Lab 8

EOSC 511

Christopher Rodell ¶

I. Do FIRST this subset of problem 3. Set loop=True, and compare the efficiency of the SOR method (currently implemented, suggest overrelaxation coefficient of 1.7) and the Jacobi iteration (you need to implement, suggest coefficient of 1; I find 1.7 unstable). Also compare to indexing the loops and doing Jacobi interation by setting loop=False. You can time functions using %timeit

See qg.py script and look at functions qg_cr and relax_jacobi. Sadly could not get this to work properly. I was having issue get everything to converge. I kept getting nan values as the iteration moves forward. Seemed to start well but then flopped :(Not sure where I was going wrong. I had attempted this by resourcing code I found off Wikipedia.

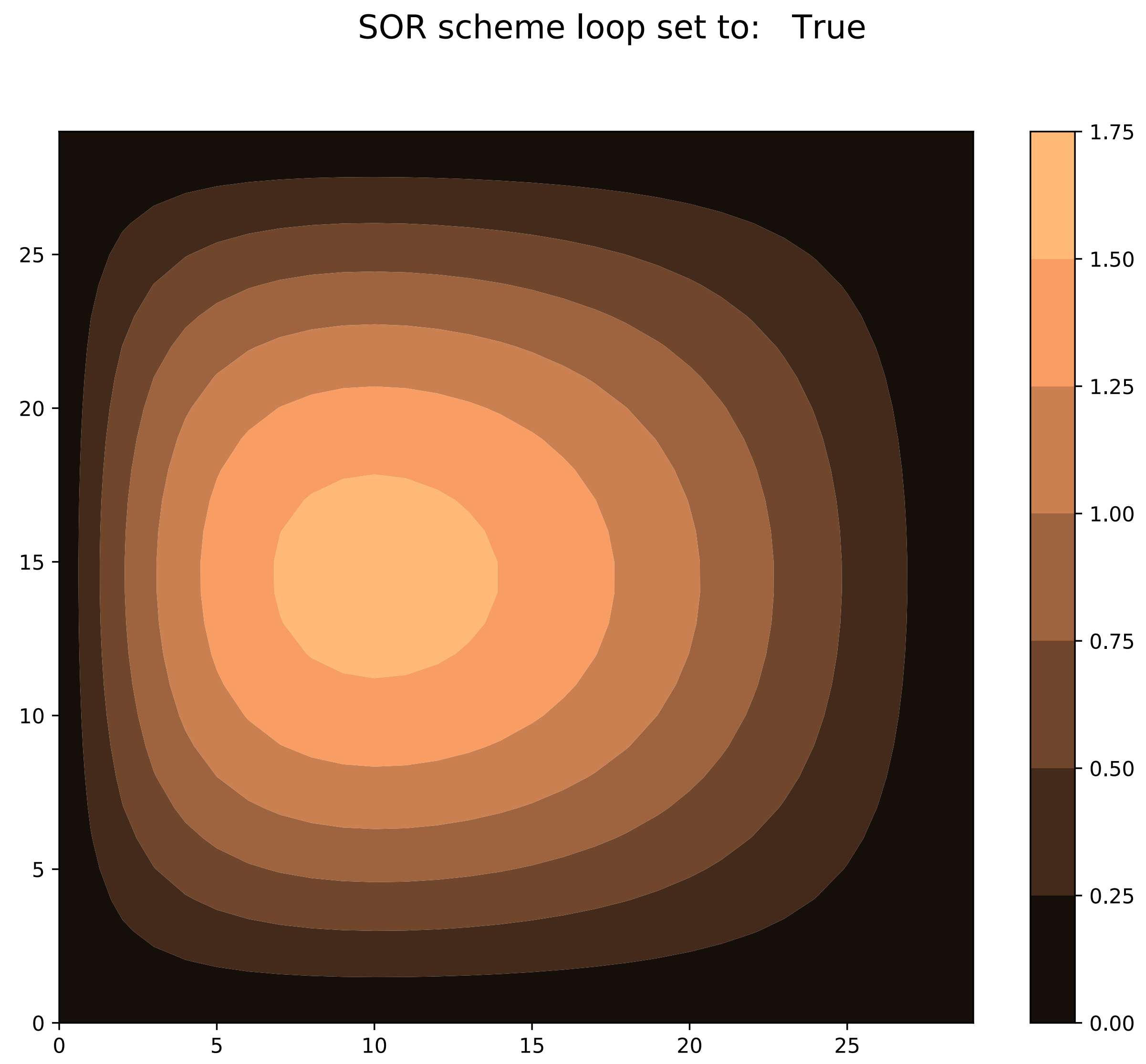
To answer the question regarding computational cost. Based on the Cost of Schemes Table. The SOR is a bit more computational expense than the simpler Jacobi method. Also, the SOR improves convergence considerably as compared to the Jacobi method.

Looking at the SOR method using the loop set to true or false we find that when set to true the function takes over one second longer to run than when set to false (with the default time step and coefficients). and also convergence to something logical that plots. Also when loop set to true the array converges to a large max value than when values 1.75 and 1.6 respectively.

Plot using Successive Over-Relaxation with loop set to True

In [1]:
```
import context
# import numlabs.lab8.qg as qg
import sys
import matplotlib.pyplot as plt
import numpy as np
import time


import qg as qg
# psi = qg(10*86400)
# param = param()

t = 10*86400
start = time.time()
qg.main(t, True, 'param')
end = time.time()
print(end - start, "Time elapsed for loop set to True")
```

```
******************************
context imported. Front of path:
/Users/rodell/repos/numeric_students
back of path: /Users/rodell/.ipython
******************************

through /Users/rodell/repos/numeric_students/numeric_notebooks/labs_7b_
8_10/context.py
Loop set to:   True
Physical Parameters:
a =  2000000.0
b =  2000000.0
epsilon =  0.00019362780879549283
wind =  -1.0
vis =  0.09978365848305341
time =  31114.932813320742
boundary_layer_width_approx =  199567.31696610682

Simulation Parameters:
nx =  30
dx =  0.034482758620689655
ny =  30
dt =  43200.0
maximum iterations =  50
tolerance = 0.005
coeff =  1.0
```

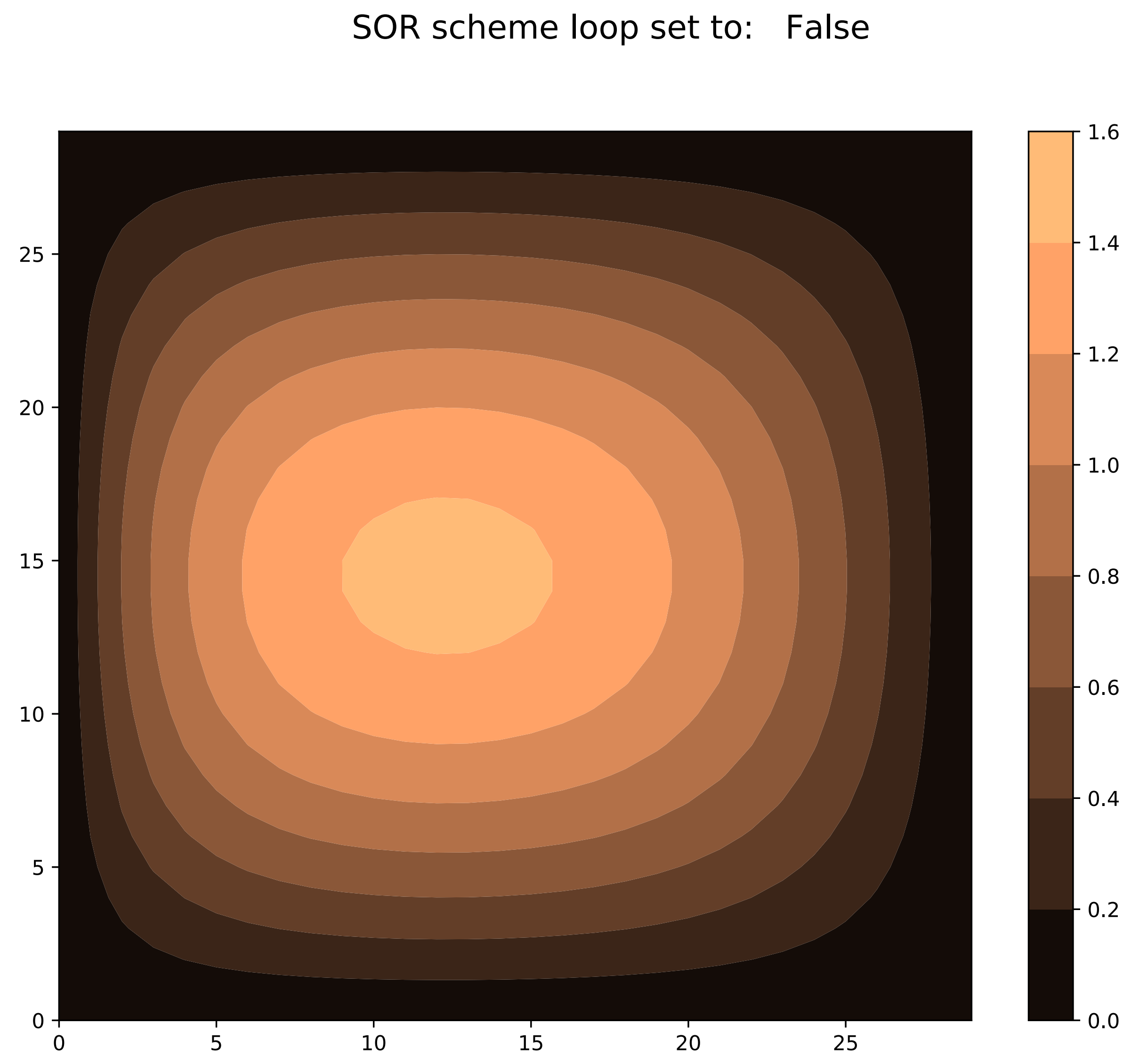


```
1.8421120643615723 Time elapsed for loop set to True
```

Plot using Successive Over-Relaxation with loop set to False

In [2]:
```
phys = 'param'
start = time.time()
qg.main(t, False, phys)
end = time.time()
print(end - start, "Time elapsed for loop set to False")
```

```
Loop set to:   False
Physical Parameters:
a =  2000000.0
b =  2000000.0
epsilon =  0.00019362780879549283
wind =  -1.0
vis =  0.09978365848305341
time =  31114.932813320742
boundary_layer_width_approx =  199567.31696610682

Simulation Parameters:
nx =  30
dx =  0.034482758620689655
ny =  30
dt =  43200.0
maximum iterations =  50
tolerance = 0.005
coeff =  1.0
```



```
0.3382148742675781 Time elapsed for loop set to False
```

Plot using the Jacobi iteration See qg.py script and look at functions qg_cr and relax_jacobi. Sadly could not get this to work properly. I was having issue get everything to converge. I kept getting nan values as the iteration moves forward. Seemed to start well but then flopped :(Not sure where I was going worng. I had attempted this by resourcing code I found off Wikipedia.

```
In [3]: ## See qg.py script if you want to see what I had tried
        # qg.qg_cr((time, False))
```

Then using the most efficient scheme, choose one parameter of the problem (eg depth, width of the ocean, vertical viscosity, wind stress, latitude) vary it (3 or 4 choices) and compare the solutions.

I am using the SOR scheme set to false as the most efficient scheme(I was unable to get the Jacobi to work properly and. the code ran fast when loop on false). Also, I chose SOR because SOR improves convergence considerably as compared to the Jacobi method.

$$

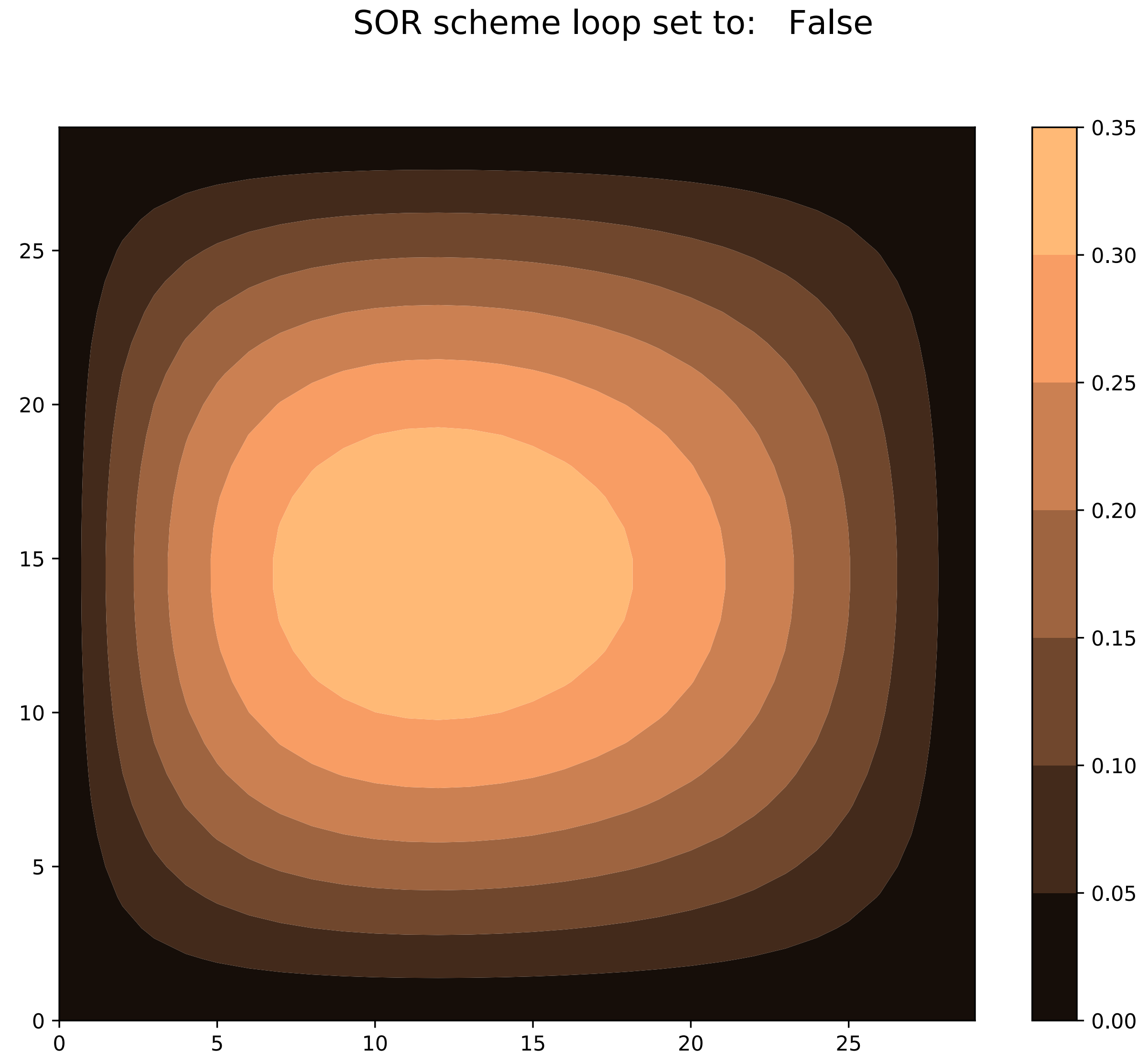
\

$$

The parameter I changed was depth from 500 meters to 100 meters. This made epsilon larger and overall made the stream function converge to a smaller number.

In [4]:
```
import qg_cr as qg_cr

start = time.time()
qg_cr.main(t, False, 'param_cr')
end = time.time()
print(end - start, "Time elapsed for loop set to False")
```

```
Loop set to:   False
Physical Parameters:
a =  2000000.0
b =  2000000.0
epsilon =  0.0009681390439774642
wind =  -1.0
vis =  0.49891829241526703
time =  31114.932813320742
boundary_layer_width_approx =  997836.5848305341

Simulation Parameters:
nx =  30
dx =  0.034482758620689655
ny =  30
dt =  43200.0
maximum iterations =  50
tolerance = 0.005
coeff =  1.0
```



```
0.3475470542907715 Time elapsed for loop set to False
```

III. Continue to use the most efficient scheme. Set the wind-stress to zero. Initialize the stream-function (both psi- 1 and psi 2) with a blob of fluid somewhere over to the east (say a Gaussian 3/4 of the way across with a radius of several grid points). Make sure that the boundary condition (psi = 0 on all walls) is enforced.
The blob should move west like a Rossby wave (the natural wave in this problem, a one-way wave, only moves west). Compare the time-scale of the westward movement to the time-scale of reaching steady state in the wind-driven case.
(Wikipedia entry on Rossby waves is currently okay if you want a quick read about them. The time scale given for Rossby waves across the ocean is for baroclinic (slow) waves. You are doing barotropic (fast) waves).