Computational Physics – Lecture 15

(8 April 2020)

Recap

In the previous lecture, we defined the Fourier Transform, the Fourier Series, and the Discrete Fourier Transform.

Fourier Transform

$$\boxed{\text{Function} \longleftrightarrow \text{Function}}$$

$$\tilde{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathrm{d}x \cdot f(x) \cdot \exp(-ikx) \tag{1}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \mathrm{d}k \cdot \tilde{f}(k) \cdot \exp(ikx) \tag{2}$$

Fourier Series

$$\boxed{\text{Periodic Function} \longleftrightarrow \text{Numbers}}$$

$$\gamma_k = \frac{1}{\sqrt{2\pi}} \int_0^{2\pi} \mathrm{d}x \cdot f(x) \cdot \exp(-ikx) \tag{3}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} \gamma_k \cdot \exp(ikx) \tag{4}$$

where $f(x + 2\pi) = f(x)$ and $k = \ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$

Discrete Fourier Transform

$$\boxed{\text{Numbers} \longleftrightarrow \text{Numbers}}$$

$$\tilde{f}(k_q) = \frac{1}{\sqrt{n}} \sum_{p=0}^{n-1} f(x_p) \cdot \exp(-ik_q x_p) \tag{5}$$

$$f(x_p) = \frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} \tilde{f}(k_q) \cdot \exp(ik_q x_p) \tag{6}$$

where $x_p = p\Delta$ for $p = 0, 1, \ldots, n-1$ and

$k_q = 2\pi q/n\Delta$ for $q = 0, 1, \ldots, n-1$

Today's plan

In today's class, we want to go deeper into the Discrete Fourier Transform.

nth roots of unity

We know that there n complex nth roots of unity, given by

$$\exp\left(\frac{i \cdot 2\pi \cdot k}{n}\right) \tag{7}$$

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

nth roots of unity

We know that there n complex nth roots of unity, given by

$$\exp\left(\frac{i \cdot 2\pi \cdot k}{n}\right) \tag{7}$$

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

Notice that

$$\begin{aligned}
&\left[\exp\left(\frac{i \cdot 2\pi \cdot k}{n}\right)\right]^n && (8)\\
&= \exp\left(i \cdot 2\pi \cdot k\right) && (9)\\
&= \cos\left(2\pi \cdot k\right) + i \cdot \sin\left(2\pi \cdot k\right) && (10)\\
&= 1 + i \cdot 0 && (11)\\
&= 1 && (12)
\end{aligned}$$

nth roots of unity

For example, the four fourth roots of unity are given by

$$\exp\left(\frac{i \cdot 2\pi \cdot 0}{4}\right) = 1 \tag{13}$$

$$\exp\left(\frac{i \cdot 2\pi \cdot 1}{4}\right) = e^{i\pi/2} \tag{14}$$

$$\exp\left(\frac{i \cdot 2\pi \cdot 2}{4}\right) = e^{i\pi} \tag{15}$$

$$\exp\left(\frac{i \cdot 2\pi \cdot 3}{4}\right) = e^{i3\pi/2} \tag{16}$$

A basis for $\mathbb{C}^n$

Remarkably, the n nth roots of unity can be used to create a basis for the finite-dimensional complex linear vector space $\mathbb{C}^n$.

A basis for $\mathbb{C}^n$

Remarkably, the n nth roots of unity can be used to create a basis for the finite-dimensional complex linear vector space $\mathbb{C}^n$.

(We are assuming the inner product

$$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a} \cdot \mathbf{b}^* \tag{17}$$

where $\mathbf{a}, \mathbf{b} \in \mathbb{C}^n$ and $\mathbf{b}^*$ is the complex conjugate of $\mathbf{b}$.)

A basis for $\mathbb{C}^n$

For $n = 4$, for instance, the four vectors

$$\begin{bmatrix} \exp\left(\frac{i\cdot 2\pi\cdot 0}{4}\cdot 0\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 0}{4}\cdot 1\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 0}{4}\cdot 2\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 0}{4}\cdot 3\right) \end{bmatrix}, \begin{bmatrix} \exp\left(\frac{i\cdot 2\pi\cdot 1}{4}\cdot 0\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 1}{4}\cdot 1\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 1}{4}\cdot 2\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 1}{4}\cdot 3\right) \end{bmatrix}, \\ \begin{bmatrix} \exp\left(\frac{i\cdot 2\pi\cdot 2}{4}\cdot 0\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 2}{4}\cdot 1\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 2}{4}\cdot 2\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 2}{4}\cdot 3\right) \end{bmatrix}, \begin{bmatrix} \exp\left(\frac{i\cdot 2\pi\cdot 3}{4}\cdot 0\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 3}{4}\cdot 1\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 3}{4}\cdot 2\right) \\ \exp\left(\frac{i\cdot 2\pi\cdot 3}{4}\cdot 3\right) \end{bmatrix} \tag{18}$$

form a basis for $\mathbb{C}^4$.

A basis for $\mathbb{C}^n$

Or in other words, the four vectors,

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ e^{i\pi/2} \\ e^{i\pi} \\ e^{i3\pi/2} \end{bmatrix}, \begin{bmatrix} 1 \\ e^{i\pi} \\ 1 \\ e^{i\pi} \end{bmatrix}, \begin{bmatrix} 1 \\ e^{i3\pi/2} \\ e^{i\pi} \\ e^{i\pi/2} \end{bmatrix} \tag{19}$$

form a basis for $\mathbb{C}^4$.

A basis for $\mathbb{C}^n$

In general, the n n-dimensional vectors

$$\mathbf{v}_k = \left[\exp\left(\frac{i \cdot 2\pi \cdot k}{n} \cdot m\right) \;\middle|\; m = 0, 1, \ldots, n-1\right]^T \tag{20}$$

for $k = 0, 1, \ldots, n-1$ form a basis for $\mathbb{C}^n$.

A basis for $\mathbb{C}^n$

In general, the n n-dimensional vectors

$$\mathbf{v}_k = \left[\exp\left(\frac{i \cdot 2\pi \cdot k}{n} \cdot m \right) \middle| \, m = 0, 1, \ldots, n-1 \right]^T \tag{20}$$

for $k = 0, 1, \ldots, n-1$ form a basis for $\mathbb{C}^n$.

(Here the index k labels the vectors and the index m labels the vector components.)

A basis for $\mathbb{C}^n$

To show that the $\mathbf{v}_k$ really do form a basis of $\mathbb{C}^n$, we first notice that these vectors are n in number.

A basis for $\mathbb{C}^n$

To show that the $\mathbf{v}_k$ really do form a basis of $\mathbb{C}^n$, we first notice that these vectors are n in number.

Now we show that these vectors are orthogonal.

A basis for $\mathbb{C}^n$

If $k_1 = k_2$

$$
\begin{aligned}
&\langle \mathbf{v}_{k_1}, \mathbf{v}_{k_2} \rangle && (21)\\
&= \mathbf{v}_{k_1} \cdot \mathbf{v}_{k_2}^* && (22)\\
&= \sum_{m=0}^{n-1} v_{k_1,m} \cdot v_{k_2,m}^* && (23)\\
&= \sum_{m=0}^{n-1} \exp\left(\frac{i \cdot 2\pi \cdot k_1}{n} \cdot m\right) \cdot \exp\left(\frac{-i \cdot 2\pi \cdot k_2}{n} \cdot m\right) && (24)\\
&= \sum_{m=0}^{n-1} 1 \qquad (\text{for } k_1 = k_2) && (25)\\
&= n > 0. && (26)
\end{aligned}
$$

A basis for $\mathbb{C}^n$

And if $k_1 \neq k_2$

$$
\begin{aligned}
&\langle \mathbf{v}_{k_1}, \mathbf{v}_{k_2} \rangle && (27)\\
&= \mathbf{v}_{k_1} \cdot \mathbf{v}_{k_2}^* && (28)\\
&= \sum_{m=0}^{n-1} v_{k_1,m} \cdot v_{k_2,m}^* && (29)\\
&= \sum_{m=0}^{n-1} \left[\exp\left(\frac{i \cdot 2\pi \cdot (k_1 - k_2)}{n}\right)\right]^m && (30)\\
&= \frac{1 - \left[\exp\left(\frac{i \cdot 2\pi \cdot (k_1 - k_2)}{n}\right)\right]^n}{1 - \left[\exp\left(\frac{i \cdot 2\pi \cdot (k_1 - k_2)}{n}\right)\right]} && (31)\\
&= \frac{1 - 1}{1 - \left[\exp\left(\frac{i \cdot 2\pi \cdot (k_1 - k_2)}{n}\right)\right]} = 0. && (32)
\end{aligned}
$$

A basis for $\mathbb{C}^n$

So we have shown that

$$\langle \mathbf{v}_{k_1}, \mathbf{v}_{k_2} \rangle = n\delta_{k_1,k_2}. \tag{33}$$

A basis for $\mathbb{C}^n$

So we have shown that

$$\langle \mathbf{v}_{k_1}, \mathbf{v}_{k_2} \rangle = n\delta_{k_1,k_2}. \tag{33}$$

The $\mathbf{v}_k$ are orthogonal and are indeed a basis for $\mathbb{C}^n$.

A basis for $\mathbb{C}^n$

So we have shown that

$$\langle \mathbf{v}_{k_1}, \mathbf{v}_{k_2} \rangle = n\delta_{k_1,k_2}. \tag{33}$$

The $\mathbf{v}_k$ are orthogonal and are indeed a basis for $\mathbb{C}^n$.

That means, any vector $\mathbf{w} \in \mathbb{C}^n$ can now be expanded in this basis to get the components of $\mathbf{w}$ as

$$\begin{aligned}
\tilde{w}_q &= \langle \mathbf{w}, \mathbf{v}_q \rangle & (34)\\
&= \mathbf{w} \cdot \mathbf{v}_q^* & (35)\\
&= \sum_{p=0}^{n-1} w_p \cdot v_{q,p}^* & (36)\\
&= \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p \right) & (37)
\end{aligned}$$

Discrete Fourier Transform

So we have

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{38}$$

where the w_p are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in the standard basis and the $\tilde{w}_q$ are a representation of the same vector in our new basis.

Discrete Fourier Transform

So we have

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{38}$$

where the w_p are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in the standard basis and the $\tilde{w}_q$ are a representation of the same vector in our new basis.

This is the Discrete Fourier Transform.

Discrete Fourier Transform

So we have

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{38}$$

where the w_p are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in the standard basis and the $\tilde{w}_q$ are a representation of the same vector in our new basis.

This is the Discrete Fourier Transform.

We have transformed n numbers w_p into n other numbers $\tilde{w}_q$.

Inverse Discrete Fourier Transform

We can also go from the new basis to the old basis.

Inverse Discrete Fourier Transform

We can also go from the new basis to the old basis.

The n basis vectors of the standard basis of $\mathbb{C}^n$ are simply

$$\mathbf{u}_x = \left[\delta_{x,m} \,\middle|\, m = 0, 1, \ldots, n-1\right]^T \tag{39}$$

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

Inverse Discrete Fourier Transform

We can also go from the new basis to the old basis.

The n basis vectors of the standard basis of $\mathbb{C}^n$ are simply

$$\mathbf{u}_x = \left[\delta_{x,m} \,\middle|\, m = 0, 1, \ldots, n-1 \right]^T \tag{39}$$

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

For example, for $n = 4$, the Standard Basis of $\mathbb{C}^4$ is

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{40}$$

Inverse Discrete Fourier Transform

Now any vector $\mathbf{w} \in \mathbb{C}^n$ can be expanded in the standard basis to get the components of $\mathbf{w}$ as

$$\begin{aligned} w_p &= \langle \mathbf{w}, \mathbf{u}_p \rangle && (41) \\ &= \mathbf{w} \cdot \mathbf{u}_p^* && (42) \end{aligned}$$

Inverse Discrete Fourier Transform

Now any vector $\mathbf{w} \in \mathbb{C}^n$ can be expanded in the standard basis to get the components of $\mathbf{w}$ as

$$w_p = \langle \mathbf{w}, \mathbf{u}_p \rangle \tag{41}$$

$$= \mathbf{w} \cdot \mathbf{u}_p^* \tag{42}$$

But now we only know $\mathbf{w}$ in the new basis.

Inverse Discrete Fourier Transform

Now any vector $\mathbf{w} \in \mathbb{C}^n$ can be expanded in the standard basis to get the components of $\mathbf{w}$ as

$$\begin{aligned} w_p &= \langle \mathbf{w}, \mathbf{u}_p \rangle &(41)\\ &= \mathbf{w} \cdot \mathbf{u}_p^* &(42) \end{aligned}$$

But now we only know $\mathbf{w}$ in the new basis.

So while calculating these inner products we must use the new-basis representation of the standard basis vectors $\mathbf{u}_p$.

Inverse Discrete Fourier Transform

Now any vector $\mathbf{w} \in \mathbb{C}^n$ can be expanded in the standard basis to get the components of $\mathbf{w}$ as

$$\begin{aligned} w_p &= \langle \mathbf{w}, \mathbf{u}_p \rangle && (41) \\ &= \mathbf{w} \cdot \mathbf{u}_p^* && (42) \end{aligned}$$

But now we only know $\mathbf{w}$ in the new basis.

So while calculating these inner products we must use the new-basis representation of the standard basis vectors $\mathbf{u}_p$.

What is the new-basis representation of the standard basis vectors?

Inverse Discrete Fourier Transform

The new-basis representation of the standard basis vectors is just their Discrete Fourier Transform, that is

$$\tilde{u}_{p,a} = \sum_{b=0}^{n-1} u_{p,b} \cdot \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot b\right) \tag{43}$$

$$= \sum_{b=0}^{n-1} \delta_{p,b} \cdot \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot b\right) \tag{44}$$

$$= \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot p\right) \tag{45}$$

Inverse Discrete Fourier Transform

The new-basis representation of the standard basis vectors is just their Discrete Fourier Transform, that is

$$\tilde{u}_{p,a} = \sum_{b=0}^{n-1} u_{p,b} \cdot \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot b\right) \quad (43)$$

$$= \sum_{b=0}^{n-1} \delta_{p,b} \cdot \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot b\right) \quad (44)$$

$$= \exp\left(\frac{-i \cdot 2\pi \cdot a}{n} \cdot p\right) \quad (45)$$

We can now use this representation of the standard basis vectors to calculate the w_p's.

Inverse Discrete Fourier Transform

We have

$$w_p = \langle \mathbf{w}, \mathbf{u}_p \rangle \tag{46}$$
$$= \mathbf{w} \cdot \mathbf{u}_p^* \tag{47}$$
$$= \sum_{q=0}^{n-1} \tilde{w}_q \cdot \tilde{u}_{p,q}^* \tag{48}$$
$$= \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right) \tag{49}$$

Inverse Discrete Fourier Transform

So we have

$$w_p = \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right) \tag{50}$$

where the $\tilde{w}_q$ are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in our new basis and the w_p are a representation of the same vector in the standard basis basis.

Inverse Discrete Fourier Transform

So we have

$$w_p = \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right) \tag{50}$$

where the $\tilde{w}_q$ are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in our new basis and the w_p are a representation of the same vector in the standard basis basis.

This is the Inverse Discrete Fourier Transform.

Inverse Discrete Fourier Transform

So we have

$$w_p = \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right) \tag{50}$$

where the $\tilde{w}_q$ are a representation of the vector $\mathbf{w} \in \mathbb{C}^n$ in our new basis and the w_p are a representation of the same vector in the standard basis basis.

This is the Inverse Discrete Fourier Transform.

We have transformed n numbers $\tilde{w}_p$ into n other numbers w_q.

Discrete Fourier Transform

So the Discrete Fourier Transform is given by

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{51}$$

and its inverse is given by

$$w_p = \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right). \tag{52}$$

Discrete Fourier Transform

Now we would expect that if we calculated the DFT of n numbers and then calculated the inverse DFT we would recover the n numbers.

Discrete Fourier Transform

Now we would expect that if we calculated the DFT of n numbers and then calculated the inverse DFT we would recover the n numbers.

Let us check this:

$$\sum_{q=0}^{n-1} \tilde{w}_q \exp\left(\frac{i2\pi qp}{n}\right) \tag{53}$$

$$= \sum_{q=0}^{n-1} \left[\sum_{r=0}^{n-1} w_r \exp\left(\frac{-i2\pi qr}{n}\right)\right] \exp\left(\frac{i2\pi qp}{n}\right) \tag{54}$$

$$= \sum_{q=0}^{n-1} \sum_{r=0}^{n-1} w_r \exp\left(\frac{i2\pi q(p-r)}{n}\right) \tag{55}$$

$$= \sum_{r=0}^{n-1} w_r \cdot n \cdot \delta_{r,p} = n \cdot w_p \neq w_p. \tag{56}$$

Discrete Fourier Transform

So our DFT does not preserve norm. It is not normalised.

Discrete Fourier Transform

So our DFT does not preserve norm. It is not normalised.

We can make it normalised and unitary by doing

$$\tilde{w}_q = \frac{1}{\sqrt{n}} \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{57}$$

and the inverse is

$$w_p = \frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right). \tag{58}$$

Discrete Fourier Transform

So our DFT does not preserve norm. It is not normalised.

We can make it normalised and unitary by doing

$$\tilde{w}_q = \frac{1}{\sqrt{n}} \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i \cdot 2\pi \cdot q}{n} \cdot p\right), \tag{57}$$

and the inverse is

$$w_p = \frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i \cdot 2\pi \cdot q}{n} \cdot p\right). \tag{58}$$

Now we can freely go back and forth between the Fourier-space representation and the configuration-space representation.

DFT using Numpy

In Numpy, the `numpy.fft` module contains various functions for Fourier analysis.

DFT using Numpy

In Numpy, the `numpy.fft` module contains various functions for Fourier analysis.

The function `numpy.fft.fft` computes the DFT and the function `numpy.fft.ifft` computes the inverse DFT.

DFT using Numpy

Numpy's DFT is not unitary, so that

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i2\pi qp}{n}\right), \tag{59}$$

and

$$w_p = \frac{1}{n}\sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i2\pi qp}{n}\right). \tag{60}$$

DFT using Numpy

Numpy's DFT is not unitary, so that

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i2\pi qp}{n}\right), \qquad (59)$$

and

$$w_p = \frac{1}{n} \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i2\pi qp}{n}\right). \qquad (60)$$

For example:

```
>>> import numpy as np
>>> w_p = (1,0,0,0)
>>> tw_q = np.fft.fft(w_p)
>>> tw_q
array([1.+0.j, 1.+0.j, 1.+0.j, 1.+0.j])
>>> np.fft.ifft(tw_q)
array([1.+0.j, 0.+0.j, 0.+0.j, 0.+0.j])
```

DFT using FFTW

FFTW is a famous C library written by Matteo Frigo (MIT) and Steven Johnson (MIT).

DFT using FFTW

FFTW is a famous C library written by Matteo Frigo (MIT) and Steven Johnson (MIT).

FFTW's DFT is not even normalised, so that

$$\tilde{w}_q = \sum_{p=0}^{n-1} w_p \cdot \exp\left(\frac{-i2\pi qp}{n}\right), \tag{61}$$

and

$$w_p = \sum_{q=0}^{n-1} \tilde{w}_q \cdot \exp\left(\frac{i2\pi qp}{n}\right). \tag{62}$$

DFT using FFTW

```
#include <stdio.h>
#include <stdlib.h>
#include <fftw3.h>

void main () {

  fftw_complex w_p[4], tw_q[4];
  fftw_plan p;

  w_p[0][0] = 1.0; w_p[0][1] = 0.0;
  w_p[1][0] = 0.0; w_p[1][1] = 0.0;
  w_p[2][0] = 0.0; w_p[2][1] = 0.0;
  w_p[3][0] = 0.0; w_p[3][1] = 0.0;

  int n = 4;
  p = fftw_plan_dft_1d(n, w_p, tw_q,
      FFTW_FORWARD, FFTW_ESTIMATE);
```

DFT using FFTW

```
fftw_execute(p);

int i;
for(i=0; i<4; i++) {
  printf("%f  %f\n", tw_q[i][0], tw_q[i][1]);
}
printf("\n");

p = fftw_plan_dft_1d(n, tw_q, w_p,
    FFTW_BACKWARD, FFTW_ESTIMATE);

fftw_execute(p);
```

DFT using FFTW

```
  for(i=0; i<4; i++) {
    printf("%f  %f\n", w_p[i][0], w_p[i][1]);
  }

  fftw_destroy_plan(p);

}
```

DFT using FFTW

This code, when compiled as

```
$ gcc -o demo demo.c -lfftw3
```

and run, produces

```
$ ./demo
1.000000  0.000000
1.000000  0.000000
1.000000  0.000000
1.000000  0.000000

4.000000  0.000000
0.000000  0.000000
0.000000  0.000000
0.000000  0.000000
```

(Notice the unnormalised output.)

Activity

Compute the DFT of $[0, 1, 0, 0]^T$ first manually and then using `numpy.fft.fft`. Then compute the inverse DFT of the result using `numpy.fft.ifft`.

Discrete Fourier Transform

We can also change our notation to think of the w's and $\tilde{w}$'s as functions of p and q (respectively).

Discrete Fourier Transform

We can also change our notation to think of the w's and $\tilde{w}$'s as functions of p and q (respectively).

So we can write

$$\tilde{f}(q) = \frac{1}{\sqrt{n}} \sum_{p=0}^{n-1} f(p) \cdot \exp\left(\frac{-i2\pi qp}{n}\right), \tag{63}$$

and the inverse as

$$f(p) = \frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} \tilde{f}(q) \cdot \exp\left(\frac{i2\pi qp}{n}\right), \tag{64}$$

for $p = 0, 1, \ldots, n-1$ and $q = 0, 1, \ldots, n-1$.

Discrete Fourier Transform

We can also introduce new variables

$$x_p = p\Delta \text{ for } p = 0, 1, \ldots, n-1, \tag{65}$$

and

$$k_q = 2\pi q/n\Delta \text{ for } q = 0, 1, \ldots, n-1. \tag{66}$$

Discrete Fourier Transform

We can also introduce new variables

$$x_p = p\Delta \text{ for } p = 0, 1, \ldots, n-1, \tag{65}$$

and

$$k_q = 2\pi q/n\Delta \text{ for } q = 0, 1, \ldots, n-1. \tag{66}$$

so that

$$\tilde{f}(k_q) = \frac{1}{\sqrt{n}} \sum_{p=0}^{n-1} f(x_p) \cdot \exp\left(-ik_q x_p\right), \tag{67}$$

and the inverse as

$$f(x_p) = \frac{1}{\sqrt{n}} \sum_{q=0}^{n-1} \tilde{f}(k_q) \cdot \exp\left(ik_q x_p\right), \tag{68}$$

for $p = 0, 1, \ldots, n-1$ and $q = 0, 1, \ldots, n-1$.