EjerciciosAproximacionCabor

February 5, 2018

```
In [1]: #n=N=numerical_approx (53 bits)
        n(pi)
```

```
Out[1]: 3.14159265358979
```

```
In [10]: #Podemos añadir un segundo argumento para especificar la precisión:
         #n(pi,prec=4) para mostrar el número con precisión de 4 bits
         #n(pi,digits=4) para mostrar el número con precisión de 4 dígitos
         a=n(pi, prec=4)
         print a
         print a.str(base=2)
         b=n(pi, digits=4)
         print b
         print b.str(base=2)
```

```
3.2
11.01
3.142
11.001001000100000
```

```
In [10]: #Podemos crear una función que especifique la precisión:
         NR = RealField(prec=1000)
         NR(pi)
```

```
Out[10]: 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
```

Ejercicio 1

```
In [1]: def pn(n):
            a = sin(pi/n)
            return a*2*n

        n(pn(100), digits =10)
```

```
Out[1]: 6.282151816
```

Ejercicio 4

```
In [34]: def raman(n):
             a = (factorial(2*n))^3
             b = 42*n + 5
             c = (factorial(n))^6
             d = 16^(3*n+1)
             return (a*b)/(c*d)

         def suma(n):
             total = 0
             for i in srange(0,n+1):
                 total += raman(i)
             return total

         def aprox(num, dig):
             err=(1/suma(num) - pi)
             err = n(err, digits = dig)
             return err

         def cifrascorr(num):
             a = aprox(num, num*5)
             cifras = 0
             while(1):
                 if (a > 1):
                     break
                 else:
                     cifras = cifras + 1
                     a = a*10
             return cifras

         def final(tope):
             for i in srange(10, tope+1, 10):
                 a = cifrascorr(i)
                 print [i, a, n(a/i, digits = 3)]

         final(2000)
```

```
[10, 20, 2.00]
[20, 38, 1.90]
[30, 57, 1.90]
[40, 75, 1.88]
[50, 93, 1.86]
[60, 111, 1.85]
[70, 129, 1.84]
[80, 147, 1.84]
[90, 165, 1.83]
[100, 183, 1.83]
[110, 201, 1.83]
[120, 219, 1.82]
```

[130, 237, 1.82]
[140, 256, 1.83]
[150, 274, 1.83]
[160, 292, 1.82]
[170, 310, 1.82]
[180, 328, 1.82]
[190, 346, 1.82]
[200, 364, 1.82]
[210, 382, 1.82]
[220, 400, 1.82]
[230, 418, 1.82]
[240, 436, 1.82]
[250, 454, 1.82]
[260, 472, 1.82]
[270, 491, 1.82]
[280, 509, 1.82]
[290, 527, 1.82]
[300, 545, 1.82]
[310, 563, 1.82]
[320, 581, 1.82]
[330, 599, 1.82]
[340, 617, 1.81]
[350, 635, 1.81]
[360, 653, 1.81]
[370, 671, 1.81]
[380, 689, 1.81]
[390, 707, 1.81]
[400, 725, 1.81]
[410, 743, 1.81]
[420, 762, 1.81]
[430, 780, 1.81]
[440, 798, 1.81]
[450, 816, 1.81]
[460, 834, 1.81]
[470, 852, 1.81]
[480, 870, 1.81]
[490, 888, 1.81]
[500, 906, 1.81]
[510, 924, 1.81]
[520, 942, 1.81]
[530, 960, 1.81]
[540, 978, 1.81]
[550, 996, 1.81]
[560, 1014, 1.81]
[570, 1033, 1.81]
[580, 1051, 1.81]
[590, 1069, 1.81]
[600, 1087, 1.81]

[610, 1105, 1.81]
[620, 1123, 1.81]
[630, 1141, 1.81]
[640, 1159, 1.81]
[650, 1177, 1.81]
[660, 1195, 1.81]
[670, 1213, 1.81]
[680, 1231, 1.81]
[690, 1249, 1.81]
[700, 1267, 1.81]
[710, 1285, 1.81]
[720, 1304, 1.81]
[730, 1322, 1.81]
[740, 1340, 1.81]
[750, 1358, 1.81]
[760, 1376, 1.81]
[770, 1394, 1.81]
[780, 1412, 1.81]
[790, 1430, 1.81]
[800, 1448, 1.81]
[810, 1466, 1.81]
[820, 1484, 1.81]
[830, 1502, 1.81]
[840, 1520, 1.81]
[850, 1538, 1.81]
[860, 1556, 1.81]
[870, 1574, 1.81]
[880, 1593, 1.81]
[890, 1611, 1.81]
[900, 1629, 1.81]
[910, 1647, 1.81]
[920, 1665, 1.81]
[930, 1683, 1.81]
[940, 1701, 1.81]
[950, 1719, 1.81]
[960, 1737, 1.81]
[970, 1755, 1.81]
[980, 1773, 1.81]
[990, 1791, 1.81]
[1000, 1809, 1.81]
[1010, 1827, 1.81]
[1020, 1845, 1.81]
[1030, 1864, 1.81]
[1040, 1882, 1.81]
[1050, 1900, 1.81]
[1060, 1918, 1.81]
[1070, 1936, 1.81]
[1080, 1954, 1.81]

```
[1090, 1972, 1.81]
[1100, 1990, 1.81]
[1110, 2008, 1.81]
[1120, 2026, 1.81]
[1130, 2044, 1.81]
[1140, 2062, 1.81]
[1150, 2080, 1.81]
[1160, 2098, 1.81]
[1170, 2116, 1.81]
[1180, 2134, 1.81]
[1190, 2153, 1.81]
[1200, 2171, 1.81]
[1210, 2189, 1.81]
[1220, 2207, 1.81]
[1230, 2225, 1.81]
[1240, 2243, 1.81]
[1250, 2261, 1.81]
[1260, 2279, 1.81]
[1270, 2297, 1.81]
[1280, 2315, 1.81]
[1290, 2333, 1.81]
[1300, 2351, 1.81]
[1310, 2369, 1.81]
[1320, 2387, 1.81]
[1330, 2405, 1.81]
[1340, 2423, 1.81]
[1350, 2442, 1.81]
[1360, 2460, 1.81]
[1370, 2478, 1.81]
[1380, 2496, 1.81]
[1390, 2514, 1.81]
[1400, 2532, 1.81]
[1410, 2550, 1.81]
[1420, 2568, 1.81]
```

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

KeyboardInterrupt                         Traceback (most recent call last)

<ipython-input-34-5fe56a846c5a> in <module>()
 33         print [i, a, n(a/i, digits = Integer(3))]
 34
---> 35 final(Integer(10000))


<ipython-input-34-5fe56a846c5a> in final(tope)
```

```
     30 def final(tope):
     31     for i in srange(Integer(10), tope+Integer(1), Integer(10)):
---> 32         a = cifrascorr(i)
     33         print [i, a, n(a/i, digits = Integer(3))]
     34 


    <ipython-input-34-5fe56a846c5a> in cifrascorr(num)
     18 
     19 def cifrascorr(num):
---> 20     a = aprox(num, num*Integer(6))
     21     cifras = Integer(0)
     22     while(Integer(1)):


    <ipython-input-34-5fe56a846c5a> in aprox(num, dig)
     13 
     14 def aprox(num, dig):
---> 15     err=(Integer(1)/suma(num) - pi)
     16     err = n(err, digits = dig)
     17     return err


    <ipython-input-34-5fe56a846c5a> in suma(n)
      9     total = Integer(0)
     10     for i in srange(Integer(0),n+Integer(1)):
---> 11         total += raman(i)
     12     return total
     13 


    <ipython-input-34-5fe56a846c5a> in raman(n)
      4     c = (factorial(n))**Integer(6)
      5     d = Integer(16)**(Integer(3)*n+Integer(1))
----> 6     return (a*b)/(c*d)
      7 
      8 def suma(n):


    src/cysignals/signals.pyx in cysignals.signals.python_check_interrupt (build/src/cysig


    src/cysignals/signals.pyx in cysignals.signals.sig_raise_exception (build/src/cysignals


    KeyboardInterrupt:
```

Ejercicio 12

```
In [14]: soluciones = solve(4*x^3+x^2+2*x+1, x, solution_dict=True)
         for i in soluciones:
             print n(i[x], digits = 4)
```

```
0.09100 - 0.7552*I
0.09100 + 0.7552*I
-0.4320
```

Ejercicio 8

```
In [14]: def F(n):
             num = (factorial(n))^2*2^(n+1)
             den = factorial(2*n+1)
             return num/den

         def func(ndig):
             k = 0 #Sumandos necesarios
             S = 0
             while 1:
                 S += F(k)
                 if(floor(abs(10^ndig*F(k)))==0):
                     break
                 k += 1
             return n(S,digits = ndig), k+1

         for i in srange(1,200):
             a = func(i)
             print [i,a[1],n(a[1]/i, digits = 3)]
```

```
[1, 5, 5.00]
[2, 8, 4.00]
[3, 11, 3.67]
[4, 14, 3.50]
[5, 17, 3.40]
[6, 20, 3.33]
[7, 23, 3.29]
[8, 27, 3.38]
[9, 30, 3.33]
[10, 33, 3.30]
[11, 36, 3.27]
[12, 40, 3.33]
[13, 43, 3.31]
[14, 46, 3.29]
[15, 49, 3.27]
[16, 53, 3.31]
[17, 56, 3.29]
```

```
[18, 59, 3.28]
[19, 62, 3.26]
[20, 66, 3.30]
[21, 69, 3.29]
[22, 72, 3.27]
[23, 76, 3.30]
[24, 79, 3.29]
[25, 82, 3.28]
[26, 85, 3.27]
[27, 89, 3.30]
[28, 92, 3.29]
[29, 95, 3.28]
[30, 99, 3.30]
[31, 102, 3.29]
[32, 105, 3.28]
[33, 109, 3.30]
[34, 112, 3.29]
[35, 115, 3.29]
[36, 118, 3.28]
[37, 122, 3.30]
[38, 125, 3.29]
[39, 128, 3.28]
[40, 132, 3.30]
[41, 135, 3.29]
[42, 138, 3.29]
[43, 142, 3.30]
[44, 145, 3.30]
[45, 148, 3.29]
[46, 152, 3.30]
[47, 155, 3.30]
[48, 158, 3.29]
[49, 161, 3.29]
[50, 165, 3.30]
[51, 168, 3.29]
[52, 171, 3.29]
[53, 175, 3.30]
[54, 178, 3.30]
[55, 181, 3.29]
[56, 185, 3.30]
[57, 188, 3.30]
[58, 191, 3.29]
[59, 195, 3.31]
[60, 198, 3.30]
[61, 201, 3.30]
[62, 204, 3.29]
[63, 208, 3.30]
[64, 211, 3.30]
[65, 214, 3.29]
```

[66, 218, 3.30]
[67, 221, 3.30]
[68, 224, 3.29]
[69, 228, 3.30]
[70, 231, 3.30]
[71, 234, 3.30]
[72, 238, 3.31]
[73, 241, 3.30]
[74, 244, 3.30]
[75, 247, 3.29]
[76, 251, 3.30]
[77, 254, 3.30]
[78, 257, 3.29]
[79, 261, 3.30]
[80, 264, 3.30]
[81, 267, 3.30]
[82, 271, 3.30]
[83, 274, 3.30]
[84, 277, 3.30]
[85, 281, 3.31]
[86, 284, 3.30]
[87, 287, 3.30]
[88, 291, 3.31]
[89, 294, 3.30]
[90, 297, 3.30]
[91, 301, 3.31]
[92, 304, 3.30]
[93, 307, 3.30]
[94, 310, 3.30]
[95, 314, 3.31]
[96, 317, 3.30]
[97, 320, 3.30]
[98, 324, 3.31]
[99, 327, 3.30]
[100, 330, 3.30]
[101, 334, 3.31]
[102, 337, 3.30]
[103, 340, 3.30]
[104, 344, 3.31]
[105, 347, 3.30]
[106, 350, 3.30]
[107, 354, 3.31]
[108, 357, 3.31]
[109, 360, 3.30]
[110, 363, 3.30]
[111, 367, 3.31]
[112, 370, 3.30]
[113, 373, 3.30]

```
[114, 377, 3.31]
[115, 380, 3.30]
[116, 383, 3.30]
[117, 387, 3.31]
[118, 390, 3.31]
[119, 393, 3.30]
[120, 397, 3.31]
[121, 400, 3.31]
[122, 403, 3.30]
[123, 407, 3.31]
[124, 410, 3.31]
[125, 413, 3.30]
[126, 417, 3.31]
[127, 420, 3.31]
[128, 423, 3.30]
[129, 426, 3.30]
[130, 430, 3.31]
[131, 433, 3.31]
[132, 436, 3.30]
[133, 440, 3.31]
[134, 443, 3.31]
[135, 446, 3.30]
[136, 450, 3.31]
[137, 453, 3.31]
[138, 456, 3.30]
[139, 460, 3.31]
[140, 463, 3.31]
[141, 466, 3.30]
[142, 470, 3.31]
[143, 473, 3.31]
[144, 476, 3.31]
[145, 480, 3.31]
[146, 483, 3.31]
[147, 486, 3.31]
[148, 490, 3.31]
[149, 493, 3.31]
[150, 496, 3.31]
[151, 499, 3.30]
[152, 503, 3.31]
[153, 506, 3.31]
[154, 509, 3.31]
[155, 513, 3.31]
[156, 516, 3.31]
[157, 519, 3.31]
[158, 523, 3.31]
[159, 526, 3.31]
[160, 529, 3.31]
[161, 533, 3.31]
```

```
[162, 536, 3.31]
[163, 539, 3.31]
[164, 543, 3.31]
[165, 546, 3.31]
[166, 549, 3.31]
[167, 553, 3.31]
[168, 556, 3.31]
[169, 559, 3.31]
[170, 562, 3.31]
[171, 566, 3.31]
[172, 569, 3.31]
[173, 572, 3.31]
[174, 576, 3.31]
[175, 579, 3.31]
[176, 582, 3.31]
[177, 586, 3.31]
[178, 589, 3.31]
[179, 592, 3.31]
[180, 596, 3.31]
[181, 599, 3.31]
[182, 602, 3.31]
[183, 606, 3.31]
[184, 609, 3.31]
[185, 612, 3.31]
[186, 616, 3.31]
[187, 619, 3.31]
[188, 622, 3.31]
[189, 626, 3.31]
[190, 629, 3.31]
[191, 632, 3.31]
[192, 635, 3.31]
[193, 639, 3.31]
[194, 642, 3.31]
[195, 645, 3.31]
[196, 649, 3.31]
[197, 652, 3.31]
[198, 655, 3.31]
[199, 659, 3.31]
```

Ejercicio 11

```
In [26]: def frac(n, k):
             i = 0
             L=[]
             while(n>1 and i<=k):
                 L.append(floor(n))
                 if(n == floor(n)):
```

```
                break
            n = (n-L[i])^(-1)
            i += 1
        return L

    def frac2(n, m):
        L=[]
        fr = n/m
        while(fr>1):
            L.append(floor(fr))
            if(fr == floor(fr)):
                break
            fr = (fr-floor(fr))^(-1)
        return L

    def convergentes(n,m):
        L1 = frac2(n,m)
        L2 = []
        L2.append(L1[0])
        L2.append((L1[0]*L1[1]+1)/L1[1])
        p1 = L1[0]*L1[1]+1
        p2 = L1[0]
        q1 = L1[1]
        q2 = 1
        i = 2
        while(i<len(L1)):
            p = L1[i]*p1+p2
            q = L1[i]*q1+q2
            L2.append(p/q)
            i += 1
            p2 = p1
            q2 = q1
            p1 = p
            q1 = q
        return L2
```

```
Out[26]: [5, 21/4, 26/5, 47/9, 120/23]
```

Ejercicio 14

```
In [35]: def dominante(num):
            flag = 0
            for i in srange(1, num+1):
                if((2^i).digits()[0]==9):
                    print i
                    flag=1
            if(flag == 0):
                print "No hay ninguna potencia de 2^n hasta n =",num,"con cifra dominante 9"
        dominante(10000)
```

```
No hay ninguna potencia de 2^n hasta n = 10000 con cifra dominante 9
```

Ejercicio 15

```
In [63]: def subint(f,a,b):
             if(f(a)*f(b)>0):
                 print "No se cumple Bolzano"
                 return
             if(f(a)*f((b+a)/2)<=0):
                 return [a,(b+a)/2]
             if(f(b)*f((b+a)/2)<=0):
                 return [(b+a)/2, b]

         def iterador(f,a,b,E):
             if(b < a):
                 iterador(f,b,a,E)

             L = subint(f,a,b)
             while(abs(L[0]-L[1])>E):
                 L = subint(f,L[0],L[1])
             L[0] = n(L[0],prec = 5)
             L[1] = n(L[1],prec = 5)
             return L

         f(x) = x^3
         iterador(f, -55, 0.653, 0.1)

Out[63]: [-0.053, 0.00082]
```

Ejericicio 16

```
In [36]: def funcion(L,Lf):
             if(len(L) != len(Lf)):
                 print("Error en longitud de listas")
                 return
             den = L[-1]-L[0]
             if(len(L) == 1):
                 return Lf[0]
             num = funcion(L[1:len(L)],Lf[1:len(L)])-funcion(L[0:len(L)-1],Lf[0:len(L)-1])
             return num/den

         def polinomio(L,Lf):
             f(x) = 0
             for i in srange(1,len(L)+1):
                 g(x) = funcion(L[0:i],Lf[0:i])
                 for j in srange(0, i-1):
                     g(x) = g(x)*(x-L[j])
                 f(x) = f(x)+g(x)
```

```
            return f

        expand(polinomio([-2,-1,1,2],[26,4,8,-2]))
```

```
Out[36]: x |--> -3*x^3 + 2*x^2 + 5*x + 4
```

```
In [28]: var('a b')
         L=[[-2,-1],[1,2],[3,-4]]
         modelo(x)=a*x+b
         find_fit(L,modelo)
```

```
Out[28]: [a == -0.47368421052953, b == -0.6842105263194624]
```

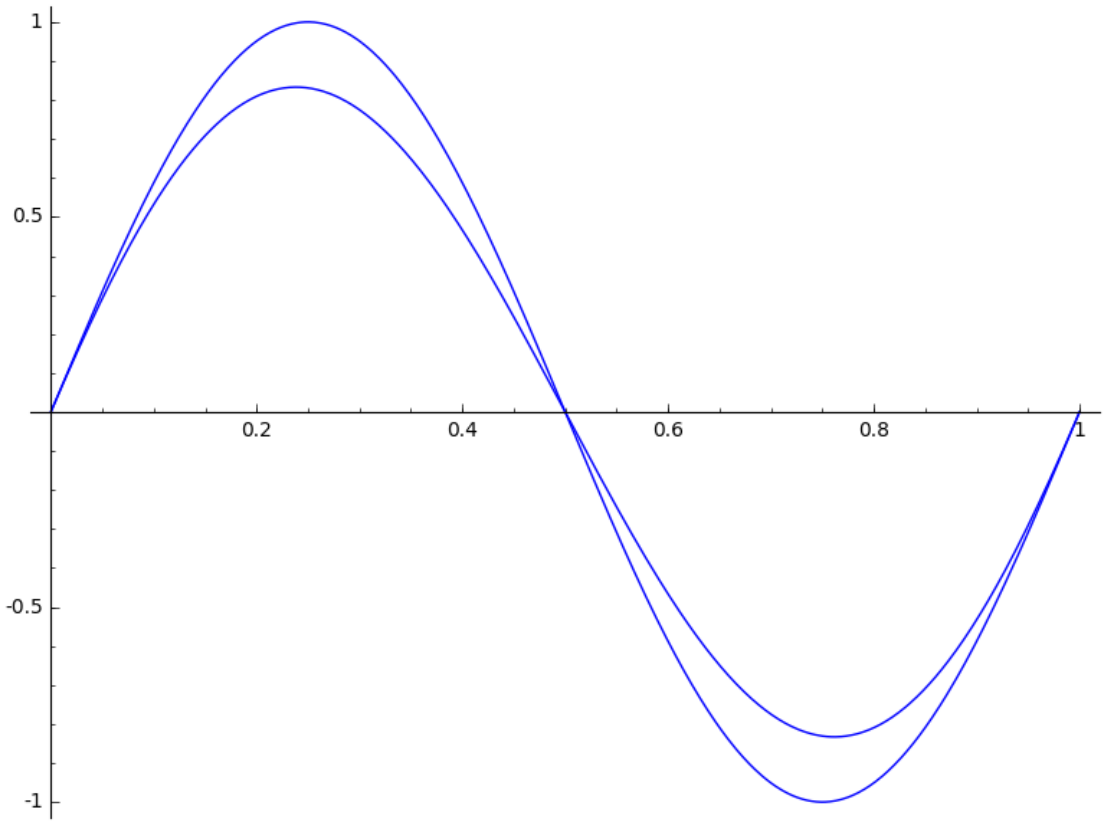
Ejercicio 17

```
In [54]: def combinaciones(a,b):
             num = factorial(a)
             den = factorial(b)*factorial(a-b)
             return num/den

         def bnf(f, n):
             g(x)=0
             for p in srange(0,n+1):
                 g(x) = g(x)+combinaciones(n,p)*f(p/n)*(1-x)^(n-p)*x^p
             return g

         f(x)=sin(2*pi*x)
         g(x) = expand(bnf(f,20))
         L = [f,g]
         sum([plot(L[i],x,xmin=0,xmax=1) for i in srange(0,2)])
```

```
Out[54]:
```



Ejercicio 21

```
In [64]: def H(n,nbits):
             suma = 0
             for i in srange(1,n+1):
                 suma += N(1/i, prec=nbits)
             return suma

         def H2(n,nbits):
             suma = 0
             for i in srange(1,n+1):
                 suma += 1/i
             return N(suma, prec=nbits)
```

Ejercicio 22

```
In [89]: def mi_gamma1(n, nbits):
             a = H2(n, nbits)
             b = N(log(n, base = e), prec = nbits)
             return a-b

         def mi_gamma2(n, nbits):
```

```
            suma = 0
            for k in srange(1,n+1):
                suma += (ceil(n/k) - n/k)/n
            return N(suma, prec = nbits)

        def mi_gamma3(n, nbits):
            suma = 0
            for k in srange(1,n+1):
                suma += 1/k - log(1+1/k, base = e)
            return N(suma, prec = nbits)

        n = 100000
        nbits = 200
        L=[]
        %time (L.append(abs(mi_gamma1(n,nbits)-N(euler_gamma,prec=nbits))))
        %time (L.append(abs(mi_gamma2(n,nbits)-N(euler_gamma,prec=nbits))))
        #%time abs(L.append(mi_gamma3(n,nbits)-N(euler_gamma,prec=nbits))) El tercero tarda m
        L
```

```
CPU times: user 2.49 s, sys: 44 ms, total: 2.54 s
Wall time: 2.41 s
CPU times: user 2.84 s, sys: 104 ms, total: 2.94 s
Wall time: 2.62 s
```

```
Out[89]: [4.99999166666666674999999999603174603216269841262265511544 90e-6,
          0.000221794764960807969731453586621931835858277722124700433296 57]
```

```
In [ ]:
```