SageMath For Calculus III

Contents

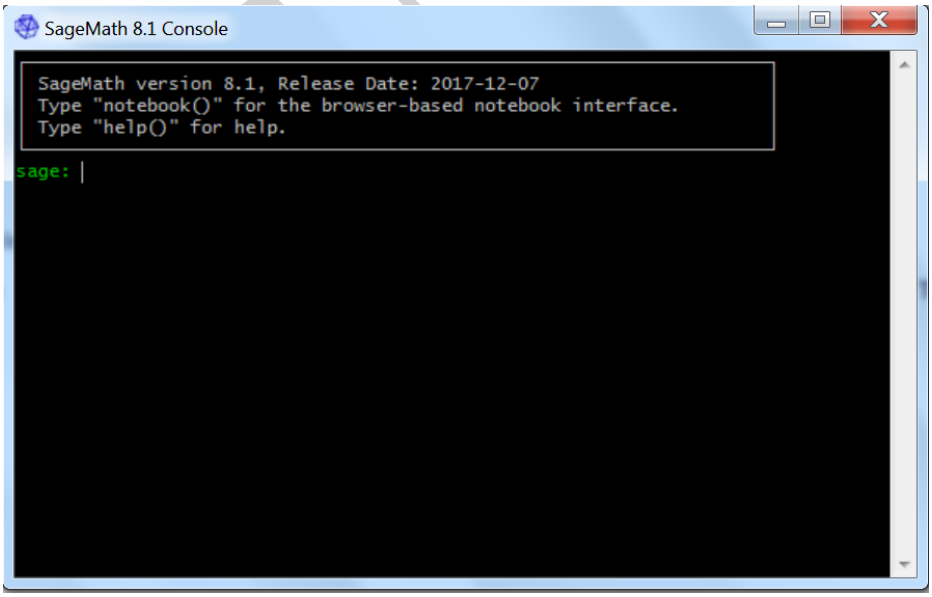
1. Introduction

1.1 Getting Started

SageMath http://www.sagemath.org/ (previously Sage or SAGE, "System for Algebra and Geometry Experimentation") is a computer algebra system with features covering many aspects of mathematics, including algebra, combinatorics, graph theory, numerical analysis, number theory, calculus and statistics. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more. Access their combined power through a common, **Python-based language** or directly via interfaces or wrappers. We will discuss basic syntax and frequently used commands in this chapter.

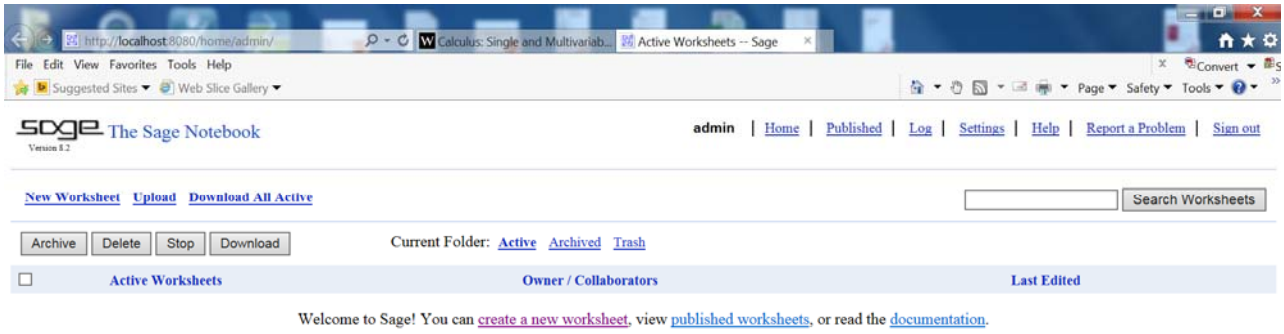
There are two ways to use Sage, you can run Sage on the SageMathCloud – COCALC (http://cloud.sagemath.org) or install SageMath (http://www.sagemath.org/download.html) and run it on your computer.

For information to install and use SageMath, go to http://doc.sagemath.org/. The latest version is SageMath 8.2 (2018-5-10). After the installation is complete, start SageMath by running SageMath 8.2  and you will see the following interface.

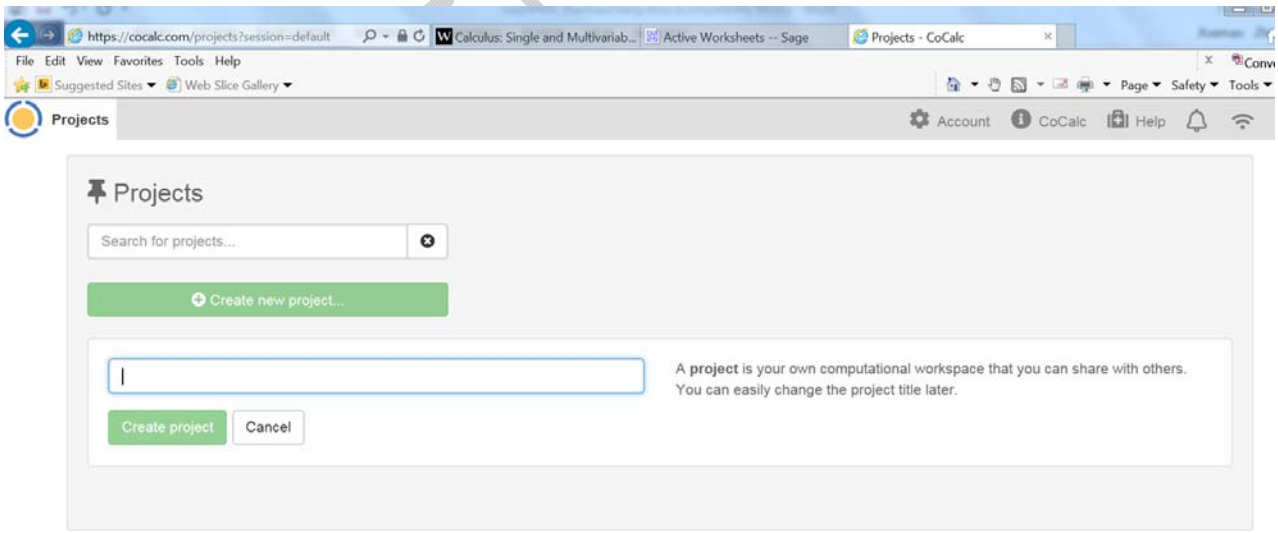


If you just run your SageMath code line by line, you can type your code in the console after the command prompt “sage:” directly.

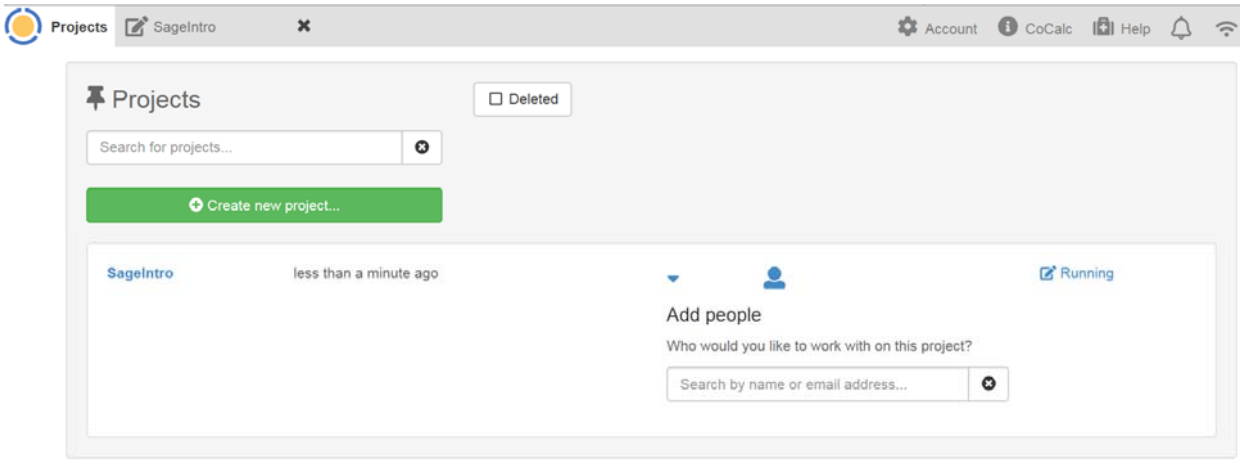
The second method to use SageMath is using a browser-based notebook. You can just type notebook() after the command prompt "sage:" to use this browser-based notebook. Keep in mind that this console should be left open while you use the browser-based note book. You need to create a password for the "admin" account. After logging in, all of your worksheets will be listed. Since it's the first time, click on "New Worksheet" to create your first worksheet. The screen will look like this:



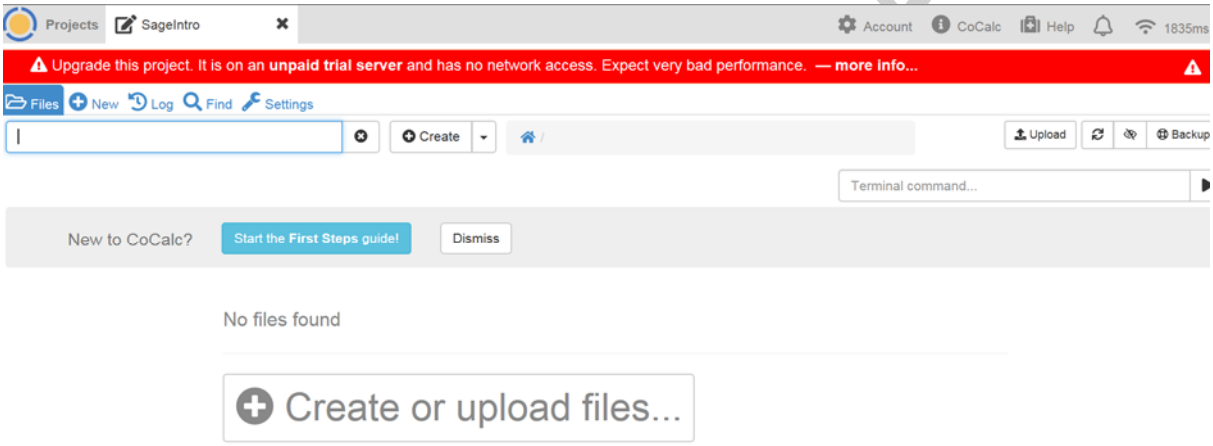
To use SageMath on the cloud, go to the Cocalc website https://cocalc.com/ or http://cloud.sagemath.org and create an account. For this class, let's run all the SageMath code on the cloud since it does not require any installations. After logging in, you will see all of your projects. Since it's the first time, click on "Create new project" . . . to create one.



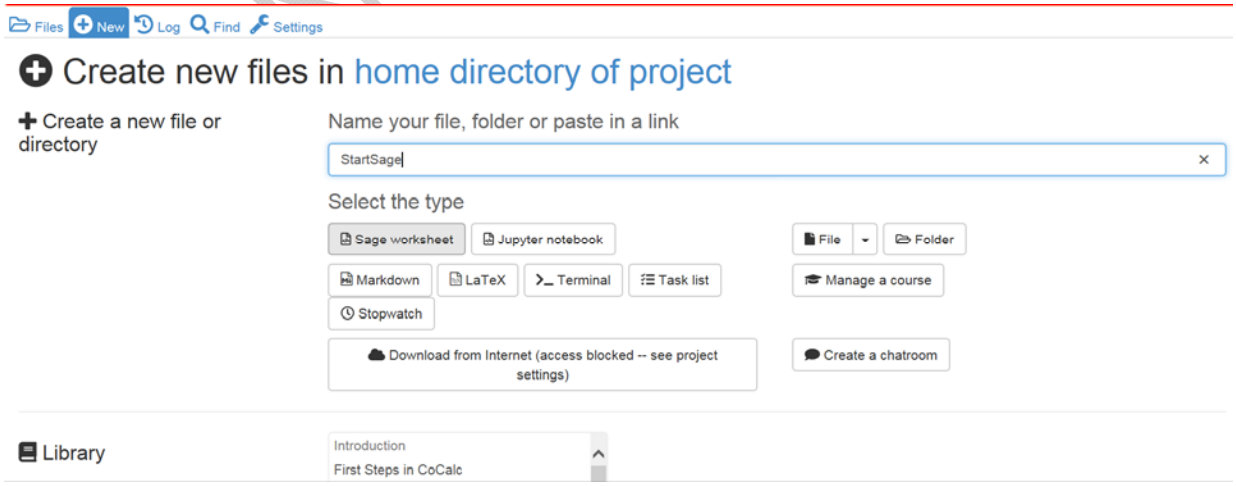
Give the project a name and click on "Create project". Your project now is created and listed under "Projects". The screen will look like this:



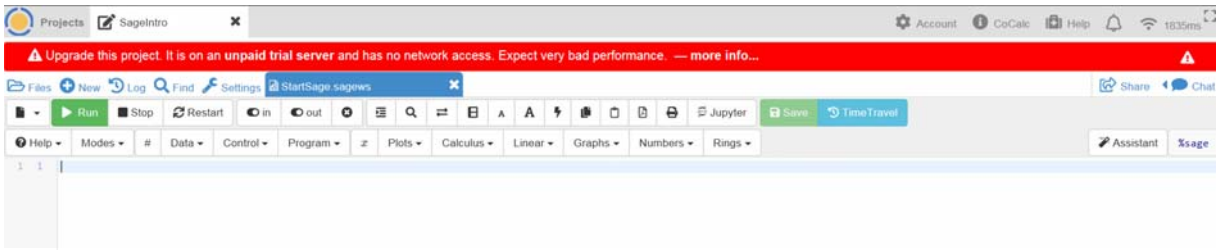
Click on the project you want to work on, and click "Create or upload files..."



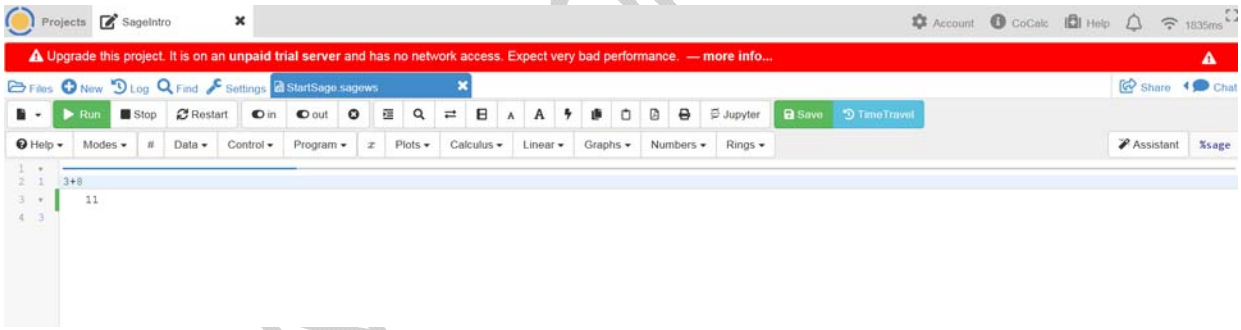
where we can create a file of upload a file from our computer. Since we want to run SageMath on cloud, we create a new file name StartSage, and select the type as Sage worksheet



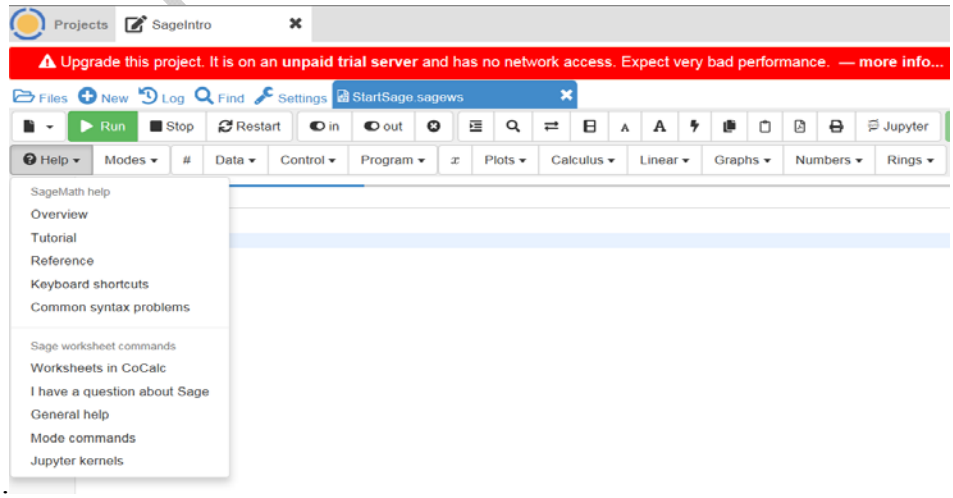
We are now in a Sage file and ready to use it. Pay attention that we are now viewing the file StartSage.sagews located inside of the project SageIntro.



Just start typing input commands (a cell formatted as an input box will be automatically created). For example, type 3+8. To evaluate this command or any other command(s) contained inside an input box, simultaneously press SHIFT+ENTER, or click on the green Run button. Be sure your mouse's cursor is positioned inside the input box or else select the input box(es) that you want to evaluate. This is how it looks:



SageMath provides an online help menu to answer many of your questions about the program.



For example, if you click "Tutorial", you will be brought to the website http://doc.sagemath.org/html/en/tutorial/index.html. If you click ""Worksheets in CoCalc", you will be brought to a GitHub website https://github.com/sagemathinc/cocalc/wiki/sagews. GitHub nowadays is a must for computer science students in job hunting. For more learning resources, please read Read Sage for Undergraduates by Gregory Bard or Mathematical Computation with Sage by Paul Zimmermann et. al.

SageMath Cloud not only lets you work anywhere as long as you have an Internet connection, but also allow you to share your file/project with your instructor or colleague. The only requirement is that the one who you want to share Sage files with should also have an account. Once he/she has it, you can give his/her permission to access your file. Notice that they have permission to access a particular file you choose, not every files you have in your account. Once you sign in, click "Add people" or the Edit icon of the project (under Projects) that you want to share, then search your colleague's name or email address in the section "Add people to project". A list of matching will show up. Choose the one you look for and add he/she to the project. That person will received an invitation email and now he/she can modify anything on that project.

1.2 SageMath Commands

Sage is built on top of the computer language Python. Therefore, you can use Python commands in Sage. In general, many important mathematical tasks can be done in SageMath without having too many lines of code.

Naming

Built-in Sage commands, functions, constants, and other expressions begin with lowercase letters and are (for the most part) one or more full-length English words (without capitalized). Furthermore, SageMath is **case sensitive**. For example, **plot, expand, print** and **show** are valid function names. **sin, def, gcd** and **max** are some of the standard mathematical abbreviations that are exceptions to the full-length English word(s) rule.

User-defined functions and variables can be any mixture of uppercase and lowercase letter and number. However, a name cannot begin with a number. User-defined functions may begin with an upper case letter, but this is not required. For example, **F1, g1, myPlot, Sol** and **Tech** are permissible function names.

brackets

SageMatg interprets various types of brackets differently.

- Parentheses, (): When there are multiple sets of parentheses in a formula, sometime mathematicians use brackets as a type of "strong parentheses". As it turns out, Sage needs the brackets for other things as well, like list or table, so you have to always use parentheses for grouping inside of formulas.
- Square brackets, []: It is used to construct a data structure with group of value such as a list or table.

Lists, Tables, and Arrays

A **list** (or string) of elements can be defined in Sage as [e1, e2, ..., en]. For example, the following command defines v = [1, 3, 5, 7,9] to be the list (set) of the first five odd positive integers.

```
v=[1, 3, 5, 7, 9]
v
```

```
[1, 3, 5, 7, 9]
```

As in Python, all lists are zero-based indexed. Here the index is 0, 1, 2, 3 and 4. You can access individual list elements. When referencing a member or the length of a list the number of list elements is always the number shown plus one. To refer to the kth element in a list name expr, just evaluate expr[k-1]. For example, to refer to the third element in v, we evaluate

```
v[2]
```

```
7
```

It is also possible to define nested lists whose elements are themselves lists, call sublists. Each sublist contains subelements. For example, the list w = [[1, 3, 5, 7,9], [2, 4, 6, 8,10]] contains two elements, each of which is a list (first five odd and even positive integers.)

```
w = [[1, 3, 5, 7,9], [2, 4, 6, 8,10]]
w
```

```
[[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

12

To refer to the kth subelement in the jth sublist of expr, just evaluate expr[j-1][k-1]. For example, to refer to the fourth subelement in the second sublist of w (or 8), we evaluate

```
w [1][3]
```

```
8
```

Lists can contain different variable types.

```
A = [ ] # This is a blank list variable
B = [1, 23, 45, 67] # this list creates an initial list of 4 numbers.
C = [2, 4, 'john']  # lists can contain different variable types.
```

```
mylist = ['Rhino', 'Grasshopper', 'Flamingo', 'Bongo']
D = len(mylist) # This will return the length of the list which is 3. The index is 0, 1, 2, 3.
print mylist[1] # This will return the value at index 1, which is 'Grasshopper'
print mylist[0:2] # This will return the first 2 elements in the list.
```

You can assign data to a specific element of the list using an index into the list. The list index starts at zero. Data can be assigned to the elements of an array as follows:

```
mylist = [0, 1, 2, 3]
mylist[0] = 'Rhino'
mylist[1] = 'Grasshopper'
mylist[2] = 'Flamingo'
mylist[3] = 'Bongo'
print (mylist)
```

You can change an individual list element:

```
mylist[0]= 'MATH'
mylist[1] = 240
print (mylist)
```

Remark: Don't change an element in a string!

```
mylist[:]= []  # this clears the list
print (mylist)
```

Tables

A **table** is used to display a rectangular array or list as a table.

```
table(list)
```

For example, the following command displays *v* in a table.

```
v=[['a', 'b', 'c'], [1 ,2 ,3] ,[4 ,5 ,6]]
table(v)
```

To highlight first row or first column, we set header-row = True or header-column = True, respectively. To put a box around each cell, set frame = True. Also, by default, align is 'left', we can change it to 'center' or 'right'.

```
v=[['a', 'b', 'c'], [1 ,2 ,3] ,[4 ,5 ,6]]
table(v, header_row =True, frame =True , align ='center')
```

Matrices

A **matrix** is used to display a rectangular array or list as a table.

```
matrix(list)
```

For example,

```
matrix([[1,2],[3,4]])
[1 2]
[3 4]
matrix.identity(2) #identity matrix of dimension 2
[1 0]
[0 1]
```

Arrays

Defined in **numpy**. Vectors and matrices for numerical data manipulation. NumPy (http://www.scipy-lectures.org/) is an extension to the Python programming language, adding support for large, multi-dimensional (numerical) arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

```
import numpy as np
data1 = [1, 2, 3, 4, 5] # list
arr1 = np.array(data1) # 1d array
arr1
        array([1, 2, 3, 4, 5])
data2 = [range(1, 5), range(5, 9)] # list of lists
arr2 = np.array(data2) # 2d array
arr2
        array([[1, 2, 3, 4], [5, 6, 7, 8]])

arr2[1,1]
        6
arr2[0,0]
        1
arr2[0,1]
        2
arr2[1,0]
        5
arr2[0, :] # row 0: returns 1d array ([1, 2, 3, 4])
        array([1, 2, 3, 4])
arr2[:, 0] # column 0: returns 1d array ([1, 5])
        array([1, 5])
arr2[:, :2] # columns strictly before index 2 (2 first columns)
        array([[1, 2], [5, 6]])
arr2[:, 2:] # columns after index 2 included
        array([[3, 4], [7, 8]])
arr2[:, 1:4] # columns between index 1 (included) and 4 (excluded)
        array([[2, 3, 4], [6, 7, 8]])
```

Commenting

One can insert comments on any input line. The comments should be follow by # sign.

1.3 Algebra

SageMath uses the standard symbols +, -,*, /, ^, ! for addition, subtraction, multiplication, division, raising powers (exponents), and factorials, respectively.

To generate numerical output in decimal form, use the command n(expr, digits = 3) to display to 3 decimal places.

NOTE: Sage can perform calculations to arbitrary precision and handle numbers that are arbitrarily large or small.

```
pi
        pi
n(pi , digits =4)
        3.142
2^5
        32
factorial (5)
        120
```

Here are Sage rules regarding the use of equal signs:
(1) A single equal sign (=) assigns a value to a variable. Thus, entering x = 3 means that x will be assigned the value 3.

```
x=3
5+x^2
        14
```

(2) A double-equal sign (==) is a test of equality between two expressions. Since we previously set x = 3, then evaluating x == 3 returns True, whereas evaluating x == 2 return False.

```
x==3
        True
x==2
        False
```

Another common usage of the double equal sign (==) is to solve equations, such as the command solve([x^2+x+1 == 0], x).

```
%var x
```

```
solve([x^2+x+1 == 0], x)
        [x == -1/2*I*sqrt(3) - 1/2, x == 1/2*I*sqrt(3) - 1/2]
```

1.4 Functions

There are two ways to represent functions in Sage, depending on how they are to be used. Consider the following example:

Example. Enter the function $\frac{x^2-x+4}{x-1}$ into SageMath.

Method 1: An explicitly way to present *f* as a function of the argument x is to enter:

```
%var x
f(x)=(x^2-x +4) /(x -1)
f(x)
        (x^2 - x + 4)/(x - 1)
f(2)
        6
```

Method 2: Define a function as:

```
def f(x): return (x^2-x +4) /(x -1)
f(x)
        (x^2 - x + 4)/(x - 1)
```

2. Graphs, Limits, Differentiation, and Integration

2.1 Plotting Graphs and Contour Plots

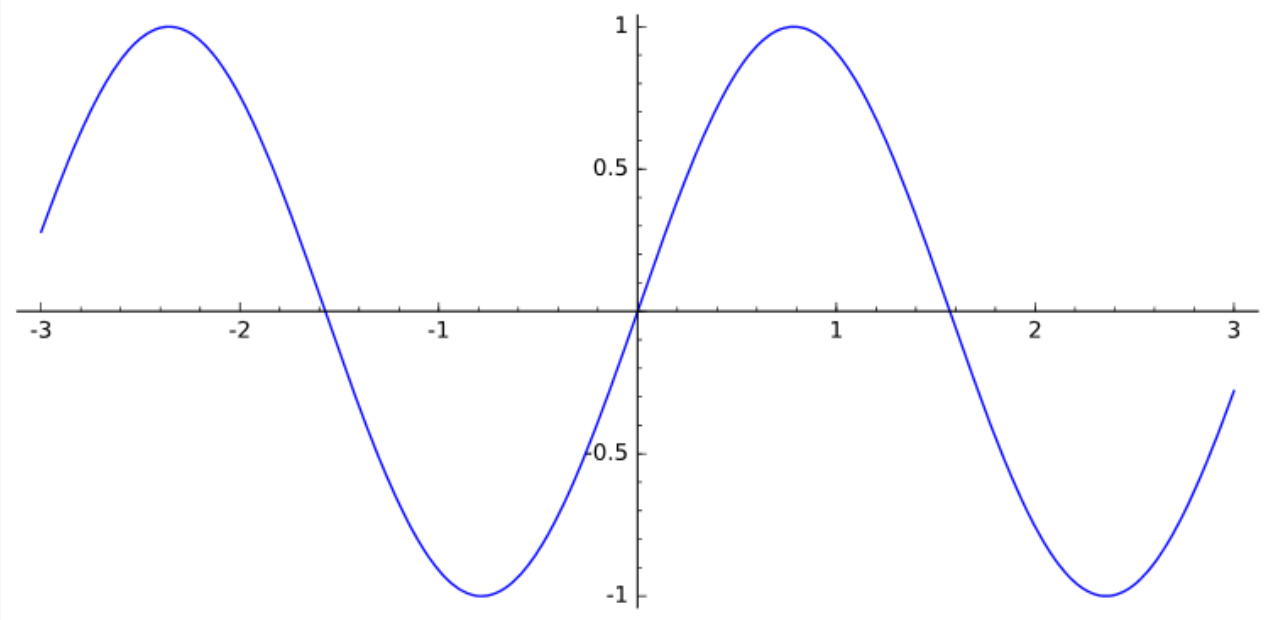
In this section, we will discuss how to plot graphs using Sage. For various options of a plot function, you can google the function or get help from the Sage Reference Manual http://doc.sagemath.org/html/en/reference/index.html.

Plot of a function:

The basic syntax for plotting the graph of a function y = f(x) with x ranging in value from a to b is plot(f,x,a,b).

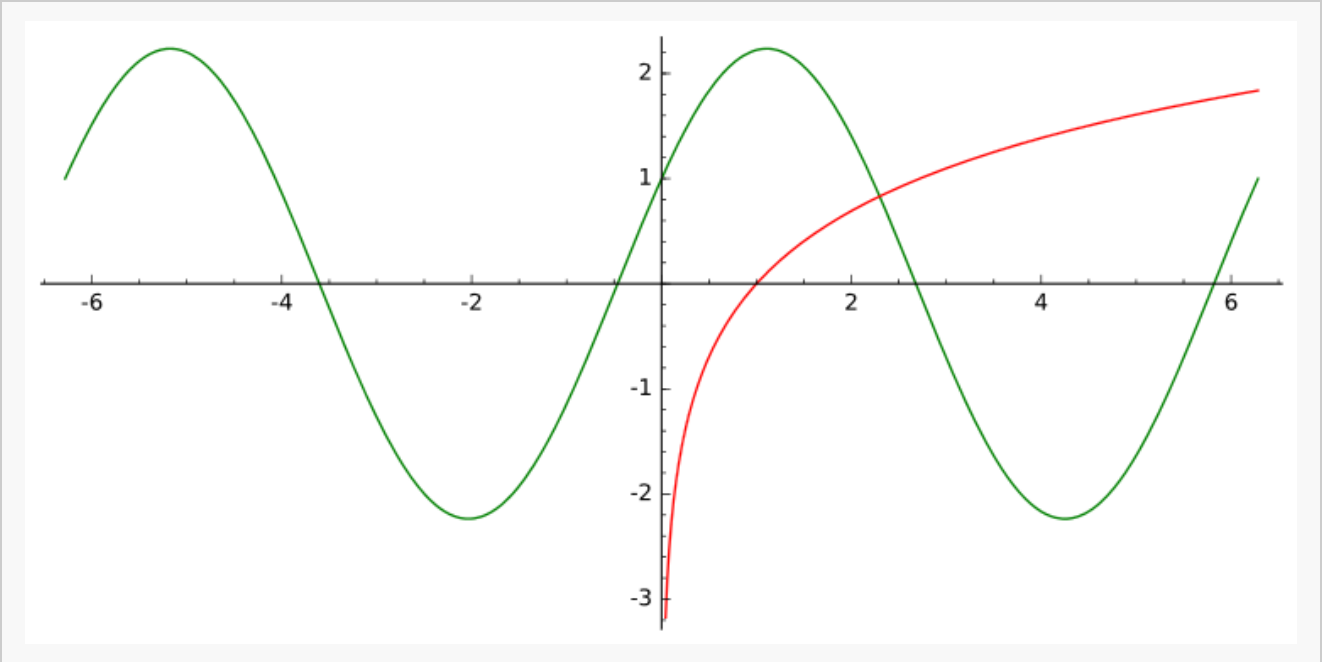
Example. Plot the graph of y = sin(2x) along the interval [-3,3]

```
g= plot (sin (2* x), x , -3 ,3)
g
```



Example. Plot the graph of f(x) = 2sin(x) +cosx and g(x) =lnx on a single graphic:

```
g1= plot (2*sin (x)+cos(x), x, -2*pi ,2* pi, color='green');
g2= plot (log(x), x,0,2* pi, color='red');
g1+g2
```



Plot Options:

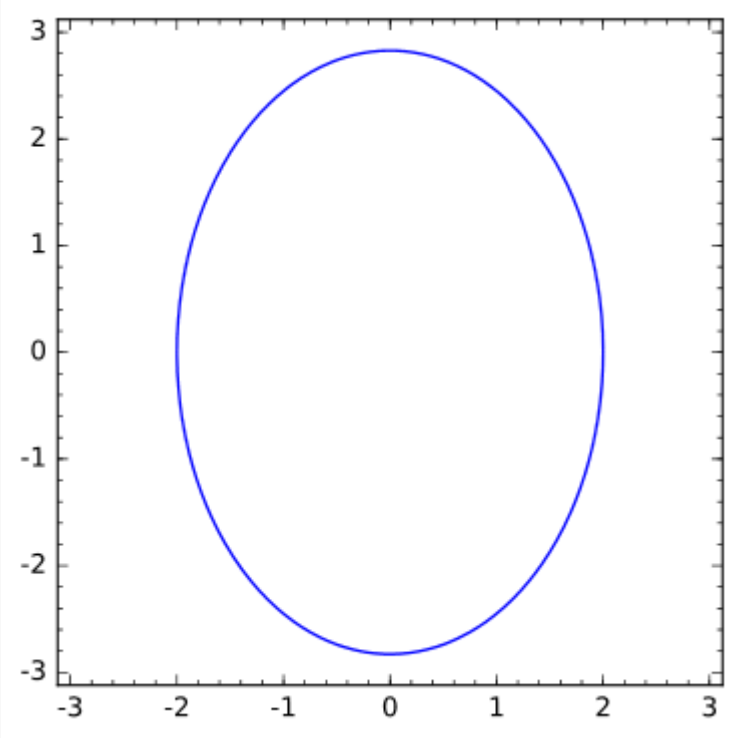
- Adding a title to a graph:
  plot(f(x),x,a,b,title="Here is a graph")
- Use figsize option to control the plot size:
  plot(f(x),x,a,b,figzie= ' a number ')
- Draw a graph with color:
  plot(f(x),x,a,b, color= 'a color')
- Draw a graph and specify its thickness:
  plot(f(x),x,a,b,thickness='a number')
- Draw graph with specify the line style and legend-label:

plot(f(x),x,a,b,color= 'a color',linestyle='–', thickness='a number',legend-label='f(x) ')

- Use frame option to puts a box around the graph
  plot(f(x),x,a,b,frame=True)
- Use axes-labels to verify the axes:
  plot(f(x),x,a,b,axes-labels=['x-axis, units', 'y-axis, units'])
- To draw an ellipse, use implicit-plot command:
  implicit-plot(f(x),(x,a,b),(y,c,d))

Example. Plot the graph of $x^2/2+y^2/4=2$.

```
%var x y
g= implicit_plot (x ^2/2+ y ^2/4==2 , (x, -3, 3) , (y, -3,3));
g
```



3D graphics:

- `plot3d()` - plot a 3d function

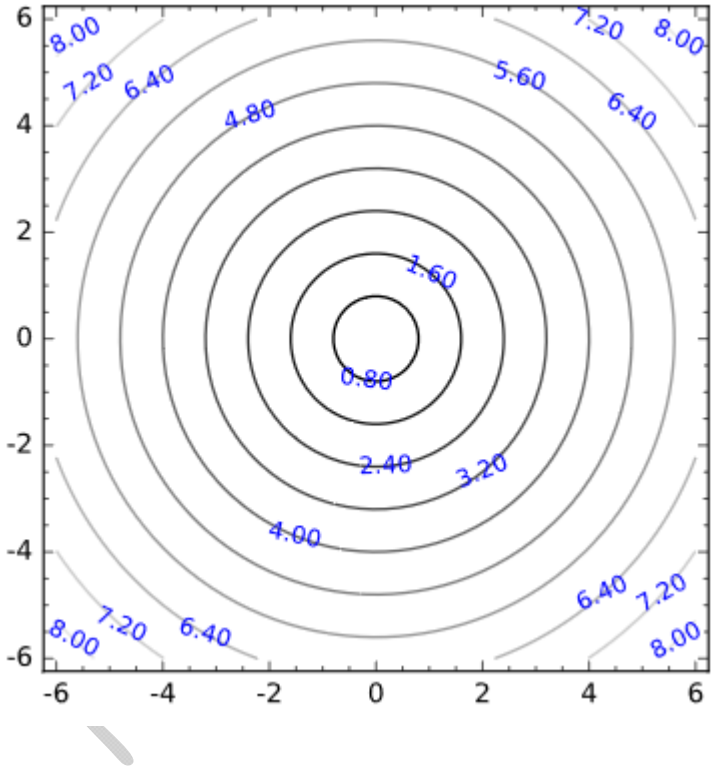
Example.

```
#Intersection of two planes
%var x y
g1=plot3d((-x-5*y+7)/3, (x,-10,10),(y,-10,10),color='green')
g2=plot3d((2*x+11*y-6)/4, (x,-10,10),(y,-10,10),color='yellow')
g1+g2
```

Contour lines are obtained from a surface by slicing it with horizontal planes. A contour plot is a collection of level curves labeled with function values. Contour plotting can be very useful when trying to get a handle on multivariable functions, as well as modeling. The basic syntax is essentially the same as for 3D plotting - simply an extension of the 2D plotting syntax.

Example. Draw a contour diagram for $f(x,y)=\sqrt{x^2+y^2}$.

```
f(x,y)=sqrt(x^2+y^2)
contour_plot(f, (x,-6,6), (y,-6,6),fill=False,labels=True,contours=10)
```



2.2 Limits

To compute the limit of function f(x) as x approaches a:

```
limit(f(x),x=a)
```

To compute the limit of function f(x) as x approaches a from the left (meaning x < a):

```
limit(f(x),x=a,dir='minus')
```

To compute the limit of function f(x) as x approaches a from the right (meaning x > a):

```
limit(f(x),x=a,dir='plus')
```

Example.

```
limit (sin (3* x)/x, x =0)
```

Example.

```
limit ( cos(x), x= infinity )
```

2.3 Differentiation

To calculate the derivative of a function, use diff() or .derivative() command:

```
diff(f(x)) or f(x).derivative()
```

To differentiate f(x,y) with respects to x::

```
diff(f(x,y),x)
```

To compute the n derivative respect to x:

```
diff(f(x),x,n)
```


Example.

```
#Derivative of f(x)
f(x)= cos (x ^2)
diff (f(x))
diff (f(x)).substitute (x= sqrt (pi /4) )
        -2*x*sin(x^2)
        -1/2*sqrt(2)*sqrt(pi)
```

2.4 Integration

The command **integral** can evaluate all rational functions and a host of transcendental functions, including exponential, logarithmic, trigonometric, and inverse trigonometric functions.

To integrate a function f(x,y) respects to x:

```
integral(f(x,y),x)
```

To integrate a f(x) over [a,b]:

```
integral(f(x,y),x,a,b)
```

Example.

```
# indefinite integral
integral (x^3 -3*x+2,x)
        1/4*x^4 - 3/2*x^2 + 2*x
```

Example.

```
# definite integral
integral (x^2*exp(x),x)
integral (x^2*exp(x),x,1,2)
                (x^2 - 2*x + 2)*e^x
                2*e^2 – e
```

3. Functions of Several Variables

In Chapter 2, 3D plots of a function of two variables and contour were illustrated. Let continue the discussion of functions of several variables using SageMath.

3.1 Limits

SageMath can be used to find the limit of a function with two variables at a point if the limit exits.

Example. Find the limit of $f(x,y) = x^2 + y^2$ at point (1,1).

```
f(x,y) = x^2 + y^2
f.limit(x=1).limit(y=1)
        (x, y) |--> 2
```

3.2 Partial Differentiation and Directional Derivatives

Example 1.

```
# partial derivatives
f(x,y)=x*sin (y) +e^(x*y)*cos(y)
fx(x,y)= f.diff(x)
fy(x,y) = f.diff(y)
fx; fy
        (x, y) |--> y*cos(y)*e^(x*y) + sin(y)
        (x, y) |--> x*cos(y)*e^(x*y) + x*cos(y) - e^(x*y)*sin(y)
f.gradient()
        (x, y) |--> (y*cos(y)*e^(x*y) + sin(y), x*cos(y)*e^(x*y) + x*cos(y)
- e^(x*y)*sin(y))
```

Calculation of directional derivatives can be done using a little bit programming. See the following example.

Example 2. Calculate the directional derivative of $f(x, y) = x^2 + y^2$ at (1, 0) in the direction of the vector $\vec{i} + \vec{j}$.

```
# directional derivative
%var x y
f = x^2+y^2;
dx(x,y) = diff(f,x).factor();
dy(x,y) = diff(f,y).factor();
d = vector([dx(1,0),dy(1,0)]);
u = vector([1,1]);
u=u/u.norm();  #normalization to a unit vector
A = d.dot_product(u)
show(A)
```

$$\sqrt{2}$$

3.3 Dot Product, Cross Product and Equation of a Plane

In SageMath, vectors are primarily linear algebra objects, but they are slowly becoming simultaneously analytic continuous functions.

Example 1.

```
#dot product
# RR denotes a matrix of reals
v = vector(RR, [1, 2, 3])
w = vector(RR, [-1, 0, -1])
v*w
```

```
-4.00000000000000
```

Example 2.

```
#cross product
v = vector(RR, [2, 1, -2])
w = vector(RR, [3, 0, 1])
vCrossW=v.cross_product(w)
vCrossW
vCrossW*v==0   #check if the cross product is perpendicular to v
vCrossW*w==0   #check if the cross product is perpendicular to w
```

```
(1.00000000000000, -8.00000000000000, -3.00000000000000)
True
True
```

Example 3. Find the equation of the plane perpendicular to $\vec{n} =< -1,3.2 >$ and passing through the point (1, 0, 4).

```
# equation of a plane
%var x y z
n = vector([-1, 3, 2])
P0 = vector([1, 0, 4])
P = vector([x, y, z])
n*(P-P0)==0;
```

```
-x + 3*y + 2*z - 7 == 0
```

3.4 The Chain Rule

The chain rule problems in multivariate calculus can be solved by SageMath.

Example. Suppose that $z = f(x, y) = x \sin y$, where $x = t^2$ and $y = 2t + 1$. Let $z = g(t)$. Compute g'(t).

```
var('t u v')
x = function('x')(t)
x=t^2
y = function('y')(t)
y=2*t+1
f(u,v) = u*sin(v)
diff(f(x,y), t)
```

```
(t, u, v)
2*t^2*cos(2*t + 1) + 2*t*sin(2*t + 1)
```

3.5 The Tangent Plane

Example. Write an equation of the plane tangent to the surface $z = x^2 + y^2$ at the point P(3, 4 25).

```
#Tangent plane Example 14.3.1
%var x y z
#var('x, y, z')
f(x, y, z)=x^2+y^2-z
grad_f=f.gradient()
fx, fy, fz =grad_f
fx(3,4,25)*(x-3)+fy(3,4,25)*(y-4)+fz(3,4,25)*(z-25)==0
```

```
6*x + 8*y - z - 25 == 0
```

3.6 Critical Points

Example. Write an equation of the plane tangent to the surface $z = x^2 + y^2$ at the point P(3, 4 25).

```
#Classification of critical points
#Example 15.1.6
%var x y
f(x,y)=x^2/2+3*y^3+9*y^2-3*x*y+9*y-9*x
fx=f.diff(x)
fy=f.diff(y)
fxx=diff(f,x,x)
fyy=diff(f,y,y)
fxy=diff(f,x,y)
#Find critical points
cpoints=solve([fx==0,fy==0],[x,y],solution_dict=True)
for sol in cpoints:
    if ((sol[x] in RR) and (sol[y] in RR) ):
        show([sol[x],sol[y]])
```

[3,-2]

[12,1]

```
#Check the first critical point (3,-2)
a,b=3,-2
show((fxx(a,b),fyy(a,b),fxy(a,b)))
D=fxx(a,b)*fyy(a,b)-fxy(a,b)^2
show(D)
```

(1, −18, −3)

−27

```
print "Since D<0, (3,-2) is a saddle point"
```

Since D<0, (3,-2) is a saddle point

```
#Check the first critical point (12,1)
a,b=12,1
show((fxx(a,b),fyy(a,b),fxy(a,b)))
D=fxx(a,b)*fyy(a,b)-fxy(a,b)^2
```

```
show(D)
```

```
(1, 36, -3)
27
```

```
print "Since D>0, and f_{xx}>0, (12,1) is a local minimum"
```

```
Since D>0, and f_{xx}>0, (12,1) is a local minimum
```

3.7 Lagrange Multiplier

Example. Find the extremal values of the function *f(x, y) = xy* subject to the constraint

$$g(x,y) = \frac{x^2}{8} + \frac{y^2}{2} - 1 = 0.$$

```
#Lagrange Multiplier
x, y, lam = var('x, y, lam')
f = x*y
g = x^2/8 + y^2/2-1
h = f - g * lam   #Lagrangian function
gradh = h.gradient([x, y, lam])
critical = solve([gradh[0] == 0, gradh[1] == 0, gradh[2] == 0, ],x, y, lam, solution_dict=True)
critical
```

```
[{y: -1, x: -2, lam: 2}, {y: 1, x: 2, lam: 2}, {y: 1, x: -2, lam: -2}, {y: -1, x: 2, lam: -2}]
```

```
f(critical[0][x], critical[0][y])
f(critical[1][x], critical[1][y])
f(critical[2][x], critical[2][y])
f(critical[3][x], critical[3][y])
```

```
2
2
-2
-2
```

4. Multiple Integrals

4.1 Double Integrals

Example 1.

```
#double integral
#var('x, y')
%var x y
f=cos(x)*sin(y)
integral(integral(f, x, 0, pi/2), y, 0, pi/2)
                1
```

Example 2.

```
#double integral, Example 16.2.7
%var x y
f=x*sqrt(y^3+1)
#integration order matters
integral(integral(f, y, x/3, 2), x, 0, 6) #this does not work
integral(integral(f, x, 0, 3*y), y, 0, 2)
               integrate(x*integrate(sqrt(y^3 + 1), y, 1/3*x, 2), x, 0, 6)
               26
```

4.2 Triple Integrals

Example 1. $\int_0^4 \int_0^4 \int_0^4 (1 + xyz) dxdydz$

```
#triple integral over a rectangular block
%var x y z
f=1+x*y*z
integral(integral(integral(f, x, 0, 4), y, 0, 4),z,0,4)
                        576
```

Example 2. $\int_{-3}^{3} \int_{-\sqrt{9-x^2}}^{\sqrt{9-x^2}} \int_{\sqrt{x^2+y^2}}^{3} z dz dy dx$

```
#triple integral over a z-simple solid
%var x y z
f(x,y)=integral(z,(z,sqrt(x^2+y^2),3))
show (f)
g(x)=integral(f(x,y),(y,-sqrt(9-x^2), sqrt(9-x^2)))
show(g)
h(x)=integral(g,(x,-3,3))
print "Answer=", (h)
```

```
Answer= x |--> 81/4*pi
```

4.3 Double Integrals in Polar Coordinates

Example . $\int_0^{\pi/4}\int_1^2 \frac{1}{r^2}\, dr d\theta$

```
# Double Integrals in Polar Coordinates
%var r theta
f=1/(r^2)
integral(integral(f,r,1,2), theta,0,pi/4)
```

```
1/8*pi
```

4.4 Triple Integrals in Cylindrical and Spherical Coordinates

Example 1. $\int_0^4 \int_0^{\pi/6} \int_0^6 1.2 r dr d\theta dz$

```
# Triple Integrals in Cylindrical Coordinates
%var z r theta
f=1.2*r
integral(integral(integral(f,z,0,4),r,0,6),theta,0,pi/6)
```

```
14.399999999999999*pi
```

Example 2. Find the volume of the uniform ``ice-cream cone" that is bounded by the cone $\phi = \pi/6$ and the sphere $\rho = 2a\cos(\phi)$ of radius a.

```
# Triple Integrals in Spherical Coordinates
%var rho phi theta a
f=(rho^2)*sin(phi)
f(phi,theta)=integral(f,(rho,0,2*a*cos(phi)))
g= integral(integral(f,phi,0,pi/6),theta,0,2*pi)
show(g)
```

$$(7/12)\pi a3$$

5. Vector Valued Functions

5.1 Parameterized Curves

Example 1. Find parametric equations for the line parallel to the vector <2,3,4> and through the point (1, 5, 7).

```
#Example 17.1.4
%var t
r0=vector([1,5,7]);
pv=vector([2,3,4]);
r0+t*pv
```

```
(2*t + 1, 3*t + 5, 4*t + 7)
```

Example 2. Graph the curve with parametric equations *x=(1+sin4t)cost, y=(1+sin4t)sint, z=1+sin4t.*

```
# Graph the curve with parametric equations
%var t
parametric_plot3d(((1+sin(4*t))*cos(t),(1+sin(4*t))*sin(t),(1+sin(4*t)))
,(t,-2*pi,2*pi))
```

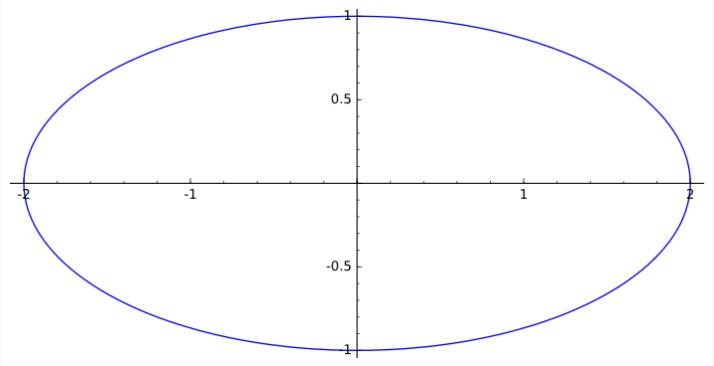
Example 3. Find the velocity and acceleration vectors for the particle whose equations of motion are given

```
#Section 17.2  Exercise 6
%var t
r(t)=(3*(cos(t))^2,3*(sin(t))^2,t^2)
v=diff(r(t),t)
a=diff(v,t)
print v
```

```
print a
        (-6*cos(t)*sin(t), 6*cos(t)*sin(t), 2*t)
        (-6*cos(t)^2 + 6*sin(t)^2, 6*cos(t)^2 - 6*sin(t)^2, 2)
```

Example 4. Find the circumference of the ellipse given by the parametric equations x=2cost, y=sint, $0 \le t \le 2\pi$.

```
#Example 17.2.6
%var t
r(t)=(2*cos(t),sin(t))
parametric_plot(r(t),(t,0,2*pi))
dr=diff(r(t),t)
s(t)=sqrt((dr[0]^2+dr[1]^2).simplify_trig());
length=integral(s(t),t,0,2*pi)
print length
n(length, digits =4)
numerical_integral(s(t),0,2*pi)
```

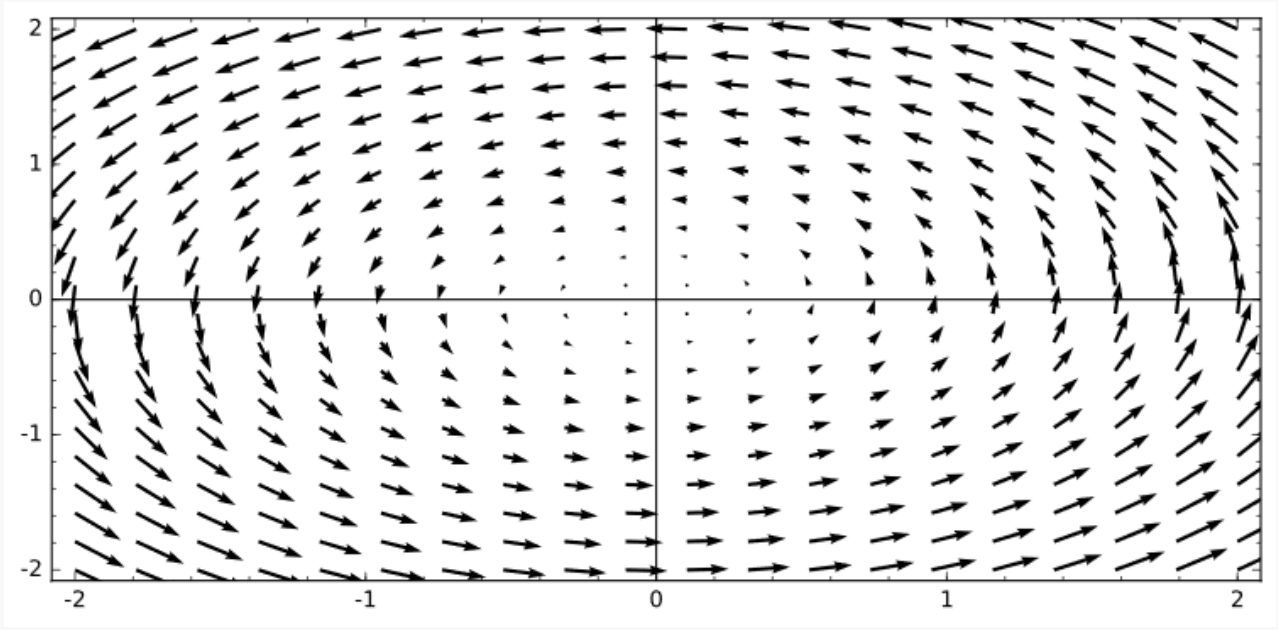



```
integrate(sqrt(-3*cos(t)^2 + 4), t, 0, 2*pi)
9.688466038349327
(9.688448220528912, 6.582441018563226e-06)
```

5.2 Vector Fields

Example 1. Sketch the vector field in 2-space given by <*-y,x*>

```
#Example 17.3.1
%var x y
plot_vector_field((-y,x), (x,-2,2), (y,-2,2))
```



Example 2. Describe the vector field in 3-space given by<*x, y, z*>

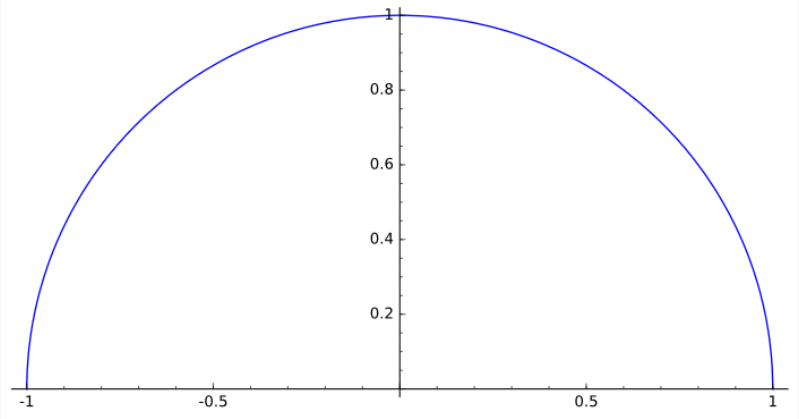
```
#Example 17.3.3
%var x y z
plot_vector_field3d((x,y,z), (x,-1,1), (y,-1,1),(z,-1,1))
```

6. Line Integrals

6.1 Line Integrals

Example 1. Evaluate $\int_C (2 + x^2 y)ds$, where C is the upper half of the unit circle traversed in the counterclockwise sense.

```
#Line integral
%var t
half_circle=parametric_plot((cos(t),sin(t)),(t,0,pi()));
show(half_circle, aspect_ratio=1)
%var x,y,t
x=cos(t)
y=sin(t)
dx=diff(x, t)
dy=diff(y,t)
ds=sqrt((dx)^2 + (dy)^2 )
h=integral((2+x^2 * y)*ds, 0, pi)
print "Answer=", (h)
```




```
Answer= 2*pi + 2/3
```

6.2 Line Integrals of Vector Fields

Example.

```
#Example 18.4.4
%var x,y
P=y^2;
Q=x;
f(x,y)=diff(Q,x)-diff(P,y);
integral(integral(f(x,y), x, 0, 2), y, 0, 3)
p1 = implicit_plot(y==0, (x,0,2), (y, 0,3),cmap=["red"])
p2 = implicit_plot(x==2, (x,0,3), (y, 0,3),cmap=["red"])
p3 = implicit_plot(y==3, (x,0,2), (y, 0,4),cmap=["red"])
p4 = implicit_plot(x==0, (x,0,2), (y, 0,3),cmap=["red"])
py = implicit_plot(x==0, (x,-4,4), (y, -4,4));
px = implicit_plot(y==0, (x,-4,4), (y, -4,4));
show(px+py+p1+p2+p3+p4, aspect_ratio=1)
```

```
-12
```