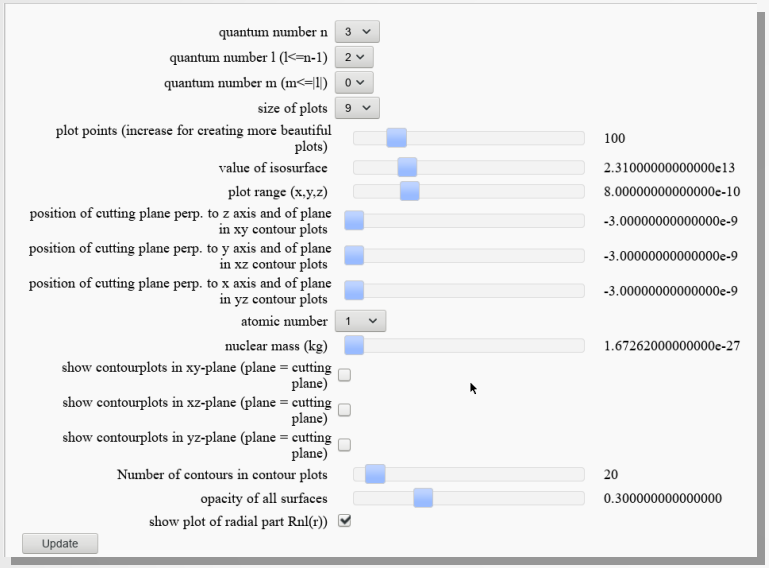
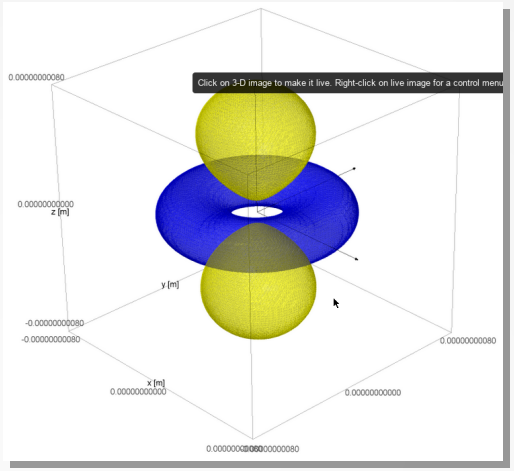
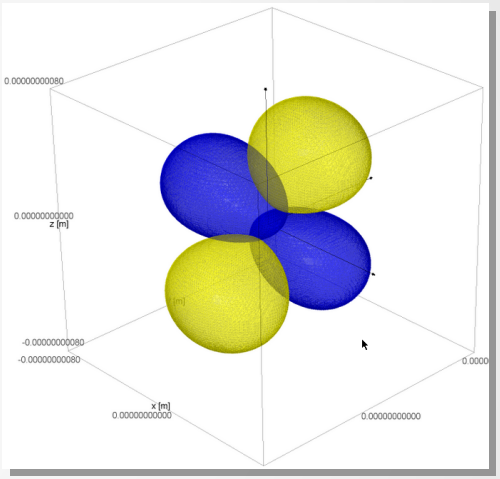
Exploración de ecuaciones con Sage







Dr. Markus Doerr

Grupo de Bioquímica Teórica
Universidad Industrial de Santander
Bucaramanga

http://ciencias.uis.edu.co/grupobioquimicateorica/

Contenido

- Instalación de Sage
- ¿Qué podemos hacer con Sage?
  - Sage como calculadora
  - Funciones
  - Integración, diferenciación
  - Ecuaciones diferenciales
- Visualización
- Visualización interactiva
- Ejercicios / tareas

Material del curso

http://ciencias.uis.edu.co/grupobioquimicateorica/clases/

→ Lectures/Clases → Software → Taller

Sage - instalación y arranque

Instalación

http://ciencias.uis.edu.co/grupobioquimicateorica/clases/sagemath/

Iniciar Sage y el notebook

```
/home/markus> sage
 SageMath version 7.5.1, Release Date: 2017-01-15
 Type "notebook()" for the browser-based notebook interface.
 Type "help()" for help.
sage:
```

```
sage: notebook()
```

Alternativa:

```
sage: notebook("/home/markus/temp/SAGE_NOTEBOOKS.sagenb")
```

Definir una contraseña

Sage - instalación y arranque

El notebook se abre en un browser.

Acceso:
http://localhost:8080/home/admin/
username: admin

nuevo worksheet

importar worksheet de un archivo

Uso de Sage - calculadora

Más información: http://www.sagemath.org/help.html

```
a=5;
a
```

evaluate

5

ejecutar:
pulsar este botón
o
tecla shift + enter

```
3*2
```
6

```
3+2
```
5

```
3^2
```
9

```
9^(1/2)
```
3

```
sqrt(9)
```
3

Más útil en la práctica:

```
### Define some natural constants
c = 2.99792558*10^8; # Speed of light c [m s-1]
h = 6.62608*10^(-34); # Planck's constant [J s]
NA = 6.02214*10^23; # Avogadro's constant [mol-1]
# now calculate the energy of one mol of photons with wavelength 400nm
lamda = 400*10^-9;
Ephoton = (NA * h * c/lamda)/1000;
Ephoton.n(digits=6)
```

```
299.067
```

Uso de Sage - calculadora

Valores exactos y aproximaciones numéricos

```
pi
```
```
pi
```

```
pi.n(digits=50)
```
```
3.1415926535897932384626433832795028841971693993751
```

```
e
```
```
e
```

```
e.n(digits=50)
```
```
2.7182818284590452353602874713526624977572470937000
```

```
e^(i*pi)
```
```
-1
```

No siempre funciona como uno espera:

```
sin(x)^2 + cos(x)^2
```
```
cos(x)^2 + sin(x)^2
```

(no simplificación)

pero:

```
(sin(x)^2 + cos(x)^2).simplify_full()
```
```
1
```

Uso de Sage - funciones

Tres maneras de definir una función

1. Función de python

importante:
sangría

```
def f(x):
    a=2
    b=3
    return a*x^2+b
f(3)
```

```
21
```

Se puede
- trazar

No se puede
- diferenciar
- integrar

2. Expresión simbólica

```
f(x) = 2*x^2+3
f(3)
```

```
21
```

Se puede
- trazar
- diferenciar
- integrar

```
f(x) = 2*x^2+3
f.derivative()
```

```
x |--> 4*x
```

```
f(x) = 2*x^2+3
f.integrate()
```

```
x |--> 2/3*x^3 + 3*x
```

3. Funciones predefinidas

```
f(x)=sin(x)
f(x)
```

```
sin(x)
```

Se puede
- trazar
- diferenciar
- integrar

```
f(x)=sin(x)
f(x).derivative(x,2)
```

```
-sin(x)
```

```
f(x)=sin(x)
f(x).integrate()
```

```
-cos(x)
```

Uso de Sage - programación en Python

Loops

for ...

```
def function(n):
    sum=0
    k=1
    for k in range(1,n+1):
        print "k =",k
        sum = sum + k
    print
    return sum

result=function(5)
print "sum = ",result
```

```
    k = 1
    k = 2
    k = 3
    k = 4
    k = 5

    sum =  15
```

while ...

```
def function(n):
    sum=0
    k=1
    while k <= n:
        print "k =",k
        sum = sum + k
        k += 1
    print
    return sum

result=function(5)
print "sum = ",result
```

```
    k = 1
    k = 2
    k = 3
    k = 4
    k = 5

    sum =  15
```

Uso de Sage - programación en Python

Condiciones:

```
def function(n):
    if n %2 == 0:
       print "n es par"
    else:
       print "n es impar"
    return
function(5)
function(4)
```

```
    n es impar
    n es par
```

Uso de Sage - álgebra y cálculo

http://doc.sagemath.org/html/en/reference/calculus/index.html

Diferenciación

```
f(theta)=sin(theta)
f.diff(theta,1)
```

```
theta |--> cos(theta)
```

```
f(theta)=sin(theta)
f.diff(theta,2)
```

```
theta |--> -sin(theta)
```

```
var ('a,b,c')
f(x) = a * x^2 + b*x + c
f.diff(x)
```

```
x |--> 2*a*x + b
```

Sage supone que x es una variable, pero no sabe que es a,b,c. Hay que decir a Sage que son variables

```
f(x,y) = 2*x^2  + 3*x*y + 4*y^2
print f.diff(x)
print f.diff(y)
```

```
(x, y) |--> 4*x + 3*y
(x, y) |--> 3*x + 8*y
```

```
f(x,y) = 3 * x^3 * y + 4 * x^2 * y^2
print f.diff(x,2,y,1)
```

```
(x, y) |--> 18*x + 16*y
```

Uso de Sage - álgebra y cálculo

Integración

```
var ('a,b,c')
f(x) = a * x^2 + b*x + c
f.integral(x)
```

```
x |--> 1/3*a*x^3 + 1/2*b*x^2 + c*x
```

```
f(theta) = sin(theta)
f.integral(theta)
```

```
theta |--> -cos(theta)
```

```
f(theta) = sin(theta)
f.integral(theta,0,pi)
```

```
2
```

```
f(x)=e^-x^2
f.integral(x)
```

```
x |--> 1/2*sqrt(pi)*erf(x)
```

```
f(x)=e^-x^2
f.integral(x,-oo,oo)
```

```
sqrt(pi)
```

Uso de Sage - álgebra y cálculo

Solución de ecuaciones

```
var('x,y')
solve([x+y==6,x-y==4],x,y)
```

```
[[x == 5, y == 1]]
```

otra notación (útil cuando hay muchos ecuaciones):

```
var('x,y')
eq1 = x+y == 6
eq2 = x-y == 4
solve([eq1,eq2],x,y)
```

```
[[x == 5, y == 1]]
```

Uso de Sage - álgebra y cálculo

Aproximación de funciones - serie Taylor

(e impresión bonita (pretty_print) de ecuaciones)

```
f(x) = sin(x)
print f.taylor(x,0,1)
print f.taylor(x,0,3)
print f.taylor(x,0,5)
```

```
x |--> x
x |--> -1/6*x^3 + x
x |--> 1/120*x^5 - 1/6*x^3 + x
```

Expansión de Taylor alrededor de x=0
con términos hasta orden 1, 2, 3

Más bonito:

```
f(x) = sin(x)
for k in range(1,6,2):
   te(x) = f.taylor(x,0,k)
   pretty_print("max. order:",k)
   pretty_print(LatexExpr("g(x)="),te(x))
```

max. order:1

$$g(x) = x$$

max. order:3

$$g(x) = -\frac{1}{6}\,x^3 + x$$

max. order:5

$$g(x) = \frac{1}{120}\,x^5 - \frac{1}{6}\,x^3 + x$$

Uso de Sage - álgebra y cálculo

Solución de ecuaciones diferenciales

http://doc.sagemath.org/html/en/reference/calculus/sage/calculus/desolvers.html

(aquí: sólo ecuaciones diferenciales lineales)

Ejemplo: descomposición del compuesto A con constante de velocidad k: $A \rightarrow P$

Cambio de la concentración [A] del compuesto A: $\frac{d[A]}{dt} = -k[A]$

Solución general:

```
# A simple first order reaction A -> P
var('t,k,A0') # tell sage that t, k and A0 are variables
A = function('A')(t) # tell sage that A (the concentration) is a function that depends on t
de = diff(A,t) == -k*A # define our differential equation
# general solution (independent variable: t)
desolve(de,A,ivar=t)
```

```
_C*e^(-k*t)
```

Solución con condiciones iniciales:

```
# A simple first order reaction A -> P
var('t,k,A0') # tell sage that t, k and A0 are variables
A = function('A')(t) # tell sage that A (the concentration) is a function that depends on t
de = diff(A,t) == -k*A # define our differential equation
# solution with initial condutions: A = A0 at t=0
desolve(de,A,ivar=t,ics=[0,A0])
```

```
A0*e^(-k*t)
```

Uso de Sage - álgebra y cálculo

Sistemas de ecuaciones diferenciales

Reacciones consecutivas $A \xrightarrow{k_a} B \xrightarrow{k_b} C$

$$\frac{d[A]}{dt} = -k_a[A] \; ; \; \frac{d[B]}{dt} = k_a[A] - k_b[B] \; ; \; \frac{d[C]}{dt} = k_b[B]$$

```
# Consecutive reactions
# A -ka-> B -kb-> C
var('t')
var('k_a,k_b,A_0,B_0,C_0')
# declare A,B,C as functions
A = function('A')(t)
B = function('B')(t)
C = function('C')(t)
# define the differential equations
de1 = diff(A,t) == -k_a * A
de2 = diff(B,t) == k_a * A - k_b * B
de3 = diff(C,t) == k_b * B
# general solution:
gensol = desolve_system([de1,de2,de3],[A,B,C],ivar=t)
# To access the solutions one by one:
pretty_print(gensol[0])
pretty_print(gensol[1])
pretty_print(gensol[2])
```

$$A(t) = A(0)\, e^{(-k_a t)}$$

$$B(t) = -\frac{k_a A(0)\, e^{(-k_a t)}}{k_a - k_b} + \frac{k_a A(0)\, e^{(-k_b t)}}{k_a - k_b}$$

$$C(t) = \frac{k_b A(0)\, e^{(-k_a t)}}{k_a - k_b} - \frac{k_a A(0)\, e^{(-k_b t)}}{k_a - k_b} + A(0)$$

```
# Consecutive reactions
# A -ka-> B -kb-> C
var('t')
var('k_a,k_b,A_0,B_0,C_0')
# declare A,B,C as functions
A = function('A')(t)
B = function('B')(t)
C = function('C')(t)
# define the differential equations
de1 = diff(A,t) == -k_a * A
de2 = diff(B,t) == k_a * A - k_b * B
de3 = diff(C,t) == k_b * B
# solution with initial conditions
sol =  desolve_system([de1,de2,de3],[A,B,C],ivar=t,ics=[0,A_0,0,0])
# To access the solutions one by one:
pretty_print(sol[0])
pretty_print(sol[1])
pretty_print(sol[2])
```

$$A(t) = A_0 e^{(-k_a t)}$$

$$B(t) = -\frac{A_0 k_a e^{(-k_a t)}}{k_a - k_b} + \frac{A_0 k_a e^{(-k_b t)}}{k_a - k_b}$$

$$C(t) = \frac{A_0 k_b e^{(-k_a t)}}{k_a - k_b} - \frac{A_0 k_a e^{(-k_b t)}}{k_a - k_b} + A_0$$

Uso de Sage - álgebra y cálculo

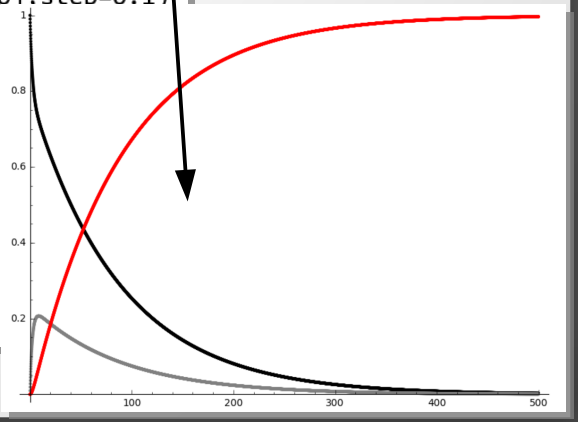
Ecuaciones diferenciales, solución *numérica*

$$A \underset{k_{ap}}{\overset{k_a}{\rightleftarrows}} B \xrightarrow{k_b} C \qquad \frac{d[A]}{dt} = -k_a[A]+k_{ap}[B] \; ; \; \frac{d[B]}{dt} = k_a[A]-k_{ap}[B]-k_b[B] \; ; \; \frac{d[C]}{dt} = k_b[B]$$

```
# Pre-equilibria. Numerical solution
# A -k_a -> B -k_b-> C
#  <-k_ap-
var('t,A,B,C')
var('k_a,k_ap,k_b,A_0,B_0,C_0')
# Define the initial conditions
A_0 = 1
B_0 = 0
C_0 = 0
k_a = 0.1
k_ap = 0.3
k_b = 0.05
# Our differential equations (actually we only need the right-hand-side, which we also define here)
de1 = diff(A,t) == -k_a * A + k_ap * B ; de1rhs = de1.rhs()
de2 = diff(B,t) == k_a * A + -k_ap * B - k_b * B ; de2rhs = de2.rhs()
de3 = diff(C,t) == k_b * B ; de3rhs = de3.rhs()
# Solve the system of equations. This gives a list of points. Each point: [t,A,B,C]
sol=desolve_system_rk4([de1rhs,de2rhs,de3rhs],[A,B,C],ics=[0,A_0,B_0,C_0],ivar=t,end_points=[0,500],step=0.1)
# Creats lists concentration vs t for each concentration
listA=[ [i,j] for i,j,k,l in sol]
listB=[ [i,k] for i,j,k,l in sol]
listC=[ [i,l] for i,j,k,l in sol]
# create plots for each concentration
plota=list_plot(listA,color='black')
plotb=list_plot(listB,color='grey')
plotc=list_plot(listC,color='red')
#show the three graphs, all in one plot
show(plota+plotb+plotc)
```

solución numérica

visualización de las soluciones



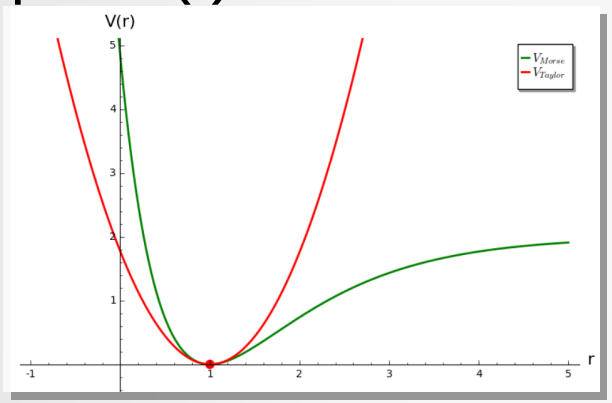
Sage - visualización de funciones

Muchos tipos de gráficas

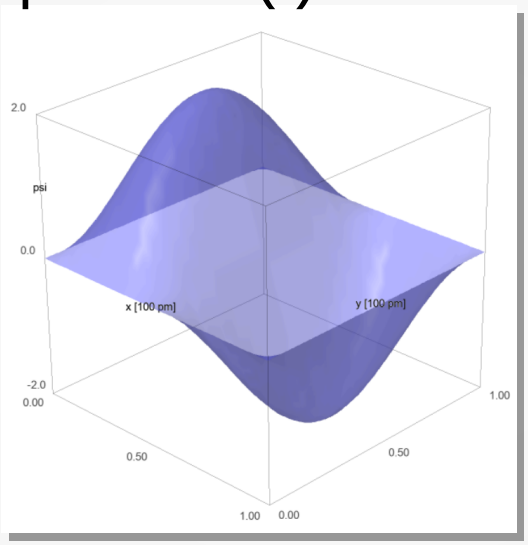
http://doc.sagemath.org/html/en/reference/plotting/sage/plot/plot.html

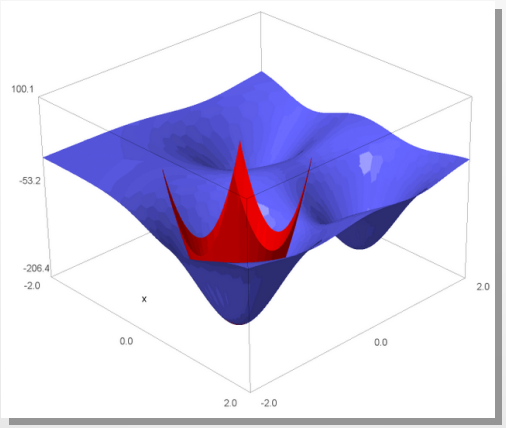
http://doc.sagemath.org/html/en/reference/plot3d/index.html

`plot()`

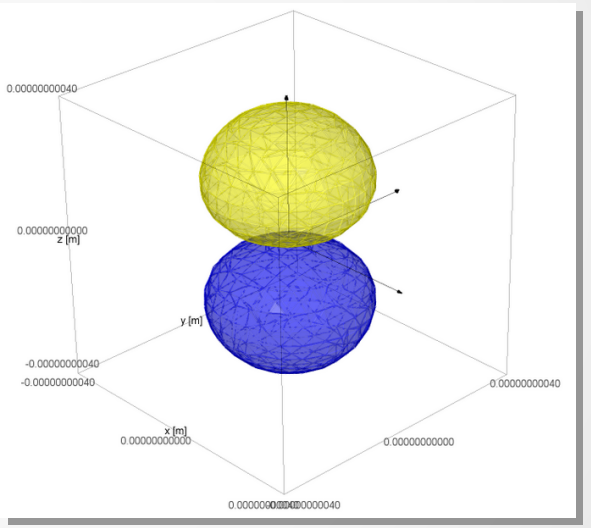


`plot3d()`





`implicit_plot3d()`

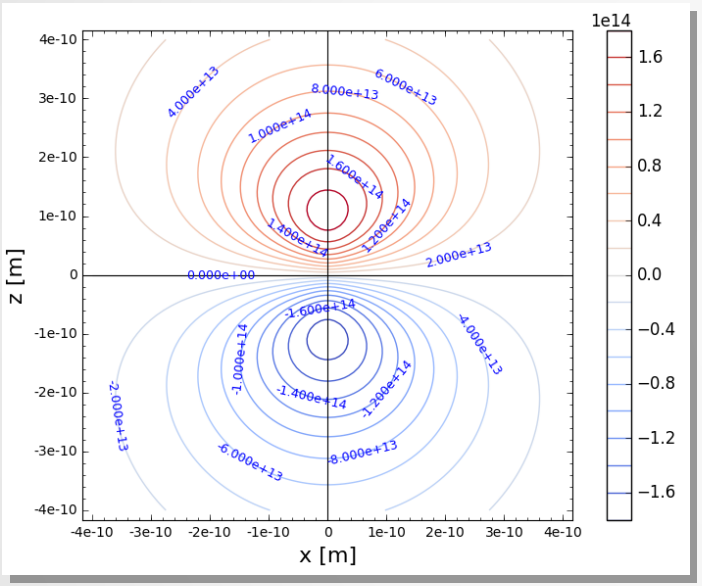


$$2p_z = \frac{1}{4(2\pi)^{1/2}}\left(\frac{Z}{a}\right)^{5/2} re^{-Zr/2a}\cos\theta$$

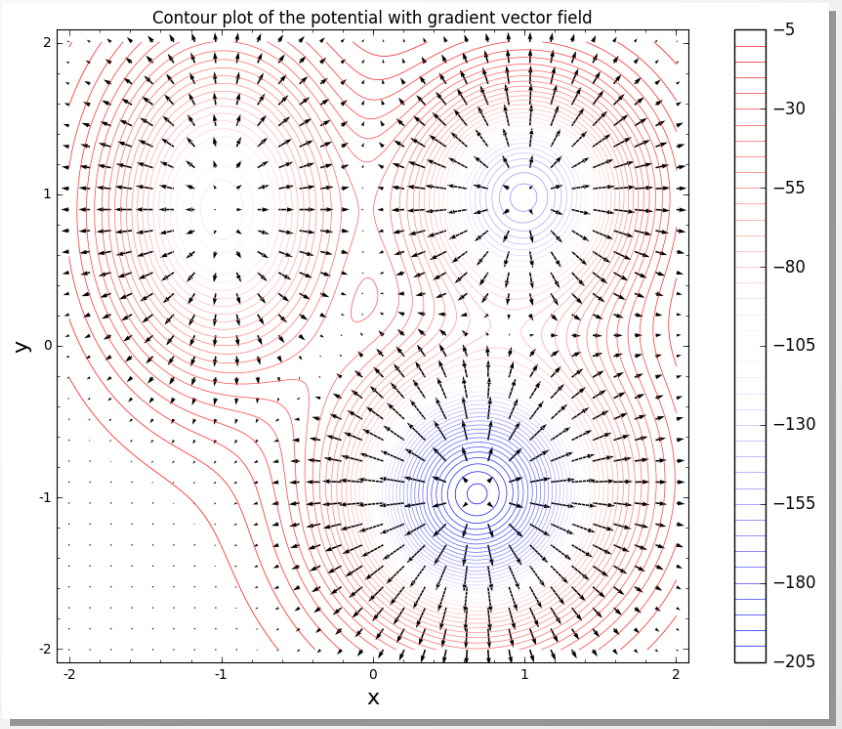
Sage - visualización de funciones

Más ejemplos

`contour_plot()`



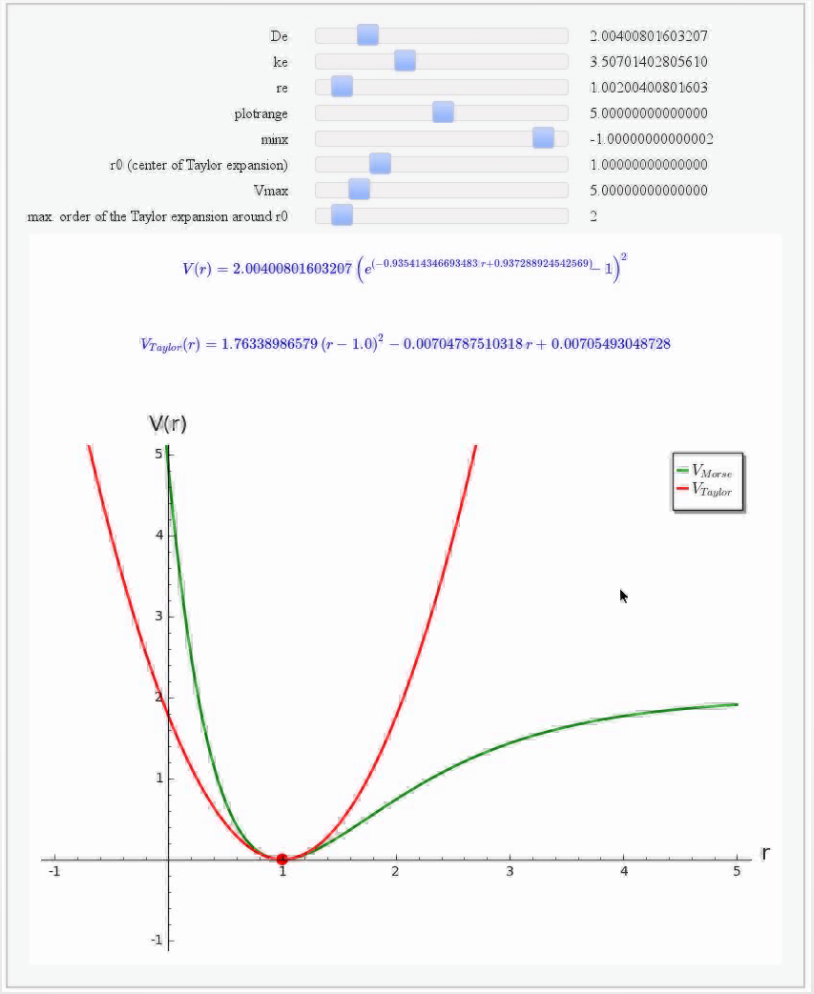
`plot_vector_field()`
`(+ contour_plot())`



Sage - visualización de funciones

Aquí: sólo plot()

Pero: interactivo



Sage - interacción

http://doc.sagemath.org/html/en/prep/Quickstarts/Interact.html

http://doc.sagemath.org/html/en/reference/notebook/sagenb/notebook/interact.html

https://wiki.sagemath.org/interact

Uso

Palabra clave

Función de Python (el nombre es arbitrario)

```
@interact
def interactfunction(
# Here we write the variables whose values are controlled interactively
variable1 = slider(0.0,1.0,step_size=0.1,default=0.5,label="variable uno")
):
    #function body
    print variable1
```

variable uno 0.500000000000000

```
0.500000000000000
```

El valor de la variable variable1
se puede cambiar interactivamente (aquí por un slider)

Aquí definimos que hacemos interactivamente

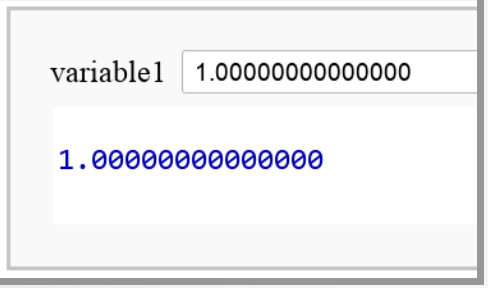
Sage - interacción

Elementos de control

http://doc.sagemath.org/html/en/reference/notebook/sagenb/notebook/interact.html#sagenb.notebook.interact.interact

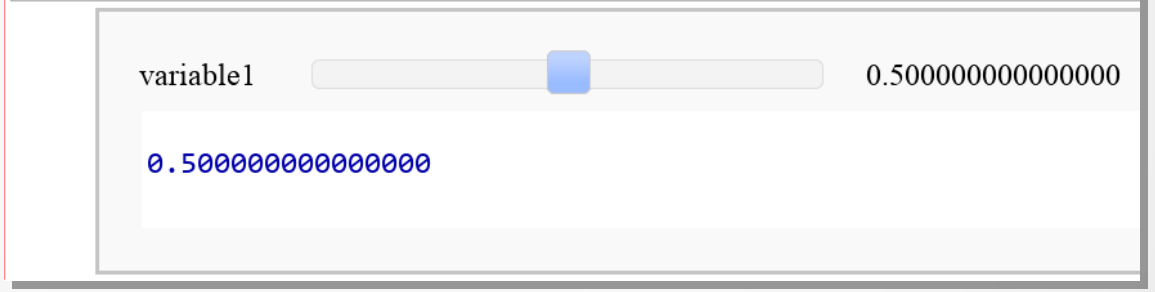
input_box()

```
@interact
def interactfunction(
variable1 = input_box(1.0)
):
    print variable1
```



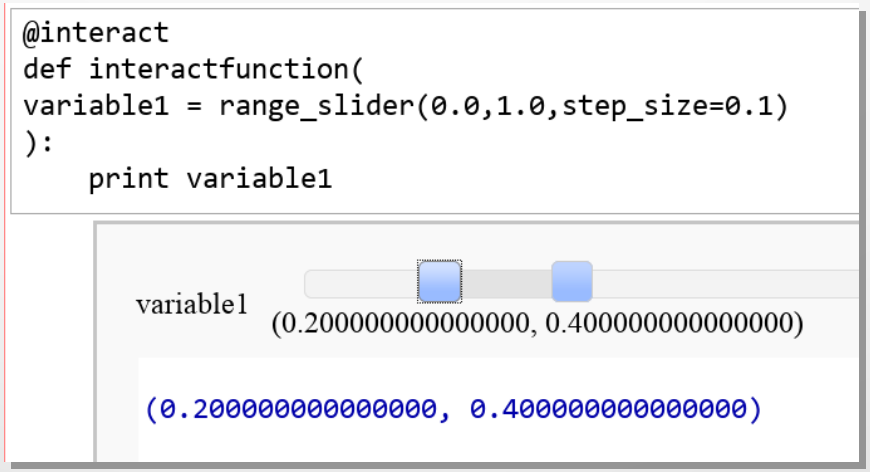
slider()

```
@interact
def interactfunction(
variable1 = slider(0.0,1.0,default=0.5,step_size=0.1)
):
    print variable1
```



range_slider()

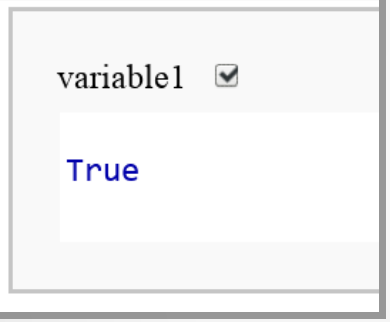
```
@interact
def interactfunction(
variable1 = range_slider(0.0,1.0,step_size=0.1)
):
    print variable1
```



Sage - interacción

checkbox()

```
@interact
def interactfunction(
variable1 = checkbox()
):
    print variable1
```

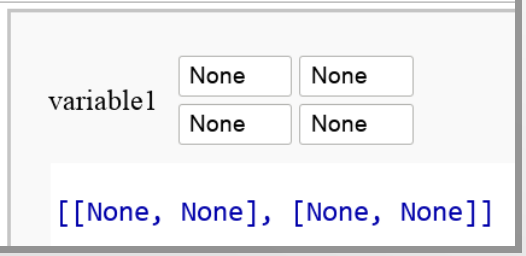


selector()

```
@interact
def interactfunction(
variable1 = selector(['a','b','c'])
):
    print variable1
```

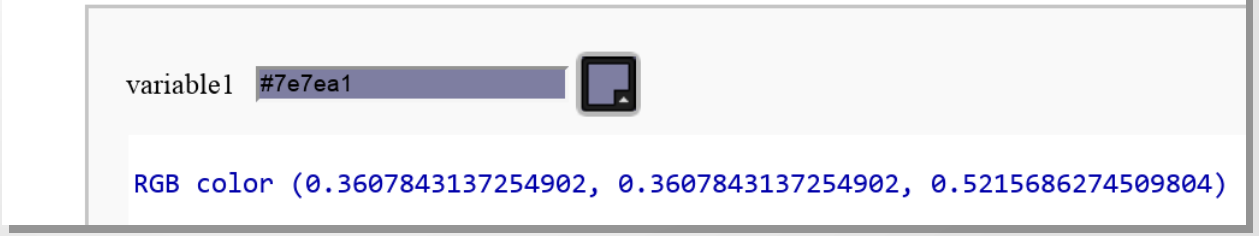
input_grid()

```
@interact
def interactfunction(
variable1 = input_grid(2,2)
):
    print variable1
```



color_selector()

```
@interact
def interactfunction(
variable1 = color_selector()
):
    print variable1
```



Sage - interacción

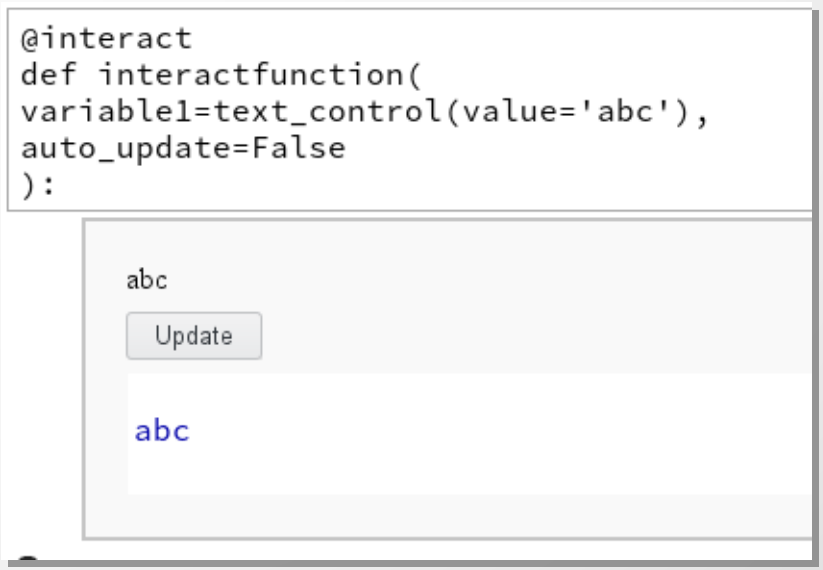
`text_control()`

```
@interact
def interactfunction(
variable1=text_control(value='abc')
):
    print variable1
```

`auto_update=True`

`auto_update=False`

```
@interact
def interactfunction(
variable1=text_control(value='abc'),
auto_update=False
):
```



Sage - interacción

Ejemplo 1:
Aproximación de la energía potencial de una molécula bimolecular

La energía potencial V de una molécula diatómica depende de la distancia r entre los núcleos. Una expresión común para V(r) es el potencial Morse:

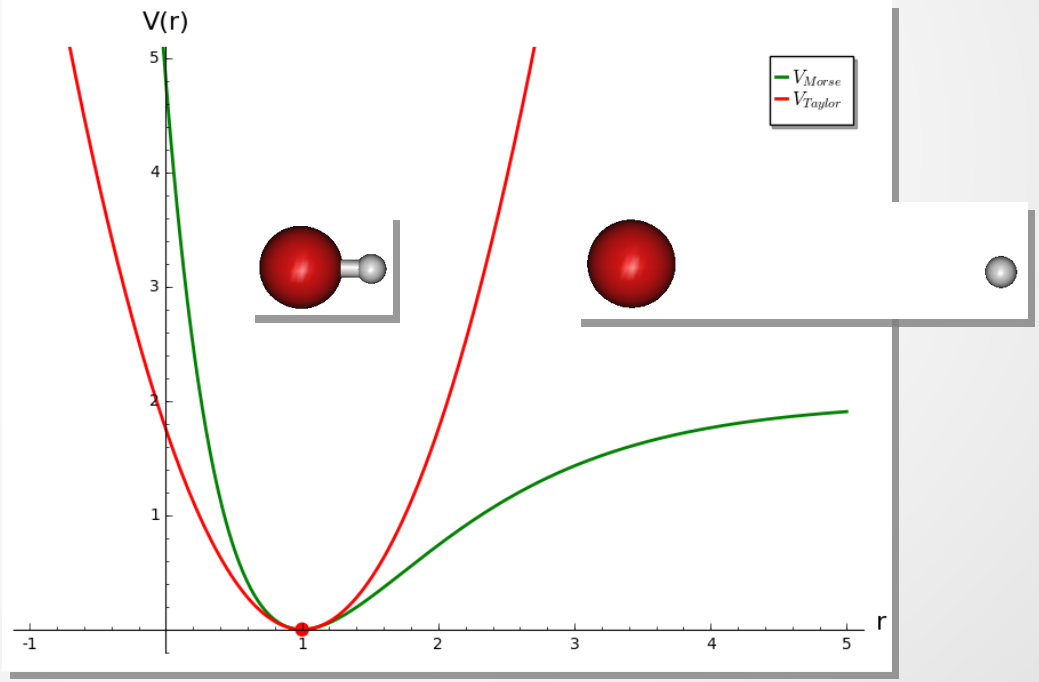
$$V(r) = D_e\left(1-e^{-a(r-r_e)}\right)^2$$

$$a=\sqrt{\frac{k_e}{2D_e}}$$

r_e: distancia con energía mínima

k_e: constante de fuerza del enlace

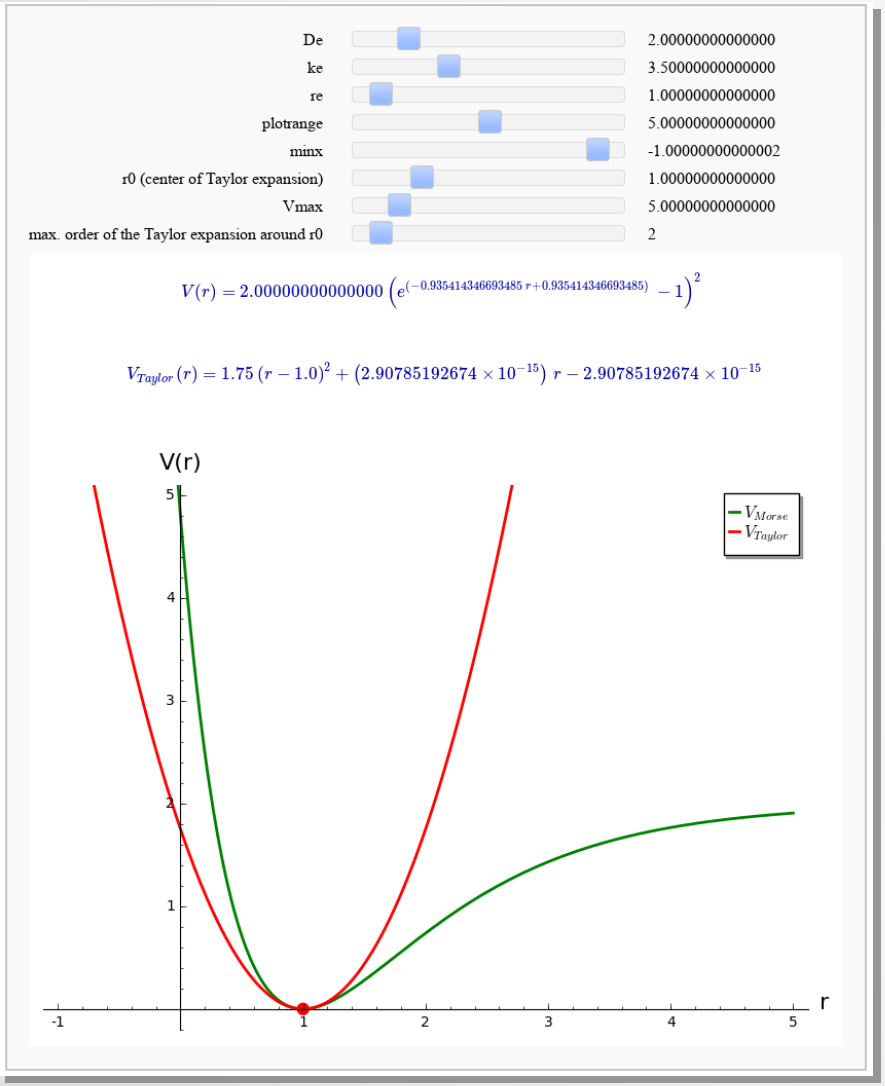
D_e: energía de disociación



Este potencial podemos aproximar por una serie de Taylor.
Alrededor del mínimo podemos aproximar el potencial bien, si incluimos términos hasta orden 2. Para aproximar el potencial a distancias más grandes debemos incluir más términos.

Sage - interacción

Tarea 1



(un ejemplo)

Programar en Sage una visualización interactiva del potencial Morse
y de la aproximación por una serie de Taylor

Experimentar con varios elementos de control

Sage - interacción

Ejemplo 2:
Cinética de reacciones complejas - solución exacta (numérica) y soluciones aproximadas

$$A+B \underset{k_{ap}}{\overset{k_a}{\rightleftarrows}} I \xrightarrow{k_b} P$$

Ecuaciones diferenciales

$$\frac{d[A]}{dt} = \frac{d[B]}{dt} = -k_a[A][B] + k_{ap}[I]$$

$$\frac{d[I]}{dt} = k_a[A][B] - k_{ap}[I] - k_b[I]$$

$$\frac{d[P]}{dt} = k_b[I]$$

Problema: encontrar una ecuación para la velocidad de la reacción total: $A+B \longrightarrow P$

Dos tipos de soluciones aproximadas:

(A): Equilibrio entre A,B y I

$$\frac{d[P]}{dt} = \frac{k_a k_b}{k_{ap}}[A][B]$$

(B): Estado estacionario ([I] es constante)

$$\frac{d[P]}{dt} = \frac{k_a k_b}{k_{ap}+k_b}[A][B]$$

Sage - interacción

$$A + B \underset{k_{ap}}{\overset{k_a}{\rightleftarrows}} I \xrightarrow{k_b} P$$

Soluciones analíticas aproximadas

(A): Equilibrio entre A,B y I

$$\frac{d[P]}{dt} = \underbrace{\frac{k_a k_b}{k_{ap}}}_{k_{ef}}[A][B]$$

(B): Estado estacionario ([I] es constante)

$$\frac{d[P]}{dt} = \underbrace{\frac{k_a k_b}{k_{ap}+k_b}}_{k_{ef}}[A][B]$$

Solución analítica (condiciones iniciales: $[A](t=0) = A_0$, $[B](t=0) = B_0$, $[P](t=0) = P_0$) :

$$P(t) = P_0 + \frac{A_0 B_0 e^{(A_0 k_{ef} t - B_0 k_{ef} t)} - A_0 B_0}{A_0 e^{(A_0 k_{ef} t - B_0 k_{ef} t)} - B_0}$$

Condición: $A_0 <> B_0$

notación en Sage (para copiar/pegar):

```
P(t) = P0 + (A0*B0*e^(A0*kef*t - B0*kef*t) - A0*B0)
/(A0*e^(A0*kef*t - B0*kef*t) - B0)
```

¿Cuándo usar aproximación A, cuándo B?

Depende de los valores de k_a, k_{ap} k_b.

→ Exploración gráfica: trazar P(t) usando las aproximaciones (A) y (B) y comparar con la solución numérica para varios valores de k_a, k_{ap} y k_b

Sage - interacción

Tarea 2

Programar una visualización interactiva de este problema.

Parámetros: A_0, B_0, I_0, P_0, k_a, k_{ap}, k_b

Funciones de trazar:

A(t), B(t), I(t), P(t) para la solución numérica

P(t) para las dos soluciones aproximadas