

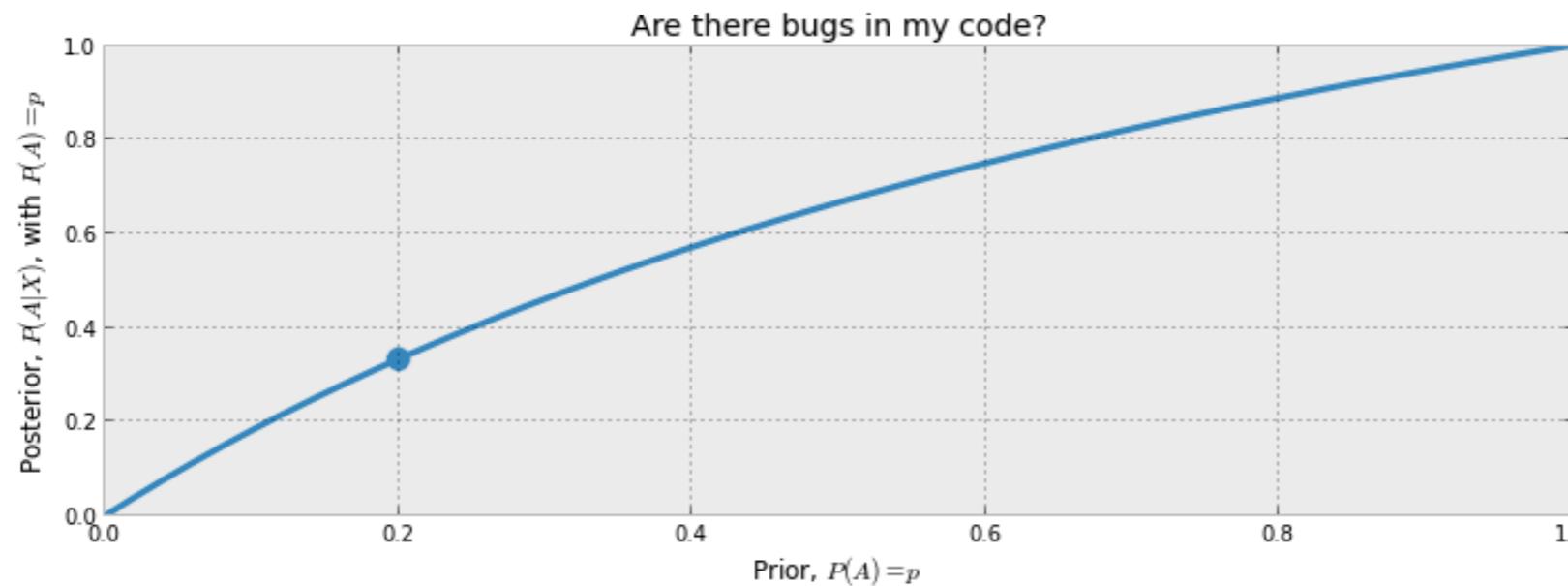
I Python is Great

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

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

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

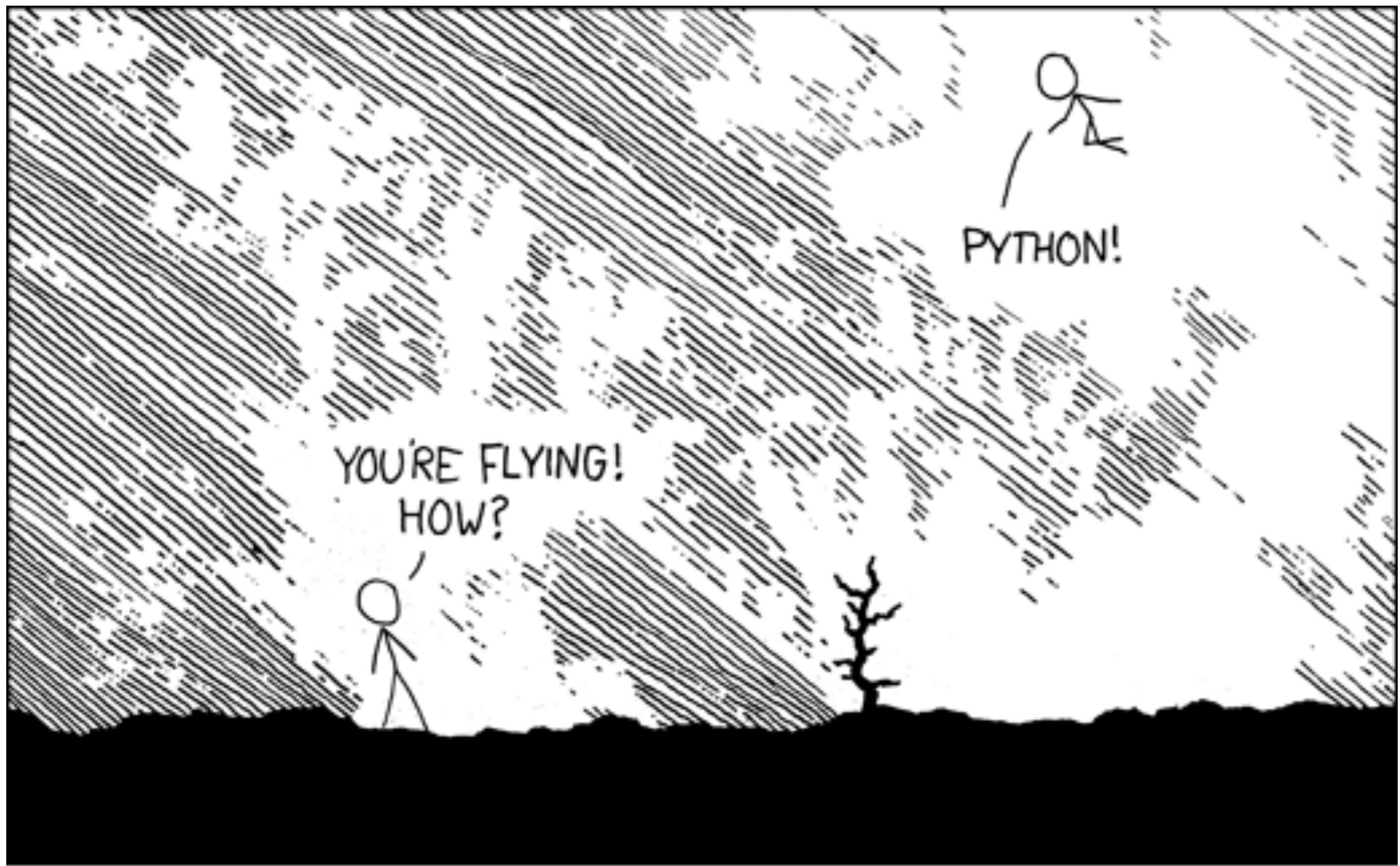
```
figsize(12.5,4)
p = np.linspace( 0,1, 50 )
plt.plot( p, 2*p/(1+p), color = "#348ABD", lw = 3 )
# plt.fill_between( p, 2*p/(1+p), alpha = .5, facecolor = ["#A60628"] )
plt.scatter( 0.2, 2*(0.2)/1.2, s = 140, c ="#348ABD" )
plt.xlim( 0, 1 )
plt.ylim( 0, 1 )
plt.xlabel( "Prior, $P(A) = p$")
plt.ylabel("Posterior,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

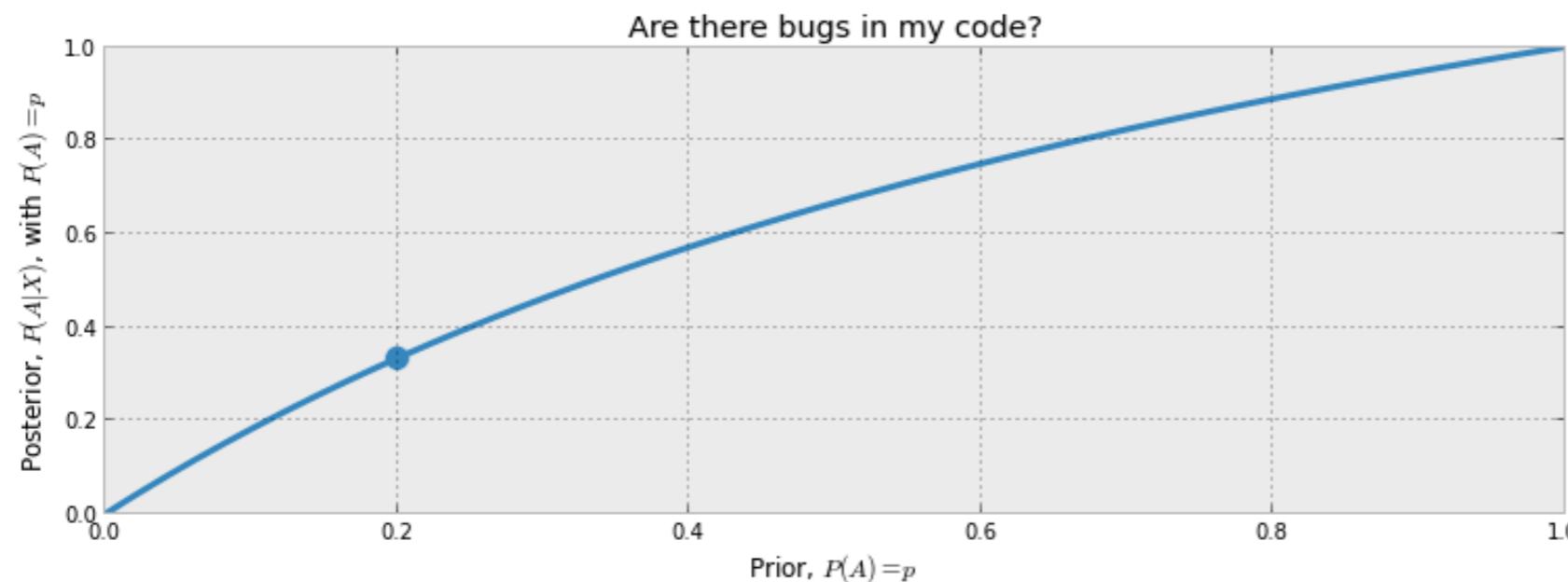
I Python is a more powerful interactive Python interpreter.

write and execute
Python code in
snippets

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

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]

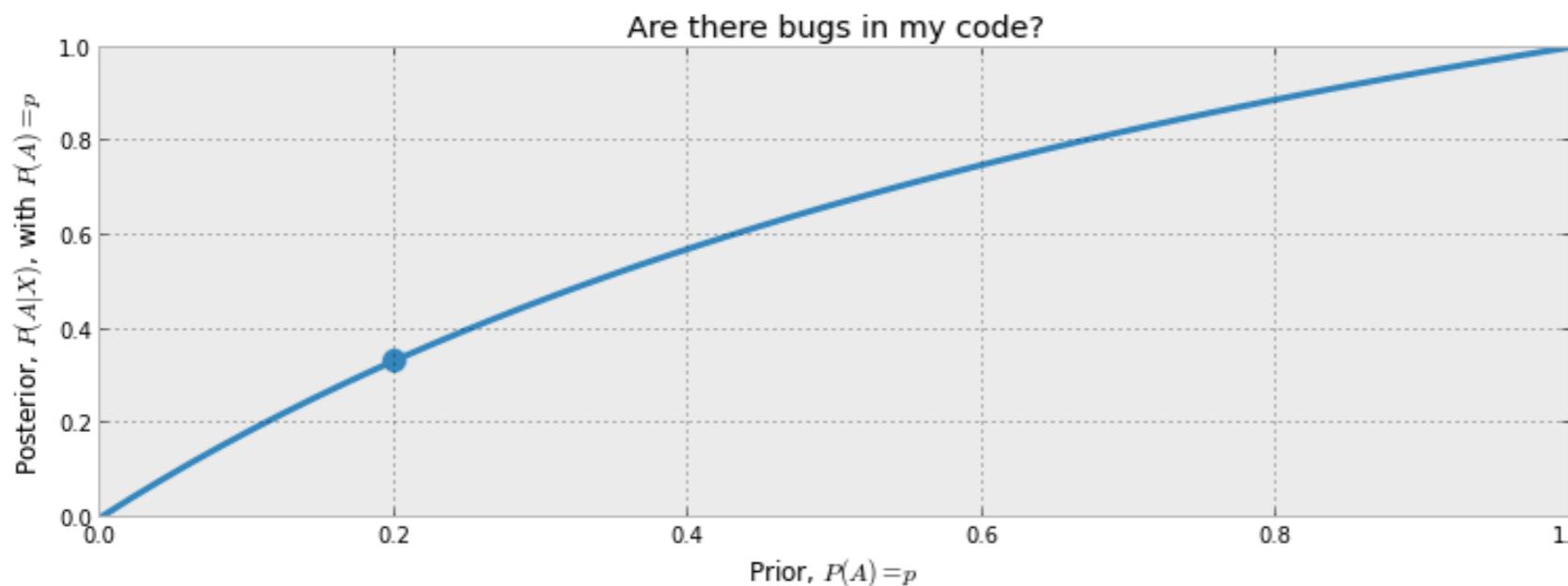
IPython is a more powerful interactive Python interpreter.

comments can include Latex math

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

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]

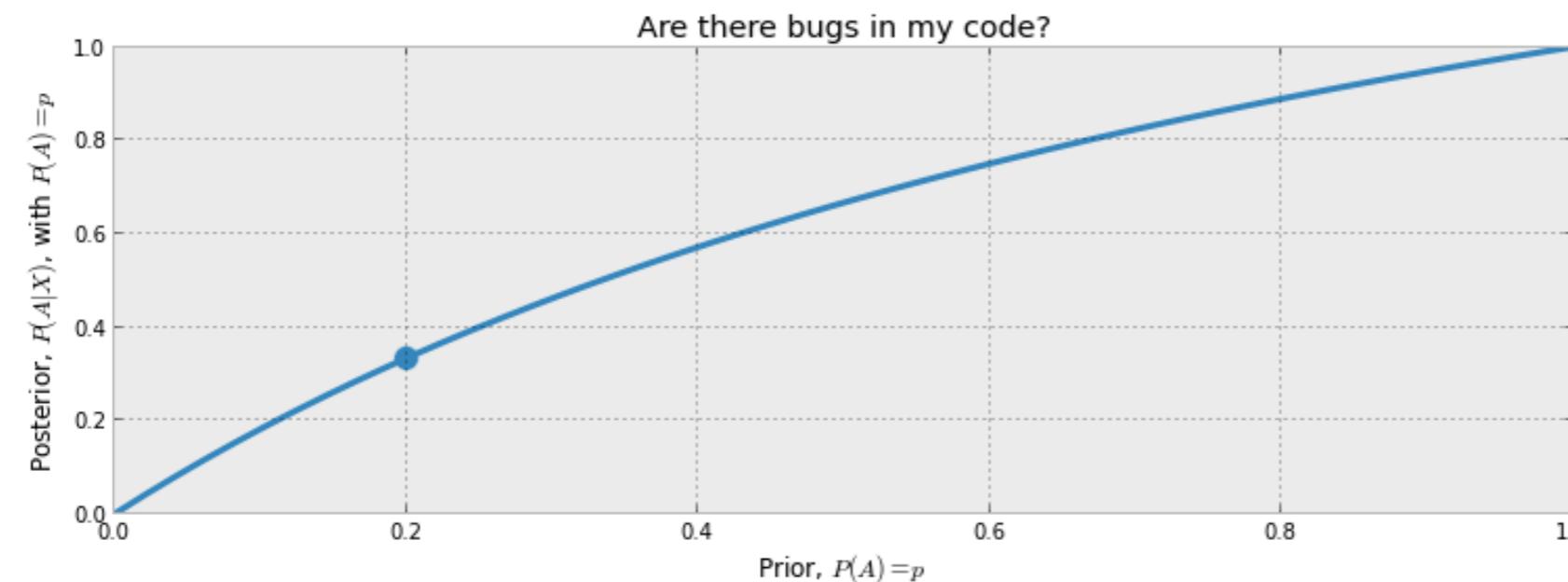
I Python is a more powerful interactive Python interpreter.

any plots generated by your code are displayed inline

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

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.

The screenshot shows a web-based IPython Notebook interface. At the top, there's a browser header with tabs for 'IPy IPython Dashboard' and 'IPy HW2 (Gaussian Blur)'. Below the header, the notebook title is 'HW2 (Gaussian Blur)' and it was last saved at 'May 21 11:24 AM'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', and 'Help'. A toolbar below the menu has buttons for cell types and execution.

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:

$$c = \frac{1}{\iint G(x,y) dx dy}$$

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)

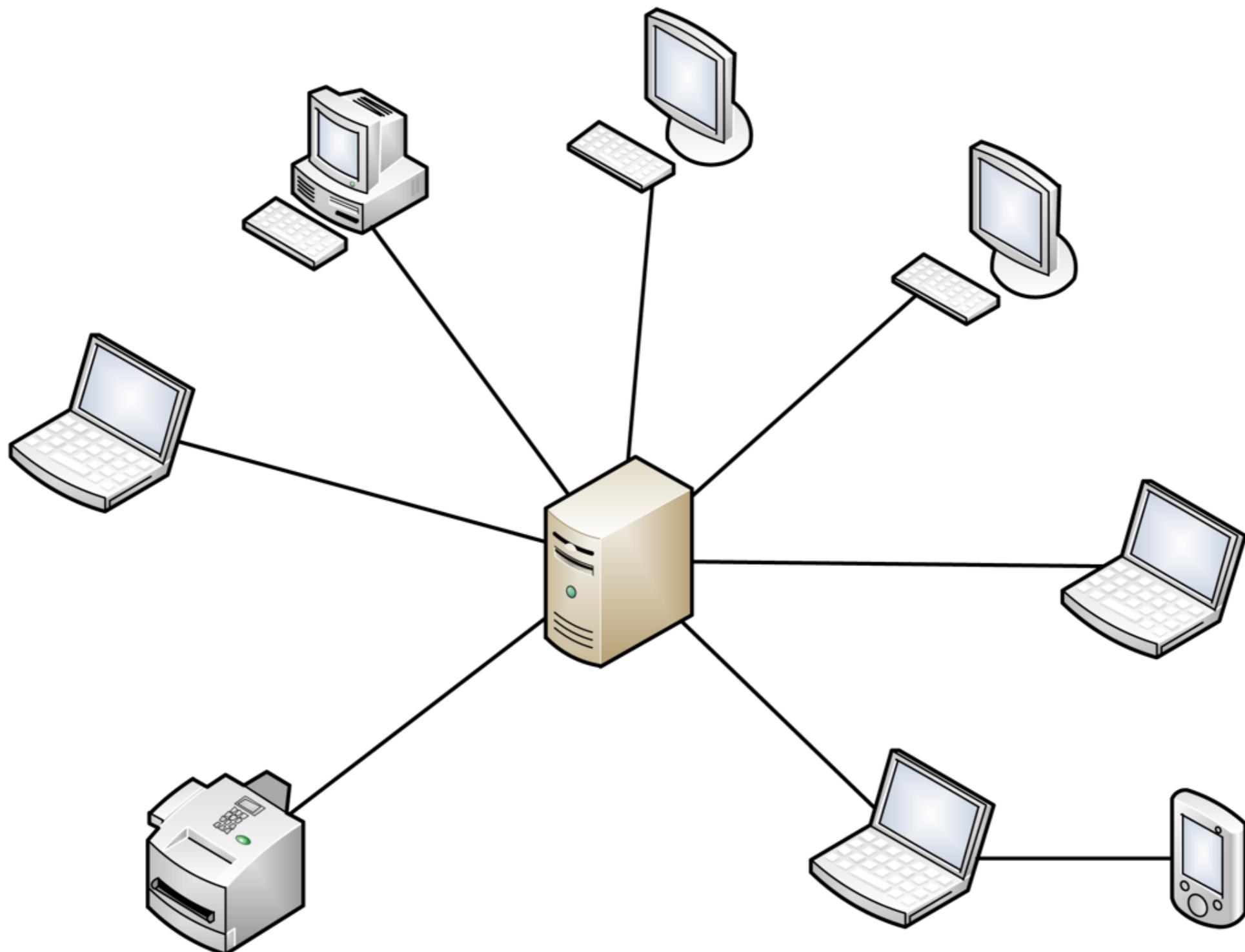
y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

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();
```

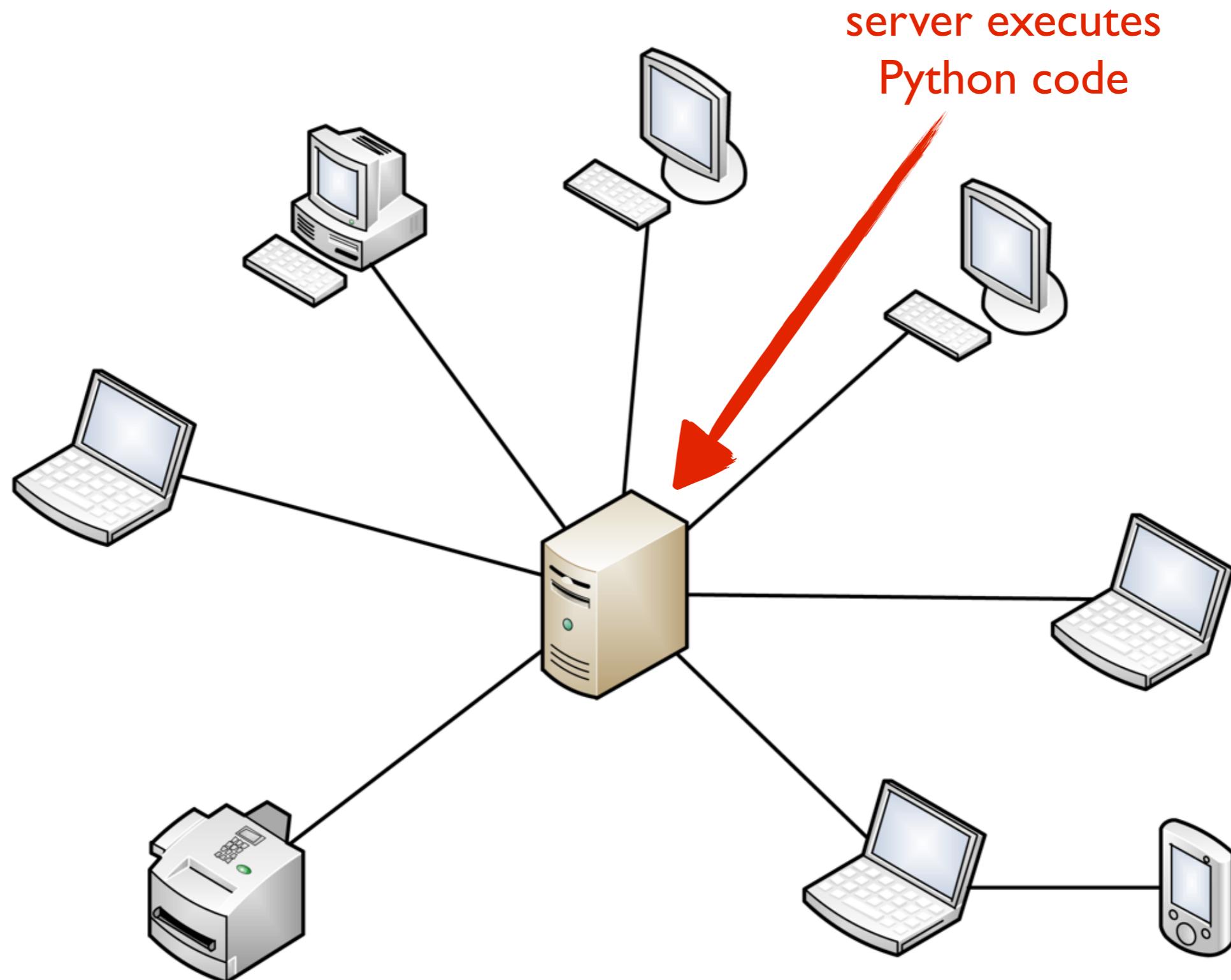
The notebook then displays a heatmap visualization of the Gaussian blur kernel. The x and y axes both range from 0 to 8. The color scale on the right ranges from 0.005 (dark gray) to 0.040 (light gray), representing the value of the kernel at each coordinate. The plot shows a symmetric bell-shaped distribution centered at (4, 4).

IPython has a decoupled client-server architecture.



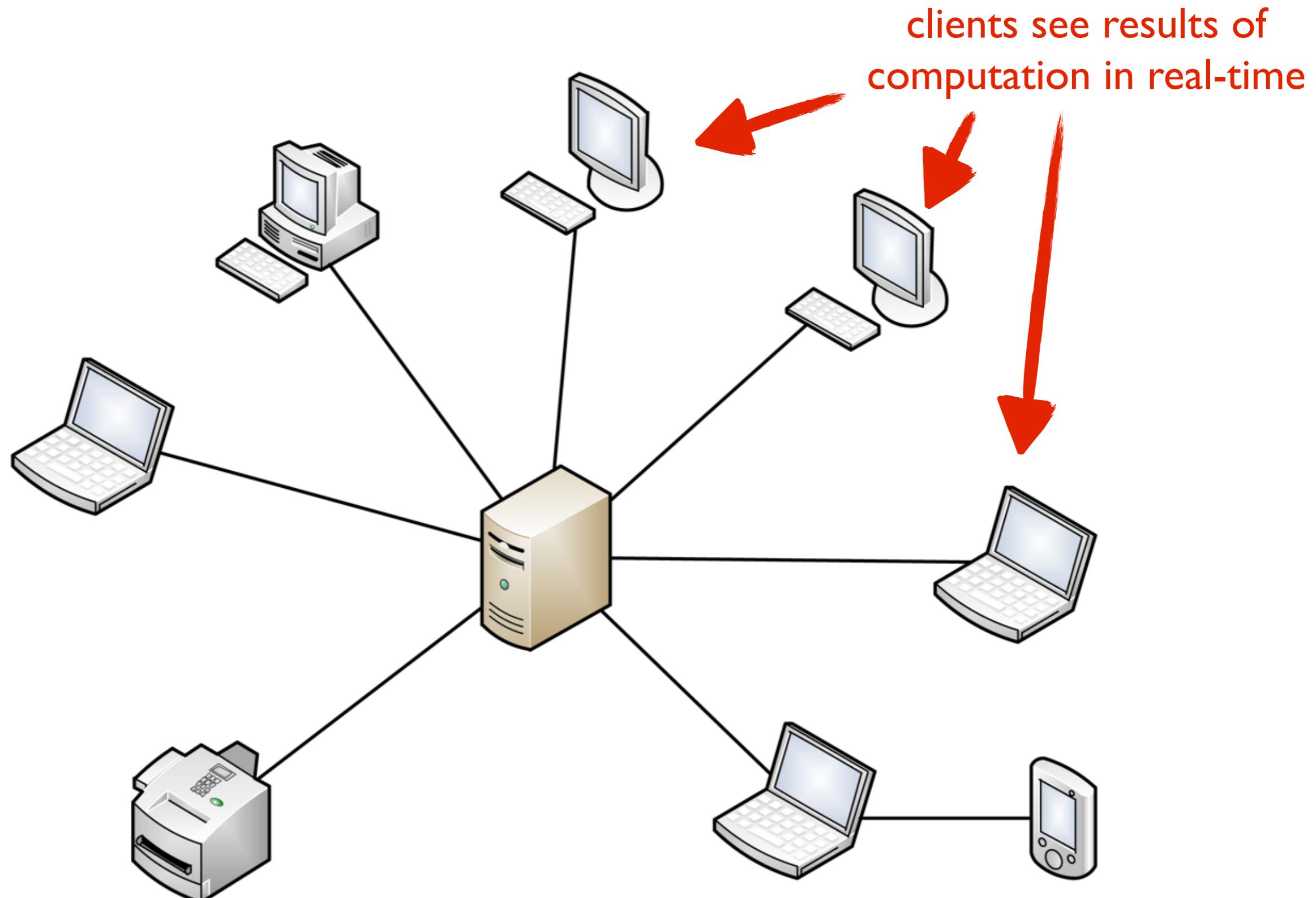
[<http://communities.intel.com/community/datastack/blog/2011/05/02/top-10-reasons-to-setup-a-client-server-network>]

I^{Python} 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.**



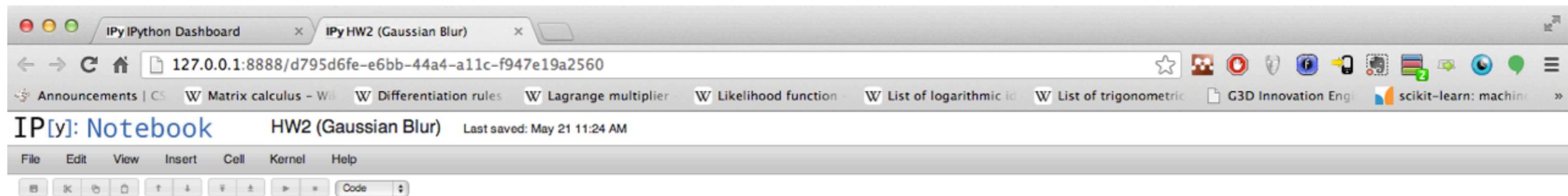
[image source unknown]

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

[image source unknown]



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



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:

$$c = \frac{1}{\int \int G(x, y) dx dy}$$

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

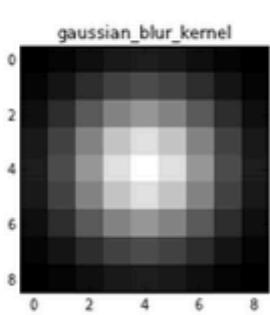
```
In [4]: import scipy

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

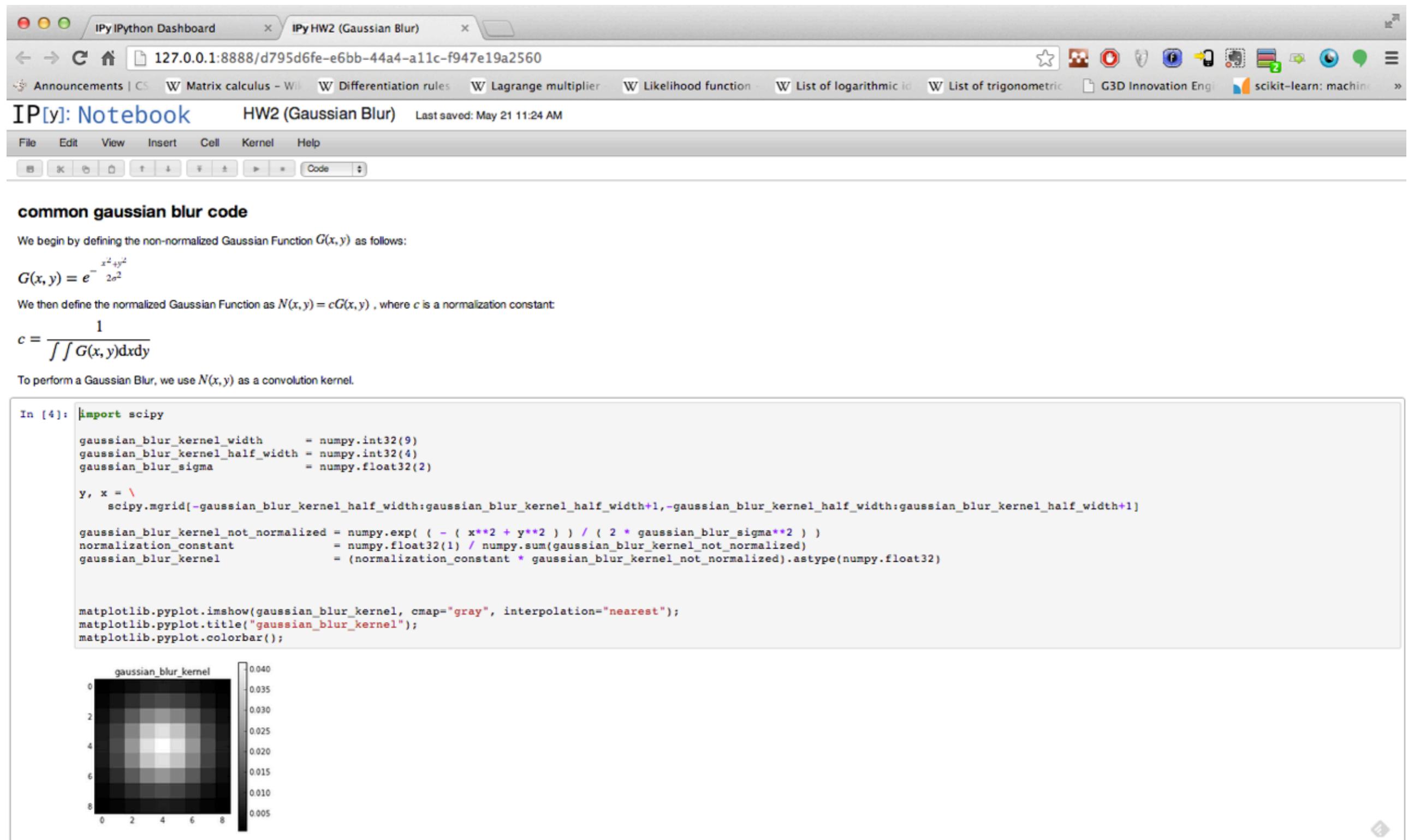
y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

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.



The screenshot shows a Mac OS X desktop with an IPython Notebook window open. The title bar says "IPy IPython Dashboard" and "IPy HW2 (Gaussian Blur)". The browser address bar shows "127.0.0.1:8888/d795d6fe-e6bb-44a4-a11c-f947e19a2560". The notebook tab bar includes "Announcements | CS", "Matrix calculus - WI", "Differentiation rules", "Lagrange multiplier", "Likelihood function", "List of logarithmic id", "List of trigonometric", "G3D Innovation Engi", and "scikit-learn: machine". The main area shows the notebook content:

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:

$$c = \frac{1}{\int \int G(x, y) dx dy}$$

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();
```

The output below the code cell is a grayscale heatmap titled "gaussian_blur_kernel". The x-axis ranges from 0 to 8, and the y-axis ranges from 0 to 8. A vertical color bar on the right indicates values from 0.005 to 0.040, with the highest intensity at the center (0,0).

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

The screenshot shows an IPython Notebook interface with the title "IPy HW2 (Gaussian Blur)". The notebook contains the following text and 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:

$$c = \frac{1}{\int \int G(x,y) dx dy}$$

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();
```

The notebook also displays a heatmap visualization of the generated Gaussian blur kernel, showing a central peak at (0,0) fading to zero at the edges. The x and y axes both range from 0 to 8, and the color bar on the right ranges from 0.005 to 0.040.

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

The screenshot shows an IPython Notebook interface with the title "IPy IPython Dashboard" and a tab titled "IPy HW2 (Gaussian Blur)". The notebook content is as follows:

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:

$$c = \frac{1}{\int \int G(x, y) dx dy}$$

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();
```

The resulting visualization is a 9x9 grayscale heatmap representing the Gaussian blur kernel. The center pixel has a value of approximately 0.040, and the values decrease towards the edges, reaching near zero at the corners. A vertical color bar on the right side of the plot indicates the scale from 0.005 to 0.040.

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

The screenshot shows an IPython Notebook interface with the title "IPy[py]: Notebook" and "HW2 (Gaussian Blur)". The notebook has been last saved on May 21 at 11:24 AM. The toolbar includes standard options like File, Edit, View, Insert, Cell, Kernel, Help, and various cell type icons. Below the toolbar is a menu bar with "Code" selected. The main content area contains text and code snippets:

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:

$$c = \frac{1}{\int \int G(x, y) dx dy}$$

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)

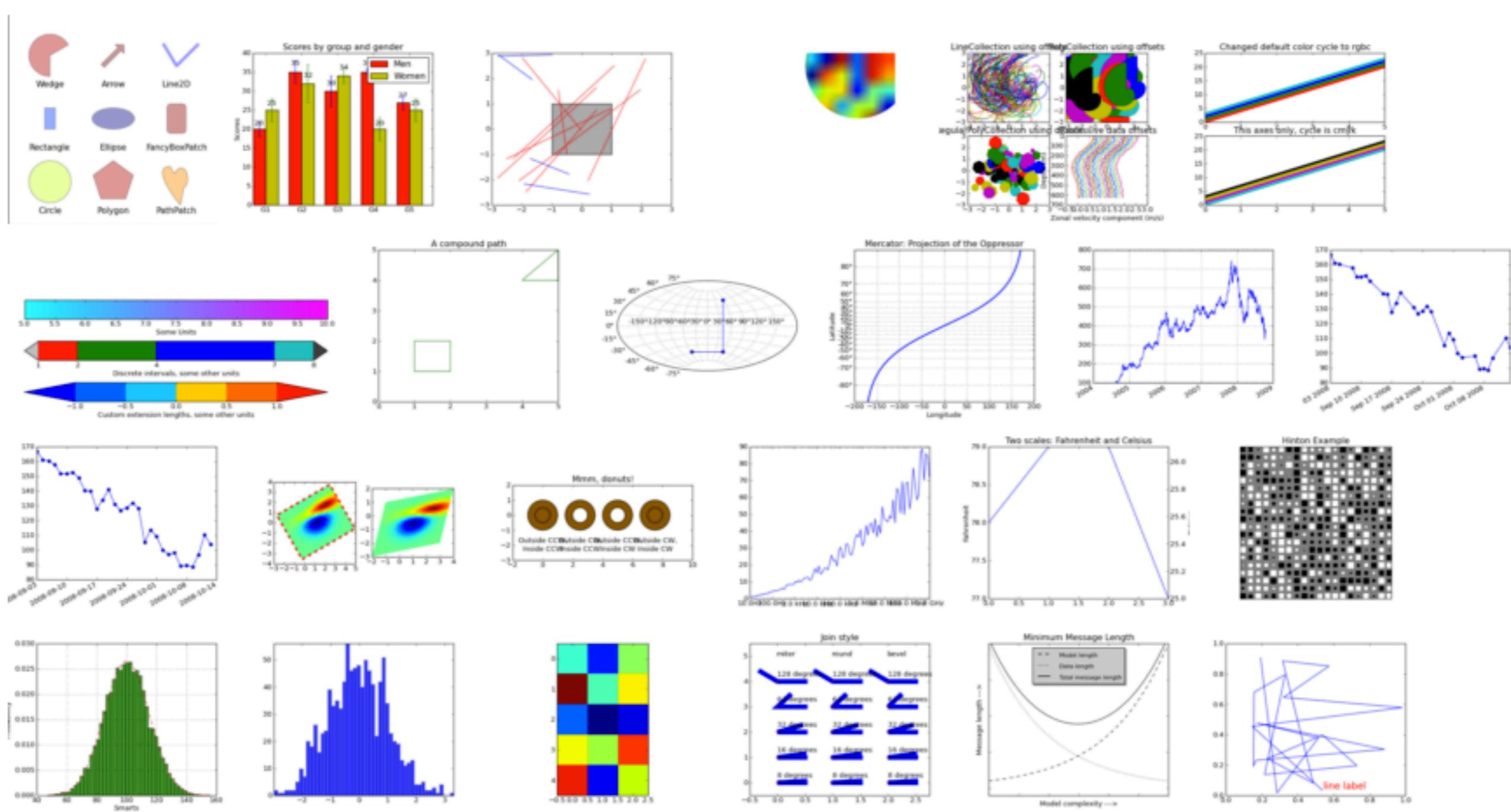
y, x = \
    scipy.mgrid[-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1,-gaussian_blur_kernel_half_width:gaussian_blur_kernel_half_width+1]

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();
```

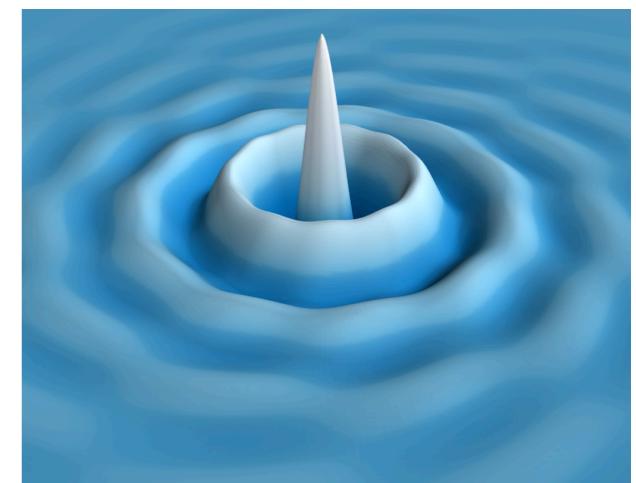
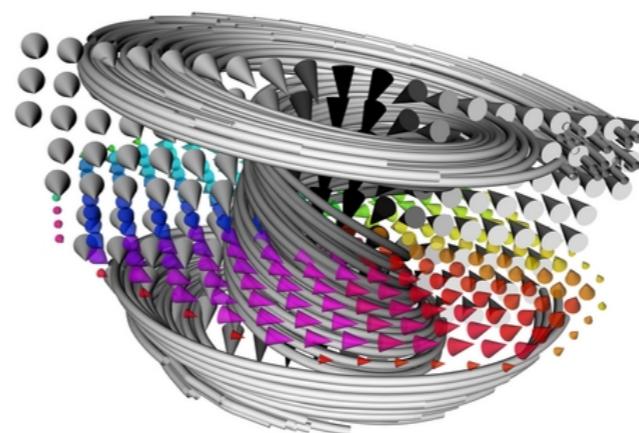
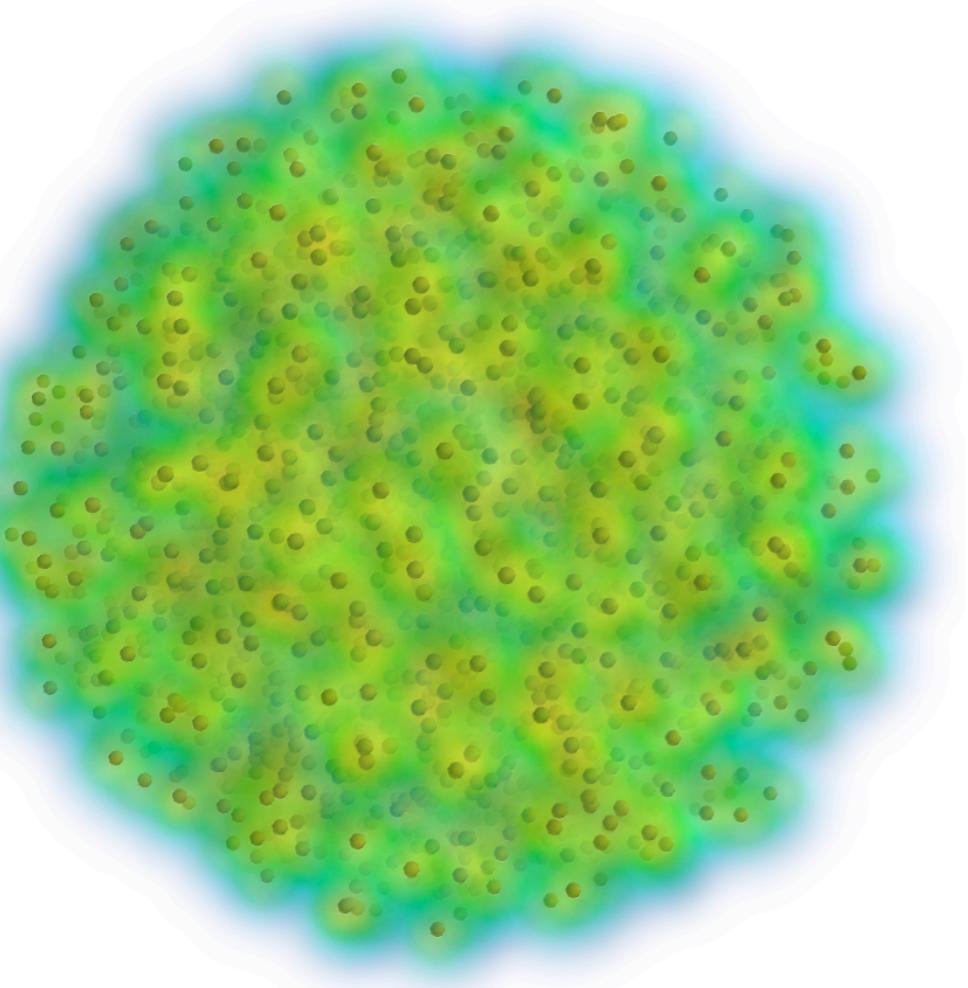
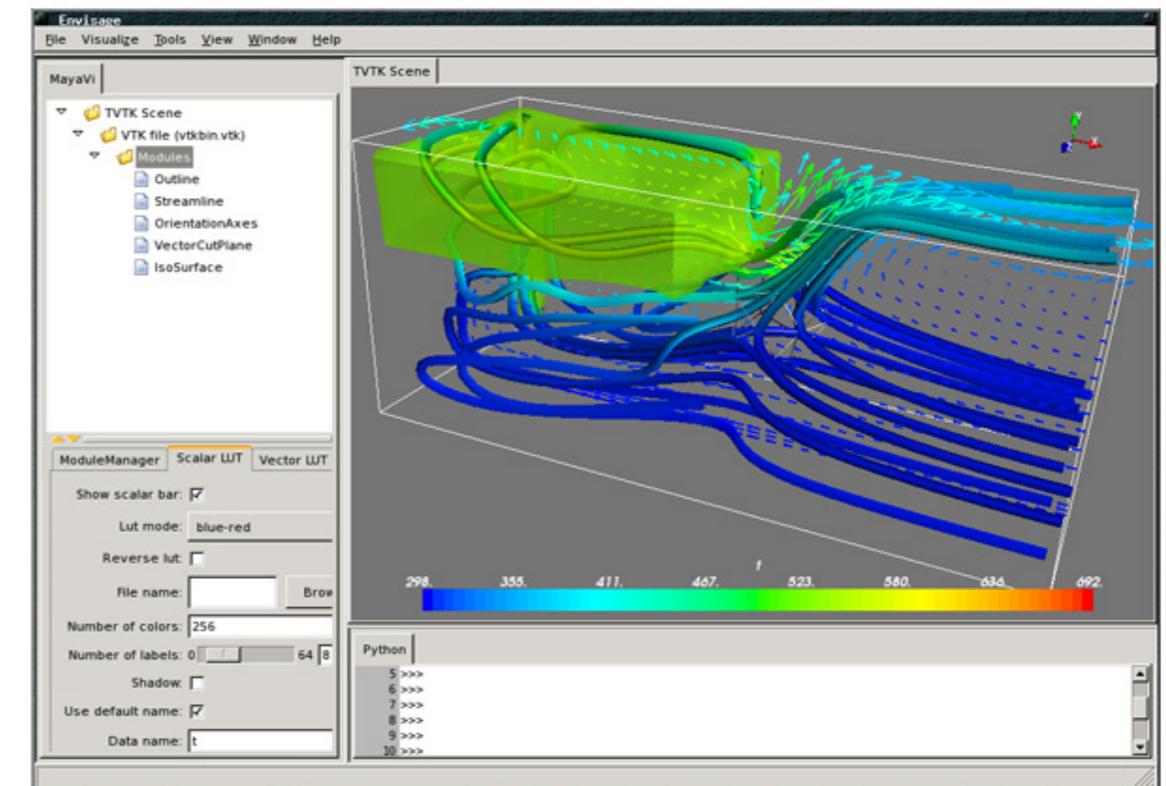
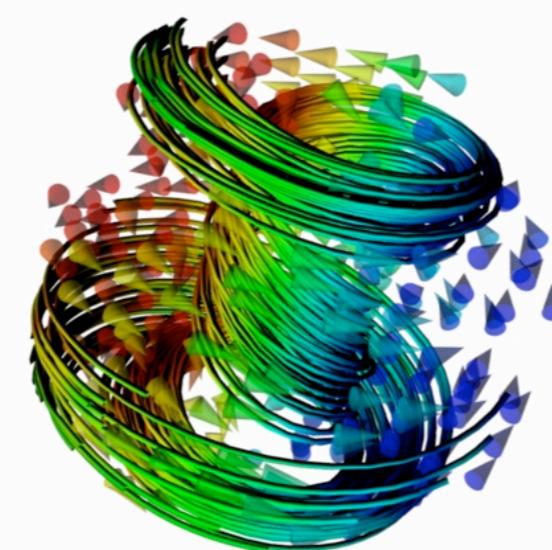
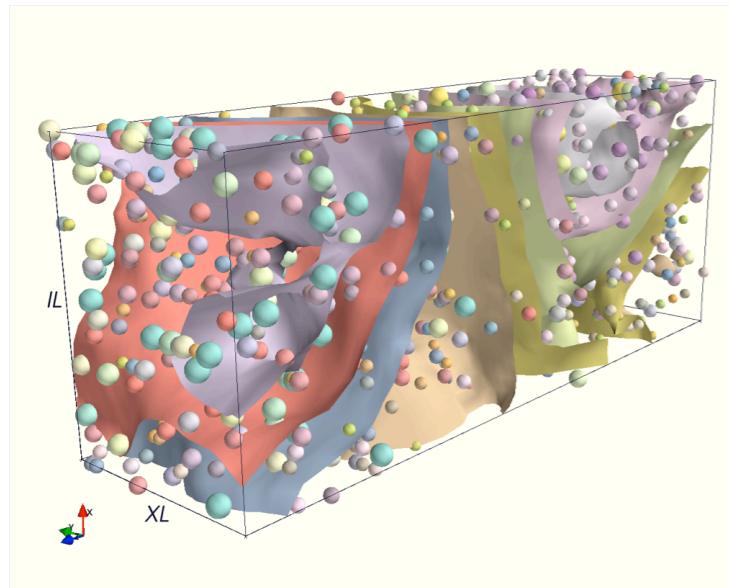
The output of the code is a grayscale heatmap titled "gaussian_blur_kernel". The x-axis ranges from 0 to 8, and the y-axis ranges from 0 to 8. A vertical color bar on the right indicates values from 0.005 to 0.040, with the highest intensity (white) at the center (4,4).

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



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

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



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

IⁿPython 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>]