量子計算的數學基礎 MA5501*

Chapter 5. The Fourier Transform

§5.1 The Classical Discrete Fourier Transform

The Fourier transform occurs in many different versions throughout classical computing, in areas ranging from signal-processing to data compression to complexity theory. For our purposes, the Fourier transform is going to be an $N \times N$ unitary matrix, all of whose entries have the same magnitude. For $N = 2$, it's just our familiar Hadamard transform:

$$F_2 = \mathrm{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Doing something similar in 3 dimensions is impossible with real numbers: we cannot give three orthogonal vectors in $\{1, -1\}^3$. However, using complex numbers allows us to define the Fourier transform for any N.

§5.1 The Classical Discrete Fourier Transform

Let $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$ be an N-th root of unity. The rows of the matrix will be indexed by $j \in \{0, \cdots, N-1\}$ and the columns by $k \in \{0, \cdots, N-1\}$ (so we use the $(0,0)$-entry to denote the usual $(1,1)$-entry). Define the (j,k)-entry of the matrix F_N by $\frac{1}{\sqrt{N}}\omega_N^{jk}$:

$$F_N = \frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}.$$

Note that F_N is a unitary matrix, since each column has norm 1, and any pair of columns (say those indexed by k and k') is orthogonal:

$$\sum_{j=0}^{N-1} \frac{1}{\sqrt{N}}\overline{\omega_N^{jk}} \cdot \frac{1}{\sqrt{N}}\omega_N^{jk'} = \frac{1}{N}\sum_{j=0}^{N-1}\omega_N^{j(k'-k)} = \begin{cases} 1 & \text{if } k' = k\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

§5.1 The Classical Discrete Fourier Transform

Let $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$ be an N-th root of unity. The rows of the matrix will be indexed by $j \in \{0, \cdots, N-1\}$ and the columns by $k \in \{0, \cdots, N-1\}$ (so we use the $(0, 0)$-entry to denote the usual $(1, 1)$-entry). Define the (j, k)-entry of the matrix F_N by $\frac{1}{\sqrt{N}}\omega_N^{jk}$:

$$F_N = \frac{1}{\sqrt{N}}\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}.$$

Note that F_N is a unitary matrix, since each column has norm 1, and any pair of columns (say those indexed by k and k') is orthogonal:

$$\sum_{j=0}^{N-1} \frac{1}{\sqrt{N}} \overline{\omega_N^{jk}} \cdot \frac{1}{\sqrt{N}} \omega_N^{jk'} = \frac{1}{N} \sum_{j=0}^{N-1} \omega_N^{j(k'-k)} = \begin{cases} 1 & \text{if } k' = k\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

§5.1 The Classical Discrete Fourier Transform

$$F_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}, \quad \omega_N = \exp\left(\frac{2\pi i}{N}\right).$$

Since F_N is unitary and symmetric, the inverse $F_N^{-1} = F_N^*$ only differs from F_N by having minus signs in the exponent of the entries. For a vector $v \in \mathbb{C}^N$, the vector $\hat{v} = F_N v$ is called the discrete Fourier transform (DFT) of v. Doing the matrix-vector multiplication, its entries are given by

$$\hat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k \,.$$

§5.1 The Classical Discrete Fourier Transform

$$F_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_N & \omega_N^2 & \cdots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \cdots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix}, \quad \omega_N = \exp\left(\frac{2\pi i}{N}\right).$$

Since F_N is unitary and symmetric, the inverse $F_N^{-1} = F_N^*$ only differs from F_N by having minus signs in the exponent of the entries. For a vector $v \in \mathbb{C}^N$, the vector $\hat{v} = F_N v$ is called the discrete Fourier transform (DFT) of v. Doing the matrix-vector multiplication, its entries are given by

$$\hat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k \,.$$

§5.2 The Fast Fourier Transform

A naive way of computing the Fourier transform $\hat{v} = F_N v$ of $v \in \mathbb{R}^N$ just does the matrix-vector multiplication to compute all the entries of $\hat{v}$. This would take $\mathcal{O}(N)$ steps (additions and multiplications) per entry, and $\mathcal{O}(N^2)$ steps to compute the whole vector $\hat{v}$. However, there is a more efficient way of computing $\hat{v}$. This algorithm is called the **Fast Fourier Transform** (**FFT**, due to Cooley and Tukey in 1965), and takes only $\mathcal{O}(N \log_2 N)$ steps. This difference between the quadratic N^2 steps and the **near-linear** $N \log_2 N$ is tremendously important in practice when N is large, and is the main reason that Fourier transforms are so widely used.

§5.2 The Fast Fourier Transform

A naive way of computing the Fourier transform $\hat{v} = F_N v$ of $v \in \mathbb{R}^N$ just does the matrix-vector multiplication to compute all the entries of $\hat{v}$. This would take $\mathcal{O}(N)$ steps (additions and multiplications) per entry, and $\mathcal{O}(N^2)$ steps to compute the whole vector $\hat{v}$. However, there is a more efficient way of computing $\hat{v}$. This algorithm is called the **Fast Fourier Transform** (**FFT**, due to Cooley and Tukey in 1965), and takes only $\mathcal{O}(N \log_2 N)$ steps. This difference between the quadratic N^2 steps and the **near-linear** $N \log_2 N$ is tremendously important in practice when N is large, and is the main reason that Fourier transforms are so widely used.

§5.2 The Fast Fourier Transform

A naive way of computing the Fourier transform $\widehat{v} = F_N v$ of $v \in \mathbb{R}^N$ just does the matrix-vector multiplication to compute all the entries of $\widehat{v}$. This would take $\mathcal{O}(N)$ steps (additions and multiplications) per entry, and $\mathcal{O}(N^2)$ steps to compute the whole vector $\widehat{v}$. However, there is a more efficient way of computing $\widehat{v}$. This algorithm is called the **Fast Fourier Transform** (**FFT**, due to Cooley and Tukey in 1965), and takes only $\mathcal{O}(N \log_2 N)$ steps. This difference between the quadratic N^2 steps and the **near-linear** $N \log_2 N$ is tremendously important in practice when N is large, and is the main reason that Fourier transforms are so widely used.

§5.2 The Fast Fourier Transform

We will assume $N = 2^n$, which is usually fine because we can add zeroes to our vector to make its dimension a power of 2 (but similar FFTs can be given also directly for most N that are not a power of 2). The key to the FFT is to rewrite the entries of $\widehat{v}$ as follows:

$$\widehat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k = \frac{1}{\sqrt{N}} \Big(\sum_{k \text{ even}} \omega_N^{jk} v_k + \sum_{k \text{ odd}} \omega_N^{jk} v_k \Big)$$

$$= \frac{1}{\sqrt{2}} \Big(\frac{1}{\sqrt{N/2}} \sum_{k \text{ even}} \omega_{N/2}^{jk/2} v_k + \frac{\omega_N^j}{\sqrt{N/2}} \sum_{k \text{ odd}} \omega_{N/2}^{j(k-1)/2} v_k \Big).$$

$$\omega_N^{jk} = \exp\left(\frac{2\pi jki}{N}\right) = \exp\left(\frac{2\pi j(k/2)i}{N/2}\right) = \omega_{N/2}^{jk/2} \quad \text{if } k \text{ is even,}$$

$$\omega_N^{jk} = \omega_N^j \omega_N^{j(k-1)} = \omega_N^j \omega_{N/2}^{j(k-1)/2} \quad \text{if } k \text{ is odd.}$$

§5.2 The Fast Fourier Transform

We will assume $N = 2^n$, which is usually fine because we can add zeroes to our vector to make its dimension a power of 2 (but similar FFTs can be given also directly for most N that are not a power of 2). The key to the FFT is to rewrite the entries of $\hat{v}$ as follows:

$$\hat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k = \frac{1}{\sqrt{N}} \Big(\sum_{k \text{ even}} \omega_N^{jk} v_k + \sum_{k \text{ odd}} \omega_N^{jk} v_k \Big)$$

$$= \frac{1}{\sqrt{2}} \Big(\frac{1}{\sqrt{N/2}} \sum_{k \text{ even}} \omega_{N/2}^{jk/2} v_k + \frac{\omega_N^j}{\sqrt{N/2}} \sum_{k \text{ odd}} \omega_{N/2}^{j(k-1)/2} v_k \Big).$$

$$\omega_N^{jk} = \exp\Big(\frac{2\pi jki}{N}\Big) = \exp\Big(\frac{2\pi j(k/2)i}{N/2}\Big) = \omega_{N/2}^{jk/2} \quad \text{if } k \text{ is even,}$$

$$\omega_N^{jk} = \omega_N^j \omega_N^{j(k-1)} = \omega_N^j \omega_{N/2}^{j(k-1)/2} \quad \text{if } k \text{ is odd.}$$

§5.2 The Fast Fourier Transform

We will assume $N = 2^n$, which is usually fine because we can add zeroes to our vector to make its dimension a power of 2 (but similar FFTs can be given also directly for most N that are not a power of 2). The key to the FFT is to rewrite the entries of $\widehat{v}$ as follows:

$$\widehat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k = \frac{1}{\sqrt{N}} \left(\sum_{k \text{ even}} \omega_N^{jk} v_k + \sum_{k \text{ odd}} \omega_N^{jk} v_k \right)$$

$$= \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{N/2}} \sum_{k \text{ even}} \omega_{N/2}^{jk/2} v_k + \frac{\omega_N^j}{\sqrt{N/2}} \sum_{k \text{ odd}} \omega_{N/2}^{j(k-1)/2} v_k \right).$$

$$\omega_N^{jk} = \exp\left(\frac{2\pi jki}{N}\right) = \exp\left(\frac{2\pi j(k/2)i}{N/2}\right) = \omega_{N/2}^{jk/2} \quad \text{if } k \text{ is even,}$$

$$\omega_N^{jk} = \omega_N^j \omega_N^{j(k-1)} = \omega_N^j \omega_{N/2}^{j(k-1)/2} \quad \text{if } k \text{ is odd.}$$

§5.2 The Fast Fourier Transform

We will assume $N = 2^n$, which is usually fine because we can add zeroes to our vector to make its dimension a power of 2 (but similar FFTs can be given also directly for most N that are not a power of 2). The key to the FFT is to rewrite the entries of $\widehat{v}$ as follows:

$$\widehat{v}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} v_k = \frac{1}{\sqrt{N}} \Big(\sum_{k \text{ even}} \omega_N^{jk} v_k + \sum_{k \text{ odd}} \omega_N^{jk} v_k \Big)$$

$$= \frac{1}{\sqrt{2}} \Big(\frac{1}{\sqrt{N/2}} \sum_{k \text{ even}} \omega_{N/2}^{jk/2} v_k + \frac{\omega_N^j}{\sqrt{N/2}} \sum_{k \text{ odd}} \omega_{N/2}^{j(k-1)/2} v_k \Big) .$$

$$\omega_N^{jk} = \exp\left(\frac{2\pi jki}{N}\right) = \exp\left(\frac{2\pi j(k/2)i}{N/2}\right) = \omega_{N/2}^{jk/2} \quad \text{if } k \text{ is even,}$$

$$\omega_N^{jk} = \omega_N^j \omega_N^{j(k-1)} = \omega_N^j \omega_{N/2}^{j(k-1)/2} \quad \text{if } k \text{ is odd.}$$

§5.2 The Fast Fourier Transform

Note that we have rewritten the entries of the N-dimensional discrete Fourier transform $\widehat{v}$ in terms of two $\frac{N}{2}$-dimensional discrete Fourier transforms, one of the even-numbered entries of v, and one of the odd-numbered entries of v. This suggests a recursive procedure for computing $\widehat{v}$: first separately compute the Fourier transform $\widehat{v_{\rm even}}$ of the $\frac{N}{2}$-dimensional vector of even-numbered entries of v and the Fourier transform $\widehat{v_{\rm odd}}$ of the $\frac{N}{2}$-dimensional vector of odd-numbered entries of v, and then compute the N entries using

$$\widehat{v}_j = \frac{1}{\sqrt{2}}\big[(\widehat{v_{\rm even}})_j + \omega_N^j(\widehat{v_{\rm odd}})_j\big] \qquad \forall\, 0 \leqslant j \leqslant \frac{N}{2}-1\,,$$

$$\widehat{v}_{j+\frac{N}{2}} = \frac{1}{\sqrt{2}}\big[(\widehat{v_{\rm even}})_j - \omega_N^j(\widehat{v_{\rm odd}})_j\big] \qquad \forall\, 0 \leqslant j \leqslant \frac{N}{2}-1\,.$$

§5.2 The Fast Fourier Transform

Note that we have rewritten the entries of the N-dimensional discrete Fourier transform $\hat{v}$ in terms of two $\frac{N}{2}$-dimensional discrete Fourier transforms, one of the even-numbered entries of v, and one of the odd-numbered entries of v. This suggests a recursive procedure for computing $\hat{v}$: first separately compute the Fourier transform $\widehat{v_{\text{even}}}$ of the $\frac{N}{2}$-dimensional vector of even-numbered entries of v and the Fourier transform $\widehat{v_{\text{odd}}}$ of the $\frac{N}{2}$-dimensional vector of odd-numbered entries of v, and then compute the N entries using

$$\hat{v}_j = \frac{1}{\sqrt{2}}\big[(\widehat{v_{\text{even}}})_j + \omega_N^j(\widehat{v_{\text{odd}}})_j\big] \qquad \forall\, 0 \leqslant j \leqslant \frac{N}{2} - 1\,,$$

$$\hat{v}_{j+\frac{N}{2}} = \frac{1}{\sqrt{2}}\big[(\widehat{v_{\text{even}}})_j - \omega_N^j(\widehat{v_{\text{odd}}})_j\big] \qquad \forall\, 0 \leqslant j \leqslant \frac{N}{2} - 1\,.$$

§5.2 The Fast Fourier Transform

The computation time $T(N)$ it takes to implement F_N this way can be written recursively as $T(N) = 2T(\frac{N}{2}) + 2N$, because we need to compute two $\frac{N}{2}$-dimensional Fourier transforms and do $2N$ additional operations (additions and multiplications) to compute $\widehat{v}$. This works out to time $T(N) = \mathcal{O}(N \log_2 N)$, as promised. Similarly, we have an equally efficient algorithm for the inverse discrete Fourier transform $F_N^{-1} = F_N^*$, whose entries are $\frac{1}{\sqrt{N}}\omega_N^{-jk}$.

§5.2 The Fast Fourier Transform

The computation time $T(N)$ it takes to implement F_N this way can be written recursively as $T(N) = 2T(\frac{N}{2}) + 2N$, because we need to compute two $\frac{N}{2}$-dimensional Fourier transforms and do $2N$ additional operations (additions and multiplications) to compute $\hat{v}$. This works out to time $T(N) = \mathcal{O}(N \log_2 N)$, as promised. Similarly, we have an equally efficient algorithm for the inverse discrete Fourier transform $F_N^{-1} = F_N^*$, whose entries are $\frac{1}{\sqrt{N}} \omega_N^{-jk}$.

§5.2 The Fast Fourier Transform

The computation time $T(N)$ it takes to implement F_N this way can be written recursively as $T(N) = 2T(\frac{N}{2}) + 2N$, because we need to compute two $\frac{N}{2}$-dimensional Fourier transforms and do $2N$ additional operations (additions and multiplications) to compute $\widehat{v}$. This works out to time $T(N) = \mathcal{O}(N \log_2 N)$, as promised. Similarly, we have an equally efficient algorithm for the inverse discrete Fourier transform $F_N^{-1} = F_N^*$, whose entries are $\frac{1}{\sqrt{N}} \omega_N^{-jk}$.

§5.3 Application: Multiplying Two Polynomials

Suppose we are given two real-valued polynomials p and q, each of degree at most d:

$$p(x) = \sum_{j=0}^{d} a_j x^j \qquad \text{and} \qquad q(x) = \sum_{k=0}^{d} b_k x^k .$$

We would like to compute the product of these two polynomials

$$p(x)q(x) = \Big(\sum_{j=0}^{d} a_j x^j\Big)\Big(\sum_{k=0}^{d} b_k x^k\Big) = \sum_{\ell=0}^{2d} \underbrace{\Big(\sum_{j=0}^{\ell} a_j b_{\ell-j}\Big)}_{c_\ell} x^\ell .$$

Clearly, each coefficient c_ℓ by itself takes $(2\ell+1)$ steps (additions and multiplications) to compute, which suggests an algorithm for computing the coefficients of $p\cdot q$ that takes $\mathcal{O}(d^2)$ steps. However, using the fast Fourier transform we can do this in $\mathcal{O}(d\log_2 d)$ steps, as follows.

§5.3 Application: Multiplying Two Polynomials

Suppose we are given two real-valued polynomials p and q, each of degree at most d:

$$p(x) = \sum_{j=0}^{d} a_j x^j \qquad \text{and} \qquad q(x) = \sum_{k=0}^{d} b_k x^k .$$

We would like to compute the product of these two polynomials

$$p(x)q(x) = \Big(\sum_{j=0}^{d} a_j x^j\Big)\Big(\sum_{k=0}^{d} b_k x^k\Big) = \sum_{\ell=0}^{2d} \underbrace{\Big(\sum_{j=0}^{\ell} a_j b_{\ell-j}\Big)}_{c_\ell} x^\ell .$$

Clearly, each coefficient c_ℓ by itself takes $(2\ell+1)$ steps (additions and multiplications) to compute, which suggests an algorithm for computing the coefficients of $p \cdot q$ that takes $\mathcal{O}(d^2)$ steps. However, using the fast Fourier transform we can do this in $\mathcal{O}(d \log_2 d)$ steps, as follows.

§5.3 Application: Multiplying Two Polynomials

Suppose we are given two real-valued polynomials p and q, each of degree at most d:

$$p(x) = \sum_{j=0}^{d} a_j x^j \qquad \text{and} \qquad q(x) = \sum_{k=0}^{d} b_k x^k .$$

We would like to compute the product of these two polynomials

$$p(x)q(x) = \Big(\sum_{j=0}^{d} a_j x^j\Big)\Big(\sum_{k=0}^{d} b_k x^k\Big) = \sum_{\ell=0}^{2d} \underbrace{\Big(\sum_{j=0}^{\ell} a_j b_{\ell-j}\Big)}_{c_\ell} x^\ell .$$

Clearly, each coefficient c_ℓ by itself takes $(2\ell+1)$ steps (additions and multiplications) to compute, which suggests an algorithm for computing the coefficients of $p \cdot q$ that takes $\mathcal{O}(d^2)$ steps. However, using the fast Fourier transform we can do this in $\mathcal{O}(d \log_2 d)$ steps, as follows.

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N} \cdot$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \ldots, N-1\}$ we have $(\widehat{a * b})_\ell = (\hat{a} .* \hat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{\ell k} (a * b)_k$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N} \,.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \ldots, N-1\}$ we have $(\widehat{a * b})_\ell = (\hat{a} .* \hat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{\ell k} (a * b)_k$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N}.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \ldots, N-1\}$ we have $(\widehat{a * b})_\ell = (\widehat{a} .* \widehat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{\ell k} \left(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(k-j) \bmod N} \right)$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N}.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \dots, N-1\}$ we have $(\widehat{a * b})_\ell = (\widehat{a} .* \widehat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{N} \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} \omega_N^{\ell j} \omega_N^{\ell(k-j)} a_j b_{(k-j) \bmod N}$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N} \,.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \ldots, N-1\}$ we have $(\widehat{a * b})_\ell = (\hat{a} .* \hat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{N} \sum_{j=0}^{N-1} \omega_N^{\ell j} a_j \sum_{k=0}^{N-1} \omega_N^{\ell(k-j)} b_{(k-j) \bmod N}$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell-j) \text{ mod } N}.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \dots, N-1\}$ we have $(\widehat{a * b})_\ell = (\hat{a} .* \hat{b})_\ell$:

$$(\widehat{a * b})_\ell = \frac{1}{N} \sum_{j=0}^{N-1} \omega_N^{\ell j} a_j \sum_{k=0}^{N-1} \omega_N^{\ell k} b_k$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell - j) \bmod N} \,.$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \ldots, N-1\}$ we have $(\widehat{a * b})_\ell = (\widehat{a} .* \widehat{b})_\ell$:

$$(\widehat{a * b})_\ell = \Big(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{\ell j} a_j \Big) \Big(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{\ell k} b_k \Big)$$

§5.3 Application: Multiplying Two Polynomials

The convolution of two vectors $a, b \in \mathbb{R}^N$ is a vector $a * b \in \mathbb{R}^N$ whose ℓ-th entry is defined by

$$(a * b)_\ell = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} a_j b_{(\ell-j) \bmod N} .$$

Let us set $N = 2d + 1$ (the number of nonzero coefficients of $p \cdot q$) and make the $(d+1)$-dimensional vectors of coefficients a and b N-dimensional by adding d zeroes. Then the coefficients of the polynomial $p \cdot q$ are proportional to the entries of the convolution: $c_\ell = \sqrt{N}(a * b)_\ell$. It is easy to show that the Fourier coefficients of the convolution of a and b are the products of the Fourier coefficients of a and b: for every $\ell \in \{0, \dots, N-1\}$ we have $(\widehat{a * b})_\ell = (\widehat{a} .* \widehat{b})_\ell$:

$$(\widehat{a * b})_\ell = \Big(\frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{\ell j} a_j\Big)\Big(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{\ell k} b_k\Big) = (\widehat{a} .* \widehat{b})_\ell .$$

§5.3 Application: Multiplying Two Polynomials

This immediately suggests an algorithm for computing the vector of coefficients c_ℓ: apply the FFT to a and b to get $\hat{a}$ and $\hat{b}$, multiply those two vectors entrywise to get $\hat{a} .* \hat{b}$, apply the inverse FFT to get $a * b$, and finally multiply $a * b$ with $\sqrt{N}$ to get the vector c of the coefficients of $p \cdot q$. Since the FFTs and their inverse take $\mathcal{O}(N \log_2 N)$ steps, and pointwise multiplication of two N-dimensional vectors takes $\mathcal{O}(N)$ steps, this whole algorithm takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(d \log_2 d)$ steps.

Note that if two numbers $a_d \cdots a_1 a_0$ and $b_d \cdots b_1 b_0$ are given in decimal notation, then we can interpret the digits as coefficients of polynomials p and q, respectively, and the two numbers will be $p(10)$ and $q(10)$. Their product is the evaluation of the product-polynomial $p \cdot q$ at the point $x = 10$.

§5.3 Application: Multiplying Two Polynomials

This immediately suggests an algorithm for computing the vector of coefficients c_ℓ: apply the FFT to a and b to get $\widehat{a}$ and $\widehat{b}$, multiply those two vectors entrywise to get $\widehat{a} .* \widehat{b}$, apply the inverse FFT to get $a * b$, and finally multiply $a * b$ with $\sqrt{N}$ to get the vector c of the coefficients of $p \cdot q$. Since the FFTs and their inverse take $\mathcal{O}(N \log_2 N)$ steps, and pointwise multiplication of two N-dimensional vectors takes $\mathcal{O}(N)$ steps, this whole algorithm takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(d \log_2 d)$ steps.

Note that if two numbers $a_d \cdots a_1 a_0$ and $b_d \cdots b_1 b_0$ are given in decimal notation, then we can interpret the digits as coefficients of polynomials p and q, respectively, and the two numbers will be $p(10)$ and $q(10)$. Their product is the evaluation of the product-polynomial $p \cdot q$ at the point $x = 10$.

§5.3 Application: Multiplying Two Polynomials

This immediately suggests an algorithm for computing the vector of coefficients c_ℓ: apply the FFT to a and b to get $\widehat{a}$ and $\widehat{b}$, multiply those two vectors entrywise to get $\widehat{a} .* \widehat{b}$, apply the inverse FFT to get $a \ast b$, and finally multiply $a \ast b$ with $\sqrt{N}$ to get the vector c of the coefficients of $p \cdot q$. Since the FFTs and their inverse take $\mathcal{O}(N \log_2 N)$ steps, and pointwise multiplication of two N-dimensional vectors takes $\mathcal{O}(N)$ steps, this whole algorithm takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(d \log_2 d)$ steps.

Note that if two numbers $a_d \cdots a_1 a_0$ and $b_d \cdots b_1 b_0$ are given in decimal notation, then we can interpret the digits as coefficients of polynomials p and q, respectively, and the two numbers will be $p(10)$ and $q(10)$. Their product is the evaluation of the product-polynomial $p \cdot q$ at the point $x = 10$.

§5.3 Application: Multiplying Two Polynomials

This suggests that we can use the above procedure (for fast multiplication of polynomials) to multiply two numbers in $\mathcal{O}(d\log_2 d)$ steps, which would be a lot faster than the standard $\mathcal{O}(d^2)$ algorithm for multiplication that one learns in primary school. However, in this case we have to be careful since the steps of the above algorithm are themselves multiplications between numbers, which we cannot count at unit cost anymore if our goal is to implement a multiplication between numbers! Still, it turns out that implementing this idea carefully allows one to multiply two d-digit numbers in $\mathcal{O}(d\log_2 d\log_2\log_2 d)$ elementary operations. This is known as the Schönhage-Strassen algorithm. We will skip the details.

§5.3 Application: Multiplying Two Polynomials

This suggests that we can use the above procedure (for fast multiplication of polynomials) to multiply two numbers in $\mathcal{O}(d \log_2 d)$ steps, which would be a lot faster than the standard $\mathcal{O}(d^2)$ algorithm for multiplication that one learns in primary school. However, in this case we have to be careful since the steps of the above algorithm are themselves multiplications between numbers, which we cannot count at unit cost anymore if our goal is to implement a multiplication between numbers! Still, it turns out that implementing this idea carefully allows one to multiply two d-digit numbers in $\mathcal{O}(d \log_2 d \log_2 \log_2 d)$ elementary operations. This is known as the Schönhage-Strassen algorithm. We will skip the details.

§5.3 Application: Multiplying Two Polynomials

This suggests that we can use the above procedure (for fast multiplication of polynomials) to multiply two numbers in $\mathcal{O}(d\log_2 d)$ steps, which would be a lot faster than the standard $\mathcal{O}(d^2)$ algorithm for multiplication that one learns in primary school. However, in this case we have to be careful since the steps of the above algorithm are themselves multiplications between numbers, which we cannot count at unit cost anymore if our goal is to implement a multiplication between numbers! Still, it turns out that implementing this idea carefully allows one to multiply two d-digit numbers in $\mathcal{O}(d\log_2 d\log_2\log_2 d)$ elementary operations. This is known as the Schönhage-Strassen algorithm. We will skip the details.

§5.4 The Quantum Fourier Transform

Since F_N is an $N \times N$ unitary matrix, we can interpret it as a quantum operation, mapping an N-dimensional vector of amplitudes to another N-dimensional vector of amplitudes. This is called the quantum Fourier transform (QFT). In case $N = 2^n$ (which is the only case we will care about), this will be an n-qubit unitary. We will see below that the QFT can be implemented by a quantum circuit using $\mathcal{O}(n^2)$ elementary gates. This is **exponentially faster** than even the FFT (which takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(2^n n)$ steps), but it achieves something different: computing the QFT will **NOT** give us the entries of the Fourier transform written down on a piece of paper, but only as the amplitudes of the resulting state.

§5.4 The Quantum Fourier Transform

Since F_N is an $N \times N$ unitary matrix, we can interpret it as a quantum operation, mapping an N-dimensional vector of amplitudes to another N-dimensional vector of amplitudes. This is called the quantum Fourier transform (QFT). In case $N = 2^n$ (which is the only case we will care about), this will be an n-qubit unitary. We will see below that the QFT can be implemented by a quantum circuit using $\mathcal{O}(n^2)$ elementary gates. This is **exponentially faster** than even the FFT (which takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(2^n n)$ steps), but it achieves something different: computing the QFT will **NOT** give us the entries of the Fourier transform written down on a piece of paper, but only as the amplitudes of the resulting state.

§5.4 The Quantum Fourier Transform

Since F_N is an $N \times N$ unitary matrix, we can interpret it as a quantum operation, mapping an N-dimensional vector of amplitudes to another N-dimensional vector of amplitudes. This is called the quantum Fourier transform (QFT). In case $N = 2^n$ (which is the only case we will care about), this will be an n-qubit unitary. We will see below that the QFT can be implemented by a quantum circuit using $\mathcal{O}(n^2)$ elementary gates. This is **exponentially faster** than even the FFT (which takes $\mathcal{O}(N \log_2 N) = \mathcal{O}(2^n n)$ steps), but it achieves something different: computing the QFT will **NOT** give us the entries of the Fourier transform written down on a piece of paper, but only as the amplitudes of the resulting state.

§5.4 The Quantum Fourier Transform

Definition

The N-dimnesional **quantum Fourier transform** F_N, where $N = 2^n$, is a linear map on the n-qubit space $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$ satisfying

$$F_N|k\rangle = \frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}\omega_N^{jk}|j\rangle \quad \forall\,|k\rangle = |k_1k_2\cdots k_n\rangle = |k_1\rangle\otimes\cdots\otimes|k_n\rangle,$$

where again $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$.

§5.4 The Quantum Fourier Transform

Theorem

Let $\phi_1, \cdots, \phi_n \in \mathbb{R}$. *For each* $n \in \mathbb{N}$,

$$\bigotimes_{\ell=1}^{n} \left(|0\rangle + e^{i\phi_\ell}|1\rangle\right) = \sum_{j=0}^{2^n-1} e^{i(j_1\phi_1+j_2\phi_2+\cdots+j_n\phi_n)}|j\rangle, \qquad (1)$$

where $|j\rangle = |j_1 j_2 \cdots j_n\rangle$ *for* $j \in \{0,1\}^n$.

Proof.

Since $|0\rangle + e^{i\phi_\ell}|1\rangle = \sum_{j_\ell=0}^{1} e^{ij_\ell\phi_\ell}|j_\ell\rangle$, we find that

$$\begin{aligned}\bigotimes_{\ell=1}^{n} \left(|0\rangle + e^{i\phi_\ell}|1\rangle\right) &= \left(\sum_{j_1=0}^{1} e^{ij_1\phi_1}|j_1\rangle\right) \otimes \cdots \otimes \left(\sum_{j_n=0}^{1} e^{ij_n\phi_n}|j_n\rangle\right) \\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1\rangle\otimes|j_2\rangle\otimes\cdots\otimes|j_n\rangle \\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1 j_2\cdots j_n\rangle.\end{aligned}$$

□

§5.4 The Quantum Fourier Transform

Theorem

Let $\phi_1, \cdots, \phi_n \in \mathbb{R}$. For each $n \in \mathbb{N}$,

$$\bigotimes_{\ell=1}^{n} \left(|0\rangle + e^{i\phi_\ell}|1\rangle\right) = \sum_{j=0}^{2^n-1} e^{i(j_1\phi_1+j_2\phi_2+\cdots+j_n\phi_n)}|j\rangle, \qquad (1)$$

where $|j\rangle = |j_1 j_2 \cdots j_n\rangle$ for $j \in \{0,1\}^n$.

Proof.

Since $|0\rangle + e^{i\phi_\ell}|1\rangle = \sum_{j_\ell=0}^{1} e^{ij_\ell\phi_\ell}|j_\ell\rangle$, we find that

$$\begin{aligned}\bigotimes_{\ell=1}^{n} \left(|0\rangle + e^{i\phi_\ell}|1\rangle\right) &= \Big(\sum_{j_1=0}^{1} e^{ij_1\phi_1}|j_1\rangle\Big) \otimes \cdots \otimes \Big(\sum_{j_n=0}^{1} e^{ij_n\phi_n}|j_n\rangle\Big) \\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1\rangle\otimes|j_2\rangle\otimes\cdots\otimes|j_n\rangle \\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1 j_2\cdots j_n\rangle.\end{aligned}$$

□

§5.4 The Quantum Fourier Transform

Theorem

Let $\phi_1, \cdots, \phi_n \in \mathbb{R}$. *For each* $n \in \mathbb{N}$,

$$\bigotimes_{\ell=1}^{n} \big(|0\rangle + e^{i\phi_\ell}|1\rangle\big) = \sum_{j=0}^{2^n-1} e^{i(j_1\phi_1+j_2\phi_2+\cdots+j_n\phi_n)}|j\rangle, \tag{1}$$

where $|j\rangle = |j_1 j_2 \cdots j_n\rangle$ *for* $j \in \{0,1\}^n$.

Proof.

Since $|0\rangle + e^{i\phi_\ell}|1\rangle = \sum_{j_\ell=0}^{1} e^{ij_\ell\phi_\ell}|j_\ell\rangle$, we find that

$$\begin{aligned}\bigotimes_{\ell=1}^{n} \big(|0\rangle + e^{i\phi_\ell}|1\rangle\big) &= \Big(\sum_{j_1=0}^{1} e^{ij_1\phi_1}|j_1\rangle\Big) \otimes \cdots \otimes \Big(\sum_{j_n=0}^{1} e^{ij_n\phi_n}|j_n\rangle\Big)\\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1\rangle\otimes|j_2\rangle\otimes\cdots\otimes|j_n\rangle\\ &= \sum_{j_1=0}^{1}\sum_{j_2=0}^{1}\cdots\sum_{j_n=0}^{1} e^{i(j_1\phi_1+j_2\phi_2+\cdots j_n\phi_n)}|j_1 j_2\cdots j_n\rangle.\end{aligned}$$

□

§5.4 The Quantum Fourier Transform

Definition

The N-dimnesional **quantum Fourier transform** F_N, where $N = 2^n$, is a linear map on the n-qubit space $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$ satisfying

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk} |j\rangle \quad \forall\, |k\rangle = |k_1 k_2 \cdots k_n\rangle = |k_1\rangle \otimes \cdots \otimes |k_n\rangle\,,$$

where again $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$.

Since $\exp\left(\frac{2\pi ijk}{2^n}\right) = \exp\left(i \sum_{\ell=1}^{n} \frac{2\pi k j_\ell}{2^\ell}\right)$ for $0 \leqslant j = (j_1 \cdots j_n)_2 \leqslant 2^n - 1$, using (1) we find that

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{2^n}} |j\rangle = \bigotimes_{\ell=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi ik}{2^\ell}} |1\rangle\right). \qquad (2)$$

§5.4 The Quantum Fourier Transform

Definition

The N-dimnesional **quantum Fourier transform** F_N, where $N = 2^n$, is a linear map on the n-qubit space $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$ satisfying

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk} |j\rangle \quad \forall\, |k\rangle = |k_1 k_2 \cdots k_n\rangle = |k_1\rangle \otimes \cdots \otimes |k_n\rangle,$$

where again $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$.

Since $\exp\left(\frac{2\pi ijk}{2^n}\right) = \exp\left(i \sum_{\ell=1}^{n} \frac{2\pi k j_\ell}{2^\ell}\right)$ for $0 \leqslant j = (j_1 \cdots j_n)_2 \leqslant 2^n - 1$, using (1) we find that

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{2^n}} |j\rangle = \bigotimes_{\ell=1}^{n} \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\frac{2\pi ik}{2^\ell}} |1\rangle\right). \qquad (2)$$

§5.4 The Quantum Fourier Transform

Definition

The N-dimnesional **quantum Fourier transform** F_N, where $N = 2^n$, is a linear map on the n-qubit space $\{|0\rangle, |1\rangle, \cdots, |N-1\rangle\}$ satisfying

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} \omega_N^{jk}|j\rangle \quad \forall\, |k\rangle = |k_1 k_2 \cdots k_n\rangle = |k_1\rangle \otimes \cdots \otimes |k_n\rangle,$$

where again $\omega_N = \exp\left(\frac{2\pi i}{N}\right)$.

Since $\exp\left(\frac{2\pi ijk}{2^n}\right) = \exp\left(i \sum_{\ell=1}^{n} \frac{2\pi k j_\ell}{2^\ell}\right)$ for $0 \leqslant j = (j_1 \cdots j_n)_2 \leqslant 2^n - 1$, using (1) we find that

$$F_N|k\rangle = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{2^n}}|j\rangle = \bigotimes_{\ell=1}^{n} \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\frac{2\pi ik}{2^\ell}}|1\rangle\right). \qquad (2)$$

§5.4 The Quantum Fourier Transform

Using the convection $0.b_1b_2\cdots b_m = \sum_{\ell=1}^{m} b_\ell 2^{-\ell}$ for $b = b_1b_2\cdots b_m \in \{0,1\}^m$ $\left(\text{for example, } 0.101 = 1\cdot\frac{1}{2} + 0\cdot\frac{1}{4} + 1\cdot\frac{1}{8} = \frac{5}{8}\right)$, by the fact that $e^{2\pi i} = 1$ we have

$$\begin{aligned}
\exp\left(\frac{2\pi i k}{2^\ell}\right) &= \exp\left(2\pi i \sum_{j=1}^{n} k_j 2^{n-j-\ell}\right) = \exp\left(2\pi i \sum_{j=n-\ell+1}^{n} k_j 2^{n-j-\ell}\right) \\
&= \exp\left(2\pi i \sum_{m=1}^{\ell} k_{n-\ell+m} 2^{-m}\right) \\
&= \exp\left(2\pi i 0.k_{n-\ell+1}k_{n-\ell+2}\cdots k_n\right)
\end{aligned}$$

so that (2) implies that

$$F_N|k\rangle = \bigotimes_{\ell=1}^{n} \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.k_{n-\ell+1}\cdots k_n}|1\rangle\right). \qquad (3)$$

§5.5 An Efficient Quantum Circuit

In the following, we will describe the efficient circuit for the n-qubit QFT. The elementary gates we will allow ourselves are Hadamards and controlled-R_s gates, where

$$\mathrm{R}_s = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^s} \end{bmatrix}.$$

Note that $\mathrm{R}_1 = \mathrm{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $\mathrm{R}_2 = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$, and

$$\mathrm{R}_s|k\rangle = e^{2\pi i \frac{k}{2^s}}|k\rangle \qquad \forall\, k \in \{0, 1\}.$$

For large s, $e^{2\pi i/2^s}$ is close to 1 and hence the R_s-gate is close to the identity-gate I. We could implement R_s-gates using Hadamards and controlled-R_s gates for $s = 1, 2, 3$, but for simplicity we will just treat each R_s as an elementary gate.

§5.5 An Efficient Quantum Circuit

In the following, we will describe the efficient circuit for the n-qubit QFT. The elementary gates we will allow ourselves are Hadamards and controlled-R_s gates, where

$$\mathrm{R}_s = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^s} \end{bmatrix}.$$

Note that $\mathrm{R}_1 = \mathrm{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $\mathrm{R}_2 = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$, and

$$\mathrm{R}_s|k\rangle = e^{2\pi i \frac{k}{2^s}}|k\rangle \qquad \forall\, k \in \{0,1\}.$$

For large s, $e^{2\pi i/2^s}$ is close to 1 and hence the R_s-gate is close to the identity-gate I. We could implement R_s-gates using Hadamards and controlled-R_s gates for $s = 1, 2, 3$, but for simplicity we will just treat each R_s as an elementary gate.

§5.5 An Efficient Quantum Circuit

Example

In this example we illustrate how to construct the quantum circuit of F_8. Using (3),

$$F_8|k_1k_2k_3\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.k_3}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.k_2k_3}|1\rangle\right) \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.k_1k_2k_3}|1\rangle\right).$$

1. To prepare the first qubit of the desired state $F_8|k_1k_2k_3\rangle$, just apply a Hadamard to $|k_3\rangle$ since

$$\mathrm{H}|k_3\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + (-1)^{k_3}|1\rangle\right) = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.k_3}|1\rangle\right).$$

§5.5 An Efficient Quantum Circuit

Example (cont.)

2. To prepare the second qubit of the desired state, we first apply a Hadamard to $|k_2\rangle$ to obtain $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_2}|1\rangle\big)$, and then conditioned on k_3 (before we apply the Hadamard to $|k_3\rangle$) apply R_2: by applying R_2 it multiplies $|1\rangle$ with a phase $e^{2\pi i 0.0k_3}$, producing the correct qubit $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_2 k_3}|1\rangle\big)$.
3. To prepare the third qubit of the desired state, we apply a Hadamard to $|k_1\rangle$, apply R_2 conditioned on k_2 and R_3 conditioned k_3. This produces the correct qubit $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_1 k_2 k_3}|1\rangle\big)$.

§5.5 An Efficient Quantum Circuit

Example (cont.)

2. To prepare the second qubit of the desired state, we first apply a Hadamard to $|k_2\rangle$ to obtain $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_2}|1\rangle\big)$, and then conditioned on k_3 (before we apply the Hadamard to $|k_3\rangle$) apply R_2: by applying R_2 it multiplies $|1\rangle$ with a phase $e^{2\pi i 0.0k_3}$, producing the correct qubit $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_2k_3}|1\rangle\big)$.
3. To prepare the third qubit of the desired state, we apply a Hadamard to $|k_1\rangle$, apply R_2 conditioned on k_2 and R_3 conditioned k_3. This produces the correct qubit $\frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.k_1k_2k_3}|1\rangle\big)$.

§5.5 An Efficient Quantum Circuit

Example (cont.)

Note that the order of the output is wrong: the first qubit should be the third and vice versa. So the final step is just to swap qubits 1 and 3. Therefore, F_8 can be achieved by the following quantum circuit:

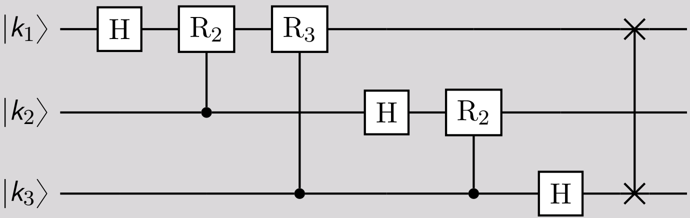

Figure 1: QFT for 3-qubits

§5.5 An Efficient Quantum Circuit

The general case works analogously: starting with $\ell = 1$, we apply a Hadamard to $|k_\ell\rangle$ and then "rotate in" the additional phases required, conditioned on the values of the later bits $k_{\ell+1}, \cdots, k_n$.

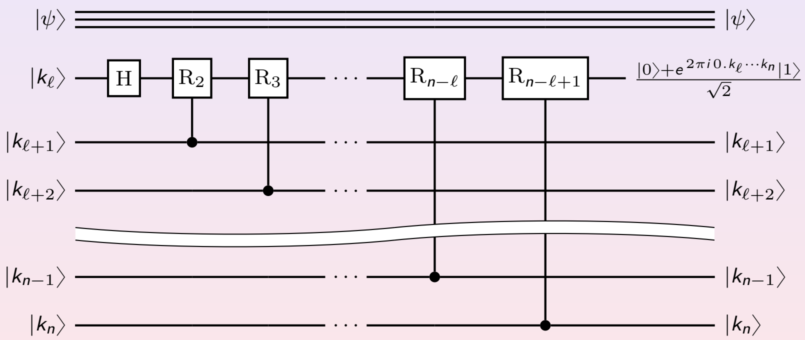

Figure 2: The ℓ-th block of QFT for n-qubits, where $|\psi\rangle$ is a $(\ell - 1)$ qubit quantum state

§5.5 An Efficient Quantum Circuit

Some swap gates at the end then put the qubits in the right order.

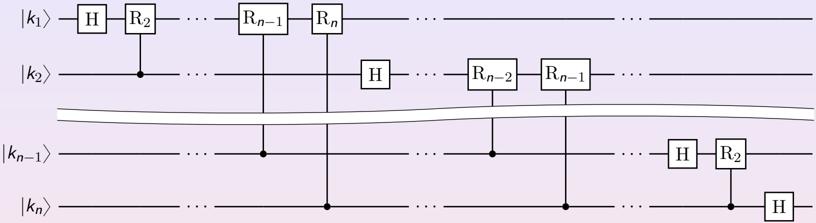

Figure 3: The quantum circuit of QFT for n-qubits (finally one should apply an order reverse operator)

§5.5 An Efficient Quantum Circuit

Since the circuit involves n qubits, and at most n gates are applied to each qubit, the overall circuit uses at most n^2 gates. In fact, many of those gates are phase gates R_s with $s \gg \log_2 n$, which are very close to the identity and hence do not do much anyway. We can actually omit those from the circuit, keeping only $\mathcal{O}(\log_2 n)$ gates per qubit and $\mathcal{O}(n \log_2 n)$ gates overall. Intuitively, the overall error caused by these omissions will be small (a homework exercise asks you to make this precise). Finally, note that by inverting the circuit (that is, reversing the order of the gates and taking the adjoint U^* of each gate U) we obtain an equally efficient circuit for the inverse quantum Fourier transform $F_N^{-1} = F_N^*$.

§5.5 An Efficient Quantum Circuit

Since the circuit involves n qubits, and at most n gates are applied to each qubit, the overall circuit uses at most n^2 gates. In fact, many of those gates are phase gates R_s with $s \gg \log_2 n$, which are very close to the identity and hence do not do much anyway. We can actually omit those from the circuit, keeping only $\mathcal{O}(\log_2 n)$ gates per qubit and $\mathcal{O}(n \log_2 n)$ gates overall. Intuitively, the overall error caused by these omissions will be small (a homework exercise asks you to make this precise). Finally, note that by inverting the circuit (that is, reversing the order of the gates and taking the adjoint U^* of each gate U) we obtain an equally efficient circuit for the inverse quantum Fourier transform $F_N^{-1} = F_N^*$.

§5.5 An Efficient Quantum Circuit

Since the circuit involves n qubits, and at most n gates are applied to each qubit, the overall circuit uses at most n^2 gates. In fact, many of those gates are phase gates R_s with $s \gg \log_2 n$, which are very close to the identity and hence do not do much anyway. We can actually omit those from the circuit, keeping only $\mathcal{O}(\log_2 n)$ gates per qubit and $\mathcal{O}(n \log_2 n)$ gates overall. Intuitively, the overall error caused by these omissions will be small (a homework exercise asks you to make this precise). Finally, note that by inverting the circuit (that is, reversing the order of the gates and taking the adjoint U^* of each gate U) we obtain an equally efficient circuit for the inverse quantum Fourier transform $F_N^{-1} = F_N^*$.

§5.5 An Efficient Quantum Circuit

Since the circuit involves n qubits, and at most n gates are applied to each qubit, the overall circuit uses at most n^2 gates. In fact, many of those gates are phase gates R_s with $s \gg \log_2 n$, which are very close to the identity and hence do not do much anyway. We can actually omit those from the circuit, keeping only $\mathcal{O}(\log_2 n)$ gates per qubit and $\mathcal{O}(n \log_2 n)$ gates overall. Intuitively, the overall error caused by these omissions will be small (a homework exercise asks you to make this precise). Finally, note that by inverting the circuit (that is, reversing the order of the gates and taking the adjoint U^* of each gate U) we obtain an equally efficient circuit for the inverse quantum Fourier transform $F_N^{-1} = F_N^*$.

§5.6 Application: phase estimation

Suppose we can apply a unitary U and we are given an eigenvector $|\psi\rangle$ of U corresponding to an unknown eigenvalue λ (that is, $U|\psi\rangle = \lambda|\psi\rangle$ for some unknown $\lambda \in \mathbb{C}$), and we would like to compute or at least approximate the λ. Since U is unitary, λ must have magnitude 1, so we can write it as $\lambda = e^{2\pi i\phi}$ for some real number $\phi \in [0, 1)$; the only thing that matters is this phase ϕ.

Suppose for simplicity that we know that $\phi = 0.\phi_1\phi_2\cdots\phi_n$ can be written exactly with n bits of precision. Then here's the algorithm for phase estimation:

1. Start with $|0^n\rangle|\psi\rangle$.
2. For $N = 2^n$, apply F_N to the first n qubits to get $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}|j\rangle|\psi\rangle$ (in fact, $\mathrm{H}^n \otimes \mathrm{I}$ would have the same effect).

§5.6 Application: phase estimation

Suppose we can apply a unitary U and we are given an eigenvector $|\psi\rangle$ of U corresponding to an unknown eigenvalue λ (that is, $U|\psi\rangle = \lambda|\psi\rangle$ for some unknown $\lambda \in \mathbb{C}$), and we would like to compute or at least approximate the λ. Since U is unitary, λ must have magnitude 1, so we can write it as $\lambda = e^{2\pi i\phi}$ for some real number $\phi \in [0, 1)$; the only thing that matters is this phase ϕ.

Suppose for simplicity that we know that $\phi = 0.\phi_1\phi_2\cdots\phi_n$ can be written exactly with n bits of precision. Then here's the algorithm for phase estimation:

1. Start with $|0^n\rangle|\psi\rangle$.
2. For $N = 2^n$, apply F_N to the first n qubits to get $\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}|j\rangle|\psi\rangle$ (in fact, $\mathrm{H}^n \otimes \mathrm{I}$ would have the same effect).

§5.6 Application: phase estimation

Suppose we can apply a unitary U and we are given an eigenvector $|\psi\rangle$ of U corresponding to an unknown eigenvalue λ (that is, $U|\psi\rangle = \lambda|\psi\rangle$ for some unknown $\lambda \in \mathbb{C}$), and we would like to compute or at least approximate the λ. Since U is unitary, λ must have magnitude 1, so we can write it as $\lambda = e^{2\pi i\phi}$ for some real number $\phi \in [0, 1)$; the only thing that matters is this phase ϕ.

Suppose for simplicity that we know that $\phi = 0.\phi_1\phi_2 \cdots \phi_n$ can be written exactly with n bits of precision. Then here's the algorithm for phase estimation:

1. Start with $|0^n\rangle|\psi\rangle$.
2. For $N = 2^n$, apply F_N to the first n qubits to get $\dfrac{1}{\sqrt{N}} \sum_{j=0}^{N-1} |j\rangle|\psi\rangle$ (in fact, $\mathrm{H}^n \otimes \mathrm{I}$ would have the same effect).

§5.6 Application: phase estimation

3. Apply the map $|j\rangle|\psi\rangle \mapsto |j\rangle U^j|\psi\rangle$. In other words, apply U to the second register for a number of times given by the first register.
4. Apply the inverse Fourier transform F_N^{-1} to the first n qubits and measure the result.

Note that after step 3, the first n qubits are in state

$$\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} e^{2\pi i\phi j}|j\rangle,$$

hence the inverse quantum Fourier transform is going to give us $|2^n\phi\rangle = |\phi_1\cdots\phi_n\rangle$ with probability 1. In case ϕ cannot be written exactly with n bits of precision, one can show that this procedure still (with high probability) spits out a good n-bit approximation to ϕ. We will omit the calculation.

§5.6 Application: phase estimation

3. Apply the map $|j\rangle|\psi\rangle \mapsto |j\rangle U^j|\psi\rangle$. In other words, apply U to the second register for a number of times given by the first register.
4. Apply the inverse Fourier transform F_N^{-1} to the first n qubits and measure the result.

Note that after step 3, the first n qubits are in state

$$\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} e^{2\pi i\phi j}|j\rangle,$$

hence the inverse quantum Fourier transform is going to give us $|2^n\phi\rangle = |\phi_1\cdots\phi_n\rangle$ with probability 1. In case ϕ cannot be written exactly with n bits of precision, one can show that this procedure still (with high probability) spits out a good n-bit approximation to ϕ. We will omit the calculation.

§5.6 Application: phase estimation

3. Apply the map $|j\rangle|\psi\rangle \mapsto |j\rangle U^j|\psi\rangle$. In other words, apply U to the second register for a number of times given by the first register.
4. Apply the inverse Fourier transform F_N^{-1} to the first n qubits and measure the result.

Note that after step 3, the first n qubits are in state

$$\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1} e^{2\pi i\phi j}|j\rangle,$$

hence the inverse quantum Fourier transform is going to give us $|2^n\phi\rangle = |\phi_1\cdots\phi_n\rangle$ with probability 1. In case ϕ cannot be written exactly with n bits of precision, one can show that this procedure still (with high probability) spits out a good n-bit approximation to ϕ. We will omit the calculation.

§5.6 Application: phase estimation

Definition

Let $U \in \mathrm{U}(2^m)$ be an $2^m \times 2^m$ unitary matrix and let $|\psi\rangle$ be one of the eigenvector of U with corresponding eigenvalue $e^{2\pi i\theta}$. The Quantum Phase Estimation algorithm, abbreviated **QPE**, takes the inputs the m-qubit quantum gate for U and the state $|0^n\rangle|\psi\rangle$ and returns the state $|\widetilde{\theta}\rangle|\psi\rangle$, where $\widetilde{\theta}$ denotes a binary approximation to $2^n\theta$ and the n subscript denotes it has been truncated to n digits. In notation, with $[\cdot]$ denoting the Gauss/floor function,

$$\mathbf{QPE}(U, |0^n\rangle|\psi\rangle) = |\widetilde{\theta}\rangle|\psi\rangle\,, \qquad \widetilde{\theta} = [2^n\theta]\,.$$

We will use $|\theta\rangle_n$ to denote $|\widetilde{\theta}\rangle$ if $\widetilde{\theta} = \big[2^n\theta\big]$.