[i] = f(i*finedeltat)
plt.plot(tarray2,farray2,’.--’,label=”Original f(t)”,
         alpha=0.2)
plt.legend()

# plot ftilde (both Real and Imaginary parts)
parray = np.arange(0,N)
ftildearrayR = np.zeros(parray.size)
for p in range(0, parray.size):
    ftildearrayR[p] = np.real(ftilde(p))
ftildearrayI = np.zeros(parray.size)
for p in range(0, parray.size):
    ftildearrayI[p] = np.imag(ftilde(p))

plt.clf()
plt.plot(parray, ftildearrayR,’.-’,label=”ftilde Real”,
         alpha=0.2)
plt.legend()
plt.plot(parray, ftildearrayI,’.--’,label=”ftilde Imag”,
         alpha=0.6)
plt.legend()
```

Discrete Fourier Transforms

Relation to Complex Fourier Series

Complex Fourier series represent the discrete spectrum of a ***continuous function*** $f(t)$, also on a domain of finite length

$$c_k = \frac{1}{T}\int_0^T f(t)\, e^{i\,\omega_k t}\,\mathrm{d}t\,, \quad f(t) = \sum_{k=-\infty}^{+\infty} c_k\, e^{-i\,\omega_k t}$$

- The allowed angular frequencies have the same spacing $\Delta\omega$, but are now infinitely many
- Intimate relationship between Discrete Fourier Transforms and Complex Fourier Series:

$$f_n = \frac{1}{N}\sum_{p=0}^{N-1} \tilde{f}_p\, e^{-i\,2\pi pn/N}$$

Discrete Fourier Transforms

In Multiple Dimensions

For a function $f_{n,m} = f(x_n, y_m)$ defined on a 2D grid $x_n = m\Delta x$ for $0 \le n \le N-1$ and $y_m = m\Delta y$ for $0 \le n \le M-1$:

- The DFT is

$$\tilde{f}_{p,q} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{n,m}\, e^{i\,2\pi pn/N}\, e^{i\,2\pi qm/M}$$

- And the backward DFT is

$$f_{n,m} = \frac{1}{N\,M} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \tilde{f}_{p,q}\, e^{-i\,2\pi pn/N}\, e^{-i\,2\pi qm/M}$$

- This can easily be extended to three and more dimensions

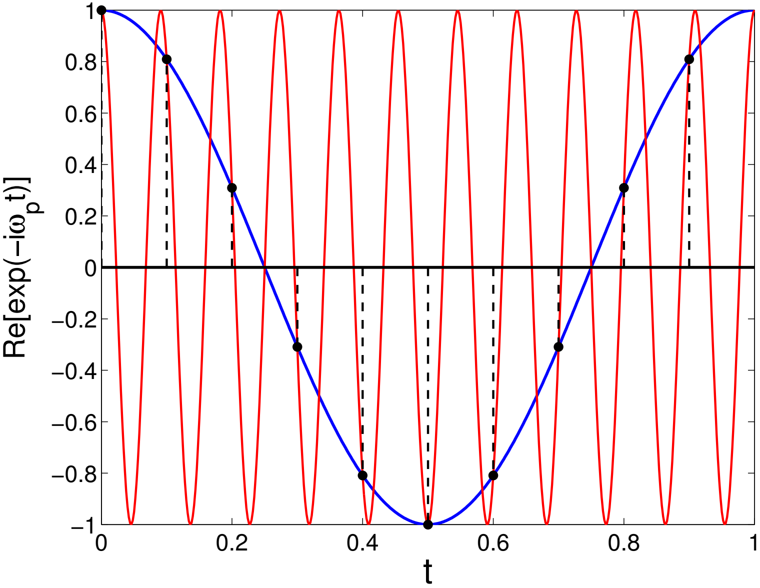
Sampling and Aliasing

- For any wave with a frequency ω_p lying in the frequency domain captured by the DFT, there are higher frequency waves with $\omega_{p'} = \omega_p + m\Omega$ (where $m \in \mathbb{Z}$ and $\Omega = 2\omega_{max}$ is the width of the frequency domain) that look exactly the same when

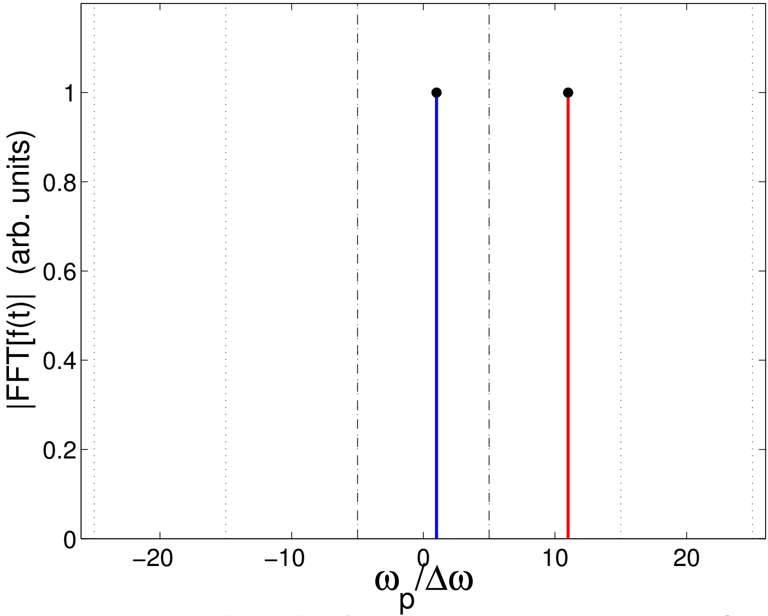
- This is clear from

$$\exp\left[i(\omega_p + m\Omega)t_n\right] = \exp\left[i\left(\frac{2\pi\, pn}{N} + \qquad\qquad\right)\right]$$

Sampling and Aliasing



Sampling and Aliasing



Sampling and Aliasing

The discrete sampling of the function to be transformed introduces some sampling effects and care has to be taken in allowing for them

- There is a minimum sampling rate that needs to be used so that we don't lose information
- If the function to be sampled fits into the finite length time (or spatial) domain of length T, and is **bandwidth limited** to below the Nyquist frequency then
- If the function has $\tilde{f}(\omega) \neq 0$ for $|\omega| > \omega_{max}$ then the frequency content for $|\omega| > \omega_{max}$ will be **aliased** down into the range $|\omega| < \omega_{max}$ and distort the DFT compared to the
- If the original function is bandwidth limited but is longer than T, then it will be ***clipped*** which will introduce a sharp cutoff at $t = 0$ and/or T
- This will effectively increase the bandwidth of the clipped function and aliasing will occur again during sampling of it

Sampling and Aliasing

- This explains:
  - Why $\tilde{f}_{-N/2} \equiv \tilde{f}$
  - Why the equation for the backwards DFT works:
    - The "negative" frequency part of the spectrum $p = \left\{-\frac{N}{2}+1, \ldots, -1\right\}$ maps to the positive spectrum at $p' = p + N = \left\{\frac{N}{2}+1, \frac{N}{2}+2, \ldots, N-1\right\}$ lying beyond the Nyquist frequency
    - In other words, although the $\sum_{p=N/2+1}^{N-1}$ parts are above the Nyquist frequency, they happen to capture the desired negative frequencies within the Nyquist range
  - Aliasing:

$$\tilde{f}_p = \sum_m \mathcal{F}(f)|_{\omega_p + m\Omega}$$

i.e., the DFT (the LHS) folds in the Fourier components from the ***exact FT of the continuous function*** from the indistinguishable set of waves $\omega_p' = p + m\Omega$.

Fast Fourier Transforms

- Directly coding the DFT results in an $\mathcal{O}(N^2)$ algorithm
- But this can be reduced to $\mathcal{O}(N \log_2 N)$ with the Fast Fourier Transform, for $N = 2^m$
- Actually invented in the early 1800s by Gauss (before Fourier), but discovered and made popular by Cooley and Tukey in 1965
- The principles are:
  - Can split an N-sample DFT into two $N/2$-sample ones

$$
\begin{aligned}
\tilde{f}_p &= \sum_{n=0}^{N-1} f_n e^{i2\pi pn/N} \\
&= \sum_{n=0,2,\ldots}^{N-2} f_n e^{i2\pi pn/N} + \sum_{n=1,3,\ldots}^{N-1} f_n e^{i2\pi pn/N} \\
&= \tilde{f}_p^{\text{even}} + e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}}
\end{aligned}
$$

  - This can be done

Fast Fourier Transforms

Outline of Algorithm

- For $0 \leq p < N$:

$$\begin{aligned}\tilde{f}_p &= \sum_{n=0}^{N-1} f_n e^{i2\pi pn/N} \\ &= \tilde{f}_p^{\text{even}} + e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}}\end{aligned}$$

- Here, $\tilde{f}_p^{\text{even,odd}}$ are $N/2$-sample DFTs of the even and odd samples of the original f_n
- These are each periodic such that $\tilde{f}_{p+N/2}^{\text{even,odd}} = \tilde{f}_p^{\text{even,odd}}$
- This results in, for $0 \leq p < N/2$:

$$\begin{aligned}\tilde{f}_p &= \tilde{f}_p^{\text{even}} + e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}} \\ \tilde{f}_{p+N/2} &= \tilde{f}_p^{\text{even}} - e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}}\end{aligned}$$

- This takes about $\mathcal{O}(N^2/4 + N)$ operations

Pseudocode

- We often need to explain an algorithm to each other, in a way that clearly shows how you would code it up (in any language)
- In this case, it helps not to be burdened by the (language-specific) overhead that real code brings with it (as well as a need to be syntactically perfect)
  - So omit things like

```
"import numpy as np"
"tarray = np.zeros(N)"
"#include <stdio.h.>"
"COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),
                      LEV,LEVIN,BLVECT(LURCOR)"
```

- We call this "Pseudocode"; it is ultimately intended to be human-friendly, but with a code-like structure
  - No formal syntax; it is up to the author to make things clear
  - Can use language-specific elements if it is not confusing (e.g., logic statements from Python or C etc.)

Pseudocode

Examples

- Examples of pseudocode of all sorts of styles are readily available online, but here are a couple of examples:

```
N = 100 # divisions
deltat = 1.0/N # goes from 0 to 1

function f(x)
  f = -100x^2+100x^4
  if x is between 0.4 and 0.5
    return f*10*abs(x-0.5)
  else
    return f

function ftilde(p)
  sum = 0
  for n from 0 to N
  sum += f(n*deltat)
         * exp(i*2*pi*p*n/N)
  return sum
```

```
def incircle(x, y):
  return True if (x,y) is
           in a circle of radius 0.5,
         False if not

total = 0
accepted = 0
loop 10 times
  assign random numbers between
           0 and 1 to x and y
  print x, y
  if incircle(x, y):
    accepted += 1
    total += 1

ratio = accepted / total
print 4 * ratio, real value of pi
```

Fast Fourier Transforms

Representation in Pseudocode

```
function FFT(f): # f is an array
         N = size of array f
         if N == 1:
             return f[0] # for a sample size of 1, the FT is the value itself
         if N is not a power of 2, exit

         # recursive calls
         farray_even = FFT(even entries of f) # size N/2
         farray_odd  = FFT(odd  entries of f) # size N/2

         return an N-element array made up of
                   farray_even[p] + exp(i 2pi p/N) * farray_odd[p]
                     (for p = 0..N/2-1)
            and of farray_even[p] - exp(i 2pi p/N) * farray_odd[p]
                     (for p = N/2..N-1)
```

The expressions in the last few lines was corrected after the lecture

Fast Fourier Transforms

- Recursive calls, while logically clear, can require substantial overheads when the code is executed
  - but *"Premature optimization is the root of all evil"*!
- Further shortcuts exist which can speed up the code by sorting the elements of the calculations in a clever way
- The use of reversing the binary representations of the indices of the discretised function is given in the lecture notes

- The FFT algorithm leads naturally to the Fast Convolution of functions, Filtering, and finding the Correlation between functions etc.

Using Fourier Transforms to Solve Problems

Spectral Methods

- Solving PDEs using FTs, e.g.: the Poisson Equation

  where ρ is a source density and u is some potential
  - appears in electrostatic and gravitational potentials, etc.
- With periodic boundary conditions, FFT methods are very fast
- Using FTs, the analytical solution for the above is

$$\tilde{u}(\vec{k}) = -\frac{\tilde{\rho}(\vec{k})}{|\vec{k}|^2}$$

- The inverse transform gives:

$$u(\vec{x}) = \mathcal{F}^{-1}\left[-\frac{\tilde{\rho}(\vec{k})}{|\vec{k}|^2}\right]$$

- Similar methods can be applied to the calculations of convolutions etc.

Today: Summary

- Introduction
- Review of Fourier Transforms (very briefly)
- Discrete Fourier Transforms
- Sampling and Aliasing
- Fast Fourier Transforms
- An Introduction to Pseudocode
- Spectral Methods

Tomorrow we will look at Random Numbers and Monte Carlo Methods