22-CAVAN-calculo-s1

November 4, 2017

Consulta avanzada
Límites de funciones o de sucesiones
El método o función limit.

```
In [1]: f1(x)=sin(x)/x
        f2(x)=sin(x)/x^2
        f1(x).limit(x=0), f2(x).limit(x=0), f2(x).limit(x=0,dir='-'), f2(x).limit(x=0,dir='rig
```

```
Out[1]: (1, Infinity, -Infinity, +Infinity)
```

```
In [2]: f3=x*sin(x)
        f4=x*(5+sin(x))
        limit(f3,x=oo), limit(f3,x=+infinity), f4.limit(x=+Infinity)
```

```
Out[2]: (und, und, und)
```

```
In [3]: var('n')
        a(n)=sqrt(n^2+3)/n^(3/2)
        limit(a(n),n=+oo)
```

```
Out[3]: 0
```

Suma (simbólica) de series
Utilizamos la función sum con la sintaxis
sum(expresion, indice, valor inicial, valor final)
en la que el valor final puede ser +Infinity (o el valor inicial -Infinity).
El resultado puede ser simbólico, como el que sigue:

```
In [4]: var('k')
        sum(1/k^2, k, 1, +oo)
```

```
Out[4]: 1/6*pi^2
```

```
In [5]: sum(1/k^3, k, 1, +oo)
```

```
Out[5]: zeta(3)
```

```
In [6]: sum(1/(k^2+k),k,1,100)
```

```
Out[6]: 100/101
```

Restricciones sobre los parámetros: la función assume. Se utiliza forget sin argumentos para borrar cualquier hipótesis anterior.

```
In [7]: var('a n')
        sum(a^n, n, 1, +oo)
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-7-7b699b682c46> in <module>()
          1 var('a n')
    ----> 2 sum(a**n, n, Integer(1), +oo)


        /usr/lib/sagemath/local/lib/python2.7/site-packages/sage/misc/functional.pyc in symboli
        562     """
        563     if hasattr(expression, 'sum'):
    --> 564         return expression.sum(*args, **kwds)
        565     elif len(args) <= 1:
        566         return sum(expression, *args)


        sage/symbolic/expression.pyx in sage.symbolic.expression.Expression.sum (/usr/lib/sagem


        /usr/lib/sagemath/local/lib/python2.7/site-packages/sage/calculus/calculus.pyc in symbo
        582
        583     if algorithm == 'maxima':
    --> 584         return maxima.sr_sum(expression,v,a,b)
        585
        586     elif algorithm == 'mathematica':


        /usr/lib/sagemath/local/lib/python2.7/site-packages/sage/interfaces/maxima_lib.py in s
        892                 raise ValueError("Sum is divergent.")
        893             elif "Is" in s: # Maxima asked for a condition
    --> 894                 self._missing_assumption(s)
        895             else:
        896                 raise


        /usr/lib/sagemath/local/lib/python2.7/site-packages/sage/interfaces/maxima_lib.py in _m
       1015              + errstr[jj+1:k] +">0)', see `assume?` for more details)\n" + errstr
       1016         outstr = outstr.replace('_SAGE_VAR_','')
    -> 1017         raise ValueError(outstr)
       1018
```

```
1019 def is_MaximaLibElement(x):


ValueError: Computation failed since Maxima requested additional constraints; using the
Is abs(a)-1 positive, negative or zero?
```

```
In [8]: var('a n')
        forget()
        assume(abs(a)<1)
        sum(a^n, n, 1, +oo)
```

```
Out[8]: -a/(a - 1)
```

```
In [9]: var('a n')
        sum(a^n, n, 1, +oo)
```

```
Out[9]: -a/(a - 1)
```

Las dos celdas que siguen son esencialmente equivalentes: tardan lo mismo y producen el mismo resultado. No siempre es así, y, en ocasiones, la segunda puede tardar menos en ejecutarse.

```
In [12]: %time
         var('k')
         Suma1=sum(1/k,k,1,10000)
         sum1 = Suma1.n(16)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.01 ţs
```

```
In [13]: %time
         Suma2=0
         for j in range(1,10001):
             Suma2+=1/j
         sum2 = Suma2.n(16)
```

```
CPU times: user 0 ns, sys: 0 ns, total: 0 ns
Wall time: 5.01 ţs
```

```
In [14]: sum1 == sum2
```

```
Out[14]: True
```

Derivadas

Utilizamos indistintamente differentiate, diff, derivative. Las tres funcionan como métodos, las dos últimas también se pueden utilizar como funciones.

```
In [15]: fsen(x)=sin(x)
```

```
In [16]: fsen(x).derivative(),fsen(x).diff(),fsen(x).differentiate(),derivative(fsen(x)),diff(
```

```
Out[16]: (cos(x), cos(x), cos(x), cos(x), cos(x))
```

```
In [17]: fsen.derivative(),fsen.diff(),fsen.differentiate(),derivative(fsen),diff(fsen)
```

```
Out[17]: (x |--> cos(x), x |--> cos(x), x |--> cos(x), x |--> cos(x), x |--> cos(x))
```

Derivadas de orden superior:

```
In [18]: fsen(x).derivative(0),fsen(x).derivative(),fsen(x).derivative(2),fsen(x).derivative(3
```

```
Out[18]: (sin(x), cos(x), -sin(x), -cos(x), sin(x))
```

Se pueden utilizar con funciones de varias variables para calcular derivadas parciales

```
In [19]: F(x,y)=x^3*sin(2*y)+y^2*cos(3*x)
```

```
In [20]: F(x,y).diff(x),F(x,y).diff(y)
```

```
Out[20]: (-3*y^2*sin(3*x) + 3*x^2*sin(2*y), 2*x^3*cos(2*y) + 2*y*cos(3*x))
```

```
In [21]: F(x,y).diff(x,y),F(x,y).diff(y,x)
```

```
Out[21]: (6*x^2*cos(2*y) - 6*y*sin(3*x), 6*x^2*cos(2*y) - 6*y*sin(3*x))
```

Se puede indicar el número de veces que se deriva respecto de cada variable:

```
In [22]: F(x,y).diff(x,2,y,3)
```

```
Out[22]: -48*x*cos(2*y)
```

Ejercicios
Hallar:

(a) $\dfrac{d}{dx}\sin(\log x)$ (b) $\dfrac{d}{dx}\arcsin\sqrt{x^2-1}$ (c) $\dfrac{d}{dx}\tan(x^2+\log x+\arctan x)$ (d) $\dfrac{d}{dx}x^{\log x}$

Cálculo integral
Para calcular la primitiva de una función definida simbólicamente podemos utilizar los métodos (o funciones) integral o integrate. El resultado es una sola primitiva; no se indica que el resultado es "salvo constante aditiva".

```
In [23]: f1(x)=sin(x)^2
         print f1.integral(x)
         print f1.integrate(x)
         print integrate(f1,x)
         print integral(f1,x)
```

```
x |--> 1/2*x - 1/4*sin(2*x)
x |--> 1/2*x - 1/4*sin(2*x)
x |--> 1/2*x - 1/4*sin(2*x)
x |--> 1/2*x - 1/4*sin(2*x)
```

Observese que efectivamente el programa elige una primitiva, ya que al evaluarla se obtiene un valor numérico.

```
In [24]: f2(x)=(x^(3/5)+x^(1/6))/sqrt(x)
         f2i=f2.integral(x)
         show(f2i)
         f2i(1)
```

```
x |--> 10/11*x^(11/10) + 3/2*x^(2/3)
```

```
Out[24]: 53/22
```

El cálculo de una integral definida se obtiene dando los extremos del intervalo de integración al método (o función), pero nos da el valor simbólico.

```
In [25]: f4(x)=sin(2*x)*cos(5*x)
         f4.integrate(x,0,1)
```

```
Out[25]: -1/14*cos(7) + 1/6*cos(3) - 2/21
```

Para obtener la aproximación numérica, tenemos que pedirla

```
In [26]: f4.integrate(x,0,1).n()
```

```
Out[26]: -0.314087005696025
```

```
In [27]: integral(sin(x)^2,x,0,pi/5), integral(sin(x)^2,x,0,pi/5).n()
```

```
Out[27]: (1/10*pi - 1/16*sqrt(2*sqrt(5) + 10), 0.0763951362851909)
```

Serie de Taylor

```
In [28]: f(x)=(1-cos(x))/x^2
```

```
In [29]: taylor(f,x,0,5)
```

```
Out[29]: x |--> 1/720*x^4 - 1/24*x^2 + 1/2
```

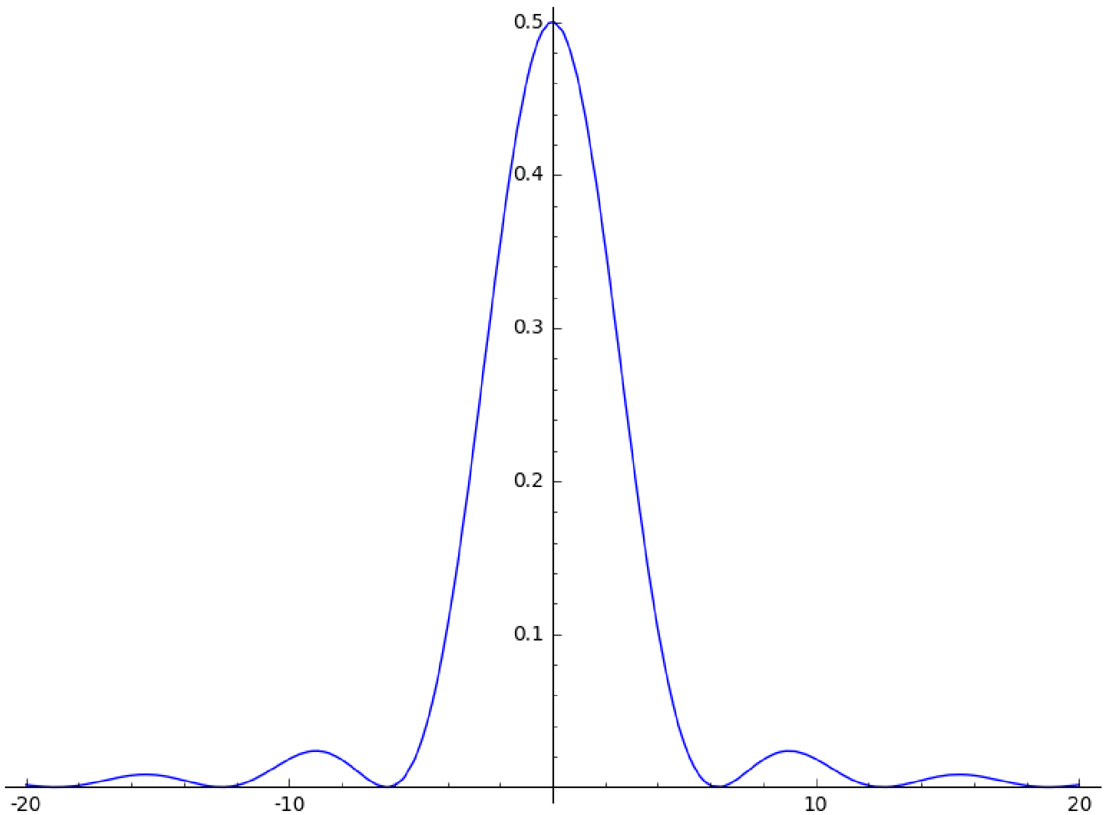
```
In [27]: taylor(f,x,0,10)
```

```
Out[27]: x |--> -1/479001600*x^10 + 1/3628800*x^8 - 1/40320*x^6 + 1/720*x^4 - 1/24*x^2 + 1/2
```

Como la función f es par ($f(x) = f(-x)$ para todo x) en su desarrollo de taylor sólo aparecen exponentes pares. Nos preguntamos si es cierto que $f(x) \leq \frac{1}{2}$ para todo x:

In [30]: plot(f,-20,20)

Out[30]:



Como la función es par sólo debemos ocuparnos del semieje $x \geq 0$. Es claro que la función se anula para $x = \pm 2\pi$ y para $|x| \geq 2\pi$ no hay gran problema porque el numerador está acotado por 1 y el denominador es suficientemente grande. Debemos estudiar la función f en el intervalo $[-2\pi, 2\pi]$ (o $[0, 2\pi]$).

De hecho, para $|x| \geq \sqrt{2}$, por la misma razón, ya es imposible que $f(x)$ sea mayor que $1/2$. Nos queda el intervalo $[0, \sqrt{2}]$), y bastaría, por ejemplo, demostrar que $f(x)$ es decreciente en todo el intervalo viendo que su derivada es negativa en todo el intervalo. £Cómo podemos usar Sage para ayudarnos a hacer esto?

Gráficas

In [31]: T2 = taylor(f,x,0,2); T2

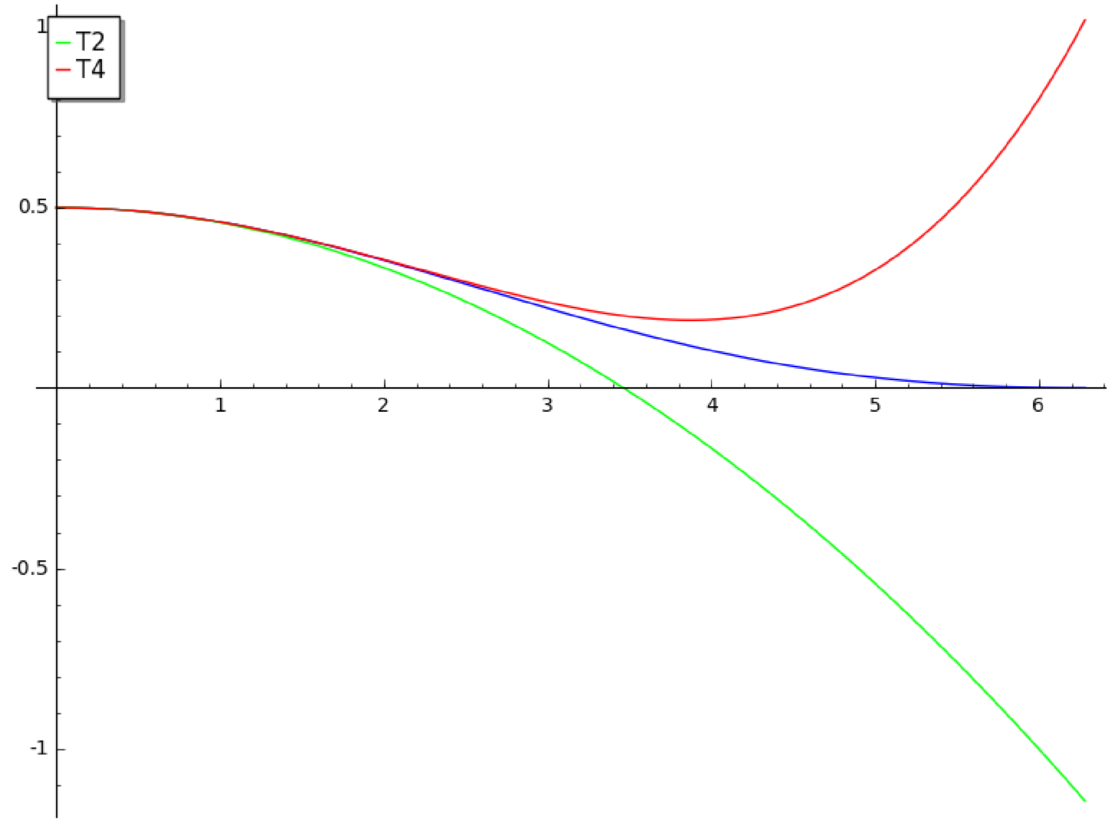
Out[31]: x |--> -1/24*x^2 + 1/2

In [32]: T4 = taylor(f,x,0,4); T4

Out[32]: x |--> 1/720*x^4 - 1/24*x^2 + 1/2

In [33]: plot(f,0,2*pi)+plot(T2,0,2*pi,rgbcolor=(0,1,0),legend_label="T2")+plot(T4,0,2*pi,rgbco

Out[33]:



Vemos claramente que las aproximaciones que da el polinomio de Taylor son locales, buenas cerca del punto en el que derivamos (en este caso $x = 0$) y mejoran al aumentar el grado del polinomio (el polinomio $T4$ aproxima mejor que el $T2$).