Universidad Nacional de Colombia

Facultad de ciencias

Departamento de matemáticas

Cadenas de Markov 2025-II

Metodos para distribución estacionaria

Estudiantes:
Jose Miguel Acuña Hernandez
Andres Puertas Londoño
Guillermo Murillo Tirado

Docente:
Freddy Hernandez

Contenido

1. Marco Teórico

1.1. Método del Autovector

1.1.1. Fundamentación Teórica

La distribución estacionaria π puede caracterizarse como el **autovector izquierdo** de la matriz P asociado al autovalor $\lambda = 1$:

$$\pi P = \pi \Leftrightarrow \pi^T P^T = \pi^T \tag{1}$$

Esto equivale a resolver el problema de autovalores:

$$P^T \mathbf{v} = \mathbf{v} \tag{2}$$

donde $\mathbf{v} = \pi^T$ es el autovector derecho de P^T correspondiente a $\lambda = 1$.
Teorema de Perron-Frobenius: Para una matriz estocástica irreducible P:

- El autovalor $\lambda = 1$ es simple y dominante: $|\lambda_i| \leq 1$ para $i \geq 2$
- Existe un único autovector izquierdo positivo π (normalizado) asociado a $\lambda = 1$
- Todos los demás autovalores satisfacen $|\lambda_i| < 1$ si P es aperiódica

1.1.2. Algoritmos de Implementación

Descomposición Espectral Directa:
El algoritmo `np.linalg.eig(P.T)` calcula todos los autovalores y autovectores mediante:

1. **Reducción a forma de Hessenberg** usando transformaciones de Householder
2. **Algoritmo QR con desplazamientos** para encontrar autovalores
3. **Cálculo de autovectores** por sustitución hacia atrás

Método de la Potencia:
Para encontrar el autovector dominante, se itera:

$$\pi^{(k+1)} = \frac{\pi^{(k)} P}{\|\pi^{(k)} P\|_1} \tag{3}$$

La convergencia está garantizada por:

$$\|\pi^{(k)} - \pi\|_1 \leq C \left|\frac{\lambda_2}{\lambda_1}\right|^k = C|\lambda_2|^k \tag{4}$$

donde λ_2 es el segundo autovalor más grande en módulo.

1.1.3. Análisis de Complejidad Computacional

Descomposición Espectral Completa:
La complejidad temporal es $\mathcal{O}(n^3)$ con las siguientes contribuciones:

- Reducción Hessenberg: $\approx \frac{10n^3}{3}$ operaciones de punto flotante
- Iteraciones QR: $\approx 6n^3$ operaciones (promedio)
- Cálculo de autovectores: $\approx 3n^3$ operaciones

Total: $\approx 10n^3$ operaciones de punto flotante.
Método de la Potencia:
Cada iteración requiere:

- Multiplicación vector-matriz: $\mathcal{O}(n^2)$ operaciones
- Normalización: $\mathcal{O}(n)$ operaciones

Para k iteraciones: $\mathcal{O}(kn^2)$ donde:

$$k = \mathcal{O}\left(\frac{\log(\epsilon)}{\log(|\lambda_2|)}\right) \tag{5}$$

En el peor caso (matrices mal condicionadas): $k = \mathcal{O}(n)$, resultando en $\mathcal{O}(n^3)$. En casos típicos: $k = \mathcal{O}(\log n)$, resultando en $\mathcal{O}(n^2 \log n)$.
Complejidad Espacial:

- Descomposición espectral: $\mathcal{O}(n^2)$ (matriz completa + autovectores)
- Método de la potencia: $\mathcal{O}(n)$ (solo vectores)

1.2. Método de Tiempos Medios de Primer Retorno

1.2.1. Fundamentación Teórica

El **tiempo medio de primer retorno** al estado j se define como:

$$E[T_j] = E[\text{mín}\{n \geq 1 : X_n = j | X_0 = j\}] \tag{6}$$

Teorema Fundamental: Para una cadena de Markov irreducible con distribución estacionaria π:

$$\pi_j = \frac{1}{E[T_j]} \tag{7}$$

Esta relación establece que la probabilidad estacionaria es inversamente proporcional al tiempo esperado de retorno.
Derivación del Sistema Lineal:
Sea $m_j^{(i)}$ el tiempo esperado de primer retorno al estado j comenzando desde el estado $i \neq j$. Entonces:

$$m_j^{(i)} = 1 + \sum_{k \neq j} P_{ik} m_j^{(k)} \tag{8}$$

Esto genera un sistema lineal de $(n-1)\times(n-1)$:

$$(I - P_{-j})\mathbf{m}_j = \mathbf{1} \tag{9}$$

donde P_{-j} es la matriz P con la fila y columna j eliminadas, y $\mathbf{m}_j$ es el vector de tiempos de retorno desde todos los estados excepto j.
El tiempo medio de retorno desde j es:

$$E[T_j] = 1 + \sum_{k\neq j} P_{jk} m_j^{(k)} \tag{10}$$

1.2.2. Implementación Algorítmica

El algoritmo estándar procede como sigue:

1. **Para cada estado** $j = 0, 1, \ldots, n-1$**:**
2. Construir matriz reducida $P_{-j} \in \mathbb{R}^{(n-1)\times(n-1)}$
3. Resolver sistema lineal $(I - P_{-j})\mathbf{m}_j = \mathbf{1}$
4. Calcular $E[T_j] = 1 + \mathbf{p}_{j,-j}^T \mathbf{m}_j$
5. Obtener $\pi_j = 1/E[T_j]$

1.2.3. Análisis Detallado de Complejidad

Implementación No Optimizada:
Paso 1 - Construcción de P_{-j}**:**

- Eliminar fila j: copiar $n\times(n-1)$ elementos $\Rightarrow \mathcal{O}(n^2)$
- Eliminar columna j: copiar $(n-1)\times(n-1)$ elementos $\Rightarrow \mathcal{O}(n^2)$

Paso 2 - Resolución del Sistema Lineal:
El sistema $(I - P_{-j})\mathbf{m}_j = \mathbf{1}$ se resuelve mediante factorización LU:
Factorización LU:

$$\text{Costo} = \sum_{k=0}^{n-2}(n-1-k)^2 \approx \frac{(n-1)^3}{3} = \mathcal{O}(n^3) \tag{11}$$

Sustitución hacia adelante $(L\mathbf{y} = \mathbf{1})$:

$$\text{Costo} = \sum_{i=0}^{n-2} i = \frac{(n-1)(n-2)}{2} = \mathcal{O}(n^2) \tag{12}$$

Sustitución hacia atrás $(U\mathbf{m}_j = \mathbf{y})$:

$$\text{Costo} = \mathcal{O}(n^2) \tag{13}$$

Costo total por sistema: $\mathcal{O}(n^3)$
Paso 3 - Cálculo de $E[T_j]$**:** Producto escalar: $\mathcal{O}(n)$
Complejidad Total: Para n estados: $n\times\mathcal{O}(n^3) = \mathcal{O}(n^4)$
Desglose de Operaciones de Punto Flotante:

- Factorización LU por sistema: $\approx \frac{2(n-1)^3}{3}$ flops
- Sustituciones: $\approx 2(n-1)^2$ flops
- Total por estado: $\approx \frac{2n^3}{3}$ flops
- Total algoritmo: $\approx \frac{2n^4}{3}$ flops

Propuesta de Optimización:
En lugar de resolver n sistemas separados, se puede usar la **matriz fundamental**:

$$Z = (I - P + \mathbf{1}\pi^T)^{-1} \tag{14}$$

Los tiempos de retorno se obtienen directamente:

$$E[T_j] = \frac{Z_{jj}}{\pi_j} \tag{15}$$

Esta optimización reduce la complejidad a $\mathcal{O}(n^3)$ (una sola factorización).

1.3. Análisis Comparativo de Complejidad Computacional

1.3.1. Complejidades Teóricas

Método	Complejidad Temporal	Complejidad Espacial	Estabilidad
Autovector (descomp. espectral)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	Excelente
Autovector (método potencia)	$\mathcal{O}(kn^2)$	$\mathcal{O}(n)$	Buena
Tiempos (no optimizado)	$\mathcal{O}(n^4)$	$\mathcal{O}(n^2)$	Excelente
Tiempos (optimizado)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$	Excelente

1.3.2. Análisis de Constantes Multiplicativas

Para una matriz $n \times n$, las operaciones de punto flotante exactas son:
Método del Autovector (descomposición espectral):

- Reducción Hessenberg: $\frac{10n^3}{3}$ flops
- Algoritmo QR: $6n^3$ flops (promedio)
- Cálculo autovectores: $3n^3$ flops
- **Total**: $\approx 10n^3$ flops

Método de Tiempos (implementación estándar):

- Factorización LU por estado: $\frac{2(n-1)^3}{3}$ flops
- Para n estados: $n \times \frac{2(n-1)^3}{3} \approx \frac{2n^4}{3}$ flops
- **Total**: $\approx 0.67n^4$ flops

1.3.3. Razón de Complejidades Empíricas

La razón de tiempos de ejecución para matrices de diferentes tamaños:

$$\frac{\text{Tiempo(Tiempos)}}{\text{Tiempo(Autovector)}} \approx \frac{0.67n^4}{10n^3} = 0.067n \tag{16}$$

Ejemplos numéricos:

- $n = 100$: Razón $\approx 6.7\times$
- $n = 500$: Razón $\approx 33.5\times$
- $n = 1000$: Razón $\approx 67\times$

1.3.4. Efectos de Optimizaciones de Hardware

Bibliotecas BLAS/LAPACK:

- Optimizaciones de caché y vectorización
- Paralelización automática para operaciones matriciales
- Speedup típico: $10 - 100\times$ vs implementación ingenua

Factores que Afectan Mediciones Empíricas:

1. **Tamaños pequeños**: Términos de orden inferior dominan para $n < 100$
2. **Jerarquía de memoria**: Matrices que caben en caché L2/L3 son significativamente más rápidas
3. **Paralelización**: BLAS puede usar múltiples threads automáticamente
4. **Overhead del intérprete**: Constante aditiva significativa para n pequeño

1.3.5. Convergencia del Método de la Potencia

La velocidad de convergencia depende del **gap espectral**:

$$\text{Error}^{(k)} \leq C \left| \frac{\lambda_2}{\lambda_1} \right|^k = C|\lambda_2|^k \tag{17}$$

Casos típicos:

- Matrices bien condicionadas: $|\lambda_2| \leq 0.9 \Rightarrow k = \mathcal{O}(\log n)$
- Matrices mal condicionadas: $|\lambda_2| \approx 1 \Rightarrow k = \mathcal{O}(n)$

1.4. Recomendaciones Algorítmicas

1.4.1. Criterios de Selección

Para matrices pequeñas ($n \leq 100$):

- Usar descomposición espectral directa
- Overhead de setup es despreciable
- Máxima precisión numérica

Para matrices medianas ($100 < n \leq 1000$):

- Descomposición espectral si se requiere precisión
- Método de la potencia si la memoria es limitada
- Evitar método de tiempos no optimizado

Para matrices grandes ($n > 1000$):

- Considerar métodos iterativos especializados
- Explotar estructura dispersa si es aplicable
- Método de la potencia con precondicionamiento

1.4.2. Consideraciones de Estabilidad Numérica

Condicionamiento de la matriz P: El número de condición $\kappa(P) = \frac{\sigma_{\text{máx}}}{\sigma_{\text{mín}}}$ afecta la precisión:

- $\kappa(P) < 10^{12}$: Precisión de máquina alcanzable
- $\kappa(P) > 10^{12}$: Pérdida significativa de dígitos significativos

Propagación de errores:

$$\frac{\|\Delta\pi\|}{\|\pi\|} \leq \kappa(P) \frac{\|\Delta P\|}{\|P\|} \tag{18}$$

El método del autovector es generalmente más robusto ante perturbaciones en P que el método de tiempos medios.

2. Función de las 3 Cadenas

2.1. Caminata Aleatoria Simple

La caminata aleatoria simple implementada corresponde a un proceso en una estructura cíclica de n estados, donde las probabilidades de transición están definidas por p (probabilidad de avanzar al siguiente estado) y q = 1-p (probabilidad de retroceder al estado anterior). La matriz de transición P presenta la forma donde cada estado i puede transitar al estado (i+1) mod n con probabilidad p, o al estado (i-1) mod n con probabilidad q.
La implementación se realiza mediante la función `generar_caminata_aleatoria(n, p)` que construye la matriz de transición correspondiente:

```
def generar_caminata_aleatoria(n, p):
    """
    Genera matriz de transición para caminata aleatoria cíclica.

    Args:
        n: Número de estados
        p: Probabilidad de ir al siguiente estado (q = 1-p al anterior)

    Returns:
        P: Matriz de transición
    """
    q = 1 - p
    P = np.zeros((n, n))

    for i in range(n):
        P[i, (i + 1) % n] = p  # Probabilidad de ir al siguiente (mod n para ciclo)
        P[i, (i - 1) % n] = q  # Probabilidad de ir al anterior (mod n para ciclo)

    return P
```

La configuración experimental utiliza un rango de estados desde n = 10 hasta n = 560 con incrementos de 50, y cuatro valores de probabilidad p = [0.2, 0.4, 0.6, 0.8]. Para cada combinación de parámetros, se genera la matriz de transición correspondiente y se aplican ambos métodos de cálculo de distribución estacionaria, midiendo los tiempos de ejecución mediante `perf_counter()` y calculando el error de convergencia entre los resultados obtenidos por ambos métodos.

3. Bloques de Código que Generan los Cambios en las Cadenas

4. Resultados de la Comparación con las Gráficas