Introduction to State Space Models

Vinay[*] **Subhadeep Sing**[*] **Vaibhav Mahore**[*] **Snehal Biswas**[*]
`{vinay2023, subh, vaibhav, snehal}@iisc.ac.in`

Abstract

This paper introduces **State Space Models** (SSMs) as a promising approach for sequential data modeling. We go through the fundamentals of SSMs, including their mathematical framework for describing dynamical systems through hidden states and observations. This paper details the **Linear State Space Layer** (LSSL) as a specific implementation. A core concept presented is the **High-Order Polynomial Projection Operators** (HiPPO) framework, which efficiently compresses historical sequence data into a fixed-size representation using polynomial basis functions. This paper then explores the **Structured State Space for Sequence Modeling** (S4) model, highlighting its efficient parameterization and computation techniques, such as **Normal Plus Low-Rank** (NPLR) decomposition to handle the HiPPO matrix diagonalization challenges. Finally, it introduces the **MAMBA** architecture, which enhances SSMs with a **selection** mechanism, making parameters input-dependent to selectively compress context and overcome the limitations of time-invariant models, thereby improving performance on tasks requiring content-aware reasoning

1 Introduction

Sequence modeling aims to learn patterns or dependencies from data over time, mapping input $x(t)$ to output $y(t)$. Common approaches include RNNs, CNNs, and Transformers, each with unique trade-offs.

Ideally, we seek models that:

- Support parallel training (like Transformers) with linear $O(N)$ complexity.
- Enable constant-time inference per token (like RNNs).

State Space Models (SSMs) emerge as a promising direction toward these goals.

2 Methodology

2.1 SSMs

A State Space Model (SSM) describes a dynamical system with a hidden state that evolves over time and an output dependent on that state, widely used in control theory and now in deep learning for modeling long-range dependencies in sequences.

2.2 Linear State Spcae Layer(LSSL)

LSSL is a **specific** way to implement SSMs. It is a simple sequence model that maps a 1-dimensional function or sequence $u(t) \to y(t)$ through a implicit state $x(t)$ by simulating a linear continuous time state space representation.

$$\dot{x}(t) = Ax(t) + Bu(t), \quad y(t) = Cx(t) + Du(t)$$

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

2.3 Discretization of Continuous-Time Models

In practice, because data is discrete (with $u(t - \Delta t) \approx u(t)$), we use the Generalized Bilinear Transform (GBT) to obtain the discrete state-space model:

$$x_t = \bar{A}\, x_{t-1} + \bar{B}\, u_t, \quad y_t = Cx_t + Du_t,$$

with

$$\bar{A} = (I - \Delta t\, \alpha A)^{-1}[I + \Delta t\, (1 - \alpha)A], \quad \bar{B} = (I - \Delta t\, \alpha A)^{-1}\Delta t\, B.$$

Complete calculation and discussion of α is provided in Appendix [A1].

2.4 LSSL: Recurrent and Convolutional Views

Recurrence. Let $x_{t-1} \in \mathbb{R}^{H\times N}$ represent the state. The update x_t and output y_t follow recurrence equations, making LSSL a recurrent model with constant per-step cost—suitable for autoregressive inference.

Convolution. The convolution kernel is defined as:

$$K_L(\bar{A}, \bar{B}, \bar{C}) = \left[\bar{C}\bar{B}, \quad \bar{C}\bar{A}\bar{B}, \quad \ldots, \quad \bar{C}\bar{A}^{L-1}\bar{B}\right] \in \mathbb{R}^{H\times L},$$

(see full derivation in Appendix [A1]).

In this *parallel* mode, the full input $u = \{u_1, \ldots, u_L\}$ is processed at once. Diagonalizing $\bar{A}$ enables efficient FFT-based computation in $\mathcal{O}(L \log L)$, ideal for fast, parallel training.

2.5 High-Order Polynomial Projection Operators(HIPPO)

2.5.1 Motivation

Storing the full history of a continuous input stream f_t is impractical. The HiPPO framework addresses this by defining fixed matrices A (used in models like LSSL) that compress the past into a low-dimensional memory. This enables sequence modeling without retaining all previous data.

The core idea is *function approximation using a probability measure*. A measure μ on $[0, \infty)$ defines an inner product $\langle f, g\rangle_\mu = \int f(x)g(x)\, d\mu(x)$, turning function space into a Hilbert space with norm $\|f\|_{L^2(\mu)} = \sqrt{\langle f, f\rangle_\mu}$. In this space, f_t is projected onto polynomial basis functions.

2.5.2 HiPPO Abstraction and Memory Update

The HiPPO framework enables online function approximation by projecting the input history f_t onto a polynomial subspace using a time-varying measure μ_t. It defines:

- A *projection operator* proj_t, which maps the restricted function $f_t(x) = f(x)$ on $[0, t]$ to a polynomial $g(t) \in \mathcal{G}$, minimizing the approximation error $\|f_t - g(t)\|_{L^2(\mu_t)}$.
- A *coefficient operator* coef_t, extracting a vector $c(t) \in \mathbb{R}^N$ representing $g(t)$ in an orthogonal polynomial basis.

The memory state $c(t)$ evolves as $\frac{d}{dt}c(t) = A_t c(t) + B_t f(t)$, discretized as $c_{k+1} = A_k c_k + B_k f_k$, enabling efficient online compression of the input history.

We primarily use three HiPPO variants: **HiPPO-LegT, HiPPO-LagT, and HiPPO-LegS**. A1

2.6 Structured State Space for Sequence Modeling (S4)

The **S4** model transforms a continuous SSM into a stable discrete recurrent or convolutional form using **NPLR** diagonalization and Cauchy-based kernel evaluation.

2.6.1 Diagonalization of HiPPO Matrix A

To overcome the bottleneck of repeated multiplications by $\bar{A}$ in the discrete-time SSM, we diagonalize A. This step is justified by the following:

Lemma 2.1. *Conjugation is an equivalence relation on SSMs:*

$$(A, B, C) \sim (V^{-1}AV,\ V^{-1}B,\ CV) \quad \textit{for any invertible matrix } V.$$

Choosing V such that $V^{-1}AV = \Lambda$ (diagonal) yields:

$$A^k = V\,\Lambda^k V^{-1}, \quad K = CA^kB = CV\Lambda^kV^{-1}B.$$

Since Λ^k is trivial to compute, the sequence $\{\Lambda^k\}_{k=0}^{L-1}$ can be represented by a Vandermonde matrix. Fast polynomial transforms allow its multiplication in $O((N+L)\log(N+L))$ time. However, because V contains the eigenvectors of A, it is highly ill-conditioned.

Lemma 2.2. *The HiPPO matrix A is diagonalized by a matrix V with entries $V_{ij} = \binom{i+j}{i-j}$, which, for example, implies that $V_{3i,i} = \binom{4i}{2i} \approx 2^{4i}$. Hence, V can have entries up to roughly $2^{4N/3}$.*

2.6.2 Normal Plus Low-Rank Decomposition

To address the numerical instability of V, we conjugate using unitary (hence well-conditioned) matrices. Although the HiPPO matrix is not normal, it can be decomposed as the sum of a normal and a low-rank matrix:

$$A = V\Lambda V^* - PQ^\top = V\Big(\Lambda - (V^*P)(V^*Q)^*\Big)V^*,$$

where $V \in \mathbb{C}^{N\times N}$ is unitary, Λ is diagonal, and $P, Q \in \mathbb{R}^{N\times r}$ with $r = 1$ or 2 (with equation (2) being $r = 1$). Although exponentiating this sum is more challenging than diagonal matrices, the S4 paper presents three efficient techniques to handle it.

S4 performs convolution efficiently by using the Fourier transform instead of computing it directly in the time domain. The steps are:

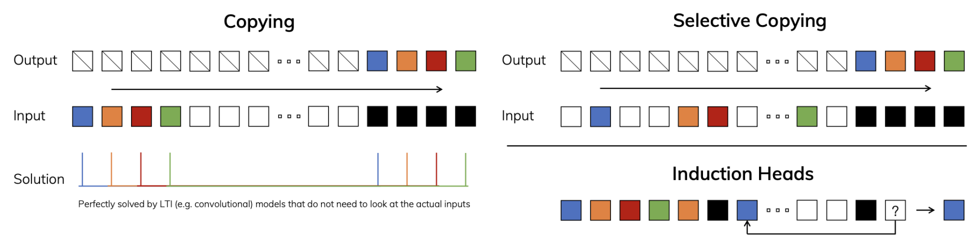
1. **Choose a length L:** Truncate the kernel to length L, since later terms often have minimal impact.
2. **Transform to frequency domain and multiply:** Use FFT to convert both the input and kernel to the frequency domain, where convolution becomes fast elementwise multiplication, avoiding costly matrix operations.
3. **Inverse FFT:** Apply IFFT to recover the final output in the time domain.

This entire process only takes $\mathcal{O}(L\log L)$ operations using FFT/IFFT, compared to the naive $\mathcal{O}(N^2L)$ time required for direct convolution involving matrix multiplications.For more detail Appendix A

2.7 Motivation for MAMBA: Selection as a Means of Compression

The paper argue that a fundamental problem of sequence modeling is *compressing context into a smaller state*. The paper propose **selectivity** as the fundamental principle for building sequence models. The selection mechanism is motivated using intuition from two synthetic task.

- **Selective Copying**: The Selective Copying task modifies the popular Copying task (Arjovsky, Shah, and Bengio 2016) by varying the position of the tokens to memorize. It requires *content-aware* reasoning to be able to memorize the relevant tokens (colored) and filter out the irrelevant ones (white).
- **Induction Head**: The Induction Heads task is a well-known mechanism hypothesized to explain the majority of in-context learning abilities of LLMs (Olsson et al. 2022). It requires *context-aware* reasoning to know when to produce the correct output in the appropriate context (black).


Copying
Output
Input
Solution
Perfectly solved by LTI (e.g. convolutional) models that do not need to look at the actual inputs
Selective Copying
Output
Input
Induction Heads
?

Algorithm 1 SSM (S4)

```
Input: $x : (B, L, D)$
Output: $y : (B, L, D)$
1: $\mathbf{A} : (D, N)$ ← Parameter
   ▷ Represents structured $N \times N$ matrix
2: $\mathbf{B} : (D, N)$ ← Parameter
3: $\mathbf{C} : (D, N)$ ← Parameter
4: $\boldsymbol{\Delta}$ : (D) ← $\tau_\Delta$(Parameter)
5: $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}$ : (D,N) ← discretize($\boldsymbol{\Delta}, \boldsymbol{A}, \boldsymbol{B}$)
6: y ← SSM($\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \boldsymbol{C}$)(x)
7: ▷ Time-invariant: recurrence or convolution
8: return $y$
```

Algorithm 2 SSM + Selection (S6)

```
Input: $x : (B, L, D)$
Output: $y : (B, L, D)$
1: $\mathbf{A} : (D, N)$ ← Parameter
   ▷ Represents structured $N \times N$ matrix
2: $\mathbf{B} : (B, L, N) \leftarrow s_B(x)$ ← Parameter
3: $\mathbf{C} : (B, L, N) \leftarrow s_C(x)$ ← Parameter
4: $\boldsymbol{\Delta}$ : $B, L, D$ ← $\tau_\Delta$(Parameter $+ s_\Delta(x)$)
5: $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}$ : (B,L,D,N) ← discretize($\boldsymbol{\Delta}, \boldsymbol{A}, \boldsymbol{B}$)
6: y ← SSM($\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}, \boldsymbol{C}$)(x)
7:     ▷ Time Varying: recurrence (scan) only
8: return $y$
```

The main difference between **S4** and **MAMBA** is simply making several parameters Δ , $\boldsymbol{B}, \boldsymbol{C}$ functions of the input, along with the associated changes to tensor shapes throughout.

2.8 Selection Mechanism

- $\boldsymbol{s_B(x_t)} = \text{Linear}_N(x_t)$ computes token-specific B_t from input x_t via a linear layer.
- $\boldsymbol{s_C(x_t)} = \text{Linear}_N(x_t)$ computes token-specific output map C_t.
- τ_Δ (Parameter $+ s_\Delta(x_t)$), where $s_\Delta(x_t) = \text{Broadcast}_D(\text{Linear}_1(x_t))$ is a learnable projection, and τ_Δ (softplus) ensures a positive step size.
- The paper states that although $\mathbf{A}$ can be selective, it only influences the model via its interaction with Δ in $\bar{\mathbf{A}} = \exp(\Delta\mathbf{A})$, so selectivity in Δ is sufficient to ensure selectivity in $\bar{\mathbf{A}}$, driving the main improvement.

The MAMBA **selection** algorithm can be nicely visualized with the picture given below:

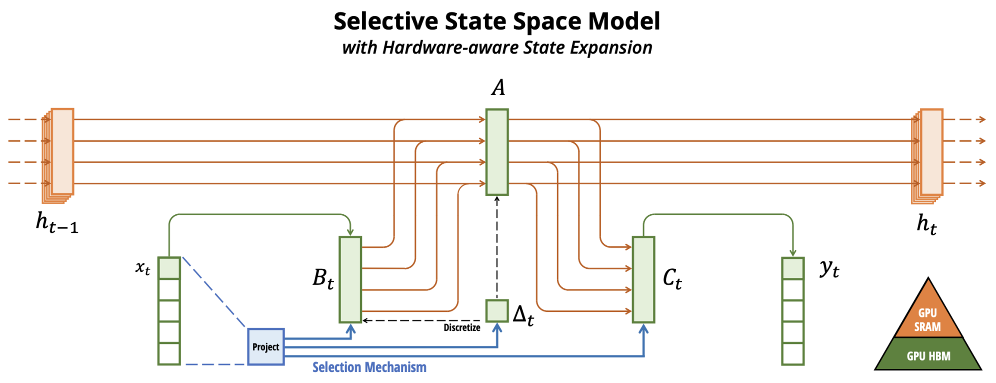


Figure 1: (**Overview**.) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the (Δ, $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$) parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

A discussion on the selection mechanism's interpretation, gating connection, and its effects (variable spacing, context filtering, boundary resetting) is provided in Appendix [3.5].

2.9 Overcoming Convolution Limitations in MAMBA

Since MAMBA cannot be evaluated using a convolution(because it's time varying), it cannot be parallelized. Our only way to calculate it to use the recurrent formulation. The paper address this problem with three techniques: kernel fusion, parallel scan and recomputation. These techniques will **not** be discussed in this project paper.

References

[1] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[2] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent Memory with Optimal Polynomial Projections. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2023.

[3] Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[4] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

]

3 Appendix

3.1 Derivation of descritization

The *Generalized Bilinear Transform (GBT)* approximates the derivative and right-hand side using a weighted average.

$$\dot{x}(t) \approx \frac{x(t) - x(t - \Delta t)}{\Delta t}$$

Assuming $u(t) \approx u(t - \Delta t)$ and letting $\alpha \in [0, 1]$, we write:

$$\frac{x(t) - x(t - \Delta t)}{\Delta t} = (1 - \alpha)[Ax(t - \Delta t) + Bu(t)] + \alpha[Ax(t) + Bu(t)]$$

Defining

$$x_{t-1} = x(t - \Delta t), \quad x_t = x(t), \quad u_t = u(t),$$

we have:

$$\frac{x_t - x_{t-1}}{\Delta t} = (1 - \alpha)[Ax_{t-1} + Bu_t] + \alpha[Ax_t + Bu_t]$$

Multiplying both sides by Δt:

$$x_t - x_{t-1} = \Delta t \left[(1 - \alpha)(Ax_{t-1} + Bu_t) + \alpha(Ax_t + Bu_t)\right]$$

Rearrange the terms to isolate x_t:

$$x_t - \Delta t \alpha A x_t = x_{t-1} + \Delta t(1 - \alpha)Ax_{t-1} + \Delta t B u_t$$

Factorizing, we get:

$$(I - \Delta t \alpha A)x_t = \big[I + \Delta t(1 - \alpha)A\big]x_{t-1} + \Delta t B u_t$$

Finally, solving for x_t:

$$x_t = (I - \Delta t \alpha A)^{-1} \left[\big(I + \Delta t(1 - \alpha)A\big)x_{t-1} + \Delta t B u_t\right]$$

3.1.1 Role of α

- $\alpha = 0$ (Forward Euler):

  $$x_t = \Big(I + \Delta t\, A\Big)x_{t-1} + \Delta t\, B\, u_t.$$

  Simple but may be less stable.

- $\alpha = 1$ (Backward Euler):
$$x_t = \Big(I - \Delta t\, A\Big)^{-1} x_{t-1} + \Big(I - \Delta t\, A\Big)^{-1} \Delta t\, B\, u_t.$$
Implicit formulation, which enhances stability.
- $\alpha = 0.5$ (Trapezoidal):
$$x_t = \left(I - \frac{\Delta t}{2} A\right)^{-1} \left[I + \frac{\Delta t}{2} A\right] x_{t-1} + \left(I - \frac{\Delta t}{2} A\right)^{-1} \Delta t\, B\, u_t.$$
This formulation balances stability and accuracy.

3.2 Convolution in LSSL

Convolution. Assume $x_{-1} = 0$. Unrolling the recurrences yields
$$y_k = \bar{C}\, \bar{A}^k\, \bar{B}\, u_0 + \bar{C}\, \bar{A}^{k-1}\, \bar{B}\, u_1 + \cdots + \bar{C}\, \bar{A}\, \bar{B}\, u_{k-1} + \bar{B}\, u_k + \bar{D}\, u_k,$$
so the output can be written as a convolution:
$$y = K_L(\bar{A}, \bar{B}, \bar{C}) * u + \bar{D}\, u,$$
with convolution kernel defined by
$$K_L(\bar{A}, \bar{B}, \bar{C}) = \begin{bmatrix}\bar{C}\, \bar{B}, & \bar{C}\, \bar{A}\, \bar{B}, & \dots, & \bar{C}\, \bar{A}^{L-1}\, \bar{B}\end{bmatrix} \in \mathbb{R}^{H \times L}.$$

3.3 Three HiPPO Operators: LegT, LagT, and LegS

3.3.1 Types of HiPPO Operators

Different choices of the measure μ_t lead to distinct operators. Here we describe three variants: LegT, LagT, and LegS. (Note: θ below is assumed to be a defined parameter).

3.3.2 LegT

For a uniform (or strict) window measure
$$\mu_t(x) = \frac{1}{\theta} \mathbf{1}_{[t-\theta, t]}(x)$$
the LegT operator projects the history. Its continuous-time dynamics are given by
$$\frac{d}{dt} c(t) = A\, c(t) + B\, f(t),$$
with coefficients defined as
$$A_{nk} = \frac{1}{\theta} \begin{cases} (-1)^{n-k}(2n+1), & \text{if } n \geq k, \\ (2n+1), & \text{if } n < k, \end{cases} \qquad B_n = \frac{1}{\theta}(2n+1)(-1)^n.$$
Discretizing (using, e.g., the bilinear transform or ZOH) yields the update:
$$c_{k+1} = A\, c_k + B\, f_k.$$

3.3.3 LagT

For an exponentially decaying window measure defined by
$$\mu^{(t)}(x) = e^{-(t-x)}\, \mathbb{I}_{(-\infty, t]}(x) \begin{cases} e^{x-t}, & \text{if } x \leq t, \\ 0, & \text{if } x > t, \end{cases}$$
the LagT operator emphasizes recent inputs more strongly. Its continuous formulation is
$$\frac{d}{dt} c(t) = A\, c(t) + B\, f(t),$$
where
$$A_{nk} = \begin{cases} 1, & \text{if } n \geq k, \\ 0, & \text{if } n < k, \end{cases} \qquad B_n = 1.$$
The corresponding discrete update is:
$$c_{k+1} = A\, c_k + B\, f_k.$$

3.3.4 LegS

The LegS operator. Its dynamics are given by:

$$\frac{d}{dt}c(t) = -\frac{1}{t}A\,c(t) + \frac{1}{t}B\,f(t),$$

with the discrete version given by

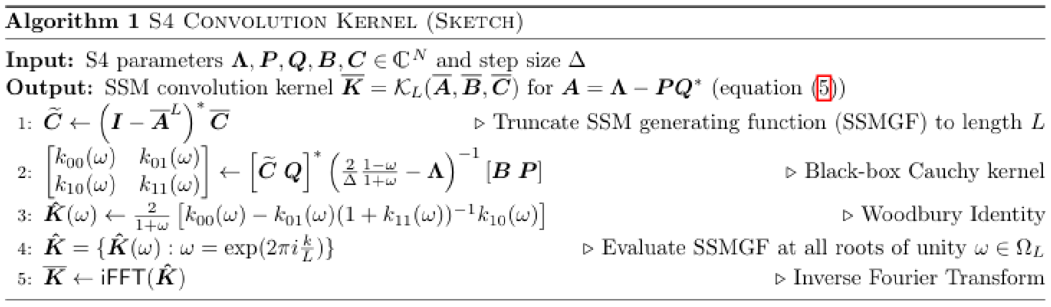
$$c_{k+1} = \left(\mathrm{Id} - \frac{1}{k}A\right)c_k + \frac{1}{k}B\,f_k$$

The entries of A and B are specified as:

$$A_{nk} = \begin{cases} \sqrt{(2n+1)(2k+1)}, & \text{if } n > k, \\ n+1, & \text{if } n = k, \\ 0, & \text{if } n < k, \end{cases} \qquad B_n = \sqrt{2n+1}.$$

Theorem 3.1 (Properties of HiPPO-LegS)**.** The HiPPO-LegS operator satisfies:

3.4 How s4 compute kernal



```
Algorithm 1 S4 CONVOLUTION KERNEL (SKETCH)
Input: S4 parameters $\boldsymbol{\Lambda}, \boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{C}^N$ and step size $\Delta$
Output: SSM convolution kernel $\overline{\boldsymbol{K}} = \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}})$ for $\boldsymbol{A} = \boldsymbol{\Lambda} - \boldsymbol{P}\boldsymbol{Q}^*$ (equation (5))
1: $\widetilde{\boldsymbol{C}} \leftarrow \left(\boldsymbol{I} - \overline{\boldsymbol{A}}^L\right)^* \overline{\boldsymbol{C}}$    ▷ Truncate SSM generating function (SSMGF) to length $L$
2: $\begin{bmatrix} k_{00}(\omega) & k_{01}(\omega) \\ k_{10}(\omega) & k_{11}(\omega) \end{bmatrix} \leftarrow \left[\widetilde{\boldsymbol{C}}\ \boldsymbol{Q}\right]^* \left(\frac{2}{\Delta}\frac{1-\omega}{1+\omega} - \boldsymbol{\Lambda}\right)^{-1} [\boldsymbol{B}\ \boldsymbol{P}]$    ▷ Black-box Cauchy kernel
3: $\hat{\boldsymbol{K}}(\omega) \leftarrow \frac{2}{1+\omega}\left[k_{00}(\omega) - k_{01}(\omega)(1 + k_{11}(\omega))^{-1}k_{10}(\omega)\right]$    ▷ Woodbury Identity
4: $\hat{\boldsymbol{K}} = \{\hat{\boldsymbol{K}}(\omega) : \omega = \exp(2\pi i \frac{k}{L})\}$    ▷ Evaluate SSMGF at all roots of unity $\omega \in \Omega_L$
5: $\overline{\boldsymbol{K}} \leftarrow \mathsf{iFFT}(\hat{\boldsymbol{K}})$    ▷ Inverse Fourier Transform
```

- Instead of computing $\overline{K}$ directly, we compute its spectrum by evaluating its **truncated generating function** $\sum_{j=0}^{L-1} K_j \zeta^j$ at the roots of unity ζ. $\overline{K}$ can then be found by applying an inverse FFT.
- This generating function is closely related to the matrix resolvent, and now involves a matrix inverse instead of power. The low-rank term can now be corrected by applying the **Woodbury identity** which reduces $(A + PQ^*)^{-1}$ in terms of A^{-1}, truly reducing to the diagonal case.
- Finally, we show that the diagonal matrix case is equivalent to the computation of a **Cauchy kernel** $\frac{1}{\omega_j - \zeta_k}$, a well-studied problem with stable near-linear algorithms (**? ?**).

Our techniques apply to any matrix that can be decomposed as **Normal Plus Low-Rank (NPLR)**.

1. **Timescale Robustness:** It is equivariant to time dilation; rescaling the input function f does not affect the approximation coefficients.
2. **Computational Efficiency:** Thanks to the triangular structure of A, each discrete update is computable in $O(N)$ operations.
3. **Stable Gradient Propagation:** The $\frac{1}{t}$ (or $\frac{1}{k}$) scaling ensures gradients remain stable, mitigating vanishing gradients.

3.5 Interpretation of Selection Mechanism

- **Connection to Gating Mechanism**: When we reduce the model to 1-Dimensional case i.e. $N = 1$, $A = -1$, $B = 1$, $s_\Delta = \mathrm{Linear}(x)$, and $\tau_\Delta = \mathrm{softplus}$, then the selective SSM recurrence(Algorithm 2) takes the form

$$g_t = \sigma(\text{Linear}(x_t)) \qquad (gate)$$
$$h_t = (1 - g_t)h_{t-1} + g_t x_t \qquad (update)$$

- **Variable Spacing**: Selectively filter out irrelevant or noisy tokens, such as language fillers, to focus on meaningful inputs.
- **Filtering Context**: Selectivity allows us to dynamically reset the state and discard irrelevant historical information, allowing performance to improve consistently with increasing context length, unlike many traditional sequence models that struggle to ignore extraneous context.
- **Boundary Resetting**: In scenarios where multiple independent sequences are stitched togethere, Linear Time-invariant(LTI) models will bleed information between the sequences. Selective SSM's can reset their state at boundaries (e.g $\Delta_t \to \infty$, or in gating when $g_t \to 1$).