

# MAT 305: Mathematical Computing

## 2-D Graphing

John Perry

University of Southern Mississippi

Spring 2016

# Outline

- ➊ Basic 2-D objects
- ➋ Plotting functions
- ➌ Options for displaying plots
- ➍ Implicit plots
- ➎ Parametric and polar plots
- ➏ Saving images and animations
- ➐ Summary

# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary

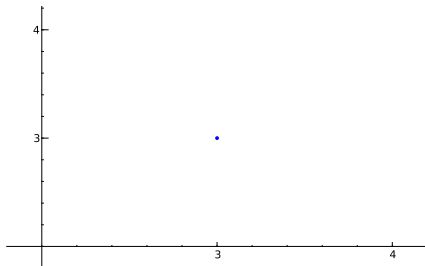
# The `point()` command

`point(( $x_0, y_0$ ), options)` where

- $(x_0, y_0)$  is a Python *tuple*
- *options* include
  - `pointsize`: size of point, default size is 10
  - `color`: rgb tuple
    - default color is blue
    - some names allowed: 'red', 'black', etc.
    - more on this later

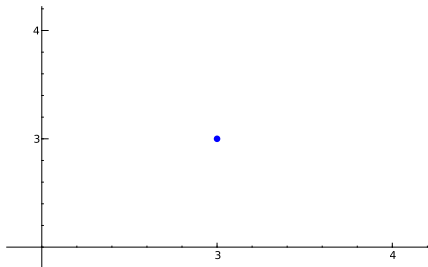
# Example

```
sage: point((3,3))
```



# Example

```
sage: point((3,3),pointsize=30)
```



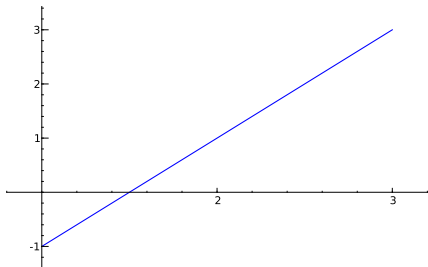
# The `line()` command

`line([(x1,y1), (x2,y2), ..., (xn,yn)], options)` where

- $(x_1, y_1)$  and  $(x_2, y_2)$  are Python *tuples*
- *options* include
  - **thickness** of curve (default is 1)
  - **linestyle**:
    - `'-'` (solid)
    - `'--'` (dashed)
    - `'-.'` (dash-dot)
    - `':'` (dots)
  - **color**

## Example

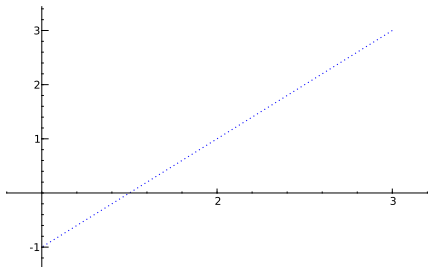
```
sage: line([(3,3),(1,-1)])
```





## Example

```
sage: line([(3,3),(1,-1)],linestyle=':')
```



# The `polygon()` command

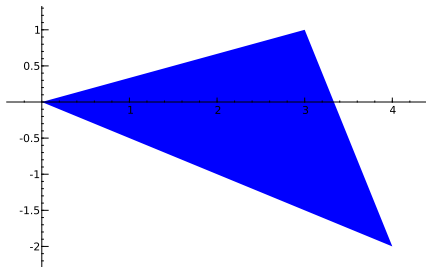
`polygon([(x1,y1), (x2,y2), ..., (xn,yn)], options)` where

- $(x_i, y_i)$  is a Python *tuple* representing a point of the polygon
- *options* include
  - **thickness** of lines (default is 1)
  - **alpha**: transparency of polygon
    - value from 0 to 1
    - 0: invisible; 1 opaque
  - **color**

The polygon will be filled. Don't want a filled polygon?  
combine lines instead. See below.

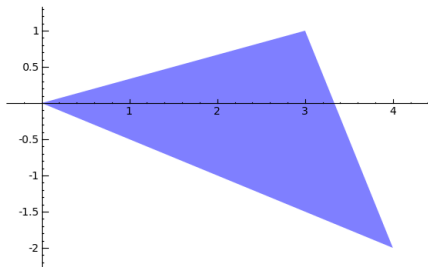
## Example

```
sage: polygon([(0,0),(3,1),(4,-2)])
```



## Example

```
sage: polygon([(0,0),(3,1),(4,-2)],alpha=0.5)
```



# The `text()` command

`text(message, (x0, y0), options)` where

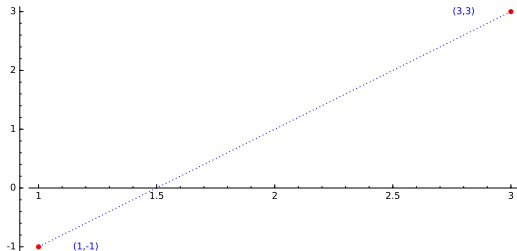
- *message* can be a number, function, or string
  - L<sup>A</sup>T<sub>E</sub>X strings allowed
- the text is centered over  $(x_0, y_0)$
- *options* include
  - `fontsize` controls text size (default 10)
  - `color`

# Combine plot objects with +

- Store graphics objects in memory using expressions
- Addition combines simple objects into complex objects
- Use `zorder` option to arrange depths

## Example

```
sage: point1 = point((3,3),pointsize=30,  
                    color='red',zorder=-10)  
sage: point2 = point((1,-1),pointsize=30,  
                    color='red',zorder=10)  
sage: my_line = line([(3,3),(1,-1)],linestyle=':')  
sage: my_label1 = text('(3,3)',(2.8,3))  
sage: my_label2 = text('(1,-1)',(1.2,-1))  
sage: point1 + point2 + my_line  
      + my_label1 + my_label2
```



# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary



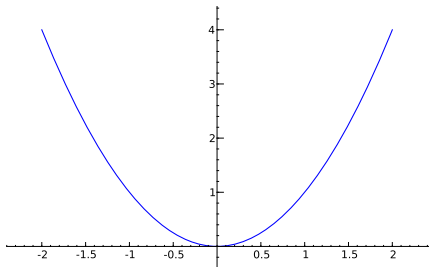
# The `plot()` command

`plot( $f(x)$ , options)` where

- $f(x)$  is an expression
- *options* include
  - `xmin`, `xmax` (*no* `ymin`, `ymax` options in `plot()`)
  - `plot_points`: minimal number of points connected
  - `fill`: to axis, min  $y$  value, max  $y$  value, a number  $c$ , or a function  $g(x)$
  - `fillcolor`
  - `color`
  - `thickness`
  - `linestyle`

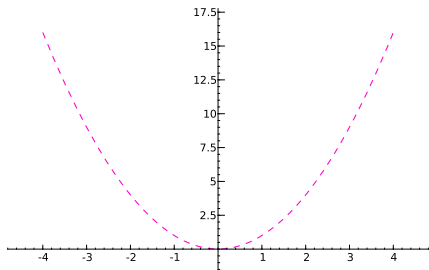
# Basic example

```
sage: plot(x**2, xmin=-2,xmax=2)
```



# Experiment with options!

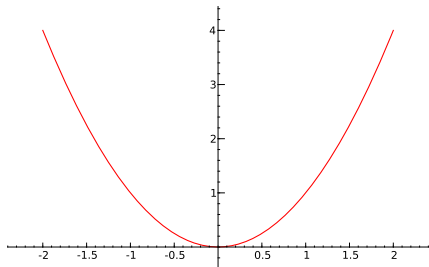
```
sage: plot(x**2, xmin=-4, xmax=4,  
          linestyle='--', color=(1,0,0.8))
```



# RGB colors

Specify colors using RGB tuples.

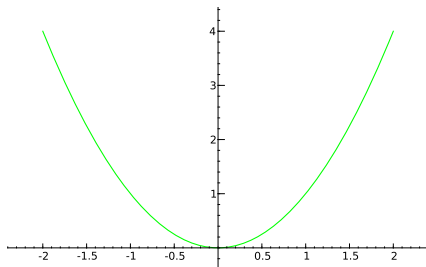
- Red, Green, Blue: primary colors of light
- Pure red:  $(1,0,0)$



# RGB colors

Specify colors using RGB tuples.

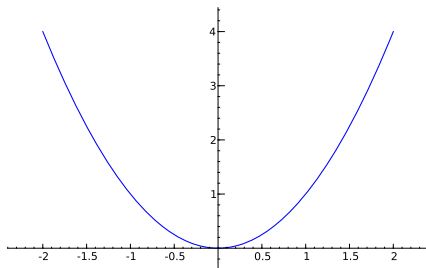
- Red, Green, Blue: primary colors of light
- Pure green: (0,1,0)



# RGB colors

Specify colors using RGB tuples.

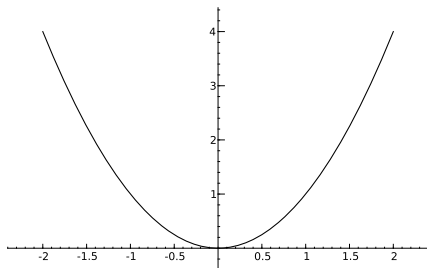
- Red, Green, Blue: primary colors of light
- Pure blue: (0,0,1)



# RGB colors

Specify colors using RGB tuples.

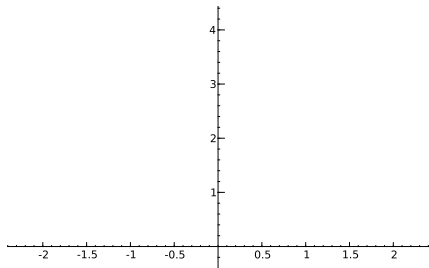
- Red, Green, Blue: primary colors of light
- Black is the absence of color:  $(0,0,0)$



# RGB colors

Specify colors using RGB tuples.

- Red, Green, Blue: primary colors of light
- White is the presence of all colors: (1,1,1)



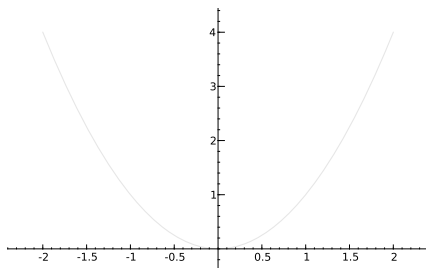
*(oops!)*



# RGB colors

Specify colors using RGB tuples.

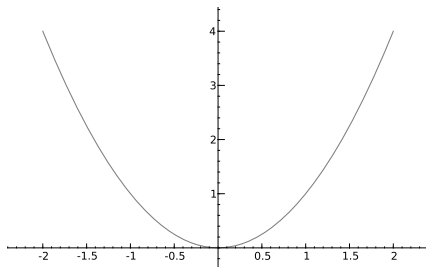
- Red, Green, Blue: primary colors of light
- Gray is an even mixture of the colors:  $(0.9, 0.9, 0.9)$



# RGB colors

Specify colors using RGB tuples.

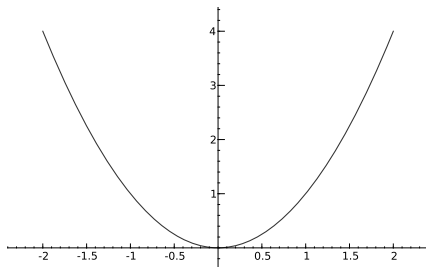
- Red, Green, Blue: primary colors of light
- Gray is an even mixture of the colors:  $(0.5, 0.5, 0.5)$



# RGB colors

Specify colors using RGB tuples.

- Red, Green, Blue: primary colors of light
- Gray is an even mixture of the colors:  $(0.2, 0.2, 0.2)$



# RGB colors

Specify colors using RGB tuples.

- Red, Green, Blue: primary colors of light
- What colors do these tuples represent?
  - $(0.8, 0.6, 0.2)$
  - $(0.9, 0.9, 0)$
  - $(0.3, 0.8, 0.9)$

# RGB colors

Specify colors using RGB tuples.

- Red, Green, Blue: primary colors of light
- What colors do these tuples represent?
  - $(0.8, 0.6, 0.2)$
  - $(0.9, 0.9, 0)$
  - $(0.3, 0.8, 0.9)$

brown

yellow

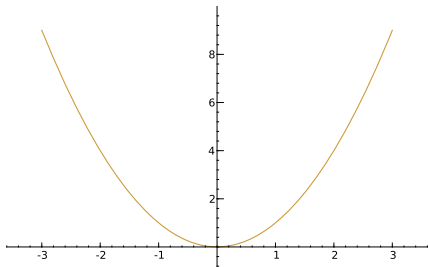
blue-green

## Remember colors

- Use expressions to remember colors

```
sage: my_brown = (0.8,0.6,0.2)
```

```
sage: plot(x**2,xmin=-3,xmax=3,color=my_brown)
```



# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots**
- 4 Implicit plots
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary

# The `show()` command

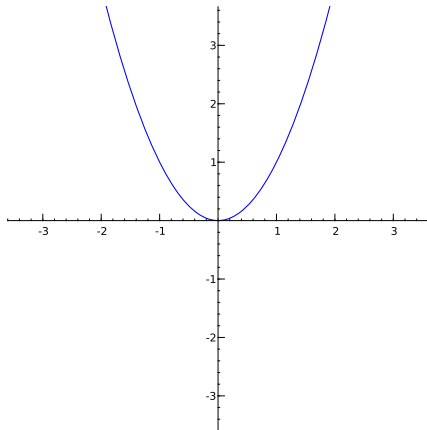
`show`(*plot object*, *options*) where

- *plot object* is any object generated by a plot
- *options* include
  - `aspect_ratio`: width/height
    - 1 makes a “square” graph
  - `xmin`, `xmax`
  - `ymin`, `ymax`



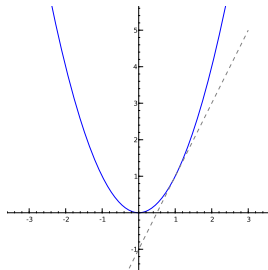
## Example

```
sage: myplot = plot(x**2,xmin=-3,xmax=3)
sage: show(myplot,ymin=-3,ymax=3,aspect_ratio=1)
```



## Combine plots

```
sage: par_plot = plot(x**2,xmin=-3,xmax=3)
sage: tan_plot = plot(2*x-1,xmin=-3,xmax=3,
                      color='gray',linestyle='--')
sage: com_plot = par_plot + tan_plot
sage: show(com_plot,ymin=-1,ymax=5,aspect_ratio=1)
```



# Different meanings of `xmin`, `xmax`

`plot`  $x$ -values for which Sage computes  $y$ -values

`show`  $x$ -values which Sage shown on the screen

Basic 2-D  
objects

Plotting  
functions

Options for  
displaying plots

Implicit plots

Parametric and  
polar plots

Saving images  
and animations

Summary

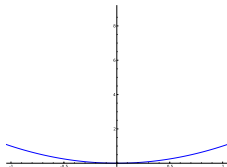
## Different meanings of `xmin`, `xmax`

`plot` `x`-values for which Sage computes `y`-values

`show` `x`-values which Sage shown on the screen

```
sage: p = plot(x**2, xmin=-3, xmax=3)
```

```
sage: show(p, xmin=-1, xmax=1)
```



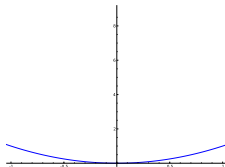
## Different meanings of `xmin`, `xmax`

`plot` x-values for which Sage computes  $y$ -values

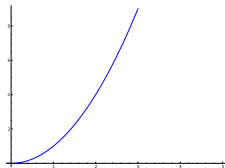
`show` x-values which Sage shown on the screen

```
sage: p = plot(x**2, xmin=-3, xmax=3)
```

```
sage: show(p, xmin=-1, xmax=1)
```



```
sage: show(p, xmin=0, xmax=5)
```



# The `animate()` command

`animate((plot1, plot2, ...), options)` where

- *plot1, plot2, ...* are graphics objects
  - each object = one frame
- *options* include
  - `xmin, xmax, ymin, ymax, aspect_ratio`

must show() animation, options:

- delay in hundredths of a second (default 20)
- iterations (0 = forever, default)
- cannot specify `xmin, xmax, ymin, ymax, aspect_ratio` for animation in `show()`; specify in `animate()` instead

## Example

```
sage: par_plot = plot(x**2,xmin=-3,xmax=3,
                      thickness=2,color=(0,0,0))
sage: tan_plot = plot(2*x-1,xmin=-3,xmax=3,thickness=2)
sage: com_plot = par_plot + tan_plot
sage: pink = (1.0,0.5,0.5)
sage: sec1_plot = plot(1,xmin=-3,xmax=3,
                      color=pink,linestyle='--')
sage: sec2_plot = plot(1/2*x+1/2,xmin=-3,xmax=3,
                      color=pink,linestyle='--')
sage: sec3_plot = plot(x,xmin=-3,xmax=3,
                      color=pink,linestyle='--')
sage: sec4_plot = plot(3/2*x-1/2,xmin=-3,xmax=3,
                      color=pink,linestyle='--')
sage: my_anim = animate((
    com_plot+sec1_plot, com_plot+sec2_plot,
    com_plot+sec3_plot, com_plot+sec4_plot,
    com_plot
))
sage: show(my_anim)
```

# The result



## Notes on xmin, xmax

- In `plot()`,  
xmin and xmax indicate  $x$  values to *compute*.
- In `show()` and `animate()`,  
xmin and xmax indicate  $x$  values to *display*.
- `plot(x^2,xmin=-3,xmax=3)` computes points on the  
interval  $[-3,3]$
- `show(my_plot,xmin=-1,xmax=1)` displays  $x \in [-1,1]$ ,  
*regardless of the  $x$  values computed in my\_plot*

# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots**
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary

# Implicit plots

- plot of an equation in  $x$  and  $y$ 
  - $x^2 + y^2 = 1$
  - might not be a function of  $x$ !
- Implicit plots handled by Python package `matplotlib`
- Behave differently than usual plots
  - look different
  - different options

# The `implicit_plot()` command

`implicit_plot( $f(x,y)$ , ( $x$ ,  $xmin$ ,  $xmax$ ), ( $y$ ,  $ymin$ ,  $ymax$ ),  
options)` where

- $f(x,y)$  a function of  $x$  and  $y$ 
  - graphs  $f(x,y) = 0$
  - variable  $y$  must be defined!
- *options* include
  - `plot_points`: number of points in each direction
  - `fill` (True or False): fill the region  $f(x,y) < 0$
  - `color` (*not* color)

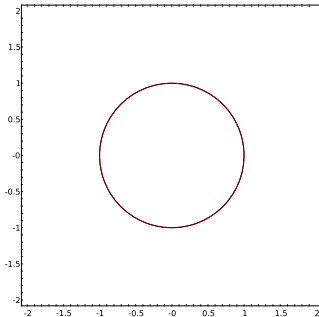
## Example

To plot a circle, rewrite the equation as  $f(x, y) = 0$ :

$$x^2 + y^2 = 1 \implies x^2 + y^2 - 1 = 0 \implies f(x, y) = x^2 + y^2 - 1$$

```
sage: icircle = implicit_plot(x**2 + y**2-1,
                             (x,-2,2),(y,-2,2),color='red')
```

```
sage: show(icircle,aspect_ratio=1)
```

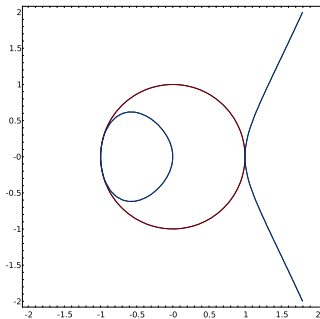


# Combining

Can combine implicit plots:

```
sage: ielliptic = implicit_plot(y**2-x**3+x,  
                                (x,-2,2),(y,-2,2),color='blue')
```

```
sage: show(icircle+ielliptic,aspect_ratio=1)
```



# Animating

Basic 2-D  
objects

Plotting  
functions

Options for  
displaying plots

Implicit plots

Parametric and  
polar plots

Saving images  
and animations

Summary

```
sage: elliptic1 = implicit_plot(y**2-x**3+x,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: elliptic2 = implicit_plot(y**2-x**3+0.75*x,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: elliptic3 = implicit_plot(y**2-x**3+0.5*x,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: elliptic4 = implicit_plot(y**2-x**3+0.25*x,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: elliptic5 = implicit_plot(y**2-x**3+0.1*x,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: elliptic6 = implicit_plot(y**2-x**3,
                                (x,-2,2),(y,-2,2),color='blue',plot_points=100)
sage: my_anim = animate([elliptic1, elliptic2, elliptic3,
                          elliptic4, elliptic5, elliptic6],
                          aspect_ratio=1)
sage: show(my_anim)
```

Basic 2-D  
objects

Plotting  
functions

Options for  
displaying plots

**Implicit plots**

Parametric and  
polar plots

Saving images  
and animations

Summary

# The result



# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots
- 5 Parametric and polar plots**
- 6 Saving images and animations
- 7 Summary

# Parametric equations

Form:

$$\begin{cases} x(t) &= \dots \\ y(t) &= \dots \end{cases}, \quad t \in [t_{\min}, t_{\max}]$$

## Example

A Bezier curve w/control pts  $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$

$$\begin{cases} x(t) &= x_0(1-t)^3 + x_1t(1-t)^2 + x_2t^2(1-t) + x_3t^3 \\ y(t) &= y_0(1-t)^3 + y_1t(1-t)^2 + y_2t^2(1-t) + y_3t^3 \end{cases}, \quad t \in [0, 1].$$

# The `parametric_plot()` command

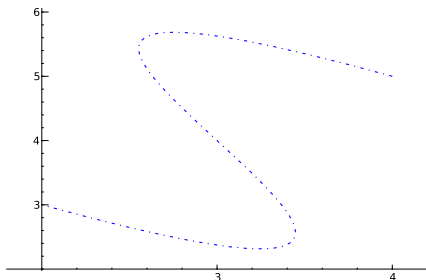
`parametric_plot(( $x(t), y(t)$ ), ( $tmin, tmax$ ), plot options)`

where

- $x(t), y(t)$  functions of  $t$
- don't forget to define  $t$  as a variable
- usual plot options apply

## Example Bezier Curve

```
sage: parametric_plot(  
      (2*t**3 + 6*3*t**2*(1 - t)  
        + 0*3*t*(1 - t)**2 + 4*(1 - t)**3,  
      3*t**3 + 0*3*t**2*(1 - t)  
        + 8*3*t*(1 - t)**2 + 5*(1 - t)**3),  
      (0,1),linestyle='-.')
```



# Polar plots

Form:  $r$  (radius) a function of  $\theta$  (angle)

## Example

A limaçon has the form

$$r = 1 + c \sin \theta.$$

It is difficult to describe this implicitly.

# The `polar_plot()` command

`polar_plot( $r(x)$ , options)` where

- $r(x)$  is a *polar* function of  $x$ 
  - $x$  stands in for  $\theta$
  - can define a variable  $\theta$  if you really want, but...
- usual plot options apply

## Example limaçon

```
sage: lim1 = polar_plot(1+2.5*sin(x),xmin=0,xmax=2*pi)
sage: lim2 = polar_plot(1+1.7*sin(x),xmin=0,xmax=2*pi)
sage: lim3 = polar_plot(1+sin(x),xmin=0,xmax=2*pi)
sage: lim4 = polar_plot(1+0.7*sin(x),xmin=0,xmax=2*pi)
sage: lim5 = polar_plot(1+0.5*sin(x),xmin=0,xmax=2*pi)
sage: lim6 = polar_plot(1+0*sin(x),xmin=0,xmax=2*pi)
sage: my_anim = animate([lim1,lim2,lim3,
                        lim4,lim5,lim6],
                        aspect_ratio=1,xmin=-2,xmax=2,
                        ymin=-1,ymax=4)
sage: show(my_anim)
```

Basic 2-D  
objects

Plotting  
functions

Options for  
displaying plots

Implicit plots

**Parametric and  
polar plots**

Saving images  
and animations

Summary

# The result



# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary

# The `.save()` command

Images and animations can be saved and used by other programs:

```
sage: my_plot = plot(x**2, -3, 3)
```

```
sage: my_plot.save('a_parabola.pdf')
```

# The `.save()` command

Images and animations can be saved and used by other programs:

```
sage: my_plot = plot(x**2, -3, 3)
```

```
sage: my_plot.save('a_parabola.pdf')
```

Can now

- open `a_parabola.pdf` in PDF reader (Adobe's Reader, Apple's Preview, Gnome's `evince`, KDE's `okular`, etc)
- import `a_parabola.pdf` into word processor (Microsoft Office, OpenOffice, Lyx, etc)
- submit `a_parabola.pdf` to instructor

# Image formats

## Different formats available

- bitmap
  - colored dots
  - does not scale well
  - gif, jpg, png
- vector
  - mathematical instructions
  - scales to any size
  - eps, pdf, svg

## Example

① When viewing previous image, zoom in & out

- quality does not degenerate

② Now try this:

```
sage: my_plot.save('a_parabola.png')
```

- open in viewer, zoom in
- notice blockiness

# Saving animation

- can save only as gif's  
`sage: my_anim.save('limacon.gif')`
- open in Firefox or other web browser

# Outline

- 1 Basic 2-D objects
- 2 Plotting functions
- 3 Options for displaying plots
- 4 Implicit plots
- 5 Parametric and polar plots
- 6 Saving images and animations
- 7 Summary

# Summary

- Sage offers many commands for plotting 2-D objects
  - points, lines
  - functions
  - equations: implicit, parametric, polar
- Most options work for all objects
- Combine objects by “adding” them together
- Animate using a list of objects
- save images