ALGORITMO PARA EL CÁLCULO DE LA EXPRESIÓN DE CUMULANTES DE ORDEN *N*

Jayro Barrera* y Fidel Hernandez**[1]
*University of Pinar del Rio.
**Technological University of Havana, Cujae.

ABSTRACT
An algorithm, focused on determining the terms comprised in the expression for computing n-th order cumulants, is proposed. This work presents some considerations about the used algorithms, as well as the results obtained by means of algorithm's implementation. At the end, the results obtained from the computation of the 8-th order cumulant terms, for a telecommunication application, are shown.

KEYWORDS: Algorithm, Cumulant, Mathematical Expression

MSC: 65T99, 65Y04

RESUMEN
En este trabajo se propone un algoritmo computacional para la determinación de los términos que componen la expresión de un cumulante de cualquier orden. En el trabajo se aporta una valoración acerca de la eficiencia de los algoritmos utilizados así como los resultados alcanzados a través de la implementación de los mismos. Finalmente se ofrecen los resultados obtenidos al determinar los términos de la expresión de cumulante de octavo orden, aplicado a un problema del área de las telecomunicaciones.

PALABRAS CLAVE: Algoritmo, Cumulante, Expresión Matemática

1. INTRODUCTION

Cada vez más, el análisis estadístico de orden superior se viene aplicando en diversos campos de la ciencia y la tecnología; por ejemplo, telecomunicaciones, sonares, radares, la biomedicina, procesamiento de datos sísmicos, reconstrucción de imágenes, estimación de retraso en el tiempo, filtrado adaptivo, ecualización ciega, entre otros. Estas herramientas estadísticas, relacionadas con las características estadísticas conocidas como cumulantes, y sus asociadas transformadas de Fourier, conocidas como poliespectro, no solo revelan información de amplitud, sino también información de fase. Esto es importante porque, como es conocido, las características estadísticas de segundo orden son ciegas a la fase (Mendel (1991), Cabezas, Cruz, Iglesias y Hernández (2013)).
Aunque está bien definida la ecuación general de cálculo de los cumulantes de diferentes órdenes (Mendel (1991)), se puede decir que en la medida que dichos órdenes se incrementan, la complejidad de la expresión correspondiente a cada orden particular se extiende y complejiza, lo cual puede conducir a errores en el desarrollo del trabajo manual algebraico orientado a su definición.
En este trabajo se propone un método dirigido a la automatización del proceso de generación de la expresión de un cumulante a partir de un orden dado. Para esto, se expone la jerarquía algorítmica seguida, una valoración acerca de la eficiencia de los algoritmos utilizados así como los resultados alcanzados a través de la implementación de los mismos.

2. MOMENTOS Y CUMULANTES

Sea x una colección de variables aleatorias, por ejemplo, $x = col(x_1, x_2, \dots, x_k)$, siendo k la cantidad de variables, I_x denota el conjunto de índices de x. Si $I \subseteq I_x$, entonces se dice que x_I es el vector integrado por las componentes de x cuyos índices pertenecen a I. De esta forma se puede denotar el momento y el cumulante de un subvector x_I del vector x como $m_x(I)$, donde $m_x(I)$ es el valor esperado del producto de los elementos de x_I, y $c_x(I)$, respectivamente. Se define como partición del conjunto I a una colección de subconjuntos I_p no ordenados, no vacíos y sin elementos en común, tales que $\bigcup_p I_p = I$. Por ejemplo, el

[1]fhernandez@tele.cujae.edu.cu.

conjunto de particiones correspondientes a $k = 3$, es: {(1, 2, 3)}, {(1), (2, 3)}, {(2), (1, 3)}, {(3), (1, 2)}, {(1), (2), (3)}. La relación momento a cumulante se define a través de la expresión (Ahmadi y Berangi (2009), Azarbad, Hakimi y Ebrahimzadeh (2012)):

$$c_x(I) = \sum_{\bigcup_{p=1}^{q} I_p = I} (-1)^{q-1}(q-1)! \prod_{p=1}^{q} m_x(I_p) \tag{1}$$

con $1 \leq q \leq k$, donde $\bigcup_{p=1}^{q} I_p = I$ denota las sumas sobre las particiones de I. Para el caso ilustrativo anterior ($k = 3$) se tiene $q = 1$ para {(1, 2, 3)}, $q = 2$ para {(1), (2, 3)}, {(2), (1, 3)}, {(3), (1, 2)}, y $q = 3$ para {(1), (2), (3)}.

De manera concreta, se definirá el momento de un proceso complejo X de orden k como (Azarbad, Hakimi y Ebrahimzadeh (2012), Li, Zhang, Wang, Xu y Xu (2012)):

$$m_{kl,X} = E[X^{k-l}\bar{X}^k] \tag{2}$$

donde $\bar{X}$ representa el valor complejo conjugado del proceso X complejo, estocástico y de media cero; l es la cantidad de veces que se tiene a $\bar{X}$ como argumento; y $E[.]$ es el operador estadístico valor esperado. De forma análoga se define el cumulante para un proceso complejo, estocástico, de media cero y orden k como (Azarbad, Hakimi y Ebrahimzadeh (2012), Li, Zhang, Wang, Xu y Xu (2012)):

$$c_{kl,X} = cum[X^{k-l}\bar{X}^k] \tag{3}$$

donde el operador $cum[.]$ es definido a través de la ecuación (1).

3. ENTORNO DE PROGRAMACIÓN

El entorno de programación utilizado para llevar a cabo la implementación de los algoritmos que se presentan en esta investigación es el software MATLAB. El mismo es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje propio (lenguaje M), disponible para las plataformas Unix, Windows y Apple Mac OS X. Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuarios (GUI) y la comunicación con programas de otros lenguajes y con otros dispositivos hardware.

MATLAB dispone de varias bibliotecas especializadas (toolbox) facilitando al usuario algoritmos implementados inherentes a la tarea que se pretende realizar. Presenta untiempo de ejecución para algunas operaciones muy rápido, en aquellas en que puede aprovechar sus capacidades de vectorización. A partir de la versión 6.5 se incorporó un acelerador JIT que mejora significativamente la velocidad de ejecución de los ficheros *.m en ciertas circunstancias. Para el desarrollo de esta investigación se utilizó la versión 7.9.0.529 (R2009b).

4. IMPLEMENTACIÓN EN MATLAB DE ALGORITMO PARA GENERAR EXPRESIÓN DE CUMULANTE DE UN ORDEN DADO

El proceso de diseño del algoritmo a ser implementado se llevó a cabo a partir del análisis de la expresión (1). Por ejemplo, el cálculo de forma manual, utilizando (1) para determinar la expresión del cumulante de tercero y cuarto orden, arroja los resultados que se muestran en Tabla 1 y Tabla 2, respectivamente. Nótese en ambos casos que si se tratara de una sola variable estocástica entonces: $x_1 = x_2 = x_3 = x_4 = x$ y se obtendrían las siguientes expresiones al sumar los términos generados por cada partición:

$$c_3 = m_3 - 3m_2m_1 + 2m_1^3$$
$$c_4 = m_4 - 4m_3m_1 - 3m_2^2 + 12m_2m_1^2 - 6m_1^4$$

Tabla 1. Cálculo manual de expresión del cumulante de tercer orden

q	Particiones I_p	Coeficientes $(-1)^{q-1}(q-1)!$	Expresión generada por la Partición
1	{(1,2,3)}	1	$m(x_1x_2x_3)$

2	{(1), (2,3)}	-1	$-m(x_1)(x_2x_3)$
2	{(2), (1,3)}	-1	$-m(x_2)m(x_1x_3)$
2	{(3), (1,2)}	-1	$-m(x_3)m(x_1x_2)$
3	{(1), (2), (3)}	2	$2m(x_1)m(x_2)m(x_3)$

Tabla 1. Cálculo manual de expresión del cumulante de cuarto orden

q	Particiones I_p	Coeficientes $(-1)^{q-1}(q-1)!$	Expresión generada por la Partición
1	{(1,2,3,4)}	1	$m(x_1x_2x_3x_4)$
2	{(1), (2,3,4)}	-1	$-m(x_1)(x_2x_3x_4)$
2	{(2), (1,3,4)}	-1	$-m(x_2)m(x_1x_3x_4)$
2	{(3), (1,2,4)}	-1	$-m(x_3)m(x_1x_2x_4)$
2	{(4), (1,2,3)}	-1	$-m(x_4)m(x_1x_2x_3)$
2	{(1,2), (3,4)}	-1	$-m(x_1x_2)m(x_3x_4)$
2	{(1,3), (2,4)}	-1	$-m(x_1x_3)m(x_2x_4)$
2	{(1,4), (2,3)}	-1	$-m(x_1x_4)m(x_2x_3)$
3	{(1), (2), (3,4)}	2	$2m(x_1)m(x_2)m(x_3x_4)$
3	{(1), (3), (2,4)}	2	$2m(x_1)m(x_3)m(x_2x_4)$
3	{(1), (4), (2,3)}	2	$2m(x_1)m(x_4)m(x_2x_3)$
3	{(2), (3), (1,4)}	2	$2m(x_2)m(x_3)m(x_1x_4)$
3	{(2), (4), (1,3)}	2	$2m(x_2)m(x_4)m(x_1x_3)$
3	{(3), (4), (1,2)}	2	$2m(x_3)m(x_4)m(x_1x_2)$
4	{(1), (2), (3), (4)}	-6	$-6m(x_1)m(x_2)m(x_3)m(x_4)$

4.1.Módulos implementados y su jerarquía

De los resultados anteriores se constatan dos etapas importantes del proceso de obtención de los cumulantes a partir de (1): la generación de las particiones de I_x en los conjuntos I_p (ver segunda columna de Tabla 1 y Tabla 2), y la obtención de la expresión analítica que genera la partición (ver cuarta columna de Tabla 1 y Tabla 2). Estas etapas fueron desarrolladas a través de la implementación de nueve módulos, en su mayoría funciones sobre MATLAB. La jerarquía bajo las cuales se rigieron se ilustra en la Figura 1.

La función *cumpart2strn* es la encargada de llevar a cabo el proceso de convertir las particiones de I_x (ver segunda columna de Tabla 1 y Tabla 2) en la notación seguida en la cuarta columna de Tabla 1 y Tabla 2. A partir de este proceso se obtiene la expresión del cumulante que se desea calcular. Para esto, la salida de la función es un dato tipo *string* con el objetivo de facilitar el trabajo con los cumulantes y momentos de procesos complejos definidos por (2) y (3). Los parámetros de entrada están relacionados con las ecuaciones citadas con anterioridad, y a través de los mismos, se le da a conocer el orden del cumulante a calcular (k) y la cantidad de señales complejas conjugadas que se tomará (l). Para esto, la función *cumpart2strn* invoca a la función *cumpart*, que a través del parámetro de entrada, que especifica el orden del cumulante, es capaz de generar las particiones de I_x en los conjuntos I_p para dicho orden. La Figura 2 muestra el diagrama de secuencia del proceso de generación de las particiones.

El proceso de generación de las particiones, a partir de (1), cuenta con dos particularidades: cómo determinar la cantidad de elementos a posicionar en cada subconjunto I_p y el modo de generar todas las formas posibles de hacerlo.

El valor de q, que denota la cantidad de conjuntos I_p de una partición de I_x en una iteración de (1), es recorrido, de forma general, desde $q = 1$ hasta el orden del cumulante que se desea calcular (k). Entiéndase por una iteración de (1) el hecho de determinar una partición de I_x en conjuntos I_p para un valor q determinado (operación de validación relativa al signo sumatorio) y obtener el producto de los respectivos momentos (operación relativa al signo de producto). En cada iteración, $I_x = \{1,2,\dots,k\}$ queda dividido en q subconjuntos I_p de forma tal que $\bigcup_{p=1}^{q} I_p = I_x$ y $\bigcap_{p=1}^{q} I_p = 0$. Esto sería matemáticamente equivalente a las formas de descomponer un número natural k en q sumandos, correspondiendo estas formas a la cantidad de

elementos a posicionar en cada partición I_p. Nótese como ejemplo, en Tabla 1, en el cálculo del cumulante de tercer orden: las llaves denotan el conjunto I_x siendo particionado en q subconjuntos denotados por paréntesis y la cantidad de elementos en cada uno de estos subconjuntos es equivalente a la partición del número natural $k = 3$ en q sumandos. En la implementación realizada, las funciones *formpartsum* y *newpartsum* son las encargadas de realizar dicha tarea. La primera función se implementó de forma recursiva, representando k el número entero positivo del cual se pretende hallar todas las formas de descomponerlo en sumandos entero positivo, y la segunda función se encarga del manejo de los datos entre la interfaz de las funciones *formpartsum* y *cumpart*, haciendo que los datos devueltos por la primera sean compatibles a la entrada para los datos requeridos por la última. La decisión de implementar la función *formpartsum* de forma recursiva fue basada en el hecho de que una solución alternativa de forma iterativa requiere del previo conocimiento de la cantidad de iteraciones a realizar o de la condición de ruptura. Esto es equivalente a saber la cantidad de formas exactas de descomponer un número en sumando. En la actualidad, dicho planteamiento constituye una interrogante en la Teoría de los Números, rama de la matemática que se encarga del estudio de las propiedades de los números. Sólo se han podido determinar relaciones asintóticas para aproximar dicho valor, cuando el número, del cual se pretende determinar las particiones, tiende a infinito.

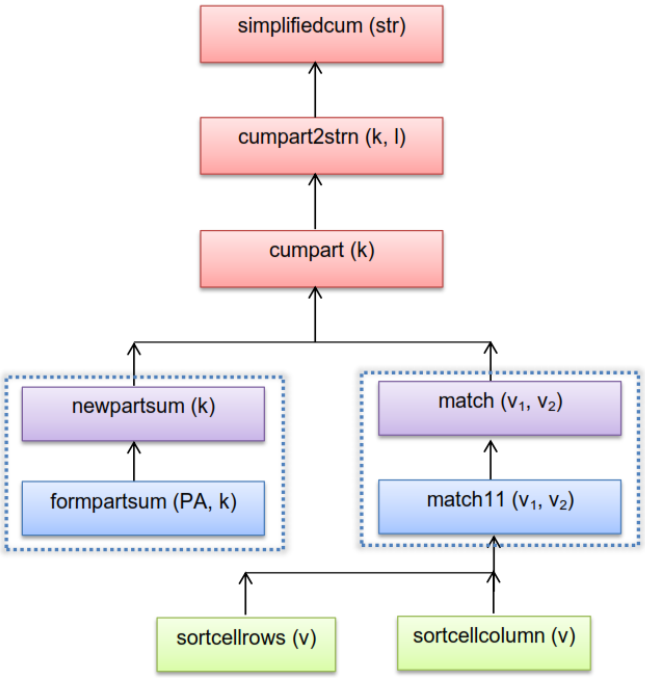

Figura 1. Jerarquía de módulos implementados para la obtención de expresión de los cumulantes

Sea $k = a_1 + a_2 + \cdots + a_q$ la partición de k en q sumandos en determinada iteración de (1). Las posibles formas de ubicar a_1 elementos en el primer subconjunto es enfocado a través de un problema de combinaciones y su valor es igual a $\binom{k}{a_1}$, siendo definido tal operación como:

$$\binom{n}{p} = \frac{n!}{(n-p)! \cdot p!}$$

donde el operador ! es el factorial del número indicado. Ejemplo, dado $I = \{1,2,3\}$ y la partición $3 = 3 + 0$. Como los conjuntos I_p no pueden ser nulos, entonces, $q = 1$ y se tendría un solo subconjunto I_p en la partición; por tanto, la cantidad de formas en que se puede tomar tres elementos de I en un subconjunto I_p, sin tener en cuenta su orden, viene dado por:

- cantidad de elementos: $n = 3$
- cantidad que se toma: $p = 3$

$$\binom{3}{3} = \frac{3!}{(3-3)! \cdot 3!} = \frac{3!}{3!} = 1$$

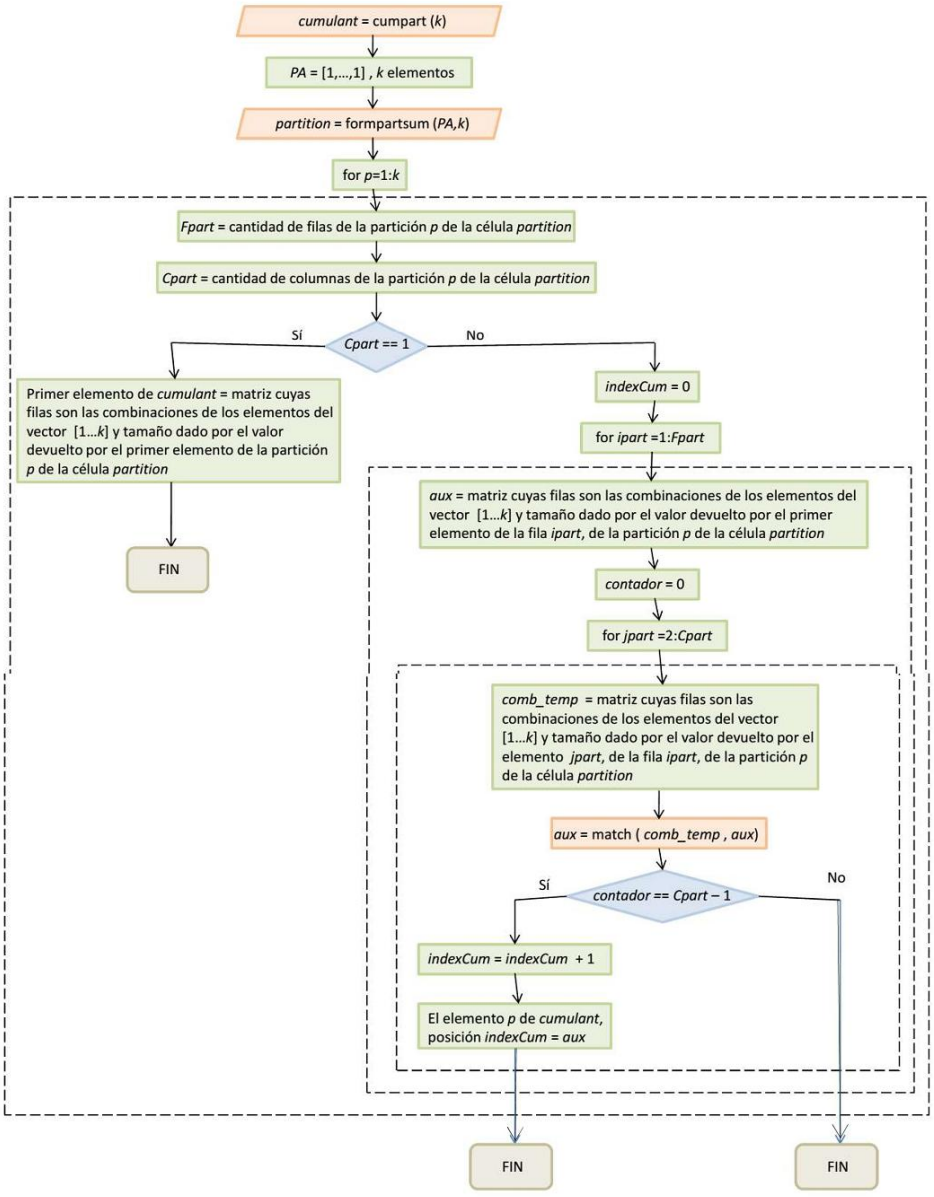

Figura 2. Algoritmo de función *cumpart*.

Esto significa que existe una forma: tomar los tres elementos del conjunto I y ubicarlos en I_p . Para el caso $3 = 1 + 2$, se está en presencia de dos subconjuntos I_p, por tanto $q = 2$. Como en el primer subconjunto se puede situar 1 elemento, esto se haría de $\binom{3}{1} = 3$ formas posibles: o bien se sitúa a 1, o bien 2, o bien 3. Para el segundo subconjunto se tendría que ubicar de $\binom{3}{2} = 3$ formas: o bien se sitúa (1,2), o bien (1,3), o bien (2,3). De esta manera, las particiones quedarían determinadas por: {(1), (2,3)}, {(2), (1,3)}, {(3), (1,2)}.

Análogamente se tendría que: las posibles formas de ubicar a_2 elementos en el segundo subconjunto viene dado por $\binom{k}{a_2}$, hasta llegar a la cantidad de formas posibles de ubicar a_q elementos en el q-ésimo subconjunto: $\binom{k}{a_n}$. No todas las formas que se generen a través de esta lógica satisfacen las condiciones de (1). Es de notar que sólo en las particiones en que se cumpla simultáneamente que $\bigcup_{p=1}^{q} I_p = I_x$ y $\bigcap_{p=1}^{q} I_p = 0$ y serán particiones a ser utilizadas. En aras de no perder ninguna forma de particionar los conjuntos, se procedió a generar todas las formas de ubicar a_i elementos en el i-ésimo subconjunto. De esto se encargó la función *cumpart*. Además, se discriminaron aquellas particiones a ser utilizadas a partir de la intersección nula de los subconjuntos construidos y de la unión unitaria de los mismos, utilizando para esto continuos llamados desde *cumpart* hacia el bloque *match* encargado de llevar a cabo este proceso de discriminación, particularizando con *match11* en el caso de que los vectores de entradas correspondan a una estructura de datos utilizadas en la programación: *cellarray*. Se generaliza con *match* para hacer compatible los datos entre *cumpart* y este bloque. Es de destacar también dos bloques auxiliares utilizados en la programación con el objetivo de ordenar las particiones generadas para hacer más viable el proceso de *cumpart2str*, estos fueron: *sortcellrows* y *sortcellcolumn*.

Con el objetivo de simplificar la expresión obtenida por *cumpart2str*, pues el orden impar de los momentos devuelve cero como resultado, la suma repetida de elementos iguales puede simplificarse a través de la multiplicación, y el producto de elementos iguales puede simplificarse a través de la potenciación, se implementó la función *simplifiedcum*, con la que se especifica la expresión del cumulante que se desea simplificar.

5. VALORACIONES SOBRE LA EFICIENCIA DE LOS ALGORITMOS

En la Figura 3 se muestra el comportamiento funcional de la cantidad de formas de descomponer un número en sumandos. Es de destacar la tasa de crecimiento que tiene dicha función, reportándose una cantidad de alrededor de 100 formas de descomposición en sumandos para el número 13 y más de 600 formas para el caso de 20.
El hecho de implementar de forma recursiva la manera de generar las particiones de un número en sumando, a través de la función desarrollada en MATLAB *formpartsum*, despertó el interés de valorar las implicaciones que traería consigo en el tiempo de ejecución. Esto se debió a la tendencia por parte de las soluciones recursivas de incurrir en tiempos prolongados de ejecución en correspondencia con las respectivas soluciones iterativas. Además, los algoritmos recursivos suelen requerir más espacio de *stack* que los algoritmos iterativos dando paso, en algunos casos, al *stackoverflow*.

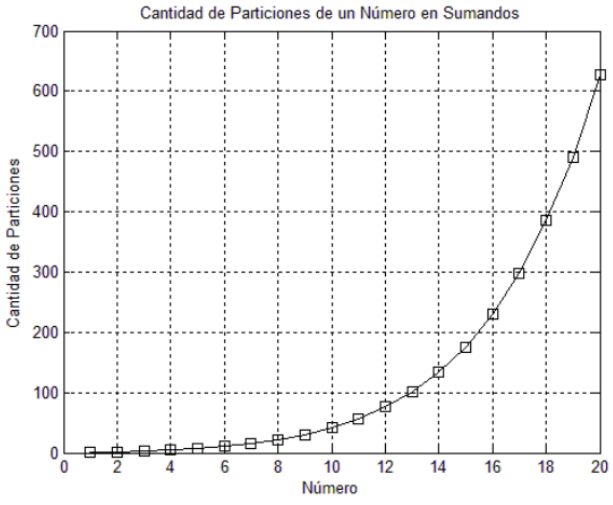

Figura 3. Cantidad de Particiones de un Número en Sumandos.

5.1. Algoritmo para generar las formas de particionar un número natural en sumandos

Para la resolución del objetivo trazado en este apartado se considerarán particiones ascendentes y ordenadas de mayor a menor.
Una partición ascendente, denotada por PA, es aquella en la cual se cumple que cada término es igual o mayor que el término a su izquierda. De esta forma, $(1+1+2)$ es PA de 3, mientras que $(1+2+1)$ no lo es.
Una partición es mayor que otra partición si su n-ésimo término es mayor que el n-ésimo término de la otra partición cuando es leído de izquierda a derecha, siendo iguales todos los términos anteriores al n-ésimo (si existen). De aquí que la PA de 4, $(1+3)$, sea mayor que $(1+1+2)$, puesto que el segundo término de la primera es mayor que el segundo término de la otra.

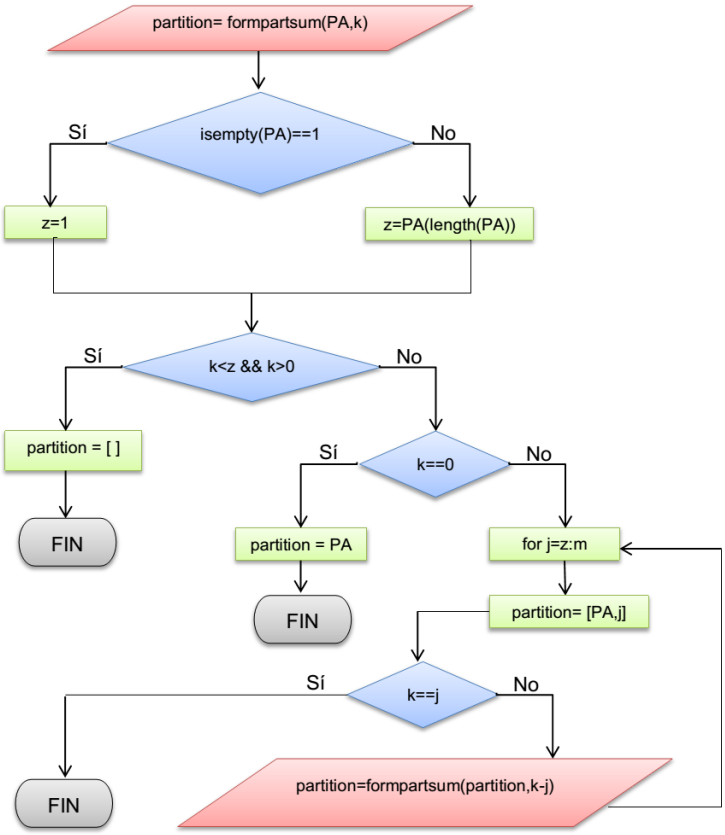

Figura 4. Diagrama de Algoritmo de función *formpartsum*

Sea f una función recursiva que devuelva las particiones ascendentes del número k, entonces f quedaría definido como:

$$f(PA,k)=\begin{cases} f(\{PA,z\},k-z)+f(\{PA,z+1\},k-z-1)+\cdots+f(\{PA,k\},0) & k\geq z,k>0 \\ 0 & k<z,k>0 \\ PA & k=0 \end{cases} \quad (4)$$

donde PA es alguna partición ascendente y z, el máximo término de PA o 1 si PA está vacía. Para poder generar las particiones de k, se pasa como primer parámetro a PA en forma de partición vacía.
La función f fue implementada en MATLAB bajo el nombre *formparsum*, recibe como argumentos a PA, vector cuyo primer parámetro a pasar será el elemento nulo o vacío, y el entero m, que indica el número del cual se determinará las particiones ascendentes. Al ser implementada de forma recursiva, se introducen las condiciones de rupturas dadas por las expresiones segunda y tercera de (4). La Figura 4 muestra el diagrama de secuencia del algoritmo propuesto.

Una vez implementada la función *formpartsum*, se pasó a realizar valoraciones referidas al tiempo de ejecución de la misma a partir de los valores recibidos en la entrada, obteniéndose los datos que se muestran en la figura 5. Se observa un crecimiento notable de los tiempos de ejecución a partir de las particiones en sumando del número 15. Estas pruebas se llevaron a cabo empleando un procesador Intel(R) Core(TM) i3 con 4GB de memoria RAM.

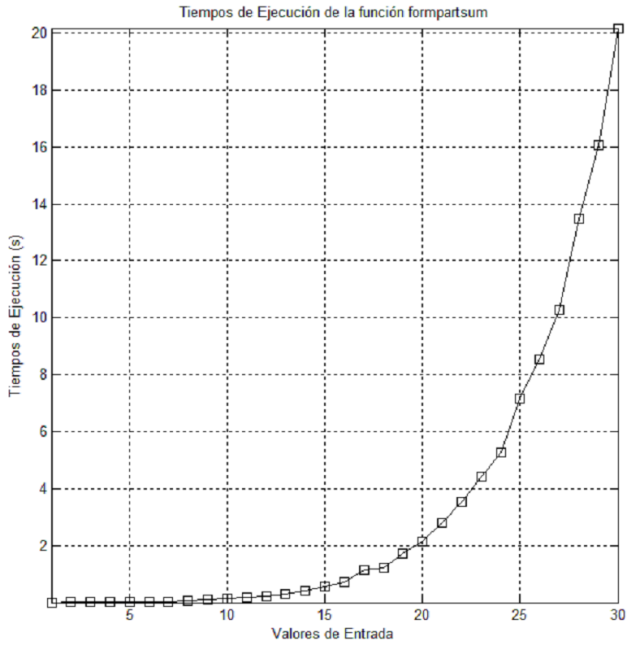

Figura 5. Tiempos de Ejecución de la función *formpartsum*.

5.2. Algoritmo para determinar las particiones en el cálculo de los cumulantes

El algoritmo que determina las particiones de I_x en conjuntos I_p, es la función *cumpart*. Para una valoración de la eficiencia del mismo, y a modo de ejemplo, fueron tomados los tiempos de ejecución hasta el valor de entrada 8 correspondiente al cumulante de octavo orden. Los resultados se muestran en la Figura 6.

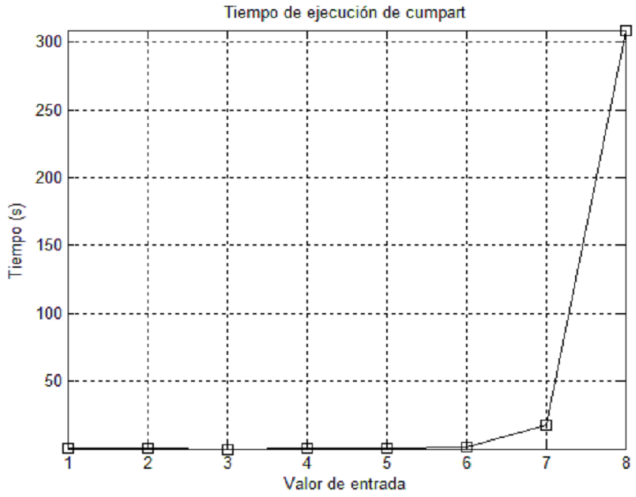

Figura 6. Tiempo de ejecución de *cumpart*.

Se observa cómo a partir de la determinación de las particiones para el valor de entrada 7 (orden 7) se experimenta un notable crecimiento en los valores del tiempo de ejecución del algoritmo, alcanzando para orden 8, un tiempo de ejecución por encima de los 300 s. Esto es debido a que a partir de 7 empieza a crecer de manera significativa la cantidad de formas de descomponer un número en sumando, y por ende, la cantidad de particiones a utilizar para la generación de las expresiones de los cumulantes (ver figura 6).
Se constató que los elevados tiempos de ejecución de *cumpart* radican en la generación de las particiones y la validación, a través de la función *match*, de cuáles de ellas constituye una partición I_x en conjuntos I_p.
Debido a esta premisa, se decidió aislar el bloque de generación de las expresiones de los cumulantes del bloque de clasificación, en aras de favorecer el proceso de clasificación con tiempos de ejecución aceptables. Además, los resultados a devolver por *cumpart2str* fueron guardados en ficheros *.mat para evitar incurrir en elevados costos computacionales por los módulos implementados que se encuentran por encima en la jerarquía expuesta con anterioridad.

6. RESULTADOS DE EJEMPLO. EXPRESIONES DE CUMULANTES DE OCTAVO ORDEN

Con el objetivo de obtener las particiones que permitan generar el cumulante de octavo orden, la implementación de *formpartsum* arrojó los resultados que se muestran en la figura 7. La misma enumera la cantidad de particiones encontradas (22 formas) y cita cada una de ellas.
La expresión del cumulante de octavo orden determinada fue utilizada en una investigación para clasificar señales de modulación digital, cuyos resultados se pueden encontrar en (Barrera y Hernández (2017)). En este caso, una vez obtenida las particiones, con el primer bloque de funciones se determinaron las expresiones para las variantes del cumulante de octavo orden que fueron de interés:

$$C_{80,x} = M_{80,x} - 28M_{20,x}M_{60,x} - 35M_{40,x}^2 + 420M_{20,x}^2M_{40,x} - 630M_{20,x}^4$$

$$C_{81,x} = M_{81,x} - 21M_{20,x}M_{61,x} - 7M_{21,x}M_{60,x} - 35M_{40,x}M_{41,x} + 210M_{20,x}^2M_{41,x} + 210M_{20,x}M_{21,x}M_{40,x} - 630M_{20,x}^3M_{21,x}$$

$$C_{84,x} = M_{84,x} - 6M_{20,x}M_{64,x} - 16M_{21,x}M_{63,x} - 6M_{22,x}M_{62,x} - M_{40,x}M_{44,x} - 16M_{41,x}M_{43,x} - 18M_{42,x}^2 + 6M_{20,x}^2M_{44,x} + 96M_{20,x}M_{21,x}M_{43,x} + 72M_{20,x}M_{22,x}M_{42,x} + 144M_{21,x}^2M_{42,x} + 96M_{21,x}M_{22,x}M_{41,x} + 6M_{22,x}^2M_{40,x} - 54M_{20,x}^2M_{22,x}^2 - 432M_{20,x}M_{21,x}^2M_{22,x} - 144M_{21,x}^4$$

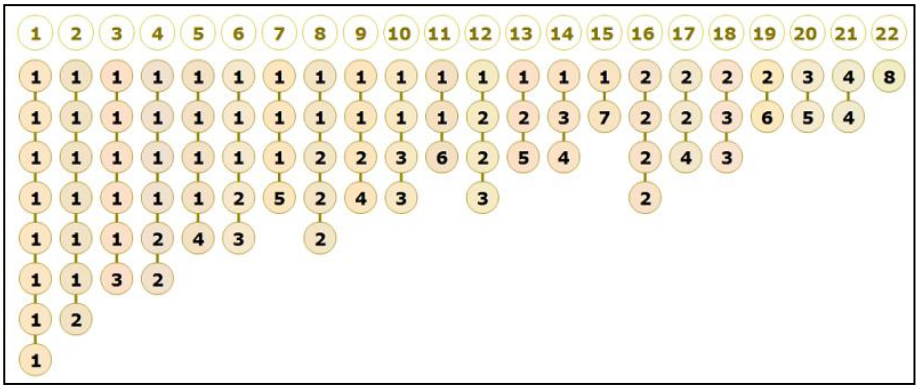

Figura 7. Particiones Ascendentes del número 8.

7. CONCLUSIONES

En este trabajo se presentó un algoritmo capaz de determinar las expresiones del cumulante de un orden dado. Se realizaron las valoraciones acerca de la eficiencia de los módulos programados propuestos, bajo las cuales se tomaron certeras decisiones en aras de reportar tiempos de ejecución aceptables en el proceso de clasificación.
Los resultados fueron aplicados de manera eficaz a la estimación del cumulante de octavo orden en una aplicación del área de las telecomunicaciones.

REVISED: MARCH, 2020.
REVISED: JULY, 2020.

REFERENCES

[1] AHMADI, N., and BERANGI R. (2009) : A Template Matching Approach to Classification of QAM Modulation using Genetic Algorithm, **Signal Processing. An International Journal,** 3, 95-109.
[2] AZARBAD, M., HAKIMI, S., and EBRAHIMZADEH, A. (2012) : Automatic Recognition of Digital Communication Signal, **International Journal of Energy, Information and Communications**, 3, 21-34.
[3] BARRERA J. L., and HERNÁNDEZ F. E. (2017) : Classification of MPSK Signals through Eighth-Order Statistical Signal Processing, IEEE Latin America Transactions, 15, 1601-1607.
[4] CABEZAS, N., CRUZ P. P., IGLESIAS M. E., and HERNANDEZ F. E. (2013) : El procesamiento estadístico de orden superior: herramienta útil para el análisis de señales, **Revista Ciencia y Tecnología**, 15, 30-37.
[5] LI P., ZHANG H., WANG X., XU N., and XU Y. (2012) : Modulation recognition of communication signals based on high order cumulnats and support vector machine. **The Journal of China Universities of Posts and Telecommunications**, 19, 61-65.
[6] MENDEL, J. M. (1991) : Tutorial on Higher-Order Statistics (Spectra) in Signal Processing and System Theory: Theoretical Results and Some Applications, **IEEE Proc.**, 79, 278-305.