Uso de SageMath en la Investigación y la Docencia Práctica

Francisco M. García Olmedo*
Pedro González Rodelas**

Resumen

Tras el auge en característica y prestación protagonizado por el sistema computacional algebraico llamado SageMath[1] y materializado en bibliotecas cada vez más elaboradas o por invocación de otro software, dicho sistema ha resultado de uso muy atractivo en laboratorio. Su utilidad, tanto para la investigación como para la docencia práctica, motiva en esta ponencia una exposición introductoria a su instalación y rasgos principales; nos centraremos en el ámbito cada vez más visitado de la Matemática Discreta, sobre todo en los estudios teóricos y técnicos relacionados con el desarrollo del software.

Palabras clave: SageMath, Python, cálculo científico y técnico.

Índice

*Dpto. de Álgebra, UGR.
**Dpto. de Matemática Aplicada, UGR.

[1] Acrónimo en parte de "System for Algebra and Geometry Experimentation".

1. Introducción

sagemath es un sistema algebraico computacional (en inglés CAS) escrito en Cython, inicialmente llamada SageX, una bifurcación de Pyrex que extiende a Python diseñada para proporcionar un rendimiento tipo C con un código escrito principalmente en Python. Reúne y unifica bajo un solo entorno, lenguaje y jerarquía de objetos toda una colección de software matemático y trata de rellenar los huecos de funcionalidad dejados por unos y otros.

La primera versión de sagemath se publicó el 24 de febrero de 2005, con el objetivo inicial de recrear un pequeño subconjunto del sistema algebraico computacional Magma, y reducir así la dependencia del software matemático propietario y cerrado. El líder del proyecto, *William A. Stein*, es un matemático en la *Universidad de Washington*, y emplea estudiantes becados para el desarrollo del mismo.

Proporciona una interfaz a software libre como GAP, Pari, Maxima, SINGULAR (todos distribuidos con sagemath). También proporciona una interfaz a software no libre: Magma, Maple, Mathematica (no distribuidos con sagemath).

Entre sus muchas características, Sagemath incluye:

- Una interfaz gráfica (notebook) para la revisión y reutilización de entradas y salidas anteriores, incluyendo gráficas y notas de texto disponibles en la mayoría de los navegadores web incluyendo Firefox, Opera, Konqueror, Chrome, Chromium y Safari.
- Una línea de comandos basada en texto usando iPython.
- El lenguaje de programación Python, actualmente en su versión 2.7, que soporta expresiones en programación orientada a objetos y "funcional".
- Procesamiento paralelo usando tanto procesadores de núcleo múltiple como multiprocesadores simétricos.
- Cálculo simbólico usando Maxima y SymPy.
- Álgebra lineal numérica usando GSL, SciPy y NumPy.
- Control interactivo de los cálculos.

- Bibliotecas de funciones elementales y especiales.
- Gráficas en 2D y 3D tanto de funciones como de datos.
- Herramientas de manipulación de datos y matrices.
- Bibliotecas de estadística multivariable.
- Una caja de herramientas para añadir interfaces de usuario a cálculos y aplicaciones.
- Herramientas para procesamiento de imágenes usando `pylab` así como `Python`.
- Herramientas para visualizar y analizar gráficas.
- Bibliotecas para funciones de teoría de números.
- Filtros para importar y exportar datos, imágenes, vídeo, sonido, `CAD` y `GIS`.
- Soporte para números complejos, aritmética de precisión arbitraria, y computación simbólica de funciones donde esto sea apropiado.
- Embeber `Sagemath` en documentos LaTeX.
- Interfaces a otro software como `Mathematica`, `Magma` y `Maple`, que le permite a los usuarios combinar software y comparar resultados y desempeño.

Álgebra	GAP, Maxima, Singular
Álgebra lineal	Linbox, IML
Álgebra lineal numérica	GSL, SciPy, NumPy
Aritmética de precisión arbitraria	GMP, MPFR, MPFI, NTL
Cálculo	Maxima, Sympy
Combinatoria	Symmetrica, MuPAD-Combinat*
Geometría algebraica	Singular, Macaulay2
Geometría aritmética	PARI, NTL, mwrank, ecm
Gráficos	Matplotlib, Tachion3d
Teoría de grafos	NetworkX
Teoría de grupos	GAP

Figura 1: Paquetes matemáticos incluidos en `sagemath`.

William Stein observó varios factores importantes al diseñar `sagemath`.

- Crear una alternativa viable a `Magma`, `Maple`, `Mathematica`, y `MATLAB`, llevaría cientos o miles de años-hombre si se empezara desde cero.

- La existencia de una amplia gama de software matemático de código abierto bien probado, pero escrito en diferentes lenguajes (siendo C, C++, Fortran y Python los más comunes).

Así que en lugar de empezar desde el inicio, sagemath integraría todo el software de código abierto sobre matemática ya existente a una interfaz común. Un usuario necesitará saber únicamente Python (que es un lenguaje bien conocido, y usado en miles de aplicaciones diferentes).

Donde no hubiera una opción de software libre disponible para algún problema, entonces sería escrito en sagemath. Pero sagemath no reinventa la rueda. La misma filosofía de diseño se usa en otros programas matemáticos (como Mathematica), pero sagemath puede utilizar un espectro más amplio de software que sus contrapartes no libres, ya que las licencias propietarias imponen serias restricciones a la reutilización del software.

El desarrollo de sagemath lo llevan a cabo tanto estudiantes como profesionales. Es apoyado tanto por trabajo voluntario como por donaciones. En 2007 sagemath ganó el primer premio en la categoría de software científico en *Les Trophées du Libre*, una competición internacional de software libre.

sagemath es software libre, distribuido bajo los términos de la GNU General Public License versión 2 ó posterior. sagemath está disponible de varias maneras:

- Se puede usar la versión en línea en cocalc.com.
- Es posible descargar binarios para GNU/Linux, OS X y Solaris (tanto x86 como SPARC). Los binarios de Solaris se consideran experimentales.
- Existen instaladores para Linux, Mac OS y recientemente también para Windows.
- El código fuente puede descargarse de la página de descargas. Aunque no se recomienda para el usuario final, las versiones en desarrollo también están disponibles.

En sagemath están incluidos los paquetes matemáticos según se refiere en la tabla de la Figura 1.

2. Aspectos Técnicos

Un aspecto central de Sagemath es soportar cálculos con objetos de diferentes sistemas algebraicos de computación "bajo un mismo techo", usando una interfax común y un lenguaje de programación limpio. Los sistemas aludidos son: GP/PARI, GAP, Singular

y Maxima. Por otra parte, Sagemath mantiene una relación muy estrecha con LaTeX para la presentación de resultados.

Los archivo de Sagemath tienen extensión .sage. Como ejemplo sea el siguiente. Supongamos que tenemos un fichero example.sage con el siguiente contenido:

```
print("Hello World")
print(2^3)
```

Puede ser leído en la consola de Sagemath usando la orden load:

```
sage: load("example.sage")
Hello World
8
```

También es posible adjuntar un fichero de Sagemath a una sesión en ejecución mediante la orden attach:

```
sage: attach("example.sage")
Hello World
8
```

Un cambio en example.sage exigirá la recarga automática en la sesión de Sagemath, frente a load que carga el fichero una vez tan solo. Cuando es cargado example.sage es convertido en Python y ejecutado por el intérprete de Python. Esta conversión es minimal; principalmente envuelve literales enteros en Integer(), literales en coma flotante en RealNumber(), reemplaza ^ por **, reemplaza R.2 por R.gen(2), etc. La versión convertida de example.sage es ubicada en el mismo directorio y es llamada example.sage.py y tiene el siguiente contenido:

```
print("Hello World")
print(Integer(2)**Integer(3))
```

Este preanálisis está implementado en sage/misc/interpreter.py.

Sin embargo también se puede crear código compilado. La velocidad es crucial en los cálculos matemáticos. Aunque Python es un lenguaje conveniente de muy alto nivel, ciertos cálculos pueden ser varios órdenes de magnitud más rápidos que en Python si se implementan usando tipos estáticos en un lenguaje compilado. Algunos aspectos de Sagemath habrían sido demasiado lentos si hubieran sido escritos completamente en Python. Para hacer frente a esto, Sagemath admite una "versión" compilada de Python

llamada Cython (cfr. [Equ18a] y [Gre06]). Cython es una extesión de Python y C, simultáneamente similar a ambos. Están permitidas la mayoría de las construcciones de Python, incluidas: listas por comprensión, expresiones condicionales, código como +=; también puede importar código que haya sido escrito en otros módulos de Python. Por otra parte, es posible declarar variables C arbitrarias y se pueden realizar directamente llamadas a biblioteca arbitrarias de C. El código resultante se convierte en código C y es compilado utilizando un compilador de C.

Para crear un código propio de Sagemath compilado, basta con dar al archivo una extensión .spyx (en lugar de .sage). Si se trabaja con la interfaz de línea de órdenes, se puede adjuntar y cargar código compilado exactamente como con el código interpretado (en este momento, no se admite adjuntar y cargar código de Cython con la interfaz del notebook). La compilación real es realizada "detrás de escena" sin que tenga que hacer nada explícitamente. La biblioteca de objetos compartidos compilados se almacena en:

$HOME/.sage/temp/hostname/pid/spyx

Estos archivos son eliminados al salir de Sagemath.

No se aplica ningún tipo de análisis previo de Sagemath a los archivos de spyx, por ejemplo, 1/3 dará como resultado 0 en un archivo spyx en lugar del número racional 1/3. Si foo es una función en la biblioteca de Sagemath, para usarla desde un archivo spyx debemos importe sage.all y usar sage.all.foo.

Es posible acceder a funciones de C definidas en un fichero *.c. Sirva de ejemplo lo siguiente. Creemos ficheros test.spyx y test.c en el mismo directorio. El código C puro estará en el fichero test.c:

```
int add_one(int n) {
  return n + 1;
}
```

El código Cython estará en test.spyx que tendrá el siguiente contenido:

```
cdef extern from "test.c":
    int add_one(int n)

def test(n):
    return add_one(n)
```

Entonces todo funcionará según el siguiente diálogo:

```
sage: attach("test.spyx")
Compiling (...)/test.spyx...
sage: test(10)
11
```

Si es necesaria una biblioteca, digamos `foo`, para compilar el código C generado por un fichero Cython, debe añadirse la línea

```
clib foo
```

a la fuente Cython. Similarmente puede ser añadido en la compilación un fichero adicional, digamos `bar`, con la declaración:

```
cfile bar
```

Finalmente, se puede interactuar con un único esquema `Python/Sagemath` como el siguiente:

```
#!/usr/bin/env sage

import sys
from sage.all import *

if len(sys.argv) != 2:
    print("Usage: %s <n>" % sys.argv[0])
    print("Outputs the prime factorization of n.")
    sys.exit(1)

print(factor(sage_eval(sys.argv[1])))
```

Para usar este script, el directorio base de `Sagemath` debe estar en el `PATH`. Si el anterior esquema es llamado `factor`, lo que sigue es un ejemplo de cómo usarlo.

```
$ ./factor 2006
2 * 17 * 59
```

Todos los objetos de `Sagemath` tienen un tipo bien definido adjudicado. Existen los tipos de Python y muchos más añadidos por `Sagemath`. Algunos tipos propios de Python han sido muy potenciados con funcionalidades propias de `Sagemath`, es el caso del tipo `Set`.

3. Instalación

El usuario de Linux puede instalar `sagemath` con su gestor de instalación habitual. Por ejemplo, para los derivados de Ubuntu existe una una PPA que añadir al sistema y entonces el procedimiento de instalación sería el siguiente:

```
sudo apt-add-repository -y ppa:aims/sagemath
sudo apt update
sudo apt install sagemath-upstream-binary
```

Si queremos cambiar "facilidad" por "software actualizado", procederemos a una instalación manual (cfr. [Yá17]). La segunda gran ventaja es que este procedimiento permite disponer de versiones.

La descarga del software se realizará desde http://www.sagemath.org/download-linux.html recibiendo, digamos en `Downloads`, un fichero acorde a nuestra arquitectura y distribución que, para fijar ideas, tendrá un identificador similar a:

`sage-8.3-Ubuntu_16.04-x86_64.tar.bz2`

El procedimiento para la instalación es básicamente una descompresión y una localización del ejecutable, según los siguientes pasos:

- Copiar en el directorio `/opt` por ser el idóneo para lo que sigue. Esto lo conseguimos con la orden (no olvidar sustituir `myUser` por su nombre de usuario en su equipo):

  ```
  sudo cp /home/myUser/Downloads/sage-8.3-Ubuntu_16.04-x86_64.tar.bz2 /opt
  ```

- Descomprimimos el fichero:

  ```
  sudo tar -xvf /opt/sage-8.1-Ubuntu_16.04-x86_64.tar.bz2
  ```

- Con el fin de suprimir lo que ya no es necesario, borrar el fichero comprimido, copiado previamente en `/opt`, mediante la orden:

  ```
  sudo rm /opt/sage-8.1-Ubuntu_16.04-x86_64.tar.bz2
  ```

- Hacer un nexo simbólico del ejecutable a un lugar incluido en el camino para que el sistema reconozca el ejecutable como orden:

  ```
  sudo ln -s /opt/SageMath/sage /usr/local/bin/sage
  ```

- Ejecutar desde la terminal la orden:

  ```
  sudo make distclean && make
  ```

- Encontrar el camino al emblema de sagemath para nuestro eventual lanzador. Para ello podemos usar la orden locate:

  ```
  sudo updatedb && locate icon48x48.png | grep ^/opt
  ```

  Al ejecutar esa orden obtedríamos una información del siguiente tenor:

  ```
  /opt/SageMath/local/lib/python2.7/site-packages/sagenb/data/sage/images/icon48x48.png
  ```

- Para trabajar cómodamente, generar un icono de lanzamiento de sagemath (si no se desea tener, podemos saltar este punto). Para ello escribiremos en la terminal:

  ```
  sudo nano /usr/share/applications/sage.desktop
  ```

  En este archivo anotaremos la información para crear nuestro lanzador. Un ejemplo sería:

  ```
  [Desktop Entry]
  Name=Sagemath
  Comment=SageMath
  Exec=sage -notebook=jupyter
  Icon=/opt/SageMath/local/lib/python2.7/site-packages/sagenb/data/sage/images/icon48x48.png
  Terminal=false
  Type=Application
  ```

  Guardamos el archivo con la combinación ctrl+x e Y y será creado.

  Aprecie que Exec=sage -notebook=jupyter podría ser sustituido por Exec=sage -notebook=sagenb si deseásemos, o necesitásemos, usar el entorno de trabajo sagenb tradicional de sagemath.

Así pues si se quisiese entrar en una sesión de sagemath desde la terminal, basta ejecutar la orden:

```
sage
```

de la que saldríamos con exit(), pero si se quiere entrar en modo gráfico con jupyter habría que ejecutar:

```
sage -notebook=jupyter
```

y si se prefiere el entorno de trabajo tradicional:

```
sage -notebook=sagenb
```

La instalación en Mac OS es la más sencilla. Consiste en descargar el paquete de instalación `.dmg` de su repositorio preferido eligiendo previamente el idóneo para su arquitectura, que casi seguro será Intel de 64 bits. Proceda ahora a instalar como lo hace habitualmente con cualquier app de Mac OS. Tome las precauciones habituales para poder instalar software de desarrolladores no identificados y valore la creación previa del directorio `/Users/myUser/.sage`, lo que no será necesario si tuvo una instalación previa o si va a utilizar `jupyter`. Si tuvo una instalación previa, lo apropiado es conservar las hojas de trabajo y borrar el contenido del directorio `.sage` antes de la primera ejecución de `sagemath`; tras la misma, recuperará las hojas de trabajo reservadas como se indicó más arriba.

Si es usuario de Windows, puede usar los contenidos de virtualización que ofrece el sitio oficial (desaconsejado) o bien descargar este instalador para Windows y seguir las instrucciones (cfr. [Sag18]).

4. Recuperación

`sagemath` guarda las hojas de trabajo de `sagenb` en la siguiente carpeta:

```
/home/myUser/.sage/sage_notebook.sagenb/home/admin
```

en los sistemas Linux; en los sistemas Mac OS son guardadas en

```
/Users/myUser/.sage/sage_notebook.sagenb/home/admin
```

de forma que si se ha tenido una instalación previa y se quieren restaurar las hojas de trabajo suyas en otra futura, debemos guardar el contenido de dicha carpeta y copiarlo sin más en la del mismo nombre de la nueva instalación.

Un inconveniente de trabajar con `sagenb` es la imposibilidad de guardar las hojas de trabajo en la nube; por ello se ha dado recientemente en utilizar `jupyter` para editar las hojas de trabajo. Pero ¿cómo convertir las antiguas hojas de trabajo de `sagenb`, de extensión `sws`, hasta tener formato `ipynb` para ser utilizadas por `jupyter`?

La idea es usar `sagenb-export` (cfr. [Vol18]). Para instalarlo:

- Si somos usuarios de Linux o Mac OS podemos ejecutar en la terminal:

  `pip install git+https://github.com/vbraun/ExportSageNB.git`

  valore, en su caso si usar 'pip3' en lugar de 'pip'. Alternativamente podemos ejecutar:

  `sage -pip install git+https://github.com/vbraun/ExportSageNB.git`

- Si somos usuarios de una distribución Linux basada en Debian, podría ejecutar la orden:

  `sudo apt install python3-sagenb-export`

  aunque siempre es recomendado instalar los complementos de `Python` mediante `pip` en lugar de con herramientas externas.

Una vez que tengamos a nuestra disposición `sagenb-export`, podremos convertir los ficheros `.sws` alojados en los directorios antes dichos a formato `.ipynb`, legible por `jupyter`. Para ello ejecutaremos la siguiente orden en la línea de la consola:

```
sagenb-export --list
```

El diálogo que produciría será algo así como éste:

```
Equipo:~ user$ sagenb-export --list
 Unique ID       | Notebook Name
 ------------------------------------
 admin:0         | lecture_00
 admin:13        | lecture_01
 admin:14        | lecture_02
 admin:15        | lecture_03
 ...
```

Entonces la orden desde consola:

```
sagenb-export --ipynb=Output.ipynb admin:0
```

produciría el fichero `lecture_00.ipynb` en la carpeta desde la que se ha ejecutado y ya podría ser abierto con, por ejemplo, la orden:

```
sage -notebook=jupyter lecture_00.ipynb
```

5. Una Sesión Típica

El sistema sagemath es especialmente apropiado para el cálculo simbólico en matemática discreta, que es la continuación de la lógica por otras vías, y ha sido muy utilizado para aplicaciones criptográfica. En el trabajo de Minh Van Nguyen [Min09] encontramos suficientemente explicado el sustento teórico de algunos sistemas criptográficos sencillos y una aplicación sobre la terminal de sagemath.

Veremos aquí un ejemplo de ese trabajo sobre Mini-AES, al que sagemath dedica una biblioteca:

sage.crypto.block cipher.miniaes.MiniAES

No tiene la fortaleza del criptosistema AES (Advanced Encryption Standard), aunque Mini-AES es un AES completo con gran valor pedagógico. El ejemplo se muestra por el siguiente diálogo.

Cifrado de texto llano:

```
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: K = FiniteField(16, "x")
sage: MS = MatrixSpace(K, 2, 2)
sage: P = MS([K("x^3 + x"), K("x^2 + 1"), K("x^2 + x"), K("x^3 + x^2")]); P
[ x^3+x x^2+1]
[ x^2+xx^3+x^2]
sage: key = MS([K("x^3 + x^2"), K("x^3 + x"), K("x^3 + x^2 + x"), K("x^2 + x + 1")]); key
[ x^3+x^2 x^3+x]
[x^3+x^2+x x^2+x+1]
sage: C = maes.encrypt(P, key); C
[            x    x^2 + x]
[x^3 + x^2 + x    x^3 + x]
```

Descifrar el resultado

```
sage: plaintxt = maes.decrypt(C, key)
sage: plaintxt; P
[ x^3+x x^2+1]
[ x^2+x x^3+x^2]
[ x^3+x x^2+1]
[ x^2+x x^3+x^2]
sage: plaintxt == P
True
```

Podemos trabajar también con cadenas binarias

```
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: bin = BinaryStrings()
sage: key = bin.encoding("KE"); key
0100101101000101
sage: P = bin.encoding("Encrypt this secret message!"); P
01000101011011100110001101110010011110010111000001110100001000000111010001\
10100001101001011100110010000001110011011001010110001101110010011001010111\
01\
00001000000110110101100101011100110111001101100001011001110110010100100001
sage: C = maes(P, key, algorithm="encrypt"); C
10001000101001101111000001111000010011001110110101000111011011010101001011\
11\
01111101011001110011100100011101100101010100010100111110110011001010001000\
01\
11011011010010000011000110001100000111000011100110101111000000001110001001
sage: plaintxt = maes(C, key, algorithm="decrypt")
sage: plaintxt == P
True
```

Ahora trabajamos con enteros n tales que $0 \leq n \leq 15$:

```
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: P = [n for n in xrange(16)]; P
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
sage: key = [2, 3, 11, 0]; key
[2, 3, 11, 0]
sage: P = maes.integer_to_binary(P); P
00000001001000110100010101100111100010011010101111001101111011111
sage: key = maes.integer_to_binary(key); key
0010001110110000
sage: C = maes(P, key, algorithm="encrypt"); C
11001000001000111110010101010101010110111001111100010000111000001
sage: plaintxt = maes(C, key, algorithm="decrypt")
sage: plaintxt == P
True
```

Podemos generar un texto llano aleatorio y una llave secreta aleatoria. Cifraremos el texto llano usando esa llave secreta y descifraremos el resultado. Seguidamente compararemos el texto llano obtenido del descifrado con el texto llano original:

```
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: MS = MatrixSpace(FiniteField(16, "x"), 2, 2)
sage: P = MS.random_element()
sage: key = maes.random_key()
sage: C = maes.encrypt(P, key)
sage: plaintxt = maes.decrypt(C, key)
sage: plaintxt == P
True
```

6. Una Sesión de Laboratorio sobre Grafos

Presentamos aquí una sesión práctica de laboratorio, inédita en cuanto a las soluciones, de la asignatura LMD del Grado en Ingeniería Informática y Matemáticas. Apreciaremos la eficacia de sagemath y la belleza estética de los resultados.

Ejercicio 6.1. *Defina una función de dos variables enteras, con nombre digamos* `grafoPasoCaballo(m,n)`*, que genere un grafo con tantos nodos como casillas tenga un tablero de ajedrez de* `m` *columnas y* `n` *filas; dados dos nodos del grafo, debe establecerse un eje entre ellos cuando, y sólo cuando, un caballo de ajedrez pueda pasar en un salto de la casilla que representa cualquiera de ellos a la casilla que representa el otro. Representar gráficamente el grafo correspondiente a un tablero* 3 *(3 columnas y 5 filas), es decir, representar el grafo* `grafoPasoCaballo(3,5)`*. Representar gráficamente el grafo correspondiente al tablero de ajedrez del juego actual.*

Solución. La solución que podemos aportar se basa en la función `pasoCaballo(m)` que damos y explicamos a continuación:

```
sage: def pasoCaballo(m):
...       """
...       pasoCaballo considera un tablero de ajedrez con
...       m columnas y genera un criterio booleano para relacionar
...       dos de sus casillas. El criterio es que un caballo de ajedrez
...       pueda viajar en un salto de la situada anterior a la situada
...       posterior en el tablero.
...       """
...       return lambda y,x: (y-x == m-2 and mod(x,m) > 1) or \
...                          (y-x == 2*m-1 and mod(x,m) > 0) or \
...                          (y-x == 2*m+1 and mod(x,m) < m-1) or \
...                          (y-x == m+2 and mod(x,m) < m-2)
sage: pasoCaballo(8)(7,16)
False
```

La función solicitada fue en tiempos:

```
sage: def grafoPasoCaballo(m,n):
...       """
...       grafoPasoCaballo genera el grafo cuyos vértices
...       representan a las casillas de un tablero de m
...       columnas y n filas. Hay un eje entre dos vértices
```

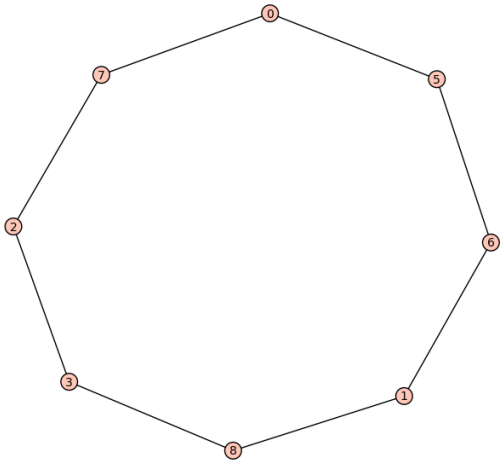
```
...       sii el caballo puede dar un salto de la casilla que
...       representa uno a la casilla que representa el otro.
...       """
...       m, n = int(m), int(n)
...       return Graph([range(m*n),pasoCaballo(m)])
```

Por desgracia se ha perdido la funcionalidad de `Graph()` y actualmente necesitaríamos una elaboración mayor. La función solicitada podría ser algo así como:

```
sage: from itertools import product
sage: def grafoPasoCaballo(m,n):
...   """
...    grafoPasoCaballo genera el grafo cuyos vértices
...    representan a las casillas de un tablero de m
...    columnas y n filas. Hay un eje entre dos vértices
...    sii el caballo puede dar un salto de la casilla que
...    representa uno a la casilla que representa el otro.
...    """
...    m, n = int(m), int(n)
...    data = {}
...    for y, j in product(range(n*m), repeat=2):
...        if y!=j :
...            k = data.get(y, [])
...            if pasoCaballo(m)(y,j):
...                k.append(j)
...            data[y] = k
...    return Graph(data)
```

que tiene el siguiente comportamiento:

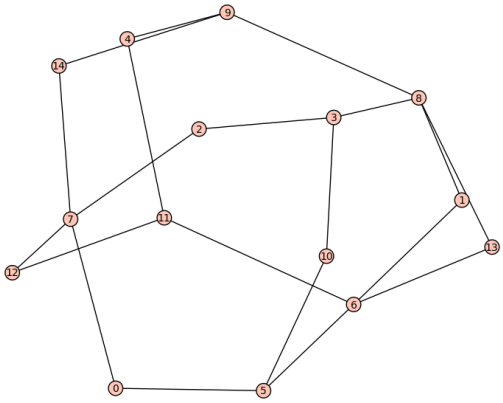
```
sage: grafoPasoCaballo(3,3).is_connected()
False
sage: grafoPasoCaballo(3,3).connected_components_number()
2
sage: grafoPasoCaballo(3,3).connected_components()
[[0, 1, 2, 3, 5, 6, 7, 8], [4]]
sage: L = grafoPasoCaballo(3,3).connected_components_subgraphs()
sage: graphs_list.show_graphs(L)
sage: L[0].plot(); L[1].plot()
```



4

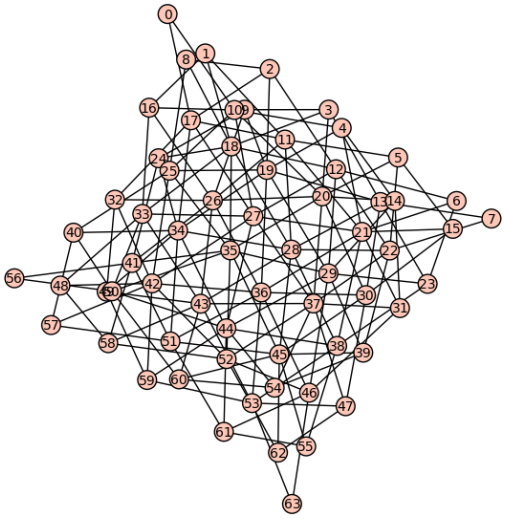
Otros ejemplos son:

```
grafoPasoCaballo(3,5).plot()
```



y

```
grafoPasoCaballo(8,8).plot()
```



Para éste último se tiene:

```
sage: grafoPasoCaballo(8,8).is_hamiltonian()
True
```

□

Ejercicio 6.2. *Definir una función, digamos* `hh(lst)`*, que dada una lista* `lst` *de números enteros determine si es la secuencia de grados de algún grafo, esto es, que implemente el algoritmo de Havel-Hakimi. Seguidamente:*

1. *Probar la función para los valores siguientes de* `lst`*:*
   - *[],*
   - *[0],*
   - *[0,0,0],*
   - *[-1,1,1,1],*
   - *[1,1,1,1,1,1],*
   - *[5,3,1,1,1,1],*
   - *[4,3,3,2,2,1],*
   - *[5,4,3,2,2],*
   - *[5,4,4,2,2,1],*
   - *[4,4,2,2,1,1,1,1] y*
   - *[4,4,3,2,2,2,1].*
2. *¿Existe un grafo regular (es decir, en el que todos sus vértices tienen el mismo grado) de grado 4 con 15 vértices?*

3. *Dado su número personal que ya usamos en la anterior entrega (las cuatro últimas cifras de su tarjeta de identificación), construir la secuencia* `lst` *formada por todos los números pares no nulos menores que él. ¿Es una secuencia gráfica?*

Solución. La definición que aportamos es la que especifica el siguiente retazo de código:

```
sage: def hh(lst):
...       """
...       hh(lst) (hh por Havel-Hakimi) devuelve True sii la lista lst
...       es una sucesión gráfica. Convenimos en que [] es una sucesión
...       gráfica.
...       """
...       sld, l = True, len(lst)
...       if l:
...           lst.sort(reverse=True)
...           b = (lst[-1] < 0) or (lst[0] > l-1) or (mod(sum(lst),2) == 1)
...           if b:
...               sld = False
...           elif lst[0] == 0:
...               sld = True
...           else:
...               sntnl = True
...               while sntnl:
...                   x = lst[0]
...                   del(lst[0])
...                   for i in range(x):
...                       lst[i] -= 1
...                   lst.sort(reverse=True)
...                   if lst[-1] < 0:
...                       sntnl, sld = False, False
...                   elif lst[0] == 0:
...                       sntnl = False
...       return sld
```

que tiene el siguiente comportamiento:

```
sage: hh([])
True
sage: hh([0])
True
```

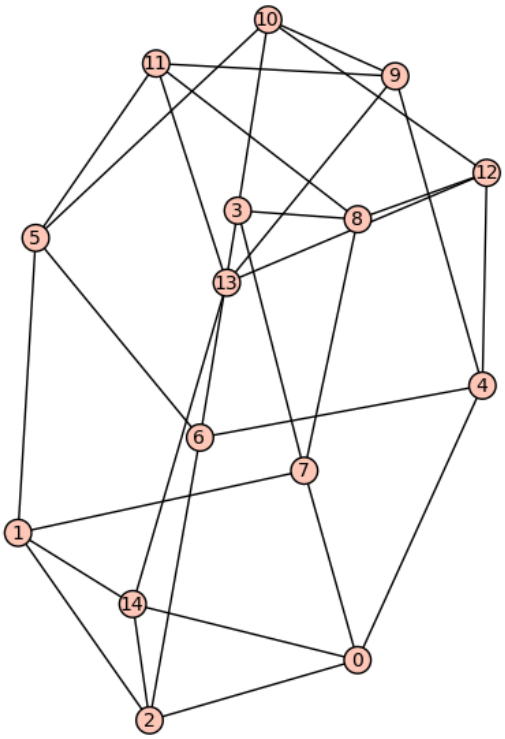
```
sage: hh([0,0,0])
True
sage: hh([1,1,1,1,1,1])
True
sage: hh([1,1,1,1,1])
False
sage: hh([4,3,3,2,2,1])
False
sage: hh([5,4,3,2,2])
False
sage: hh([5,4,4,2,2,1])
False
sage: hh([5,4,4,2,2,0])
False
sage: hh([4,4,2,2,1,1,1,1])
True
sage: hh([4,4,3,2,2,2,1])
True
sage: hh([4,4,4,4,4,4,4,4,4,4,4,4,4,4,4])
True
sage: hh([5,3,1,1,1,1])
False
```

Para la segunda pregunta, la respuesta es afirmativa puesto que:

```
sage: hh([4]*15)
True
```

de hecho uno de tales grafos es el siguiente:

```
graphs.RandomRegular(4, 15).plot()
```



En cuanto a la tercera prengunta, la respuesta es negativa pues:

```
sage: hh(range(2,7543,2))
False
```

□

7. Conclusión

Sage es una herramienta de cálculo que puede cubrir con creces las necesidades universitarias en los laboratorios técnicos, compitiendo incluso en esa labor con afamados sistemas de cálculo. Su licencia GNU General Public License es quizá la razón definitiva para su adopción.

Sage es un sistema relativamente fácil de usar por la posibilidad de programarlo mediante Python. Éste es un lenguaje que vive un periodo de gran auge y desarrollo, con notables y justificadas capacidades para el cálculo científico y técnico.

Pese a servirse de otros sistemas de cálculo, más arriba nombrados, la intervención de Cython, que fue desarrollado a propósito como bifurcación de Pyrex, hace a Sage veloz

en los cálculos. Esto es considerado como un gran acierto y puede que la única opción.

Por todo lo dicho consideramos que Sage una opción muy a tener en cuenta para la labor docente e investigadora en el ámbito universitario.

Referencias

[Equ18a] Equipo de desarrollo de Cython: *Cython C-Extensions for Python*. http://cython.org/. Version: august 03 2018. – Wiki for Cython

[Equ18b] Equipo de desarrollo de Sage: *Sage Tutorial (Release 8.3)*. https://doc.sagemath.org/pdf/en/tutorial/SageTutorial.pdf. Version: august 04 2018. – Exhaustive tutorial

[Gre06] Greg Ewing: *Pyrex - a Language for Writing Python Extension Modules*. http://www.cosc.canterbury.ac.nz/greg.ewing/python/Pyrex/. Version: march 2006. – Guide for Pyrex

[Min09] Minh Van Nguyen: *Exploring Cryptography Using the Sage Computer Algebra System*. https://www.sagemath.org/files/thesis/nguyen-thesis-2009.pdf. Version: 2009. – Thesis for the Degree of Bachelor of Science.

[Sag18] Sagemath: *Sagemath*. https://github.com/sagemath/sage-windows/releases. Version: 2018. – Windows Installer

[Vol18] Volker Braun: *Convert SageNB Notebooks*. https://github.com/vbraun/ExportSageNB. Version: 2018. – Export SageNB Notebooks (to Jupyter)

[Yá17] Yábir García Benchakhtir: *Instalar SageMath en Mac OS , Windows, Ubuntu y otros Linux*. https://wildunix.es/posts/instalar-sage-en-mac-os-x-ubuntu-y-otros-linux-uso-bajo-windows/. Version: 2017. – Iniciativas para difusión del software libre