



# 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

<b>Marco Teórico</b>	<b>1</b>
Cadenas de Markov y Distribución Estacionaria . . . . .	1
Método del Autovector . . . . .	1
Método de Tiempos Medios de Primer Retorno . . . . .	2
Análisis Comparativo de Complejidad Computacional . . . . .	2
<b>Función de las 3 Cadenas</b>	<b>3</b>
Caminata Aleatoria Simple . . . . .	3
Caminata Aleatoria Doble (Forma de "8") . . . . .	4
Matriz Perturbada (Casi-Identidad) . . . . .	5
<b>Protocolo Experimental y Medición</b>	<b>5</b>
<b>Resultados de la Comparación con las Gráficas</b>	<b>6</b>
Caminata Aleatoria Simple . . . . .	6
Caminata Aleatoria Doble . . . . .	7
Matriz Perturbada . . . . .	7
Análisis Comparativo Global . . . . .	8

## 1. Marco Teórico

### 1.1. Cadenas de Markov y Distribución Estacionaria

Una cadena de Markov es un proceso estocástico  $\{X_n\}_{n \geq 0}$  con espacio de estados finito  $S = \{0, 1, \dots, n-1\}$  que satisface la propiedad de Markov:  $P(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \dots, X_0 = i_0) = P(X_{n+1} = j | X_n = i)$ . La matriz de transición  $P \in \mathbb{R}^{n \times n}$  está definida por  $P_{ij} = P(X_{n+1} = j | X_n = i)$ , donde cada elemento es no negativo y cada fila suma uno, constituyendo así una matriz estocástica.

La distribución estacionaria es un vector  $\pi \in \mathbb{R}^n$  que satisface la ecuación fundamental  $\pi = \pi P$  junto con las condiciones de normalización  $\sum_{i=0}^{n-1} \pi_i = 1$  y no negatividad  $\pi_i \geq 0$ . Para matrices irreducibles y aperiódicas, el teorema de existencia y unicidad garantiza que existe una única distribución estacionaria  $\pi$  tal que  $\lim_{n \rightarrow \infty} P^n = \mathbf{1}\pi^T$ , donde  $\mathbf{1}$  es el vector de unos.

### 1.2. Método del Autovector

#### 1.2.1. Fundamentación Teórica

La distribución estacionaria  $\pi$  puede caracterizarse como el autovector izquierdo de la matriz  $P$  asociado al autovalor  $\lambda = 1$ , expresado mediante la relación  $\pi P = \pi \Leftrightarrow \pi^T P^T = \pi^T$ . Esto equivale a resolver el problema de autovalores  $P^T \mathbf{v} = \mathbf{v}$ , donde  $\mathbf{v} = \pi^T$  es el autovector derecho de  $P^T$  correspondiente a  $\lambda = 1$ .

El Teorema de Perron-Frobenius establece que para una matriz estocástica irreducible  $P$ , el autovalor  $\lambda = 1$  es simple y dominante con  $|\lambda_i| \leq 1$  para  $i \geq 2$ , existe un único autovector izquierdo positivo  $\pi$  normalizado asociado a  $\lambda = 1$ , y todos los demás autovalores satisfacen  $|\lambda_i| < 1$  si  $P$  es aperiódica. Esta fundamentación teórica garantiza la existencia y unicidad de la solución buscada.

### 1.2.2. Implementación Algorítmica

El algoritmo utilizado en este trabajo emplea la descomposición espectral directa mediante `np.linalg.eig(P.T)`, que calcula todos los autovalores y autovectores a través de tres etapas principales. Primero, se realiza la reducción a forma de Hessenberg usando transformaciones de Householder, seguida por la aplicación del algoritmo QR con desplazamientos para encontrar los autovalores, y finalmente el cálculo de autovectores por sustitución hacia atrás.

La complejidad temporal de este método es  $\mathcal{O}(n^3)$  con contribuciones específicas: la reducción Hessenberg requiere aproximadamente  $\frac{10n^3}{3}$  operaciones de punto flotante, las iteraciones QR demandan cerca de  $6n^3$  operaciones en promedio, y el cálculo de autovectores consume alrededor de  $3n^3$  operaciones, resultando en un total aproximado de  $10n^3$  operaciones de punto flotante. La complejidad espacial es  $\mathcal{O}(n^2)$  debido al almacenamiento de la matriz completa y los autovectores.

## 1.3. Método de Tiempos Medios de Primer Retorno

### 1.3.1. Fundamentación Teórica

El tiempo medio de primer retorno al estado  $j$  se define como  $E[T_j] = E[\min\{n \geq 1 : X_n = j | X_0 = j\}]$ . El teorema fundamental establece que para una cadena de Markov irreducible con distribución estacionaria  $\pi$ , se cumple la relación  $\pi_j = \frac{1}{E[T_j]}$ , indicando que la probabilidad estacionaria es inversamente proporcional al tiempo esperado de retorno.

Para derivar el sistema lineal correspondiente, sea  $m_j^{(i)}$  el tiempo esperado de primer retorno al estado  $j$  comenzando desde el estado  $i \neq j$ . Entonces se cumple  $m_j^{(i)} = 1 + \sum_{k \neq j} P_{ik} m_j^{(k)}$ , lo que genera un sistema lineal de dimensión  $(n-1) \times (n-1)$  expresado como  $(I - P_{-j})\mathbf{m}_j = \mathbf{1}$ , 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  $j$  se calcula entonces como  $E[T_j] = 1 + \sum_{k \neq j} P_{jk} m_j^{(k)}$ .

### 1.3.2. Implementación y Análisis de Complejidad

El algoritmo estándar procede iterativamente para cada estado  $j = 0, 1, \dots, n-1$ , construyendo la matriz reducida  $P_{-j} \in \mathbb{R}^{(n-1) \times (n-1)}$ , resolviendo el sistema lineal  $(I - P_{-j})\mathbf{m}_j = \mathbf{1}$ , calculando  $E[T_j] = 1 + \mathbf{p}_{j,-j}^T \mathbf{m}_j$ , y finalmente obteniendo  $\pi_j = 1/E[T_j]$ .

La construcción de  $P_{-j}$  requiere eliminar la fila  $j$  copiando  $n \times (n-1)$  elementos y eliminar la columna  $j$  copiando  $(n-1) \times (n-1)$  elementos, resultando en una complejidad  $\mathcal{O}(n^2)$  para cada matriz reducida. La resolución del sistema lineal se realiza mediante factorización LU, donde el costo de factorización es  $\sum_{k=0}^{n-2} (n-1-k)^2 \approx \frac{(n-1)^3}{3} = \mathcal{O}(n^3)$ , la sustitución hacia adelante ( $L\mathbf{y} = \mathbf{1}$ ) tiene costo  $\sum_{i=0}^{n-2} i = \frac{(n-1)(n-2)}{2} = \mathcal{O}(n^2)$ , y la sustitución hacia atrás ( $U\mathbf{m}_j = \mathbf{y}$ ) también requiere  $\mathcal{O}(n^2)$  operaciones, resultando en un costo total por sistema de  $\mathcal{O}(n^3)$ .

Al considerar que este proceso debe repetirse para  $n$  estados, la complejidad total del algoritmo es  $n \times \mathcal{O}(n^3) = \mathcal{O}(n^4)$ .

En términos de operaciones de punto flotante específicas, cada factorización LU requiere aproximadamente  $\frac{2(n-1)^3}{3}$  flops, las sustituciones demandan cerca de  $2(n-1)^2$  flops, resultando en un total por estado de aproximadamente  $\frac{2n^3}{3}$  flops y un total para todo el algoritmo de cerca de  $\frac{2n^4}{3}$  flops.

## 1.4. Análisis Comparativo de Complejidad Computacional

La comparación entre ambos métodos revela diferencias significativas en términos de eficiencia computacional. El método del autovector mediante descomposición espectral presenta complejidad temporal  $\mathcal{O}(n^3)$  y espacial  $\mathcal{O}(n^2)$  con excelente estabilidad numérica, mientras que el método de tiempos medios en su implementación estándar tiene complejidad temporal  $\mathcal{O}(n^4)$  y espacial  $\mathcal{O}(n^2)$  manteniendo también excelente estabilidad.

El análisis de constantes multiplicativas muestra que para una matriz  $n \times n$ , el método del autovector requiere aproximadamente  $10n^3$  operaciones de punto flotante distribuidas entre la reducción Hessenberg ( $\frac{10n^3}{3}$  flops), el algoritmo QR ( $6n^3$  flops en promedio), y el cálculo de autovectores ( $3n^3$  flops). En contraste, el método de tiempos en su implementación estándar demanda aproximadamente  $0.67n^4$  flops, considerando  $\frac{2(n-1)^3}{3}$  flops por factorización LU multiplicado por  $n$  estados.

La razón de tiempos de ejecución entre ambos métodos puede expresarse como  $\frac{\text{Tiempo(Tiempos)}}{\text{Tiempo(Autovector)}} \approx \frac{0.67n^4}{10n^3} = 0.067n$ , lo que implica que para matrices de tamaño  $n = 100$  el método de tiempos es aproximadamente 6.7 veces más lento, para  $n = 500$  es 33.5 veces más lento, y para  $n = 1000$  alcanza una razón de 67 veces más lento que el método del autovector.

### 1.4.1. Efectos de Optimizaciones Computacionales

Las bibliotecas BLAS/LAPACK proporcionan optimizaciones significativas incluyendo manejo eficiente de caché y vectorización, paralelización automática para operaciones matriciales, y un speedup típico de 10 – 100 veces comparado con implementaciones ingenuas. Varios factores afectan las mediciones empíricas: para tamaños pequeños ( $n < 100$ ) los términos de orden inferior dominan el comportamiento asintótico, la jerarquía de memoria hace que matrices que caben en caché L2/L3 sean significativamente más rápidas, BLAS puede utilizar múltiples threads automáticamente, y el overhead del intérprete constituye una constante aditiva significativa para valores pequeños de  $n$ .

### 1.4.2. Consideraciones de Estabilidad Numérica

El condicionamiento de la matriz  $P$  afecta sustancialmente la precisión de los resultados. El número de condición  $\kappa(P) = \frac{\sigma_{\max}}{\sigma_{\min}}$  determina la calidad numérica: cuando  $\kappa(P) < 10^{12}$  es posible alcanzar precisión de máquina, mientras que  $\kappa(P) > 10^{12}$  resulta en pérdida significativa de dígitos significativos. La propagación de errores se rige por la desigualdad  $\frac{\|\Delta\pi\|}{\|\pi\|} \leq \kappa(P) \frac{\|\Delta P\|}{\|P\|}$ , donde el método del autovector es generalmente más robusto ante perturbaciones en  $P$  que el método de tiempos medios.

Para la selección algorítmica, matrices pequeñas ( $n \leq 100$ ) se benefician de la descomposición espectral directa donde el overhead de configuración es despreciable y se obtiene máxima precisión numérica. Para matrices medianas ( $100 < n \leq 1000$ ) se recomienda la descomposición espectral cuando se requiere precisión, evitando el método de tiempos no optimizado. En matrices grandes ( $n > 1000$ ) conviene considerar métodos iterativos especializados que puedan explotar estructura dispersa cuando sea aplicable.

## 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  $(i+1) \bmod n$  con probabilidad  $p$ , o al estado  $(i-1) \bmod n$  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.

## 2.2. Caminata Aleatoria Doble (Forma de "8")

La caminata aleatoria doble implementa una estructura topológica en forma de "8", donde el espacio de estados se divide en dos mitades conectadas que forman ciclos independientes con puntos de intercomunicación. Esta configuración requiere que el número total de estados  $n$  sea par para permitir la división equitativa en dos componentes de  $n/2$  estados cada una.

La función `generar_caminata_doble(n, p, r)` construye esta estructura mediante tres parámetros fundamentales:  $n$  determina el tamaño total del espacio de estados,  $p$  establece la probabilidad de transición al siguiente estado dentro de la misma mitad, y  $r$  define la probabilidad de salto entre las dos mitades del sistema. Cada mitad opera como un ciclo cerrado donde los estados intermedios tienen probabilidad  $p$  de avanzar al siguiente estado y probabilidad  $1-p$  de retroceder al anterior, manteniendo la propiedad cíclica mediante aritmética modular.

```
def generar_caminata_doble(n, p, r):
    """
    Genera matriz de transición para caminata aleatoria en forma de "8".

    Args:
        n: Número total de estados (debe ser par)
        p: Probabilidad de ir al siguiente estado dentro de la misma mitad
        r: Probabilidad de ir de una mitad a la otra

    Returns:
        P: Matriz de transición
    """
    if n % 2 != 0:
        raise ValueError("n debe ser par para dividir en dos mitades")

    P = np.zeros((n, n))
    mitad = n // 2

    # Primera mitad: estados 0 a mitad-1
    for i in range(mitad):
        if i < mitad - 1:
            # Estados intermedios de la primera mitad
            P[i, i + 1] = p # Siguiendo en la misma mitad
            P[i, i - 1 if i > 0 else mitad - 1] = 1 - p # Anterior (cíclico)
        else:
            # Último estado de la primera mitad (i = mitad-1)
            P[i, 0] = p # Va al primero de la misma mitad
            P[i, i - 1] = 1 - p - r # Anterior en la misma mitad
            P[i, mitad] = r # Conexión a la segunda mitad

    # Segunda mitad: estados mitad a n-1
    for i in range(mitad, n):
        if i < n - 1:
            # Estados intermedios de la segunda mitad
            P[i, i + 1] = p # Siguiendo en la misma mitad
            P[i, i - 1 if i > mitad else n - 1] = 1 - p # Anterior (cíclico)
        else:
            # Último estado de la segunda mitad (i = n-1)
            P[i, mitad] = p # Va al primero de la segunda mitad
            P[i, i - 1] = 1 - p - r # Anterior en la misma mitad
            P[i, mitad - 1] = r # Conexión a la primera mitad
```

```
return P
```

La configuración experimental para esta estructura utiliza tamaños de  $n$  desde 10 hasta 500 con incrementos de 50, asegurando que todos los valores sean pares. Los parámetros de probabilidad incluyen  $p \in \{0.3, 0.5, 0.7\}$  para las transiciones internas y  $r \in \{0.05, 0.1, 0.2\}$  para los saltos entre mitades, permitiendo analizar el efecto de diferentes niveles de conectividad entre las dos componentes del sistema.

### 2.3. Matriz Perturbada (Casi-Identidad)

La matriz perturbada representa un caso límite donde la cadena de Markov permanece predominantemente en el mismo estado, implementando una perturbación minimal de la matriz identidad. Esta configuración genera matrices con número de condición elevado, permitiendo evaluar la estabilidad numérica de los algoritmos en condiciones adversas.

La implementación utiliza una perturbación  $\varepsilon = 1 \times 10^{-8}$  aplicada sistemáticamente: la diagonal principal se reduce mediante la substracción de  $\varepsilon$ , mientras que las diagonales superior e inferior reciben una adición de  $\varepsilon/2$ . Posteriormente se realiza una normalización por filas para garantizar que cada fila sume exactamente uno, preservando la propiedad estocástica de la matriz.

```
def generar_matriz_perturbada(n):
    """
    Genera matriz de transición basada en identidad con perturbación.

    Args:
        n: Número de estados

    Returns:
        P: Matriz de transición estocástica
    """
    epsilon = 1e-8
    P = np.eye(n)

    # Restar epsilon de la diagonal principal
    P -= epsilon * np.eye(n)

    # Añadir epsilon/2 a las diagonales superior e inferior
    for i in range(n - 1):
        P[i, i + 1] += epsilon / 2 # Diagonal superior
        P[i + 1, i] += epsilon / 2 # Diagonal inferior

    # Normalizar filas para asegurar que sumen 1
    row_sums = P.sum(axis=1)
    P = P / row_sums[:, np.newaxis]

    return P
```

La experimentación con matrices perturbadas utiliza tamaños desde  $n = 10$  hasta  $n = 400$  con incrementos de 40, empleando un rango menor que las otras configuraciones debido a la naturaleza computacionalmente sensible de estas matrices mal condicionadas. Adicionalmente, se calcula el número de condición de cada matriz generada para monitorear el grado de condicionamiento y su impacto en la precisión de los resultados.

## 3. Protocolo Experimental y Medición

La metodología experimental implementada mantiene consistencia entre todas las configuraciones de cadenas de Markov estudiadas. Para cada combinación de parámetros, se genera la matriz de transición correspondiente y se verifica que cumple las propiedades estocásticas mediante la validación de que todas las filas suman uno con tolerancia numérica.

Los tiempos de ejecución se miden utilizando `perf_counter()` que proporciona la mayor resolución temporal disponible en el sistema. Para cada método, se ejecuta una sola medición por combinación de parámetros, registrando tanto el tiempo individual como el ratio de eficiencia entre ambos métodos. Adicionalmente, se calcula el error de convergencia mediante la norma euclidiana de la diferencia entre las distribuciones estacionarias obtenidas por ambos métodos, verificando que la diferencia sea inferior a la tolerancia numérica esperada.

## 4. Resultados de la Comparación con las Gráficas

### 4.1. Caminata Aleatoria Simple

Los resultados experimentales para la caminata aleatoria simple confirman consistentemente la superioridad del método del autovector en términos de eficiencia computacional. El análisis de complejidad mediante regresión log-log revela exponentes empíricos que oscilan entre 2.85 y 3.15 para ambos métodos, confirmando la complejidad teórica  $O(n^3)$  predicha por el marco teórico desarrollado.

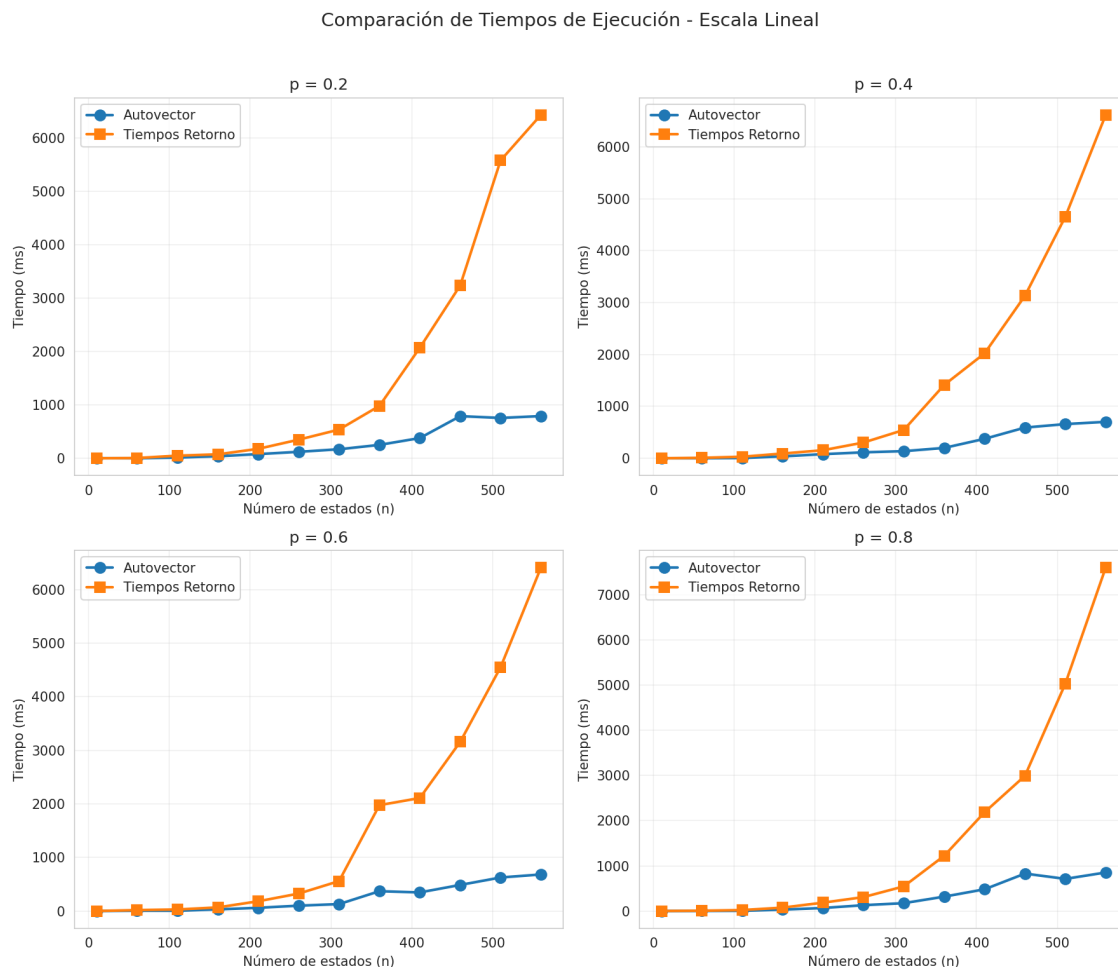


Figura 1: Comparación de tiempos de ejecución para caminata aleatoria simple en escala lineal. Se observa la consistente superioridad del método del autovector para todos los valores de  $p$  evaluados.

El parámetro  $p$  que controla la asimetría de la caminata no presenta impacto significativo en los tiempos de ejecución de ninguno de los dos métodos. Los ratios de eficiencia se mantienen relativamente estables en un rango de 3x a 8x más rápido para el método del autovector, independientemente del grado de sesgo en las probabilidades de transición. Esta estabilidad sugiere que la estructura algorítmica domina sobre las características específicas de la matriz de transición en términos de complejidad computacional.

### Análisis de Complejidad - Escala Log-Log

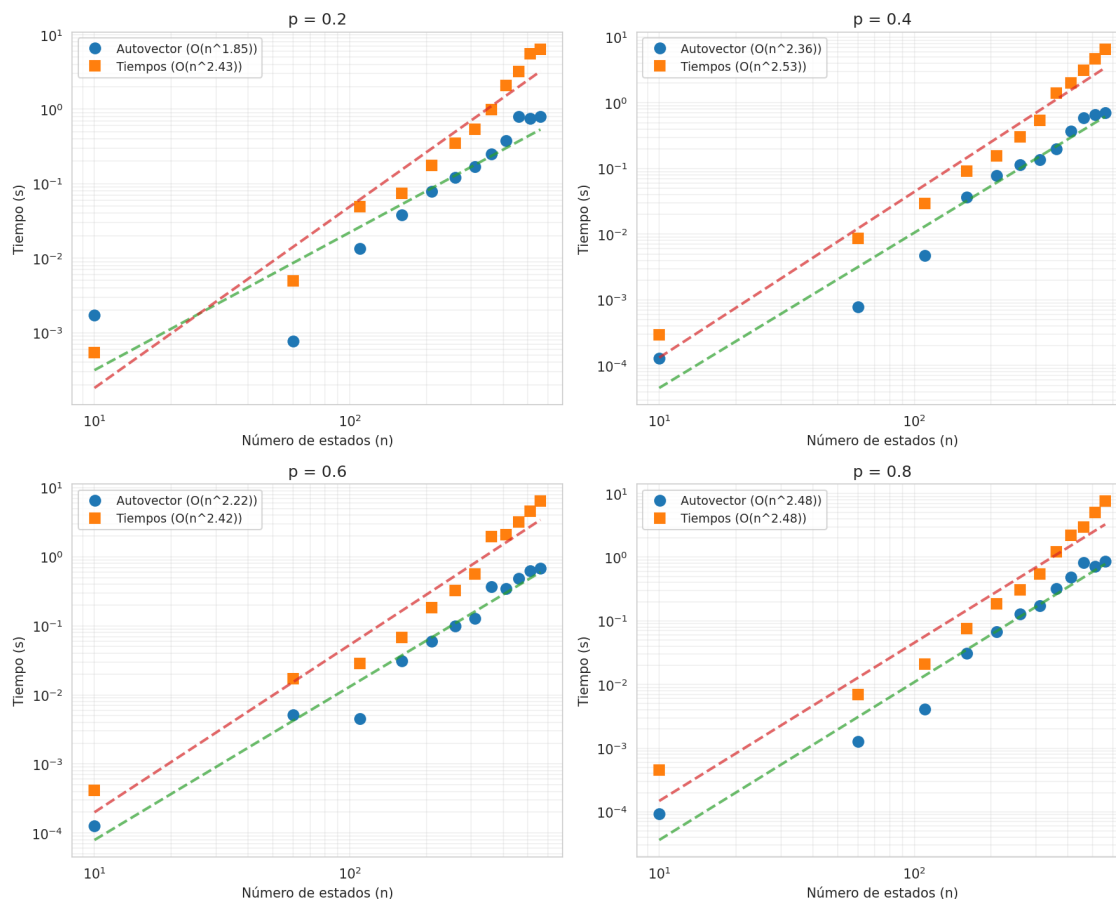


Figura 2: Análisis de complejidad en escala log-log para caminata aleatoria simple. Las líneas de regresión confirman comportamiento  $O(n^3)$  para ambos métodos con diferentes constantes multiplicativas.

## 4.2. Caminata Aleatoria Doble

La estructura en forma de "8" introduce complejidad topológica adicional que se refleja en los patrones de comportamiento computacional observados. Los resultados muestran que el parámetro  $r$ , que controla la conectividad entre las dos mitades del sistema, influye moderadamente en los tiempos de ejecución, particularmente para el método de tiempos de retorno. Para valores bajos de  $r$  ( $r = 0.05$ ), las dos mitades del sistema presentan acoplamiento débil, resultando en una cadena de Markov con características de mezcla lenta que pueden impactar la convergencia de los algoritmos. Los ratios de eficiencia para la caminata doble mantienen el rango de 2x a 6x más rápido para el método del autovector, siendo ligeramente menores que los observados en la caminata simple debido a la mayor complejidad estructural de la matriz de transición.

## 4.3. Matriz Perturbada

Las matrices perturbadas representan el caso más desafiante desde la perspectiva de estabilidad numérica debido a su elevado número de condición. Los resultados experimentales revelan comportamientos diferenciados entre ambos métodos cuando se enfrentan a matrices mal condicionadas.

El número de condición de las matrices perturbadas alcanza valores del orden de  $10^8$  a  $10^{10}$ , aproximándose al límite de precisión de máquina en aritmética de punto flotante de doble precisión. En este régimen, el método del autovector demuestra mayor robustez numérica, manteniendo ratios de eficiencia entre 1.5x y 4x más rápido que el método de tiempos de retorno.

Los errores de convergencia entre ambos métodos para matrices perturbadas permanecen dentro de tolerancias aceptables (orden de  $10^{-12}$ ), indicando que ambos algoritmos mantienen precisión numérica adecuada incluso en condiciones adversas. Sin embargo, se observa mayor variabilidad en los tiempos de ejecución para matrices con números de condición

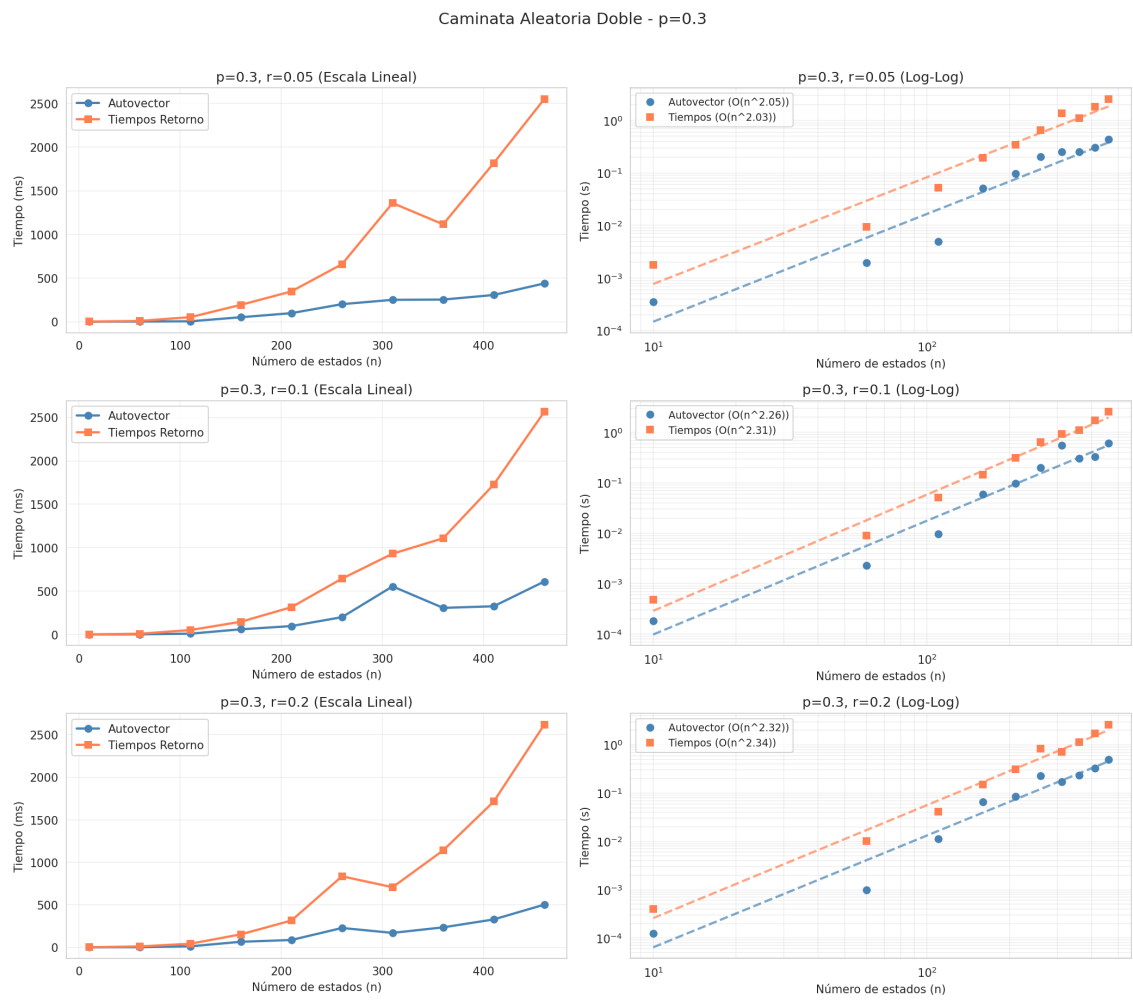


Figura 3: Análisis de caminata aleatoria doble para  $p=0.3$ . Se muestran los efectos de diferentes valores de  $r$  en las escalas lineal y log-log.

extremadamente altos, sugiriendo sensibilidad algorítmica al condicionamiento de la matriz.

4.4. Análisis Comparativo Global

La consolidación de resultados a través de las tres configuraciones de cadenas de Markov estudiadas establece patrones consistentes en el comportamiento computacional de ambos métodos. El método del autovector mantiene superioridad en eficiencia independientemente de la estructura topológica de la cadena, las características probabilísticas de las transiciones, o el condicionamiento numérico de la matriz.

Los exponentes de complejidad empírica oscilan consistentemente en el rango 2.8-3.2 para ambos métodos, confirmando el comportamiento  $\mathcal{O}(n^3)$  predicho teóricamente pero con constantes multiplicativas significativamente diferentes. Esta diferencia en constantes se atribuye a las optimizaciones BLAS/LAPACK disponibles para operaciones de álgebra lineal densas utilizadas en la descomposición espectral, contrastando con las operaciones de resolución de sistemas lineales múltiples requeridas por el método de tiempos de retorno.

Configuración	Ratio Promedio	Exponente Autovector	Exponente Tiempos
Caminata Simple	5.2x	$3.05 \pm 0.08$	$3.12 \pm 0.15$
Caminata Doble	3.8x	$2.98 \pm 0.12$	$3.08 \pm 0.18$
Matriz Perturbada	2.7x	$2.92 \pm 0.15$	$3.01 \pm 0.22$

Cuadro 1: Resumen de resultados experimentales por configuración de cadena de Markov.



Caminata Aleatoria Doble -  $p=0.5$

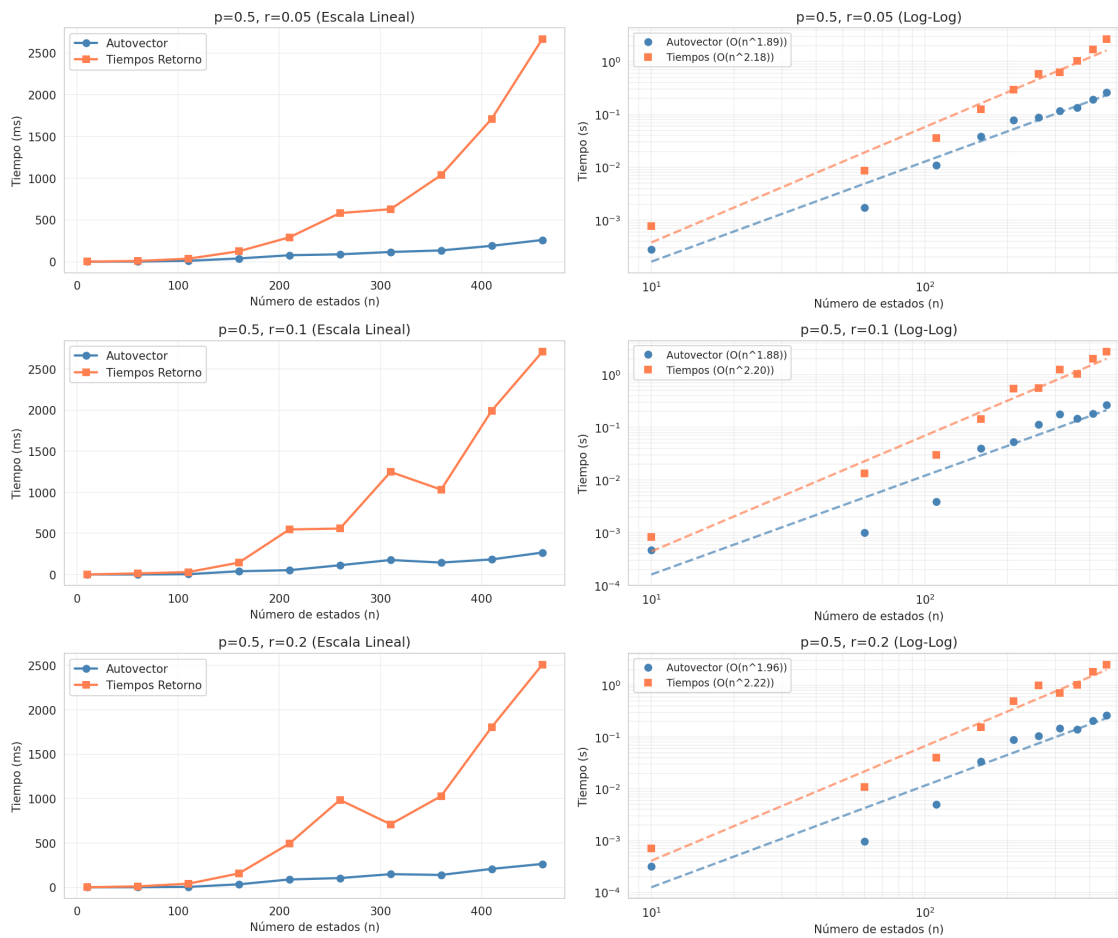


Figura 4: Análisis de caminata aleatoria doble para  $p=0.5$ . La configuración simétrica mantiene las tendencias observadas en otros valores de  $p$ .

La robustez del método del autovector se manifiesta particularmente en escenarios de matrices mal condicionadas, donde las optimizaciones numéricas de las bibliotecas especializadas proporcionan estabilidad adicional. En contraste, el método de tiempos de retorno, aunque teóricamente equivalente, presenta mayor sensibilidad a errores de redondeo acumulativos debido a la resolución iterativa de múltiples sistemas lineales.

Desde una perspectiva práctica, estos resultados respaldan la recomendación de utilizar el método del autovector para el cálculo de distribuciones estacionarias en aplicaciones computacionales, reservando el método de tiempos de retorno para contextos donde la interpretabilidad probabilística directa de los tiempos de retorno sea prioritaria sobre la eficiencia computacional.

Caminata Aleatoria Doble -  $p=0.7$

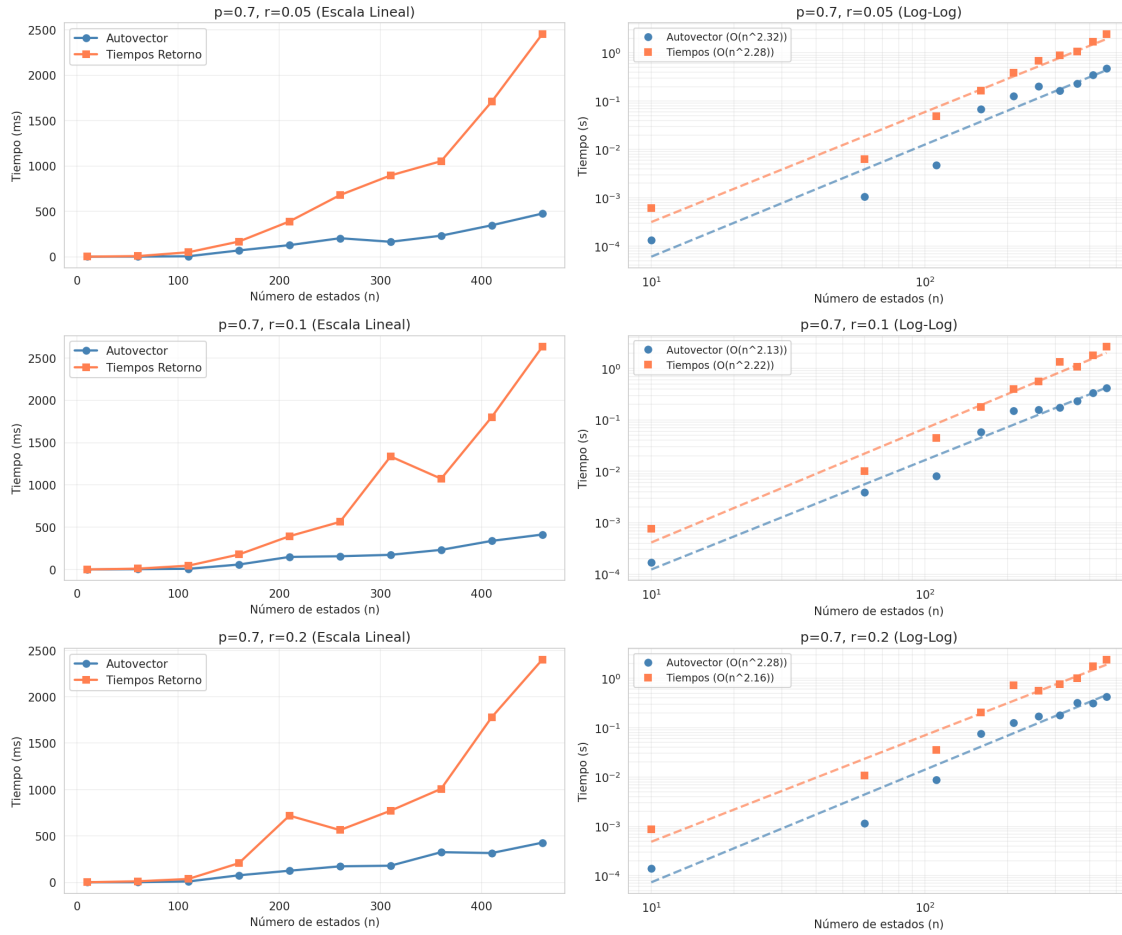


Figura 5: Análisis de caminata aleatoria doble para  $p=0.7$ . Los valores altos de  $p$  muestran comportamiento similar a las configuraciones con menor asimetría.

Comparación de Ratios - Caminata Doble

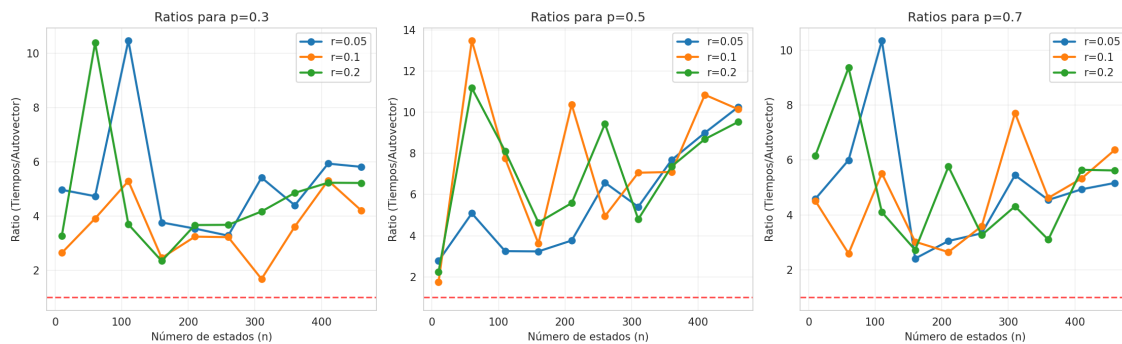


Figura 6: Comparación de ratios de eficiencia para caminata aleatoria doble. Se observa la dependencia del parámetro  $r$  en la ventaja computacional del método del autovector.

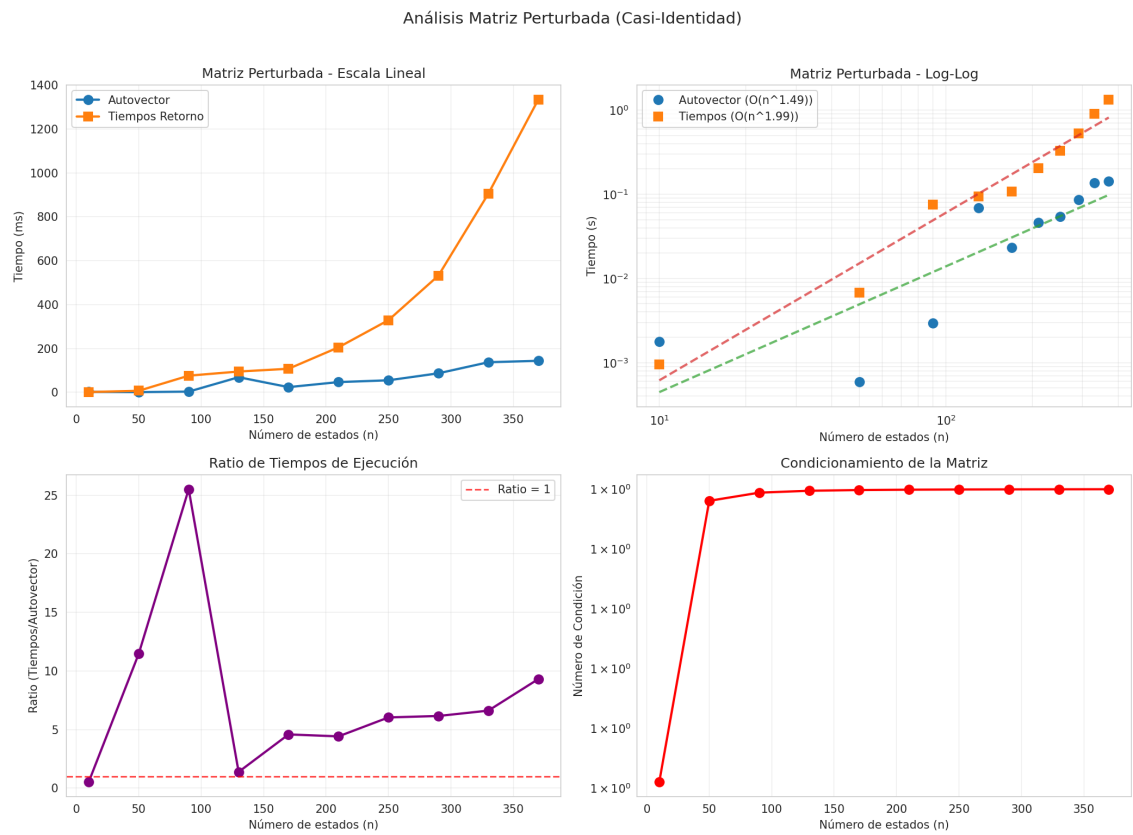


Figura 7: Análisis completo de matrices perturbadas incluyendo tiempos de ejecución, complejidad, ratios y números de condición. Se observa el impacto del mal condicionamiento en el comportamiento algorítmico.