Jupiter's red spot and 2D turbulence

Here we implement Onsager's vortex model for two-dimensional turbulent flow. Two-dimensional turbulence is quite different from turbulence in three dimensions. In three dimensions, large eddies stretch and fold into smaller eddies, leading to whirly motion on all scales. In two dimensions, one often finds that the swirly eddies combine into one big eddy. Jupiter's red spot and hurricanes are thought to be approximately explained as the result of 2D turbulence in their atmospheres.

Refer to the problem description. Do part (a) analytically.

Import packages.

(Note: no %pylab inline, for compatibility with our animations. Plots will appear in separate windows.)

In [ ]:
```
%pylab
from matplotlib import animation
from scipy import *
from scipy.integrate import odeint
```

Setting up the Hamiltonians, the equations of motion, and various routines for solving and plotting...

In [ ]:

```
def H(x,y,Gamma):
    Rij2 = (x[:,newaxis]-x[newaxis,:])*(x[:,newaxis]-x[newaxis,:])
\
            + (y[:,newaxis]-y[newaxis,:])*(y[:,newaxis]-y[newaxi
s,:])
    fill_diagonal(Rij2,1.)
    GammaGamma = tensordot(Gamma,Gamma, axes=0)
    H1 = -(1./(2.*pi))*tensordot(GammaGamma,log(Rij2))
    return H1

def Hi(i,x,y,Gamma):
    """
    Returns vortex energy for vortex #i; H = 1/2 sum(H(i)) because
of double counting in sum
    """
    Rij2 = (x[i]-x)**2 + (y[i]-y)**2
    Rij2[i]=1.
    H1 = -(Gamma[i]/pi)*(sum(Gamma[:i]*log((x[i]-x[:i])**2 + (y[i]-
y[:i])**2)) \
                                +sum(Gamma[i+1:]*log((x[i]-x[i+1:])**
2 + (y[i]-y[i+1:])**2)))
    return H1

def Enstrophy(x,y,Gamma):
    return dot(Gamma,x*x+y*y)

def DrDt(r,t):
    nVortices = int(len(r)/2)
    x = r[:nVortices]
    y = r[nVortices:]
    xij = x[:,newaxis]-x[newaxis,:]
    yij = y[:,newaxis]-y[newaxis,:]
    Rij2 = xij*xij+yij*yij
    fill_diagonal(Rij2,1.)
    dxdt = -(1/(2*pi))*dot(yij/Rij2,Gamma)
    dydt = -(1/(2*pi))*dot(-xij/Rij2,Gamma)
    return concatenate((dxdt, dydt))

def animate(n,Points,xOft,yOft):
    Points.set_offsets(transpose(array([xOft[n], yOft[n]])))
    return Points

def RunSimulation(x0,y0,tMax,nTimes):
    nVortices = len(x0)
    r0 = concatenate((x0,y0))
    times = linspace(0.,tMax,nTimes);
    rOft = odeint(DrDt,r0,times)
    xOft = rOft[:,:nVortices];
    yOft = rOft[:,nVortices:];
    return xOft, yOft

def RandomScatter(nVortices):
    Gamma = random.uniform(-1,1, size=nVortices)
    r = sqrt(random.uniform(size=nVortices))
```

```
    theta = random.uniform(-pi,pi,size=nVortices)
    x = r*cos(theta)
    y = r*sin(theta)
    return x, y, Gamma

def minDistance(x0,y0):
    distances = array([[sqrt((x0[i] - x0[j])**2 + (y0[i]-y0[j])**2)
for i in range(j)] for j in range(1,len(x0))])
    return min(array([min(row) for row in distances]))
```

(b) Run the simulation with one vortex with $\Gamma = 1$ and a `tracer vortex' with $\Gamma = 0$. Does the tracer vortex rotate around the test vortex with velocity $\mathbf{u}_0$?

In [ ]:
```
Gamma = array([1.,0])
x0 = array([0.,0.5])
y0 = array([0.,0.])
tMax=100.
nTimes=400

xOft, yOft = RunSimulation(x0,y0,tMax,nTimes)
fig = figure()
ax = axes( xlim=(-1.,1.), ylim = (-1.,1.) )
ax.set_aspect('equal')
Points = ax.scatter(xOft[0], yOft[0],c=Gamma,s=200)
show()
anim = animation.FuncAnimation(fig, animate, frames=nTimes, interva
l=1, repeat = True, fargs=(Points,xOft,yOft))
show()
```

Do parts (c) and (d) analytically.

(e) Start with $n = 20$ vortices with a random distribution of vortex strengths $\Gamma_i \in [-1, 1]$ and random positions $\mathbf{r}_i$ within the unit circle. Print the original configuration. Run for a time $t = 10$, and print the final configuration. Do you see the spontaneous formation of a giant whirlpool? Are the final positions roughly also randomly arranged? Measure and report the energy H of your vortex configuration. (Warning: Sometimes the differential equation solver crashes. Just restart again with a new set of random initial conditions.)

In [ ]:
```
x0, y0, Gamma = RandomScatter(20)
tMax = 10
nTimes = 400

print("Initial energy =", H(x0, y0, Gamma))

xOft, yOft = RunSimulation(x0,y0,tMax,nTimes)

print("Final energy =", H(xOft[-1], yOft[-1], Gamma))

fig = figure()
ax = axes( xlim=(-2,2), ylim = (-2,2) )
ax.set_aspect('equal')
Points = ax.scatter(xOft[0], yOft[0],c=Gamma,s=200)
show()
anim = animation.FuncAnimation(fig, animate, frames=nTimes, interva
l=1, repeat = True, fargs=(Points,xOft,yOft))
show()
```

Do parts (f), (g), and (h) analytically.

(i) Thermalize the Monte-Carlo simulation for your $n = 20$ vortices at temperature $\beta = 1/(k_B T) = 2$, report the final energy, and print out your configuration. Does the vortex simulation look similar to the ones you found in part~(e)? Thermalize again at temperature $\beta = -2$, report the energy, and print out your final configuration. Do the vortices separate out into clumps of positive and negative vorticity?

In [ ]:
```
def Metropolis(nSweeps,beta,x,y,Gamma):
    """
    Picks vortices at random, attempts to shift each into a new ran
dom position, and accepts
    according to a Metropolis rule at temperature given by beta=1/k
b T
    """
    nVortices = len(x)
    nTries = nSweeps*nVortices
    iTry = random.randint(0,nVortices,size=nTries)
    r = sqrt(random.uniform(size=nTries))
    theta = random.uniform(-pi,pi,size=nTries)
    xTry = r*cos(theta)
    yTry = r*sin(theta)
    rand = random.uniform(size=nTries)
    for trial in range(nTries):
        i = iTry[trial]
        Ei = Hi(i,x,y,Gamma)
        xi, yi = x[i],y[i]
        x[i], y[i] = xTry[trial], yTry[trial]
        Ef = Hi(i,x,y,Gamma)
        bDE = beta*(Ef-Ei)
        if ((bDE > 0.) and (exp(-bDE)<rand[trial])):
            x[i] = xi
            y[i] = yi
    return
```

In [ ]:
```
x0, y0, Gamma = RandomScatter(20)
beta = -2.
print("Initial energy = ", H(x0,y0,Gamma))
Metropolis(1000,beta,x0,y0,Gamma)
print("Final energy = ", H(x0,y0,Gamma))
print("Min distance = ", minDistance(x0,y0))
```

In [ ]:
```
figure()
ax = axes( xlim=(-1,1), ylim = (-1,1) )
ax.set_aspect('equal')
Points = ax.scatter(x0, y0,c=Gamma,s=200)
```

(j) Re-run the Monte Carlo simulation with $n = 20$ vortices until you find a configuration with a clear separation of positive and negative vortices (say, by examining the energy), but where the minimum distance between vortices is not too small (say, bigger than 0.01). Print this initial configuration of vortices. Run the simulation for $t = 10$ with this state as the initial condition. How many hurricanes do you find? Print out the final configuration of vortices.

In [ ]:
```
x0, y0, Gamma = RandomScatter(20)
beta = ...
print("Initial energy = ", H(x0,y0,Gamma))
Metropolis(1000,beta,x0,y0,Gamma)
print("Final energy = ", H(x0,y0,Gamma))
print("Min distance = ", minDistance(x0,y0))
```

In [ ]:
```
print("Initial energy =", H(x0, y0, Gamma))

xOft, yOft = RunSimulation(x0,y0,tMax,nTimes)

print("Final energy =", H(xOft[-1], yOft[-1], Gamma))

fig = figure()
ax = axes( xlim=(-2,2), ylim = (-2,2) )
ax.set_aspect('equal')
Points = ax.scatter(xOft[0], yOft[0],c=Gamma,s=200)
show()
anim = animation.FuncAnimation(fig, animate, frames=nTimes, interva
l=1, repeat = True, fargs=(Points,xOft,yOft))
show()
```