Melissa Robles
Optimización estocástica
Universidad de los Andes
Bogotá, marzo de 2021

Taller 2

1. Introducción al problema

Se considera un objeto volador que se mueve en una retícula 199×199 bajo la influencia del viento. El problema consiste en llegar a la región $[155, 157] \times [155, 157]$ en el menor tiempo posible. Por lo tanto, definimos los estados posibles como puntos en la retícula:

$$s = \{(a, b) : 0 \leq a, b \leq 199\}$$

Y por cada estado, se consideran 4 acciones posibles: moverse hacia arriba u, hacia abajo d, hacia la izquierda l o a la derecha r. Por lo tanto, para todo $s \in S$,

$$A(s) = \{u, d, l, r\}$$

Las probabilidades de transición para los estados $s = (a, b)$ con $0 < a, b < 198$, están dadas por:

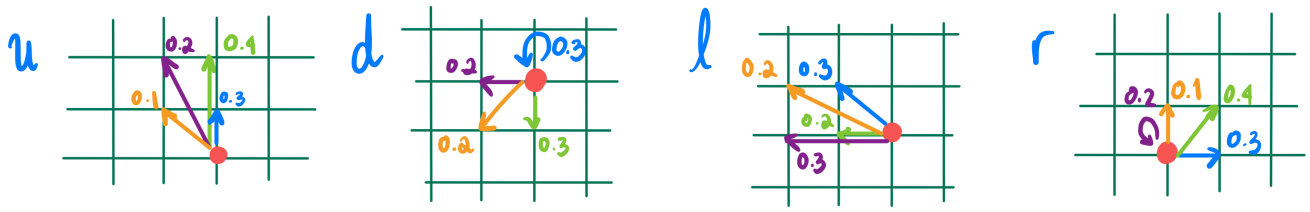
$$Q((a', b') \mid s, u) = \begin{cases} 0{,}3 & \text{si } a' = a, b' = b + 1 \\ 0{,}4 & \text{si } a' = a, b' = b + 2 \\ 0{,}2 & \text{si } a' = a - 1, b' = b + 2 \\ 0{,}1 & \text{si } a' = a - 1, b' = b + 1 \end{cases} \tag{1}$$

$$Q((a', b') \mid s, d) = \begin{cases} 0{,}3 & \text{si } a' = a, b' = b \\ 0{,}3 & \text{si } a' = a, b' = b - 1 \\ 0{,}2 & \text{si } a' = a - 1, b' = b \\ 0{,}2 & \text{si } a' = a - 1, b' = b - 1 \end{cases} \tag{2}$$

$$Q((a',b') \mid s,l) = \begin{cases} 0{,}3 & \text{si } a' = a-1, b' = b+1 \\ 0{,}2 & \text{si } a' = a-1, b' = b \\ 0{,}3 & \text{si } a' = a-2, b' = b \\ 0{,}2 & \text{si } a' = a-2, b' = b+1 \end{cases} \tag{3}$$

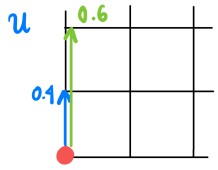
$$Q((a',b') \mid s,r) = \begin{cases} 0{,}3 & \text{si } a' = a+1, b' = b \\ 0{,}4 & \text{si } a' = a+1, b' = b+1 \\ 0{,}2 & \text{si } a' = a, b' = b \\ 0{,}1 & \text{si } a' = a, b' = b+1 \end{cases} \tag{4}$$

Estas probabilidades se pueden ver gráficamente de la siguiente manera:



Notemos que estas probabilidades no cuentan los casos en que $x = 0, 1$ o $y = 0, 1$. Veamos qué sucede en esos casos. Las probabilidades $Q((a',b')|s,X)$ para los $X \in \{u,d,l,r\}$ que no se enuncien, son los mismos que en (1), (2), (3) o (4).

- Si $a = 0, b = 0$:



$$Q((a',b') \mid s,u) = \begin{cases} 0{,}4 & \text{si } a' = a, b' = b+1 \\ 0{,}6 & \text{si } a' = a, b' = b+2 \end{cases}$$

$$Q((a',b') \mid s,d) = \{ \ 1 \quad \text{si } a' = a, b' = b$$

$$Q((a',b') \mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b+1 \\ 0{,}5 & \text{si } a' = a, b' = b \end{cases}$$

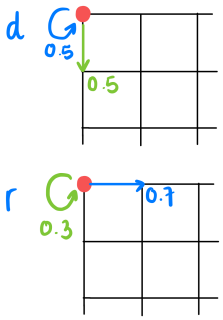
- Si $a = 0, b = 199$:

$$Q((a',b')\,|\,s,u) = \left\{ 1 \quad \text{si } a' = a, b' = b \right.$$

$$Q((a',b')\,|\,s,d) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b-1 \end{cases}$$

$$Q((a',b')\,|\,s,l) = \left\{ 1 \quad \text{si } a' = a, b' = b \right.$$

$$Q((a',b')\,|\,s,r) = \begin{cases} 0{,}7 & \text{si } a' = a+1, b' = b \\ 0{,}3 & \text{si } a' = a, b' = b \end{cases}$$

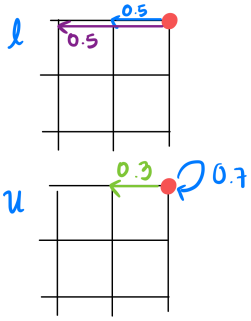


- Si $a = 199, b = 199$:

$$Q((a',b')\mid s,u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b \\ 0{,}7 & \text{si } a' = a-1, b' = b \end{cases}$$

$$Q((a',b')\mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a-1, b' = b \\ 0{,}5 & \text{si } a' = a-2, b' = b \end{cases}$$

$$Q((a',b')\,|\,s,r) = \left\{ 1 \quad \text{si } a' = a, b' = b \right.$$

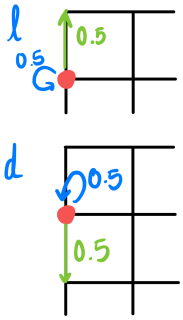


- Si $a = 0, b = 198$:

$$Q((a',b')\mid s,u) = \left\{ 1 \quad \text{si } a' = a, b' = b+1 \right.$$

$$Q((a',b')\mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b+1 \end{cases}$$

$$Q((a',b')\mid s,d) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b-1 \end{cases}$$

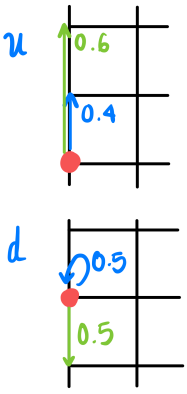


- Si $a = 0, 0 < b < 198$:

$$Q((a',b')\mid s,u) = \begin{cases} 0{,}6 & \text{si } a' = a, b' = b+2 \\ 0{,}4 & \text{si } a' = a, b' = b+1 \end{cases}$$

$$Q((a',b')\mid s,d) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b-1 \end{cases}$$

$$Q((a',b')\mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b+1 \end{cases}$$

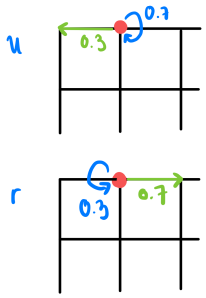


- Si a = 1, b = 199:

$$Q((a',b') \mid s,u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b \\ 0{,}3 & \text{si } a' = a-1, b' = b \end{cases}$$

$$Q((a',b') \mid s,l) = \begin{cases} 1 & \text{si } a' = a-1, b' = b \end{cases}$$

$$Q((a',b') \mid s,r) = \begin{cases} 0{,}3 & \text{si } a' = a, b' = b \\ 0{,}7 & \text{si } a' = a+1, b' = b \end{cases}$$

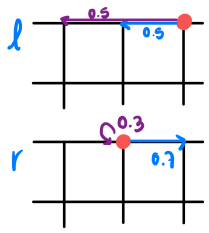


- Si $1 < a < 199$, $b = 199$:

$$Q((a',b') \mid s,u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b \\ 0{,}3 & \text{si } a' = a-1, b' = b \end{cases}$$

$$Q((a',b') \mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a-1, b' = b \\ 0{,}5 & \text{si } a' = a-2, b' = b \end{cases}$$

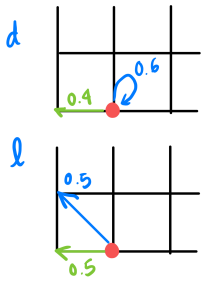
$$Q((a',b') \mid s,r) = \begin{cases} 0{,}3 & \text{si } a' = a, b' = b \\ 0{,}7 & \text{si } a' = a+1, b' = b \end{cases}$$



- Si $a = 1, b = 0$:

$$Q((a',b') \mid s,d) = \begin{cases} 0{,}6 & \text{si } a' = a, b' = b \\ 0{,}4 & \text{si } a' = a-1, b' = b \end{cases}$$

$$Q((a',b') \mid s,l) = \begin{cases} 0{,}5 & \text{si } a' = a-1, b' = b+1 \\ 0{,}5 & \text{si } a' = a-1, b' = b \end{cases}$$



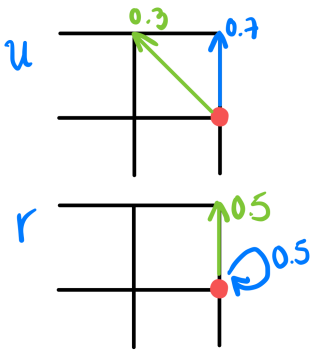
- Si $1 < a < 199$ y $b = 0$:

$$Q((a',b') \mid s,d) = \begin{cases} 0{,}6 & \text{si } a' = a, b' = b \\ 0{,}4 & \text{si } a' = a-1, b' = b \end{cases}$$

- Si $a = 199$ y $b = 198$:

$$Q((a',b') \mid s,u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b+1 \\ 0{,}3 & \text{si } a' = a-1, b' = b+1 \end{cases}$$

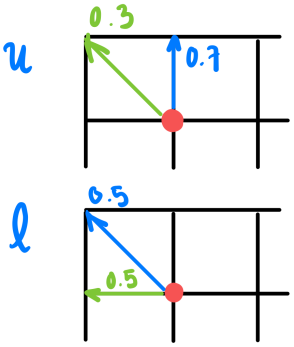
$$Q((a',b') \mid s,r) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b+1 \end{cases}$$



- Si $a = 199$ y $0 < b < 198$

$$Q((a', b') \mid s, r) = \begin{cases} 0{,}5 & \text{si } a' = a, b' = b \\ 0{,}5 & \text{si } a' = a, b' = b + 1 \end{cases}$$

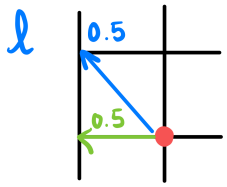
- Si $a = 1$ y $b = 198$:



$$Q((a', b') \mid s, u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b + 1 \\ 0{,}3 & \text{si } a' = a - 1, b' = b + 1 \end{cases}$$

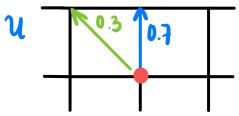
$$Q((a', b') \mid s, l) = \begin{cases} 0{,}5 & \text{si } a' = a - 1, b' = b + 1 \\ 0{,}5 & \text{si } a' = a - 1, b' = b \end{cases}$$

- Si $a = 1$ y $0 < b < 198$:



$$Q((a', b') \mid s, l) = \begin{cases} 0{,}5 & \text{si } a' = a - 1, b' = b + 1 \\ 0{,}5 & \text{si } a' = a - 1, b' = b \end{cases}$$

- Si $1 < a < 199$ y $b = 198$:



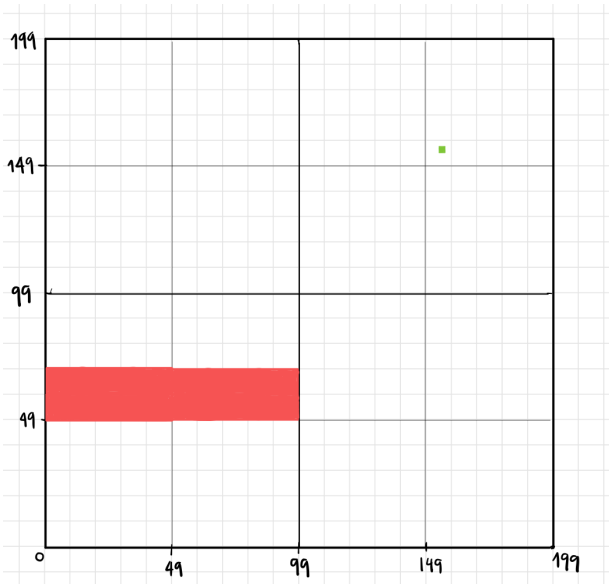
$$Q((a', b') \mid s, u) = \begin{cases} 0{,}7 & \text{si } a' = a, b' = b + 1 \\ 0{,}3 & \text{si } a' = a - 1, b' = b + 1 \end{cases}$$

El resto de casos tienen las probabilidades dadas en (1),(2),(3) y (4).

Adicionalmente, se define una zona que el objeto quiere evitar. Esta zona ocupa el 5 % del retículo, es decir, la conforma 2000 puntos del retículo. La zona va a ser la siguiente:

$$[0, 99] \times [49, 69]$$

El siguiente diagrama muestra las ubicaciones de la zona que se quiere evitar (en rojo), y la zona a donde se quiere llegar (en verde).



Como lo que queremos lograr es llegar a la región objetivo, definiremos las recompensas como distancias. En principio, para $s = (a, b) \in S$ y $X \in \{u, d, l, r\}$:

$$r(s, X) = - \sum_{(a',b') \text{ posibles}} Q((a', b') \mid s, X) \cdot \min_{(ro_x, ro_y) \in [155,157] \times [155,157]} ||(ro_x, ro_y) - (a', b')|| \qquad (5)$$

Esto sería en el caso en que ninguno de los (a', b') posibles estén en la región prohibida. Si este es el caso, y uno de los puntos posibles $(a', b') \in [0, 99] \times [49, 69]$, entonces $r(s, X) = -500$. Esto para garantizar que no se escoja esa acción.

Nota: El $-$ en la ecuación es para poder volver el problema de minimizar distancias, uno de maximizar.

1.1. Problema descontado de horizonte infinito

Como un problema descontado de horizonte infinito, queremos maximizar

$$\mathbb{E}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(X_t, Y_t)\right]$$

En este caso, tomaremos inicialmente $\lambda = 0{,}85$ y después analizaremos la sensibilidad del cambio de esta constante. Resolveremos este problema utilizando iteración por política.

1.2. Problema de optimización lineal

Como un problema de optimización lineal, podemos verlo como

$$\text{Maximizar} \sum_{s \in S} \alpha(s) v(s)$$

Sujeto a

$$v(s) - \sum_{j \in S} \lambda p(j|s,a) v(j) \geq r(s,a) \quad \forall s = (x,y) \in S, \forall a \in \{u, d, l, r\}$$

Las restricciones que debe cumplir la función α son:

$$\alpha(s) > 0 \quad \text{y} \quad \sum_{s \in S} \alpha(s) = 1$$

Como la distribución de los estados en la cuadrícula es uniforme, utilizaremos $\alpha(s) = \frac{1}{|S|}$ para todo estado s, es decir

$$\alpha(s) = \frac{1}{40000}$$

Este problema se resolverá con el algoritmo Simplex.

2. Implementación

Inicialmente, para evitar cálculos repetitivos, se creó una arreglo tres dimensional de $200 \times 200 \times 4$ llamado *recompensas*. Este arreglo guarda en la entrada (i, j, k), la recompensa de estar en el estado $s = (i, j)$ y tomar la acción k. Se usó la fórmula (5) y las restricciones para las áreas prohibidas. Para llenar esta matriz de recompensas, se utilizaron cinco funciones auxiliares. La primera de estas calcula la distancia de un punto a la región objetivo:

```
#Dado un estado (x,y) la función retorna su distancia mínima
# al cuadrado objetivo.
def minDistanciaAObjetivo(x,y):
    dis = 300
    for i in range(155,158):
        for j in range(155, 158):
            n1 = norm( array([x,y]) - array([i,j]) )
            if( n1 < dis ):
                dis = n1
    return dis
```

Figura 1: Función auxiliar: distancia mínima a la región óptima

Las otras cuatro funciones auxiliares *recU(a,b)*, *recD(a,b)*, *recL(a,b)* y *recR(a,b)* calculan para un estado $s = (a, b)$ la recompensa de tomar un estado u, d, l o r. La implementación de *recD(a,b)*, *recL(a,b)* y *recR(a,b)* son muy similares a *recU(a,b)* pero con los casos que se vieron en la Sección 1.

```
def recU(a,b):
    dist= minDistanciaAObjetivo(a,b)
    rta = -500

    #Tengo seis casos posibles:
    if( a == 0 and b == 199 ):
        rta = -dist
    elif ( a == 0 and b == 198 ):
        rta = -minDistanciaAObjetivo(a,b+1)
    elif ( a == 0 and b < 198 ):
        if a <= 99 and b <= 69 and b >= 49:
            rta = -500
        else:
            rta = -(0.4*minDistanciaAObjetivo(a,b+1) + 0.6*minDistanciaAObjetivo(a,b+2))
    elif ( a > 0 and b == 199 ):
        rta = -(0.3*minDistanciaAObjetivo(a-1,b) + 0.7*dist)
    elif( a > 0 and b == 198 ):
        rta = -(0.3*minDistanciaAObjetivo(a-1,b+1) + 0.7*minDistanciaAObjetivo(a,b+1))
    else:
        if a <= 99 and b <= 69 and b >= 49:
            rta = -500
        else:
            rta = -(0.3*minDistanciaAObjetivo(a,b+1) + 0.4*minDistanciaAObjetivo(a,b+2) +
                0.2*minDistanciaAObjetivo(a-1,b+2) + 0.1*minDistanciaAObjetivo(a-1,b+1))
    return rta
```

Figura 2: Función auxiliar: calcula $r((a, b), u)$

Con estas funciones auxiliares se llenó la matriz de recompensas:

```
#Itero sobre toda la matriz
for a in range(0,200):
    for b in range(0,200):
        #Para cada entrada de la matriz calculo las posibles recompensas

        #Primero considero la acción u.
        recompensas[a,b,0] = recU(a,b)

        #Ahora considero la acción d.
        recompensas[a,b,1] = recD(a,b)

        #Ahora considero la acción l.
        recompensas[a,b,2] = recL(a,b)

        #Ahora considero la acción r.
        recompensas[a,b,3] = recR(a,b)
```

Figura 3: Llena la matriz de recompensas.

2.1. Iteración por política

Para la implementación de este método, se debía crear para cada dn a considerar la matriz P_{dn} de tamaño 40000×40000. Como la gran mayoría de las entradas de la matriz son 0's, se utilizó la librería *Scipy* para manejar la matriz como dispersa.

Para poder ver las políticas como un arreglo, se utilizó la siguiente conversión:

$$dn[k] = \text{ acción para el estado } \left(\left\lfloor \frac{k}{200} \right\rfloor , k - \left\lfloor \frac{k}{200} \right\rfloor \cdot 200 \right)$$

Y, por lo tanto, dado un estado $s = (i, j)$, la acción a tomar bajo d_n sería:

$$dn[i \cdot 200 + j]$$

Para el cálculo de esa matriz, se creó una función auxiliar *calculo_Pdn_y_rdn(dn, lamb)* que tiene como input un arreglo $dn \in D$ de tamaño 40000 y $lamb \in (0, 1)$, y retorna la matriz $(I - \lambda P_{dn})$ y el arreglo r_{dn}. La implementación de este método se puede encontrar en anexos.

Otra función auxiliar que se utilizó fue *Psa (i,j,a,lamb)* que calcula la fila $(i * 200 + j)$-ésima de la matriz P_{d_a}, donde $d_a = [a, a, \ldots, a]$.

Finalmente, la implementación del ciclo principal es:

```
lamb = 0.9
#Acá se guardará d_n
d0 = np.zeros(40000)
#Acá se guardará d_n+1
dn = np.zeros(40000)
#Condición de parada
parar = False
iteraciones = 1

while( parar == False ):
    dn = np.zeros(40000)
    #Calculamos qué es P_dn y r_dn
    (P_dn,r_dn) = calculo_Pdn_y_rdn(d0,lamb)
    #Pasamos P_dn a una matriz csr pues es más rápida para hacer cálculos.
    P_dn = csr_matrix(P_dn)
    #Quiero resolver el sistema de ecuacones (P_dn)v = r_dn.
    v_n = scipy.sparse.linalg.spsolve(P_dn,r_dn)

    #Cálculo de d_n+1
    #Iteramos sobre todos los estados posibles
    for i in range(0,200):
        for j in range(0,200):
            val = np.zeros(4)
            #Itero sobre las posibles acciones
            for a in range(0,4):
                val[a] = recompensas[i,j,a] + np.dot( Psa(i,j,a,lamb) , v_n )
            valorMaximo = np.max(val)
            if( val[ int(d0[i*200+j]) ] == valorMaximo ):
                dn[i*200+j] = d0[i*200+j]
            else:
                dn[i*200+j] = np.argmax(val)
    #Condición de parada:
    parar = (dn==d0).all()
    d0 = dn
    iteraciones += 1
    #Fin del ciclo.
print(dn)
```

Figura 4: Implementación de iteración por política.

2.2. Implementación: Problema de optimización lineal

Para este caso se utilizó el paquete Scipy.linprog. Para utilizarlo se debía tener dos arreglos y una matriz:

- *obj*: Arreglo de tamaño 40000 que contiene en cada entrada $\alpha(j)$. En este caso es un arreglo donde todas sus entradas son $\frac{1}{40000}$.
- *lhs_ ineq*: Matriz de tamaño 160000×40000 donde cada fila tiene los coeficientes de los $v(j)$ en el lado izquierdo de las desigualdades

$$v(s)(1 - \lambda p(s|s,a)) - \sum_{j \neq s} \lambda p(j|s,a)v(j) \geq r(s,a) \quad \forall s = (x,y) \in S, \forall a \in \{u,d,l,r\}$$

  Notemos que son las mismas entradas que las matrices $(I - \lambda P_d)$.

- *rhs_ ineq*: Arreglo de tamaño 160000 que contiene para todo $s \in S$ y $a \in A$, el lado derecho de la anterior desigualdad.

```
lamb = 0.85
obj = 1/40000 * np.ones(40000)

#Primero agrego el lado izquierdo de las
#desigualdades con acción up para todo estado s.
d = np.zeros(40000)
(Pd, rd) = calculo_Pdn_y_rdn(d, lamb)
lhs_ineq = csr_matrix( Pd )

for i in range(1,4):
    d = np.full(40000, i)
    (Pd, rd) = calculo_Pdn_y_rdn(d, lamb)
    conv = csr_matrix(Pd)
    lhs_ineq = scipy.sparse.vstack( (lhs_ineq, conv) )

rhs_ineq = np.zeros(160000)
for a in range(0,4):
    for i in range(0,200):
        for j in range(0,200):
            rhs_ineq[ a*40000 + i*200 +j ] = recompensas[i,j,a]

opt = scipy.optimize.linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq, bounds = (-8000,80),options={'sparse': True} )
```

Figura 5: Implementación Linear Programming.

```
vn = opt.x

print(opt)

dn = np.zeros(40000)

#Calculamos política óptima
#Iteramos sobre todos los estados posibles
for i in range(0,200):
    for j in range(0,200):
        val = np.zeros(4)
        #Itero sobre las posibles acciones
        for a in range(0,4):
            val[a] = recompensas[i,j,a] + np.dot( Psa(i,j,a,lamb) , vn )
        dn[i*200+j] = np.argmax(val)

visualizarD( dn )
```

Figura 6: Implementación Linear Programming.

3. Resultados numéricos

3.1. Iteración por política

Al correr el algoritmo de Iteración por política con $\lambda = 0.85$, se obtuvieron los siguientes resultados:

Tiempo de ejecución	90.71 seg
Número de iteraciones	10 iteraciones
Tiempo promedio por iteración	9.071 seg

La política óptima se puede ver en la siguiente gráfica, donde los colores representan las acciones a tomar en ese estado:

$$u = 0 \qquad d = 1 \qquad l = 2 \qquad r = 3.$$

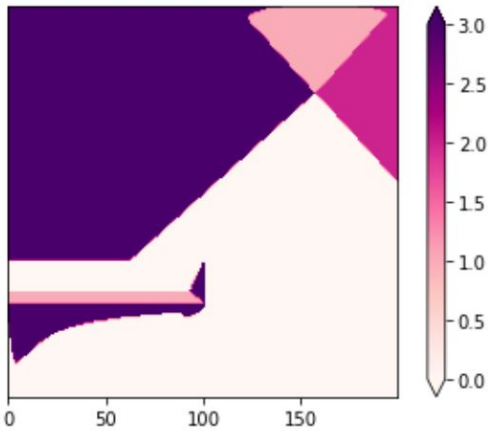


Figura 7: Resultado Iteración por política.

Para la simulación de 100 trayectorias para dos estados iniciales, se tomaron los puntos

$$(50, 25) \text{ y } (50, 100)$$

Los resultados obtenidos fueron los siguientes:

Para $(50, 25)$:

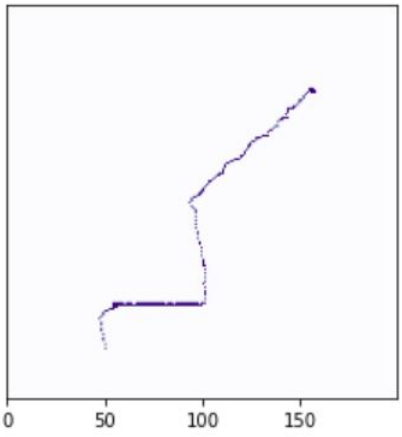


Figura 8: Simulación de una trayectoria comenzando en $(50, 25)$.

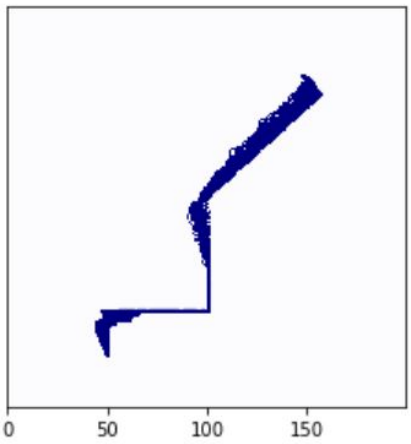


Figura 9: Simulación de 100 trayectorias comenzando en $(50, 25)$.

Para $(50, 200)$:

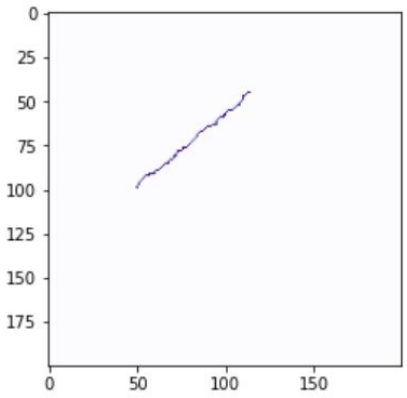


Figura 10: Simulación de una trayectoria comenzando en $(50, 100)$.

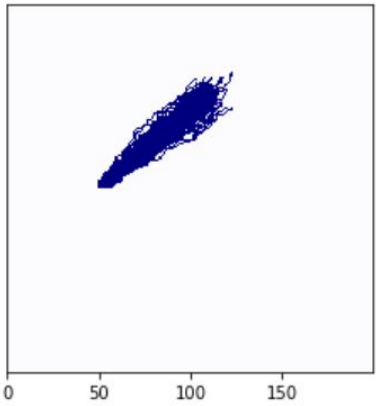


Figura 11: Simulación de 100 trayectorias comenzando en $(50, 100)$.

3.2. Optimización lineal

Al correr el algoritmo con $\lambda = 0{,}85$, se obtuvieron los siguientes resultados:

Tiempo de ejecución: 408.05 seg.

Política obtenida:

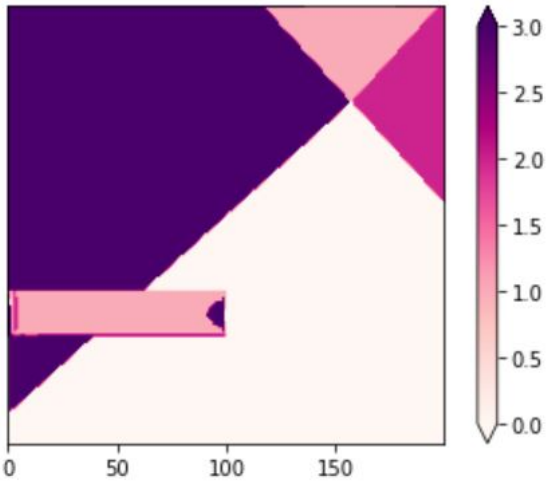


Figura 12: Gráfica de la política óptima obtenida.

Ahora, cambiamos la ditribución inicial de los α para ver cómo cambia la política obtenida. En este

caso se tomó para $s = (a, b)$:

$$\alpha'(s) = \begin{cases} \frac{1}{8} & \text{si } s \in \{(50, 50), (50, 51)\} \\ \frac{3}{4 \times 39998} & \text{de lo contrario} \end{cases}$$

Y los resultados fueron:

Tiempo de ejecución: 515.92 seg.

Política obtenida:

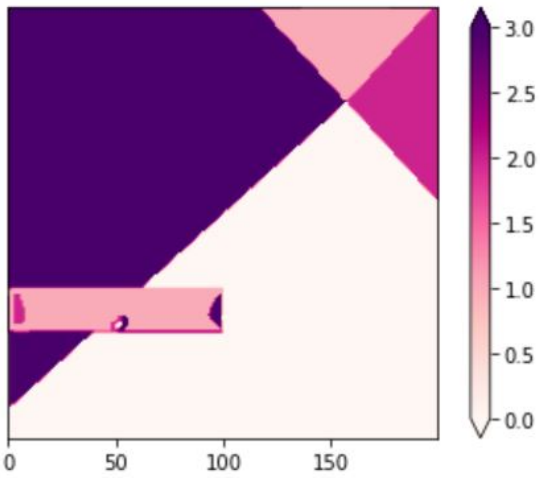


Figura 13: Gráfica de la política óptima obtenida para α'.

3.3. Sensibilidad de λ

Para estudiar la sensibilidad del λ se eligió utilizar el método Iteración por política, pues para este no se utilizó una librería sino se realizó la implementación directamente. Además, se puede hacer una comparación también por número de iteraciones. Se realizó un análisis para los siguientes valores de λ:

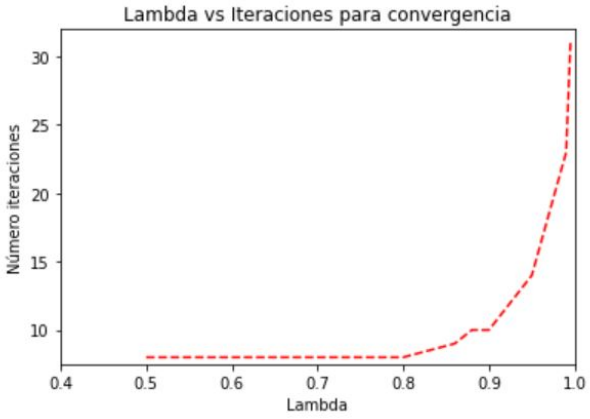
0.5, 0.7, 0.8, 0.86, 0.88, 0.9, 0.95, 0.99 y 0.995

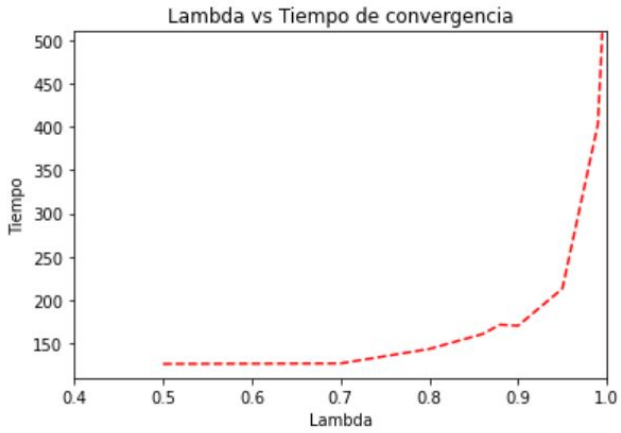
Al igual que para el taller pasado, se realizó una comparación por número de iteraciones, tiempo de convergencia y política obtenida.

λ	0.5	0.7	0.8	0.86	0.88
Iteraciones	8	8	8	9	10
Tiempo ejecución (seg)	126.16	126.70	143.37	160.60	171.48

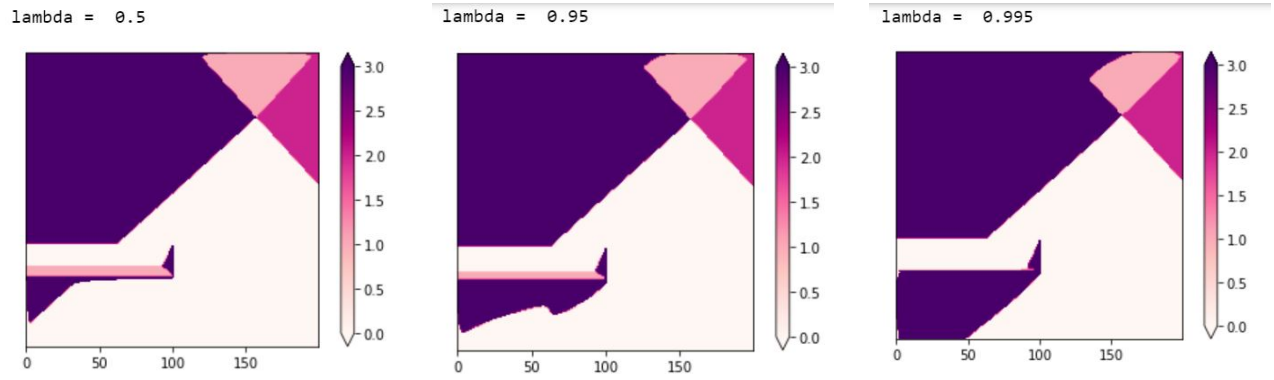
λ	0.9	0.95	0.99	0.995
Iteraciones	10	14	23	31
Tiempo ejecución (seg)	170.18	212.99	402.47	508.49

El comportamiento que muestran de tablas anteriores se puede evidenciar en las siguientes gráficas.



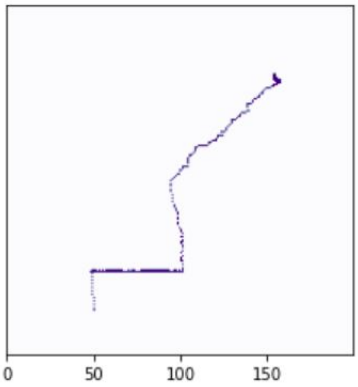


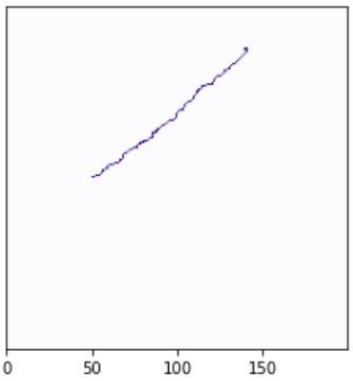
Además de los efectos del cambio de λ en el tiempo de convergencia y el número de iteraciones, el resultado de la política óptima también cambió. A continuación se muestran tres diferentes políticas obtenidas. Las demás se pueden encontrar en anexos.



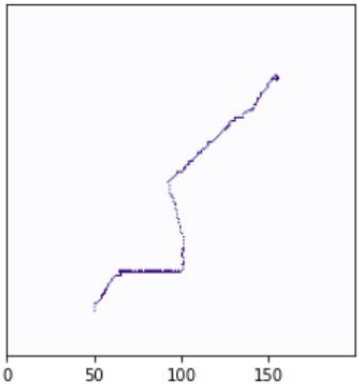
Las trayectorias que se analizaron en la Sección 3.1 con estos resultados son:

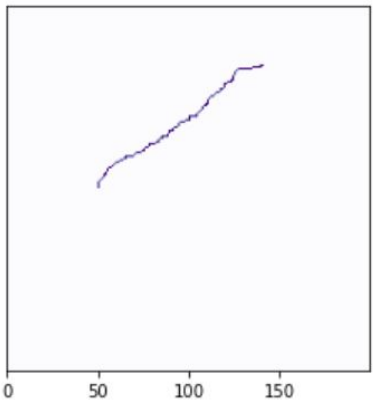
- $\lambda = 0{,}5$:



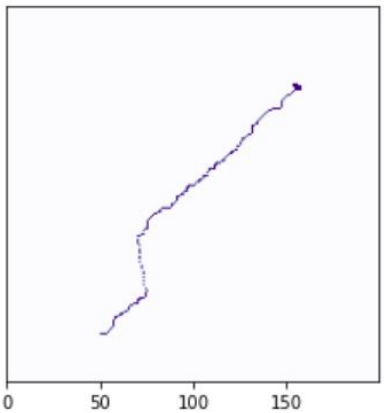


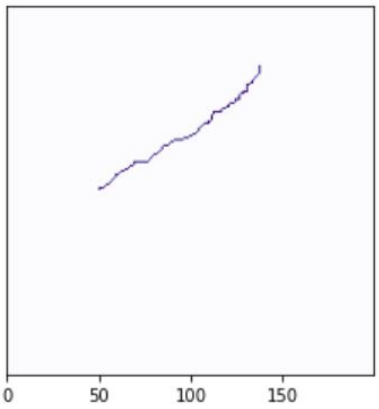
- $\lambda = 0{,}95$:





- $\lambda = 0{,}995$:





4. Discusión de resultados y conclusiones

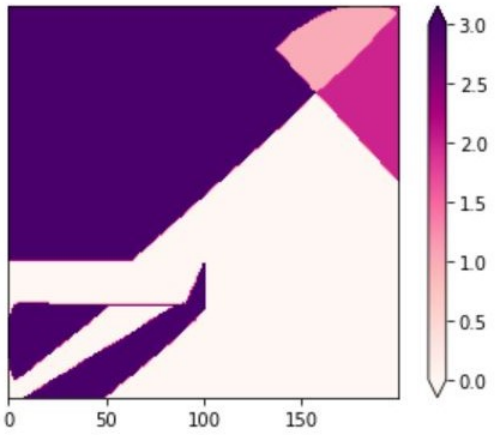
De las políticas obtenidas con los dos métodos analizados, podemos ver que en principio son coherentes con el problema. Se pueden evidenciar cuatro zonas divididas por una X, donde el centro es la zona a la que se quiere llegar. Los puntos más evidentes donde las políticas óptimas difieren en cualquiera de los métodos, son aquellos en las zonas *prohibidas*.

En principio, aunque ambos métodos retornaran políticas acordes al problema inicial, se puede evidenciar que el método Iteración por política retorna un resultado más preciso: en los puntos al borde superior de la zona *prohibida*, no tendría sentido que la política óptima fuera d (lo que se ve en la gráfica como rosado).

También se evidenció un mayor rendimiento con el método Iteración por política, pues el mayor tiempo de ejecución (con $\lambda = 0{,}995$) fue de 508.49 seg., mientras que viendo el problema como un problema de optimización lineal, el tiempo de ejecución máximo fue de 515.92 seg. con $\lambda = 0{,}85$ y α' como distribución elegida.

Para la solución con programación lineal, se evidenció que la distribución α escogida influye un poco en la política óptima escogida, pero en gran medida influye en el rendimiento del algoritmo. Cuando se cambió la distribución de S aumentó el tiempo de ejecución en más de 25 %.

Al igual que en el taller anterior, se pudo evidenciar que el cambio de λ afecta significativamente tanto en la política obtenida, como en el tiempo de ejecución del algoritmo y el número de iteraciones necesarias. En las gráficas *Lambda vs Iteraciones para convergencia* y *Lambda vs Tiempo de convergencia* se puede evidenciar el comportamiento de esto. El cambio que se evidenció en la política obtenida a partir del cambio del valor del λ, se ve principalmente en las zonas prohibidas y en las cercanas a estas. Esto afecta principalmente las trayectorias cuando el punto de partida está debajo de la zona prohibida, como se puede evidenciar en las gráficas de las trayectorias en la Sección 3.3. Para ver si la política óptima se estabiliza al igual que en el taller pasado, se corrió la política con valores aún más grandes de λ. Para valores de $\lambda = 0.999$, 0.9999, 0.99999 y 0.999999 se obtuvo el mismo resultado para la política óptima, que es la siguiente:



Podemos entonces concluir que, al igual que el taller pasado, después de un valor de λ fijo, la política se estabiliza.

5. Anexos

5.1. Implementación

```
def calculo_Pdn_y_rdn (dn, lamb):
    #Matrix a retornar. La inicializamos como la matriz identidad
    identidad = scipy.sparse.identity(40000)
    P_dn = lil_matrix(identidad)
    #Inicializamos r_dn como un arreglo de 0's.
    r_dn = np.zeros(40000)
    #Iteramos sobre todos los estados posibles para llenar por columnas la matriz
    #En P_dn vamos a guardar realmente I - lambda P_dn
    for i in range(0,200):
        for j in range (0,200):
            accion = dn[i*200+j]
            index = i*200+j

            #Para r_dn:
            r_dn[index] = recompensas[i,j,int(accion)]

            #Para P_dn:

            #Si la acción es up
            if( accion == 0 ):
                if ( i == 0 and j < 198 ):
                    P_dn[index,i*200+(j+1)] = - lamb*0.4
                    P_dn[index,i*200+(j+2)] = - lamb*0.6
                elif ( i > 0 and j == 199 ):
                    P_dn[index,(i-1)*200+j] = - lamb*0.3
                    #Está en la diagonal
                    P_dn[index,index] = 1- lamb*0.7
                elif( i > 0 and j == 198 ):
                    P_dn[index,(i-1)*200+(j+1)] = - lamb*0.3
                    P_dn[index,i*200+(j+1)] = - lamb*0.7
                elif( i == 0 and i == 199 ):
                    #Está en la diagonal
                    P_dn[index,index] = 1 - lamb
                elif ( i == 0 and j == 198 ):
                    P_dn[index,i*200+(j+1)] = - lamb
                else:
                    P_dn[index,i*200+(j+1)] = - lamb*0.3
                    P_dn[index,i*200+(j+2)] = - lamb*0.4
                    P_dn[index,(i-1)*200+(j+2)] = - lamb*0.2
                    P_dn[index,(i-1)*200+(j+1)] = - lamb*0.1
```

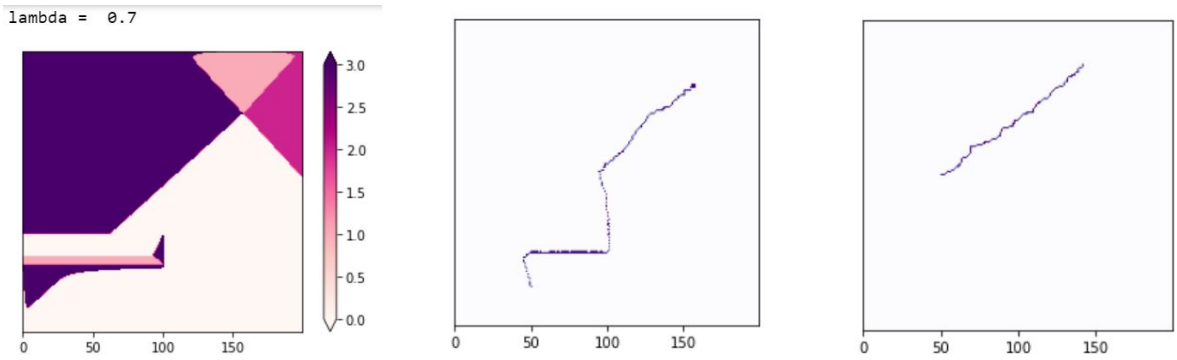
```
  #Si la acción es down
  elif( accion == 1 ):
      if( i == 0 and j > 0 ):
          #Diagonal
          P_dn[index,index] = 1- lamb*0.5
          P_dn[index,i*200+(j-1)] = - lamb*0.5
      elif( i > 0 and j == 0 ):
          #Diagonal
          P_dn[index,index] = 1- lamb*0.6
          P_dn[index,(i-1)*200+j] = - lamb*0.4
      elif( i == 0 and j == 0 ):
          #Diagonal
          P_dn[index,index] = 1 - lamb
      else:
          #Diagonal
          P_dn[index,index] = 1- lamb*0.3
          P_dn[index,i*200+(j-1)] = - lamb*0.3
          P_dn[index,(i-1)*200+j] = - lamb*0.2
          P_dn[index,(i-1)*200+(j-1)] = - lamb*0.2

  #Si la acción es left
  elif( accion == 2 ):
      if ( i > 1 and j == 199 ):
          P_dn[index,(i-1)*200+j] = - lamb*0.5
          P_dn[index,(i-2)*200+j] = - lamb*0.5
      elif( i == 0 and j < 199 ):
          #Diagonal
          P_dn[index,index] = 1 - lamb*0.5
          P_dn[index,i*200+(j+1)] = - lamb*0.5
      elif( i == 1 and j < 199 ):
          P_dn[index,(i-1)*200+j] = - lamb*0.5
          P_dn[index,(i-1)*200+(j+1)] = - lamb*0.5
      elif( i == 0 and j == 199 ):
          #Diagonal
          P_dn[index,index] = 1 - lamb
      elif( i == 1 and j == 199 ):
          P_dn[index,(i-1)*200+j] = - lamb
      else:
          P_dn[index,(i-1)*200+(j+1)] = - lamb*0.3
          P_dn[index,(i-2)*200+j] = - lamb*0.3
          P_dn[index,(i-1)*200+j] = - lamb*0.2
          P_dn[index,(i-2)*200+(j+1)] = - lamb*0.2
```

```
        #Si la acción es right.
        elif( accion == 3 ):
            if( i < 199 and j == 199 ):
                #Diagonal
                P_dn[index,index] = 1- lamb*0.3
                P_dn[index,(i+1)*200+j] = - lamb*0.7
            elif( i == 199 and j < 199 ):
                #Diagonal
                P_dn[index,index] = 1- lamb*0.5
                P_dn[index,i*200+(j+1)] = - lamb*0.5
            elif( i == 199 and j == 199 ):
                #Diagonal
                P_dn[index,index] = 1- lamb
            else:
                P_dn[index,(i+1)*200+j] = - lamb*0.3
                P_dn[index,(i+1)*200+(j+1)] = - lamb*0.4
                #Diagonal
                P_dn[index,index] = 1 - lamb*0.2
                P_dn[index,i*200+(j+1)] = - lamb*0.1
return (P_dn,r_dn)
```

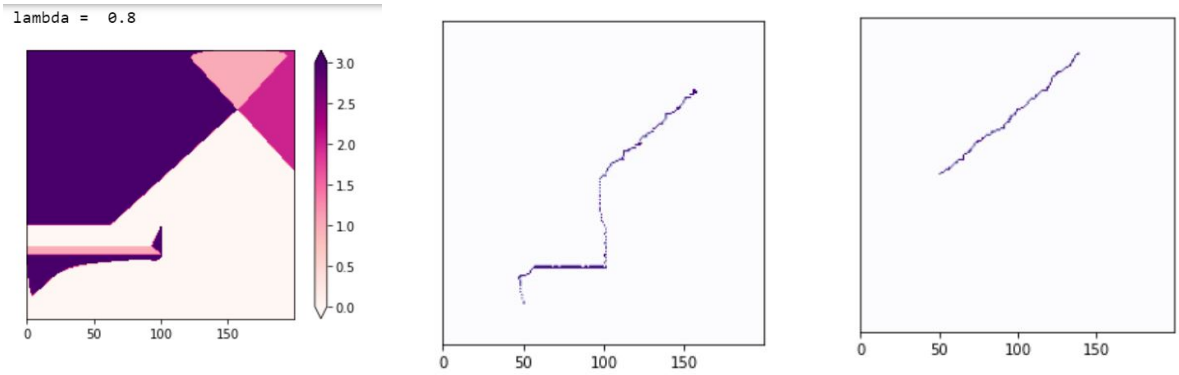
Figura 14: Función auxiliar: calcula $(I - \lambda P_{dn})$ y r_{dn}.

5.2. Sensibilidad de λ

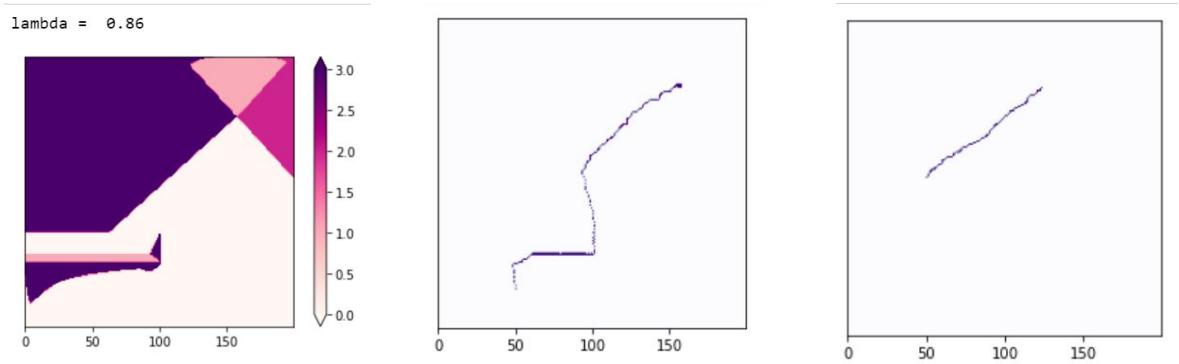
- $\lambda = 0{,}7$



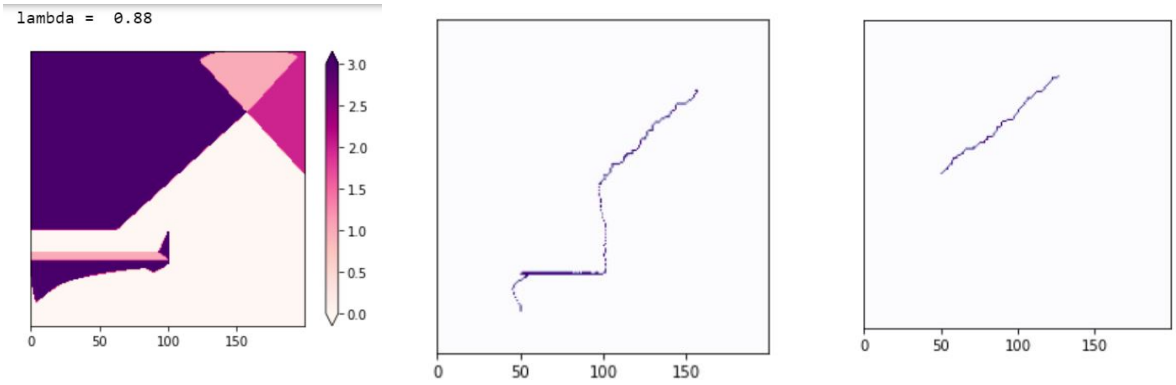
- $\lambda = 0{,}8$



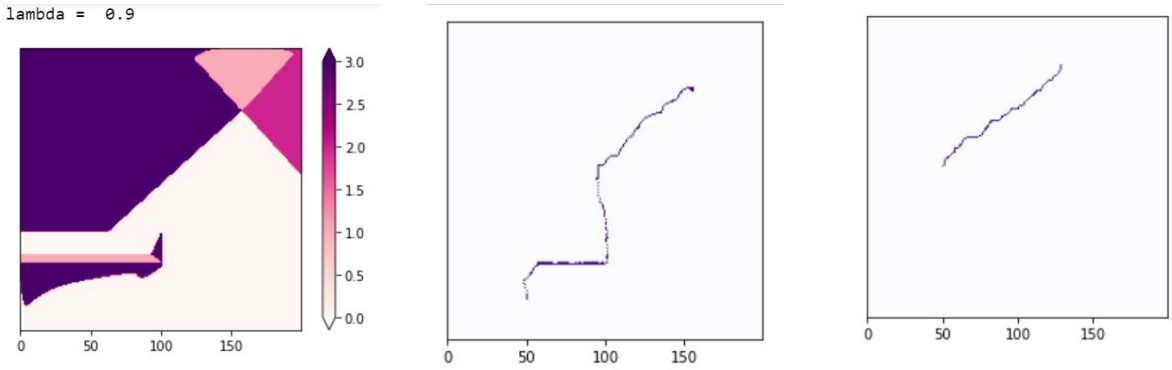
- $\lambda = 0{,}86$:



- $\lambda = 0{,}88$:



- $\lambda = 0{,}9$:



- $\lambda = 0{,}99$: