Personal Notes on FFT-Accelerated Toeplitz CG

Ishan Patwardhan

Contents

1 **Motivation** 1

2 **Toeplitz → Circulant Embedding** 1

2.1 Fast Matvec Routine 1

3 **CG Complexity With FFT** 2

4 **Worked Example: AR(1) Factor Regression** 2

5 **Practice Problems (with Sketch Solutions)** 3

6 **Things to Explore Next** 3

1 Motivation

Large time-series covariance matrices in intraday factor models are *Toeplitz*. Each CG matrix–vector multiply normally costs $O(n^2)$, but by embedding the Toeplitz matrix into a circulant one we can use the FFT and bring that down to $O(n \log n)$. That's low-latency ammo every HFT desk loves.

2 Toeplitz → Circulant Embedding

A Toeplitz matrix $T \in \mathbb{R}^{n\times n}$ has entries $T_{ij} = t_{i-j}$. Define a $(2n) \times (2n)$ circulant matrix C via

$$C = \begin{bmatrix} T & R \\ R & T \end{bmatrix}, \quad R_{ij} = t_{n+i-j}.$$

Because C is circulant it diagonalises by the discrete Fourier matrix F: $C = F^* \operatorname{diag}(\hat{c})F$. Thus Cx can be computed as $\mathrm{IFFT}(\hat{c}\,\mathrm{FFT}(x))$ in $O(n \log n)$.

2.1 Fast Matvec Routine

Given $x \in \mathbb{R}^n$:

1. Form $x_{\text{emb}} = (x, 0, \ldots, 0)^T \in \mathbb{R}^{2n}$.
2. $y \leftarrow \mathrm{IFFT}(\hat{c}\,\mathrm{FFT}(x_{\text{emb}}))$.
3. Return first n entries.

Cost: two FFTs length $2n \Rightarrow O(n \log n)$.

3 CG Complexity With FFT

CG needs one matvec per iteration. Total cost to reach residual ε:

$$O\big(k(n \log n) + (n + \text{nonzeros}(P))\big),$$

where P is any preconditioner. For AR(1) covariances T is already diagonally dominant so plain CG + FFT is fine.

4 Worked Example: AR(1) Factor Regression

We model $y_t = \phi y_{t-1} + \varepsilon_t$ with $\phi = 0.9$, variance $\sigma^2 = 1$. The covariance of $(y_1, \ldots, y_n)$ is Toeplitz with $t_k = \sigma^2 \phi^{|k|}/(1 - \phi^2)$.

Goal. Solve $Tx = b$ for $n = 2^{12}$ using:
- (A) dense CG (baseline)
- (B) FFT-CG (this note)

Timing on my laptop (Python, NumPy FFT):

$$\text{Dense CG: } 1.46\,\text{s} \qquad \text{FFT-CG: } 0.35\,\text{s}.$$

Both reach residual 10^{-8} in 200 iterations; solutions match to 10^{-12}.

Python Listing

```
import numpy as np, time
phi, sigma2, n = 0.9, 1.0, 4096                 # AR(1) params, grid
col = sigma2 * phi ** np.arange(n) / (1-phi**2)  # first column of T
m = 2*n
circ_col = np.r_[col, 0, col[:0:-1]]
fft_c = np.fft.rfft(circ_col)                    # precompute FFT of column

def fft_toeplitz_mv(x):                          # O(n log n) matvec
    xp = np.zeros(m); xp[:n] = x
    y = np.fft.irfft(fft_c * np.fft.rfft(xp))
    return y[:n]

T = col[np.abs(np.subtract.outer(np.arange(n), np.arange(n)))]  # dense T
b = np.random.default_rng(0).standard_normal(n)

def cg(matvec, b, tol=1e-8):
    x = np.zeros_like(b)
    r, p = b - matvec(x), b - matvec(x)
    rs = r @ r
    for k in range(200):
        Ap = matvec(p)
        alpha = rs / (p @ Ap)
        x += alpha * p
        r -= alpha * Ap
        rs_new = r @ r
        if rs_new**0.5 < tol: return x, k+1
        p = r + (rs_new/rs) * p
        rs = rs_new
    return x, 200
```

```
start = time.perf_counter(); x_dense,_ = cg(lambda v: T@v, b); t_dense = time.perf_counter()-sta
start = time.perf_counter(); x_fft,_   = cg(fft_toeplitz_mv, b); t_fft   = time.perf_counter()-s
print(f"Dense CG {t_dense:.3f}s  vs  FFT CG {t_fft:.3f}s  | diff={np.linalg.norm(x_dense-x_fft):
```

Run-time output:

```
Dense CG 1.462s  vs  FFT CG 0.354s  | diff=1.3e-12
```

5 Practice Problems (with Sketch Solutions)

P1: *Bandwidth-b Toeplitz Preconditioner.* Show that truncating to b off-diagonals yields M s.t. $\kappa(M^{-1}T) = O(\log n)$.

P2: *Block-Toeplitz Case.* Extend the embedding to block-Toeplitz-with-Toeplitz-blocks (BTTB) and outline a $2D$ FFT routine.

P3: *GPU FFT Benchmark.* Use CuPy to compare CPU vs GPU matvec throughput for $n = 2^{18}$. Report speed-ups.

P4: *Toeplitz Least-Squares.* Combine FFT-CG with normal-equation conditioning to fit a rolling AR(p) in real time.

6 Things to Explore Next

- Try iterative refinement with double vs single precision FFT.
- Investigate hierarchical-matrix ($\mathcal{H}$) preconditioners.
- Implement real-time covariance updates using Woodbury + FFT.