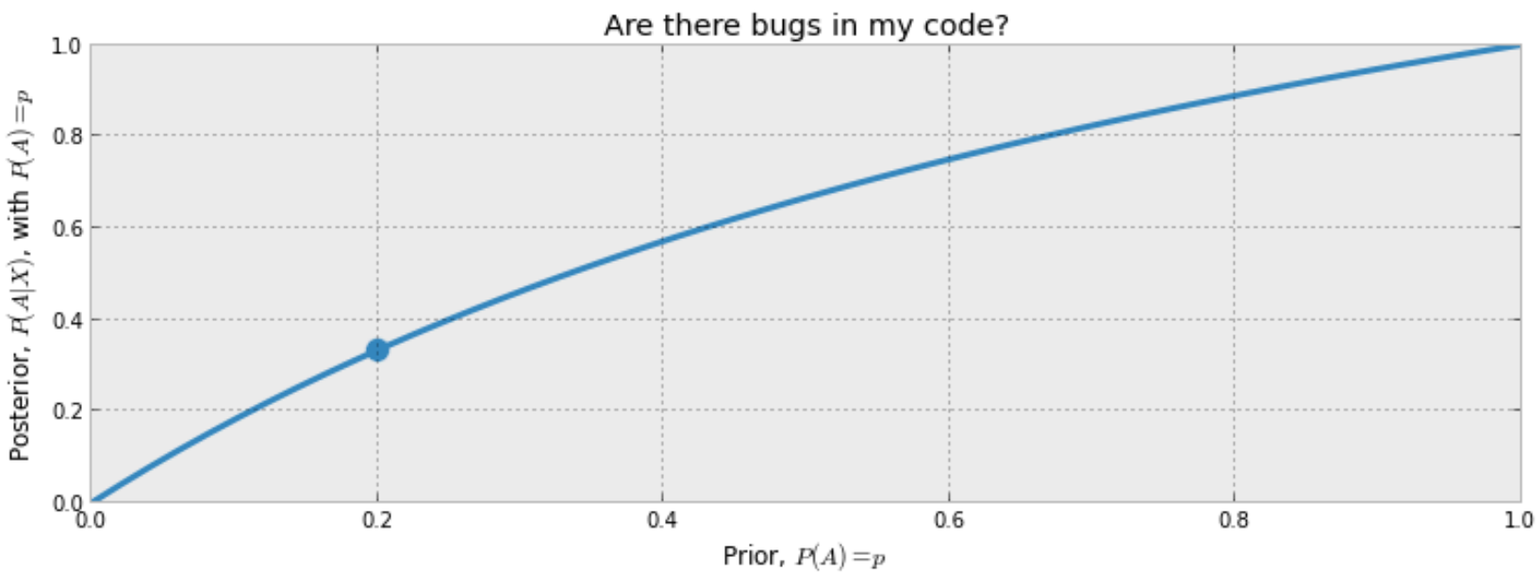
IPython is Great

(for large-scale computation, data exploration, and creating reproducible research artifacts)

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)}$$
$$= \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

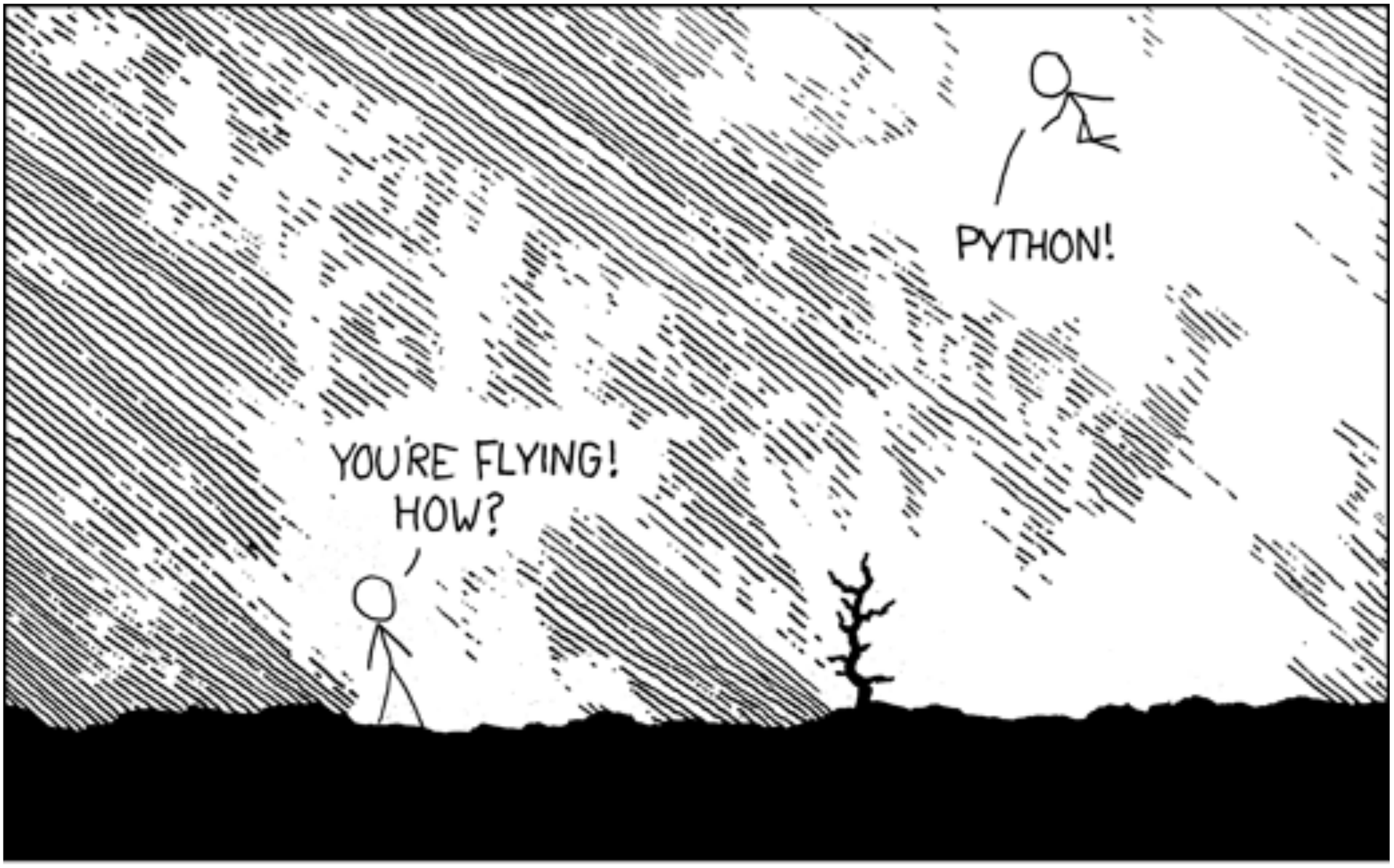
```
figsize(12.5,4)
p = np.linspace( 0,1, 50)
plt.plot( p, 2*p/(1+p), color = "#348ABD", lw = 3 )
#plt.fill_between( p, 2*p/(1+p), alpha = .5, facecolor = ["#A60628"])
plt.scatter( 0.2, 2*(0.2)/1.2, s = 140, c ="#348ABD"  )
plt.xlim( 0, 1)
plt.ylim( 0, 1)
plt.xlabel( "Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title( "Are there bugs in my code?");
```



[http://nbviewer.ipython.org/urls/raw.github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/master/Chapter1_Introduction/Chapter1_Introduction.ipynb]

Mike Roberts, Stanford University

You probably all know Python.



[http://xkcd.com/353/]

You probably all know the default Python interpreter.

```
Last login: Mon May 20 17:53:32 on ttys000
dn0a2100e6:~ mike$ python
Enthought Python Distribution -- www.enthought.com
Version: 7.3-2 (32-bit)

Python 2.7.3 |EPD 7.3-2 (32-bit)| (default, Apr 12 2012, 11:28:34)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "credits", "demo" or "enthought" for more information.
>>> 2 + 2
4
>>>
```

Python is already awesome.

IPython is a framework for making Python even more awesome, especially in a research setting.

More specifically...

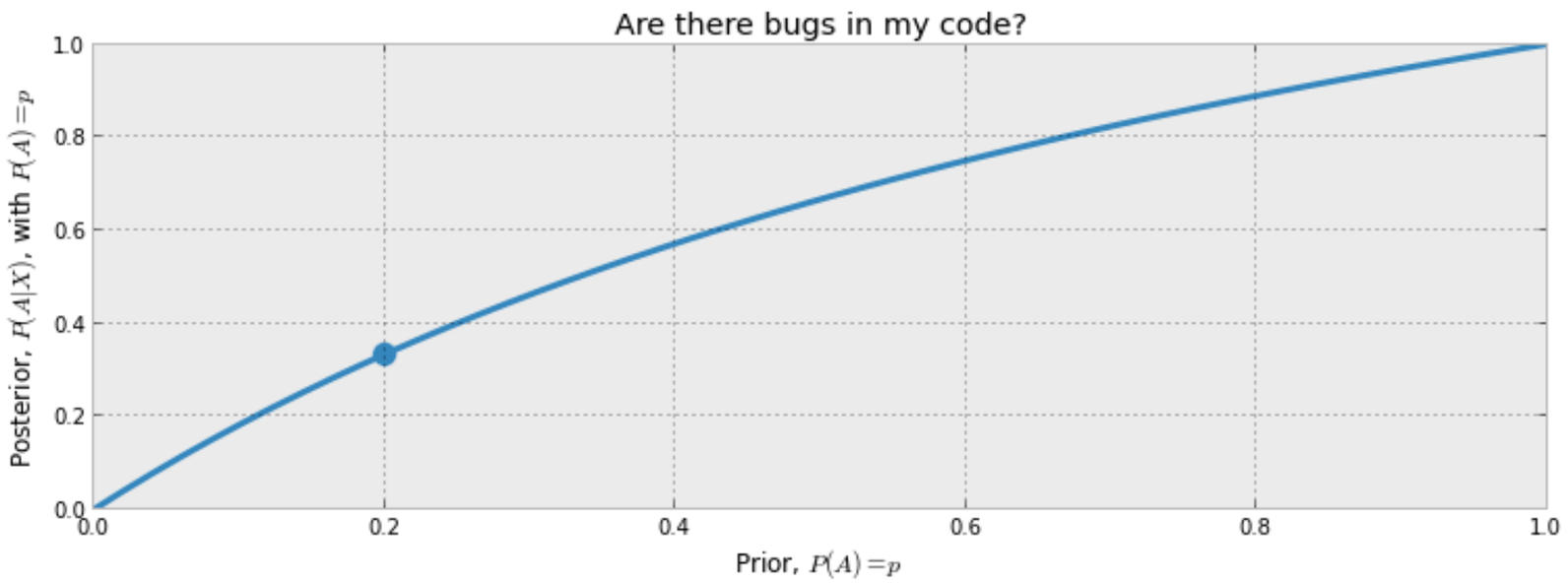
IPython is a more powerful interactive Python interpreter.

write and execute Python code in *snippets*

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)} = \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```
figsize(12.5,4)
p = np.linspace( 0,1, 50)
plt.plot( p, 2*p/(1+p), color = "#348ABD", lw = 3 )
#plt.fill_between( p, 2*p/(1+p), alpha = .5, facecolor = ["#A60628"])
plt.scatter( 0.2, 2*(0.2)/1.2, s = 140, c ="#348ABD"  )
plt.xlim( 0, 1)
plt.ylim( 0, 1)
plt.xlabel( "Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title( "Are there bugs in my code?");
```



[http://nbviewer.ipython.org/urls/raw.github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/master/Chapter1_Introduction/Chapter1_Introduction.ipynb]

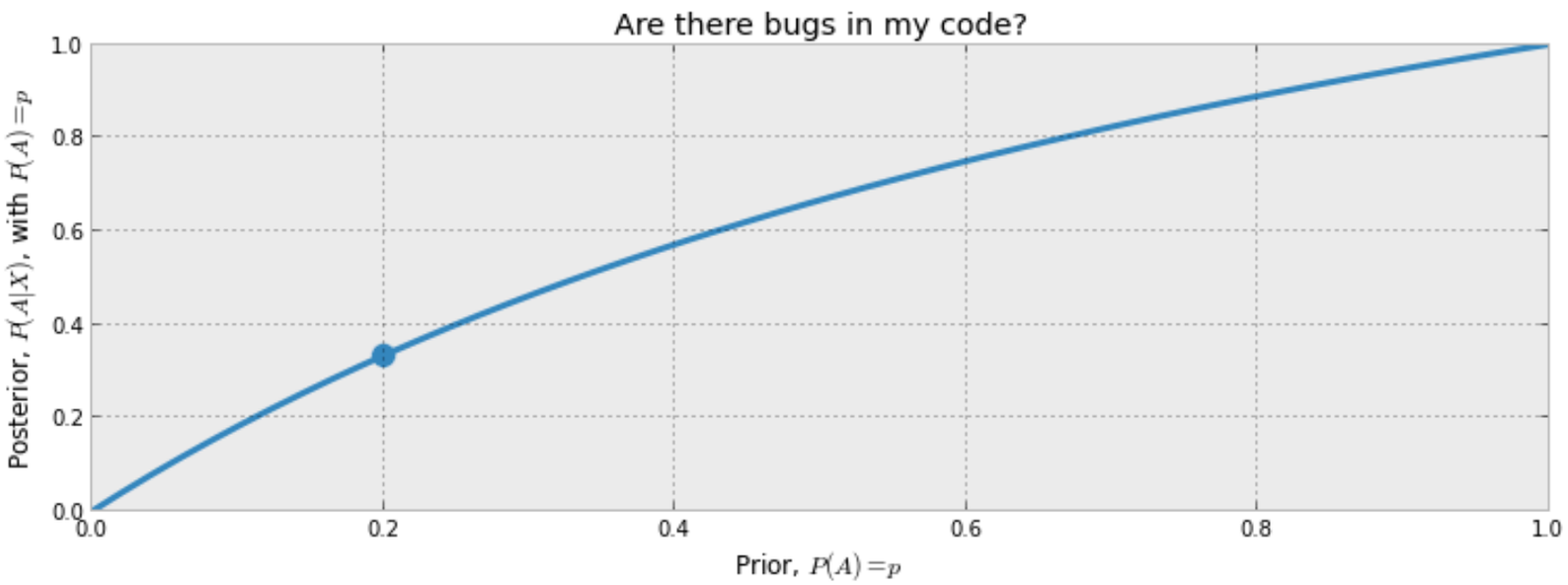
IPython is a more powerful interactive Python interpreter.

comments can include Latex math

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)}$$
$$= \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```
figsize(12.5,4)
p = np.linspace( 0,1, 50)
plt.plot( p, 2*p/(1+p), color = "#348ABD", lw = 3 )
#plt.fill_between( p, 2*p/(1+p), alpha = .5, facecolor = ["#A60628"])
plt.scatter( 0.2, 2*(0.2)/1.2, s = 140, c ="#348ABD"  )
plt.xlim( 0, 1)
plt.ylim( 0, 1)
plt.xlabel( "Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title( "Are there bugs in my code?");
```



[http://nbviewer.ipython.org/urls/raw.github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/master/Chapter1_Introduction/Chapter1_Introduction.ipynb]

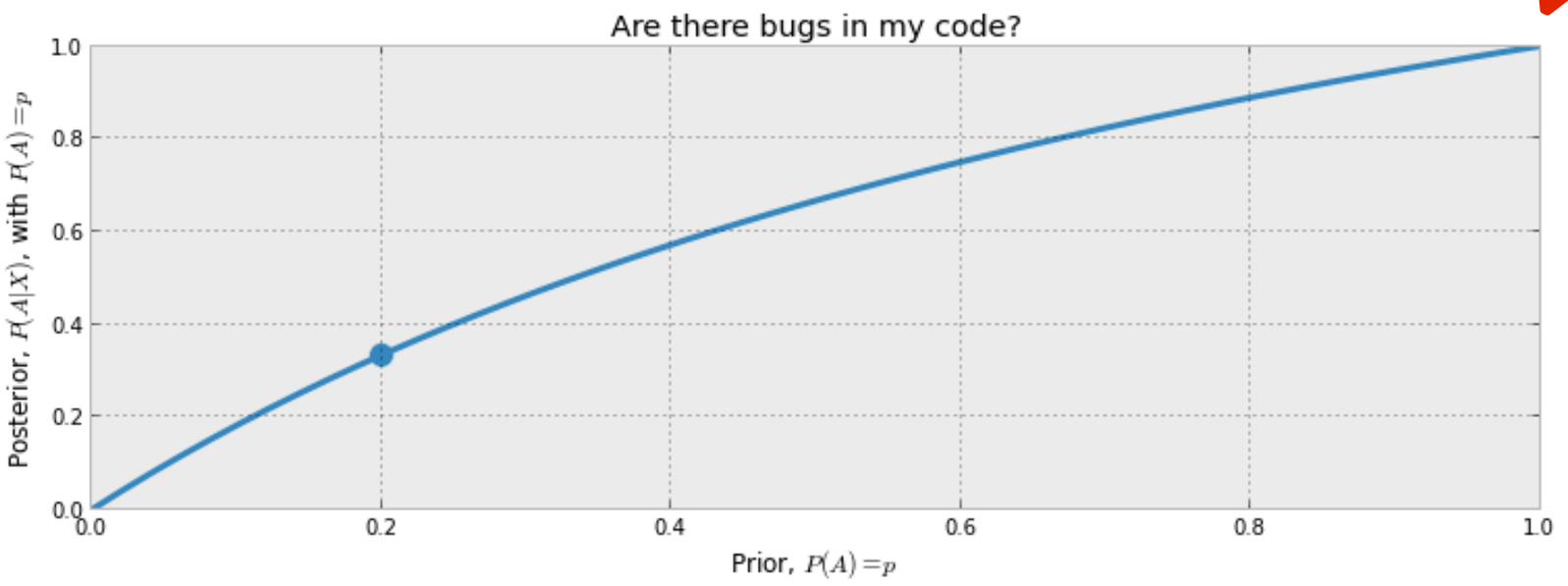
IPython is a more powerful interactive Python interpreter.

any plots generated by your code are displayed inline

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)} = \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```
figsize(12.5,4)
p = np.linspace( 0,1, 50)
plt.plot( p, 2*p/(1+p), color = "#348ABD", lw = 3 )
#plt.fill_between( p, 2*p/(1+p), alpha = .5, facecolor = ["#A60628"])
plt.scatter( 0.2, 2*(0.2)/1.2, s = 140, c ="#348ABD"  )
plt.xlim( 0, 1)
plt.ylim( 0, 1)
plt.xlabel( "Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title( "Are there bugs in my code?");
```



[http://nbviewer.ipython.org/urls/raw.github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/master/Chapter1_Introduction/Chapter1_Introduction.ipynb]

Write Python code interactively in a web browser instead of a terminal window.

IPy IPython Dashboard | IPy HW2 (Gaussian Blur)

127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560

Announcements | CS · Matrix calculus - Wi · Differentiation rules · Lagrange multiplier · Likelihood function · List of logarithmic ic · List of trigonometric · G3D Innovation Engi · scikit-learn: machine

IP[y]: Notebook HW2 (Gaussian Blur) Last saved: May 21 11:24 AM

File Edit View Insert Cell Kernel Help

Code

common gaussian blur code

We begin by defining the non-normalized Gaussian Function $G(x, y)$ as follows:

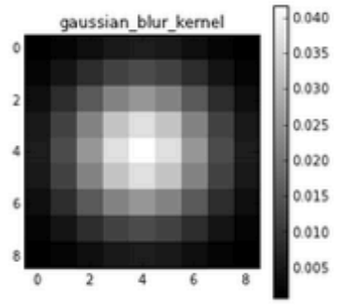
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then define the normalized Gaussian Function as $N(x, y) = cG(x, y)$, where c is a normalization constant:

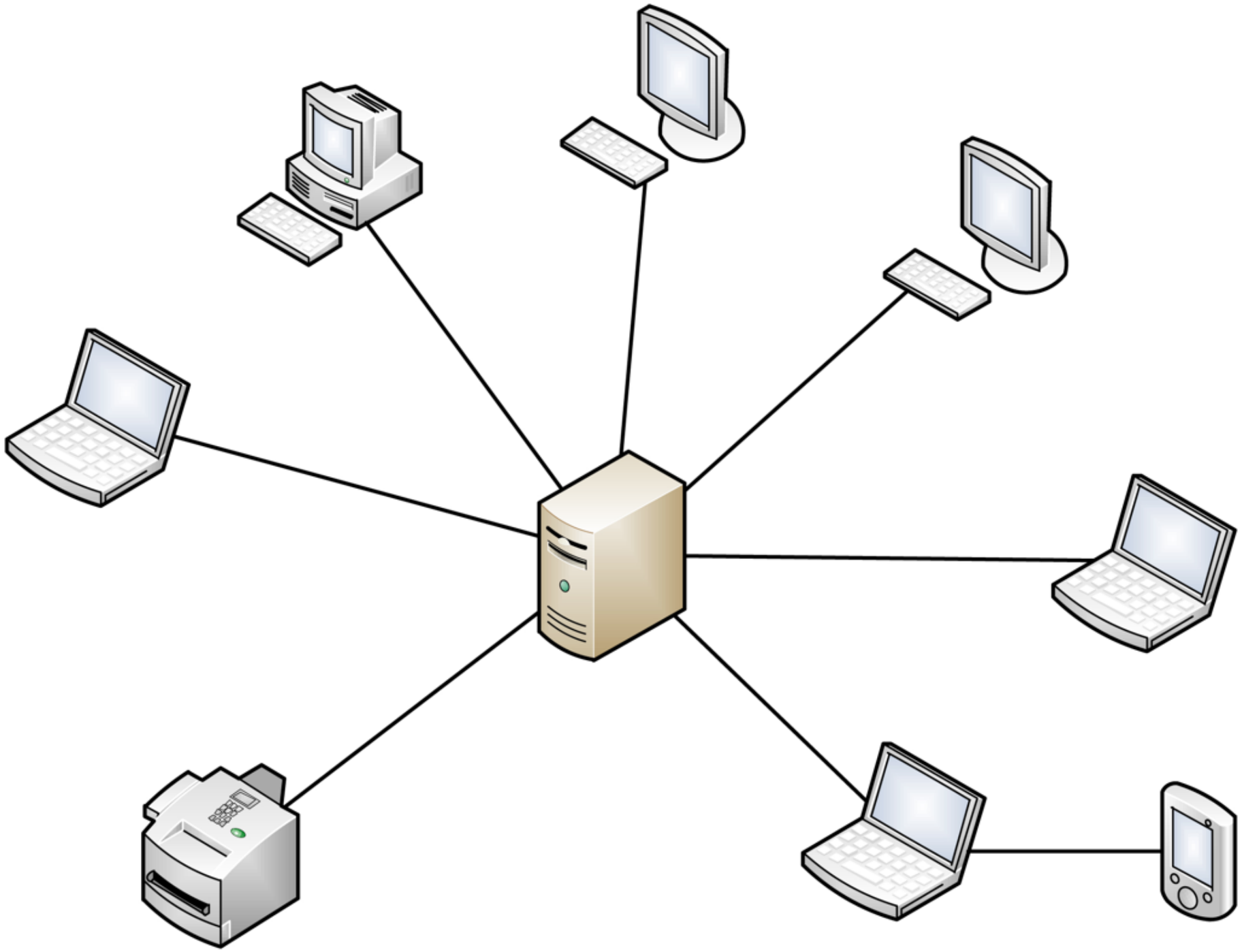
$$c = \frac{1}{\int\int G(x, y)\mathrm{d}x\mathrm{d}y}$$

To perform a Gaussian Blur, we use $N(x, y)$ as a convolution kernel.

In [4]:
```
import scipy

gaussian_blur_kernel_width      = numpy.int32(9)
gaussian_blur_kernel_half_width = numpy.int32(4)
gaussian_blur_sigma             = numpy.float32(2)

y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

gaussian_blur_kernel_not_normalized = numpy.exp( ( - ( x**2 + y**2 ) ) / ( 2 * gaussian_blur_sigma**2 ) )
normalization_constant              = numpy.float32(1) / numpy.sum(gaussian_blur_kernel_not_normalized)
gaussian_blur_kernel                = (normalization_constant * gaussian_blur_kernel_not_normalized).astype(numpy.float32)



matplotlib.pyplot.imshow(gaussian_blur_kernel, cmap="gray", interpolation="nearest");
matplotlib.pyplot.title("gaussian_blur_kernel");
matplotlib.pyplot.colorbar();
```

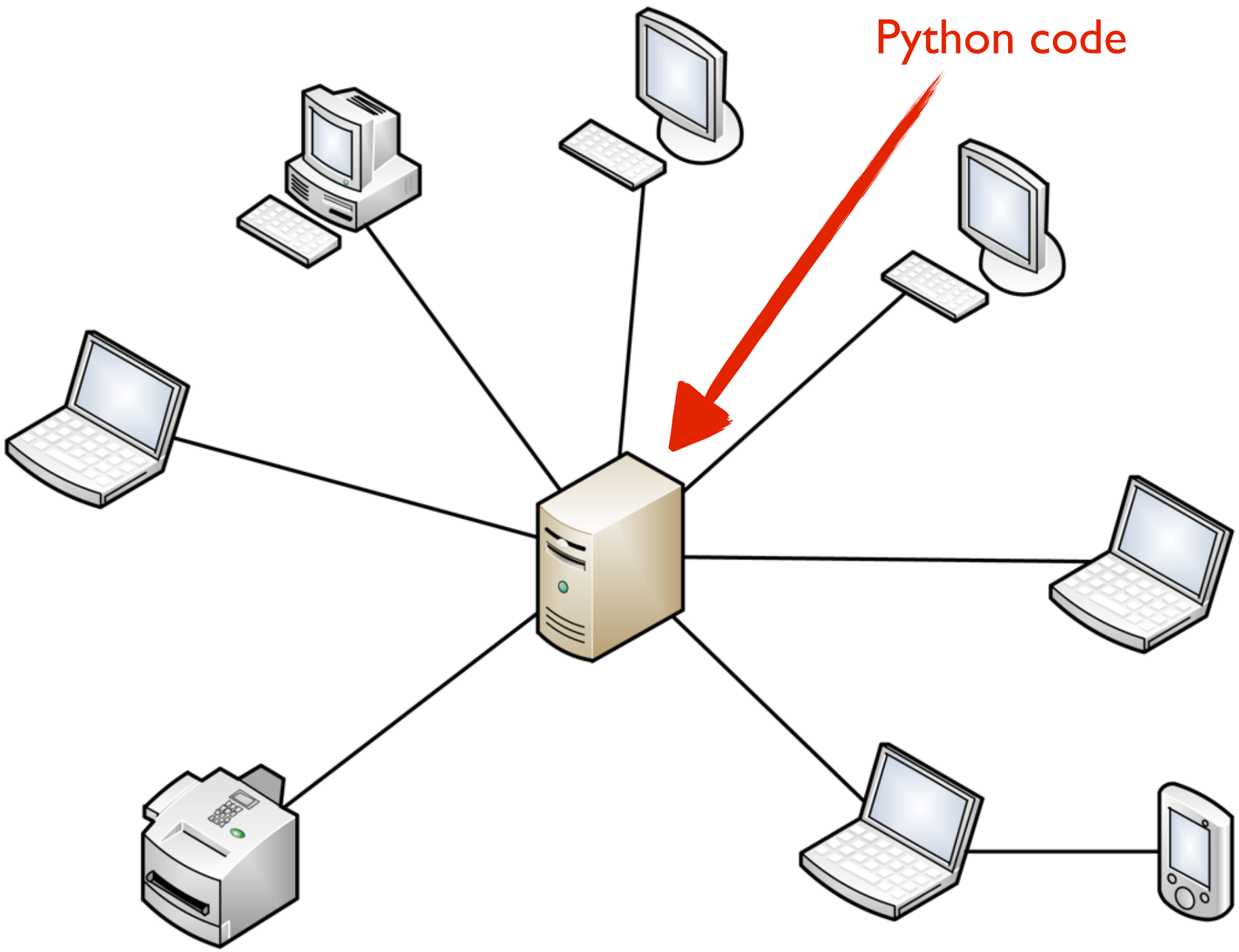


IPython has a decoupled client-server architecture.



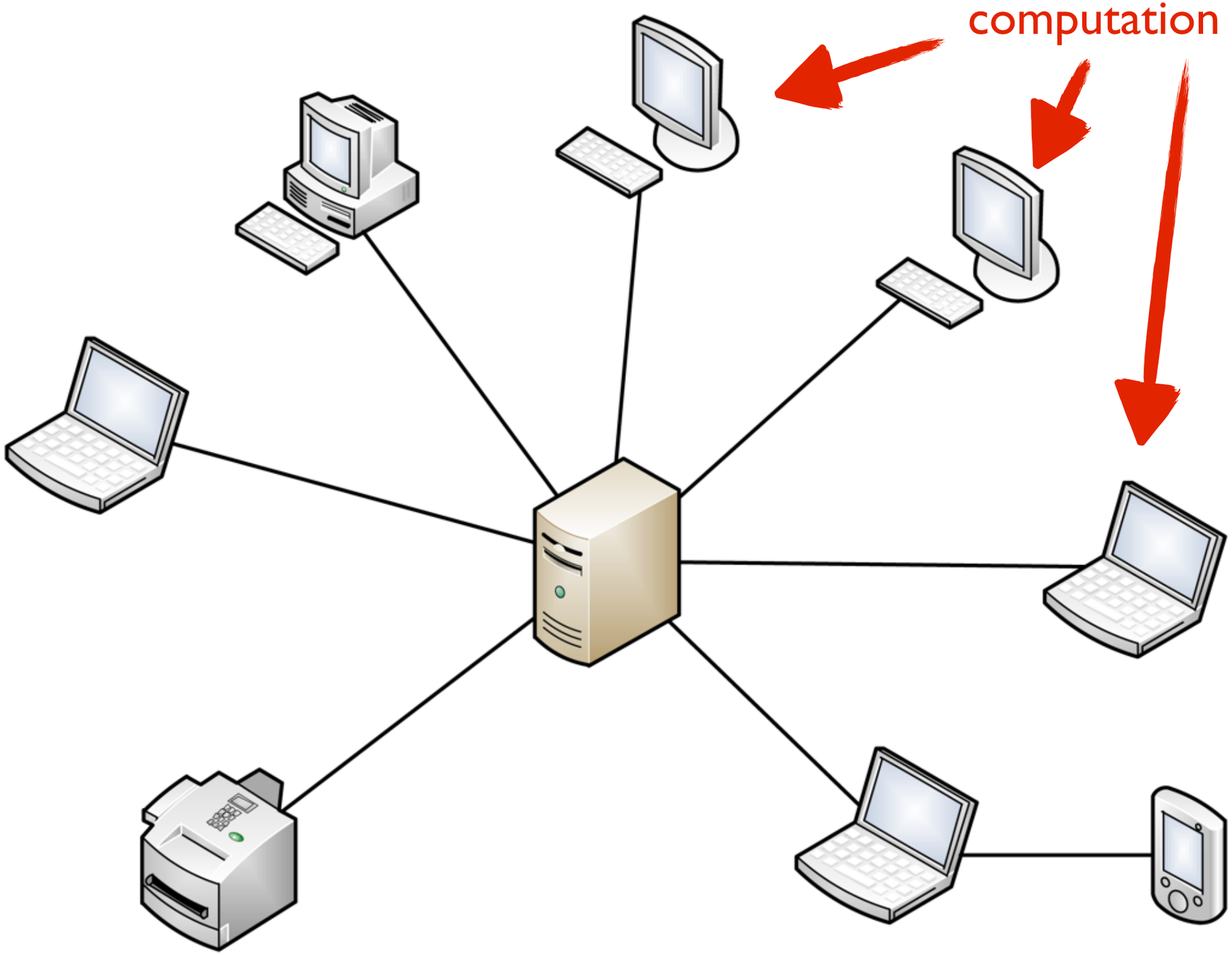
[http://communities.intel.com/community/datastack/blog/2011/05/02/top-10-reasons-to-setup-a-client-server-network]

IPython has a decoupled client-server architecture.



[http://communities.intel.com/community/datastack/blog/2011/05/02/top-10-reasons-to-setup-a-client-server-network]

IPython has a decoupled client-server architecture.



[http://communities.intel.com/community/datastack/blog/2011/05/02/top-10-reasons-to-setup-a-client-server-network]

You can stay productive on any computer with an internet connection and a web browser.

[image source unknown]

Your Python session is persistent for the lifetime of the *server*, so you can abruptly shut down your *client* without losing any progress.

The default way to write IPython code is in an *IPython notebook*.

IPy IPython Dashboard | IPy HW2 (Gaussian Blur)

127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560

Announcements | CS | Matrix calculus - Wi | Differentiation rules | Lagrange multiplier | Likelihood function | List of logarithmic ic | List of trigonometric | G3D Innovation Engi | scikit-learn: machine

IP[y]: Notebook HW2 (Gaussian Blur) Last saved: May 21 11:24 AM

File Edit View Insert Cell Kernel Help

Code

common gaussian blur code

We begin by defining the non-normalized Gaussian Function $G(x, y)$ as follows:

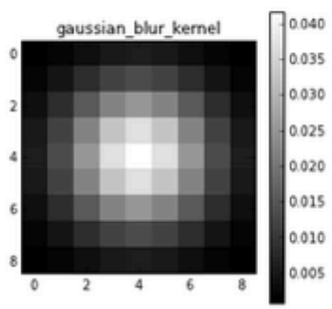
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then define the normalized Gaussian Function as $N(x, y) = cG(x, y)$, where c is a normalization constant:

$$c = \frac{1}{\int\int G(x, y)\mathrm{d}x\mathrm{d}y}$$

To perform a Gaussian Blur, we use $N(x, y)$ as a convolution kernel.

In [4]:
```
import scipy

gaussian_blur_kernel_width      = numpy.int32(9)
gaussian_blur_kernel_half_width = numpy.int32(4)
gaussian_blur_sigma             = numpy.float32(2)

y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

gaussian_blur_kernel_not_normalized = numpy.exp( ( - ( x**2 + y**2 ) ) / ( 2 * gaussian_blur_sigma**2 ) )
normalization_constant              = numpy.float32(1) / numpy.sum(gaussian_blur_kernel_not_normalized)
gaussian_blur_kernel                = (normalization_constant * gaussian_blur_kernel_not_normalized).astype(numpy.float32)



matplotlib.pyplot.imshow(gaussian_blur_kernel, cmap="gray", interpolation="nearest");
matplotlib.pyplot.title("gaussian_blur_kernel");
matplotlib.pyplot.colorbar();
```



An *IPython notebook* is a standalone file that contains code snippets, math, comments, and output.

IPy IPython Dashboard | IPy HW2 (Gaussian Blur)

127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560

Announcements | CS | Matrix calculus - Wi | Differentiation rules | Lagrange multiplier | Likelihood function | List of logarithmic ic | List of trigonometric | G3D Innovation Engi | scikit-learn: machine

IP[y]: Notebook HW2 (Gaussian Blur) Last saved: May 21 11:24 AM

File Edit View Insert Cell Kernel Help

Code

common gaussian blur code

We begin by defining the non-normalized Gaussian Function $G(x, y)$ as follows:

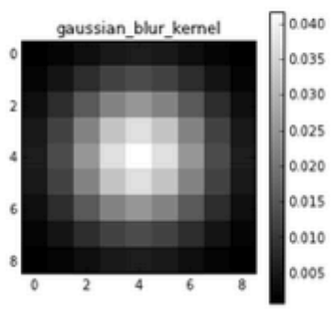
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then define the normalized Gaussian Function as $N(x, y) = cG(x, y)$, where c is a normalization constant:

$$c = \frac{1}{\int\int G(x, y)\mathrm{d}x\mathrm{d}y}$$

To perform a Gaussian Blur, we use $N(x, y)$ as a convolution kernel.

In [4]:
```
import scipy

gaussian_blur_kernel_width      = numpy.int32(9)
gaussian_blur_kernel_half_width = numpy.int32(4)
gaussian_blur_sigma             = numpy.float32(2)

y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

gaussian_blur_kernel_not_normalized = numpy.exp( ( - ( x**2 + y**2 ) ) / ( 2 * gaussian_blur_sigma**2 ) )
normalization_constant              = numpy.float32(1) / numpy.sum(gaussian_blur_kernel_not_normalized)
gaussian_blur_kernel                = (normalization_constant * gaussian_blur_kernel_not_normalized).astype(numpy.float32)



matplotlib.pyplot.imshow(gaussian_blur_kernel, cmap="gray", interpolation="nearest");
matplotlib.pyplot.title("gaussian_blur_kernel");
matplotlib.pyplot.colorbar();
```



But an *IPython notebook* is more than a static collage of snippets related to your research...

IPy IPython Dashboard | IPy HW2 (Gaussian Blur)

127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560

Announcements | CS | Matrix calculus - Wi | Differentiation rules | Lagrange multiplier | Likelihood function | List of logarithmic ic | List of trigonometric | G3D Innovation Engi | scikit-learn: machine

IP[y]: Notebook HW2 (Gaussian Blur) Last saved: May 21 11:24 AM

File Edit View Insert Cell Kernel Help

Code

common gaussian blur code

We begin by defining the non-normalized Gaussian Function $G(x, y)$ as follows:

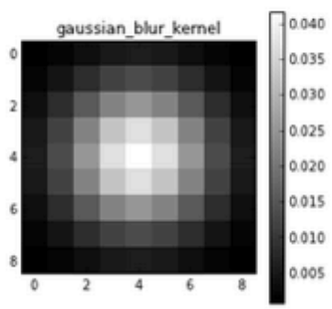
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then define the normalized Gaussian Function as $N(x, y) = cG(x, y)$, where c is a normalization constant:

$$c = \frac{1}{\int\int G(x, y)\mathrm{d}x\mathrm{d}y}$$

To perform a Gaussian Blur, we use $N(x, y)$ as a convolution kernel.

In [4]:
```
import scipy

gaussian_blur_kernel_width      = numpy.int32(9)
gaussian_blur_kernel_half_width = numpy.int32(4)
gaussian_blur_sigma             = numpy.float32(2)

y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

gaussian_blur_kernel_not_normalized = numpy.exp( ( - ( x**2 + y**2 ) ) / ( 2 * gaussian_blur_sigma**2 ) )
normalization_constant              = numpy.float32(1) / numpy.sum(gaussian_blur_kernel_not_normalized)
gaussian_blur_kernel                = (normalization_constant * gaussian_blur_kernel_not_normalized).astype(numpy.float32)



matplotlib.pyplot.imshow(gaussian_blur_kernel, cmap="gray", interpolation="nearest");
matplotlib.pyplot.title("gaussian_blur_kernel");
matplotlib.pyplot.colorbar();
```



...because it is so easy for someone to re-run your code, run your code on new data, and plot different things.

IPy IPython Dashboard | IPy HW2 (Gaussian Blur)

127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560

Announcements | CS | Matrix calculus - Wi | Differentiation rules | Lagrange multiplier | Likelihood function | List of logarithmic ic | List of trigonometric | G3D Innovation Engi | scikit-learn: machine

IP[y]: Notebook HW2 (Gaussian Blur) Last saved: May 21 11:24 AM

File Edit View Insert Cell Kernel Help

Code

common gaussian blur code

We begin by defining the non-normalized Gaussian Function $G(x, y)$ as follows:

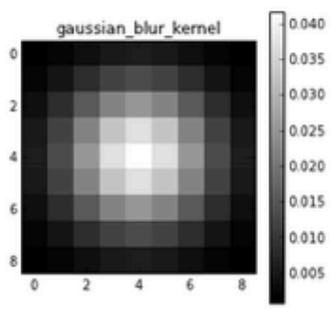
$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We then define the normalized Gaussian Function as $N(x, y) = cG(x, y)$, where c is a normalization constant:

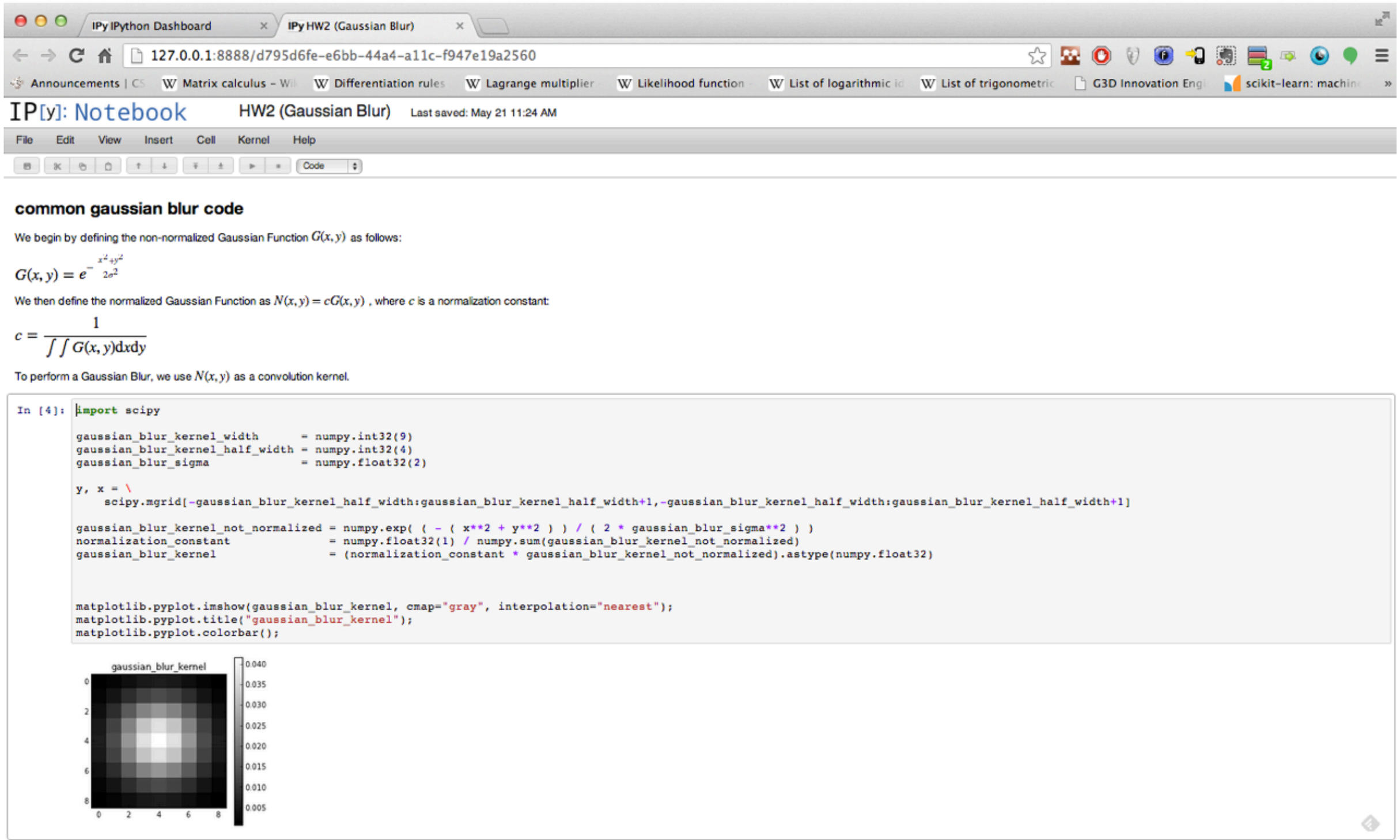
$$c = \frac{1}{\int\int G(x, y)\mathrm{d}x\mathrm{d}y}$$

To perform a Gaussian Blur, we use $N(x, y)$ as a convolution kernel.

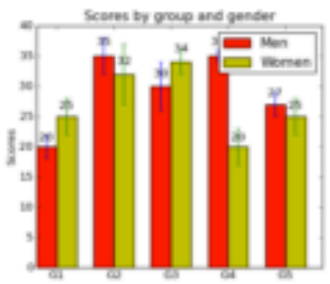
In [4]:
```
import scipy

gaussian_blur_kernel_width      = numpy.int32(9)
gaussian_blur_kernel_half_width = numpy.int32(4)
gaussian_blur_sigma             = numpy.float32(2)

y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

gaussian_blur_kernel_not_normalized = numpy.exp( ( - ( x**2 + y**2 ) ) / ( 2 * gaussian_blur_sigma**2 ) )
normalization_constant              = numpy.float32(1) / numpy.sum(gaussian_blur_kernel_not_normalized)
gaussian_blur_kernel                = (normalization_constant * gaussian_blur_kernel_not_normalized).astype(numpy.float32)



matplotlib.pyplot.imshow(gaussian_blur_kernel, cmap="gray", interpolation="nearest");
matplotlib.pyplot.title("gaussian_blur_kernel");
matplotlib.pyplot.colorbar();
```

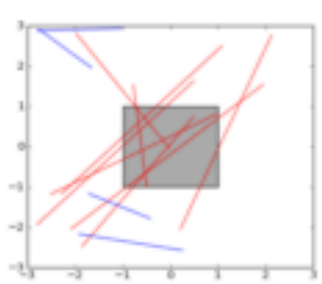


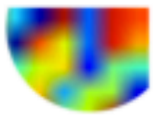
Because of this, IPython notebooks document your research journey in a reproducible way.

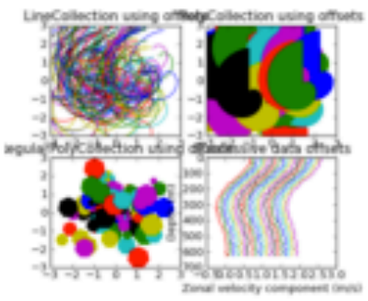


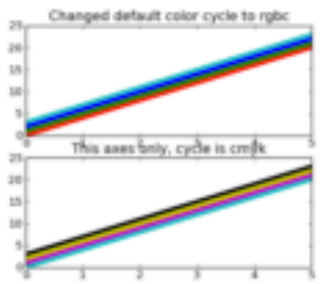
IPython integrates seamlessly with Matplotlib, making it well-suited for data exploration.

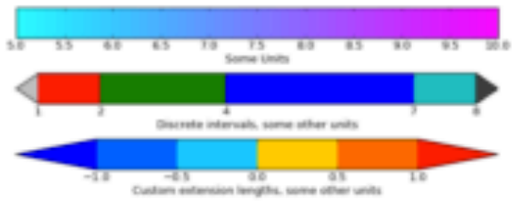


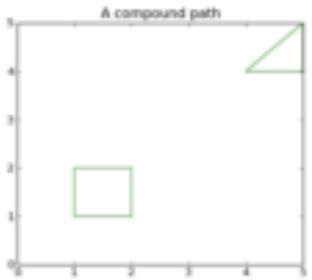


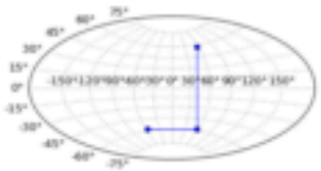


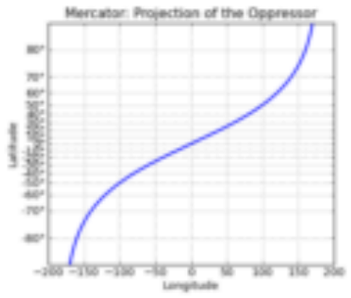


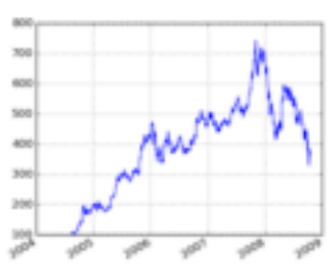


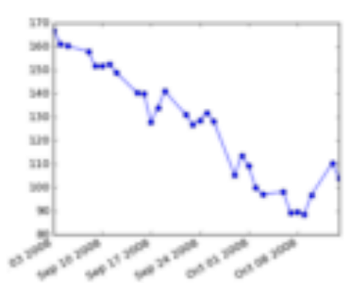


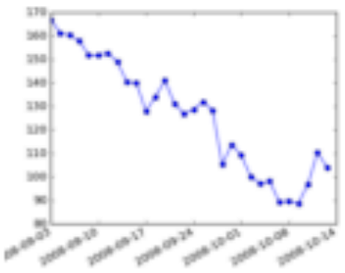


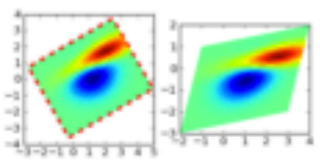


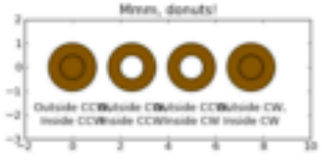


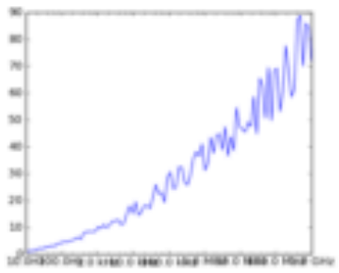


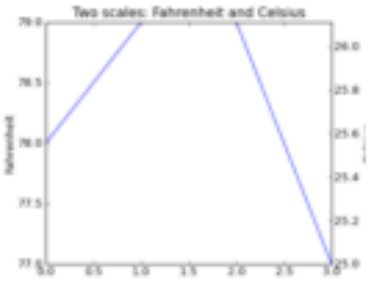


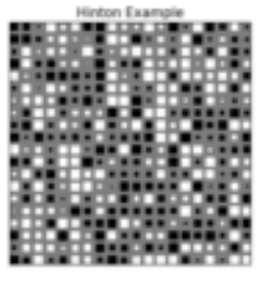


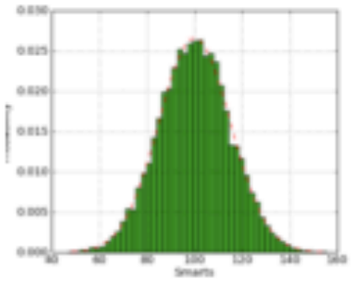


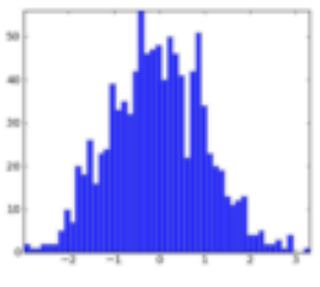


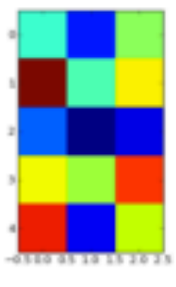


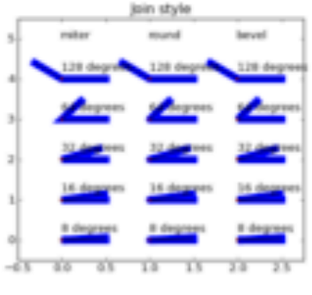


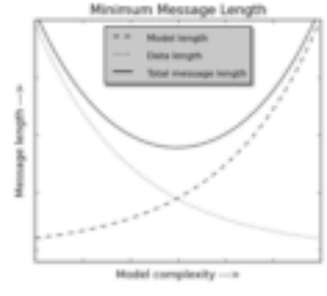


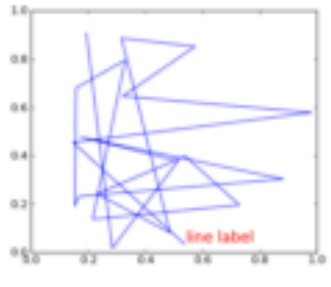






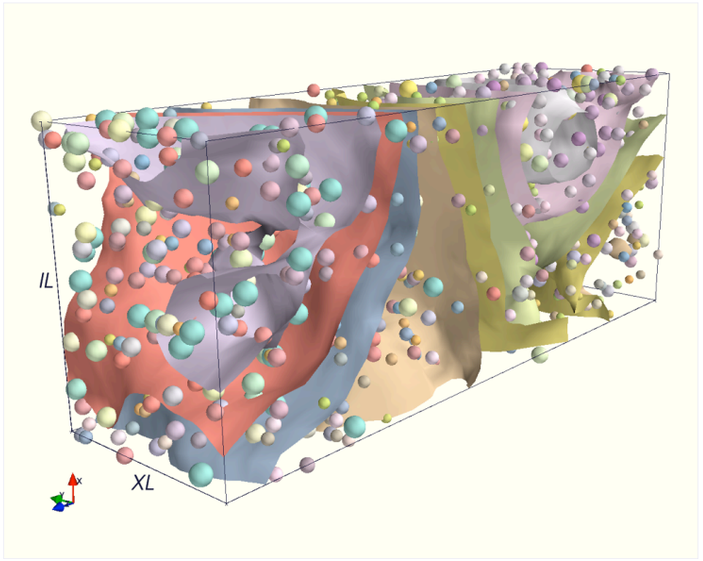


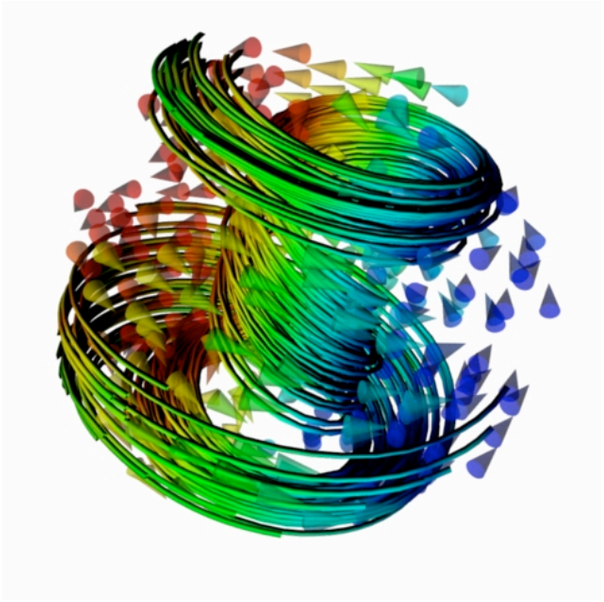


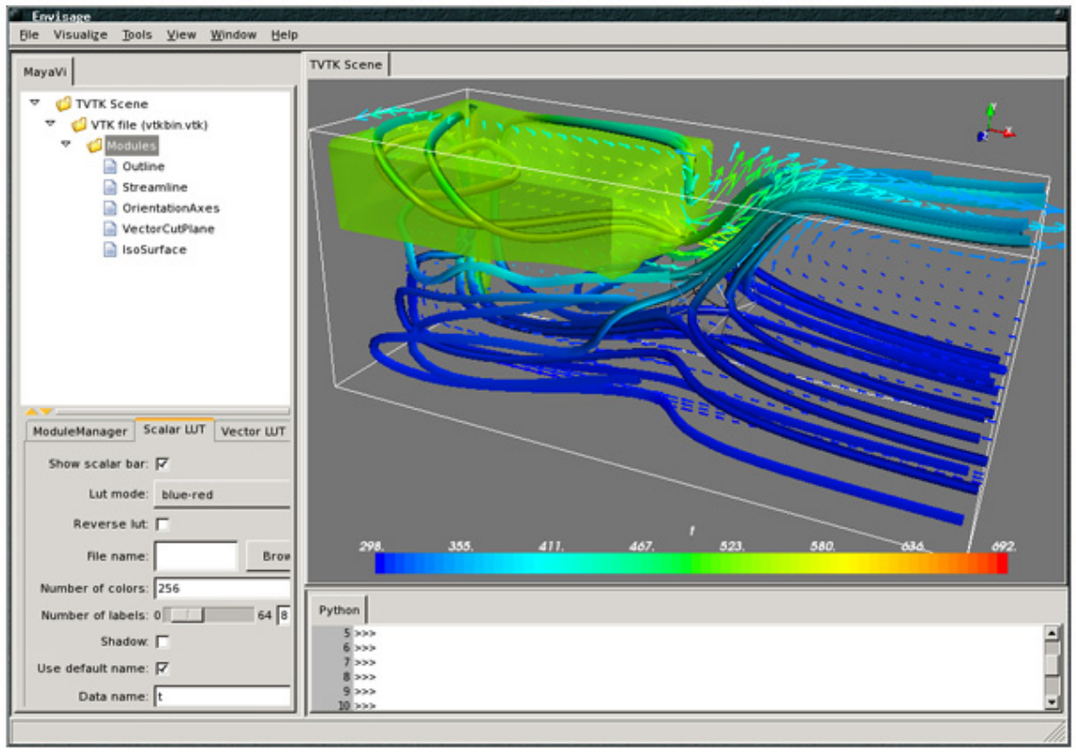


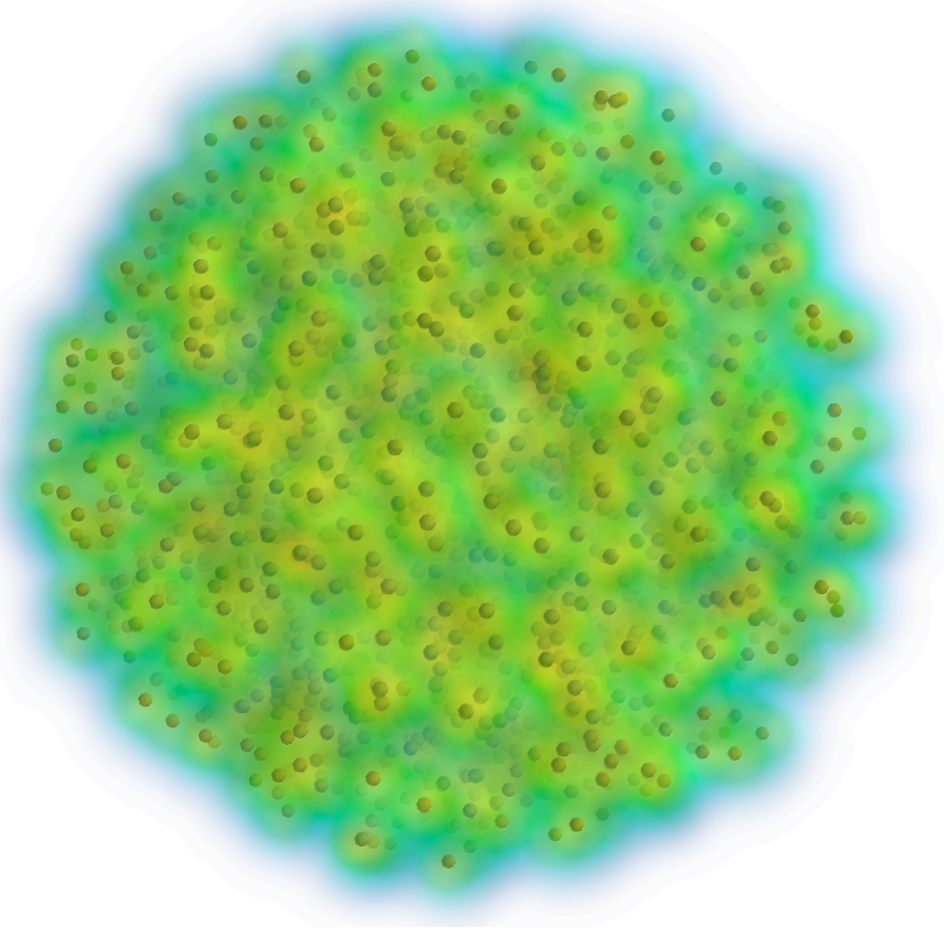
[http://matplotlib.org/gallery.html]

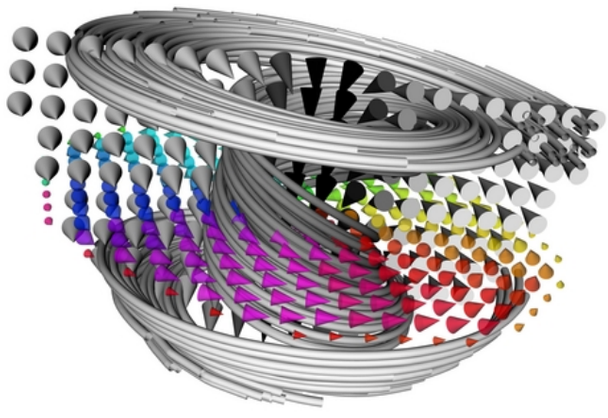
IPython also supports realtime exploration of your data even if it is complicated and high-dimensional.











[http://docs.enthought.com/mayavi/mayavi/auto/examples.html]

IPython has builtin support for cluster-level parallelism.

[http://commons.wikimedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg]

So you can scale up your experiments without needing to re-write your code in C++.

[http://commons.wikimedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg]

If you want to reclaim even more performance, IPython integrates seamlessly with *Cython*. Using *Cython*, you can usually get to 0.9x of native C++ just by adding a few lightweight annotations to your existing Python code.

[image source unknown]

Demo Time

[image source unknown]

IPython is Great

(for large-scale computation, data exploration, and creating reproducible research artifacts)

“I totally just used IPython to debug my Metropolis Hastings stuff. Having the ability to keep changing little snippets of code, then rerun and plot the results, is just great.”

-Niels Joubert

[https://www.facebook.com/njoubert/photos]