Lance, Gabriel Pérez

Working Paper

Propagación de la información en un sistema interconectado de nodos

Serie Documentos de Trabajo, No. 686

Provided in Cooperation with:
University of CEMA, Buenos Aires

Suggested Citation: Lance, Gabriel Pérez (2019) : Propagación de la información en un sistema interconectado de nodos, Serie Documentos de Trabajo, No. 686, Universidad del Centro de Estudios Macroeconómicos de Argentina (UCEMA), Buenos Aires

This Version is available at:
https://hdl.handle.net/10419/203826

UNIVERSIDAD DEL CEMA
Buenos Aires
Argentina

Serie
DOCUMENTOS DE TRABAJO

Área: Ingeniería

PROPAGACIÓN DE LA INFORMACIÓN EN UN SISTEMA INTERCONECTADO DE NODOS

Gabriel Pérez Lance

Febrero 2019
Nro. 686

www.cema.edu.ar/publicaciones/doc_trabajo.html
UCEMA: Av. Córdoba 374, C1054AAP Buenos Aires, Argentina
ISSN 1668-4575 (impreso), ISSN 1668-4583 (en línea)
Editor: Jorge M. Streb; asistente editorial: Valeria Dowding <jae@cema.edu.ar>

Propagación de la información en un sistema interconectado de nodos

Gabriel Pérez Lance [(1)]

Resumen

En un sistema formado por una red de nodos emisores y receptores de información, es necesario conocer el estado de señal que tendría cada uno de esos nodos en todo momento con el fin de lograr un flujo de información adecuado.
El objetivo de este trabajo es obtener un modelo matemático que describa las interacciones entre los nodos, y a partir de dicho modelo, determinar las señales temporales en cada uno de ellos.
Para la elaboración del modelo se considera tiempo discreto, y para la obtención de las señales en los nodos se utiliza transformada Z.
Se aplica luego el modelo para resolver un problema concreto, y posteriormente se lleva a cabo una simulación con computadora sobre ese mismo problema. Se observa que las señales que devuelve el modelo y la que surgen de la simulación coinciden perfectamente.
Se podría adicionalmente ampliar el modelo para que tenga en cuenta los fenómenos de atenuación de las señales en la interacción entre nodos, y analizar cómo impacta dicha atenuación en la convergencia de las señales temporales.

1. Introducción

En un sistema de nodos que interactúan entre sí intercambiando información, resulta necesario administrar el flujo de la misma de manera tal que los nodos puedan procesarla sin que existan problemas de saturación, tanto debido a la cantidad de energía que reciben, como así también por la velocidad con que dicha información debe ser recibida, procesada y retransmitida.

La distribución de los nodos estará dada desde un punto de vista físico con una topología concreta. Con respecto al medio de interconexión entre ellos, podrá ser el espacio o bien un medio material como por ejemplo un cable coaxil, o un tendido de red, o cualquier otro.
Los retardos con que llega la información de un nodo a otro se deberán a la velocidad de propagación de la señal por el canal utilizado, y además deberemos sumar el tiempo de procesamiento del nodo emisor.

El sistema de nodos tendrá una condición inicial determinada por generadores de exitación asociados a cada uno de los nodos.
El objetivo es proponer y desarrollar un modelo que permita - dado un conjunto de nodos, sus respectivos tiempos de retardos, y un conjunto de generadores que determinen el estado inicial - conocer la señal temporal de cada nodo en cualquier momento.

[(1)] Los puntos de vista expresados en esta publicación son los del autor y no necesariamente los de la Universidad del CEMA.

2. Desarrollo

Consideremos un nodo transmisor de señal en determinado punto del espacio. En cierto momento dicho nodo emite un pulso. Este pulso viaja por el canal físico utilizado, hasta que alcanza a otro nodo. Luego, ese nodo toma la información del pulso recibido y emite a su vez otro pulso que viaja hacia los demás nodos, y así recursivamente. Estos pulsos pueden representar señales digitales o analógicas.

En nuestro enfoque vamos a considerar señales de tiempo discreto descriptas mediante funciones $x_j: \mathbb{N}_0 \to \mathbb{R}$
Si definimos con x_i a la señal presente en cada nodo *i* en el tiempo discreto *n*, y siendo m_{ij} el tiempo que le requiere a un pulso ir desde el nodo *j* hasta el nodo *i* (debido a retardos por la propagación en base al tipo y características del canal, o bien por el tiempo que le requiere al nodo j procesar la información a emitir), entonces para todo *n*, se va a verificar el siguiente sistema que permite modelizar el comportamiento de las señales en cada uno de los nodos:

$$\begin{cases} x_1[n] = x_2[n-m_{12}] + x_3[n-m_{13}] + \ldots + x_k[n-m_{1k}] + \ldots + x_r[n-m_{1r}] + g_1[n] \\ x_2[n] = x_1[n-m_{21}] + x_3[n-m_{23}] + \ldots + x_k[n-m_{2k}] + \ldots + x_r[n-m_{2r}] + g_2[n] \\ x_3[n] = x_1[n-m_{31}] + x_2[n-m_{32}] + \ldots + x_k[n-m_{3k}] + \ldots + x_r[n-m_{3r}] + g_3[n] \\ \ldots \\ x_k[n] = x_1[n-m_{k1}] + x_2[n-m_{k2}] + x_k[n-m_{k3}] + \ldots + x_r[n-m_{kr}] + g_k[n] \\ \ldots \\ x_r[n] = x_1[n-m_{r1}] + x_2[n-m_{r2}] + \ldots + x_k[n-m_{rk}] + \ldots + x_{r-1}[n-m_{r\,r-1}] + g_r[n] \end{cases} \quad (1)$$

donde $g_i[n]$ es la función que representa la generación de señal en cada nodo. Para el caso en que el nodo 1 fuera el único que emite un pulso inicial, entonces $g_1[n]$ sería la delta de Dirac, y el resto de las $g_i[n]$ serían nulas.

Consideraremos que el sistema es causal, es decir que la señal en cualquier nodo sólo depende del estado actual o anterior de todos ellos.
Por tratarse de un sistema lineal de ecuaciones en diferencias, las soluciones serán combinaciones lineales de autofunciones que tendrán un crecimiento no mayor al exponencial, por lo tanto existitrá una región de convergencia para la sumatoria que define a la transformada Z.
Aplicando entonces transformada Z al sistema (1), obtenemos:

$$\begin{cases} X_1(z) = z^{-m_{12}} X_2(z) + z^{-m_{13}} X_3(z) + \ldots + z^{-m_{1k}} X_k(z) + \ldots + z^{-m_{1r}} X_r(z) + G_1(z) \\ X_2(z) = z^{-m_{21}} X_1(z) + z^{-m_{23}} X_3(z) + \ldots + z^{-m_{2k}} X_k(z) + \ldots + z^{-m_{2r}} X_r(z) + G_2(z) \\ X_3(z) = z^{-m_{31}} X_1(z) + z^{-m_{32}} X_2(z) + \ldots + z^{-m_{3k}} X_k(z) + \ldots + z^{-m_{3r}} X_r(z) + G_3(z) \\ \ldots \\ X_k(z) = z^{-m_{k1}} X_2(z) + z^{-m_{k2}} X_2(z) + z^{-m_{k3}} X_3(z) + \ldots + z^{-m_{kr}} X_r(z) + G_k(z) \\ \ldots \\ X_r(z) = z^{-m_{r1}} X_1(z) + z^{-m_{r2}} X_2(z) + \ldots + z^{-m_{rk}} X_k(z) + \ldots + z^{-m_{r\,r-1}} X_{r\,r-1}(z) + G_r(z) \end{cases} \quad (2)$$

Si definimos las matrices:

$$A_{(z)}=\begin{bmatrix} 0 & z^{-m_{12}} & z^{-m_{13}} & \dots & z^{-m_{1k}} & \dots & z^{-m_{1r}} \\ z^{-m_{21}} & 0 & z^{-m_{23}} & \dots & z^{-m_{2k}} & \dots & z^{-m_{2r}} \\ z^{-m_{31}} & z^{-m_{32}} & 0 & \dots & z^{-m_{3k}} & \dots & z^{-m_{3r}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ z^{-m_{k1}} & z^{-m_{k2}} & z^{-m_{k3}} & \dots & \dots & z^{-m_{kr-1}} & z^{-m_{kr}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ z^{-m_{r1}} & z^{-m_{r2}} & z^{-m_{r3}} & \dots & \dots & z^{-m_{rr-1}} & 0 \end{bmatrix} \quad (3)$$

$$X_{(z)}=\begin{bmatrix} X_1(z) \\ X_2(z) \\ X_3(z) \\ \dots \\ X_k(z) \\ \dots \\ X_r(z) \end{bmatrix} \quad (4a) \qquad \text{y} \qquad G_{(z)}=\begin{bmatrix} G_1(z) \\ G_2(z) \\ G_3(z) \\ \dots \\ G_k(z) \\ \dots \\ G_r(z) \end{bmatrix} \quad (4b)$$

entonces el sistema (2) se puede escribir matricialmente como $X_{(z)}=A_{(z)}.X_{(z)}+G_{(z)}$ (5)

por lo tanto, $X_{(z)}-A_{(z)}.X_{(z)}=G_{(z)}$, y entonces $I.X_{(z)}-A_{(z)}.X_{(z)}=G_{(z)}$

=> $(I-A_{(z)}).X_{(z)}=G_{(z)}$

y definiendo $B_{(z)}=I-A_{(z)}$ podemos escribir: $B_{(z)}.X_{(z)}=G_{(z)}$ (6)

siendo:

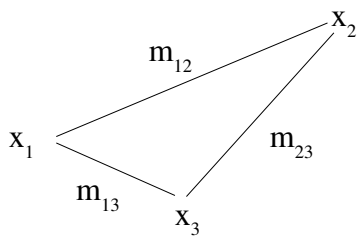
$$B_{(z)}=\begin{bmatrix} 1 & -z^{-m_{12}} & -z^{-m_{13}} & \dots & -z^{-m_{1k}} & \dots & -z^{-m_{1r}} \\ -z^{-m_{21}} & 1 & -z^{-m_{23}} & \dots & -z^{-m_{2k}} & \dots & -z^{-m_{2r}} \\ -z^{-m_{31}} & -z^{-m_{32}} & 1 & \dots & -z^{-m_{3k}} & \dots & -z^{-m_{3r}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ -z^{-m_{k1}} & -z^{-m_{k2}} & z^{-m_{k3}} & \dots & \dots & -z^{-m_{kr-1}} & -z^{-m_{kr}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ -z^{-m_{r1}} & -z^{-m_{r2}} & -z^{-m_{r3}} & \dots & \dots & -z^{-m_{rr-1}} & 1 \end{bmatrix} \quad (7)$$

Por lo tanto, dadas las excitaciones $G_{(z)}$, bastará con resolver el sistema lineal (6), y luego obtener la transformada inversa Z para conocer las señales en todos los nodos para cualquier *n*.

3. Aplicación a un sistema de tres nodos con características isotrópicas

El sistema bajo estudio consta de los nodos 1, 2 y 3. El nodo 1 emite un pulso en el momento inicial, y luego el sistema comienza a recibir la información y emitir nuevos pulsos.

El siguiente grafo representa el sistema considerado:



asumiremos que $m_{ij} = m_{ji}$ por tratarse de un sistema isotrópico

Entonces, por lo explicado en la sección anterior y según (1), podemos escribir:

$$\begin{aligned} x_1[n] &= x_2[n-5] + x_3[n-1] + \delta[n] \\ x_2[n] &= x_1[n-5] + x_3[n-3] \\ x_3[n] &= x_1[n-1] + x_2[n-3] \end{aligned} \qquad (8)$$

por lo tanto, si aplicamos tranformada Z, se obtiene el sistema:

$$\begin{aligned} X_1(z) &= z^{-5} X_2(z) + z^{-1} X_3(z) + 1 \\ X_2(z) &= z^{-5} X_1(z) + z^{-3} X_3(z) \\ X_3(z) &= z^{-1} X_1(z) + z^{-3} X_2(z) \end{aligned} \qquad (9)$$

resolviendo dicho sistema obtenemos:

$$\begin{aligned} X_1(z) &= z^4 \frac{z^5 - z^4 + z^3 - z^2 + z - 1}{z^9 - z^8 - z^3 + z^2 - z - 1} \\ X_2(z) &= \frac{z^5}{z^9 - z^8 - z^3 + z^2 - z - 1} \\ X_3(z) &= z^2 \frac{z^6 - z^5 + z^4 - z^3 + z^2 - z + 1}{z^9 - z^8 - z^3 + z^2 - z - 1} \end{aligned} \qquad (10)$$

Las aproximaciones numéricas de las raices del denominador (polos) de estas expresiones son:

$$
\begin{aligned}
p_1 &= -1.0 \\
p_2 &= -0.5478998554250795 \\
p_3 &= -0.5584757873686504 + 0.9050397522824599\,i \\
p_4 &= -0.5584757873686504 - 0.9050397522824599\,i \\
p_5 &= 0.2992477778422136 + 0.9708722639349872\,i \\
p_6 &= 0.2992477778422136 - 0.9708722639349872\,i \\
p_7 &= 0.8747274032272649 + 0.649702931158183\,i \\
p_8 &= 0.8747274032272649 - 0.649702931158183\,i \\
p_9 &= 1.316901068023423
\end{aligned}
\qquad (11)
$$

Siendo $X_1(z) = \frac{P(z)}{Q(z)}$ una función racional, y dado que además tiene todos polos de primer orden, entonces los residuos en cada uno de sus polos, se pueden calcular como

$$Res[p_j] = \lim_{z \to p_j} \frac{P(z)}{Q'(z)} \qquad (12)$$

de este modo, los residuos en cada polo p_j resultan:

$$
\begin{aligned}
Res[p_1] &\approx -0.5454545454545454 \\
Res[p_2] &\approx 0.0691465403217533 \\
Res[p_3] &\approx -0.03319910091206479 + 0.004036283082005585\,i \\
Res[p_4] &\approx -0.03319910091206479 + 0.004036283082005585\,i \\
Res[p_5] &\approx -0.01006924308272961 + 0.07442317890963517\,i \\
Res[p_6] &\approx -0.01006924308272961 - 0.07442317890963517\,i \\
Res[p_7] &\approx 0.1618838583363631 - 0.01165380087630015\,i \\
Res[p_8] &\approx 0.1618838583363631 + 0.01165380087630015\,i \\
Res[p_9] &\approx 0.2390769764496538
\end{aligned}
\qquad (13)
$$

El grado de *P(z)* no es menor que el grado de *Q(z)*, por lo tanto se puede escribir

$$X_1(z) = C(z) + \frac{R(z)}{Q(z)} \qquad (14)$$

luego, aplicando fracciones simples sobre $\frac{R(z)}{Q(z)}$, tenemos que

$$\frac{R(z)}{Q(z)} = \frac{A_1}{z-p_1} + \frac{A_2}{z-p_2} + ... + \frac{A_r}{z-p_r} \quad (15)$$ con A_j: residuo de $X_1(z)$ en el polo p_j

pero $$\frac{A_j}{z-p_j} = \frac{z^{-1}A_j}{1-p_j z^{-1}} \quad (16)$$ además, para funciones causales, se cumple que

$$u[n]\alpha^n \overset{z}{\rightarrow} \frac{1}{1-\alpha z^{-1}} \quad (17)$$

y por la propiedad del desplazamiento en el tiempo $$x[n-m] \overset{z}{\rightarrow} z^{-m}.X(z) \quad (18)$$

$$\Rightarrow \quad \frac{z^{-1}A_j}{1-p_j z^{-1}} \overset{Z^{-1}}{\rightarrow} A_j\, p_j^{n-1} u[n-1] \quad (19)$$

aplicando linealidad podemos obtener la transformada Z inversa de $\frac{R(z)}{Q(z)}$:

$$\frac{R(z)}{Q(z)} \overset{Z^{-1}}{\rightarrow} \sum_{j=1}^{r} A_j\, p_j^{n-1}\, u[n-1] \quad (20)$$

u[n] es la función de Heaviside, o escalón unitario, en tiempo discreto

Finalmente, siendo *C(z)=1*, entonces su transformada Z inversa es la delta de dirac de tiempo discreto $\delta[n]$

Así, de acuerdo con (14) y aplicando Z^{-1}, la señal temporal en el nodo 1 es:

$$x_1[n] = \delta[n] + u[n-1] \sum_{j=1}^{r} A_j\, p_j^{n-1} \quad (21)$$

Partiendo de $X_2(z)$ y $X_3(z)$ y procediendo de modo similar se obtienen expresiones para $x_2[n]$ y $x_3[n]$, que describen las señales temporales en los nodos respectivos.

4. Resultados

Se utiliza el modelo matemático desarrollado reemplazando en (21) el valor de los polos y residuos calculados anteriormente, obteniéndose la secuencia $x_1[n]$:

1, -9.992007221626409 10^{-16}, 0.9999999999999998, 9.992007221626409 10^{-16},
1.000000000000002, 5.051514762044462 10^{-15}, 1.000000000000011,
2.065014825802791 10^{-14}, 2.000000000000034, 2.000000000000048,
4.000000000000062, 4.000000000000083, 6.000000000000115, 6.000000000000164,
9.000000000000238, 10.00000000000035, 14.00000000000052, 18.00000000000077,
26.00000000000107, 34.00000000000144, 47.00000000000196, 60.00000000000276,
79.00000000000371, 102.000000000005, 134.0000000000069, 174.00000000001,
231.0000000000131, 304.0000000000186, 404.0000000000256, 534.0000000000339,
705.000000000046, 926.0000000000641, 1219.000000000088, 1600.000000000122,
2105.00000000016 (22a)

análogamente $x_2[n]$:

0, -2.609024107869118 10^{-15}, -3.33066907387547 10^{-15}, -3.608224830031759 10^{-15},
0.9999999999999969, 0.9999999999999969, 0.9999999999999969, 0.999999999999998,
0.9999999999999998, 1.000000000000004, 2.000000000000013, 2.00000000000003,
3.000000000000052, 5.00000000000008, 7.000000000000113, 9.000000000000153,
12.00000000000021, 14.00000000000031, 18.00000000000046, 24.0000000000007,
31.00000000000101, 41.0000000000014, 56.00000000000207, 74.00000000000294,
99.00000000000401, 131.0000000000053, 170.000000000008, 222.0000000000106,
292.0000000000142, 382.0000000000207, 504.0000000000289, 666.000000000038,
878.0000000000538, 1160.000000000071, 1531.000000000102 (22b)

y la secuencia $x_3[n]$:

0, 0.9999999999999987, -1.609823385706477 10^{-15}, 0.9999999999999979,
-1.575128916186941 10^{-15}, 0.9999999999999991, 1.609823385706477 10^{-15},
2.000000000000008, 1.000000000000018, 3.00000000000003, 3.000000000000045,
5.000000000000062, 5.000000000000088, 8.00000000000013, 8.000000000000194,
12.00000000000029, 15.00000000000044, 21.00000000000064, 27.00000000000091,
38.00000000000125, 48.00000000000176, 65.00000000000249, 84.00000000000338,
110.0000000000046, 143.0000000000065, 190.0000000000092, 248.0000000000124,
330.0000000000175, 435.0000000000243, 574.0000000000325, 756.000000000045,
997.000000000062, 1308.000000000084, 1723.000000000115, 2266.000000000157
(22c)

Se lleva a cabo ahora una simulación en computadora, implementada en Python, para emular el problema planteado y observar cuales serían las secuencias que se obtendrían en un contexto real. La simulación representa el comportamiento que tendría el sistema atendiendo a su descripción básica, y no en base al modelo obtenido. La intención es utilizar esta simulación para poder contrastar los resultados que se obtuvieron utilizando el modelo.

La simulación realizada devuelve las siguientes secuencias:

nodo 1: 1, 0, 1, 0, 1, 0, 1, 0, 2, 2, 4, 4, 6, 6, 9, 10, 14, 18, 26, 34, 47, 60, 79, 102, 134, 174, 231, 304, 404, 534, 705, 926, 1219, 1600, 2105 (23a)

nodo 2: 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 3, 5, 7, 9, 12, 14, 18, 24, 31, 41, 56, 74, 99, 131, 170, 222, 292, 382, 504, 666, 878, 1160, 1531 (23b)

nodo 3: 0, 1, 0, 1, 0, 1, 0, 2, 1, 3, 3, 5, 5, 8, 8, 12, 15, 21, 27, 38, 48, 65, 84, 110, 143, 190, 248, 330, 435, 574, 756, 997, 1308, 1723, 2266 (23c)

El código del programa utilizado para la simulación se encuentra en el Anexo I.

5. Análisis de los resultados obtenidos y conclusiones

A continuación estudiaremos la propagación de errores en el modelo debido a la utilización de aritmética finita en punto flotante. También buscamos una cota del error presente en las secuencias devueltas por el modelo utilizado.

La expresión a^n, tiene una cota en el error absoluto según:

$$\Delta(a^n) = n\ a^{n-1}\ \Delta a + a^n\ (n-1)\ u \qquad (24)$$

donde Δa : cota del error absoluto de a

u : eps de la máquina de punto flotante utilizada, $u = eps = 0.5\ 10^{1-t}$ (25)

t : número de dígitos de la mantisa

además la cota del error absoluto en un producto es:

$$\Delta(A.B) = B.\Delta(A) + A.\Delta(B) + u.A.B \qquad (26)$$

(24) y (26) => $\Delta(A.p^n) = p^n.\Delta(A) + A.n.p^{n-1}.\Delta(p) + A.p^n\ u$ (27)

además una cota del error en $\sum_{j=1}^{r} Y_j$ es:

$$\Delta\left(\sum_{j=1}^{r} Y_j\right) = \sum_{j=1}^{r} \Delta(Y_j)\ +\ (Y_r + 2\ Y_{r-1} + 3\ Y_{r-2} + ... + (r-1)\ Y_1).u \qquad (28)$$

=>

$$\Delta\left(\sum_{j=1}^{r} Y_j\right) \le \sum_{j=1}^{r} \Delta(Y_j)\ +\ u\ (r-1) \sum_{j=1}^{r} Y_j \qquad (29)$$

en el caso particular de nuestro modelo, en la (21) sólo debemos considerar la parte real de la sumatoria, por lo tanto

$$Y_j = |A_j|.|p_j|^n \cos(\phi_j + n\ \psi_j) \qquad (30) \qquad \text{con} \quad \begin{matrix} \phi_j = Arg(A_j) \\ \psi_j = Arg(p_j) \end{matrix} \qquad (31)$$

entonces aplicando propagación de errores y punto flotante, las cotas para los argumentos son:

$$\Delta(\phi_j)=\frac{3.u.\Im(A_j)/\Re(A_j)}{1+\Im^2(A_j)/\Re^2(A_j)}+u.\phi_j$$
$$\Delta(\psi_j)=\frac{3.u.\Im(p_j)/\Re(p_j)}{1+\Im^2(p_j)/\Re^2(p_j)}+u.\psi_j \qquad (32)$$

además $\Delta(\phi_j+n\ \psi_j)=\Delta(\phi_j)+\Delta(n\ \psi_j)+u.(\phi_j+n\ \psi_j)$ (33)

=> $\Delta(\cos(\phi_j+n\ \psi_j))=\sin(\phi_j+n\ \psi_j).\Delta(\phi_j+n\ \psi_j)+u.\cos(\phi_j+n\ \psi_j)$ (34)

Utilizando un *eps* del orden de 10^{-16} , considerando las fórmulas de las cotas obtenidas y de acuerdo con los valores de residuos y polos en el caso bajo estudio, encontramos que 10^{-9} es una cota superior del error de Y_j considerando el caso más desfavorable de todos los polos y residuos. Finalmente, según (29) entonces 10^{-8} es una cota para *x[n]*.

Se observa entonces que los resultados arrojados por el modelo coinciden perfectamente, dentro de los márgenes de error, con los que devuelve la simulación. Esto representa un respaldo en favor del enfoque utilizado para desarrollar el modelo.

6. Anexo I

Código (Pyhton) utilizado para la simulación

```
#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

class Pulse:

    def __init__(self,sourceNode,asignedNode, emitionTimeCoordinate):

        self.sourceNode = sourceNode
        self.asignedNode = asignedNode
        self.emitionTimeCoordinate = emitionTimeCoordinate
        self.timeRemainingForArrival = self.travelDuration()

#---------------------------------------------------------------------

    def getEmitionTimeCoordinate(self):

        return self.emitionTimeCoordinate

#---------------------------------------------------------------------

    def travelDuration(self):

        travelDuration = self.sourceNode.distanceWithRespectTo(self.asignedNode)

        return travelDuration
```

```
#-----------------------------------------------------------------------

    def arrivalTimeCoordinate(self):

        emitionTimeCoordinate = self.getEmitionTimeCoordinate()
        timeGap = self.travelDuration()

        arrivalTimeCoordinate = timeGap + emitionTimeCoordinate

        return arrivalTimeCoordinate

#-----------------------------------------------------------------------

    def actualize(self):

        self.timeRemainingForArrival = self.timeRemainingForArrival - 1

        if self.timeRemainingForArrival == 0:
            self.asignedNode.recive(self)

#-----------------------------------------------------------------------

    def erase(self):

        self.sourceNode.erase(self)


#|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

class Secuence:

    def __init__(self):

        self.timeLine = {}

#-----------------------------------------------------------------------

    def addOnePulseOn(self,time):

        if time in self.timeLine:
            self.timeLine[time] += 1
        else:
            self.timeLine[time] = 1

#-----------------------------------------------------------------------

    def actualize(self,pulse):

        arrivalTimeCoordinate = pulse.arrivalTimeCoordinate()

        self.addOnePulseOn(arrivalTimeCoordinate)

#-----------------------------------------------------------------------

    def displayUpTo(self,time):
```

```
        representation = ""

        for i in range(time):

            if i in self.timeLine:
                representation += str(self.timeLine[i]) + " ,"
            else:
                representation += "0, "

        representation = representation[:-2]

        print(representation)

#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

class Node:

    def __init__(self,name,distances,initialTime = 0):

        self.initialTime = 0
        self.name = name
        self.nodes = None
        self.distances = distances
        self.secuence = Secuence()
        self.pulsesToEmit = []
        self.pulsesOnCourse = []

#-----------------------------------------------------------------------

    def addNodes(self,nodes):

        self.nodes = nodes

#-----------------------------------------------------------------------

    def prepareForInitialEmition(self):

        for node in self.nodes:

            self.pulsesOnCourse.append(Pulse(self,node,0))

        self.secuence.addOnePulseOn(0)

#-----------------------------------------------------------------------

    def actualize(self):

        for pulse in list(self.pulsesOnCourse):
            pulse.actualize()

#-----------------------------------------------------------------------

    def turnPulseTowards(self,pulse,node):

        arrivalTimeCoordinate = pulse.arrivalTimeCoordinate()

        turnedPulse = Pulse(pulse.asignedNode,node,arrivalTimeCoordinate)
```

```
        return turnedPulse

#---------------------------------------------------------------------

    def recive(self,pulse):

        self.secuence.actualize(pulse)

        for node in self.nodes:

            newPulse = self.turnPulseTowards(pulse,node)
            self.pulsesOnCourse.append(newPulse)

        pulse.erase()

#---------------------------------------------------------------------

    def displaySecuenceUpTo(self,time):

        print(self.name)
        self.secuence.displayUpTo(time)
        print()

#---------------------------------------------------------------------

    def distanceWithRespectTo(self,asignedNode):

        distance = self.distanceWith(asignedNode)

        return distance

#---------------------------------------------------------------------

    def distanceWith(self,anotherNode):

        for nodePair in self.distances:

            if (nodePair == (self.getName(), anotherNode.getName()) or
                nodePair == (anotherNode.getName(), self.getName())):
                return self.distances[nodePair]

#---------------------------------------------------------------------

    def erase(self,pulse):

        self.pulsesOnCourse.remove(pulse)

#---------------------------------------------------------------------

    def getName(self):

        return self.name

#|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

class Simulation:
```

```
    def __init__(self):

        self.amountOfTime = None
        self.amountOfNodes = None
        self.distances = {}
        self.nodes = []

#---------------------------------------------------------------------

    def setAmountOfTime(self,amount):

        self.amountOfTime = amount

#---------------------------------------------------------------------

    def setAmountOfNodes(self,amount):

        self.amountOfNodes = amount

#---------------------------------------------------------------------

    def setDistances(self):

        for i in range(1,self.amountOfNodes+1):
            for j in range(1,self.amountOfNodes+1):

                if i == j:
                    self.distances[(str(i),str(j))] = 0
                    continue

                if ((not (str(i),str(j)) in self.distances) and
                    (not (str(j),str(i)) in self.distances)):

                    print("distance between {} and {}: ".format(i,j))
                    self.distances[(str(i),str(j))] = int(input())

#---------------------------------------------------------------------

    def initializeNodes(self):

        for i in range(1,self.amountOfNodes+1):
            self.nodes.append(Node(str(i),self.distances))

        for node in self.nodes:

            listOfNodesToAssign = list(self.nodes)
            listOfNodesToAssign.remove(node)
            node.addNodes(listOfNodesToAssign)


#---------------------------------------------------------------------

    def setStartingNode(self,name):

        startingNode = self.findNodeWithName(str(name))
        startingNode.prepareForInitialEmition()
```

```
#------------------------------------------------------------------------

    def findNodeWithName(self,name):

        for node in self.nodes:

            if node.getName() == name:
                return node

#------------------------------------------------------------------------

    def execute(self):

        print("loading",end = "")
        for second in range(self.amountOfTime):
            for node in self.nodes:
                node.actualize()
            print(".",end = "")
        print()
        print()

        for node in self.nodes:
            node.displaySecuenceUpTo(self.amountOfTime)

#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||

simulation = Simulation()
simulation.setAmountOfTime(35)
simulation.setAmountOfNodes(3)
simulation.setDistances()
simulation.initializeNodes()
simulation.setStartingNode(1)
simulation.execute()
```

7. Referencias

Ruel V. Churchil, James Ward Brown, Variable Compleja y Aplicaciones, 5ta edicion, cap 5 y 6, McGraw-Hill, 2004. 7a.

Murray R. spiegel, Variable Compleja, cap 6 y 7, McGraw-Hill, 2005. 1a.

Oppenheim, Alan V. , Willsky, Alan S, Nawab, S. Hamid. Signals and Systems. 2a ed. cap 10 México: Pretince Hall.

Burden, Richard L., Faires, J. Douglas, Análisis Numérico, 2002. 7a. Thompson Learning, México.