Work Sheet 3

Sajeev G P

1/18/2019

```
# Basic Linear Algebra
```

Sage is able to build vectors of elements in a ring. When no ring is specified, it deduces it from the given elements.

Look at the different parents of v_1 and v_2.

```
v1 = vector([1,2,3])
print v1
print v1.parent()
v2 = vector([-1.1, 2.2, 3])
print v2
print v2.parent()
```

```
(1, 2, 3)
Ambient free module of rank 3 over the principal ideal domain Integer Ring
(-1.10000000000000, 2.20000000000000, 3.00000000000000)
Vector space of dimension 3 over Real Field with 53 bits of precision
```

Sage is able to build matrices of elements in a ring. Similar to vectors, the ring is deduced from the elements.

In this case here, we explicitly tell Sage that the ring is RDF, though.

```
m1 = matrix([[1, 2], [4, -1]])
print m1
print m1.parent()
m2 = matrix(RDF, [[1, 2], [4, -1]])
print m2
print m2.parent()
```

```
[ 1  2]
[ 4 -1]
Full MatrixSpace of 2 by 2 dense matrices over Integer Ring
[ 1.0  2.0]
[ 4.0 -1.0]
Full MatrixSpace of 2 by 2 dense matrices over Real Double Field
```

```
A = matrix([[ 1, 2, 3, 4],
            [ 5, 6, 7, 8],
```

```
               [  9,10,11,12]])
print A[1, 3] # 2nd row, 4th element: 8
print A[:, 1] # 2nd column: [2, 6, 10] transposed
```

8
[2]
[6]
[10]

Here, we add, multiply and dot-product two rows or columns of the matrix A.

```
A = matrix([[ 1, 2, 3, 4],
            [ 5, 6, 7, 8],
            [ 9,10,11,12]])
# sum of second and third column
print A[:, 1] + A[:, 2]
# dot-product of first and second row (note, we have to transpose it\
   !)
print A[0, :] * A[1, :].transpose()
```

[5]
[13]
[21]
[70]

If you actually mean to multiply elements and not the dot-product, use the method elementwise_product.

```
A = matrix([[ 1, 2, 3, 4],
            [ 5, 6, 7, 8],
            [ 9,10,11,12]])
A.elementwise_product(A)
```

[1 4 9 16]
[25 36 49 64]
[81 100 121 144]

You can use start:end to specify a range of elements to pick. The end index is not inclusive!

```
A = matrix([[ 1, 2, 3, 4],
             [ 5, 6, 7, 8],
             [ 9,10,11,12]])
# Look carefully at the output of the following to understand what \
   part of the matrix got extracted.
# Feel free to play with the indices to see how they work.
A[0:2, 1:3]
```

[2 3]
[6 7]

You can either all the method .inverse() or use the   operator.
Note, that the inverse depends on the ring!

```
m1 = matrix([[1, 2], [4, -1]])
print m1.inverse()
m2 = matrix(RDF, [[1, 2], [4, -1]])
print ~m2
```

[1/9 2/9]
[4/9 -1/9]
[0.1111111111111111 0.2222222222222222]
[0.4444444444444444 -0.1111111111111111]

Kernel

```
A = matrix(RDF, [[1, -2, 2],
                 [4, -1, 1],
                 [1, -1, 1]])
kernel(A)
```

Vector space of degree 3 and dimension 1 over Real Double Field
Basis matrix:
[1.0 0.33333333333333326 -2.3333333333333333]

You can solve a linear system either left or right. The backslash \ is a shortcut.

```
A = matrix(RDF, [[1, -2, 9],
                 [4,  0, 2],
                 [1, -2, 1]])
b = vector(RDF, [9, 5, -1])
x = A.solve_right(b)
print "x:          {}".format(x)
print "check A*x: {}".format(A * x)
print "bashslash: {}".format(A \ b)
```

x: (0.625, 1.4375, 1.25)
check A*x: (9.0, 5.0, -1.0)
bashslash: (0.625, 1.4375, 1.25)

This code iterates over the left eigenvalues/eigenvectors of a matrix B. The method returns a triple of eigenvalues, vectors and multiplicity.

```
B = matrix([[1, 3, 2],
            [1, 3, 2],
            [1, 3, 6]])
for ev, evec, mult in B.eigenvectors_left():
    print("Eigenvalue: {} with eigenvector: {} of multiplicity {}".\
   format(ev, evec[0], mult))
```

Eigenvalue: 8 with eigenvector: (1, 3, 4) of multiplicity 1
Eigenvalue: 2 with eigenvector: (1, 3, -2) of multiplicity 1
Eigenvalue: 0 with eigenvector: (1, -1, 0) of multiplicity 1

```
# Plotting
```

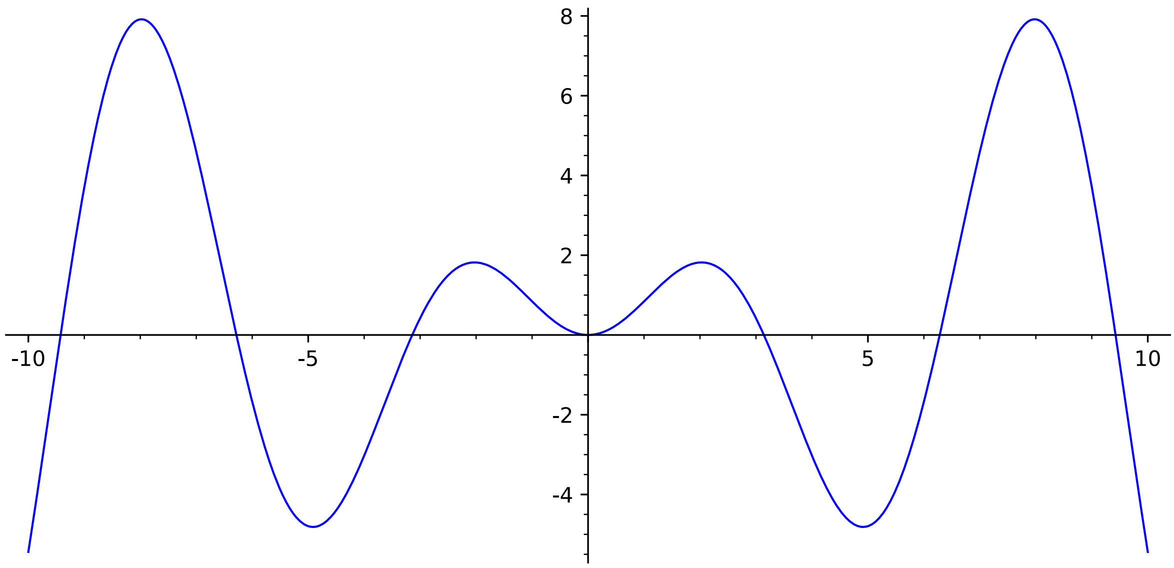
This is a very simple example how to plot a function.

For plots of symbolic functions you need to make sure, that x is a symbolic variable. x is defined as such a variable by default, but you might have overwritten it. In such a case, x = var('x') redefines x as such a symbolic variable with the name "x".

The second argument (x, -10, 10) defines the interval where to evaluate the symoblic expression. In this case here, it says to evaluate $x \in [-10, 10]$.

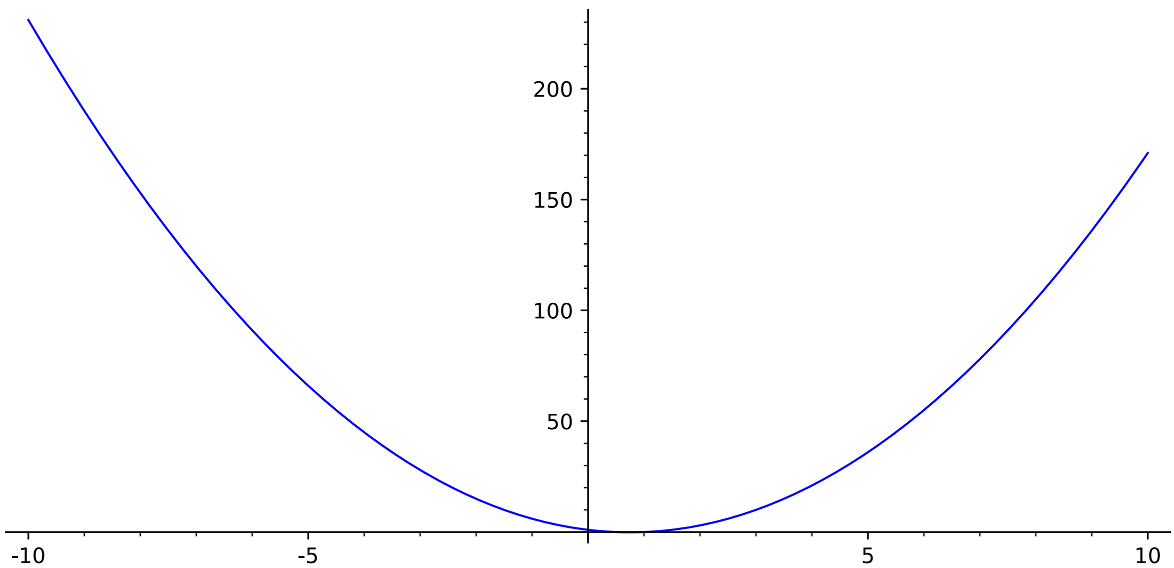
The Sage Reference guide to plotting is here: http://doc.sagemath.org/html/en/reference/plotting/index.html

```
x = var('x')
plot(x * sin(x), (x, -10, 10))
```



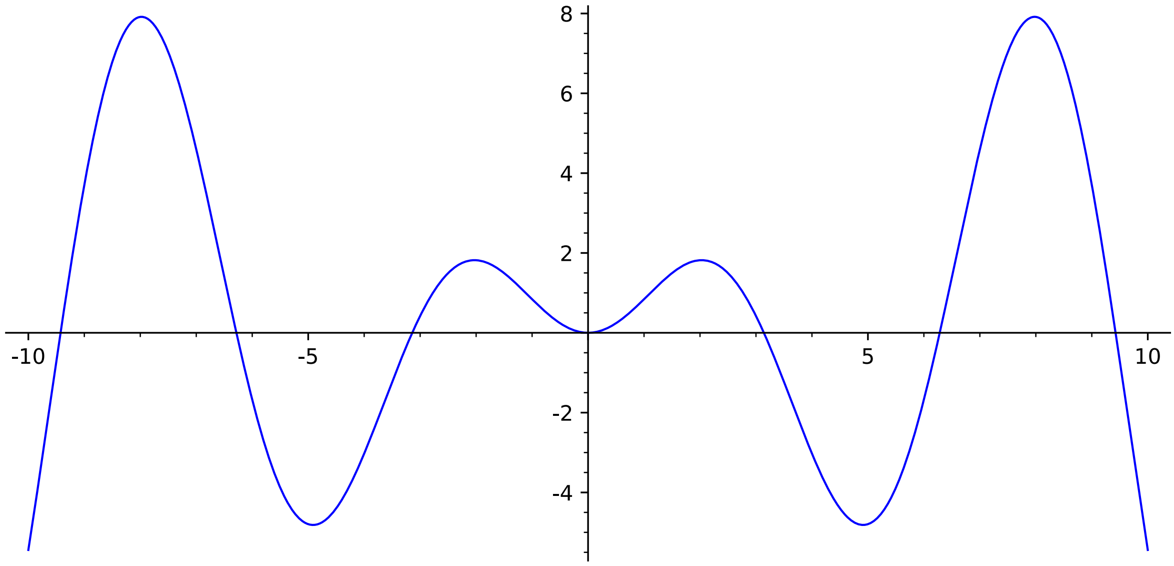
Plot of $f(x) = 2x^2 - 3x + 1$.

```
plot(2*x^2 - 3*x + 1, (x, -10, 10))
```



Plot of $f(x) = x\sin(x)$.

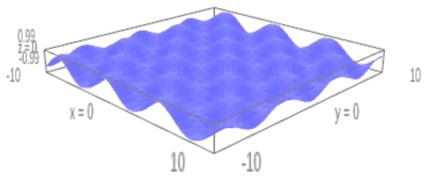
```
plot(x * sin(x), (x, -10, 10))
```



3D plot of $f(x, y) = \sin(x)\cos(y)$.

```
var('x y')
plot3d(sin(x) * cos(y), (x, -10, 10), (y, -10, 10))
```

(x, y)




```
# Reading and Writing Files
```

```
import csv
import sys

f = open('example.csv', 'rt')
try:
```

```
    reader = csv.reader(f)
    for row in reader:
        print row
finally:
    f.close()
```

['Title 1', 'Title 2', 'Title 3']
['1', 'a', '08/01/07']
['2', 'b', '08/02/07']
['3', 'c', '08/03/07']
['4', 'd', '08/04/07']
['5', 'e', '08/05/07']
['6', 'f', '08/06/07']
['7', 'g', '08/07/07']
['8', 'h', '08/08/07']
['9', 'i', '08/09/07']
['10', 'j', '08/10/07']

```
import csv
import sys

f = open('example.csv', 'wt')
try:
    writer = csv.writer(f)
    writer.writerow( ('Matrix', 'Example', '1') )
    for i in range(3):
        writer.writerow( (i+1,i,i-1 ))
finally:
    f.close()

print open('example.csv', 'rt').read()
```

Matrix,Example,1
1,0,-1
2,1,0
3,2,1