IMPLEMENTACIÓN E DESENVOLVEMENTO DE AULAS DE MATEMÁTICAS AVANZADAS EN SAGE

Francisco de Arriba Pérez, franfuco4444@gmail.com
Eusebio Corbacho Rosas, corbacho@uvigo.es
Mª Carmen Somoza López, carmensomoza@uvigo.es
Ricardo Vidal Vázquez, rvidal@uvigo.es

Universidade de Vigo
Servizo de Publicacións
2018

MANUAIS DA UNIVERSIDADE DE VIGO, Nº. 72

Edición
Servizo de Publicacións da Universidade de Vigo
Edificio da Biblioteca Central
Campus de Vigo
36310 Vigo

ISBN: 978-84-8158-796-8
D.L.: VG 558-2018

Impresión: Tórculo Comunicación Gráfica, S.A.

PRÓLOGO

Cando os avaliadores do comité EAC (Engineering Acreditation Comission) de ABET (Accreditation Board for Engineering and Technology) dos EE.UU., visitaron a Escola de Enxeñería Industrial de Vigo co obxecto de acreditar o máster en Enxeñería Industrial, interesoulles saber como se ensinaban as matemáticas aos futuros profesionais da Enxeñería Industrial. O coordinador do máster contestoulles que se abordaba dunha forma bastante orixinal e cun seguimento individualizado por parte do profesor. Por iso quixeron entrevistarse coa persoa que explicaba matemáticas aos alumnos deste máster da Universidade de Vigo. Seica esperaban a algún novo profesor, recén licenciado, con ideas frescas sacadas das novas pedagoxías. E quedaron un pouco sorprendidos cando comprobaron que o profesor era alguén veterano…, en fin, digamos, xa "*maduro*" nestas lides. Despois das súas explicacións cheas de entusiasmo, de ganas por transmitir os seus coñecementos, de ver ese rostro que se acende cando fala de matemáticas, que rezuma paixón pola súa profesión de profesor, o avaliador de ABET non puido máis que afirmar: "*gustaríame que vostede tivera sido o meu profesor de matemáticas na carreira de enxeñería*". Ese é o noso querido profesor Eusebio Corbacho Rosas que encabeza o grupo de autores deste libro.

Esta obra que o lector ten entre as súas mans ou na súa pantalla do computador, é o froito da vocación dos seus autores pola transmisión do coñecemento, pola docencia, pero é tamén o froito da adaptación aos novos tempos. É o froito do gusto polo traballo ben feito, do amor aos seus alumnos.

O libro está especialmente dirixido aos alumnos das materias Matemáticas da Especialidade pertencente ao terceiro curso do Grao en Enxeñería en Tecnoloxías Industriais; Complementos de Formación, do terceiro curso do Grao en Enxeñería en Electrónica Industrial e Automática e tamén da materia Métodos Matemáticos, pertencente ao primeiro curso do Máster en Enxeñería Industrial. Todas elas materias ofertadas pola Universidade de Vigo na Escola de Enxeñería Industrial.

Como ben constataron os avaliadores de ABET, esta obra é un exercicio de adaptación aos novos tempos, pero saiba o lector que nesta modesta obra sintetízanse máis de 40 anos de profesión, de vocación docente en institucións de educación superior e sempre tratando de transmitir aos demais o seu amor polas matemáticas.

Trátase dunha guía eminentemente práctica, que vai levando da man a quen se queira deixar, pola fascinante aventura de resolver problemas físicos e de enxeñería mediante unha ferramenta tan útil e versátil como o Sage. O lector terá ocasión de utilizar esta ferramenta gratuíta desde o primeiro momento seguindo as instrucións aquí recollidas.

Esta obra incita ao lector a abordar os problemas en primeira persoa, é dicir, a elaborar os seus propios programas na linguaxe Phyton que é na que traballa Sage. De tal maneira que se produza un exercicio de formación non só no campo das matemáticas, senón tamén na capacidade de abstracción, na capacidade do futuro profesional da enxeñería para ser capaz de pasar dos problemas individuais, concretos ás solucións xerais. E unha vez realizada a formulación teórica xeral, é posible individualizar as solucións, resolver os problemas concretos sen máis que incorporar as condicións particulares que se cumpren para ese determinado problema.

Tan só esperamos que o Profesor Corbacho e o seu equipo formado por Francisco de Arriba, Carmen Somoza e Ricardo Vidal, sigan abrindo camiño nesta fascinante aventura que é achegar as matemáticas ao mundo da enxeñería e que o lector disfrute percorréndoo.

Vigo, no Día de Galicia de 2018.

Juan M. Pou Saracho
Catedrático de Física Aplicada
Exdirector da Escola de Enxeñería Industrial de Vigo

Táboa de contidos

1. Introdución

SAGE (Software for Algebra and Geometry Experimentation), é un software matemático de acceso gratuíto, que conta con varias versións descargables a través da súa páxina web para facelo compatible cos sistemas operativos máis comúns: Windows, Linux, MacOS. Esta ferramenta vainos permitir desenvolver dende simples ecuacións ata complexos problemas n-dimensionais. A linguaxe de programación na que traballa Sage é Python, esta linguaxe en auxe nos últimos anos conta co apoio de multitude de creadores de software independentes que enriquecen as súas librerías de cálculo dunha forma constante. Ademais Sage inclúe as súas propias funcións que facilitan a realización de operacións matemáticas dunha forma moi similar a outros softwares comerciais, pero na nosa opinión dunha forma moito máis intuitiva.

A principal vantaxe deste software é que é gratuíto e accesible a calquera docente ou estudante que desexe utilizalo. É habitual que cada profesor teña que dar clases en diferentes grupos de laboratorio e Sage nas súas versións 6.x permite crear no PC/pórtatil do profesor un "usuario-ubuntu" por cada laboratorio que imparte que será un espazo novo de traballo, illado do resto, con utilidades adecuadas a cada unidade temática almacenada nos arquivos DATA e cos que os alumnos poden confeccionar as súas propias caixas de ferramentas matemáticas. Os exercicios realizados polos alumnos ao longo do curso vanse almacenando no servidor do profesor tendo así constancia dos mesmos e facilitando a avaliación continua dos alumnos.

Este servidor permite aos alumnos usando tan só un navegador, coma por exemplo Firefox, usar esta ferramenta na clase sen necesidade de instalar nada, co único requisito de atoparse conectados á mesma rede de Internet que a do profesor. Estando limitado o tempo de uso como servidor, ao período da clase, onde os usuarios deben realizar labores pautadas, estimamos que as exixencias de seguridade nestes contextos non son de máxima prioridade.

Ao alumno tamén se lle indica como instalar unha distribución de Sage no seu ordenador para traballar pola súa conta fora da clase. Polo tanto podemos distinguir dous tipos de instalacións:

- Instalación de Sage en Ubuntu. Isto outorgaranos todas as funcionalidades de Sage, poderemos usalo de forma individual ou como servidor. Esta modalidade é a que deben utilizar os profesores que dexesen dar funcionalidades de servidor.
- Instalación de Sage en MacOS ou en Windows a través dun sistema operativo virtualizado. En Windows, este tipo de instalación só permitiranos un uso persoal. É a instalación preferida por alumnos non familiarizados con outros sistemas operativos.

Este manual inclúe, no primeiro capítulo, unha indicación detallada de como preparar a aula informática para usar o portátil do profesor coma un servidor para o desenvolvemento da clase e con iso facilitar así a incorporación doutros profesores a esta iniciativa de baixo custo e baixo mantemento, usando un potente software para a investigación e o traballo en matemáticas. Tamén inclúe as instrucións pertinentes para

que os alumnos instalen no seu ordenador unha versión de Sage a través dunha máquina virtual.

Neste texto todos os cálculos foron comprobados usando a versión 6.10 de Sage, última no momento da primeira redacción do manual e por tanto, é a que compre instalar os alumnos. A gran actividade dos creadores de software de Sage fai que nestes momentos dispoñamos das versións 7.x e 8.x pero nestas versións aparecen distintos tipos de erros nos exercicios presentados. Por exemplo, na versión 7.4 a orde **rank(**A) colapsa o sistema nalgunhas matrices elementais. Nas versións 8.x non é posible visualizar os arquivos DATA dificultando que os alumnos se acostumaren a preparar a súas propias caixas de ferramentas matemáticas. Tampouco se poden crear servidores multi-conta no Sage Notebook entorpecendo o uso do portátil do profesor como servidor nas diferentes clases que teña encargadas. Cónstanos que se está a traballar na resolución destes problemas pero só cando estean superados poderemos adaptar o noso manual a novas versións.

Os seguintes capítulos, que denominamos Worksheet 0 e Worksheet 1, son unha copia facsimilar en PDF dunha selección dos arquivos de Sage onde os alumnos dos catro últimos anos nas materias de Matemáticas da Especialidade e Complementos de Formación dos Grado de Tecnoloxías Industriais e Electrónica Industrial e Automática e Métodos Matemáticos do Mestrado de Enxeñería Industrial, manexaron no ordenador nas aulas informáticas da Escola de Enxeñería Industrial da Universidade de Vigo. Con este manual achégase unha copia das citadas prácticas en formato electrónico accesibles na dirección web http://corbacho.webs.uvigo.es/ e que lle permitirá practicar pola súa conta, comprobar resultados e ampliar con novos exercicios que a súa formación requira.

Os autores deste manual constitúen un equipo multidisciplinar no que Francisco de Arriba, Enxeñeiro de Telecomunicacións que foi becario do departamento de Matemática Aplicada I, se ocupou do software soporte das aulas virtuais, Eusebio Corbacho e Ricardo Vidal, profesores do mencionado departamento, desenvolveron estas prácticas sobre os fundamentos teóricos recollidos nos textos [4] e [5] da Bibliografía, onde poden verse en detalle as demostracións que neste manual se presentan sucintamente. Carmen Somoza, doutora do citado departamento, exerceu de coordinadora e ocuparase de trasladar está iniciativa, tanto presencial coma a distancia, á educación secundaria onde é profesora.

Para os autores significa unha recompilación do traballo realizado durante este tempo que se mostrou frutífero para a formación dos alumnos. Comprobamos que con estas prácticas os alumnos lograron realizar traballos de desenvolvemento, algúns dos cales están recollidos nos diferentes problemas e exercicios.

Neste manual puidéronse incorporar unicamente dúas worksheets (follas de traballo ou prácticas) por razóns de extensión:

- A Worksheet 0, de carácter preliminar, con cuestións xerais de Sage e repaso de conceptos matemáticos que o alumno debe ter previamente a cursar as materias mencionadas.
- A Worksheet 1 trata de problemas inversos lineais e non lineais, incluíndo

problemas de axuste lineal e de axuste a un modelo.

Nos DATA destas Worksheets inclúense unha grande cantidade de funcións de elaboración propia plantexadas como pura implementación en Sage da teoría presentada. Chamarémolas simplemente funcións para distinguilas das implementadas en Sage que chamaremos ordes. A creación destes DATA proporciónanlle ao alumno solvencia en programación Phyton e permítenlle construír a súa propia caixa de ferramentas para abordar futuros problemas. Os DATA por figurar na versión electrónica non se inclúen neste manual xa que se poden ver se se usa a versión de Sage 6.10 recomendada.

Debido á extensión do manual, preténdese presentar en futuros volumes o resto das prácticas das materias mencionadas. O segundo, trataría de ecuacións diferenciais e xeometría discreta, e o terceiro de variable complexa e transformadas integrais.

2. Instalación de Sage

2.1. Instalación de Sage sobre Ubuntu

Esta instalación é a preferida polos autores do libro ao presentar varias vantaxes. Unha das máis importantes é que ao usar Ubuntu (Software libre) nos independizamos de sistemas operativos privativos (Windows, MacOS), desta forma non estamos atados a actualizacións que supoñen un sobre custo ou a políticas de empresas que moven os seus intereses en función do mercado. Ademais, Sage, desde as súas orixes, nace con esta mesma filosofía de Software libre e está optimizado para contornas Linux, ambas cuestións fan que nos decantemos por este sistema operativo. Isto vainos a permitir construír a custo 0 (unicamente o custo do PC/portátil) nosas aulas virtuais de ensino.

Para empezar é necesario instalar o sistema operativo Ubuntu no noso PC. Esta instalación require de certos coñecementos sobre computadores, para a súa realización, existen multitude de guías e foros en Internet dende onde poder realizar o proceso completo e consultar erros ou peculiaridades de cada dispositivo, algúns exemplos son:

- https://www.Ubuntu .com/ - Páxina oficial
- http://computerhoy.com/paso-a-paso/software/como-instalar-Ubuntu -equipo-windows-35619
- https://www.genbeta.com/paso-a-paso/linux-paso-a-paso-instalar-Ubuntu -junto-a-windows-7
- https://www.muycomputer.com/2017/10/24/windows-10-y-Ubuntu -17/

Para atopar información sobre como instalar Ubuntu con Windows, ou Ubuntu de forma única, podemos utilizar Google e as palabras crave “instalar Ubuntu [versión] en Windows [versión]”, onde versión será a versión de cada un dos sistemas operativos. Para problemas comúns relacionados con Windows 8 ou Windows 10, como que trala instalación non aparece o selector de sistema operativo automaticamente e sempre nos inicie Windows, podemos buscar máis información introducindo a seguinte frase en Google: “Non aparece o GRUB, diríxeme automaticamente a Windows”.

Neste caso esforzámonos por mostrar como sería unha instalación sen erros, situación que debería ocorrer na maioría dos casos.

2.1.1. Instalar Ubuntu

1. IMPORTANTE! Ao realizar os pasos que se presentan a continuación eliminarase Windows e todos os arquivos que conteñan, quedando só instalado Ubuntu. Ademáis é necesario o uso dun pendrive e recomendamos que teña unha capacidade igual ou superior a 8GB.

2. Descargamos a imaxe de Ubuntu da páxina oficial clicando no seguinte link:

https://www.Ubuntu .com/download/desktop e pulsamos sobre o botón **Download** da versión **Ubuntu 16.04.2 LTS**, tal como mostra a imaxe:

3. Descargar **Universal USB Installer**, esta ferramenta para Windows permítenos configurar un USB para que lance un sistema operativo almacenado no seu interior, ao arrincar un PC (usb bootloader)

- Accedemos a https://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/
- Pulsamos sobre o botón 

- Automaticamente iniciarase a descarga.

NOTA: A día 26-04-2018 a descarga de Virtual Box iniciarase directamente accedendo á seguinte url: https://www.pendrivelinux.com/downloads/Universal-USB-Installer /Universal-USB-Installer-1.9.8.1.exe

4. Executamos o arquivo **Universal-USB-Installer**, o que abrirá unha fiestra similar á da figura. No "**Step 1**" seleccionamos Ubuntu. No "**Step 2**" seleccionamos o arquivo de ubuntu.iso que descargamos. E no "**Step 3**" seleccionamos o pendrive onde imos gravar o instalador de Ubuntu. **IMPORTANTE!**. Ao darlle a **Create** o pendrive borrarase por completo perdendo todos os arquivos que conteña.

5. Unha vez finalizado o proceso debemos apagar o ordenador.

6. Acendemos o ordenador e pulsamos **ESC**, **F12**, **Supr** ou **DEL** (**retroceso**) para acceder á **BIOS** e á configuración de arranque (dependerá dun PC a outro a tecla que permitirá acceso á BIOS, repetir o apagado e encendido ata acceder a esta e de non aparecer nada, buscar en Internet o botón que use o seu modelo de dispositivo para acceder á BIOS).

7. Na BIOS debemos buscar unha opción que sexa **Boot/Arranque/Opciones** de **Arranque**, nesta pestana mostrarase a orde en que o ordenador revisa onde hai un sistema operativo e arríncao, debemos mover a opción do USB á primeira posición.

8. Saímos e gardamos (habitualmente é pulsando a tecla F10 e aceptando a fiestra emerxente).

9. Ao saír o ordenador reiniciarase. Se se configurou a BIOS de forma correcta e o pendrive está ben gravado, iniciarase a carga de Ubuntu dende o pendrive.

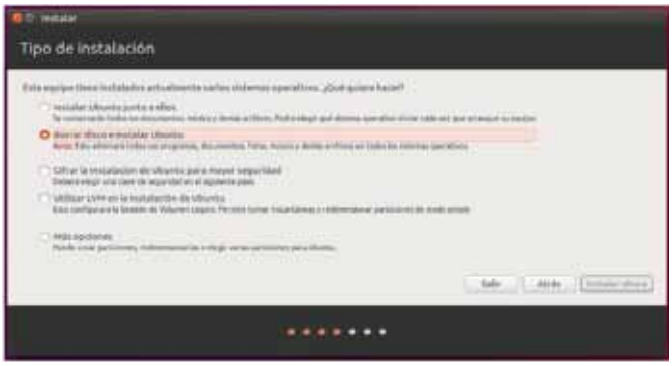
10. Na fiestra que aparece debemos pulsar na opción que indica **"Install Ubuntu "/"Instalar Ubuntu"**.

11. Eliximos o idioma e dámoslle a continuar.

12. Se posuímos algunha rede wifi coñecida seleccionámola e dámoslle a continuar.

13. Seleccionamos a opción **"Instalar software de terceiros para multimedia, MP3,Flash e compatibilidade con gráficas e Wi-Fi"** e pulsamos continuar.

14. Chegaremos a unha fiestra na que se nos pregunta de que forma queremos instalar Ubuntu. Existen dúas opcións, "**Instalar Ubuntu xunto a eles**" (outros sistemas operativos como Windows), isto conservará Windows e permitirá ao iniciar o ordenador elixir que sistema operativo queremos cargar, ou a opción que recomendamos para evitar erros e deixar o ordenador 100% preparado unicamente con Ubuntu que é "**Borrar disco e instalar Ubuntu**" e pulsamos en "**Instalar agora**". Esta opción eliminará todo o que teñamos almacenado no disco duro e instalará Ubuntu encima. Tal como aparece na imaxe



15. A partir deste momento comenzará o proceso de instalación no que se nos pedirá un nome de usuario e un contrasinal.

16. Unha vez acabado debemos apagar o equipo e quitar o pendrive do PC

17. Volvemos acender o PC, nesta ocasión iniciarase Ubuntu cunha interfaz de escritorio semellante a Windows. A partir de aquí xa estamos preparados para instalar Sage.

2.1.2. Instalación de Sage en Ubuntu

O proceso de instalación de Sage unha vez que xa temos Ubuntu, perfeccionámolo ao longo destes anos. De ser unha tarefa que obrigaba a posuír certos coñecementos de sistemas, reducímolo a unha tarefa que consiste en tan só descargar un arquivo comprimido e executar uns poucos comandos nun terminal. Isto deixará unha instalación completamente limpa e funcional deste software.

NOTA: É importante comentar para os lectores máis experimentados na xestión de sistemas, que existe a posibilidade de descargar o instalador de Sage comprimido desde a propia web de Sage, descomprimilo, executar un arquivo con nome Sage e deixar preparado Sage para a súa execución dentro do mesmo usuario onde se instala. Esta modalidade, aínda que fácil de levar a cabo, non prepara o sistema operativo para crear as aulas virtuais das que falamos neste libro. A nosa instalación o que propón é crear un Sage executable a nivel de sistema operativo, non a nivel de usuario. Cada usuario de Ubuntu será considerado un aula virtual (Matemáticas 1, Física 1) e con só unha instalación permitiremos que cada aula virtual nova poida executar un servidor Sage independente do resto, con tan só un comando moi sinxelo e fácil de executar.

1. Primeiro descargamos o ficheiro comprimido https://drive.google.com /open?id=0B9GfBC4n4aiVTnVSX19qWm1ONW8.

2. Descomprimimos o ficheiro pulsando o botón dereito sobre o arquivo "**Instalar_Sage**" e pulsando sobre "**Extraer aquí**". Este cartafol conten 5 arquivos diferentes:

- install: este script instala todo o necesario para lanzar Sage en Ubuntu (instala ferramentas necesarias para a correcta execución de Sage, descarga os ficheiros de Sage, instala Sage, etc.). Componse do seguinte código:

```
Liña necesaria para ser interpretado como un ficheiro executable
#!/bin/bash

Paquetes de Ubuntu necesarios para o correcto funcionamento de Sage
sudo apt-get install libatlas-base-dev
sudo apt-get install openssh-server -y
sudo apt-get install openssh-client -y
sudo apt-get install gnutls-bin ssl-cert -y
sudo make-ssl-cert -y

Descarga desde unha url funcional o archivo comprimido de Sage
wget ftp://ftp.fu-berlin.de/unix/misc/sage/linux/64bit/sage-7.6-Ubuntu_16.04-
x86_64.tar.bz2

Descomprime o ficheiro e o move ao cartafol do sistema /opt
```

```
tar xvf sage*
sudo cp -r SageMath /opt

Elimina o cartafol temporal SageMath e crea un enlace simbólico de Sage desde opt a local
rm -r SageMath
sudo ln -s /opt/SageMath/sage /usr/local/bin/sage

Elimina o arquivo descomprimido
rm sage*

Dá permisos de usuario do sistema aos scripts de lanzar Sage, crear e eliminar aulas virtuais
chmod +x 1
sudo cp 1 /usr/local/bin/
chmod +x nuevo
sudo cp nuevo /usr/local/bin/
chmod +x eliminar
sudo cp eliminar /usr/local/bin/

Executa por primeira vez Sage como servidor
sudo 1
```

- novo: permite crear un grupo de laboratorio novo só indicando o nome e o contrasinal. Componse do seguinte código:

```
#!/bin/bash

Texto que se mostrará por pantalla como guía para o usuario que executa o script
sudo echo Escribe o nome do laboratorio e pulsa enter

Recolle na variable ao nome do laboratorio
read a

Executa o comando de Ubuntu adduser para crear o usuario
sudo adduser "$a"

Engade ao novo usuario ao grupo de usuarios root do sistema
sudo adduser "$a" sudo
```

- eliminar: permite eliminar un laboratorio de traballo só especificando o nome. Componse do seguinte código:

```
#!/bin/bash

Texto que se mostrará por pantalla como guía para o usuario que executa o *script
```

sudo echo Escribe o nome do laboratorio a ELIMINAR

Recolle na variable a o nome do laboratorio
read a

Elimina o usuario do sistema
sudo deluser --remove-home "$a"

- 1: permite a execución sinxela de Sage como servidor. Componse do seguinte código:

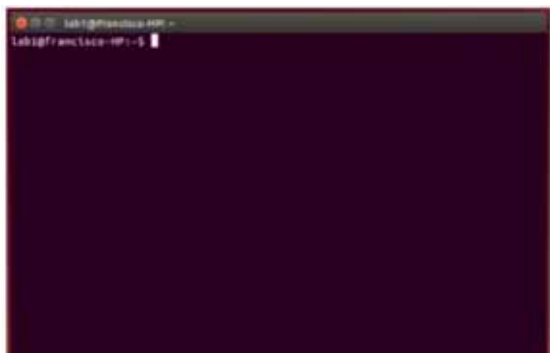
#!/bin/bash

Executa o servidor de Sage. Se queremos executar este mesmo servidor con https en lugar de http, introducir no seguinte comando secure=True, separado cunha coma. Isto fará uso das librerías openssh instaladas con anterioridade.

sage -c "notebook(interface='', port=8080, accounts=True, timeout=7200)"

3. Facemos clic en  , buscamos a palabra terminal e clicamos nunha icona similar a  .

4. Abriránsenos unha fiestra como a mostrada na figura:



5. Movémonos ata o cartafol onde están descomprimidos os arquivos con **cd Descargas/**

6. Escribimos **sudo chmod +x install**, isto iniciará a execución da instalación de Sage, nada máis escribir este comando e pulsar **enter** pedirásenos o contrasinal do usuario, introducímolo e volvemos a pulsar **enter**.

7. A medida que se vaia realizando a instalación irán aparecendo mensaxes por pantalla, algunha delas pode ser que solicite que o usuario escriba "**yes**" no terminal e pulse **enter** para continuar coa instalación.

8. Unha vez finalizou o proceso xa dispoñeremos da nosa distribución de Sage perfectamente instalada.

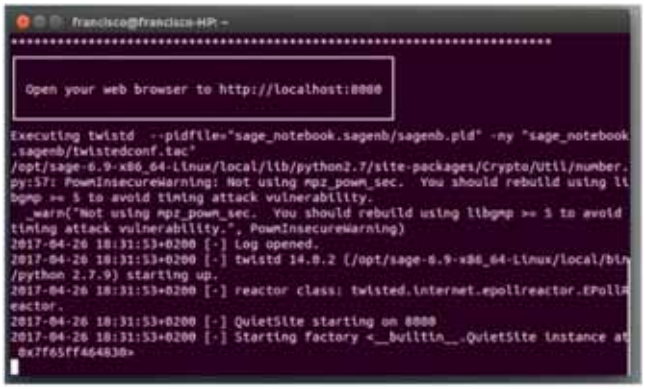
9. Automaticamente cando a instalación finalizou lanzarase o servidor Sage no propio terminal, tal como se mostra na figura. Pedirase que se introduza un contrasinal (contrasinal do usuario admin, este usuario vai ser o que utilice o profesor).

2.1.3. Lanzar Sage

Para lanzar o servidor Sage hai que realizar os seguintes pasos:

1. Abrir un terminal, escribir o número 1 e pulsar enter.

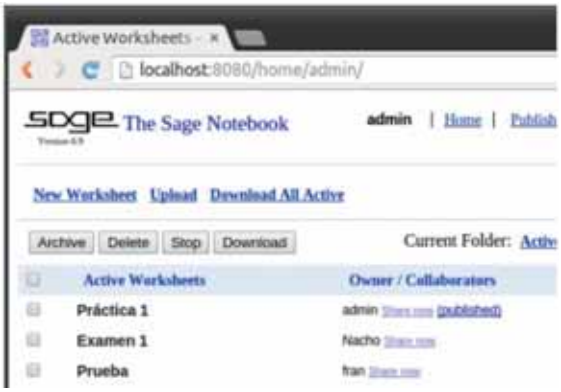
2. O terminal mostrará unha mensaxe como a representada na figura:



3. Abrimos un navegador (por exemplo Firefox) e escribimos na barra de direccións http://localhost:8080.

4. Abrirase unha páxina coa sesión iniciada ou para iniciar sesión, tal como se mostra na figura:

5. Introducimos de nome de usuario **admin** e contrasinal o que asignemos na nosa primeira execución e xa teremos dispoñible a nosa contorna de traballo, tal como se mostra na seguinte figura. Algúns exemplos de uso explicaranse en maior detalle na sección "Primeiros pasos en Sage".



6. Para finalizar o servidor pódese apagar o ordenador directamente ou no propio terminal pulsando **ctrl+c**.

2.1.4. Crear laboratorios de traballo

É habitual que cada profesor teña que dar clases en diferentes grupos de laboratorio. Se só tivésemos un só usuario no noso PC/portátil iríansenos acumulando todos os alumnos nun só Sage, facendo a tarefa de xestión de laboratorios moi complicada. Isto tamén o contemplamos, para evitalo creamos un usuario-ubuntu por cada laboratorio que impartimos. Por exemplo se teño clase en Telecomunicacións na materia de programación 1 e aos grupos de laboratorio 1 e 2, crearemos dous usuarios en Ubuntu. Cada "usuario-ubuntu" será un espazo novo de traballo illado do resto, e neles irase almacenando os exercicios que se vaian realizando cos alumnos ao longo do curso.

1. Para crear un espazo de traballo abriremos un terminal e escribiremos **nuevo** e pulsaremos **enter**.

2. Pedirásenos o contrasinal do usuario no que nos atopamos, introducímola e pulsamos **enter**.

3. Pedirásenos o nome do novo laboratorio, por exemplo **lab_lunes1** e un contrasinal.

4. Pulsamos **enter** varias veces deixando o resto de campos baleiros

5. Finalmente crearase o novo usuario quedando o terminal como mostra a figura

6. Unha vez foi creado poderase acceder a el dende a vista de inicio de Ubuntu ou dende a parte superior dereita pulsando na icona  e logo no nome do novo usuario creado.

2.1.5. Eliminar laboratorios de traballo

Unha vez finalice o curso e se o profesor quere recuperar o espazo ocupado por un laboratorio, poderá eliminar dito grupo realizando as seguintes accións.

1. Abrimos un terminal, escribimos **eliminar**, pulsamos **enter**.

2. Pedirásenos o contrasinal do usuario no que nos atopamos (debe ser diferente do que imos borrar), introducímolo e pulsamos **enter**.

3. Pedirásenos o nome do laboratorio a borrar, por exemplo **lab_lunes1**.

4. Desta forma ese grupo de laboratorio e todos os seus ficheiros serían eliminados.

2.1.6. Usar a propia wifi do móbil para ofrecer Sage

Tamén é común atoparnos con que hai alumnos que levan o seu propio portátil á clase de laboratorio. Nestes casos ideamos unha forma que lles permitirá acceder ao noso servidor facendo uso unicamente dun dispositivo móbil, sen necesidade de conectar ningún cable de rede.

Para iso só é necesario activar unha zona wifi portátil usando un smartphone, este proceso en Android realízase da seguinte forma.

1. Pulsamos en **axustes**.

2. Dámoslle a ...**Máis**.

3. **Compartir internet e zona Wi-Fi**.

4. **Configurar/Activar Zona Wi-Fi portátil**.

5. Neste punto poderemos configurar o nome e o contrasinal da nosa propia wifi e activala para o seu uso. Este proceso representámolo de forma visual na seguinte figura

Unha vez despregada a wifi, dende o PC/portátil, o profesor deberá conectarse a ela pulsando en  , buscando o nome da rede e introducindo o contrasinal que definiu. O alumno tamén deberá facer o mesmo (poderá estar usando calquera tipo de sistema operativo ou tablet, só necesitará un navegador para usar Sage).

IMPORTANTE! Recoméndase usar para esta tarefa un smartphone que o profesor non use, ou un smartphone persoal coa tarifa de datos desactivada, se non, o usuario podería consumir os datos do profesor accedendo a Internet. O importante é que aínda que o profesor non teña Internet no seu smartphone, este sistema permitiralle enlazar

aos seus alumnos co seu ordenador e co servidor Sage que se está executando no seu interior, o que non lles permitirá é acceder a Internet, polo tanto é un método ideal para a realización de exames.

NOTA: Isto mesmo poderíase lograr con routers de pequenas dimensións que con só conectalos ao ordenador poden crear unha wifi LAN.

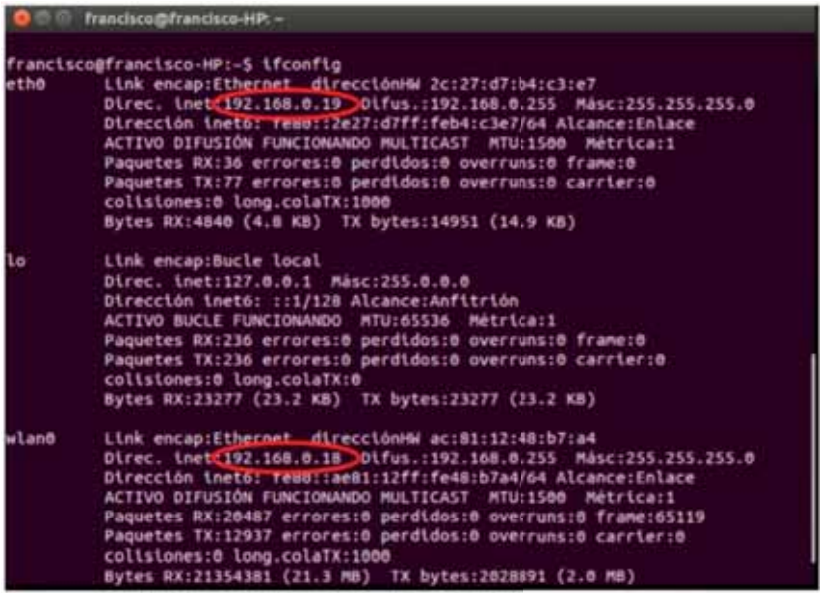
2.1.7. Coñecer a IP do meu computador

Para que un alumno se conecte ao computador do profesor (tal como veremos na seguinte sección), o profesor debe coñecer e mostrar a súa IP ao resto dos estudantes, tanto a IP que consegue desde unha conexión cableada, como a que ten a través da wifi do smartphone.

Para facer isto é necesario seguir os seguintes pasos:

1. Abrir un terminal e escribimos **ifconfig** e pulsamos **enter**.

2. Automaticamente devolveranos a información que vemos na figura. Debemos centrarnos en **eth0** e **wlan0**, estas dúas etiquetas corresponden á rede cableada e á rede wifi e o número que aparece xusto á dereita de **Direc. ine** (aparece marcado en vermello na figura) é a IP que se debe dar aos alumnos (dependendo de se están conectados usando os ordenadores da aula ou por wifi).



2.1.8. Primeiros pasos en Sage

2.1.8.1. Accedendo a Sage/Creando un usuario persoal

Imos mostrar o proceso que deben seguir os usuarios que se queiran conectar ao noso servidor. Mentres os computadores que se queiran conectar se atopen dentro da nosa Área Local (**LAN**), non necesitarán ter nada instalado, bastará cun navegador como

Firefox.

1. Unha vez o servidor Sage foi lanzado cada ususario debe introducir no seu navegador http://IP_LOCAL_ORDENADOR_PROFESOR:8080/.

2. Si un usuario xa creou unha conta usando este servidor, debe introducir o seu nome e contrasinal, e así acceder á súa área de traballo. En caso contrario, para crear unha nova conta, clicar sobre "**Sing up for a new Sage Notebook account**".

3. Posteriormente na nova fiestra, só terán que reencher os campos como mostra a imaxe e pulsar sobre o botón "**create account**". Unha vez feito isto xa poderán iniciar sesión.

2.1.8.2. Crear unha worksheet

Antes de creala, **que é unha worksheet?** unha worksheet non é máis que unha contorna, onde se permitirá escribir e executar funcións matemáticas, crear variables, xogar con números complexos, etc., basicamente é a pantalla de traballo de programas como Matlab.

Para creala hai que facer clic sobre **New Worksheet** (parte superior esquerda da pantalla), e na nova fiestra que se mostra, definir un novo nome, por exemplo "**Proba**". Pulsamos "**Rename**" e a nova folla de traballo estará dispoñible.

Para empezar realizaremos unha proba tan simple como introducir os valores $2+2$ como na imaxe e pulsamos o botón **evaluate**. O resultado se todo sae ben debe ser 4.

2.1.8.3. Coñecendo a interface

Chegados a este punto, é necesario familiarizarse coa interface presente no interior da worksheet. É aconsellable coñecer a contorna de desenvolvemento, polo menos desde unha perspectiva moi xeral.

Empezaremos analizando a parte superior esquerda:

1. **File**: Mostra opcións como: "**crear novo worksheet**", "**descargar worksheet**"..., todas relacionadas co tratamento de novas follas de traballo, tarefas que realizaremos mellor dende a fiestra principal e non dense aquí.

2. **Action**: A acción máis importante é "**Interrupt**", esta finalizará calquera cálculo en execución, problema resultado dun bucle infinito ou ben un algoritmo demasiado custoso de computar, o resultado é que o voso traballo non avance e sexa necesario interrompelo.

3. **Data**: Permite crear arquivos .sage, que serven para programar funcións e darlles uso dentro dun worksheet.

4. **Sage**: Despregará unha lista de opcións que fan referencia á posibilidade que posúe Sage, non só de executarse como unha simple extensión de Python, se non á posibilidade de interpretar outras linguaxes como son HTML, Latex, R.

Pasemos analizar os botóns da dereita:

1. **Print**: Mostra todo o que escribimos na worksheet, nun formato previo a impresión, esta característica permitiranos utilizar as opcións de impresión e entre elas, se está dispoñible, a de exportalo en formato pdf.

2. **Worksheet**: É o botón máis importante de todos os aquí presentes, permitiranos volver á zona de traballo.

3. **Edit**: Opcionalmente, permítesenos traballar en formato editor de texto (nada recomendado para un usuario que utiliza Sage por primeira vez, e que non está afeito ás linguaxes de programación).

4. **Text**: Permitirá visualizar o noso contido en formato terminal, isto é, funciona do mesmo modo que se escribísemos todos os comandos sobre un terminal como o proporcionado por Linux ou Mac (opción non recomendada).

5. **Revisions**; Mostra a información sobre a nosa worksheet.

6. **Share** e **Publish**: Estes botóns permitirannos compartir o noso traballo, no primeiro pedirannos o nome dos usuarios aos que queremos dar permisos para compartir, un exemplo práctico: compartir un arquivo co profesor. Mentres que Publish, daranos a opción de mostrar o noso contido a calquera persoa de Internet xerando automaticamente un enlace para publicalo en calquera web ou aula e desta forma que os alumnos poidan acceder e descargar unha worksheet preparada previamente.

2.1.8.4. Gardar unha worksheet

Todo o traballo que realicemos é necesario gardalo.

Para isto existen os botóns da parte superior esquerda:

1. **Save**: Mentres esteemos traballando de xeito continuado, será máis que recomendable, que se vaia gardando o traballo a medida que se vai realizando, para evitar posibles perdas relacionadas con caídas do servidor, desconexións por problemas na rede, ou apagados inesperados do computador. Desta forma, Save, permítenos ir gardando aos poucos o que imos realizando sen perigo de perdas.

2. **Save & Quit**: Mentres Save garda e mantennos na nosa páxina de traballo, Save & Quit garda e envíanos directamente á nosa páxina principal (tamén chamada Home, onde visualizamos todas as worksheets creadas), esta opción é ideal para cando queremos finalizar a sesión e marcharnos, ou simplemente para abandonar gardando os cambios e dirixirnos a outra folla de traballo.

3. **Discard & Quit**: Permite descartar e volver á nosa páxina principal, ideal para modificacións que danaron o correcto funcionamento dalgunha das nosas funcións.

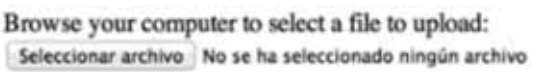
2.1.8.5. Xerar un zip cas miñas worksheets

Supoñamos que tivésedes, como mostra a imaxe, dúas follas de traballo. Proba e Proba_Sage, para gardalas todas nun .zip bastaría con pulsar sobre o botón superior **Download All Active**.

Con todo se só desexamos un deses arquivos, é suficiente con facer clic sobre o cadro en branco que aparece á esquerda do nome (para así seleccionalo) e pulsar sobre o botón Download.

2.1.8.6. Engadindo worksheets almacenadas nun .zip

É necesario pulsar na fiestra principal sobre **Upload**, isto levaranos a outra nova fiestra onde pulsando sobre o botón **Seleccionar archivo** (como se mostra na imaxe), permitiranos seleccionar o noso arquivo .zip e incorporar á nosa sesión, todos os traballos nel almacenados.



2.1.8.7. Borrar unha worksheet

IMPORTANTE!, isto non só borrará a folla de traballo visible, tamén eliminará todos os arquivos DATA ".Sage", creados. Este paso non se pode desfacer e por iso rógase que esta acción só se realice se se posúe total entendemento do que se está facendo.

É suficiente con seleccionar o cadro branco que se atopa á esquerda do nome, e pulsar sobre o botón **Delete**.

2.1.8.8. Creando unha worksheet colaborativa

Outra gran vantaxe de Sage é a posibilidade de realizar un mesmo traballo sobre unha única worksheet, da mesma forma que o administrador ou profesor o fai co seu alumno, pero esta vez entre alumnos.

Para logralo, o "alumno", terá que pulsar sobre Share e incorporar aos compañeiros cos que realizar tarefas, a seguinte figura ilustra como se verá a worksheet, tanto para o administrador como para os alumnos implicados.

2.2. Instalación de Sage en MacOS

A última versión dispoñible na páxina web de SageMath é a 7.4. O método para a instalación de versións anteriores de Sage pasa polo compilado dos arquivos de Sage accesibles a través deste link: http://old.files.sagemath.org/src-old/sage-6.10.tar.gz. Unha vez descargado débense realizar os pasos indicados no arquivo **README.txt**. O contido do arquivo dita o seguinte proceso:

1. Descomprimir o ficheiro.

2. Instalar **Xcode** de forma gratuíta a través da tenda de aplicacións de Apple.

3. Abrir un terminal e executar a seguinte liña de código: **xcode-select --install**.

4. Desde terminal utilizar o comando **cd** ata o directorio onde se atopa o cartafol descomprimido.

5. Entrar no cartafol co comando: **cd sage-*/**

6. Escribir **make** e pulsar **enter**.

Isto executará o compilador de Sage para esa máquina Apple. Este proceso tarda varias horas dependendo do procesador do dispositivo, pode ata demorarse ata 12h.

Para as últimas versións débense seguir os seguintes pasos que deixarán unha páxina Sage operativa nuns poucos minutos:

1. Descargar o arquivo .exe de Sage.

- Para iso accedemos a http://www.sagemath.org/download.html
- Pulsamos no enlace aloxado nalgún dos países cercanos ao noso, por exemplo Portugal, https://mirrors.up.pt/pub/sage/index.html
- Clicamos sobre a categoría "Apple Mac OS X"
- Descargamos a versión de Sage desexada, por exemplo sage-7.4-OSX_10.11.6-x86_64.dmg.

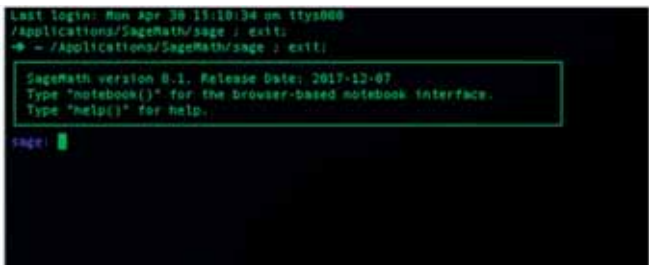
NOTA: A día 26-04-2018 a descarga da versión 7.4 de Sage está disponible accedendo á seguinte url: https://mirrors.up.pt/pub/sage/osx/intel/sage-7.4-OSX_10.11.6-x86_64.dmg.

2. Pulsamos dúas veces sobre o arquivo .dmg. Seleccionamos o cartafol SageMath e a arrastramos ao noso cartafol de aplicacións.

3. Abrimos o cartafol de aplicacións, clicamos có botón secundario sobre a icona de

Sage e seleccionamos **Abrir con Terminal**.

4. Esperar que apareza, na fiestra que se abre, a palabra sage, tal como se mostra na seguinte imaxe:



5. Escribir **notebook()** e pulsar **enter**. Isto iniciará o servidor de Sage.

2.3. Instalación de Sage 6.10 e 7.x en Windows

Dentro desta sección vemos a instalación a través dunha máquina virtual (compatible para calquera versión de Windows)

1. Descargar Virtual Box.

- Acceder a https://www.virtualbox.org/

- Pulsar sobre o botón de Download Virtual Box.
- Na nova páxina, debaixo da sección “VirtualBox 5.2.10 platform packages”, pulsamos sobre a versión de Windows para iniciar a descarga.

NOTA: A día 26-04-2018 a descarga de Virtual Box se iniciará directamente accedendo á seguinte url: https://download.virtualbox.org/virtualbox/5.1.34/VirtualBox-5.1.34-121010-Win.exe

2. Executar e instalar coas opcións por defecto o arquivo VirtualBox.exe.

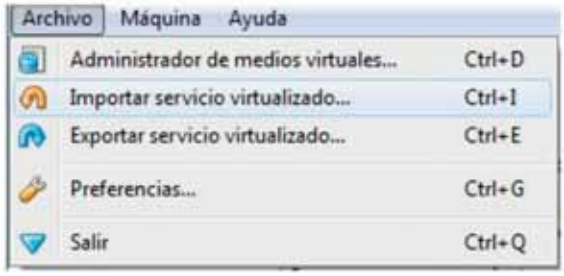
3. Descargar o arquivo **.ova** de Sage.

- Para iso accedemos http://www.sagemath.org/download.html.
- Pulsamos no enlace aloxado nalgún dos países próximos ao noso, por exemplo Portugal, https://mirrors.up.pt/pub/sage/index.html.
- Pulsamos sobre a categoría “OVA Image — Image for virtualization, also a binary distribution for Windows”
- Descargamos a versión de Sage desexada, por exemplo **Sage-6.10.ova**.

NOTA: A día 26-04-2018 a descarga da versión 6.10.ova de Sage está disponible accedendo á seguinte url: https://mirrors.up.pt/pub/sage/ova/sage-6.10.ova.

4. Abrir Virtual Box clicando sobre a icona "**Oracle VM VirtualBox**".

5. Pulsar en arquivo a opción "**importar servizo virtualizado**":



6. Na nova fiestra pulsar en **abrir servicio** e buscar o cartafol onde se dercargou o arquivo **.ova**.

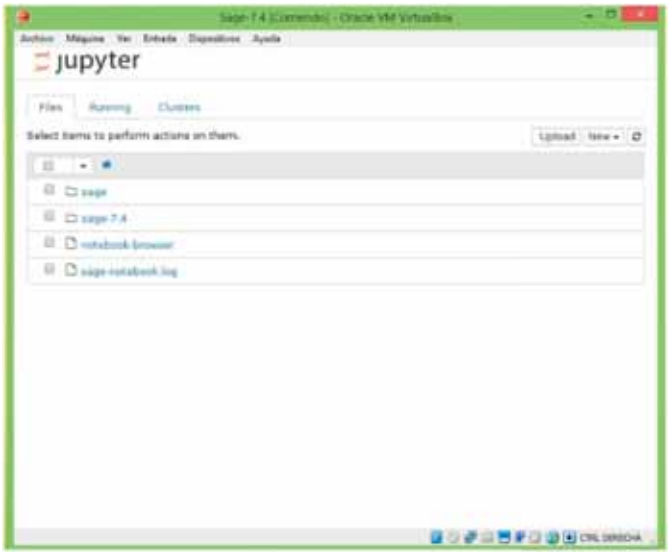
7. Pulsar en **next**.

8. Pulsar **importar**.

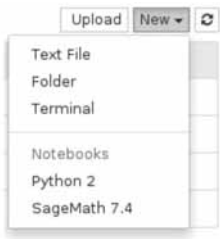
9. Unha vez feito iso pulsar na fiestra de Virtual Box sobre  e no botón  , desta forma lanzarase Sage.

10. A primeira vez que o executemos temos que realizar os seguintes pasos adicionais.

- É necesario ir aceptando as diferentes fiestras emerxentes ata que se mostre por pantalla unha imaxe similar á seguinte:



- Pulsamos en **new** e logo en **Terminal** tal como se mostra na figura.



- Na fiestra de fondo negro que aparece escribimos **nano .xinitrc** e pulsamos **enter**.
- Movémonos coas frechas de teclado ata a liña resaltada na figura adxunta e engadímoslle un **#** ao comenzo da liña, desta forma estaremos comenzando a liña.

```
xset s off
xset dpms force on
setterm -powersave off -blank 0

openbox &

sleep 1
/usr/bin/VBoxClient --display
/usr/bin/VBoxClient --seamless
/usr/bin/VBoxClient --clipboard

export SAGE_BROWSER='/usr/bin/chromium-browser --no-first-run --kiosk'
export BROWSER='/home/sage/notebook-browser'

LOG="$HOME/sage-notebook.log"

# The following line starts the old SageNB notebook
# $HOME/sage/sage --notebook=sagenb interface=\'\' port=8000 automatic_login=True >& $LOG

# The following line starts the new Jupyter notebook
$HOME/sage/sage --notebook=jupyter --ip='*' --port=8000 >& $LOG
```

- Movémonos á liña sombreada na seguinte figura e neste caso eliminamos o **#**, deste modo permitiremos que Sage se inicie de forma automática nas próximas ocasións.

```
 GNU nano 2.0.9                File: .xinitrc                          Modified

xset s off
xset dpms force on
setterm -powersave off -blank 0

openbox &

sleep 1
/usr/bin/VBoxClient --display
/usr/bin/VBoxClient --seamless
/usr/bin/VBoxClient --clipboard

export SAGE_BROWSER='/usr/bin/chromium-browser --no-first-run --kiosk'
export BROWSER='/home/sage/notebook-browser'

LOG="$HOME/sage-notebook.log"

# The following line starts the old SageNB notebook
#$HOME/sage/sage --notebook=sagenb interface=\'\' port=8000 automatic_login=True >& $LOG

# The following line starts the new Juyyter notebook
#$HOME/sage/sage --notebook=jupyter --ip='*' --port=8000 >& $LOG
```

- Para gardar os cambios pulsamos **ctrl+x**, preguntarásenos se queremos gardar, pulsamos a tecla **y** do teclado, preguntarásenos se queremos gardalo co nome actual e sen tocar nada pulsamos **enter**.
- De novo no terminal escribimos **reboot** e pulsamos **enter**, isto iniciará de forma correcta o servidor de Sage.
- A partir de agora sempre que se inicie Sage só haberá que seguir dende o paso 11 en diante (**continuar no paso 11**)

11. Mentres carga Sage débese ir aceptando as diferentes fiestras emerxentes ata que se mostre por pantalla unha imaxe similar á seguinte figura:

12. Abrir un navegador (preferiblemente **Firefox**) e escribir como url: **localhost:8000**. Na nova páxina que aparece escribir como usuario **admin** e como contrasinal **Sage**.

13. E xa poderedes crear os vosos worksheets.

2.4. Instalación das versións Sage 8.1 e posteriores

Dentro desta sección vemos a instalación como unha aplicación máis de Escritorio de Windows (só compatible coas versións máis actualizadas de este sistema operativo)

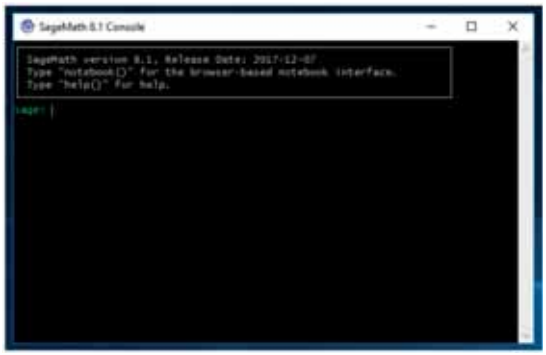
1. Descargar o arquivo .exe de Sage.

- Para iso accedemos a http://www.sagemath.org/download.html
- Pulsamos no enlace aloxado nalgún dos países pretos ao noso, por exemplo Portugal (https://mirrors.up.pt/pub/sage/index.html)
- Pulsamos sobre a categoría “Microsoft Windows”
- Descargamos a versión de Sage desexada, por exemplo Sage-8.1.ova

NOTA: A día 26-04-2018 a descarga da versión 8.1 exe de Sage está dispoñible accedendo a seguinte url: https://mirrors.up.pt/pub/sage/win/SageMath-8.1.exe

2. Executar e instalar coas opcións por defecto o arquivo SageMath-8.1.exe.

3. Ao finalizar executar o lanzador que se creou en escritorio co nome Sagemath8.1  e esperar que apareza, na fiestra que se abre, a palabra sage, tal como se mostra na seguinte imaxe



4. Escribir **notebook()** e pulsar **enter**. Isto iniciará o servidor de Sage.

NOTA: Esta instalación non está exenta de problemas, ademais dos xa mencionados na introdución, nunha proba realizada polo noso equipo en Windows 8 impedía lanzar o servidor, é por iso que ata que non exista unha versión 100% testada, compatible con varias versións de Windows, aconsellamos o uso de Sage a través dun sistema operativo virtualizado explicado na sección anterior.

2.5. Consideracións sobre seguridade nun servidor Sage

Tal como vimos no apartado “Instalando Sage en Ubuntu” podemos ter máis ou menos seguridade lanzando os nosos servidores se modificamos o noso script “1” engadindo “secure=True” na sentenza que inicia o servidor (notebook) de Sage. Isto fai que a comunicación entre calquera ordenador e o servidor realícese de forma encriptada a través de SSL. Isto evita que ordenadores mal intencionados poidan interceptar a comunicación e ler o contido do que se está enviado nun e outro sentido. Desta forma os ordenadores mal intencionados non poderían modificar esta información, ler os contrasinais e nomes de usuarios, etc. Certo é que este tipo de consideracións son moi importantes para calquera web que actualmente estea dispoñible de forma pública. No noso caso o servidor é restrinxido a aulas educativas con tan só un acceso local, iso impide que axentes externos poidan interceptar estas comunicacións. Certo é que os

alumnos máis avantaxados con coñecementos en Hackeo de ordenadores, que saiban capturar paquetes e realizar tarefas de suplantación poderían poñer en perigo as clases virtuais se a encriptación non está activa. A desvantaxe de activar esta encriptación é que o ordenador que actúa como servidor pasa a ter que encriptar todas as comunicacións que se establecen con el, o que causa un aumento no consumo de recursos da CPU que acaba por afectar ao rendemento de cálculo nas prácticas de laboratorio máis pesadas. Ademais, ao activar a encriptación SSL, os alumnos, no seu acceso á IP local do servidor do profesor deben aceptar o certificado auto firmado para poder entrar, isto causa confusión entre os alumnos menos experimentados. A nosa recomendación é que cada docente active ou desactive esta encriptación en función dos coñecementos que considera que posúe os seus alumnos, debemos recordar que en todo momento o obxectivo é ensinar, non enfocar todos os nosos esforzos a previr un improbable e pouco prexudicial ataque informático.

Outra consideración na seguridade sería a de utilizar por parte do usuario administrador e por parte dos alumnos de clase un contrasinal que inclúa maiúsculas, minúsculas, letras, números e carácteres especiais como puntos ou barra baixa. De novo isto faise de uso obrigado en servidores expostos, non é tan crítico en contornas moi controladas onde os alumnos deben realizar actividades académicas xa de por si nun tempo bastante axustado.

3.1. Worksheet 0

```
load (DATA + 'worksheet0.sage')
```

3.1.1. Xeneralidades

Para crear unha cela de texto como a que estás lendo debes clicar, pulsando Maiúsculas, sobre a raia azul que aparece nos espazos en branco, ao desprazar o punteiro do rato sobre eles. Para modificala, debes clicar dúas veces sobre calquera punto dela, aquí, ○, por exemplo. Cando acabes as modificacións, salva os cambios.

Para abrir unha cela de cálculo que nos permita facer operacións matemáticas, clica sobre a raia azul que está abaixo. Escribe nela números, signos aritméticos cos parénteses adecuados ou calquera expresión alfanumérica (sen acentos nin eñes) e evalúa.

```
print 41237*(7653-253)/23
print cos
print 5*cos(2*x+7)
```

```
305153800/23
cos
5*cos(2*x + 7)
```

Para saber como interpreta Sage un resultado R escribe type(R) ou parent(R) nunha cela de cálculo

```
R1=41237*(7653-253)/23
R2=cos
R3=5*cos(2*x + 7)

print type(R1)
print parent(R1)

print type(R2)
print parent(R2)

print type(R3)
print parent(R3)
```

```
<type 'sage.rings.rational.Rational'>
Rational Field
<class 'sage.functions.trig.Function_cos'>
<class 'sage.functions.trig.Function_cos'>
<type 'sage.symbolic.expression.Expression'>
Symbolic Ring
```

Para coñecer as ordes implementadas en Sage podemos preguntarllo nunha cela de cálculo

```
cos??
```

File: /home/sage/sage-6.10/local/lib/python2.7/site-packages/sage/functions/trig.py

Source Code (starting at line 85):

```
class Function_cos(GinacFunction):
    def __init__(self):
        """
        The cosine function.

        EXAMPLES::

            sage: cos(pi)
            -1
            sage: cos(x).subs(x==pi)
            -1
            sage: cos(2).n(100)
            -0.41614683654714238699756822950
            sage: loads(dumps(cos))
            cos

        We can prevent evaluation using the ``hold`` parameter::

            sage: cos(0,hold=True)
            cos(0)

        To then evaluate again, we currently must use Maxima via
        :meth:`sage.symbolic.expression.Expression.simplify`::

            sage: a = cos(0,hold=True); a.simplify()
            1

        TESTS::

            sage: conjugate(cos(x))
            cos(conjugate(x))
            sage: cos(complex(1,1))      # rel tol 1e-15
            (0.8337300251311491-0.9888977057628651j)

        """
        GinacFunction.__init__(self, "cos", latex_name=r"\cos",
                conversions=dict(maxima='cos',mathematica='Cos'))
```

Nós tamén podemos definir funcións propias e usalas en calquera punto da folla de traballo se as cargamos na primeira cela de cálculo, como fixemos con **load (DATA + 'worksheet0.sage')**. Podemos ver o que contén worksheet0.sage se clicamos na caixa Data (terceira superior esquerda) e seleccionamos esa opción. Para volver á folla de traballo clicamos en Worksheet.

Ás veces Sage pode simplificar algunha expresión simbólica E coa orde **E.simplify()**.

Para expresións simbólicas racionais ou trigonométricas pódense usar as ordes máis específicas **E.rational_simplify()** e **E.trig_simplify()** ou a orde **E.full_simplify()**.

```
E1(x)=x^2/x
print E1(x).simplify()
E2(x,y)= (x^(y+1)-x)/x
print E2(x,y).simplify()
print E2(x,y).rational_simplify()
print E2(x,y).full_simplify()
E3(x)=cos(x)^2+sin(x)^2
print E3(x).simplify()
print E3(x).trig_simplify()
print E3(x).full_simplify()
```

```
x
(x^(y + 1) - x)/x
x^y - 1
x^y - 1
cos(x)^2 + sin(x)^2
1
1
```

En Sage dispoñemos da función $hue : [0,1] \times [0,1] \times [0,1] \to \mathbb{R}^3$ que leva un punto (h,s,v) do mapa de color Munsell nunha terna (r,g,b) do sistema de cor RGB. Na seguinte cela de cálculo podemos cambiar os valores de h e ver as diferentes tonalidades

```
h=.6
circle((0,1),1,axes=False,fill=True,
rgbcolor=hue(h,1,1),figsize=[1,1])
```

3.1.2. Conxuntos

En Sage podemos expresar un conxunto finito escribindo, entre chaves, o nome entre aspas dos seus elementos.

```
S1={'Alberto','Bartolo','Carlos','Daniel'}
S2={'a','b','c','d'}
print type(S1), type(S2)
```

```
<type 'set'> <type 'set'>
```

Se os elementos son números ou variables predefinidas omítense as aspas.

```
S3={0,1,3,8,pi}
var('Maria Luisa Carmen Lucia')
S4={Maria, Luisa, Carmen, Lucia}
print type(S3)
print type(S4)
```

```
<type 'set'>
<type 'set'>
```

Non entende { } como o conxunto baleiro senón como o *dicionario* baleiro (ver apartado seguinte).

```
S5={}
type(S5)
```

```
<type 'dict'>
```

Tamén se pode escribir o nome dos elementos dun conxunto entre corchetes e parénteses precedidos da palabra set. Neste caso si que entende set([ ]) como o conxunto baleiro.

```
var('Alberto Bartolo Carlos Daniel a b c d')
S6=set([Alberto,Bartolo,Carlos,Daniel])
S7=set([a,b,c,d])
S8=set([])
print type(S6);print type(S7);print type(S8)
```

```
<type 'set'>
<type 'set'>
<type 'set'>
```

Aínda que deamos un conxunto entre chaves, Sage devólvenolo en formato set:

```
{1,2,3,4}
```

```
{1, 2, 3, 4}
```

Se queremos velo, de novo, entre chaves debemos empregar a función **llavea()** aínda que o resultado sexa de tipo string

```
S=set([1, 2, 3, 4]);SL=llavea(S)
print SL
type(SL)
```

```
{1, 2, 3, 4}
<type 'str'>
```

Sage pode obter a unión (|), intersección (&), diferenza (-) e produto cartesiano de conxuntos finitos

```
print S2|S3;print S3&S4;print S6-S7
print cartesian_product([S1,S2])
```

```
set(['a', 0, 'c', 'b', 'd', 1, 8, pi, 3])
set([])
set([Alberto, Bartolo, Carlos, Daniel])
The cartesian product of ({'Bartolo', 'Daniel', 'Alberto',
'Carlos'}, {'a', 'c', 'b', 'd'})
```

Sage recoñece os conxuntos numéricos infinitos **NN**, **Primes**, **ZZ**, **QQ**, **RR** e **CC**. Para saber como os entende basta introducir o nome seguido de dous signos de interrogación nunha cela de cálculo:

```
Primes??
```

File: /opt/sage-6.1.1-x86_64-Linux/local/lib/python2.7/site-packages/sage/sets/primes.py

Source Code (starting at line 29):

```
class Primes(Set_generic, UniqueRepresentation):
    """
    The set of prime numbers.

    EXAMPLES::

        sage: P = Primes(); P
        Set of all prime numbers: 2, 3, 5, 7, ...

    We show various operations on the set of prime numbers::

        sage: P.cardinality()
        +Infinity
        sage: R = Primes()
        sage: P == R
        True
        sage: 5 in P
        True
        sage: 100 in P
        False

        sage: len(P)          # note: this used to be a TypeError
Traceback (click to the left of this block for traceback)
...
```

```
1 in Primes()
```

```
    False
```

```
11 in Primes()
```

```
    True
```

```
111 in Primes()
```

```
    False
```

Sage tamén dispón da orde **factor**()

```
factor(24)
```
```
2^3 * 3
```

A orde **var**('x',domain=RR) créanos unha variable simbólica real $x \in RR$.

```
var('x',domain=RR)
factor(x^5-x)
```
```
(x^2 + 1)*(x + 1)*(x - 1)*x
```

Para a representación decimal dun número real r dispoñemos das ordes r.n() e **round**(r,n)

```
print pi
print pi.n()
print pi.n(digits=40)
print round(pi,4)
```
```
pi
3.14159265358979
3.141592653589793238462643383279502884197
3.1416
```

A orde **var**('z') crea unha variable simbólica complexa $z \in CC$. Sage calcula a súa parte real coa orde **real(z)**, a súa parte imaxinaria coa orde **imag(z)**, o seu módulo coa función **mod**(z) ou a orde **abs(z)** e a determinación do seu argumento en $(-\pi, \pi]$ coa orde **arg(z)**.

```
var('x y', domain=RR);z=x+y*I
print 'real(z)=',real(z)
print 'imag(z)=',imag(z)
print 'conjugate(z)=',conjugate(z)
print 'mod(z)=',mod(z)
print 'abs(z)=',abs(z)
print 'arg(z)=',arg(z)
```
```
real(z)= x
imag(z)= y
conjugate(z)= x - I*y
mod(z)= sqrt(x^2 + y^2)
abs(z)= abs(x + I*y)
arg(z)= arg(x + I*y)
```
```
z=CC.random_element()
print z
print 'real(z)=',real(z)
print 'imag(z)=',imag(z)
print 'conjugate(z)=',conjugate(z)
print 'mod(z)=',mod(z)
print 'abs(z)=',abs(z)
print 'arg(z)=',arg(z)
```

```
-0.512388015397364 + 0.941535186016151*I
real(z)= -0.512388015397364
imag(z)= 0.941535186016151
conjugate(z)= -0.512388015397364 - 0.941535186016151*I
mod(z)= 1.07192816215888
abs(z)= 1.07192816215888
arg(z)= 2.06917940203008
```

Definimos a función **roundc(**z,n**)** que redondea a n cifras decimais a parte real e a imaxinaria do complexo z porque a orde **round(**z,n**)** non funciona cando $z \in CC$

```
print 'roundc(z,5)=',roundc(z,5)
```

```
roundc(z,5)= -0.51239 + 0.94154*I
```

Tamén introducimos as seguintes funcións:

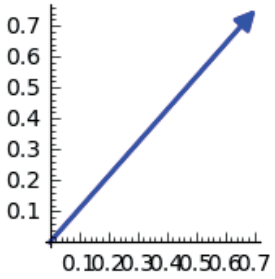
argumento(z**)** que dá a determinación do argumento en $(0, 2\pi]$

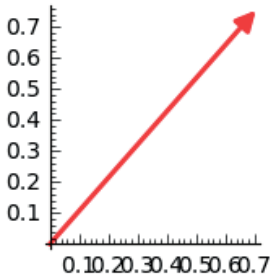
Argumento(r,z**)** que dá a determinación do argumento en $(r, r + 2\pi]$

pc(z,h**)** que debuxa o complexo z no plano de Argand en rgbcolor=hue(h,1,1) e, se soamente escribimos **pc(**z**)**, debúxao en rgbcolor=(1,0,0) .

```
z=1
print 'arg(z)=',arg(z)
print 'argumento(z)=',argumento(z)
print 'Argumento(-.9,z)=',Argumento(.9,z)
z=CC.random_element()
G1=show(pc(z,.7),figsize=[2,2]);G2=show(pc(z),figsize=[2,2])
```

```
arg(z)= 0
argumento(z)= 2*pi
Argumento(-.9,z)= 2*pi + 0.900000000000000
```

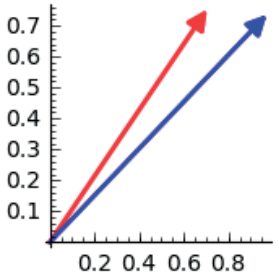




Para z,w en **CC** a función **angle(**z,w**)** dános o ángulo menor que π radiáns subtendido entre eles.

```
w=CC.random_element()
print angle(z,w)
(pc(z)+pc(w,.7)).show(figsize=[2,2])
```

```
0.170524220329213
```



3.1.3. Listas e Dicionarios

Sexa X un conxunto con boa ordenación e cardinal $|X| = n$ que chamaremos alfabeto e podemos identificar con $\{0, 1, \cdots, n-1\}$.

Unha lista L de lonxitude n nun conxunto C é a imaxe dunha aplicación $V : X \to C$ escrita entre corchetes, na orde correspondente á de X:

$$L = [V(0), V(1), , \ldots, V(n-1)]$$

A orde **len(**L**)** dá a lonxitude da lista L.

O primeiro elemento de L é L[0], o segundo L[1], o terceiro L[2],... , o penúltimo L[-2] e o último L[-1].

A orde L[i:j] devólvenos a sublista [L[i],L[i+1],...,L[j-1]].

A orde L[i:] devólvenos a sublista [L[i],L[i+1],...,L[-1]] e a orde L[:j], a sublista [L[0],L[1],...,L[j-1]].

Se z$\in$ CC a función **ri(z)** devolve a lista $[real(z), imag(z)]$.

A función **listasim(**'x',n**)** xera a lista de variables simbólicas [x0,...,xn-1]

```
L=[1,2,8,'a','b','c','d'];print len(L)
print L[2]
print L[-2]
print L[1:3]
print L[2:]
print L[:3]
z=2+3*I
print ri(z)
print listasim('x',10)
```

```
7
8
c
[2, 8]
[8, 'a', 'b', 'c', 'd']
[1, 2, 8]
[2, 3]
[x0, x1, x2, x3, x4, x5, x6, x7, x8, x9]
```

As funcións **S(**L**)**, **RS(**L**)**, **TS(**L**)** e **FS(**L**)** simplifican as expresións simbólicas da lista L.

```
var('x y')
FS([cos(x)^2+sin(x)^2,4*x+6*y+12*x-2*y])
```

```
[1, 16*x + 4*y]
```

A orde **range(**n**)** dános a lista de naturais [0,1,...,n-1].

A orde **range(**n_1, n_2**)** dános a lista de naturais $[n_1, n_1+1, \ldots, n_1+k]$ sendo k o maior natural tal que $n_1 + k < n_2$.

A orde **range(**n_1, n_2, p**)** dános a lista de naturais $[n_1, n_1+p, n_1+2p, \ldots, n_1+kp]$ sendo k o maior natural tal que $n_1 + kp < n_2$.

Se N é unha sublista de **range**(len(L)), a función **selenlist(**L,N**)** dános a sublista de L con índices en N.

```
print range(10)
print range(2,8)
print range(2,8,2)
N=[2,3];selenlist(L,N)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[2, 3, 4, 5, 6, 7]
[2, 4, 6]
[8, 'a']
```

A función **veces(**a,L**)** dinos cantas veces aparece o elemento a na lista L.

A orde L**.index(**a**)** dános o índice do primeiro a da lista L.

A función **indices(**a,L**)** dános todos os índices de a na lista L.

A función **siguientes(**a,L**)** devólvenos todos os seguintes de a na lista L.

A función **sig(**a,L**)** dános o seguinte do primeiro a da lista L.

A función **depuralista(**L**)** suprime da lista L os elementos repetidos.

```
L=[0,1,3,3,'a',5,'b','a','c'];print veces(6,L)
print L.index('a')
print indices('a',L)
print siguientes('c',L)
print sig('c',L)
print depuralista(L)
```

```
0
4
[4, 7]
[]
c
[0, 1, 3, 'a', 5, 'b', 'c']
```

A orde **reverso(**L**)** dános a reordenada de diante a atrás da lista L.

```
print L
reverso(L)
```

```
[0, 1, 3, 3, 'a', 5, 'b', 'a', 'c']
['c', 'a', 'b', 5, 'a', 3, 3, 1, 0]
```

Para saber se, os elementos dunha lista M están noutra lista L podemos usar a función **conliset(**M,L**)**.

Para saber se, conxuntistamente coinciden dúas listas L e M podemos usar a función **igualiset**(L,M).

```
M=[1,0,3,'a','b']
print conliset(M,L)
print igualiset(M,L)
```

```
True
False
```

A concatenación das listas L e M realízase coa orde L+M.

A supresión na lista L dos elementos da lista M, pódese conseguir coa función **restalistas(**L,M**).**

```
L=[3,7,8,9,'a','b',5]
M=[3,7,8,9]
print L+M
print restalistas(L,M)
```

```
[3, 7, 8, 9, 'a', 'b', 5, 3, 7, 8, 9]
['a', 'b', 5]
```

A orde L**.append(**a**)** xera a lista L+[a].

A función **Atila(**n**)** devólvenos o número natural de n cifras todas iguais a 1.

A función **zip(**L,M**)** constrúe a lista de tuplas [(L[0],M[0]),(L[1],M[1]),...(L[k],M[k])] ata que se acaba a lista máis curta.

A función **mezclalistas**(L,M) constrúe unha lista tomando alternativamente un elemento da de maior lonxitude e outro da de menor lonxitude. Cando se acaban os elementos da menor, engade os elementos que quedan da máis longa. Se ambas listas teñen a mesma lonxitude, toma alternativamente un elemento de L e outro de M.

```
L.append(12)
print L
print Atila(12)
print zip(L,M)
print mezclalistas(L,M)
```

```
[3, 7, 8, 9, 'a', 'b', 5, 12]
111111111111
[(3, 3), (7, 7), (8, 8), (9, 9)]
[3, 3, 7, 7, 8, 8, 9, 9, 'a', 'b', 5, 12]
```

Exemplo:

Canto tempo necesita Sage para comprobar que Atila(1031) é primo?

Solución:

Fixando un tempo inicial $T_0 =$**walltime**(), facemos os cálculos e expresamos o tempo transcorrido $T_1 =$**walltime**(T_0) desde o instante T_0.

```
T0=walltime()
print Atila(1031) in Primes()
T1=walltime(T0)
T1
```

```
True
1563.3019180297852
```

Exemplos:

1) Listar os números primos entre 50 e 100.

2) Listar os 4 primeros primos de Atila.

```
print '1)'
PP=[]
k=1
for n in range(50,100):
    if n in Primes():
        PP.append(n)
print PP
print
print '2)'
PA=[]
k=1
for n in range(1000):
    if k<5:
        if Atila(n) in Primes():
            PA.append(n)
        k=len(PA)
PA
```

```
1)
[53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

2)
[2, 19, 23, 317]
```

Sage pode listar os elementos dun produto cartesiano

```
A=set([3,'a','b'])
B=set(['d',2,5])
LAB=list(cartesian_product([A,B]))
print LAB[:5]
print LAB[5:]
```

```
[('a', 2), ('a', 'd'), ('a', 5), (3, 2), (3, 'd')]
[(3, 5), ('b', 2), ('b', 'd'), ('b', 5)]
```

Tamén pode listar as permutacións dunha lista L, con ou sen repeticións, mediante a orde **Permutations(**L**).list()**

```
L=[1,1,3,4]
PL=Permutations(L).list()
print PL[:4]
print PL[4:8]
print PL[8:]
```

```
[[1, 1, 3, 4], [1, 1, 4, 3], [1, 3, 1, 4], [1, 3, 4, 1]]
[[1, 4, 1, 3], [1, 4, 3, 1], [3, 1, 1, 4], [3, 1, 4, 1]]
[[3, 4, 1, 1], [4, 1, 1, 3], [4, 1, 3, 1], [4, 3, 1, 1]]
```

Tamén pode listar as combinacións de n elementos dunha lista L, con ou sen repeticións, mediante a orde **Combinations(**L,n**).list()**

```
Combinations(L,2).list()
```

```
[[1, 1], [1, 3], [1, 4], [3, 4]]
```

A imaxe da aplicación $V : X \to C$ tamén pode vir dada polo seu grafo ou lista de pares:

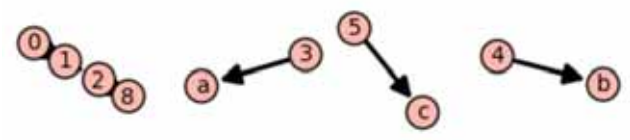
$$GV = [(a, V(a)) \; for \; a \; in \; X]$$

```
L=[1,2,8,'a','b','c']
X=range(len(L))
GV=zip(X,L);GV
```

```
[(0, 1), (1, 2), (2, 8), (3, 'a'), (4, 'b'), (5, 'c')]
```

A orde **DiGraph(**GV**)** dános unha representación gráfica da correspondencia

```
show(DiGraph(GV))
```



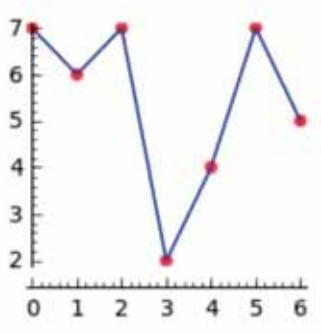
Tamén podemos entender a imaxe como a valoración das palabras dunha letra do alfabeto X no conxunto C e, mediante a función **g2dict(**GV**)**, obter un xenuíno dicionario D de Sage cuxa utilidade queda claro se observamos o funcionamento das ordes D.**keys()** e D.**values():**

```
D=g2dict(GV)
print type(D)
print D.keys()
print llavea(D.values())
```

```
<type 'dict'>
[0, 1, 2, 3, 4, 5]
[{1}, {2}, {8}, {'a'}, {'b'}, {'c'}]
```

Se C é un conxunto numérico, a orde **point(**GV**)** dános unha representación bidimensional dos puntos de GV e a orde **line(**GV**)** traza a poligonal que os une:

```
L=[7,6,7,2,4,7,5]
X=range(len(L))
GV=zip(X,L)
point(GV,hue=0,pointsize=30,figsize=[2.2,2.2])+line(GV)
```



A unha aplicación $k : X \times X \to C$ chamarémola núcleo en C. Se en $X \times X$ consideramos a orde produto da boa orde de X, podemos escribir a imaxe de k en forma dunha lista L de lonxitude n^2 e coa orde **matrix(**n,L**),** transformala nunha matriz $n \times n$ con entradas en C.

```
k(x,y)=2*x^2*y
C=listasim('a',130)
X=range(5)
L=[]
for i in X:
    for j in X:
        L.append(C[k(i,j)])
show(L)
M=matrix(5,L)
show(M)
```

$$[a_0, a_0, a_0, a_0, a_0, a_0, a_0, a_2, a_4, a_6, a_8, a_0, a_8, a_{16}, a_{24}, a_{32}, a_0, a_{18}, a_{36}, a_{54}, a_{72}, a_0, a_{32}, a_{64}, a_{96}, a_{128}]$$

$$\begin{pmatrix} a_0 & a_0 & a_0 & a_0 & a_0 \\ a_0 & a_2 & a_4 & a_6 & a_8 \\ a_0 & a_8 & a_{16} & a_{24} & a_{32} \\ a_0 & a_{18} & a_{36} & a_{54} & a_{72} \\ a_0 & a_{32} & a_{64} & a_{96} & a_{128} \end{pmatrix}$$

Tamén podemos describir a imaxe de k mediante o seu grafo ou lista de tripletas

$$Gk = [(a, b, k(a, b))\ for\ (a, b)\ in\ X \times X]$$

que podemos achar directamente:

```
Gk=[]
for i in X:
    for j in X:
        Gk.append((i,j,C[k(i,j)]))
```

```
print 'Gk='
print Gk[:4]
print Gk[4:8]
print Gk[8:12]
print Gk[12:16]
print Gk[16:20]
print Gk[20:24]
print Gk[24:]
```

```
Gk=
[(0, 0, a0), (0, 1, a0), (0, 2, a0), (0, 3, a0)]
[(0, 4, a0), (1, 0, a0), (1, 1, a2), (1, 2, a4)]
[(1, 3, a6), (1, 4, a8), (2, 0, a0), (2, 1, a8)]
[(2, 2, a16), (2, 3, a24), (2, 4, a32), (3, 0, a0)]
[(3, 1, a18), (3, 2, a36), (3, 3, a54), (3, 4, a72)]
[(4, 0, a0), (4, 1, a32), (4, 2, a64), (4, 3, a96)]
[(4, 4, a128)]
```

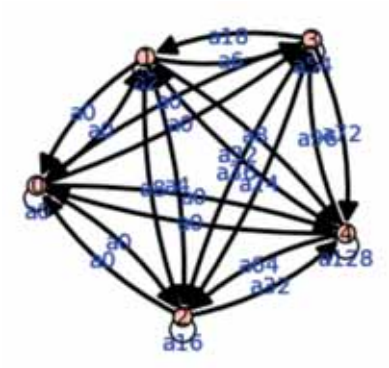
ou obtela a partir da súa expresión matricial mediante a función **matrixtok**(M):

```
MGk=matrixtok(M)
print
print 'MGk='
print MGk[:4]
print MGk[4:8]
print MGk[8:12]
print MGk[12:16]
print MGk[16:20]
print MGk[20:24]
print MGk[24:]
```

```
MGk=
[(0, 0, a0), (0, 1, a0), (0, 2, a0), (0, 3, a0)]
[(0, 4, a0), (1, 0, a0), (1, 1, a2), (1, 2, a4)]
[(1, 3, a6), (1, 4, a8), (2, 0, a0), (2, 1, a8)]
[(2, 2, a16), (2, 3, a24), (2, 4, a32), (3, 0, a0)]
[(3, 1, a18), (3, 2, a36), (3, 3, a54), (3, 4, a72)]
[(4, 0, a0), (4, 1, a32), (4, 2, a64), (4, 3, a96)]
[(4, 4, a128)]
```

A orde **DiGraph(**Gk,edge_lalels=True**)** dános unha representación bidimensional coas etiquetas dos arcos en C.

```
D=DiGraph(Gk)
D.graphplot(edge_labels=True).show(figsize=[2.5,2.5])
```



Podemos entender a imaxe de k como a valoración en C das palabras de dúas letras do alfabeto X e mediante a función **g3dict(**Gk**)** representala como un xenuíno dicionario D:

```
D=g3dict(Gk)
print 'escribímolo como un dicionario de dicionarios'
print D[0]
print D[1]
print D[2]
print D[3]
print D[4]
```

```
escribímolo como un dicionario de dicionarios
{0: a0, 1: a0, 2: a0, 3: a0, 4: a0}
{0: a0, 1: a2, 2: a4, 3: a6, 4: a8}
{0: a0, 1: a8, 2: a16, 3: a24, 4: a32}
{0: a0, 1: a18, 2: a36, 3: a54, 4: a72}
{0: a0, 1: a32, 2: a64, 3: a96, 4: a128}
```

Os núcleos mais interesantes son os que toman valores reais $k : X \times X \to \mathbb{R}$.

Entre eles destacamos o de Kronecker $\delta : X \times X \to \mathbb{R}$ tal que
$\delta(x,y) = \begin{cases} 1 & \text{se} \quad x = y \\ 0 & \text{se} \quad x \neq y \end{cases}$ e cuxa matriz é a identidade.

Para calquera núcleo $k : X \times X \to \mathbb{R}$ defínese o seu núcleo laplaciano
$\ell_k : X \times X \to \mathbb{R}$ tal que

$$\ell_k(x,y) = -k(x,y) + \delta(x,y) \sum_{z \in X} k(x,z)$$

que é de interese en problemas de redes eléctricas e hidráulicas. Se k vén representado pola lista L, a función **laplacian**(L) calcula a lista correspondiente ao núcleo ℓ_k.

Para os núcleos reais, ademais das representacións mencionadas antes, as ordes **point3d(**Gk**)** e **line3d(**Gk**)** dannos a súa representación gráfica tridimensional e a poligonal correspondente.

Exemplo:

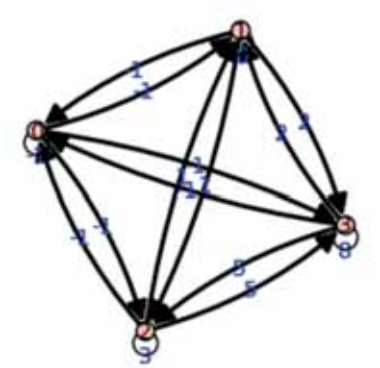
X=range(4), k(x ,y)=x·y -1

```
k(x,y)=x*y-1
L=[]
Gk=[]
for i in range(4):
    for j in range(4):
        L.append(k(i,j))
        Gk.append((i,j,k(i,j)))
print 'Gk='
print Gk[:4]
print Gk[4:8]
print Gk[8:12]
print Gk[12:]
print 'M='
show(matrix(4,L))
print 'Digrafo='
D=DiGraph(Gk)
D.graphplot(edge_labels=True).show(figsize=[2.5,2.5])
```

```
Gk=
[(0, 0, -1), (0, 1, -1), (0, 2, -1), (0, 3, -1)]
[(1, 0, -1), (1, 1, 0), (1, 2, 1), (1, 3, 2)]
[(2, 0, -1), (2, 1, 1), (2, 2, 3), (2, 3, 5)]
[(3, 0, -1), (3, 1, 2), (3, 2, 5), (3, 3, 8)]
M=
```

$$\begin{pmatrix} -1 & -1 & -1 & -1 \\ -1 & 0 & 1 & 2 \\ -1 & 1 & 3 & 5 \\ -1 & 2 & 5 & 8 \end{pmatrix}$$

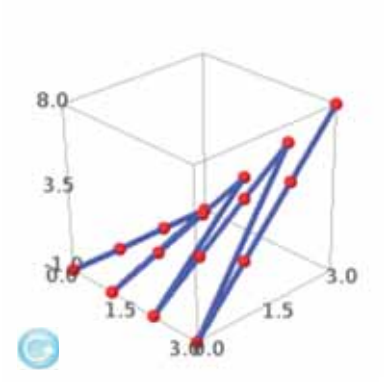
Digrafo=



```
print 'Representación 3D='
show(point3d(Gk,color='red',size=16,figsize=
```

```
[2.5,2.5,2.5])+line3d(Gk,thickness=7))
```

Representación 3D=



Se nun núcleo real $k : X \times X \to \mathbb{R}$, as palabras de dúas letras con valor nulo carecen de significado ou importancia, podemos suprimilas e representar unicamente a imaxe do soporte de k co conseguinte aforro de memoria.

Exemplo:

X=range(4), k(x,y)=x·y

```
k(x,y)=x*y
L=[]
Sk=[]
for i in range(4):
    for j in range(4):
        L.append(k(i,j))
        if k(i,j)!=0:
            Sk.append((i,j,k(i,j)))
print 'Sk=',Sk[:4]
print Sk[4:]
print

print 'M='
show(matrix(4,L))
D=g3dict(Sk)
print 'D='
print D[1]
print D[2]
print D[3]
G=DiGraph(Sk)
G.graphplot(edge_labels=True).show(figsize=[2.2,2.2])
```

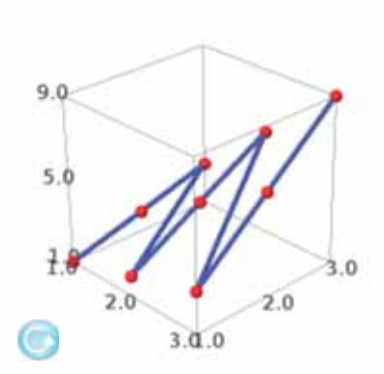
```
Sk= [(1, 1, 1), (1, 2, 2), (1, 3, 3), (2, 1, 2)]
[(2, 2, 4), (2, 3, 6), (3, 1, 3), (3, 2, 6), (3, 3, 9)]

M=
```

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 4 & 6 \\ 0 & 3 & 6 & 9 \end{pmatrix}$$

```
D=
{1: 1, 2: 2, 3: 3}
{1: 2, 2: 4, 3: 6}
{1: 3, 2: 6, 3: 9}
```

```
print 'Representación 3D='
show(point3d(Sk,color='red',size=16,figsize=
[2.5,2.5,2.5])+line3d(Sk,thickness=6))
```

```
Representación 3D=
```



Dar un núcleo real $k : X \times X \to \mathbb{R}$ con $k(i,j) \in \{0,1\}$ é equivalente a dar unha relación en X ou fixar un subconxunto $P \subset X \times X$:

$$i \sim j \quad \Leftrightarrow \quad (i,j) \in P \quad \Leftrightarrow \quad k(i,j) = 1$$

Nun alfabeto X de cardinal n podemos xerar:

1. A relación identidade Δ coa función **rid(**n**)**.

2. Unha relación aleatoria P, coa función **relacion(**n**)**.

3. A relación trasposta P^t coa función **rtranspose(**P**)**.

As representacións dunha relación poden ser diversas:

```
print rid(7)
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6)]
```

```
P=relacion(5)
html("<h4>Representación en forma de lista</h4>")
print 'P=';print P[:6];print P[6:]
html("<h4>Representación en forma de matriz</h4>")
A=matrix(5)
for i in range(5):
    for j in range(5):
        if (i,j) in P:
            A[i,j]=1
show(A)
html("<h4>Representación en forma de dicionario</h4>")
D=g2dict(P);show(llavea(D))
html("<h4>Representación bidimensional</h4>")
G=DiGraph(D)
G.graphplot().show(figsize=[2.2,2.2])
```

Representación en forma de lista

```
P=
[(0, 0), (0, 2), (0, 3), (0, 4), (1, 1), (2, 3)]
[(3, 1), (3, 3), (4, 2)]
```

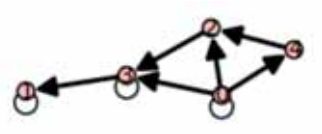
Representación en forma de matriz

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Representación en forma de dicionario

```
{0: {0, 2, 3, 4}, 1: {1}, 2: {3}, 3: {1, 3}, 4: {2}}
```

Representación bidimensional



```
Pt=rtranspose(P)
html("<h4>Representación en forma de lista</h4>")
print 'Pt=',print Pt[:6]
print Pt[6:]
html("<h4>Representación en forma de matriz</h4>")
A=matrix(5)
for i in range(5):
    for j in range(5):
        if (i,j) in Pt:
            A[i,j]=1
show(A)
html("<h4>Representación bidimensional</h4>")
G=DiGraph(Pt);G.graphplot().show(figsize=[2.2,2.2])
```

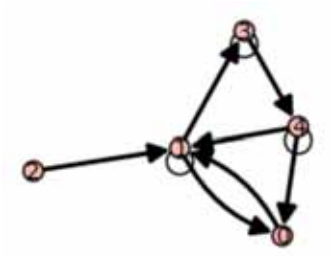
Representación en forma de lista

```
Pt=
[(1, 0), (4, 0), (0, 1), (1, 1), (2, 1), (4, 1)]
[(1, 3), (3, 3), (3, 4), (4, 4)]
```

Representación en forma de matriz

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Representación bidimensional



```
html("<h4>Representación en forma de dicionario</h4>")
D=g2dict(Pt);show(llavea(D))
```

Representación en forma de dicionario

```
{0: {1}, 1: {0, 1, 3}, 2: {1}, 3: {3, 4}, 4: {0, 1, 4}}
```

Sexan P e Q dúas relacións nun alfabeto X de cardinal n. A función **comprel**(P,Q) calcúlanos a súa composición

$$P \circ Q = [(a,b) \in X \times X \,|\, \exists c \in X \text{ tal que } (a,c) \in Q \text{ e } (c,b) \in P]$$

e a función **rct**(P) calcúlanos a clausura transitiva de P que definimos por recorrencia, como segue:

$$P_1 = P, \quad P_2 = P_1 \circ P_1, \ldots, P_{k+1} = P_k \circ P_1 \quad \text{e} \quad rtc(P) = \bigcup_{k=1}^{n} P_k \quad \text{con} \quad n \in \mathbb{N}$$

```
P=relacion(4)
print 'P=',P[:6]
print P[6:]
print
Q=relacion(4)
print 'Q=',Q
print
PoQ=comprel(P,Q)
print 'PoQ=',PoQ[:5]
print PoQ[5:]
RCT=sorted(rct(P))
print 'rct(P)=',RCT[:4]
print RCT[4:9]
print RCT[9:]
```

```
P= [(0, 2), (1, 2), (1, 3), (2, 0), (2, 2)]
[]

Q= [(0, 2), (2, 0), (2, 1), (3, 1), (3, 2)]
```

```
PoQ= [(0, 0), (0, 2), (2, 2), (2, 3), (3, 0)]
[(3, 2), (3, 3)]

rct(P)= [(0, 0), (0, 2), (1, 0), (1, 2)]
[(1, 3), (2, 0), (2, 2)]
[]
```

Sabemos que unha relación P é:

1. Reflexiva se $\Delta \subset P$.

2. Simétrica se $P = P^t$.

3. Antisimétrica se $P \cap P^t = \Delta$.

4. Transitiva se $P \circ P \subset P$.

As funcións **rreflex(**P), **rsim(**P), **rantisim(**P) y **rtrans(**P) dinnos se a relación P verifica estas propiedades

```
P=relacion(3)
print 'P=',P

print 'reflexiva?',rreflex(P)
print 'simétrica?',rsim(P)
print 'antisimétrica?',rantisim(P)
print 'transitiva?',rtrans(P)

print 'clausura_transitiva(P)='
print sorted(rct(P))
print 'transitiva?',rtrans(rct(P))
```

```
P= [(0, 0), (0, 1), (1, 0), (1, 1)]
reflexiva? True
simétrica? True
antisimétrica? False
transitiva? True
clausura_transitiva(P)=
[(0, 0), (0, 1), (1, 0), (1, 1)]
transitiva? True
```

Sexa X un alfabeto de cardinal n. A imaxe dunha aplicación $D : X \times \overset{m}{\cdots} \times X \to C$ pódese interpretar como un dicionario de palabras de m letras do alfabeto X. Para $m \geq 3$ soamente utilizaremos a representación asociada á función **gdict(**GD**)**.

Por exemplo, para $n = 5$, $m = 3$, $C = listasim('a', 65)$, $D(x, y, z) = C[x * y * z]$ obteremos

```
D(x,y,z)=x*y*z
C=listasim('a',65)
```

```
GD=[]
for i in range(5):
    for j in range(5):
        for k in range(5):
            GD.append((i,j,k,C[D(i,j,k)]))
DIC=gdict(GD)
show(llavea(DIC))
```

{0:{0:{4:{a0}},1:{4:{a0}},2:{4:{a0}},3:{4:{a0}},4:{4:{a0}}},

1:{0:{4:{a0}},1:{4:{a4}},2:{4:{a8}},3:{4:{a12}},4:{4:{a16}}},

2:{0:{4:{a0}},1:{4:{a8}},2:{4:{a16}},3:{4:{a24}},4:{4:{a32}}},

3:{0:{4:{a0}},1:{4:{a12}},2:{4:{a24}},3:{4:{a36}},4:{4:{a48}}},

4:{0:{4:{a0}},1:{4:{a16}},2:{4:{a32}},3:{4:{a48}},4:{4:{a64}}}}

Para n=5, m=4, C=RR, $D(x,y,z,t) = 4x^3y + 2xyz - 5xyzt^2$, cinguíndonos ao soporte, obtemos

```
G(x,y,z,t)=4*x^3*y+2*x*y*z-5*x*y*z*t^2
L=[]
for a in range(5):
    for b in range(5):
        for c in range(5):
            for d in range(5):
                if G(a,b,c,d)!=0:
                    L.append((a,b,c,d,G(a,b,c,d)))
DIC=gdict(L)
show(llavea(DIC))
```

{1:{1:{4:{4:-308}},2:{4:{4:-616}},3:{4:{4:-924}},4:{4:{4:-1232}}},

2:{1:{4:{4:-592}},2:{4:{4:-1184}},3:{4:{4:-1776}},4:{4:{4:-2368}}},

3:{1:{4:{4:-828}},2:{4:{4:-1656}},3:{4:{4:-2484}},4:{4:{4:-3312}}},

4:{1:{4:{4:-992}},2:{4:{4:-1984}},3:{4:{4:-2976}},4:{4:{4:-3968}}}}

3.1.4. Digrafos, grafos e multidigrafos

Un digrafo D é unha terna (V, E, w) onde V é un alfabeto, E é una relación en V e $w : V \times V \to \mathbb{R}$ é un núcleo de soporte E. Designando $\top : V \times V \to V \times V$ á trasposición canónica diremos que o digrafo é simple se $\Delta \cap E = \emptyset$, que é unidireccional si $E \cap \top(E) = \emptyset$ e que é conexo se $\Delta \cup T = V \times V$ sendo T a clausura transitiva de E.

As funcións **simple**(w), **unidir**(w) e **conexo**(w) dinnos se o digrafo definido polo núcleo w é ou non da condición indicada.

Toda a información sobre o digrafo D está contida na lista que define o núcleo w, pero na definición clásica prefírese que aparezan o conxunto de vértices V e o conxunto de arcos E.

```
w=[[0,1,12],[0,2,11],[1,2,3],[2,3,6],[4,1,5]]
V=vertices(w)
E=[(a[0],a[1]) for a in w]
print 'É simple?', simple(w)
print 'É unidireccional?', unidir(w)
print 'É conexo?', conexo(w)
```

```
É simple? True
É unidireccional? True
É conexo? False
```

Dado un digrafo $D = (V, E, w)$ definimos:

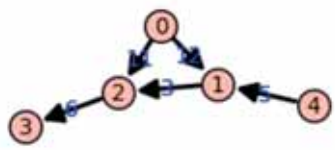
O seu trasposto $D^t = (V, \top(E), w^t)$ onde $w^t = w \circ \top$

O seu simetrizado $D_s = (V, E \cup \top(E), w_s)$ sendo $w_s = \frac{w+w^t}{2}$.

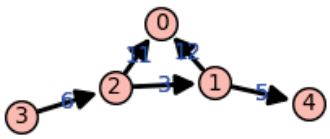
E se D é unidireccional, o seu bidireccional $G = (V, E \cup \top(E), w + w^t)$.

As funcións **trans**(w), **sim**(w) e **bidir**(w) constrúen estes digrafos asociados ao núcleo w.

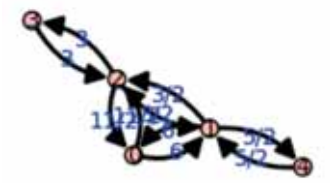
```
G1=DiGraph(w)
G1.graphplot(edge_labels=True).show(figsize=[2.2,2.2])
```



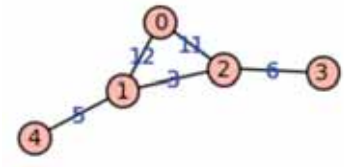

```
G2=DiGraph(trans(w))
G2.graphplot(edge_labels=True).show(figsize=[2.2,2.2])
```




```
G3=DiGraph(sim(w))
G3.graphplot(edge_labels=True).show(figsize=[2.2,2.2])
```



```
G4=Graph(w)
G4.graphplot(edge_labels=True).show(figsize=[2.2,2.2])
```



Un digrafo D tal que $D = D_s$ chamóuselle clasicamente grafo. Se (u, v) é un arco, (v, u) tamén o será e co mesmo peso. En E podemos considerar a relación de equivalencia (u, v)~(v, u) que substitúe os dous arcos opostos (u, v) e (v, u) pola arista $\{u, v\}$ e ao cociente $A = E/$~ chámaselle conxunto de aristas $\{u, v\}$ do grafo $G = (V, A, w)$.

Moitas situacións da vida real poden ser modelizadas sobre digrafos e grafos. Os vértices poden ser cidades, os arcos autoestradas, as aristas vías de dobre sentido e os pesos os correspondentes peaxes ou calidades da viaxe; ou os vértices poden ser nodos nun circuíto eléctrico, as aristas os condutores e os pesos as correspondentes resistencias; ou os vértices poden ser os nodos dunha rede hidráulica, os arcos tuberías de auga e os pesos, fluxos ou caudais.

Dúas cidades poden estar conectadas por máis dunha autoestrada, dous nodos eléctricos poden estar conectados por máis dun condutor ou dous nodos hidráulicos poden estar conectados por máis dun tubo. Estas situacións non poden ser representadas por un núcleo pero poden ser arquivados nunha lista que teña diferentes tripletas coas mesmas dúas primeiras coordenadas e correspóndelle unha gráfica que denominamos multidigrafo porque entre dous vértices pode haber arcos paralelos de igual ou diferente peso.

Aínda que se pode tratar a teoría de multidigrafos directamente, para a maioría dos nosos intereses logramos formular un problema equivalente , tratable en termos de digrafos, reemplazando os conxuntos de arcos paralelos por un só arco cun un peso adecuado. Por exemplo, se temos diferentes autoestradas entre dúas cidades, collemos

a máis barata ou a que teña mellores calidades para a viaxe. Se tenemos varios tubos entre dous nodos hidráulicos, consideramos un tubo cun fluxo igual á suma dos fluxos. Se temos m condutores entre dous nodos eléctricos con resistencias $r_1, r_2, \cdots, r_m$, consideramos unicamente un condutor con resistencia $r = (\frac{1}{r_1} + \cdots + \frac{1}{r_m})^{-1}$.

Podemos facer todo isto ca función **deplist**(L,c) onde c é o critero de reunificación de arcos paralelos que pode ser tomar o máximo dos seus pesos (M), ou o mínimo (m), ou a suma (S) ou a resistencia (R) pero o usuario pode engadir outros criterios se o considera necesario.

A utilización de pesos adecuados para os arcos dun digrafo ou as aristas dun grafo permítenos tratar desde un punto de vista formal as teorías de grafos e multidigrafos dentro da teoría de digrafos.

3.1.5. Listas reais

Para construír unha lista aleatoria en NN procedemos como segue:

```
L=[]
for n in range(15):
    L.append(abs(ZZ.random_element()))
print L
```

```
    [1, 0, 6, 4, 1, 1, 1, 8, 2, 0, 8, 0, 0, 0, 0]
```

En RR podemos fabricala como segue

```
R=[]
for n in range(15):
    R.append(RR.random_element())
R
```

```
    [0.858923811094704,
     0.221860174343157,
     0.947988369672898,
     -0.282423620836550,
     0.133926487890108,
     0.640727726885428,
     0.988965176119890,
     -0.859935337572387,
     -0.606895363140605,
     0.908609604495559,
     -0.284793852887377,
     0.398016777872859,
     0.741318108797200,
     0.143728156643913,
     -0.356716146145913]
```

e redondear a k decimais coa orde **Round(**R,k)

```
LR=Round(R,1); show(LR)
```

$$[0.9, 0.2, 0.9, -0.3, 0.1, 0.6, 1.0, -0.9, -0.6, 0.9, -0.3, 0.4, 0.7, 0.1, -0.4]$$

En CC podemos fabricala como segue

```
C=[]
for n in range(15):
    C.append(CC.random_element())
C
```

```
[0.110957131022016 + 0.113647609547805*I,
 0.161995148448700 + 0.998437815476345*I,
 0.102979810817442 + 0.261817105442985*I,
 0.234796225849073 - 0.944119251380028*I,
 -0.439315337674340 - 0.0443227327628162*I,
 0.438007051471576 - 0.537162499827203*I,
 -0.0553369386391618 - 0.870630642433996*I,
 0.726836561251005 - 0.508560205984939*I,
 0.370490107996605 + 0.770430162812737*I,
 0.0229893863590009 + 0.679537772422878*I,
 -0.0260167683985322 + 0.527424975645744*I,
 0.693151454389015 + 0.581802480063583*I,
 -0.712020232740675 + 0.948841129569493*I,
 0.433587928850306 + 0.436776532727521*I,
 0.595426593841810 - 0.0811120308867825*I]
```

e redondear a k decimais coa orde **Roundc(**C,k**)**

```
LC=Roundc(C,2);LC
```

```
[0.11 + 0.11*I,
 0.16 + 1.0*I,
 0.1 + 0.26*I,
 0.23 - 0.94*I,
 -0.44 - 0.04*I,
 0.44 - 0.54*I,
 -0.06 - 0.87*I,
 0.73 - 0.51*I,
 0.37 + 0.77*I,
 0.02 + 0.68*I,
 -0.03 + 0.53*I,
 0.69 + 0.58*I,
 -0.71 + 0.95*I,
 0.43 + 0.44*I,
 0.6 - 0.08*I]
```

e construír as listas das súas partes reais e imaxinarias coas funcións **Real(**C**)** e **Imag(**C**)**.

```
print Real(LC)
print Imag(LC)
```

```
[0.11, 0.16, 0.1, 0.23, -0.44, 0.44, -0.06, 0.73, 0.37, 0.02,
0.69, -0.71, 0.43, 0.6]
[0.11, 1.0, 0.26, -0.94, -0.04, -0.54, -0.87, -0.51, 0.77, 0.
0.53, 0.58, 0.95, 0.44, -0.08]
```

As ordes **sum(**L**)** e **prod(**L**)** dannos a suma e o produto de todos os elementos da lista numérica L.

Para modificar a lista **L** sumando, restando, multiplicando ou dividindo cada un dos seus elementos por un z dado, usamos a función **oplista(**L,o,z**)** onde o pode ser s (suma), r (resta), m (multiplicación) ou d (división).

```
print sum(LR)
print prod(LR)
print sum(LC)
print prod(LC)
print
print Roundc(oplista(LC,m,1+I),3)
```

```
3.3
-4.76171136e-06
2.6400000000000006 + 2.3400000000000003*I
-0.0010226719764006114 - 0.0007129372099000458*I

[0.22*I, -0.84 + 1.16*I, -0.16 + 0.36*I, 1.17 - 0.71*I, -0.4 +
0.48*I, 0.98 - 0.1*I, 0.81 - 0.93*I, 1.24 + 0.22*I, -0.4 + 1.
-0.66 + 0.7*I, -0.56 + 0.5*I, 0.11 + 1.27*I, -1.66 + 0.24*I, -
0.87*I, 0.68 + 0.52*I]
```

Dados tres números reais a, b, h tales que $a < b$ e $h < b - a$, a orde $[a, a + h..b]$ dános a lista L de números reais tal que $L[k] = a + kh$ sempre que $a + kh \leq b$.

```
A=[pi,pi+1..20]
print A
print
print num(A)
print
print Round(num(A),2)
```

```
[pi, pi + 1, pi + 2, pi + 3, pi + 4, pi + 5, pi + 6, pi + 7, p
pi + 9, pi + 10, pi + 11, pi + 12, pi + 13, pi + 14, pi + 15,
16]

[3.14159265358979, 4.14159265358979, 5.14159265358979,
6.14159265358979, 7.14159265358979, 8.14159265358979,
9.14159265358979, 10.1415926535898, 11.1415926535898,
12.1415926535898, 13.1415926535898, 14.1415926535898,
15.1415926535898, 16.1415926535898, 17.1415926535898,
18.1415926535898, 19.1415926535898]

[3.14, 4.14, 5.14, 6.14, 7.14, 8.14, 9.14, 10.14, 11.14, 12.1
```

```
13.14, 14.14, 15.14, 16.14, 17.14, 18.14, 19.14]
```

A función **listareales(**m,M,n**)** devólvenos unha lista aleatoria de n números reais no intervalo [m,M].

```
L=listareales(3,2*pi,15)
NL=num(L)
print Roundc(NL,3)
```

```
[6.237, 5.28, 6.142, 3.561, 3.088, 3.777, 3.524, 5.604, 3.447,
5.822, 4.228, 3.528, 4.071, 5.045, 4.771]
```

Dada unha lista L de reais, as ordes **min(**L**)** e **max(**L**)** dannos o mínimo e o máximo e as funcións **minimos(**L**)** e **maximos(**L**)** engádennos os índices en que se alcanzan.

```
print min(NL)
print max(NL)
print minimos(NL)
print maximos(NL)
```

```
3.08800053972573
6.23672166620490
(3.08800053972573, [4])
(6.23672166620490, [0])
```

Dada unha función $f:\mathbb{R}\to\mathbb{R}$ a orde **map(**f,L**)** dános a lista $f\circ L$ e as funcións **MIN(**f,L**)** e **MAX(**f,L**)** dannos, respectivamente os **minimos(**$f\circ L$**)** e **maximos(**$f\circ L$**)**.

```
f(x)=sin(x)
print Roundc(map(f,NL),3)
print
print MIN(f,NL)
print
print MAX(f,NL)
```

```
[-0.046, -0.843, -0.141, -0.407, 0.054, -0.593, -0.373, -0.62
-0.301, -0.445, -0.885, -0.377, -0.801, -0.945, -0.998]

(-0.998300488197310, [14])

(0.0535664637651222, [4])
```

A orde L**.sort()** converte a lista L de números reais na súa ordenada ascendente. Para preservar a lista L podemos usar a orde **sorted(**L**).**

```
M=copy(NL)
M.sort()
print M
print sorted(NL)
```

```
[3.23329416502709, 3.25014301234386, 3.31223772212883,
3.52491697680188, 3.54217826417719, 3.68225604396834,
3.69509512080487, 4.24009559372445, 4.27361230118604,
```

```
4.50676135066974, 4.54328373850604, 4.89701938211608,
5.00634221285981, 5.93933624846053, 6.21439238287031]

[3.23329416502709, 3.25014301234386, 3.31223772212883,
3.52491697680188, 3.54217826417719, 3.68225604396834,
3.69509512080487, 4.24009559372445, 4.27361230118604,
4.50676135066974, 4.54328373850604, 4.89701938211608,
5.00634221285981, 5.93933624846053, 6.21439238287031]
```

3.1.5.1. Sucesións recurrentes

A sucesión de Fibonacci é a sucesión recurrente máis famosa da Historia. Defínese como segue:

$$x_0 = 1, \quad x_1 = 1 \quad , \ldots , \quad x_{n+2} = x_{n+1} + x_n$$

A función **Fibonacci**(n) dános o término x_n

Exemplo:

a) Calcular x_{15} e x_{90}.

b) Canto tempo lle leva a Sage achar a lista dos 50 primeiros términos da sucesión de Fibonacci?

c) Comprobar que $\lim_{n\to\infty} \frac{x_{n+1}}{x_n} = \phi$, o número áureo.

```
print 'a)'
print 'Fibonacci(15)=',Fibonacci(15),','
'Fibonacci(90)=',Fibonacci(90)
print 'b)'
T0=walltime()
F=[Fibonacci(n) for n in range(50)]
T1=walltime(T0)
print T1
print 'c)'
F=[Fibonacci(n) for n in range(100)]
A=[(sig(a,F)/a).n() for a in F]
print Roundc(A[90:-1],2)
```

```
a)
Fibonacci(15)= 987 ,Fibonacci(90)= 4660046610375530309
b)
0.00388097763062
c)
[1.62, 1.62, 1.62, 1.62, 1.62, 1.62, 1.62, 1.62, 1.62]
```

Unha sucesión de números reais $(x_n) = x_0, x_1, x_2, x_3, x_4, \ldots$ xera:

A sucesión de primeiras diferenzas

$$(d_n^1) = x_1 - x_0, x_2 - x_1, x_3 - x_2, x_4 - x_3, \ldots$$

A sucesión de segundas diferenzas

$$(d_n^2) = d_1^1 - d_0^1, d_2^1 - d_1^1, d_3^1 - d_2^1, d_4^1 - d_3^1, \ldots$$

A sucesión de terceiras diferenzas

$$(d_n^3) = d_1^2 - d_0^2, d_2^2 - d_1^2, d_3^2 - d_2^2, d_4^2 - d_3^2, \ldots$$

En xeral, unha sucesión de k-ésimas diferenzas

$$(d_n^k) = d_1^{k-1} - d_0^{k-1}, d_2^{k-1} - d_1^{k-1}, d_3^{k-1} - d_2^{k-1}, d_4^{k-1} - d_3^{k-1}, \ldots$$

Cando a sucesión de primeiras diferenzas é a constante non nula (r), (x_n) é aritmética de diferenza r.

Cando a sucesión de segundas diferenzas é a constante non nula (r), (x_n) é aritmética de orde 2 e diferenza r.

Cando a sucesión de k-ésimas diferenzas é a constante non nula (r), (x_n) é aritmética de orde k e diferenza r.

Exemplos:

1) Sexa o polinomio $P(x) = x^2 + 2x + 5$. Probar que a sucesión (P(n)) é aritmética de orde 2 e diferenza 2.

```
P(x)=x^2+2*x+5; S=[P(n) for n in range(20)]
D1=[sig(a,S)-a for a in S]; D2=[sig(a,D1)-a for a in D1]
D2[:-2]
```

```
[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]
```

2) Sexa o polinomio $Q(x) = 3x^5 - 4x + 10$. Probar que a sucesión (Q(n)) é aritmética de orde 5 e diferenza 360.

```
Q(x)=3*x^5-4*x+10; S=[Q(n) for n in range(20)]
D1=[sig(a,S)-a for a in S]
D2=[sig(a,D1)-a for a in D1]
D3=[sig(a,D2)-a for a in D2]
D4=[sig(a,D3)-a for a in D3]
D5=[sig(a,D4)-a for a in D4]
D5[:-5]
```

```
[360, 360, 360, 360, 360, 360, 360, 360, 360, 360, 360, 360,
360, 360]
```

3) Sexa $P(x)$ un polinomio de grado k cuxo monomio de maior grado é ax^k. Probar

que $(P(n))$ é unha sucesión aritmética de orde k e diferenza $ak!$.

Solución:

É claro que $D^1(x) = P(x+1) - P(x) = \sum_{k=1}^{5} \frac{1}{k!} P^{k)}(x)$ é un polinomio de grado $k-1$. De igual modo, $D^2(x) = D^1(x+1) - D^1(x)$ é un polinomio de grado $k-2$ e, inductivamente, $D^k(x) = D^{k-1}(x+1) - D^{k-1}(x)$ é un polinomio de grado 0, é dicir, unha constante.

A sucesión de k-ésimas diferenzas de $(P(n))$ é a sucesión constante $(D^k(n))$. É dicir, $(P(n))$ é aritmética de orde k. É fácil ver que a súa diferenza é $ak!$.

4) Cantas sucesións aritméticas de orde 3 e diferenza 7 hai?

Solución:

Cada unha delas debe cumprir a relación de recurrencia:

$$X[n+3] - 3X[n+2] + 3X[n+1] - X[n] = 7.$$

Calquera tripleta inicial $[x_0, x_1, x_2]$ pode ser estendida de modo único a unha sucesión aritmética de orde 3 e diferenza 7.

Por exemplo, podemos achar o término 20 da estendida de $[3, 2, 7]$

```
X=[3,2,7]
for k in range(20):
        X.append(X[-3]-3*X[-2]+3*X[-1]+7)
show(X[20])
```

9103

3.1.6. Listas complexas

A función **listacomplejos(**T,n**)** devólvenos unha lista aleatoria de n números complexos no cadrado $[-T, T] \times [-T, T]$.

As funcións **plse(**L**)** e **plce(**L**)** pintan en vermello no plano de Argand, con ou sen eixos, os puntos correspondentes aos elementos de L. Se escribimos **plse(**L,h**)** e **plce(**L,h**)** píntannolos en ton h.

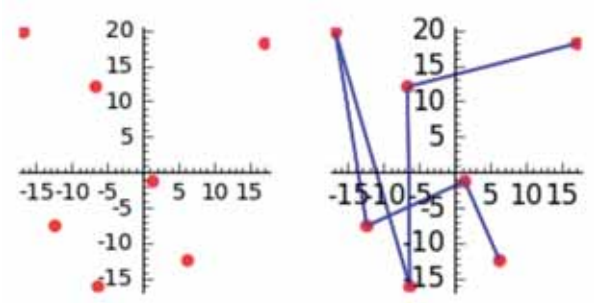
A función **poligonal(**L,h,k**)** representa a poligonal determinada por L pintando en ton h as aristas e en ton k os vértices.

A función **lon(**L**)** devólvenos a lonxitude da poligonal de L e a lonxitude da arista maior.

A función **envolturaconvexa**(L) dános os vértices expostos da envoltura convexa de L,

ordenados en sentido antihorario. Debemos ter en conta que a lista L se modifica ao executar esta función.

```
L=listacomplejos(20,7)
CL=copy(L)
P=poligonal(L,.7)
PL=plce(L)
C=envolturaconvexa(CL)
show(graphics_array([PL,P]),figsize=[4,2])
```



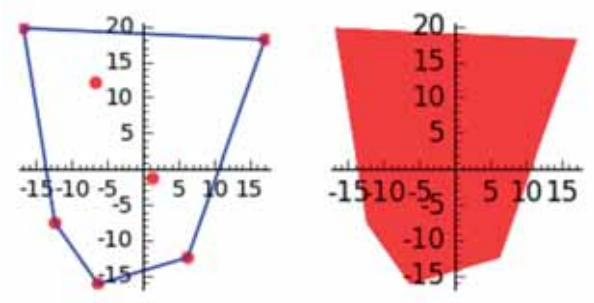
```
lon(L)
```

```
(144.945019384742, 37.3378682790149)
```

A orde de Sage **polygon2d(**C**)** pinta o polígono de vértices da lista C, reencho ou non segundo que a opción fill sexa True ou False.

```
CR=polygon2d(C, fill=True, color='red')
CSR=polygon2d(C, fill=False)
print 'lon(C)[0]=',lon(C)[0]
show(graphics_array([CSR+PL,CR]),figsize=[4,2])
```

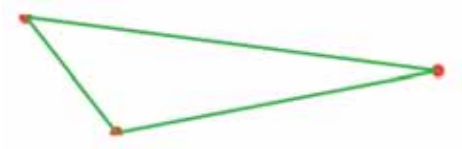
```
lon(C)[0]= 117.631506064754
```



Cando **len**(L)=3 a función **triangle**(L) dános os lados e os ángulos do triángulo cuxos vértices están en L. A función **heron**(L) dános a área do triángulo e a función **pt**(L,h,k) píntanos o triángulo cos lados en tonalidade hue=h e os vértices en hue=k.

```
T=listacomplejos(5,3)
print 'lados=',triangle(T)[0]
print 'ángulos=',triangle(T)[1]
print 'área=', heron(T)
show(pt(T,.3,1),figsize=[3,3])
```

```
lados= [4.44122039083169, 5.63931450972944, 1.99077226720016]
ángulos= [0.775026834939715, 2.04752537374692, 0.319040444903
área= 3.92782068679971
```



Exemplo:

Se Z=listacomplejos(20,10)

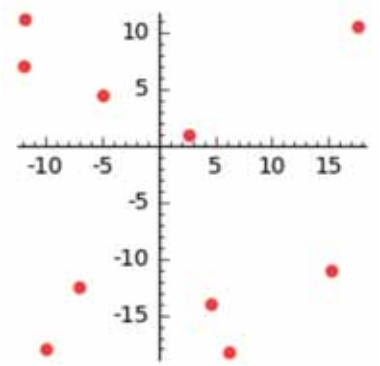
a) Cantos elementos de Z cumpren que $-5 \leq \Re(z) \leq 7$?

b) Achar o elemento de Z de módulo máximo.

c) Achar un $z_0 \in Z$ tal que $arg(z_0) = \min\{arg(z) \mid z \in Z\}$.

d) Achar un $z_0 \in Z$ tal que $argumento(z_0) = \min\{argumento(z) \mid z \in Z\}$.

```
Z=listacomplejos(20,10)
show(plce(Z),figsize=[2.5,2.5])
```




```
print 'a)'
Za=[z for z in Z if -5<=real(z)<=7]
len(Za)
```

```
a)
4
```

```
print 'b)'
Zb=[abs(z) for z in Z]
Z[Zb.index(max(Zb))]
```

```
b)
-9.87944010399016 - 18.0676849983995*I
```

```
print 'c)'
Zc=[arg(z) for z in Z]
Z[Zc.index(min(Zc))]
```

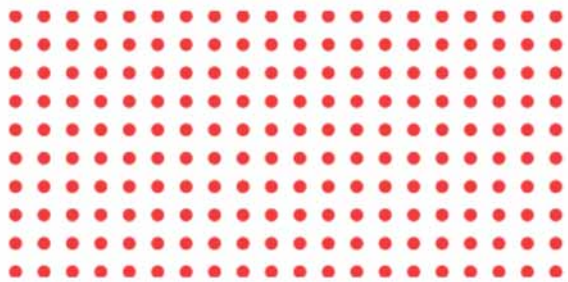
```
c)
-6.96027396199439 - 12.5774217465963*I
```

```
print 'd)'
Zd=[argumento(z) for z in Z]
Z[Zd.index(min(Zd))]
```

```
d)
2.72784626437312 + 0.895048594018917*I
```

Son de utilidade as funcións **rectangular(**n,m,b,h**)**, **triangular(**n,b,h**)**, **hexagonal(**m,n,r**)** que nos dan as siguintes configuracións de puntos no plano complexo:

```
R=rectangular(20,10,1,1)
show(plse(R),figsize=[4,2])
```

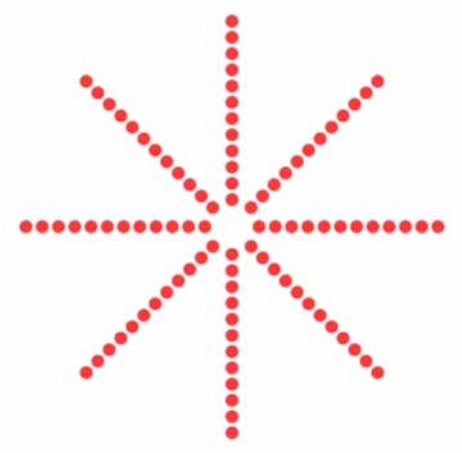


```
T1=triangular(20,1,sqrt(3)/2)
show(plse(T1),figsize=[4,2])
```

```
h=hexagonal(7,16,1)
show(h[1],figsize=[4,3])
```

A función **radial(**n,c,r,d**)** dános os vértices de c polígonos regulares de n vértices, homotéticos respecto de 0, sendo o primeiro de radio r, o segundo de radio r+d, o terceiro de radio r+2d e, así sucesivamente.

```
R=radial(8,12,5,3)
show(plse(R),figsize=[3,3])
```



3.1.6.1. Números poligonais

Se buscamos esta entrada en Internet podemos ler que é aquel número que pode ser representado como puntos dispostos en forma de polígono regular empezando polo 1.

A función **numpol**(k,n) calcula o n-ésimo número poligonal de k lados e clarifica a anterior definición:

Exemplo 1:

Valorar e debuxar o décimo número triangular, o oitavo número cadrado, o sétimo número pentagonal e o noveno número octogonal.

```
print '1)'; T10=numpol(3,10)
print len(T10)
show(plse(T10),figsize=[2,2])
```

```
1)
55
```

```
print '2)'; Q8=numpol(4,8)
print len(Q8)
show(plse(Q8),figsize=[2,2])
```

```
2)
64
```

```
print '3)'
P7=numpol(5,7)
print len(P7)
show(plse(P7),figsize=[2,2])
```

```
3)
70
```

```
print '4)'
O9=numpol(8,9)

print len(O9)
show(plse(O9),figsize=[2,2])
```

```
4)
225
```

Exemplo 2:

Achar o término xeral da sucesión de números triangulares, cadrados, pentagonais e octogonais,

```
print 'triangulares'
t(n)=n*(n+1)/2
show(expand(t(n)))
print 'cadrados'
q(n)=n^2
show(q(n))
print 'pentagonais'
p(n)=2*t(n)-1+t(n-2)
show(expand(p(n)))
print 'octogonais'
p(n)=2*t(n)-1+3*t(n-1)+t(n-2)
show(expand(p(n)))
```

```
triangulares
```

$$\frac{1}{2}n^2+\frac{1}{2}n$$

```
cadrados
```

$$n^2$$

```
pentagonais
```

$$\frac{3}{2}n^2-\frac{1}{2}n$$

```
octogonais
```

$$3n^2-2n$$

Os razonamentos anteriores xeralízanse facilmente e podemos asegurar que o término xeral de calquera número poligonal vén dado por un polinomio de grado 2. Así, as sucesións **numpol**(k,n) con $n = 1, 2, 3, \ldots$ son sucesións aritméticas de orde 2 e diferenza $k - 2$.

Se meditamos sobre a utilidade dos números poligonais chegamos á conclusión de que nos procuran unha técnica biparamétrica para ordenar conxuntos finitos de puntos en polígonos regulares. Con todo, é claro que a distribución de devanditos puntos no polígono, salvo no triángulo e no cadrado, non é moi harmónica.

A nosa función **numpolc**(k,n) introduce o n-ésimo número poligonal centrado de k lados que tamén nos proporciona unha distribución harmónica de puntos no caso hexagonal

Exemplo 3:

Valorar e debuxar o décimo número triangular centrado, o oitavo número cadrado centrado, o sétimo número pentagonal centrado e o noveno número hexagonal centrado.

```
print '1)'
T10=numpolc(3,10)
print len(T10)
show(plse(T10),figsize=[2,2])
```

```
1)
136
```

```
print '2)'; Q8=numpolc(4,8)
print len(Q8)
show(plse(Q8),figsize=[2,2])
```

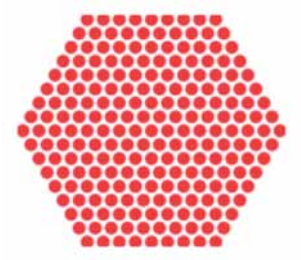
```
2)
113
```

```
print '3)';
P7=numpolc(5,7)
print len(P7)
show(plse(P7),figsize=[2,2])
```

```
3)
106
```

```
print '4)'
O9=numpolc(6,9)
print len(O9)
show(plse(O9),figsize=[2,2])
```

```
4)
217
```



Exercicio:

Comprobar que os términos xerais das sucesións de números triangulares centrados, cadrados centrados, pentagonais centrados e hexagonais centrados son, respectivamente:

TC) $\frac{3}{2}n^2 - \frac{3}{2}n + 1$

QC) $2n^2 - 2n + 1$

PC) $\frac{5}{2}n^2 - \frac{5}{2}n + 1$

HC) $3n^2 - 3n + 1$

3.1.7. Listas alfanuméricas

Os elementos dunha lista poden ser, á súa vez, listas numéricas ou simbólicas, de igual ou diferente lonxitude. Sage manéxaas con enorme facilidade e iso constitúe un dos potenciais esenciais deste software.

Presentaremos algunhas funcións que nos permitan xerar listas para modelizar problemas de selección de persoal ou de xestión de almacéns.

A función **Poblacion(**Ai,Af,n**)** xera unha lista aleatoria de n españois nacidos entre os anos Ai e Af cos datos que aparecen no DNI. A letra do DNI obtense a partir do seu número N coa funcion **letradni**(**N**).

A función **PoblacionQ(**A,e,E,n) xera unha lista aleatoria de n españois que no ano A teñen entre e e E anos de idade cos datos que aparecen no DNI, a súa titulación académica e o ámbito da súa formación.

A función **almacen('**X',n,p,P,u,U,V**)** dános unha lista A de lonxitude n cuxos elementos son listas de lonxitude 4:

$$[X_i, p_i, e_i, V(p_i, e_i)]$$

que describen os artigos almacenados: X_i é o identificativo do i-ésimo artigo, p_i o seu prezo de custo unitario, e_i o número de unidades existentes e $V(p_i, e_i)$ o seu prezo de venda unitario que pode depender do prezo de custo e das unidades existentes. A lista A xérase aleatoriamente con $p_i \in [p, P]$ e $e_i \in [u, U]$ unha vez que se especifican os intervalos e a función V. Os artigos aparecen en orde crecente do prezo de custo.

Para editar unha poboación ou un almacén utilizaremos a función **Publilista()**.

Exemplo 1:

Xerar unha lista aleatoria de 20 galegos nacidos entre 1980 e 1990 e ordenala alfabeticamente.

```
P=Poblacion(1980,1990,20)
PA=[a[0] for a in P]
Publilista(PA)
```

```
[1, [Feijoo, Bonan, Gertrudis]]
[2, [Zaragoza, Indurain, Alicia]]
[3, [Martinez, Canido, Alicia]]
[4, [Jubiol, Iglesias, Monica]]
[5, [Artigas, Cerrudo, Timoteo]]
[6, [Artigas, Somoza, Fernando]]
[7, [Gonzalez, Corbacho, Narciso]]
[8, [Goyanes, Castejon, Matilde]]
[9, [Corbacho, Benjumea, Emilio]]
[10, [Armada, Aznar, Liana]]
[11, [Lopez, Goyanes, Xiana]]
```

```
[12, [Garcia, Armada, Nieves]]
[13, [Cerrudo, Iglesias, Luis]]
[14, [Devesa, Latices, Matilde]]
[15, [Manero, Gea, Serena]]
[16, [Linares, Bonan, Patricia]]
[17, [Pujol, Castejon, Carla]]
[18, [Zaragoza, Corbacho, Julio]]
[19, [Somoza, Benjumea, Victor]]
[20, [Perez, Lago, Ana]]
```

```
PO=sorted([[str(a[0])]+[a] for a in P])
POC=[a[1][0] for a in PO]
Publilista(POC)
```

```
[1, [Armada, Aznar, Liana]]
[2, [Artigas, Cerrudo, Timoteo]]
[3, [Artigas, Somoza, Fernando]]
[4, [Cerrudo, Iglesias, Luis]]
[5, [Corbacho, Benjumea, Emilio]]
[6, [Devesa, Latices, Matilde]]
[7, [Feijoo, Bonan, Gertrudis]]
[8, [Garcia, Armada, Nieves]]
[9, [Gonzalez, Corbacho, Narciso]]
[10, [Goyanes, Castejon, Matilde]]
[11, [Jubiol, Iglesias, Monica]]
[12, [Linares, Bonan, Patricia]]
[13, [Lopez, Goyanes, Xiana]]
[14, [Manero, Gea, Serena]]
[15, [Martinez, Canido, Alicia]]
[16, [Perez, Lago, Ana]]
[17, [Pujol, Castejon, Carla]]
[18, [Somoza, Benjumea, Victor]]
[19, [Zaragoza, Corbacho, Julio]]
[20, [Zaragoza, Indurain, Alicia]]
```

Exemplo 2:

Xerar unha lista duns 20 españois para ocupar postos docentes, ordenados polos ámbitos Sanitario, Xurídico-Social, Científico, Tecnolóxico, Humanístico e Filolóxico.

```
PQ=PoblacionQ(2017,25,55,100)
PQD=[a for a in PQ if a[4]==Grado or a[4]==Mestrado or
a[4]==Doutor]
PQDS=[a[0]+[a[5]] for a in PQD if a[5]==Sanitario]
PQDJ=[a[0]+[a[5]] for a in PQD if a[5]==Xuridico_Social]
PQDC=[a[0]+[a[5]] for a in PQD if a[5]==Cientifico]
PQDT=[a[0]+[a[5]] for a in PQD if a[5]==Tecnoloxico]
PQDH=[a[0]+[a[5]] for a in PQD if a[5]==Humanistico]
PQDF=[a[0]+[a[5]] for a in PQD if a[5]==Filoloxico]
P=PQDS+PQDJ+PQDC+PQDT+PQDH+PQDF
Publilista(P)
```

```
[1, [Manero, Carnero, Miguel, Sanitario]]
[2, [Jubiol, Jubiol, Leticia, Sanitario]]
[3, [Iniesta, Aznar, Eduardo, Xuridico_Social]]
[4, [Barbera, Hinojosa, Elena, Xuridico_Social]]
[5, [Goyanes, Barbera, Ana, Xuridico_Social]]
[6, [Perez, Zapatero, Yago, Xuridico_Social]]
[7, [Iglesias, Gea, Matilde, Xuridico_Social]]
[8, [Fernandez, Perez, Mercedes, Xuridico_Social]]
[9, [Barbera, Devesa, Eduardo, Cientifico]]
[10, [Martinez, Cespon, Zoraida, Cientifico]]
[11, [Gonzalez, Gil, Victoria, Cientifico]]
[12, [Martinez, Barbera, Cristina, Tecnoloxico]]
[13, [Fernandez, Rajoy, Liana, Tecnoloxico]]
[14, [Pelaez, Esteban, Pedro, Tecnoloxico]]
[15, [Rodriguez, Varela, Fernando, Tecnoloxico]]
[16, [Borbon, Esteban, Isabel, Tecnoloxico]]
[17, [Mas, Gea, Berta, Tecnoloxico]]
[18, [Gea, Goyanes, Patricia, Tecnoloxico]]
[19, [Cerrudo, Fernandez, Paloma, Filoloxico]]
```

Exemplo 3:

```
V(x,y)=3*x+100/(21-y)
A=almacen('X',13,1,20,10,20,V)
Publilista(A)
```

```
[1, [X0, 2.7, 17, 33.1]]
[2, [X1, 4.3, 12, 24.0]]
[3, [X2, 4.5, 12, 24.6]]
[4, [X3, 4.8, 14, 28.7]]
[5, [X4, 4.9, 16, 34.7]]
[6, [X5, 6.5, 19, 69.5]]
[7, [X6, 8.0, 17, 49.0]]
[8, [X7, 11.1, 11, 43.3]]
[9, [X8, 14.7, 14, 58.4]]
[10, [X9, 15.5, 13, 59.0]]
[11, [X10, 15.8, 18, 80.7]]
[12, [X11, 18.6, 12, 66.9]]
[13, [X12, 19.9, 19, 109.7]]
```

3.1.8. Polinomios ortogonais

As familias de polinomios ortogonais cumpren leis de recurrencia a tres términos específicas. Por exemplo,

1) Legendre $(n+1)P_{n+1} = (2n+1)xP_n - nP_{n-1}$, con $P_0 = 1, P_1 = x$.

2) Hermite $P_{n+1} = 2xP_n - 2nP_{n-1}$, con $P_0 = 1, P_1 = 2x$.

3) Laguerre $(n+1)P_{n+1} = (2n+1-x)P_n - nP_{n-1}$, con $P_0 = 1, P_1 = -x+1$.

4) Tchebycheff $P_{n+1} = 2xP_n - P_{n-1}$. Tipo 1, con $P_0 = 1$, $P_1 = x$. Tipo 2, con $P_0 = 1, P_1 = 2x$.

As funcións **Legendre**(n), **Hermite**(n), **Laguerre**(n), **Tchebycheff1**(n) e **Tchebycheff2**(n) calcúlannolas.

```
Publilista(Legendre(6))
```

```
[1, 1]
[2, x]
[3, 3/2*x^2 - 1/2]
[4, 5/2*x^3 - 3/2*x]
[5, 35/8*x^4 - 15/4*x^2 + 3/8]
[6, 63/8*x^5 - 35/4*x^3 + 15/8*x]
[7, 231/16*x^6 - 315/16*x^4 + 105/16*x^2 - 5/16]
```

```
Publilista(Hermite(6))
```

```
[1, 1]
[2, 2*x]
[3, 4*x^2 - 2]
[4, 8*x^3 - 12*x]
[5, 16*x^4 - 48*x^2 + 12]
[6, 32*x^5 - 160*x^3 + 120*x]
[7, 64*x^6 - 480*x^4 + 720*x^2 - 120]
```

```
Publilista(Laguerre(6))
```

```
[1, 1]
[2, -x + 1]
[3, x^2 - 4*x + 2]
[4, -x^3 + 9*x^2 - 18*x + 6]
[5, x^4 - 16*x^3 + 72*x^2 - 96*x + 24]
[6, -x^5 + 25*x^4 - 200*x^3 + 600*x^2 - 600*x + 120]
[7, x^6 - 36*x^5 + 450*x^4 - 2400*x^3 + 5400*x^2 - 4320*x + 7
```

```
Publilista(Tchebycheff1(6))
```

```
[1, 1]
[2, x]
[3, 2*x^2 - 1]
[4, 4*x^3 - 3*x]
[5, 8*x^4 - 8*x^2 + 1]
[6, 16*x^5 - 20*x^3 + 5*x]
[7, 32*x^6 - 48*x^4 + 18*x^2 - 1]
```

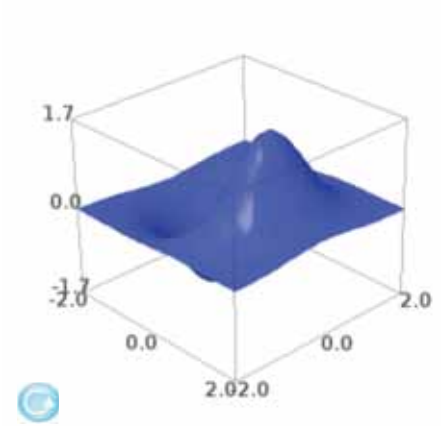
```
Publilista(Tchebycheff2(6))
```

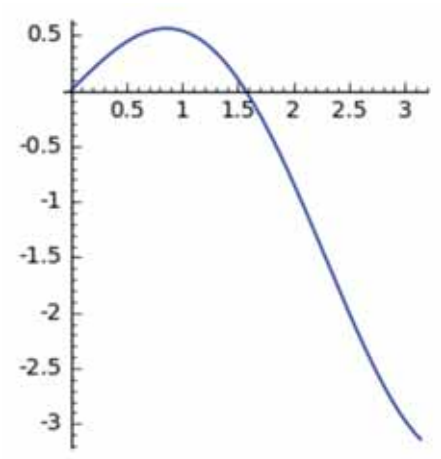
```
[1, 1]
[2, 2*x]
[3, 4*x^2 - 1]
[4, 8*x^3 - 4*x]
[5, 16*x^4 - 12*x^2 + 1]
[6, 32*x^5 - 32*x^3 + 6*x]
[7, 64*x^6 - 80*x^4 + 24*x^2 - 1]
```

3.1.9. Listas gráficas

Presentamos xa algúns obxectos gráficos como **point**(), **line**(), **plse**(), **poligonal**(), etc. En Sage as funcións $f : D \subset \mathbb{R} \to \mathbb{R}$ ou $F : D \subset \mathbb{R}^2 \to \mathbb{R}$ introdúcense moi sinxelamente e as ordes **plot**(f) e **plot3d(**F**)** proporciónannos novos obxectos gráficos ben coñecidos polo lector.

```
f(x)=x*cos(x)
F(x,y)=4*x*exp(-x^2-y^2)
G1=plot(f,(x,0,pi),aspect_ratio=1, figsize=[3,3])
G2=plot3d(F,(x,-2,2),(y,-2,2),aspect_ratio=[1,1,1], figsize=
[3,3,3])
show(G1)
show(G2)
```





Se G é unha lista de obxectos gráficos, A=**animate**(G) é a película constituída pola presentación secuencial de devanditos obxectos e podemos visionala coa orde **show**(A).

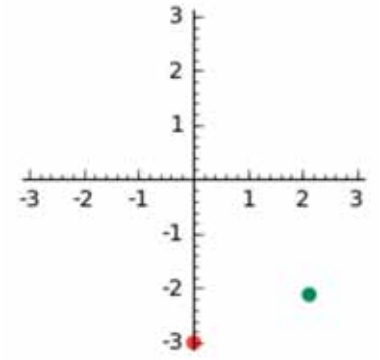
Exemplo 1:

a) Presenta un punto verde describindo unha circunferencia de radio r e outro punto vermello que a describa a dobre velocidade angular.

b) Presenta a traza do punto verde.

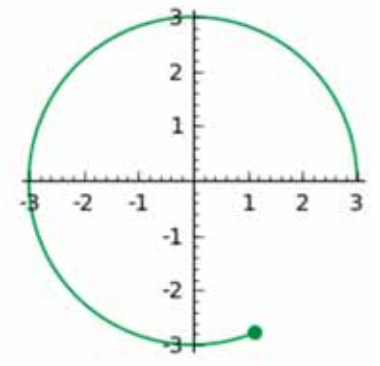
```
r=3

v(t)=[r*cos(t),r*sin(t)]
c(t)=[r*cos(2*t),r*sin(2*t)]
P0=point([-r,0],color='white')
P1=point([0,-r],color='white')
P2=point([r,0],color='white')
P3=point([0,r],color='white')
print 'Dous puntos en movemento:'
P=[]
T=[0,0.1,..,2*pi]
for i in T:
    PV=point(v(i),color='green',pointsize=40)
    PR=point(c(i),color='red',pointsize=40)
    P.append(P0+P1+PV+PR+P2+P3)
AP=animate(P,figsize=[3,3])
#show(AP)
show(P[55],figsize=[2.5,2.5])
```

```
Dous puntos en movemento:
```




```
print 'Traxectoria do punto verde:'
T=[0.1,0.2,..,2*pi]
ES=[]
for k in T:
    PV=point(v(k),color='green',pointsize=40)
    P=parametric_plot(v(t),
(t,0,k),color='green',aspect_ratio=1)
    ES.append(P0+P1+PV+P+P2+P3)
ANES=animate(ES,figsize=[3,3])
#show(ANES)
show(ES[50],figsize=[2.5,2.5])
```

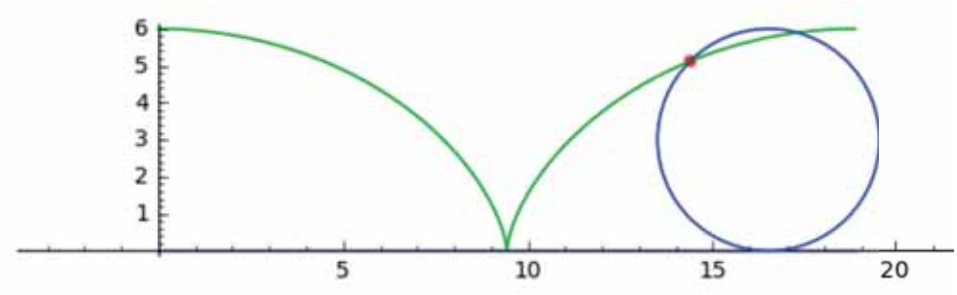
Traxectoria do punto verde:



Exemplo 2:

A cicloide de parámetro r é a curva que describe un punto dunha circunferencia de radio r que roda sen deslizar sobre o eixe OX.

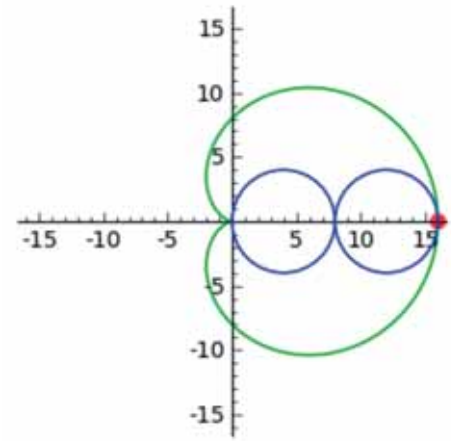
```
r=3

P1=point((-r,0),color='white')
P2=point((2*r*pi,0),color='white')
P3=point((-r,r),color='white')
P4=point((4*r*pi,r),color='white')
PS=P1+P2+P3+P4

R(t)=[r*t,r]
G(t)=[r*sin(t),r*cos(t)]
X=R+G


CUR=parametric_plot(X,(t,0,2*pi),hue=.3)
R=line([(0,0),(2*pi*r,0)])
T=[0,0.1,..,2*pi]
CIC=[]
for t in T:

CIC.append(CUR+PS+R+circle((r*t,r),r)+point(X(t),hue=0,pointsize=3
D=animate(CIC,figsize=[10,2])
show(CIC[55],figsize=[10,2])
```



Exemplo 3:

A cardiode de parámetro r é a curva que describe un punto dunha circunferencia de radio r que roda sen deslizar sobre outra circunferencia do mesmo radio.

```
r=4
P1=point((0,4*r),color='white')
P2=point((0,-4*r),color='white')
P3=point((-4*r,0),color='white')
P4=point((4*r,0),color='white')
PS=P1+P2+P3+P4
C(t)=[r+2*r*cos(t),2*r*sin(t)]
G(t)=[r*cos(2*t),r*sin(2*t)]
X=C+G
TQ=parametric_plot(X,(t,0,2*pi),hue=.3)
puntos=[]
T=[0,.1..2*pi]
for a in T:

puntos.append(circle((r,0),r)+circle((r+2*r*cos(a),2*r*sin(a)),r)+
a=animate(puntos,figsize=[3,3])
a.show(delay=10)
```



Podemos aproximar as lonxitudes das curvas anteriores coa función **loncurva**(A,c).

```
print 'circunferencia de radio r=3:'
r=3
A=[0,2*pi]
c(t)=[r*cos(t),r*sin(t)]
loncurva(A,c)
```

```
    circunferencia de radio r=3:
    18.8494774175970
```

```
print 'cicloide de parametro r=3:'
r=3
c(t)=[r*(t+sin(t)),r*(1+cos(t))]
loncurva(A,c)
```

```
    cicloide de parametro r=3:
    23.9999674086634
```

```
print 'cardioide de parametro r=3:'
r=3
A=[0,2*pi]
c(t)=[r+2*r*cos(t)+r*cos(2*t), 2*r*sin(t)+r*sin(2*t)]
loncurva(A,c)
```

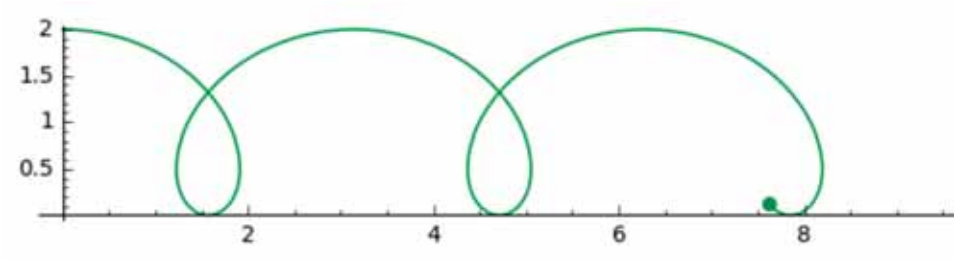
```
    cardioide de parametro r=3:
    47.9995351112404
```

Exemplo 4:

Un punto P desprázase pola recta y=1 a $1\frac{m}{s}$ e outro punto Q describe unha circunferencia de centro P e radio 1 a $2\frac{rad}{s}$.

Presentar a traza de Q nos primeiros 4π segundos se no instante inicial P está en (0,1) e Q en (0,2).

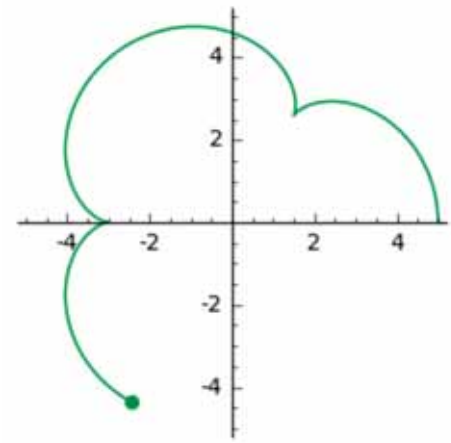
```
P(t)=[t,1]; Q(t)=[sin(2*t),cos(2*t)]
X=P+Q
P0=point((0,0),color='white')
P1=point((0,2),color='white')
P2=point((4*pi,0),color='white')
P3=point((4*pi,2),color='white')
MARCAS=P0+P1+P2+P3

A=[0.1,0.2,..,4*pi]
ES=[]
for k in A:
    PC=point(X(k),color='green',pointsize=40)
    TC=parametric_plot(X,(t,0,k),color='green',aspect_ratio=1)
    ES.append(PC+TC+MARCAS)
ANES=animate(ES)
show(ES[80])
```



Exemplo 5:

Un punto P describe unha circunferencia de centro (0,0) e radio 4 a $1\frac{rad}{s}$ e outro punto Q describe unha circunferencia de centro P e radio 1 a $4\frac{rad}{s}$. Se no instante inicial P está en (4,0) e Q en (5,0), presentar a primeira traza pechada de Q.

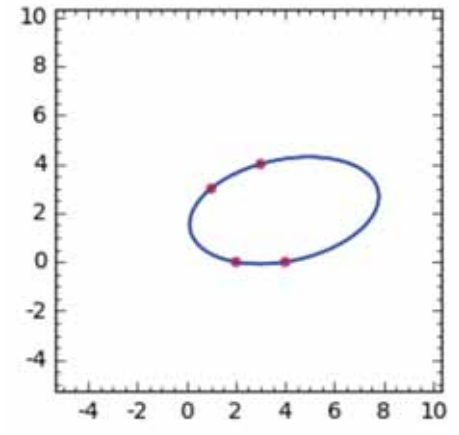
```
P(t)=[4*cos(t),4*sin(t)]
Q(t)=[cos(4*t),sin(4*t)]
X=P+Q
P0=point((-5,0),color='white')
P1=point((5,0),color='white')
P2=point((0,-5),color='white')
P3=point((0,5),color='white')
MARCAS=P0+P1+P2+P3
A=[0.0001]+[0.1,0.2,..,2*pi]+[2*pi]
ES=[]
for k in A:
    PC=point(X(k),color='green',pointsize=40)
    TC=parametric_plot(X,(t,0,k),color='green',aspect_ratio=1)
    ES.append(PC+TC+MARCAS)
ANES=animate(ES,figsize=[3,3])
show(ES[42],figsize=[3,3])
```



Exemplo 6:

Sexan P_0, P_1, P_2, P_3 catro puntos de $\mathbb{R}^2$. Xerar o feixe de cónicas que pasan por eles e animalo.

```
P=[[2,0],[4,0],[1,3],[3,4]]
PS=sum([point(p,color='red',pointsize=20) for p in P])

C1(x,y)=((x-P[0][0])*(P[1][1]-P[0][1])-(y-P[0][1])*(P[1][0]-
P[0][0]))*((x-P[2][0])*(P[3][1]-P[2][1])-(y-P[2][1])*(P[3][0]-
P[2][0]))
C2(x,y)=((x-P[0][0])*(P[2][1]-P[0][1])-(y-P[0][1])*(P[2][0]-
P[0][0]))*((x-P[1][0])*(P[3][1]-P[1][1])-(y-P[1][1])*(P[3][0]-
P[1][0]))
C=[]
L=[0,.05,..,1]
for r in L:
    C.append(PS+implicit_plot(r*C1+(1-r)*C2,(x,-5,10),
(y,-5,10),figsize=[3,3]))
A=animate(C)
#show(A)
show(C[18])
```

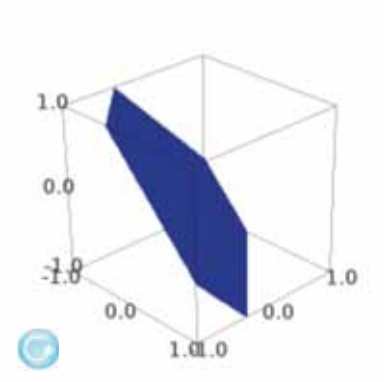


Exemplo 7:

Representar as seccións que determinan sobre o cubo $[-1,1]\times[-1,1]\times[-1,1]$ os planos paralelos ao x+y+z=1.

```
var('x y z')
T=[-3,-2.9,..,3]
A=[]
for k in T:
    A.append(implicit_plot3d(x+y+z-k, (x,-1,1),(y,-1,1),
(z,-1,1),figsize=[2.5,2.5,2.5]))
```

```
ANA=animate(A)
show(A[27])
```



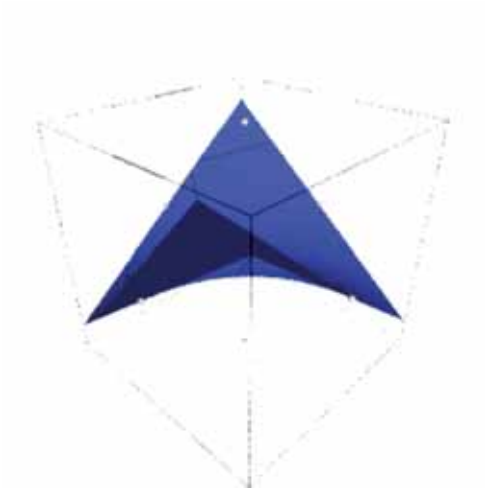
Exemplo 8:

Se K=[-2,-1.9,..,2], animar a familia de superficies $\{f_k \mid k \in K\}$ con $f_k : (-2,2) \times (-2,2) \to \mathbb{R}^3$ e $f_k(u,v) = [u,v,kuv]$

Representar a imaxe da función $f : (-2,2) \times (-2,2) \times (-2,2) \to \mathbb{R}^3$ con $f(u,v,w) = [u,v,uvw]$

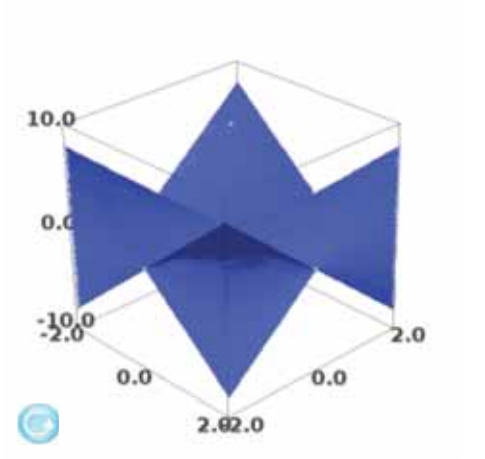
```
K=[-2,-1.9,..,2]
f(u,v,w)=[u,v,u*v*w]

FXI=point3d([-2,0,0],color='white',figsize=[3,3,3])
FXD=point3d([2,0,0],color='white',figsize=[3,3,3])
FYI=point3d([0,-2,0],color='white',figsize=[3,3,3])
FYD=point3d([0,2,0],color='white',figsize=[3,3,3])
FZB=point3d([0,0,-10],color='white',figsize=[3,3,3])
FZA=point3d([0,0,10],color='white',figsize=[3,3,3])
A=[]

for k in K:
    P=parametric_plot3d(f(u,v,k),(u,-2,2),
(v,-2,2),aspect_ratio=1,figsize=[3,3,3])
    A.append(FXI+FYI+FZB+P+FXD+FYD+FZA)
AN=animate(A)

print 'Animacion da familia de superficies:'
show(AN)
```

```
Animacion da familia de superficies:
```



```
print 'im(f):'
show(sum(A),figsize=[3,3])
```

```
im(f):
```



3.1.10. Vectores e Matrices

Se $X = \{0, 1, \cdots, n-1\}$ é un alfabeto de cardinal n, xa dixemos que unha lista L nun conxunto C é a imaxe dunha aplicación $V : X \to C$ escrita entre corchetes, na orde correspondente á de X:

$$L = [V(0), V(1), , \ldots, V(n-1)]$$

A orde **vector**(L) devólvenos ao propio concepto de aplicación V:

Se C ten estructura de anel conmutativo unitario $(C, +, *, 1)$, o conxunto C^X de todas as aplicacións $V : X \to C$, pode ser dotado de estructura de grupo conmutativo coa suma puntual $\oplus$:

$$V_1 \oplus V_2(i) = V_1(i) + V_2(i),$$

e pode ser dotado dun produto externo $\odot$ por elementos de C,

$$c \odot V(i) = c * V(i)$$

que, claramente, cumpre as seguintes propiedades:

1. $c \odot (V_1 \oplus V_2) = c \odot V_1 \oplus c \odot V_2$.

2. $(c_1 + c_2) \odot V = c_1 \odot V \oplus c_2 \odot V$.

3. $c_1 \odot (c_2 \odot V) = (c_1 * c_2) \odot V$.

4. $1 \odot V = V$.

5. $0 \odot V$=**0**.

A estructura $(C^X, \oplus, \odot)$ chámase módulo sobre o anel C. Nela é importante o subconxunto $\mathfrak{K} = \{\delta_i \mid i \in X\}$ das deltas de Kronecker onde

$$\delta_i : X \to C \quad \text{é tal que } \delta_i(j) = \begin{cases} 1 \text{ se } i = j \\ 0 \text{ se } i \neq j \end{cases}$$

pois calquera $V \in C^X$ pode escribirse como combinación lineal de elementos de $\mathfrak{K}$, é dicir, como suma puntual de produtos externos dun elemento de C por un elemento de $\mathfrak{K}$. Por exemplo, se a imaxe de V é a lista $[c_0, \ldots, c_{n-1}]$, é claro que

$$V = c_0 \odot \delta_0 \oplus \ldots \oplus c_{n-1} \odot \delta_{n-1}$$

Por iso dise que $\mathfrak{K}$ é sistema xerador de C^X e é importante o caso en que a expresión de calquera V como combinación lineal de elementos de $\mathfrak{K}$, é única, cousa que sucede cando

$$c_0 \odot \delta_0 + \cdots + c_{n-1} \odot \delta_{n-1} = \mathbf{0} \quad \Leftrightarrow \quad c_i = 0 \quad \forall i \in X.$$

Neste caso dicimos que tanto o conxunto $\mathfrak{K}$ como o módulo $(C^X, \oplus, \odot)$, son libres.

En particular, se o anel $(C, +, *, 1)$ é un corpo, o conxunto $\mathfrak{K}$ sempre é libre e a estrutura $(C^X, \oplus, \odot)$ chámase espazo vectorial sobre o corpo C.

Isto explica que para unha lista numérica L, as respostas á orde de Sage **type**(**vector**(L)) sexan:

```
X=vector(listasim('x',5)); A=vector(listareales(5,7,4))
print type(X); print type(A)
print X.parent()
print A.parent()
```

```
    <class
    'sage.modules.vector_symbolic_dense.FreeModule_ambient_field_
```

```
tegory.element_class'>
<type
'sage.modules.free_module_element.FreeModuleElement_generic_d
t;
Vector space of dimension 5 over Symbolic Ring
Vector space of dimension 4 over Real Field with 53 bits of
precision
```

Aínda que C pode ser o corpo racional QQ ou o corpo complexo CC, supoñeremos habitualmente que C=RR.

Sage emprega o signo + para a suma puntual en C^X e o signo * para o produto externo por elementos de C:

```
Y=vector(listasim('y',5))
B=vector(listareales(-10,10,4))
r=RR.random_element()
var('t',domain=RR)
print X+Y
print A+B
print t*X
print r*A
```

```
(x0 + y0, x1 + y1, x2 + y2, x3 + y3, x4 + y4)
(1.44124441493054, 1.50020950786274, 10.9718795096736,
1.48096528199362)
(t*x0, t*x1, t*x2, t*x3, t*x4)
(-0.873744794952428, -1.16448327745398, -1.04530245807147,
-1.12126116765308)
```

Sage tamén emprega o signo * para o produto escalar que é a operación $* : C^X \times C^X \to C$ que ao par de vectores (V,W) lle fai corresponder o elemento de C:

$$V * W = \sum_{i \in X} V(i) * W(i)$$

```
print X*Y
print A*B
```

```
x0*y0 + x1*y1 + x2*y2 + x3*y3 + x4*y4
-63.3636387794488
```

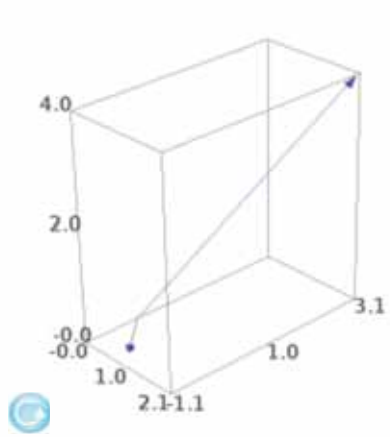
Un conxunto $\{V_1, \cdots, V_k\} \subset C^X$ dise ortogonal se $V_i * V_j = 0 \quad \forall i \neq j$.

O conxunto $\Re$ é ortogonal e é inmediato comprobar que todo conxunto ortogonal de vectores non nulos é libre.

Se X ten cardinal n, o espazo vectorial RR^X identifícase co espazo $\mathbb{R}^n$ das n-tuplas reais que o lector coñece ben nos casos n=1,2 e 3 por ser o modelo de espazo físico.

Por exemplo, en $\mathbb{R}^3$ os vectores A=(2,3,4) e B=(1,-1,1/4) son ortogonais e iso significa que na súa representación cartesiana son perpendiculares: $A * B = 0 \quad \Rightarrow A \perp B$

```
A=vector([2,3,4])
B=vector([1,-1,1/4])

show(arrow3d([0,0,0],list(A))+arrow3d([0,0,0],list(B)),
aspect_ratio=1, figsize=[3,3,3])
```



O lector coñece en $\mathbb{R}^3$ a operación interna produto vectorial $\wedge : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$ que ten, entre outras, as dúas propiedades seguintes:

1. $X \wedge Y = -Y \wedge X$.

2. $X \wedge Y \perp X$ e $X \wedge Y \perp Y$.

En Sage obtemos $X \wedge Y$ coa orde X.**cross_product(**Y**)** .

Ademais, esta orde de Sage tamén funciona como operación interna en $\mathbb{R}^7$.

```
X=vector(listasim('x',3))
Y=vector(listasim('y',3))
print X.cross_product(Y)
print X.cross_product(Y)==-Y.cross_product(X)
print (X.cross_product(Y)*X).rational_simplify()
print (X.cross_product(Y)*Y).rational_simplify()

X7=vector(listasim('x',7))
Y7=vector(listasim('y',7))
print X7.cross_product(Y7)
print X7.cross_product(Y7)==-Y7.cross_product(X7)
print (X7.cross_product(Y7)*X7).rational_simplify()
print (X7.cross_product(Y7)*Y7).rational_simplify()
```

```
(-x2*y1 + x1*y2, x2*y0 - x0*y2, -x1*y0 + x0*y1)
True
0
0
(-x3*y1 - x6*y2 + x1*y3 - x5*y4 + x4*y5 + x2*y6, x3*y0 - x4*y
x0*y3 + x2*y4 - x6*y5 + x5*y6, x6*y0 + x4*y1 - x5*y3 - x1*y4
- x0*y6, -x1*y0 + x0*y1 + x5*y2 - x6*y4 - x2*y5 + x4*y6, x5*y
x2*y1 + x1*y2 + x6*y3 - x0*y5 - x3*y6, -x4*y0 + x6*y1 - x3*y2
x2*y3 + x0*y4 - x1*y6, -x2*y0 - x5*y1 + x0*y2 - x4*y3 + x3*y4
x1*y5)
True
0
0
```

A función **norma(V, p)** calcula a norma p de calquera $V \in \mathbb{R}^n$ con $1 \leq p < \infty$. No caso euclídeo, $p = 2$, podemos suprimir a p e escribir, simplemente, **norma(V).**

Un conxunto $\{V_1, \cdots, V_k\} \subset \mathbb{R}^n$ dise ortonormal se é ortogonal e norma(V_i)=1 para todo i.

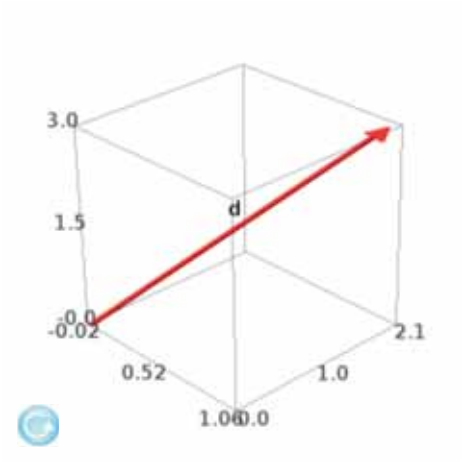
```
X=vector(listasim('x',5))
print norma(X)
print norma(X,1)
print norma(X,sqrt(2))
print

V=vector(listareales(-10,10,7))
print norma(V).n()
print norma(V,1).n()
print norma(V,sqrt(2)).n()
```

```
sqrt(x0^2 + x1^2 + x2^2 + x3^2 + x4^2)
abs(x0) + abs(x1) + abs(x2) + abs(x3) + abs(x4)
(abs(x0)^sqrt(2) + abs(x1)^sqrt(2) + abs(x2)^sqrt(2) +
abs(x3)^sqrt(2) + abs(x4)^sqrt(2))^(1/2*sqrt(2))

17.4335078704201
40.4035601874169
24.3193291562795
```

Pitágoras permítenos interpretar en $\mathbb{R}^3$ **norma(**X**)** como a distancia entre a orixe **0** e o punto X.

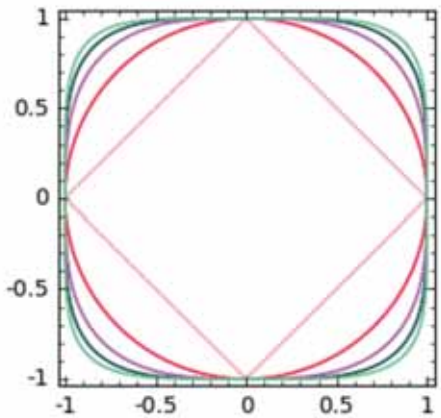
```
O=vector([0,0,0])
X=vector([1,2,3])
vX=arrow3d(O,X,color='red')
d=round(norma(X),3)

print 'd=',d
show(vX+text3d('d',(.5,1,1.8),fontsize=20),figsize=[3,3,3])
```

```
d= 3.742
```



A función **bola2d(**p**)** debúxanos en $\mathbb{R}^2$ a bóla unidade para a norma p no ceil(p)-ésimo cor da lista **colors.keys()** de Sage.

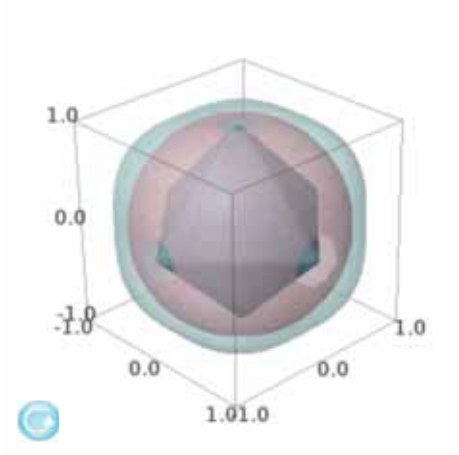
```
D=[]
for n in range(1,6):
    D.append(bola2d(n))
show(sum(D),figsize=[3,3])
```



A función **bola3d(**p) debúxanos en $\mathbb{R}^3$ a bóla unidade para a norma p no ceil(p)-ésimo cor da lista **colors.keys()** de Sage. Para permitir a visión de varias bólas ao mesmo tempo facemos o parámetro *opacity* igual a $1/p$.

```
D=[]
for n in range(1,4):
    D.append(bola3d(n))
```

```
show(sum(D),figsize=[3,3,3])
```

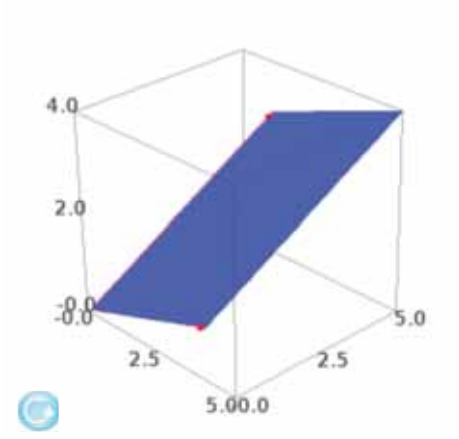


En $\mathbb{R}^3$ podemos representar o paralelogramo subtendido por dous vectores X, Y mediante a función **lelo(**X,Y**)**, o paralelepípedo subtendido por tres vectores X, Y, Z mediante a función **lele(**X,Y,Z**)** e o tetraedro de vértices 0,X,Y,Z, mediante a función **tetra(**X,Y,Z**)**.

Se L=[X,Y] ou L=[X,Y,Z], a función **gram(**L**)** dános a área ou o volume subtendidos.

```
X=vector([1,5,3])
Y=vector([4,0,1])
L=[X,Y]
A=gram(L).n()
print 'A=',A
show(lelo(X,Y),figsize=[3,3,3])
```

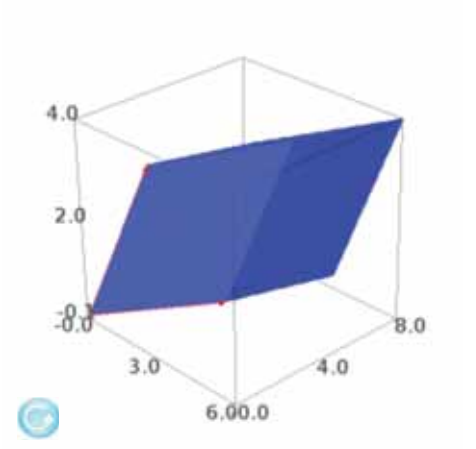
```
A= 23.3666428910958
```




```
X=vector([1,2,3])
Y=vector([4,2,1])
```

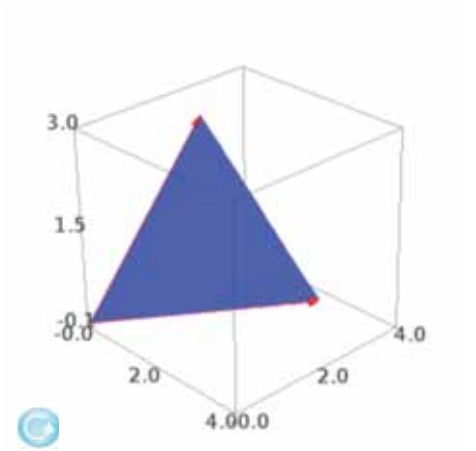
```
Z=vector([1,4,0])
L=[X,Y,Z]; V=gram(L).n()
print 'V=',V
show(lele(X,Y,Z),figsize=[3,3,3])
```

```
V= 40.0000000000000
```




```
V=(gram(L)/6).n()
print 'V=',V
show(tetra(X,Y,Z),figsize=[3,3,3])
```

```
V= 6.66666666666667
```



En xeral, dada unha lista $L = [V_1, \cdots, V_k]$ de vectores de $\mathbb{R}^n$ a función **gram(**L**)** dános a medida euclídea do k-paralelepípedo subtendido por devanditos vectores. En particular, **gram(**L**)**=0 se e só se os vectores da lista L non son libres.

A función **liberag(**L**)** dános unha sublista maximal de L que é libre.

Se L é unha lista de vectores libres, a función **GS(**L**)** dá unha base ortonormal do subespacio xerado por L.

```
L=[vector([1,2,3,4]),vector([2,4,6,8]),vector([2,3,5,0]),
vector([3,7,-1,2]),vector([1,0,2,-1])]
print liberag(L)
num(GS(L))
```

```
[(1, 2, 3, 4), (2, 3, 5, 0), (3, 7, -1, 2), (1, 0, 2, -1)]
[(0.182574185835055, 0.365148371670111, 0.547722557505166,
0.730296743340221),
 (0.273287955463098, 0.324991082172333, 0.598279037635431,
-0.679526808178514),
 (0.289716806266858, 0.764900856712917, -0.574827236534493,
-0.0237592025223030),
 (0.898912972557410, -0.419492720526791, -0.107869556706889,
0.0659202846542100)]
```

Unha lista L, simbólica ou numérica, de lonxitude $n \times m$ sometida á orde **matrix(**n,L**)** dános unha táboa M de n filas e m columnas que Sage entende como iremos vendo:

```
L=listasim('a',28)
M=matrix(7,L)
show(M)
```

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \\ a_{16} & a_{17} & a_{18} & a_{19} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{24} & a_{25} & a_{26} & a_{27} \end{pmatrix}$$

A función **matrixsim**('a',n,m) xera unha matriz simbólica subindicada de forma habitual.

```
M=matrixsim('a',3,5)
show(M)
```

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \end{pmatrix}$$

A orde M.**transpose()** dános a táboa trasposta M^t obtida cambiando filas por columnas.

```
show(M.transpose())
```

$$\begin{pmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{02} & a_{12} & a_{22} \\ a_{03} & a_{13} & a_{23} \\ a_{04} & a_{14} & a_{24} \end{pmatrix}$$

As ordes **matrix(**n**)** e **identity_matrix(**n) xérannos a matriz nula O_n e a matriz unidade U_n. A orde **diagonal_matrix(**L**)** xéranos a matriz de ceros con diagonal L:

```
print 'O4='
show(matrix(4))
print 'U4='
show(identity_matrix(4))
print
show(diagonal_matrix(listasim('a',4)))
```

O4=

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

U4=

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a_0 & 0 & 0 & 0 \\ 0 & a_1 & 0 & 0 \\ 0 & 0 & a_2 & 0 \\ 0 & 0 & 0 & a_3 \end{pmatrix}$$

A función **diagonalsmatrix**(L,D,n,m) xéranos unha matriz $n \times m$ que ten nas diagonais D[i] os elementos L[i] cortando ou reenchendo con ceros, segundo sexa necesario.

```
A=listasim('a',7)
B=listasim('b',4)
C=listasim('c',5)
D=[-1,0,2]
show(diagonalsmatrix([A,B,C],D,9,10))
```

$$\begin{pmatrix} b_0 & 0 & c_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_0 & b_1 & 0 & c_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_1 & b_2 & 0 & c_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_2 & b_3 & 0 & c_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_3 & 0 & 0 & c_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Unha lista de lonxitude n, de tuplas, listas ou vectores de lonxitude m, sometida á orde **matrix(**L**)** tamén nos dá una táboa de n filas e m columnas:

```
T=[(1,2,3),(2,3,4),(4,5,6)]
L=[[1,2,3],[2,3,4],[4,5,6],[6,7,8]]
V=[vector([1,2,3]),vector([2,3,4]),vector([4,5,6]),
vector([6,7,8]),vector([8,9,9])]
MT=matrix(T)
ML=matrix(L)
MV=matrix(V)
show(MT)
show(ML)
show(MV)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \\ 8 & 9 & 9 \end{pmatrix}$$

As ordes Q.**trace()** e **det(**Q**)** dannos a traza e o determinante dunha matriz cadrada Q.

```
Q=matrixsim('a',4,4)
print 'trace(Q)=',Q.trace()
print
print 'det(Q)=',det(Q)
```

```
trace(Q)= a00 + a11 + a22 + a33

det(Q)= a03*a12*a21*a30 - a02*a13*a21*a30 - a03*a11*a22*a30 +
a01*a13*a22*a30 + a02*a11*a23*a30 - a01*a12*a23*a30 -
a03*a12*a20*a31 + a02*a13*a20*a31 + a03*a10*a22*a31 -
a00*a13*a22*a31 - a02*a10*a23*a31 + a00*a12*a23*a31 +
a03*a11*a20*a32 - a01*a13*a20*a32 - a03*a10*a21*a32 +
a00*a13*a21*a32 + a01*a10*a23*a32 - a00*a11*a23*a32 -
a02*a11*a20*a33 + a01*a12*a20*a33 + a02*a10*a21*a33 -
a00*a12*a21*a33 - a01*a10*a22*a33 + a00*a11*a22*a33
```

Para unha matriz calquera M, a función **menoresom(**M**)** dános a lista de todos os menores de orde máxima.

```
M=matrixsim('a',4,3)
menoresom(M)
```

```
[-(a12*a21 - a11*a22)*a00 + (a02*a21 - a01*a22)*a10 - (a02*a1
a01*a12)*a20,
 -(a12*a31 - a11*a32)*a00 + (a02*a31 - a01*a32)*a10 - (a02*a1
a01*a12)*a30,
 -(a22*a31 - a21*a32)*a00 + (a02*a31 - a01*a32)*a20 - (a02*a2
a01*a22)*a30,
 -(a22*a31 - a21*a32)*a10 + (a12*a31 - a11*a32)*a20 - (a12*a2
a11*a22)*a30]
```

Dada unha matriz M as ordes M.**nrows**() e M.**ncols**() dannos o seu número de filas e columnas e a función **size(**M**)** devólvenos esa mesma información en forma de lista $[n, m]$.

As ordes M.**rows()** e M.**columns()** dannos as listas de vectores formadas polas filas e as columnas de M.

As ordes M.**row(**i**)** e M.**column(**j**)** dannos os vectores correspondentes á i-ésima fila e a j-ésima columna de M.

As funcións **suprimef(**M,i**)** e **suprimefc(**M,i,j**)** suprimen a i-ésima fila e a i-ésima fila e a j-ésima columna.

A orde **list(**M**)** devólvenos a lista de vectores formados polas filas de M e a función **listar(**M**)** devólvenos unha lista con todas as entradas da matriz, fila por fila.

```
M=matrixsim('b',5,7)
show(M)
print 'size =',size(M)
print 'filas='
print M.rows()
```

```
print 'columnas='
print M.columns()
print 'fila 2=',M.row(2)
print 'columna 3=', M.column(3)
print 'suprime fila 2'
show(suprimef(M,2))
print 'suprime fila 2 e columna 3'
show(suprimefc(M,2,3))
print 'list ='
print list(M)
print 'listar ='
print listar(M)
```

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} & b_{06} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \end{pmatrix}$$

```
size = [5, 7]
filas=
[(b00, b01, b02, b03, b04, b05, b06), (b10, b11, b12, b13, b1
b16), (b20, b21, b22, b23, b24, b25, b26), (b30, b31, b32, b3
b35, b36), (b40, b41, b42, b43, b44, b45, b46)]
columnas=
[(b00, b10, b20, b30, b40), (b01, b11, b21, b31, b41), (b02, b
b22, b32, b42), (b03, b13, b23, b33, b43), (b04, b14, b24, b3
b44), (b05, b15, b25, b35, b45), (b06, b16, b26, b36, b46)]
fila 2= (b20, b21, b22, b23, b24, b25, b26)
columna 3= (b03, b13, b23, b33, b43)
suprime fila 2
```

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} & b_{06} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \end{pmatrix}$$

```
suprime fila 2 e columna 3
```

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{04} & b_{05} & b_{06} \\ b_{10} & b_{11} & b_{12} & b_{14} & b_{15} & b_{16} \\ b_{30} & b_{31} & b_{32} & b_{34} & b_{35} & b_{36} \\ b_{40} & b_{41} & b_{42} & b_{44} & b_{45} & b_{46} \end{pmatrix}$$

```
list =
[(b00, b01, b02, b03, b04, b05, b06), (b10, b11, b12, b13, b1
b16), (b20, b21, b22, b23, b24, b25, b26), (b30, b31, b32, b3
```

```
b35, b36), (b40, b41, b42, b43, b44, b45, b46)]
listar =
[b00, b01, b02, b03, b04, b05, b06, b10, b11, b12, b13, b14, b
b16, b20, b21, b22, b23, b24, b25, b26, b30, b31, b32, b33, b
b35, b36, b40, b41, b42, b43, b44, b45, b46]
```

Se dúas matrices M1 e M2 teñen o mesmo número de filas pódense pegar horizontalmente coa función **pegah(**M1,M2**).** Se teñen o mesmo número de columnas pódense pegar verticalmente coa función **pegav(**M1,M2**)**.

Se **size(**M1**)**=[n,m], **size(**M2**)**=[n,k], **size(**M3**)**=[h,m] e **size(**M4**)**=[h,k] a orde **block_matrix([**[M1,M2],[M3,M4]**])** xera a matriz por bloques de tamaño [n+h,m+k]

$$\begin{pmatrix} M1 & M2 \\ M3 & M4 \end{pmatrix}$$

```
n=5;m=3;k=4;h=2
M1=matrixsim('a',5,3)
M2=matrixsim('b',5,4)
M3=matrixsim('c',2,3)
M4=matrixsim('d',2,4)
print 'M1=';show(M1)
print 'M2=';show(M2)
print 'pegah(M1,M2)='
show(pegah(M1,M2))
print 'M3=';show(M3)
print 'pegav(M1,M3)='
show(pegav(M1,M3))
print 'M4=';show(M4)
print 'block_matrix([[M1,M2],[M3,M4]])='
show(block_matrix([[M1,M2],[M3,M4]]))
```

M1=

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \\ a_{40} & a_{41} & a_{42} \end{pmatrix}$$

M2=

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \\ b_{40} & b_{41} & b_{42} & b_{43} \end{pmatrix}$$

pegah(M1,M2)=

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & b_{00} & b_{01} & b_{02} & b_{03} \\ a_{10} & a_{11} & a_{12} & b_{10} & b_{11} & b_{12} & b_{13} \\ a_{20} & a_{21} & a_{22} & b_{20} & b_{21} & b_{22} & b_{23} \\ a_{30} & a_{31} & a_{32} & b_{30} & b_{31} & b_{32} & b_{33} \\ a_{40} & a_{41} & a_{42} & b_{40} & b_{41} & b_{42} & b_{43} \end{pmatrix}$$

M3=

$$\begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \end{pmatrix}$$

pegav(M1,M3)=

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \\ a_{30} & a_{31} & a_{32} \\ a_{40} & a_{41} & a_{42} \\ c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \end{pmatrix}$$

M4=

$$\begin{pmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \end{pmatrix}$$

block_matrix([[M1,M2],[M3,M4]])=

$$\left(\begin{array}{ccc|cccc} a_{00} & a_{01} & a_{02} & b_{00} & b_{01} & b_{02} & b_{03} \\ a_{10} & a_{11} & a_{12} & b_{10} & b_{11} & b_{12} & b_{13} \\ a_{20} & a_{21} & a_{22} & b_{20} & b_{21} & b_{22} & b_{23} \\ a_{30} & a_{31} & a_{32} & b_{30} & b_{31} & b_{32} & b_{33} \\ a_{40} & a_{41} & a_{42} & b_{40} & b_{41} & b_{42} & b_{43} \\ \hline c_{00} & c_{01} & c_{02} & d_{00} & d_{01} & d_{02} & d_{03} \\ c_{10} & c_{11} & c_{12} & d_{10} & d_{11} & d_{12} & d_{13} \end{array}\right)$$

As matrices con entradas en RR ou CC, de tamaño [n,m], dotadas da suma de entradas correspondentes e do produto por elementos do corpo, teñen estrutura de espazo vectorial de dimensión nxm sobre o devandito corpo.

Unha base de este espazo vectorial é a formada por todas as matrices [n,m] con entradas nulas salvo unha con valor 1.

```
var('r')
M1=matrixsim('a',5,6)
M2=matrixsim('b',5,6)
print 'M1='
show(M1)
print 'M2='
show(M2)
print 'M1+M2='
show(M1+M2)
print 'r*M1='
show(r*M1)
M=matrixsim('c',2,3)
print 'M='
show(M)
B=[]
for i in range(2):
    for j in range(3):
        D=matrix(2,3)
        D[i,j]=1
        B.append(D)
print 'B='
show(B)
```

M1=

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{pmatrix}$$

M2=

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} & b_{05} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{pmatrix}$$

M1+M2=

$$\begin{pmatrix} a_{00}+b_{00} & a_{01}+b_{01} & a_{02}+b_{02} & a_{03}+b_{03} & a_{04}+b_{04} & a_{05}+b_{05} \\ a_{10}+b_{10} & a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} & a_{15}+b_{15} \\ a_{20}+b_{20} & a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{30}+b_{30} & a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{40}+b_{40} & a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \end{pmatrix}$$

```
r*M1=
```

$$\begin{pmatrix} a_{00}r & a_{01}r & a_{02}r & a_{03}r & a_{04}r & a_{05}r \\ a_{10}r & a_{11}r & a_{12}r & a_{13}r & a_{14}r & a_{15}r \\ a_{20}r & a_{21}r & a_{22}r & a_{23}r & a_{24}r & a_{25}r \\ a_{30}r & a_{31}r & a_{32}r & a_{33}r & a_{34}r & a_{35}r \\ a_{40}r & a_{41}r & a_{42}r & a_{43}r & a_{44}r & a_{45}r \end{pmatrix}$$

```
M=
```

$$\begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \end{pmatrix}$$

```
B=
```

$$\left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right]$$

e, claramente,

$$M = c_0 * B[0] + \cdots + c_5 * B[5]$$

Tamén podemos definir o produto dunha matriz M_1 de tamaño [n,m] por outra M_2 de tamaño [m,k] obtendo como resultado a matriz $M_1 * M_2$ de tamaño [n,k]. Como se pode comprobar esta operación é asociativa.

```
M1=matrixsim('a',3,4)
M2=matrixsim('b',4,2)
print 'M1 size',size(M1)
show(M1)
print 'M2 size', size(M2)
show(M2)
print 'M1*M2 size', size(M1*M2)
show(M1*M2)
```

```
M1 size [3, 4]
```

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{pmatrix}$$

```
M2 size [4, 2]
```

$$\begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \\ b_{30} & b_{31} \end{pmatrix}$$

M1*M2 size [3, 2]

$$\begin{pmatrix} a_{00}b_{00}+a_{01}b_{10}+a_{02}b_{20}+a_{03}b_{30} & a_{00}b_{01}+a_{01}b_{11}+a_{02}b_{21}+a_{03}b_{31} \\ a_{10}b_{00}+a_{11}b_{10}+a_{12}b_{20}+a_{13}b_{30} & a_{10}b_{01}+a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} \\ a_{20}b_{00}+a_{21}b_{10}+a_{22}b_{20}+a_{23}b_{30} & a_{20}b_{01}+a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} \end{pmatrix}$$

Os conxuntos de matrices reais ou complexas de tamaño [n,n] teñen, ademais da estrutura de espazo vectorial, a operación interna produto de matrices e, xa que logo, son álxebras non conmutativas con unidade Un=**identity_matrix**(n) que designaremos, respectivamente, $\mathcal{M}_n(RR)$ e $\mathcal{M}_n(CC)$.

```
M1=matrixsim('a',3,3)
M2=matrixsim('b',3,3)
print 'M1='
show(M1)
print 'M2='
show(M2)

html("<h4>En xeral, o produto non é conmutativo</h4>")
M=M1*M2
print 'M1*M2='
show(M)
N=M2*M1
print 'M2*M1='
show(N)
```

M1=

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

M2=

$$\begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \\ b_{20} & b_{21} & b_{22} \end{pmatrix}$$

En xeral, o produto non é conmutativo

```
M1*M2=
```

$$\begin{pmatrix} a_{00}b_{00}+a_{01}b_{10}+a_{02}b_{20} & a_{00}b_{01}+a_{01}b_{11}+a_{02}b_{21} & a_{00}b_{02}+a_{01}b_{12}+a_{02}b_{22} \\ a_{10}b_{00}+a_{11}b_{10}+a_{12}b_{20} & a_{10}b_{01}+a_{11}b_{11}+a_{12}b_{21} & a_{10}b_{02}+a_{11}b_{12}+a_{12}b_{22} \\ a_{20}b_{00}+a_{21}b_{10}+a_{22}b_{20} & a_{20}b_{01}+a_{21}b_{11}+a_{22}b_{21} & a_{20}b_{02}+a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

```
M2*M1=
```

$$\begin{pmatrix} a_{00}b_{00}+a_{10}b_{01}+a_{20}b_{02} & a_{01}b_{00}+a_{11}b_{01}+a_{21}b_{02} & a_{02}b_{00}+a_{12}b_{01}+a_{22}b_{02} \\ a_{00}b_{10}+a_{10}b_{11}+a_{20}b_{12} & a_{01}b_{10}+a_{11}b_{11}+a_{21}b_{12} & a_{02}b_{10}+a_{12}b_{11}+a_{22}b_{12} \\ a_{00}b_{20}+a_{10}b_{21}+a_{20}b_{22} & a_{01}b_{20}+a_{11}b_{21}+a_{21}b_{22} & a_{02}b_{20}+a_{12}b_{21}+a_{22}b_{22} \end{pmatrix}$$

Nas álxebras $\mathcal{M}_n(RR)$ e $\mathcal{M}_n(CC)$ podemos definir normas. Por exemplo, a orde de Sage **norm**() por cumprir que

$$norm(M1 * M2) \leq norm(M1) * norm(M2)$$

convérteas en álxebras normadas.

```
M1=random_matrix(CC,5); M2=random_matrix(CC,5)
norm(M1*M2)<=norm(M1)*norm(M2)
```

```
True
```

Toda matriz M de $\mathcal{M}_n(RR)$ ou de $\mathcal{M}_n(CC)$ con determinante non nulo ten unha M.**inverse()** para o produto tal que M*M.**inverse**()=M.**inverse**()*M=**identity_matrix**(n)

```
n=5; M=random_matrix(RR,n)
MI=M.inverse(); Un=identity_matrix(n)
show(norm(M*MI-Un));show(norm(MI*M-Un))
```

$$3.033335872 \times 10^{-15}$$

$$3.81662521041 \times 10^{-15}$$

Con ordes do tipo **random_matrix**(X,n,m) podemos xerar matrices M aleatorias de tamaño [n,m] e entradas en ZZ, QQ, RR ou CC.

Se M é real, podemos esixirlle que det(M)=1 (unimodular), que M=M^t (simétrica), que M=-M^t (antisimétrica) ou que M^{-1}=M^t (ortogonal) e se é complexa acharemos as súas partes real e imaxinaria coas funcións **parterm(**M) e **parteim(**M) e podemos esixirlle que M^{-1}=M^t.**conjugate**() (hermítica).

Aproveitamos para presentar a ferramenta **@interact** de Sage:

```
html("<h2>Matriz nxm simbólica ou numérica</h2>")
var('Simbolica Unimodular Simetrica Antisimetrica Ortogonal
Hermitica')
@interact
def matriz(T=input_box(default=Simbolica,label='tipo='),
           n=input_box(default=2,label='n='),
           m=input_box(default=2,label='m='),
           k=input_box(default=2,label='k=')):
              if T==Simbolica:
                  L=listasim('a',n*m)
                  M=matrix(n,L)
              elif T==ZZ:
                  M=random_matrix(ZZ,n,m)
              elif T==QQ:
                  M=random_matrix(QQ,n,m)
              elif T==RR:
                  M=random_matrix(RR,n,m)
              elif T==CC:
                  M=random_matrix(CC,n,m)
                  show(parterm(M))
                  show(parteim(M))
              elif T==Unimodular:
                  M=random_matrix(ZZ,n,algorithm=
                  'unimodular',upper_bound=k)
              elif T==Simetrica:
                  M=random_simmetric_matrix(n)
              elif T==Antisimetrica:
                  M=random_antisimmetric_matrix(n)
              elif T==Ortogonal:
                  M=random_orthogonal_matrix(n)
              elif T==Hermitica:
                  M=random_hermitic_matrix(n)
              return show(M)
```

En calquera matriz numérica de tamaño [n,m] cúmprese que

$$len(liberag(M.columns())) = len(liberag(M.rows()))$$

Esta cantidade pódese obter directamente coa orde rank(M), por exemplo para a matriz

$$\begin{pmatrix} 3 & 2 & 7 & 46 \\ 15 & 5 & 7 & -12 \\ 3 & 25 & -63 & 9 \\ 12 & 3 & 2 & -4 \\ 23 & 35 & 22 & -5 \\ 7 & 3 & 12 & -17 \\ 2 & 33 & 24 & -43 \end{pmatrix}$$

```
M=matrix([[3,2,7,46],[15,5,7,-12],[3,25,-63,9],[12,3,2,-
4],[23,35,22,-5],[7,3,12,-17],[2,33,24,-43]])
print len(liberag(M.rows()))==len(liberag(M.columns()))
print len(liberag(M.rows()))
print rank(M)
```

```
True
4
4
```

3.1.11. Aplicacións lineais

Sexan X e Y dous alfabetos de cardinais m, n e sexan C^X e C^Y os correspondentes espazos vectoriais sobre o corpo C. Diremos que unha aplicación $F : C^X \to C^Y$ é lineal se

1. $F(u_1 + u_2) = F(u_1) + F(u_2) \quad \forall u_1, u_2 \in C^X$.

2. $F(\lambda u) = \lambda F(u) \quad \forall u \in C^X$ e $\forall \lambda \in C$

O conxunto $\{u \in C^X \mid F(u) = 0\}$ é un subespazo vectorial de C^X que se denomina ker F.

O conxunto $\{F(u) \mid u \in C^X\}$ é un subespacio vectorial de C^Y que se denomina Im F.

É importante recordar a ecuación de dimensións:

$$m = dim C^X = dim\, ker F + dim\, Im F$$

A aplicación $F^\star : C^Y \to C^X$ tal que

$$F(x) * y = x * F^\star(y) \quad \forall (x, y) \in C^X \times C^Y$$

é, claramente, lineal e chámase aplicación adxunta de F. Evidentemente, $F^{\star\star} = F$.

Unha aplicación lineal $F : C^X \to C^X$ tal que $F = F^\star$ dise autoadxunta.

Tipo 1:

Sexan $U=[u_1,\ldots,u_k]$ e $V=[v_1,\ldots,v_k]$ listas de vectores de C^X e C^Y e sexa $R=[\lambda_1,\ldots,\lambda_k]$ unha lista de elementos de C. A aplicación $F=\sum_{i=1}^{k}\lambda_i u_i\otimes v_i : C^X\to C^Y$ tal que

$$F(x)=\sum_{i=1}^{k}\lambda_i(u_i*x)*v_i \quad \forall x\in C^X$$

é claramente lineal polas propiedades do produto escalar.

Se U e V son listas de vectores libres e R non ten elementos nulos cúmprese que $kerF=lin(u_1,\ldots,u_k)^{\perp}$ e $imF=lin(v_1,\ldots,v_k)$ e, xa que logo,

$$\dim kerF=m-k \quad \text{e} \quad \dim imF=k$$

co que constatamos fácilmente a ecuación de dimensións.

Podemos xerar unha aplicación lineal de Tipo 1 coa función **oplin**(U,V,R,X).

```
U=[vector([1,2,3]),vector([0,1,2]),vector([0,0,1])]
V=[vector([1,2,3,4]),vector([0,1,2,5]),vector([0,0,1,6])]
R=[1,3,5]
n=len(U[0])
X=vector(listasim('x',n))
oplin(U,V,R,X)
```

```
(x0 + 2*x1 + 3*x2, 2*x0 + 7*x1 + 12*x2, 3*x0 + 12*x1 + 26*x2,
23*x1 + 72*x2)
```

Tipo 2:

Sexa M unha matriz real [n,m] e sexan $x\in\mathbb{R}^m$ e $y\in\mathbb{R}^n$ vectores calquera.

a) A orde de Sage $M*x$ dános un vector de $\mathbb{R}^n$. A aplicación $G:\mathbb{R}^m\to\mathbb{R}^n$ tal que $G(x)=M*x$, coincide coa de Tipo 1 para U=M.rows(), V=identity_matrix(n).rows() e R unha lista de n uns.

G é, xa que logo, unha aplicación lineal que diremos asociada á matriz M.

```
n=8; m=6
M=matrixsim('a',n,m)
Un=identity_matrix(n)
show(M)
U=M.rows()
V=Un.rows()
R=[1 for k in range(n)]
X=vector(listasim('x',m))
oplin(U,V,R,X)==M*X
```

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \\ a_{60} & a_{61} & a_{62} & a_{63} & a_{64} & a_{65} \\ a_{70} & a_{71} & a_{72} & a_{73} & a_{74} & a_{75} \end{pmatrix}$$

```
True
```

b) A orde de Sage $y * M$ dános un vector de $\mathbb{R}^m$ que coincide con $M.transpose() * y$. A aplicación $D : \mathbb{R}^n \to \mathbb{R}^m$ tal que

$$D(y) = y * M = M^t * y$$

coincide coa aplicación lineal de Tipo 1 para U=M.columns(), V=identity_matrix(m).columns() e R unha lista de m uns.

```
Um=identity_matrix(m)
U1=M.columns()
V1=Um.columns()
R1=[1 for k in range(m)]
Y=vector(listasim('y',n))

oplin(U1,V1,R1,Y)==Y*M
```

```
True
```

c) $D = G^\star$

Demostración:

Debemos probar que $G(x) * y = x * D(y) \quad \forall (x, y) \in \mathbb{R}^m \times \mathbb{R}^n$

```
((M*X)*Y-X*(Y*M)).rational_simplify()
```

```
0
```

d) A aplicación composta $A = D \circ G : \mathbb{R}^m \to \mathbb{R}^m$ ten por matriz asociada $M^t * M$

$$A(x) = D(M * x) = M^t * M * x$$

e é unha aplicación autoadxunta posto que

$$A(x_1) * x_2 = M^t * M * x_1 * x_2 =$$

$$x_1 * (M^t * M)^t * x_2 = x_1 * M^t * M * x_2 = x_1 * A(x_2)$$

e) Unha aplicación lineal de Tipo 1 con listas U, V, R é unha aplicación de Tipo 2 asociada á matriz M:

$$M = matrix(V).transpose() * diagonal\_matrix(R) * matrix(U)$$

```
m=5; n=7
U=[vector(listasim('a',m)),vector(listasim('b',m)),
    vector(listasim('c',m))]
V=[vector(listasim('d',n)),vector(listasim('e',n)),
    vector(listasim('f',n))]
R=listasim('r',len(U))
M=matrix(V).transpose()*diagonal_matrix(R)*matrix(U)
X=vector(listasim('x',m))
RS(oplin(U,V,R,X)-M*X)
```

```
[0, 0, 0, 0, 0, 0, 0]
```

f) Toda aplicación lineal $F : \mathbb{R}^m \to \mathbb{R}^n$ é de Tipo 2 asociada a unha matriz real de tamaño [n,m].

Demostración:

Se $\{e_1, \ldots, e_m\}$ é unha base de $\mathbb{R}^m$, F está determinada pola lista $[F(e_1), \ldots, F(e_m)]$ ou pola matriz real M=**matrix**($[F(e_1), \ldots, F(e_m)]$) de tamaño [m,n]. Comprobaremos que F é unha aplicación lineal de Tipo 2 asociada á matriz M^t:

```
m=4; n=6
L=[vector(listasim('a0',n)),vector(listasim('a1',n)),
    vector(listasim('a2',n)),vector(listasim('a3',n))]
M=matrix(L).transpose();show(M)
Um=identity_matrix(m)
for i in range(m):
    M*Um.column(i)==L[i]
```

$$\begin{pmatrix} a_{00} & a_{10} & a_{20} & a_{30} \\ a_{01} & a_{11} & a_{21} & a_{31} \\ a_{02} & a_{12} & a_{22} & a_{32} \\ a_{03} & a_{13} & a_{23} & a_{33} \\ a_{04} & a_{14} & a_{24} & a_{34} \\ a_{05} & a_{15} & a_{25} & a_{35} \end{pmatrix}$$

```
True
True
True
True
```

3.1.12. Teoría espectral finito dimensional

Sexa $F : \mathbb{R}^n \to \mathbb{R}^n$ unha aplicación lineal. É importante saber se existe un vector non

nulo $x \in \mathbb{R}^n$ e un $k \in \mathbb{R}$ tal que $F(x) = k\,x$. Se $I_n : \mathbb{R}^n \to \mathbb{R}^n$ é a identidade, dita existencia darase se e só se $\ker(F - kI_n) \neq \{0\}$.

Fixada unha base en $\mathbb{R}^n$ poderemos asociar a aplicación F a unha matriz M de tamaño [n,n] e a identidade I_n á matriz unidade U_n e teremos:

$$\ker(F - kI_n) \neq \{0\} \quad \Leftrightarrow \quad \det(M - kU_n) = 0$$

A última expresión é unha ecuación polinómica de grado n en k, independente da base elixida, que podemos obter coa orde P= **charpoly**(M). As raíces deste polinomio, en xeral complexas, obtémolas coa orde P.**complex_roots**()

```
M=random_matrix(QQ,5)
P=charpoly(M)
var('k')
print 'P=',expand(P(x=k))
P.complex_roots()
show(P.complex_roots())
```

P= k^5 - 11/2*k^3 + 3/2*k^2 - 23*k + 26

[−3.05889898617754,1.00000000000000,2.61459891371236, −0.277849963767410

−1.78148697608977*i*,−0.277849963767410+1.78148697608977*i*]

A existencia dun autovalor nulo indica que $\ker F \neq \{0\}$ e, xa que logo, que F non é inxectiva.

A existencia dun autovalor real r asegura a dun autovector $x \in \mathbb{R}^n$ tal que $F(x) = rx$ e, xa que logo, a dunha recta invariante por F, a recta $lin(x)$.

A existencia dun autovalor complexo $a + ib$, asegura a de dous vectores $u, v \in \mathbb{R}^n$ tales que

$$F(u + iv) = (a + ib)(u + iv) = (au - bv) + i(bu + av) \quad \Rightarrow$$

$$\begin{cases} F(u) = au - bv \\ F(v) = bu + av \end{cases}$$

e, en consecuencia, a dun plano invariante por F, o plano $lin(u, v)$.

Ademais, como $F(u - iv) = (a - ib)(u - iv)$, vemos $a - ib$ tamén é autovalor de F e está ligado ao mesmo plano invariante. Isto é importante para entender a relación entre o número de autovalores e a suma das dimensións dos subespacios invariantes.

Se a aplicación lineal $F : \mathbb{R}^n \to \mathbb{R}^n$ está asociada a unha matriz M, con entradas en QQ, a orde de Sage $D, A = M$.**eigenmatrix_right**() devólvenos as matrices complexas D e A. A primeira é unha matriz diagonal formada polos autovalores de F e a segunda, é unha matriz inversible, coas columnas formadas polos autovectores

correspondentes. En consecuencia, $M * A = A * D$.

```
n=5
M=random_matrix(QQ,n)
show(M)
D,A=M.eigenmatrix_right()
DR=Roundmc(D,3)
AR=Roundmc(A,3)
show(DR)
show(AR)

for i in range(n):
    print M*A.column(i)==A.column(i)*D[i,i]
print
print 'En consecuencia'
print
Roundmc(M*A,16)==Roundmc(A*D,16)
```

$$\begin{pmatrix} -1 & -2 & \frac{1}{2} & -1 & \frac{1}{2} \\ -1 & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & -\frac{1}{2} & 0 \\ 0 & -1 & 1 & 0 & -\frac{1}{2} \\ 0 & -1 & 0 & 1 & \frac{1}{2} \end{pmatrix}$$

$$\begin{pmatrix} -1.822 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.868 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.774 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.208 - 1.013i & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.208 + 1.013i \end{pmatrix}$$

$$\begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \\ 0.373 & 1.172 & -1.254 & 0.778 - 0.796i & 0.778 + 0.796i \\ 0.104 & 2.376 & -0.059 & 0.847 + 0.315i & 0.847 - 0.315i \\ 0.171 & -0.629 & 0.325 & -2.683 + 0.957i & -2.683 - 0.957i \\ 0.087 & 1.317 & 1.239 & -0.687 - 3.614i & -0.687 + 3.614i \end{pmatrix}$$

```
True
True
True
True
True

En consecuencia
```

```
True
```

```
S1=Roundmc(svd(M)[0],2)
S2=Roundmc(svd(M)[1],2)
S3=Roundmc(svd(M)[2],2)
show(S1)
show(S2)
show(S3)
```

$$\begin{pmatrix} -0.81 & -0.54 & 0.02 & -0.05 & -0.22 \\ 0.31 & -0.52 & 0.1 & -0.73 & 0.31 \\ 0.1 & -0.43 & 0.24 & 0.63 & 0.59 \\ -0.41 & 0.33 & -0.46 & -0.18 & 0.69 \\ -0.27 & 0.38 & 0.85 & -0.19 & 0.18 \end{pmatrix}$$

$$\begin{pmatrix} 2.92 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.75 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.27 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.05 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.44 \end{pmatrix}$$

$$\begin{pmatrix} 0.17 & 0.89 & -0.37 & 0.17 & -0.11 \\ 0.61 & -0.08 & 0.43 & 0.65 & -0.14 \\ -0.1 & -0.26 & -0.58 & 0.56 & 0.52 \\ 0.74 & -0.25 & -0.45 & -0.43 & -0.03 \\ -0.21 & -0.26 & -0.37 & 0.23 & -0.83 \end{pmatrix}$$

A función **eigreal**(M) devólvenos dúas matrices reais Dr e Ar: A primeira diagonal por bloques e a segunda inversible que tamén cumpren que $M * Ar = Ar * Dr$.

```
Dr,Ar=eigreal(M)
A=Roundmc(eigreal(M)[1],4)
show(Dr)
show(A)
Roundm(M*Ar,12)==Roundm(Ar*Dr,12)
```

$$\begin{pmatrix} -\frac{79590843}{43694119} & 0 & 0 & 0 & 0 \\ 0 & -\frac{102179609}{117775753} & 0 & 0 & 0 \\ 0 & 0 & \frac{242003429}{136415825} & 0 & 0 \\ 0 & 0 & 0 & \frac{38593063}{185941085} & \frac{68391161}{67486158} \\ 0 & 0 & 0 & -\frac{68391161}{67486158} & \frac{38593063}{185941085} \end{pmatrix}$$

$$\begin{pmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 0.0 \\ 0.3729 & 1.1715 & -1.2542 & 0.778 & 0.7963 \\ 0.1043 & 2.3757 & -0.0585 & 0.847 & -0.3147 \\ 0.1713 & -0.6292 & 0.3247 & -2.6834 & -0.9565 \\ 0.0868 & 1.3167 & 1.2393 & -0.6868 & 3.6137 \end{pmatrix}$$

```
True
```

Se unha aplicación lineal $F:\mathbb{R}^n \to \mathbb{R}^n$ é autoadxunta todos os seus autovalores son reais pois, si $a+ib$ é autovalor,

$$\begin{cases} F(u) = au - bv \\ F(v) = bu + av \end{cases}$$

e, como $F(u)*v = F(v)*u$, teremos que $-b(v*v) = b(u*u)$. Logo b=0.

Ademais, dous autovalores distintos, $a \neq b$ teñen autovectores ortogonais pois

$$\begin{cases} F(u) = au \\ F(v) = bv \end{cases}$$

e, como $F(u)*v = F(v)*u$, teremos que $(a-b)(u*v) = 0$. Logo $u*v = 0$.

En particular, para calquera aplicación lineal $f:\mathbb{R}^n \to \mathbb{R}^m$, a composición $F = f^\star f : \mathbb{R}^n \to \mathbb{R}^n$ é autoadxunta e calquera autovalor a é real e non negativo:

$$a(u*u) = F(u)*u = (f^\star f(u)*u) = (f(u)*f(u) \geq 0 \Rightarrow \quad a \geq 0.$$

En particular, unha aplicación $F = f^\star f$ estará asociada a unha matriz $M = A^t * A$.

A función **svd**(M) devólvenos tres matrices reais O, D, O^t. D é unha matriz diagonal formada polos autovalores de F, O é unha matriz ortogonal cuxas columnas son os autovectores correspondentes e O^t é a súa traposta de modo que $M = O*D*O^t$.

```
A=random_matrix(RR,6,4)
M=A.transpose()*A
show(Roundmc(M,4))
S=svd(M)
show(Roundmc(S[0],4))
show(Roundmc(S[1],4))
show(Roundmc(S[2],4))
norm(M-prod(S))
```

$$\begin{pmatrix} 1.8035 & 0.4051 & 0.7057 & 1.0707 \\ 0.4051 & 2.4852 & -0.8859 & -0.382 \\ 0.7057 & -0.8859 & 2.2159 & 0.1965 \\ 1.0707 & -0.382 & 0.1965 & 2.1797 \end{pmatrix}$$

$$\begin{pmatrix} -0.4215 & -0.5503 & -0.2351 & -0.6813 \\ 0.4717 & -0.6874 & -0.3848 & 0.3962 \\ -0.5786 & 0.216 & -0.6686 & 0.4142 \\ -0.5149 & -0.4219 & 0.5913 & 0.4553 \end{pmatrix}$$

$$\begin{pmatrix} 3.6269 & 0.0 & 0.0 & 0.0 \\ 0.0 & 2.8534 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.7804 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.4236 \end{pmatrix}$$

$$\begin{pmatrix} -0.4215 & 0.4717 & -0.5786 & -0.5149 \\ -0.5503 & -0.6874 & 0.216 & -0.4219 \\ -0.2351 & -0.3848 & -0.6686 & 0.5913 \\ -0.6813 & 0.3962 & 0.4142 & 0.4553 \end{pmatrix}$$

```
1.9319117345855928e-15
```

Teorema espectral para aplicacións lineais autoadxuntas non negativas:

Toda aplicación lineal $F = f^\star f : \mathbb{R}^n \to \mathbb{R}^n$ é do Tipo 1 para unha lista de vectores ortonormais $[\mathbf{u}_1, \cdots, \mathbf{u}_n]$ e unha lista de reais non negativos $[r_1, \cdots, r_n]$ de modo que

$$F = \sum_{i=1}^{n} r_i \mathbf{u}_i \otimes \mathbf{u}_i$$

Demostración:

A aplicación $F : \mathbb{R}^n \to \mathbb{R}^n$ está asociada a unha matriz $M = A^t * A$ que admite unha diagonalización ortogonal, $M = O * D * O^t$ e, xa que logo,

$$F(\mathbf{x}) = M * \mathbf{x} = ODO^t * \mathbf{x} = (\mathbf{u}_1 \cdots \mathbf{u}_n) \begin{pmatrix} r1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & r_n \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 * \mathbf{x} \\ \vdots \\ \mathbf{u}_n * \mathbf{x} \end{pmatrix} =$$

$$\sum_{i=1}^{n} r_i(\mathbf{u}_i * \mathbf{x}) * \mathbf{u}_i.$$

A expresión de $F : \mathbb{R}^n \to \mathbb{R}^n$, asociada á matriz M, como aplicación de Tipo 1, obtémola coa función **teann**(M):

```
n=5
m=4
A=random_matrix(RR,4,5)
M=A.transpose()*A
print teann(M)
print
X=vector(listasim('x',n))
print M*X
```

```
    (0.9320131564996694*x0 - 0.0804854526394895*x1 -
    0.09378218170826072*x2 + 0.1543769101942127*x3 -
    0.48290811278512663*x4, -0.0804854526394895*x0 +
    1.3462920060383026*x1 - 1.4633559744860205*x2 + 0.81566962281
    + 0.09064956602928358*x4, -0.09378218170826069*x0 -
    1.4633559744860205*x1 + 1.7756223013202883*x2 -
    0.7265366581855923*x3 + 0.1636245064036366*x4, 0.154376910194
    + 0.8156696228160689*x1 - 0.7265366581855924*x2 +
    1.1247871194373575*x3 + 0.4799601855078581*x4,
    -0.4829081127851266*x0 + 0.09064956602928356*x1 +
    0.1636245064036366*x2 + 0.4799601855078582*x3 +
    0.7323295984430518*x4)

    (0.932013156499670*x0 - 0.0804854526394895*x1 -
    0.09378218170826080*x2 + 0.154376910194213*x3 - 0.482908112785
    -0.0804854526394895*x0 + 1.34629200603830*x1 - 1.463355974486
    0.815669622816069*x3 + 0.09064956602928354*x4, -0.093782181708
    - 1.46335597448602*x1 + 1.77562230132029*x2 - 0.7265366581855
    0.163624506403637*x4, 0.154376910194213*x0 + 0.81566962281606
    0.726536658185592*x2 + 1.12478711943736*x3 + 0.47996018550785
    -0.482908112785127*x0 + 0.09064956602928354*x1 + 0.16362450640
    + 0.479960185507858*x3 + 0.732329598443052*x4)
```

Exemplo:

Sendo $A \in \mathcal{M}(n \times n)$ con $det(A) \neq 0$, e $b \in \mathbb{R}^n$, utilizar o teorema anterior para estudar o sistema $Ax = b$.

```
L=[1..8]
n=choice(L)
detA=0
while detA<0.0001:
    A=random_matrix(QQ,n)
```

```
    detA=det(A)
b=random_vector(QQ,n)
M=A.transpose()*A
B=A.transpose()*b
P=diagonalizacion_ortogonal(M)
R=[P[0][i,i] for i in range(n)]
U=[P[1].column(i) for i in range(n)]
x=sum([((U[i]*B)/R[i])*U[i] for i in range(n)])
print 'A=';show(A)
print 'b=';show(b)
print 'x=';show(Roundc(x,3))
```

A=

$$\begin{pmatrix} 1 & -\frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$$

b=

$$(-2,\ 7)$$

x=

$$[1.2, 6.4]$$

Teorema espectral xeral:

Toda aplicación lineal $f:\mathbb{R}^m \to \mathbb{R}^n$ asociada a unha matriz A de tamaño $[n,m]$ e $rank(A)=k$ é de Tipo 1 para unha lista ortonormal $[\mathbf{u}_1,\cdots,\mathbf{u}_k]\subset\mathbb{R}^m$, outra lista ortonormal $[\mathbf{v}_1,\cdots,\mathbf{v}_k]\subset\mathbb{R}^n$ e unha lista de reais positivos $[s_1,\cdots,s_k]$ de modo que

$$f=\sum_{i=1}^{k} s_i\mathbf{u}_i\otimes\mathbf{v}_i$$

Demostración:

A aplicación lineal $F=f^\star f:\mathbb{R}^m\to\mathbb{R}^m$ é autoadxunta e definida non negativa. O teorema anterior asegura a existencia dunha lista ortonornal $[\mathbf{u}_1,\cdots,\mathbf{u}_m]$ de $\mathbb{R}^m$ constituída polos vectores propios de F. Se supoñemos que

$$lin(\mathbf{u}_1,\cdots,\mathbf{u}_k)=\ \mathrm{im}F \qquad \text{e} \qquad lin(\mathbf{u}_{k+1},\cdots,\mathbf{u}_m)=\ker F$$

a lista dos autovalores asociados será $[r_1 > 0, \cdots, r_k > 0, 0, \cdots, 0]$ e, en consecuencia,

$$F = \sum_{i=1}^{k} r_i \mathbf{u_i} \otimes \mathbf{u_i}.$$

A aplicación lineal $\mathcal{P} = F^{\frac{1}{2}} = \sum_{i=1}^{k} \sqrt{r_i}\mathbf{u_i} \otimes \mathbf{u_i}$ cumpre que

$$\|\mathcal{P}\mathbf{x}\|^2 = F\mathbf{x} * \mathbf{x} = \|f\mathbf{x}\|^2 \quad \forall \mathbf{x} \in \mathbb{R}^m$$

e, en consecuencia, existe unha aplicación lineal isométrica $U_0 : \mathcal{P}(\mathbb{R}^m) \to \mathbb{R}^n \quad \text{tal que} \quad U_0(\mathcal{P}\mathbf{x}) = f\mathbf{x}$ que nos asegura a factorización de $f : \mathbb{R}^m \xrightarrow{\mathcal{P}} \mathcal{P}(\mathbb{R}^m) \xrightarrow{U_0} \mathbb{R}^n$. Entón,

$$f\mathbf{x} = U_0\left(\sum_{i=1}^{k} \sqrt{r_i}(\mathbf{u}_i * \mathbf{x}) * \mathbf{u}_i\right) = \sum_{i=1}^{k} s_i(\mathbf{u}_i * \mathbf{x}) * U_0(\mathbf{u}_i) \quad \forall \mathbf{x} \in \mathbb{R}^m$$

O teorema queda probado con $\mathbf{v}_i = U_0(\mathbf{u}_i)$ e $s_i = \sqrt{r_i} \quad \forall i = 1, \cdots, k$.

A expresión de $f : \mathbb{R}^m \to \mathbb{R}^n$, asociada á matriz A, como aplicación de Tipo 1, obtémola coa función **TEG**(A).

```
A=random_matrix(RR,3,5)
X=vector(listasim('x',5))
T=TEG(A)
print A*X
print
oplin(T[0],T[1],T[2],X)
```

```
(-0.304444122611807*x0 + 0.442019399743148*x1 + 0.66682347144
+ 0.365773807562896*x3 - 0.483996783189247*x4, 0.527171133684
+ 0.208307148205914*x1 + 0.514143777852839*x2 -
0.0573810339003855*x3 - 0.543248446705786*x4, -0.343937702798
- 0.550497072483248*x1 - 0.827711209510973*x2 + 0.96030633950
- 0.485250041593122*x4)

(-0.3044441226118066*x0 + 0.44201939974314775*x1 +
0.666823471441348*x2 + 0.3657738075628962*x3 -
0.4839967831892466*x4, 0.5271711336845122*x0 +
0.20830714820591412*x1 + 0.5141437778528392*x2 -
0.05738103390038558*x3 - 0.5432484467057859*x4,
-0.3439377027983366*x0 - 0.5504970724832481*x1 -
0.8277112095109735*x2 + 0.9603063395014665*x3 -
0.4852500415931215*x4)
```

Teorema de Moore-Penrose:

Se $L : \mathbb{R}^m \to \mathbb{R}^n$ é lineal, con L= $\sum_{i=1}^{k} s_i \mathbf{u}_i \otimes \mathbf{v}_i$, a función lineal $L^+ : \mathbb{R}^n \to \mathbb{R}^m$ con L^+= $\sum_{i=1}^{k} \frac{1}{s_i} \mathbf{v}_i \otimes \mathbf{u}_i$ cumpre:

1. $L \cdot L^+ \cdot L = L$.

2. $L^+ \cdot L \cdot L^+ = L^+$.

3. $L \cdot L^+ = (L \cdot L^+)^t$.

4. $L^+ \cdot L = (L^+ \cdot L)^t$.

5. Por cumprir as anteriores, o operdor L^+ é único.

6. Si L é inversible, $L^+ = L^{-1}$.

Por cumprir 5. e 6., L^+ chámase inverso xeneralizado de Moore-Penrose de L.

Demostración:

1,2,3 e 4 compróbanse facilmente nos vectores básicos $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ e $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$.

5. Se supoñemos que existen dous inversos de Moore-Penrose L_1^+ e L_2^+ teremos:

$$L_1^+ = L_1^+ \cdot L \cdot L_1^+ = L_1^+ \cdot (L \cdot L_1^+)^t = L_1^+ \cdot (L_1^+)^t \cdot L^t =$$

$$L_1^+ \cdot (L_1^+)^t \cdot (L \cdot L_2^+ \cdot L)^t = L_1^+ \cdot (L_1^+)^t \cdot L^t \cdot (L_2^+)^t \cdot L^t =$$

$$L_1^+ \cdot (L \cdot L_1^+)^t \cdot (L \cdot L_2^+)^t = L_1^+ \cdot L \cdot L_2^+.$$

Por outra banda, de modo similar:

$$L_2^+ = L_2^+ \cdot L \cdot L_2^+ = (L_2^+ \cdot L)^t \cdot L_2^+ = L^t \cdot (L_2^+)^t \cdot L_2^+ =$$

$$(L \cdot L_1^+ \cdot L)^t \cdot (L_2^+)^t \cdot L_2^+ = L^t \cdot (L_1^+)^t \cdot L^t \cdot (L_2^+)^t \cdot L_2^+ =$$

$$(L_1^+ \cdot L)^t \cdot (L_2^+ \cdot L)^t \cdot L_2^+ = L_1^+ \cdot L \cdot L_2^+.$$

Así, $L_1^+ = L_2^+$ e a unicidade está probada.

6. Se L^{-1} existe, é claro que cumpre as propiedades 1.2.3.4. Por 5., é claro que $L^+ = L^{-1}$.

A orde **pinv(**M**)** importada de *NumPy*, como pode verse nos DATA, calcula a inversa de Moore Penrose dunha matriz M, mentres que a nosa función **mpinv(**M**)** calcula a

inversa de Moore Penrose da matriz M a partir da súa descomposición en valores singulares, segundo explicouse na teoría.

```
A=matrix([[1,2,3,4],[2,0,1,-1],[5,12,1,0]])
MPA=pinv(A)
MPMA=mpinv(A)
print 'MPA=',MPA
print 'MPMA=',MPMA
```

```
MPA= [0.016760297119617462   0.3204119801521301
0.005524346139281988]
[-0.01750936359167099 -0.15037453174591064  0.083614237606525
[ 0.12631087005138397   0.2024344503879547 -0.030992509797215
[  0.1598314642906189 -0.15674157440662384 -0.019943820312619

MPMA= [ 0.01676029962546817?   0.3204119850187266?
0.005524344569288397?]
[-0.01750936329588015?  -0.1503745318352060?  0.0836142322097
[  0.12631086142322107?   0.2024344569288390? -0.0309925093632
[  0.1598314606741573?  -0.1567415730337079? -0.0199438202247
```

Exemplo:

Determinar a solución, no sentido de mínimos cadrados do sistema $A\mathbf{x} = \mathbf{b}$ onde

$$A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 3 \\ 1 \\ 3 \\ 1 \end{pmatrix}$$

```
A=matrix([[1,0,1],[0,-1,1],[1,0,1],[0,1,-1]])
b=vector([3,1,3,1])
show(mpinv(A)*b)
```

$$(2,\ 1,\ 1)$$

3.1.13. Funcións diferenciables

Sexa A un aberto de $\mathbb{R}^m$. Toda función $f : A \to \mathbb{R}^n$ será escrita como unha lista de n expresións simbólicas nas variables $(x_0, \cdots, x_{m-1})$

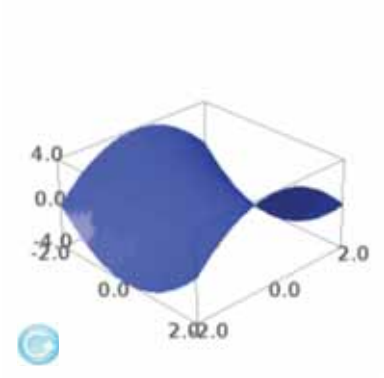
```
f(x0,x1,x2,x3)=[x0*x1+x2*x3]
g(x0,x1,x2)=[x0^2,x0*x1,x1*x2,cos(x0*x1*x2)]
show(f)
show(g)
```

$$(x_0, x_1, x_2, x_3) \mapsto (x_0x_1 + x_2x_3)$$

$$(x_0, x_1, x_2) \mapsto \left(x_0^2,\ x_0x_1,\ x_1x_2,\ \cos\left(x_0x_1x_2\right)\right)$$

Debemos advertir, con todo, que para representar graficamente funcións reais debemos substituír a lista polo elemento.

```
h(x,y)=[x^2-y^2]
show(plot3d(h[0],(x,-2,2),(y,-2,2)),figsize=[2.5,2.5])
```



Se $X = [1, 2, 3, 4]$ e $Y = [\frac{\pi}{2}, 2, 3]$ podemos obter os valores $f(X)$ e $g(Y)$ de dous xeitos:

```
show(f(1,2,3,4))
show(g(pi/2,2,3))
X=vector([1,2,3,4])
Y=vector([pi/2,2,3])
show(EV(f,X))
show(EV(g,Y))
```

$$(14)$$

$$\left(\frac{1}{4}\,\pi^2,\ \pi,\ 6,\ -1\right)$$

$$(14)$$

$$\left(\frac{1}{4}\,\pi^2,\ \pi,\ 6,\ -1\right)$$

Podemos obter a lista de variables que interveñen na expresión das funcións mediante a función **VAR()**.

```
show(VAR(f))
show(VAR(g))
```

$$[x_0, x_1, x_2, x_3]$$

$$[x_0, x_1, x_2]$$

Diremos que f é diferenciable en $\mathbf{x} \in A$ se existe unha aplicación lineal $L : \mathbb{R}^m \to \mathbb{R}^n$ tal que $f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) \approx L(\mathbf{h})$. O sentido preciso deste 'ser aproximadamente igual' é o seguinte:

$$\lim_{\mathbf{h}\to 0} \frac{\|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - L(\mathbf{h})\|}{\|\mathbf{h}\|} = 0$$

En caso de existir a aplicación lineal L, é única e denotámola $Df(\mathbf{x})$, diferencial de f en $\mathbf{x}$.

Evidentemente, si f é diferenciable en $\mathbf{x}$, é continua en $\mathbf{x}$.

Se f é lineal, é claro que $f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) = f(\mathbf{h})$ e, xa que logo, $Df(\mathbf{x}) = f \quad \forall \mathbf{x} \in A$.

• Se $m = 1$, a función f é diferenciable en $t \in A \subset \mathbb{R}$ se e só se existe o vector derivada

$$f'(t) = \lim_{h\to 0} \frac{f(t+h) - f(t)}{h}$$

que calculamos indistintamente coas ordes **derivative(**f,t**)**(t) ou **diff(**f,t**)**(t)

```
f(t)=[1,t,t^2,cos(t)]
show(derivative(f,t)(t))
show(diff(f,t)(t))
```

$$(0,\ 1,\ 2\,t,\ -\sin(t))$$

$$(0,\ 1,\ 2\,t,\ -\sin(t))$$

• Se $n = 1$, a función f é diferenciable en $\mathbf{x} \in A \subset \mathbb{R}^m$ se e só se existe o vector gradiente $\nabla f(\mathbf{x}) \in \mathbb{R}^m$ tal que

$$\lim_{\mathbf{h}\to 0} \frac{|f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}) - \nabla f(\mathbf{x}) * \mathbf{h}|}{\|\mathbf{h}\|} = 0$$

que calculamos coa función **EV(**derivative(f),VAR(f)).row(0)

```
f(x0,x1,x2,x3)=[x0*x3+x1*x2^2]
EV(derivative(f),VAR(f)).row(0)
```

```
(x3, x2^2, 2*x1*x2, x0)
```

Toda $f : A \subset \mathbb{R}^m \to \mathbb{R}$ pode considerarse, tamén, como unha función $f : A \times \mathbb{R}^k \subset \mathbb{R}^{m+k} \to \mathbb{R}$ pola inclusión natural $\mathbb{R}^m \subset \mathbb{R}^{m+k}$. Interésanos, xa que logo, calcular o gradiente de f en calquera das situacións anteriores.

Facémolo mediante a función **GRAD**(f[0],X).

```
f(x0,x1,x2,x3,x4,x5)=[x0*x3+x1*x2^2]
X=vector(listasim('x',6))
GRAD(f[0],X)
```

```
(x3, x2^2, 2*x1*x2, x0, 0, 0)
```

En xeral, a matriz asociada á aplicación lineal $Df(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$ calculámola coa función **JAC(f)**. A k-ésima fila desta matriz é o vector **GRAD(**f[k],**x)** sendo **x=VAR(f)**.

Para un punto concreto $\mathbf{a} \in A$, obtémola con **EV(JAC(**f),**a)**.

```
f(x0,x1,x2)=[x0^2,x0*x1,x1*x2,x2^2,x1-x2]
M=JAC(f)
show(M)

X=VAR(f)
for k in range(len(f)):
    print GRAD(f[k],X)

a=vector([1,2,3])
show(EV(M,a))
```

$$\begin{pmatrix} 2x_0 & 0 & 0 \\ x_1 & x_0 & 0 \\ 0 & x_2 & x_1 \\ 0 & 0 & 2x_2 \\ 0 & 1 & -1 \end{pmatrix}$$

```
(2*x0, 0, 0)
(x1, x0, 0)
(0, x2, x1)
```

```
(0, 0, 2*x2)
(0, 1, -1)
```

$$\begin{pmatrix} 2 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 3 & 2 \\ 0 & 0 & 6 \\ 0 & 1 & -1 \end{pmatrix}$$

Se $f : A \to \mathbb{R}^n$ é diferenciable en todo $\mathbf{x} \in A$ podemos definir a aplicación $Df : A \to L(\mathbb{R}^m, \mathbb{R}^n)$. Se esta fose diferenciable nun punto $\mathbf{x}$, existirá a aplicación lineal $D(Df)(\mathbf{x}) : \mathbb{R}^m \to L(\mathbb{R}^m, \mathbb{R}^n)$ que denotamos $D^2 f(\mathbf{x})$ e interpretamos como unha aplicación bilineal

$$D^2 f(\mathbf{x}) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$$

Diremos, entón, que f é dúas veces diferenciable en $\mathbf{x}$ e que $D^2 f(\mathbf{x})$ é a diferencial segunda de f en $\mathbf{x}$. As compoñentes desta aplicación, son as formas bilineales $D^2 f[k](\mathbf{x}) : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$. Para cada $k = 0, 1, \ldots, n-1$ podémola obter coa orde f[k].**hessian**() e, a lista de todas elas, coa función **HES**(f).

```
f(x,y)=[x^2*y^2,x*y^2,x+x*y]
H=HES(f)
show(H)
```

$$\left[\begin{pmatrix} 2\,y^2 & 4\,xy \\ 4\,xy & 2\,x^2 \end{pmatrix}, \begin{pmatrix} 0 & 2\,y \\ 2\,y & 2\,x \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right]$$

Para un punto concreto $\mathbf{a} \in A$, obtemos o valor de $D^2 f(\mathbf{a})$ no par $(\mathbf{x}, \mathbf{y})$ como segue:

```
a=vector([2,3])
H0=[EV(f[k].hessian(),a) for k in range(3)]
show(H0)
X=vector(listasim('x',2))
Y=vector(listasim('y',2))
D2fa(x0,x1,y0,y1)=[X*A*Y for A in H0]
D2fa
```

$$\left[\begin{pmatrix} 18 & 24 \\ 24 & 8 \end{pmatrix}, \begin{pmatrix} 0 & 6 \\ 6 & 4 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right]$$

```
(x0, x1, y0, y1) |--> (6*(3*x0 + 4*x1)*y0 + 8*(3*x0 + x1)*y1,
6*x1*y0 + 2*(3*x0 + 2*x1)*y1, x1*y0 + x0*y1)
```

Para unha función $f : A \to \mathbb{R}^n$ dúas veces diferenciable en todo $\mathbf{x} \in A$ podemos mellorar a aproximación lineal

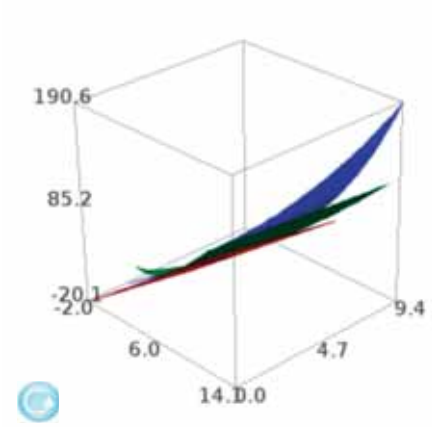
$$f(\mathbf{a}+\mathbf{h}) \approx f(\mathbf{a}) + Df(\mathbf{a})(\mathbf{h})$$

coa aproximación cuadrática

$$f(\mathbf{a}+\mathbf{h}) \approx f(\mathbf{a}) + Df(\mathbf{a})(\mathbf{h}) + \tfrac{1}{2}D^2 f(\mathbf{a})(\mathbf{h},\mathbf{h})$$

como podemos ver no seguinte exemplo:

```
f(x,y)=[x^2*y^2,x*y^2,exp(x+x*y)]
a=vector([1,2])
h=vector(listasim('h',2))
D=parametric_plot3d(EV(f,a+h),(h0,-.5,.5),(h1,-.5,.5))
T=parametric_plot3d(EV(f,a)+EV(JAC(f),a)*h,(h0,-.5,.5),
(h1,-.5,.5),color='red')
H0=[EV(f[k].hessian(),a) for k in range(3)]
Q=parametric_plot3d(EV(f,a)+EV(JAC(f),a)*h+(1/2)*
    vector([h*A*h for A in H0]),(h0,-.5,.5),
(h1,-.5,.5),color='green')
show(D+T+Q, figsize=[3,3,3])
```



En teoría poderíamos falar de aproximacións de orde superior pero no mundo técnico non adoitan utilizarse: No estudo do movemento chegamos só á segunda variación da posición con respecto ao tempo. Nas ecuacións da Física-Matemática só aparecen derivadas de ata o orde dous. É máis, coa reiteración de aproximacións lineais (algoritmos tipo Newton) puidéronse resolver importantes problemas non lineais.

3.1.14. Variedades diferenciables

Un conxunto $M \subset \mathbb{R}^n$ é unha variedade diferenciable q-dimensional ($1 \leq q \leq n$) de clase $\mathcal{C}^k$ ($k \geq 1$) se para cada $x \in M$ existe un par (U, f) que cumpre:

1. U é un aberto de $\mathbb{R}^q$.

2. $f : U \to M \subset \mathbb{R}^n$ é unha aplicación de clase $\mathcal{C}^k$ con $rankDf(u) = q \quad \forall u \in U$.

3. $f(U)$ é un entorno de x en M.

Escribimos $M \in \mathcal{V}_q^k(\mathbb{R}^n)$ e dicimos que (U, f) é unha carta local do punto $x \in M$. O conxunto de cartas locais necesarias para describir toda a variedade é o atlas de la variedade.

O feito de que $rankDf(u) = q$ indica que os vectores $\{\frac{\partial f}{\partial u_0}, \ldots, \frac{\partial f}{\partial u_{q-1}}\}$ constitúen unha base de $imDf(u) = T_x(M)$ que é o espazo tanxente á variedade no punto $x = f(u)$. O subespacio ortogonal $N_x(M) = T_x(M)^{\perp}$, é o espazo normal á variedade no punto x. A variedade afín $x + T_x(M)$ é a máis próxima a M na entorna de x.

Aquí trataremos de variedades diferenciables dunha soa carta. Salvo no primeiro exemplo, estudaremos variedades en $\mathbb{R}^2$ ou $\mathbb{R}^3$. As unidimensionais chámanse curvas, as bidimensionais, superficies e as tridimensionais, sólidos.

Podemos calcular a medida xeométrica dunha variedade $M \subset \mathbb{R}^3$ dada pola carta (U, f) integrando a función **med**(f) no aberto U. Esta función que está incluída nos DATA procede recordala aquí

```
def med(f):
    V=VAR(f)
    n=len(V)
    M=JAC(f)
    T=[M.column(i) for i in range(n)]
    G=gram(T).full_simplify()
    return G
```

para deixar claro que o que fai é medir no espazo tanxente á variedade, calculando o gramiano dos vectores tanxentes.

Se a variedade é unidimensional a súa medida xeométrica chamase lonxitude $\lambda(M)$, se é bidimensional, área $\sigma(M)$ e se é tridimensional, volume $v(M)$ e, en xeral, denótase $\mu(M)$. O **centroide** de M é o vector

$$\mathbf{c}(M) = \frac{1}{\mu(M)} \int_U f \cdot \mathbf{med}(f)$$

Se $\delta : M \to [0, \infty]$ é calquer función de densidade, de masa, de carga eléctrica, etc., ao integrar $(\delta \circ f) \cdot med(f)$ no aberto U obtemos unha nova medida da variedade M, a masa total $m(M)$, a carga total $q(M)$, etc.

Se δ é a densidade de masa, o **baricentro** de M é o vector

$$\mathbf{b}(M) = \frac{1}{m(M)} \int_U (\delta \circ f) \cdot f \cdot \mathbf{med}(f)$$

Se L_0 é unha recta en $\mathbb{R}^3$ que pasa pola orixe e $D_{L_0} : M \to \mathbb{R}_+$ é tal que $D_{L_0}(x) = \inf_{y \in L_0} \{\|x - y\|^2\}$, é doado ver que $D_{L_0}(x) = \mathbf{gram}^2([x, e])$ sendo e o vector direccional unitario de L_0. O momento de inercia de M respecto de L_0

$$I(M, L_0) = \int_U (\delta \circ f) \cdot (D_{L_0} \circ f) \cdot \mathbf{med}(f) = \int_U (\delta \circ f) \cdot \mathbf{gram}^2([f, e]) \cdot \mathbf{med}(f$$

O teorema de Steiner asegura que o momento de inercia respecto de calquera outra recta L paralela a L_0 é

$$I(M, L) = d^2 m(M) + I(M, L_0)$$

onde d é a distancia entre as rectas L e L_0 (Véxase [13]).

Exemplo 1:

Sexa $n \geq q$ e sexa A unha matriz real de tamaño [n,q] con $rank(A) = q$. Se $U = \mathbb{R}^q$ e $f : U \to \mathbb{R}^n$ é tal que $f(X) = A * X \quad \forall X \in U$ verifícase que $im\, f \in \mathcal{V}_q^\infty(\mathbb{R}^n)$.

```
n=4; q=3
E1=random_matrix(QQ,n,q)
print 'E1='
show(E1)
X=vector(listasim('u',q))
print 'f='
f(u0,u1,u2)=list(E1*X)
show(f)
Df(u0,u1,u2)=JAC(f)
print 'Df='
show(Df)
print 'rank Df(X)=',rank(JAC(f))
```

E1=

$$\begin{pmatrix} -2 & 1 & -2 \\ -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -2 \\ 2 & -1 & -1 \end{pmatrix}$$

f=

$$(u_0, u_1, u_2) \mapsto \left(-2\,u_0 + u_1 - 2\,u_2,\ -\frac{1}{2}\,u_0,\ u_1 - 2\,u_2,\ 2\,u_0 - u_1 - u_2\right)$$

Df=

$$(u_0, u_1, u_2) \mapsto \begin{pmatrix} -2 & 1 & -2 \\ -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -2 \\ 2 & -1 & -1 \end{pmatrix}$$

rank Df(X)= 3

Exemplo 2:

Sexa $U = (0, 2\pi) \subset \mathbb{R}$, $f : U \to \mathbb{R}^2$ tal que $f(t) = [cos(t), sin(t)] \quad \forall t \in U$.

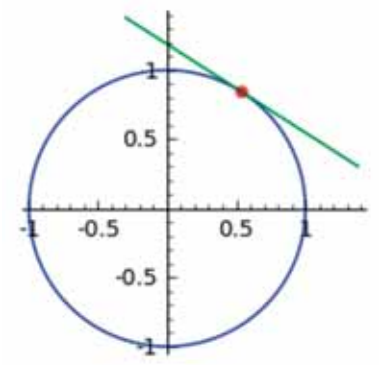
Probar que $imf \in \mathcal{V}_1^{\infty}(\mathbb{R}^2)$.

Representar a variedade e a súa tanxente no punto $f(1)$.

Presentar a animación da tanxente ao longo da curva.

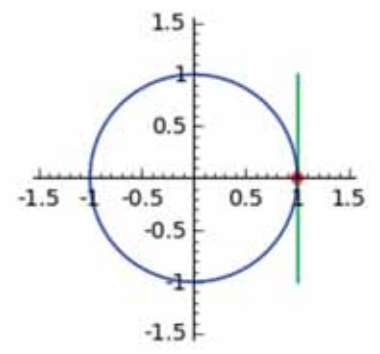
```
U=(0,2*pi)
f(t)=[cos(t),sin(t)]
print 'f(t)=',f(t)
Df=diff(f,t)
print 'Df(t)=',Df(t)
print 'rank Df(t)=',rank(matrix(Df(t)))
print 'Representación gráfica:'
G1=parametric_plot(f,U)
G2=point(f(1), hue=0,pointsize=30)
VAT(p)=list(f(1)+p*Df(1))
G3=parametric_plot(VAT,(p,-1,1),color='green')
show(G1+G2+G3,figsize=[2.5,2.5])
```

f(t)= (cos(t), sin(t))
Df(t)= (-sin(t), cos(t))
rank Df(t)= 1
Representación gráfica:



```
print 'Animacion da tanxente:'
R=[0,0.4..2*pi]
AG3=[]
for r in R:
    G2=point(f(r), hue=0,pointsize=30)
    VAT(p)=list(f(r)+p*Df(r))
    G3=parametric_plot(VAT,(p,-1,1),color='green')
    AG=G1+G2+G3
    AG3.append(AG)
ANG3=animate(AG3,xmin=-1.5,ymin=-1.5,xmax=1.5,ymax=1.5,figsize=
[2.5,2.5])
ANG3.show()
```

Animacion da tanxente:



Exemplo 3:

Sexa $U=(0,2\pi)\times(0,1)\subset\mathbb{R}^2$, $f:U\to\mathbb{R}^2$ tal que $f(t,v)=[vcos(t),vsin(t)]\quad\forall(t,v)\in U$. Probar que $imf\in\mathcal{V}_2^{\infty}(\mathbb{R}^2)$ e representala.

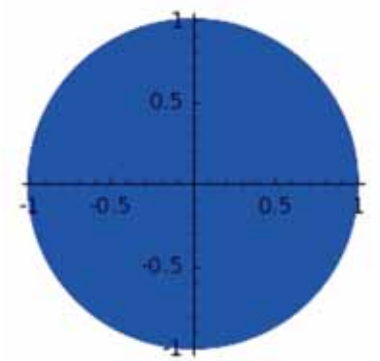
```
f(t,v)=[v*cos(t),v*sin(t)]
print 'f(t,v)=',f(t,v)
X=vector([t,v])
Df(t,v)=EV(jacobian(f,X),X)
print 'Df(t,v)='
show(Df(t,v))
print 'rank Df(X)=',rank(EV(jacobian(f,X),X))
print
print 'Representación gráfica'

V=[0,10^(-3)..1]
G=[]
for r in V:
    G.append(parametric_plot(f(t,r),(t,0,2*pi)))
show(sum(G),figsize=[2.5,2.5])
```

```
f(t,v)= (v*cos(t), v*sin(t))
Df(t,v)=
```

$$\begin{pmatrix} -v\sin(t) & \cos(t) \\ v\cos(t) & \sin(t) \end{pmatrix}$$

```
rank Df(X)= 2
```

Representación gráfica

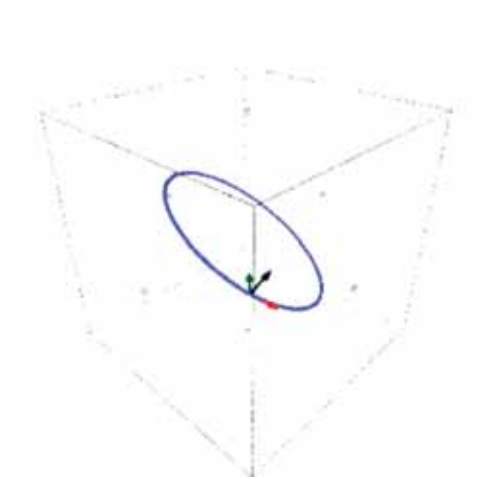


Exemplo 4:

Para representar en $\mathbb{R}^3$ unha circunferencia de centro C, radio r e vector director D podemos utilizar a función **circunferencia**(C,r,D). Tamén podemos usar a función **tnb**(f) para achar o triedro de Frenet dunha curva $f : (a, b) \to \mathbb{R}^3$.

```
c=circunferencia([1,2,3],3,[1,1,2])
C=parametric_plot3d(c,(t,0,2*pi),thickness=3,xmin=-2,ymin=-
1,zmin=0,xmax=4,ymax=5,zmax=6,aspect_ratio=1,figsize=[3,3,3])
FXI=point3d([-3,2,3],color='white',figsize=[3,3,3])
FXD=point3d([5,2,3],color='white',figsize=[3,3,3])
FYI=point3d([1,-2,3],color='white',figsize=[3,3,3])
FYD=point3d([1,6,3],color='white',figsize=[3,3,3])
FZB=point3d([1,2,-1],color='white',figsize=[3,3,3])
FZA=point3d([1,2,7],color='white',figsize=[3,3,3])

A=[]
R=[0,.5,..,2*pi]
for r in R:

A.append(FXI+FYI+FZB+C+arrow(c(t=r),c(t=r)+vector(num(tnb(c)
[0](t=r))),
     color='red',width=2)+arrow(c(t=r),c(t=r)+
     vector(num(tnb(c)
[1](t=r))),color='green',width=2)+arrow(c(t=r),c(t=r)+
     vector(num(tnb(c)
```

```
[2](t=r))),color='black',width=2)+FXD+FYD+FZA)
AN=animate(A,aspect_ratio=1)
show(AN)
```



Exemplo 5:

Sexa $U = (0, 6\pi) \subset \mathbb{R}$, $f : U \to \mathbb{R}^3$ tal que $f(t) = [cos(t), sin(t), t] \quad \forall t \in U$.

Probar que $imf \in \mathcal{V}_1^\infty(\mathbb{R}^3)$.

Representar a curva e a súa tanxente en f(1).

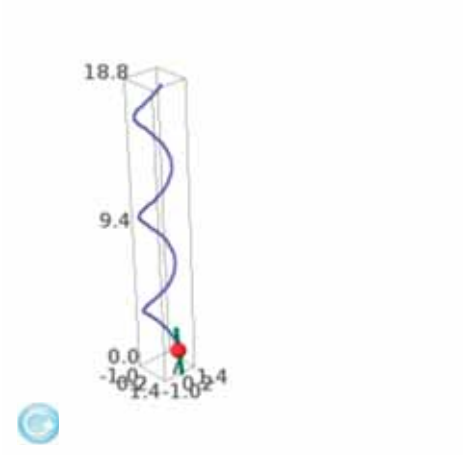
Presentar a animación da tanxente ao longo da curva.

Presentar a animación de Frenet ao longo da curva.

```
U=(0,6*pi)
f(t)=[cos(t),sin(t),t]
print 'f(t)=',f(t)
Df=diff(f,t)

print 'Df(t)=',Df(t)
print 'rank Df(t)=',rank(matrix(Df(t)))
print 'Curva e tanxente en f(1):'
G1=parametric_plot3d(f,U,thickness=3)
G2=point(f(1), color='red',size=20)
VAT(p)=list(f(1)+p*Df(1))
G3=parametric_plot(VAT,(p,-1,1),color='green',thickness=5)
show(G1+G2+G3,figsize=[3,3,20])
```

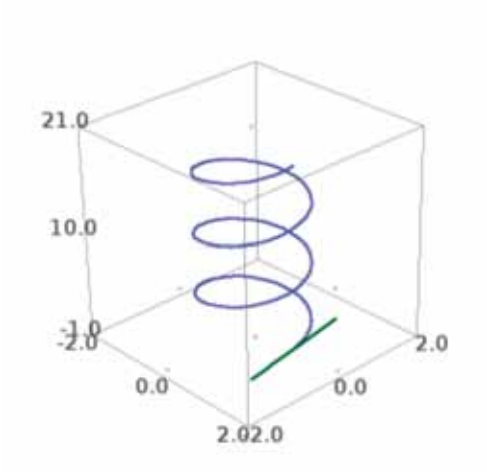
```
f(t)= (cos(t), sin(t), t)

Df(t)= (-sin(t), cos(t), 1)

rank Df(t)= 1
```

Curva e tanxente en f(1):




```
print 'Animación da tanxente:'
FXI=point3d([-2,0,0],color='white',figsize=[3,3,3])
FXD=point3d([2,0,0],color='white',figsize=[3,3,3])
FYI=point3d([0,-2,0],color='white',figsize=[3,3,3])
FYD=point3d([0,2,0],color='white',figsize=[3,3,3])
FZB=point3d([0,0,-1],color='white',figsize=[3,3,3])
FZA=point3d([0,0,21],color='white',figsize=[3,3,3])
R=[0,0.4..6*pi];AT=[]
for r in R:
    VAT(p)=list(f(r)+p*Df(r))
    G3=parametric_plot3d(VAT,
(p,-1,1),color='green',thickness=3,figsize=[2.5,2.5,2.5])
    AG=FXI+FYI+FZB+G1+G3+FXD+FYD+FZA
    AT.append(AG)
ANT=animate(AT); show(ANT)
```

Animación da tanxente:



```
print 'Animación de Frenet:'
A=[]
R=[0,.4,..,6*pi]
for r in R:

A.append(FXI+FYI+FZB+G1+arrow(f(t=r),f(t=r)+2*vector(num(tnb(f)
[0](t=r))),
    color='red',width=4)+arrow(f(t=r),f(t=r)+
    2*vector(num(tnb(f)
[1](t=r))),color='green',width=4)+arrow(f(t=r),f(t=r)+
    2*vector(num(tnb(f)
[2](t=r))),color='black',width=4)+FXD+FYD+FZA)
AN=animate(A,aspect_ratio=1)
show(AN)
```

Animación de Frenet:

Exemplo 6:

Sexa $U = (0, 2\pi) \times (-\frac{\pi}{2}, \frac{\pi}{2}) \subset \mathbb{R}^2$, $f : U \to \mathbb{R}^3$ tal que

$$f(u, v) = [cos(u)cos(v), sin(u)cos(v), sin(v)] \quad \forall (u, v) \in U.$$

Probar que $f(U)$ é unha variedade bidimensional.

Representar a superficie co seu plano tanxente e o seu vector normal en $f(\frac{\pi}{4}, \frac{\pi}{6})$.

Presentar a animación do plano tanxente e o vector normal ao longo da curva $z = \frac{1}{2}$.

Presentar a animación do plano tanxente e o vector normal ao longo da curva v=cos(u).

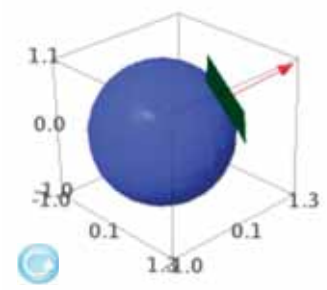
```
f(u,v)=[cos(u)*cos(v),sin(u)*cos(v),sin(v)]
print 'f(u,v)=',f(u,v)
X=vector([u,v])
```

```
Df(u,v)=EV(jacobian(f,X),X)
print 'Df(u,v)='
show(Df(u,v))
print 'rank Df(X)=',rank(EV(jacobian(f,X),X))
print 'Superficie con plano tanxente e vector normal en
f(pi/4.pi/6):'
G1=parametric_plot3d(f(u,v),(u,0,2*pi),(v,-pi/2,pi
/2),aspect_ratio=1)
TM=EV(jacobian(f,X),[pi/4,pi/6])
VAT(p,q)=list(f(pi/4,pi/6)+p*TM.column(0)+q*TM.column(1))
G2=parametric_plot3d(VAT(p,q),(p,-1/2,1/2),(q,-
1/2,1/2),color='green')
N=TM.column(0).cross_product(TM.column(1))
/norma(TM.column(0).cross_product(TM.column(1)))
G3=arrow(f(pi/4,pi/6),f(pi/4,pi/6)+N,color='red')
show(G1+G2+G3,aspect_ratio=1,figsize=[2.2,2.2,2.2])
```

```
f(u,v)= (cos(u)*cos(v), cos(v)*sin(u), sin(v))
Df(u,v)=
```

$$\begin{pmatrix} -\cos(v)\sin(u) & -\cos(u)\sin(v) \\ \cos(u)\cos(v) & -\sin(u)\sin(v) \\ 0 & \cos(v) \end{pmatrix}$$

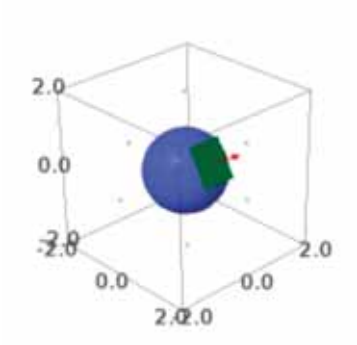
```
rank Df(X)= 2
Superficie con plano tanxente e vector normal en f(pi/4.pi/6)
```




```
print 'Animación do plano tanxente e vector normal na curva
z=1/2:'
FXI=point3d([-2,0,0],color='white',figsize=[2.2,2.2,2.2])
FXD=point3d([2,0,0],color='white',figsize=[2.2,2.2,2.2])
FYI=point3d([0,-2,0],color='white',figsize=[2.2,2.2,2.2])
FYD=point3d([0,2,0],color='white',figsize=[2.2,2.2,2.2])
FZB=point3d([0,0,-2],color='white',figsize=[2.2,2.2,2.2])
FZA=point3d([0,0,2],color='white',figsize=[2.2,2.2,2.2])
P=[0,.1,..,2*pi]
A=[]
for u in P:
    TM=EV(jacobian(f,X),[u,pi/6])
```

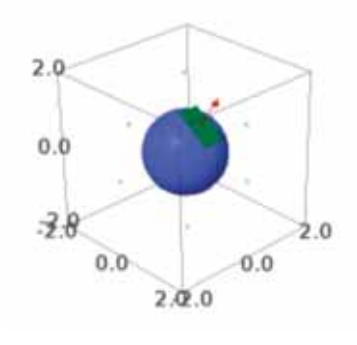
```
    VAT(p,q)=list(f(u,pi/6)+p*TM.column(0)+q*TM.column(1))
    G3=parametric_plot3d(VAT(p,q),(p,-1/2,1/2),(q,-
1/2,1/2),color='green',figsize=[2.2,2.2,2.2])
    N=TM.column(0).cross_product(TM.column(1))
    N=N/norma(N)
    G4=arrow(f(u,pi/6),f(u,pi/6)+N,color='red')
    A.append(FXI+FYI+FZB+G1+G3+G4+FXD+FYD+FZA)
AN=animate(A,aspect_ratio=1)
show(AN)
```

Animación do plano tanxente e vector normal na curva z=1/2:




```
print 'Animación do plano tanxente e vector normal na curva
v=cos(u):'
P=[0,.1,..2*pi]; B=[]
for u in P:
    v=cos(u)
    TM=EV(jacobian(f,X),[u,v])
    VAT(p,q)=list(f(u,v)+p*TM.column(0)+q*TM.column(1))
    G3=parametric_plot3d(VAT(p,q),(p,-1/2,1/2),(q,-
1/2,1/2),color='green',figsize=[2.2,2.2,2.2])
    N=TM.column(0).cross_product(TM.column(1))
    N=N/norma(N)
    G4=arrow(f(u,v),f(u,v)+N,color='red')
    B.append(FXI+FYI+FZB+G1+G3+G4+FXD+FYD+FZA)
BN=animate(B,aspect_ratio=1)
show(BN)
```

Animación do plano tanxente e vector normal na curva v=cos(u)



Exemplo 7:

Construír unha banda de Möebius sobre a circunferencia de centro (0,0,0) e radio 3 do plano $z = 0$.

Probar que é unha variedade diferenciable 2-dimensional.

Representala co seu plano tanxente e vector normal no punto (3,0,0).

Presentar a animación do plano tanxente e vector normal ao longo da circunferencia mencionada.

```
c=circunferencia([0,0,0],3,[0,0,1])
F=tnb(c)
b(t,v)=TS(list(c+v*(F[1]*cos(t/2)+F[2]*sin(t/2))))
print 'b(t,v)='
print b(t,v)
print
X=vector([t,v])
Db(t,v)=EV(jacobian(b,X),X)
print 'Db(t,v)='
show(Db(t,v))
print
print 'rank Db(X)=',rank(EV(jacobian(b,X),X))
print
print 'Banda con plano tanxente e vector normal en b(0,0):'
G1=parametric_plot3d(b(t,v),(t,0,2*pi),(v,-1,1),aspect_ratio=1)
TM=EV(jacobian(b,X),[0,0])

VAT(p,q)=list(b(0,0)+p*TM.column(0)+q*TM.column(1))
G2=parametric_plot3d(VAT(p,q),(p,-1,1),(q,-1,1),color='green')

N=TM.column(0).cross_product(TM.column(1))/norma(TM.column(0).
    cross_product(TM.column(1)))
G3=arrow(b(0,0),b(0,0)+N,color='red',width=2)
show(G1+G2+G3,aspect_ratio=1,figsize=[2.2,2.2,2.2])
```

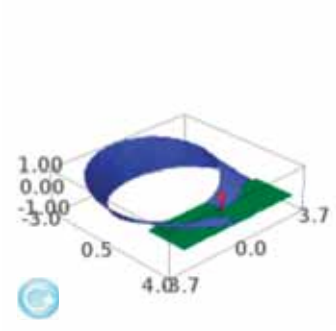
```
b(t,v)=
(-v*cos(1/2*t)*cos(t) + 3*cos(t), -v*cos(1/2*t)*sin(t) + 3*si
v*sin(1/2*t))
```

Db(t,v)=

$$\begin{pmatrix} \frac{1}{2}\,v\cos(t)\sin\left(\frac{1}{2}\,t\right)+v\cos\left(\frac{1}{2}\,t\right)\sin(t)-3\,\sin(t) & -\cos\left(\frac{1}{2}\,t\right)\cos(t) \\ -v\cos\left(\frac{1}{2}\,t\right)\cos(t)+\frac{1}{2}\,v\sin\left(\frac{1}{2}\,t\right)\sin(t)+3\,\cos(t) & -\cos\left(\frac{1}{2}\,t\right)\sin(t) \\ \frac{1}{2}\,v\cos\left(\frac{1}{2}\,t\right) & \sin\left(\frac{1}{2}\,t\right) \end{pmatrix}$$

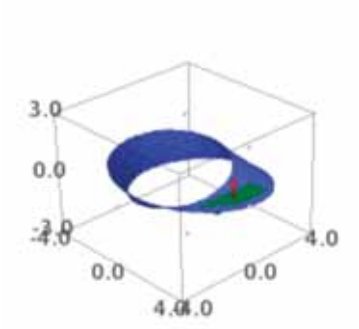
```
rank Db(X)= 2
```

Banda con plano tanxente e vector normal en b(0,0):




```
print 'Animación do plano tanxente e vector normal'
print 'na circunferencia z=0:'
FXI=point3d([-4,0,0],color='white',figsize=[2.2,2.2,2.2])
FXD=point3d([4,0,0],color='white',figsize=[2.2,2.2,2.2])
FYI=point3d([0,-4,0],color='white',figsize=[2.2,2.2,2.2])
FYD=point3d([0,4,0],color='white',figsize=[2.2,2.2,2.2])
FZB=point3d([0,0,-3],color='white',figsize=[2.2,2.2,2.2])
FZA=point3d([0,0,3],color='white',figsize=[2.2,2.2,2.2])
A=[]
P=[0,.1,..,2*pi]+[2*pi]
for u in P:
    TM=EV(jacobian(b,X),[u,0])
    VAT(p,q)=list(b(u,0)+p*TM.column(0)+q*TM.column(1))
    G3=parametric_plot3d(VAT(p,q),(p,-1/2,1/2),(q,-
1/2,1/2),color='green',figsize=[2.2,2.2,2.2])
    N=TM.column(0).cross_product(TM.column(1))
    N=N/norma(N)
    G4=arrow(b(u,0),b(u,0)+N,color='red',width=2)
    A.append(FXI+FYI+FZB+G1+G3+G4+FXD+FYD+FZA)
AN=animate(A,aspect_ratio=1)
show(AN)
```

Animación do plano tanxente e vector normal
na circunferencia z=0:



Tralos exemplos 6 e 7 podemos dicir que cando o vector normal se move de forma continua ao longo de calquera curva pechada nunha superficie, a superficie é orientable e cando non, como sucede na banda de Möbius, a superficie non é orientable.

Exemplo 8:

Sexa $U=(\pi,2\pi)\times(-\frac{\pi}{2},\frac{\pi}{2})\times(0,1)\subset\mathbb{R}^3$, $f:U\to\mathbb{R}^3$ tal que $f(t)=[rcos(u)cos(v),rcos(u)sin(v),rsin(u)]\quad\forall(u,v,r)\in U.$ Probar que $imf\in\mathcal{V}_3^\infty(\mathbb{R}^3)$ e representala.

```
f(u,v,r)=[r*sin(u)*cos(v),r*sin(u)*sin(v),r*cos(u)]
print 'f='; show(f)
X=vector([u,v,r])
Df(u,v,r)=EV(jacobian(f,X),X)
print 'Df=';show(Df)
print 'rank Df(X)=',rank(EV(jacobian(f,X),X))
print
print 'Representación gráfica'
R=[0,10^(-2)..1]
G=[]
for t in R:
    G.append(parametric_plot3d(f(u,v,t),(u,pi,2*pi),(v,-pi/2,pi
/2)))
show(sum(G),aspect_ratio=1,figsize=[2.2,2.2,2.2])
```

f=

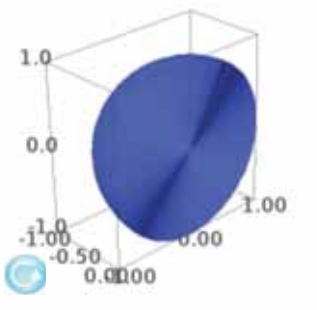
$$(u,v,r)\mapsto\left(r\cos(v)\sin(u),\ r\sin(u)\sin(v),\ r\cos(u)\right)$$

Df=

$$(u,v,r)\mapsto\begin{pmatrix}r\cos(u)\cos(v) & -r\sin(u)\sin(v) & \cos(v)\sin(u)\\ r\cos(u)\sin(v) & r\cos(v)\sin(u) & \sin(u)\sin(v)\\ -r\sin(u) & 0 & \cos(u)\end{pmatrix}$$

rank Df(X)= 3

Representación gráfica



Os conceptos de difeomorfismo global e local e os teoremas TFIN e TFIM, permitirannos unha definición equivalente de variedade:

Sexan X e Y dous espazos de Banach, A un aberto de X e $f : A \to Y$ unha función de clase $\mathcal{C}^1$. Dicimos que f é un difeomorfismo global en A, $f \in \mathcal{D}(A)$, se $B = f(A)$ é aberto en Y e existe unha $g : B \to X$ de clase $\mathcal{C}^1$ tal que $g \circ f = I_A$ e $f \circ g = I_B$.

Dicimos que f é un difeomorfismo local en x, $f \in \mathcal{D}(x)$, se existe un aberto $E \subset A$ tal que $x \in E$ e $f|_E \in \mathcal{D}(E)$.

Teorema da función inversa (FIN):

Nas condicións anteriores $f \in \mathcal{D}(x) \quad \Leftrightarrow \quad Df(x) : X \to Y$ é inversible e acotada.

Teorema da función implícita (FIM):
Sexan X_1 , X_2 , Y espazos de Banach e sexa $A \subset X_1 \times X_2$ un aberto. Se $F : A \to Y$ é de clase $\mathcal{C}^k$ e $D_2F(\mathbf{x}_0) : X_2 \to Y$ é inversible e acotada, cúmprese que
• Existe un aberto $W \subset A$ tal que $\mathbf{x}_0 = (x_{01}, x_{02}) \in W$.
• Existe un aberto $U_1 \subset X_1$ tal que $x_{01} \in U_1$.
• Existe unha $g : U_1 \to X_2$ de clase $\mathcal{C}^k$ con $Dg(x_{01}) = -(D_2F(\mathbf{x}_0))^{-1} \circ D_1F(\mathbf{x}_0)$ tal que

$$\{\mathbf{x} \in W \,|\, F(\mathbf{x}) = F(\mathbf{x}_0)\} = \{(x_1, g(x_1)) \,|\, x_1 \in U_1\}.$$

Definición equivalente de variedade diferenciable:

$M \in \mathcal{V}_q^k(\mathbb{R}^n) \quad \Leftrightarrow \quad M \subset \mathbb{R}^n$ e $\forall \mathbf{x} \in M$ existe un par (A, F) tal que
• A é un aberto de $\mathbb{R}^n$ que contén a $\mathbf{x}$.
• $F : A \to \mathbb{R}^{n-q}$ é de clase $\mathcal{C}^k$ e $rank\, DF(\mathbf{x}) = n - q \quad \forall \mathbf{x} \in A$.
• $A \cap M = \{\mathbf{x} \in A | F(\mathbf{x}) = 0\}$

Demostración:

$\Rightarrow$)Para todo $\mathbf{x} \in M$ existe un aberto $U \subset \mathbb{R}^q$ e unha $f : U \to \mathbb{R}^n$ que, posiblemente cunha reordenación de variables, cumpre $f(\mathbf{u}) = \mathbf{x}$ e $\left|\frac{\partial(f_0 \cdots f_{q-1})}{\partial(u_0 \cdots u_{q-1})}(\mathbf{u})\right| \neq 0$. Segundo o TFIN a restricción de f ás q primeras coordenadas, $\bar{f} : U \to \mathbb{R}^q$, cumpre que $\bar{f} \in \mathcal{D}(\mathbf{u})$. Así, existe un aberto U_0 tal que $\mathbf{u} \in U_0 \subset U$ e outro aberto $V_0 = \bar{f}(U_0)$ onde $\bar{f}$ ten inversa $\phi : V_0 \to U_0$.

A restricción de f a U_0 é inxectiva e, xa que logo, $f : U_0 \to f(U_0)$ é bixectiva. A súa inversa é a composición $f(U_0) \to V_0 \to U_0$ que leva $(x_0, \cdots, x_{n-1}) \mapsto (x_0, \cdots, x_{q-1}) \mapsto \phi(x_0, \cdots, x_{q-1})$ e, xa que logo, os puntos de $f(U_0)$ admiten as ecuacións explícitas

$$\begin{cases} x_q = f_q \circ \phi(x_0, \cdots, x_{q-1}) \\ \vdots \\ x_{n-1} = f_{n-1} \circ \phi(x_0, \cdots, x_{q-1}) \end{cases}$$

Como $f(U_0)$ é contorna aberta de $\mathbf{x}$ en M, existirá un aberto $A \subset \mathbb{R}^n$ tal que $f(U_0) = A \cap M$. Ademais, $\mathbf{x} \in A$ pois $\mathbf{x} \in f(U_0)$.
Se definimos $F : A \to \mathbb{R}^{n-q}$ tal que $F(\mathbf{x}) = (x_q - f_q \circ \phi(x_0, \cdots, x_{q-1}), \cdots, x_{n-1} - f_{n-1} \circ \phi(x_0, \cdots, x_{q-1}))$ é claro que $F \in \mathcal{C}^k$ igual que f e $rank\, DF(\mathbf{x}) = n - q$. Ademais,

$$\{\mathbf{x} \in A \,|\, F(\mathbf{x}) = 0\} = f(U_0) = A \cap M$$

$\Leftarrow$) Se para $\mathbf{x}_0 \in M$ existe $F : A \subset \mathbb{R}^{q+p} \to R^p$ de clase $\mathcal{C}^k$ tal que $F(\mathbf{x}_0) = \mathbf{0}$ e $\frac{\partial(F_0, \cdots, F_{p-1})}{\partial(x_q, \cdots, x_{q+p-1})}(\mathbf{x}_0) \neq 0$, o TFI asegura a existencia dunha contorna aberta W de $\mathbf{x}_0$ onde o lugar dos puntos $\mathbf{x}$ que satisfán a ecuación $F(\mathbf{x}) = \mathbf{0}$, é a imaxe dun aberto $U_1 \subset \mathbb{R}^q$ por unha función $f \in \mathcal{C}^k$ cuxa diferencial $Df(u) : \mathbb{R}^q \to \mathbb{R}^{q+p}$ ten rango q $\forall u \in U_1$. En efecto:

Se W, U_1 e g son exactamente como no FIM e definimos $f : U_1 \to \mathbb{R}^{q+p}$ de modo que $u \mapsto (u, g(u))$, é claro que $Df(u) : \mathbb{R}^q \to \mathbb{R}^{q+p}$ é a aplicación lineal asociada á matriz $\begin{pmatrix} U_q \\ Dg(u) \end{pmatrix}$ que ten rango q. En consecuencia, $M \in \mathcal{V}_q^k(\mathbb{R}^n)$.

Observación:

Se M é unha variedade diferenciable q-dimensional de clase $\mathcal{C}^k$ en $\mathbb{R}^n$ dada implicitamente pola ecuación $F(\mathbf{x}) = \mathbf{0}$ e parametricamente pola carta local $f : U \to \mathbb{R}^n$, a función $F \circ f : U \to \mathbb{R}^{n-q}$ toma constantemente o valor $\mathbf{0}$ e, xa que logo, se $\mathbf{x} = f(\mathbf{u})$, temos que $DF(\mathbf{x}) \circ Df(\mathbf{u}) : \mathbb{R}^q \to \mathbb{R}^{n-q}$ é a aplicación lineal nula. En consecuencia, $imDf(\mathbf{u}) \subset \ker DF(\mathbf{x})$ e, por ser ambos subespacios de $\mathbb{R}^n$ da mesma dimensión, deben coincidir. Así, $T_{\mathbf{x}}(M) = imDf(\mathbf{u}) = \ker DF(\mathbf{x})$.

Exemplo 9:

Atopar a ecuación implícita da variedade do exemplo 1 utilizando a función **complort**(E1):

```
E1
X=vector(listasim('u',3))
Y=vector(listasim('x',4))
f(u0,u1,u2)=list(E1*X)
COE1=complort(E1)
F(x0,x1,x2,x3,x4)=list(Y*COE1)
```

```
print 'Ecuacións parametricas:'
show(f(u0,u1,u2))
print 'Ecuacións implícitas:'
show([F(x0,x1,x2,x3,x4),RS(list(EV(F,f(u0,u1,u2))))])
```

Ecuacións parametricas:

$$\left(-2\,u_0+u_1-2\,u_2,\ -\frac{1}{2}\,u_0,\ u_1-2\,u_2,\ 2\,u_0-u_1-u_2\right)$$

Ecuacións implícitas:

$$\left[\left(\frac{1}{3}\sqrt{\frac{1}{2}}\,x_0-\frac{4}{3}\sqrt{\frac{1}{2}}\,x_1-\frac{1}{3}\sqrt{\frac{1}{2}}\,x_2\right),[0]\right]$$

Exemplo 10:

Dada a variedade lineal en $\mathbb{R}^5$ de ecuacións implícitas:

$$\begin{cases} x_0+2x_1-x_3+x_4 &= 0\\ 2x_0-x_1+3x_2-x_4 &= 0\\ 3x_1-2x_3+x_4 &= 0 \end{cases}$$

achar unha representación paramétrica.

```
U=vector(listasim('u',2))
B=matrix([[1,2,0,-1,1],[2,-1,3,0,-1],[0,3,0,-2,1]])
OB=complort(B.transpose())
print OB*U
X=vector(listasim('x',5))
F(x0,x1,x2,x3,x4)=list(B*X)
f(u0,u1)=list(OB*U)
show(RS(list(EV(F,f(u0,u1,u2)))))
```

```
(3*sqrt(1/67)*u0, -sqrt(1/67)*u0 + sqrt(1/3)*u1, -4*sqrt(1/67)*u0,
-4*sqrt(1/67)*u0 + sqrt(1/3)*u1, -5*sqrt(1/67)*u0 - sqrt(1/3)*u1)

[0,0,0]
```

Exemplo 11:

Debuxar a intersección do cono oblicuo de base $\begin{cases} x^2+y^2=25\\ z=0 \end{cases}$ e vértice (2,2,10) co plano $z=k$ para calquera $k\in(0,10)$.

Solución:

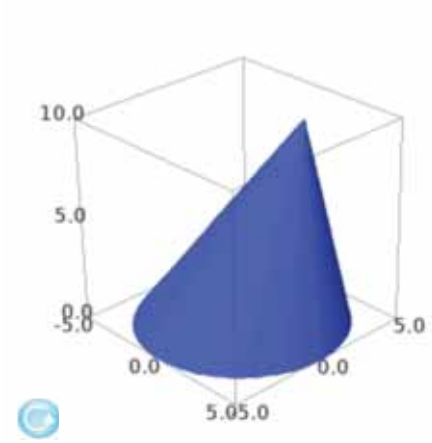
O cono sólido é a imaxe do aberto $U=(0,5)\times(0,2\pi)\times(0,1)$ pola función

$f : U \to \mathbb{R}^3$ tal que

$$f(r,t,h) = [(1-h)r\cos(t) + 2h, (1-h)r\sin(t) + 2h, 10h]$$

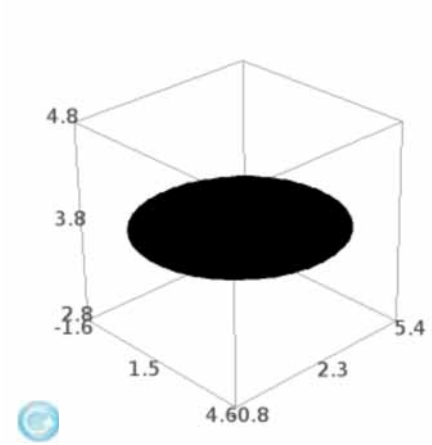
```
f(r,t,h)=[(1-h)*r*cos(t)+2*h,(1-h)*r*sin(t)+2*h,10*h]

print 'A superficie lateral do cono é'
s(t,h)=[(1-h)*5*cos(t)+2*h,(1-h)*5*sin(t)+2*h,10*h]
parametric_plot(s(t,h),(t,0,2*pi),(h,0,1),figsize=[3,3,3])
```

```
A superficie lateral do cono é
```




```
print 'O corte co plano z=3.8 é'
k=3.8
c(r,t)=[(1-k/10)*r*cos(t)+2*k/5,(1-k/10)*r*sin(t)+3*k/5,k]
parametric_plot3d(c(r,t),(r,0,5),
(t,0,2*pi),color='black',figsize=[3,3,3])
```

```
O corte co plano z=3.8 é
```



3.1.15. EXERCICIOS

CONXUNTOS

Exercicio 1:

Calcular a suma de todos os naturais menores ou iguales que 11000.

```
sum(range(11001))
```

```
60505500
```

Exercicio 2:

Calcular a suma dos números primos menores que 1000.

```
sum([p for p in range(1000) if p in Primes()])
```

```
76127
```

Exercicio 3:

Programar o cálculo do primo máis próximo a un natural dado:

```
def primoprox(n):
    m=max([p for p in range(n) if p in Primes()])
    M=max([p for p in range(2*n-m+1) if p in Primes()])
    if n-m<M-n:
        return m
    elif n-m>M-n:
        return M
    else:
        return m, M
```

```
primoprox(207)
```

```
211
```

Exercicio 4:

a) Cantos números de cinco cifras distintas se poden formar coas cifras 1,2,3,4,5?

b) Canto suman todos eles?

```
L=[1,2,3,4,5];P=Permutations(L).list()
N=[p[0]*10000+p[1]*1000+p[2]*100+p[3]*10+p[4] for p in P]
S=sum(N)
print 'a)', len(N)
print 'b)', sum(N)
```

```
a) 120
b) 3999960
```

Exercicio 5:

De cantos xeitos se poden repartir 15 libros diferentes entre tres estudantes de modo que a cada un lle toquen 5 libros?

```
len(Combinations(15,5).list())*len(Combinations(10,5).list())
```

```
756756
```

Exercicio 6:

Os seguintes datos proceden dun ensaio de tracción realizado cunha probeta cilíndrica de 13 mm de diámetro inicial. Logo da rotura, a lonxitude calibrada foi de 76,5 mm e o seu diámetro 9,6 mm.

Carga (kN)	Lonxitude calibrada (mm)
0	50,8
13,35	50,842
26,7	50,884
33,36	50,905
40	51,028
46,7	51,816
53,4	57,404
55,15	63,5
50,7	76,5

a) Representar os datos e calcular o límite elástico e a resistencia a tracción.

b) Achar o módulo de Young.

c) Achar a porcentaxe de alongamento ata rotura e a porcentaxe de estricción.

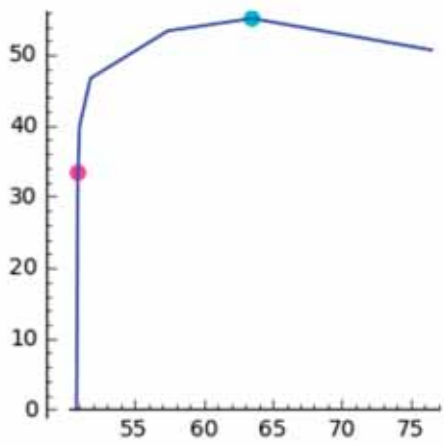
Solución:

a) O punto maxenta representa o límite elástico Le. A partir del comenza a deformación plástica. Superado o Le, ao deixar de aplicar un esforzo, a recuperación non é total senón paralela á recta de pendente constante.

O punto cian representa a resistencia a tracción Rm. A partir deste punto a deformación xa non é uniforme e comenza a producirse na probeta o fenómeno de estricción.

```
X=[50.8,50.842,50.884,50.905,51.028,51.816,57.404,63.50,76.5]
Y=[0,13.35,26.7,33.36,40,46.7,53.4,55.15,50.7]
grafica=line(zip(X,Y))

LP=[round((sig(a,X)-a)/(sig(b,Y)-b),3) for (a,b) in zip(X,Y)]
```

```
n=veces(LP[0],LP[0:-1])
PuntoLe=(X[n],Y[n])
Le=points(PuntoLe,hue=.9, pointsize=50)

PuntoRm=(X[Y.index(max(Y))],Y[Y.index(max(Y))])
Rm=points(PuntoRm,hue=.5, pointsize=50)

show(grafica+Le+Rm, figsize=[3,3])
```



b) Para determinar o módulo de Young E usamos a lei de Hooke: $\sigma = E\varepsilon$ sendo $\sigma = \frac{F}{S}$ e $\varepsilon = \frac{\Delta L}{L}$

```
sigma=(33360/(3.1416*(6.5)^2))
epsilon=(50.905-50.8)/50.8
E=sigma/epsilon
print 'E=',E
```

```
E= 121596.998132223
```

c) As porcentaxes de alongamento e de estricción defínense como segue:

$$PA = 100\frac{Lf - Li}{Li} \qquad PE = 100\frac{Si - Sf}{Si}$$

```
Li=50.8
Lf=76.5
Si=132.7
Sf=72.38
PA=100*(Lf-Li)/Li
print'PA=',PA
PE=100*(Si-Sf)/Si
print'PE=',PE
```

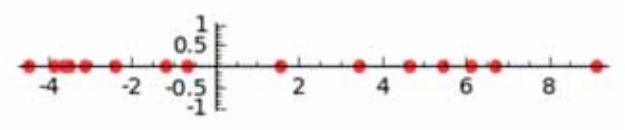
```
PA= 50.5905511811024
PE= 45.4559155990957
```

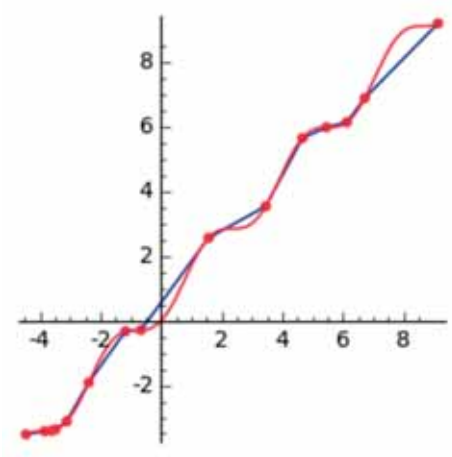
LISTAS REAIS. SUCESIÓNS RECURRENTES

Exercicio 7:

Obter unha lista aleatoria L de 15 números reais no intervalo [-5,10]. Calcular os valores que toma a función x+sin2x en L e representar a poligonal correspondente. Comparala coa representación da función no intervalo [min(L),max(L)].

```
L=listareales(-5,10,15)
M=copy(L)
M.sort()
show(plce(M),figsize=[4,3])
f(x)=x+(sin(x))^2
F=map(f,M)
P=[[M[i],F[i]] for i in range(15)]
DP=point2d(P,color='red',pointsize=20)
POL=line(P)
CUR=plot(f,(x,min(L),max(L)),color='red')
show(DP+POL+CUR, figsize=[3,3])
```





Exercicio 8:

Achar a suma dos términos da sucesión de Fibonacci menores que 10^6.

```
print 'a) Utilizando break:'
M=[]
for n in NN:
    Fn=Fibonacci(n)
    if Fn<10^6:
        M.append(Fn)
```

```
    else:
        break
sum(M)
```

```
a) Utilizando break:
2178308
```

```
print 'b) Utilizando while:'
M=[]
n=0
Fn=1
while Fn<10^6:
    M.append(Fn)
    n=n+1
    Fn=Fibonacci(n)
sum(M)
```

```
b) Utilizando while:
2178308
```

Exercicio 9:

Elaborar unha función **TG**(n,CI) que nos dea o termo n-ésimo da lista X de condicións iniciais X[0:2]=CI e relación de recurrencia

$$X[n+2]+X[n+1]-2X[n]=1 \quad \forall n \in N.$$

```
def TG(n,CI):
    X=CI
    for k in range(n):
        X.append(2*X[-2]+1-X[-1])
    return X[n]
```

```
CI=[0,1]
TG(23,CI)
```

```
1864143
```

Exercicio 10:

Elaborar unha función **TG**2(n,CI1,CI2) que nos dea o termo n-ésimo das listas X e Y de condicións iniciais X[0]=2,X[1]=3, Y[0]=1,Y[1]=-2 e relacións de recurrencia

$$\begin{cases} X[n+1]= & \frac{X[n]+2Y[n-1]}{3}+\sqrt{|X[n]Y[n-1]|} \\ Y[n+1]= & \frac{Y[n]-X[n-1]}{2}+\sqrt{|Y[n]X[n-1]|} \end{cases}$$

Calcular X[20], Y[20] e estimar os límites de X e Y.

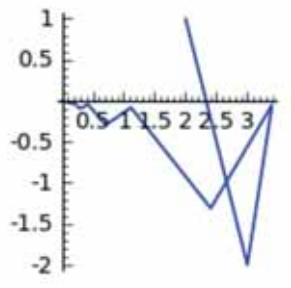
```
def TG2(n,CI1,CI2):
```

```
    X=CI1
    Y=CI2
    for k in range(n):

X.append(((X[-1]+2*Y[-2])/3+sqrt(abs(X[-1]*Y[-2]))).n())
        Y.append(((Y[-1]-X[-2])/2+sqrt(abs(Y[-1]*X[-2]))).n())
    return X[n],Y[n]
```

```
CI1=[2,3]
CI2=[1,-2]
TG2(20,CI1,CI2)
```

```
    (0.00191410132458635, -0.000405481358022525)
```

```
line([TG2(n,CI1,CI2) for n in range(100)],figsize=[2,2])
```



LISTAS COMPLEXAS

Exercicio 11:

Achar o triángulo de área máxima entre todos os que teñen os seus vértices na lista complexa

$$[-5*I-14,-7*I-16,-12*I-4,15*I+7,13*I+2,14*I-12]$$

```
L=[-5*I - 14, -7*I - 16, -12*I - 4, 15*I + 7, 13*I + 2, 14*I -
12]
C=Combinations(L,3).list()
A=[heron(a) for a in C]
M=maximos(A)
print 'Os seus vertices son', C[M[1][0]]
print 'e a súa área vale', M[0]
```

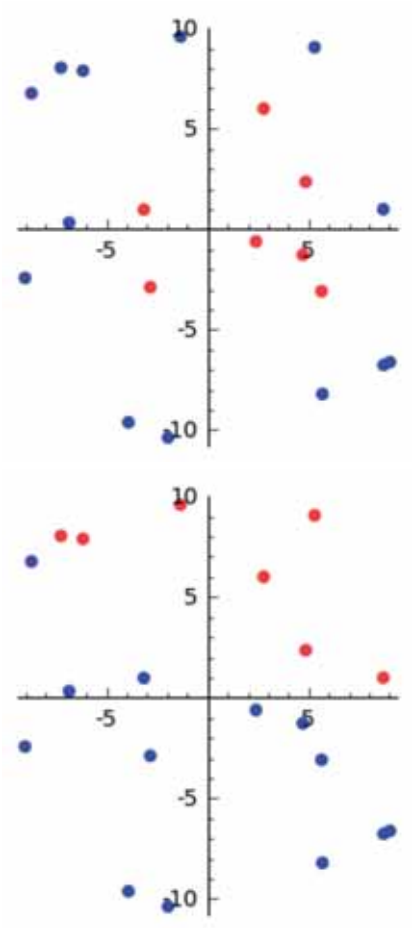
```
    Os seus vertices son [-12*I - 4, 15*I + 7, 14*I - 12]
    e a súa área vale 251.000000000000
```

Exercicio 12:

Reordenar unha lista aleatoria L de 20 complexos de módulo menor que 15 segundo os seus módulos e segundo os seus argumentos.

```
L=listacomplejos(sqrt(15^2/2),20)
ML=[[z,abs(z)] for z in L]
ML1=[a[1] for a in ML]
ML1.sort()
MLL=[]
for m in ML1:
    for a in L:
        if [a,m] in ML:
            MLL.append(a)
print Roundc(MLL,3)
A=plce(MLL[0:7])
B=plce(MLL[7:],.7)
show(A+B,figsize=[3,3])

AL=[[z,argumento(z)] for z in L]
AL1=[a[1] for a in AL]
AL1.sort()
ALL=[]
for m in AL1:
    for a in L:
        if [a,m] in AL:
            ALL.append(a)
print
print Roundc(ALL,3)
A=plce(ALL[0:7])
B=plce(ALL[7:],.7)
show(A+B,figsize=[3,3])
```

```
[2.4 - 0.614*I, -3.156 + 0.975*I, -2.838 - 2.887*I, 4.717 - 1
4.866 + 2.356*I, 5.647 - 3.087*I, 2.781 + 6.002*I, -6.86 + 0.
8.707 + 0.99*I, -9.043 - 2.43*I, -1.341 + 9.585*I, 5.69 - 8.2
-6.173 + 7.884*I, -3.921 - 9.616*I, 5.302 + 9.054*I, -1.958 -
10.373*I, -7.265 + 8.039*I, 8.724 - 6.764*I, -8.728 + 6.765*I,
- 6.619*I]

[8.707 + 0.99*I, 4.866 + 2.356*I, 5.302 + 9.054*I, 2.781 + 6.
-1.341 + 9.585*I, -6.173 + 7.884*I, -7.265 + 8.039*I, -8.728 +
6.765*I, -3.156 + 0.975*I, -6.86 + 0.322*I, -9.043 - 2.43*I, -
- 2.887*I, -3.921 - 9.616*I, -1.958 - 10.373*I, 5.69 - 8.208*
8.724 - 6.764*I, 9.032 - 6.619*I, 5.647 - 3.087*I, 4.717 - 1.
2.4 - 0.614*I]
```



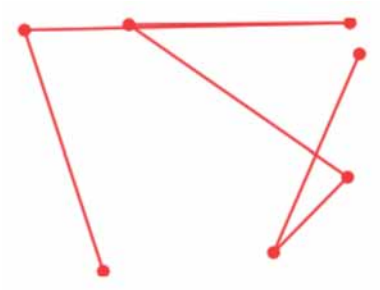
Exercicio 13:

Programar unha función **spin**(L) que nos dea a suma dos ángulos de xiro necesarios para percorrer a poligonal determinada pola lista de complexos L.

```
def spin(L):
    n=len(L)
    return sum([angle(sig(a,L)-a,sig(sig(a,L),L)-sig(a,L)) for
a in L[0:n-2]]).n()
```

```
L=listacomplejos(20,7)
show(poligonal(L),figsize=[2.5,2.5])
spin(L)
```

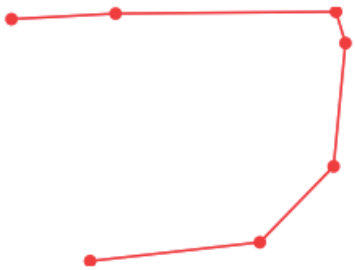
```
12.0252876647938
```



Exercicio 14:

Dada a lista de complexos L determinar a ordenación que produza a poligonal de mínima lonxitude e mínimo spin.

```
PL=Permutations(L).list()
MPL=[[round(lon(a)[0],5),round(spin(a),5)] for a in PL]
m=sorted(MPL)[0]
show(poligonal(PL[MPL.index(m)]),figsize=[2.5,2.5])
```



LISTAS ALFANUMÉRICAS

Exercicio 15:

1) Obter unha lista cualificada de 20 españois que teñan entre 18 e 55 anos.

2) Obter a sublista das mulleres co seu DNI.

3) Obter a sublista do ámbito tecnolóxico coa súa titulación.

4) Obter a lista completa de apelidos e nome ordenada alfabéticamente.

5) Obter a lista completa de apelidos e nome ordenada por idades.

```
PQ=PoblacionQ(2016,18,55,20)
PA=[a[0]+[a[1]] for a in PQ]
Publilista(PA)
```

```
[1, [Aguirre, Indurain, Bartolo, M]]
[2, [Lopard, Bonan, Bartolo, M]]
[3, [Pitarch, Linares, Monica, F]]
```

```
[4, [Pujol, Hinojosa, Zoraida, F]]
[5, [Amat, Cespon, Sixto, M]]
[6, [Cespedes, Benjumea, Nicanora, F]]
[7, [Taura, Zaragoza, Julio, M]]
[8, [Izquierdo, Rodriguez, Victor, M]]
[9, [Armada, Zapatero, Tomas, M]]
[10, [Pelaez, Varela, Nicanora, F]]
[11, [Lopez, Carballido, Bernardo, M]]
[12, [Garcia, Diaz, Yago, M]]
[13, [Artigas, Gil, Ines, F]]
[14, [Armada, Benitez, Arturo, M]]
[15, [Perez, Cespedes, Ursula, F]]
[16, [Calero, Lopard, Bernardo, M]]
[17, [Arias, Latices, Victor, M]]
[18, [Leon, Cerrudo, Matilde, F]]
[19, [Gimpera, Aguirre, Narciso, M]]
[20, [Perez, Gea, Olegario, M]]
```

```
print 'Mulleres con DNI'
print' '
Publilista([[a[0],a[3]] for a in PQ if a[1]==F])
print' '
print 'Ámbito tecnolóxico con titulación'
print' '
PQGT=[[a[0],a[4]] for a in PQ if a[5]==Tecnoloxico]
Publilista(PQGT)
print' '
print'Todos por orde alfabético'
print' '
PQO1=sorted([[str(a[0])]+[a] for a in PQ])
PQO=[a[1:][0] for a in PQO1]
Publilista([a[0][:-1] for a in PQO])
print' '
print'De maior a menor'
print' '
PQE1=sorted([[a[2][2]*10000+a[2][1]*100+a[2][0]]+[a] for a in
PQ])
PQE=[a[1:] for a in PQE1]
Publilista([a[0][0] for a in PQE])
```

```
Mulleres con DNI

[1, [[Pitarch, Linares, Monica], [44140804, D]]]
[2, [[Pujol, Hinojosa, Zoraida], [22909181, P]]]
[3, [[Cespedes, Benjumea, Nicanora], [45158673, J]]]
[4, [[Pelaez, Varela, Nicanora], [22135107, E]]]
[5, [[Artigas, Gil, Ines], [22556164, H]]]
[6, [[Perez, Cespedes, Ursula], [36933657, G]]]
[7, [[Leon, Cerrudo, Matilde], [47185439, L]]]

Ámbito tecnolóxico con titulación
```

```
[1, [[Lopard, Bonan, Bartolo], Secundarios]]
[2, [[Cespedes, Benjumea, Nicanora], Grado]]
[3, [[Artigas, Gil, Ines], FP1]]
[4, [[Leon, Cerrudo, Matilde], Primarios]]

Todos por orde alfabético

[1, [Aguirre, Indurain]]
[2, [Amat, Cespon]]
[3, [Arias, Latices]]
[4, [Armada, Benitez]]
[5, [Armada, Zapatero]]
[6, [Artigas, Gil]]
[7, [Calero, Lopard]]
[8, [Cespedes, Benjumea]]
[9, [Garcia, Diaz]]
[10, [Gimpera, Aguirre]]
[11, [Izquierdo, Rodriguez]]
[12, [Leon, Cerrudo]]
[13, [Lopard, Bonan]]
[14, [Lopez, Carballido]]
[15, [Pelaez, Varela]]
[16, [Perez, Cespedes]]
[17, [Perez, Gea]]
[18, [Pitarch, Linares]]
[19, [Pujol, Hinojosa]]
[20, [Taura, Zaragoza]]

De maior a menor

[1, [Artigas, Gil, Ines]]
[2, [Calero, Lopard, Bernardo]]
[3, [Leon, Cerrudo, Matilde]]
[4, [Amat, Cespon, Sixto]]
[5, [Lopez, Carballido, Bernardo]]
[6, [Armada, Zapatero, Tomas]]
[7, [Perez, Gea, Olegario]]
[8, [Aguirre, Indurain, Bartolo]]
[9, [Perez, Cespedes, Ursula]]
[10, [Arias, Latices, Victor]]
[11, [Taura, Zaragoza, Julio]]
[12, [Garcia, Diaz, Yago]]
[13, [Pelaez, Varela, Nicanora]]
[14, [Pitarch, Linares, Monica]]
[15, [Gimpera, Aguirre, Narciso]]
[16, [Pujol, Hinojosa, Zoraida]]
[17, [Lopard, Bonan, Bartolo]]
[18, [Izquierdo, Rodriguez, Victor]]
[19, [Armada, Benitez, Arturo]]
[20, [Cespedes, Benjumea, Nicanora]]
```

LISTAS GRÁFICAS

Exercicio 16:

Pintar a gráfica da función $f(x) = x^3 - 2x^2 + 3$ no intervalo [0,2], en cores diferentes segundo a ordenada e programar unha función **barracolor** que asigne un valor a cada cor.

Solución:

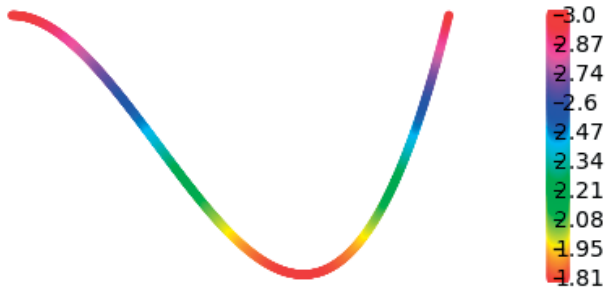
A función **barracolor** depende da función real f, do seu intervalo de definición R, das coordenadas $P = [P[0], P[1]]$ do punto inferior da barra, da lonxitude da barra L e da distancia d dos valores á barra:

```
def barracolor(f,R,P,L,d):
    C=[]
    for n in range(L*100):
        c=n/((L*100)-1)

C.append(point([P[0],P[1]+c*L],hue=c,pointsize=100,axes=False))
    TC=add(C)
    M=[]
    for n in range(10):
        c=n/9
        M.append(line([[-.02+P[0],P[1]+c*L],
[.02+P[0],P[1]+c*L]],color='black',aspect_ratio=1))
    TM=add(M)
    T=[R[0],R[0]+1/100,..,R[1]]
    Vf=[f(t) for t in T]
    M=max(Vf).n()
    m=min(Vf).n()
    V=[]
    for k in range(10):
        c=k/9
        v=m+(k*(M-m)/9)
        w=round(v,2)
        V.append(text(w,
(d+P[0],P[1]+c*L),color='black',fontsize=10))
    TV=add(V)
    return TC+TM+TV
```

Aplicándoa ao noso caso, temos

```
f(x)=x^3-2*x^2+3
X=[0,1/300..2]
N=len(X)
Y=map(f,X)
k=max(Y)-min(Y)
```

```
show(sum([point([X[i],Y[i]],hue=(Y[i]-min(Y))/k,pointsize=15)
for i in range(N)])+barracolor(f,[0,2],
[2.5,1.8],1.2,.1),figsize=[4,4])
```



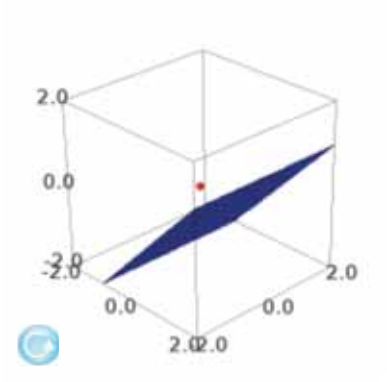
VECTORES E MATRICES

Exercicio 17:

Estudar cales dos seguintes conxuntos son subespacios de $\mathbb{R}^3$:
(a) U1 = $\{(x,y,z) \in R^3 | x - z = 1\}$.
(b) U2 = $\{(x,y,z) \in R^3 | x - z = y - 2z = 0\}$.
(c) U3 = $\{(x,y,z) \in R^3 | |y| = |z|\}$.

```
print '(a)'
var('x,y,z')
P=implicit_plot3d(x-z-1,(x,-2,2),(y,-2,2),(z,-2,2))
U=point3d([0,0,0],rgbcolor=(1,0,0),size=10)
show(P+U,figsize=[2.5,2.5,2.5])
```

(a)

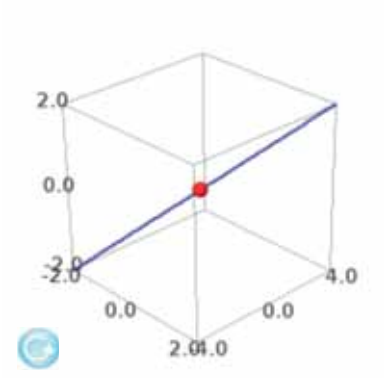


Non é subespacio porque a orixe (punto vermello) non está no conxunto.

```
print '(b)'
var('t')
P=parametric_plot3d([t,2*t,t],(t,-2,2),thickness=0.8)
```

```
U=point3d([0,0,0],rgbcolor=(1,0,0),size=20)
show(P+U,figsize=[2.5,2.5,2.5])
```

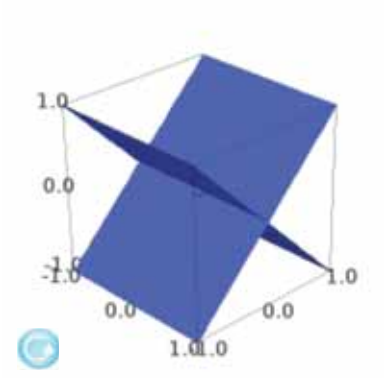
(b)



É unha recta que pasa pola orixe, daquela un subespacio unidimensional.

```
print '(c)'
var('x,y,z')
P=implicit_plot3d(abs(y)-abs(z),(x,-1,1),(y,-1,1),(z,-1,1))
U=point3d([0,0,0],gbcolor=(1,0,0),size=20)
show(P+U,figsize=[2.5,2.5,2.5])
```

(c)



É a unión conxuntista de dous subespacios pero non é subespacio porque hai vectores que están no conxunto cuxa suma non o está.

Exercicio 18:

No espazo dos polinomios reais de grado ≤ 3, $\Pi_3(R)$, considéranse as bases

$$\mathcal{B} = \{1+x^3, 1+x^2, 1+x, 1\} \quad \text{e} \quad \mathcal{C} = \{1, x, x^2, x^3\}$$

(a) Determinar as matrices de cambio de base de $\mathcal{C}$ a $\mathcal{B}$ e de $\mathcal{B}$ a $\mathcal{C}$.

(b) Achar as coordenadas do polinomio $p(x) = x + x^2 + x^3$ na base $\mathcal{B}$.

```
print '(a)'
print 'A matriz de cambio da base B a C é'
PBC=matrix([[1,0,0,1],[1,0,1,0],[1,1,0,0],
[1,0,0,0]]).transpose()
show(PBC)
print 'A matriz de cambio da base C a B é a súa inversa'
show(PBC.inverse())
print '(b)'
show(PBC.inverse()*vector([0,1,1,1]))
```

```
(a)
A matriz de cambio da base B a C é
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

```
A matriz de cambio da base C a B é a súa inversa
```

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & -1 & -1 \end{pmatrix}$$

```
(b)
```

$$(1, 1, 1, -3)$$

Exercicio 19:

Comprobar en $\mathbb{R}^7$ que **norma(**$X \wedge Y$**) = gram(**[X,Y]**).**

```
X=vector(listasim('x',7))
Y=vector(listasim('y',7))
Z=X.cross_product(Y)
L=[X,Y]
(norma(Z)-gram(L)).rational_simplify()
```

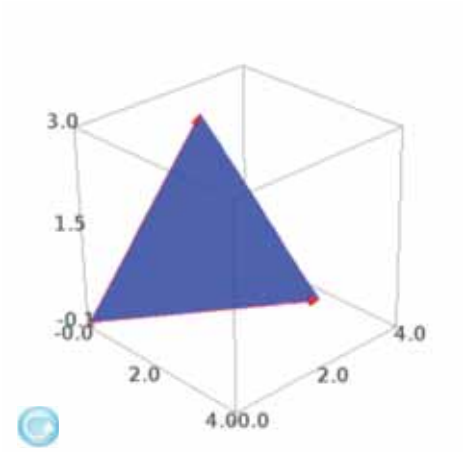
```
0
```

Exercicio 20:

Sexan os vectores X=(1,2,3), Y=(4,2,1), Z=(1,4,0). Representar o tetraedro de vértices **0** X,Y,Z e calcular o seu volume.

```
X=vector([1,2,3])
Y=vector([4,2,1])
Z=vector([1,4,0])
L=[X,Y,Z]
V=gram(L)
print 'V=',V/6
show(tetra(X,Y,Z),figsize=[3,3,3])
```

```
V= 20/3
```



Exercicio 21:

Sexan $A = (a_{ij})$ e $B = (b_{ij})$ en $\mathcal{M}_4(\mathbb{R})$ tales que

$$a_{ij} = |i - j| \qquad e \qquad b_{ij} = \begin{cases} ij & se \quad i \neq j \\ 0 & se \quad i = j \end{cases}$$

Achar $A * B$ e $B * A$

```
A=matrix(5)
for i in [1..4]:
    for j in [1..4]:
        A[i,j]=abs(i-j)
A=suprimefc(A,0,0)
B=matrix(5)
for i in [1..4]:
    for j in [1..4]:
        if i!=j:
            B[i,j]=i*j
B=suprimefc(B,0,0)
print 'A*B='
show(A*B)
print
print 'B*A='
show(B*A)
```

A*B=

$$\begin{pmatrix} 20 & 36 & 42 & 32 \\ 11 & 24 & 27 & 16 \\ 6 & 12 & 24 & 16 \\ 7 & 12 & 21 & 40 \end{pmatrix}$$

B*A=

$$\begin{pmatrix} 20 & 11 & 6 & 7 \\ 36 & 24 & 12 & 12 \\ 42 & 27 & 24 & 21 \\ 32 & 16 & 16 & 40 \end{pmatrix}$$

Exercicio 22:

Se A é unha matriz cadrada, demostrar que $A + A^t$ e $A^t * A$ son simétricas.

Solución:

É claro que

$(A + A^t)^t = A^t + A^{tt} = A^t + A = A + A^t$. Logo, $A + A^t$ é simétrica.

$(A^t * A)^t = A^t * A^{tt} = A^t * A$. Logo, $A^t * A$ é simétrica.

En Sage podemos comprobalo automaticamente para matrices de tamaño menor ou igual que 20.

```
L=[1..20]
n=choice(L)
A=random_matrix(RR,n)
C=A+A.transpose()
D=A.transpose()*A
print C==C.transpose()
print D==D.transpose()
```

```
True
True
```

Exercicio 23:

Probar que $A + iB$ é hermítica se e só se A é simétrica e B é antisimétrica.

Solución:

Se $H = A + iB$ é hermítica, $H^\star = A^t - iB^t = A + iB$. Logo, $A = A^t$ e $B = -B^t$.

Reciprocamente, se $A = A^t$ e $B = -B^t$, é claro que $\overline{(A+iB)}^t = A^t - iB^t = A + iB$. Logo $A + iB$ é hermítica.

En Sage podemos comprobalo automaticamente para matrices de tamaño menor ou igual que 20.

```
E=[1..20]
n=choice(E)
A0=random_matrix(RR,n)
A=(A0+A0.transpose())/2
B0=random_matrix(RR,n)
B=(B0-B0.transpose())/2
H=A+I*B
print H==(H.transpose()).conjugate()
```

```
True
```

```
E=[1..20]
n=choice(E)
A0=random_matrix(RR,n)
A1=(A0+A0.transpose())/2
B0=random_matrix(RR,n)
B1=(B0-B0.transpose())/2
C=A1+I*B1
A=parterm(C)
B=parteim(C)
print A==A.transpose()
print B==-B.transpose()
```

```
True
True
```

Exercicio 24:

Sexa $\mathbf{u} \in \mathbb{R}^n$ un vector unitario, I_n a matriz unidade e $H = I_n - 2\mathbf{u}^t\mathbf{u}$ a matriz de Householder de $\mathbf{u}$. Probar que $det(H) = -1$.

Solución:

Como $H(u^t) = (I_n - 2u^tu)(u^t) = u^t - 2u^t = -u^t$, resulta que u^t é un autovector correspondente ao autovalor -1.

Como H é simétrica, os autovectores asociados a outros autovalores son ortogonais a u^t. Calquera v^t ortogonal a u^t cumpre que $H(v^t) = (I_n - 2u^tu)(v^t) = v^t - 2u^tuv^t = v^t$ e, xa que logo, 1 é autovalor con multiplicidade $n-1$. Xa que logo, $det(H) = (-1)\cdot 1^{n-1} = -1$.

En Sage podemos comprobalo automaticamente para $n \leq 10$:

```
n=choice(range(1,10))
V=vector(listasim('v',n))
```

```
U=V/norma(V)
M=matrix(U)
H=identity_matrix(n)-2*M.transpose()*M
det(H).rational_simplify()
```

```
-1
```

APLICACIÓNS LINEAIS

Exercicio 25:

Sexa $L : R^4 \to R^4$ a aplicación lineal definida por:

$$L(x, y, z, t) = (y - 2z - t, -2x + 3y - 4z - 2t, -x + y - z - t, t)$$

(a) Achar a matriz asociada a L.
(b) Achar unha base de ker L e outra de im L.

```
print '(a)'
L=matrix([[0,1,-2,-1],[-2,3,-4,-2],[-1,1,-1,-1],
[0,0,0,1]]).transpose()
show(L)
print '(b)'
print 'Unha base de ker L é'
show (L.right_kernel().basis())
print 'Unha base de im L é'
IL=libera([[0,1,-2,-1],[-2,3,-4,-2],[-1,1,-1,-1],[0,0,0,1]])
show([vector(a) for a in IL])
```

```
(a)
```

$$\begin{pmatrix} 0 & -2 & -1 & 0 \\ 1 & 3 & 1 & 0 \\ -2 & -4 & -1 & 0 \\ -1 & -2 & -1 & 1 \end{pmatrix}$$

```
(b)
Unha base de ker L é
```

$$[(1, -1, 2, 1)]$$

```
Unha base de im L é
```

$$[(0, 1, -2, -1), (-2, 3, -4, -2), (-1, 1, -1, -1)]$$

Exercicio 26:

Nunha empresa obsérvase que, cada ano, o 50% dos fumadores deixa de fumar e o 10% dos non fumadores empeza a fumar. Estudar a proporción de fumadores a longo

prazo.

Solución:

Se no ano inicial hai F_0 fumadores e N_0 non fumadores, ao cabo dun ano haberá

$$F_1 = \tfrac{1}{2}F_0 + \tfrac{1}{10}N_0 \qquad N_1 = \tfrac{1}{2}F_0 + \tfrac{9}{10}N_0$$

Xa que logo,

$$\begin{pmatrix} F_1 \\ N_1 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{10} \\ \frac{1}{2} & \frac{9}{10} \end{pmatrix} \begin{pmatrix} F_0 \\ N_0 \end{pmatrix} \qquad \text{e} \qquad \begin{pmatrix} F_n \\ N_n \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{10} \\ \frac{1}{2} & \frac{9}{10} \end{pmatrix}^n \begin{pmatrix} F_0 \\ N_0 \end{pmatrix}$$

```
var('F N')
A=matrix([[1/2,1/10],[1/2,9/10]])
D,V=A.eigenmatrix_right()
LD=matrix([[1,0],[0,0]])
LA=V*LD*V.inverse()
FF=vector(LA[0])*vector([F,N])
show(FF)
```

$$\frac{1}{6}F + \frac{1}{6}N$$

Deixa de fumar $\frac{1}{6}$ dos traballadores.

TEORÍA ESPECTRAL FINITO DIMENSIONAL

Exercicio 27:

Considérase a aplicación lineal $L : \mathbb{R}^3 \to \mathbb{R}^3$ que respecto da base canónica vén dada pola matriz

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

(a) Achar unha matriz diagonal D e unha matriz inversible V tal que $A = VDV^{-1}$

(b) Probar que existe unha base de $\mathbb{R}^3$ en que a aplicación L vén dada por unha matriz diagonal.

(c) Achar unha raíz cadrada de A.

```
A=matrix([[2,1,1],[1,2,1],[1,1,2]])
print '(a)'
D,V=A.eigenmatrix_right()
```

```
show(D)
show(V)
show(V*D*V.inverse()==A)
```

(a)

$$\begin{pmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{pmatrix}$$

True

(b)

Respecto da base $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ de columnas de V, L vén dada pola matriz diagonal D:

$$L\mathbf{v}_1 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 4 \end{pmatrix} = 4\mathbf{v}_1$$

$$L\mathbf{v}_2 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = \mathbf{v}_2$$

$$L\mathbf{v}_3 = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix} = \mathbf{v}_3$$

(c)

D ten 8 raíces cadradas segundo a elección dos signos:

$$\sqrt{D} = \begin{pmatrix} \pm 2 & 0 & 0 \\ 0 & \pm 1 & 0 \\ 0 & 0 & \pm 1 \end{pmatrix}$$

É claro que $\sqrt{A} = V\sqrt{D}V^{-1}$ porque $\sqrt{A}\sqrt{A} = V\sqrt{D}V^{-1}V\sqrt{D}V^{-1} = VDV^{-1} = A$.

Exercicio 28:

Dados dous reais distintos e non nulos a, b considérase a matriz $A = (a_{ij}) \in \mathbb{M}_n$ tal que

$$a_{ij} = \begin{cases} a & se \quad i = j \\ b & se \quad i \neq j \end{cases}$$

Calcular o seu determinante

Solución:

Sage permítenos calcular en menos de 10 segundos, os 10 primeiros casos.

```
T0=walltime()
var('a b')
for n in range(1,10):
    A=b*ones_matrix(n)
    for i in range(n):
        A[i,i]=a
    show(factor(det(A)))
T1=walltime(T0)
T1
```

$$a$$

$$(a+b)(a-b)$$

$$(a+2\,b)(a-b)^2$$

$$(a+3\,b)(a-b)^3$$

$$(a+4\,b)(a-b)^4$$

$$(a+5\,b)(a-b)^5$$

$$(a+6\,b)(a-b)^6$$

$(a+7\,b)(a-b)^7$

$(a+8\,b)(a-b)^8$

```
14.054022789001465
```

Estes resultados suxiren que a fórmula xeral é

$$det(A) = (a + (n-1)b) \cdot (a-b)^{n-1}.$$

En efecto:

A matriz A é simétrica e todos os seus autovalores son reais. É evidente que $a-b$ é un autovalor de orde $n-1$ e que $a+(n-1)b$ é autovalor de orde 1. Como o determinante de A é o produto dos seus autovalores, a tese queda probada.

Exercicio 29:

Considérase a matriz

$$A = \begin{pmatrix} -3 & 3 & 3 & -3 \\ -1 & 1 & 1 & -1 \\ -2 & a & 3 & -4 \\ -1 & 1 & 2 & -3 \end{pmatrix}$$

(a) Sabendo que o vector $v = (3,1,1,0)$ é un autovector de A, calcular o valor de a.

(b) Achar o espectro de A.

(c) Estudar se A é diagonalizable.

(d) Calcular unha raíz cadrada de $-A$.

```
print'(a)'
var('a t')
A=matrix([[-3,3,3,-3],[-1,1,1,-1],[-2,a,3,-4],[-1,1,2,-3]])
L=(A*vector([3,1,1,0])-t*vector([3,1,1,0]))
show(L)
```

```
(a)
```

$(-3\,t-3,\ -t-1,\ a-t-3,\ 0)$

Para que este vector sexa nulo, é evidente que $t=-1$ e $a=2$.

```
print'(b)'
A=matrix([[-3,3,3,-3],[-1,1,1,-1],[-2,2,3,-4],[-1,1,2,-3]])
```

```
show(factor(A.charpoly()))
print '(c)'
D,V=A.eigenmatrix_right()
show(D);show(V)
show(A==V*D*V.inverse())
print'(d)'
show(-A*-A==-A)
```

(b)

$$x^2 \cdot (x+1)^2$$

(c)

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & \frac{1}{3} & 0 \\ 2 & -2 & 0 & 1 \\ 1 & -1 & -\frac{1}{3} & 1 \end{pmatrix}$$

True

(d)

True

Exercicio 30:

Diagonalizar as seguintes matrices simétricas:

$$A_1 = \begin{pmatrix} -1 & -1 & 2 \\ -1 & -1 & -2 \\ 2 & -2 & 2 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 5 & 1 & -2 \\ 1 & 5 & 2 \\ -2 & 2 & 2 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} -2 & -1 & 0 & 2 \\ -1 & -2 & 0 & -2 \\ 0 & 0 & -3 & 0 \\ 2 & -2 & 0 & 1 \end{pmatrix}$$

```
A1=matrix([[-1,-1,2],[-1,-1,-2],[2,-2,2]])
A2=matrix([[5,1,-2],[1,5,2],[-2,2,2]])
A3=matrix([[-2,-1,0,2],[-1,-2,0,-2],[0,0,-3,0],[2,-2,0,1]])
A=[A1,A2]
print 'A1 e A2'
for a in A:
    show([Roundm(svd(a)[0],2),Roundm(svd(a)[1],2),Roundm(svd(a)
[2],2)])
print 'A3'
show(Roundm(svd(A3)[0],2))
show(Roundm(svd(A3)[1],2))
show(Roundm(svd(A3)[2],2))
```

A1 e A2

$$\left[\begin{pmatrix} 0.41 & 0.91 & 0.0 \\ -0.41 & 0.18 & 0.89 \\ 0.82 & -0.37 & 0.45 \end{pmatrix}, \begin{pmatrix} 4.0 & 0.0 & 0.0 \\ 0.0 & 2.0 & 0.0 \\ 0.0 & 0.0 & 2.0 \end{pmatrix}, \begin{pmatrix} 0.41 & -0.41 & 0.82 \\ -0.91 & -0.18 & 0.37 \\ -0.0 & -0.89 & -0.45 \end{pmatrix}\right]$$

$$\left[\begin{pmatrix} 0.0 & -0.91 & 0.41 \\ 0.89 & -0.18 & -0.41 \\ 0.45 & 0.37 & 0.82 \end{pmatrix}, \begin{pmatrix} 6.0 & 0.0 & 0.0 \\ 0.0 & 6.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}, \begin{pmatrix} 0.0 & 0.89 & 0.45 \\ -0.91 & -0.18 & 0.37 \\ -0.41 & 0.41 & -0.82 \end{pmatrix}\right]$$

A3

$$\begin{pmatrix} -0.67 & 0.75 & 0.0 & 0.0 \\ -0.33 & -0.3 & -0.89 & 0.0 \\ 0.0 & -0.0 & -0.0 & -1.0 \\ 0.67 & 0.6 & -0.45 & 0.0 \end{pmatrix}$$

$$\begin{pmatrix} 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 3.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 3.0 \end{pmatrix}$$

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.45 & 0.0 & 0.89 \\ -0.0 & 0.89 & 0.0 & 0.45 \\ -0.0 & -0.0 & 1.0 & -0.0 \end{pmatrix}$$

Podemos comprobar que os resultados son correctos

```
A=[A1,A2,A3]
for a in A:
    show(prod(svd(a)))
```

$$\begin{pmatrix} -1.0 & -1.0 & 2.0 \\ -1.0 & -1.0 & -2.0 \\ 2.0 & -2.0 & 2.0 \end{pmatrix}$$

$$\begin{pmatrix} 5.0 & 1.0 & -2.0 \\ 1.0 & 5.0 & 2.0 \\ -2.0 & 2.0 & 2.0 \end{pmatrix}$$

$$\begin{pmatrix} -2.0 & -1.0 & 0.0 & 2.0 \\ -1.0 & -2.0 & 0.0 & -2.0 \\ 0.0 & 5.95808196779 \times 10^{-16} & -3.0 & 2.9790409839 \times 10^{-16} \\ 2.0 & -2.0 & 0.0 & 1.0 \end{pmatrix}$$

Exercicio 31:

Determinar a descomposición en valores singulares dos operadores dados polas seguintes matrices:

$$A_1 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 0 & 0 \\ 2 & 2 \\ -2 & 2 \\ 2 & -2 \end{pmatrix} \qquad A_4 = \begin{pmatrix} 3 & -1 & 3 \\ 1 & 1 & -1 \\ 1 & -1 & -1 \\ 3 & 1 & 3 \end{pmatrix}$$

$$A_5 = \begin{pmatrix} 1 & 0 & -1 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix}$$

```
A1=matrix([[1,1,0],[0,0,1]])
A2=matrix([[1,1],[1,0],[0,1]])
A3=matrix([[0,0],[2,2],[-2,2],[2,-2]])
A4=matrix([[3,-1,3],[1,1,-1],[1,-1,-1],[3,1,3]])
A5=matrix([[1,0,-1],[-1,0,1],[0,0,0],[1,0,-1]])
A=[A1,A2,A3,A4,A5]
F=[]
for i in range(len(A)):
    F.append(factorizacion_vs(A[i]))
    show(F[-1])
    show(A[i]==prod(F[-1]))
```

$$\left[\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} \sqrt{2} & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}\right]$$

True

$$\left[\begin{pmatrix} \frac{1}{3}\sqrt{3}\sqrt{2} & 0 \\ \frac{1}{6}\sqrt{3}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{6}\sqrt{3}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}, \begin{pmatrix} \sqrt{3} & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}\right]$$

True

$$\left[\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ -\frac{1}{2}\sqrt{2} & 0 \\ \frac{1}{2}\sqrt{2} & 0 \end{pmatrix}, \begin{pmatrix} 4 & 0 \\ 0 & 2\sqrt{2} \end{pmatrix}, \begin{pmatrix} \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \end{pmatrix}\right]$$

True

$$\left[\begin{pmatrix} \frac{1}{2}\sqrt{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2}\sqrt{2} & \frac{1}{2} \\ 0 & \frac{1}{2}\sqrt{2} & -\frac{1}{2} \\ \frac{1}{2}\sqrt{2} & 0 & \frac{1}{2} \end{pmatrix}, \begin{pmatrix} 6 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}, \begin{pmatrix} \frac{1}{2}\sqrt{2} & 0 & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & 0 & -\frac{1}{2}\sqrt{2} \\ 0 & 1 & 0 \end{pmatrix}\right]$$

True

$$\left[\begin{pmatrix} \frac{1}{6}\sqrt{6}\sqrt{2} \\ -\frac{1}{6}\sqrt{6}\sqrt{2} \\ 0 \\ \frac{1}{6}\sqrt{6}\sqrt{2} \end{pmatrix}, (\sqrt{6}), \begin{pmatrix} \frac{1}{2}\sqrt{2} & 0 & -\frac{1}{2}\sqrt{2} \end{pmatrix}\right]$$

True

Exercicio 32:

Achar as inversas de Moore-Penrose das matrices do exercicio anterior.

```
A1=matrix([[1,1,0],[0,0,1]])
A2=matrix([[1,1],[1,0],[0,1]])
A3=matrix([[0,0],[2,2],[-2,2],[2,-2]])
A4=matrix([[3,-1,3],[1,1,-1],[1,-1,-1],[3,1,3]])
A5=matrix([[1,0,-1],[-1,0,1],[0,0,0],[1,0,-1]])
A=[A1,A2,A3,A4,A5]
F=[]
for i in range(len(A)):
    F.append(mpinv(A[i]))
    show(F[-1])
```

$$\begin{pmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ \frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

$$\begin{pmatrix} 0 & \frac{1}{4} & -\frac{1}{8} & \frac{1}{8} \\ 0 & \frac{1}{4} & \frac{1}{8} & -\frac{1}{8} \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{12} & \frac{1}{4} & \frac{1}{4} & \frac{1}{12} \\ -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \\ \frac{1}{12} & -\frac{1}{4} & -\frac{1}{4} & \frac{1}{12} \end{pmatrix}$$

$$\begin{pmatrix} \frac{1}{6} & -\frac{1}{6} & 0 & \frac{1}{6} \\ 0 & 0 & 0 & 0 \\ -\frac{1}{6} & \frac{1}{6} & 0 & -\frac{1}{6} \end{pmatrix}$$

VARIEDADES DIFERENCIABLES

Exercicio 33:

Achar a medida da variedade diferenciable cuxa carta (U, f) é:

a) $U = (0, 2\pi), \quad f : U \to \mathbb{R}^2$ con $f(t) = [5\cos(t), 5\sin(t)]$.

b) $U = (0, 6\pi), \quad f : U \to \mathbb{R}^3$ tal que $f(t) = [cos(t), sin(t), t]$.

c) $U = (0, 2\pi) \times (0, 6), \quad f : U \to \mathbb{R}^2$ con $f(t, v) = [vcos(t), vsin(t)]$.

d) $U = (0, 2\pi) \times (-\frac{\pi}{2}, \frac{\pi}{2}), f : U \to \mathbb{R}^3$ con

$f(u, v) = [2cos(u)cos(v), 2sin(u)cos(v), 2sin(v)]$.

e) $U = (0, r) \times (0, \pi) \times (0, 2\pi), f : U \to \mathbb{R}^3$ con

$f(l, u, v) = [lsin(u)cos(v), lsin(u)sin(v), lcos(u)]$.

```
print'a)'
f(t)=[5*cos(t),5*sin(t)]
M=med(f)
show(integrate(M,t,0,2*pi))
print'b)'
f(t)=[cos(t),sin(t),t]
M=med(f)
show(integrate(M,t,0,6*pi))
print'c)'
f(t,v)=[v*cos(t),v*sin(t)]
M=med(f)
show(integrate(integrate(M,t,0,2*pi),v,0,6))
```

```
print'd)'
f(u,v)=[2*cos(u)*cos(v),2*cos(u)*sin(v),2*sin(u)]
M=med(f)
show(M)
print'A funcion med(f) é 4|cos(u)|'
print'A súa integral en (0,2pi) é 4 veces a integral de 4cos(u)
en (0,pi/2)'
show(integrate(4*integrate(4*cos(u),u,0,pi/2),v,-pi/2,pi/2))
print'e)'
var('r',domain=RR)
f(l,u,v)=[l*sin(u)*cos(v),l*sin(u)*sin(v),l*cos(u)]
M=med(f)
show(M)
print'A funcion med(f) é l^2|sin(u)|'
print'A súa integral en (0,pi) é 2 veces a integral de
l^2sin(u) en (0,pi/2)'
show(integrate(integrate(2*integrate(l^2*sin(u),u,0,pi
/2),v,0,2*pi),l,0,r))
```

a)

$$10\,\pi$$

b)

$$3\,(2\,\sqrt{2})\,\pi$$

c)

$$36\,\pi$$

d)

$$4\,\sqrt{\cos(u)^2}$$

A funcion med(f) é 4|cos(u)|
A súa integral en (0,2pi) é 4 veces a integral de 4cos(u) en (0,pi/2)

$$16\,\pi$$

e)

$$\sqrt{l^4\,\sin(u)^2}$$

A funcion med(f) é l^2|sin(u)|

```
A súa integral en (0,pi) é 2 veces a integral de l^2sin(u) en
(0,pi/2)
```

$$\frac{4}{3}\pi r^3$$

Exercicio 34:

Para unha lista L de $n+1$ complexos distintos, definimos a curva de Bézier $B_L : [0,1] \to \mathbb{C}$ pola ecuación:

$$B_L(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k L[k]$$

Comprobar para $n > 2$ as seguintes propiedades:

a) $B_L(0) = L[0]$ e $B_L(1) = L[n]$.

b) $\frac{B'_L(0)}{L[1]-L[0]} \in \mathbb{R}^+$ e $\frac{B'_L(1)}{L[n]-L[n-1]} \in \mathbb{R}^+$.

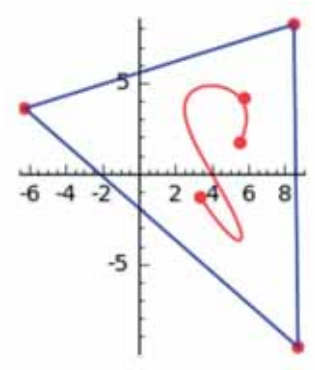
c) $\{B_L(t) \,|\, t \in [0,1]\} \subset pco(L)$

Para cada $2 < n \leq 6$, acoutar a razón entre a lonxitude da fronteira da envoltura convexa de L e a lonxitude da curva de Bezier correspondente.

```
def Bezier(L):
    var('t',domain=RR)
    n=len(L)-1
    C=[factorial(n)/(factorial(k)*factorial(n-k))*((1-t)^(n-
k))*(t^(k))*vector(ri(L[k])) for k in range(n+1)]
    D=parametric_plot(sum(C),(t,0,1),color='red')+plce(L)
    return sum(C), D
```

```
L=listacomplejos(10,6)
B=Bezier(L)
show(B[1]+pco(L),figsize=[2.5,2.5])
print 'lonxitude da curva=',integrate(med(B[0]),t,0,1).n()
C=envolturaconvexa(L)
print 'lonxitude da fronteira =',lon(C)[0]
```

```
    lonxitude da curva= 17.8531524183
    lonxitude da fronteira = 53.2177282036728
```



Para establecer estas acotacións podemos realizar experimentos de K probas, do tipo seguinte:

```
n=6; K=10; r=[]
for k in range(K):
    L=listacomplejos(10,n)
    B=Bezier(L)
    CU=integrate(med(B[0]),t,0,1).n()
    C=envolturaconvexa(L)
    E=lon(C)[0]
    r.append(E/CU)
[min(r),max(r)]
```

```
[1.88477592186517, 3.13617414464007]
```

Exercicio 35:

Calcular a superficie e o volume dunha esfera de radio r.

```
var('r',domain=RR)
s(u,v)=[cos(u)*cos(v),cos(u)*sin(v),sin(u)]
med(s)
```

```
sqrt(cos(u)^2)
```

```
show(r^2*integrate(2*integrate(cos(u),u,0,pi/2),v,0,2*pi))
```

$$4\,\pi r^2$$

```
w(l,u,v)=[l*cos(u)*cos(v),l*cos(u)*sin(v),l*sin(u)]
med(w)
```

```
sqrt(l^4*cos(u)^2)
```

```
show(integrate(integrate(2*integrate(l^2*cos(u),u,0,pi/2)
    ,v,0,2*pi),l,0,r))
```

$$\frac{4}{3}\,\pi r^3$$

Exercicio 36:

Calcular a área lateral e o volume dun cono de vértice (0,0,5) e base a circunferencia $\begin{cases} x^2+y^2=1 \\ z=0 \end{cases}$

```
P=vector([0,0,5])
C(t)=[cos(t),sin(t),0]
s(t,u)=list(P+u*(C-P))
med(s)
```

```
sqrt(26)*sqrt(u^2)
```

```
sqrt(26)*integrate(integrate(u,u,0,1),v,0,2*pi)
```

```
sqrt(26)*pi
```

```
f(u,v)=[u*cos(v),u*sin(v)]
med(f)
```

```
sqrt(u^2)
```

```
var('r',domain=RR)
assume(r>0)
integrate(integrate(med(f),u,0,r),v,0,2*pi)
```

```
pi*r^2
```

```
c(t)=[cos(t),sin(t),t]
M=med(c)
integrate(M,t,0,4*pi)
```

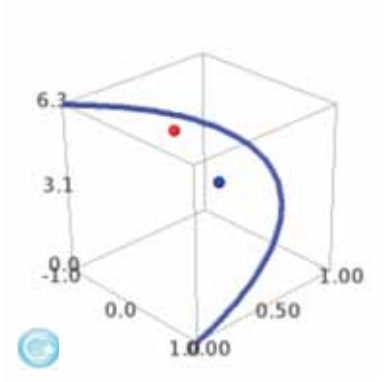
```
(4*sqrt(2))*pi
```

Exercicio 37:

Achar o centroide (azul) e o baricentro (vermello) da hélice H con carta local $f:(0,,\pi)\to\mathbb{R}^3$ onde $f(t)=[cos(t),sin(t),2t]$ se a densidade da masa é $\delta(x,y,z)=x^2+y^2+z^2$.

```
f(t)=[cos(t),sin(t),2*t]
Mf=med(f)
LH=integral(Mf,t,0,pi)
C1=integral(f[0]*Mf,t,0,pi).n()
C2=integral(f[1]*Mf,t,0,pi).n()
C3=integral(f[2]*Mf,t,0,pi).n()
CH=vector([C1,C2,C3])/LH
delta(x,y,z)=x^2+y^2+z^2
D=EV(delta,f(t)).trig_simplify()
MH=integral(D*Mf,t,0,pi)
B1=integral(f[0]*D*Mf,t,0,pi).n()
B2=integral(f[1]*D*Mf,t,0,pi).n()
B3=integral(f[2]*D*Mf,t,0,pi).n()
BH=vector([B1,B2,B3])/MH
parametric_plot3d(f,(t,0,pi),thickness=7,figsize=
```

```
[2.5,2.5,2.5])+point3d(CH,size=15,color='blue')+point3d(BH,size=15
```

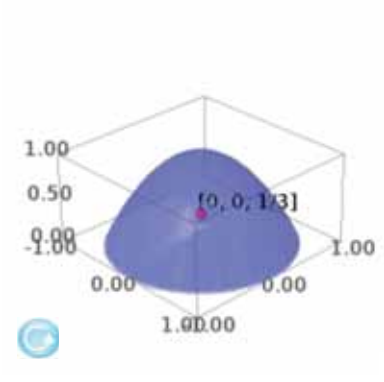


Exercicio 38:

Achar o centroide do sólido S cuxa carta local é (U, f) onde $U = (0,1) \times (0,1) \times (0,2\pi)$ e $f : U \to \mathbb{R}^3$ vale

$$f(p,r,t) = [prcos(t), prsin(t), p(1-r^2)]$$

```
f(p,r,t)=[p*r*cos(t),p*r*sin(t),p*(1-r^2)]
Mf=med(f)
VS=integral(integral(integral(Mf,p,0,1),r,0,1),t,0,2*pi)
X=(1/VS)*integral(integral(integral(f[0]*Mf,p,0,1),r,0,1),
t,0,2*pi)
Y=(1/VS)*integral(integral(integral(f[1]*Mf,p,0,1),r,0,1),
t,0,2*pi)
Z=(1/VS)*integral(integral(integral(f[2]*Mf,p,0,1),r,0,1),
t,0,2*pi)
CS=vector([X,Y,Z])
PAR=parametric_plot3d(f(1,r,t),(r,0,1),(t,0,2*pi),opacity=1
/2,figsize=[3,3,3])+parametric_plot3d(f(p,1,t),(p,0,1),
(t,0,2*pi),opacity=1/2,figsize=[2.5,2.5,2.5])
show(PAR+point3d(CS,size=15,color='red')+text3d([0,0,1/3],[2
/3,0,5/7],color='black'),aspect_ratio=1)
```



Exercicio 39:

Calcular o momento de inercia dunha circunferencia de radio r e densidade de masa uniforme $\delta(x,y,z)=k$, respecto da recta que pasa por un dos seus puntos e é perpendicular ao plano que a contén.

```
var('r k',domain='positive')
e=vector([0,0,1])
f(t)=[r*cos(t),r*sin(t),0]
Mf=med(f)
delta(x,y,z)=k
D=EV(delta,f(t))
MC=integral(D*Mf,t,0,2*pi)
MI0=integral(D*gram([f(t),e])^2*Mf,t,0,2*pi)
d=r
MI=d^2*MC+MI0
show(MI)
```

$$4\,\pi k r^3$$

Exercicio 40:

Achar o momento de inercia da variedade do exercicio 38 respecto á recta que pasa polo punto $(-2,1,1)$ e ten vector direccional $(0,0,1)$ se a densidade de masa é $\delta(x,y,z)=x+y+z$.

Solución:

Utilizando o teorema de Steiner, calculamos o momento de inercia respecto do eixe Z e sumámoslle $5m$ (M)

```
f(p,r,t)=[p*r*cos(t),p*r*sin(t),p*(1-r^2)]
e=vector([0,0,1])
delta(x,y,z)=x+y+z
D=EV(delta,f(p,r,t))
MS=integral(integral(integral(D*Mf,p,0,1),r,0,1),t,0,2*pi)
MI0=integral(integral(integral(D*gram([f(p,r,t),e])^2*Mf,p,0,1),r,
d=sqrt(5)
MI=d^2*MS+MI0
show(MI)
```

$$\frac{7}{8}\,\pi$$

3.2. Worksheet 1

```
load (DATA + 'worksheet0.sage')
load (DATA + 'worksheet1.sage')
```

3.2.1. Problemas inversos

Sexan dous espazos vectoriais reais X, Y, un subconxunto $\Omega \subset X$ e un dato $b \in Y$. Dada unha función $f : \Omega \to Y$, existe algún $x \in \Omega$ tal que $f(x) = b$?

Sobreentendendo o resto das concrecións, adoitamos dicir: Resolver a ecuación $f(x) = b$.

En xeral, este problema pode non ter solución e, se a ten, pode non ser única.

Se podemos asegurar a existencia de solucións, para concluílo, debemos determinar todas as pertencentes a Ω.

Aínda que o problema non teña solución, si pode tela o seguinte problema alternativo:

Achar os mínimos da función

$$F : \Omega \to Y \overset{\| \ \|^2}{\to} \mathbb{R} \quad \text{con} \quad F(x) = \|b - f(x)\|^2 \quad \forall x \in \Omega$$

sendo $\| \ \|$ a norma euclídea en Y.

3.2.1.1. Problema inverso lineal

É o problema inverso en que X, Y son de dimensión finita e $f : X \to Y$ é lineal.

Se dim X=m e dim Y=n, fixadas senllas bases de Hamel, a función f vén dada por unha matriz $A \in \mathfrak{M}_{n \times m}(\mathbb{R})$ e o clásico teorema de Rouché-Frobenius asegura:

1.- O problema ten solución se $rang(A) = rang([A, b])$.

2.- O problema ten solución única se $rang(A) = rang([A, b]) = m$.

Cando o problema non ten solución, a través do espazo euclídeo $(Y, \| \ \|)$, acharemos os mínimos da función

$$Q : X \to Y \overset{\| \ \|^2}{\to} \mathbb{R} \quad \text{con} \quad Q(x) = \|b - Ax\|^2 \quad \forall x \in X$$

Este problema alternativo é un problema de mellor aproximación nun subespazo cerrado

dun espazo de Hilbert e, xa que logo, sempre ten solución. Os mínimos de Q podemos obtelos anulando a diferencial $DQ(x)$:

Como $Q(x) = (b|b) - 2(b|Ax) + (Ax|Ax) = (b|b) - 2(A^t b|x) + (A^t Ax|x)$, é claro que $DQ(x) = -2A^t b + 2A^t Ax$ e, xa que logo,

$$DQ(x) = 0 \quad \Leftrightarrow A^t Ax = A^t b$$

As solucións deste sistema autoadxunto son as mellores aproximacións do sistema inicial $Ax = b$. A función **rouche**(A,b) dinos se o sistema é compatible ou non e dános a dimensión da variedade de solucións ou mellores aproximacións.

Teorema de mellor aproximación de Moore-Penrose:

Sexa $L : \mathbb{R}^m \to \mathbb{R}^n$ unha función lineal e sexa $L\mathbf{x} = \mathbf{b}$ un problema inverso. Se L^+ é a inversa de Moore-Penrose de L, $L^+\mathbf{b}$ é o vector de norma euclídea mínima entre os da variedade de solucións $S_b = \{\mathbf{x} \in \mathbb{R}^m \mid L\mathbf{x} = \mathbf{b}\}$ ou entre os da variedade de mellores aproximacións $A_b = \{\mathbf{x} \in \mathbb{R}^m \mid L^t L\mathbf{x} = L^t\mathbf{b}\}$.

Demostración:

Sexan $L = \sum_{i=1}^k s_i \mathbf{u}_i \otimes \mathbf{v}_i$ e $L^+ = \sum_{i=1}^k \frac{1}{s_i}\mathbf{v}_i \otimes \mathbf{u}_i$. Se existe $\mathbf{x}_0$ tal que $L\mathbf{x}_0 = \mathbf{b}$, é suficiente probar que

$$L^+\mathbf{b} \in (\mathbf{x}_0 + ker L) \cap [ker L]^\perp$$

Isto é fácil porque $LL^+\mathbf{b} = LL^+L\mathbf{x}_0 = L\mathbf{x}_0$ e, xa que logo, $L^+\mathbf{b} \in \mathbf{x}_0 + ker L$. Ademais, $L^+\mathbf{b} = \sum_{i=1}^k \frac{1}{s_i}(\mathbf{v}_i|\mathbf{b})\mathbf{u}_i \in [ker L]^\perp$.

Exemplos:

Resolver os sistemas lineales $A\mathbf{x} = \mathbf{b}$, onde

1) $A = \begin{pmatrix} 4 & 8 & 4 & 0 \\ 1 & 5 & 4 & -3 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 8 \\ -4 \\ 10 \\ -4 \end{pmatrix}$

```
A=matrix([[4,8,4,0],[1,5,4,-3]])
b=vector([8,-4,10,-4])
rouche(A,b)
```

```
Problema mal suscitado
```

2) $A = \begin{pmatrix} 4 & 8 & 4 & 0 \\ 1 & 5 & 4 & -3 \\ 1 & 4 & 7 & 2 \\ 1 & 3 & 0 & -2 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 8 \\ -4 \\ 10 \\ -4 \end{pmatrix}$

```
A=matrix([[4,8,4,0],[1,5,4,-3],[1,4,7,2],[1,3,0,-2]])
b=vector([8,-4,10,-4])
rouche(A,b)
```

```
O sistema ten a solucion unica
(3, -1, 1, 2)
```

3) $A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$, $\mathbf{b}_1 = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$

```
A=matrix([[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
b1=vector([32,23,33,31])
show(rouche(A,b1).n())
```

```
O sistema ten a solucion unica

(1.00000000000000, 1.00000000000000, 1.00000000000000, 1.00000000000000)
```

4) $A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$, $\mathbf{b}_2 = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$

```
A=matrix([[10,7,8,7],[7,5,6,5],[8,6,10,9],[7,5,9,10]])
b2=vector([32.1,22.9,33.1,30.9])
show(rouche(A,b2).n())
```

```
O sistema ten a solucion unica

(9.19999999890609, −12.5999999981888, 4.49999999954511, −1.09999999973018)
```

A notable diferenza entre as solucións de 3) e 4), para a pequena diferenza entre $\mathbf{b_1}$ e $\mathbf{b_2}$ débese ao enorme número de condición da matriz A.

```
print 'cond(A)=', cond(A)
```

```
cond(A)= 2984.09270168
```

5) $A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 8 & -9 & -6 \\ 3 & 5 & -6 & 4 & 7 \\ 1 & 3 & 2 & -4 & -12 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 10 \\ 13 \\ 25 \\ 31 \end{pmatrix}$

```
A=matrix([[1,2,3,4,5],[2,5,8,-9,-6],[3,5,-6,4,7],[1,3,2,-
4,-12]])
b=vector([10,13,25,31])
R=rouche(A,b)
show(R)
```

```
O sistema ten unha variedade de dimension 1
de solucions cuxa expresion parametrica e
```

$$\left(5103\,p_0 - \frac{233}{4},\ -2551\,p_0 + \frac{415}{12},\ 241\,p_0 - \frac{37}{12},\ 59\,p_0 + \frac{25}{12},\ -192\,p_0\right)$$

A solución de norma euclídea mínima pódese obter con **pinv**(A).

```
show(pinv(A)*b)
```

$$(2.10009169092,\ 4.41420068518,\ -0.233172232508,\ 2.78109061528,\ -2.27066776497)$$

6) $A = \begin{pmatrix} 3 & 2 & 7 & 46 \\ 15 & 5 & 7 & -12 \\ 3 & 25 & -63 & 9 \\ 12 & 3 & 2 & -4 \\ 23 & 35 & 22 & -5 \\ 7 & 3 & 12 & -17 \\ 2 & 33 & 24 & -43 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 28 \\ 57 \\ 25 \\ 35 \\ 43 \\ 5 \\ 47 \end{pmatrix}$

```
A=matrix([[3,2,7,46],[15,5,7,-12],[3,25,-63,9],[12,3,2,-
4],[23,35,22,-5],[7,3,12,-17],[2,33,24,-43]])
b=vector([28,57,25,35,43,5,47])
show(rouche(A,b).n())
```

```
O sistema non ten solucion
pero ten mellor aproximacion unica
```

$$(1.78504265380862,\ 0.754698902822306,\ 0.0293005955206407,\ 0.0348590050046829)$$

7) $A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}$

```
A=matrix([[1,2,3],[1,2,3],[1,2,3]])
b=vector([3,4,5])
R=rouche(A,b)
show(R)
```

```
O sistema non ten solucion
pero ten unha variedade de dimension 2
de mellores aproximacions con expresion parametrica
```

$$(p_0 + 4,\ p_0 + 3\,p_1,\ -p_0 - 2\,p_1)$$

A solución de norma euclídea mínima pódese obter con **mpinv**(A).

```
show(mpinv(A)*b)
```

$$\left(\frac{2}{7},\ \frac{4}{7},\ \frac{6}{7}\right)$$

8) $A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & -1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & -1 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 3 \\ 1 \\ 3 \\ 1 \end{pmatrix}$

```
A=matrix([[1,0,1],[0,-1,1],[1,0,1],[0,1,-1]])
b=vector([3,1,3,1])
show(rouche(A,b))
```

```
O sistema non ten solucion
pero ten unha variedade de dimension 1
de mellores aproximacions con expresion parametrica
```

$$(p_0 + 3,\ -p_0,\ -p_0)$$

A mellor aproximación de norma euclídea mínima pódese obter con **mpinv**(A).

```
show(mpinv(A)*b)
```

$$(2,\ 1,\ 1)$$

9) Escribir o sistema de ecuacións

$$\begin{cases} x_2 - 5x_3 + 7x_1 & = 5 \\ x_1 + 17x_3 + 5x_2 & = 12 \\ 4x_3 - 2x_1 + 12x_2 & = 0 \end{cases}$$

en forma matricial e resolvelo.

```
X=listasim('x',4)
X=X[1:]
E1=x2-5*x3+7*x1
E2=x1+17*x3+5*x2
E3=4*x3-2*x1+12*x2
f(X)=[E1,E2,E3]
A=JAC(f);b=vector([5,12,0])
show(A)
rouche(A,b)
```

$$\begin{pmatrix} 7 & 1 & -5 \\ 1 & 5 & 17 \\ -2 & 12 & 4 \end{pmatrix}$$

```
O sistema ten a solucion unica
(422/359, -13/718, 461/718)
```

3.2.1.2. Problema inverso non lineal

Métodos directos para achar as raíces de polinomios de grado menor que 5:

Para un polinomio $p : \mathbb{R} \to \mathbb{R}$, podemos usar a función **Polinomio**(P) onde P é a lista dos coeficientes dos monomios, ordenada por grados decrecentes. Esta función usa como auxiliares as funcións **AlKhwarizmi**(a,b,c), **Tartaglia**(a,b,c,d) e **Euler**(a,b,c,d,e) para os polinomios de grado 2, 3 e 4, respectivamente.

Exemplo 1:

Achar as raíces do polinomio $p(x) = 8$.

```
C=[8]
show(Polinomio(C))
```

```
Polinomio constante
```

Exemplo 2:

Achar as raíces do polinomio $p(x) = x - 8$.

```
C=[1,-8]
show(Polinomio(C))
```

```
8
```

Exemplo 3:

Achar as raíces do polinomio $p(x) = 4x^2 - 3x + 12$.

```
C=[4,-3,12]
show(num(Polinomio(C)))
```

$$[0.375000000000000 + 1.69096865730859i, 0.375000000000000 - 1.69096865730859i]$$

Exemplo 4:

Achar as raíces do polinomio $p(x) = 13x^3 - 10x^2 - x + 5$.

```
C=[13,-10,-1,5]
show(Polinomio(C))
```

$$[-0.566230510840716, 0.667730640035743 + 0.483106239099904i, 0.667730640035743 - 0.483106239099904i]$$

Exemplo 5:

Achar as raíces do polinomio $p(x) = x^4 - 3x^3 + 4x^2 - 2x + 3$.

```
C=[1,-3,4,-2,3]
Polinomio(C)
```

```
[-0.0781471302651938 + 0.899807460564854*I, 1.57814713026519 -
1.08949615540615*I, -0.0781471302651938 - 0.899807460564854*I,
1.57814713026519 + 1.08949615540615*I]
```

Exemplo 6:

Achar as raíces do polinomio $p(x) = x^5 - 3x^3 + 4x^2 - 2x + 3$.

```
C=[1,0,-3,4,-2,3]
show(Polinomio(C))
```

```
Non existe metodo directo
```

Teorema de Bolzano:

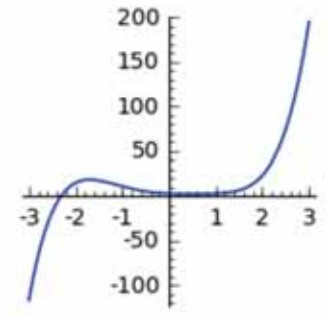
Sexan X=Y=$\mathbb{R}$, $\Omega = [a, b]$ e $f : [a, b] \to \mathbb{R}$ continua con f(a)·f(b)<0.

Existe polo menos unha $x \in (a, b)$ tal que $f(x) = 0$ á que podemos aproximarnos controladamente coa función **biseccion**(f,a,b,T).

Exemplo 1:

Achar a raíz real do polinomio $p(x) = x^5 - 3x^3 + 4x^2 - 2x + 3$

```
p(x)=[x^5-3*x^3+4*x^2-2*x+3]
show(plot(p[0],(x,-3,3)),figsize=[2.2,2.2])
```




```
biseccion(p[0],-3,-2,10^(-6))
```

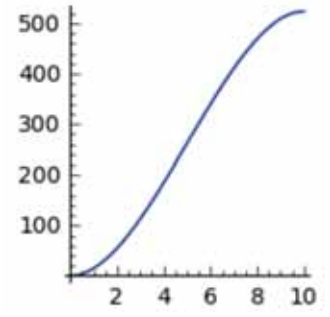
```
-2.31227779388428
```

Exemplo 2:

Achar a altura que alcanza a auga contida nun depósito esférico de radio r=5 m si está ao 70% da súa capacidade.

```
var('x')
V(h)=integrate(pi*(10-x)*x,x,0,h)
print 'O volume V en función da altura h é'
print 'V(h)=',V(h)
show(plot(V,(h,0,10)),figsize=[2.2,2.2])
print
V0=(7/10)*V(10)
p(h)=[V(h)-V0]
print biseccion(p[0],5,10,10^(-6))
```

```
O volume V en función da altura h é
V(h)= -1/3*pi*(h^3 - 15*h^2)

6.36742532253265
```



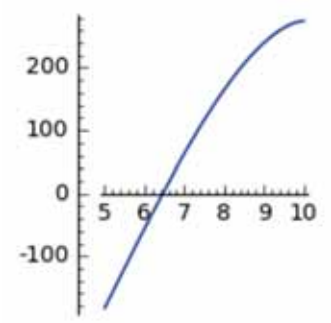
Exemplo 3:

Cortamos o depósito esférico anterior por un meridiano e volvemos a soldar as dúas

metades nos extremos dun cilindro de modo que obtemos un novo depósito convexo que cuadriplica a capacidade do inicial . Achar a altura do auga no novo depósito cando estea ao 70 % da súa capacidade.

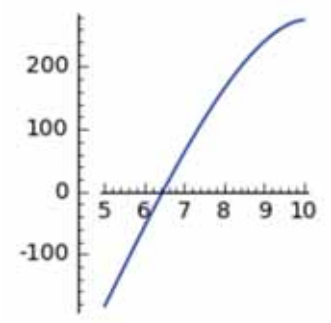
```
assume(h>0)
NV(h)=V(h)+10*integrate(sqrt((10-x)*x),x,0,h)
show(plot(NV(h)-(7/10)*NV(10),(h,5,10)),figsize=[2.2,2.2])
show((NV(h)-(7/10)*NV(10)).find_root(6,8))
```

6.45917713755




```
assume(h>0)
NV(h)=V(h)+10*integrate(sqrt((10-x)*x),x,0,h)
show(plot(NV(h)-(7/10)*NV(10),(h,5,10)),figsize=[2.2,2.2])
show((NV(h)-(7/10)*NV(10)).find_root(6,8))
```

6.45917713755



Teorema do punto fixo de Banach:

Sexa X=Y un espazo normado completo, $\Omega \subset X$ pechado e $f(\Omega) \subset \Omega$ cumprindo que existe $C \in (0,1)$ tal que

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq C\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \Omega$$

Existe unha e só unha solución $\mathbf{s} \in \Omega$ do problema inverso $(f - I)(\mathbf{x}) = \mathbf{0}$ que se obtén elixindo calquera $\mathbf{x}_0 \in \Omega$ e iterando:

$$\mathbf{x}_1 = f(\mathbf{x}_0), \ldots, \mathbf{x}_n = f(\mathbf{x}_{n-1}) \text{ posto que } \mathbf{s} = \lim \mathbf{x}_n.$$

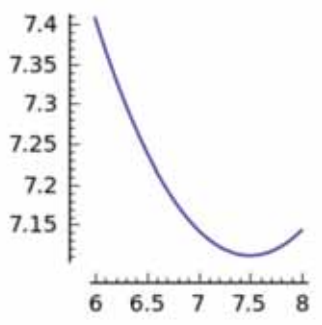
A funcion **banach**(f,$\mathbf{x}_0$,T) implementa este algoritmo con tolerancia T e dános o punto fixo e o número de iteracións realizadas.

Exemplos:

1) Achar un punto fixo da función $g(x) = \frac{400}{15x - x^2}$

É fácil ver que $g([6,8]) \subset [6,8]$:

```
g(x)=[400/(15*x-x^2)]
show(plot(g[0],(x,6,8),hue=.7),figsize=[2.2,2.2])
```



Como g é derivable e

$$|g(x_1) - g(x_2)| \leq |g'(x)| \cdot |x_1 - x_2| \text{ para algún } x \in [x_1, x_2]$$

podemos asegurar que $g : [6,8] \to [6,8]$ é contractiva con constante de contractividade $C = 0.42$.

```
gprim(x)=diff(g[0],x)
A=[6,6.01..8]
max([abs(gprim(a)) for a in A])
```

```
0.411522633744856
```

Podemos achar o único punto fixo en [6,8] tomando 7 como punto inicial en 5 iteracións

```
Banach(g,vector([7]),10^(-6))
```

```
((7.12859264097418), 5)
```

Exemplo 2:

Sexa $F : \mathbb{R}^2 \to \mathbb{R}^2$ a función dada por

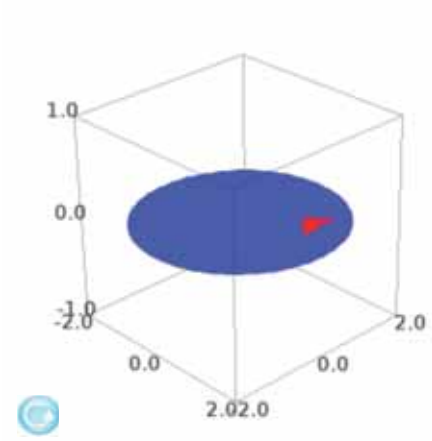
$$F(x,y) = [x^2 - 10x + y^2 + 8, xy^2 + x - 10y + 8]$$

Achar unha solución de F(**x**)= **0**.

Buscamos unha solución de G(**x**)=**x** sendo $G:\mathbb{R}^2\to\mathbb{R}^2$ a función dada por $G(x,y)=[\frac{x^2+y^2+8}{10},\frac{xy^2+x+8}{10}]$.

Si B é a bóla pechada de centro **0** e radio 2 é fácil ver que $G(B)\subset B$:

```
c(r,t)=[r*cos(t),r*sin(t),0]
B=parametric_plot3d(c,(r,0,2),(t,0,2*pi),figsize=[3,3,3])
G(x,y)=[(x^2+y^2+8)/10,(x*y^2+x+8)/10,0]
GB=parametric_plot3d(G(x=c(r,t)[0],y=c(r,t)[1]),(r,0,2),
(t,0,2*pi),color='red',figsize=[3,3,3])
show(B+GB)
```



O teorema dos incrementos finitos asegura que

$$||G(\mathbf{x}_1)-G(\mathbf{x}_2)||\leq||DG(\mathbf{x})||\cdot||\mathbf{x}_1-\mathbf{x}_2||$$

para algún $\mathbf{x}\in[\mathbf{x}_1,\mathbf{x}_2]$ e podemos ver que $G:B\to B$ é contractiva con $C=0.4$.

```
G(x,y)=[(x^2+y^2+8)/10,(x*y^2+x+8)/10]
J=[]
for a in [-1,-.99..1]:
    for b in [-1,.99..1]:
        J.append(norm(jacobian(G,[x,y])(a,b)))
max(J)
```

```
0.4
```

Podemos achar o único punto fixo de G en B con tolerancia $T=10^{-16}$ e punto inicial $P=(0,0)$, en 38 iteracións:

```
Banach(G,vector([0,0]),10^(-16))
```

```
((1.00000000000000, 1.00000000000000), 38)
```

Método de Newton

Sexan X, Y, espazos normados completos, $\Omega \subset X$ aberto e $f : \Omega \to Y$ diferenciable.

Se queremos resolver o problema inverso $f(x) = b$ e coñecemos a función nun punto $x_0 \in \Omega$ podemos utilizar que

$$f(x) \approx f(x_0) + Df(x_0)(x - x_0)$$

e suscitar o problema inverso lineal

$$Df(x_0)x = b - f(x_0) + Df(x_0)x_0$$

Se unha solución ou mellor aproximación $x_1 \in \Omega$ deste, cumprise que $\|b - f(x_1)\| < \|b - f(x_0)\|$ podemos repetir o proceso e suscitar un segundo problema inverso lineal

$$Df(x_1)x = b - f(x_1) + Df(x_1)x_1$$

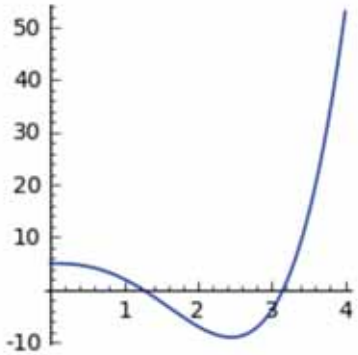
Se unha solución ou mellor aproximación $x_2 \in \Omega$ deste cumprise que $\|b - f(x_2)\| < \|b - f(x_1)\|$ podemos repetir o proceso e suscitar un terceiro problema inverso lineal....

Podemos obter así unha sucesión $(x_n) \subset \Omega$ tal que $\|b - f(x_n)\|$ chegue a ser menor que a tolerancia T permitida no noso problema inicial mediante a función **newton**(f,x_0,T).

Exemplo 1:

Achar as raíces reais do polinomio $p(x) = x^4 - 3x^3 - x^2 + 5$.

```
p(x)=[x^4-3*x^3-x^2+5]
plot(p[0],(x,0,4),figsize=[2.5,2.5])
```




```
P=vector([1])
show(newton(p,P,10^(-12)))
P=vector([4])
show(newton(p,P,10^(-12)))
```

$[(1.25223051069514), 4]$

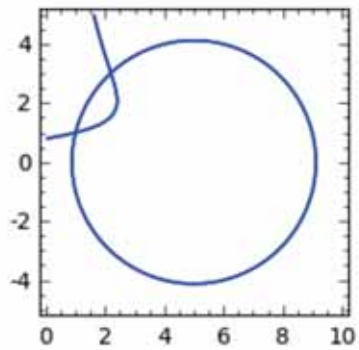
$[(3.15789345364414), 6]$

Exemplo 2:

Achar as solucións reais do sistema de ecuacións non lineais:

$$\begin{cases} x^2 - 10x + y^2 + 8 = 0 \\ xy^2 + x - 10y + 8 = 0 \end{cases}$$

```
f(x,y)=[x^2-10*x+y^2+8,x*y^2+x-10*y+8]
F0=implicit_plot(f[0], (x, 0, 10), (y, -5,5))
F1=implicit_plot(f[1], (x, 0, 10), (y, -5,5))
show(F0+F1,figsize=[2.5,2.5])
```



O debuxo invítanos a iniciar a procura das dúas solucións reais desde os puntos (0,0) e (4,4):

```
P=vector([0,0])
show(newton(f,P,10^(-12))[0])
P=vector([4,4])
show(newton(f,P,10^(-12))[0])
```

$(1.00000000000000, 1.00000000000000)$

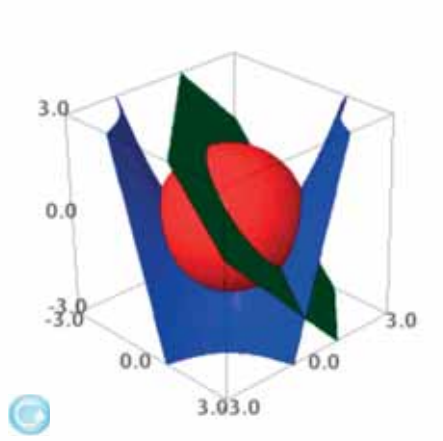
$(2.19343941541531, 3.02046646812303)$

Exemplo 3:

Achar as solucións reais do sistema de ecuacións non lineais:

$$\begin{cases} x^2+y^2+z^2=4 \\ xy-z=1 \\ x+y+z=1 \end{cases}$$

```
f(x,y,z)=[x^2+y^2+z^2-4,x*y-z-1,x+y+z-1]
D=implicit_plot3d(f[0],(x,-3,3),(y,-3,3),
(z,-3,3),color='red',figsize=[3,3,3])
DD=implicit_plot3d(f[1],(x,-3,3),(y,-3,3),
(z,-3,3),color='blue',figsize=[3,3,3])
DDD=implicit_plot3d(f[2],(x,-3,3),(y,-3,3),
(z,-3,3),color='green',figsize=[3,3,3])
show(D+DD+DDD)
```



O debuxo invítanos a iniciar a procura das dúas solucións reais nos puntos (2,0,-1) e (0,2,-1)

```
P=vector([2,0,-1])
show(newton(f,P,10^(-12))[0])
P=vector([0,2,-1])
show(newton(f,P,10^(-12))[0])
```

$$(1.79902808754467,\ 0.0718006058423045,\ -0.870828693386971)$$

$$(0.0718006058423045,\ 1.79902808754467,\ -0.870828693386971)$$

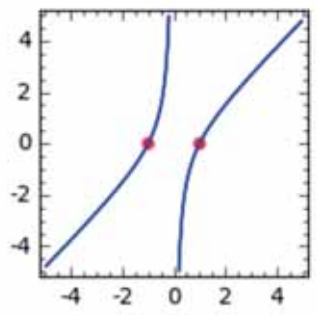
Exemplo 4:

Achar as solucións da ecuación non lineal:

$$x^2-xy=1$$

A seguinte gráfica dános todos os puntos do plano que verifican a ecuación:

```
f(x,y)=[x^2-x*y-1]
show(implicit_plot(f[0],(x,-5,5),
(y,-5,5))+point((-1,0),color='red',pointsize=30)+
point((1,0),color='red',pointsize=30),figsize=[2.2,2.2])
```



Co método de Newton podemos obter algunha solución. Por exemplo, iniciando en (1.3,1) e (1.3,3) obtemos os puntos marcados en vermello

```
P=vector([1.3,1])
show(newton(f,P,10^(-16)))
P=vector([1.3,3])
show(newton(f,P,10^(-16)))
```

$$[(-1.00000000000000,\ 0.000000000000000),\ 9]$$

$$[(1.00000000000000,\ 0.000000000000000),\ 10]$$

Con todo, o método pode non funcionar aínda que tomemos o punto inicial moi próximo a unha solución.

```
P=vector([2,1.5])
show(newton(f,P,10^(-16)))
P=vector([1.99,1.5])
show(newton(f,P,10^(-16)))
```

$$[(2.00000000000000,\ 1.50000000000000),\ 0]$$

```
Traceback (click to the left of this block for traceback)
...
ValueError: matrix equation has no solutions
```

Ordes directas de Sage:

A orde p.roots(), pode darnos as raíces do polinomio p coas súas multiplicidades, só algunhas ou non atopar ningunha.

Exemplo 1:

Achar as raíces do polinomio $p(x) = x^7 - 1$.

```
p(x)=x^7-1
p.roots()
```

```
[(1, 1)]
```

Con Sage 6.3 esta mesma orde dábanos as 7 raíces que sabemos existen:

```
[(e^(2/7*I*pi), 1), (e^(4/7*I*pi), 1), (e^(6/7*I*pi), 1),
(e^(-6/7*I*pi), 1), (e^(-4/7*I*pi), 1), (e^(-2/7*I*pi), 1), (1, 1)]
```

Exemplo 2:

Achar as raíces do polinomio $p(x) = x^4 - 3x^3 - x^2 + 5$.

```
p(x)=x^4-3*x^3-x^2+5
R=p.roots()
[(a[0].n(),a[1]) for a in R]
```

```
[(1.25223051069514, 1), (3.15789345364414, 1)]
```

Con Sage 6.3 esta mesma orde dábanos as 4 raíces que sabemos existen:

```
[(-0.705061982169642 - 0.875955796226703*I, 1), (-0.705061982169642 +
0.875955796226703*I, 1), (1.25223051069514, 1), (3.15789345364414, 1)]
```

Exemplo 3:

Achar as raíces do polinomio $p(x) = x^7 + x^4 - x - 1$.

```
p(x)=x^7+x^4-x-1
p.roots()
```

```
[(1, 1), (-1, 1)]
```

Exemplo 4:

Achar as raices do polinomio $x^7 + x^4 - x + 1$.

```
p(x)=x^7+x^4-x+1
p.roots()
```

```
Traceback (click to the left of this block for traceback)
...
RuntimeError: no explicit roots found
```

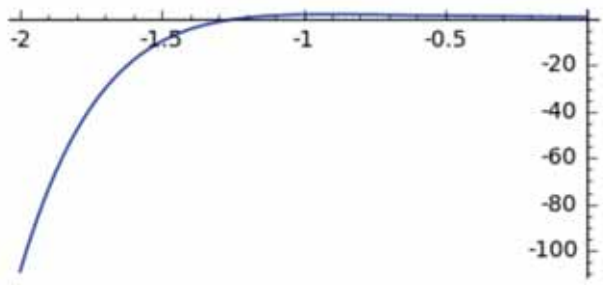
A orde f.**find_root(**a,b) busca unha solución do problema inverso f(x)=0 sendo $f:(a,b)\to\mathbb{R}$

Exemplo 1:

Achar a raíz real do polinomio $p(x)=x^7+x^4-x+1$ en (-2,0).

```
f(x)=x^7+x^4-x+1
show(plot(f,(x,-2,0),figsize=[4,2]))
f.find_root(-2,0)
```

```
-1.245640190632102
```

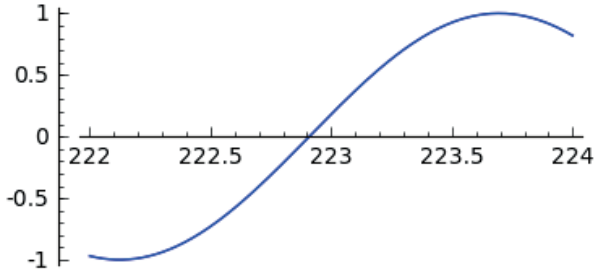


Exemplo 2:

Achar a raíz de cos(2x+5)=0 máis próxima a 223:

```
f(x)=cos(2*x+5)
show(plot(f,(x,222,224)),figsize=[4,2])
f.find_root(222,223)
```

```
222.90927289506766
```



A orde **solve**(f,x,solution_dict=True) busca todas as solucións do problema inverso f(x)=0. Se as atopa, dánolas en forma de dicionario.

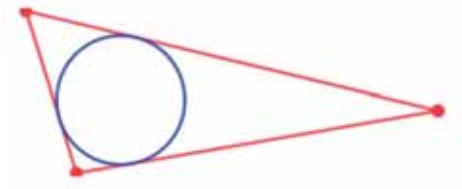
As nosas funcións **circuncentro**(T), **incentro**(T) e **ortocentro**(T) utilizan **solve** para determinar estes puntos distinguidos dun triángulo T.

Exemplo 1:

Xerar un triángulo calquera T en $[-10,10]\times[-10,10]$ e debuxalo coa súa

circunferencia inscrita.

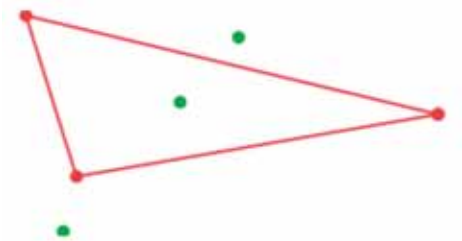
```
T=listacomplejos(10,3)
IN=incentro(T)
show(pt(T)+circle(ri(IN[0]),IN[1]),figsize=[3,3])
```



Exemplo 2:

Comprobar graficamente que o baricentro, o circuncentro e o ortocentro de T están aliñados (Euler 1765)

```
B=baricentro(T)
C=circuncentro(T)
OR=ortocentro(T)
show(plse([B,C[0],OR],.3)+pt(T),figsize=[3,3])
```



Exemplo 3:

Achar os términos xerais das sucesións **numpolc**(k,n) para k=3,4,5 e 6.

Solución:

Xa dixemos no Worksheet 0 que os devanditos términos xerais deben ser da forma $P(n) = an^2 + bn + c$. Podemos determinar as variables [a,b,c] para que, en cada caso, $[P(1), P(2), P(3)]$ coincida con [len(numpolc(k,n)) for n in range(1,4)]:

```
var('a b c')
V=[a,b,c]
for k in range(3,7):
    S=[len(numpolc(k,n)) for n in range(1,4)]
    E1=a+b+c-S[0]
    E2=4*a+2*b+c-S[1]
    E3=9*a+3*b+c-S[2]
    R=solve([E1,E2,E3],V,solution_dict=True)
```

```
    P(n)=R[0][a]*n^2+R[0][b]*n+R[0][c]
    print k,'-gonales centrados'
    show(P(n))
```

```
3 -gonales centrados
```

$$\frac{3}{2}n^2-\frac{3}{2}n+1$$

```
4 -gonales centrados
```

$$2n^2-2n+1$$

```
5 -gonales centrados
```

$$\frac{5}{2}n^2-\frac{5}{2}n+1$$

```
6 -gonales centrados
```

$$3n^2-3n+1$$

En consecuencia, a sucesión **numpolc**(k,n) para $n=1,2,3,\ldots$ é unha sucesión aritmética de orde 2 e diferencia k.

Exemplo 4:

Achar as raíces do polinomio $p(x)=x^4-3x^3-x^2+5$.

```
p(x)=x^4-3*x^3-x^2+5
S=solve(p,x,solution_dict=True)
n=len(S)
for i in range(n):
    show(S[i][x].n())
```

```
1.25223051069514
```

```
3.15789345364414
```

Exemplo 5:

Tratar con **solve**() o segundo exemplo do método de Newton:

$$\begin{cases} x^2-10x+y^2+8=0 \\ xy^2+x-10y+8=0 \end{cases}$$

```
f(x,y)=[x^2-10*x+y^2+8,x*y^2+x-10*y+8]
S=solve([x^2-10*x+y^2+8,x*y^2+x-10*y+8],
[x,y],solution_dict=True)
print 'número de solucións=', len(S)
for i in range(len(S)):
    print ((S[i][x],S[i][y]))
```

```
número de solucións= 6
(1, 1)
(2.19343945972, 3.02046644455)
(-0.878299184279 + 1.94927226015*I, -2.54686272964 - 4.4990275304*I)
(-0.878299184279 - 1.94927226015*I, -2.54686272964 + 4.4990275304*I)
(9.28157947657 + 0.158281165555*I, 0.536629495575 - 1.26287018428*I)
(9.28157947657 - 0.158281165555*I, 0.536629495575 + 1.26287018428*I)
```

Exemplo 6:

Tratar con **solve**() o terceiro exemplo do método de Newton:

$$\begin{cases} x^2+y^2+z^2=4 \\ xy-z=1 \\ x+y+z=1 \end{cases}$$

```
f(x,y,z)=[x^2+y^2+z^2-4,x*y-z-1,x+y+z-1]
S=solve([x^2+y^2+z^2-4,x*y-z-1,x+y+z-
1],[x,y,z],solution_dict=True)
print 'número de solucións=', len(S)
for i in range(len(S)):
    print ((S[i][x],S[i][y],S[i][z]))
```

```
número de solucións= 4
(-0.935414346693 + 1.73084623621*I, -0.935414346693 - 1.73084623621*I,
2.87082869339 + 6.66133814775e-16*I)
(-0.935414346693 - 1.73084623621*I, -0.935414346693 + 1.73084623621*I,
2.87082869339 + 2.6645352591e-15*I)
(0.0718006061854, 1.79902811524, -0.870828676353)
(1.79902811524, 0.0718006061854, -0.870828676353)
```

Exemplo 7:

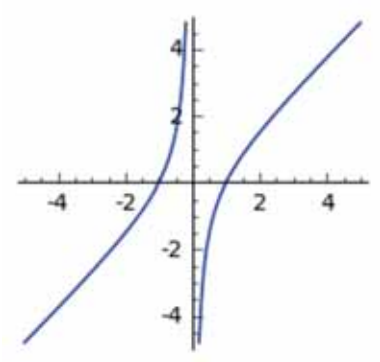
Tratar con **solve**() o cuarto exemplo do método de Newton:

$$x^2-xy=1$$

Resolvendo en función de y obtemos a solución explícita $y=F(x)$ cuxa gráfica coincide co conxunto de solucións da ecuación de partida:

```
f(x,y)=[x^2-x*y-1]
S=solve(f[0],y,solution_dict=True)
F(x)=S[0][y]
print 'F(x)=',S[0][y]
show(plot(F,(x,-5,-0.2))+plot(F,
(x,0.2,5),aspect_ratio=1),figsize=[2.5,2.5])
```

```
F(x)= (x^2 - 1)/x
```



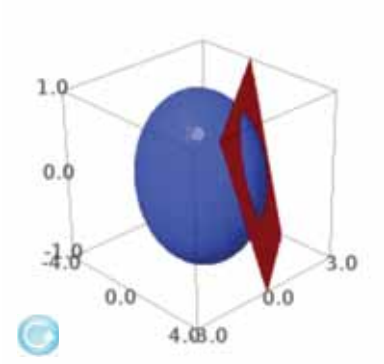
Exemplo 8:

Achar as solucións do sistema de ecuacións non lineais

$$\begin{cases} 4x^2 + 9y^2 + 36z^2 = 36 \\ x + y + z = 3 \end{cases}$$

Evidentemente as solucións son os puntos da elipse intersección do elipsoide e o plano:

```
implicit_plot3d(4*x^2+9*y^2+36*z^2-36,(x,-4,4),(y,-3,3),
(z,-1,1))+implicit_plot3d(x+y+z-3,(x,-4,4),(y,-3,3),(z,-
1,1),color='red',figsize=[2.5,2.5,2.5])
```



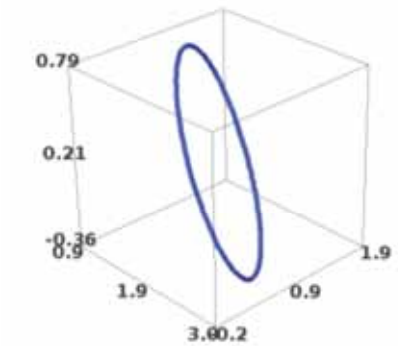
Para achar a expresión paramétrica desta elipse utilizamos dúas cartas:

```
f(x,y,z)=[4*x^2+9*y^2+36*z^2-36,x+y+z-3]
S=solve(list(f(x,y,z)),[x,z],solution_dict=True)
C0(y)=[S[0][x],y,3-S[0][x]-y]
C1(y)=[S[1][x],y,3-S[1][x]-y]
show(C0)
show(C1)
parametric_plot3d(C0,(y,-.153,1.86729),thickness=7,figsize=
[2,2,2])+parametric_plot3d(C1,
(y,-.153,1.86729),thickness=7,figsize=[2,2,2])
```

$$y \mapsto \left(-\frac{9}{10}\,y-\frac{3}{20}\,\sqrt{-14y^2+24y+4}+\frac{27}{10},\,y,\,-\frac{1}{10}\,y+\frac{3}{20}\,\sqrt{-14y^2+24y+4}+\frac{3}{10}\right)$$

$$y \mapsto \left(-\frac{9}{10}\,y+\frac{3}{20}\,\sqrt{-14y^2+24y+4}+\frac{27}{10},\,y,\,-\frac{1}{10}\,y-\frac{3}{20}\,\sqrt{-14y^2+24y+4}+\frac{3}{10}\right)$$

Sleeping... Make Interactive



Exemplo 9:

Achar as solucións do sistema de ecuacións non lineais

$$\begin{cases} x^2+t^3=1 \\ xy+zt=2 \\ x+y+zt=3 \end{cases}$$

Facendo un **solve** en [x,y,z] obtemos:

```
f(x,y,z,t)=[x^2+t^3-1,x*y+z*t-2,x+y+z*t-3]
S=solve(list(f(x,y,z,t)),[x,y,z],solution_dict=True); show(S)
```

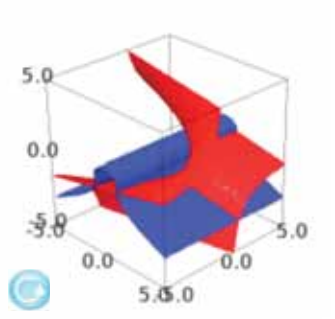
$$\left[\left\{x:\sqrt{-t^3+1},\,z:-\frac{\sqrt{t^2+t+1}\sqrt{-t+1}-2}{t},\,y:1\right\},\left\{x:-\sqrt{-t^3+1},\,z:\frac{\sqrt{t^2+t+1}\sqrt{-t+1}+2}{t},\,y:1\right\}\right]$$

Como as solucións cumpren que y=1, o sistema anterior pódese reducir ao seguinte

$$\begin{cases} x^2+t^3=1 \\ x+zt=2 \end{cases}$$

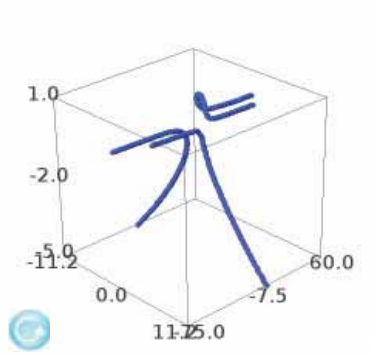
cuxas solucións son os puntos da intersección de dúas superficies en $\mathbb{R}^3$;

```
A=implicit_plot3d(x^2+t^3-1,(x,-5,5),(z,-5,5),(t,-5,5),figsize=
[2.2,2.2,2.2])
B=implicit_plot3d(x+z*t-2,(x,-5,5),(z,-5,5),(t,-
5,5),color='red',figsize=[2.2,2.2,2.2])
show(A+B)
```



Para achar a expresión paramétrica desta intersección utilizamos as cartas:

```
C0(t)=[S[0][x],S[0][z],t]
C1(t)=[S[1][x],S[1][z],t]
A=parametric_plot3d(C0,(t,-5,-.02),thickness=7,figsize=
[2.5,2.5,2.5])
B=parametric_plot3d(C0,(t,.02,1),thickness=7,figsize=
[2.5,2.5,2.5])
C=parametric_plot3d(C1,(t,-5,-.04),thickness=7,figsize=
[2.5,2.5,2.5])
D=parametric_plot3d(C1,(t,.05,1),thickness=7,figsize=
[2.5,2.5,2.5])
show(A+B+C+D)
```



```
L=list(C0(t))
V0=vector([L[0]]+[1]+L[1:])
L=list(C1(t))
V1=vector([L[0]]+[1]+L[1:])
show([V0,V1])
```

$$\left[\left(\sqrt{-t^3+1},\,1,\,-\frac{\sqrt{t^2+t+1}\sqrt{-t+1}-2}{t},\,t\right),\left(-\sqrt{-t^3+1},\,1,\,\frac{\sqrt{t^2+t+1}\sqrt{-t+1}+2}{t},\,t\right)\right]$$

```
show(EV(f,V0)(t=-4).n())
show(EV(f,V1)(t=-7).n())
```

(0.000000000000000, 0.000000000000000, 0.000000000000000)

(0.000000000000000, 0.000000000000000, 0.000000000000000)

3.2.1.3. Mellor aproximación dun problema inverso non lineal

Cando non logramos achar unha solución do problema $f(x)=b$ podemos plantexarnos achar os mínimos da función

$$Q:\Omega \overset{b-f}{\to} Y \overset{\|\ \|^2}{\to} \mathbb{R} \quad \text{con} \quad Q(x)=\|b-fx\|^2 \quad \forall x\in\Omega$$

sendo $\|\ \|$ a norma euclídea de Y. Utilizamos a función **RASNL**(f,b).

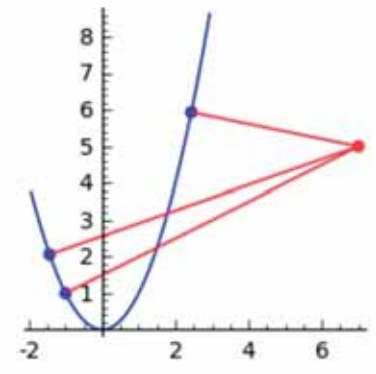
Exemplo 1:

Achar a distancia do punto (7,5) á parábola $y=x^2$:

```
f(t)=[t,t^2]
P=[7,5]
b=vector(P)
F=RASNL(f,b)
S=solve(F,t,solution_dict=True)
n=len(S)
R=[]
for k in range(n):
    R.append(S[k][t].n())
R.sort()
m=min(R)-.5
M=max(R)+.5
DC=parametric_plot(f,(t,m,M),aspect_ratio=1)
PC=[list(f(r)) for r in R]
```

```
DPC=point(PC,hue=.7,pointsize=30)+point(P,hue=1,pointsize=30)
L=[line([PC[k],P],hue=1,thickness=1) for k in range(n)]
show(DC+DPC+sum(L),figsize=[2.5,2.5])
print 'distancia=lonxitude do mínimo segmento=', abs(PC[2]
[0]+I*PC[2][1]-(P[0]+I*P[1]))
```

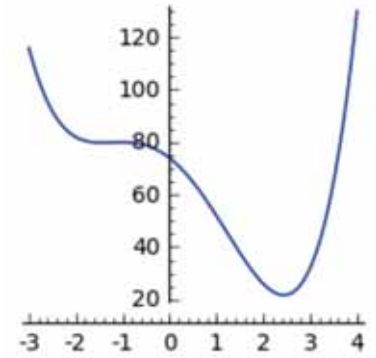
```
distancia=lonxitude do mínimo segmento= 4.65860761423887
```



Outra forma simplificada de resolver o problema é:

```
P(t)=(t-7)^2+(t^2-5)^2
show(plot(P,(t,-3,4)),figsize=[2.5,2.5])
Pprime=derivative(P)
t0=Pprime.find_root(2,3)
print 'distancia=',sqrt(P(t0)).n()
```

```
distancia= 4.65860761423887
```



Exemplo 2:

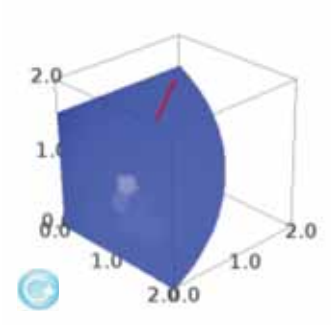
Achar a distancia do punto (1,1,2) ao helicoide

$$x = r\cos(t), \quad y = r\sin(t), \quad z = t.$$

```
f(r,t)=[r*cos(t),r*sin(t),t]
b=vector([1,1,2])
```

```
af(r,t)=RASNL(f,b)
N=newton(af,[1,pi/2],10^(-6))[0]
PU=EV(f,N)
print 'distancia=',norma(PU-b)
S=parametric_plot3d(f,(r,0,2),(t,0,pi/2),figsize=[2.2,2.2,2.2])
L=line3d([list(PU),list(b)],rgbcolor=
(1,0,0),thickness=.3,figsize=[2.2,2.2,2.2])
show(S+L)
```

```
distancia= 0.981249353983967
```

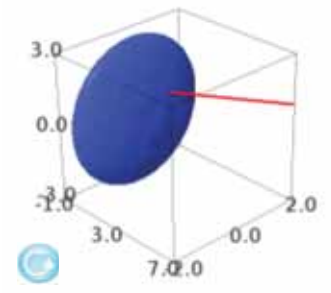


Exemplo 3:

Achar a distancia do punto (7,2,1) ao elipsoide $x^2 + \frac{y^2}{4} + \frac{z^2}{9} = 1$.

```
f(a,b)=[cos(a)*cos(b),2*cos(a)*sin(b),3*sin(a)]
v=vector([7,2,1])
af(a,b)=RASNL(f,v)
N=newton(af,[0,pi/2],10^(-6))[0]
PU=EV(f,N)
print 'distancia=',norma(PU-v)
S=parametric_plot3d(f,(a,-pi,pi),(b,0,2*pi),figsize=
[2.2,2.2,2.2])
L=line3d([list(PU),list(v)],rgbcolor=
(1,0,0),thickness=.3,figsize=[2.2,2.2,2.2])
show(S+L)
```

```
distancia= 6.23440479101348
```



3.2.2. Problemas de axuste

Sexa Ω un aberto de $\mathbb{R}^n$, $I \subset \mathbb{R}$ un intervalo e $\Psi : \Omega \to \mathcal{C}(I)$ unha carta local da variedade diferenciable de funcións reais

$$\{\Psi(\mathbf{w}) : I \to \mathbb{R} \quad | \quad \mathbf{w} \in \Omega\}$$

Iso quere dicir que o conxunto de derivadas parciais

$$\left\{ \frac{\partial \Psi}{\partial w_i}(\mathbf{w}) : I \to \mathbb{R} \quad | \quad i = 1, \cdots, n \right\}$$

constitúe unha base do espazo tanxente á variedade no punto $\Psi(\mathbf{w})$.

Achar a función da variedade que mellor aproxima en norma euclídea os datos $X = [x_1, \cdots, x_k]$ e $Y = [y_1, \cdots, y_k]$.

Solución:

Necesitamos achar o $\mathbf{w}_0 \in \Omega$ que minimice a función

$$\Phi(\mathbf{w}) = (\mathbf{y}|\mathbf{y}) - \mathbf{2}(\mathbf{\Psi}(\mathbf{w})(\mathbf{x})|\mathbf{y}) + (\mathbf{\Psi}(\mathbf{w})(\mathbf{x})|\mathbf{\Psi}(\mathbf{w})(\mathbf{x}))$$

Así pois, en $\mathbf{w}_0 \in \Omega$ debe anularse a función gradiente $\nabla\Phi : \Omega \to \mathbb{R}^n$ e, xa que logo, debe cumprirse o sistema de n ecuacións:

$$\frac{\partial \Psi}{\partial w_i}(\mathbf{w}_0)(\mathbf{x}) \cdot (\mathbf{y} - \Psi(\mathbf{w}_0)(\mathbf{x})) = 0 \quad \forall i = 1, \cdots, n$$

Iso quere decir que $\mathbf{w}_0$ é un punto de Ω para o que se cumpre que o vector $\Psi(\mathbf{w}_0)(\mathbf{x}) - \mathbf{y}$ é ortogonal ao espazo tanxente á variedade en $\Psi(\mathbf{w}_0)$.

Un caso especialmente sinxelo é o dunha variedade lineal de funcións $F = [f_1, \cdots, f_n]$. En cada punto

$$\Psi(\mathbf{w}) = w_1 f_1 + \cdots + w_n f_n$$

o espazo tanxente é a propia variedade lineal F e o cálculo de $\mathbf{w}_0$ realízase mediante a nosa función **nube(**X,Y,F**).**

Outro caso interesante é o de unha variedade de funcións que seguen un modelo f como, por exemplo,

$$f(\mathbf{w}, x) = \frac{w_1 x + w_2}{3 + cos(w_3 x) + sin(w_4 x)}.$$

A nosa función **ortotangente**(X,Y,f) obtén as n expresións dependentes de $\mathbf{w}$

$$\frac{\partial \Psi}{\partial w_i}(\mathbf{w})(x_1) \cdot (\Psi(\mathbf{w})(x_1) - y_1) + \cdots + \frac{\partial \Psi}{\partial w_i}(\mathbf{w})(x_k) \cdot (\Psi(\mathbf{w})(x_k) - y_k) \quad \forall i = 1$$

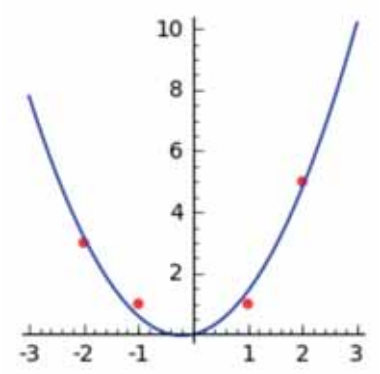
e definindo a función $af(\mathbf{w}) = ortotangente(X, Y, f)$ podemos obter unha solución $\mathbf{w}_0$ facendo un **newton**(af,P,T) cun adecuado inicio P. Podémolo buscar mediante a función **ajustamodel**(X,Y,m,h) baseada na orde importada de Scipy $find\_fit$.

Ejemplo 1:

Achar o polinomio de grado 2 que mellor axusta os seguintes datos X=[-2,-1,1,2] Y=[3,1,1,5].

```
X=[-2,-1,1,2]; Y=[3,1,1,5]
F(x)=[1,x,x^2]
N=nube(X,Y,F)
show(N[0]+N[1],figsize=[2.5,2.5])
show(N[2])
```

$$1.00000000000000\,x^2 + 0.400000000000000\,x$$

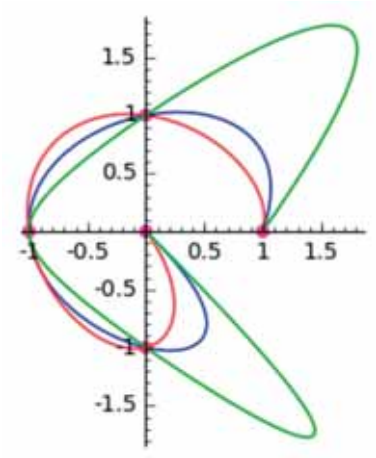


Exemplo 2:

Achar tres curvas $c : [0, 4] \to \mathbb{R}^2$ tales que $c(0) = (1, 0)$, $c(1) = (0, 1)$, $c(2) = (-1, 0)$, $c(3) = (0, -1)$ e $c(4) = (0, 0)$.

```
X=[0,1,2,3,4]
AB=[1,0,-1,0,0]
OR=[0,1,0,-1,0]
F1(x)=[1,x,x^2,x^3,x^4]
c1(x)=[nube(X,AB,F1)[2],nube(X,OR,F1)[2]]
C1=parametric_plot(c1,
(x,0,4))+point(zip(AB,OR),hue=0,pointsize=30)
F2(x)=[1,sin(x),sin(2*x),cos(x),cos(2*x)]
c2(x)=[nube(X,AB,F2)[2],nube(X,OR,F2)[2]]
C2=parametric_plot(c2,
(x,0,4),hue=0)+point(zip(AB,OR),hue=.7,pointsize=30)
F3(x)=[1,exp(2*x),exp(-2*x),exp(3*x),exp(-3*x)]
c3(x)=[nube(X,AB,F3)[2],nube(X,OR,F3)[2]]
C3=parametric_plot(c3,
```

```
(x,0,4),hue=.3)+point(zip(AB,OR),hue=.9,pointsize=30)
show(C1+C2+C3,figsize=[3,3])
```

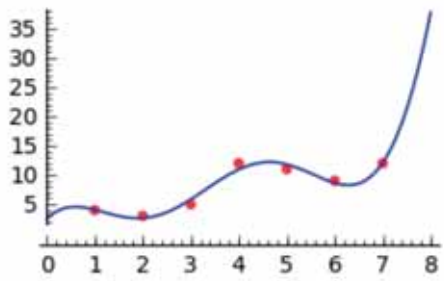


Exemplo 3:

Achar a función da variedade lineal $[x, x^2, x^3, sin(x), cos(x)]$ que mellor axusta os datos X=[1,2,3,4,5,6,7], Y=[4,3,5,12,11,9,12].

```
F(x)=[x,x^2,x^3,sin(x), cos(x)]
X=[1,2,3,4,5,6,7]
Y=[4,3,5,12,11,9,12]
N=nube(X,Y,F)
show(N[0]+N[1],figsize=[3,2])
show(vector(Round(list(N[3]),5))*vector(F(x)))
```

$$0.77247\,x^3 - 7.91034\,x^2 + 20.12104\,x + 2.51303\,\cos(x) - 12.20553\,\sin(x)$$

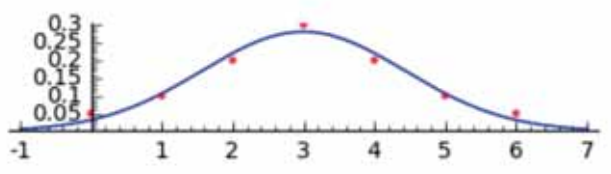


Exemplo 4:

Axustar mediante unha campá de Gauss, os datos $X = [0, 1, 2, 3, 4, 5, 6]$ e $Y = [0.05, 0.1, 0.2, 0.3, 0.2, 0.1, 0.05]$.

```
f(x,a,b)=(1/(2*pi*b)^(1/2))*exp(-((x-a)/b)^2)
X=[0,1,2,3,4,5,6]
Y=[.05,.1,.2,.3,.2,.1,.05]
af(a,b)=ortotangente(X,Y,f)
N=newton(af,[3,1],10^(-6))
g=EtV(f,x,N[0])
print 'a=',N[0][0], 'b=',N[0][1]
DC=plot(g,(x,min(X)-1,max(X)+1))
D=[[X[i],Y[i]] for i in range(len(X))]
DD=point2d(D,rgbcolor=(1,0,0),pointsize=10,aspect_ratio=1)
show(DC+DD,aspect_ratio=5,figsize=[4,3])
```

```
a= 3.00000000000000 b= 2.02934466671219
```



Exemplo 5:

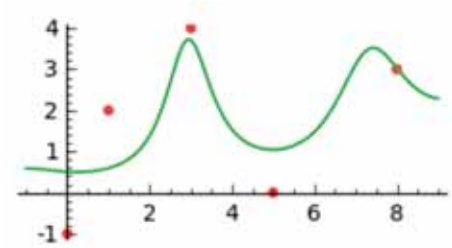
Axustar os datos X=[0,1,3,5,8], Y=[-1,2,4,0,3] ao seguinte modelo

$$f(x) = \frac{ax + b}{3 + cos(cx) + sin(dx)}$$

```
f(x,a,b,c,d)=(a*x+b)/(3+cos(c*x)+sin(d*x))
X=[0,1,3,5,8]
Y=[-1,2,4,0,3]
m(x)=f
S=ajustamodel(X,Y,m,.3)
html("<h4> Con ajustamodel obtemos:</h4>")
show(S[0]+S[1],figsize=[3,3])
show(S[2])
```

Con ajustamodel obtemos:

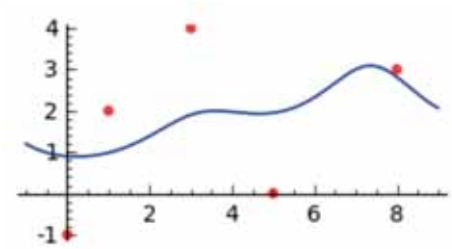
$$\frac{0.592915378627\,x + 2.03094382541}{\cos\left(1.12530942358\,x\right) + \sin\left(1.58966064275\,x\right) + 3}$$




```
html("<h4> Con ortotangente e inicio P=[1,1,1,1] obtemos un
distinto:</h4>")
D=[[X[i],Y[i]] for i in range(len(X))]
af(a,b,c,d)=ortotangente(X,Y,f)
N=newton(af,[1,1,1,1],10^(-6))
g=EtV(f,x,N[0])
DC=plot(g,(x,min(X)-1,max(X)+1))
DD=point(D,hue=1,pointsize=20,aspect_ratio=1)
show(DC+DD,figsize=[3,3])
show(g)
```

Con ortotangente e inicio P=[1,1,1,1] obtemos un distinto:

$$\frac{0.462095097136045\,x + 3.59610370016335}{\cos\left(1.19413556903563\,x\right) + \sin\left(0.863007441853387\,x\right) + 3}$$

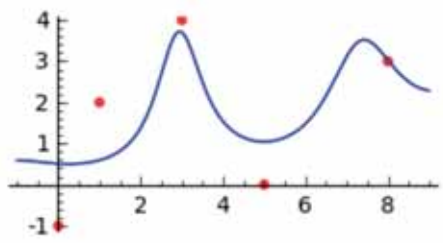



```
html("<h4> Con ortotangente e inicio P=[.6,2,1,1.5] obtemos
</h4>")
html("<h4> un axuste igual ao primeiro:</h4>")
D=[[X[i],Y[i]] for i in range(len(X))]
af(a,b,c,d)=ortotangente(X,Y,f)
N=newton(af,[.6,2,1,1.5],10^(-6))
g=EtV(f,x,N[0])
DC=plot(g,(x,min(X)-1,max(X)+1))
DD=point(D,hue=1,pointsize=20,aspect_ratio=1)
show(DC+DD,figsize=[3,3])
show(g)
```

Con ortotangente e inicio P=[.6,2,1,1.5] obtemos

un axuste igual ao primeiro:

$$\frac{0.592833367431769\,x + 2.03122310394649}{\cos\left(1.12532306801961\,x\right) + \sin\left(1.58965224580056\,x\right) + 3}$$



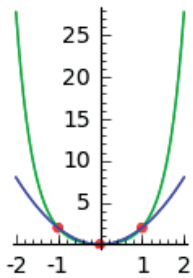
Exemplo 6:

Axusta os datos X=[-1,0,1], Y=[2,0,2] cunha catenaria

$$f(a,b,c,x) = \frac{cosh(ax+b)}{a} + c$$

e compara o resultado co de axustalos cunha parábola.

```
f(x,a,b,c)=c+cosh(a*x+b)/a
p(x,a,b,c)=a*x^2+b*x+c
cat(x)=f
par(x)=p
X=[-1,0,1]
Y=[2,0,2]
C=ajustamodel(X,Y,cat,.3)
P=ajustamodel(X,Y,par,.7)
show(C[0]+C[1]+P[0]+P[1],aspect_ratio=2/10,figsize=[3,2])
```



3.2.3.Exercicios

3.2.3.1. Exercicios Problemas Lineais

CONTROL DE SISTEMAS

En $\mathbb{R}^n$ chamamos sistema a un par de listas, unha de vectores $V = [v_1, \ldots, v_m]$ e outra de escalares $L = [r_1, \ldots, r_m]$.

Dicimos que o sistema $[V, L]$ é controlable se existe unha forma lineal $f : \mathbb{R}^n \to \mathbb{R}$ tal que $f(v_i) = r_i \quad \forall i = 1, \ldots, m.$

Un control de norma mínima chámase control óptimo.

Exercicio 1:

Dado o sistema en $\mathbb{R}^4$, V=[(1, 3, 5, 3), (2, 0, 12, 5), (−2, 7, 1, 3)] e L=[1, 3, 12]

1. Determinar o conxunto de controis.

2. Achar o control óptimo na norma euclídea.

3. Achar o control óptimo na norma 1.

```
V=[vector([1,3,5,3]),vector([2,0,12,5]),vector([-2,7,1,3])]
L=[1,3,12]
C=listasim('c',4)
VC=vector(C)
E0=V[0]*VC-L[0]
E1=V[1]*VC-L[1]
E2=V[2]*VC-L[2]
S=solve([E1,E2,E2],C,solution_dict=True)
print '1)','Os controis forman una variedade de dimensión 2'
CON=[S[0][c] for c in C]
f(r1,r2)=CON
show(f(r1,r2))
print '2)','O control óptimo na norma euclídea é'
g(X)=[E0,E1,E2]
A=JAC(g)
b=vector(L)
show(pinv(A)*matrix(b).transpose())
print '3)','A gráfica da norma 1 dos controis é'
N1(r1,r2)=sum([abs(S[0][c]) for c in C])
plot3d(N1,(r1,-3,3),(r2,-3,3),figsize=[3,3,3])
```

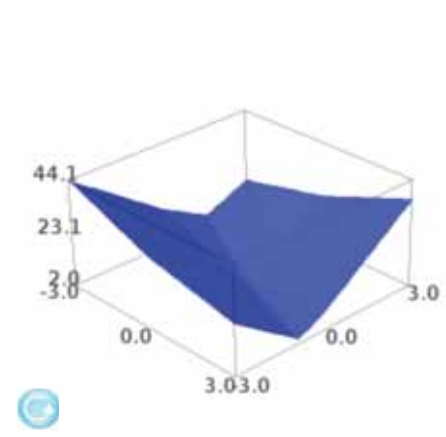
```
1) Os controis forman una variedade de dimensión 2
```

$$\left(-\frac{5}{2}\,r_1 - 6\,r_2 + \frac{3}{2},\ -\frac{8}{7}\,r_1 - \frac{13}{7}\,r_2 + \frac{15}{7},\ r_2,\ r_1\right)$$

2) O control óptimo na norma euclídea é

$$\begin{pmatrix} -4.64703777432 \\ -0.139743743464 \\ 0.595534585416 \\ 1.02953209728 \end{pmatrix}$$

3) A gráfica da norma 1 dos controis é



O mínimo alcánzase na contorna de $(0, 0)$. Para estimalo, tomamos o mínimo da lista:

```
VAL=[]
for j in [-.5,-.49,..,.5]:
    for k in [-.5,-.49,..,.5]:
        VAL=VAL+[[norma(f(j,k),1),[j,k]]]
min(VAL)
```

```
[1.92857142857143, [4.44089209850063e-16, 0.250000000000001]]
```

```
print 'O control óptimo é'
show(f(0,.25))
print 'e a súa norma 1 vale'
show(min(VAL)[0])
```

O control óptimo é

$(0.000000000000000, 1.67857142857143, 0.250000000000000, 0)$

e a súa norma 1 vale

1.92857142857143

REDES HIDRAÚLICAS

Un digrafo (V, E, w) pode ser interpretado como unha rede de tubos entre os vértices de V con caudais $w(u, v)$, se admitimos en cada vértice u a posibilidade dunha alimentación externa de caudal a_u e un desaugadoiro ao exterior de caudal d_u de modo que se cumpra a lei dos nodos de Kirchhoff:

$$(\star) \qquad \sum_{v \in V} w(v,u) - \sum_{v \in V} w(u,v) + a_u - d_u = 0 \quad \forall u \in V.$$

Sempre podemos elexir unha enumeración $(v_1, \cdots, v_n)$ de V tal que

$$a = (a_1, \cdots, a_i, 0, \cdots 0) \geq 0 \quad \text{e} \quad d = (0, \cdots, 0, d_{i+k}, \cdots, d_n) \geq 0$$

Se D é a matriz diagonal do vector d e M é a matriz do laplaciano de w, o sistema lineal $(\star)$ pódese escribir na forma

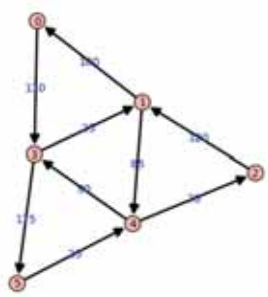
$$(M^t + D) \cdot 1 = a$$

Se nos caudais de alimentación $a_1, \ldots, a_i$ hai concentracións $e_1, \ldots, e_i$ de certa substancia, poderemos calcular as concentracións $c_1, \ldots, c_n$ de dita substancia nos vértices da rede. Eses valores virán dados pola función lineal

$F : \mathbb{R}^i \to \mathbb{R}^n \to \mathbb{R}^n$ tal que $e \mapsto \begin{pmatrix} e \\ 0 \end{pmatrix} \mapsto (M^t + D)^+ \cdot A \cdot \begin{pmatrix} e \\ 0 \end{pmatrix}$ sendo A a matriz diagonal de vector a e sendo $(M^t + D)^+$ a inversa de Moore-Penrose de $M^t + D$.

Exercicio 2:

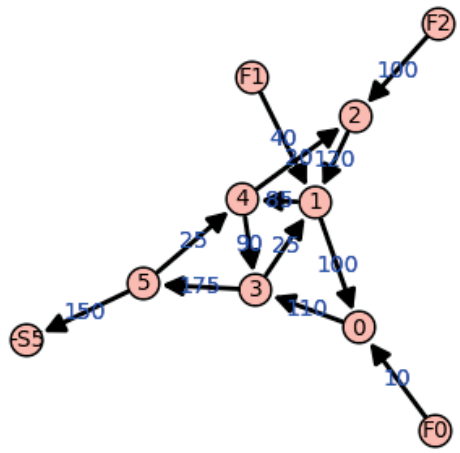
Engadir fontes e sumidoiros para convertir o digrafo



nunha rede hidraúlica cos caudais indicados.

```
G=[[1,0,100],[0,3,110],[3,1,25],[1,4,85],[2,1,120],[4,2,20],
[3,5,175],[4,3,90],[5,4,25]]
V=vertices(G)
AD=[sum([a[2] for a in G if a[0]==e])-sum([a[2] for a in G if
a[1]==e]) for e in V]
F=listasim('F',6)
```

```
S=listasim('S',6)
ka=[[F[k],k,AD[k]] for k in range(6) if AD[k]>0]
kd=[[k,-S[k],-AD[k]] for k in range(6) if AD[k]<0]
H=G+ka+kd
D=DiGraph(H)
D.graphplot(edge_labels=True).show(figsize=[3,3])
```



Exercicio 3:

Se nas fontes engadidas (F_0, F_1, F_2) hai concentracións de sal (c_0, c_1, c_2), cal é a concentración de sal en cada nodo?

Avalialas no caso particular $(30, 10, 3)$.

```
var('c0 c1 c2')
A=diagonal_matrix([a[2] for a in ka]+[0,0,0])
D=diagonal_matrix([0,0,0,0,0]+[d[2] for d in kd])
M=ktomatrix(laplacian(G)).transpose()
CN=pinv(M+D)*A*vector([c0,c1,c2,0,0,0]);CN
```

```
(0.10147046876269392*c0 + 0.25672272321065875*c1 +
0.6418068080266467*c2, 0.011617515638963346*c0 +
0.28239499553172476*c1 + 0.7059874888293116*c2,
0.0040214477211795 78*c0 + 0.04647006255585323*c1 +
0.9495084897229663*c2, 0.06666666666666667*c0 +
0.2666666666666666*c1 + 0.6666666666666663*c2,
0.024128686327077743*c0 + 0.2788203753351205*c1 +
0.6970509383378012*c2, 0.06666666666666664*c0 +
0.26666666666666655*c1 + 0.666666666666666*c2)
```

```
EV(CN,[30,10,3])
```

```
(7.536761719067345, 5.290437890974083, 3.4338695263628187,
6.666666666666665, 5.603217158176941, 6.666666666666625)
```

Exercicio 4:

Dúas empresas miden directamente as concentracións de sal nos nodos dándonos resultados diferentes:

$R_1 = [7.51, 5.30, 3.41, 6.71, 5.65, 6.72]$ $R_2 = [7.55, 5.28, 3.43, 6.61, 5.55, 6.0($

Con cal quedamos?

```
B=pinv(M+D)*A
b=vector([7.51, 5.30, 3.41, 6.71, 5.65, 6.72])
pinv(B)*b
```

```
(29.69049145189905, 10.244938784820667, 2.964184443911329, 0.(
0.0)
```

```
v=EV(CN,[29.6904914518990, 10.2449387848207, 2.96418444391133])
norma(v-b)
```

```
0.06222133817772289
```

```
B=pinv(M+D)*A
b=vector([7.55, 5.28, 3.43, 6.61, 5.55, 6.00])
pinv(B)*b
```

```
(28.691989994769337, 9.641789978644482, 3.01899573283203, 0.0,
0.0)
```

```
v=EV(CN,[32.0455623419476, 8.95430715547043, 3.23854211517815])
norma(v-b)
```

```
0.723591849599671
```

REDES ELÉCTRICAS

Unha rede eléctrica, con xeradores de tensión e resistencias, pode ser modelada por un grafo $G(V, A, r)$ simple e conexo onde os pesos das aristas son as resistencias. Alternativamente, se eliximos unha orientación en A tomando unha orde (e^-, e^+) en cada $\{e^-, e^+\} \in A$, podemos modelalo por un digrafo unidireccional $D(V, E, r)$.

Se o potencial eléctrico é $p : V \to \mathbb{R}$, a diferenza de potencial $\delta p : E \to \mathbb{R}$ onde $\delta p(e) = p(e^+) - p(e^-)$, produce unha corrente eléctrica cuxa intensidade $\omega : E \to \mathbb{R}$ debe cumprir as seguintes leis:

- Ohm:

$$\delta p(e) = r(e)\omega(e) \quad \forall e \in E$$

- Nodos de Kirchoff:

$$\sum_{v \in V} w(v, u) - \sum_{v \in V} w(u, v) = 0 \quad \forall u \in V.$$

- Ciclos de Kirchoff:

$$\delta p \in Z(D)^{\perp}$$

onde $Z(D)$ é o subespacio de ciclos do digrafo D que definimos como segue:

Un ciclo L de D é unha lista de arcos $[e_1, \ldots, e_k]$ tal que $e_i^+ = e_{i+1}^-$, $e_1{}^- = e_k{}^+$ e determina a función

$$g_L : E \to \mathbb{R} \quad \text{tal que} \quad g_L(e) = \begin{cases} 1 & \text{se} \quad e \in L \\ -1 & \text{se} \quad \top(e) \in L \\ 0 & \text{noutro caso} \end{cases}$$

Z(D) é o subespacio de $\mathbb{R}^E$ xerado por $\{g_L \mid L \text{ é ciclo de } D\}$.

O operador *diferencia* $\delta : R^V \to R^E$ tal que $\delta(p) = \delta p$ é claramente lineal e, fixada unha ordenación de vértices $(v_1, \ldots, v_n)$ e de arcos $(e_1, \ldots, e_m)$, ten por matriz asociada a trasposta da matriz de incidencia U que se define

$$U \in \mathcal{M}(n \times m) \quad \text{tal que} \quad U[i,j] = \begin{cases} 1 & \text{se} \quad v_i = e_j^- \\ -1 & \text{se} \quad v_i = e_j^+ \\ 0 & \text{noutro caso} \end{cases}$$

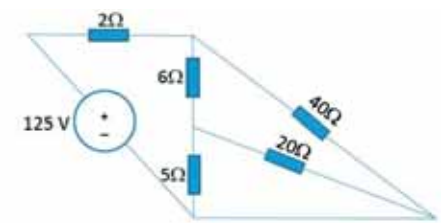
e calcúlase coa función **incidencia**(V,E).

É útil observar que $Z(D) = \ker \delta^\star$ e, xa que logo, $\dim Z(D) = |E| - rank(U)$.

En lugar de xeradores de tensión podemos usar xeradores de corrente. En tal caso, é necesario saber que un xerador de tensión de T voltios e unha resistencia de R_T ohmios montados en serie, equivale a un xerador de corrente de I amperios e unha resistencia de R_I ohmios montados en paralelo, sempre que $R_I = R_T \quad \text{e} \quad I = \frac{T}{R_T}$.

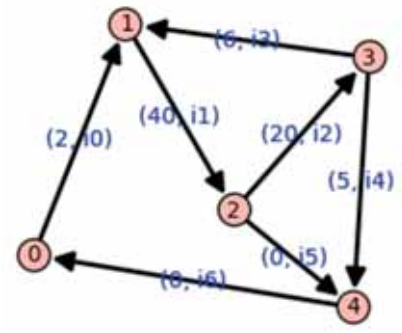
Exercicio 5:

Estudar o circuíto eléctrico:



Podemos considerar o circuíto como un digrafo unidireccional con nodos [0,1,2,3,4] situados, respectivamente, no vértice superior esquerdo, superior dereito, inferior dereito, medio central e inferior esquerdo. Nas súas aristas poñeremos etiquetas coa resistencia e a intensidade.

```
W=listasim('i',7)
RE=[[0,1,(2,i0)],[1,2,(40,i1)],[2,3,(20,i2)],[3,1,(6,i3)],
[3,4,(5,i4)],[2,4,(0,i5)],[4,0,(0,i6)]]
D=DiGraph(RE)
D.graphplot(edge_labels=True).show(figsize=[2.5,2.5])
```



Sobre este esquema podemos escribir as ecuacións N0, N1, N2, N3 , N4, dadas pola lei de nodos de Kirchoff.

```
N0=i6-i0
N1=i0-i1+i3
N2=i1-i2-i5
N3=i2-i3-i4
N4=i4+i5-i6
```

Para escribir as ecuacións dadas pola lei de ciclos de Kirchoff, calculamos a dimensión de Z(D) a partir da matriz de incidencia.

```
V=vertices(RE)
E=[[a,b] for [a,b,c] in RE]
U=incidencia(V,E)
print 'dim Z(D)=',len(E)-rank(U)
```

```
dim Z(D)= 3
```

Eliximos como base de Z(D) os tres ciclos interiores do esquema anterior, obtendo as ecuacións C0, C1, C2 dadas pola lei de ciclos de Kirchoff:

```
dp=vector([2*i0,40*i1,20*i2,6*i3,5*i4,0*i5,-125])
c0=vector([1,0,0,-1,1,0,1])
c1=vector([0,1,1,1,0,0,0])
c2=vector([0,0,1,0,1,-1,0])
C0=c0*dp
C1=c1*dp
C2=c2*dp
print 'C0=',C0
print 'C1=',C1
```

```
print 'C2=',C2
```

```
C0= 2*i0 - 6*i3 + 5*i4 - 125
C1= 40*i1 + 20*i2 + 6*i3
C2= 20*i2 + 5*i4
```

Resolvemos con **solve**() o sistema de todas as ecuacións de Kirchoff e obtemos as intensidades nos arcos.

```
S=solve([N0,N1,N2,N3,N4,C0,C1,C2],W,solution_dict=True)
VW=[S[0][a] for a in W]
show(VW)
```

$$\left[\frac{25}{2}, \frac{5}{2}, -2, -10, 8, \frac{9}{2}, \frac{25}{2}\right]$$

Se expresamos as ecuacións en forma vectorial A **x= b** podemos resolver o sistema con **rouche**(A,**b**):

```
f(W)=[N0,N1,N2,N3,N4,C0,C1,C2]
A=JAC(f)
b=vector([0,0,0,0,0,125,0,0])
rouche(A,b)
```

```
O sistema ten a solucion unica
(25/2, 5/2, -2, -10, 8, 9/2, 25/2)
```

Coñecidas as intensidades podemos calcular o vector de tensións $t \in \mathbb{R}^E$.

```
t=dp(i0=VW[0],i1=VW[1],i2=VW[2],i3=VW[3],i4=VW[4],i5=VW[5],i6=VW[6
show(t)
```

$$(25,\ 100,\ -40,\ -60,\ 40,\ 0,\ -125)$$

Para calcular o potencial debemos resolver o problema lineal inverso $\delta : \mathbb{R}^V \to \mathbb{R}^E$ para o dato t. Como é natural, o potencial está determinado salvo unha constante aditiva:

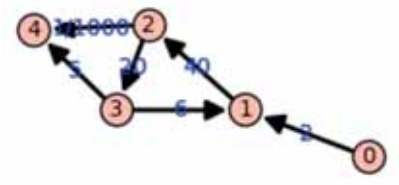
```
A=transpose(U)
P=rouche(A,t)
show(P)
```

```
O sistema ten unha variedade de dimension 1
de solucions cuxa expresion parametrica e
```

$$(p_0 + 125,\ p_0 + 100,\ p_0,\ p_0 + 40,\ p_0)$$

Finalmente, se suprimimos de RE a arista con xerador e substituímos as resistencias de 0 ohmios por resistencias de 1/1000 ohmios, obtemos a rede RESG.

```
ASG=[k for k in range(len(dp)) if dp[k] not in RR or dp[k]==0]
RESG=[[a,b,c[0]] for [a,b,c] in selenlist(RE,ASG)]
for a in RESG:
    if a[2]==0:
        a[2]=1/1000
D=DiGraph(RESG)
D.graphplot(edge_labels=True).show(figsize=[2.5,2.5])
```



Achando a conductancia C do grafo biriireccional de RESG, a función **kleinrandic**(C) devólvenos a matriz de resistencias equivalentes entre os nodos de RE. Podemos corroborar que $i_0 = \frac{125}{EQ[0,4]} = \frac{25}{2}$.

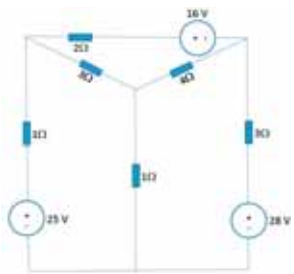
```
B=bidir(RESG)
C=conductance(B)
EQ=Roundm(kleinrandic(C),3)
show(EQ)
print 'i0=',125/EQ[0,4]
```

$$\begin{pmatrix} 0.0 & 2.0 & 10.0 & 7.28 & 10.0 \\ 2.0 & 0.0 & 8.0 & 5.28 & 8.0 \\ 10.0 & 8.0 & 0.0 & 3.681 & 0.001 \\ 7.28 & 5.28 & 3.681 & 0.0 & 3.68 \\ 10.0 & 8.0 & 0.001 & 3.68 & 0.0 \end{pmatrix}$$

```
i0= 12.5
```

Exercicio 6:

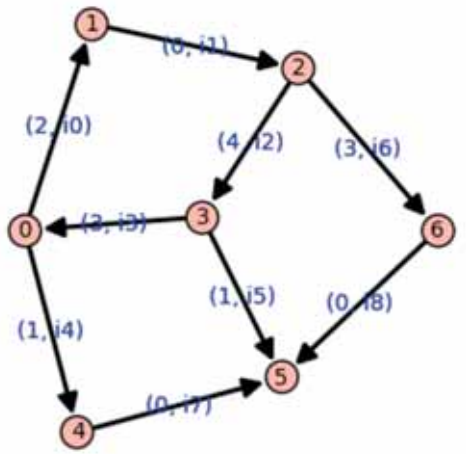
Estudar o circuíto eléctrico:



Consideramos o circuíto como un digrafo unidireccional con nodos [0,1,2,3,4,5,6]

situados, respectivamente, no vértice superior esquerdo, no medio da arista superior separando a resistencia de 2 Ω do xerador de 16 V, no vértice superior dereito, no vértice interior, no medio da arista esquerda separando a resistencia de 1Ω do xerador de 25 V, na intersección da arista central coa inferior e no medio da arista dereita separando a resistencia de 3Ω do xerador de 28 V. Nos arcos do digrafo poñemos etiquetas coa resistencia e a intensidade:

```
W=listasim('i',9)
RE=[[0,1,(2,i0)],[1,2,(0,i1)],[2,3,(4,i2)],[3,0,(3,i3)],
[0,4,(1,i4)],[3,5,(1,i5)],[2,6,(3,i6)],[4,5,(0,i7)],
[6,5,(0,i8)]]
D=DiGraph(RE)
D.graphplot(edge_labels=True).show(figsize=[3,3])
```



Sobre este esquema escribimos as ecuacións dos nodos.

```
N0=i3-i0-i4
N1=i0-i1
N2=i1-i2-i6
N3=i2-i5-i3
N4=i4-i7
N5=i5+i7+i8
N6=i6-i8
```

Achamos a dimensión de Z(D):

```
V=vertices(RE)
E=[[a,b] for [a,b,c] in RE]
U=incidencia(V,E)
print 'dim Z(D)=',len(E)-rank(U)
```

```
dim Z(D)= 3
```

Escribimos as ecuacións dos ciclos:

```
dp=vector([2*i0,-16,4*i2,3*i3,i4,i5,3*i6,-25,-28])
c0=vector([1,1,1,1,0,0,0,0,0])
c1=vector([0,0,-1,0,0,-1,1,0,1])
c2=vector([0,0,0,1,1,-1,0,1,0])
C0=c0*dp
C1=c1*dp
C2=c2*dp
print 'C0=',C0
print 'C1=',C1
print 'C2=',C2
```

```
C0= 2*i0 + 4*i2 + 3*i3 - 16
C1= -4*i2 - i5 + 3*i6 - 28
C2= 3*i3 + i4 - i5 - 25
```

E calculamos todas as intensidades con **solve**().

```
S=solve([N0,N1,N2,N3,N4,N5,N6,C0,C1,C2],W,solution_dict=True)
VW=[S[0][a] for a in W]
show(VW)
```

$$\left[\frac{568}{175}, \frac{568}{175}, -\frac{283}{175}, \frac{932}{175}, \frac{52}{25}, -\frac{243}{35}, \frac{851}{175}, \frac{52}{25}, \frac{851}{175}\right]$$

Con elas calculamos o vector de tensións $t \in \mathbb{R}^E$.

```
t=dp(i0=VW[0],i1=VW[1],i2=VW[2],i3=VW[3],i4=VW[4],i5=VW[5],
    i6=VW[6],i7=VW[7],i8=VW[8]);t
```

```
(1136/175, -16, -1132/175, 2796/175, 52/25, -243/35, 2553/175, -25, -28)
```

Achamos o potencial resolvendo o problema lineal inverso $\delta : \mathbb{R}^V \to \mathbb{R}^E$ para o dato t.

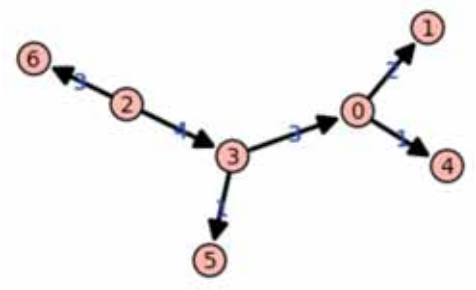
```
A=transpose(U)
P=rouche(A,t)
show(P)
```

```
O sistema ten unha variedade de dimension 1
de solucions cuxa expresion parametrica e
```

$$\left(p_0 + \frac{127}{25}, p_0 - \frac{247}{175}, p_0 + \frac{2553}{175}, p_0 + \frac{737}{35}, p_0 + 3, p_0 + 28, p_0\right)$$

Finalmente, calculamos a RESG.

```
ASG=[k for k in range(len(dp)) if dp[k] not in RR or dp[k]==0]
RESG=[[a,b,c[0]] for [a,b,c] in selenlist(RE,ASG)]
for a in RESG:
    if a[2]==0:
        a[2]=1/1000
D=DiGraph(RESG)
D.graphplot(edge_labels=True).show(figsize=[3,3])
```



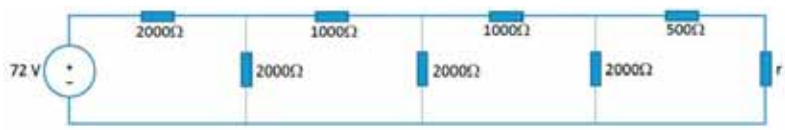
e a matriz de resistencias equivalentes entre os nodos de RE.

```
B=bidir(RESG)
C=conductance(B)
EQ=Roundm(kleinrandic(C),3)
show(EQ)
```

$$\begin{pmatrix} 0.0 & 2.0 & 7.0 & 3.0 & 1.0 & 4.0 & 10.0 \\ 2.0 & 0.0 & 9.0 & 5.0 & 3.0 & 6.0 & 12.0 \\ 7.0 & 9.0 & 0.0 & 4.0 & 8.0 & 5.0 & 3.0 \\ 3.0 & 5.0 & 4.0 & 0.0 & 4.0 & 1.0 & 7.0 \\ 1.0 & 3.0 & 8.0 & 4.0 & 0.0 & 5.0 & 11.0 \\ 4.0 & 6.0 & 5.0 & 1.0 & 5.0 & 0.0 & 8.0 \\ 10.0 & 12.0 & 3.0 & 7.0 & 11.0 & 8.0 & 0.0 \end{pmatrix}$$

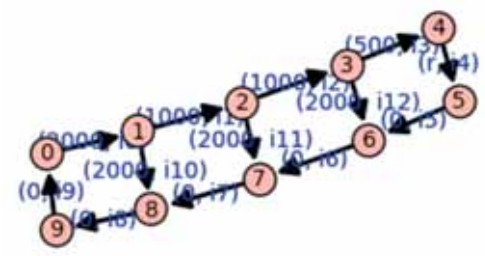
Exercicio 7:

Estudar os equivalentes de Thévenin e Norton da resistencia r no seguinte circuíto eléctrico:



Consideramos o circuíto como un digrafo unidireccional con nodos [0,1,2,3,4,5,6,7,8,9]. O 0 situado no vértice superior esquerdo e os restantes, nos correspondentes vértices do circuíto percorrido en sentido horario. Nos arcos do digrafo poñemos etiquetas coa resistencia e a intensidade:

```
var('r')
W=listasim('i',13)
RE=[[0,1,(2000,i0)],[1,2,(1000,i1)],[2,3,(1000,i2)],
[3,4,(500,i3)],[4,5,(r,i4)],[5,6,(0,i5)],[6,7,(0,i6)],
[7,8,(0,i7)],[8,9,(0,i8)],[9,0,(0,i9)],[1,8,(2000,i10)],
[2,7,(2000,i11)],[3,6,(2000,i12)]]
D=DiGraph(RE)
D.graphplot(edge_labels=True).show(figsize=[3,3])
```



Escribimos as ecuacións de Kirchoff dos nodos.

```
N0=i9-i0
N1=i0-i1-i10
N2=i1-i2-i11
N3=i2-i3-i12
N4=i3-i4
N5=i4-i5
N6=i5+i12-i6
N7=i6+i11-i7
N8=i7+i10-i8
N9=i8-i9
```

Calculamos a dimensión de Z(D) e escribimos as ecuacións dos ciclos.

```
V=vertices(RE)
E=[[a,b] for [a,b,c] in RE]
U=incidencia(V,E)
print 'dim Z(D)=',len(E)-rank(U)
dp=vector([2000*i0,1000*i1,1000*i2,500*i3,r*i4,0,0,0,0,-
72,2000*i10,2000*i11,2000*i12])
c0=vector([1,0,0,0,0,0,0,0,1,1,1,0,0])
c1=vector([0,1,0,0,0,0,0,0,1,0,0,-1,1,0])
c2=vector([0,0,1,0,0,0,0,1,0,0,0,0,-1,1])
c3=vector([0,0,0,1,1,1,0,0,0,0,0,0,-1])
C0=c0*dp
C1=c1*dp
C2=c2*dp
```

```
C3=c3*dp
print 'C0=',C0
print 'C1=',C1
print 'C2=',C2
print 'C3=',C3
```

```
dim Z(D)= 4
C0= 2000*i0 + 2000*i10 - 72
C1= 1000*i1 - 2000*i10 + 2000*i11
C2= -2000*i11 + 2000*i12 + 1000*i2
C3= i4*r - 2000*i12 + 500*i3
```

Resolvemos con **solve** e interesámosnos só por i4.

```
S=solve([N0,N1,N2,N3,N4,N5,N6,N7,N8,N9,C0,C1,C2,C3],W,
         solution_dict=True)
VW=[S[0][a] for a in W]
i4(r)=VW[4]
print 'i4(r)=',i4(r)
```

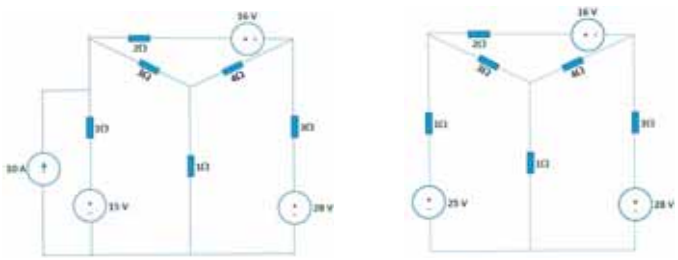
```
i4(r)= 9/(r + 1500)
```

De acordo con Thévenin, podemos substituír o circuíto por un triángulo, unha de cuxas aristas é a resistencia de r ohmios, outra ten un xerador de tensión de 9 voltios e a terceira unha resistencia de 1500 ohmios.

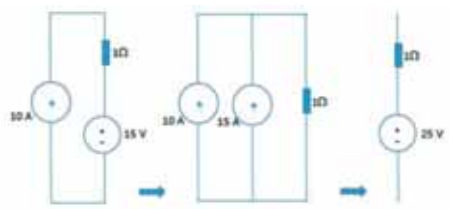
De acordo con Norton, podemos substituír o circuito inicial por un formado por tres aristas en paralelo, a propia resistencia de r ohmios, unha resistencia de 1500 ohmios e, a terceira, cun xerador de corrente de 6 miliamperios.

Exercicio 8:

Demostrar que os seguintes circuítos son equivalentes:



De acordo con Thévenin-Norton podemos establecer a seguinte cadea de equivalencias:



ESTÁTICA

Exercicio 9:

Unha barra perfectamente ríxida de lonxitude $2L$ e peso P está en repouso apoiada en n puntos de abscisas distintas $x_0, \cdots, x_{n-1}$. Determinar as forzas verticais $f_0, \cdots, f_{n-1}$ nos devanditos puntos.

Solución:

Aplicando as leis de Newton obtemos as ecuacións E0 e E1 que podemos resolver nas variables $F = [f_0, \ldots, f_{n-1}]$.

Se n=2 o sistema é compatible e determinado:

```
n=2
var('P L')
X=listasim('x',n)
F=listasim('f',n)
E0=sum(F)-P
E1=vector(X)*vector(F)-P*L
S=solve([E0,E1],F,solution_dict=True)
[S[0][a] for a in F]
```

```
[(L - x1)*P/(x0 - x1), -(L - x0)*P/(x0 - x1)]
```

Se n>2 o sistema ten unha variedade de solucións de dimensión n-2:

```
n=5
var('P L')
X=listasim('x',n)
F=listasim('f',n)
E0=sum(F)-P
E1=vector(X)*vector(F)-P*L
S=solve([E0,E1],F,solution_dict=True)
FR=[S[0][a] for a in F]
FR
```

```
[((L - x1)*P + (r1 + r2 + r3)*x1 - r3*x2 - r2*x3 - r1*x4)/(x0 - x1),
-((L - x0)*P + (r1 + r2 + r3)*x0 - r3*x2 - r2*x3 - r1*x4)/(x0 - x1), r3,
r2, r1]
```

Nesa variedade de solucións existe unha soa de norma mínima que podemos calcular coa inversa de Moore-Penrose. En particular, para $L = 10$ m, $P = 100$ kg e $X = [2, 4, 10, 12, 18]$ obtemos

```
P=100
L=10
X=[2,4,10,12,18]
F=listasim('f',n)
E0=sum(F)-P
E1=vector(X)*vector(F)-P*L
f(F)=[E0,E1]
A=JAC(f)
b=vector([P,P*L])
SNM=pinv(A)*matrix(b).transpose()
show(SNM)
```

$$\begin{pmatrix} 16.5048547089 \\ 17.4757279456 \\ 20.3883487731 \\ 21.3592208922 \\ 24.2718428373 \end{pmatrix}$$

Se no apoio x_i a forza f_i procedese dun resorte de constante elástica k_i que sufre unha elongación vertical y_i tal que $f_i = k_i y_i$, a condición de perfecta rixidez da barra impón a existencia dunha recta y = mx+b que pase polos puntos $(x_0, y_0), \cdots, (x_{n-1}, y_{n-1})$ e, xa que logo, as n incógnitas $b, m, r_1, \ldots, r_{n-2}$ han de cumprir as n ecuacións K[i](b+mX[i])-FR[i] con $i = 0, \ldots, n-1$ quedando o problema determinado como parece exixir a nosa intuición física:

```
K=listasim('k',n)
var('b m')
R=listasim('r',50)
SF=solve([K[i]*(b+m*X[i])-FR[i] for i in range(n)],
[b,m,r1,r2,r3],solution_dict=True)
SFR=[SF[0][a] for a in [b,m,r1,r2,r3]]
```

No caso particular anterior, L=10 m, P=100 kg, X=[2,4,10,12,18], si todas as constantes elásticas son iguais, obtemos a antedita solución de norma mínima:

```
var('k')
LL=[SFR[i]
(L=10,P=100,x0=2,x1=4,x2=10,x3=12,x4=18,k0=k,k1=k,k2=k,
    k3=k,k4=k) for i in range(5)]
SOL=Round([FR[i]
(L=10,P=100,r1=LL[2],r2=LL[3],r3=LL[4],x0=2,x1=4,x2=10,
    x3=12,x4=18) for i in range(5)],6)
```

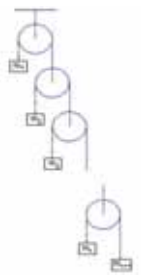
```
show(SOL)
```

$$[16.504854, 17.475728, 20.38835, 21.359223, 24.271845]$$

DINÁMICA

Exercicio 10:

Estudar o movemento de $n+1$ masas conectadas nun sistema de n poleas como o considerado na figura:



Supoñemos que as poleas son de masas despreciables e actúan sen rozamentos.

Solución:

No caso dunha soa polea as leis de Newton proporciónannos as seguintes relacións entre as aceleracións a_1, a_2 e as tensións T_1, T_2:

$$\begin{array}{rrrrrrr} m_0a_0 & & & - & T_0 & = & -m_0g \\ & m_1a_1 & & & - \; T_1 & = & -m_1g \\ & & & T_0 & - \; T_1 & = & 0 \\ a_0 & + & a_1 & & & = & 0 \end{array}$$

que podemos resolver con **solve**():

```
var('g')
M=listasim('m',2)
A=listasim('a',2)
T=listasim('T',2)
E0=m0*a0-T0+m0*g
E1=m1*a1-T1+m1*g
E2=T0-T1
E3=a0+a1
S=solve([E0,E1,E2,E3],A+T,solution_dict=True)
AT=[S[0][a] for a in A+T]
print 'aceleracións'
show(AT[0:2])
print 'tensións'
show(AT[2:])
```

```
aceleracións
```

$$\left[-\frac{gm_0 - gm_1}{m_0 + m_1}, \frac{gm_0 - gm_1}{m_0 + m_1}\right]$$

tensións

$$\left[\frac{2\,gm_0m_1}{m_0 + m_1}, \frac{2\,gm_0m_1}{m_0 + m_1}\right]$$

No caso de 2 poleas:

$$\begin{array}{lllllllll}
m_0a_0 & & & & - & T_0 & & & = & -m_0g \\
 & m_1a_1 & & & & - & T_1 & & = & -m_1g \\
 & & m_2a_2 & & & & - & T_2 & = & -m_2g \\
 & & & & T_0 & - & 2T_1 & & = & 0 \\
 & & & & & & T_1 & - T_2 & = & 0 \\
2a_0 & + & a_1 & + & a_2 & & & & = & 0
\end{array}$$

e podemos escribilas:

```
var('g')
M=listasim('m',3)
A=listasim('a',3)
T=listasim('T',3)
E0=m0*a0-T0+m0*g
E1=m1*a1-T1+m1*g
E2=m2*a2-T2+m2*g
E3=T0-2*T1
E4=T1-T2
E5=2*a0+a1+a2
S=solve([E0,E1,E2,E3,E4,E5],A+T,solution_dict=True)
AT=[S[0][a] for a in A+T]
print 'aceleracións'
show(AT[0:3])
print 'tensións'
show(AT[3:])
```

aceleracións

$$\left[-\frac{gm_0(m_1 + m_2) - 4\,gm_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}, -\frac{gm_0(m_1 - 3\,m_2) + 4\,gm_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}, \frac{gm_0(3\,m_1 - m_2) - 4\,gm_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}\right]$$

tensións

$$\left[\frac{8\,gm_0m_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}, \frac{4\,gm_0m_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}, \frac{4\,gm_0m_1m_2}{m_0(m_1 + m_2) + 4\,m_1m_2}\right]$$

No caso de 3 poleas:

$$
\begin{array}{cccccccccccc}
m_0a_0 & & & & & & -T_0 & & & & = & -m_0g \\
& m_1a_1 & & & & & & -T_1 & & & = & -m_1g \\
& & m_2a_2 & & & & & & -T_2 & & = & -m_2g \\
& & & m_3a_3 & & & & & & -T_3 & = & -m_3g \\
& & & & & T_0 & -2T_1 & & & & = & 0 \\
& & & & & & T_1 & -2T_2 & & & = & 0 \\
& & & & & & & T_2 & -T_3 & & = & 0 \\
4a_0 & + 2a_1 & + a_2 & & a_3 & & & & & & = & 0
\end{array}
$$

que escribimos:

```
var('g')
M=listasim('m',4)
A=listasim('a',4)
T=listasim('T',4)
E0=m0*a0-T0+m0*g
E1=m1*a1-T1+m1*g
E2=m2*a2-T2+m2*g
E3=m3*a3-T3+m3*g
E4=T0-2*T1
E5=T1-2*T2
E6=T2-T3
E7=4*a0+2*a1+a2+a3
S=solve([E0,E1,E2,E3,E4,E5,E6,E7],A+T,solution_dict=True)
AT=[S[0][a] for a in A+T]
print 'aceleracións'
show(AT[0:4])
print
print 'tensións'
show(AT[4:])
```

aceleracións

$$\left[\frac{16g m_1 m_2 m_3 - (m_1(m_2+m_3)+4m_2m_3)gm_0}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{16g m_0 m_2 m_3+(m_1(m_2+m_3)-12m_2m_3)gm_0}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{16g m_0 m_1 m_3+(m_1(m_3-7m_2)+4m_2m_3)gm_0}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{16g m_0 m_1 m_2-(m_1(7m_2-m_3)-4m_2m_3)gm_0}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}\right]$$

tensións

$$\left[\frac{32g m_0 m_1 m_2 m_3}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{16g m_0 m_1 m_2 m_3}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{8g m_0 m_1 m_2 m_3}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}, \frac{8g m_0 m_1 m_2 m_3}{16m_1m_2m_3+(m_1(m_2+m_3)+4m_2m_3)m_0}\right]$$

Para o caso de n poleas é máis fácil inferir a ecuación matricial $Ax = b$ onde:

$$A = \begin{pmatrix} M & -I \\ L & R \end{pmatrix}, \quad \mathbf{b} = -g*\text{vector}([m_0, \cdots, m_n, 0, \cdots, 0]) \quad \text{y} \quad \mathbf{x} = \text{vector}([a_0, \cdots, a_n, T$$

sendo:

$$M = \begin{pmatrix} m_0 & & \\ & \ddots & \\ & & m_n \end{pmatrix} \qquad -I = \begin{pmatrix} -1 & & \\ & \ddots & \\ & & -1 \end{pmatrix}$$

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ & & \ddots & & \\ 0 & 0 & \cdots & 0 & 0 \\ 2^{n-1} & 2^{n-2} & \cdots & 1 & 1 \end{pmatrix} \qquad R = \begin{pmatrix} 1 & -2 & & & \\ & \ddots & \ddots & & \\ & & 1 & -2 & \\ 0 & 0 & & 1 & -1 \\ 0 & 0 & & 0 & 0 \end{pmatrix}$$

Para o caso particular de sete masas [1.5, 2, 3.5, 4, 2, 3.2, 4] e $g = 9.81$, teremos:

```
g=9.81
M=diagonal_matrix([1.5, 2, 3.5, 4, 2, 3.2, 4])
I=-identity_matrix(7)
L=matrix(7,7)
for j in range(6):
    L[6,j]=2^(5-j)
L[6,6]=1
R=diagonalsmatrix([[1,1,1,1,1,1,0],[-2,-2,-2,-2,-2,-1]],
[0,1],7,7)
A=block_matrix([[M,I],[L,R]])
b=g*vector([1.5, 2, 3.5, 4, 2, 3.2, 4,0,0,0,0,0,0,0])
AT=rouche(A,b)
print 'aceleracións'
show(Round(AT[0:7],3))
print 'tensións'
show(Round(AT[7:],3))
```

```
O sistema ten a solucion unica
aceleracións
```

$$[-6.221, 3.798, 8.092, 9.059, 9.059, 9.575, 9.622]$$

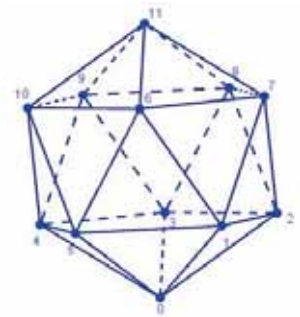
```
tensións
```

$$[-24.046, -12.023, -6.012, -3.006, -1.503, -0.751, -0.751]$$

PROBABILIDADE

Exercicio 11:

Unha formiga avanza polas aristas dun icosaedro de modo que, ao chegar a un vértice, volve sobre os seus pasos ou continúa por calquera das outras aristas que inciden nel, coa mesma probabilidade.



Se designamos P_k á probabilidade de que partindo do vertice k chegue ao polo norte (11) antes que ao polo sur (0), é claro que $P_0 = 0$ e $P_{11} = 1$. Achar P_k para $k = 1, \ldots, 10$.

Cada vértice k ten un conxunto de 5 vértices adxacentes A(k) e é claro que

$$P_k = \frac{1}{5} \sum_{i \in A(k)} P_i \quad \forall k = 1, \ldots, 10$$

Debemos escribir e resolver este sistema de ecuacións lineais:

```
P=listasim('P',11)
P=P[1:]
E1=5*P1-P2-P5-P6-P10
E2=5*P2-P1-P3-P6-P7
E3=5*P3-P4-P8-P7-P2
E4=5*P4-P3-P8-P9-P5
E5=5*P5-P1-P10-P9-P4
E6=5*P6-P1-P2-P7-P10-1
E7=5*P7-P6-P2-P3-P8-1
E8=5*P8-P7-P9-P3-P4-1
E9=5*P9-P8-P10-P4-P5-1
E10=5*P10-P9-P5-P6-P1-1
```

Podemos usar **solve**().

```
S=solve([E1,E2,E3,E4,E5,E6,E7,E8,E9,E10],P,solution_dict=True)
R=[S[0][a] for a in P]
show(R)
```

$$\left[\frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}\right]$$

ou escribilo en forma matricial $A\mathbf{x} = \mathbf{b}$ e usar **rouche**(A,b) para resolvelo

```
f(P)=[E1,E2,E3,E4,E5,E6,E7,E8,E9,E10]
A=JAC(f)
print 'A='
show(A)
b=vector([0,0,0,0,0,1,1,1,1,1])
print 'b='
show(b)
show(rouche(A,b))
```

A=

$$\begin{pmatrix} 5 & -1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & -1 \\ -1 & 5 & -1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 5 & -1 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 5 & -1 & 0 & 0 & -1 & -1 & 0 \\ -1 & 0 & 0 & -1 & 5 & 0 & 0 & 0 & -1 & -1 \\ -1 & -1 & 0 & 0 & 0 & 5 & -1 & 0 & 0 & -1 \\ 0 & -1 & -1 & 0 & 0 & -1 & 5 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 5 & -1 \\ -1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & -1 & 5 \end{pmatrix}$$

b=

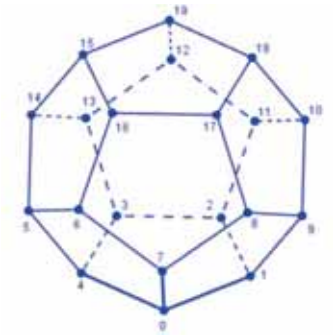
$$(0, 0, 0, 0, 0, 1, 1, 1, 1, 1)$$

O sistema ten a solucion unica

$$\left(\frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{2}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}, \frac{3}{5}\right)$$

Exercicio 12:

Unha formiga avanza polas aristas dun dodecaedro de modo que, ao chegar a un vértice, volve sobre os seus pasos ou continua por calquera das outras aristas que inciden nel, coa mesma probabilidade



Se P_k é a probabilidade de que partindo de k chegue ao 19 (arriba) antes que ao 0

(abaixo), é claro que $P_0 = 0$ e $P_{19} = 1$.

Achar P_k para $k = 1, \ldots, 18$.

Solución:

Cada vértice k ten un conxunto de 3 vértices adxacentes A(k) e é claro que

$$P_k = \frac{1}{3} \sum_{i \in A(k)} P_i \quad \forall k = 1, \ldots, 18$$

Debemos escribir e resolver o seguinte sistema de ecuacións lineais:

```
P=listasim('P',19)
P=P[1:]
E1=3*P1-P2-P9
E2=3*P2-P1-P3-P11
E3=3*P3-P2-P4-P13
E4=3*P4-P3-P5
E5=3*P5-P4-P6-P14
E6=3*P6-P5-P7-P16
E7=3*P7-P6-P8
E8=3*P8-P7-P9-P17
E9=3*P9-P1-P8-P10
E10=3*P10-P9-P11-P18
E11=3*P11-P10-P12-P2
E12=3*P12-P11-P13-1
E13=3*P13-P12-P14-P3
E14=3*P14-P15-P13-P5
E15=3*P15-P14-P16-1
E16=3*P16-P15-P17-P6
E17=3*P17-P16-P18-P8
E18=3*P18-P10-P17-1
```

Podemos usar **solve**()

```
S=solve([E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,
 E12,E13,E14,E15,E16,E17,E18],P,solution_dict=True)
R=[S[0][a] for a in P]
show(R)
```

$$\left[\frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}\right]$$

ou escribilo en forma matricial $A\mathbf{x} = \mathbf{b}$ e usar **rouche**(A,b) para resolvelo

```
f(P)=[E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15,E16,E17,E
A=JAC(f)
print 'A='
show(A)
b=vector([0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1])
print 'b='
show(b)
show(rouche(A,b))
```

A=

$$\begin{pmatrix}
3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 3
\end{pmatrix}$$

b=

$$(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1)$$

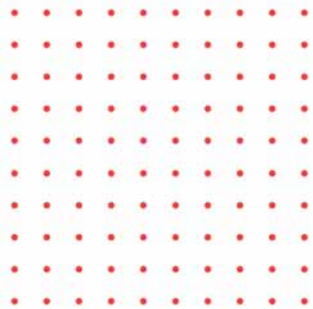
O sistema ten a solucion unica

$$\left(\frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{2}{7}, \frac{3}{7}, \frac{3}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}, \frac{4}{7}, \frac{4}{7}, \frac{5}{7}\right)$$

TEMPERATURAS ESTACIONARIAS EN REIXAS

Exercicio 13

Sexa unha reixa cadrada como a seguinte:



Se os puntos dos bordos están a temperaturas B=[N, S, E, W], a que temperatura están os puntos interiores?

Solución:

Os problemas de temperatura estacionaria están gobernados pola ecuación de Laplace. O teorema do valor medio para funcións armónicas implica que, discretizando o problema, a temperatura nun punto debe ser o promedio das temperaturas dos puntos adxacentes.

Por exemplo, no caso dunha reixa cadrada $n \times n$ co bordo a temperaturas B, a función **rejillac**(n) proporciónanos un esquema coas variables adecuadas

```
n=10
T=matrixsim('t',n,n)
R=rejillac(n)
show(sum(R),axes=False,figsize=[3,3])
```

t00 t01 t02 t03 t04 t05 t06 t07 t08 t09

t10 t11 t12 t13 t14 t15 t16 t17 t18 t19

t20 t21 t22 t23 t24 t25 t26 t27 t28 t29

t30 t31 t32 t33 t34 t35 t36 t37 t38 t39

t40 t41 t42 t43 t44 t45 t46 t47 t48 t49

t50 t51 t52 t53 t54 t55 t56 t57 t58 t59

t60 t61 t62 t63 t64 t65 t66 t67 t68 t69

t70 t71 t72 t73 t74 t75 t76 t77 t78 t79

t80 t81 t82 t83 t84 t85 t86 t87 t88 t89

t90 t91 t92 t93 t94 t95 t96 t97 t98 t99

para suscitar con comodidade as ecuacións da temperatura:

```
B=[100,500,300,1000]
Eq=[]
F=[1,2,..,n-2]
```

```
for i in  F:
    Eq.append(T[i,0]-B[3])
    Eq.append(T[i,n-1]-B[2])
    Eq.append(T[0,i]-B[0])
    Eq.append(T[n-1,i]-B[1])
for i in F:
    for j in F:
        Eq.append(4*T[i,j]-T[i-1,j]-T[i+1,j]-T[i,j-1]-
T[i,j+1])
```

Se V son as variables que interveñen en Eq, podemos resolver o sistema lineal con **solve**(Eq,V,solution_dict=True).

```
V=VAR(Eq)
L=solve(Eq,V,solution_dict=True)
R=[L[0][a] for a in V]
R3=Round(R,3)
```

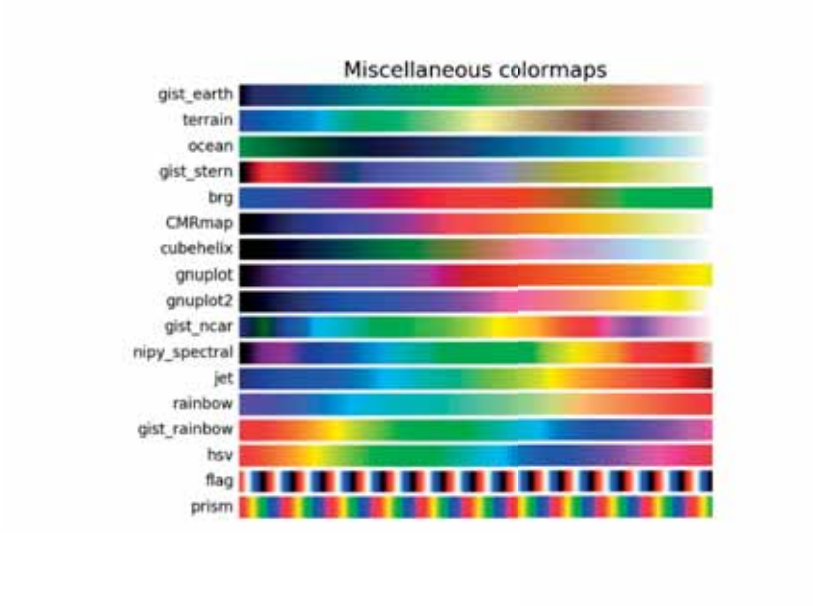
```
R3C=[550,100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
100.0, 200,1000.0,
545.887, 364.52, 278.542, 232.056, 205.445, 191.913, 191.735,
215.08,
300.0, 1000.0, 719.029, 533.651, 417.592, 344.237, 297.812,
270.473,
259.948, 268.585, 300.0, 1000.0, 796.578, 633.461, 513.94,
429.487,
371.092, 332.219, 308.996, 299.314, 300.0, 1000.0, 833.82,
689.677,
575.221, 488.677, 424.852, 378.315, 344.505, 319.675, 300.0,
1000.0,
849.026, 716.205, 608.589, 525.148, 461.323, 411.683, 371.033,
334.881,
300.0, 1000.0, 846.08, 717.53, 617.781, 542.004, 483.61,
436.06,
393.065, 348.816, 300.0, 1000.0, 817.763, 690.052, 603.001,
541.478,
495.053, 455.881, 416.349, 367.32, 300.0, 1000.0, 734.92,
621.916,
562.693, 525.853, 499.243, 476.064, 449.131, 404.113, 300.0,
750,500.0,
500.0, 500.0, 500.0, 500.0, 500.0, 500.0, 500.0,400]
show(Roundm(matrix(n,R3C),2))
```

550.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	200.0
1000.0	545.89	364.52	278.54	232.06	205.44	191.91	191.74	215.08	300.0
1000.0	719.03	533.65	417.59	344.24	297.81	270.47	259.95	268.58	300.0
1000.0	796.58	633.46	513.94	429.49	371.09	332.22	309.0	299.31	300.0
1000.0	833.82	689.68	575.22	488.68	424.85	378.31	344.5	319.68	300.0
1000.0	849.03	716.21	608.59	525.15	461.32	411.68	371.03	334.88	300.0
1000.0	846.08	717.53	617.78	542.0	483.61	436.06	393.06	348.82	300.0
1000.0	817.76	690.05	603.0	541.48	495.05	455.88	416.35	367.32	300.0
1000.0	734.92	621.92	562.69	525.85	499.24	476.06	449.13	404.11	300.0
750.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	500.0	400.0

Como as temperaturas dos vértices non son relevantes podémolas supoñer iguais ao promedio das temperaturas dos lados que os xeran e completar adecuadamente a lista R3 para presentar a matriz de temperaturas en todos os puntos da reixa.

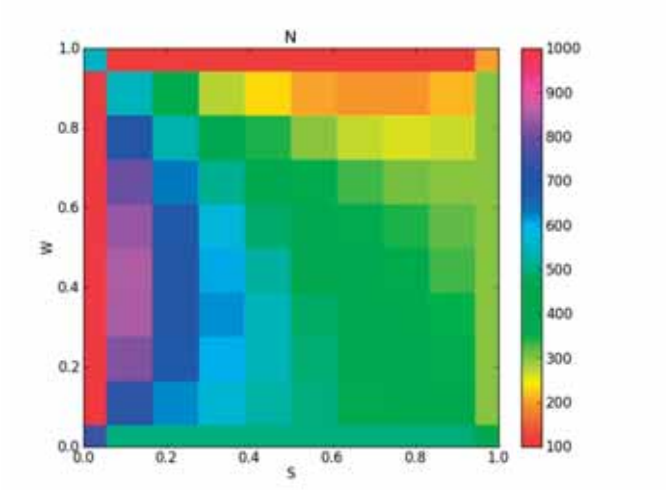
Programamos a solución deste problema na función **placaq**(l,n-1,mapa,N,S,E,W) para un cadrado de lado l con n puntos equidistantes. En lugar dunha solución numérica, devólvenos unha solución colorística construída sobre calquera dos mapa de cores que elixamos coa variable 'mapa' entre os importados coa seguinte orde

```
cargar(['http://matplotlib.org/mpl_examples/color
/colormaps_reference_05.hires.png'])
```



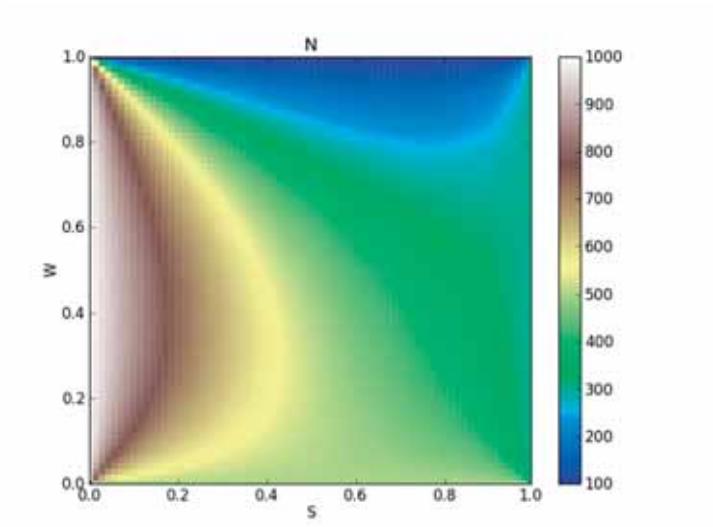
Utilizando, por exemplo, o mapa de cor de Munssell 'hsv' introducido no Worksheet 0, obtemos o resultado anterior na forma

```
T=placaq(1,9,'hsv',100,500,300,1000)
```



Ademais, podemos aumentar o número de puntos de cada lado para obter unha representación colorística mais suave

```
T=placaq(1,70,'terrain',100,500,300,1000)
```



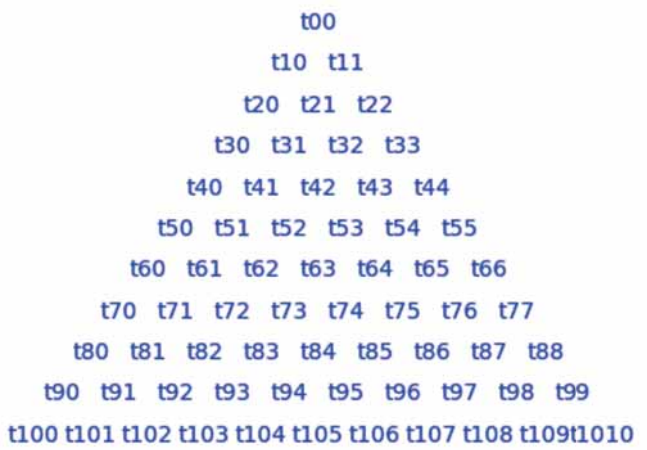
Exercicio 14:

Sexa unha reixa triangular equilátera como a seguinte

Se os puntos dos bordos están a temperaturas [S, E , W], a que temperatura están os puntos interiores?

No caso do triángulo de vértices X=(0,0), Y=(1,0), Z=($\frac{1}{2}$, $\frac{1\sqrt{3}}{2}$), con n puntos na base e [S,E,W]=[500,300,1000], a función **rejillat**(X,Y,Z,n-1) proporciónanos un esquema de variables

```
X=vector([0,0])
Y=vector([1,0])
Z=vector([1/2,sqrt(3)/2])
R=rejillat(X,Y,Z,10)
show(sum(R),axes=False,figsize=[4,3])
```



que nos permiten suscitar con comodidade as ecuacións da temperatura:

```
n=10
[E,S,W]=[300,500,1000]
T=matrixsim('t',n+1,n+1)
Eq=[]
L=[1,2,..,n-1]
for i in L:
    Eq.append(T[i,0]-W)
```

```
    Eq.append(T[n,i]-S)
    Eq.append(T[i,i]-E)
for i in L:
    for j in [1,2,..,i-1]:
        Eq.append(6*T[i,j]-T[i-1,j-1]-T[i-1,j]-T[i,j-1]-
T[i,j+1]-T[i+1,j]-T[i+1,j+1])
```

Se V son as variables que interveñen en Eq, podemos resolver o sistema lineal con **solve**(Eq,V,solution_dict=True).

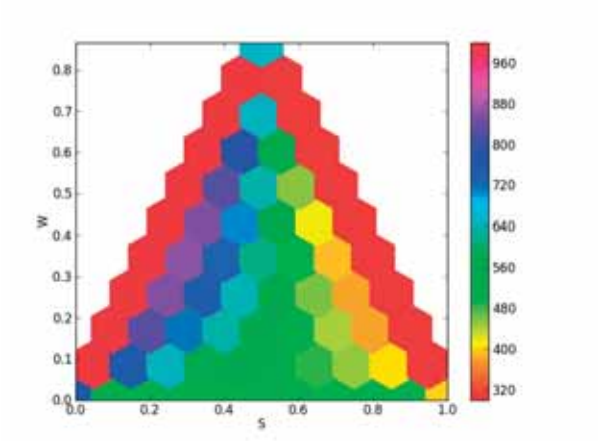
```
V=VAR(Eq)
SOL=solve(Eq,V,solution_dict=True)
R=[SOL[0][a] for a in V]
R3=Round(R,3)
```

Como as temperaturas dos vértices non son relevantes, podemos supoñelas iguais ao promedio das temperaturas dos lados que os xeran e completar adecuadamente a lista R3 para presentar as temperaturas en cada punto da reixa.

```
R3C=[650,1000.0, 300.0, 1000.0, 648.632, 300.0, 1000.0,
772.603, 519.187, 300.0,
1000.0, 828.738, 639.06, 454.826, 300.0, 1000.0, 856.028,
704.737,
554.272, 416.437, 300.0, 1000.0, 865.694, 737.0, 613.323,
497.25,
392.272, 300.0, 1000.0, 858.886, 742.251, 639.968, 546.709,
460.487,
379.456, 300.0, 1000.0, 828.19, 717.181, 634.776, 565.75,
503.477,
443.758, 380.218, 300.0, 1000.0, 745.895, 647.179, 591.806,
551.7,
517.869, 486.288, 452.623, 405.474, 300.0, 750,500.0, 500.0,
500.0, 500.0,
500.0, 500.0, 500.0, 500.0, 500.0,400]


V=[Z+i*(X-Z)/n for i in range(n+1)]
W=[Z+i*(Y-Z)/n for i in range(n+1)]
L=[text(round(R3C[0],2),Z,color='black',fontsize=9)]
for i in [1,2,..,n]:
    for j in range(i+1):
        L.append(text(round(R3C[(i^2+i+2*j)/2],2),V[i]+(j/i)*
(W[i]-V[i]),color='black',fontsize=9))
show(sum(L),axes=False,figsize=[5,3])
```

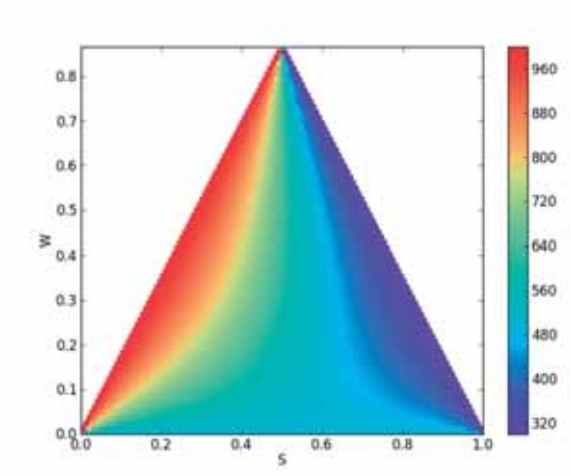
```
                                650.0
                           1000.0  300.0
                      1000.0 648.63  300.0
                 1000.0  772.6 519.19  300.0
            1000.0 828.74 639.06 454.83  300.0
       1000.0 856.03 704.74 554.27 416.44  300.0
  1000.0 865.69 737.0  613.32 497.25 392.27  300.0
1000.0 858.89 742.25 639.97 546.71 460.49 379.46  300.0
1000.0 828.19 717.18 634.78 565.75 503.48 443.76 380.22  300.0
1000.0 745.89 647.18 591.81 551.7  517.87 486.29 452.62 405.47  300.0
750.0  500.0  500.0  500.0  500.0  500.0  500.0  500.0  500.0  500.0  400.0
```

```
T=placat(1,.866,9,'hsv',1000,500,300)
```



Tamén podemos aumentar o número de puntos da reixa para obter unha representación colorística máis suave.

```
T=placat(1,.866,70,'rainbow',1000,500,300)
```



Exercicio 15

Sexa unha reixa hexagonal regular como a que corresponde ao número hexagonal centrado **numpolc**(6.10):

Se os puntos dos bordos están a temperaturas [N, NE, SE, S, SW , NW], a que temperatura están os puntos interiores?

3.2.3.2. Exercicios Problemas Inversos non Lineais

Exercicio 1:

Sexa $\Omega = [0,1] \times [0, 2\pi]$. Atopar o lugar dos puntos fixos das funcións da familia

$$\{f_p : \Omega \to \mathbb{R}^2 \,|\, p \in [\frac{1}{2}, 2\pi - \frac{1}{2}]\}$$

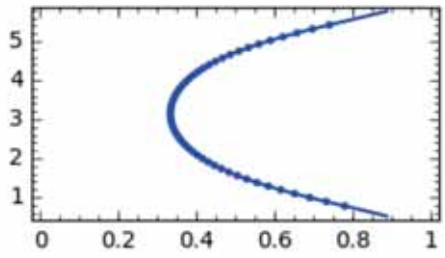
onde

$$f_p(x,e) = [\frac{1 + x\cos(y)}{2}, p + \frac{x\sin(y)}{2}]$$

Solución:

É evidente que $f_p(\Omega)$ é o disco de radio 1/2 e centro $(1/2, p)$. Xa que logo, $f_p(\Omega) \subset \Omega$ e f_p é contractiva en Ω. Consideramos os p da lista L=[.5,.6,..,$2\pi - \frac{1}{2}$] do intervalo [$\frac{1}{2}, 2\pi - \frac{1}{2}$] e calculamos o punto fixo de cada f_p:

```
L=[.5,.6,..,2*pi-1/2]
PT=[]
for p in L:
    f(x,y)=[(1+x*cos(y))/2,p+x*sin(y)/2]
    v=vector([.5,p])
    Bv=Banach(f,v,10^(-12))[0]
    PT.append(Bv)
show(point(PT)+implicit_plot(f[0]-x,(x,0,1),(y,1/2,2*pi-
(1/2))),aspect_ratio=1/10, figsize=[3,2])
```



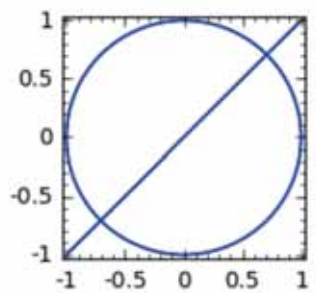
Vemos que o lugar dos puntos está sobre a curva $x = \frac{1+x\cos y}{2}$

Exercicio 2:

Resolver o sistema de ecuaciones non lineais:

$$\begin{cases} x^2 + y^2 - 1 = 0 \\ x = y \end{cases}$$

```
f(x,y)=[x^2+y^2-1,x-y]
F0=implicit_plot(f[0], (x, -1, 1), (y, -1,1))
F1=implicit_plot(f[1], (x, -1, 1), (y, -1,1))
show(F0+F1, figsize=[2.2,2.2])
```



O debuxo invítanos a iniciar a procura en (1,1) e (-1,-1).

```
P=vector([1,1])
show(newton(f,P,10^(-12))[0].n())
P=vector([-1,-1])
show(newton(f,P,10^(-12))[0].n())
```

$$(0.707106781186548,\ 0.707106781186548)$$

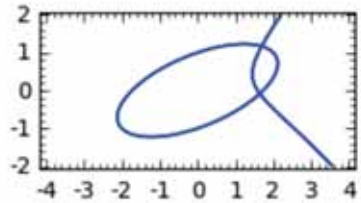
$$(-0.707106781186548,\ -0.707106781186548)$$

Exercicio 3:

Achar as solucións reais do sistema de ecuacións non lineais

$$\begin{cases} x^3 + 3xy - 5y^2 = 4 \\ x^2 + 3y^2 - 2xy = 3 \end{cases}$$

```
f(x,y)=[x^3+3*x*y-5*y^2-4,x^2+3*y^2-2*x*y-3]
F0=implicit_plot(f[0], (x, -4, 4), (y, -2,2))
F1=implicit_plot(f[1], (x, -4, 4), (y, -2,2))
show(F0+F1,figsize=[2.5,2.5])
```




```
P=vector([2,-1])
show(newton(f,P,10^(-12))[0])
P=vector([2,2])
show(newton(f,P,10^(-12))[0])
```

$$(1.64446677519593,\ -0.0835491609064199)$$

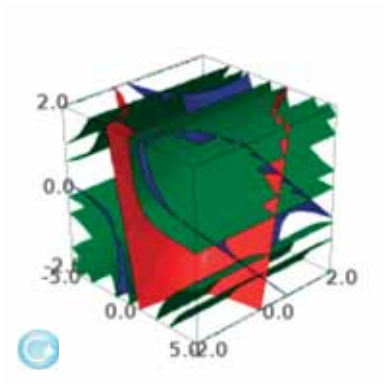
$$(1.69505707066860,\ 1.16627355039090)$$

Exercicio 4:

Resolver o sistema de ecuacións non lineais:

$$\begin{cases} xy - z - 1 = 0 \\ xcos(y+z) - 1 = 0 \\ sin(xz) = \frac{1}{2} \end{cases}$$

```
f(x,y,z)=[x*y-z-1,x*cos(y+z)-1,sin(x*z)-.5]
D=implicit_plot3d(f[0],(x,-5,5),(y,-2,2),
(z,-2,2),color='red',figsize=[2.5,2.5,2.5])
DD=implicit_plot3d(f[1],(x,-5,5),(y,-2,2),
(z,-2,2),color='blue',figsize=[2.5,2.5,2.5])
DDD=implicit_plot3d(f[2],(x,-5,5),(y,-2,2),
(z,-2,2),color='green',figsize=[2.5,2.5,2.5])
show(D+DD+DDD)
```



O debuxo indícanos que existe un conxunto infinito de puntos illados que verifican o sistema. Algúns deles podemos calculalos iniciando **newton** dende (3,-1,2), (3,0,.2), (3,0,4) ou (4,4,4).

```
P=vector([3,-1,2])
show(newton(f,P,10^(-12))[0])
P=vector([3,0,.2])
show(newton(f,P,10^(-12))[0])
P=vector([3,0,4])
show(newton(f,P,10^(-12))[0])
P=vector([4,4,4])
show(newton(f,P,10^(-12))[0])
```

$$(3.43453030147893,\ 0.513099779980445,\ 0.762256742025008)$$

$$(1.82617982146930,\ 0.704595498626721,\ 0.286718081890218)$$

$$(3.55602356367661,\ 1.31637748680249,\ 3.68106936176306)$$

$(1.96961341107623,\ 2.80220061798983,\ 4.51925191771888)$

Exercicio 5:

Elabora unha función **prestamo**(C,r,n) que nos dea a cantidade fixa que hai que aboar mensualmente para liquidar en n meses un préstamo de capital C ao r% mensual e a cantidade que debemos ao final dun mes calquera. Calcula a débeda trala mensualidade 37 no caso dun préstamo de 20000 € ao 1% mensual a pagar en 60 meses.

```
def prestamo(C,r,n):
    var('Q')
    X=[C]
    for k in range(n):
        X.append((1+r)*X[k:][0]-Q)
    S=solve(X[-1],Q,solution_dict=True)
    cuota=S[0][Q].n()
    Y=[C]
    for k in range(1,n):
        Y.append(X[k](Q=cuota).n())
    return cuota,Y
```

```
P=prestamo(20000,1/100,60)
print 'Cota'
show(P[0])
print 'Débeda 37'
show(P[1][37])
```

```
Cota

444.888953698036


Débeda 37

9100.56885965068
```

Exercicio 6:

Sexan A e B en $\mathcal{M}_2(\mathbb{R})$ tales que:

$$A * B = \begin{pmatrix} 1 & 2 \\ 8 & a \end{pmatrix} \quad \text{e} \quad B * A = \begin{pmatrix} 2 & 1 \\ 4 & b \end{pmatrix}$$

Determinar os valores de a e b.

Solución:

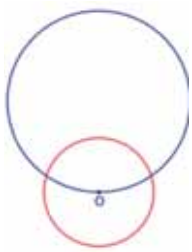
Como $det(A*B) = det(B*A)$ e $trace(A*B) = trace(B*A)$, solve determínanos a solución:

```
var('a b')
AB=matrix([[1,2],[8,a]])
BA=matrix([[2,1],[4,b]])
E=[det(AB)-det(BA),AB.trace()-BA.trace()]
S=solve(E,a,b,solution_dict=true)
print 'a=',S[0][a]
print 'b=',S[0][b]
```

```
a= -10
b= -11
```

Exercicio 6:

Con centro nun punto O da circunferencia azul de radio R, trázase unha circunferencia vermella de radio $r \leq 2R$.



1) Cal é a área $A(r,R)$ da rexión convexa encerrada por ambas circunferencias?

2) Resolver o problema inverso $A(r,5) = 60$.

3) Resolver os problemas inversos $A(r,R) = \frac{\pi R^2}{2} \quad \forall R \in \mathbb{R}$

Solución:

1) É fácil ver que a función $A : [0,2R] \times \{R\} \to \mathbb{R}$ é

$$A(r,R) = 4R^2 \int_0^{asin(\frac{r}{2R})} sen^2 t dt + r^2(\frac{\pi}{2} - asin(\frac{r}{2R}))$$

Podemos evaluala como segue:

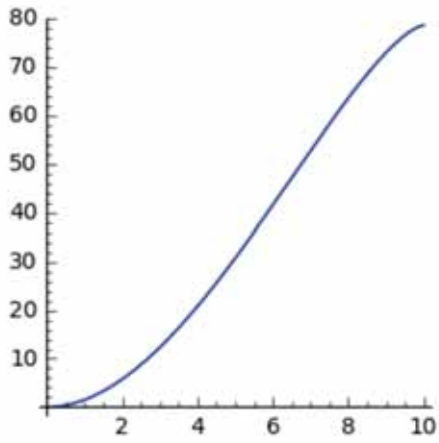
```
var('t b')
assume (b>0)
B(r,R,b)=4*R^2*integral((sin(t))^2,t,0,b)+r^2*((pi/2)-b)
A(r,R)=B(r,R,asin(r/(2*R)))
show(A(r,R))
```

$$\frac{1}{2}\left(\pi - 2\arcsin\left(\frac{r}{2R}\right)\right)r^2 + R^2\left(2\arcsin\left(\frac{r}{2R}\right) - \sin\left(2\arcsin\left(\frac{r}{2R}\right)\right)\right)$$

2) Para R=5 podemos graficarla e buscar a solución do problema inverso en [6,8].

```
show(plot(A(r,5),(r,0,10)),figsize=[3,3])
show(find_root(A(r,5)-60,6,8))
```

7.66221277006



3) Empezamos resolvendo os problemas inversos $A(r,R) = \frac{\pi R^2}{2}$ para R=1,2,3,4 e 5.

```
L=[]
for R in range(1,6):
    L.append(find_root(A(r,R)-(1/2)*pi*R^2,R,2*R))
L
```

```
[1.1587284730181215,
 2.317456946036243,
 3.4761854190543646,
 4.634913892072486,
 5.793642365090608]
```

Esta lista suxírenos que a solución do problema inverso $A(r,R) = \frac{\pi R^2}{2}$ é L[0]*R. Comprobámolo:

```
(A(L[0]*R,R)-pi*R^2/2).n()
```

```
-1.77635683940025e-15
```

```
(sqrt(4/3)).n()
```

```
1.15470053837925
```

M=1.1587284730181215 é a constante de Mataix. Obsérvese que MR non é o lado do

hexágono circunscrito á circunferencia de radio R.

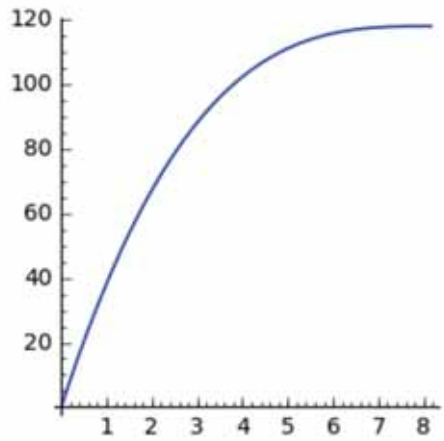
Exercicio 7:

Sexa H a altura dun tetraedro regular de 10 m de arista.

Representar a función $V : [0, H] \to \mathbb{R}$ onde V(h) é o volume de auga que contén o tetraedro, apoiado no chan sobre unha das súas caras, se o enchemos ata a altura h. Resolver a ecuación $V(h) = \frac{V(H)}{3}$

```
a=10
H=sqrt(2/3)*a
V(h)=(sqrt(3)/4)*(a/H)^2*integrate((H-x)^2,x,0,h)
show(plot(V,(h,0,H),figsize=[3,3]))
show(find_root(V(h)-(V(H)/3),0,2))
```

1.03221118305



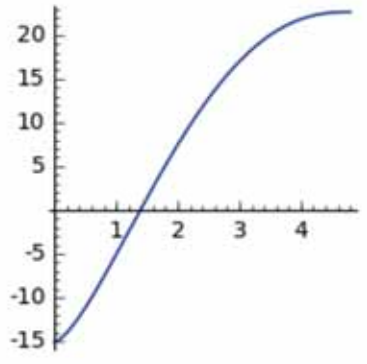
Exercicio 8:

Un depósito cónico de radio 3 e altura 4, apoiado no chan sobre a súa base, contén auga ata o 40% da súa capacidade. A que altura chegará a auga se o envorcamos e mantémolo apoiado sobre unha das súas xeratrices?

```
var('x,h')
H=24/5
hx=5*(H-x)/H
ax=6*x/H
rx=sqrt(ax*(6-ax))
assume(h>0)
Vg(h)=(4/3)*integrate(hx*rx,x,0,h)
```

```
f(h)=Vg(h)-24*pi/5
show(plot(f,(h,0,H)),figsize=[2.5,2.5])
print 'A auga chegará a', f.find_root(1,2),'m  de altura'
```

```
A auga chegará a 1.39400680898 m  de altura
```

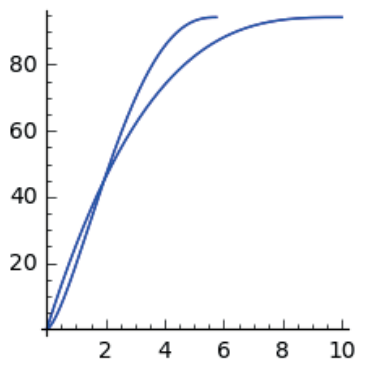


Exercicio 9:

Ata que altura debemos encher de auga un depósito cónico de 3 m de radio e 10 m de altura, colocado no chan sobre a súa base, para que, ao tombalo sobre unha xeratriz, a auga manteña a súa altura?

```
Vb(h)=(9*pi/100)*integrate((10-x)^2,x,0,h)
Db=plot(Vb,(h,0,10))

H=60/sqrt(109)
hx=6*x/H
assume(h>0)
Vg(h)=(4/3)*integrate((109/60)*(H-x)*sqrt(hx*(6-hx)),x,0,h)
Dg=plot(Vg,(h,0,H))
show(Db+Dg, figsize=[2.5,2.5])
show((Vb-Vg).find_root(1,3))
```

```
1.93990004999
```

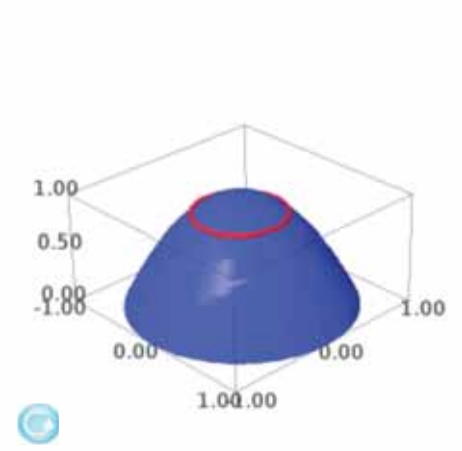


Exercicio 10:

Sexa S o sólido do exercicio 37 da Worksheet 0 no que supoñemos unha densidade de masa constante. Cal é a altura do seu centro de gravidade se o supoñemos en equilibrio apoiado no chan sobre a súa cara alabeada?

```
F(p,r,t)=[p*r*cos(t),p*r*sin(t),p*(1-r^2)]
V1=derivative(F(1,r,t),r)
V2=derivative(F(1,r,t),t)
NOR=vector(TS(V1.cross_product(V2)))
CDG=vector([0,0,1/3])
S=solve(list(F(1,r,t)+p*NOR-CDG),[p,r,t],solution_dict=True)
r0=S[0][r]
html("<h4>A linea vermella é o lugar dos puntos de apoio no
chan</h4>")
PAR=parametric_plot3d(F(1,r,t),(r,0,1),(t,0,2*pi),figsize=
[3,3,3])+parametric_plot3d(F(p,1,t),(p,0,1),(t,0,2*pi),figsize=
[3,3,3])
L=parametric_plot3d(F(1,r0,t),
(t,0,2*pi),color='red',thickness=5,figsize=[3,3,3])
show(PAR+L,aspect_ratio=1)
```

A linea vermella é o lugar dos puntos de apoio no chan




```
html("<h4>A altura do CDG sobre o chan cando se apoia na cara
alabeada é</h4>")
show((norma(CDG-F(1,r0,t)).simplify_trig()).n())
```

A altura do CDG sobre o chan cando se apoia na cara alabeada é

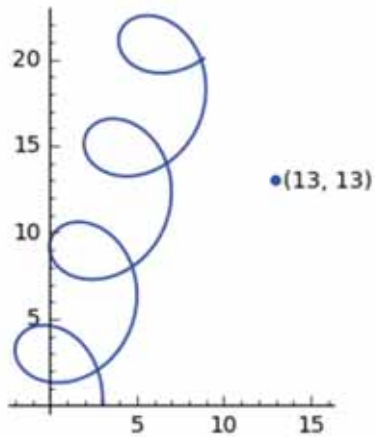
0.645497224367903

Exercicio 11:

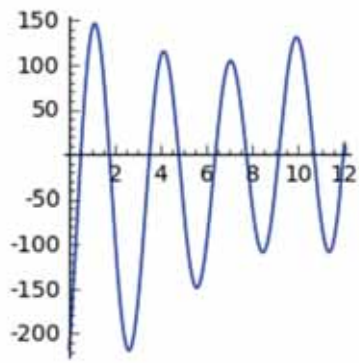
No plano euclídeo, un punto P percorre a recta y=3x a $2\frac{m}{s}$ e un punto Q vira en torno a P a $2\frac{rad}{s}$ e $3m$ de radio.

Se no instante inicial P está en (0,0) e Q en (3,0), achar a posición de P cando Q estea á mínima distancia de (13,13).

```
c(t)=[t,3*t]
M=norma(diff(c,t))
P(t)=[(2/M)*t, (2/M)*3*t]
G(t)=[3*cos(2*t),3*sin(2*t)]
Q=P+G
parametric_plot(Q,(t,0,12),figsize=
[3,3])+point([13,13],pointsize=20)+text((13,13),
(16,13),color='black')
```



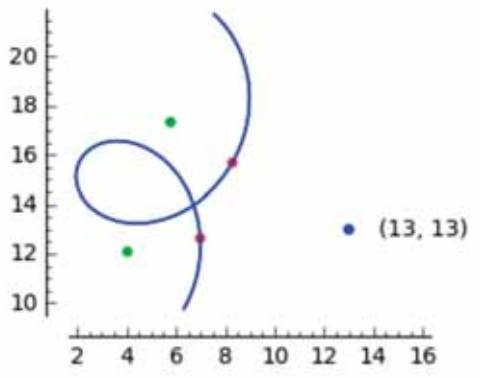

```
DQ(t)=(Q(t)-vector([13,13]))*(Q(t)-vector([13,13]))
d(t)=diff(DQ,t)
plot(d,(t,0,12),figsize=[2.5,2.5])
```




```
t0=d.find_root(6,7)
D1=parametric_plot(Q,(t,6,10),figsize=[3,3])+
```

```
point([13,13],pointsize=20)+
point(Q(t0),hue=0,pointsize=20)+point(P(t0),hue=0.3,pointsize=20)
t1=d.find_root(9,10)
D2=parametric_plot(Q,(t,6,10),figsize=[3,3])+
point([13,13],pointsize=20)+
point(Q(t1),hue=0,pointsize=20)+point(P(t1),hue=0.3,pointsize=20)+
text((13,13),(16,13),color='black')
show(D1+D2)
print 'O mínimo absoluto é o menor dos dous mínimos relativos.'
print 'Alcánzase en t=',t1
print 'P=',P(t1).n()
```

```
O mínimo absoluto é o menor dos dous mínimos relativos.
Alcánzase en t= 9.13390883727
P= (5.77679117322218, 17.3303735196666)
```



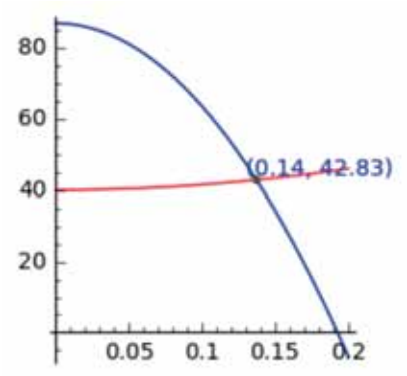
Exercicio 12:

Achar a ecuación característica dunha bomba hidraúlica sabendo que para caudais en litros por segundo Q=[40,100,180] dános alturas en metros H=[83.26,63.58,11.07].

Se a bomba se instala nun sistema de distribución de ecuación característica $H = 40 + 150Q^2$, cal será o punto de funcionamento?

```
C=[.04,.1,.18]
A=[83.26,63.58,11.07]
F(Q)=[1,Q,Q^2]
N=nube(C,A,F)
H(Q)=N[2](x=Q)
show(H(Q))
H1(Q)=40+150*Q^2
S=solve(H(Q)-H1(Q),Q,solution_dict=True)
Q0=S[1][Q].n()
H0=H(Q0)
R=vector(Round([Q0,H0],2))
show(plot(H(Q),(Q,0,.2))+plot(H1(Q),
(Q,0,.2),color='red')+point((Q0,H0),color='green',pointsize=15)+te
[0.18,46]),figsize=[2.5,2.5])
```

$$-2345.53571428569\,Q^2 + 0.374999999994657\,Q + 86.9978571428573$$



Exercicio 13:

Achar o prezo de equilibrio entre a demanda D e a oferta O ao prezo unitario P.

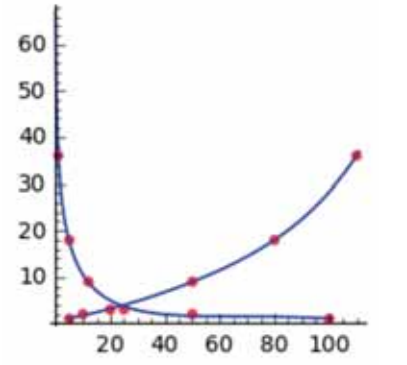
D=[100,50,25,12,5,1]

O=[5,10,20,50,80,110]

P=[1,2,3,9,18,36]

```
D=[100,50,25,12,5,1]
O=[5,10,20,50,80,110]
P=[1,2,3,9,18,36]
FD(x)=[1,x,x^2,log(1+10*x)]
ND=nube(D,P,FD)
FO(x)=[1,x,x^2,exp(x/50),exp(x/100)]
NON=nube(O,P,FO)
show(ND[0]+ND[1]+NON[0]+NON[1],figsize=[2.5,2.5])
s=find_root(ND[2]-NON[2],24,25)
print 'Prezo de equilibrio=',NON[2](x=s)
```

```
Prezo de equilibrio= 3.83179199045834
```

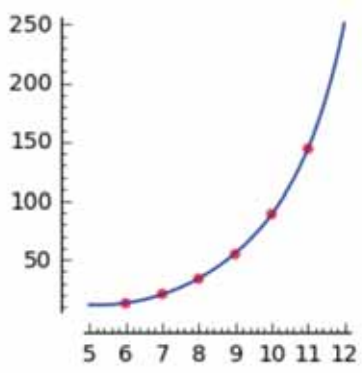


Exercicio 14:

Sexa X=[6,7,8,9,10,11] e Y=[F[n] for n in X] sendo F a lista Fibonacci. Cal é o axuste dos datos X,Y na variedade $[1, x, x^2, exp(x)]$?

```
X=range(6,12)
Y=[Fibonacci(x) for x in X]
G(x)=[1,x,x^2,exp(x)]
N=nube(X,Y,G)
show(N[0]+N[1],figsize=[2.5,2.5])
show(N[2])
```

$$2.74331015960817\,x^2 - 28.8711978038863\,x + 0.000706944943916312\,e^x + 87.40045283$$



Exercicio 15:

Sabemos que X=[1,3,5,7,9,,...] é unha sucesión aritmética de orde 5 e diferencia 7. Achar o seu termo xeral.

Solución:

Por ser aritmética de orde 5 e diferencia 7 ha de cumprir a relación de recurrencia

$$X[n+5] - 5X[n+4] + 10X[n+3] - 10X[n+2] + 5X[n+1] - X[n] = 7 \quad \forall n$$

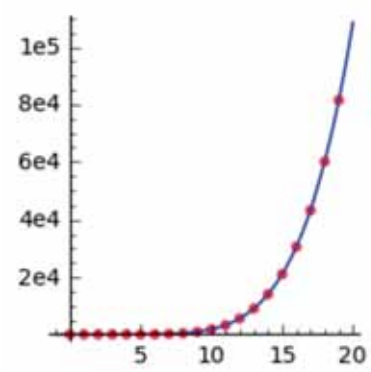
Podemos achar os n primeiros termos coa función **TG5**(n,CI)

```
def TG5(n,CI):
    X=CI
    for k in range(n-len(CI)):
        X.append(5*X[-1]-10*X[-2]+10*X[-3]-5*X[-4]+X[-5]+7)
    return X
```

Sendo CI=[1,3,5,7,9], buscamos o polinomio de grado 5 que mellor axuste os datos X=range(20), Y=TG5(20,CI):

```
CI=[1,3,5,7,9]
X=range(0,20)
Y=TG5(20,CI)
F(x)=[1,x,x^2,x^3,x^4,x^5]
N=nube(X,Y,F)
show(N[0]+N[1],figsize=[2.5,2.5])
show(N[2])
```

$$0.0583333333333333\,x^5 - 0.583333333333333\,x^4 + 2.04166666666667\,x^3 - 2.916666666($$



O termo n-ésimo da sucesión vén dado por $f(n)$ e os 20 primeiros termos son:

```
f(n)=N[2](x=n)
show(f(n))
show([round(f(n),3) for n in range(20)])
```

$$0.0583333333333333\,n^5 - 0.583333333333333\,n^4 + 2.04166666666667\,n^3 - 2.916666666($$

$$[1.0, 3.0, 5.0, 7.0, 9.0, 18.0, 55.0, 162.0, 409.0, 901.0, 1785.0, 3257.0, 5569.0, 9036.0, 140$$

Exercicio 16:

Achar as curvas de Wöhler dos seguintes ensaios de fatiga de materiais onde X son ciclos e Y tensións:

a)

$X = [400000, 300000, 250000, 200000, 150000, 100000, 50000]$

$Y = [100, 200, 300, 400, 500, 600, 700]$

b)

$X = [13000, 17000, 25000, 35000, 50000, 100000, 200000, 500000]$

$Y = [400, 350, 300, 250, 200, 150, 100, 50]$

c)

$X = [38543, 98738, 289322, 820410, 98732127, 73421023]$

$Y = [170, 150, 130, 110, 90, 70]$.

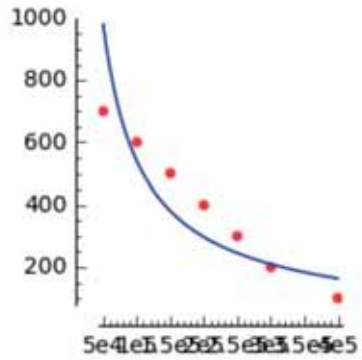
Solución:

A curva de Wöhler é un axuste segundo o modelo $f(x, a, b) = ax^b$. Empezamos tomando logaritmos e axustando na variedade $[1.log(x)]$.

```
print 'a)'
X=[400000,300000,250000,200000,150000,100000,50000]
Y=[100,200,300,400,500,600,700]
LY=[log(y) for y in Y]
F(x)=[1,log(x)]
N=nube(X,LY,F)
a=exp(N[2](x=1))
b=(N[2](x=exp(1))-N[2](x=1)).n()
W(x)=a*x^b
show(point(zip(X,Y),hue=0,pointsize=20)+plot(W,
(x,X[-1],X[0])),figsize=[2.5,2.5])
show(W(x))
```

a)

$$\frac{1.08937468005287 \times 10^7}{x^{0.860983478324387}}$$

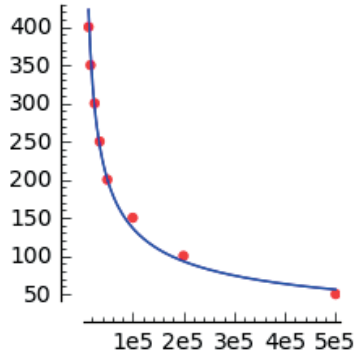



```
print 'b)'
X=[13000,17000,25000,35000,50000,100000,200000,500000]
Y=[400,350,300,250,200,150,100,50]
LY=[log(y) for y in Y]
F(x)=[1,log(x)]
N=nube(X,LY,F)
a=exp(N[2](x=1))
```

```
b=(N[2](x=exp(1))-N[2](x=1)).n()
W(x)=a*x^b
show(point(zip(X,Y),hue=0,pointsize=20)+plot(W,
(x,X[0],X[-1])),figsize=[2.5,2.5])
show(W(x))
```

b)

$$\frac{80182.9578974038}{x^{0.553775725837834}}$$

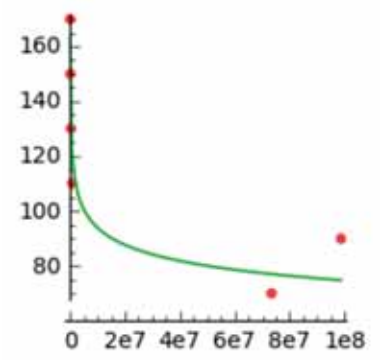


Podemos axustar directamente os datos ao modelo $f(x,a,b) = ax^b$:

```
print 'c)'
f(x,a,b)=a*x^b
C=[38543,98738,289322,820410,98732127,73421023]
Y=[170,150,130,110,90,70]
m(x)=f
S=ajustamodel(C,Y,m,.3)
show(S[0]+S[1],aspect_ratio=10^6, figsize=[2.5,2.5])
show(S[2])
```

c)

$$\frac{468.744524232}{x^{0.0996826270543}}$$



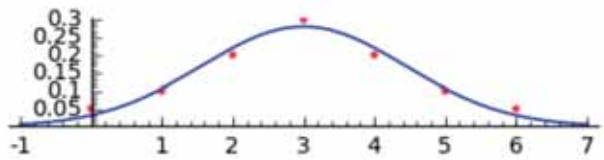
Nos casos a) e b) propoñemos buscar mellores axustes nas variedes $[1, x, x^2, log(x)]$ e $[1, x, log(x), log(log(x))]$.

Exercicio 17:

Axustar mediante unha campá de Gauss, os datos $X = [0, 1, 2, 3, 4, 5, 6]$ e $Y = [0.05, 0.1, 0.2, 0.3, 0.2, 0.1, 0.05]$.

```
f(x,a,b)=(1/(2*pi*b)^(1/2))*exp(-((x-a)/b)^2)
X=[0,1,2,3,4,5,6]
Y=[.05,.1,.2,.3,.2,.1,.05]
af(a,b)=ortotangente(X,Y,f)
N=newton(af,[3,1],10^(-6))
g=EtV(f,x,N[0])
print 'a=',N[0][0], 'b=',N[0][1]
DC=plot(g,(x,min(X)-1,max(X)+1))
D=[[X[i],Y[i]] for i in range(len(X))]
DD=point2d(D,rgbcolor=(1,0,0),pointsize=10,aspect_ratio=1)
show(DC+DD,aspect_ratio=5,figsize=[4,3])
```

```
a= 3.00000000000000 b= 2.02934466671219
```



Bibliografía

1. Bollobás, B. Modern graph theory. New York, Springer-Verlag, 1998.
2. Casamayou, N. Cohen, G. Connan, T. Dumont, L. Fousse, F. Maltey, M. Meulien, M. Mezzarobba, C. Pernet, N. M. Thiéry, P. Zimmermann. Calcul mathématique avec Sage. http://sagebook.gforge.inria.fr
3. Ciarlet, P.G. Introduction à l'analyse numérique matricielle et à l'optimisation. Paris, Masson, 1990.
4. Corbacho Rosas, E., Rodríguez de Miguel, V. Cálculo avanzado, 2017. https://corbacho.webs.uvigo.es
5. Corbacho Rosas, E., Vidal Vázquez, R. Matemáticas de la Especialidad y Métodos Matemáticos, 2017. https://corbacho.webs.uvigo.es.
6. Dedeckind. Que son y para que sirven los números.
7. Harary, F., Graph theory. Encyclopedia of Mathematics and its applications, vol. 21, Gian-Carlo Rota Editor. Department of Mathematics. Massachusetts Institute of Technology. Cambridge. Published by the Press Syndicate of the University of Cambridge 1984.
8. Javaheri, M. Dirichlet problems in locally finite graphs. Discrete Appl. Math., 155, (2007), 2496--2506.
9. Luenberger, D.G. Optimization by vector space methods. New York, J. Wiley, 1969.
10. Merino, L., Santos, E. Álgebra Lineal con Métodos Elementales. Universidad de Granada. 1997.
11. Rubio de Francia, J.L., Memoria Analisis II. Zaragoza, 1975.
12. Rudin, W. Análisis real y complejo (3ª edición). Madrid, Mc. Graw-Hill, 1988.
13. Somoza López. M.C. Estructuras métricas en grafos, Tesis. Departamento de Matemática Aplicada I, Universidade de Vigo, 2015. https://corbacho.webs.uvigo.es.
14. Santaló, L. La matemática: Una filosofía y una técnica. Barcelona, Editorial Ariel, 1994.
15. SageMath http://www.sagemath.org/index.html.
16. Stein, W. SAGE for Power Users, 2012. http://www.wstein.org/books/sagebook/sagebook.pdf