高等数值算法与应用（十三）

Advanced Numerical Algorithms & Applications

计算机科学与技术系 喻文健

Today

- Discrete Fourier Transform, FFT
- Applications of DFT (课本第12章)
- Fast Solver for Poisson Equation
- About the Final Project

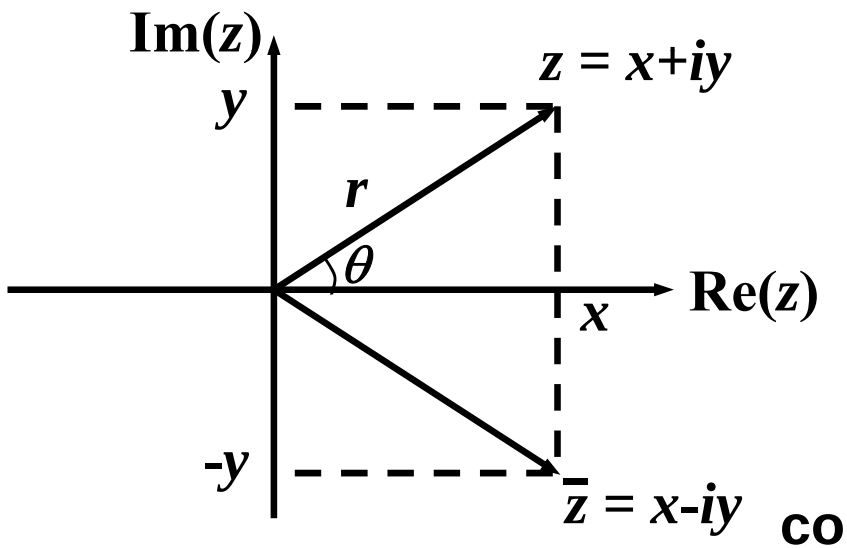
Discrete Fourier Transform and FFT

Wenjian Yu

Review of Complex Arithmetic

- 复平面与复数表示

$$z = x + iy$$ imaginary unit $\sqrt{-1}$



complex conjugate

根据欧拉定理： ??

$$e^{i\theta} = \cos\theta + i\sin\theta$$

极坐标形式 $z = re^{i\theta}$

其中 $r = |z|,\ \theta = \arctan(y/x)$

复数的模： $|z| = \sqrt{z\bar{z}} = \sqrt{x^2 + y^2}$, $\bar{z} = re^{-i\theta}$

Review of Complex Arithmetic

- 复数的算术运算

$$z_1 \times z_2 = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1)$$

两个复数的+,-,x,/运算需要2到11次实数运算

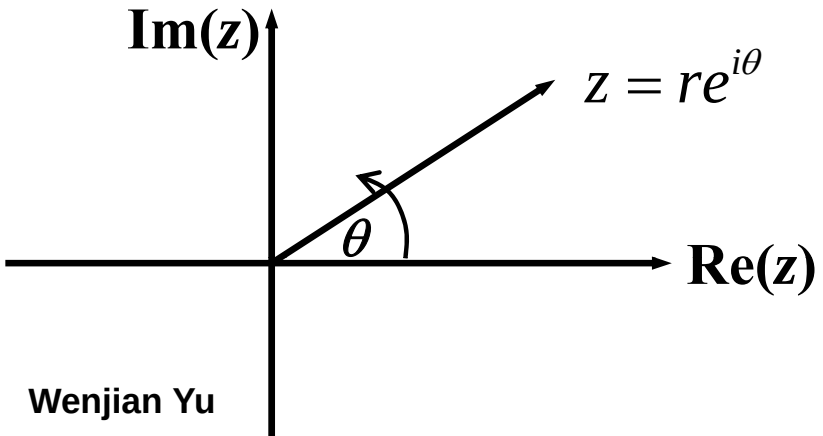
$$z_1 / z_2 = \frac{x_1 x_2 + y_1 y_2}{x_2^{\,2} + y_2^{\,2}} + i\frac{x_2 y_1 - x_1 y_2}{x_2^{\,2} + y_2^{\,2}}$$

极坐标形式 $z = re^{i\theta}$

$$z_1 \times z_2 = r_1 r_2 e^{i(\theta_1 + \theta_2)}$$

$$z_1 / z_2 = (r_1 / r_2) e^{i(\theta_1 - \theta_2)}$$

一个复数乘以 *i*，等价于将其在复平面上逆时针旋转π/2



离散周期函数的三角函数逼近

- 三角函数适合于逼近带周期性的函数
- 离散周期函数(表格函数)的三角函数逼近

复函数 $x(t)$ 的周期为**2π**，将**[0, 2π]**作**n**等分, 得到表格函数:

$x_k = x(2\pi k/n), \quad k = 0,1,...,n-1.$ 因为 $\{1, \cos t, \sin t, \cdots, \cos jt, \sin jt, \cdots\}$ 为 **[0, 2π]**上实函数空间的正交函数族,

函数族 $\{1, e^{it}, \cdots, e^{i(n-1)t}\}$ 在区间**[0, 2π]**上的复函数空间是正交的.

并且在离散点集 $\{2\pi k/n, \quad k = 0,1,...,n-1\}$ 上，这些函数也具有正交性.

因此，$x(t)$ 在这**n**个点上的最小二乘傅立叶逼近为

$$S(t) = \sum_{m=0}^{n-1} y_m e^{imt}, \text{ 其中 } y_m = \frac{(x(t), e^{imt})}{(e^{imt}, e^{imt})} = \frac{1}{n}\sum_{k=0}^{n-1} x_k e^{-im\frac{2\pi k}{n}}, \ m = 0,1,...,n-1$$

(注意复向量内积定义)

由于**n**阶逼近就是插值，所以 $x_k = S(t_k) = \sum_{m=0}^{n-1} y_m e^{im\frac{2\pi k}{n}}$

Roots of Unity

为了记号简单，引入“旋转因子”的概念

For given integer n, we use notation

$$\omega_n = \cos(2\pi/n) - i\sin(2\pi/n) = e^{-2\pi i/n}$$

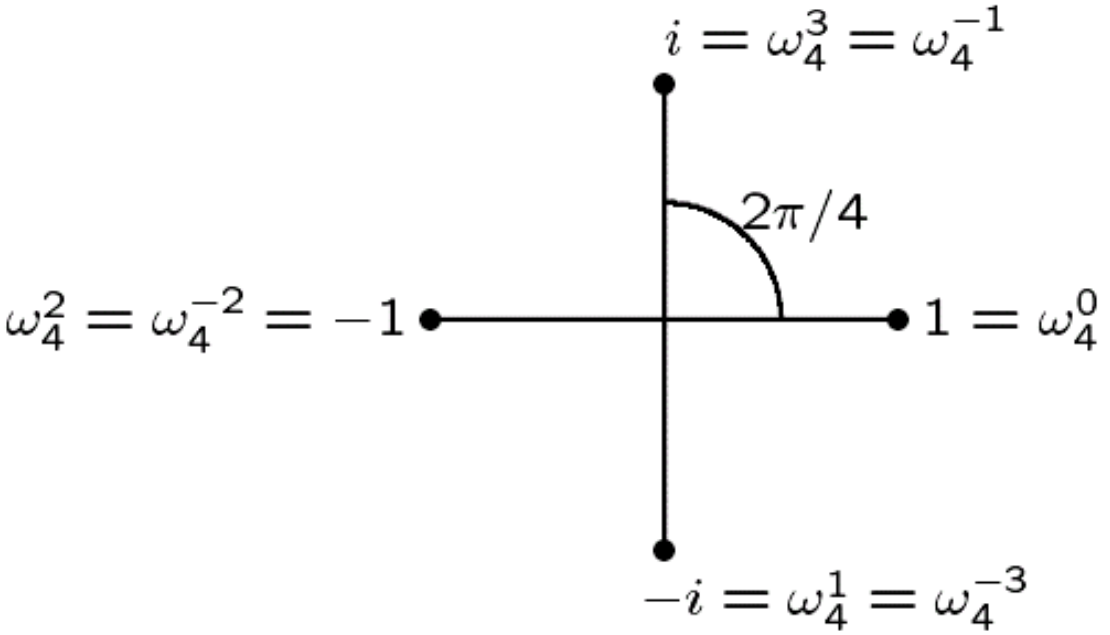
注意幅角中的”–”

for primitive nth root of unity

n次单位根

nth roots of unity, sometimes called *twiddle factors* in this context, are then given by ω_n^k or by ω_n^{-k}, $k = 0, \ldots, n-1$

旋转因子



$x^4 = 1$ 的四个根

Clockwise

Discrete Fourier Transform

DFT把一个复数序列变换为另一个复数序列

Given sequence $\boldsymbol{x} = [x_0, \ldots, x_{n-1}]^T$, its *discrete Fourier transform*, or *DFT*, is sequence $\boldsymbol{y} = [y_0, \ldots, y_{n-1}]^T$ given by

$$y_m = \sum_{k=0}^{n-1} x_k \omega_n^{mk}, \quad m = 0, 1, \ldots, n-1,$$

比较前面离散周期函数的逼近:

$$S(t) = \sum_{m=0}^{n-1} \frac{1}{n} y_m e^{imt},$$

or, written more compactly,

$$\boldsymbol{y} = \boldsymbol{F}_n \boldsymbol{x},$$

with entries of Fourier matrix $\boldsymbol{F}_n$ given by

$$\{\boldsymbol{F}_n\}_{mk} = \omega_n^{mk}$$

$\boldsymbol{F}_n$ 为对称矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 4 & 6 \\ 0 & 3 & 6 & 9 \end{pmatrix}$$

For example,

$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$$\omega_4^1 = -i$$

Inverse DFT

DFT反变换

Note that

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \omega^{-3} \\ 1 & \omega^{-2} & \omega^{-4} & \omega^{-6} \\ 1 & \omega^{-3} & \omega^{-6} & \omega^{-9} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^{1} & \omega^{2} & \omega^{3} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} \end{bmatrix}$$

$$\overline{F}_n = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot n$$

F_n为对称阵，所有元素模为1

In general, $F_n^{-1} = (1/n)\,\overline{F}_n$

Inverse DFT is therefore given by

三角插值

$$x_k = \frac{1}{n} \sum_{m=0}^{n-1} y_m \, \omega_n^{-mk}, \quad k = 0, 1, \ldots, n-1$$

将x看成k的函数；则插值函数为1，

$\omega_n^{-k} = e^{i2\pi k/n}$, ...

$\omega_n^{-(n-1)k} = e^{i2\pi(n-1)k/n}$

DFT gives trigonometric interpolant using only matrix-vector multiplication, which costs only $\mathcal{O}(n^2)$

求插值系数 y_m

DFT, continued

DFT of sequence, even purely real sequence, is in general complex

$$y_k = \sum_{m=0}^{n-1} x_m \omega_n^{km}$$

Components of DFT y of real sequence x of length n are *conjugate symmetric*: y_k and y_{n-k} are complex conjugates for $k = 1, \ldots, (n/2) - 1$

$$y_{n-k} = \sum_{m=0}^{n-1} x_m \omega_n^{(n-k)m}$$

$$\omega_n^{(n-k)m} \cdot \omega_n^{km} = \omega_n^{nm} = 1$$

共扼, 对任意 ***m***

Two components of special interest are:

- y_0, whose value is sum of components of x, is sometimes called DC component, corresponding to zero frequency (i.e., constant function)

$$S(t) = \sum_{m=0}^{n-1} \frac{1}{n} y_m e^{imt},$$

- $y_{n/2}$, corresponding to *Nyquist frequency*, which is highest frequency representable at given sampling rate

$\omega_n^{-(n-m)k}$ 与 ω_n^{-mk} 代表相同频率三角函数

n为偶数

Example: DFT

For randomly chosen sequence $\boldsymbol{x}$,

$$\boldsymbol{F}_8\,\boldsymbol{x} = \boldsymbol{F}_8 \begin{bmatrix} 4 \\ 0 \\ 3 \\ 6 \\ 2 \\ 9 \\ 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 35 \\ -5.07 + 8.66i \\ -3 + 2i \\ 9.07 + 2.66i \\ -5 \\ 9.07 - 2.66i \\ -3 - 2i \\ -5.07 - 8.66i \end{bmatrix} = \boldsymbol{y}$$

普遍成立，只要**n**为偶数

Transformed sequence is complex, but y_0 and y_4 are real, while y_5, y_6, and y_7 are complex conjugates of y_3, y_2, and y_1, respectively

There appears to be no discernible pattern to frequencies present, and y_0 is indeed equal to sum of elements of $\boldsymbol{x}$

频率上没有明显特征

Example: DFT

For cyclic sequence x,

$$F_8\, x = F_8 \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 8 \\ 0 \\ 0 \\ 0 \end{bmatrix} = y$$

Sequence has highest possible rate of oscillation (between 1 and -1) for this sampling rate

8个采用点可能表示的频率最高的周期性

In transformed sequence, only nonzero component is at Nyquist frequency (in this case y_4)

从频域上看，仅最高频率分量非零**(**单纯的四倍频**)**

Idea of Fast DFT (FFT)

设$\alpha = \omega_n^k,\ 0 \le k \le n-1$

多项式插值的形式

$$y_k = \sum_{m=0}^{n-1} x_m \omega_n^{km} = x_0 + x_1\alpha + x_2\alpha^2 + \cdots + x_{n-1}\alpha^{n-1}$$

$$= (x_0 + x_2\alpha^2 + x_4\alpha^4 + \cdots) + \alpha(x_1 + x_3\alpha^2 + x_5\alpha^4 + \cdots)$$

$$= DFT_{n/2}(x_{even})_k + \alpha \cdot DFT_{n/2}(x_{odd})_k$$

$DFT_n(x)_k$

设n为偶数，注意到旋转因子（1的复数根）ω_n有下述性质：

$$\alpha = \omega_n^k = e^{-2k\pi i/n},\quad \alpha^2 = e^{-2k\pi i/(n/2)} = \omega_{n/2}^k$$

For k=0, …, n/2-1,
转换为n/2阶的DFT计算

当$k \ge n/2$时，怎么办？

注意到$\alpha^2 = \omega_n^{2k} = \omega_n^n \cdot \omega_n^{2(k-n/2)},\ k \ge n/2$

1

$k' = k - n/2$ → 仍是那两个n/2阶的DFT计算

又$\alpha = \omega_n^k = \omega_n^{n/2} \cdot \omega_n^{k-n/2} = -\omega_n^{k'}$,

可再简化$DFT_n(x)$后半段计算

FFT Algorithm

```
procedure fft(x, y, n, ω)
    if n = 1 then
        y[0] = x[0]
    else
        for k = 0 to (n/2) − 1
            p[k] = x[2k]
            s[k] = x[2k + 1]
        end
        fft(p, q, n/2, ω^2)
        fft(s, t, n/2, ω^2)
        for k = 0 to n − 1
            y[k] = q[k mod (n/2)] +
                   ω^k t[k mod (n/2)]
        end
    end
```

计算量分析：

$$
\begin{aligned}
C(n) &= 2C(\frac{n}{2}) + 5n \\
&= 4C(\frac{n}{4}) + 2 \cdot 5n \\
&= 8C(\frac{n}{8}) + 3 \cdot 5n \\
&= \cdots \\
&= 5n\log_2 n \quad < \quad n^2
\end{aligned}
$$

进一步简化

$$
w = \left[\omega^0, \cdots, \omega^{n/2-1}\right]^T
$$

$$
y = \begin{bmatrix} q + w.*t \\ q - w.*t \end{bmatrix}
$$

$$
\because \ \omega_n^k = \omega_n^{n/2} \cdot \omega_n^{k-n/2} = -\omega_n^{k'}
$$

FFT算法说明

- 考虑的是理想情况，$n=2^k$
- 事先算出**n/2**个旋转因子
- 若仔细考虑，可不增加额外存储量 **(**原地工作**)**
- 若输入序列**x**为实数序列，可使计算**/**存储量减半
- 在某些算法实现中，输出并非按正常的顺序，可有两种处理方法：在进行反**FFT**变换时把顺序调整回来；将输出再进行排序（也是**nlogn**的运算量）
- 实际程序中将递归变为循环，见netlib/fftpack

$\mathcal{O}(n^2)$与$\mathcal{O}(n\log_2 n)$的区别

Use of FFT algorithm reduces work to only $\mathcal{O}(n \log_2 n)$, which makes enormous practical difference in time required to transform large sequences

n	$n \log_2 n$	n^2
64	384	4096
128	896	16384
256	2048	65536
512	4608	262144
1024	10240	1048576

反变换同样可以用FFT算法

$$x_k = \frac{1}{n} \sum_{m=0}^{n-1} y_m \underline{\omega_n^{-mk}}, \quad k = 0, 1, \ldots, n-1$$

Limitations of FFT

FFT algorithm is not always applicable or maximally efficient

Input sequence assumed to be:

- Equally spaced
- Periodic （频域分析的含义）
- Power of two in length

First two of these follow from definition of DFT, while third is required for maximal efficiency of FFT algorithm

Care must be taken in applying FFT algorithm to produce most meaningful results as efficiently as possible

For example, transforming sequence that is not really periodic or padding sequence to make its length power of two may introduce spurious noise and complicate interpretation of results

铺垫

伪的

Mixed-Radix FFT

混合基数FFT

It is possible to define "mixed-radix" FFT algorithm that does not require number of points n to be power of two

More general algorithm is still based on divide-and-conquer; sequence is not necessarily split exactly in half at each level, but by smallest prime factor of remaining sequence length

质数因子

Efficiency depends on whether n is product of small primes (ideally power of two)

If not, then much of computational advantage of FFT may be lost

For example, if n itself is prime, then sequence cannot be split at all, and "fast" algorithm becomes standard $\mathcal{O}(n^2)$ matrix-vector multiplication

Applications of DFT

Wenjian Yu

FFT的应用

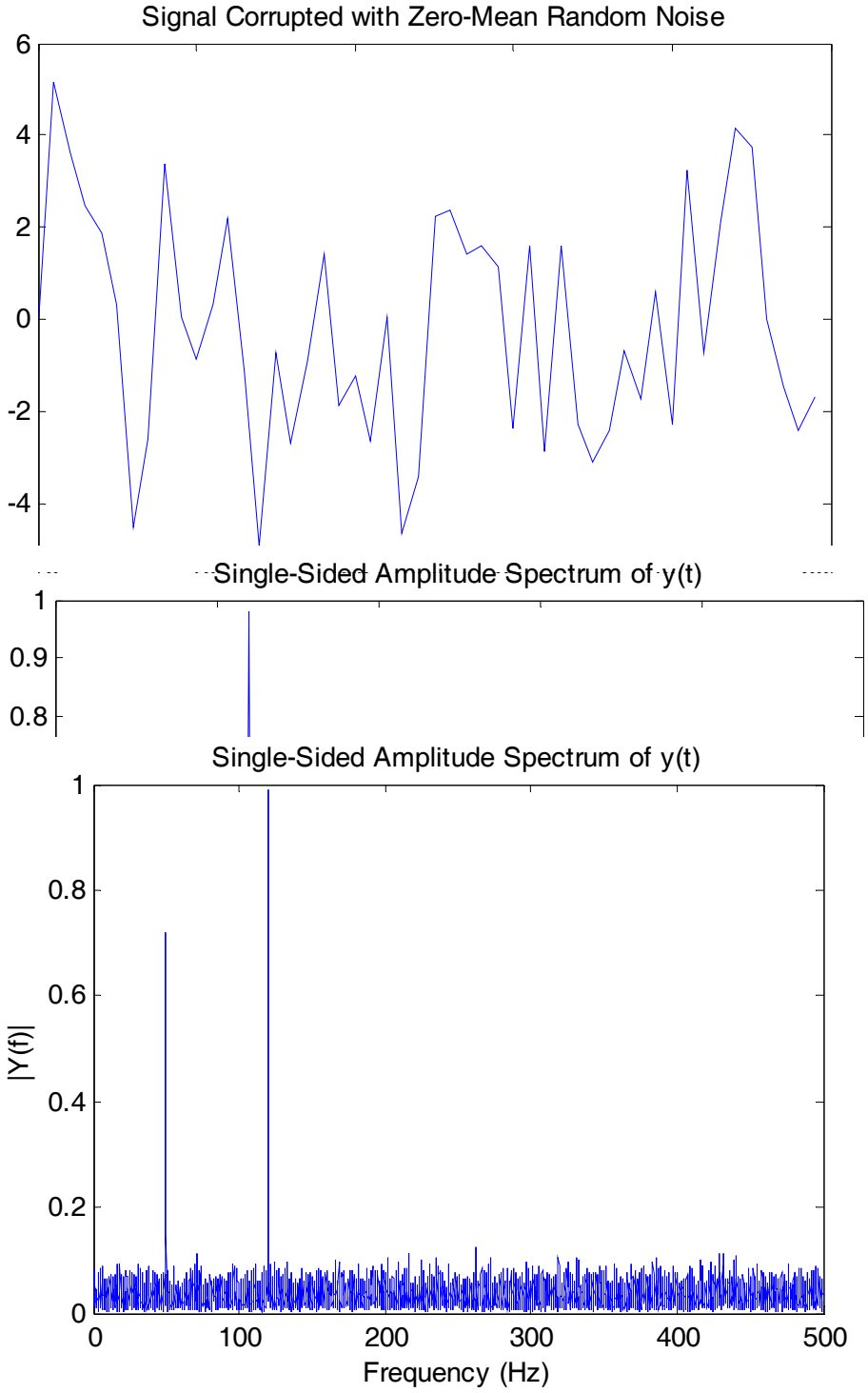
- 信号周期检测
  - 检测离散数据的周期
- 数字滤波
  - 过滤掉不需要的频率分量(比如噪声)
  - 隔离出某个频率的数据 (天气数据有日周期, 年周期)
- 多项式相乘
- 离散卷积计算
- 泊松方程快速求解

信号周期检测

$$x(t) = 0.7\sin(2\pi \cdot 50t) + \sin(2\pi \cdot 120t)$$

- 采样频率: f_s=1000
- 采样长度: L=1000, t=(0:L-1)/f_s
- 带噪声的波形
  y = x + 2*randn(size(t));
- 做**FFT:** NFFT = 2^nextpow2(L
  Y = fft(y,NFFT)/L;
- 输出: f = f_s /2*linspace(0,1,NF
  plot(f,2*abs(Y(1:NFFT/2+1)))
- f=50, 120两个频率点, 幅度
- 修改L=10000, 结果更好

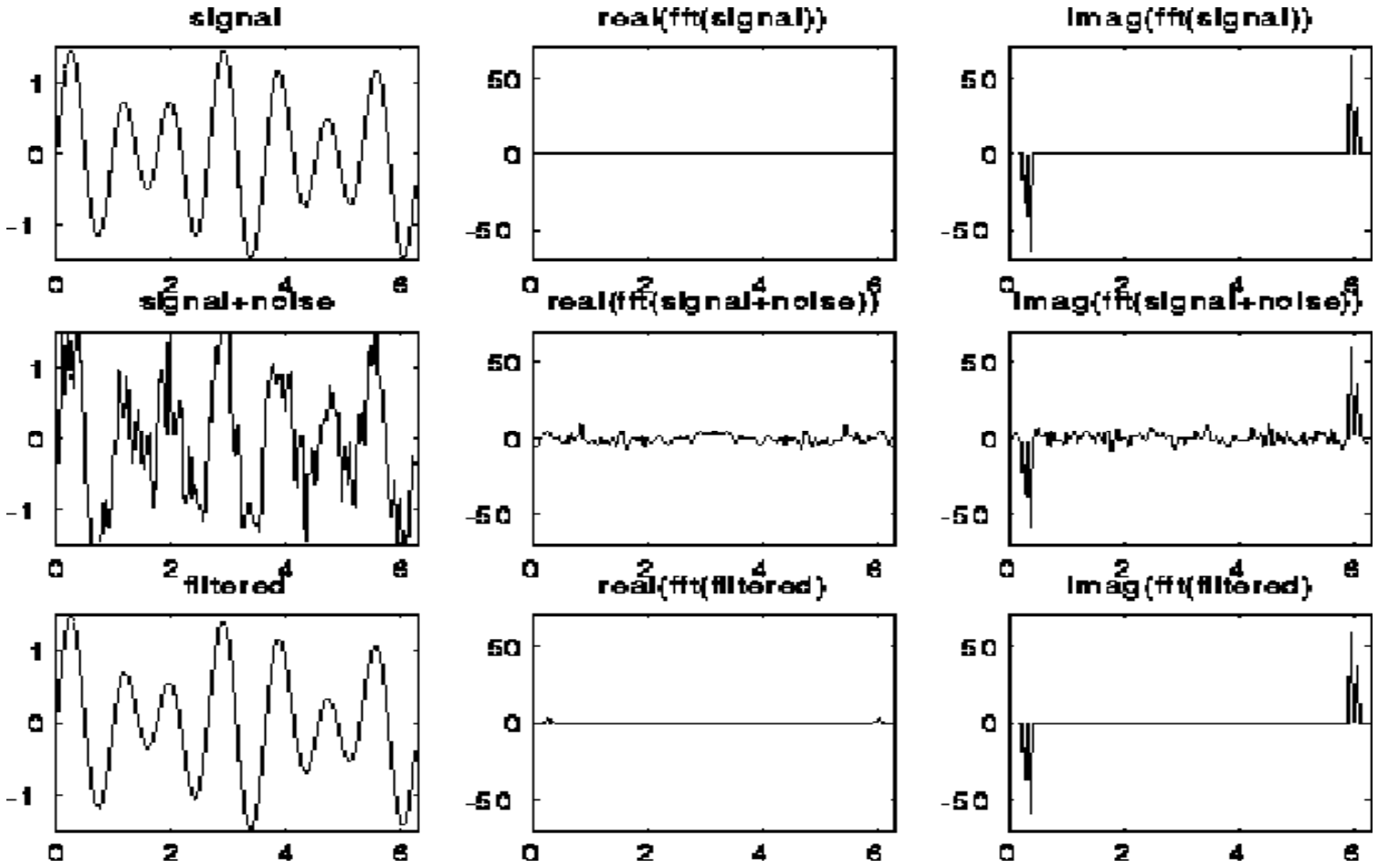
演示： **fftexp1.m**



Using the 1D FFT for filtering

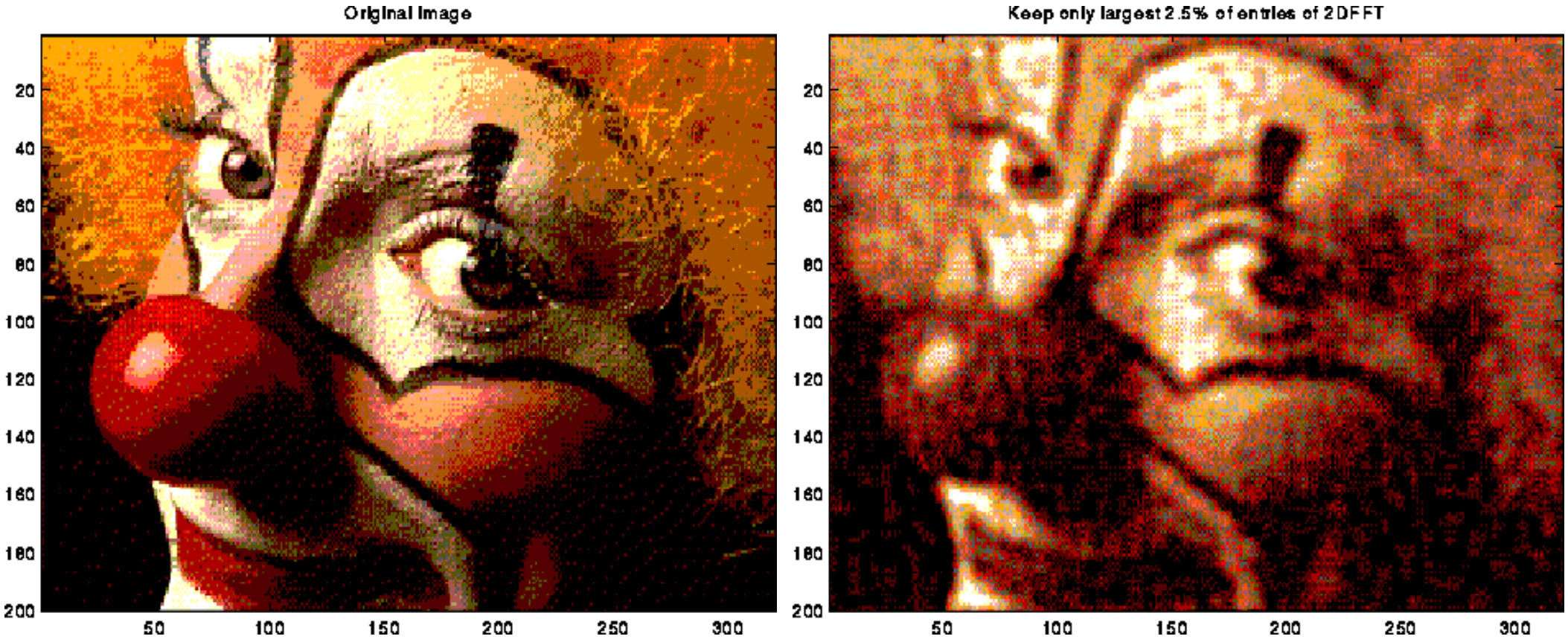
- Signal = sin(7t) + 0.5 sin(5t) at 128 points
- Noise = random number bounded by 0.75
- Filter by zeroing out FFT components < 0.25

(或者去除高频分量)



Using the 2D FFT for image compression

- Image = 200x320 matrix of values
- Compress by keeping largest 2.5% of FFT components



Fast Polynomial Multiplication

FFT algorithm also provides fast method for some computations that might not seem related to it

For example, complexity of straightforward multiplication of two polynomials is proportional to product of their degrees

However, polynomial of degree $n-1$ is uniquely determined by its values at n distinct points

Thus, product polynomial can be determined by interpolation from pointwise product of factor polynomials at n points

Both polynomial evaluation and interpolation using n points would normally require $\mathcal{O}(n^2)$ operations, but by choosing points to be nth roots of unity, FFT algorithm can be used to reduce complexity to $\mathcal{O}(n \log_2 n)$

$$p(t) = x_0 + x_1 t + \cdots + x_{l-1} t^{l-1}$$

$$\times$$

$$q(t) = y_0 + y_1 t + \cdots + y_{m-1} t^{m-1}$$

计算 $p \cdot q \sim \mathcal{O}(lm)$

n-1阶多项式在n点求值;
已知n点多项式的值求系数

$$\begin{bmatrix} 1 & t_0 & \cdots & t_0^{n-1} \\ 1 & t_1 & \cdots & t_1^{n-1} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & t_{n-1} & \cdots & t_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

取这n点为 ω_n^k, k=0,...,n-1

n-1≥ l+m-2, n为2的整数幂

DFT和inverse DFT

$$\sim \mathcal{O}(n \log_2 n) + \mathcal{O}(n)$$

Matlab code: polymul.m

Applications of DFT, continued

Some computations are simpler or more efficient in frequency domain than in time domain

Examples include discrete convolution of two sequences $\boldsymbol{u}$ and $\boldsymbol{v}$ of length n,

$$\{\boldsymbol{u} \star \boldsymbol{v}\}_m = \sum_{k=0}^{n-1} v_k u_{m-k}, \quad m = 0, 1, \ldots, n-1,$$

and related quantities such as cross correlation of two sequences or autocorrelation of a sequence with itself

离散卷积

实际上是两个**n-1**阶多项式乘积的系数

低次项系数

相关性

$$DFT^{-1}(DFT(\boldsymbol{u}).*DFT(\boldsymbol{v}))$$

Equivalent operation in frequency domain is simply pointwise multiplication

$$\mathcal{O}(n^2)$$

$$\downarrow$$

$$\mathcal{O}(n\log_2 n) + \mathcal{O}(n)$$

FFT Resource and FFT Variations

- Netlib/fftpack, www.netlib.org
- FFT in the west, www.fftw.org
- Produces FFT implementation optimized for
  - Your version of FFT (complex, real,…)
  - Your value of n (arbitrary, possibly prime)
  - Your architecture
- Won 1999 Wilkinson Prize for Numerical Software
- FFT Variations (for various applications)
  - The sin, cos transform – imaginary, real part of FFT
  - Wavelet, precorrected FFT for integral equation

Fast Solver for Poisson Equation

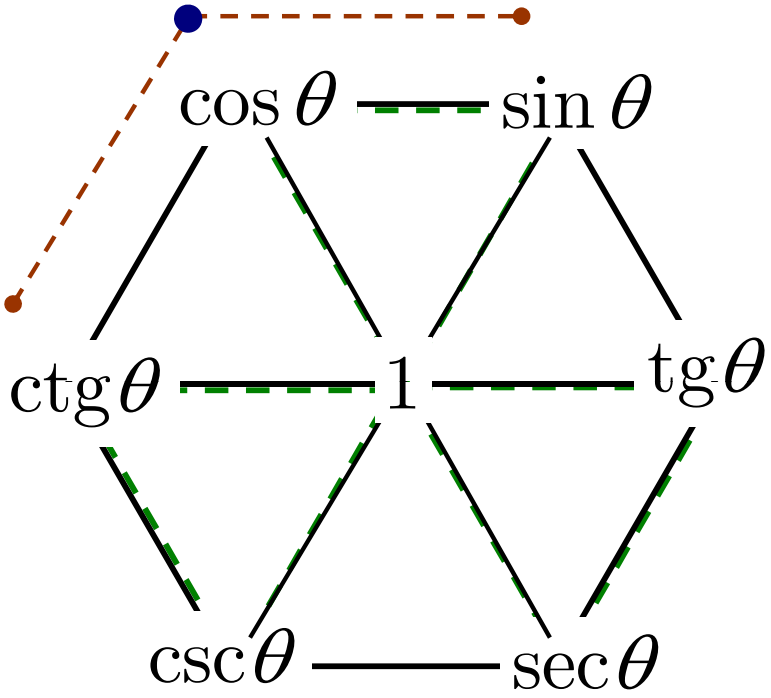
Wenjian Yu

Fast Solver for Poisson Equation

- 1-D model problem
  - Eigenvalues and Eigenvectors
- 2-D model problem
  - Two ways to present the equations as a single matrix equation
  - Eigenvalues and Eigenvectors
  - Solve the Poisson equation using the eigen-decomposition
- Fast Solver with FFT

规则区域的PDE问题

Review of trigonometric functions



三角函数的基本性质（图解释）：

- 对角的顶点互为倒数
- 三角形上面两顶点平方和为下面一顶点
- 连续的三顶点，中间为两边点乘积

和差公式：

$$\sin(\alpha+\beta) = \sin\alpha\cos\beta + \sin\beta\cos\alpha$$
$$\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta.$$

倍角公式：

$$\sin(2\alpha) = 2\sin\alpha\cos\alpha$$
$$\cos(2\alpha) = \cos^2\alpha - \sin^2\alpha$$

$$\sin(3\alpha) = 3\sin\alpha - 4\sin^3\alpha$$
$$\cos(3\alpha) = 4\cos^3\alpha - 3\cos\alpha$$

Review of trigonometric functions

和差化积—积化和差公式：

$$\sin\alpha+\sin\beta=2\sin[\tfrac{1}{2}(\alpha+\beta)]\cos[\tfrac{1}{2}(\alpha-\beta)]$$

$$\sin\alpha-\sin\beta=2\cos[\tfrac{1}{2}(\alpha+\beta)]\sin[\tfrac{1}{2}(\alpha-\beta)]$$

$$\cos\alpha+\cos\beta=2\cos[\tfrac{1}{2}(\alpha+\beta)]\cos[\tfrac{1}{2}(\alpha-\beta)]$$

$$\cos\alpha-\cos\beta=-2\sin[\tfrac{1}{2}(\alpha+\beta)]\sin[\tfrac{1}{2}(\alpha-\beta)].$$

$$\sin\alpha\cos\beta=\tfrac{1}{2}[\sin(\alpha-\beta)+\sin(\alpha+\beta)]$$

$$\cos\alpha\cos\beta=\tfrac{1}{2}[\cos(\alpha-\beta)+\cos(\alpha+\beta)]$$

$$\cos\alpha\sin\beta=\tfrac{1}{2}[\sin(\alpha+\beta)-\sin(\alpha-\beta)]$$

$$\sin\alpha\sin\beta=\tfrac{1}{2}[\cos(\alpha-\beta)-\cos(\alpha+\beta)].$$

1-D Poisson Problem

$$-\frac{d^2u(x)}{dx^2}=f(x) \quad \Longrightarrow \quad -\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}=f_i$$

$$T_N\cdot\begin{bmatrix}u_1\\ \vdots\\ \vdots\\ u_N\end{bmatrix}\equiv\begin{bmatrix}2 & -1 & & 0\\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1\\ 0 & & -1 & 2\end{bmatrix}\cdot\begin{bmatrix}u_1\\ \vdots\\ \vdots\\ u_N\end{bmatrix}=h^2\begin{bmatrix}f_1\\ \vdots\\ \vdots\\ f_N\end{bmatrix}$$

分析T_N的特征值：

1. 根据圆盘定理：$|\lambda-2|\le 2$ ，设 $\lambda_k=2-2\cos\theta_k,\ \theta_k\in[0,\ \pi]$
2. 特征多项式 $P_N=det(\lambda I-T_N)$ ： $P_1=\lambda-2=-\dfrac{\sin 2\theta}{\sin\theta}$

$$P_2=(\lambda-2)^2-1=4\cos^2\theta-1=3-4\sin^2\theta=\frac{\sin 3\theta}{\sin\theta}$$

1-D Poisson Problem

$$T_N \equiv \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix}$$

特征多项式

$$P_N = (\lambda - 2)P_{N-1} - P_{N-2} = -2\cos\theta P_{N-1} - P_{N-2}$$

$$= (-1)^N \frac{\sin(N+1)\theta}{\sin\theta}$$

在 $\theta_k \in [0,\ \pi]$ 范围内的解： $\theta_k = \dfrac{k\pi}{N+1},\ k = 1,\ 2,\ \cdots,\ N$

特征值为： $\lambda_k = 2 - 2\cos\theta_k$

特征向量为 z_k，其归一化后的分量 $z_k(j) = \sqrt{\dfrac{2}{N+1}}\sin\left(\dfrac{jk\pi}{N+1}\right)$

矩阵特征值分解为 $T_N = Z\Lambda Z^T$

j =1, …，验证?

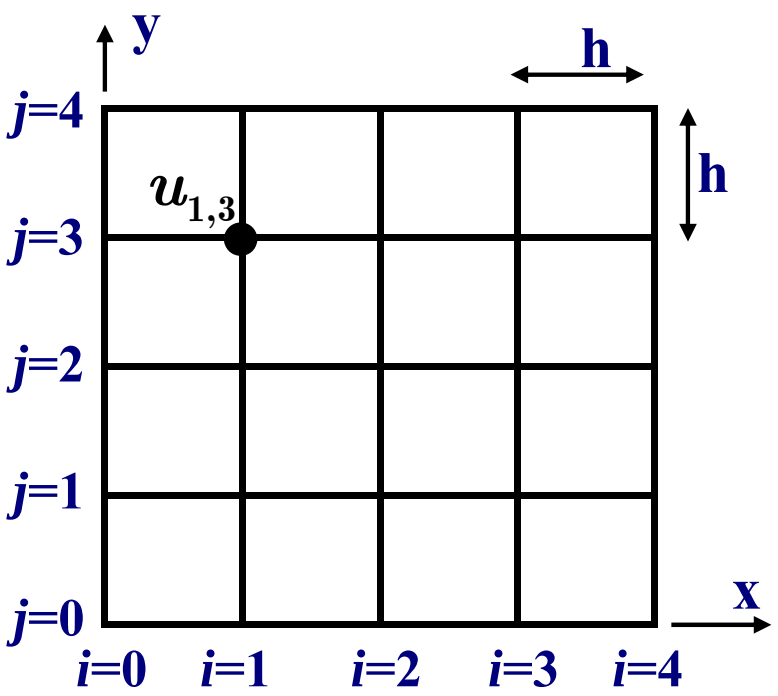
其实Z为正交、对称矩阵

2-D Poisson Problem

$$-\frac{\partial^2 u(x,y)}{\partial x^2}-\frac{\partial^2 u(x,y)}{\partial y^2}=f(x,y)$$

二阶中心差分　　$1 \le i,j \le N$

$$4u_{i,j}-u_{i-1,j}-u_{i+1,j}-u_{i,j-1}-u_{i,j+1}=h^2 f_{i,j}$$



$$T_{N\times N}=\left[\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right]$$

变量按列分成N组

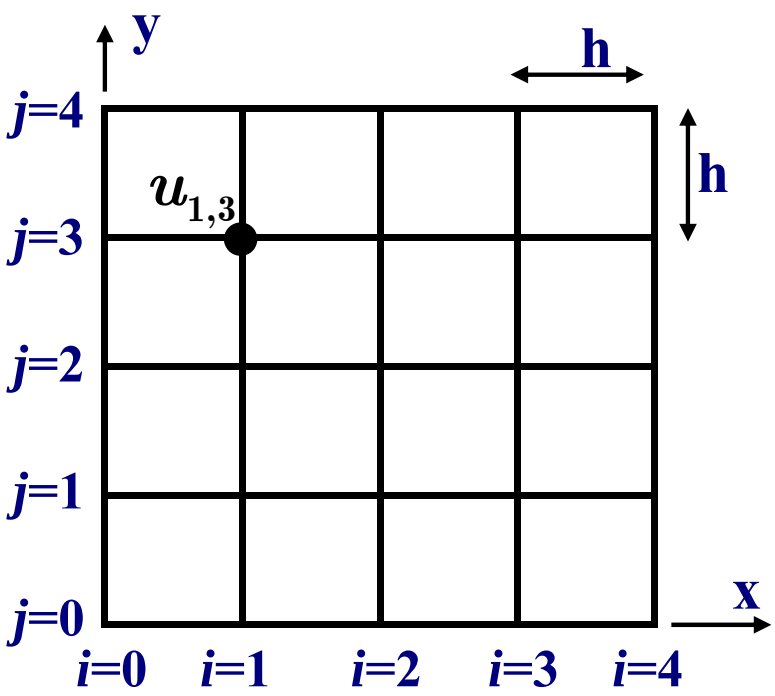
$$T_{N\times N}\equiv\begin{bmatrix} T_N+2I_N & -I_N & & \\ -I_N & \ddots & \ddots & \\ & \ddots & \ddots & -I_N \\ & & -I_N & T_N+2I_N \end{bmatrix}$$

矩阵维数: N^2, 直接解法 ~ $\mathcal{O}(N^4)$

2-D Poisson Problem

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{i,j}$$

$$1 \le i, j \le N$$



为了采用特征值方法，将方程写为另一种形式

设$u_{i,j}$组成一个$N \times N$的矩阵U(每行对应于一列网格点)

$$2u_{i,j} - u_{i\text{-}1,j} - u_{i+1,j} = (T_N \cdot U)_{i,j}$$

$$2u_{i,j} - u_{i,j\text{-}1} - u_{i,j+1} = (U \cdot T_N)_{i,j}$$

$$T_N \cdot U + U \cdot T_N = h^2 F$$

$$T_N \equiv \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix}$$

2-D Poisson Problem

Solve the Poisson equation with eigen-decomposition

$$T_N \cdot U + U \cdot T_N = h^2 F$$

将 $T_N = Z \Lambda Z^T$ 代入，左乘 Z^T 右乘 Z

$$Z^T (Z \Lambda Z^T) U Z + Z^T U (Z \Lambda Z^T) Z = Z^T (h^2 F) Z$$

$$\Lambda Z^T U Z + Z^T U Z \Lambda = h^2 Z^T F Z$$

$$\Lambda U' + U' \Lambda = h^2 F' \text{ ,} \qquad \text{设 } U' = Z^T U Z,\ F' = Z^T F Z$$

解出 $u'_{jk} = \dfrac{h^2 f'_{jk}}{\lambda_j + \lambda_k}$

$1 \le j, k \le N$

算法：

1) $F' = ZFZ$

2) For all j and k, $u'_{jk} = \dfrac{h^2 f'_{jk}}{\lambda_j + \lambda_k}$

3) $U = ZU'Z$

计算量 $\sim \mathcal{O}(N^3)$

(Z矩阵对称)

Fast Solver with FFT

在上一个算法中，

$$Z_{jk} = \sqrt{\frac{2}{N+1}} \sin \frac{jk\pi}{N+1}, \qquad 1 \le j,k \le N$$

考虑**2*N*+2**阶的旋转因子$\omega_{2N+2} = e^{-2\pi i/(2N+2)}$，对应**DFT**变换矩阵为：

$$\left\{F_{2N+2}\right\}_{jk} = \omega_{2N+2}^{jk} = \cos \frac{jk\pi}{N+1} - i\sin \frac{jk\pi}{N+1}, \qquad 0 \le j,k \le 2N+1$$

说明矩阵**Z**为**DFT**矩阵 F_{2N+2}的**(2:N+1, 2:N+1)**子矩阵的虚部的$\sqrt{\frac{2}{N+1}}$倍

算法：

1) $F' = ZFZ$ $\quad Z(ZF^T)^T$

2) For all j and k, $u'_{jk} = \dfrac{h^2 f'_{jk}}{\lambda_j + \lambda_k}$

3) $U = ZU'Z$

计算 Zx：
- 用零扩充x向量为2N+2维;
- FFT(x);
- 结果的N分量虚部× $\sqrt{\frac{2}{N+1}}$

计算量 ~ $\mathcal{O}(N^2 log_2 N)$

Further Discussion

- Extend the eigenvalue analysis to 3-D model problem
  - Use Kronecker product

$$A \otimes B = \begin{bmatrix} a_{1,1}B & \cdots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,n}B \end{bmatrix}$$

- Fast PDE solvers
  - Fast sine transform (multiple by Z directly)
  - FFT-based algorithm is **direct solver (直接法)**
  - The FFT-based solver is very limited: uniformed discretization、power of 2、boundary condition
  - Other approaches: cyclic reduction, multi-grid (迭代法)
- Reference
  - J. Demmel, *Applied Numerical Linear Algebra*, SIAM Press, 1997

About the Final Project

- 选题情况
  - : 最速下降法与共轭梯度法求解二维线性方程组的演示程序
  - 矩阵特征值计算的演示程序(QR方法vs Jacobi方法, 改编eigsvdgui程序)
  - : 求解下三角稀疏矩阵的Lsolve算法的演示
  - 欧拉方法求解ODE初值问题的演示程序(改编Heath网站程序)
  - : 求解常微分方程组时的混沌现象(改编lorenzgui程序)
  - 有关矩阵特征值的圆盘定理与幂法演示程序(改编Heath网站程序)
- 要求
  - 提交实验报告, 程序 (deadline: 1/18/2011)；
  - 程序演示：元旦后，1月17日晚