北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# 任意点数FFT算法探究

BEIJING INSTITUTE OF TECHNOLOGY

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

01 研究背景与意义

02 基p-FFT算法

03 混合基FFT算法

04 仿真验证

05 总结

德以明理 学以精工

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# Part 1

## 研究背景与意义

德以明理 学以精工

# 01 研究背景与意义

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

## FFT混合基算法的引入

- 在课堂上我们学习的是序列点数$N = 2^L$，即以2为基数的FFT算法。这种算法具有程序简单、效率高、使用方便的优点，因而得到了广泛的应用。
- 若序列的点数不满足$N = 2^L$，可以在原序列后面补0，使序列点数满足$N = 2^L$，然后对其使用基2-FFT算法求解。由DFT的性质可知，有限长序列补0并不影响频谱的包络，只是频率采样点数增加了。但有时补0点数太多，造成计算量增加太多，从而使求解速度变慢。
- 在有些情况下，我们需要求解准确的N点DFT，并且MATLAB中fft函数也能对任意点数的序列求解FFT。在这种情况下基2-FFT算法不再适用，如果使用DFT的定义直接进行计算，计算量将非常庞大，因此需要对任意点数N的FFT快速算法进行探索。

FFT混合基算法是Cooley-Tukey算法的通用形式，其核心思想是将一个长度为复合数N的DFT，递归地分解为多个更小长度的DFT组合进行计算。它不像基-2算法那样要求长度必须是2的幂，而是能灵活高效地处理任意包含较小质因子的序列，从而在通用性和效率间取得平衡。

因此，若想实现混合基算法，首先需要对基2-FFT算法进行推广，得到以任意点数p为基数的FFT算法，即基p-FFT算法。在此基础上对序列长度N进行素因数分解，逐层使用基p-FFT算法进行求解。基p-FFT算法仍分为时间抽取和频率抽取两种方式，两种方式计算量相同。为了方便使用递归的算法编写程序进行验证，本次报告采用时间抽取的方法。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# Part 2

## 基p-FFT算法

德以明理 学以精工

# 02 基p-FFT算法



## □ 基3-FFT算法推导

在推导基p-FFT算法时，首先对基3-FFT算法进行推导

设序列$x[n]$的长度为$N$，且满足$N=3^L$，将$x[n]$分成3个$N/3$的子序列：

$$x_0[n]=x[3n],\quad x_1[n]=x[3n+1],\quad x_2[n]=x[3n+2],\quad n=0,1,\cdots,\frac{N}{3}-1$$

于是$x[n]$的DFT可以表示为：

$$X[k]=\sum_{n=0}^{N-1}x[n]W_N^{kn}=\sum_{n=0}^{N/3-1}x[3n]W_N^{k(3n)}+\sum_{n=0}^{N/3-1}x[3n+1]W_N^{k(3n+1)}+\sum_{n=0}^{N/3-1}x[3n+2]W_N^{k(3n+2)}$$

$$=\sum_{n=0}^{N/3-1}x_0[n]W_{N/3}^{kn}+W_N^k\sum_{n=0}^{N/3-1}x_1[n]W_{N/3}^{kn}+W_N^{2k}\sum_{n=0}^{N/3-1}x_2[n]W_{N/3}^{kn}$$

$$=X_0[k]+W_N^kX_1[k]+W_N^{2k}X_2[k]\qquad k=0,1,\cdots,\mathrm{N}-1$$

其中$X_0[k]$、$X_1[k]$和$X_2[k]$分别是$x_0[n]$、$x_1[n]$和$x_2[n]$的$N/3$点DFT，注意到$X_0[k]$、$X_1[k]$和$X_2[k]$均是以$N/3$为周期的，于是可得

# 02 基p-FFT算法



## □ 基3-FFT算法推导

$$X[k] = X_0[k] + W_N^k X_1[k] + W_N^{2k} X_2[k],$$

$$X[k+N/3] = X_0[k] + W_N^{k+N/3} X_1[k] + W_N^{2(k+N/3)} X_2[k]$$

$$X[k+2N/3] = X_0[k] + W_N^{k+2N/3} X_1[k] + W_N^{2(k+2N/3)} X_2[k], \quad k = 0,1,\cdots,N/3-1$$

注意到$W_N^{N/3} = W_3^1$，$W_N^{2N/3} = W_3^2$，将其带入上式得

$$X[k] = X_0[k] + W_N^k X_1[k] + W_N^{2k} X_2[k],$$

$$X[k+N/3] = X_0[k] + W_3^1 W_N^k X_1[k] + W_3^2 W_N^{2k} X_2[k]$$

$$X[k+2N/3] = X_0[k] + W_3^2 W_N^k X_1[k] + W_3^4 W_N^{2k} X_2[k], \quad k = 0,1,\cdots,N/3-1$$

$x[n]$的N点DFT可以转化为$x_0[n]$, $x_1[n]$和$x_2[n]$的$N/3$点DFT的加权组合。这种方式是将序列以3的模分成3个子列，即在时域上进行3倍抽取，这就是基-3时间抽取FFT算法。

# 02 基p-FFT算法



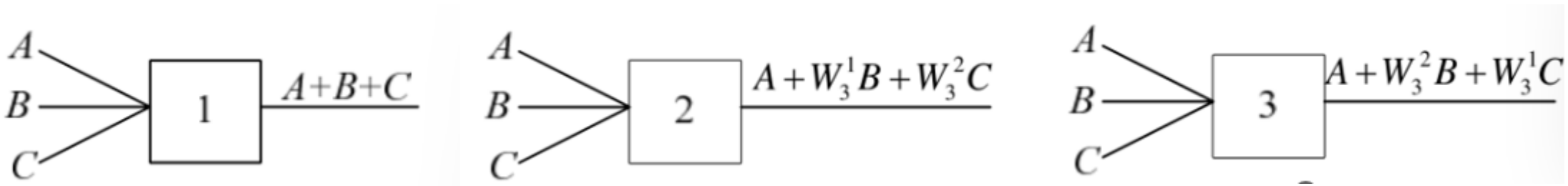
## □ 基3-FFT蝶形运算单元

若写成矩阵形式，即：

$$\boldsymbol{X} = \boldsymbol{W_3 D_3 X'}$$

其中$\boldsymbol{W_3}$为3点DFT矩阵，$\boldsymbol{D_3} = \boldsymbol{diag}\left(W_N^0, W_N^k, W_N^{2k}\right)$，$\boldsymbol{X} = \begin{bmatrix} X[k] \\ X[k+N/3] \\ X[k+2N/3] \end{bmatrix}$，$\boldsymbol{X'} = \begin{bmatrix} X_0[k] \\ X_1[k] \\ X_2[k] \end{bmatrix}$，$k = 0,1,\cdots,N/3-1$

则基3-FFT蝶形运算单元定义如下图所示



# 02 基p-FFT算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

## 基3-FFT多级蝶形运算

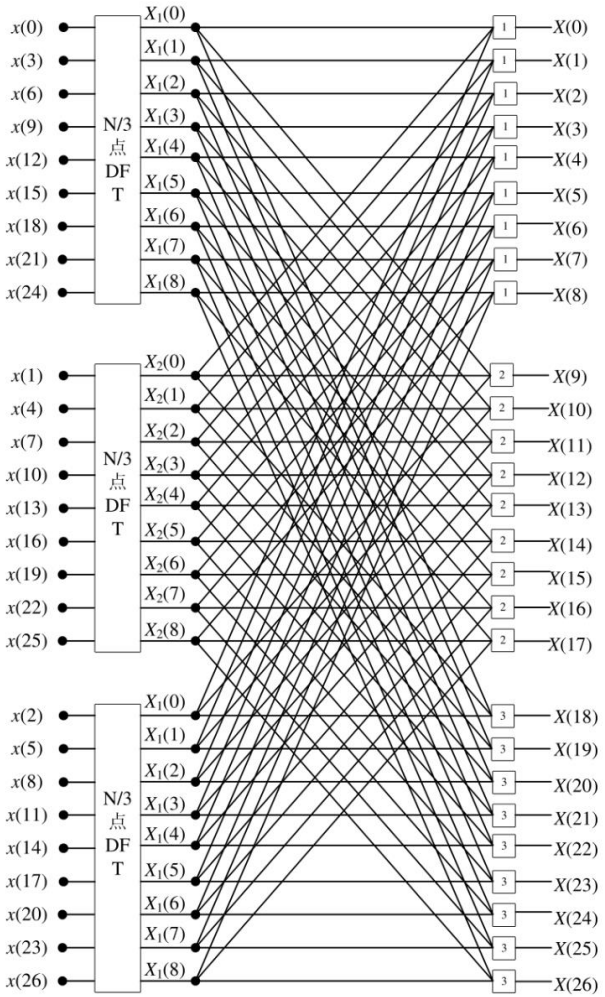
- 进一步，如果N/3也是偶数，则可将N/3点DFT分解为N/9点DFT。以此类推，直到DFT点数不是3的倍数为止。当点数$N = 3^s$时，可持续进行分解，总共需要s级运算。按照上述规律，27点序列基3-FFT算法结构如右图所示：

- 在基3-FFT算法中，输入序列的序号可以通过三进制“码位倒置”的方法得到，即先将序号N表示为三进制数$(r_{L-1}, r_{L-2}, \dots, r_1, r_0)_3$，其中$r_{\mathrm{i}} = 0,1$或2，此时有

$$N = r_{L-1}3^{L-1} + r_{L-2}3^{L-2} + \cdots + r_1 3 + r_0$$

  然后将该三进制数的首尾颠倒为 $(r_0, r_1, \dots, r_{L-2}, r_{L-1})_3$，重新按照上式计算新的序号值，最后转换为十进制数。

- 但是在具体编程过程中我们并不一定需要知道输入序列的序号，根据基3-FFT多级蝶形运算的特点，我们可以采用递归调用的方式对N点DFT进行逐级分解，最终分解为3点DFT。这种方法既降低了编程难度，也可以提高程序代码的可读性，并且仍可以大幅降低DFT的计算量。




27点序列基3-FFT算法结构

# 02 基p-FFT算法



## □ 基p-FFT算法推导

在基3-FFT算法的基础上，可以推导基p-FFT算法。将序列以p的模分成p个子列，即在时域上进行p倍抽取，即可得

$$x_0[n]=x[pn], x_1[n]=x[pn+1], x_{p-2}[n]=x[pn+p-2], x_{p-1}[n]=x[pn+p-1] \qquad n=0,1,\cdots,\frac{N}{p}-1$$

仿照基3-FFT算法推导过程，设$X_0[k]$、 $X_1[k]\cdots X_{p-1}[k]$是$x_0[n]$、 $x_1[n]\cdots x_{p-1}[n]$的$N/p$点DFT，则：

$$X[k]=X_0[k]+W_N^kX_1[k]+\cdots+W_N^{(p-2)k}X_{p-2}[k]+W_N^{(p-1)k}X_{p-1}[k],$$

$$X[k+N/p]=X_0[k]+W_p^1W_N^kX_1[k]+\cdots+W_p^{(p-2)}W_N^{(p-2)k}X_{p-2}[k]+W_p^{(p-1)}W_N^{(p-1)k}X_{p-1}[k]$$

$$\vdots$$

$$X[k+(p-2)N/p]=X_0[k]+W_p^{p-2}W_N^kX_1[k]+\cdots+W_p^{(p-2)(p-2)}W_N^{(p-2)k}X_{p-2}[k]+W_p^{(p-2)(p-1)}W_N^{(p-1)k}X_{p-1}[k]$$

$$X[k+(p-1)N/p]=X_0[k]+W_p^{p-1}W_N^kX_1[k]+\cdots+W_p^{(p-1)(p-2)}W_N^{(p-2)k}X_{p-2}[k]+W_p^{(p-1)(p-1)}W_N^{(p-1)k}X_{p-1}[k]$$

由此可观察到基3-FFT算法是基p-FFT算法在p=3时的特例， 此时$x[n]$的N点DFT可以转化为$x_0[n], x_1[n],\cdots,x_{p-1}[n]$的$N/p$点DFT的加权组合。这种方式是将序列以p的模分成p个子列，即在时域上进行p倍抽取，这就是基-p时间抽取FFT算法。

# 02 基p-FFT算法



## □ 基p-FFT算法推导

若写成矩阵形式，可得：

$$\boldsymbol{X} = \boldsymbol{W_p D_p X'}$$

其中$\boldsymbol{W_p}$为p点DFT矩阵，$\boldsymbol{D_p} = \boldsymbol{diag}\left(W_N^0, W_N^k, \cdots, W_N^{(p-2)k}, W_N^{(p-1)k}\right)$，$\boldsymbol{X} = \begin{bmatrix} X[k] \\ X[k+N/p] \\ \vdots \\ X[k+(p-2)N/p] \\ X[k+(p-1)N/p] \end{bmatrix}$，$\boldsymbol{X'} = \begin{bmatrix} X_0[k] \\ X_1[k] \\ \vdots \\ X_{p-2}[k] \\ X_{p-1}[k] \end{bmatrix}$

$k = 0,1,\cdots,N/p-1$

- 基p-FFT算法同样具有类似的蝶形运算单元，由于较为复杂这里不再展示。由于基p-FFT算法是基3-FFT算法的推广，因此当点数$N = p^s$时，可持续按照p的倍数进行逐级分解，总共需要s级运算，在具体编程中采取递归算法实现。
- 在基p-FFT算法中，输入序列的序号同样可以通过p进制“码位倒置”的方法得到，即先将序号N表示为p进制数$(r_{L-1}, r_{L-2}, \ldots, r_1, r_0)_p$，然后将该p进制数的首尾颠倒为$(r_0, r_1, \ldots, r_{L-2}, r_{L-1})_p$，据此计算新的序号值，最后转换为十进制数即可。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# Part 3

## 混合基FFT算法

德以明理 学以精工

# 03 混合基FFT算法

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

## □ $N = p_1 p_2$形式的混合基算法

当点数N不再是某个数的幂次时，基p-FFT算法的分解不能持续进行。在此情况下可对N进行质因数分解，即$N = p_1 p_2 \cdots p_{L-1} p_L$($p_1, p_2 \cdots p_{L-1}, p_L$均为质数)，因此可以采取混合基算法加快运算速度。

当$N = p_1 p_2$时，按照基p-FFT算法的思路，将序列以$p_2$的模分成$p_2$个子列，即

$$x_0[n] = x[p_2 n], x_1[n] = x[p_2 n + 1], x_{p_2-2}[n] = x[p_2 n + p_2 - 2], x_{p_2-1}[n] = x[p_2 n + p_2 - 1] \qquad n = 0,1,\cdots,p_1 - 1$$

设$X_0[k]$、 $X_1[k] \cdots X_{p_2-1}[k]$是$x_0[n]$、 $x_1[n] \cdots x_{p_2-1}[n]$的$p_1$点DFT，重复基p-FFT算法推导方法，则有

$$\boldsymbol{X} = \boldsymbol{W}_{\boldsymbol{p_2}} \boldsymbol{D}_{\boldsymbol{p_2}} \boldsymbol{X}'$$

其中$\boldsymbol{W}_{\boldsymbol{p_2}}$为$p_2$点DFT矩阵， $\boldsymbol{D}_{\boldsymbol{p_2}} = \boldsymbol{diag}\left(W_N^0, W_N^k, \cdots, W_N^{(p_2-2)k}, W_N^{(p_2-1)k}\right)$， $\boldsymbol{X} = \begin{bmatrix} X[k] \\ X[k+p_1] \\ \vdots \\ X[k+(p_2-2)p_1] \\ X[k+(p_2-1)p_1] \end{bmatrix}$， $\boldsymbol{X}' = \begin{bmatrix} X_0[k] \\ X_1[k] \\ \vdots \\ X_{p_2-2}[k] \\ X_{p_2-1}[k] \end{bmatrix}$

$k = 0,1,\cdots,p_1 - 1$

$x[n]$的N点DFT可以转化为$x_0[n], x_1[n], \cdots, x_{p_2-1}[n]$ $p_2$个子列的$p_1$点DFT的加权组合，由于分解的$p_1$和$p_2$并不相同，所以这种算法叫做混合基算法。

# 03 混合基FFT算法



## □ $N = p_1 p_2$形式的混合基算法

当$N = p_1 p_2$时，混合基FFT算法可看作由$p_2$个子列DFT的加权组合和求解这两部分组成。

由$p_2$个子列DFT加权组合的矩阵计算式可知，这一步骤可看作先乘N个旋转因子再进行$p_1$个$p_2$点DFT操作，复数乘法的运算量为$p_1 {p_2}^2 + N$，复数加法的运算量为$p_1 p_2 (p_2 - 1)$

在求$p_2$个子列的$p_1$点DFT时，复数乘法的运算量为$p_2 {p_1}^2$，复数乘法的运算量为$p_2 p_1 (p_1 - 1)$，因此总计计算量：

$$\text{复数乘法} = N(p_1 + p_2 + 1)，\text{复数加法} = N(p_1 + p_2 - 2)$$

直接计算DFT的计算量：复数乘法 $= N^2$，复数加法 $= N(N-1)$，因此混合基运算可节省的计算量倍数为：

$$\text{复数乘法} = N/(p_1 + p_2 + 1)，\text{复数加法} = (N-1)/(p_1 + p_2 - 2)$$

观察上式可知，一般情况下$p_1 + p_2 + 1 \ll N$，使用混合基算法可明显降低计算量。

注意到当$p_1$或$p_2$等于2时上式不再成立，这是由于基2-FFT蝶形运算单元运算结构比较简单，每次只需进行1次乘法和2次加法即可，因此相较于上式运算量还可以进一步降低。进一步推导可知，相较于所有基p-FFT算法，基2-FFT算法的运算量最低，计算速度最快。

# 03 混合基FFT算法



## □ $N = p_1 p_2 \cdots p_L$形式的混合基算法

当$N = p_1 p_2 \cdots p_L$时，可按照$N = p_1 p_2$的方法，先将序列以$p_L$的模分成$p_L$个子列，则$x[n]$的N点DFT可以转化为$x_0[n]$, $x_1[n], \cdots, x_{p_L-1}[n]$这$p_L$个子列的$p_1 p_2 \cdots p_{L-1}$点DFT的加权组合，每个子列的$p_1 p_2 \cdots p_{L-1}$点DFT又可按照上述同样的方式继续分解，持续进行下去，直到最后会分解为$p_1$点DFT的加权组合。

此时输入序列的序号仍然满足类似于基2-FFT算法的多基多进制“码位倒置”的关系，但是在具体编程过程中我们可以借鉴之前基3-FFT的思路，采用递归调用的方式对N点DFT根据分解的素因子进行混合基分解，最终分解为$p_1$点DFT。

与$N = p_1 p_2$的情况类似，$N = p_1 p_2 \cdots p_L$形式的混合基算法的总计算量为：

$$\text{复数乘法} = N\left[\left(\sum_{i=1}^{L} r_i\right) + L - 1\right]，\text{节省的计算量倍数} = \frac{N}{\sum_{i=1}^{L} r_i + L - 1}$$

同样地，上式也必须在$p_1, p_2, \cdots, p_L > 2$时成立。

# 03 混合基FFT算法



## 混合基算法的问题和改进

- 混合基算法存在的问题：由上述分析可知，混合基FFT算法可以通过对序列点数N进行质因数分解，从而大幅减小计算量。然而当序列长度N为质数或质因数分解含有大质数因子，此时N不能得到有效分解，混合基算法就变为计算质数点数的DFT，这会导致计算速度大幅降低。

- 雷德算法：为了加快质数点数FFT的计算速度，可使用雷德算法进行改进。通过将素数长度的DFT转化为循环相关，而循环相关可通过FFT高效计算，通过调整计算循环相关的FFT点数可以满足快速计算的要求，从而加快质数点数FFT的计算速度。

由于雷德算法基本原理的推导涉及到原根等数论知识，推导起来较为复杂，由于课堂时间有限不再展示。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# Part 4

## 仿真验证

德以明理 学以精工

# 04 仿真验证

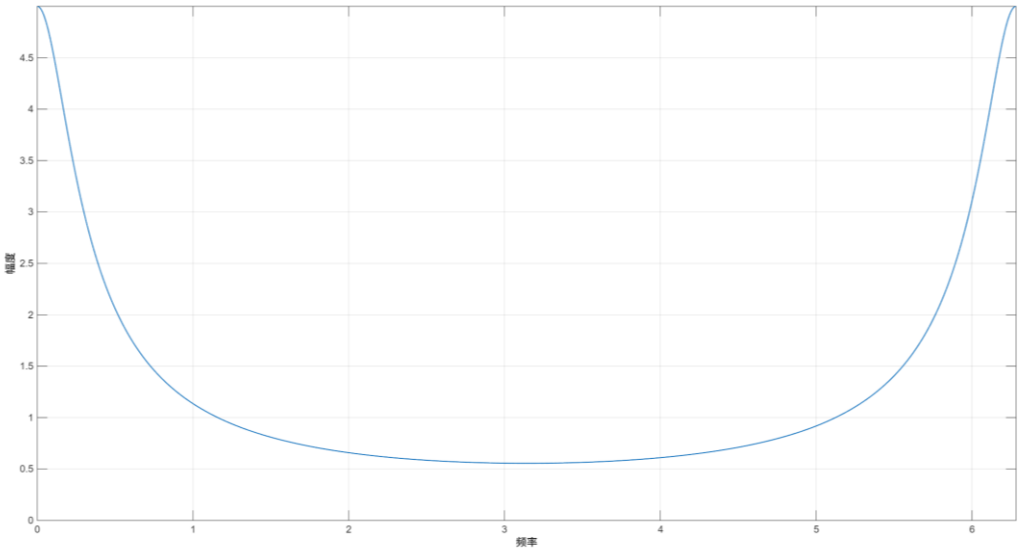


## 编程仿真实现

根据上述对任意点数FFT算法原理的介绍，可以利用递归方式对基p-FFT算法、混合基算法和雷德算法进行编程实现。在这里仿真验证中，为了使结果更直观，我们以有限长的单边指数序列为例，探究不同FFT算法的运行速度。单边指数序列为形如下式的序列：

$$x[n] = a^n u[n],\ |a| < 1$$

仿真验证时取$a = 0.8$，单边指数序列DTFT频谱图如右图所示。当序列较长时后面的项几乎趋近于0，对DFT的求解结果影响可以忽略不计，并且DFT是对DTFT采样的结果，因此绘制的DFT频谱图的形状也应与右图相同。

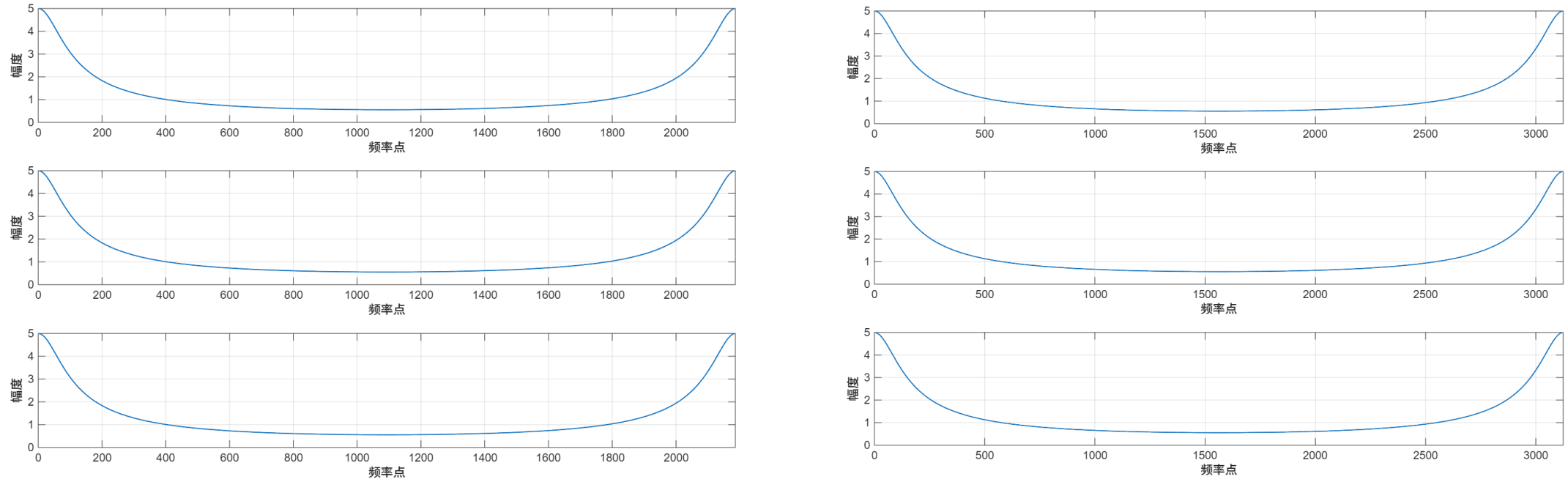


# 04 仿真验证



## □ 基p-FFT算法

分别取序列的长度为$3^7$和$5^5$，利用MATLAB内置的fft函数、DFT定义和基p-FFT算法分别求序列的DFT，三种求解方式绘制的频谱图如右图所示。




可见这三种求解方式绘制的频谱图几乎完全一样，并于DTFT的形状相符，验证了求解结果的正确性

# 04 仿真验证



## □ 基p-FFT算法

分别取序列的长度为$3^7$和$5^5$，利用MATLAB内置的fft函数、DFT定义和混合基算法分别求序列的DFT，三种求解方式消耗的时间如下所示。

```
序列的长度N=2187
MATLAB内置fft函数历时 0.000134 秒。
dft直接计算历时 1.842886 秒。
基p-fft算法历时 0.011947 秒。
```

```
序列的长度N=3125
MATLAB内置fft函数历时 0.000130 秒。
dft直接计算历时 3.866020 秒。
基p-fft算法历时 0.021457 秒。
```

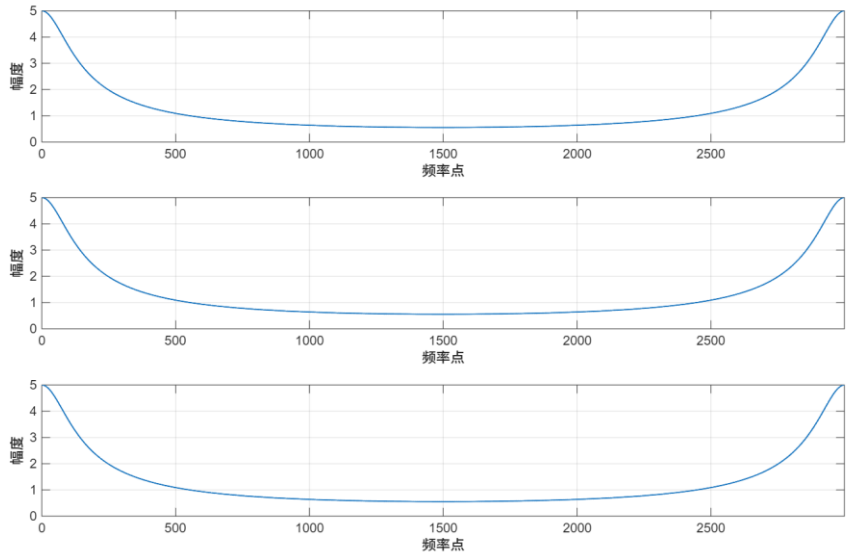
由上图可知，基p-FFT算法运算速度虽然不及MATLAB内置fft函数，但是相较于通过DFT定义直接计算要快得多，因此对于长度满足$N = p^L$高度复合数序列，基p-FFT算法可以大幅提升运算速度

# 04 仿真验证



## 混合基FFT算法

取序列的长度为3000，其中$3000 = 2^3 \times 3 \times 5^3$，在此情况下基p-FFT算法不再适用，需借助混合基算法。利用MATLAB内置的fft函数、DFT定义和混合基算法分别求序列的DFT，三种求解方式绘制的频谱图以及计算时间如下所示。




```
序列的长度N=3000
MATLAB内置fft函数历时  0.000194  秒。
dft直接计算历时  3.550939  秒。
混合基fft算法历时  0.012813  秒。
```

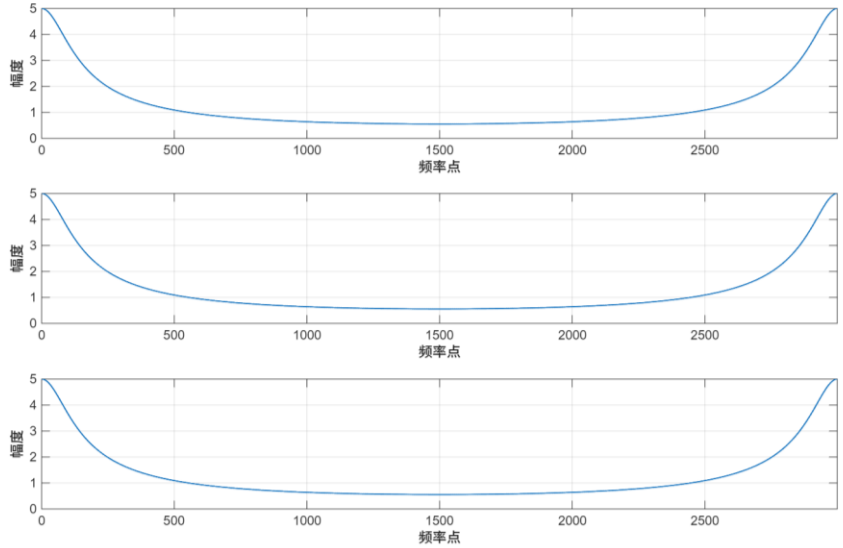
- 可见三种求解方式绘制的频谱图几乎完全一样，并于DTFT的形状相符，验证了求解结果的正确性
- 由上图可知，编写的混合基算法运算速度虽然不及MATLAB内置fft函数，但是相较于通过DFT定义直接计算要快得多，因此对许长度满足$N = p_1 p_2 \cdots p_L$的序列，混合基算法可以大幅提升运算速度

# 04 仿真验证

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

## □ 雷德-混合基FFT算法

取序列的长度为3001，其中3001是个质数。利用MATLAB内置的fft函数、DFT定义、混合基算法分别求序列的DFT，三种求解方式绘制的频谱图以及消耗的时间如下所示。




```
序列的长度N=3001
MATLAB内置fft函数历时  0.000250  秒。
dft直接计算历时  3.542324  秒。
混合基fft算法历时  3.613346  秒。
雷德-混合基fft算法历时  0.067168  秒。
```

- 当序列长度为3001时，混合基算法和通过DFT直接计算用时几乎直接相等。
- 这说明当DFT的点数为质数时，混合基算法就相当于直接求解DFT，不能加快计算速度。
- 当使用雷德算法改进后，计算速度显著提升，这与仿真预期相符，但计算速度仍不及MATLAB内置fft函数。

北京理工大学
BEIJING INSTITUTE OF TECHNOLOGY

# Part 5

## 总结

德以明理 学以精工

# 05 总结



- 综合以上探究，可以采取以下算法策略实现任意点数FFT的快速计算：
  1. 当序列长度$N = 2^L$时，可以直接采用基2-FFT算法进行计算，此时计算速度最快。
  2. 当序列长度$N \neq 2^L$但$N = p^L$(p为质数)时，应采用基p-FFT算法。
  3. 当序列长度$N = p_1 p_2 \cdots p_{L-1} p_L$ ($p_1, p_2 \cdots p_{L-1}, p_L$均为质数)时，应对N进行质因数分解并采用混合基算法。
  4. 当序列长度N为质数或质因数分解含有较大素数，应在混合基算法的基础上使用雷德算法进行优化。
- MATLAB内置函数fft主要基于FFTW来实现快速傅里叶变换，根据输入序列的长度不同采取不同的算法。FFTW在计算fft仍采用了混合基算法和雷德算法等，但其算法种类更多样，使用更灵活，并且底层采用C语言编写。这大大提升了fft函数的运行速度，与仿真实验中实现的FFT算法相比具有更高的效率。
- 混合基算法可以高效计算任意长度数据序列的DFT，因此在信号处理、数字通信、音频处理等领域得到了广泛的应用。

# 05 总结

北京理工大学 BEIJING INSTITUTE OF TECHNOLOGY

## 口 参考文献


[1] Cooley, James W.,Tukey,et al. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of Computation,1965,Vol.19(90): 297-301

[2] Rader, C.M. Discrete Fourier transforms when the number of data samples is prime[J]. Proceedings of the IEEE,1968,Vol.56(6): 1107-1108.

[3] Frigo, M.,Johnson,et al. FFTW: an adaptive software architecture for the FFT[C]//Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181). 1998.

[4] Oppenheim, A. V., Schafer, R. W. Discrete-Time Signal Processing (3rd ed.)[M]. Upper Saddle River (N.J.): Prentice Hall, 2010. ISBN 9780132067099.

[5] 程佩青编著. 数字信号处理教程 经典版[M]. 北京：清华大学出版社,2015

[6] 冷狗。自己实现混合基 fft，适用于任意长度序列 [EB/OL]. (2025-01-17) [2025-12-6]. https://zhuanlan.zhihu.com/p/6871666826.

[7] 通信考研加油哥。数字信号处理 - 基 3 的 FFT 算法 [EB/OL]. (2025-09-02) [2025-12-6]. https://zhuanlan.zhihu.com/p/1946000854160250159.

北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

# 汇报结束，感谢聆听！

THANKS FOR YOUR LISTENING!