SAGE

Como entrar al servidor:

cd Desktop/SAGE-noteb/

sage –notebook=jupyter

Para cambiar de SAGE a Jupyter: carpeta bin

CAPITULO 2: SAGE COMO CALCULADORA AVANZADA

1. Como obtener **información sobre los parámetros de las funciones**

Nombre_funcion**(** y tabulamos

Si no conocemos el nombre de la funcion: escribimos el nombre y tabulamos

Si necesitamos informacion sobre los parametros: nombre_funcion**?** y evaluamos

2. Hay otra clase de instrucciones a las que llamamos **metodos**

(objeto).metodo()

Ej.: Instrucción-> factor(2^137+1)

Metodo-> (2^137+1).factor()

2.1 Aritmética elemental

- Cociente: / (se muestra en notación racional)
- División entera: //
- Resto: %

2.2 Listas y tuplas

En el Capítulo 3

Listas -> L=[….]

Tuplas-> t=(….) las coordenadas de puntos son tuplas

2.3 Funciones

- f(x)=log(x) es el ln
- f(x)=sin/cos/tan(x)
- f(x)=sqrt(x)

Para definir funciones:

- f(x,y,z)=….
- **def** f(x): **return** la funcion

Variables y expresiones simbolicas

- Para cualquier variable distinta de x: **var**(‘a b c’)
- <u>**Imprimir expresiones**</u>
    - codigo: **print** s; s2; s3; s4; p
    - formato matematico: **show**([s, s2, s3, s4, p])
    - sustituyendo valores: **print** s(a=1, b=1, c=1), s(a=1, b=2, c=3)
    - a veces simplificadas: **show**(s)
- <u>**Simplificar expresiones**</u>
    - Nombre.**expand**()
    - Nombre.**factor**()
    - Nombre.**simplify** ->tabular para ver todos los tipos

Variables booleanas

El ==, <, <= sirven para comparar y devuelven True/False

<u>**2.4 Graficas**</u>

- **Point**(punto o lista), **points**(lista), **point2d**(lista), **point**(3d) -> <u>paradoja de Bertrand</u>
- **Line**(lista), **line2d**(lista), **line3d**(lista) -><u>la ruina del jugador</u>
- **plot**(f(a=0, b=1),(x,-2,2))) grafica de la funcion de una variable f(x,a,b) con x en el intervalo [-2,2], a=0 y b=1.
- **plot3d**(g,(x,-10,10),(y,-10,10)) grafica de la funci´on de dos variables g sobre el cuadrado [-10,10]x[-10,10].
- **Parametric_plot**([f(t),g(t)],(t,0,2)) grafica de la curva dada en parametricas mediante dos funciones de t, f y g, con la variable t variando en el intervalo [0,2].
- **Parametric_plot3d**([f(u,v),g(u,v),h(u,v)],(u,0,2),(v,0,2)) esboza una grafica de la superficie dada en parametricas mediante tres funciones de u y v, f, g y h, con las variables u y v en el intervalo [0,2]. Los puntos de la superficie son entonces los que se obtienen mediante x=f(u,v),y=g(u,v),z=h(u,v)

- **Implicit_plot**(f,(x,-5,5),(y,-5,5)) representacion de la parte de la curva f(x,y)=0 contenida en el cuadrado [-5,5]x[-5,5]. Representacion de una curva “en implicitas”.

- **Implicit_plot3d**(f,(x,-5,5),(y,-5,5),(z,-5,5)) se representa una parte de la superficie f(x, y, z) = 0 contenida en el cubo [-5,5]x[-5,5]x[-5,5]. Representacion de una superficie “en implicitas”.

2.5 Cálculo

Para ver el valor numérico de una expresión: expresion**.n()** ó **n**(expresion)

Ecuaciones

- **Solve**(ecuacion)   acordarse de declarar las variables antes

- **Find_root**(f, a,b) busca sol de la ecuacion f en el intervalo [a,b]

Límites

- **.limit()** o **limit**(expresion) si no existe devuelve **und** o **ind** o la definicion de la funcion

Series

- **sum**(expresion, variable, limite inferior, limite superior)

Calculo diferencial

- **.derivative()**, o **derivative**(expresion)
  - Para derivar funciones de varias variables, es decir calcular derivadas parciales (i.e. derivar una funcion de varias variables respecto a una de ellas manteniendo las otras variables en valores constantes), basta especificar la variable con respecto a la que derivamos y el orden hasta el que derivamos:
    F(x,y)=xˆ2∗sin(y)+yˆ2∗cos(x)
    show([F.derivative(y,2),derivative(F,x,1).derivative(y,1)])

Desarrollo de Taylor

- **taylor**(f,x,0,20) donde x0=0 y el polinoimio lo queremos de grado 20

Calculo integral

- **.integral()** e **.integrate(**variable, valor1, valor2**)**

- **Numerical_integral**(f,pi/5,pi/4) o N(f**.integrate**(x,pi/5,pi/4)) para aproximacion numerica
  - Numerical_integral devuelve una tupla con el valor aprox y una cota para el error

2.5 Algebra lineal

Construccion de matrices

- **Matrix(**conjunto al que pertenecen los numeros(ZZ), filas como listas[[1],[2]…]**)**

Submatrices

- Para acceder a un elemento: A[0,0]
- En la notacion slice, dos indices separados por dos puntos, i:j, indican el corte entre el primer indice, incluido, hasta el segundo, que se excluye
- La matriz A[7:] consiste en las filas desde la octava, que tiene indice 7 porque hemos empezado a contar en 0, hasta la decima.
- La matriz A[:2] consiste en las filas primera y segunda, es decir, las de indice menor estrictamente que 2.
- Utilizando el doble iındice, podemos recortar recuadros mas concretos.
- A[2:5:2,4:7:2] produce [25 27] [45 47] se queda con las filas con indice del 2 al 4, saltando de dos en dos debido al 2 que aparece en tercer lugar en 2:5:2, y con las columnas con indices del 4 al 6 tambien saltando de dos en dos por el que aparece en 4:7:2. Siempre hay que recordar que las filas y columnas se numeran empezando en el cero.
- Por otra parte, tambien podemos usar indices negativos lo que conduce a quedarnos con filas o columnas que contamos hacia atras, de derecha a izquierda, empezando por las ultimas: A[-1] para la ultima fila y **A.column**(-1) para la ultima columna

Operaciones con matrices

- **Identity_matrix**(n) matriz identidad nxn
- **A.det()**, **det**(A)
- **A.rank()**, **rank**(A)
- **A.trace()**
- **A.inverse()**
- **A.transpose()**
- **A.adjoint()**
- **A.echelon form()**matriz escalonada de A, en el menor anillo en que vivan sus entradas. Importante para resolver sistemas de ecs

Espacios vectoriales

- E=**VectorSpace**(cuerpo,dimension) y sus elementos v=vector([1,2,3]) o v=E([1,2,3])

- M=**MatrixSpace**(cuerpo,filas,columnas)

Subespacios vectoriales

- L1 = V1**.subspace**([v1,v2])
- print **dim**(L1)
- L1**.degree()** dimensión de su espacio ambiente, L1**.ambient vector space()**
- L1**.intersection**(L2)

Bases y coordenadas

- L1 = V1**.subspace with basis**([v1,v2,v3])
- print L1**.basis()** o L1**.basis matrix()**
- print (L1**.coordinates**(v1), L2**.coordinates**(v1)) nos da las coordenadas de un vector en la base del espacio

Producto escalar

- Podemos definir un espacio vectorial con una forma bilineal mediante V = VectorSpace(QQ,2, **inner_product_matrix**=[[1,2],[2,1]])
- u**.dot product**(v) producto escalar
- u**.cross product**(v) producto vectorial
- u**.pairwise product**(v) producto elemento a elemento
- **norm**(u), u**.norm()**, u**.norm(2)** norma Euclidea
- u**.norm(1)** suma de coordenadas en valor absoluto
- u**.norm(Infinity)** coordenada con mayor valor absoluto
- u**.inner product**(v) producto escalar utilizando la matriz del producto escalar

CAPITULO 3: ESTRUCTURAS DE DATOS

Las estructuras de datos son iterables, para todas tiene sentido hacer un bucle for: **for** item **in** <estructura_de_datos>: …..

Para comprobar si un dato está: dato **in** <estructra_de_datos>

Para obtener el numero de datos que hay: **len()**

3.1 Datos básicos: tipos

- A**.type()** o **type**(A) nos informa del tipo que tiene el objeto
- Hay dos tipos diferentes de enteros: los enteros de Python y los enteros de Sage. La unica diferencia, en la practica, es que no es posible aplicar a los enteros de Python los metodos disponibles para los enteros de Sage. Por ejemplo, los enteros de Sage tienen el metodo **.digits()**

3.2 Listas

Seleccionamos con L[j:k:d], las sublistas. Si no se especifica el tercer argumento, los saltos son de uno en uno. La ausencia de alguno de los dos primeros argumentos, manteniendo los dos puntos de separacion, indica que su valor es el mas extremo (primero o ultimo).

El signo **+** concatena listas. Se abrevia la concatenacion k veces de una misma lista L con k∗L

Metodos y funciones con listas

- **list()** nos permite crear una copia exacta de una lista
- L**.sort()** cambia la lista L de numeros enteros por la lista ordenada pero mantiene el nombre L
- **srange(j,k,d)**: devuelve los numeros entre j (inclusive) y k (exclusive), pero contando de d en d elementos
    - **srange(k)**: devuelve srange(0,k,1). Si k es un natural, devuelve los naturales entre 0 (inclusive) y k (exclusive); y si k es negativo, devuelve una lista vacıa.
    - **srange(j,k)**: devuelve la lista srange(j,k,1). Si j y k son enteros, devuelve los enteros entre j (inclusive) hasta el anterior a k.
- **.reverse()** cambia la lista por la resultante de reordenar sus elementos dandoles completamente la vuelta.
- **.append(**elem**)** se añade un nuevo elemento, y solo uno, al final de la lista. Para mas de un elemento **.extend()**
- **sum(lista)** calcula la suma de los elementos de la lista

- numero de apariciones se averigua con el metodo **.count()**, la primera aparicion la ofrece **.index()**
- **.pop()** o .**remove()** borran
- **.insert()** añade en una posicion determinada
- **Del(L[3:5])** borra los elementos del rango indicado

Otras listas

[j^2 for j in [1..5]]   [j^2 for j in [1..20] if j.is prime()]

- k**.digits()** devuelve una lista cuyos elementos son sus dıgitos. Por ejemplo, 123.digits() devuelve la lista [3,2,1] (<u>atencion al orden</u>). Por defecto los dıgitos se consideran en la base 10, y una lista sin ceros a la izquierda. <u>Otra base de numeracion se indica como primer argumento</u>, y un <u>numero de dıgitos fijo</u>, m, se indica asignando al parametro padto dicho valor: **padto=m**.
- **.factor()** factoriza en primos

<u>3.3 Tuplas</u>

<u>No</u> podemos asignar nuevo valor a los elementos de una tupla

Notacion slice, **+** (contatena), **.index()** y **.count()** → como en tablas

Con el del producto, se repite la tuplas: 2∗(1,5) devuelve (1,5,1,5).

Para **cambiar un valor**:

1. t1,t2=t[:1],t[2:] ##cortamos la tupla en dos evitando el elemento a sustituir
2. t1+(3**,**)+t2 ## concatenamos intercalando la tupla (3**,**)

tuple() copia una tupla, **len()** <u>aplicada a tuplas</u> me da su tamaño

list(factor()), aplicada a un natural, devuelve una lista de pares

xgcd(a,b) devuelve una tupla con 3 valores: (d,u,v) maximo comun divisor y se verifica la identidad de Bezout: d=a·u+b·v.

<u>3.4 Cadenas de caracteres</u>

Se delimitan con comillas simples

str(k) convierte el entero k en una cadena de caracteres

+ (concatenacion) y ∗ (repeticion) se utilizan y comportan como en el caso de listas

len() es el numero de caracteres de la cadena

Para el acceso a subcadenas, se utiliza la notacion slice

cadena.count(cadena1) devuelve el numero de veces que la cadena1 aparece “textualmente” dentro de la cadena.

cadena.index(cadena1) devuelve el lugar en que la cadena1 aparece por primera vez dentro de la cadena

.split() trocea una cadena de caracteres. Si no hay argumento, se utiliza, por defecto, el espacio en blanco para trocear

Inverso a .split() viene dado por el metodo **.join()** o **join()**

.find() devuelve el primer indice en que aparece una subcadena en la cadena a que se aplica; si no aparece, devuelve -1

3.5 Conjuntos

Se utilizan las llaves para delimitar un conjunto

No podemos crear conjuntos con conjuntos o listas entre sus elementos

Otra manera de crear conjuntos es con el constructor **set(**contenedor**)**, que toma los elementos de cualquier contenedor. A=**set()** crea un conjunto sin elementos, el conjunto vacio

Los elementos de un conjunto no est´an ordenados ni indexados

Para ver si un elemento está en un conjunto: **elem in conjunto**

Operaciones

- **A|B** union
- **A&B** interseccion
- **A-B** diferencia
- **A<=B** comparacion. True si todos los elementos de A estan en B

len(A) es el numero de elementos de un conjunto

Añadimos un elemento a un conjunto mediante **A.add(1)**, y lo suprimimos con **A.remove(1)**. Si se quieren añadir mas elementos, contenidos en cualquier otro contenedor, basta aplicar el metodo **.update()**

3.6 Diccionarios

Definimos un diccionario mediante: diccionario = {clave1:valor1, clave2:valor2, ...}

Las claves son los identificadores de las entradas del diccionario y, por tanto, han de ser todas diferentes.

Definimos una nueva entrada, o cambiamos su valor, mediante **diccionario[clave]=valor** y suprimimos una entrada con **del diccionario[clave]**

Podemos crear un diccionario vacio con el constructor **dict()**

Si pasamos al constructor una lista de pares, este creara un diccionario con claves los primeros objetos de cada par, y valores los segundos

Un buen constructor de listas de pares es **zip().** Si se tienen dos listas, L1 y L2, la lista zip(L1,L2) esta formada por las parejas (L1[j],L2[j]) para j=0,1,..., el menor indice final posible

Los metodos **.keys()** y **.values()** producen listas con las claves y los valores, respectivamente, en el diccionario. El metodo **.items()** devuelve la lista de pares (clave,valor)

La instruccion **x in diccionario** devuelve True si x es una de las claves del diccionario

3.7 Conversiones

- <u>Tupla a lista o conjunto</u>: Para una tupla T **list(T) set(T)**
- <u>Lista a tupla</u>: **tuple(L)**
- <u>Cadena a lista</u>: C = 'abc' **list(C)**
- <u>Lista a cadena</u>: **join(list(C),sep="")**
- <u>Lista a conjunto</u>: para una lista L, **set(L)**. Suprime repeticiones en la lista
- <u>Conjunto a lista</u>: para un conjunto A, **list(A)**.
- <u>Diccionario a lista de pares</u>: Para un diccionerio D, **D.items()**.
- <u>Lista de pares a diccionario</u>: def convert list dict(L): / dict = {} / for item in L: // dict[item[0]]=item[1] / return dict

CAPITULO 4: TECNICAS DE PROGRAMACION

4.1 Funciones

La sintaxis para definir funciones es: def nombre funcion(arg 1,arg 2,...,arg n): / '''Comentarios sobre la funcion''' / instruccion 1 / instruccion 2 / etc. / return res 1,res 2,...,res m

4.2 Control del flujo

Bucles for

- Sintaxis: **for** elemento **in** contenedor: / instruccion 1 / instruccion 2 / etc ...
- **for j in srange(10)** va del 0 al 9

Contadores y acumuladores

En ocasiones el rango aparece como **xsrange()** en lugar de **srange()**. La diferencia fundamental es que el segundo rango genera una lista de enteros y luego la recorre, y el primero no genera la lista y va aumentando el valor del contador en cada vuelta. Para rangos grandes la segunda forma de la instrucci´on genera una lista enorme que puede saturar la memoria ram de la maquina

Función time: **%time** encima de la función

Otra sintaxis para bucles

- **[f(x) for x in L]** → lista
- **[f(x) for x in L if Condicion]** → lista con condición
- **map(f,L)** que es equivalente a **[f(item) for item in L]**

Bucles if

- Sintaxis: **if condicion1**: instruccion 1 instruccion 2 etc ... / **elif condicion2**: instruccion 3 instruccion 4 etc ... / **else:** instruccion 5 instruccion 6 etc ...
- Las diversas condiciones deben ser booleanas y puede haber tantas l´ıneas elif como queramos

Bucles while

- Sintaxis: **while condicion1:** instruccion 1 instruccion 2 etc ... / actualizacion de la condicion
- Cuando, dentro de la ejecucion de un bucle, el programa llega a una lınea con un **break** el bucle termina y el programa sigue ejecutandose a continuacion del bucle

Recursion

Igual que en prog, pero aquí son poco eficientes y tardan mucho

Orbitas

o(x0):={x0,f(x0)=:x1,f(f(x0))=:f2 (x0):=x2,f(f(f(x0)))=:f3 (x0):=x3,...,fn(x0):=xn,...}⇢X

Cuando trabajamos con el ordenador el conjunto X debe ser finito, y en ese caso todas las orbitas, siendo subconjuntos de X, seran tambien finitas y, por tanto, aparece necesariamente un punto xi0 =fi0(x0) de la orbita de x0 al que se vuelve cuando seguimos iterando, es decir, tal que xi0 =fi0(x0)=fi1(x0)=xi1. Decimos que se ha producido un ciclo, y cuando X es finito todas las ´orbitas terminan en un ciclo, que puede consitir en un ´unico punto fijo por f (f(x)=x).

def orbita(ini,f): / L = [] / while not(ini in L): // L.append(ini) //ini = f(ini) #Actualiza el valor de ini /return L

Ordenacion

Primer algoritmo: InsertSort

Segundo algoritmo: MergeSort

Tercer algoritmo: QuickSort

Cuarto algoritmo: BucketSort

Sage tiene la función **.sort()**

CAPITULO 5: COMPLEMENTOS

5.1 Sistemas de numeracion

Cambios de base (mirar apartado sigiente)

1. Un entero escrito en base b, an an-1 an-2...a1a0 se pasa a base 10 evaluando en base 10 su correspondiente polinomio en la variable b an·b^n+an-1·b^n-1+an-2·b^n-2+···+a1·b+a0.

2. Al reves, si tenemos un entero N en base 10 y lo queremos pasar a base b, es decir, escribirlo en la forma an·b^n+an-1·b^n-1+an-2·b^n-2+···+a1·b+a0.

Debemos, en primer lugar dividir N entre b y el resto es el digito de las unidades a0 en base b. Esto se debe a que podemos escribir an·b^n+···+a1·b+a0=b·(an·b^n-1+···+a1)+a0.

Sistemas de numeracion en Sage

En Sage podemos obtener los dıgitos en una base b de un entero decimal D mediante la instruccion **(D).digits(base=b)**

Las instrucciones **bin(D)**, **oct(D)** y **hex(D)** devuelven cadenas de caracteres con la expresi´on de D en las bases 2, 8 y 16

Para convertir un entero en base b a decimal podemos usar **ZZ('expresion en base b',b)**

5.2 Trucos

Potencias

¿Como calcular de manera eficiente potencias de la forma a^2^k ?

- def potencia(a,k): / if k==0: // return 1 / elif k %2 == 0: // b = potencia(a,k/2) // return (b∗b) / else: // b = potencia(a,(k!1)/2) // return (a∗b∗b)

Sin calcular a^k, calcular el resto (aˆk%m) de la division de a^k entre un entero m

- def potencia mod(a,k,m): / if k==0: // return 1 / elif k%2 == 0: // b = potencia mod(a,k/2,m) // return (b∗b)%m / else: // b = potencia mod(a,(k!1)/2,m) // return (a∗b∗b)%m

- **power_mod(a,n,m)** para calcular el resto de dividir a^n entre m

5.5 Eficiencia

Control de tiempo

Podemos usar la instruccion **time** al comienzo de una lınea de codigo en la que se ejecute una funcion

En segundo lugar tenemos la instruccion timeit, que se ejecuta en la forma **timeit('instruccion')**, y difiere de time en que ejecuta la instruccion varias veces y devuelve un promedio

Cython

cython traduce codigo escrito en Python a C y, una vez que se declaran los tipos de las variables, consigue mejoras importantes en el rendimiento. Para usarlo dentro de Sage basta escribir en la primera lınea de la celda **%cython**

CAPITULO 6: TEORIA DE NUMEROS

6.1 Clases de restos

Fijamos un entero m al que llamamos modulo. En el anillo de los numeros enteros la relacion nRn' si y solo si n y n' tienen el mismo resto al dividir por m → relacion de equivalencia

El conjunto cociente tiene m elementos que corresponden a los m posibles restos

Teorema de Bezout: y dado que el MCD(k,p) = 1 si p es primo y k<p sabemos que existen enteros a y b tales que a·k+b·p=1

Teorema de Fermat-Euler

Si a y n son enteros primos relativos, entonces $a^{\phi(n)} \equiv 1 \pmod{n}$ donde $\phi(n)$ es la función ϕ de Euler

6.2 Fibonacci

La sucesion de Fibonacci es la de numeros enteros definida por $F0=0$, $F1=1$, $Fm=Fm-1+Fm-2$

funcion **fibonacci(m)** que, como es esperable, devuelve el numero de Fibonacci m-esimo

6.3 Algoritmo de Euclides

El algoritmo de Euclides es el que utilizamos para calcular el MCD

MCM

Se calcular haciendo Euclides ya que p*q=MCD*mcm

MCD

- **gcd(a,b)** devuelve el MCD entre a y b
- **xgcd(a,b)** devuelve (d,u,v) donde d=MCD y u,v → d=u*a+v*b

6.4 Numeros primos

- **.is_prime()**
- **.next_prime()**
- **Prime_range(n1,n2)** lista de primos en el intervalo [N1,N2)
- **Nth_prime(m)**
- **Primes(n1,n2)** iterador sobre lista de primos. Sirve para definir un bucle que itere sobre enteros primos en el rango [N1,N2) utilizando poca RAM

6.5 Enteros representables como suma de cuadrados

6.6 Desarrollos decimales

- **Floor(m)** suelo de un numero
- **Ceil(m)** techo de un numero