Conteo de figuras geométricas en un mosaico de líneas.

Alejandro Casares M.

16 de mayo de 2020

1. PRESENTACIÓN.

En estos virulentos días de confinamiento obligado se ha popularizado en las redes un tipo de pasatiempo que propone hallar dentro de un mosaico plano, generalmente cuadrado o rectangular, formado usualmente por una malla de segmentos, el número de elementos de un cierto tipo geométrico. El problema, académicamente trivial, no por eso resulta sencillo, a causa de que las figuras buscadas se superponen parcialmente, se complementan unas a otras, y, aunque de una misma forma, pueden tener muchos tamaños y posiciones dentro del mosaico que les sirve de marco.

El objetivo de este ensayo es analizar el problema en forma global, proponer un algoritmo general de solución e implementarlo en un lenguaje de programación de alto nivel. El lector, así no llegue a implementarlo, puede adquirir una perspectiva del método, y aún aprovechar su nomenclatura y herramientas para sistematizar su resolución, aunque lo haga manualmente.

No se ha pretendido deducir y desarrollar matemáticamente fórmulas que entreguen la solución del problema y no puedan ser consideradas de "fuerza bruta", aunque en algunos casos sí sea posible encontrarlas.

Sin llegar a ser un desarrollo en *Inteligencia Artificial* o *Aprendizaje Automático*[1], pues no utiliza algoritmos de aprendizaje ni pretende entrenar al programa para que adquiera experiencia y mejore su capacidad de solución mientras mayor número de problemas diferentes resuelva, con un poco de imaginación y un toque de ironía, podrían – puestos a asignar etiquetas – adjudicársele las etiquetas (*tags*) de **reconocimiento de patrones** y **minería de datos** (como minúsculo ejemplo).

2. EJEMPLOS.

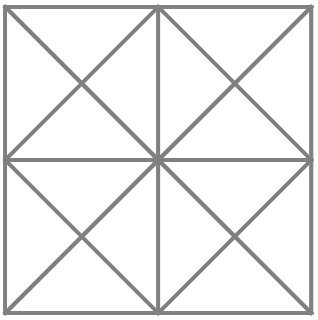
Para entrar rápidamente en materia, vienen a continuación algunos ejemplos que han sido resueltos con el programa, típicos de los que circulan en las redes.

En cada uno, pueden buscarse figuras planas adecuadas. Algunos esquemas se prestan a más de un tipo de búsqueda. Así, en el ejemplo 2 pueden buscarse triángulos o cuadrados, y en el 6 triángulos o rombos. Pero cada búsqueda debe ser independiente. Si la figura buscada no existe en el mosaico, el conteo, naturalmente, será nulo.

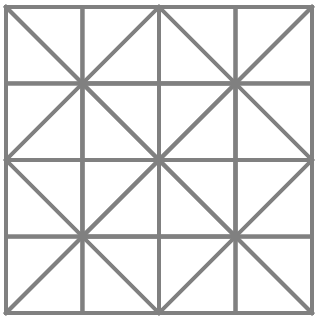
[1] Pobre, pero muy usada traducción de *Machine Learning* al castellano

Ejemplos propuestos

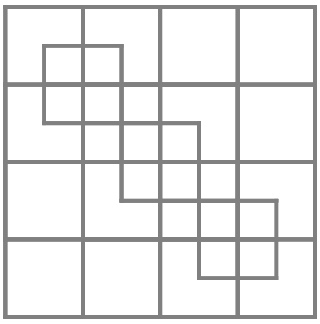
Ejemplo 1



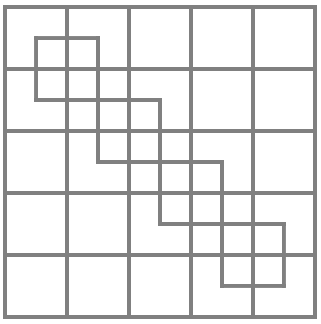
Ejemplo 2



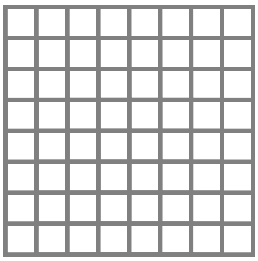
Ejemplo 3



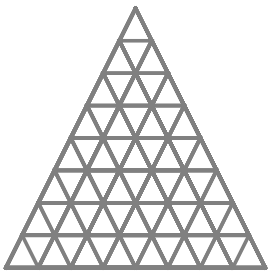
Ejemplo 4



Ejemplo 5



Ejemplo 6



3. DESCRIPCIÓN BÁSICA DEL ALGORITMO.

Utilizaremos dos elementos auxiliares cuya construcción, en cada caso, es la parte primera y esencial del algoritmo: los vértices numerados y la matriz de conexiones.

3.1. Vértices

En cada esquema, los vértices son los puntos de intersección de los segmentos que forman el mosaico. Para identificar a cada vértice, se le asignará un número natural en orden secuencial, comenzando por 1. Aunque se puede hacerlo siguiendo cualquier recorrido, conviene comenzar desde la esquina inferior izquierda del esquema, avanzar por la primera fila de vértices, columna por columna, hacia la derecha; luego se continúa con la segunda fila, y así sucesivamente hasta terminar. Si hay estructuras del gráfico, como en los ejemplos 3 y 4, que no comienzan en la primera columna, se puede numerarlas, continuando con la secuencia, una vez terminada la malla principal.

El programa utilizará esta numeración para identificar las figuras internas del esquema. Así por ejemplo, un triángulo quedará identificado por los números de sus tres vértices, y un cuadrado por los de sus cuatro vértices. Se ha convenido, además, en usar un sentido levógiro[2] para ordenar los vértices de una figura al referirse a ella.

En las soluciones obtenidas por el programa se puede encontrar la numeración de los vértices del esquema y las referencias a las figuras identificadas.

3.2. Matriz de conexiones

Para especificar las característica del mosaico de segmentos rectilíneos dentro del cual se buscarán las figuras, una vez identificados los puntos extremos de esos segmentos, que son los vértices, es necesario describir de qué manera ellos están alineados y conectados.

Se asignan un sistema de coordenadas cartesianas ortogonales en el plano, con origen en el vértice inferior izquierdo (que llevará el número 1 si se usó la sugerencia sobre el recorrido), y una unidad de medida, por conveniencia igual a la anchura de las divisiones de la malla. Entonces las dos coordenadas de cada vértice señalan unívocamente su posición en el plano.

La matriz de conexiones es una tabla cuadrada y simétrica cuyos elementos son ceros o unos: será 1 el elemento situado en la fila i, columna j, de esa matriz, cuando exista en el esquema un segmento rectilíneo que contenga a los vértices i y j; y será 0 en caso contrario. Nótese que no se requiere que i y j sean contiguos para estar conectados; sólo que estén sobre el mismo segmento.

3.3. Algoritmo de conteo (*contador general*)

La idea del algoritmo es sencilla pero potente, y funcionará en todos los casos en que se tengan los datos indicados. Sea iv el número de vértices del esquema y $\mathbf{C}$ la matriz de conexiones. Supongamos que se desea contar el número de figuras de NL lados. Al respecto, cabe indicar que se puede pedir figuras equiláteras o escalenas, pero en los ejercicios se ha definido por conveniencia que los triángulos buscados puedan ser cualesquiera, pero cuando NL es mayor a tres, los polígonos buscados son convexos y equiláteros (aunque pueden no ser equiangulares, como en el

[2] Es decir, en el sentido contrario al de las agujas del reloj

caso del rombo)[3].

El algoritmo genera, entonces, un arreglo V de todas las combinaciones de los números $\{1,2,\ldots,iv\}$ tomados de NL en NL. Luego recorre V, elemento por elemento, y para cada uno (es decir, para cada combinación posible de NL vértices), examina si la figura correspondiente es o no un segmento rectilíneo (vértices colineales); si no lo es, averigua si tiene un perímetro cerrado (es decir, si cada vértice tiene conexión con otros dos vértices perimetrales[4]) y si todos sus lados tienen igual longitud (por la hipótesis introducida), en cuyo caso incrementa el contador de figuras halladas.

Si V ha sido bien calculado, entonces no tiene repeticiones, es completo, y, además, sigue un orden conveniente, al que ya nos referiremos. De modo que no hay razón para que el conteo no sea correcto. Nótese que, teóricamente hablando, nada impide que el algoritmo se aplique manualmente, pero en la práctica, la cardinalidad de V – que es el coeficiente binomial C_{iv}^{NL} – es tan grande para revisarlo con lápiz y papel[5], que recorrer el arreglo manual se vuelve utópico empeño; y no se diga si le añadimos la ubicación y reconocimiento de cada figura dentro del mosaico. Es ahí donde la velocidad y exactitud de la computadora hacen la diferencia con respecto a la limitada capacidad humana para realizar sin equivocarse tareas repetitivas y rutinarias con gran cantidad de elementos[6].

4. Funciones del programa y módulos que las ejecutan

El programa bautizado *Counting figures* dispone a la fecha de los siguientes módulos:

1. *contador_general*
2. *crea_conteos*
3. *fcrea_conexion*
4. *work_link*
5. *fpanel_triangular*

Veamos de qué manera el programa *Counting_figures* está organizado para llevar a cabo el algoritmo descrito. A continuación se revisan someramente los pasos y se indica el módulo que implementa cada uno, con una breve descripción de las características que se han considerado esenciales.

Conviene indicar que la implementación aquí descrita se ha orientado hacia un usuario ilustrado, es decir, familiarizado con el ambiente de programación MATLAB, quien dispondrá de los módulos fuente, y que estará en capacidad – si lo desea – de crear e ingresar nuevos ejemplos trabajando en ese ambiente de programación, y de mantener y administrar el sistema.

[3]Limitar la búsqueda a triángulos equiláteros resulta muy restrictivo en los ejemplos 1, 2 y 6, mientras ampliarla a cuadriláteros cualesquiera en el ejemplo 6 produce demasiados resultados y la solución pierde precisión

[4]En la implementación final se prefirió calcular el área: si es mayor que cero, se trata de una figura cerrada.

[5]En el ejemplo 5, con 81 vértices, V tiene 1663740 elementos, mientras en el 6, de 45 vértices, hay 14190 cuando se buscan triángulos y 148995 si son rombos.

[6]Sin embargo, el humano, utilizando criterios simples, puede saltarse la revisión de muchos elementos, evitando, por ejemplo, en un solo paso todas las figuras que comiencen con al menos tres puntos colineales o no conectados.

4.1. Preparación de la información para el algoritmo:

Ingreso de datos del esquema:

- Vértices: Su numeración se hace desde la esquina inferior izquierda. El módulo que acoge esta parte es *crea_conteos*, mediante líneas de programa que recogen las características del mosaico en cuestión, asignando en orden secuencial coordenadas x y y a cada punto.
- Matriz de conexiones: También a cargo del módulo *crea_conteos*, que puede tomar la matriz **C** de líneas de ceros y unos introducidas por el programador (como en los ejemplos 1 y 2), o llamar a una función ad-hoc creada por él, como *fcrea_conexion* en los ejemplos 3, 4 y 5, o *fpanel_triangular* en el ejemplo 6. Estas funciones no leen ceros y unos, sino que contienen estrategias para generarlos de acuerdo con la geometria del problema. Adicionalmente, para examinar y editar, si se desea, la matriz de conexiones, se ha implementado la función *work_link*. El resultado de cada proceso llevado a cabo en *crea_conteos* se guarda en un archivo determinado, al que el programa principal tiene acceso.
- Selección de la figura que se busca: Se la especifica en el guion principal *contador_general*, para cada opción contemplada.

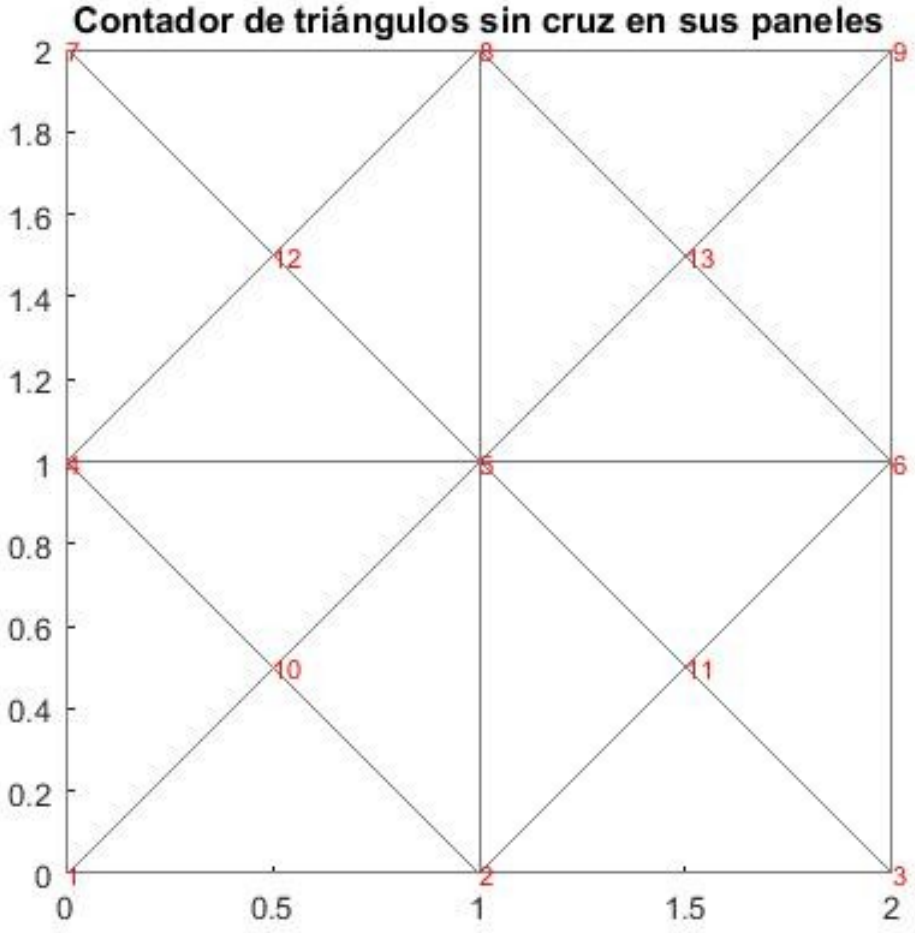
4.2. Algoritmo en sí: Proceso de conteo.

Lo lleva a cabo *contador_general* aplicando las ideas básicas ya expuestas. Este módulo admite parámetros para el grado de detalle en que se desean los resultados (únicamente los vértices que forman figuras cerradas, o también los colineales y los que no forman figuras continuas); y para los bloques de salida requeridos: impresión de la matriz de conexiones, gráfico del esquema.

4.3. Resultados obtenidos.

El programa presenta en pantalla en primer lugar el menú de selección del ejemplo que se desea resolver.
Cuando se ha solicitado todos los bloques de salida, viene a continuación el gráfico del esquema, con los vértices ubicados y numerados con caracteres pequeños de color rojo. Este gráfico permanece a la vista durante el resto del proceso, de manera que el usuario puede ir ubicando en él las figuras conforme van apareciendo como resultados, lo cual puede darle una intuición visual acerca del orden de avance en la selección de grupos de vértices que hace el algoritmo.
A continuación aparece la matriz de conexión, de orden igual al número de vértices. Dado que éste puede ser alto (en el caso del tablero de ajedrez, por ejemplo, es una matriz de 81 x 81, que no resulta cómoda de examinar y además puede exceder el espacio de la pantalla), en esta presentación se ha escogido el ejemplo 1, que aparece en la página siguiente (Contador de triángulos sin cruz en sus paneles), con sólo 13 vértices, y sin embargo suficiente para la presente explicación.
Luego vienen las figuras encontradas, cada una con su respectivo número secuencial y la lista de sus vértices perimetrales, que es el resultado final y fundamental. En el ejemplo 1 se puede ver que son 44 los triángulos encontrados[7], y se puede localizar cada uno de ello en el gráfico, que en la impresión ha sido colocado en primer lugar.

[7]He podido ver, en varios lugares de la red y en concursos televisivos, que erróneamente se afirma que son 40 triángulos.



Creación de esquemas de conteo
==========

1.-Contador de triángulos con cruz añadida
2.-Contador de triángulos sin cruz añadida
3.-Contador de cuadrados 4 x 4
4.-Contador de cuadrados 5 x 5
5.-Cuadrados en esquema con cruz añadida
6.-Cuadrados en esquema sin cruz añadida
7.-Cuadrados en tablero de ajedrez
8.-Triángulo de triángulos 8 x 8
9.-Salir del programa

Escoja su opción:2

Contador de triángulos sin cruz en sus paneles

MATRIZ DE CONEXIONES RECTILINEAS:

```
                    1
    1 2 3 4 5 6 7 8 9 0 1 2 3
-------------------------------
 1.-0 1 1 1 1 0 1 0 1 1 0 0 1
 2.-1 0 1 1 1 1 0 1 0 1 1 0 0
 3.-1 1 0 0 1 1 1 0 1 0 1 1 0
 4.-1 1 0 0 1 1 1 1 0 1 0 1 0
 5.-1 1 1 1 0 1 1 1 1 1 1 1 1
 6.-0 1 1 1 1 0 0 1 1 0 1 0 1
 7.-1 0 1 1 1 0 0 1 1 0 1 1 0
 8.-0 1 0 1 1 1 1 0 1 0 0 1 1
 9.-1 0 1 0 1 1 1 1 0 1 0 0 1
10.-1 1 0 1 1 0 0 0 1 0 0 0 1
11.-0 1 1 0 1 1 1 0 0 0 0 1 0
12.-0 0 1 1 1 0 1 1 0 0 1 0 0
13.-1 0 0 0 1 1 0 1 1 1 0 0 0
```

Número de vértices: 13

1 .- Triángulo 1 2 4
2 .- Triángulo 1 2 5
3 .- Triángulo 1 2 10
4 .- Triángulo 1 3 5
5 .- Triángulo 1 3 7
6 .- Triángulo 1 3 9
7 .- Triángulo 1 5 4
8 .- Triángulo 1 10 4
9 .- Triángulo 1 5 7
10 .- Triángulo 1 9 7
11 .- Triángulo 2 3 5
12 .- Triángulo 2 3 6
13 .- Triángulo 2 3 11
14 .- Triángulo 2 5 4
15 .- Triángulo 2 6 4
16 .- Triángulo 2 8 4
17 .- Triángulo 2 6 5
18 .- Triángulo 2 5 10
19 .- Triángulo 2 11 5
20 .- Triángulo 2 6 8
21 .- Triángulo 3 6 5
22 .- Triángulo 3 9 5
23 .- Triángulo 3 6 11
24 .- Triángulo 3 9 7
25 .- Triángulo 4 5 7
26 .- Triángulo 4 5 8
27 .- Triángulo 10 5 4
28 .- Triángulo 4 5 12
29 .- Triángulo 4 6 8
30 .- Triángulo 4 8 7
31 .- Triángulo 4 12 7
32 .- Triángulo 5 6 8
33 .- Triángulo 5 6 9
34 .- Triángulo 11 6 5
35 .- Triángulo 5 6 13
36 .- Triángulo 5 8 7
37 .- Triángulo 5 9 7
38 .- Triángulo 5 9 8
39 .- Triángulo 5 8 12
40 .- Triángulo 5 13 8
41 .- Triángulo 6 9 8
42 .- Triángulo 6 9 13
43 .- Triángulo 12 8 7
44 .- Triángulo 13 9 8

Encontró 44 Triángulos

Como ya se ha mencionado, a veces es posible buscar figuras de varias formas en un mismo esquema, y procesar cada una independientemente. Si en el ejemplo que acabamos de ver se cambia la figura buscada de triángulos a cuadrados, con la misma malla, vértices y matriz de conexiones, el resultado es el siguiente:

```
 1 .- Cuadrado  1  2  5  4
 2 .- Cuadrado  1  3  9  7
 3 .- Cuadrado  2  3  6  5
 4 .- Cuadrado  2  6  8  4
 5 .- Cuadrado  2 11  5 10
 6 .- Cuadrado  4  5  8  7
 7 .- Cuadrado 10  5 12  4
 8 .- Cuadrado  5  6  9  8
 9 .- Cuadrado 11  6 13  5
10 .- Cuadrado  5 13  8 12

Encontró 10 Cuadrados
```

5. Algunas notas sobre la programación, mantenimiento y administración del sistema.

Como ya se dijo, el programa ha sido elaborado en **MATLAB**, de modo que estas anotaciones tendrán básicamente ese enfoque. Sin embargo, pueden ser útiles también para el caso de que se desee traducirlo a otro paquete funcionalmente equivalente, como **Python** o **R**.

- El menú del programa *contador_general* determina los esquemas que el sistema puede resolver. Si se intenta añadir un nuevo caso, la primera acción que debe emprenderse es aumentarlo en este menú, lo cual implica incrementar la variable *nopc* y añadir con *fprintf* una nueva opción a la lista del menú, actualizando la numeración secuencial.

  Luego, en la lista de casos que viene a continuación en el mismo programa, hay que incluir el número del nuevo caso, y las acciones y parámetros que le van a corresponder: el nombre del archivo de donde cargará los datos en la variable P[8], el número NL de lados, el indicador *reg* de equilateralidad y el nombre *fig* de la figura buscada, junto con la identificación amplia del nuevo esquema.
  Ejemplo:

  ```
  case 10
     load connectT10 P; NL = 4; reg = 1; fig = 'Rombos';
     name = 'Nuevo esquema de rombos 10 x 10';
  ```

- Como la matriz de conexiones **C** puede ser muy grande, y es además simétrica, se prefirió archivar y recuperar sólo lo indispensable, es decir, su triángulo superior [9], que se guarda, fila por fila, en la variable *link*, prácticamente de la mitad de su tamaño. Durante el proceso se arma **C** completa, pero en el archivo en disco sólo se conserva *link*. Conocida esta

[8] P es un arreglo de celdas que contiene el triángulo superior de la matriz de conexiones, por filas, y las coordenadas x e y de los vértices

[9] Por comodidad se incluye también la diagonal principal, llena de ceros.

característica, en lo que viene nos referiremos siempre a **C**, sin preocuparnos de su forma de almacenamiento en memoria.

La creación de la matriz de conexiones puede ser llevada a cabo de dos formas:

1. Manualmente, cuando es difícil definir la malla de segmentos por una característica general que pueda programarse, como en los casos 1 y 2. El proceso manual requiere paciencia y precisión, y debe llevarse a cabo con cuidado para que represente fielmente el modelo.
2. Por programa, en los casos en que se pueda identificar una característica geométrica común para todos los pares de puntos conectados. Así, en el ejemplo 7, del tablero de ajedrez, si dos puntos tienen la misma abscisa o la misma ordenada, deben estar conectados, lo cual se programa fácilmente en la función *fcrea_conexion*, llamada desde *crea_conteos*:

```
% Para cada punto P se buscan los puntos con igual abscisa que P y se los guarda en
% X. Se hace lo mismo para cada ordenada, y sus resultados se guardan en Y.
% Finalmente se pone la unión de X y Y como fila correspondiente de C.
C = zeros(iv);
for k = 1:iv
   X = x == x(k);
   Y = y == y(k);
   C(k,:) = or(X,Y);
end
```

3. Eventualmente, podría ensayarse un procedimiento mixto, basado en la malla básica creada por programa, con algunos puntos añadidos, que se conectan en una forma especial, como en los ejemplos 2 y 3. Para este caso, y también para el puramente manual, se ha habilitado la función *work_link*, la cual permite editar uno por uno elementos (i, j) de la matriz, a la vez que facilitar este trabajo, al mostrar, si se desea, el gráfico del esquema y la matriz **C** completa, en su estado actual, información que toma del archivo correspondiente. Tal como sucede con *contador_general*, en esta función habrá que añadir, en una estructura *switch / case* los nuevos casos en los que se quiera poder editar **C**. La llamada de la función es:

```
[link,x,y] = work_link(caso,i,j,imp);
```

   El significado y uso de sus argumentos de entrada y salida pueden consultarse en el archivo *work_link.m*.

- Creación del arreglo de combinaciones de vértices: la primera de las dos técnicas fundamentales del algoritmo de conteo – según ya se explicó en 3.3 – es construir el arreglo V, con todas las combinaciones de NL vértices que se puedan formar con los iv vértices del esquema. Aunque se podría hacerlo utilizando lazos anidados, es mucho más cómodo y eficiente utilizar la función que **MATLAB** proporciona para este fin, y que tiene sus análogas en otros lenguajes de programación equiparables. Se trata de la función *combnk(V,k)*, que entrega todas las combinaciones de los n elementos en V tomados de k en k.

En *contador_general* encontramos las dos líneas:

```
% Combinaciones de iv vértices, tomados de NL en NL:
V = combnk(1:iv,NL); nV = size(V,1);
```

La segunda línea crea el arreglo V deseado, a partir de los números de vértices {1,2,...,iv}. Su tamaño, almacenado en nV, corresponde al valor del coeficiente binomial

$$C_{iv}^{NL} = \frac{iv!}{NL!(iv-NL)!},$$

como se puede comprobar en cualquier ejemplo.

Cada elemento del arreglo V está formado por NL coeficientes. El primero será $\{1, 2, \ldots, NL\}$, y el último $\{iv-(NL-1), iv-(NL-2), \ldots, iv\}$.

- Determinación de si un elemento de V corresponde a una figura buscada. Este proceso es ya más largo para explicarlo aquí detalladamente en palabras, pero hay algunos puntos claves que vale la pena examinar:
  - Estamos recorriendo los grupos G de NL vértices, para saber cuáles forman la figura buscada y cuáles no. Primero dejemos fuera los puntos colineales. Usaremos la función

    ```
    pointsAreCollinear = @(xy) rank(bsxfun(@minus,xy(2:end,:),xy(1,:))) == 1;
    ```

    Esta función en línea, definida al comienzo del programa, busca el rango de una matriz *NL-1* × *2* formada con las abscisas y ordenadas de *NL-1* de los NL vértices escogidos, de cada una de cuyas filas se restan las coordenadas del vértice restante[10]. Si el rango es 1, la variable calculada es 1 y los puntos son colineales. Se la utiliza en nuestro proceso en la llamada:

    ```
    colin = pointsAreCollinear([x(G(1:NL));y(G(1:NL))]');
    ```

  - Si los puntos no están en línea recta, entonces se averigua si forman un polígono cerrado de NL vértices. La forma escogida de hacerlo, por ser la más general, es calcular la extensión del área encerrada, mediante la función embebida *polyarea(X,Y)*, a la cual se entregan las coordenadas de los vértices, en un recorrido perimetral, que puede ser levógiro o dextrógiro[11]. En general, la numeración automática de los vértices en cada grupo, que viene de la función *combnk*, no se preocupa de que el recorrido se haga en uno de estos dos sentidos, de modo que, para cumplir con ese requisito, dado un grupo G de vértices no colineales, antes de calcular el área hay que ordenarlos.
    Para hacerlo, a partir del centroide CG – calculado como el punto cada una de cuyas coordenadas es el promedio de las respectivas coordenadas de los vértices – se hallan los ángulos de los vectores que parten de CG y van a cada uno de los vértices. Usando la función $atan2(Y, X)$ (arco tangente entre 0 y 2π) y ordenando en forma creciente los ángulos obtenidos, se obtiene un orden levógiro, que se traslada mediante un índice de clasificación a los vértices en sí.

  - El último requisito para aceptar que G contiene los vértices de una figura buscada es comprobar que, si *reg = 1*, tenga todos sus lados congruentes. En este caso se requiere que se calculen las distancias entre los vértices – que son las longitudes de los lados – para lo cual se ha definido la función en línea

    ```
    distanceConnected = @(x,y,pares) sqrt(diff(x(pares(:)))^2 + diff(y(pares(:)))^2);
    ```

[10] https://la.mathworks.com/matlabcentral/answers/438506-can-we-check-points-are-in-straight-line?s_tid=srchtitle

[11] Si los puntos no están en uno de estos órdenes de recorrido, o no encierran un polígono de NL vértices, la función devuelve un área nula

y se la llama oportunamente con:

```
R = [];R(1:NL,1) = G';R(1:NL-1,2)=G(2:NL)';R(NL,2)=G(1);
for id = 1:NL
    ds(id) = distanceConnected(x,y,R(id,:));
end
```

- Como ya se dijo, en la concepción del sistema implementado toca al programador encargarse también de su mantenimiento y administración. Si algún requerimiento lo ameritase, se podría invertir tiempo y esfuerzo para mejorar y modernizar la interfaz del sistema con el usuario, estableciendo entre ambos una forma gráfica de comunicación conocida como *GUI* (graphic user interface), la cual podría facilitar, por ejemplo, la añadidura de nuevos esquemas sin tener que escribir líneas de código. El mantenimiento de la *GUI* sería más fácil y liberaría al usuario de la necesidad de conocer MATLAB, lo cual lo haría más atractivo para la mayoría de los usuarios.

6. Soluciones de los ejemplos

Los lectores habrán sin duda resuelto – o al menos intentado hacerlo – algunos de los ejercicios planteados en la sección 2, y sin duda asimismo, tendrán curiosidad por verificar si sus resultados coinciden con los obtenidos por el programa. Para no dejarlos en la incertidumbre, en la tabla siguiente se resumen los del programa, utilizando, para abreviar, la nomenclatura de las variables expuesta en las secciones anteriores. Recuérdese además que se especificó que los triángulos buscados no tengan que ser equiláteros, pero los polígonos de más de tres lados sí lo sean.

Esquema	**iv**	**fig**	**NL**	**nV**	**N°**	**fig**	**NL**	**nV**	**N°**
Ejemplo 1	13	Triángulo	3	286	**44**	Cuadrado	4	715	**10**
Ejemplo 2	25	Triángulo	3	2300	**96**	Cuadrado	4	12650	**35**
Ejemplo 3	47	Triángulo	3	1081	**0**	Cuadrado	4	178365	**51**
Ejemplo 4	65	Triángulo	3	43680	**0**	Cuadrado	4	677040	**85**
Ejemplo 5	81	Triángulo	3	85320	**0**	Cuadrado	4	1663740	**204**
Ejemplo 6	45	Triángulo	3	14190	**170**	Rombo	4	148995	**50**

De estas soluciones, al menos una es comprobable matemáticamente: no es difícil apreciar que los cuadrados que se pueden encontrar en un tablero de ajedrez, debido a su simetría respecto a la diagonal que pasa por el origen son: uno de 8 casillas de lado, que es el propio tablero; 4 con 7 casillas de lado; 9 con 6, y así sucesivamente, hasta llegar a los 64 escaques individuales. El número total de cuadrados puede, entonces, escribirse como:

$$N = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2 + 7^2 + 8^2 = \sum_{k=1}^{k=8} k^2 = \frac{8 \times (8+1) \times (2*8+1)}{6} = 204.$$

Usando la fórmula de la suma de cuadrados consecutivos $N = \sum_{k=1}^{k=n} k^2 = \frac{n \times (n+1) \times (2n+1)}{6}$, que se puede demostrar por inducción.

7. Repositorio de los programa del sistema

Los programadores interesados pueden obtener la última versión de los programas fuente **MATLAB** de esta aplicación en el repositorio **Git Hub**, ampliamente usado para colaboración

en el mundo del software, ante todo para el control de versiones.

La URL del repositorio del sistema es https://github.com/acasares/Counting_Figures. Para descargar los archivos recomiendo a quienes no estén familiarizados con **Git Hub** usar, por su sencillez, el botón verde del extremo derecho `"CLONE OR DOWNLOAD"`, escoger `"Clone with HTTPS"` y finalmente `"Download ZIP"`, que empaquetará todos los archivos y los guardará en el directorio de descargas del usuario, en su computador local.

Se agradecerá cualquier pregunta o sugerencia, y todas serán debidamente consideradas y respondidas.