

# **Tratado de Implementación Numérica y Algorítmica de Predictores Estocásticos Universales**

Consortio de Desarrollo de Meta-Predicción Adaptativa

18 de febrero de 2026

# Índice

<b>1</b>	<b>Fundamentos de Discretización y Simulaciones de Monte Carlo</b>	<b>2</b>
1.1	Generación de Números Pseudo-Aleatorios . . . . .	2
1.1.1	Algoritmo de Chambers-Mallows-Stuck (CMS) . . . . .	2
1.2	Esquemas de Integración Estocástica . . . . .	2
1.2.1	Esquema de Euler-Maruyama . . . . .	2
1.2.2	Esquema de Milstein . . . . .	3
1.3	Simulación de Procesos de Salto (Rama C) . . . . .	3
<b>2</b>	<b>Implementación del Motor de Identificación (SIA)</b>	<b>4</b>
2.1	Estimación Multifractal (WTMM) . . . . .	4
2.2	Detección de Cambios de Régimen (Test CUSUM) . . . . .	4
2.3	Cálculo de Sensibilidades (Malliavin/AAD) . . . . .	4
2.3.1	Procesos Tangenciales y Bismut-Elworthy-Li . . . . .	5
2.3.2	Algoritmo Delta-Malliavin por Monte Carlo . . . . .	5
<b>3</b>	<b>Solvers Numéricos para Núcleos de Predicción</b>	<b>6</b>
3.1	Rama A: Proyección de Hilbert y Filtrado de Wiener . . . . .	6
3.1.1	Algoritmo Recursivo de Levinson-Durbin . . . . .	6
3.2	Rama B: Ecuación HJB y Métodos de Viscosidad . . . . .	6
3.2.1	Esquemas de Diferencias Finitas Monótonas . . . . .	6
3.2.2	Deep Galerkin Method (DGM) . . . . .	7
3.3	Rama C: Ecuación Integro-Diferencial de Saltos . . . . .	7
3.3.1	Algoritmo Delta-Malliavin en Espacios de Poisson . . . . .	7
3.3.2	Esquema IMEX (Implícito-Explícito) para PIDEs . . . . .	7
3.4	Rama D: Computación de Signatures . . . . .	8
3.4.1	Identidad de Chen y Truncamiento . . . . .	8
3.4.2	Log-Signatures . . . . .	8
<b>4</b>	<b>Orquestador: Transporte Óptimo Regularizado</b>	<b>9</b>
4.1	Circuit Breaker de Robustez (Pre-Orquestador) . . . . .	9
4.2	Algoritmo de Sinkhorn-Knopp (Espacio Dual) . . . . .	9
4.3	Esquema JKO Proximal . . . . .	10
<b>5</b>	<b>Arquitectura de Software y Paralelismo</b>	<b>11</b>
5.1	Patrones de Construcción Orientados a Objetos . . . . .	11
5.1.1	Estructura de Clases Sugerida . . . . .	11
5.2	Computación Heterogénea y Aceleración . . . . .	11
5.2.1	GPU (CUDA/OpenCL) . . . . .	11
5.2.2	FPGA (Field-Programmable Gate Array) . . . . .	11

<b>6</b>	<b>Consideraciones de Estabilidad Numérica</b>	<b>12</b>
6.1	Condición CFL (Courant-Friedrichs-Lewy) . . . . .	12
6.2	Estabilidad de la Signatura Logarítmica . . . . .	12
<b>7</b>	<b>Gobernanza de Metaparámetros Heurísticos</b>	<b>13</b>
7.1	Taxonomía y Acotación Analítica (Safe Harbors) . . . . .	13
7.1.1	Parámetros de Discretización y Truncamiento . . . . .	13
7.1.2	Parámetros de Regularización y Estabilidad . . . . .	13
7.1.3	Umbrales de Decisión (Fronteras Duras) . . . . .	14
7.2	Validación Cruzada Causal (Walk-Forward Validation) . . . . .	14
7.3	Meta-Optimización Libre de Derivadas (Optimización Bayesiana) . . . . .	15

# Capítulo 1

## Fundamentos de Discretización y Simulaciones de Monte Carlo

### 1.1 Generación de Números Pseudo-Aleatorios

La base del integrador estocástico es la fuente de entropía  $\xi \sim \mathcal{D}$ .

- **Gaussianos:** Para el movimiento Browniano  $dW_t \approx \sqrt{\Delta t}Z$ , con  $Z \sim \mathcal{N}(0, 1)$ . Se recomienda el generador *Mersenne Twister* o *PCG64* para periodos largos.
- **Levy/Saltos:** Utilizar el método de Chambers-Mallows-Stuck para simular variables estables  $S(\alpha, \beta, \gamma, \delta)$ .

#### 1.1.1 Algoritmo de Chambers-Mallows-Stuck (CMS)

Para generar una variable aleatoria  $\alpha$ -estable estándar  $S(\alpha, \beta = 0, \gamma = 1, \delta = 0)$  con  $\alpha \neq 1$ :

1. Generar  $U \sim \text{Uniforme}(-\pi/2, \pi/2)$  y  $W \sim \text{Exponencial}(1)$ .
2. Computar:

$$X = \frac{\sin(\alpha U)}{(\cos U)^{1/\alpha}} \cdot \left[ \frac{\cos((1 - \alpha)U)}{W} \right]^{(1-\alpha)/\alpha}$$

3. Retornar  $X$ . Para el caso general  $Y \sim S(\alpha, \beta, \gamma, \delta)$ , aplicar la transformación afín correspondiente.

### 1.2 Esquemas de Integración Estocástica

#### 1.2.1 Esquema de Euler-Maruyama

Para la EDO estocástica  $dX_t = b(X_t)dt + \sigma(X_t)dW_t$ , la discretización de primer orden es:

---

**Algorithm 1** Integrador Euler-Maruyama

---

- 1: **Input:**  $X_0, T, N, b(\cdot), \sigma(\cdot)$
  - 2:  $\Delta t \leftarrow T/N$
  - 3:  $X \leftarrow$  array de longitud  $N + 1$
  - 4: **for**  $k \leftarrow 0$  **to**  $N - 1$  **do**
  - 5:      $Z \sim \mathcal{N}(0, 1)$
  - 6:      $X_{k+1} \leftarrow X_k + b(X_k)\Delta t + \sigma(X_k)\sqrt{\Delta t}Z$
  - 7: **end for**
  - 8: **Return**  $X$
-

### 1.2.2 Esquema de Milstein

Mejora la convergencia fuerte a orden 1.0. Requiere la derivada de la volatilidad  $\sigma'(x)$ .

$$\hat{X}_{k+1} = \hat{X}_k + b_k \Delta t + \sigma_k \Delta W_k + \frac{1}{2} \sigma_k \sigma'_k ((\Delta W_k)^2 - \Delta t)$$

**Nota:** Si  $\sigma(x)$  es constante (volatilidad aditiva), Milstein se reduce a Euler-Maruyama.

### 1.3 Simulación de Procesos de Salto (Rama C)

Para  $dX_t = b(X_t)dt + \sigma(X_t)dW_t + dJ_t$ , donde  $J_t$  es un proceso de Poisson Compuesto con intensidad  $\lambda$  y tamaño de salto  $Y \sim F_Y$ :

1. Simular el número de saltos en  $[t, t + \Delta t]$ :  $N_{\text{jump}} \sim \text{Poisson}(\lambda \Delta t)$ .
2. Si  $N_{\text{jump}} > 0$ , generar tamaños  $Y_1, \dots, Y_{N_{\text{jump}}}$ .
3. Actualizar:  $X_{k+1} = X_{k+1}^{\text{diff}} + \sum Y_i$ .

## Capítulo 2

# Implementación del Motor de Identificación (SIA)

### 2.1 Estimación Multifractal (WTMM)

El algoritmo WTMM (Wavelet Transform Modulus Maxima) permite extraer el espectro de singularidades  $D(h)$  en tiempo (cuasi) real.

---

**Algorithm 2** WTMM Discreto Detallado - Con Trazado de Máximos

---

- 1: **Input:** Serie temporal  $X$ , escalas  $a_i \in \{2^0, 2^{0.1}, \dots, 2^J\}$  (escalas dyádicas densas).
  - 2: **Paso 1: CWT (FFT) y Máximos Locales**
  - 3: Para cada escala  $a_j$ , extraer conjunto de máximos  $M_j = \{(b, |W_{a_j}(b)|)\}$ .
  - 4: **Paso 2: Enlace de Máximos (Tracking)**
  - 5: Inicializar líneas  $\mathcal{L} = \{(b, |W_{a_J}(b)|)\}_{b \in M_J}$  (desde escala gruesa).
  - 6: **for**  $j \leftarrow J - 1$  **downto** 1 **do**
  - 7:     **for** cada línea  $L \in \mathcal{L}$  con último punto  $(b_{\text{last}}, \text{mod})$  **do**
  - 8:         Buscar  $(b_{\text{curr}}, \text{mod}_{\text{curr}}) \in M_j$  tal que  $|b_{\text{curr}} - b_{\text{last}}| < C \cdot a_j$  (Cono de influencia).
  - 9:         Si hay múltiples candidatos, elegir el de mayor módulo.
  - 10:        Extender  $L \leftarrow L \cup \{(b_{\text{curr}}, \text{mod}_{\text{curr}})\}$ .
  - 11:     **end for**
  - 12: **end for**
  - 13: **Paso 3: Función de Partición** Para momentos  $q \in [-5, 5]$ :
  - 14:  $Z(q, a) = \sum_{L \in \mathcal{L}} (\sup_{(b, \text{mod}) \in L \cap \text{scale}(a)} \text{mod})^q$
  - 15: **Paso 4: Exponentes**
  - 16:  $\tau(q) \leftarrow$  pendiente de la regresión lineal  $\log Z(q, a)$  vs  $\log a$ .
  - 17: **Output:** Espectro de Legendre  $D(h) = \min_q(qh - \tau(q))$ .
- 

### 2.2 Detección de Cambios de Régimen (Test CUSUM)

El evento `RegimeChangedEvent` se dispara cuando la estadística de Page de la suma acumulada de residuos excede un umbral adaptativo.

### 2.3 Cálculo de Sensibilidades (Malliavin/AAD)

En lugar de perturbar entradas (Diferencias Finitas), computamos la derivada exacta del grafo computacional.

---

**Algorithm 3** Algoritmo CUSUM Discreto

---

```
1: Input: Residuos estandarizados  $e_t$ , umbral  $h$ .
2:  $S_0 \leftarrow 0$ ,  $G_0^+ \leftarrow 0$ ,  $G_0^- \leftarrow 0$ 
3: for  $t \leftarrow 1$  to  $N$  do
4:    $G_t^+ \leftarrow \max(0, G_{t-1}^+ + e_t - k)$ 
5:    $G_t^- \leftarrow \max(0, G_{t-1}^- - e_t - k)$ 
6:   if  $G_t^+ > h$  or  $G_t^- > h$  then
7:     Emit RegimeChangedEvent
8:      $G_t^+, G_t^- \leftarrow 0$  (Reset CUSUM)
9:   end if
10: end for
```

---

### 2.3.1 Procesos Tangenciales y Bismut-Elworthy-Li

Para una difusión general  $dX_t = b(X_t)dt + \sigma(X_t)dW_t$ , la fórmula del peso de Malliavin debe generalizarse:

$$\partial_{X_0} E[f(X_T)] = E \left[ f(X_T) \int_0^T (\sigma^{-1}(X_s) Y_s \nabla b(X_s))^\top dW_s \right]$$

donde  $Y_t = \nabla_{X_0} X_t$  es el **Primer Proceso de Variación**, que satisface la EDO linealizada:

$$dY_t = \nabla b(X_t) Y_t dt + \sum_{k=1}^d \nabla \sigma_k(X_t) Y_t dW_t^k, \quad Y_0 = I$$

Se requiere resolver el sistema acoplado  $(X_t, Y_t)$  o utilizar **\*\*Diferenciación Automática\*\*** (Forward-Mode AD) para propagar la matriz Jacobiana a lo largo de la trayectoria.

### 2.3.2 Algoritmo Delta-Malliavin por Monte Carlo

Para calcular  $\Delta = \partial_{X_0} E[f(X_T)]$  en el caso simplificado:

$$\Delta \approx E \left[ f(X_T) \frac{W_T}{\sigma_{X_0 T}} \right]$$

En grafos de computación (TensorFlow/PyTorch):

1. Definir el grafo computacional del payoff  $L = f(X_T)$ .
2. Simular caminos forward  $X_0 \rightarrow X_1 \cdots \rightarrow X_T$ .
3. Ejecutar paso backward (Backprop) para obtener  $\nabla_{X_0} L$ .
4. Promediar  $\nabla_{X_0} L$  sobre  $M$  caminos.

## Capítulo 3

# Solvers Numéricos para Núcleos de Predicción

### 3.1 Rama A: Proyección de Hilbert y Filtrado de Wiener

#### 3.1.1 Algoritmo Recursivo de Levinson-Durbin

Para resolver el sistema de ecuaciones normales discretas de Yule-Walker (equivalente discreto a Wiener-Hopf) y encontrar el predictor lineal óptimo de orden  $p$ ,  $\hat{X}_{t+1} = \sum_{k=1}^p \phi_k X_{t-k+1}$ : **Nota:**

---

**Algorithm 4** Recursión de Levinson-Durbin

---

```
1: Input: Autocorrelaciones  $R_0, R_1, \dots, R_p$ .
2:  $E_0 \leftarrow R_0$ 
3: for  $k \leftarrow 1$  to  $p$  do
4:    $\lambda_k \leftarrow (R_k - \sum_{j=1}^{k-1} \phi_j^{(k-1)} R_{k-j}) / E_{k-1}$ 
5:    $\phi_k^{(k)} \leftarrow \lambda_k$ 
6:   for  $j \leftarrow 1$  to  $k-1$  do
7:      $\phi_j^{(k)} \leftarrow \phi_j^{(k-1)} - \lambda_k \phi_{k-j}^{(k-1)}$ 
8:   end for
9:    $E_k \leftarrow E_{k-1}(1 - \lambda_k^2)$ 
10: end for
11: Output: Coeficientes del filtro  $\phi^{(p)}$ .
```

---

Para eficiencia  $O(N \log N)$  en convoluciones largas, utilizar FFT (Teorema de Convolución) en lugar de recursión temporal directa.

### 3.2 Rama B: Ecuación HJB y Métodos de Viscosidad

#### 3.2.1 Esquemas de Diferencias Finitas Monótonas

El Teorema de Barles-Souganidis (1991) establece condiciones necesarias para la convergencia a soluciones de viscosidad.

**Esquema Numérico 3.1 (Esquema Upwind Generalizado)** Para la ecuación  $H(u, u_x, u_{xx}) = 0$ , utilizamos:

$$D_x^+ u_i = \frac{u_{i+1} - u_i}{\Delta x}, \quad D_x^- u_i = \frac{u_i - u_{i-1}}{\Delta x}$$
$$D_{xx} u_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2}$$



El paso de tiempo se actualiza explícitamente:

$$u_i^{n+1} = u_i^n - \Delta t \cdot H_{num}(u_i^n, D_x^+ u_i^n, D_x^- u_i^n, D_{xx} u_i^n)$$

**Condición de Monotonicidad:** El hamiltoniano numérico  $H_{num}(u, p, q, r)$  debe ser no decreciente en  $u$ ,  $p$ ,  $q$ , y  $r$  (dependiendo de la dirección del flujo de características).

### 3.2.2 Deep Galerkin Method (DGM)

Para alta dimensionalidad ( $d > 3$ ), donde las mallas son inviables ("Maldición de la Dimensionalidad").

---

#### Algorithm 5 Entrenamiento de Red Neuronal DGM

---

```

1: Input: Red  $f_\theta(t, x)$ , PDE  $\mathcal{L}u = 0$ , dominio  $\Omega$ , pasos  $M$ .
2: for  $i \leftarrow 1$  to  $M$  do
3:   Muestrear puntos aleatorios:
4:    $\{t_j, x_j\}_j \sim \text{Unif}([0, T] \times \Omega)$  (Interior)
5:    $\{\tau_k, \xi_k\}_k \sim \text{Unif}(\{T\} \times \Omega)$  (Condición Final)
6:    $\{\zeta_l, \gamma_l\}_l \sim \text{Unif}([0, T] \times \partial\Omega)$  (Frontera)
7:   Calcular Loss:
8:    $L_1 = \frac{1}{N} \sum (\partial_t f + \mathcal{L}f(t_j, x_j))^2$ 
9:    $L_2 = \frac{1}{K} \sum (f(T, \xi_k) - g(\xi_k))^2$ 
10:   $L_3 = \frac{1}{L} \sum (f(\zeta_l, \gamma_l) - h(\gamma_l))^2$ 
11:   $L(\theta) = L_1 + L_2 + L_3$ 
12:  Actualizar  $\theta \leftarrow \theta - \eta \nabla_\theta L(\theta)$  (Adam/SGD)
13: end for
```

---

## 3.3 Rama C: Ecuación Integro-Diferencial de Saltos

### 3.3.1 Algoritmo Delta-Malliavin en Espacios de Poisson

Para procesos con componente de salto  $J_t$ , la sensibilidad se basa en la Integración por Partes de Malliavin usando pesos de probabilidad:

$$\partial_{X_0} E[f(X_T)] \approx E \left[ f(X_T) \left( \frac{W_T}{\sigma T} + \sum_{i=1}^{N_T} \frac{\partial_X \Delta X_{\tau_i}}{\Delta X_{\tau_i}} \right) \right]$$

La implementación requiere rastrear los tiempos de salto  $\tau_i$  y sus amplitudes  $\Delta X_{\tau_i}$  durante el paso forward de Monte Carlo.

### 3.3.2 Esquema IMEX (Implícito-Explícito) para PIDEs

Para resolver la ecuación de Fokker-Planck con término integral  $\mathcal{I}[p](x) = \int p(y) \nu(dy)$ :

$$\frac{p_i^{n+1} - p_i^n}{\Delta t} = \underbrace{\mathcal{L}_{\text{diff}} p_i^{n+1}}_{\text{Implícito}} + \underbrace{\mathcal{I}[p^n]_i}_{\text{Explícito}}$$

La parte difusiva se resuelve invirtiendo una matriz tridiagonal (algoritmo Thomas). La integral de convolución se evalúa explícitamente usando FFT en cada paso de tiempo  $O(N \log N)$ .

## 3.4 Rama D: Computación de Signatures

### 3.4.1 Identidad de Chen y Truncamiento

El tensor de signatura  $\mathbf{S}(X)_{0,t}$  hasta nivel  $M$  vive en  $T^{(M)}(\mathbb{R}^d)$ . **Algoritmo Iterativo:** Dado un camino discretizado con incrementos  $\Delta X_k = X_{t_{k+1}} - X_{t_k}$ : 1. Calcular la signatura del segmento lineal  $\mathbf{S}(\Delta X_k) = \exp(\Delta X_k)$  en el álgebra tensorial. - Nivel 1:  $\Delta X_k$  - Nivel 2:  $\frac{1}{2}\Delta X_k \otimes \Delta X_k$  2. Concatenar usando la propiedad multiplicativa de Chen:

$$\mathbf{S}(X)_{0,t_{k+1}} = \mathbf{S}(X)_{0,t_k} \otimes \mathbf{S}(\Delta X_k)$$

Este producto tensorial se implementa eficientemente explotando la estructura triangular de los tensores.

### 3.4.2 Log-Signatures

Para reducir la dimensión del vector de características, proyectamos la signatura al álgebra de Lie libre mediante la fórmula de Baker-Campbell-Hausdorff (BCH). Librería recomendada: `iisignature` (Python/C++) o `signatory` (PyTorch, diferenciable).

## Capítulo 4

# Orquestador: Transporte Óptimo Regularizado

### 4.1 Circuit Breaker de Robustez (Pre-Orquestador)

Antes de ejecutar el pesaje de Wasserstein, se aplica una lógica condicional fuerte basada en el Postulado de Robustez ante Singularidades.

1. **Input:** vector  $V_s$  del SIA y pesos actuales  $w_t$ .
2. Si  $\alpha(t) < \alpha_{\text{threshold}}$  (Rugosidad Crítica) o  $d > 1.5$ :
  - Forzar  $w_D \leftarrow 1.0$  (Signature).
  - Cambiar función de costo Wasserstein a métrica Huber  $\rho_\delta(x - y)$ .
3. Si RegimeChangedEvent:
  - Reiniciar entropía:  $w_t \leftarrow \text{Softmax}(\mathbf{0})$  (Uniforme).
4. **Output:** Pesos ajustados para inicializar Sinkhorn.

### 4.2 Algoritmo de Sinkhorn-Knopp (Espacio Dual)

El algoritmo clásico es numéricamente inestable para pequeños  $\varepsilon$ . Implementar mediante reducción LogSumExp para variables potenciales duales  $f = \varepsilon \log u, g = \varepsilon \log v$ .

---

**Algorithm 6** Iteraciones de Sinkhorn Estabilizadas (Log-Domain)

---

- 1: **Input:** Costo  $C$ , marginales  $a, b$  (en log:  $\alpha = \log a, \beta = \log b$ ),  $\varepsilon$ .
  - 2: Inicializar duales  $f \leftarrow \mathbf{0}_N, g \leftarrow \mathbf{0}_N$
  - 3: **function** SMIN( $M, \epsilon$ )
  - 4:     **Return**  $-\epsilon \cdot \text{LogSumExp}(-M/\epsilon)$  operando por filas.
  - 5: **end function**
  - 6: **while** no convergencia **do**
  - 7:      $f \leftarrow \text{Smin}(C - g^\top, \epsilon) + \alpha$
  - 8:      $g \leftarrow \text{Smin}(C - f, \epsilon) + \beta$
  - 9: **end while**
  - 10: Distancia Sinkhorn  $W_\varepsilon \approx \langle \exp(f/\varepsilon), (K \odot C) \exp(g/\varepsilon) \rangle$
-

### 4.3 Esquema JKO Proximal

La actualización de pesos  $w^{(k+1)} = \operatorname{argmin}_w \dots$  requiere diferenciación a través del bucle de Sinkhorn.

**Implementación Diferenciable:** Utilizar librerías de autodiff (JAX/PyTorch) con soporte para `custom_vjp` (Vector-Jacobian Product) en el punto fijo de Sinkhorn, evitando desenrollar el bucle para ahorrar memoria:

$$\partial L / \partial C = P^* \quad (\text{Plan de Transporte Óptimo})$$

Esto permite alimentar el gradiente exacto  $\nabla_{W_2} \mathcal{F}$  al optimizador L-BFGS. La actualización de pesos  $w^{(k)}$  se implementa como un paso de gradiente implícito en la variedad de Wasserstein.

$$w^{(k+1)} = \operatorname{Prox}_{\tau \mathcal{F}}^{W_2}(w^{(k)})$$

Esto se resuelve anidando un bucle de Sinkhorn dentro de un optimizador L-BFGS o mediante descenso de gradiente proyectado si la regularización entrópica es suficiente para suavizar el paisaje de energía.

## Capítulo 5

# Arquitectura de Software y Paralelismo

### 5.1 Patrones de Construcción Orientados a Objetos

El sistema se diseña bajo principios SOLID para garantizar modularidad y extensibilidad de los núcleos predictivos.

#### 5.1.1 Estructura de Clases Sugerida

1. **AbstractStochasticProcess**: Clase base que define la interfaz `simulate(dt, steps)`.
2. **ModelIdentifier (SIA)**: Singleton que consume flujos de datos y emite eventos `RegimeChangedEvent`. Utiliza el patrón *Strategy* para intercambiar métodos de estimación (WTMM, DFA).
3. **PredictionKernel**: Interfaz abstracta para los predictores (A, B, C, D).
  - `fit(historical_data)`: Calibración de parámetros.
  - `predict(horizon)`: Generación de trayectorias futuras.
  - `compute_risk()`: Cálculo de VaR/ES.
4. **Orchestrator**: Implementa el patrón *Mediator*. Posee una instancia de `WassersteinOptimizer` y coordina la ponderación de los núcleos.

### 5.2 Computación Heterogénea y Aceleración

#### 5.2.1 GPU (CUDA/OpenCL)

El entrenamiento de redes neuronales (DGM) y las simulaciones de Monte Carlo masivas se delegan a la GPU.

- **Kernels**: Implementación de generadores de números aleatorios (coalesced memory access) y reducción paralela para el cálculo de esperanzas.
- **Sinkhorn**: Las operaciones matriciales ( $K \cdot v$ ) se realizan mediante librerías BLAS optimizadas (cuBLAS).

#### 5.2.2 FPGA (Field-Programmable Gate Array)

Para aplicaciones de ultra-baja latencia (HFT), la Rama D (Signatures) se sintetiza en hardware reconfigurable.

- **Pipeline**: El cálculo iterativo de la signatura  $S_{0,t} \otimes \Delta X$  se implementa como un pipeline sistólico.
- **Fixed-Point Arithmetic**: Se utiliza aritmética de punto fijo para maximizar el throughput, analizando previamente el rango dinámico de los tensores.

## Capítulo 6

# Consideraciones de Estabilidad Numérica

### 6.1 Condición CFL (Courant-Friedrichs-Lewy)

Para esquemas de diferencias finitas explícitos en la ecuación HJB (Rama B), el paso de tiempo debe satisfacer:

$$\Delta t \leq \frac{(\Delta x)^2}{2 \max \sigma^2}$$

Si la volatilidad es alta, el paso de tiempo se vuelve prohibitivamente pequeño. En tal caso, se debe cambiar a un esquema **Implícito** o **Semi-Lagrangiano**.

### 6.2 Estabilidad de la Signatura Logarítmica

El cálculo de log-signatures involucra la serie de Baker-Campbell-Hausdorff, que converge solo si los incrementos son pequeños.

---

**Algorithm 7** Control de Paso Adaptativo para Signatures

---

```
1: Input: Camino  $X$ , tolerancia  $\epsilon$ .  
2: function COMPUTESIG( $X$ )  
3:   if  $\|\Delta X\| > \epsilon$  then  
4:      $X_{\text{mid}} \leftarrow \text{Interpolate}(X)$  (Punto medio)  
5:      $S_1 \leftarrow \text{ComputeSig}(X_{\text{left}})$   
6:      $S_2 \leftarrow \text{ComputeSig}(X_{\text{right}})$   
7:     Return  $S_1 \otimes S_2$   
8:   else  
9:     Return  $\exp(\Delta X)$   
10:  end if  
11: end function
```

---

## Capítulo 7

# Gobernanza de Metaparámetros Heurísticos

La implementación de sistemas estocásticos en hardware finito exige la introducción de parámetros de regularización y truncamiento que no existen en la teoría de probabilidad continua. Este capítulo define la **Taxonomía de Control** para garantizar que la instanciación numérica del predictor sea estable, reactiva y causal.

### 7.1 Taxonomía y Acotación Analítica (Safe Harbors)

Los siguientes límites matemáticos son mandatorios para evitar colapso numérico (NaNs), explosión de gradientes o violación de causalidad.

#### 7.1.1 Parámetros de Discretización y Truncamiento

Definen la resolución del mundo simulado.

- **Paso de Tiempo** ( $\Delta t$ ): No es una variable libre. Debe satisfacer la condición de Courant-Friedrichs-Lewy (CFL) generalizada para PIDE estocásticas.

$$\Delta t \leq \frac{C_{\text{safe}} \cdot (\Delta x)^2}{2 \cdot \sup |\sigma(x)|^2 + \sup |b(x)| \cdot \Delta x}$$

Donde  $C_{\text{safe}} \approx 0.9$ . Esta es una condición CFL mixta (Advectiva-Difusiva), necesaria porque la dinámica posee tanto un término de deriva (advección) regido por  $\sup |b(x)|$  como un término de volatilidad (difusión) regido por  $\sup |\sigma(x)|^2$ . Una violación de este límite inducirá oscilaciones espurias en el solver DGM/IMEX.

- **Profundidad de Signatura** ( $M$ ): El truncamiento del álgebra tensorial  $T((\mathbb{R}^d))$  define la memoria topológica.
  - **Rango Seguro:**  $M \in [3, 5]$ .
  - **Justificación:**  $M < 3$  pierde información de no-conmutatividad (el orden de los eventos).  $M > 5$  invoca la maldición de la dimensionalidad (el vector de características crece como  $d^M$ ), saturando la RAM sin ganancia predictiva marginal.

#### 7.1.2 Parámetros de Regularización y Estabilidad

Controlan la suavidad de las soluciones en problemas mal planteados.

- **Entropía de Sinkhorn** ( $\varepsilon$ ): Convierte el transporte de Wasserstein duro en un problema convexo suave.

- **Inicialización:**  $\varepsilon \approx 10^{-2}$ .
- **Límite Inferior:**  $\varepsilon \geq 10^{-4}$  (para float32). Valores menores provocan *underflow* numérico en la exponenciación de la matriz de costos  $K = e^{-C/\varepsilon}$ .
- **Impacto:**  $\varepsilon \rightarrow \infty$  genera una mixtura uniforme (máxima incertidumbre).  $\varepsilon \rightarrow 0$  genera un "Winner-takes-all" inestable.
- **Paso Proximal JKO** ( $\tau$ ): Dicta la velocidad de cambio de la distribución de pesos  $\rho$  sobre la variedad de Wasserstein.

$$\rho_{k+1} = \text{Prox}_{\tau E}^W(\rho_k)$$

Un  $\tau$  alto permite adaptación rápida pero ruidosa. Un  $\tau$  bajo induce memoria inercial excesiva. Se recomienda  $\tau$  adaptativo inversamente proporcional a la volatilidad del error de predicción.

### 7.1.3 Umbrales de Decisión (Fronteras Duras)

Convierten probabilidades continuas en acciones discretas (ej. activar Circuit Breaker).

- **Umbral CUSUM** ( $h$ ): No debe ser una constante mágica. Debe definirse dinámicamente como  $k \cdot \sigma_{\text{resid}}$ , donde  $\sigma_{\text{resid}}$  es la desviación estándar histórica de los residuos de predicción y  $k \approx 3$  (regla de tres sigma).
- **Tolerancia de Singularidad** ( $H_{\min}$ ): Umbral del exponente de Hölder para activar el modo de emergencia (Signatures). Típicamente  $H_{\min} \in [0.4, 0.5]$  para detectar regímenes de reversión a la media violentos o crash de mercado.

## 7.2 Validación Cruzada Causal (Walk-Forward Validation)

Los métodos de validación estática (K-Fold tradicional) están prohibidos pues violan la flecha del tiempo y filtran información futura (Look-ahead bias). El único esquema de validación aceptable es *Rolling Walk-Forward* con ventana deslizante para evitar la dilución de regímenes recientes.

---

#### Algorithm 8 Protocolo de Validación Walk-Forward Estricto (Rolling Window)

---

- 1: **Input:** Stream de datos  $\mathcal{D} = \{x_1, \dots, x_T\}$ , ventana inicial  $L_{\text{train}}$ , horizonte  $H$ , memoria máxima  $W_{\text{max}}$ .
  - 2: **Output:** Error de Generalización Agregado  $\mathcal{E}$ .
  - 3:  $t \leftarrow L_{\text{train}}$
  - 4: errors  $\leftarrow []$
  - 5: **while**  $t + H \leq T$  **do**
  - 6:    $start\_idx \leftarrow \max(1, t - W_{\text{max}})$
  - 7:    $\mathcal{D}_{\text{train}} \leftarrow \{x_{start\_idx}, \dots, x_t\}$  ▷ Ventana Deslizante (Rolling)
  - 8:    $\mathcal{D}_{\text{test}} \leftarrow \{x_{t+1}, \dots, x_{t+H}\}$  ▷ Futuro inmediato desconocido
  - 9:   **Training:** Optimizar Meta-Predictor ( $\theta$ ) en  $\mathcal{D}_{\text{train}}$
  - 10:   **Inference:**  $\hat{y} \leftarrow \text{Predecir}(\mathcal{D}_{\text{test}}, \theta)$
  - 11:   **Evaluate:**  $e_t \leftarrow \text{Metric}(\hat{y}, \mathcal{D}_{\text{test}})$
  - 12:   errors.append( $e_t$ )
  - 13:    $t \leftarrow t + H$  ▷ Avanzar el tiempo paso a paso
  - 14: **end while**
  - 15: **return** Media(errors)
-



### 7.3 Meta-Optimización Libre de Derivadas (Optimización Bayesiana)

Dado que muchos hiperparámetros son discretos (profundidad de árbol  $M$ , umbrales de decisión) o la superficie de error es ruidosa y no-convexa, el descenso de gradiente es inaplicable.

Se prescribe el uso de **\*\*Procesos Gaussianos (GP)\*\*** para la búsqueda eficiente del siguiente candidato óptimo  $\theta_{\text{next}}$ :

$$\theta_{\text{next}} = \arg \max_{\theta \in \Theta} \text{Expected Improvement}(\theta | \mathcal{D}_{\text{obs}})$$

Donde la función objetivo es el retorno negativo de la Validación Walk-Forward ( $-\mathcal{E}$ ). Tras  $N$  iteraciones, el óptimo global estimado  $\theta^*$  será el candidato que minimizó empíricamente el error  $\mathcal{E}$ .

1. **Prior:** Definir rangos seguros (Sección 7.1) para cada hiperparámetro.
2. **Surrogate Model:** Entrenar un GP sobre pares  $(\theta_i, \text{Performance}_i)$  observados hasta el momento.
3. **Acquisition Function:** Seleccionar el siguiente  $\theta_{\text{next}}$  que maximice la probabilidad de mejorar el mejor resultado actual (balanceando exploración vs. explotación).
4. **Evaluación Costosa:** Ejecutar el protocolo Walk-Forward completo solo para  $\theta_{\text{next}}$ .

Este enfoque reduce drásticamente el coste computacional comparado con Grid Search o Random Search en espacios de alta dimensión, convergiendo a la "personalidad" óptima del predictor en pocas iteraciones.