Álgebra Lineal Computacional

2do Cuatrimestre 2025

Laboratorio N° 7: Matrices de transición, Markov.

Una matriz de $T \in \mathbb{R}^{n \times n}$ se dice de estocástica[1] si cumple con las condiciones

$$\sum_{i=1}^{n} T_{ij} = 1$$
$$T_{ij} \geq 0$$

para todo $1 \leq i, j \leq n$. Si partimos de un vector $v \in \mathbb{R}^n$ que representa cantidades (pueden ser probabilidades, poblaciones, niveles de llenado, entre otras cosas), tenemos que las componentes del resultado de aplicar T a v, $w_i = (Tv)_i = \sum_{j=1}^{n} T_{ij} v_j$. Es decir, el vector w resultante de aplicar T a un vector v puede pensarse como la proporción del flujo enviado desde la componente j hacia la componente i. Esto justifica la condición de normalización sobre cada columna de T. Otra propiedad interesante es que T preserva los totales: si $v_i \geq 0 \; \forall i$, entonces $||w||_1 = ||v||_1$. Esta condición puede interpretarse como *conservación del caudal*. Como última propiedad a mencionar, ocurre que los autovalores de T cumplen $1 \geq |\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$. Los autovectores con autovalor $\lambda = 1$ representan los *puntos fijos* o *estados estacionarios del sistema*.

En este laboratorio generaremos métodos para analizar las matrices de transición, y consideraremos algunas de sus aplicaciones.

Ejercicio 1 (Matrices de transición al azar)**.** Construyan las siguientes funciones

a) `transiciones_al_azar_continuas(n)` que genere una matriz de transición al azar de $n \times n$ con elemento reales. Una forma posible de hacer esto es:

- Generen n^2 números al azar y acomodenlos en forma de matriz. Los números deben ser generados en el intervalo $[0, 1]$
- Normalicen cada columna de forma tal que sume 1.

b) `transicion_al_azar_uniforme(n,thres)` que tenga probabilidades de transición uniformes para cada columna. Una forma de hacer esto es:

- Generen n^2 números al azar y acomodenlos en forma de matriz. Los números deben ser generados en el intervalo $[0, 1]$.
- Fijen en 1 todos los valores que sean menores a `thres` y en 0 los que sean mayores.
- Normalicen cada columna de forma tal que sume 1.

Ejercicio 2 (Estado estacionario y puntos fijos)**.** Para encontrar todos los puntos fijos de una matriz de transición T, la forma más directa es buscar el núcleo de $T - I$, con I la matriz identidad. De esta forma, obtenemos a la vez todos los autovectores con autovalor 1. Una forma robusta de hacer esto, se basa en reemplazar $T - I$ con $(T - I)^t (T - I)$ (les queda a ustedes demostrar que el núcleo de ambas transformaciones lineales es el mismo). Gracias a que $(T - I)^t (T - I)$ es una matriz simétrica, podemos aplicar el método de diagonalización basado en el método de la potencia y reflectores de Householder que desarrollamos el laboratorio pasado. Entonces,

- Construyan una función `nucleo(A,tol)` que encuentre los vectores en el núcleo de A mediante encontrar los autovectores con autovalor 0 (en realidad, de módulo menor a `tol`) de $A^t A$.

[1] A.k.a. *de Markov* o *de transición*

- Usando `nucleo(A,tol)`, armen `puntos_fijos(T,tol)` que encuentre los puntos fijos de una matriz de transición T. Esta función debe retornar vectores v tales que $||v||_1 = 1$.
- Exploren los puntos fijos de distintas matrices de transición. Grafiquen los resultados mediante gráficos de barras (`bar` de `matplotlib.pyplot`) para observar cuánto se concentra el flujo en una dada componente. Consideren los siguientes casos:
  - Matrices de transición continuas generadas al azar, en función de n
  - Matrices de transición uniformes, en función de n y $thres$.
  - Matrices de transición estilo *PageRank*: $T = (1-\alpha)T_0 + \alpha/NE$, con $E_{ij} = 1$ para todo i, j y T_0 una matrices de transición al azar uniforme, en función de α y $thres$ para $n = 100$.

Ejercicio 3 (Difusión en la grilla)**.** Una de las aplicaciones de las matrices de transición es estudiar procesos de difusión. Consideren una grilla cuadrada de $n \times n$ posiciones. Supongan que un caminante puede moverse de una celda a cualquiera de sus vecinas con igual probabilidad. Noten que la matriz en cuestión corresponde entonces a una matriz de las que denominamos *uniformes* previamente. Esta matriz $T \in R^{k \times k}$ con $k = n^2$. Entonces,

a) Construyan para valores de $n \leq 50$ (más grande es mejor, pero elijan en base al poder de cómputo disponible), una matriz T que representen el movimiento en la grilla. Para llevar las posiciones (i, j) en la grilla a un vector de R^k, pueden usar la transformación $k = n * i + j$ $(0 \leq i, j \leq n-1)$. Consideren que el caminante puede moverse desde (i, j) hacia las posiciones $(i \pm 1, j)$ y $(i, j \pm 1)$. Asuman condiciones de contorno periódicas, de forma tal que (i, n) equivale a $(i, 0)$ y (n, j) equivale a $(0, j)$.

b) Visualicen el la distribución de probabilidad usando la función `plot_grilla(v,n)` que acompaña este archivo. Consideren 10, 11, 100, 101, 1000, 1001, 10000 y 10001 pasos del caminante. ¿Llega el caminante a una distribución estable en el con el número de pasos? ¿Qué pueden concluir respecto a la existencia de P^∞, y los autovalores de la matriz de transición?

c) (**Ejercicio extra**) Con las matrices de transición resultantes, computen la distancia promedio recorrida en función de la cantidad de pasos para un caminante que comienza en la mitad de la grilla. Si luego de p pasos el caminante tiene probabilidad $w_k = (T^p v)_k$ de estar en la posición k, entonces la distancia promedio recorrida puede calcularse como $\sigma_d = \sqrt{\sum_k w_k (i_k^2 + j_k^2)}$, con $i_k = \lfloor k/n \rfloor$ y $j_k = k \mod n$.

Ejercicio 4 (Matrices ralas)**.** En muchos problemas de donde se estudian transiciones entre elementos, la principal dificultad es el número muy grande de estados posibles (como habrán notado en el punto anterior). Por ejemplo, al modelar el proceso de navegación entre página web, se debe trabajar con matrices de $n \approx 10^8$. Sin embargo, estos sistemas están muy estructurados, existiendo sólo unos pocos elementos distintos de cero para cada página (es decir, cada página web conecta únicamente con otras pocas páginas). Esto naturalmente lleva a buscar la representación de *matriz rala* o *sparse matrix*. En esta representación, se piensa a la matriz como un diccionario o lista indexada. La matriz es un conjunto de elementos de la forma $\{i : (j, a_{ij})\}$, donde i indica la fila en la que hay un elemento distinto de cero, j la columna en la que ese elemento es distinto de cero y a_{ij} el valor que se encuentra en ese elemento. De esta forma, si en promedio hay k páginas web alcanzables desde una página cualquiera, esta representación requiere guardar $3kn$ elementos. Una matriz normal requeriría n^2, haciendo que el ahorro vaya como $1 - 3k/n$, que aumenta con n. Con esto en mente:

a) Construyan una función `crea_rala(listado)` que reciba tres listas con elementos i, j y a_{ij} y retorne un diccionario de la forma $\{i : (j, a_{ij}\}$, que represente la matriz en formato ralo.

b) Construyan una función `multiplica_rala_vector(A,v)` que reciba una matriz rala generada con `crea_rala` y un vector genérico (lista o numpy.array unidimensional) y calcule el producto $w = Av$ entre ambos. El resultado debe ser un vector w de la misma clase que v.

c) Implementen el método de la potencia para matrices ralas `metpot_rala(A,v,maxiter,tol)` que aplique hasta `maxiter` iteraciones del método de la potencia a un vector dado v, o se detenga anticipadamente si la diferencia entre una iteración y la siguiente es menor a `tol`.

d) Repitan el ejercicio anterior, para grillas de tamaño más grande, por ejemplo $n = 100$ o $n = 1000$. Comparen para un mismo tamaño de la matriz el tiempo de corrida con un método y el otro.

e) (**Ejercicio extra**) Utilicen el método implementado para estudiar la difusión de un caminante aleatorio sobre la red de calles de la Ciudad de Autónoma de Buenos Aires. Analicen cuán lejos puede llegar el caminante luego de 100, 1000 y 10000 pasos dependiendo de su punto de partida inicial. En analogía con el punto anterior, en una red de calles pensamos a las intersecciones entre las mismas (las *esquinas*) como los centros de la grilla, y las calles que las conectan como las vias para moverse de una intersección a la otra. Para construir la matriz de transición, empleen los dos archivos que acompañan este documento: un listado de conexiones entre las intersecciones de las calles (`ejes.csv`), donde ya se indica el peso de cada conexión, y un listado de las posiciones de las intersecciones con su posición en el espacio (`intersecciones.csv`). Pueden apoyarse en el script `calles_tips.py` que acompaña este archivo ¿En qué región logra llegar más lejos el caminante de su punto de partida?

Módulo ALC

Para el módulo ALC, deben programar las siguientes funciones. Tests para las mismas acompañan en el archivo `L07-Tests.py`.

```
def transiciones_al_azar_continuas(n):
    """
    n la cantidad de filas (columnas) de la matriz de transici n.
    Retorna matriz T de n x n normalizada por columnas, y con entradas al azar en el
    intervalo [0,1]
    """

def transiciones_al_azar_uniforme(n,thres):
    """
    n la cantidad de filas (columnas) de la matriz de transici n.
    thres probabilidad de que una entrada sea distinta de cero.
    Retorna matriz T de n x n normalizada por columnas. El elemento i,j es distinto de
    cero si el n mero generado al azar para i,j es menor o igual a thres. Todos los
    elementos de la columna $j$ son iguales (a 1 sobre el n mero de elementos distintos
     de cero en la columna).
    """

def nucleo(A,tol=1e-15):
    """
    A una matriz de m x n
    tol la tolerancia para asumir que un vector esta en el nucleo.
    Calcula el nucleo de la matriz A diagonalizando la matriz traspuesta(A) * A (* la
    multiplicacion matricial), usando el medodo diagRH. El nucleo corresponde a los
    autovectores de autovalor con modulo <= tol.
    Retorna los autovectores en cuestion, como una matriz de n x k, con k el numero de
    autovectores en el nucleo.
    """

def crea_rala(listado,m_filas,n_columnas,tol=1e-15):
    """
    Recibe una lista listado, con tres elementos: lista con indices i, lista con indices
     j, y lista con valores A_ij de la matriz A. Tambien las dimensiones de la matriz a
    traves de m_filas y n_columnas. Los elementos menores a tol se descartan.
    Idealmente, el listado debe incluir unicamente posiciones correspondientes a valores
     distintos de cero. Retorna una lista con:
    - Diccionario {(i,j):A_ij} que representa los elementos no nulos de la matriz A. Los
     elementos con modulo menor a tol deben descartarse por default.
    - Tupla (m_filas,n_columnas) que permita conocer las dimensiones de la matriz.
    """

def multiplica_rala_vector(A,v):
    """
    Recibe una matriz rala creada con crea_rala y un vector v.
    Retorna un vector w resultado de multiplicar A con v
    """
```

Nota: no está permitodo el uso de la multiplicación matricial de numpy (`@`, `np.matmul`, etc)