第一讲　Toeplitz 矩阵相关

本章主要介绍与 Toeplitz 矩阵相关的一些计算, 包括

- Toeplitz 矩阵
- 循环 (Circulant) 矩阵
- 快速 Fourier 变换和快速三角变换
- 循环矩阵和 Toeplitz 矩阵与向量乘积的快速算法

1.1　三类具有特殊结构的矩阵

1.1.1　Toeplitz 矩阵

具有下面形式的矩阵称为 Toeplitz 矩阵 (即在同一条对角线上的元素都相等)

$$T = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-n+1} \\ t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{1.1}$$

- Toeplitz 矩阵只有 $2n-1$ 个独立元素, 因此在存储一个 Toepliz 矩阵时, 只需存储第一列和第一行 (或第一列和最后一列);
- 如果 T 对称, 则只需存储第一列.

定义 1.1 (Persymmetry)　如果矩阵 $A \in \mathbb{R}^{n\times n}$ 满足

$$A^T = JAJ \quad 或 \quad (JA)^T = JA,$$

则称 A 是 *persymmetric*, 其中 J 是 n 阶反转矩阵 *(reversal matrix)* 或交换矩阵 *(exchange matrix, [6, page 20])*, 即

$$J = \begin{bmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{bmatrix}. \tag{1.2}$$

- 如果 A 是 persymmetric, 则 JA 和 AJ 都是对称的;
- A 是 persymmetric 当且仅当

$$a_{ij} = a_{n+1-j,n+1-i}, \quad i,j = 1,2,\ldots,n,$$

即 A 关于反对角线 (antidiagonal) 是对称的.

引理 1.1 设 $A \in \mathbb{R}^{n\times n}$ 是 *persymmetric*. 如果 A 非奇异, 则 A^{-1} 也是 *persymmetric*.

证明. 由 $J^{-1} = J = J^T$ 和 $(AJ)^T = AJ$ 可得

$$(JA^{-1})^T = (J^{-1}A^{-1})^T = ((AJ)^{-1})^T = ((AJ)^T)^{-1} = (AJ)^{-1} = J^{-1}A^{-1} = JA^{-1}.$$

□

推论 1.1 非奇异 *Toeplitz* 矩阵的逆是 *persymmetric*. (但一般不是 *Toeplitz* 矩阵)

1.1.2 循环矩阵

循环矩阵 (circulant matrix) 是一类特殊的 Toeplitz 矩阵, 具有下面的形式

$$C = \begin{bmatrix} z_0 & z_{n-1} & \cdots & z_1 \\ z_1 & z_0 & \cdots & z_2 \\ \vdots & \vdots & & \vdots \\ z_{n-2} & z_{n-3} & \cdots & z_{n-1} \\ z_{n-1} & z_{n-2} & \cdots & z_0 \end{bmatrix} \triangleq C(z),$$

其中 $z = [z_0, z_1, \ldots, z_{n-1}]^T$. 易知循环矩阵的第二列是由第一列往下移一位所得到, 第三列则是由第二列再往下移一位所得到, 依此类推.

☞ 循环矩阵 C 由其第一列 z 所确定, 因此只需存储第一列.

引理 1.2 设 $C = C(z)$ 是循环矩阵, 则

$$C = \sum_{k=0}^{n} z_k L^k,$$

其中 L 是 *downshift permutation [6, page 20]*, 即

$$L = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & 0 & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}. \tag{1.3}$$

1.1.3 Hankel 矩阵

具有下面形式的矩阵称为 Hankel 矩阵 (即在同一条反对角线上的元素都相等)

$$H = \begin{bmatrix} h_{-n+1} & \cdots & h_{-1} & h_0 \\ \vdots & \iddots & \iddots & h_1 \\ h_{-1} & \iddots & \iddots & \vdots \\ h_0 & h_1 & \cdots & h_{n-1} \end{bmatrix} \in \mathbb{R}^{n\times n}. \tag{1.4}$$

易知 Hankel 是对称矩阵.

☞ 若 H 是 Hankel 矩阵, 则 HJ 是 Toeplitz 矩阵, 其中 J 是反转矩阵 (见 (1.2)).

1.2 循环矩阵与快速 Fourier 变换

1.2.1 Fourier 变换

Fourier 变换是函数空间上的一类非常重要的线性变换, 它是通过 Fourier 积分来定义的, 而 Fourier 积分则是周期函数的 Fourier 级数的自然推广. 为了简单起见, 我们只考虑实数域上函数.

Fourier 级数

设 $f(x)$ 是周期为 $T=2l$ 的实值函数, 则其在 $[-l,l]$ 上的 Fourier 展开为

$$f(x)\sim\frac{a_0}{2}+\sum_{k=1}^{\infty}\left(a_k\cos\frac{k\pi x}{l}+b_k\sin\frac{k\pi x}{l}\right) \tag{1.5}$$

其中

$$\begin{cases} a_0 &= \dfrac{1}{l}\displaystyle\int_{-l}^{l} f(\tau)\,\mathrm{d}\tau, \\ a_k &= \dfrac{1}{l}\displaystyle\int_{-l}^{l} f(\tau)\cos\frac{k\pi\tau}{l}\,\mathrm{d}\tau, \\ b_k &= \dfrac{1}{l}\displaystyle\int_{-l}^{l} f(\tau)\sin\frac{k\pi\tau}{l}\,\mathrm{d}\tau, \quad k=1,2,\dots. \end{cases}$$

当 Fourier 展开式 (1.5) 右边的三角级数一致收敛时, 符号 "$\sim$" 可以改成 "$=$".

定理 1.2 如果 $f(x)$ 满足 *Dirichlet* 条件, 即:

- 连续或只有有限多个第一类间断点 (跳跃点);
- 只有有限多个极值点;

则 $f(x)$ 在其连续点上可以展开为 *Fourier* 级数, 即等号成立. 而在其间断点 $\tilde{x}$ 处, 其 *Fourier* 级数收敛到

$$\frac{f(\tilde{x}+0)+f(\tilde{x}-0)}{2}.$$

利用 Euler 公式, 可以将 Fourier 级数写成复数形式, 即

$$f(x)=\sum_{k=-\infty}^{+\infty}c_k e^{ik\omega x}, \tag{1.6}$$

其中 $\omega=\pi/l$,

$$c_k=\frac{1}{2l}\int_{-l}^{l}f(\tau)e^{-ik\omega\tau}\,\mathrm{d}\tau. \tag{1.7}$$

对于非周期函数, 可视周期为 $T=\infty$, 在一定条件下可展开为 Fourier 积分. 由 (1.6) 和 (1.7) 可知

$$f(x)=\lim_{l\to\infty}\sum_{k=-\infty}^{+\infty}\left(\frac{1}{2l}\int_{-l}^{l}f(\tau)e^{-ik\omega\tau}\,\mathrm{d}\tau\right)e^{ik\omega x}.$$

由于 $l=\pi/\omega$, 因此上式可写为

$$f(x)=\lim_{\omega\to0}\sum_{k=-\infty}^{+\infty}\left(\frac{\omega}{2\pi}\int_{-\frac{\pi}{\omega}}^{\frac{\pi}{\omega}}f(\tau)e^{-ik\omega\tau}\,\mathrm{d}\tau\right)e^{ik\omega x}.$$

记 $\Delta\xi=\omega$, $\xi_k=k\omega=k\Delta\xi$, 则

$$f(x)=\frac{1}{2\pi}\lim_{\Delta\xi\to0}\sum_{k=-\infty}^{+\infty}\left(\int_{-\frac{\pi}{\Delta\xi}}^{\frac{\pi}{\Delta\xi}}f(\tau)e^{-i\xi_k\tau}\,\mathrm{d}\tau\right)e^{i\xi_k x}\Delta\xi.$$

这是一个和式的极限. 根据积分的定义, 在一定条件下, 上式可写为

$$f(x)=\frac{1}{2\pi}\int_{-\infty}^{+\infty}\int_{-\infty}^{\infty}f(\tau)e^{-i\xi\tau}\,\mathrm{d}\tau\,e^{i\xi x}\,\mathrm{d}\xi. \tag{1.8}$$

这就是函数 $f(x)$ 的 Fourier 积分公式.

☞ 上面只是一个形式上的推导, 并不是每个函数都存在 Fourier 积分公式.

定理 1.3 如果 $f(x)$ 在 $(-\infty,+\infty)$ 上满足

- 在任意有限区间上都满足 *Dirichlet* 条件;
- $f(x)$ 在 $(-\infty,+\infty)$ 上绝对可积;

则 $f(x)$ 在其连续点上有

$$f(x)=\frac{1}{2\pi}\int_{-\infty}^{+\infty}\int_{-\infty}^{\infty}f(\tau)e^{-i\xi\tau}\,\mathrm{d}\tau\,e^{i\xi x}\,\mathrm{d}\xi.$$

1.2.2 离散 Fourier 变换

设 $u=[\ldots,u_{-1},u_0,u_1,\ldots]\in l_2$, 则其离散 Fourier 变换定义为

$$\hat{u}(\xi)=\frac{1}{\sqrt{2\pi}}\sum_{k=-\infty}^{+\infty}u_k e^{-ik\xi},\quad \xi\in[-\pi,\pi].$$

易知 $\hat{u}(\xi)\in L_2[-\pi,\pi]$.

离散 Fourier 逆变换:

$$u_k=\frac{1}{\sqrt{2\pi}}\int_{-\pi}^{\pi}\hat{u}(\xi)e^{ik\xi}\,\mathrm{d}\xi.$$

$$\|u\|_{l_2}=\|\hat{u}\|_{L_2}$$

1.2.3 DFT 与 FFT

快速 Fourier 变换 (FFT, Fast Fourier Transform) 是用来计算周期离散信号或有限长离散信号 Fourier 变换 (DFT, Discrete Fourier Transform) 的一种快速算法.

有限长 DFT

周期离散信号或有限长离散信号的 Fourier 变换定义如下:

定义 1.2 我们称向量 $y=[y_0,y_1\ldots,y_{n-1}]\in\mathbb{C}^n$ 是 $x=[x_0,x_1\ldots,x_{n-1}]\in\mathbb{C}^n$ 的 *DFT*, 其中

$$y_k=\sum_{j=0}^{n-1}x_j\omega_n^{kj},\quad k=0,1,\ldots,n-1. \tag{1.9}$$

这里

$$\omega_n=e^{-\frac{2\pi\mathrm{i}}{n}}=\cos\left(\frac{2\pi}{n}\right)-\mathrm{i}\sin\left(\frac{2\pi}{n}\right)$$

是 1 的一个 n 次根. [a]

[a] 有的文献里取 $\omega_n=e^{2\pi\mathrm{i}/n}$, 这没有实质性的区别.

☞ 如果没有特殊说明, 下面所涉及的 DFT 均指周期离散信号或有限长离散信号的 Fourier 变换.

DFT (1.9) 可以表示成如下的矩阵与向量的乘积:

$$y=F_n x,$$

其中

$$F_n=[f_{kj}]_{n\times n}=\begin{bmatrix}1&1&1&\cdots&1\\1&\omega_n&\omega_n^2&\cdots&\omega_n^{n-1}\\1&\omega_n^2&\omega_n^4&\cdots&\omega_n^{2(n-1)}\\\vdots&\vdots&&\ddots&\vdots\\1&\omega_n^{n-1}&\omega_n^{(n-1)2}&\cdots&\omega_n^{(n-1)(n-1)}\end{bmatrix}$$

即

$$f_{kj} = \omega_n^{(k-1)(j-1)}, \qquad k, j = 1, 2, \ldots, n.$$

我们称 F_n 为 n 阶 DFT 矩阵 [6, page 34].

容易看出, DFT 矩阵是一类特殊的 Vandermonde 矩阵, 因此有

$$\det(F_n) = \prod_{j \neq k} |\omega_n^k - \omega_n^j| \neq 0.$$

所以 F_n 是非奇异的. 因此有

$$x = F_n^{-1} y,$$

这就是离散 Fourier 逆变换 (Inverse DFT, 简记 IDFT).

由 F_n 的表示形式可以看出, F_n 是对称的, 但不是 Hermite 对称的, 因此 $F_n^* = \bar{F}_n$. 由于 $\bar{\omega}_n = \omega_n^{-1}$, 通过计算可得

$$(F_n(:, p+1))^* F_n(:, q+1) = \sum_{k=0}^{n-1} \bar{\omega}_n^{kp} \omega_n^{kq} = \sum_{k=0}^{n-1} \omega_n^{k(q-p)}.$$

若 $p = q$, 则上式右端等于 n, 否则

$$\sum_{k=0}^{n-1} \omega_n^{k(q-p)} = \frac{1 - \omega_n^{n(q-p)}}{1 - \omega_n^{q-p}} = 0.$$

于是有

引理 1.3 设 F_n 是 n 阶 *DFT* 矩阵, 则

$$F_n^* F_n = nI, \quad F_n^{-1} = \frac{1}{n} F_n^* = \frac{1}{n} \bar{F}_n,$$

即 $\dfrac{1}{\sqrt{n}} F_n$ 是酉矩阵.

DFT 与 FFT

DFT 和卷积是数字信号处理中两个最基本也是最常用的运算, 而卷积可化为 DFT 来实现.

如果采用传统的方法计算 DFT (1.9), 则需要 $\mathcal{O}(n^2)$ 次复数运算. 当 n 很大时, 其计算量是相当可观的, 特别是在进行二维或三维数字图像处理时, 计算量更是大得惊人, 难以 "实时" 实现.

但通过观察发现, 在 DFT 运算中包含大量的重复运算. 通过巧妙利用 ω_n 的周期性和对称性, Cooley 和 Tukey [4] 于 1965 年提出了快速 Fourier 变换算法, 使得 n 个点的 DFT 的乘法计算量由 n^2 降到 $\frac{1}{2} N \log_2 N$. 这一重要发现被公认为是数字信号处理发展史上的一个转折点, 也可以称之为一个里程碑.

自 Cooley-Tukey 算法提出之后, 新的算法不断涌现. 总的来说, FFT 算法的发展方向有两个: 一是针对 n 为 2 的整数次幂的算法, 如基 2 算法, 基 4 算法, 实因子算法, 分裂基算法等; 另一个是 n 不是 2 的整数次幂的算法, 如素因子算法, Winograd 算法等. 关于 FFT 算法的详细介绍可以参见 [5, 8]. 事实上, FFT 算法的历史可追溯到 Gauss 的工作 [7]. FFT 算法被评为二十世纪的十大优秀算法之一 [3].

在 MATLAB 中, 可通过函数 `fft` 和 `ifft` 来实现 DFT 和 IDFT 的计算.

MATLAB 源代码 1.1 FFT

```
1  y=fft(x); % y=F*x
2  x=ifft(y); % y=F^{-1}*x
```

1.2.4 循环矩阵与 DFT

由引理 1.2 可知,

$$C = \sum_{k=0}^{n} z_k L^k,$$

其中 L 是 downshift permutation (1.3). 通过直接计算可得

$$F_n L = W F_n,$$

其中

$$W \triangleq \mathrm{diag}(1, \omega_n, \omega_n^2, \dots, \omega_n^{n-1}). \tag{1.10}$$

于是有

$$L^k = (F_n^{-1} W F_n)^k = F_n^{-1} W^k F_n = \frac{1}{n} F_n^* W^k F_n \triangleq \tilde{F}_n^* W^k \tilde{F}_n,$$

其中 $\tilde{F}_n \triangleq \dfrac{1}{\sqrt{n}} F_n$ 是酉矩阵. 所以

$$C = \sum_{k=0}^{n} z_k L^k = \sum_{k=0}^{n} z_k \tilde{F}_n^* W^k \tilde{F}_n \triangleq \tilde{F}_n^* \Lambda \tilde{F}_n, \tag{1.11}$$

其中 $\Lambda = \sum_{k=0}^{n} z_k W^k \triangleq \mathrm{diag}(\lambda_0, \lambda_1, \dots, \lambda_{n-1})$ 是对角矩阵, 其对角线元素为

$$\lambda_k = \sum_{j=0}^{n-1} z_j \omega_n^{kj}, \quad k = 0, 1, 2, \dots, n-1.$$

容易验证

$$[\lambda_1, \lambda_2, \dots, \lambda_n]^T = F_n z. \tag{1.12}$$

又 $\tilde{F}_n$ 是酉矩阵, 故等式 (1.11) 即为循环矩阵 C 的特征值分解, 并且还是酉分解. 所以 λ_k 即为 C 的特征值, 并且这些特征值可以通过 (1.12) 进行计算, 即对向量 z 做一次 FFT 即可.

因此, 对任意一个向量 $v \in \mathbb{R}^n$, 通过下面的方式来计算循环矩阵 $C = C(z)$ 与 v 的乘积:

$$Cv = \tilde{F}_n^* \Lambda \tilde{F}_n v = F_n^{-1}(\Lambda(F_n v)).$$

算法 1.1 计算循环矩阵 $C(z)$ 与向量 v 的乘积

1: Lam = fft(z) % 计算循环矩阵的特征值
2: y = fft(v) % 计算 $F_n v$
3: y = Lam .* y % 计算 Λy
4: y = ifft(y) % 计算 $F_n^{-1} y$

MATLAB **源代码 1.2** 循环矩阵与向量的乘积

```
Lam = fft(z);
y = ifft(Lam .* fft(v));
```

1.2.5 Toeplitz 矩阵与向量的乘积

设 $T \in \mathbb{R}^{n \times n}$ 是 n 阶 Toeplitz 矩阵 (1.1), 则 T 可以嵌入到一个 $2n$ 阶的循环矩阵

$$C(z) = \begin{bmatrix} T & C_{12} \\ C_{21} & T \end{bmatrix} \in \mathbb{R}^{2n \times 2n}$$

中. 通过选取适当的 C_{12} 和 C_{21}, 可以使得 $C(z)$ 是一个循环矩阵. 事实上, 只需令 z 取下面的向量即可

$$z = [t_0, t_1, \dots, t_{n-1}, 0, t_{-n+1}, \dots, t_{-2}, t_{-1}]^T.$$

于是, 计算 T 与任一向量 $v \in \mathbb{R}^n$ 的乘积时, 可通过循环矩阵 $C(z)$ 与向量 $\begin{bmatrix} v \\ 0 \end{bmatrix}$ 的乘积来实现, 即

$$C(z)\begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} T & * \\ * & T \end{bmatrix}\begin{bmatrix} v \\ 0 \end{bmatrix} = \begin{bmatrix} Tv \\ * \end{bmatrix}.$$

算法 1.2 计算 Toeplitz 矩阵 T 与向量 v 的乘积

1: 构造向量 $z = [t_0, t_1, \ldots, t_{n-1}, 0, t_{-n+1}, \ldots, t_{-2}, t_{-1}]^T$
2: Lam = fft(z)　% 计算循环矩阵 $C(z)$ 的特征值
3: ytmp = fft([v; zeros(n,1)])
4: ytmp = Lam .* ytmp
5: ytmp = ifft(ytmp)
6: y = ytmp(1:n)　% 取前 n 个分量即可

MATLAB **源代码 1.3** Toeplitz 矩阵与向量的乘积

```
function y = Tx(t,v)
% t : 长度为 2n-1 的列向量, 包含 T 的第一列和第一行
%    即 t=[t_0,t_1,...,t_{n-1},t_{-1},t_{-2},...,t_{-n+1}]
  n = length(v);
  z = [t(1:n); 0; t(end:-1:n+1)];
  Lam = fft(z);
  v = [v; zeros(n,1)];
  y = ifft(Lam .* fft(v));
  y = y(1:n);
```

1.2.6 BCCB 矩阵

设 C 是如下的块循环矩阵

$$C = \begin{bmatrix} C_0 & C_{n-1} & \cdots & C_1 \\ C_1 & C_0 & \cdots & C_2 \\ \vdots & \vdots & & \vdots \\ C_{n-2} & C_{n-3} & \cdots & C_{n-1} \\ C_{n-1} & C_{n-2} & \cdots & C_0 \end{bmatrix},$$

其中 $C_i \in \mathbb{R}^{m\times m}$ 都是循环矩阵. 此时我们称 C 是 BCCB (Block Circulant matrix with Circulant Blocks).

为了简单起见, 我们这里假设 $m = n$. 如果 $m \neq n$, 可以等到类似的结论.

首先, 由于 C_i 都是循环矩阵, 因此

$$C_i = \tilde{F}^* \Lambda_i \tilde{F}, \quad i = 0, 1, 2, \ldots, n-1, \tag{1.13}$$

其中 $\Lambda_i = \mathrm{diag}(\lambda_0^{(i)}, \lambda_1^{(i)}, \ldots, \lambda_{n-1}^{(i)})$. 这里我们省略了酉矩阵 $\tilde{F}_n^*$ 的下标 n. 于是

$$C = \begin{bmatrix} \tilde{F}^* & & & \\ & \tilde{F}^* & & \\ & & \ddots & \\ & & & \tilde{F}^* \end{bmatrix} \begin{bmatrix} \Lambda_0 & \Lambda_{n-1} & \cdots & \Lambda_1 \\ \Lambda_1 & \Lambda_0 & \cdots & \Lambda_2 \\ \vdots & \vdots & & \vdots \\ \Lambda_{n-2} & \Lambda_{n-3} & \cdots & \Lambda_{n-1} \\ \Lambda_{n-1} & \Lambda_{n-2} & \cdots & \Lambda_0 \end{bmatrix} \begin{bmatrix} \tilde{F} & & & \\ & \tilde{F} & & \\ & & \ddots & \\ & & & \tilde{F} \end{bmatrix}.$$

将等式右端中间的矩阵记为 Λ, 则

$$C = (I \otimes \tilde{F}^*)\Lambda(I \otimes \tilde{F}). \tag{1.14}$$

下面考虑 Λ 的对角化. 构造矩阵

$$\tilde{C}_k = \begin{bmatrix} \lambda_k^{(0)} & \lambda_k^{(n-1)} & \cdots & \lambda_k^{(1)} \\ \lambda_k^{(1)} & \lambda_k^{(0)} & \cdots & \lambda_k^{(2)} \\ \vdots & \vdots & & \vdots \\ \lambda_k^{(n-2)} & \lambda_k^{(n-3)} & \cdots & \lambda_k^{(n-1)} \\ \lambda_k^{(n-1)} & \lambda_k^{(n-2)} & \cdots & \lambda_k^{(0)} \end{bmatrix}, \quad k = 0, 1, 2, \ldots, n-1. \tag{1.15}$$

易知 $\tilde{C}_k$ 均是循环矩阵, 因此有

$$\tilde{C}_k = \tilde{F}^* \tilde{\Lambda}_k \tilde{F}, \quad k = 0, 1, 2, \ldots, n-1, \tag{1.16}$$

其中 $\tilde{\Lambda}_k = \operatorname{diag}(\tilde{\lambda}_0^{(k)}, \tilde{\lambda}_1^{(k)}, \ldots, \tilde{\lambda}_{n-1}^{(k)})$. 记

$$\tilde{F}^* = [f_0, f_1, \ldots, f_{n-1}],$$

即 f_i 为 $\tilde{F}^*$ 的第 $i+1$ 列 (或 $\tilde{F}$ 第 $i+1$ 行的转置). 由 (1.15) 和 (1.16) 可以验证

$$\Lambda(f_i \otimes e_k) = \tilde{\lambda}_i^{(k-1)}(f_i \otimes e_k), \quad i = 0, 1, 2, \ldots, n-1, k = 1, 2, \ldots, n,$$

其中 e_k 为单位矩阵的第 k 列. 所以 $\tilde{\lambda}_i^{(k)}$ 是 Λ 的特征值, 对应的特征向量为 $f_i \otimes e_k$. 于是可得到 Λ 的特征值分解

$$\Lambda = (\tilde{F}^* \otimes I)\tilde{\Lambda}(\tilde{F} \otimes I), \tag{1.17}$$

其中

$$\tilde{\Lambda} = \operatorname{diag}\left(\tilde{\lambda}_0^{(0)}, \tilde{\lambda}_0^{(1)}, \ldots, \tilde{\lambda}_0^{(n-1)}, \tilde{\lambda}_1^{(0)}, \tilde{\lambda}_1^{(1)}, \ldots, \tilde{\lambda}_1^{(n-1)}, \ldots, \tilde{\lambda}_{n-1}^{(0)}, \tilde{\lambda}_{n-1}^{(1)}, \ldots, \tilde{\lambda}_{n-1}^{(n-1)}\right).$$

最后, 将 (1.17) 代入 (1.14) 可得

$$\begin{aligned} C &= (I \otimes \tilde{F}^*)\Lambda(I \otimes \tilde{F}) \\ &= (I \otimes \tilde{F}^*)(\tilde{F}^* \otimes I)\tilde{\Lambda}(\tilde{F} \otimes I)(I \otimes \tilde{F}) \\ &= (\tilde{F}^* \otimes \tilde{F}^*)\tilde{\Lambda}(\tilde{F} \otimes \tilde{F}) \end{aligned} \tag{1.18}$$

BCCB 矩阵 C 与向量 x 的乘积

这里考虑矩阵 C 与向量乘积的具体实现过程. 设 $C \in \mathbb{R}^{n^2 \times n^2}$, $x \in \mathbb{R}^{n^2}$, 则

$$Cx = (\tilde{F}^* \otimes \tilde{F}^*)\Lambda(\tilde{F} \otimes \tilde{F})x = (F^{-1} \otimes F^{-1})\Lambda(F \otimes F)x.$$

具体操作过程如下:

(1) 计算 C 的特征值 $\tilde{\lambda}_i^{(k)}$: 首先计算 $\lambda_k^{(i)}$, 即 C_i 的特征值, 这可以通过对 C_i 的第一列做一次 FFT 求得. 然后再计算 $\tilde{\lambda}_i^{(k)}$, 即 $\tilde{C}_i$ 的特征值. 只需对 $\tilde{C}_i$ 的第一列做 FFT 即可. 在 MATLAB 中, 整个过程可以通过一次 2-D (二维) FFT 实现, 即 `fft2`.

(2) 计算 $(F \otimes F)x$: 这个可以通过两次 FFT 实现, 也可以通过一次 2-D FFT 实现.

(3) 计算 $\tilde{\Lambda}$ 与 $(F \otimes F)x$ 的乘积: 通过一次向量与向量的数组乘积即可实现.

(4) 计算 $F^{-1} \otimes F^{-1}$ 与之前计算结果的乘积: 可以通过两次 IFFT 实现, 也可以通过一次 2-D IFFT 实现.

MATLAB 源代码 1.4 BCCB 矩阵 C 与向量 x 的乘积

```
function y = BCCBx(Z,x)
% Z : n 阶矩阵, 第 1 列为 C_0 的第一列,
%     第 2 列为 C_1 的第一列, 依次类推
%
% x : 长度为 n^2 的列向量
%
```

```
n = size(Z,1);
Lam = fft2(Z); % 计算 C 的特征值
X = reshape(x,n,n); % 将 x 改写为 n 阶矩阵
Y = fft2(X); % 计算 (F\otimes F)x
Y = Lam .* Y; % 计算 \tilde\Lam 与 (F\otimes F)x 的乘积:
Y = ifft2(Y);
y = Y(:);
```

1.2.7 分块循环矩阵

设 A 是如下的块循环矩阵

$$A = \begin{bmatrix} A_0 & A_{n-1} & \cdots & A_1 \\ A_1 & A_0 & \cdots & A_2 \\ \vdots & \vdots & & \vdots \\ A_{n-2} & A_{n-3} & \cdots & A_{n-1} \\ A_{n-1} & A_{n-2} & \cdots & A_0 \end{bmatrix},$$

其中 $A_i \in \mathbb{R}^{m\times m}$. 这里不要求 A_i 是循环矩阵.

可以验证

$$\begin{aligned} A &= \sum_{k=0}^{n-1} L^k \otimes A_k \\ &= \sum_{k=0}^{n-1} \tilde{F}^* W^k \tilde{F} \otimes A_k \\ &= \sum_{k=0}^{n-1} (\tilde{F}^* \otimes I)(W^k \otimes A_k)(\tilde{F} \otimes I) \\ &= (F^{-1} \otimes I) \left(\sum_{k=0}^{n-1} W^k \otimes A_k \right) (F \otimes I), \end{aligned}$$

其中 L 是 downshift permutation (1.3), W 是对角矩阵 (1.10). 由于矩阵 $F \otimes I$ 和 $F^{-1} \otimes I$ 与向量的乘积可以通过 FFT 与 IFFT 实现, 因此只需考虑如何计算中间矩阵与向量的乘积. 直接验证可知

$$\left(\sum_{k=0}^{n-1} W^k \otimes A_k \right) = \text{blkdiag}((F \otimes I)\tilde{A}),$$

其中 blkdiag 表示块对角矩阵, $\tilde{A}$ 是块矩阵 C 的第一列 (以块为单位), 即

$$\tilde{A} = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{n-1} \end{bmatrix}.$$

下面给出所涉及的运算的 MATLAB 代码: (假定 $A \in \mathbb{R}^{n^2\times n^2}$, $x \in \mathbb{R}^{n^2}$)

- $y = (F \otimes I)x \to$ `y=fft(reshape(x,n,n).').'; y=y(:);`
  或 `y=fft(reshape(x,n,n),[],2); y=y(:)`
- $y = (F^{-1} \otimes I)x \to$ `y=ifft(reshape(x,n,n),[],2); y=y(:);`

这里 `fft(reshape(x,n,n),[],2)` 表示按行计算 FFT.

☞ 一个类似的计算是 $y = (I \otimes F)x$, 对应的 MATLAB 代码是
`y=fft(reshape(x,n,n)); y=y(:);`

1.2.8 快速三角变换

本小节内容主要参考 [8]. FFT 能用来计算下面的三角变换:

- **The Inverse Real Periodic Transform**: Given $x \in \mathbb{R}^n$ with $n = 2m$, compute $a \in \mathbb{R}^{m+1}$ and $b \in \mathbb{R}^{m-1}$ such that

$$x_k = \frac{a_0}{2} + \sum_{j=1}^{m-1} \left(a_j \cos\left(\frac{kj\pi}{m}\right) + b_j \sin\left(\frac{kj\pi}{m}\right)\right) + \frac{(-1)^k a_{n+1}}{2}, \quad k = 0, 1, \ldots, n-1.$$

- **离散 Sine 变换** (DST): Given $x \in \mathbb{R}^n$, compute $y \in \mathbb{R}^n$ such that

$$y_k = \sum_{j=1}^{n} \sin\left(\frac{kj\pi}{n+1}\right) x_j, \quad k = 0, 1, \ldots, n-1.$$

- **离散 Cosine 变换** (DCT): Given $x \in \mathbb{R}^n$, compute $y \in \mathbb{R}^n$ such that

$$y_k = \frac{x_0}{2} + \sum_{j=1}^{n-2} \cos\left(\frac{kj\pi}{n-1}\right) x_j + \frac{(-1)^k x_{n-1}}{2}, \quad k = 0, 1, \ldots, n-1.$$

下面介绍两个特殊矩阵:

- **cosine matrix** C_n:

$$[C_n]kj = \cos\left(\frac{2kj\pi}{n}\right), \quad k, j = 0, 1, \ldots, n-1.$$

- **sine matrix** S_n:

$$[S_n]kj = \sin\left(\frac{2kj\pi}{n}\right), \quad k, j = 0, 1, \ldots, n-1.$$

易知 $F_n = C_n - iS_n$. 可以看出 DST 和 DCT 都可以用上面的矩阵写出矩阵 -向量的乘积, 即: 设 $x, y \in \mathbb{R}^n$, 则有

$$y = DST(x) = S_{2(n+1)}(1:n, 1:n)x$$
$$y = DCT(x) = C_{2(n-1)}(1:n, 1:n)[\frac{x_0}{2}, x(1:n-2), x_{n-1}]^T$$

参考文献

[1] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.

[2] R. Barrett, et.al, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, 1994. (http://www.netlib.org/templates/index.html)

[3] Barry A. Cipra, The Best of the 20th Century: Editors Name Top 10 Algorithms, SIAM News, 2000.

[4] J. W. Cooley and J. W. Tukey, An algorithm for the machine calculation of complex Fourier series, *Math. Comput.*, 19 (1965), 297–301.

[5] P. Duhamel and M. Vetterli, Fast fourier transform: A tutorial review and a state of the art, *Signal Processing*, (19) 1990, 259–299.

[6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The 4th Editon, The Johns Hopkins University Press, Baltimore, MD, 2013.

[7] M. T. Heideman, D. H. Johnson and C. S. Burrus, Gauss and the history of the fast Fourier transform, *IEEE ASSP Magazine*, 1 (1984), 14–21.

[8] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.