Session-21

October 10, 2019

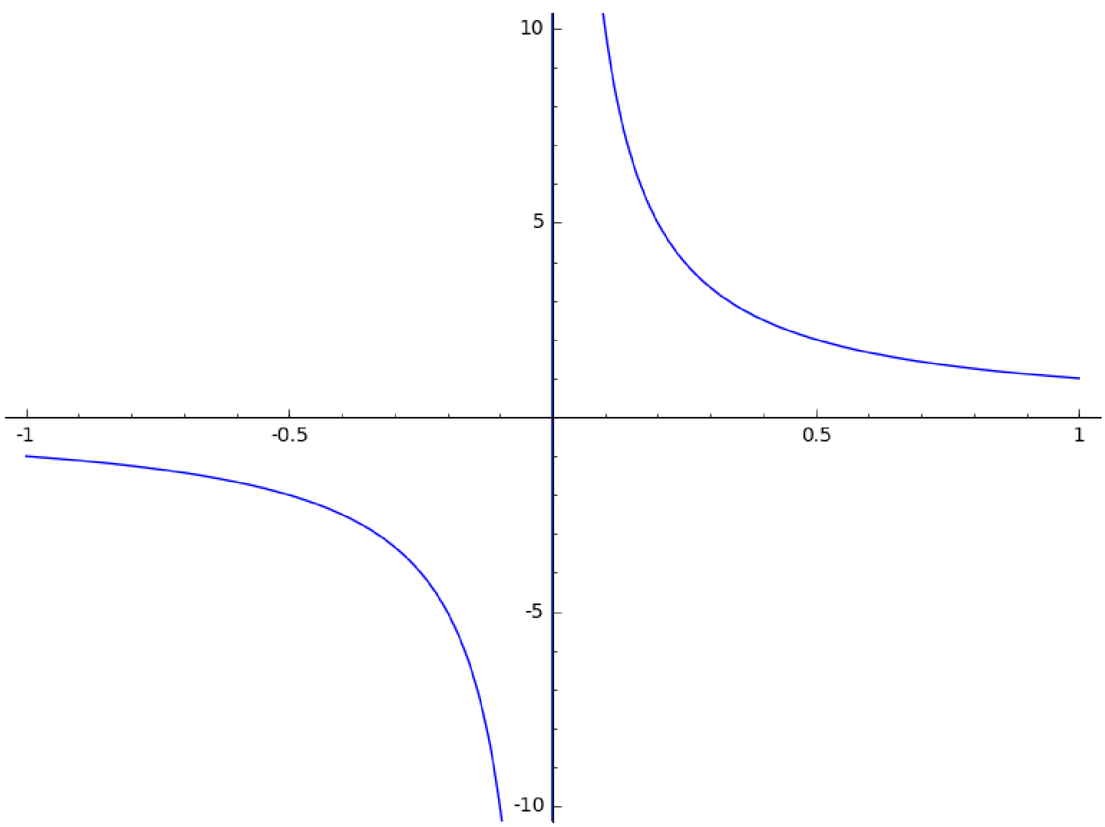
0.1 Parametric plots

```
In [1]: var('theta', latex_name="\\theta")
```

```
Out[1]: theta
```

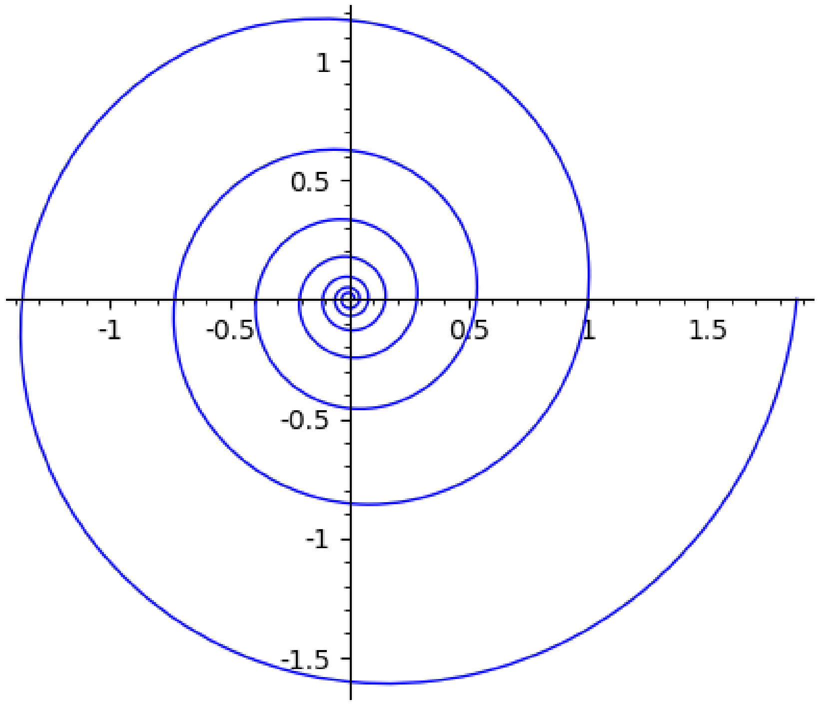
```
In [2]: polar_plot(2*cos(7*theta), (theta, 0, 2*pi))
```

```
Out[2]:
```




```
In [3]: polar_plot( exp(theta/10), (theta, -12*pi, 2*pi), plot_points=1000)
```

Out[3]:



0.2 Sudoku using Sage

```
In [4]: A = matrix(9, 9, [5,0,0, 0,8,0, 0,4,9, 0,0,0, 5,0,0, 0,3,0,
                          0,6,7, 3,0,0, 0,0,1, 1,5,0, 0,0,0, 0,0,0, 0,0,0, 2,0,8,
                          0,0,0, 0,0,0, 0,0,0, 0,1,8, 7,0,0, 0,0,4, 1,5,0, 0,3,0,
                          0,0,2, 0,0,0, 4,9,0, 0,5,0, 0,0,3])
```

```
In [5]: print A
```

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

```
In [6]: sudoku(A)
```

```
Out[6]: [5 1 3 6 8 7 2 4 9]
        [8 4 9 5 2 1 6 3 7]
        [2 6 7 3 4 9 5 8 1]
        [1 5 8 4 6 3 9 7 2]
        [9 7 4 2 1 8 3 6 5]
        [3 2 6 7 9 5 4 1 8]
        [7 8 2 9 3 4 1 5 6]
        [6 3 5 1 7 2 8 9 4]
        [4 9 1 8 5 6 7 2 3]
```

0.3 Laplacian of a scalar field

```
In [7]: var('x,y,z')
```

```
Out[7]: (x, y, z)
```

```
In [8]: g(x, y, z) = x^2 + y^3 + z^4 + x*y*z^2
```

```
In [9]: a=diff(g,2)
```

The output of the above call is Hessian of a scalar field.

```
In [10]: print a
```

```
[            (x, y, z) |--> 2              (x, y, z) |--> z^2             (x, y, z) |--> 2*y*z]
[          (x, y, z) |--> z^2              (x, y, z) |--> 6*y             (x, y, z) |--> 2*x*z]
[        (x, y, z) |--> 2*y*z          (x, y, z) |--> 2*x*z (x, y, z) |--> 2*x*y + 12*z^2]
```

Laplacian is the trace of Hessian

```
In [11]: L = diff(g,2).trace()
```

```
In [12]: L.show()
```

```
(x, y, z) |--> 2*x*y + 12*z^2 + 6*y + 2
```

0.4 Divergence of Vector Field

```
In [13]: f(x,y,z) = (x*y^2, y*z^2, z*x^2)
```

```
In [14]: derivative(f)
```

```
Out[14]: [  (x, y, z) |--> y^2 (x, y, z) |--> 2*x*y      (x, y, z) |--> 0]
         [    (x, y, z) |--> 0   (x, y, z) |--> z^2 (x, y, z) |--> 2*y*z]
         [(x, y, z) |--> 2*x*z     (x, y, z) |--> 0   (x, y, z) |--> x^2]
```

Divergence is the trace of above matrix.

```
In [15]: derivative(f).trace()
```

```
Out[15]: (x, y, z) |--> x^2 + y^2 + z^2
```

0.5 Arbitrary precision arithmetic

In sage you don't have to worry about the number of digits for precision.

```
In [42]: factorial(1000)
```

```
Out[42]: 4023872600770937735437024339230039857193748642107146325437999104299385123986290205920
```

Sage knows about prime numbers quite well.

```
In [17]: prime_range(1700, 1750)
```

```
Out[17]: [1709, 1721, 1723, 1733, 1741, 1747]
```

You can do linear algebra using sage.

```
In [18]: A=matrix(3,3,[1,2,3,4,5,6,7,8,0])
```

```
In [19]: print A
```

```
[1 2 3]
[4 5 6]
[7 8 0]
```

```
In [20]: A.det()
```

```
Out[20]: 27
```

```
In [21]: A.eigenvalues()
```

```
Out[21]: [-5.734509942225074?, -0.3883838424073199?, 12.12289378463240?]
```

```
In [22]: A.trace()
```

```
Out[22]: 6
```

```
In [23]: A*A
```

```
Out[23]: [30 36 15]
         [66 81 42]
         [39 54 69]
```

```
In [24]: A^3-5*A+4
```

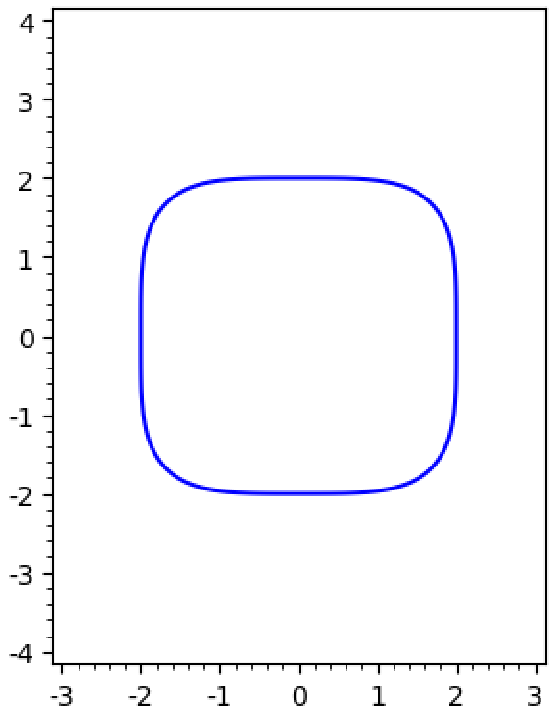
```
Out[24]: [278 350 291]
         [664 852 654]
         [703 860 445]
```

0.6 Implicit plots

g(x,y)=x^{4+y}4-16

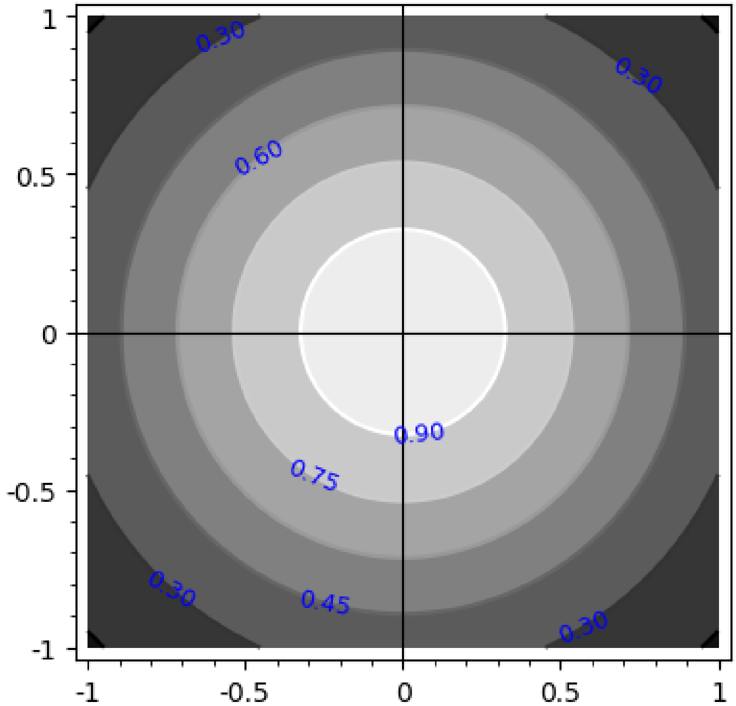
In [26]:
```
implicit_plot(g, (x,-3,3), (y,-4,4))
```

Out[26]:



In [27]:
```
f(x,y)=exp(-(x^2+y^2))
contour_plot(f,(x,-1,1),(y,-1,1),fill=True,axes=True,labels=True)
```

Out[27]:



0.7 Vector algebra

```
In [28]: u = vector( [1, 2, 3] )
         v = vector( [2, 3, 4] )
         a = v.dot_product(u)

In [29]: a

Out[29]: 20

In [30]: w=v.cross_product(u)

In [31]: w

Out[31]: (1, -2, 1)

In [32]: v.dot_product(w)

Out[32]: 0

In [33]: norm(w)

Out[33]: sqrt(6)

In [34]: norm(u)
```

Out[34]: sqrt(14)

In [35]: norm(v)

Out[35]: sqrt(29)

In [36]: u+v

Out[36]: (3, 5, 7)

In [37]: u-v

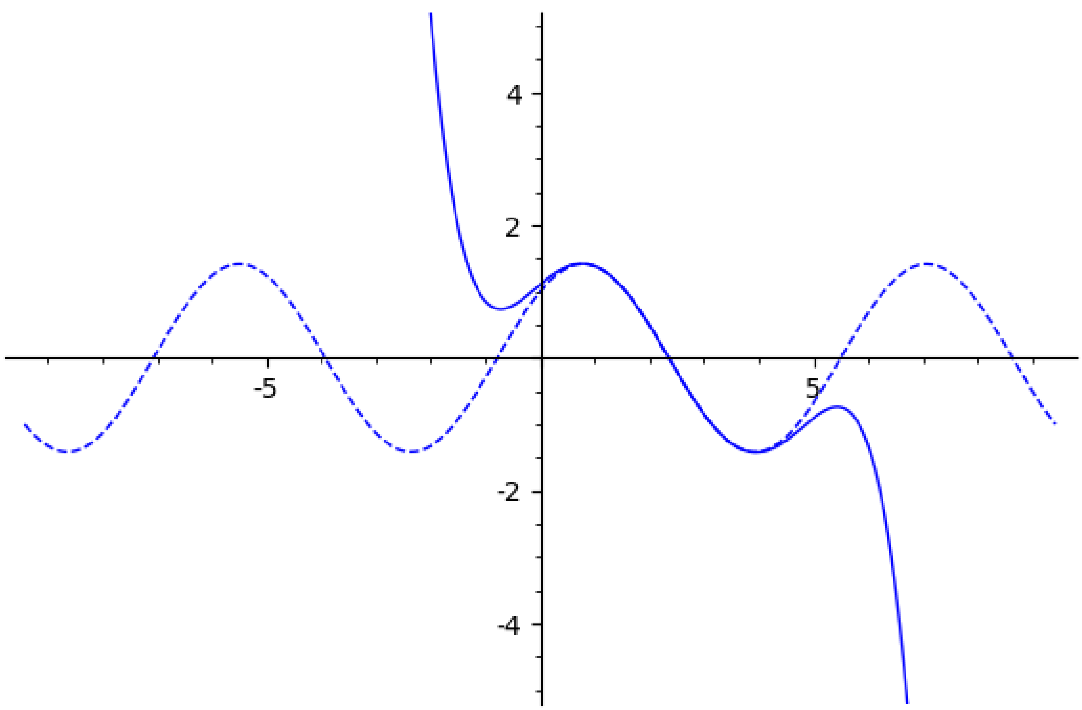
Out[37]: (-1, -1, -1)

0.8 Taylor series for approximation of a function

In [38]:
```
f(x) = cos(x) + sin(x)
p(x) = taylor(f(x), x, 3*pi/4, 5)
print p(x)
```

```
1/122880*sqrt(2)*(3*pi - 4*x)^5 - 1/384*sqrt(2)*(3*pi - 4*x)^3 + 1/4*sqrt(2)*(3*pi - 4*x)
```

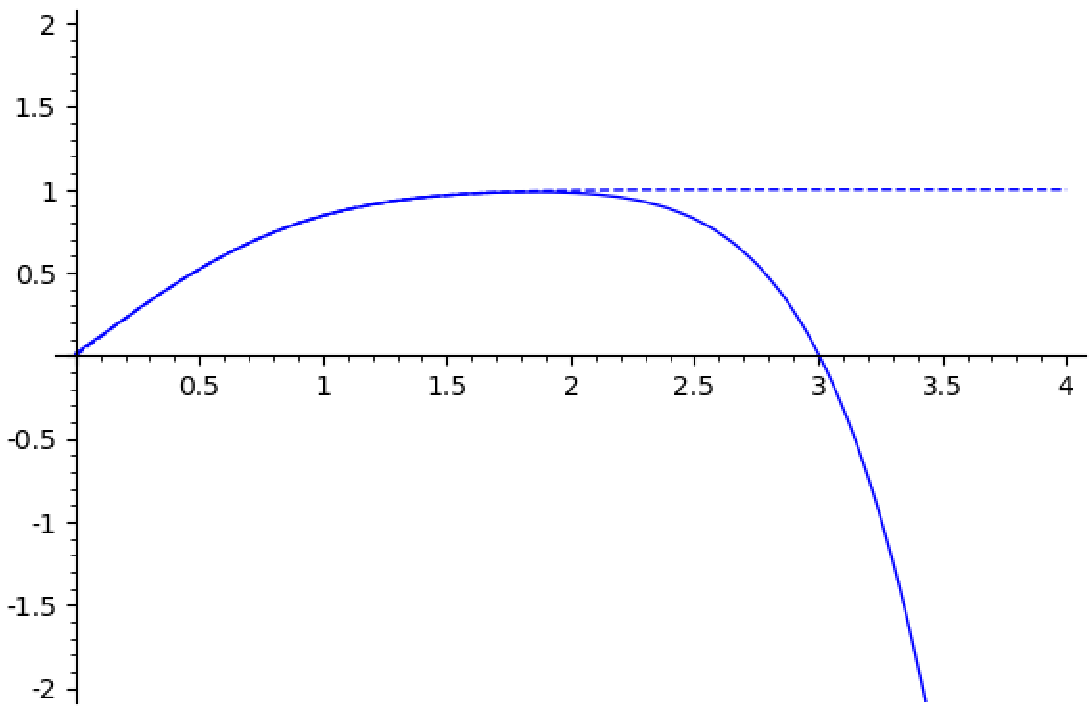
In [39]:
```
p1 = plot( p(x), -3*pi, 3*pi, ymax=5, ymin=-5)
p2 = plot( f(x), -3*pi, 3*pi, linestyle='--')
(p1+p2).show()
```



```
In [40]: f2 = erf(x)
         p2(x) = taylor(f2(x), x, 1, 5)
         print p2(x)
```

```
-1/3*(x - 1)^5*e^(-1)/sqrt(pi) + 1/3*(x - 1)^4*e^(-1)/sqrt(pi) + 2/3*(x - 1)^3*e^(-1)/sqrt(pi)
```

```
In [41]: p3a = plot( p2(x), 0, 4, ymax=2, ymin=-2)
         p3b = plot( f2(x), 0, 4, linestyle='--')
         (p3a+p3b).show()
```



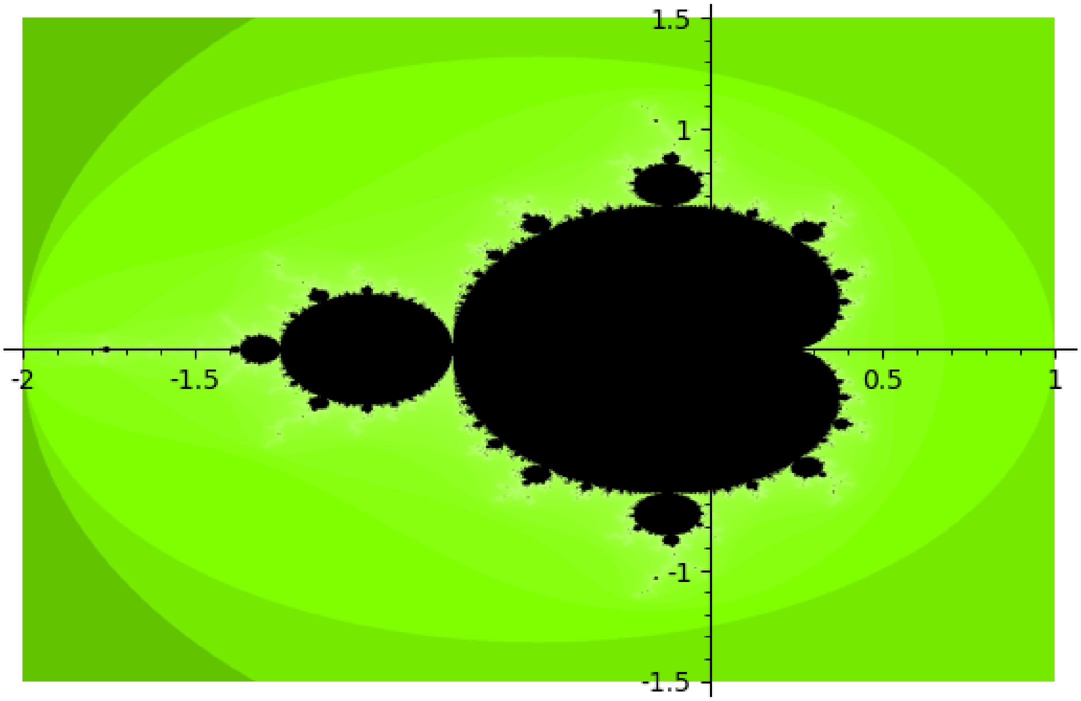
0.9 Fancy plots

0.10 Mandelbrot fractal

```
In [43]: def m(c):
             z = complex(0,0)
             c = complex(c)
             for j in range(100):
                 if abs(z) > 2 :
                     break
                 z = z^2 + c
             else:
                     return complex(0,0)
             return complex(0, j)
```

```
In [44]: complex_plot(m, (-2,1), (-1.5, 1.5), plot_points=1000)
```

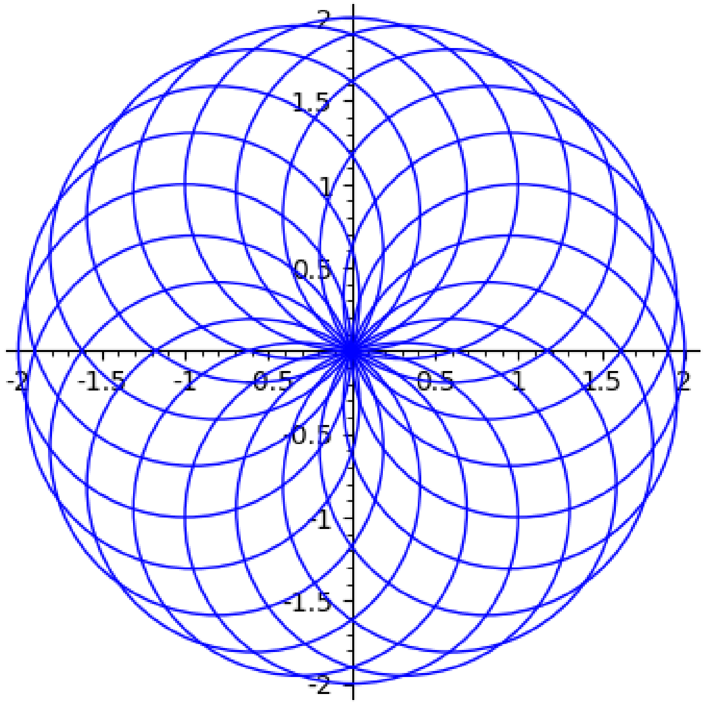
Out[44]:



0.11 Circling around

```
In [45]: v = vector([1, 0])
         var('j')
         k = 20
         disp = sum(circle(matrix([[cos(j*2*pi/k),-sin(j*2*pi/k)], [sin(j*2*pi/k), cos(j*2*pi/l
                      for j in range(k))
```

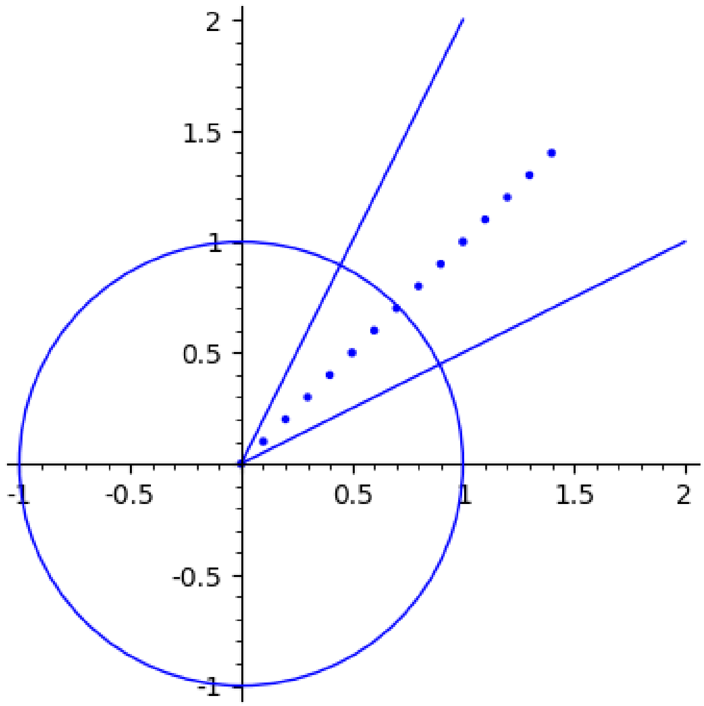
```
In [46]: disp.show()
```



0.12 Graphical elements

In [47]:
```
L = []
for i in range(15):
    L.append([i/10, i/10])

G = list_plot(L)
G += line([[0,0], [2,1]])
G += line([[0,0], [1,2]])
G += circle((0,0), 1)
G.show()
```



0.13 Platonic solids

```
In [48]: G=icosahedron((0, -2, 0), size=0.8, color='red', opacity=0.5)
         G.show()
```

```
Graphics3d Object
```

0.14 3D parametric plots

```
In [49]: var('s,t')
         fx = (3*sin(t) + cos(s) + 5)* cos(2*t)
         fy = (3*sin(t) + cos(s) + 5)* sin(2*t)
         fz = sin(s) + 2*cos(t)
         G = parametric_plot3d([fx, fy, fz], (s, 0, 2*pi), (t, 0, 2*pi), color='green', axes=T
         G.show()
```

```
Graphics3d Object
```

```
In [ ]:
```