Taller 2 Optimización Estocástica

Esteban Leiva

Octubre 2022

1 Introducción

Considere un avión que se mueve en una retícula mxm (m=20) (por razones de complejidad se tomó 0.1 del tamaño del problema) bajo la influencia del viento. El objetivo del avión es llegar a las coordenadas [17,19]x[17,19]. En cada posición en el interior de la retícula el objeto tiene las acciones $A = u, d, l, r$ con probabilidades de transición dadas por el enunciado.

2 Modelación

Las probabilidades de transición dadas en el enunciado tienen que ser complementadas teniendo en cuenta el borde de la retícula. Esto se debe a que, por ejemplo, si no se modifica el kernel del enunciado y el avión se encuentra en (x,y) en el borde izquierdo y la acción es "l" (left) la probabilidad de que el estado siguiente (z,w) donde $z = x - 1$ ó $z = x - 2$ debe ser igual a 0 porque si no lo es, se saldría de la retícula; pero en el kernel dado no es así.

Por ejemplo, acá se muestra un fragmento del kernel modificado para la acción "l" en el caso en el que el avión ha llegado al objetivo y en el que se encuentra en el borde izquierdo de la retícula menos el punto [0]x[m-1]

```
if a == "l":
    if s[0] in arrived[0] and s[1] in arrived[1]:
        if s1[0] == s[0] and s1[1] == s[0]:
            return 1
        else:
            return 0
    elif s[0] == 0 and s[1] != m-1:
        if s1[0] == s[0] and s1[1] == s[1]+1:
            return 0.5
        elif s1[0] == s[0] and s1[1] == s[1]:
            return 0.5
        else:
            return 0
```

Este es un problema descontado de horizonte infinito:

$T = \mathbb{N}$, tiempos de decisión con horizonte infinito

$S = [0, m]x[0, m]$, cada estado es una tupla (i,j) que representa la posición en la cuadrícula.

$A = \{u, d, l, r\}$, los movimientos que se pueden realizar en un estado, arriba, abajo, izquierda, derecha.

$Q(s1, s, a)$, el kernel de transición del estado s, al estado s1, usando la acción a. Este kernel está definido como en el enunciado y modificado como se explico anteriormente para asegurar que estuviera bien definido en toda la cuadrícula.

$r(s)$ definido de la siguiente manera:

```
def r(s,a):
    r = -1
    if s[0] in avoid[0] and s[1] in avoid[1]:
        r = -1000
    elif s[0] in arrived[0] and s[1] in arrived[1]:
        r = 100
    return r
```

Inicializamos la cuadrícula 20x20 y definimos una zona roja (que queremos evitar y representa el 5 por ciento de la cuadrícula) y una zona verde (la zona terminal a la que queremos llegar en el menor tiempo posible).

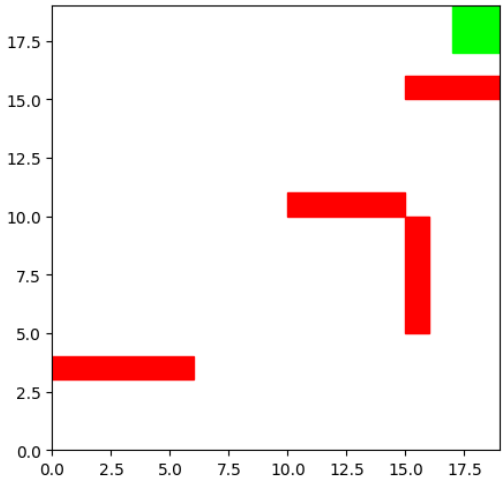

Figure 1: Modelo

3 Policy Iteration

Construya la matriz P_{d_n} iterando sobre $Q(s1, s, a)$ y conviértala a la estructura cscmatrix() de scipy para poder trabajar con algoritmos construidos para matrices sparse. Construya el vector r_{d_n} iterando sobre $r(s)$.

Detalles de la implementación:

1. Fije $n = 0$ y seleccione una política arbitraria $d_0 \in D$ para comenzar con la iteración por política. En este caso, se utilizó la política que consiste en "u" (ir arriba) para todo estado. La política arbitraria está implementada como una matriz de numpy mxm que asigna una de las cuatro acciones a cada casilla (estado).

2. Policy evaluation: fije $\lambda = 0.9$ obtenga v^n utilizando la función de scipy.sparse.linalg.spsolve para resolver $(I - \lambda P_{d_n})v = r_{d_n}$

3. Policy improvement: construya d_{n+1} de tal manera que

$$d_{n+1} \in argmax_{d \in D}\{r_d + \lambda P_d v^n\}$$

   Esto hágalo por componentes, para cada estado calcule la recompensa esperada si se utiliza cada acción y con la función argmax de numpy escoja la acción que maximiza la recompensa para cada estado.

4. Pare si $d_{n+1} = d_n$, lo cual se implementó con la función arrayequal() de numpy. De lo contrario, incremente n por 1 y devuélvase al paso 2.

Con esto, obtenemos la siguiente política:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	r	r	r	r	r	r	r	r	r	u	u	l	l	l	l	l	l	l	l	l
1	d	d	r	r	r	r	r	r	r	u	l	l	l	l	l	l	l	l	l	l
2	d	d	d	d	d	d	d	r	r	u	l	d	d	d	d	d	d	d	d	d
3	d	d	d	d	d	r	r	r	r	u	l	l	l	d	d	d	d	d	d	d
4	d	d	d	d	d	r	r	r	r	u	l	l	l	l	l	d	d	d	d	d
5	d	d	d	d	d	r	r	r	r	u	l	l	d	d	d	d	d	d	d	d
6	u	u	u	u	u	r	r	r	r	u	l	l	l	l	u	u	u	u	u	u
7	u	u	u	u	u	u	r	r	r	u	l	l	l	u	u	u	u	u	u	u
8	u	u	u	u	u	u	r	r	r	u	l	l	u	u	u	u	u	u	u	u
9	u	u	u	u	u	u	r	r	r	u	l	u	u	u	u	u	u	u	u	u
10	u	u	u	u	u	u	u	r	r	u	u	u	u	u	u	u	u	u	u	u
11	r	r	r	r	r	r	r	r	r	u	u	u	r	r	r	r	r	u	u	u
12	d	r	r	r	r	r	r	r	r	u	u	l	l	r	r	r	r	u	u	u
13	d	d	r	r	r	r	r	r	r	u	l	l	l	r	r	r	r	r	r	l
14	d	d	d	d	d	d	d	r	r	u	l	d	d	d	d	u	u	u	u	u
15	u	u	u	u	u	u	u	r	r	u	u	u	u	u	u	u	u	u	u	u
16	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	l
17	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	u	l
18	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	u	d
19	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	d	d	d

Figure 2: Política

Ahora, definimos dos puntos de salida (0,0) y (19,0) correspondientes a las dos esquinas inferiores de la cuadrícula. A partir de estos puntos, realizamos 100 simulaciones para cada uno utilizando la política y el kernel de transición para avanzar de estado a estado (con ayuda de la función random.choice() de python):

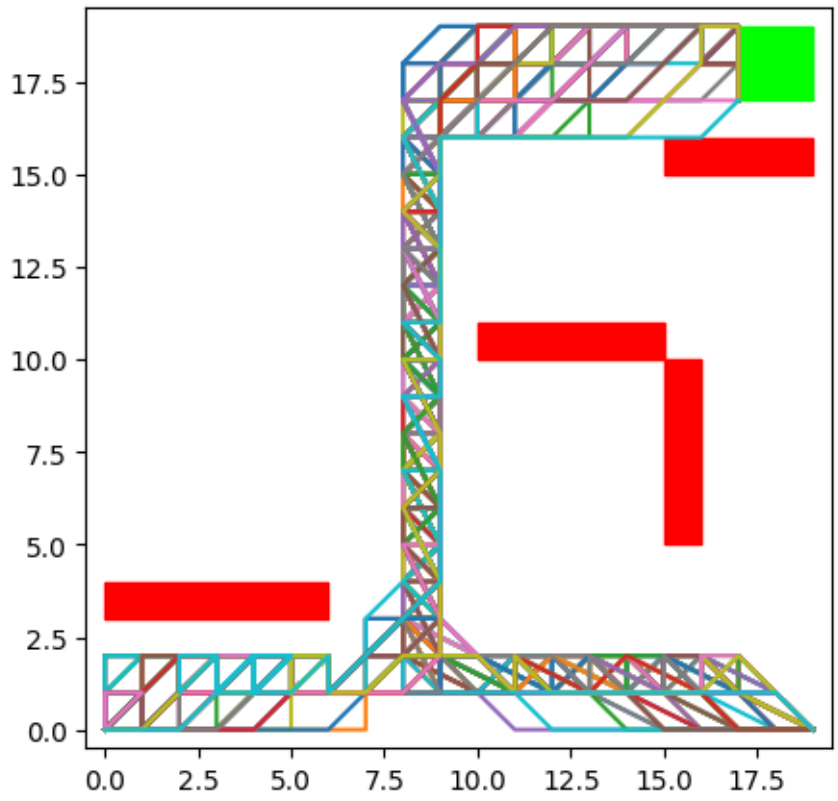

Figure 3: Política

4 Programación lineal

Planteamiento del dual:

$$Maximize \sum_{s \in S} \sum_{a \in A_s} r(s)x(s,a)$$

sujeto a

$$\sum_{a \in A_j} x(j,a) - \sum_{s \in S} \sum_{a \in A_s} \lambda Q(j,s,a)x(s,a) = \alpha(j)$$

y

$$x(s,a) >= 0$$

Por razones de simplicidad, se decidió implementar el problema primal y con la librería CVXPY usar la funcion .dualvalue para obtener la solución al problema dual:

$$Minimize \sum_{j \in S} \alpha(j) v(j)$$

sujeto a

$$v(s) - \sum_{j \in S} \lambda p(j|s,a) v(j) \geq r(s,a)$$

$\alpha(j)$ es tal que $\sum_{j \in S} \alpha(j) = 1$ y en nuestro caso $r(s,a) = r(s)$

Definimos el problema en CVXPY de la siguiente manera donde P representa el kernel de transición, R representa el vector de recompensa y alfa representa la distribución inicial (que en este caso es uniforme):

```
alfa = np.asarray([1/(m*m) for s in states])
v = cp.Variable(m*m)
objective = cp.Minimize(cp.matmul(v,alfa.transpose()))

constraints = []
for a in range(4):
    constraints.append(R<=v-cp.matmul(P[a],v))

prob = cp.Problem(objective,constraints)
result = prob.solve(solver='SCS',verbose=True)

dual = []
for i in range(0,4):
    dual.append(constraints[i].dual_value)

dual = np.array(dual)
```

Con esta implementación llegamos a la siguiente política estacionaria aleatoria: (las filas representan los estados enumerados de 0 a $m^2 - 1$ y las columnas las probabilidades de tomar una de las acciones enumeradas del 0 al 4:

	0	1	2	3
0	0	0	0	1
1	0	1	0	0
2	0	1	0	0
3	0	1	0	-3.341548207e-17
4	0	1	0	0
5	0	1	0	0
6	1	0	0	0
7	1	0	0	0
8	1	3.479035834e-16	1.739517917e-16	0
9	1	0	0	-9.567658351e-17
10	1	0	0	0

Figure 4: Política estacionaria aleatoria

Sin embargo, númericamente esta política se puede considerar determinística porque la probabilidad de que se tome una acción es 1 y el resto es 0 o un número tan pequeño que es despreciable. Si graficamos esta política determinística podemos ver que es igual a la obtenida por Policy Iteration:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	r	r	r	r	r	r	r	r	u	u	u	l	l	l	l	l	l	l	l	l
1	d	d	r	r	r	r	r	r	u	u	l	l	l	l	l	l	l	l	l	l
2	d	d	d	d	d	d	d	r	u	u	l	d	d	d	d	d	d	d	d	d
3	d	d	d	d	d	r	r	r	u	u	l	l	l	d	d	d	d	d	d	d
4	d	d	d	d	d	r	r	r	u	u	l	l	l	l	l	d	d	d	d	d
5	d	d	d	d	d	r	r	r	u	u	l	l	d	d	d	d	d	d	d	d
6	u	u	u	u	u	r	r	r	u	u	l	l	l	l	u	u	u	u	u	u
7	u	u	u	u	u	u	r	r	u	u	l	l	l	u	u	u	u	u	u	u
8	u	u	u	u	u	u	r	r	u	u	l	l	u	u	u	u	u	u	u	u
9	u	u	u	u	u	u	r	r	u	u	l	u	u	u	u	u	u	u	u	u
10	u	u	u	u	u	u	u	r	u	u	u	u	u	u	u	u	u	u	u	u
11	r	r	r	r	r	r	r	r	r	u	u	u	r	r	r	r	r	u	u	u
12	d	r	r	r	r	r	r	r	r	u	u	l	l	r	r	r	r	u	u	u
13	d	d	r	r	r	r	r	r	r	u	l	l	l	r	r	r	r	r	r	l
14	d	d	d	d	d	d	d	r	r	u	l	d	d	d	d	u	u	u	u	u
15	u	u	u	u	u	u	u	u	r	r	u	u	u	u	u	u	u	u	u	u
16	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	l
17	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	u	l
18	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	u	u	d
19	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	d	d	d

Figure 5: Política estacionaria determinística

Más aún, se cambió la distribución inicial de una distribución uniforme (como estaba definida en la parte anterior) a uno en el cual hay componentes que valen 0 y otros que valen $\frac{2}{m^2}$ tal que $\sum_{j \in S} \alpha(j) = 1$ y se encontró que estos cambios en la distribución inicial no influyen en la política generada por el algoritmo de programación lineal.

5 Análisis sensibilidad factor de descuento

Se va a utilizar el algoritmo de Policy Iteration porque en la implementación es el que se puede modificar más rápidamente para hacer el cambio de λ porque no se necesita cargar matrices. Vamos a acercar λ a 1 de la siguiente manera: 0.3, 0.5, 0.7, 0.8, 0.9, 0.93, 0.95, 0.98, 0.99 y vamos a revisar las simulacione tomando como punto inicial los mismos dos valores del punto 3:

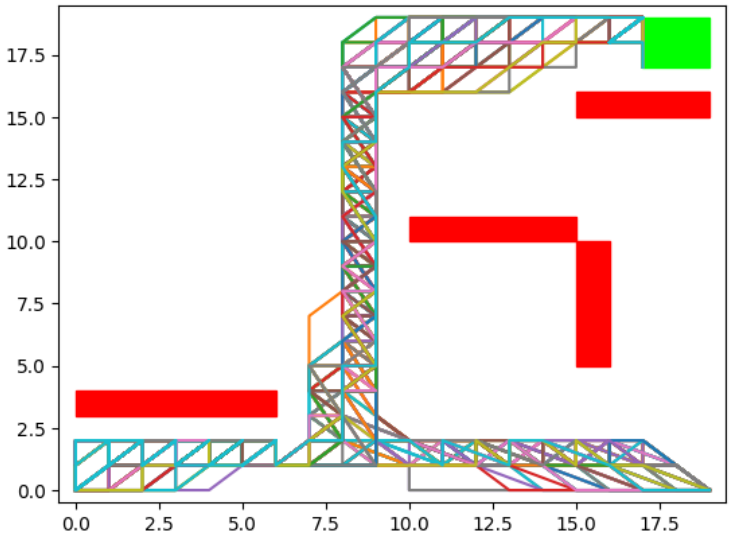

Figure 6: $\lambda = 0.5$

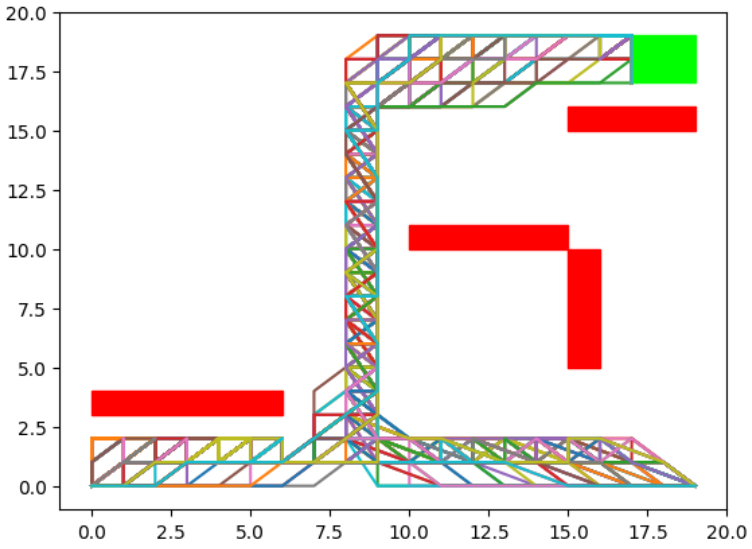

Figure 7: $\lambda = 0.7$

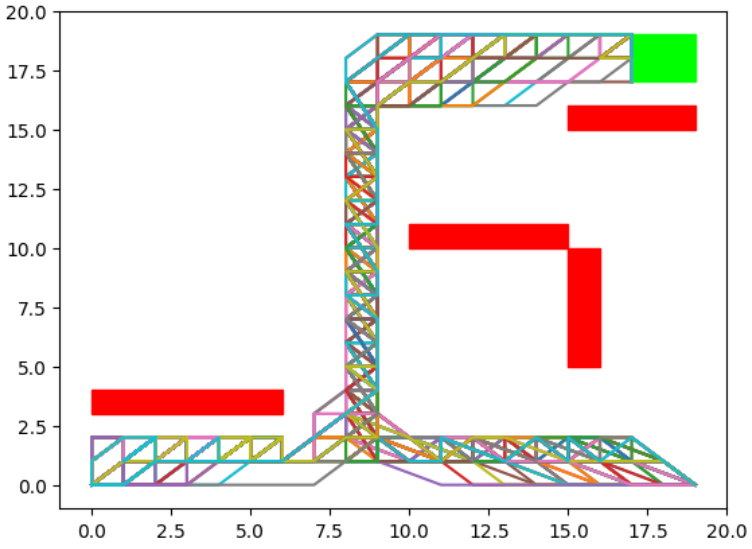

Figure 8: $\lambda = 0.8$

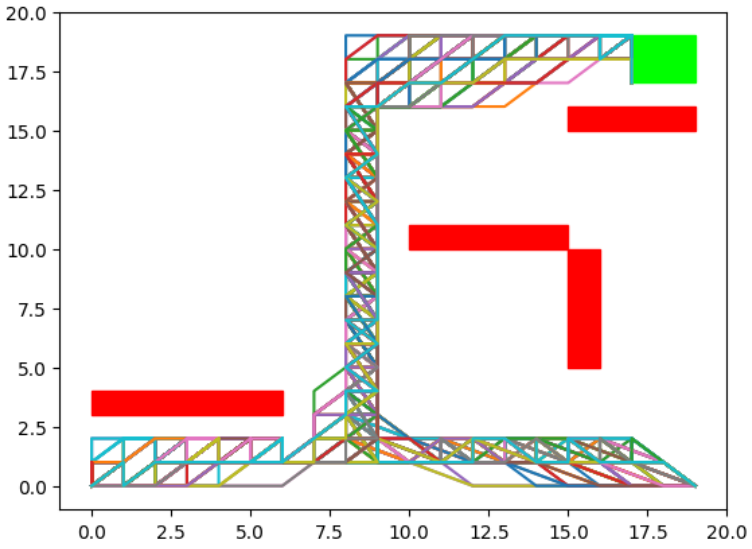

Figure 9: $\lambda = 0.9$

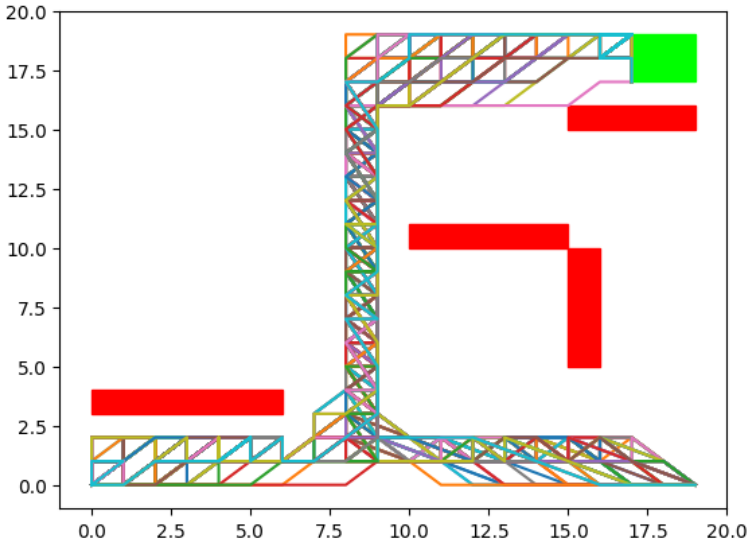

Figure 10: $\lambda = 0.93$

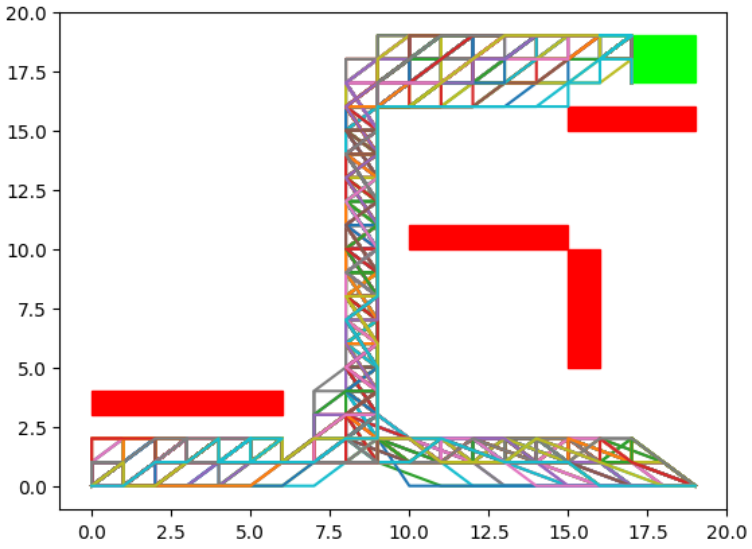

Figure 11: $\lambda = 0.95$

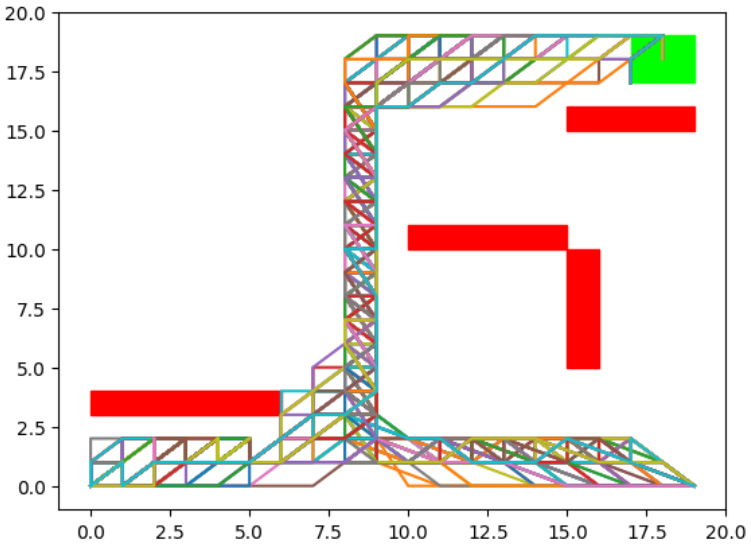

Figure 12: $\lambda = 0.98$

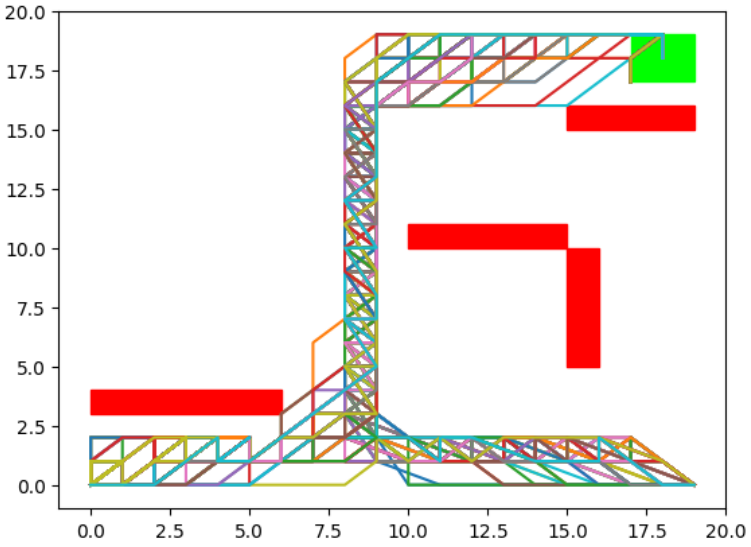

Figure 13: $\lambda = 0.99$

Como se puede evidenciar en la anterior lista de gráficas, a medida que el factor de descuento λ se va acercando a 1; los caminos de las simulaciones se van haciendo más cortos. Por ejemplo, comparando 0.5 y 0.99 se puede ver que el factor de descuento más cercano a 1 es más compacto en la línea de ascenso y en la línea paralela al eje x que está a la altura de la región verde. Esto es coherente con la teoría porque un λ mayor implica que los costos futuros tienen una mayor importancia, lo cual hace que el costo de los caminos más largo sea más negativo.

6 Conclusiones

Los algoritmos de Policy Iteration y Programación Lineal llegaron a la misma política y esta política estacionaria, a la hora de resolver el problema del avión, es efectiva esquivando los obstáculos y llegando al punto de llegada de una manera razonable. En adición, se encontró que el método por programación lineal, en la forma como se implementó usando CVXPY, es más adecuado para manejar un conjunto de estados más grandes. Además, a medida que el factor de descuento se va a acercando a 1, los caminos de la simulación se van volviendo más cortos debido a que los costos futuros tienen mayor peso en el problema. Sin embargo, esta mejora en los caminos implica un mayor costo computacional.