MAT 305: Mathematical Computing

2-D Graphing

John Perry

University of Southern Mississippi

Spring 2017

Outline

Outline

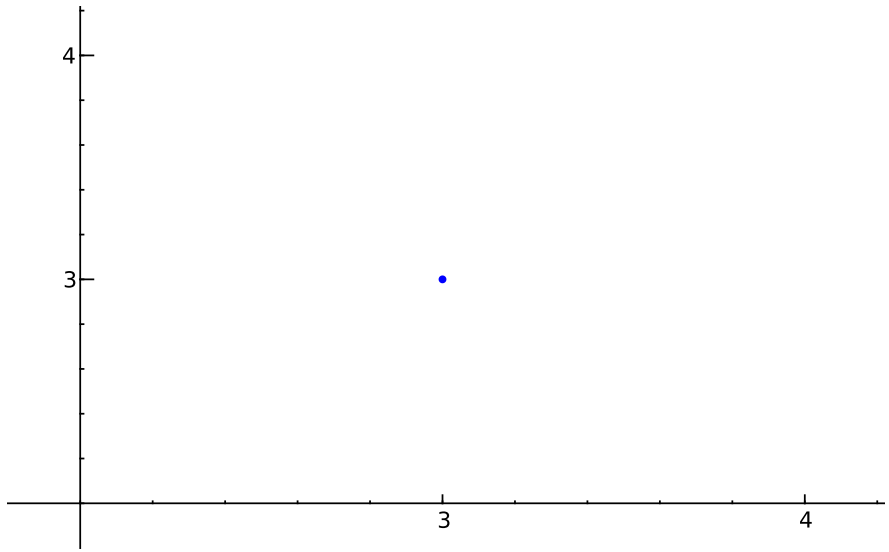
The `point()` command

`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

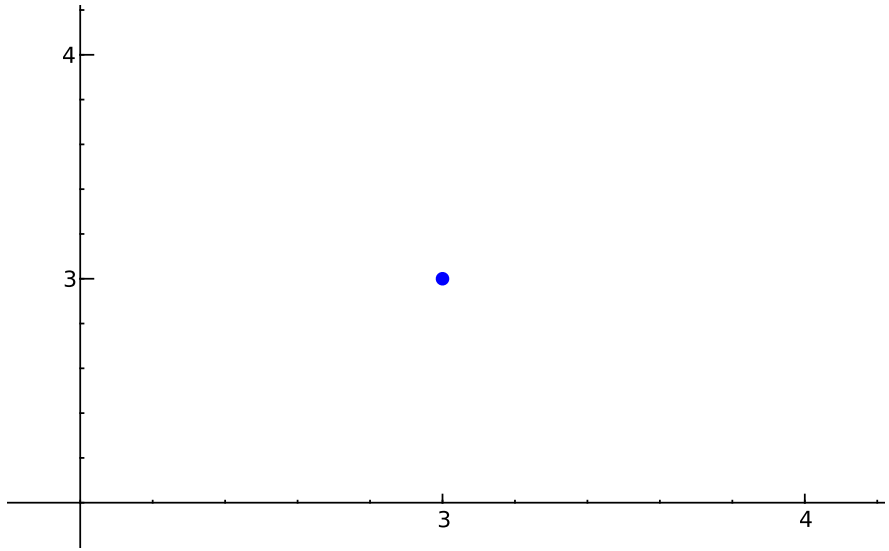
Example

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



Example

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



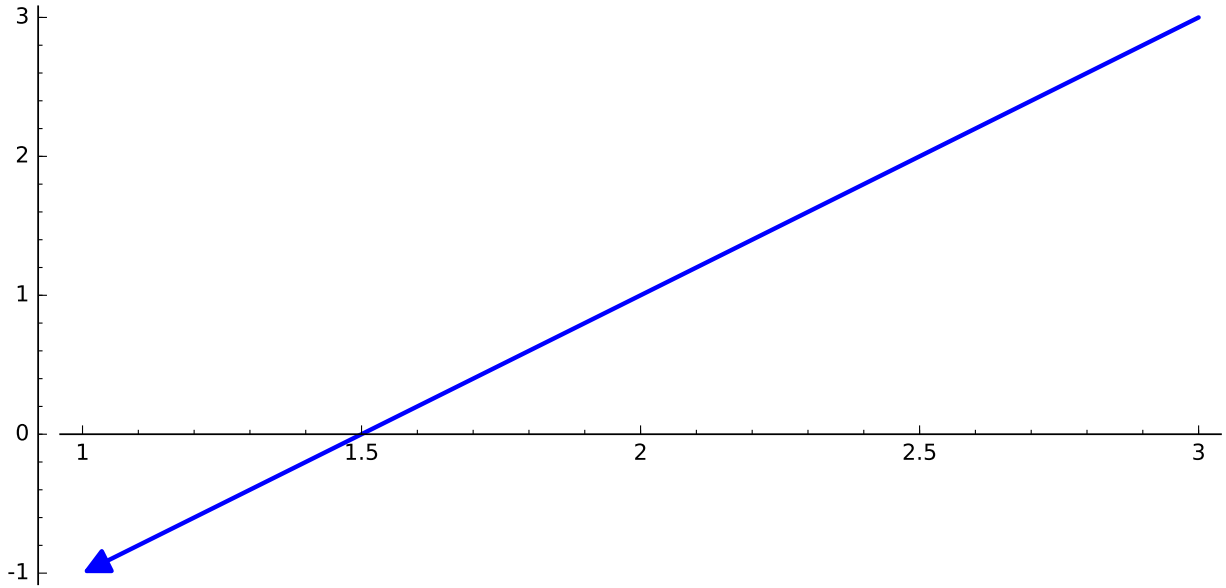
The `arrow()` command

`arrow(`$((x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n))$, *options*`)` where

- (x_i,y_i) are Python *tuples*
- *options* include
  - `arrowsize` (default is 5)
  - `width` of stem (default is 2)
  - `head`: location of arrowhead
    - `0` (at tailpoint)
    - `1` (at headpoint)
    - `2` (both endpoints)

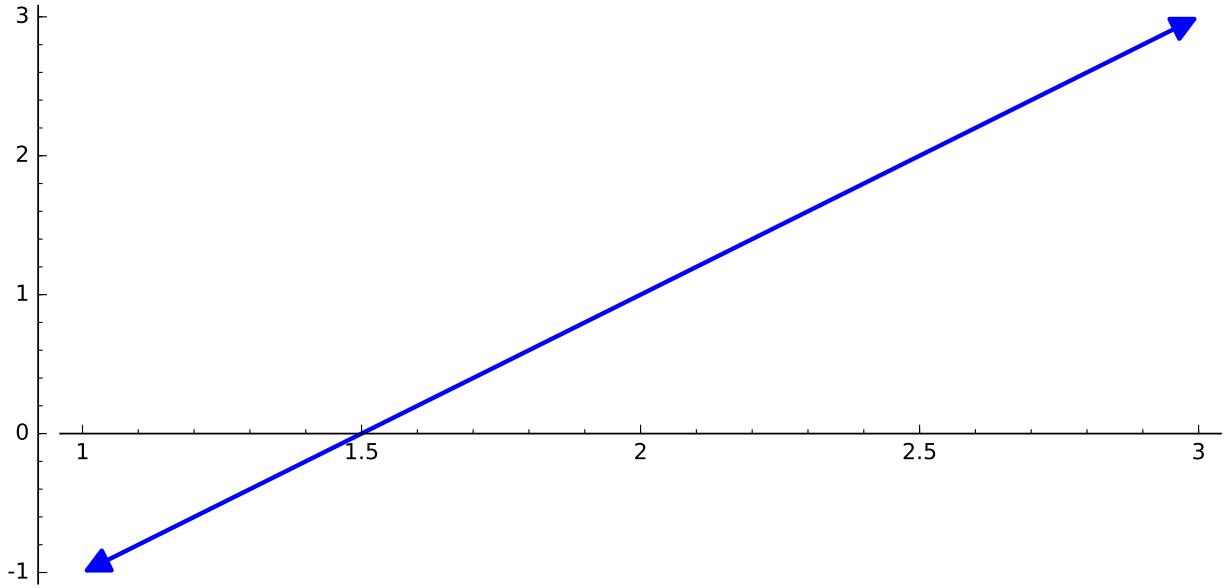
Example

```
sage: arrow(((3,3),(1,-1),(3,3)))
```



Example

```
sage: arrow(((3,3),(1,-1),(3,3)),head=2)
```



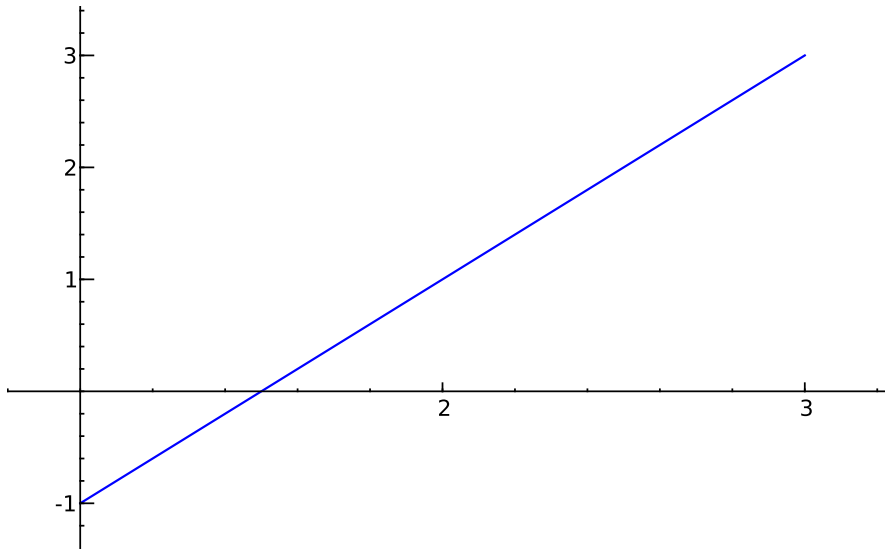
The `line()` command

`line(`$((x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n))$, *options*`)` where

- (x_i,y_i) are Python *tuples*
- *options* include
  - `thickness` of curve (default is 1)

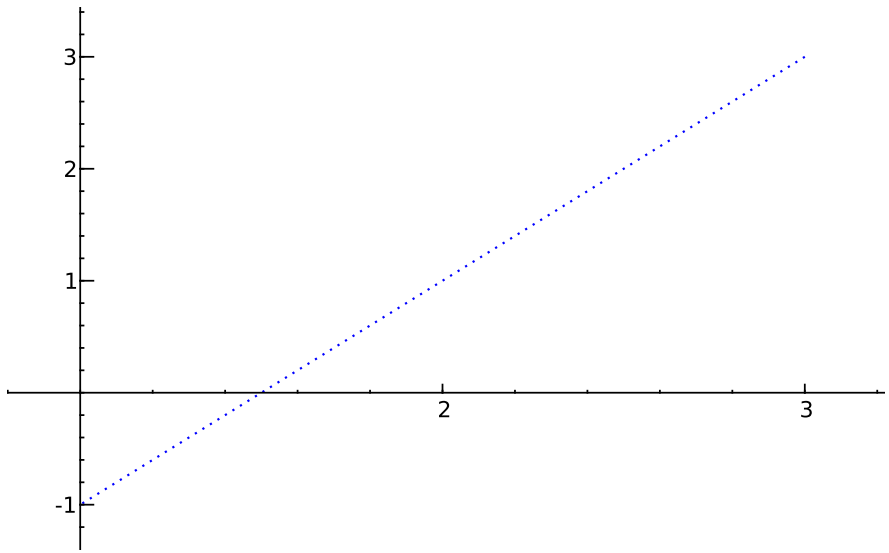
Example

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



Example

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



The `polygon()` command

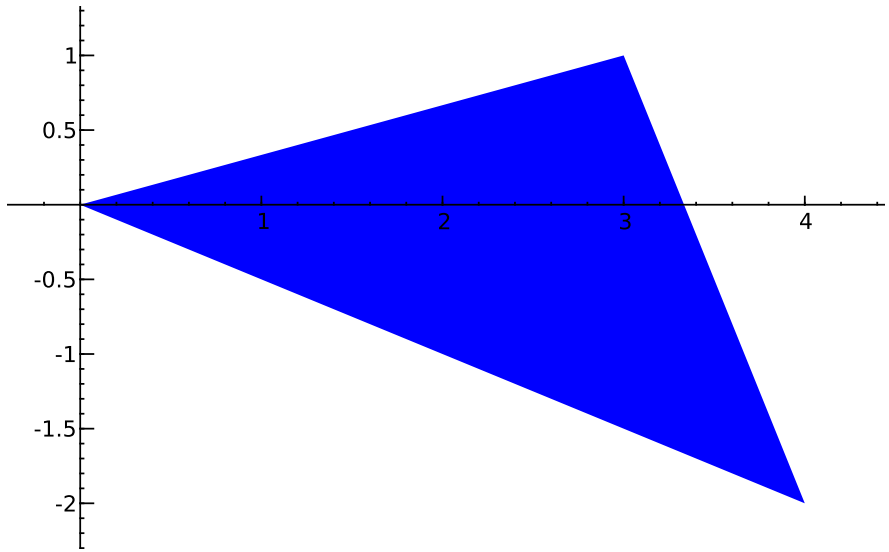
`polygon(`$[(x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)]$, *options*`)` where

- (x_i,y_i) is a Python *tuple* representing a point of the polygon
- *options* include
  - `thickness` of lines (default is 1)

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

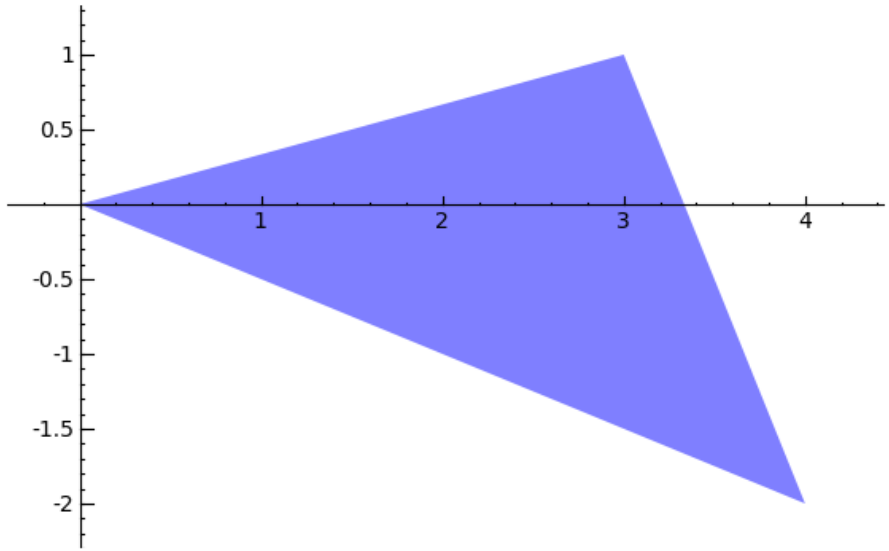
Example

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



Example

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



Common to arrows, lines, polygons

- `linestyle`:
  - `'solid'` (solid)
  - `'dashed'` (dashed)
  - `'dashdot'` (dash-dot)
  - `'dotted'` (dots)
- `alpha`: transparency of polygon
  - value from `0` to 1
  - `0`: invisible; 1 opaque
- `color`: (r, g, b) tuple
  - default color is blue
  - some names allowed: `‘red‘`, `‘black‘`, etc.
  - more on this later

The `text()` command

`text(`*message*`,` (x_0, y_0)`,` *options*`)` where

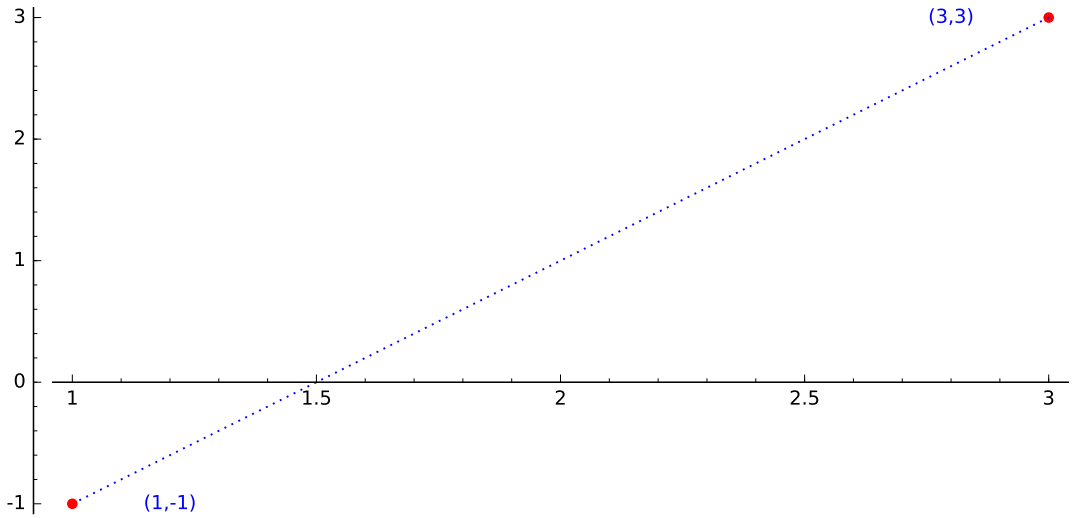
- *message* can be a number, function, or string
  - LaTeX strings allowed
- the text is centered over (x_0, y_0)
- *options* include
  - `fontsize` controls text size (default 10)
  - `color`

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

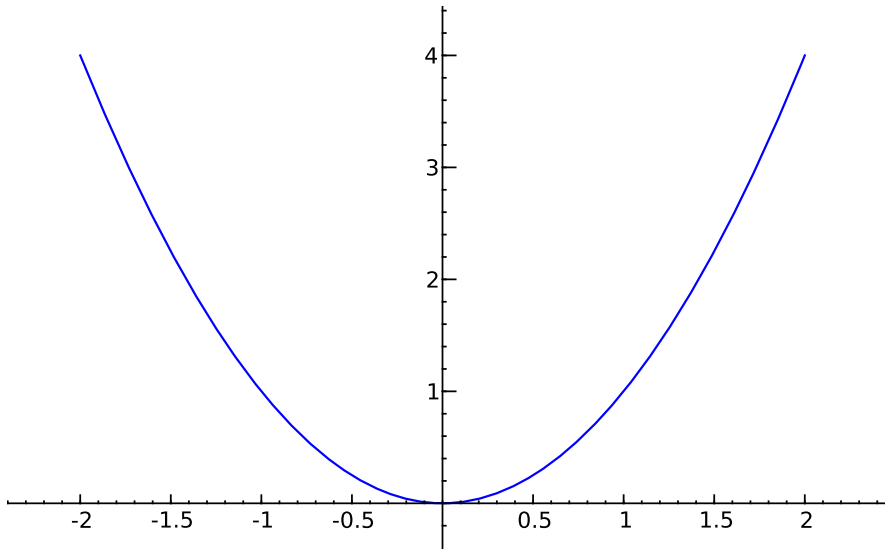
The `plot()` command

`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`

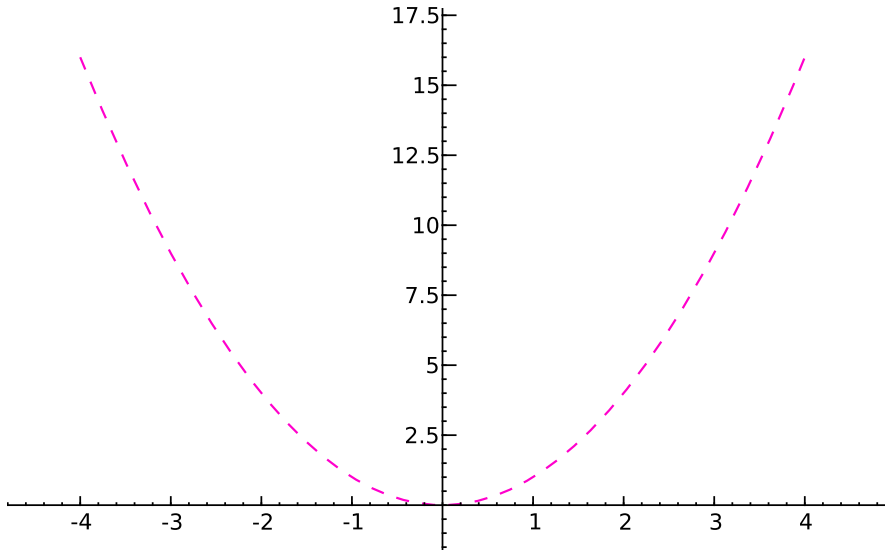
Basic example

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



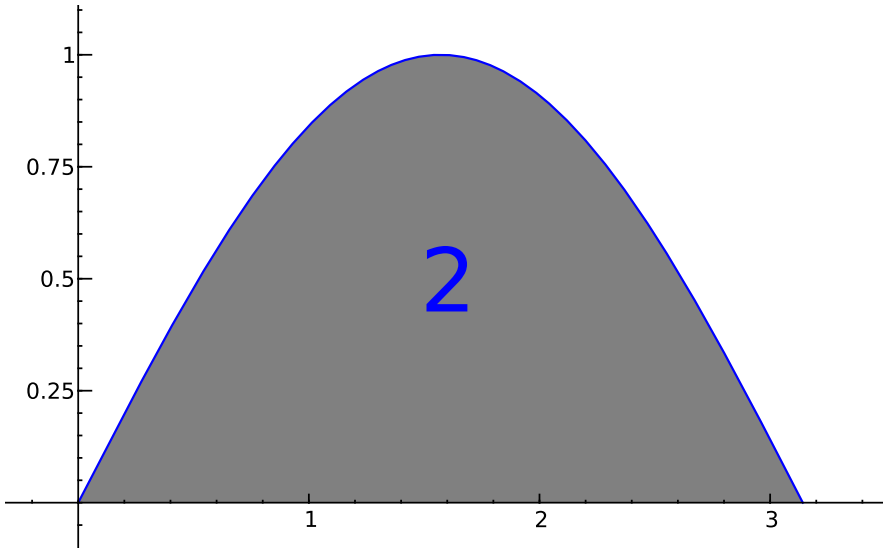
Experiment with options!

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



1 picture $\stackrel{?}{=}$ 1000 words

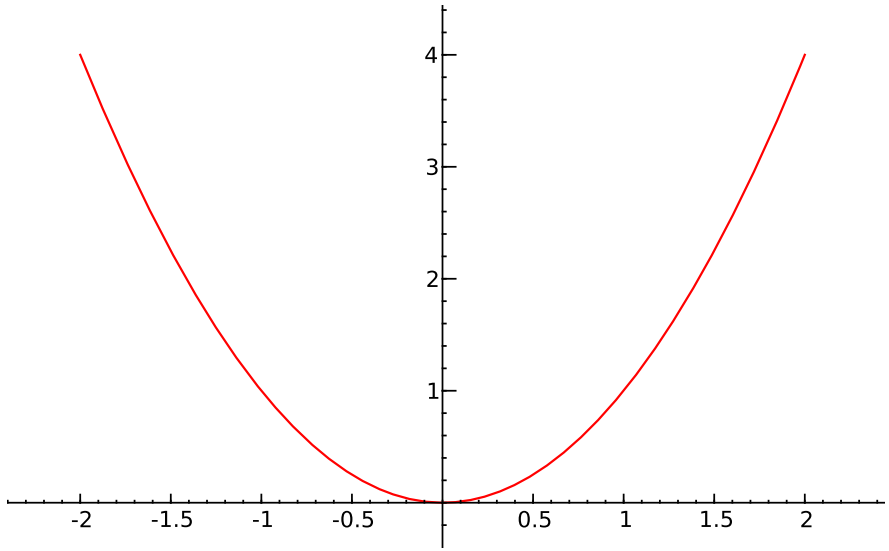
```
sage:  fplot = plot(sin(x),xmin=0,xmax=pi,
                    fill='axis')

sage:  farea = integral(sin(x),x,0,pi)

sage:  areatext = text(farea,(pi/2,0.5),fontsize=40)

sage:  fplot+areatext
```



RGB colors

Specify colors using RGB tuples.

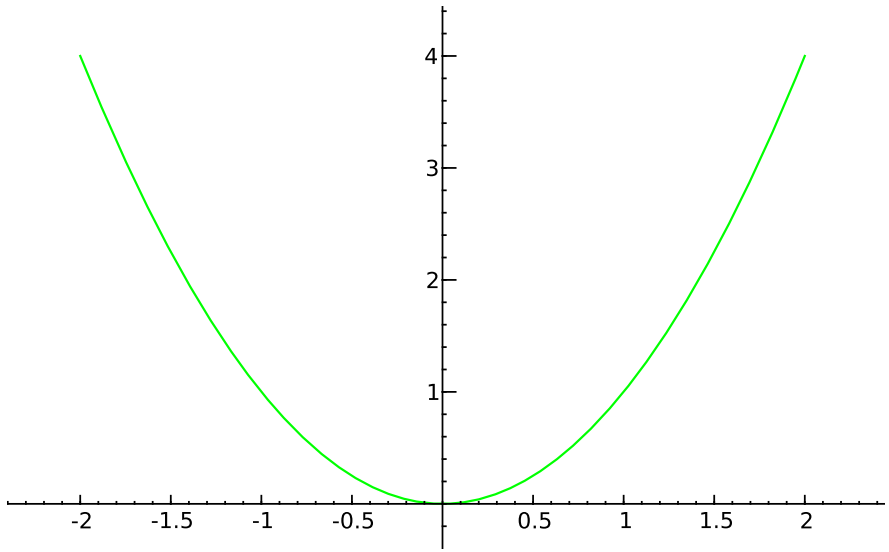
- Red, Green, Blue: primary colors of light
- Pure red: (1,0,0)



RGB colors

Specify colors using RGB tuples.

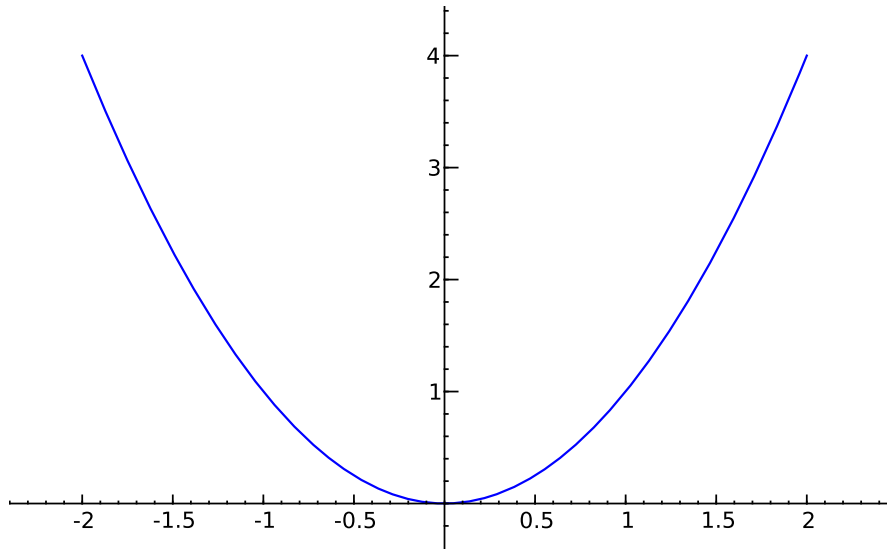
- Red, Green, Blue: primary colors of light
- Pure green: (0,1,0)



RGB colors

Specify colors using RGB tuples.

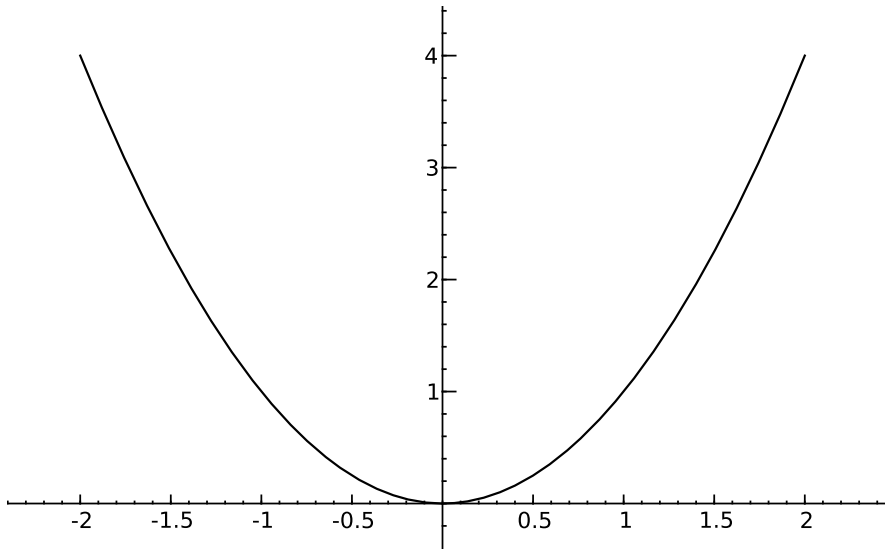
- Red, Green, Blue: primary colors of light
- Pure blue: (0,0,1)



RGB colors

Specify colors using RGB tuples.

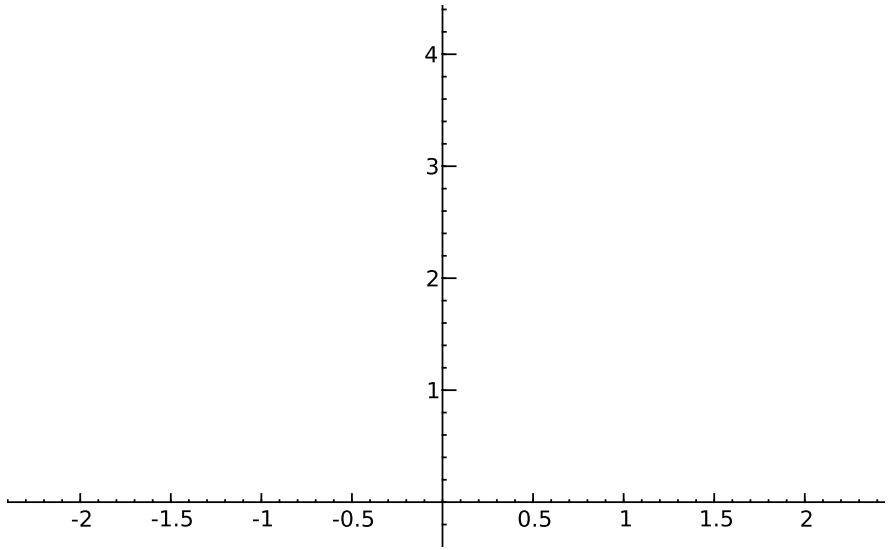
- Red, Green, Blue: primary colors of light
- Black is the absence of color: (0,0,0)



RGB colors

Specify colors using RGB tuples.

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

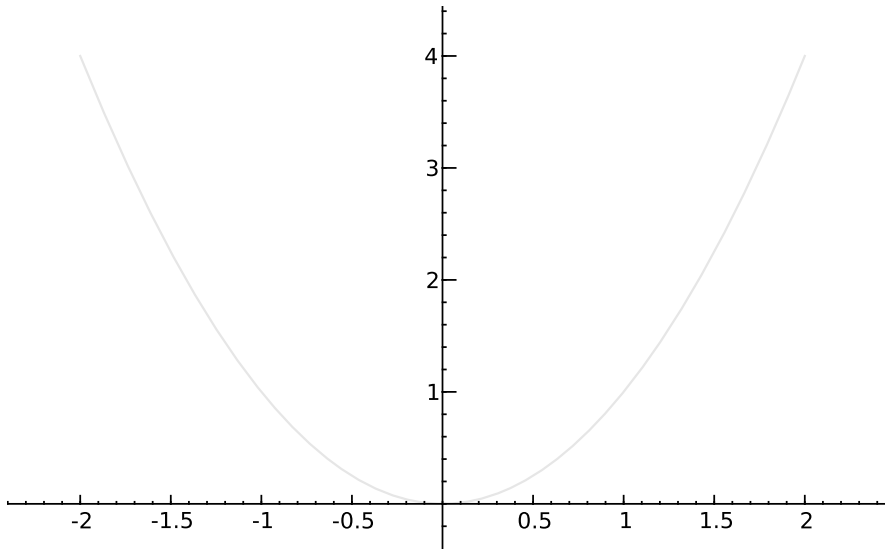


(oops!)

RGB colors

Specify colors using RGB tuples.

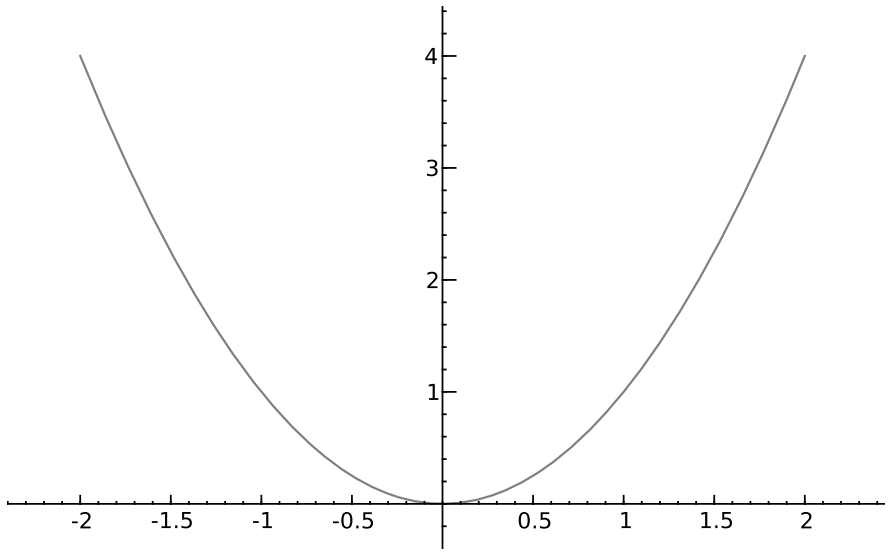
- Red, Green, Blue: primary colors of light
- Gray is an even mixture of the colors: (0.9,0.9,0.9)



RGB colors

Specify colors using RGB tuples.

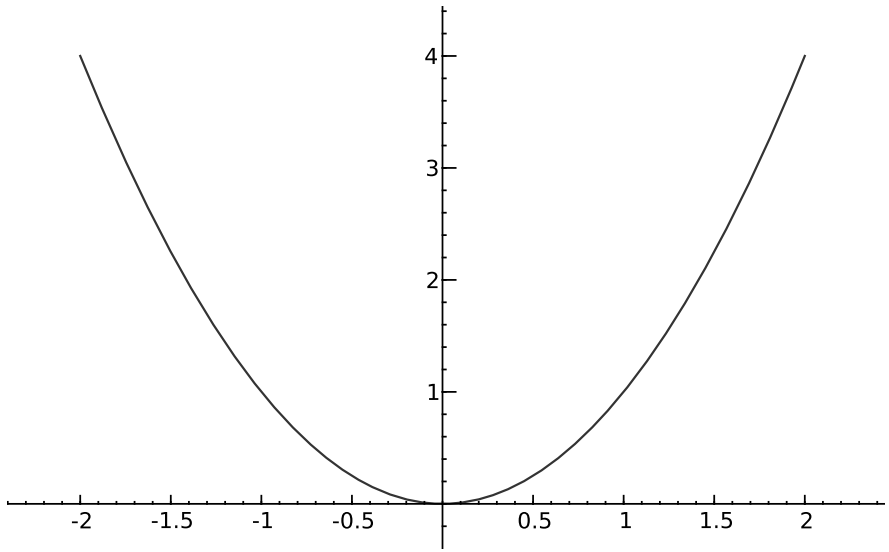
- Red, Green, Blue: primary colors of light
- Gray is an even mixture of the colors: (0.5,0.5,0.5)



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

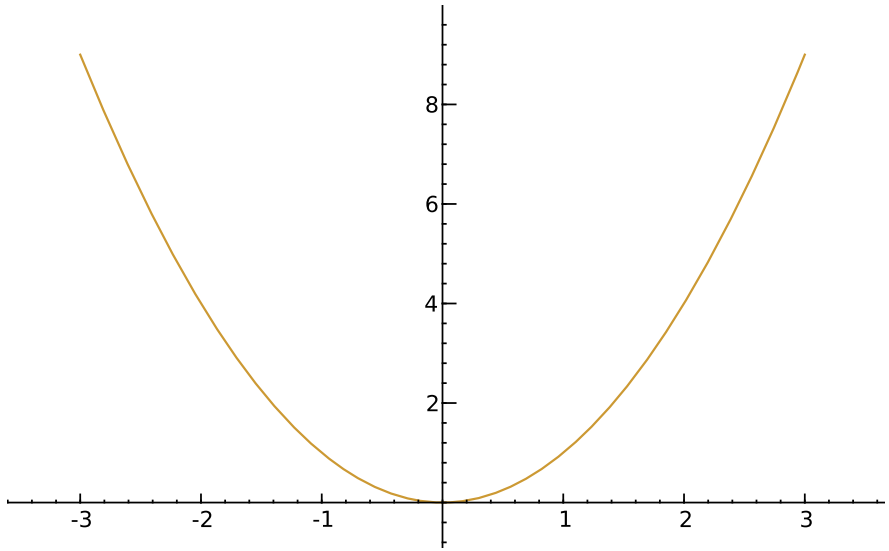
Specify colors using RGB tuples.

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

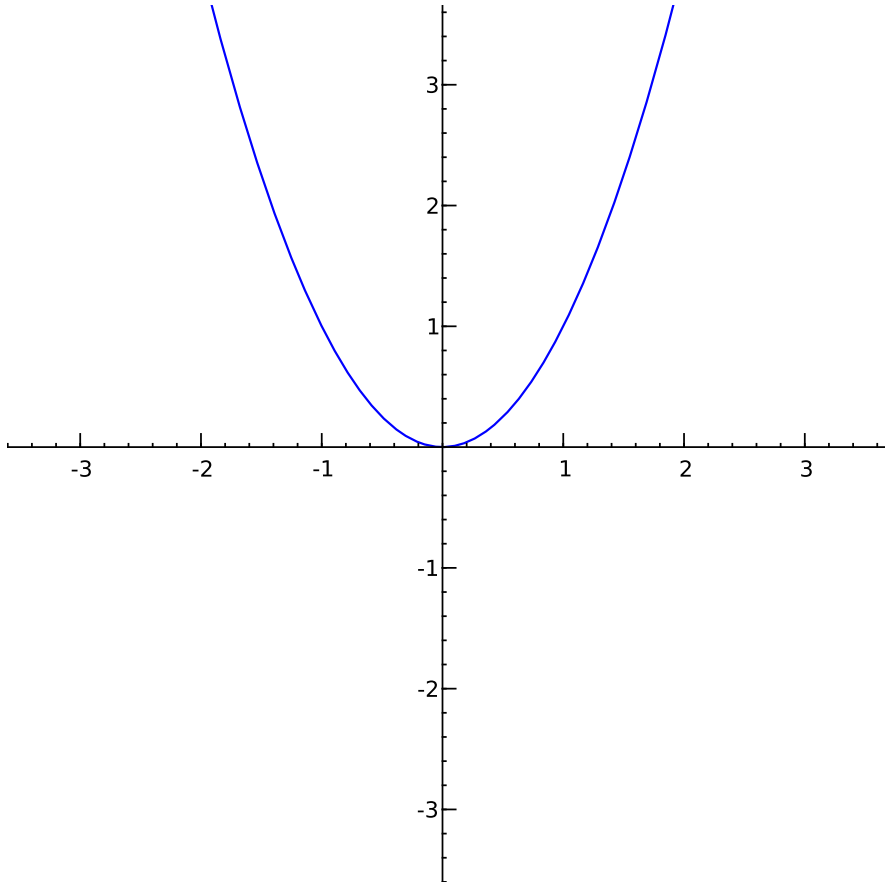
The `show()` command

`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

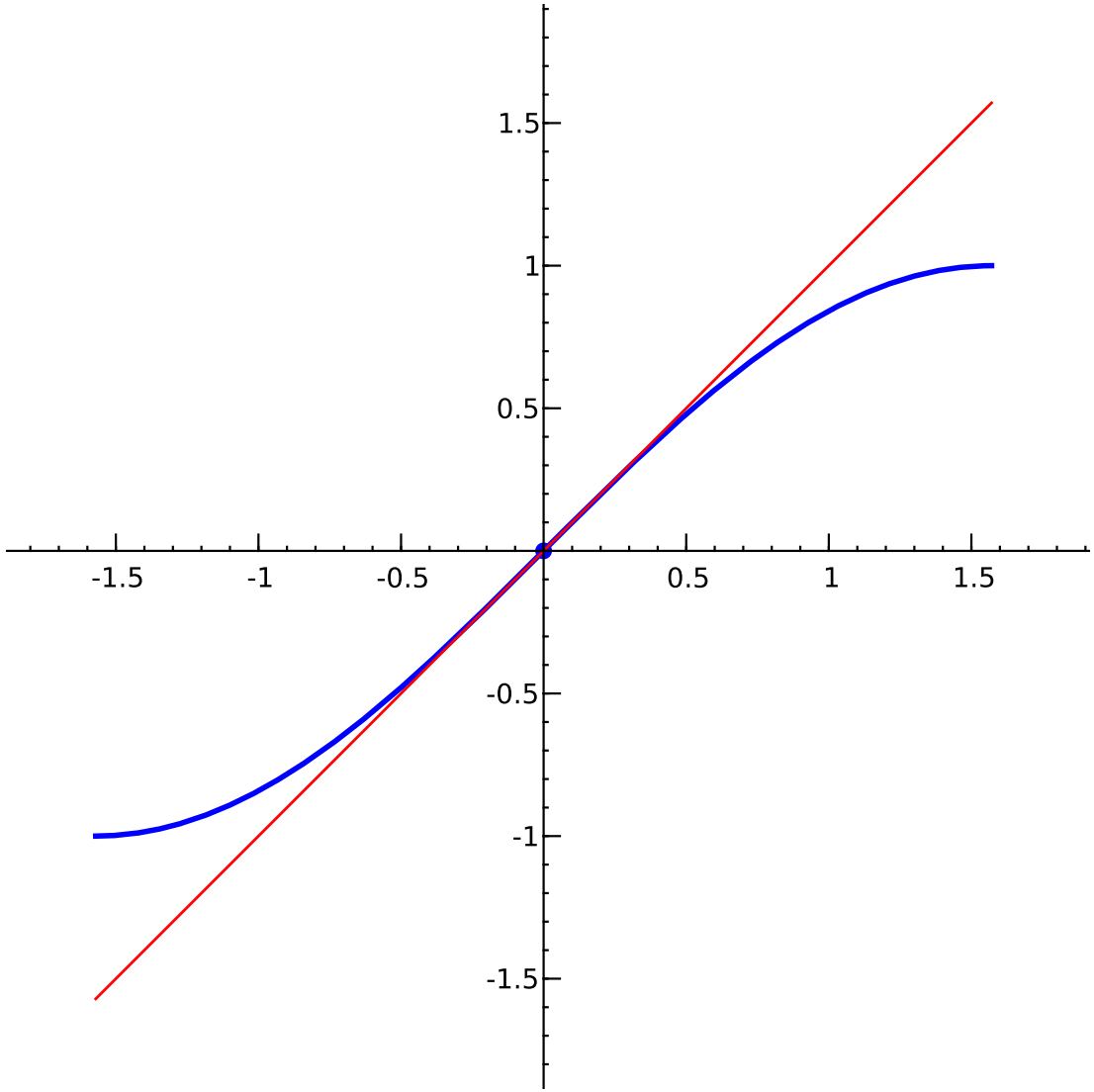
```
sage: myplot = plot(x**2,xmin=-3,xmax=3)
sage: show(myplot,ymin=-3,ymax=3,aspect_ratio=1)
```



Calculus: 1 picture = 1000 words

```
sage:  f(x) = sin(x)

sage:  df(x) = diff(f)

sage:  m0 = df(0)

sage:  p0 = point((0,f(0)),pointsize=30)

sage:  fplot = plot(f,xmin=-pi/2,xmax=pi/2,
                    thickness=2)

sage:  tan_line = plot(m0*(x-0)+f(0),xmin=-pi/2,
                       xmax=pi/2,rgbcolor=(1,0,0))

sage:  show(p0+fplot+tan_line,aspect_ratio=1)
```

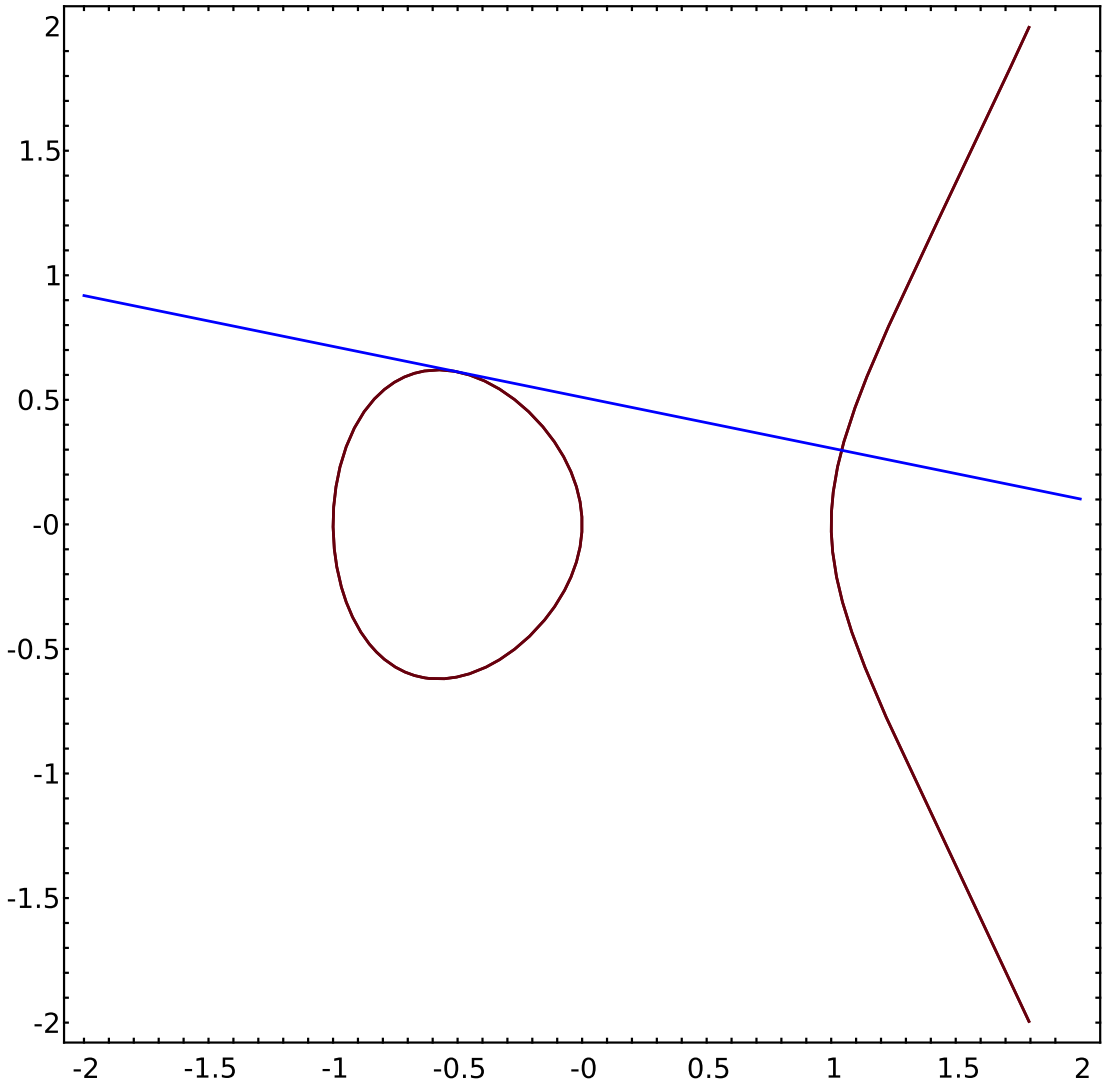
Calculus: 1 picture = 1000 words



Calculus: 2 pictures = 2000 words

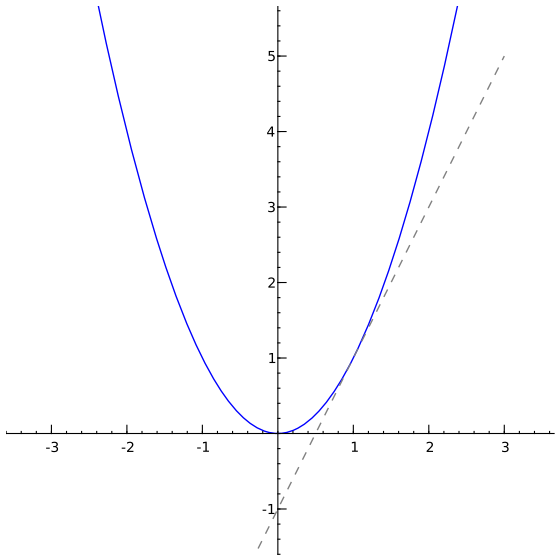
```
sage:  yvar = var('y')
sage:  yf = function('y',x)
sage:  yprime = diff(yf)
sage:  f = yf**2 - x**3 + x
sage:  df = diff(f)
sage:  solve(df,yprime)
[D[0](y)(x) == 1/2*(3*x^2 - 1)/y(x)]
sage:  mx(x,yvar) = 1/2*(3*x**2 - 1)/yvar
sage:  fplot = implicit_plot(yvar**2-x**3+x,
                 (x,-2,2),(yvar,-2,2),color='red')
sage:  tan_line = plot(mx(-0.5,sqrt(3/8))
                          *(x+0.5)+sqrt(3/8),
                          xmin=-2,xmax=2)
sage:  show(fplot+tan_line,aspect_ratio=1)
```

Calculus: 2 picture = 2000 words



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

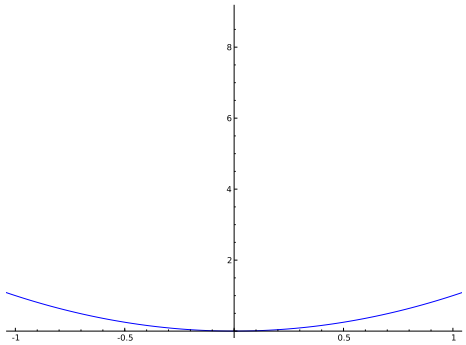
`show` x-values which Sage shown on the screen

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)
```

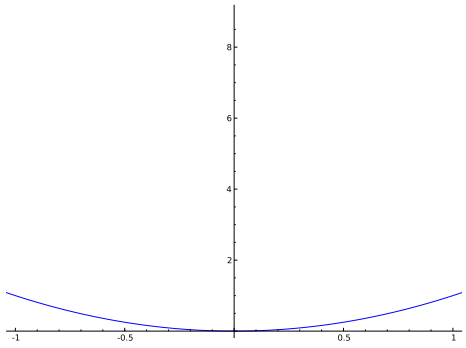


Different meanings of `xmin`, `xmax`

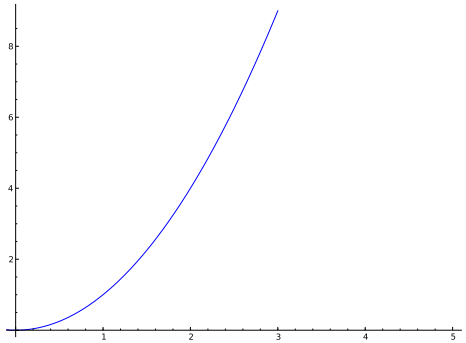
`plot` x-values for which Sage computes y-values

`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`

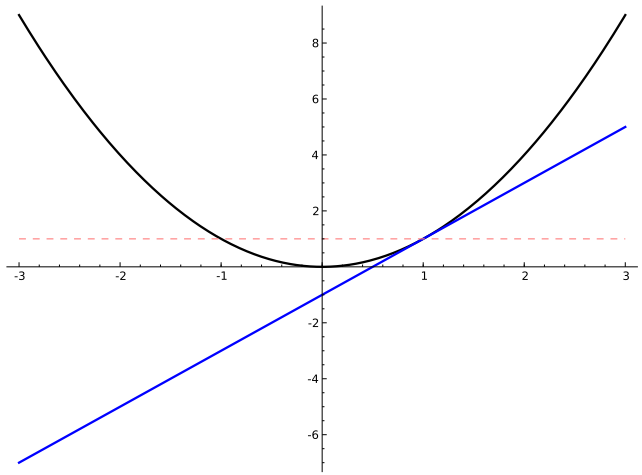
must `show()` animation, options:

- `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

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

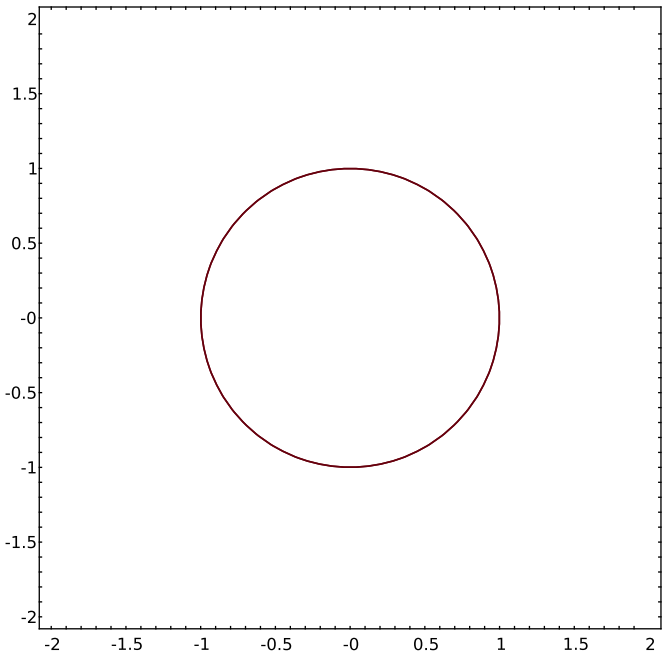
`implicit_plot(` $f(x,y)$, `(x,` *xmin*, *xmax*`)`, `(y,` *ymin*, *ymax*`)`, *options*`)` where

- $f(x,y)$ 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`

Example

Unit circle $x^2 + y^2 = 1$

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

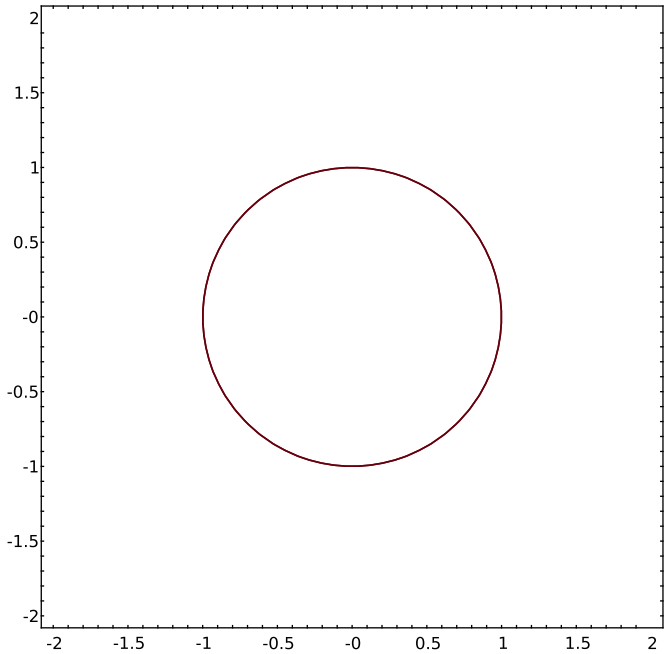


Example

Alternately, rewrite as $f(x,y)=0$:

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

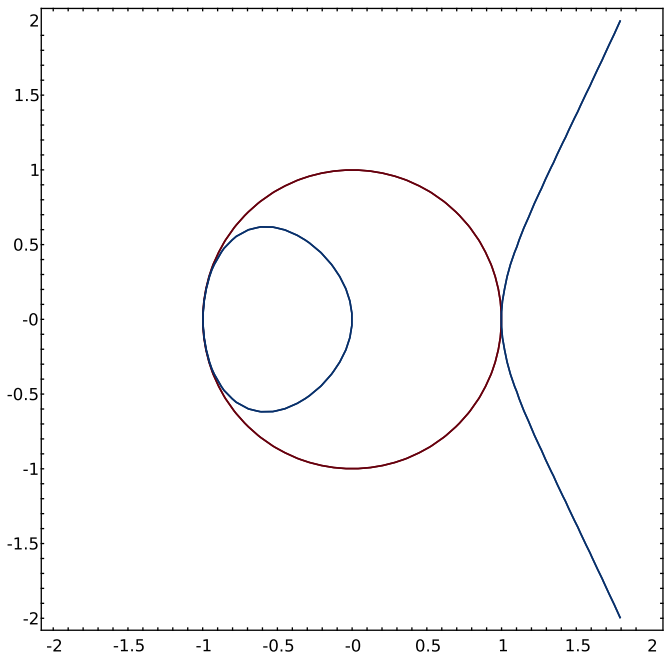
```
sage: icircle = implicit_plot(x**2 + y**2 - 1,
                              (x,-2,2),(y,-2,2),color='red')
sage: show(icircle,aspect_ratio=1)
```



Combining

Can combine implicit plots:

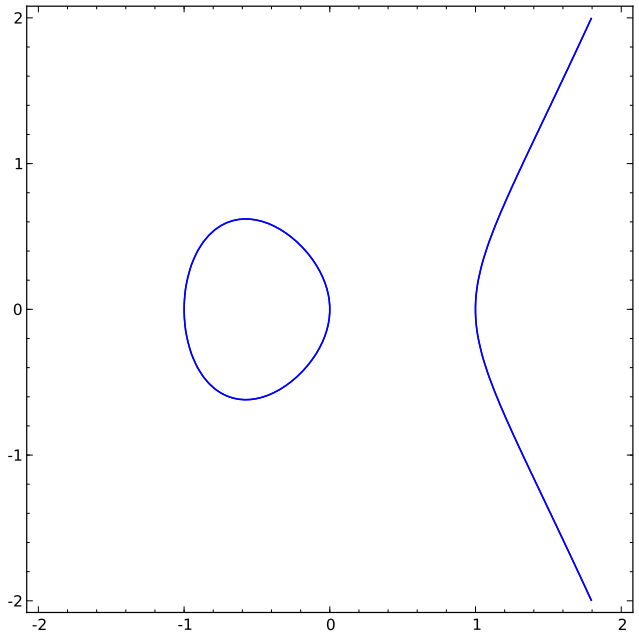
```
sage:  ielliptic = implicit_plot(y**2-x**3+x,
                    (x,-2,2),(y,-2,2),color='blue')
sage:  show(icircle+ielliptic,aspect_ratio=1)
```



Animating

```
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)
```

The result



Outline

Parametric equations

Form:

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

Example

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

$$\begin{cases} x(t) &= x_0(1-t)^3 + x_1 t(1-t)^2 + x_2 t^2(1-t) + x_3 t^3 \\ y(t) &= y_0(1-t)^3 + y_1 t(1-t)^2 + y_2 t^2(1-t) + y_3 t^3 \end{cases}, \; t \in [0,1].$$

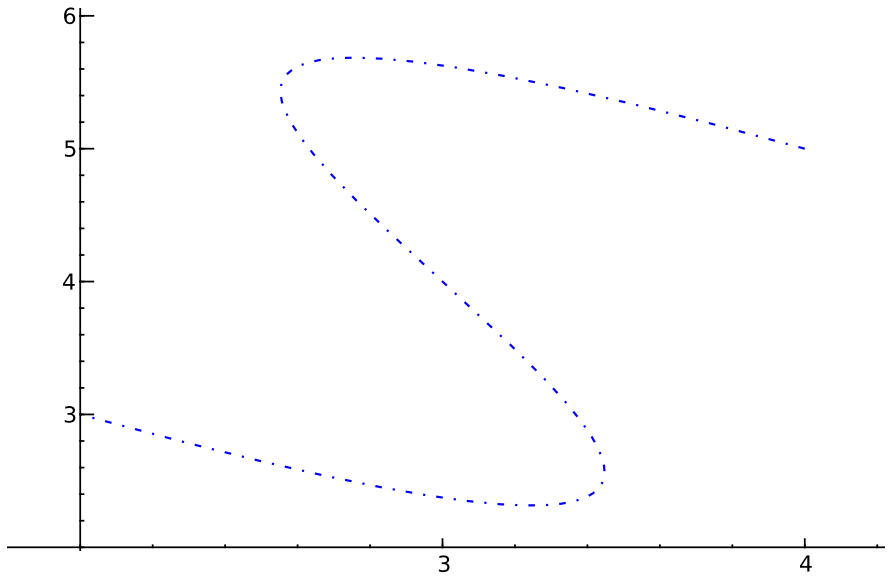
The `parametric_plot()` command

`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 θ (angle)

Example

A limaçon has the form

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

It is difficult to describe this implicitly.

The `polar_plot()` command

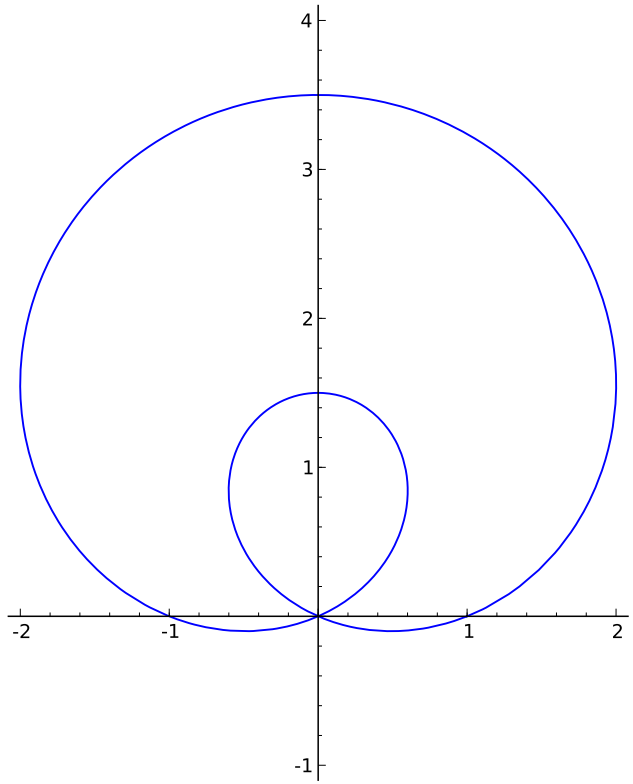
`polar_plot(`$r(x)$, *options*`)` where

- $r(x)$ is a *polar* function of x
  - x stands in for θ
  - can define a variable θ 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)
```

The result



Outline

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

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