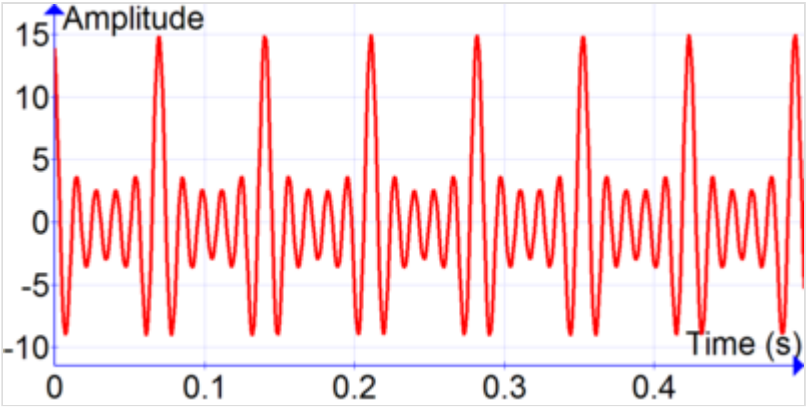
快速傅里叶变换

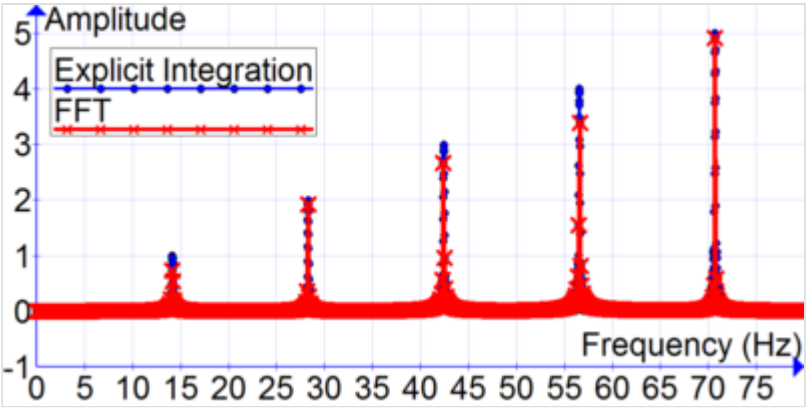
维基百科，自由的百科全书

快速傅里叶变换（英語：**Fast Fourier Transform, FFT**），是快速计算序列的离散傅里叶变换（DFT）或其逆变换的方法[1]。傅里叶分析将信号从原始域（通常是时间或空间）转换到频域的表示或者逆过来转换。FFT会通过把DFT矩阵分解为稀疏（大多为零）因子之积来快速计算此类变换。[2] 因此，它能够将计算DFT的复杂度从只用DFT定义计算需要的 $O(n^2)$，降低到 $O(n \log n)$，其中 n 为数据大小。



一个五项余弦级数的时域信号。频率为 $10\sqrt{2}$ 的倍数。振幅为 1, 2, 3, 4, 5。时间步长为 0.001 s

快速傅里叶变换广泛的应用于工程、科学和数学领域。这里的基本思想在1965年才得到普及，但早在1805年就已推导出来。[3] 1994年美國數學家吉爾伯特·斯特朗把FFT描述为“我们一生中最重要的数值算法”[4]，它还被IEEE科学与工程计算期刊列入20世纪十大算法。[5]



用FFT（~40,000次运算）五项余弦级数及用显式积分（~1亿次运算）得出的DFT。时间窗口是10秒。FFT是用FFTW3（http://www.fftw.org/ （页面存档备份 (https://web.archive.org/web/20190925140738/http://www.fftw.org/)，存于互联网档案馆））计算的。显式积分DFT是用 https://sourceforge.net/projects/amoreaccuratefouriertransform/ （页面存档备份 (https://web.archive.org/web/20160613013603/https://sourceforge.net/projects/amoreaccuratefouriertransform/)，存于互联网档案馆） 计算的。

定义和速度

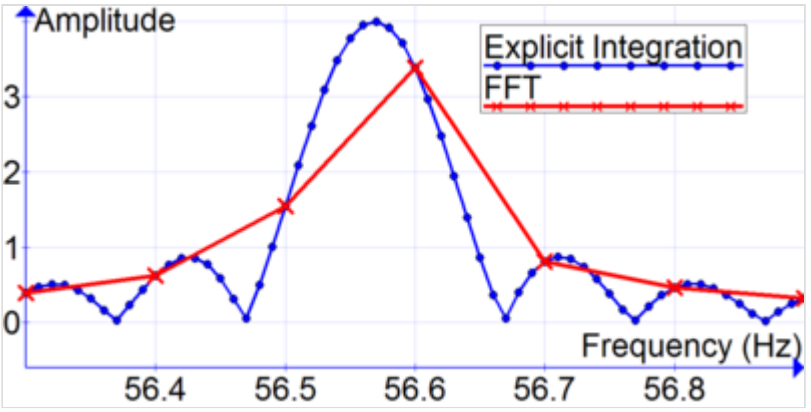
用FFT计算DFT会得到与直接用DFT定义计算相同的结果；最重要的区别是FFT更快。（由于捨入誤差的存在，许多FFT算法还会比直接运用定义求值精确很多，后面会讨论到这一点。）

令 $x_0, ..., x_{N-1}$ 為复数。DFT由下式定义

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1.$$

直接按这个定义求值需要 O(N^2) 次运算：X_k 共有 N 个输出，每个输出需要 N 项求和。直接使用DFT運算需使用N個複數乘法(4N 個實數乘法)與N-1個複數加法(2N-2個實數加法)，因此，計算使用DFT所有N點的值需要N^2複數乘法與N^2-N 個複數加法。FFT则是能够在 O(N log N) 次操作计算

出相同结果的任何方法。更准确的说，所有已知的FFT算法都需要O(*N* log *N*)次运算（技术上O只标记上界），虽然还没有已知的证据证明更低的复杂度是不可能的。[6]



缩放后的五项余弦级数的DFT。注意到显式积分更细的步长大小比FFT更精确地再现了峰值（4）和频率（56.569 Hz），代价是速度慢了上千倍。

要说明FFT节省时间的方式，就得考虑复数相乘和相加的次数。直接计算DFT的值涉及到 N^2 次复数相乘和 $N(N-1)$ 次复数相加（可以通过削去琐碎运算（如乘以1）来节省O(*N*)次运算）。众所周知的基2库利-图基算法，*N* 为2的幂，可以只用 $(N/2)\log_2(N)$ 次复数乘法（再次忽略乘以1的简化）和 $N\log_2(N)$ 次加法就可以得到相同结果。在实际中，现代计算机通常的实际性能通常不受算术运算的速度和对复杂主体的分析主导[7]，但是从 $O(N^2)$ 到 $O(N \log N)$ 的总体改进仍然能够体现出来。

一般的簡化理論

假設一個M*N型矩阵S可分解成列向量以及行向量相乘：

$$\mathbf{S} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$$

若$\begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix}^T$有$M_0$個相異的非平凡值（$a_m \neq \pm 2^k, a_m \neq \pm 2^k a_n$ where $m \neq n$）

$\begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix}$有$N_0$個相異的非平凡值

則S共需要$M_0 * N_0$個乘法。

$$\begin{bmatrix} Z[1] \\ Z[2] \\ \vdots \\ Z[N] \end{bmatrix} = \mathbf{S} \begin{bmatrix} X[1] \\ X[2] \\ \vdots \\ X[N] \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} \begin{bmatrix} X[1] \\ X[2] \\ \vdots \\ X[N] \end{bmatrix}$$

Step 1：$Z_a = b_1 X[1] + b_2 X[2] + \cdots + b_n X[N]$

Step 2：$Z[1] = a_1 Z_a, Z[2] = a_2 Z_a, \cdots, Z[N] = a_m Z_a$

簡化理論的變型：

$$\mathbf{S} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} \begin{bmatrix} b_1 & b_2 & \cdots & b_n \end{bmatrix} + \mathbf{S_1}$$

S_1也是一個M*N的矩陣。

若S_1有P_1個值不等於0，則$\mathbf{S}$的乘法量上限為$M_0 + N_0 + P_1$。

快速傅立葉變換乘法量的計算

假設$N = P_1 \times P_2 \times \cdots \times P_k$，其中$P_1, P_2, \cdots, P_k$彼此互質

$\mathbf{P_k}$點DFT的乘法量為$\mathbf{B_k}$，則$\mathbf{N}$點DFT的乘法量為：

$$\frac{N}{P_1}B_1 + \frac{N}{P_2}B_2 + \cdots\cdots + \frac{N}{P_k}B_k$$

假設$\mathbf{N = P^c}$，P是一個質數。

若$\mathbf{N_1 = P^a}$點的DFT需要的乘法量為$\mathbf{B_1}$

且$n_1 \times n_2$當中（$n_1 = 0, 1, \cdots, N_1 - 1, \quad n_2 = 0, 1, \cdots, N_2 - 1$）

有D_1個值不為$\frac{N}{12}$及$\frac{N}{8}$的倍數，

有D_2個值為$\frac{N}{12}$及$\frac{N}{8}$的倍數，但不為$\frac{N}{4}$的倍數，

則N點DFT的乘法量為：

$$\mathbf{N_2B_1 + N_1B_2 + 3D_1 + 2D_2}$$

库利-图基算法

库利-图基算法是最常见的FFT算法。这一方法以分治法为策略递归地将长度为$N = N_1N_2$的离散傅里叶变换分解为长度为N_1的N_2个较短序列的离散傅里叶变换，以及与$\mathrm{O}(N)$个旋转因子的复数乘法。

这种方法以及FFT的基本思路在1965年J. W. Cooley和J. W. Tukey合作发表*An algorithm for the machine calculation of complex Fourier series*之后开始为人所知。但后来发现，实际上这两位作者只是重新发明了高斯在1805年就已经提出的算法（此算法在历史上数次以各种形式被再次提出）。

库利-图基算法最有名的应用，是将序列长为*N* 的DFT分割为两个长为*N/2* 的子序列的DFT，因此这一应用只适用于序列长度为2的幂的DFT计算，即基2-FFT。实际上，如同高斯和Cooley与Tukey都指出的那样，Cooley-Tukey算法也可以用于序列长度*N* 为任意因数分解形式的DFT，即混合基FFT，而且还可以应用于其他诸如分裂基FFT等变种。尽管Cooley-Tukey算法的基本思路是采用递

归的方法进行计算，大多数传统的算法实现都将显式的递归算法改写为非递归的形式。另外，因为Cooley-Tukey算法是将DFT分解为较小长度的多个DFT，因此它可以同任一种其他的DFT算法联合使用。

设计思想

下面，我们用**N次单位根**W_N来表示$e^{-j\frac{2\pi}{N}}$。

W_N的性质：

1. **周期性**，W_N具有周期N，即$W_N^{k+N}=W_N^k$
2. **对称性**：$W_N^{k+\frac{N}{2}}=-W_N^k$。
3. 若m是N的约数，$W_N^{mkn}=W_{\frac{N}{m}}^{kn}$

为了简单起见，我们下面设待变换序列长度$n=2^r$。根据上面单位根的对称性，求级数$y_k=\sum_{n=0}^{N-1}W_N^{kn}x_n$时，可以将求和区间分为两部分：

$$
\begin{aligned}
y_k &= \textstyle\sum_{n=2t} W_N^{kn}x_n + \sum_{n=2t+1} W_N^{kn}x_n \\
&= \textstyle\sum_t W_{\frac{N}{2}}^{kt}x_{2t} + W_N^k \sum_t W_{\frac{N}{2}}^{kt}x_{2t+1} \\
&= F_{even}(k) + W_N^k F_{odd}(k) \qquad\qquad (i\in\mathbb{Z})
\end{aligned}
$$

$F_{odd}(k)$和$F_{even}(k)$是两个分别关于序列$\{x_n\}_0^{N-1}$奇数号和偶数号序列N/2点变换。由此式只能计算出y_k的前N/2个点，对于后N/2个点，注意$F_{odd}(k)$和$F_{even}(k)$都是周期为N/2的函数，由单位根的对称性，于是有以下变换公式：

- $y_{k+\frac{N}{2}}=F_{even}(k)-W_N^kF_{odd}(k)$
- $y_k=F_{even}(k)+W_N^kF_{odd}(k)$。

这样，一个N点变换就分解成了两个N/2点变换。照这样可继续分解下去。这就是**库利-图基快速傅里叶变换**算法的基本原理。根据主定理不难分析出此时算法的时间复杂度为$\mathrm{O}(N\log N)$

算法实现

- 蝶形结网络和位反转（Bit Reversal）：

  首先将$n=2^N$个输入点列按二进制进行编号，然后对各个编号按位倒置并按此重新排序。例如，对于一个8点变换，
  001倒置以后变成 100
  000 → 000
  001 → 100
  010 → 010
  011 → 110
  100 → 001
  101 → 101

110 → 011
111 → 111
倒置后的编号为{0,4,2,6,1,5,3,7}。
然后将这n个点列作为输入传送到蝶形结网络中，注意将因子W_N^k逐层加入到蝶形网络中。

算法复杂度

由于按蝶形结网络计算n点变换要进行log *n*层计算，每层计算n个点的变换，故算法的时间复杂度为$O(n \log n)$。

其他算法

在Cooley-Tukey算法之外还有其他DFT的快速演算法。对于长度$N = N_1 N_2$且N_1与N_2互质的序列，可以采用基于中国剩余定理的互质因子算法将N长序列的DFT分解为两个子序列的DFT。与Cooley-Tukey算法不同的是，互质因子算法不需要旋转因子。

Rader-Brenner算法是类似于Cooley-Tukey算法，但是采用的旋转因子都是纯虚数，以增加加法运算和降低了数值稳定性为代价减少了乘法运算。这方法之后被split-radix variant of Cooley-Tukey所取代，与Rader-Brenner演算法相比，有一样多的乘法量，却有较少的加法量，且不牺牲数值的准确性。

Bruun以及QFT演算法是不断的把DFT分成许多较小的DFT运算。（Rader-Brenner以及QFT演算法是为了2的指数所设计的演算法，但依然可以适用在可分解的整数上。Bruun演算法则可以运用在可被分成偶数个运算的数字）。尤其是Bruun演算法，把FFT看成是$z^N - 1$，并把它分解成z^{M-1}与$z^{2M} + az^M + 1$的形式。

另一个从多项式观点的快速傅立叶变换法是Winograd算法。此演算法把$z^N - 1$分解成cyclotomic多项式，而这些多项式的系数通常为1，0，-1。这样只需要很少的乘法量（如果有需要的话），所以winograd是可以得到最少乘法量的快速傅立叶演算法，对于较小的数字，可以找出有效率的算方式。更精确地说，winograd演算法让DFT可以用2^k点的DFT来简化，但减少乘法量的同时，也增加了非常多的加法量。Winograd也可以利用剩余值定理来简化DFT。

Rader演算法提出了利用点数为N（N为质数）的DFT进行长度为N-1的回旋摺积来表示原本的DFT，如此就可利用摺积用一对基本的FFT来计算DFT。另一个prime-size的FFT演算法为chirp-Z演算法。此法也是将DFT用摺积来表示，此法与Rader演算法相比，能运用在更一般的转换上，其转换的基础为Z转换（Rabiner et al., 1969）。

实数或对称资料专用的演算法

在许多的运用当中，要进行DFT的资料是纯实数，如此一来经过DFT的结果会满足对称性：

$$\mathbf{X}_{N-k} = \mathbf{X}_k^*$$

而有一些演算法是专门为这种情形设计的（e.g. Sorensen, 1987）。另一些则是利用旧有的演算法（e.g. Cooley-Tukey），删去一些不必要的演算步骤，如此省下了记忆体的使用，也省下了时间。另一方面，也可以把一个偶数长度且纯实数的DFT，用长度为原本一半的复数型态DFT来表示（实数项为原本纯实数资料的偶数项，虚数项则为奇数项）。

在Matlab上用一次N點FFT計算兩個N點實數信號x,y的FFT:

```
function [X,Y] = Real2FFT( x,y )
% N-point FFT is used for 2 real signals both with length N
N = length(x);
z = x+y*i ;
Z = fft(z);
X = 0.5*(Z+conj(Z(1+mod(0:-1:1-N,N))));
Y = 0.5*(Z-conj(Z(1+mod(0:-1:1-N,N))))/i;
end
```

一度人们认为，用离散哈特利转换（Discrete Hartley Transform）来处理纯实数的DFT会更有效率，但接着人们发现，对于同样点数的纯实数DFT，经过设计的FFT，可以比DHT省下更多的运算。Bruun演算法是第一个试着从减少实数输入的DFT运算量的演算法，但此法并没有成为人们普遍使用的方法。

对于实数输入，且输入为偶对称或奇对称的情形，可以更进一步的省下时间以及记忆体，此时DFT可以用离散余弦转换或离散正弦转换来代替（Discrete cosine/sine transforms）。由于DCT/DST也可以设计出FFT的演算法，故在此种情形下，此方法取代了对DFT设计的FFT演算法。

DFT可以应用在频谱分析以及做摺积的运算，而在此处，不同应用可以用不同的演算法来取代，列表如下：

用来做频谱分析的情况下，DFT可用下列的演算法代替：

- DCT
- DST
- DHT
- 正交基底的扩展（orthogonal basis expantion）包括正交多项式（orthogonal polynomials）以及CDMA.
- Walsh（Hadamard）转换
- Haar转换
- 小波（wavelet）转换
- 时频分布（time-frequency distribution）

用来做摺积的情况下，DFT可用下列的演算法代替：

- DCT
- DST
- DHT
- 直接做摺积（direct computing）
- 分段式DFT摺积（sectioned DFT convolution）
- 威諾格拉德快速傅立葉變換演算法
- 沃尔什（Walsh、Hadamard）转换

- 数论转换

複雜度以及運算量的極限

長久以來，人們對於求出快速傅立葉變換的複雜度下限以及需要多少的運算量感到很有興趣，而實際上也還有許多問題需要解決。即使是用較簡單的情形，即2^k點的DFT，也還沒能夠嚴謹的證明出FFT至少需要$\Omega(N \log N)$（不比$N \log N$小）的運算量，目前也沒有發現複雜度更低的演算法。通常數學運算量的多寡會是運算效率好壞最主要的因素，但在現實中，有許多因素也會有很大的影響，如快取記憶體以及CPU均有很大的影響。

在1978年，Winograd率先導出一個較嚴謹的FFT所需乘法量的下限：$\Theta(N)$。當$N = 2^k$時，DFT只需要$4N - 2\log_2^2 N - 2\log_2 N - 4$次無理實數的乘法即可以計算出來。更詳盡，且也能趨近此下限的演算法也一一被提出（Heideman & Burrus, 1986; Duhamel, 1990）。很可惜的是，這些演算法，都需要很大量的加法計算，目前的硬體無法克服這個問題。

對於所需加法量的數目，雖然我們可以在某些受限制的假設下，推得其下限，但目前並沒有一個精確的下限被推導出來。1973年，Morgenstern在乘法常數趨近巨大的情形下（對大部分的FFT演算法為真，但不是全部）推導出加法量的下限：$\Omega\left(N \log N\right)$。Pan（1986）在假設FFT演算法的不同步的情形有其極限下證明出加法量的下限$\Omega(NlogN)$，但一般來說，此假設相當的不明確。長度為$N = 2^k$的情形下，在某些假設下，Papadimitriou（1979）提出使用Cooley-Tukey演算法所需的複數加法量$N \log_2 N$是最少的。到目前為止，在長度為$N = 2^k$情況，還沒有任何FFT的演算法可以讓複數的加法量比$N \log_2 N$還少。

還有一個問題是如何把乘法量與加法量的總和最小化，有時候稱作"演算複雜度"（在這裡考慮的是實際的運算量，而不是漸近複雜度）。同樣的，沒有一個嚴謹下限被證明出來。從1968年開始，$N = 2^k$點DFT而言，split-radix FFT演算法需要最少的運算量，在$N > 1$的情形下，其需要$4N \log_2 N - 6N + 8$個乘法運算以及加法運算。最近有人導出更低的運算量：$\frac{34}{9} N \log_2 N$。（Johnson and Frigo, 2007; Lundy and Van Buskirk, 2007）

大多數嘗試要降低或者證明FFT複雜度下限的人都把焦點放在複數資料輸入的情況，因其為最簡單的情形。但是，複數資料輸入的FFT演算法，與實數資料輸入的FFT演算法，離散餘弦轉換（DCT），離散哈特列轉換（DHT），以及其他的演算法，均有很大的關連性。故任何一個演算法，在複雜度上有任何的改善的話，其他的演算法複雜度也會馬上獲得改善（Duhamel & Vetterli, 1990）。

快速演算法查表

當輸入信號長度為N時:

N = 1~60

N	乘法數	加法數	N	乘法數	N	乘法數	N	乘法數
1	0	0	11	46	24	28	39	182
2	0	4	12	8	25	148	40	100
3	2	12	13	52	26	104	42	124
4	0	16	14	32	27	114	44	160
5	10	34	15	40	28	64	45	170
6	4	36	16	20	30	80	48	92
7	16	72	18	32	32	72	52	208
8	4	52	20	40	33	160	54	228
9	16	72	21	62	35	150	56	156
10	20	88	22	80	36	64	60	160

$N < 1000$

N	乘法數	N	乘法數	N	乘法數	N	乘法數
63	256	96	280	192	752	360	1540
64	204	104	468	204	976	420	2080
66	284	108	456	216	1020	480	2360
70	300	112	396	224	1016	504	2300
72	164	120	380	240	940	512	3180
80	260	128	560	252	1024	560	3100
81	480	144	436	256	1308	672	3496
84	248	160	680	288	1160	720	3620
88	412	168	580	312	1324	784	4412
90	340	180	680	336	1412	840	4580

$N > 1000$

N	乘法數	N	乘法數	N	乘法數	N	乘法數
1008	5356	1440	8680	2520	16540	4032	29488
1024	7436	1680	10420	2688	19108	4096	37516
1152	7088	2016	12728	2880	20060	4368	35828
1260	7640	2048	16836	3369	24200	4608	36812
1344	8252	2304	15868	3920	29900	5040	36860

参阅

- 离散傅里叶变换
- 并行快速傅里叶变换

- 快速數論變換

参考资料

1. 杨毅明. 数字信号处理（第2版）. 北京: 机械工业出版社. 2017年: 第95页. ISBN 9787111576235.
2. Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform* (SIAM, 1992).
3. Heideman, M. T.; Johnson, D. H.; Burrus, C. S. Gauss and the history of the fast Fourier transform. IEEE ASSP Magazine. 1984, **1** (4): 14–21. doi:10.1109/MASSP.1984.1162257.
4. Strang, Gilbert. Wavelets. American Scientist. May–June 1994, **82** (3): 253. JSTOR 29775194.
5. Dongarra, J.; Sullivan, F. Guest Editors Introduction to the top 10 algorithms. Computing in Science Engineering. January 2000, **2** (1): 22–23. ISSN 1521-9615. doi:10.1109/MCISE.2000.814652.
6. Johnson and Frigo, 2007
7. Frigo & Johnson, 2005

延伸阅读

- Brenner, N.; Rader, C. A New Principle for Fast Fourier Transformation. IEEE Acoustics, Speech & Signal Processing. 1976, **24** (3): 264–266. doi:10.1109/TASSP.1976.1162805.
- Brigham, E. O. The Fast Fourier Transform. New York: Prentice-Hall. 2002.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. 30. (Polynomials and the FFT). Introduction to Algorithms 2. MIT Press and McGraw-Hill. 2001. ISBN 0-262-03293-7.
- Duhamel, Pierre. Algorithms meeting the lower bounds on the multiplicative complexity of length-2^n DFTs and their connection with practical algorithms. IEEE Trans. Acoust. Speech. Sig. Proc. 1990, **38** (9): 1504–151. doi:10.1109/29.60070.
- Duhamel, P.; Vetterli, M. Fast Fourier transforms: a tutorial review and a state of the art. Signal Processing. 1990, **19**: 259–299. doi:10.1016/0165-1684(90)90158-U.
- Edelman, A.; McCorquodale, P.; Toledo, S. The Future Fast Fourier Transform?. SIAM J. Sci. Computing. 1999, **20**: 1094–1114. doi:10.1137/S1064827597316266.
- D. F. Elliott, & K. R. Rao, 1982, *Fast transforms: Algorithms, analyses, applications*. New York: Academic Press.
- Funda Ergün, 1995, Testing multivariate linear functions: Overcoming the generator bottleneck (https://doi.org/10.1145%2F225058.225167), *Proc. 27th ACM Symposium on the Theory of Computing*: 407–416.
- Frigo, M.; Johnson, S. G. The Design and Implementation of FFTW3 (PDF). Proceedings of the IEEE. 2005, **93**: 216–231 [2008-06-22]. doi:10.1109/jproc.2004.840301. （原始内容存档 (PDF)于2019-07-16）.
- H. Guo and C. S. Burrus, 1996, Fast approximate Fourier transform via wavelets transform (https://doi.org/10.1117%2F12.255236), *Proc. SPIE Intl. Soc. Opt. Eng.* **2825**: 250–259.
- H. Guo, G. A. Sitton, C. S. Burrus, 1994, The Quick Discrete Fourier Transform (https://doi.org/10.1109%2FICASSP.1994.389994), *Proc. IEEE Conf. Acoust. Speech and Sig. Processing (ICASSP)* **3**: 445–448.
- Steve Haynal and Heidi Haynal, "Generating and Searching Families of FFT Algorithms (https://web.archive.org/web/20120426031804/http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_13_Haynal.pdf)", *Journal on Satisfiability, Boolean Modeling and Computation* vol. 7, pp. 145–187 (2011).

- Heideman, Michael T.; Burrus, C. Sidney. On the number of multiplications necessary to compute a length-2^n DFT. IEEE Trans. Acoust. Speech. Sig. Proc. 1986, **34** (1): 91–95. doi:10.1109/TASSP.1986.1164785.
- Johnson, S. G.; Frigo, M. A modified split-radix FFT with fewer arithmetic operations (PDF). IEEE Trans. Signal Processing. 2007, **55** (1): 111–119 [2008-06-22]. doi:10.1109/tsp.2006.882087.（原始内容存档 (PDF)于2021-02-25）.
- T. Lundy and J. Van Buskirk, 2007. "A new matrix approach to real FFTs and convolutions of length 2^k," *Computing* **80** (1): 23–45.
- Kent, Ray D. and Read, Charles (2002). *Acoustic Analysis of Speech*. ISBN 978-0-7693-0112-9. Cites Strang, G. (1994)/May–June). Wavelets. *American Scientist, 82,* 250–255.
- Morgenstern, Jacques. Note on a lower bound of the linear complexity of the fast Fourier transform. J. ACM. 1973, **20** (2): 305–306. doi:10.1145/321752.321761.
- Mohlenkamp, M. J. A fast transform for spherical harmonics (PDF). J. Fourier Anal. Appl. 1999, **5** (2–3): 159–184. doi:10.1007/BF01261607.（原始内容 (PDF)存档于2007年2月6日）.
- Nussbaumer, H. J. Digital filtering using polynomial transforms. Electronics Lett. 1977, **13** (13): 386–387. doi:10.1049/el:19770280.
- V. Pan, 1986, The trade-off between the additive complexity and the asyncronicity of linear and bilinear algorithms (https://doi.org/10.1016%2F0020-0190%2886%2990035-9), *Information Proc. Lett.* **22**: 11–14.
- Christos H. Papadimitriou, 1979, Optimality of the fast Fourier transform (https://doi.org/10.1145%2F322108.322118), *J. ACM* **26**: 95–102.
- D. Potts, G. Steidl, and M. Tasche, 2001. "Fast Fourier transforms for nonequispaced data: A tutorial (http://www.tu-chemnitz.de/~potts/paper/ndft.pdf)", in: J.J. Benedetto and P. Ferreira (Eds.), *Modern Sampling Theory: Mathematics and Applications* (Birkhauser).
- Press, W.H.; Teukolsky, S.A.; Vetterling, W.T.; Flannery, B.P., Chapter 12. Fast Fourier Transform, Numerical Recipes: The Art of Scientific Computing 3, New York: Cambridge University Press, 2007 [2015-12-11], ISBN 978-0-521-88068-8,（原始内容存档于2011-08-11）
- Rokhlin, Vladimir; Tygert, Mark. Fast algorithms for spherical harmonic expansions. SIAM J. Sci. Computing. 2006, **27** (6): 1903–1928. doi:10.1137/050623073.
- Schatzman, James C. Accuracy of the discrete Fourier transform and the fast Fourier transform. SIAM J. Sci. Comput. 1996, **17**: 1150–1166. doi:10.1137/s1064827593247023.
- Shentov, O. V.; Mitra, S. K.; Heute, U.; Hossen, A. N. Subband DFT. I. Definition, interpretations and extensions. Signal Processing. 1995, **41** (3): 261–277. doi:10.1016/0165-1684(94)00103-7.
- Sorensen, H. V.; Jones, D. L.; Heideman, M. T.; Burrus, C. S. Real-valued fast Fourier transform algorithms. IEEE Trans. Acoust. Speech Sig. Processing. 1987, **35** (35): 849–863. doi:10.1109/TASSP.1987.1165220. See also Sorensen, H.; Jones, D.; Heideman, M.; Burrus, C. Corrections to "Real-valued fast Fourier transform algorithms" . IEEE Transactions on Acoustics, Speech, and Signal Processing. 1987, **35** (9): 1353–1353. doi:10.1109/TASSP.1987.1165284.
- Welch, Peter D. A fixed-point fast Fourier transform error analysis. IEEE Trans. Audio Electroacoustics. 1969, **17** (2): 151–157. doi:10.1109/TAU.1969.1162035.
- Winograd, S. On computing the discrete Fourier transform. Math. Computation. 1978, **32** (141): 175–199. JSTOR 2006266. doi:10.1090/S0025-5718-1978-0468306-4.