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

Taller 1

1. Introducción al problema

Cada mes, el manager de un almacén determina el inventario de un producto y decide si surtirlo o no. Se enfrenta así a un *trade-off* entre el costo de mantener el inventario y las pérdidas que pueden derivarse de no tener suficiente inventario para satisfacer la demanda. El objetivo de este problema es maximizar la ganancia.

En el problema, la demanda del producto es aleatoria con una distribución conocida, y más precisamente el caso que se analizará en este documento será cuando ésta tiene una distribución Poisson(10).

Para formalizar el problema debemos hacer algunas suposiciones iniciales:

- El almacén tiene capacidad para $M = 100$ unidades.
- Se venden únicamente cantidades enteras del producto.
- Los costos, ganancias y la distribución de la demanda no varían de mes a mes.
- Si la demanda excede el inventario, el cliente comprará el producto en otra parte.
- La demanda de los productos se da en el transcurso del mes.
- La decisión de ordenar más producto o no, se hace al inicio del mes y la entrega es inmediata.

Ahora, con esta información, se definen algunas variables para poder modelar el problema:

Variable	**Explicación**
s_t	inventario al inicio del mes t
a_t	número de unidades ordenadas por el manager en el mes t.
D_t	demanda aleatoria en el mes t
p_j	$P(D_t = j) = \frac{e^{-10} \cdot 10^j}{j!}, j = 0, 1, 2, \ldots$ distribución de probabilidad de la demanda

Tenemos entonces que el cálculo de s_{t+1} en función de s_t, a_t y D_t es $s_{t+1} = \text{máx}\{s_t + a_t - D_t, 0\}$.

Ahora, para formalizar las ganancias definimos las siguientes funciones:

Variable	**Explicación**
$O(u)$	Costo de ordenar u unidades.
$h(u)$	Costo de mantener un inventario con u unidades. En este caso, $h(u) = u$.
$f(j)$	Si la demanda es j, $f(j)$ son los ingresos. En este caso $f(j) = 10 \cdot j$.
$r(t)$	Ganancia.
$F_t(u)$	Valor esperado al inicio del mes de los ingresos en el mes t, donde u es el inventario inicial.

$O(u)$ está está dado por:

$$O(u) = \begin{cases} 2 + 2u & si \quad u > 0 \\ 0 & si \quad u = 0 \end{cases}$$

Y por lo tanto, la ganancia se puede expresar como:

$$\begin{aligned} r_t(s_t, a_t, s_{t+1}) &= -O(a_t) - s_t - a_t + 10(s_t + a_t - s_{t+1}) \\ &= 9s_t + 9a_t - O(a_t) - 10s_{t+1} \end{aligned}$$

Si tomamos $q_u = \sum_{j=u}^{\infty} p_j = \sum_{j=u}^{\infty} \frac{e^{-10} \cdot 10^j}{j!}$ como la probabilidad de que la demanda exceda el inventario, entonces

$$F(u) = \sum_{j=0}^{u-1} f(j)p_j + f(u)q_u = \sum_{j=0}^{u-1} 10j \cdot p_j + 10u \cdot q_u$$

Entonces, la recompensa esperadas para todo t:

$$\begin{aligned} r(s, a) &= F(s + a) - O(a) - h(s + a) \\ &= \sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(a + s) \cdot q_{s+a} - O(a) - s - a \\ &= \begin{cases} \sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(s + a) \cdot q_{s+a} - 3a - s - 2 & si \quad a > 0 \\ \sum_{j=0}^{s-1} 10j \cdot p_j + 10s \cdot q_s - s & si \quad a = 0 \end{cases} \end{aligned}$$

Los estados, por lo tanto, modelarán la cantidad de inventario disponible al comienzo de un mes:

$$S = \{0, 1, 2, \ldots, 100\}$$

Y las acciones representan la cantidad de stock adicional a ordenar en el mes t:

$$A_s = \{0, 1, 2, \ldots, 100 - s\}$$

En este proyecto se utilizará un tiempo infinito, $H = \infty$ y una función de desempeño descontada, es decir, queremos minimizar para $\lambda = 0{,}9$:

$$\mathbb{E}\left[\sum_{t=1}^{\infty} \lambda^{t-1} r(X_t, Y_t)\right] = \mathbb{E}\left[\sum_{t=1}^{\infty} 0{,}9^{t-1} r(X_t, Y_t)\right]$$

La función de valor es

$$\begin{aligned} v_{n+1}(s) &= \sup_{a \in A(s)} \left\{ r(s,a) + \sum_{j \in S} \lambda p(j|s,a) v_n(j) \right\} \\ &= \text{máx} \left\{ \begin{array}{l} \displaystyle\max_{a \in A(s), a \neq 0} \left\{ \left(\sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(s+a) \cdot q_{s+a} - 3a - s - 2 \right) + \sum_{j \in S} 0{,}9 \cdot p(j|s,a) v_n(j) \right\}, \\ \left(\displaystyle\sum_{j=0}^{s-1} 10j \cdot p_j + 10s \cdot q_s - s \right) + \sum_{j \in S} 0{,}9 \cdot p(j|s,0) v_n(j) \end{array} \right\} \end{aligned}$$

Nos falta únicamente calcular qué es $p(j|s,a)$. Esta es la probabilidad de que en un tiempo t el inventario sea igual a j dado que en el tiempo $t-1$ se tenían s unidades en inventario y se pidieron a unidades. Si $100 \geq s + a \geq j > 0$, esto es lo mismo que la probabilidad que la demanda en el mes t sea igual a $s + a - j$:

$$p(j|s,a) = P(D_t = s + a - j) = p_{s+a-j} = \frac{e^{-10} \cdot 10^{s+a-j}}{(s+a-j)!}$$

Si $M \geq s + a$ y $j = 0$, significa que se vendieron todas las unidades, por lo que

$$p(j|s,a) = q_{s+a} = 1 - \mathbb{P}(X < s + a)$$

Por último, si $j > s + a$ es imposible, por lo que $p(j|s,a) = 0$.

2. Código y detalles de implementación

2.1. Iteración por valor

Inicialmente se crearon dos arreglos psubj y qsubj y se llenaron con todos los valores de p_j y de q_j respectivamente. Esto con el fin de no tener que calcularlos cada vez que los llamamos. Para esto se utilizaron los métodos dpois y ppois para distribuciones de Poisson. Es importante notar que a lo largo de la implementación, como los índices de los arreglos en R empiezan en 1, psub[1] $= p_0$ y, en general psub[$n+1$] $= p_n$ y sucede lo mismo para las q's.

```
######################################################################

#Calcula los valores de pj para j=0,1...200 y los guarda en un vector
psubj = rep(0, 201)
for( j in 1:201 ){
  #psubj[1] = p_0
  psubj[j] = dpois(x=(j-1), lambda = 10)
}

#Calcula los valores de qj para j = 1...200 y los guarda en un vector
qsubj = rep(0,201)
for( j in 1:201 ){
  qsubj[j] = 1 - ppois(q = (j-2), lambda = 10, lower.tail = TRUE)
}

######################################################################
```

Figura 1: Cálculo de p_j y q_j.

El siguiente es el ciclo principal de Iteración por valor. Se utilizaron dos funciones auxiliares: recompensa(s, a) y suma$(s, a, v0)$ que respectivamente calculan $r(s, a)$ y $\sum_{j \in S} p(j|s, a)v_0(j)$.

```
#M, maximo stock
M = 100

#lambda
lambda = 0.9

#Definimos v0(s) = 0 para todo s.
#Esta variable la utilizaremos para las iteraciones de forma recursiva. Es vn.
v0 = rep(0,M+1)

#Este va a representar vn+1 cuando calculemos iterativamente.
vn = rep(0,M+1)

#epsilon lo tomamos como 0.1
epsilon = 0.1

#Inicializamos n como 0.
n = 0

#Condición de parada del ciclo.
parar = FALSE
```

Figura 2: Inicialización de variables.

```
while(parar == FALSE){
  print(n)
  #Itero sobre los s
  for( s in 0:M ) {
    #Los valores a encontrar el máximo (r(s,a) + sum...)
    valores <- rep(0, (M-s))

    #Comenzamos la iteración sobre A(i)
    for( a in 0:(M-s) ){
      valores[a+1] = recompensa(s,a) + (lambda * suma(s,a,v0))
    }
    #Tomamos el maximo entre los valores calculados
    vn[s] = max(valores)
  }

  #Condición de parada
  norma = max(abs(vn-v0))

  if(norma < ((epsilon*(1-lambda))/(2*lambda))){
    parar = TRUE
  }
  v0 <- vn #Actualizo mi v0 como vn para poder utilizarlo en el siguiente cálculo de vn+1
  n = n+1  #Actualizo el n
}
```

Figura 3: Ciclo principal.

```
#Comienza el paso 4.
d = rep(0,M+1)

#Itero sobre los s
for( s in 0:M )
{
  #Los valores a encontrar el máximo (r(s,a) + sum...)
  valores = rep(0, (M-s))

  #Comenzamos la iteración sobre A(i)
  for( a in 0:(M-s) ){
    valores[a+1] = recompensa(s,a) + (0.9 * suma(s,a,vn))
  }

  d[s+1] = (which.max(valores)-1)

}
```

Figura 4: Paso 4, cálculo de la política.

```
recompensa <- function(s,a){
  rta = 0
  if(s+a-1 >= 0){
    #Con el ciclo se calcula suma 10j p_j para j=0...s+a-1
    for(j in 0:(s+a-1)){
      rta = rta + (10*j*psubj[j+1])
    }

    rta = rta + (10*(s+a)*qsubj[s+a+1] - 3*a - s)             #Se suma el resto.
    #Si a>0 se debe hacer el descuento de 2 que es el valor de realizar el pedido.
    if(a > 0){ rta = rta - 2  }
  }
  return(rta)
}

suma <- function(s,a,v0){
  rta = 0
  for (j in 0:M){
    pjsa = 0 #p(j|s,a)
    #Vemos los casos posibles para p(j|s,a)
    if( j > 0 && j <= (s+a) && (s+a) <= 100){
      pjsa = psubj[s+a-j+1]
    }
    else if( (s+a) <= M && j == 0 ){
      pjsa = qsubj[s+a+1]
    }
    rta = rta + (pjsa*v0[j+1])
  }
  return(rta)
}
```

Figura 5: Funciones auxiliares de recompensa y suma.

2.2. Gauss Seidel

En la implementación de Iteración por valor con Gauss-Seidel, lo único que cambia es el método suma$(s, a, v0)$ pues ahora lo que buscamos maximizar es

$$r(s,a) + \lambda \left(\sum_{j<s} p(j|s,a) v^{n+1}(j) + \sum_{i \geq j} p(j|s,a) v^{n}(j) \right)$$

Además de esto, lo único que cambia en la iteración principal (Figura 3), es que no se hace llamado a la función suma, sino a la función sumaGS.

```r
sumaGS <- function(s,a,v0,vn){
  rta = 0
  for (j in 0:M){
    pjsa = 0 #p(j|s,a)

    #Vemos los casos posibles para p(j|s,a)
    if( j > 0 && j <= (s+a) && (s+a) <= 100){
      pjsa = psubj[s+a-j+1]
    }
    else if( (s+a) <= M && j == 0 ){
      pjsa = qsubj[s+a+1]
    }

    #Verificamos por qué multiplicar, dependiendo si j < s o j >= s.
    if( j < s ){
      rta = rta + (pjsa * vn[j+1])
    }
    else{
      rta = rta + (pjsa * v0[j+1])
    }

  }
  return(rta)
}
```

Figura 6: Función de suma para GS.

2.3. Jacobi

El método de Jacobi propone otra forma de definir un *regular splitting* de la matriz $I - \lambda P_d$. Esta es (Q_d, R_d) con:

$$Q_d = \begin{bmatrix} 1-\lambda p_{1,1} & 0 & \dots & 0 \\ 0 & 1-\lambda p_{2,2} & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & 1-\lambda p_{100,100} \end{bmatrix} \quad R_d = \lambda \begin{bmatrix} 0 & p_{1,2} & \dots & p_{1,100} \\ p_{2,1} & 0 & \dots & p_{1,100} \\ \vdots & \vdots & \dots & \vdots \\ p_{100,1} & p_{100,2} & \dots & 0 \end{bmatrix}$$

Por lo tanto, al igual que en el método anterior, lo que queremos encontrar en el paso 2 es

$$v^{n+1} = \max_{d \in D} \{Q_d^{-1} r_d + Q_d^{-1} R_d v^n\}$$

Como la inversa de Q_d se puede encontrar explícitamente al ser una matriz diagonal, es lo que haremos:

$$(Q_d^{-1})_{ij} = \begin{cases} \frac{1}{1-\lambda p_{ii}} & si \quad i = j \\ 0 & si \quad i \neq j \end{cases}$$

Por lo tanto,

$$(Q_d^{-1} R_d)_{ij} = \sum_{k=1}^{100} (Q_d^{-1})_{ik} (R_d)_{kj} = (Q_d^{-1})_{ii} (R_d)_{ij} = \begin{cases} \frac{\lambda p_{ij}}{1-\lambda p_{ii}} & si \quad i \neq j \\ 0 & si \quad i = j \end{cases}$$

Así,

$$(Q_d^{-1}R_d v^n)_i = \sum_{k=1}^{100}(Q_d^{-1}R_d)_{ik}v_k^n = \sum_{k\neq i}\frac{\lambda p_{ij}}{1-\lambda p_{ii}}v_k^n$$

De igual manera, tenemos que

$$(Q_d^{-1}r_d)_i = \sum_{k=1}^{100}(Q_d^{-1})_{ik}(r_d)_k = (Q_d^{-1})_{ii}(r_d)_i = \frac{1}{1-\lambda p_{ii}}(r_d)_i$$

En conclusión,

$$v^{n+1} = \max_{d\in D}\left\{\left(\frac{1}{1-\lambda p_{ii}}(r_d)_i + \sum_{k\neq i}\frac{\lambda p_{ij}}{1-\lambda p_{ii}}v_k^n\right)_i\right\}$$

Tenemos entonces viendolo componente por componente, que

$$\begin{aligned} v^{n+1}[s] &= \max_{a\in A(s)}\left\{\frac{1}{1-\lambda p(s|s,a)}r(s,a) + \sum_{j\neq s}\frac{\lambda p(j|s,a)}{1-\lambda p(s|s,a)}v^n[j]\right\} \\ &= \max_{a\in A(s)}\left\{\frac{1}{1-\lambda p_a}r(s,a) + \lambda\sum_{j\neq s}\frac{p(j|s,a)}{1-\lambda p_a}v^n[j]\right\} \end{aligned}$$

Por lo tanto, en la implementación podemos cambiar únicamente los métodos suma(s, a, v0) y recompensa(s, a) por los siguientes:

```
recompensaJacobi <- function(s,a){
  rta = 0
  if(s+a-1 >= 0){
    #Con el ciclo se calcula suma 10j p_j para j=0...s+a-1
    for(j in 0:(s+a-1)){
      rta = rta + (10*j*psubj[j+1])
    }

    rta = rta + (10*(s+a)*qsubj[s+a+1] - 3*a - s)             #Se suma el resto.
    #Si a>0 se debe hacer el descuento de 2 que es el valor de realizar el pedido.
    if(a > 0){ rta = rta - 2  }

    #Hasta aquí rta = r(s,a), falta multiplicar por 1/1-lambda p(s|s,a)
    rta = rta * (1/(1- (lambda*psubj[a+1])) )

    }
  return(rta)
}
```

Figura 7: Función suma para Jacobi.

```
sumaJacobi <- function(s,a,v0){
  rta = 0
  for (j in 0:M){
    #Sumamos sobre los estados diferentes de s.
    if(j != s){
      pjsa = 0 #p(j|s,a)
      #Vemos los casos posibles para p(j|s,a)
      if( j > 0 && j <= (s+a) && (s+a) <= 100){
        pjsa = psubj[s+a-j+1]
      }
      else if( (s+a) <= M && j == 0 ){
        pjsa = qsubj[s+a+1]
      }
      rta = rta +  ( (pjsa*v0[j+1])/ (1-lambda * psubj[a+1]))
    }
  }
  return(rta)
}
```

Figura 8: Función recompensa para Jacobi.

3. Resultados numéricos

3.1. Iteración por valor

Con un $\epsilon = 0{,}1$ y $v_0(s) = 0$ para todo s, se obtuvieron los siguientes valores para $d_\epsilon(s)$:

s	$d_\epsilon(s)$		s	$d_\epsilon(s)$		s	$d_\epsilon(s)$
0	13		4	9		8	5
1	12		5	8		9	4
2	11		6	7		10	3
3	10		7	6		≥ 11	0

De lo que podemos concluir que la política óptima para un mes con stock s es:

$$\begin{cases} \text{Pedir } 13 - s \text{ unidades.} & si \quad s \leq 10 \\ \text{No realizar pedido} & si \quad s > 10 \end{cases} \tag{1}$$

La cual es una política de tipo (σ, Σ) con $\sigma = 10$ y $\Sigma = 13$.

3.2. Gauss-Sediel

Se obtuvo el mismo resultado anterior para la política óptima. Lo único que se pudo evidenciar fue una disminución en el número de iteraciones necesarias. En particular, con Iteración por valor se necesitaron 90 iteraciones, mientras que con Gauss-Sediel fueron 52.

3.3. Jacobi

Se obtuvo el mismo resultado que para Iteración por valor y Gauss-Sediel. El número de iteraciones fue de 84, un poco menos que el número de iteraciones para Iteración por valor, pero significativamente más que para Gauss-Sediel.

3.4. Gráfica comparativa entre VI, GS y J

La siguiente gráfica compara la convergencia en los tres métodos previamente analizados. En particular se puede evidenciar que Gauss-Sediel es el que tiene una mayor tasa de convergencia, y esto se pudo evidenciar al comparar el número de iteraciones que se necesitaron para que el algoritmo se detuviera. En segundo lugar está Jacobi y por último Value Iteration.

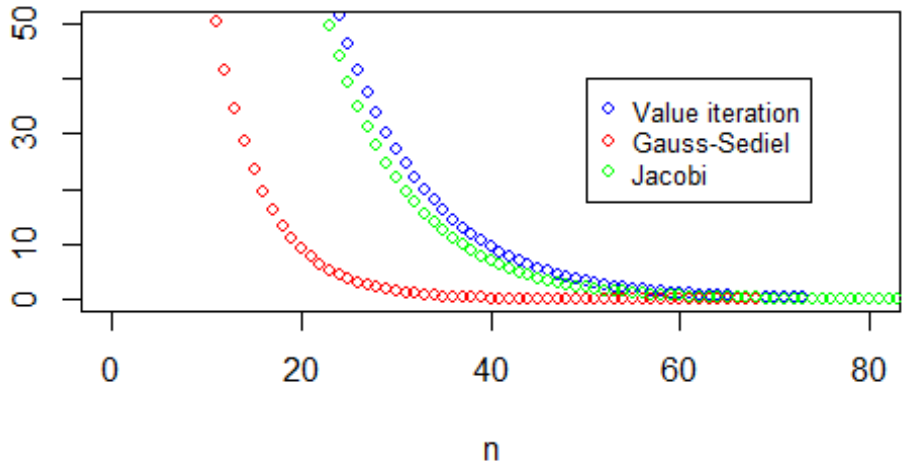

Figura 9: Comparación de convergencia: $||v_n - v^*_\lambda||$ vs n

3.5. Alteración del problema: $c(s) = -0{,}01s^2 + 3s$

Cambiamos un poco el problema y ahora el costo de ordenar u unidades es igual a

$$O(u) = \begin{cases} 2 + 3u - 0{,}01u^2 & si \quad u > 0 \\ 0 & si \quad u = 0 \end{cases}$$

Lo único que se debe modificar es el el cálculo de la recompensa, pues la recompensa esperadas para todo t será:

$$
\begin{aligned}
r(s,a) &= F(s+a) - O(a) - h(s+a) \\
&= \sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(s+a) \cdot q_{s+a} - O(a) - s - a \\
&= \begin{cases} \sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(s+a) \cdot q_{s+a} - 2 - 3a + 0{,}01a^2 - s - a & si \quad a > 0 \\ \sum_{j=0}^{s-1} 10j \cdot p_j + 10s \cdot q_s - s & si \quad a = 0 \end{cases} \\
&= \begin{cases} \sum_{j=0}^{s+a-1} 10j \cdot p_j + 10(s+a) \cdot q_{s+a} - 4a + 0{,}01a^2 - s - 2 & si \quad a > 0 \\ \sum_{j=0}^{s-1} 10j \cdot p_j + 10s \cdot q_s - s & si \quad a = 0 \end{cases}
\end{aligned}
$$

Utilizaremos el método Gauss-Sediel y lo único que tenemos que cambiar de la implementación es la función recompensa:

```
recompensa <- function(s,a){
  rta = 0
  if(s+a-1 >= 0){
    #Con el ciclo se calcula suma 10j p_j para j=0...s+a-1
    for(j in 0:(s+a-1)){
      rta = rta + (10*j*psubj[j+1])
    }

    rta = rta + (10*(s+a)*qsubj[s+a+1] - 4*a + 0.01*(a**2) - s )             #Se suma el resto.
    #Si a>0 se debe hacer el descuento de 2 que es el valor de realizar el pedido.
    if(a > 0){ rta = rta - 2  }
  }
  return(rta)
}
```

Figura 10: Modificación del problema.

El resultado numérico obtenido fue exactamente el mismo que en los casos anteriores, es decir la política descrita en (1).

3.6. Qué tanto afecta el valor del λ

Para analizar qué tanto afecta el valor del λ elegido, correremos el algoritmo de $Gauss - Sediel$ para varios valores de λ acercandose a 1. En particular, se correrá para diez valores de lambda: 0.5, 0.7, 0.8, 0.9, 0.99, 0.992, 0.994, 0.996, 0.998 y 0.999. Para cada caso se calculará la política óptima, el número de iteraciones y el valor de $(1-\lambda)V_\lambda$. Como este último es un vector con 100 entradas se pondrán en anexos al final del documento.

λ	0.5	0.7	0.8	0.9	0.99
Política óptima	(2) $\sigma = 9, \Sigma = 12$	(2)	(1)	(1)	(3) $\sigma = 11, \Sigma = 14$
Iteraciones	9	15	24	52	620
Tiempo ejecución (seg)	3.35	4.20	6.15	13.07	162.70

λ	0.992	0.994	0.996	0.998	0.999
Política óptima	(3)	(3)	(3)	(3)	(3)
Iteraciones	791	1109	1720	3631	7477
Tiempo ejecución (seg)	218.71	284.16	448.49	1394.09	2908.08

4. Discusión de resultado y conclusiones

De la primera parte de los resultados numéricos, podemos concluir que efectivamente el algoritmo de *Gauss-Sediel* es significativamente más eficiente que otros dos analizados. Este era un resultado esperado pues lo mismo ocurrió en el ejemplo del libro, cuya gráfica 6.3.1 de [P] es similar la que se obtuvo en la Figura 9. Es por esta razón que se eligió utilizar este algoritmo para las siguientes dos partes.

Del resultado obtenido en la sección 3.5 al cambiar la función de costo, se pudo evidenciar que el cambio que se hizo no fue tan significativo pues no cambió la política óptima obtenida. Diferente a este fue el resultado obtenido al cambiar el valor de λ en la sección 3.6. Se pudo evidenciar que no sólo hubo un cambio en la política óptima (hubo tres diferentes, (1), (2) y (3)), sino se vio un efecto importante en la eficiencia del algoritmo. Esto se puede evidenciar de forma más visual en la siguiente gráfica que muestra el valor de λ vs el tiempo de ejecución:

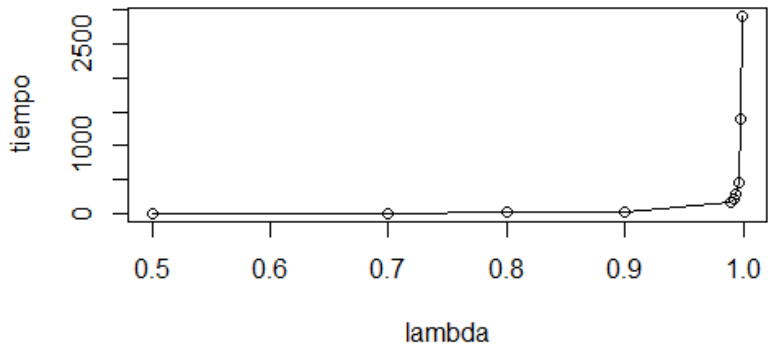

Figura 11: Valor de λ vs tiempo de ejecución (seg).

Este resultado de nuevo es esperado pues como vimos en clase, el caso más difícil de resolver es cuando $\lambda = 1$.

Por último, el resultado obtenido en el cálculo de $(1-\lambda)V_\lambda$ se puede ver que no difere significativamente con el cambio del valor del λ. De hecho, $||(1-0{,}5)V_{0,5} - (1-0{,}999)V_{0,999}|| = 17$, que es un valor alto pero no tan significativo como el efecto que se tuvo sobre el tiempo de ejecución y número de iteraciones.

5. Anexos

Vectores $(1-\lambda)V_\lambda$: Se pondrán únicamente las primeras 10 entradas para cada λ.

- $\lambda = 0{,}5$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 52,48 \\ 53,87 \\ 55,28 \\ 56,69 \\ 58,12 \\ 59,55 \\ 61,00 \\ 62,45 \\ 63,92 \\ 65,86 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}7$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 53,90 \\ 54,74 \\ 55,59 \\ 56,44 \\ 57,29 \\ 58,15 \\ 59,02 \\ 59,89 \\ 60,77 \\ 61,72 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}8$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 54,70 \\ 55,26 \\ 55,81 \\ 56,38 \\ 56,94 \\ 57,51 \\ 58,09 \\ 58,67 \\ 59,25 \\ 60,71 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}9$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 55,63 \\ 55,91 \\ 56,19 \\ 56,47 \\ 57,04 \\ 57,33 \\ 57,62 \\ 57,91 \\ 58,20 \\ 58,60 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}99$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 55,63 \\ 55,91 \\ 56,19 \\ 56,47 \\ 56,75 \\ 57,04 \\ 57,33 \\ 57,62 \\ 57,91 \\ 58,20 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}992$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 56,51 \\ 56,54 \\ 56,56 \\ 56,58 \\ 56,60 \\ 56,63 \\ 56,67 \\ 56,70 \\ 56,72 \\ 56,75 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}994$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 56,53 \\ 56,55 \\ 56,56 \\ 56,58 \\ 56,60 \\ 56,64 \\ 56,66 \\ 56,66 \\ 56,67 \\ 56,68 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}996$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 56,54 \\ 56,56 \\ 56,56 \\ 56,58 \\ 56,60 \\ 56,63 \\ 56,64 \\ 56,63 \\ 56,62 \\ 56,62 \\ \vdots \end{pmatrix}$$

- $\lambda = 0{,}998$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 56,58 \\ 56,59 \\ 56,59 \\ 56,59 \\ 56,60 \\ 56,60 \\ 56,61 \\ 56,61 \\ 56,61 \\ 56,62 \\ \vdots \end{pmatrix}$$

$\lambda = 0{,}999$

$$(1-\lambda)V_\lambda = \begin{pmatrix} 56,59 \\ 56,59 \\ 56,59 \\ 56,60 \\ 56,60 \\ 56,60 \\ 56,61 \\ 56,61 \\ 56,61 \\ 56,62 \\ \vdots \end{pmatrix}$$