Computational Physics HW3

Luke Bouma

August 9, 2015

1 The Stefan-Boltzmann constant

In the angular frequency interval ω to $\omega + d\omega$, black bodies of unit area radiate a thermal energy per second equal to $I(\omega)d\omega$, where

$$I(\omega) = \frac{\hbar}{4\pi^2 c^2} \frac{\omega^3}{\left(e^{\hbar\omega/k_B T} - 1\right)}.$$

a) Show the total energy per unit area radiated by a black body is

$$W = \int_0^\infty \frac{x^3}{e^x - 1} dx.$$

We know the energy radiated per second is

$$I(\omega)d\omega = \frac{\hbar}{4\pi^2 c^2} \frac{\omega^3 d\omega}{\left(e^{\hbar\omega/k_B T} - 1\right)}.$$

Let $x = \hbar\omega/k_B T$. Then $x k_B T/\hbar = \omega \implies d\omega = dx(k_B T)/\hbar$

$$\begin{aligned} I(\omega)d\omega &= \frac{\hbar}{4\pi^2 c^2} \frac{k_B^4 T^4}{\hbar^4} \frac{x^3 dx}{(e^x - 1)} \\ \implies W = \int_{\omega=0}^\infty I(\omega)d\omega &= \frac{k_B^4 T^4}{4\pi^2 c^2 \hbar^3} \int_{x=0}^\infty \frac{x^3}{(e^x - 1)} dx. \end{aligned}$$

b) Write a program to evaluate the integral. We'll use analytic methods to express the integral as a sum (ideally, we would change of variables, but I couldn't figure out a good one). Note that

$$\begin{aligned} I = \int_{x=0}^\infty \frac{x^3}{e^x - 1} dx &= \int_{x=0}^\infty \frac{x^3 e^{-x}}{1 - e^{-x}} dx, \\ \text{but} \sum_{n=1}^\infty e^{-nx} &= \frac{e^{-x}}{1 - e^{-x}} \text{ (Geometric series, first term } e^{-x}) \\ \implies I &= \sum_{n=1}^\infty \int_0^\infty x^3 e^{-nx} dx \\ &= \sum_{n=1}^\infty \left(-n e^{-nx} \left. \frac{x^4}{4} \right|_0^\infty - \int_0^\infty 3x^2 \frac{e^{-nx}}{n} dx \right) \\ &= \sum_{n=1}^\infty \left(- \int_0^\infty 3x^2 \frac{e^{-nx}}{n} dx \right) \end{aligned}$$

and repeating the integration by parts (ignore the first terms, since they go to zero anyway): ,

$$\begin{aligned}
&= \sum_{n=1}^{\infty}\left(\int_0^{\infty} 6x\frac{e^{-nx}}{n^2}x\right)\\
&= 6\sum_{n=1}^{\infty}\int_0^{\infty}\frac{e^{-nx}}{n^3}dx\\
&= 6\sum_{n=1}^{\infty}\left.\frac{-e^{-nx}}{n^4}\right|_{x=0}^{\infty}\\
&= 6\sum_{n=1}^{\infty}\frac{1}{n^4}
\end{aligned}$$

```
def nastySum(maxIter):
    s = 0.
    n = 1
    while n < maxIter:
        s += 1/n**4
        n += 1
    return 6*s

for i in range(2000000, 2000001):
    print(nastySum(i))
```

gives $6\sum_{n=1}^{\infty} 1/n^4 \approx 6.493939402265166$, good to the analytic $\pi^4/4$ to 11 decimals.

c) Compute Stefan-Boltzmann constant to 3 decimals (SI units). We know (and experimentalists knew from observations) that $W = \sigma T^4 = \frac{k_B^4 T^4}{4\pi^2 c^2 \hbar^3}\int_{x=0}^{\infty}\frac{x^3}{(e^x-1)}dx$. Thus

$$\begin{aligned}
\sigma &= \frac{k_B^4}{4\pi^2 c^2 \hbar^3} I\\
&= 6.6703726 \times 10^{-8}\mathrm{W}/(\mathrm{K}^4\mathrm{m}^2)
\end{aligned}$$

which is good to experimental accuracy.

2 Gravitational pull of uniform sheet

a) Text Consider a square sheet of metal, with $L = 10$m sides and $m = 10$tonnes mass. Consider the gravitational force due to the plane felt by a point mass of 1kg at a distance of z from the center of the square, in the direction perpendicular to the sheet. Show the component of the force along the $z-$axis is

$$F_x = G\sigma z \iint_{-L/2}^{L/2} \frac{dxdy}{(x^2+y^2+z^2)^{3/2}},$$

for G the gravitational constant and σ the mass per unit area of the sheet.

Writing the math in a picture,

Like you're doing $dm = \sigma \, dx \, dy$. So

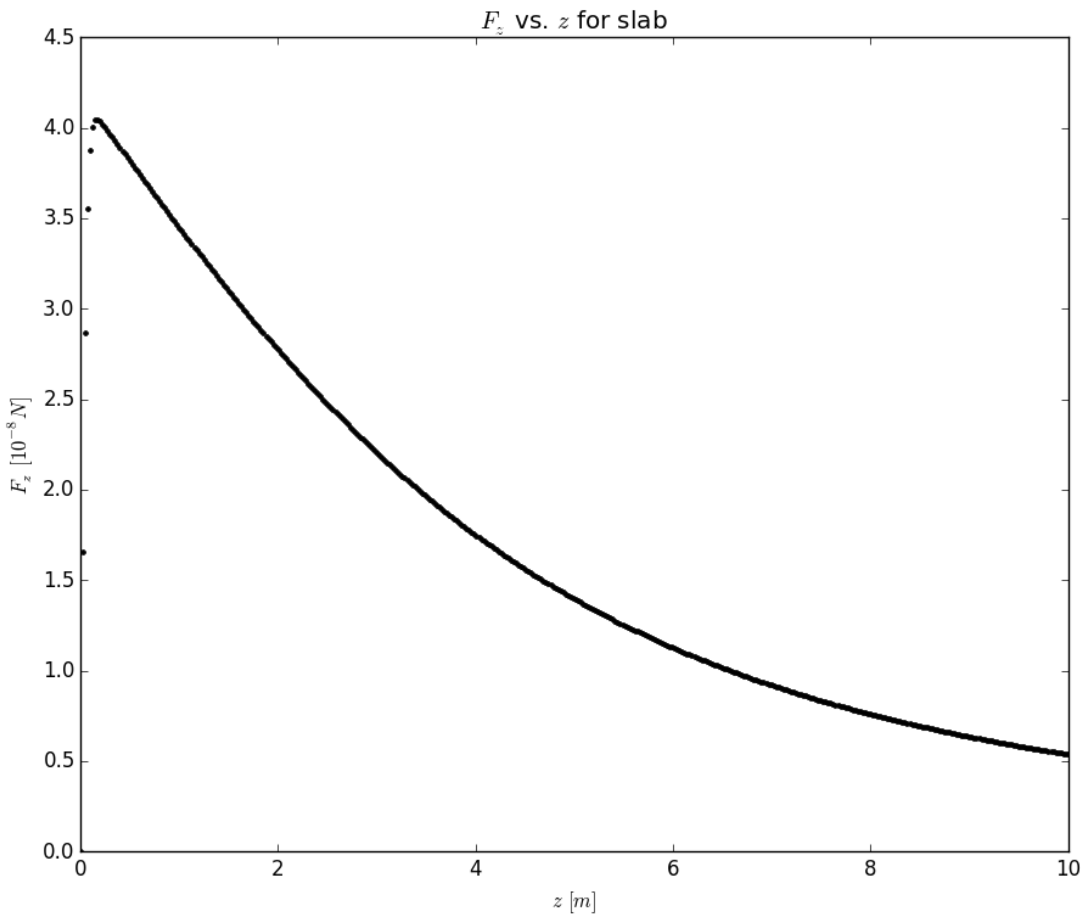
$$\hat{r} = \frac{\langle x, y, z \rangle}{(x^2+y^2+z^2)^{3/2}}$$

Newton, then

$$\vec{F} = G\int \frac{dm \, \vec{r}}{r^3} = G \iint \frac{\sigma \, dx \, dy \, \vec{r}}{r^3}$$

$$F_z = G\sigma z \int_{-L/2}^{L/2} \int_{L/2}^{L/2} \frac{dx \, dy}{(x^2+y^2+z^2)^{3/2}} \qquad \boxed{2}$$

b) Writing a program to calculate and plot the force as a function of z from $z = 0$ to $z =$ 10m. Calculate the integral with double Gaussian quadrature, with 100 sample points on each axis. We get (sampling 400 points in the z axis):



by using the program

```
from gaussxw import gaussxwab
from numpy import linspace, size
import matplotlib.pyplot as plt

G = 6.67384e-11
L = 10
area = 100          #SI, m^2
mass = 10*1000      #10tons, 1000kg per ton
sigma = mass/area

def f(x,y,z):
    return (x**2 + y**2 + z**2)**(-3/2)

N = 100
x, weight = gaussxwab(N, -L/2, L/2)
y, weight = gaussxwab(N, -L/2, L/2)

z = linspace(0, 10, 400)
for zi in range(size(z)):
    s = 0.
    for i in range(N):
        for j in range(N):
            s += weight[i] * weight[j] * \
                f(x[i], y[j], z[zi])
    plt.plot(z[zi], s*G*sigma*z[zi] * 1e8, 'k.')
    print(zi)

plt.xlabel("$z$ $[m]$")
plt.ylabel("$F_z$ $[10^{-8} N]$")
plt.title("$F_z$ vs. $z$ for slab")
plt.show()
```

c) So why does F_z drop to zero for small z? Well, run with $N = 100$, we are fitting the integral to a 100th order polynomial with points given by the zeros of the Legendre polynomial. This is not exact: the effect is amplified for small z because for a fixed N, when you get close enough to $z = 0$ you'll reach a point where the gravitational force diverges faster than is possible to fit with N zeros in the Legendre polynomial. *[Could be more quantitative]*

Indeed, testing with $N = 150$ reduces the error.

A nice thing to note about this calculation is that this is also approximately the gravitational pull exerted by a galaxy on objects above it.

3 `f(x)` to return $1 + \frac{1}{2}\tanh 2x$, then take derivative

Take the derivative with a central difference formula, on $-2 \le x \le 2$. Work out an analytic formula for the derivative, and plot the numerical vs analytic results. Plot one as a line, the other as dots. The analytic derivative derivation is:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\therefore \frac{d}{dx}(\tanh x) = \frac{d}{dx}\left(\sinh x \cdot (\cosh x)^{-1}\right)$$

$$= \frac{1}{\cosh x}\frac{d}{dx}(\sinh x) + \sinh x \frac{d}{dx}\left(\frac{1}{\cosh x}\right)$$

$$= \frac{1}{e^{x}+e^{-x}}\frac{d}{dx}\left(e^{x}-e^{-x}\right) + \left(e^{x}-e^{-x}\right)\frac{d}{dx}\left(\frac{1}{e^{x}+e^{-x}}\right)$$

$$= 1 + \left(e^{x}-e^{-x}\right)\cdot\frac{-1}{(e^{x}+e^{-x})^{2}}\cdot\left(e^{x}-e^{-x}\right)$$

$$= 1 - \frac{(e^{x}-e^{-x})^{2}}{(e^{x}+e^{-x})^{2}}$$

$$= 1 - \tanh^{2} x$$

---

so the code is

```
from math import tanh
from numpy import linspace, zeros
import matplotlib.pyplot as plt

def f(x):
    return 1 + tanh(2*x)/2

def fPnum(x, h):
    return (f(x+h/2)-f(x-h/2))/h

num = 200
x = linspace(-2, 2, num)
analyt = zeros(num)
h = 1e-8

for i in range(num):
    analyt[i] = 1-tanh(2*x[i])**2
    plt.plot(x[i], fPnum(x[i],h), 'k.')

plt.xlabel('$x$')
plt.ylabel('$d/dx(1+tanh(2x))$')
plt.plot(x, analyt, 'r-')
plt.show()
```

and the plot then becomes

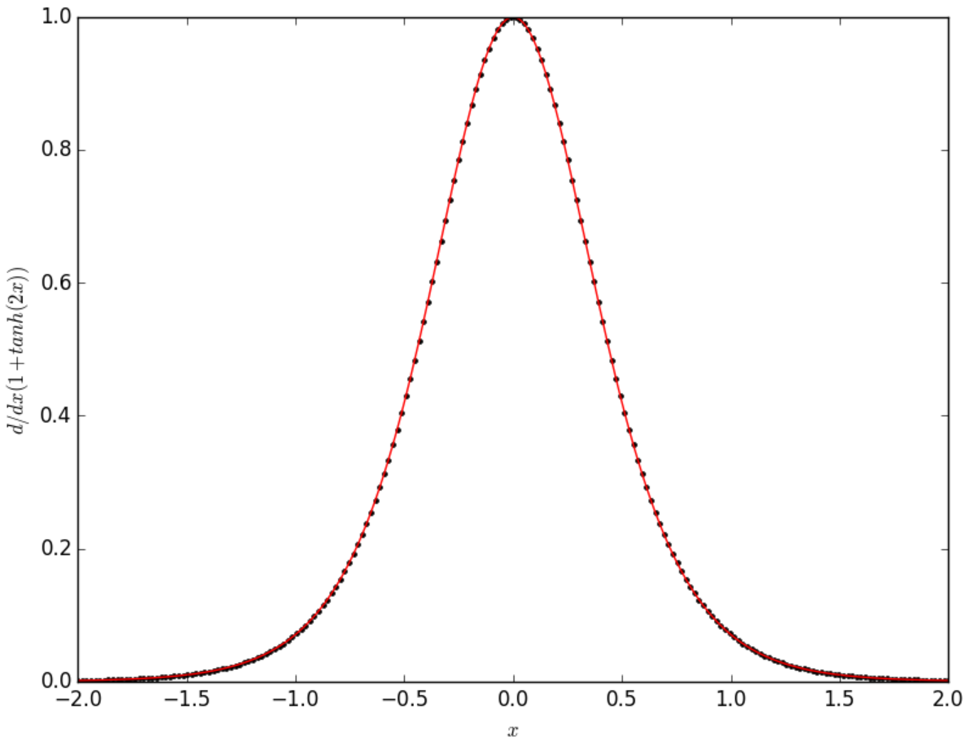


Figure 1: Dots are numerical results, and the line is the analytic result.

4 The gamma function, $\Gamma(a)$

The gamma function is defined as

$$\Gamma(a) = \int_0^\infty x^{a-1}e^{-x}dx$$

which has no closed form, but can be calculated by performing the integral numerically.

a) We plot $x^{a-1}e^{-x}$ as a function of x from $x = 0$ to $x = 5$, with three separate curves for $a = 2, 3, 4$, all on the same axes. Quick and dirty:

```
from numpy import linspace , size , zeros
import matplotlib.pyplot as plt
from math import exp

num = 100
a = [2,3,4]
x = linspace(0, 5, num)
integrand = zeros(num)
for a in a:
    powx = a-1
    for i in range(size(x)):
        integrand[i] = x[i]**powx * exp(-x[i])
    plt.plot(x, integrand)

plt.show()
```

gives

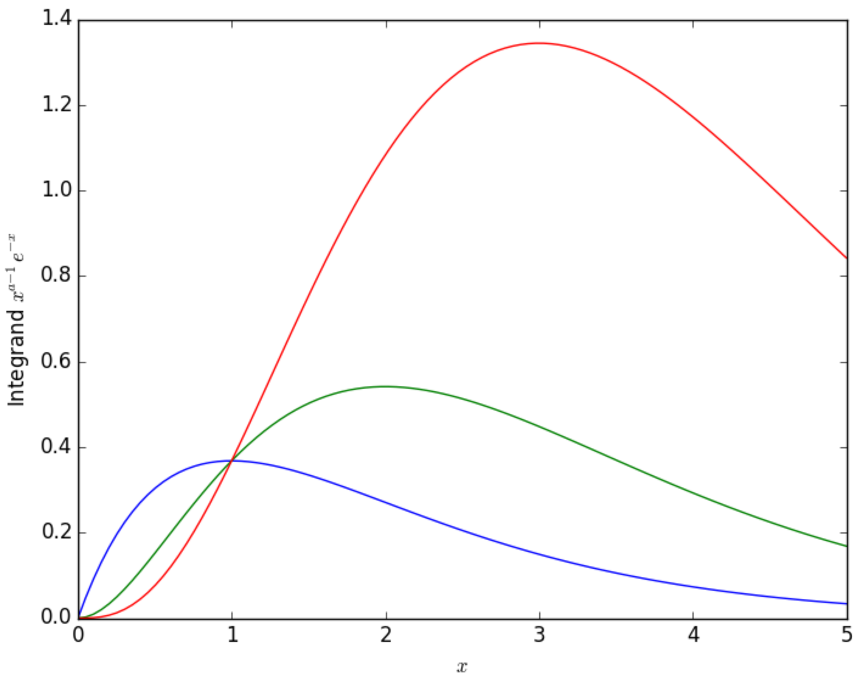


Figure 2: Blue: $a = 2$, green: $a = 3$, red: $a = 4$.

b) Showing analytically that the maximum is at $x = a - 1$, note that

$$\begin{aligned}\frac{d}{dx}\left(x^{a-1}e^{-x}\right) &= (a-1)x^{a-2}e^{-x} - x^{a-1}e^{-x} = 0\\ \implies (a-1)x^{a-2} &= x^{a-1}\\ \therefore\; a-1 &= x\end{aligned}$$

c) Use the change of variables $z = x/(c + x)$. The value of x that then gives $z = 1/2$ is

$$\begin{aligned}\frac{1}{2} &= \frac{x}{c+x}\\ \frac{1}{2}c &= \frac{1}{2}x\\ x &= c,\end{aligned}$$

so then we should pick $c = a - 1$ to center the peak at $z = 1/2$.

d) Rewriting $x^{a-1} = e^{(a-1)\ln x}$ to derive an expression for the integrand that suffers less numerical issues, we get that

$$\begin{aligned}
\Gamma(a) &= \int_0^\infty x^{a-1}e^{-x}dx \\
&= \int_0^\infty e^{(a-1)\ln x}e^{-x}dx \\
&= \int_0^\infty \exp((a-1)\ln x - x)dx
\end{aligned}$$

and then the change of variables $z = \frac{x}{x+(a-1)}$ or $x = \frac{(a-1)z}{1-z}$, we get

$$\begin{aligned}
\frac{dz}{dx} &= \frac{1}{x+(a-1)} - \frac{x}{(x+(a-1))^2} \\
dz &= \frac{a-1}{(x+(a-1))^2}dx \\
\implies dx &= \frac{(x+(a-1))^2}{a-1}dz
\end{aligned}$$

so

$$\Gamma(a) = \int_0^1 \exp\left((a-1)\ln\left(\frac{(a-1)z}{1-z}\right) - \frac{(a-1)z}{1-z}\right)\frac{(\frac{(a-1)z}{1-z}+(a-1))^2}{a-1}dz.$$

This is clearly crap. Both $z = 0,\ z = 1$ will give problems. So... what gives? This is generic to this kind of change of variables! I should ask someone about this... I'll take it as my single opt-out.

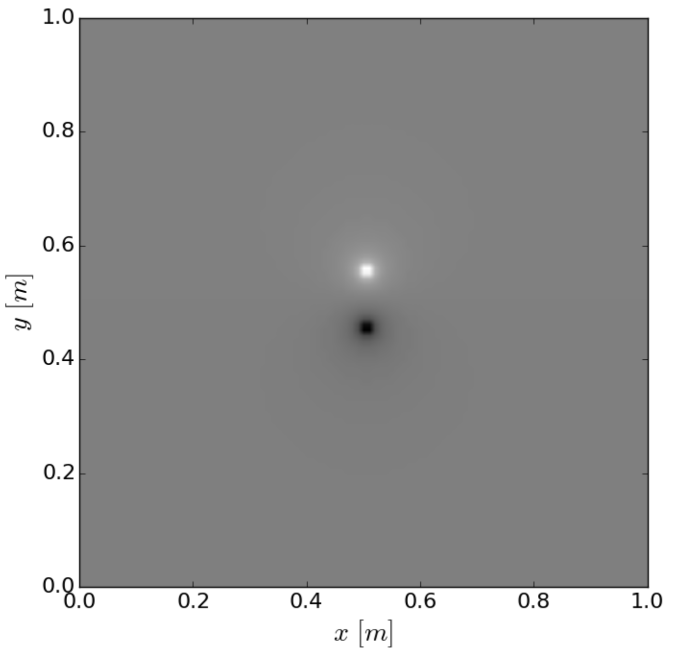
5 Electric field of charge distribution

We have a distribution of charges, and want to calculate the resulting electric field. We do this by calculating the electric potential, $\phi = q/4\pi\epsilon_0 r$, and taking its gradient $\boldsymbol{E} = -\nabla\phi$.

a) Two charges, of ±1C, 10cm apart. Calculate the resulting electric potential on a 1m by 1m square plane surrounding the charges, and passing through them. Do it at 1cm spacing points in a grid, and make a visualization using a density plot.

```
from math import pi, sqrt
from numpy import arange, copy, zeros
import matplotlib.pyplot as plt

eps0 = 8.854187817e-12
q1, q2 = 1, -1
dist = 0.1              #between charges [m]
prefac = 4*pi*eps0

num = 101               #Num grid pts, 1cm per sq
xMin, xMax, yMin, yMax = 0, 1, 0, 1
hx, hy = (xMax-xMin)/num, (yMax-yMin)/num
x, y = arange(xMin, xMax, hx), arange(yMin, yMax, hy)
```

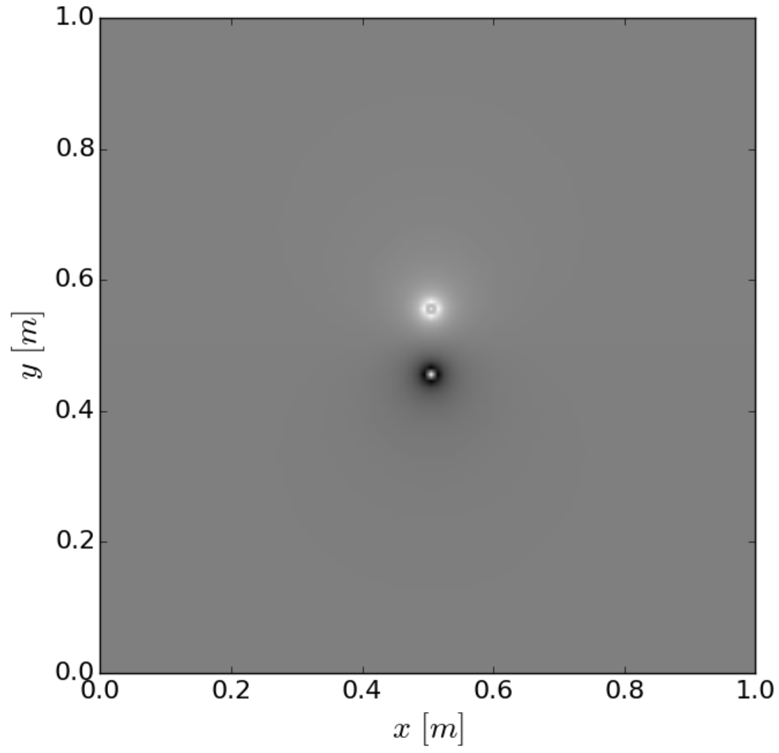
```
r1 = zeros([num, num], float)
r2 = copy(r1)

x1, y1 = xMax/2 + dist/2, yMax/2
x2, y2 = xMax/2 - dist/2, yMax/2

phi = copy(r2)
for i in range(num):
    for j in range(num):
        r1[i,j] = sqrt((x[i]-x1)**2+(y[j]-y1)**2)
        r2[i,j] = sqrt((x[i]-x2)**2+(y[j]-y2)**2)
        phi[i,j] = q1 / (prefac * r1[i,j]) + q2 / (prefac * r2[i,j])

plt.imshow(phi, origin='lower', extent=[xMin, xMax, yMin, yMax])
plt.xlabel('$x$␣$[m]$')
plt.ylabel('$y$␣$[m]$')
plt.gray()
plt.show()
```



which doesn't look *terrible*. Note that something funny is going on with the grid though. It can't be true 1cm squares, since if it were we would get a dividing-by-zero error on the grid points with the charges. With `print(x,y)` we see this is true – `arange` doesn't include the max, so as written in the code above, this is wrong.

Modifying this so that the grid is truly 101×101 points, with distance between squares of 1cm, we begin to see something that makes *numeric* sense:

```
num = 100              #Num grid pts, 1cm per sq
xMin, xMax, yMin, yMax = 0, 1, 0, 1
hx, hy = (xMax-xMin)/num, (yMax-yMin)/num
x, y = arange(xMin, xMax+hx, hx), arange(yMin, yMax+hy, hy)

r1 = zeros([num, num], float)
r2 = copy(r1)

x1, y1 = xMax/2 + dist/2, yMax/2
x2, y2 = xMax/2 - dist/2, yMax/2

phi = copy(r2)
for i in range(num):
    for j in range(num):
        r1[i,j] = sqrt((x[i]-x1)**2+(y[j]-y1)**2)
        r2[i,j] = sqrt((x[i]-x2)**2+(y[j]-y2)**2)
        phi[i,j] = q1 / (prefac * r1[i,j]) + q2 / (prefac * r2[i,j])
```



What we see here is a similar plot, but with diverging potential on the grid points with the charges. This makes sense: $\phi = q/4\pi\varepsilon_0 r$, which tends to ∞ at $r = 0$. We get the error `divide by zero encountered in double scalars phi[i,j] = q1 / (prefac * r1[i,j]) + ...` which means we're doing it right.

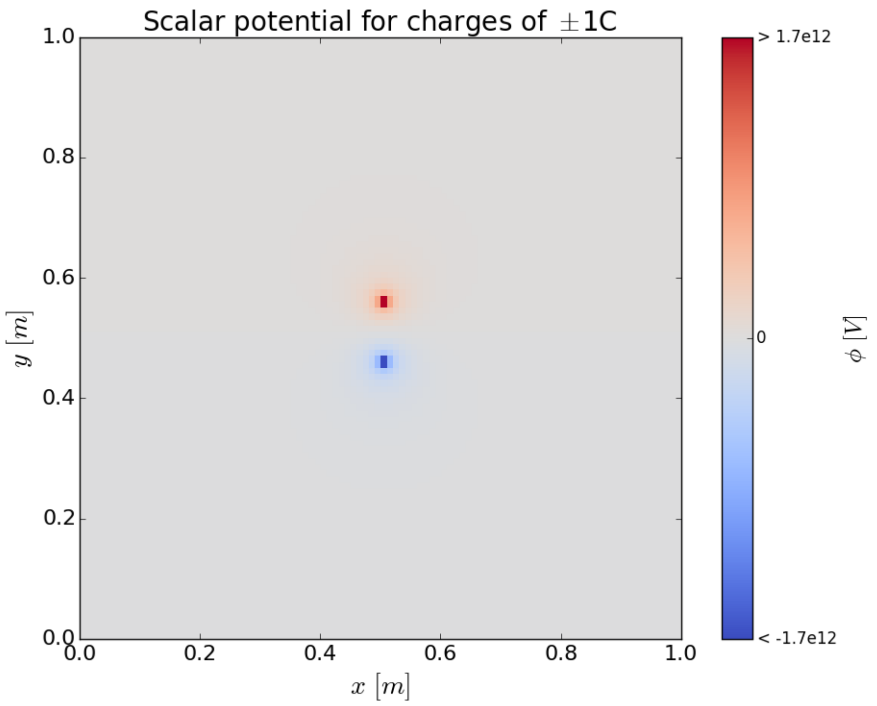
To work around this entirely, put the charges off-grid! Do that by changing

```
x1, y1 = xMax/2 + dist/2 + hx/2, yMax/2
x2, y2 = xMax/2 - dist/2 + hx/2, yMax/2
```

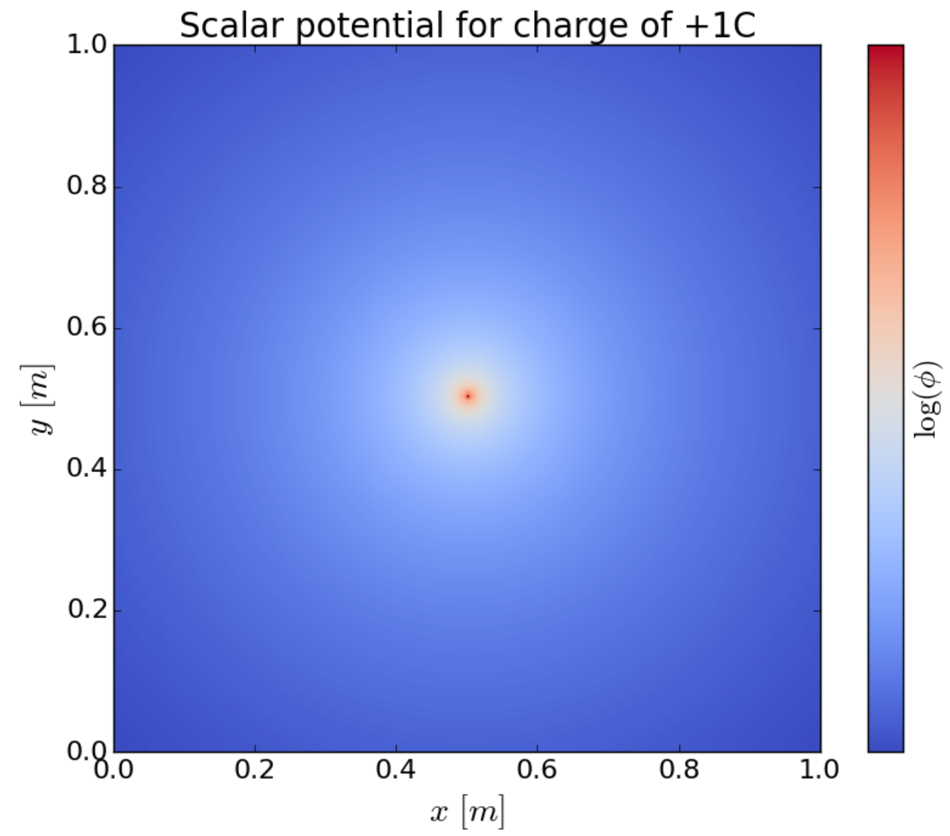
and then replotting with

```
fig, ax = plt.subplots()
cax = ax.imshow(phi, interpolation='nearest', cmap=cm.coolwarm,
                origin='lower', extent=[xMin, xMax, yMin, yMax])
ax.set_title('Scalar potential map for charges of $\pm$1C',
             fontsize=20)

#Add colorbar
cbar = fig.colorbar(cax, ticks=[min(phi), 0, max(phi)])
cbar.ax.set_yticklabels(['< -1.7e12 $V$', '0',
                         '> 1.7e12 $V$']) #vertical colorbar

plt.xlabel('$x$ $[m]$', fontsize=20)
plt.ylabel('$y$ $[m]$', fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

we get



which is the most sensible thing we've seen in a bit. As a final check (where we can take the log of the potentials), the plot for a single positive charge gives

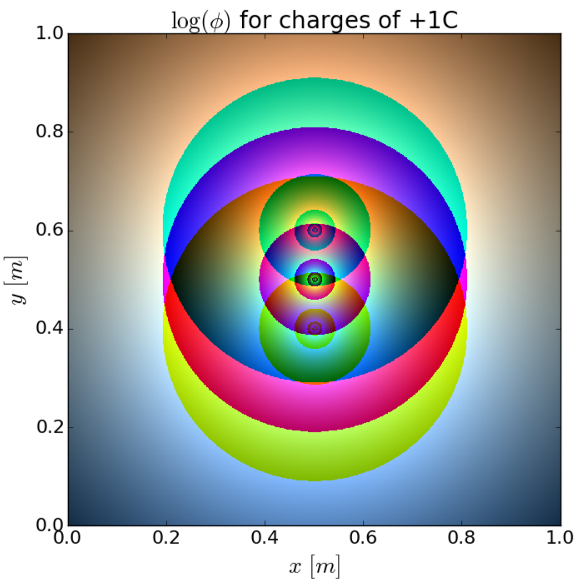


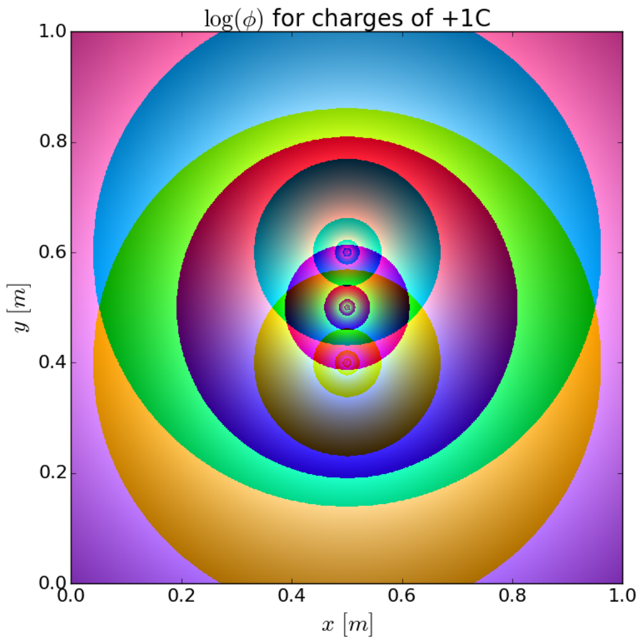
and the code we used to generate this was:

```
from math import pi, sqrt
from numpy import arange, copy, zeros, max, min, log
import matplotlib.pyplot as plt
from matplotlib import cm

eps0 = 8.854187817e-12
q1, q2 = 1, -1
dist = 0.1              #between charges [m]
prefac = 4*pi*eps0

num = 500               #Num grid pts, 1cm per sq
xMin, xMax, yMin, yMax = 0, 1, 0, 1
hx, hy = (xMax-xMin)/num, (yMax-yMin)/num
x, y = arange(xMin, xMax+hx, hx), arange(yMin, yMax+hy, hy)

r1 = zeros([num, num], float)
r2 = copy(r1)

#x1, y1 = xMax/2 + dist/2 + hx/2, yMax/2
x1, y1 = xMax/2 + hx/2, yMax/2
x2, y2 = xMax/2 - dist/2 + hx/2, yMax/2
```

```
phi = copy(r2)
for i in range(num):
    for j in range(num):
        r1[i,j] = sqrt((x[i]-x1)**2+(y[j]-y1)**2)
        r2[i,j] = sqrt((x[i]-x2)**2+(y[j]-y2)**2)
        phi[i,j] = q1 / (prefac * r1[i,j]) #\
                  # + q2 / (prefac * r2[i,j])

logPhi = log(phi)

fig, ax = plt.subplots()
cax = ax.imshow(logPhi, interpolation='nearest', cmap=cm.coolwarm,
                origin='lower', extent=[xMin, xMax, yMin, yMax])
ax.set_title('Scalar potential for charge of +1C',
             fontsize=20)

#Add colorbar
cbar = fig.colorbar(cax, ticks=[min(phi), 0, max(phi)])
cbar.ax.set_yticklabels(['< -1.7e12', '0',
                        '> 1.7e12']) #vertical colorbar
cbar.set_label('$\log(\phi)$', size=18)

plt.xlabel('$x$ $[m]$', fontsize=20)
plt.ylabel('$y$ $[m]$', fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

Cool. Since we're doing this for my own enjoyment, let's go on tangent-trip here. What happens if we vectorize the code, and make it artsy? This:

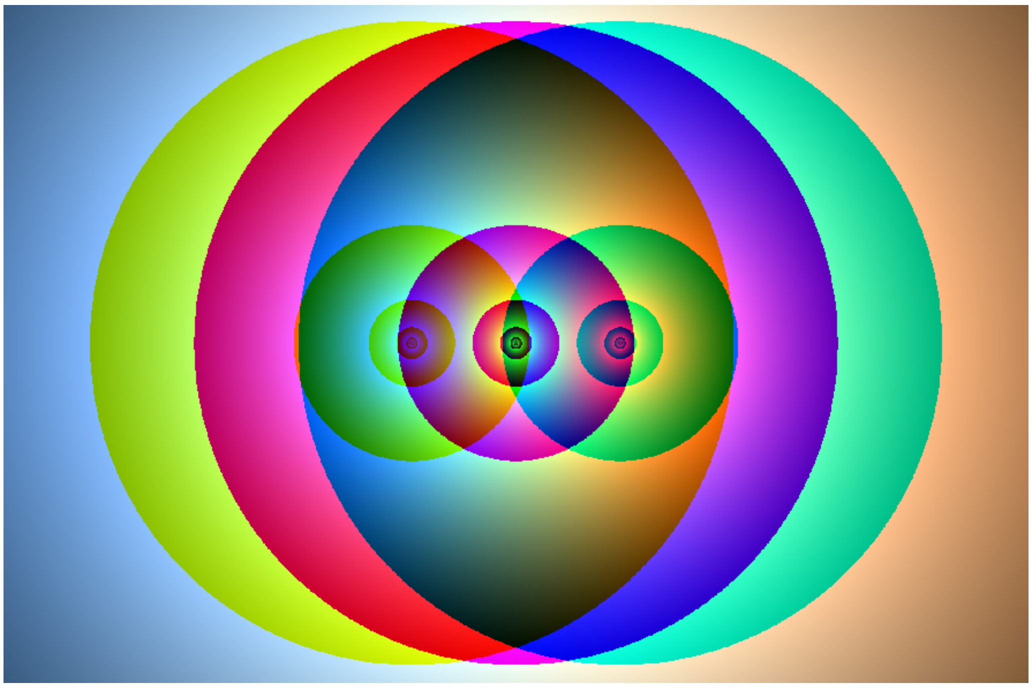




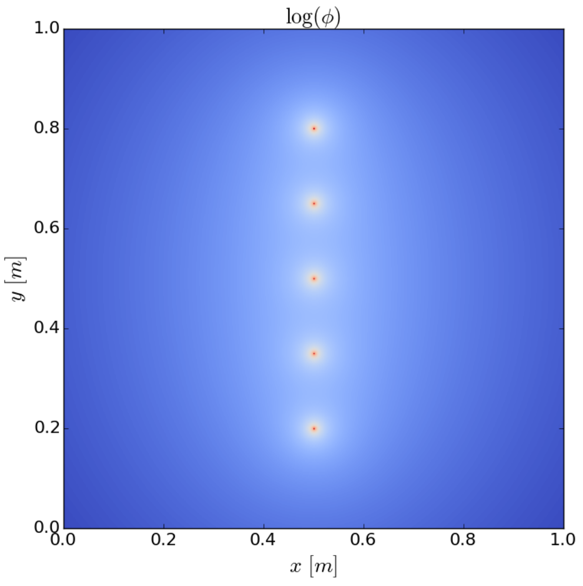
I'm actually surprised this works at all. My plotting code has a bug in it:

```
from math import pi, sqrt
from numpy import arange, copy, zeros, max, min, log, array
import matplotlib.pyplot as plt
from matplotlib import cm

eps0 = 8.854187817e-12
numCharges = 3
q = array([0.2, 1, 0.2])  # charge on point charges
dist = 0.1                # between charges [m]
prefac = 4*pi*eps0

num = 1000                # num grid pts per axis
xMin, xMax, yMin, yMax = 0, 1, 0, 1
h = array([(xMax-xMin)/num, (yMax-yMin)/num])

# grids for x,y. The +h_x, +h_y is for spacing
x, y = arange(xMin, xMax+h[0], h[0]), arange(yMin, yMax+h[1], h[1])

# r stores distance of grid point from each charge
r = zeros([num, num, numCharges], float)
phi = copy(r)

# rCharge stores positions of charges. Note it's best to
# place charges off-grid to avoid numerical issues as r->0
rCharge = zeros([2, numCharges], float)
for k in range(numCharges):
    rCharge[0, k], rCharge[1, k] = xMax/2 - dist + dist*k + h[0]/2, yMax/2

for k in range(numCharges):
    for i in range(num):
        for j in range(num):
            r[i,j,k] = sqrt((x[i]-rCharge[0, k])**2
                            +(y[j]-rCharge[1, k])**2)
            phi[i,j, k] += q[k] / (prefac * r[i,j,k])
    print(k)

logPhi = log(phi)

fig, ax = plt.subplots()
cax = ax.imshow(logPhi,
                interpolation='nearest',
                cmap=cm.coolwarm,
                origin='upper',
                extent=[xMin, xMax, yMin, yMax])
ax.set_title('$\log(\phi)$',
             fontsize=20)

plt.xlabel('$x$␣$[m]$', fontsize=20)
plt.ylabel('$y$␣$[m]$', fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

notice that I made `phi[i,j,k]` dependent on k unnecessarily. So I'm kind of surprised `imshow` doesn't return an error when I tell it to plot. But for whatever reason...



These look pretty good. Anyway, fixing the bug it becomes less sexy, but obviously it makes a lot more physical sense:



b) Find the electric field Okay! So now that we're done messing around and we have a nice vectorized version of the code, let's move on. We now want to visualize the field, where $\boldsymbol{E} = -\nabla\phi$. In other words,

$$(E_x, E_y) = \left(-\frac{\partial\phi}{\partial x}, -\frac{\partial\phi}{\partial y}\right).$$

The trick here is the the electric field has both magnitude and direction. To visualize it then, we'll use two density plots: one for magnitude, and one for direction. We'll calculate the derivatives with the central difference formula (modified for our grid):

$$\frac{\partial\phi}{\partial x} \approx \frac{\phi(x+h) - \phi(x-h)}{2h}, \quad \frac{\partial\phi}{\partial y} \approx \frac{\phi(y+h) - \phi(y-h)}{2h}.$$

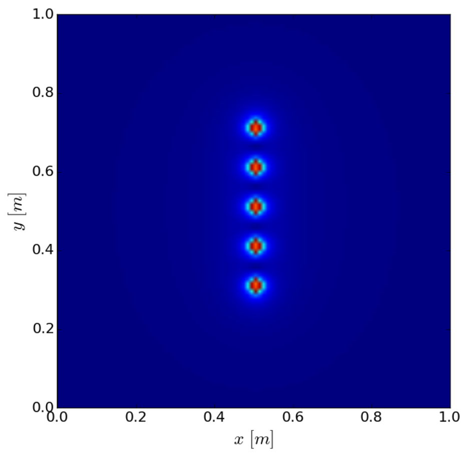
Then

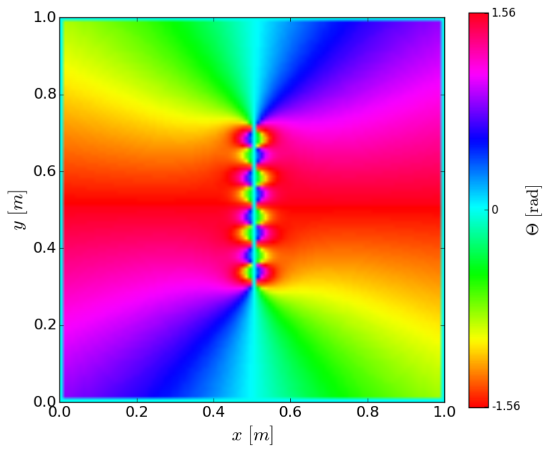
$$E = \sqrt{\left(\frac{\partial\phi}{\partial x}\right)^2 + \left(\frac{\partial\phi}{\partial y}\right)^2}, \quad \hat{E} = \frac{\vec{E}}{E} = \frac{\left(-\frac{\partial\phi}{\partial x}, -\frac{\partial\phi}{\partial y}\right)}{E}.$$

E is easy to visualize with a density plot. To visualize $\hat{E}$, note that $\tan\theta \equiv \hat{E}_y/\hat{E}_x$. So we can visualize the direction of the field in a density plot just by looking at

$$\theta = \tan^{-1}\left(\frac{\hat{E}_y}{\hat{E}_x}\right),$$

plotted with the `hsv` color scheme. We get:





where the left figure is the magnitude, and the right figure is the angle of the direction of E.

```
from math import pi, sqrt
from numpy import arange, copy, zeros, max, min, log, array, arctan
import matplotlib.pyplot as plt
from matplotlib import cm

eps0 = 8.854187817e-12
numCharges = 5
q = array([1, 1, 1, 1, 1])  # charge on point charges
dist = 0.1         # between charges [m]
prefac = 4*pi*eps0

num = 100              # num grid pts per axis
xMin, xMax, yMin, yMax = 0, 1, 0, 1
h = array([(xMax-xMin)/num, (yMax-yMin)/num])

# grids for x,y. The +h_x, +h_y is for spacing
x, y = arange(xMin, xMax+h[0], h[0]), arange(yMin, yMax+h[1], h[1])

# r stores distance of grid point from each charge
r = zeros([num, num, numCharges], float)
phi = zeros([num, num], float)
phi_x, phi_y = copy(phi), copy(phi) #could vectorize, but whatever
magE = zeros([num, num], float)
thetaE = copy(magE)

# rCharge stores positions of charges. Note it's best to
# place charges off-grid to avoid numerical issues as r->0
rCharge = zeros([2, numCharges], float)
for k in range(numCharges):
    rCharge[0, k], rCharge[1, k] = xMax/2 - 2*dist + dist*k + h[0]/2, yMax/2

for k in range(numCharges):
    for i in range(num):
        for j in range(num):
            r[i,j,k] = sqrt((x[i]-rCharge[0, k])**2
                            +(y[j]-rCharge[1, k])**2)
            phi[i,j] += q[k] / (prefac * r[i,j,k])
    print(k)

#Find derivatives, get E-field info:
for i in range(num):
    for j in range(num):
        if i != 0 and i!= num-1 and j != 0 and j != num-1:
            phi_x[i,j] = (phi[i+1,j] - phi[i-1,j])/(2*h[0])
            phi_y[i,j] = (phi[i,j+1] - phi[i,j-1])/(2*h[1])
            magE[i,j] = sqrt(phi_x[i,j]**2 + phi_y[i,j]**2)

            thetaE[i,j] = arctan( (-phi_y[i,j] / magE[i,j]) /
                                  (-phi_x[i,j] / magE[i,j]) )


plt.imshow(magE, origin = 'lower', extent=[xMin, xMax, yMin, yMax])
plt.xlabel('$x$ $[m]$', fontsize=20)
plt.ylabel('$y$ $[m]$', fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

and the plot script we wrote for the second one was:

```
fig, ax = plt.subplots()
cax = ax.imshow(thetaE,
                origin='lower',
                extent=[xMin, xMax, yMin, yMax],
                cmap=cm.hsv)

tMin, tMax = min(thetaE), max(thetaE)
cbar = fig.colorbar(cax, ticks=[-1.56, 0, 1.56])
cbar.ax.set_yticklabels(['-1.56', '0', '1.56'])
cbar.set_label('$\Theta$ $[\mathrm{rad}]$', fontsize=18)

plt.xlabel('$x$ $[m]$', fontsize=20)
plt.ylabel('$y$ $[m]$', fontsize=20)
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.show()
```

c) Continuous charge distribution, find electric field. Now let the charge density in Coulombs per meter-squared be

$$\sigma(x,y) = q_0 \sin\frac{2\pi x}{L} \sin\frac{2\pi y}{L} \theta_s(\boldsymbol{r} - \boldsymbol{r}_c),$$

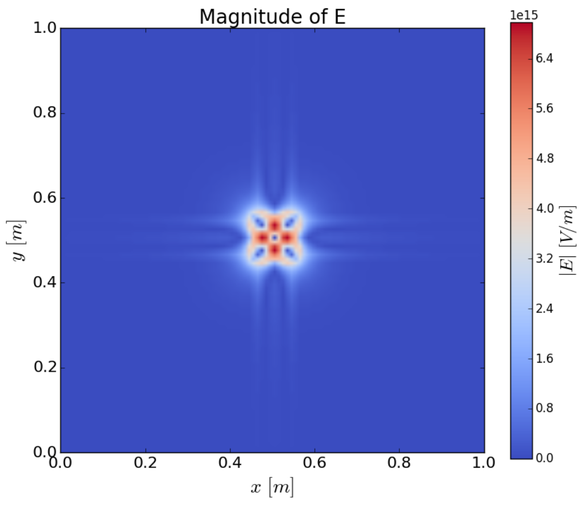
where $\mathbf{r}_c$ is the center of a square charge distribution, and θ_s is a step function (0 to 1) defined on the plane so that the charge steps up where we put the continuous charge distribution, and is zero everywhere else.

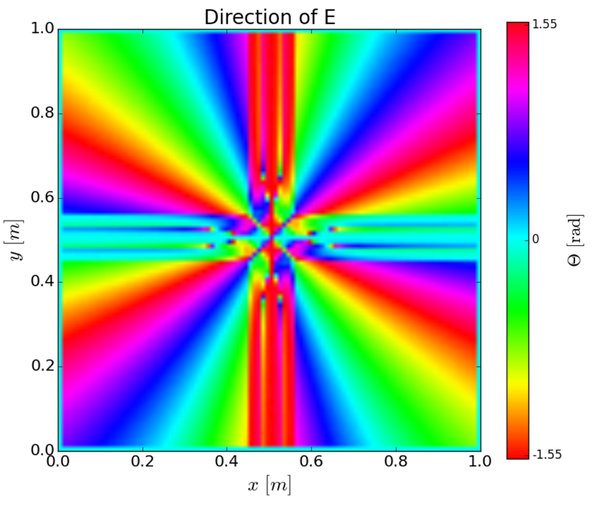
Okay. Then

$$\begin{aligned} \phi &= \frac{1}{4\pi\varepsilon_0}\int \frac{\sigma dA}{r} \\ \phi(x,y) &= \frac{q_0}{4\pi\varepsilon_0}\iint_0^1 \frac{\sigma(x,y)\cdot dxdy}{(x^2+y^2)^{1/2}} \end{aligned}$$

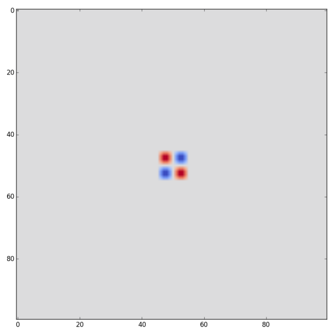
There might be a nice way to evaluate this thing numerically with double-gauss quadrature, just like in problem 2. I couldn't figure it out. I did the dumb thing: for every point (10,000 of them), we're saying 'okay, the potential at this point is this sum over all 9999 other points'. So literally, just treating the thing point-by-point (without even optimizing for the fact that we only have charge in the center of the grid), we need to do 10^8 operations to work out the potential for the full grid.

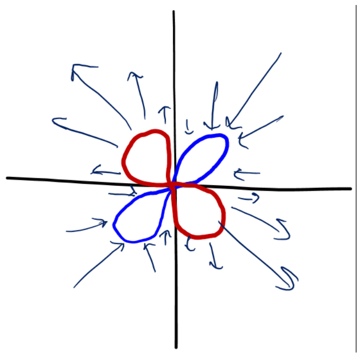
This takes like 4 minutes on my laptop, and yields:





Let's analyze these plots for a second. The charge density, and our estimate for the field direction can be crudely visualized as follows (n.b., the diagram on the right is a naive guess):





From this interpretation, the direction plot looks reasonable, but slightly different from what our first guess would be. Start with the first quadrant. Going counterclockwise, we see the angle Θ decrease, but with the expected 'flips' between quadrants, close to the charges.

The magnitude of the field map is also less-than-obvious for me. Why do those 8 bright lines exist where they do? Is that physical? Yea, I don't like it. The direction map wouldn't make any sense if the fields were off though, so I think it's reasonable to assume that any errors we might have introduced aren't screwing with the physics...

6 Gauss elimination

I don't have the textbook, or access to these equations/examples. The program is probably one that performs Gauss elimination on some input matrix. I feel comfortable with that problem (the algorithm is just: start with first row of the augmented matrix. Is the first coefficient of that row 1? If not, make it 1 by dividing through by the first element. Then you have a one in that position. Use this to eliminate all the first column coefficients below this one (arranging any duplicated–> zero rows to the bottom). Repeat this on the following rows).

7 QR algorithm

Here we're going to write a program to compute the eigenpairs (values and vectors) for a real symmetric matrix with the QR algorithm.

Firstly, what does this algorithm do? It expresses a real square matrix A in the form $A = QR$, where Q is an orthogonal matrix ($Q^T = Q^{-1}$, i.e., its columns and rows are orthonormal vectors), and R is upper triangular. We think of A as a set of N column vectors (each of length N):

$$A = \begin{pmatrix} | & | & | & \cdots \\ a_0 & a_1 & a_2 & \cdots \\ | & | & | & \cdots \end{pmatrix},$$

and then define a new set of vectors $u_0, ..., u_{N-1}$ and $q_0, .., q_{N-1}$ as follows:

$$\begin{aligned} u_0 &= a_0, & q_0 &= \frac{u_0}{|u_0|} \\ u_1 &= a_1 - (q_0 \cdot a_1)q_0 & q_1 &= \frac{u_1}{|u_1|} \\ u_2 &= a_2 - (q_0 \cdot a_2)q_0 - (q_1 \cdot a_2)q_1 & q_2 &= \frac{u_2}{|u_2|} \end{aligned}$$

and so on. The general formulas for calculating u_i and q_i are:

$$u_i = a_i - \sum_{j=0}^{i-1}(q_j \cdot a_i)q_j, \quad q_i = \frac{u_i}{|u_i|}.$$

7.1 Show, by induction or otherwise, that the vectors q_i are orthonormal.

To do this, we want to show that $q_i q_j = \delta_{ij}$, where $\delta_{ij} = 1$ if $i = j$, else it is 0. In the $N = 1$ case, this is true by inspection. So assume it's true for $N = k$, and we want to show it works for $N = k + 1$.

Note that up to a normalization factor, our assumption means that

$$q_i q_j = \left(a_i - \sum_{k=0}^{i-1} (q_k \cdot a_i) q_k \right) \cdot \left(a_j - \sum_{l=0}^{i-1} (q_l \cdot a_j) q_l \right) = \delta_{ij}$$

for $i, j \leq k$. We then want to consider two cases: $i = k+1, j = k$, and $i = j = k+1$. If we show the condition holds in both of these cases, then we'll have extended to the next logical case, confirming the inductive guess. In order to avoid wasting a bunch of time, let's skip it and get writing the code.

If we rearrange the definitions of the vectors, we'll have

$$\begin{aligned} a_0 &= |u_0| q_0 \\ a_1 &= |u_1| q_1 + (q_0 \cdot a_1) q_0 \\ a_2 &= |u_2| q_2 + (q_0 \cdot a_2) q_0 + (q_1 \cdot a_2) q_1, ... \end{aligned}$$

If we write all this out as a single matrix equation, we'll get

$$A = \begin{pmatrix} | & | & | & \dots \\ a_0 & a_1 & a_2 & \dots \\ | & | & | & \dots \end{pmatrix} = \begin{pmatrix} | & | & | & \dots \\ q_0 & q_1 & q_2 & \dots \\ | & | & | & \dots \end{pmatrix} \begin{pmatrix} |u_0| & q_0 \cdot a_1 & q_0 \cdot a_2 & \dots \\ 0 & a_1 & q_1 \cdot a_2 & \dots \\ 0 & 0 & |u_2| & \dots \end{pmatrix}.$$

But then this is precisely the QR decomposition!

7.2 Write a Python function that takes as its argument a real square matrix, and returns Q and R that form its QR decomposition.

```
from numpy import zeros, shape, dot
from numpy.linalg import norm

def QRdecomp(A):
    N = shape(A)[0]
    Q, u, R = zeros([N, N], float), zeros([N, N], float), \
              zeros([N, N], float)
    for i in range(N):
        u[:, i] = A[:, i] - sum((dot(Q[:, j], A[:, i])*Q[:, j]) \
                          for j in range(i))
        Q[:, i] = u[:, i] / norm(u[:, i])

    for i in range(N):
        for j in range(N):
            if i == j:
                R[i, i] = norm(u[:, i])
            if j > i:
                R[i, j] = dot(Q[:, i], A[:, j])

    return Q, R

N = int(input('Enter dimension of square matrix A: '))
```

```
A = zeros([N, N], float)
for i in range(N):
    for j in range(N):
        val = float(input('Enter matrix value in A: '))
        A[i, j] = val

Q, R = QRdecomp(A)

print('A - Q.R: ', A - dot(Q, R))
```

this code yields the zero matrix (up to numerical error) on the given test-case.

7.3 Using this function, write a program to calculate the eigenvalues and vectors of A

We want to continue the calculation until the magnitude of every off-diagonal elements is smaller than 10^{-6}. Recall the algorithm for finding eigenpairs with QR decomposition:

1. Given a $N \times N$ matrix A, create $N \times N$ matrix V and set $V = I$. Choose $\varepsilon > 0$ as desired for accuracy.
2. Calculate QR decomp as $A = QR$.
3. Update A to $A = RQ$.
4. Set $V = VQ$.
5. Check magnitude of all off-diagonal elements of A. If all are $< \varepsilon$, then stop. Otherwise, go to step 2.

There isn't much more to add to the program we already have from part b). We just tag on the code

```
V = eye(N)
eps = 1e-7
delta = 1.
while delta > eps:
    Q, R = QRdecomp(A)
    A, V = dot(R, Q), dot(V, Q)
    for i in range(N):
        for j in range(N):
            if i != j and abs(A[i, j]) < delta:
                    delta = A[i,j]
    print(delta)

print(A)
```

and then for our eigenvalue matrix we get the expected (21, -8, -3, 1),

```
[[  2.09999689e+01   3.00151296e-02   3.48747494e-05   1.99329587e-08]
 [  3.00151296e-02  -7.99996725e+00   2.90479623e-03   8.23483888e-06]
 [  3.48747494e-05   2.90479623e-03  -2.99999834e+00  -3.65798131e-03]
 [  1.99329814e-08   8.23483888e-06  -3.65798131e-03   9.99996655e-01]]
```

and for the eigenvectors we have

```
[[ 0.43111805   0.38446684   0.77461018  -0.25749086]
 [ 0.38401649  -0.43096919   0.25774167   0.77483296]
 [ 0.62384839  -0.52691446  -0.25741911  -0.51663308]
 [ 0.52676535   0.62354788  -0.51699487   0.25772582]]
```

I'll take it! So on this homework we skimped a bit in calculating the gamma function, and probably also in analyzing our results on the electric field of the charge distribution (probably because we spent so long screwing around on the earlier parts of the problem). We also went pretty heavily on the analytical side for the first problem, instead of doing brute-force numerics. We also didn't bother with the Gaussian elimination problem, but I'm fine with that.