

# PHYS13 WS2010 Final Project Writeup

Antonio Lorenzo

27 January 2010

## 1 A Particle in a Two-Dimensional Square Box

The particle in a box problem is perhaps the most simple problem in quantum mechanics. Likewise, a particle in a 2-D box also has a simple solution which is the product of the two 1-D boxes.

To solve the particle in a box, one simply solves the time-independent Schrödinger equation given by

$$\hat{H}\psi = E\psi. \quad (1)$$

$\hat{H}$  is the quantum mechanics Hamiltonian given by

$$\hat{H} = -\frac{\hbar^2}{2m}\hat{\nabla}^2 + V(\mathbf{r}, t). \quad (2)$$

In our case of a 2-D box with an infinite potential outside the box and zero inside, we simply solve the second-order ordinary differential equation

$$-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\psi_E(x) = E\psi_E(x). \quad (3)$$

The solutions,  $\psi_E(x)$ , are wavefunctions corresponding to the energy eigenstates of the system. My program plots these wavefunctions.

## 2 The Code

My main goal in writing the program was not to find a solution to the 2-D infinite square box, but to gain experience with scientific computing in C and Python. I wrote the code to generate the data in C just to gain experience with extending Python. I used the `math3d` package of the `matplotlib` library in Python to plot the C data. In my C code, I simply used the analytical solutions to the 2-D box, although it would not be too difficult to use one of the various ODE solvers written in C/C++ or to write your own solver.

## 2.1 The C Code

I wrote a Python extension in C that calculates the values to put into arrays to send to Python. The arrays contain all the information Python needs to make 3-D plots.

Below is the C code to create the square arrays used as the x and y axes. `create_Yarray( p )` creates the array for the y axis with p number of points between 0 and 1. `create_Xarray( p )` uses a similar process for the x axis.

```
/* Create the Y array from 0 to 1 with points number of points */
static PyObject* create_Yarray(PyObject* self, PyObject* args)
{
    int points;

    if(!PyArg_ParseTuple(args,"i",&points))
        return NULL;

    int dim[] = {points+1,points+1};
    int i,j;
    double *y;
    yarray =(PyArrayObject *) PyArray_SimpleNew(2,dim,NPY_DOUBLE);

    for(j=0; j<=points; j++)
    {
        for(i=0; i<=points; i++)
        {
            y =(double* ) PyArray_GETPTR2(yarray,i,j);
            *y= (double) i * (double) 1/(double) points;
        }
    }

    PyArray_Return(yarray);
}

/* Create the X array from 0 to 1 with points number of points */
static PyObject* create_Xarray(PyObject* self, PyObject* args)
{
    int points;

    if(!PyArg_ParseTuple(args,"i",&points))
        return NULL;
```

```

int dim[] = {points+1,points+1};
xarray =(PyObject *) PyArray_SimpleNew(2,dim,NPY_DOUBLE);
int i,j;
double *x;

for(j=0; j<=points; j++)
{
for(i = 0; i<=points; i++)
{
x =(double* ) PyArray_GETPTR2(xarray,i,j);
*x=(double) j * 1/(double) points;
}
}

PyArray_Return(xarray);
}

```

The code to create the z array just uses the analytical sine solutions to construct the z axis for a specific wavefunction given by the quantum numbers n and m.

```

/* Create the Z array using the X and Y arrays and the defined function */
static PyObject* create_Zarray(PyObject* self, PyObject* args)
{
int n,m;

if(!PyArg_ParseTuple(args,"ii",&n,&m))
return NULL;

numpy_int *dims;
dims = PyArray_DIMS(xarray);

int dim[] = {(int) dims[0],(int) dims[1]};
int l,k;
l = dim[0]-1;
k = dim[1]-1;
zarray =(PyObject *) PyArray_SimpleNew(2,dim,NPY_DOUBLE);

int i,j;
double *x, *y, *z;

```

```

for(j=0; j<=k; j++)
{
for(i=0; i<=l; i++)
{
x =(double* ) PyArray_GETPTR2(xarray,i,j);
y =(double* ) PyArray_GETPTR2(yarray,i,j);
z =(double* ) PyArray_GETPTR2(zarray,i,j);
*z = sin((double) m * PI * *x)* sin((double) n * PI * *y);
}
}
PyArray_Return(zarray);
}

```

## 2.2 The Python Code

The Python code utilizes the mplot3d package of the matplotlib library to make 3-D plots. The Python code just uses the C code to make the arrays and plot them. I've included user inputs to get specific wavefunctions as designated by n and m, along with the number of points to use per axis, and the type of plot to create. I've also included code to make the arrays with Python and the numpy package just to compare computing time with the C code.

```

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np
import square_well_arrays as s
import time
from pylab import title
from math import sin, pi

## Creates numpy arrays of a particle in an infinite 2D square well
## in a C extension and then plots the arrays
n = input("Enter a value for n (the x level of the wavefunction): ")
m = input("Enter a value for m (the y level of the wavefunction): ")
points = input("Enter the number of points per axis to use to plot the function: ")
plotype = raw_input("Enter a plot type (wireframe, surface, or contour): ")

fig = plt.figure()
ax = Axes3D(fig)
inc = 1./points

```

```

##Create the arrays in C and time
time.clock()
X = s.create_Xarray(points)
Y = s.create_Yarray(points)
Z = s.create_Zarray(n,m)
tc = time.clock()

#Create the arrays in Python just to compare the how much longer python takes
x2 = y2 = np.arange(0,1 + inc, inc)
X2, Y2 = np.meshgrid(x2,y2)
Z2 = np.zeros((points+1,points+1))
for j in range(points+1):
    for i in range(points+1):
        Z2[i][j] = sin(m * pi * X2[i][j]) * sin(n * pi * Y2[i][j])

tp = time.clock()
rtp = tp - tc

print 'The time to make the arrays using C code was', tc, 'seconds.'
print "The time to make the arrays using all Python code was", rtp, "seconds."
print "My C code is", rtp/tc, "times faster!"

if plottype=='wireframe':
    ax.plot_wireframe(X,Y,Z)
elif plottype=='surface':
    ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.winter)
elif plottype=='contour':
    ax.contour(X, Y, Z)
else:
    print "Please enter a valid type of plot."

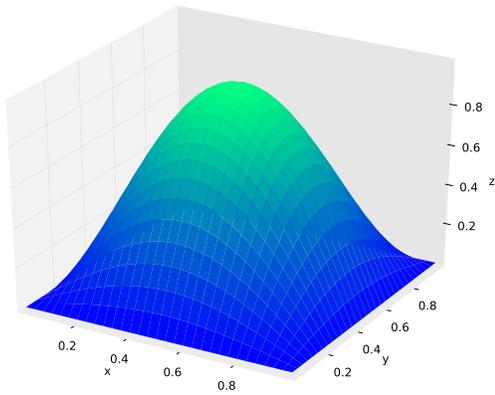
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.title("Particle in 2-D Infinite Well",fontsize=56)
plt.show()

```

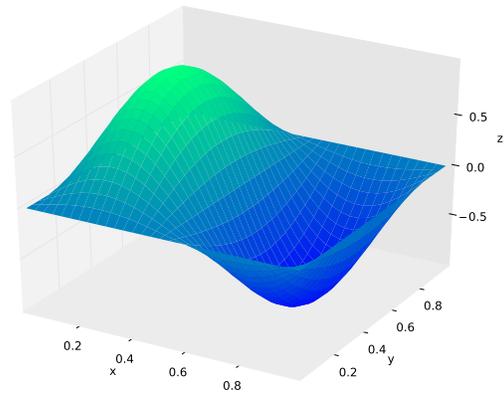
### 3 The Results

#### 3.1 Some Plots

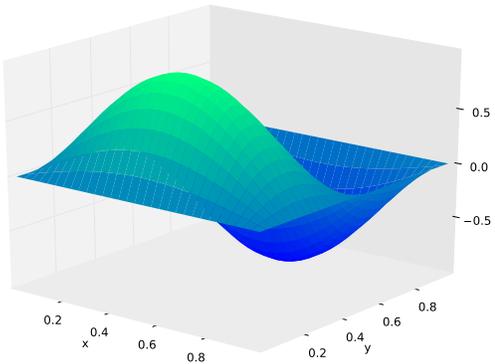
In the figures below are a few representative plots using the above program.



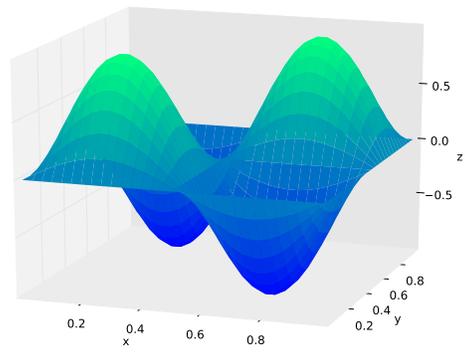
(a) Wavefunction for  $n=1, m=1$



(b) Wavefunction for  $n=1, m=2$



(c) Wavefunction for  $n=2, m=1$



(d) Wavefunction for  $n=2, m=2$

Figure 1: Some representative wavefunctions of a particle in a 2-D infinite box

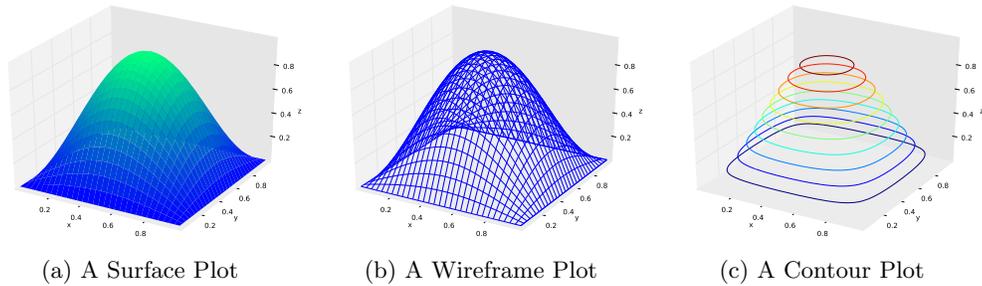


Figure 2: The different plot styles the program is capable of producing

### 3.2 Speed

The real advantage to writing an extension for Python in C/C++ is speed. Since C is a compiled and lower level language it is much faster at computational tasks than Python is. Using the built in time module of Python, 30 points along each axis, and running all simulations on my PC with Windows 7, 4GB of ram, and a 1.83GHz Intel Core Duo, my C code is on average 35x faster than similar Python code. Using the same method to create tables in Mathematica takes 140x longer than the C code and 4x longer than Python code. It is amazing to consider since Mathematica is programmed to make use of multi-core CPUs whereas my C and Python code can only use one core. This time savings could be incredibly helpful when dealing with very large datasets. As I am not a computer scientist, I'm sure the code for C and Python could be optimized, with the speed gap becoming even greater.

## 4 Conclusion

My program works well to create plots of the wavefunctions of a particle in a 2-D infinite square well. I did experience some problems with mplot3d, such as not being able to see a title, but I'm sure there are other graphing utilities for Python that work excellently. Mathematica and Matlab are much easier to create beautiful plots in, but Python is free and available to anyone. Writing C/C++ extensions for Python is useful if you need to do intense calculations or if you want to use a C library that someone else took the time to write and debug. For my purposes it was overkill, but still a good learning experience. Python seems to be the way to go when writing scientific programs with its ease of use, the possibilities to incorporate a number of tried and true C/C++ or Fortran libraries, and its availability leading to simple collaboration.