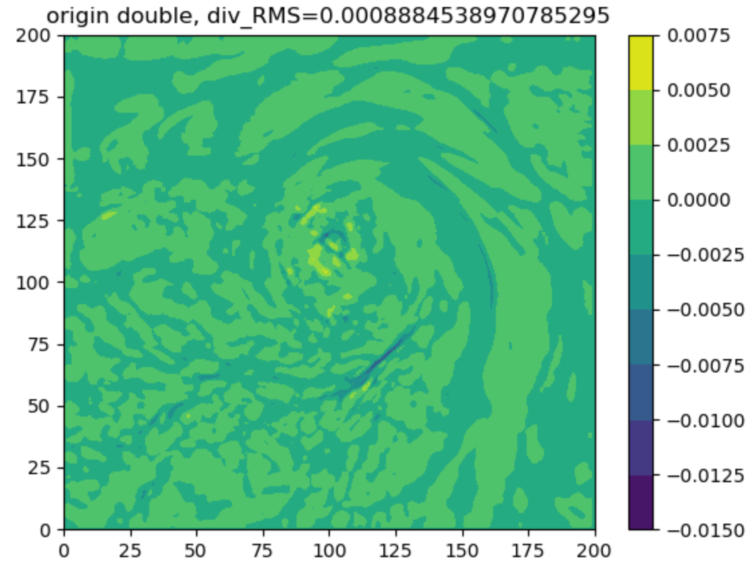
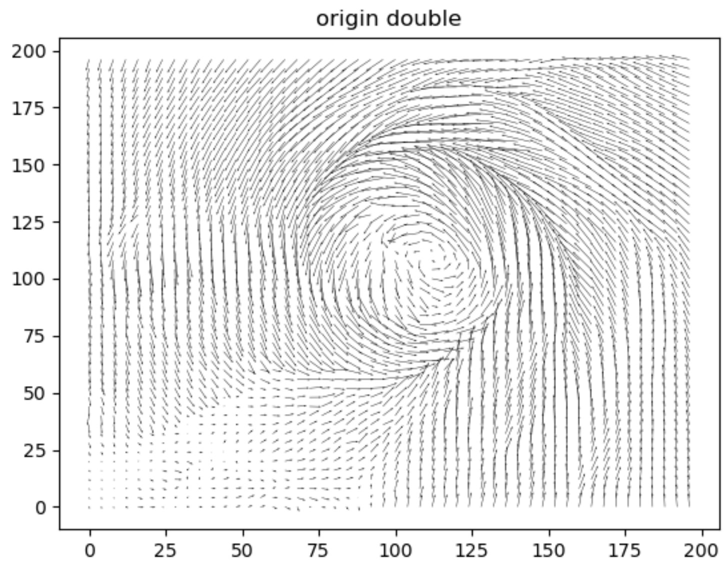
高等應用數學　　HWS　　黃展皇 110621013

1. Check the magnitude of the averaged divergence. You can use the root-mean-square magnitude.

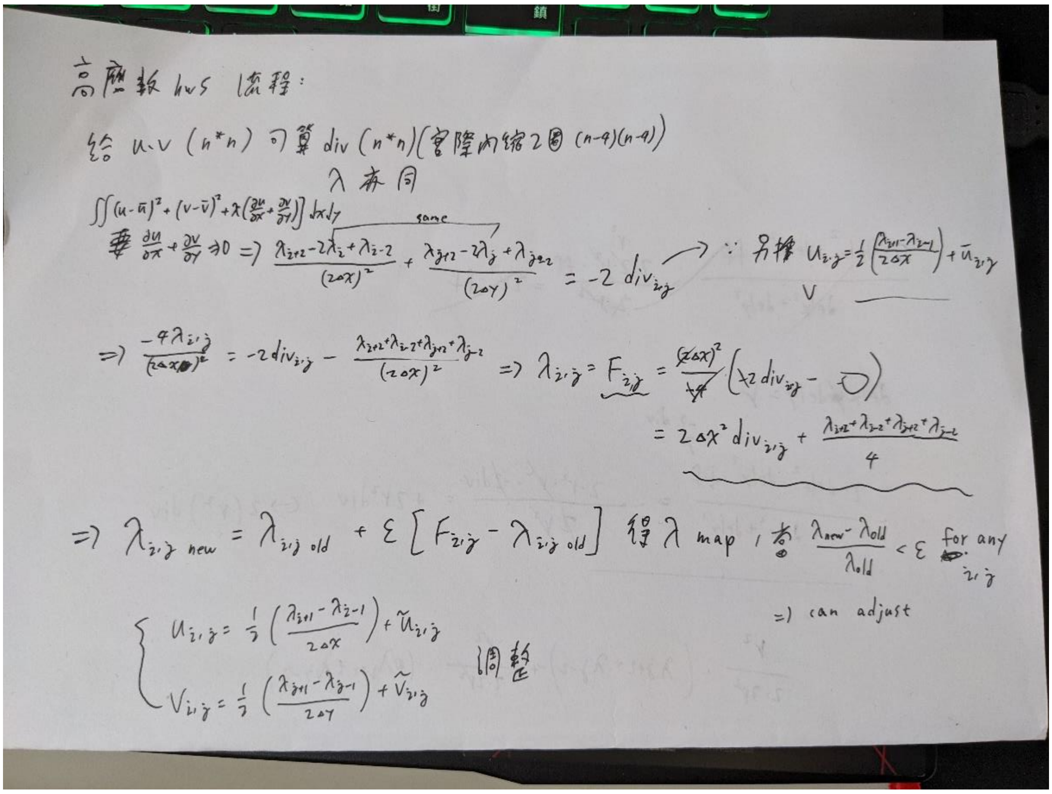


root-mean-square divergence is 8.8845*10^-4

2. Plot the wind field.



3. Use the “correct” scheme to variationally adjust the wind field under **strong constraint**, so that the final wind field becomes completely incompressible. **Please repeat this problem twice using “single-precision” and “double-precision” for your computer code, respectively.**



高應拔 hws 流程：

給 u.v $(n*n)$ 可算 div $(n*n)$ (邊際內縮2圈 $(n-4)(n-4)$)

λ 亦同

$$\iint \left[(u-\bar{u})^2+(v-\bar{v})^2+\lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right] dx\,dy$$

要 $\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \Rightarrow \frac{\lambda_{i+2}-2\lambda_i+\lambda_{i-2}}{(2\Delta x)^2}+\frac{\lambda_{j+2}-2\lambda_j+\lambda_{j-2}}{(2\Delta y)^2}=-2\,div_{i,j}$ ∵ 另擇 $u_{i,j}=\frac{1}{2}\left(\frac{\lambda_{i+1}-\lambda_{i-1}}{2\Delta x}\right)+\bar{u}_{i,j}$

same

V

$\Rightarrow \frac{-4\lambda_{i,j}}{(2\Delta x)^2}=-2div_{i,j}-\frac{\lambda_{i+2}+\lambda_{i-2}+\lambda_{j+2}+\lambda_{j-2}}{(2\Delta x)^2} \Rightarrow \lambda_{i,j}=F_{i,j}=\frac{(2\Delta x)^2}{4}\left(2div_{i,j}-\bigcirc\right)$

$=2\Delta x^2 div_{i,j}+\frac{\lambda_{i+2}+\lambda_{i-2}+\lambda_{j+2}+\lambda_{j-2}}{4}$

$\Rightarrow \lambda_{i,j\,new}=\lambda_{i,j\,old}+\varepsilon\left[F_{i,j}-\lambda_{i,j\,old}\right]$ 得 λ map，當 $\frac{\lambda_{new}-\lambda_{old}}{\lambda_{old}}<\varepsilon$ for any i,j

$\Rightarrow$ can adjust

$$\begin{cases} u_{i,j}=\frac{1}{2}\left(\frac{\lambda_{i+1}-\lambda_{i-1}}{2\Delta x}\right)+\tilde{u}_{i,j} \\ v_{i,j}=\frac{1}{2}\left(\frac{\lambda_{j+1}-\lambda_{j-1}}{2\Delta y}\right)+\tilde{v}_{i,j} \end{cases}$$ 調整

The source code is attached at the end.(eps=1.5, max_rela_e < 0.0001)

Use python np.array as precision and calculation matrix.

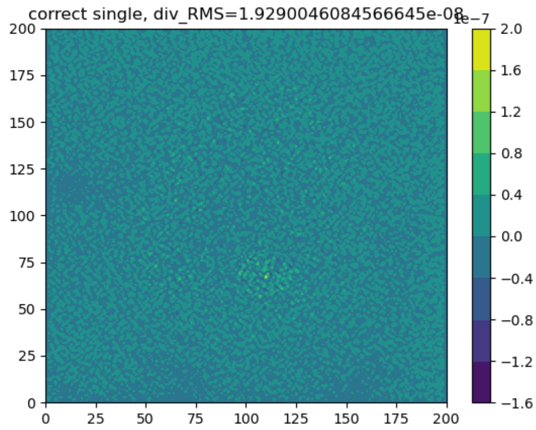
Key lambda renew process：

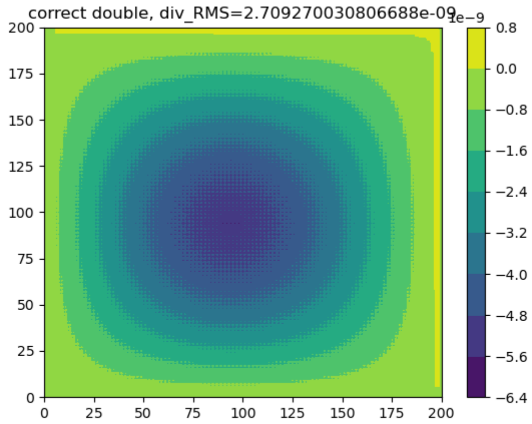
```
F = 2*(resolution**2)*div[i][j] + (lamb[i+2][j]+lamb[i-2][j]+lamb[i][j+2]+lamb[i][j-2])/4

lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])
```

4. Compute the magnitude of the averaged divergence again after the adjustment.

single-precision : **double-precision :**





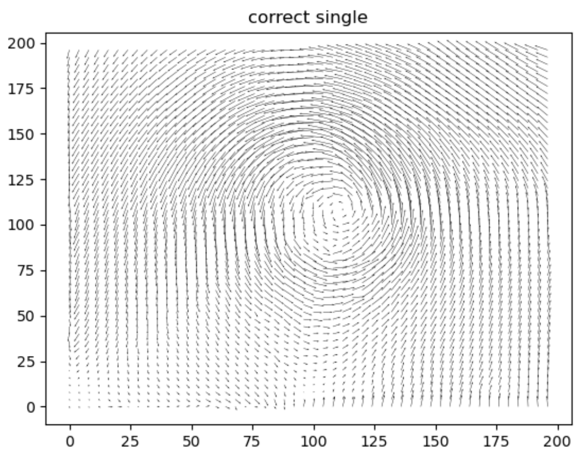

```
origin double divergence_RMS 0.0008884538970785295
correct single divergence_RMS 1.9290046084566645e-08
correct double divergence_RMS 2.709270030806688e-09
```

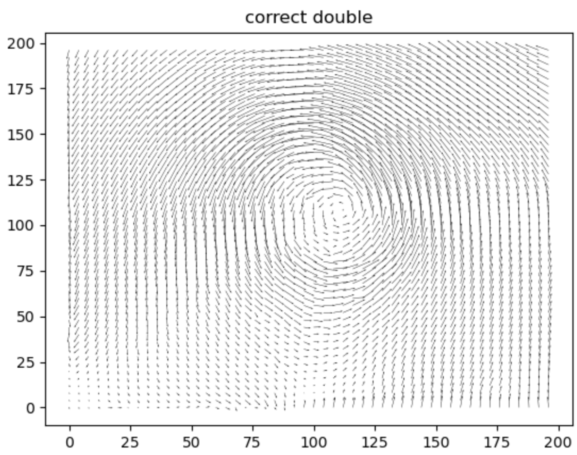
After adjustment, single-precision root-mean-square divergence is reduced to 1.92*10**-8, double-precision root-mean-square divergence is reduced to 2.71*10**-9.

Observing the distribution of divergence in space, we can find that single-precision single-point value jumps more obviously, while double-precision presents a situation where the inner circle deviates from 0.

5. Plot the wind field after the adjustment.

single-precision : **double-precision :**





It can be seen that the distribution of the two wind fields is very close, and both are spiral wind fields.

6. Repeat (3) to (5), but use the "wrong" and yet more accurate scheme.

(eps=1.5, max_rela_e < 0.0001)

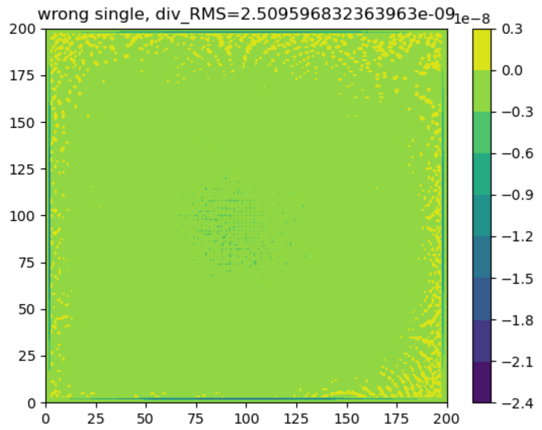
Same as above, but we consider lamb[+-1] Instead of lamb[+-2]

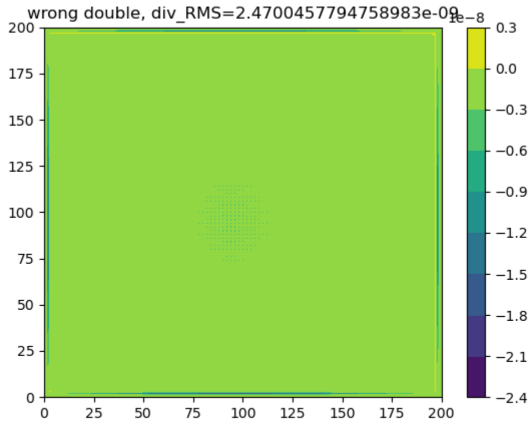
```
F = (resolution**2)*div[i][j]/2 + (lamb[i+1][j]+lamb[i-1][j]+lamb[i][j+1]+lamb[i][j-1])/4

lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])
```

Compute the magnitude of the averaged divergence again after the adjustment.

single-precision： **double-precision：**





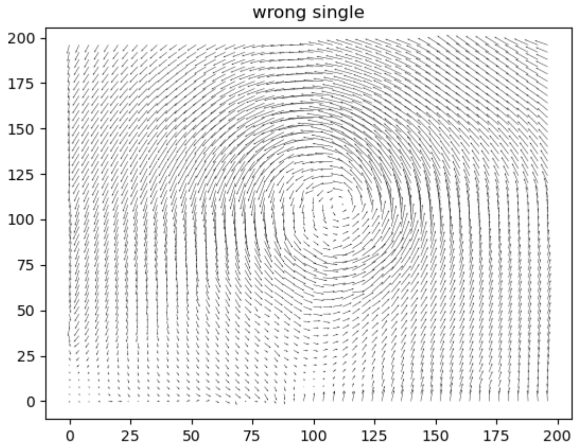

```
origin double divergence_RMS 0.0008884538970785295
correct single divergence_RMS 1.9290046084566645e-08
correct double divergence_RMS 2.709270030806688e-09
wrong single divergence_RMS 8.26582341088901e-09
wrong double divergence_RMS 8.096105002240557e-09
```

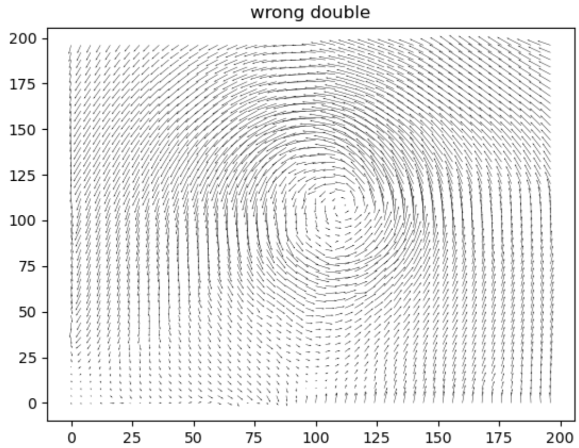
After adjustment, single-precision root-mean-square divergence is reduced to 8.27*10**-9, double-precision root-mean-square divergence is reduced to 8.10*10**-9.

Observing the distribution of divergence in space, we can find that single-precision single-point value jumps more obviously at edge, double-precision is smoother, and there is a larger value of convergence and divergence at the edge.

7. Plot the wind field after the adjustment.

single-precision： **double-precision：**





It can be seen that the distribution of the two wind fields is very close, and both are spiral wind fields.

In summary, the variational method and numerical SOR can effectively achieve the constraints we want to complete

In addition, students feel that there is room for improvement on "the relative error of each point of the lambda must be less than the threshold to end the SOR", so they wrote an additional function to change the use of RMS error as the condition to end the SOR process, so that the end when RMS<0.01 , And the results are as follows:

```
origin double divergence_RMS 0.0008884538970785295
correct single divergence_RMS 1.9290046084566645e-08
correct double divergence_RMS 2.709270030806688e-09
wrong single divergence_RMS 2.509596832363963e-09
wrong double divergence_RMS 2.4700457794758983e-09
correct test double divergence_RMS 1.0582003325471752e-11
```

updated version

SOR is limited to 4000 iterations. Compared with the original about 2000 iterations, the div should be able to converge.

```
origin double divergence_RMS 0.0008884538970785295
correct single divergence_RMS 1.5105978720468576e-08
correct double divergence_RMS 8.082506958148227e-10
wrong single divergence_RMS 1.2789661771830076e-09
wrong double divergence_RMS 1.2244987655464533e-09
correct test double divergence_RMS 5.967794024157393e-16
```

Correct single : 1.93->1.51 (*10**-8)

Correct double : 2.71*10**-9->8.08*10**-10

Wrong single : 2.51->1.28 (*10**-9)

Wrong double : 2.47->1.22 (*10**-9)

Correct double test(RMS loss) : 1.06*10**-11->5.97*10**-16

Code change:

```
#if max_rela_e < threshold:
 #    break
 if count == 4000:
     break
```

Source code：(env:win10, conda4.10.3, python3.8.8, numpy1.19.5)

```
import os
import numpy as np
import matplotlib.pyplot as plt
n = 201
# u, v : n*n
# div, lamb = n*n [usful:(n-4)(n-4)]

# read data
def read_data():
    f = np.loadtxt('data.txt')
    u, v = f[:201], f[201:]
    return u, v
# thinning data for beautiful wind quiver
def thinning(field, thinning_times=5):
    new_field = np.zeros((n//thinning_times, n//thinning_times))
    for i in range(n):
        if i%thinning_times == 0:
            for j in range(n):
                if j%thinning_times == 0 and i//thinning_times <
new_field.shape[0] and j//thinning_times < new_field.shape[1]:
                    new_field[i//thinning_times, j//thinning_times] =
field[i, j]
    return new_field
# Input pre, post, and d and output interpolation differential.
def median_interpolation(front, behind, d):
    return (behind-front)/(2*d)

# double/single version, calculate div_RMS and plot div and savefig
def q1and4_check_averaged_divergence(u, v, title=''): # out 2 lines
didn't used, midinter
    if title.split(' ')[-1] == 'double':
        resolution = np.array([2000], dtype=np.float64)[0]
        div = np.zeros((n, n), dtype=np.float64)
        for i in range(2, n-2):
            for j in range(2, n-2):
```

```
                div[i][j] = median_interpolation(u[i][j-1], u[i][j+1], 
resolution) + median_interpolation(v[i-1][j], v[i+1][j], resolution)

        div_RMS = 0.0
        for i in range(2, n-2):
            for j in range(2, n-2):
                div_RMS += div[i][j]**2
        div_RMS = (div_RMS/(n-4)**2)**0.5
        print(title+' divergence_RMS', div_RMS)
    elif title.split(' ')[-1] == 'single':
        resolution = np.array([2000], dtype=np.float32)[0]
        div = np.zeros((n, n), dtype=np.float32)
        for i in range(2, n-2):
            for j in range(2, n-2):
                div[i][j] = median_interpolation(u[i][j-1], u[i][j+1], 
resolution) + median_interpolation(v[i-1][j], v[i+1][j], resolution)

        div_RMS = np.array([0.0], dtype=np.float32)[0]
        for i in range(2, n-2):
            for j in range(2, n-2):
                div_RMS += div[i][j]**2
        div_RMS = (div_RMS/(n-4)**2)**0.5
        print(title+' divergence_RMS', div_RMS)
    else:
        print('---q1and4_check_averaged_divergence problem---')
        return

    plt.contourf(div)
    plt.colorbar()
    plt.title(title+', div_RMS='+str(div_RMS))
    plt.savefig('5-'+title+'_div')
    #plt.show()
    plt.close()

# plot wind quiver
def q2and5_plot_wind_quiver(u, v, thinning_times=4, title=''): # 
thinning and plot
```

```
    thinning_u, thinning_v = thinning(u, thinning_times), thinning(v, 
thinning_times)
    x = np.arange(0, 201-thinning_times, thinning_times)
    y = np.arange(0, 201-thinning_times, thinning_times)
    X, Y = np.meshgrid(x, y)
    plt.quiver(X, Y, thinning_u, thinning_v, scale=5, units='xy', 
width=0.2) # , headwidth=1, headlength=2
    plt.title(title)
    plt.savefig('5-'+title)
    #plt.show()
    plt.close()

def q3_correct_variationally_adjust(u, v, sd='double'): #強約束風場變分調
整，使風場變得完全不可壓縮。上下左右各兩點
# TODO: threshold/mae/mse? eps<1 or >2?
    if sd == 'double': # 雙精度調整:更新完的 lamb 直接進到下個 lamb 的計算, 
eps=1.5 + threshold
        resolution = np.array([2000], dtype=np.float64)[0]
        # check 64bits
        assert type(u[0][0]) == np.float64
        assert type(resolution) == np.float64

        # if lamb has been cal
        filepath = './correct_double_lamb.npy'
        if os.path.isfile(filepath):
            lamb = np.load('correct_double_lamb.npy')
        else:
            # lamb, div init
            lamb = np.zeros((n, n), dtype=np.float64)
            div = np.zeros((n, n), dtype=np.float64)
            for i in range(2, n-2):
                for j in range(2, n-2):
                    div[i][j] = median_interpolation(u[i][j-1], 
u[i][j+1], resolution) + median_interpolation(v[i-1][j], v[i+1][j], 
resolution)

            # iterate, until every lamb - old_lamb < threshold
            count = 0
```

```
            eps = np.array([1.5], dtype=np.float64)[0]
            threshold = np.array([10**-3], dtype=np.float64)[0]
            while True:
                old_lamb = np.array(lamb)
                count += 1

                for i in range(2, n-2):
                    for j in range(2, n-2):
                        F = 2*(resolution**2)*div[i][j] +
(lamb[i+2][j]+lamb[i-2][j]+lamb[i][j+2]+lamb[i][j-2])/4
                        lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])

                skip_flag = False
                max_rela_e = np.array([0.0], dtype=np.float64)[0]
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        if old_lamb[i][j] == np.array([0.0],
dtype=np.float64)[0]:
                            max_rela_e = 10*threshold # abs fail
                            skip_flag = True
                        if skip_flag == True:
                            break
                        max_rela_e = max(max_rela_e, abs((lamb[i][j]-
old_lamb[i][j])/old_lamb[i][j]))
                    if skip_flag == True:
                        break

                print(count, skip_flag, max_rela_e)

                if max_rela_e < threshold:
                    break
            np.save('correct_double_lamb', lamb)
            print('correct double count:', count)

        #plt.contourf(lamb)
        #plt.colorbar()
        #plt.title('lamb')
        #plt.show()
```

```
        # adjust
        for i in range(1, n-1):
            for j in range(1, n-1):
                u[i][j] += 0.5*((lamb[i][j+1]-lamb[i][j-
1])/(2*resolution))
                v[i][j] += 0.5*((lamb[i+1][j]-lamb[i-
1][j])/(2*resolution))

    else:# 單精度調整 # dtype=np.float32
        resolution = np.array([2000], dtype=np.float32)[0]
        # convert and check 32bits
        u = u.astype(np.float32)
        v = v.astype(np.float32)
        assert type(u[0][0]) == np.float32
        assert type(resolution) == np.float32

        # if lamb has been cal
        filepath = './correct_single_lamb.npy'
        if os.path.isfile(filepath):
            lamb = np.load('correct_single_lamb.npy')
        else:
            # lamb, div init
            lamb = np.zeros((n, n), dtype=np.float32)
            div = np.zeros((n, n), dtype=np.float32)
            for i in range(2, n-2):
                for j in range(2, n-2):
                    div[i][j] = median_interpolation(u[i][j-1],
u[i][j+1], resolution) + median_interpolation(v[i-1][j], v[i+1][j],
resolution)

            # iterate, until every lamb - old_lamb < threshold
            count = 0
            eps = np.array([1.5], dtype=np.float32)[0]
            threshold = np.array([10**-3], dtype=np.float32)[0]
            while True:
                old_lamb = np.array(lamb)
                count += 1
```

```
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        F = 2*(resolution**2)*div[i][j] +
(lamb[i+2][j]+lamb[i-2][j]+lamb[i][j+2]+lamb[i][j-2])/4
                        lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])

                skip_flag = False
                max_rela_e = np.array([0.0], dtype=np.float32)[0]
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        if old_lamb[i][j] == np.array([0.0],
dtype=np.float32)[0]:
                            max_rela_e = 10*threshold # abs fail
                            skip_flag = True
                        if skip_flag == True:
                            break
                        max_rela_e = max(max_rela_e, abs((lamb[i][j]-
old_lamb[i][j])/old_lamb[i][j]))
                    if skip_flag == True:
                        break

                print(count, skip_flag, max_rela_e)

                if max_rela_e < threshold:
                    break
            np.save('correct_single_lamb', lamb)
            print('correct double count:', count)

        #plt.contourf(lamb)
        #plt.colorbar()
        #plt.title('lamb')
        #plt.show()

        # adjust
        for i in range(1, n-1):
            for j in range(1, n-1):
```

```
                u[i][j] += 0.5*((lamb[i][j+1]-lamb[i][j-
1])/(2*resolution))
                v[i][j] += 0.5*((lamb[i+1][j]-lamb[i-
1][j])/(2*resolution))

    return u, v

def q6_wrong_variationally_adjust(u, v, sd='double'): #強約束風場變分調
整，使風場變得完全不可壓縮。上下左右各一點
    if sd == 'double': # 雙精度調整:更新完的 lamb 直接進到下個 lamb 的計算,
eps=1.5 + threshold
        resolution = np.array([2000], dtype=np.float64)[0]
        # check 64bits
        assert type(u[0][0]) == np.float64
        assert type(resolution) == np.float64

        # if lamb has been cal
        filepath = './wrong_double_lamb.npy'
        if os.path.isfile(filepath):
            lamb = np.load('wrong_double_lamb.npy')
        else:
            # lamb, div init
            lamb = np.zeros((n, n), dtype=np.float64)
            div = np.zeros((n, n), dtype=np.float64)
            for i in range(2, n-2):
                for j in range(2, n-2):
                    div[i][j] = median_interpolation(u[i][j-1],
u[i][j+1], resolution) + median_interpolation(v[i-1][j], v[i+1][j],
resolution)

            # iterate, until every lamb - old_lamb < threshold
            count = 0
            eps = np.array([1.5], dtype=np.float64)[0]
            threshold = np.array([10**-3], dtype=np.float64)[0]
            while True:
                old_lamb = np.array(lamb)
                count += 1
```

```
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        F = (resolution**2)*div[i][j]/2 +
(lamb[i+1][j]+lamb[i-1][j]+lamb[i][j+1]+lamb[i][j-1])/4
                        lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])

                skip_flag = False
                max_rela_e = np.array([0.0], dtype=np.float64)[0]
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        if old_lamb[i][j] == np.array([0.0],
dtype=np.float64)[0]:
                            max_rela_e = 10*threshold # abs fail
                            skip_flag = True
                        if skip_flag == True:
                            break
                        max_rela_e = max(max_rela_e, abs((lamb[i][j]-
old_lamb[i][j])/old_lamb[i][j]))
                    if skip_flag == True:
                        break

                print(count, skip_flag, max_rela_e)

                if max_rela_e < threshold:
                    break
            np.save('wrong_double_lamb', lamb)
            print('wrong double count:', count)

        #plt.contourf(lamb)
        #plt.colorbar()
        #plt.title('lamb')
        #plt.show()

        # adjust
        for i in range(1, n-1):
            for j in range(1, n-1):
                u[i][j] += 0.5*((lamb[i][j+1]-lamb[i][j-
1])/(2*resolution))
```

```
                v[i][j] += 0.5*((lamb[i+1][j]-lamb[i-
1][j])/(2*resolution))

    else:# 單精度調整 # dtype=np.float32
        resolution = np.array([2000], dtype=np.float32)[0]
        # convert and check 32bits
        u = u.astype(np.float32)
        v = v.astype(np.float32)
        assert type(u[0][0]) == np.float32
        assert type(resolution) == np.float32

        # if lamb has been cal
        filepath = './wrong_single_lamb.npy'
        if os.path.isfile(filepath):
            lamb = np.load('wrong_single_lamb.npy')
        else:
            # lamb, div init
            lamb = np.zeros((n, n), dtype=np.float32)
            div = np.zeros((n, n), dtype=np.float32)
            for i in range(2, n-2):
                for j in range(2, n-2):
                    div[i][j] = median_interpolation(u[i][j-1],
u[i][j+1], resolution) + median_interpolation(v[i-1][j], v[i+1][j],
resolution)

            # iterate, until every lamb - old_lamb < threshold
            count = 0
            eps = np.array([1.5], dtype=np.float32)[0]
            threshold = np.array([10**-3], dtype=np.float32)[0]
            while True:
                old_lamb = np.array(lamb)
                count += 1

                for i in range(2, n-2):
                    for j in range(2, n-2):
                        F = (resolution**2)*div[i][j]/2 +
(lamb[i+1][j]+lamb[i-1][j]+lamb[i][j+1]+lamb[i][j-1])/4
                        lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])
```

```
            skip_flag = False
            max_rela_e = np.array([0.0], dtype=np.float32)[0]
            for i in range(2, n-2):
                for j in range(2, n-2):
                    if old_lamb[i][j] == np.array([0.0],
dtype=np.float32)[0]:
                        max_rela_e = 10*threshold # abs fail
                        skip_flag = True
                    if skip_flag == True:
                        break
                    max_rela_e = max(max_rela_e, abs((lamb[i][j]-
old_lamb[i][j])/old_lamb[i][j]))
                if skip_flag == True:
                    break

            print(count, skip_flag, max_rela_e)

            if max_rela_e < threshold:
                break
        np.save('wrong_single_lamb', lamb)
        print('wrong double count:', count)

    #plt.contourf(lamb)
    #plt.colorbar()
    #plt.title('lamb')
    #plt.show()

    # adjust
    for i in range(1, n-1):
        for j in range(1, n-1):
            u[i][j] += 0.5*((lamb[i][j+1]-lamb[i][j-
1])/(2*resolution))
            v[i][j] += 0.5*((lamb[i+1][j]-lamb[i-
1][j])/(2*resolution))

return u, v
```

```
#### test
def q3_cva_test(u, v, sd='double'): # mse test
    if sd == 'double': # 雙精度調整:更新完的 lamb 直接進到下個 lamb 的計算,
eps=1.5 + mse
        resolution = np.array([2000], dtype=np.float64)[0]
        # check 64bits
        assert type(u[0][0]) == np.float64
        assert type(resolution) == np.float64

        # if lamb has been cal
        filepath = './test_correct_double_lamb.npy'
        if os.path.isfile(filepath):
            lamb = np.load('test_correct_double_lamb.npy')
        else:
            # lamb, div init
            lamb = np.zeros((n, n), dtype=np.float64)
            div = np.zeros((n, n), dtype=np.float64)
            for i in range(2, n-2):
                for j in range(2, n-2):
                    div[i][j] = median_interpolation(u[i][j-1],
u[i][j+1], resolution) + median_interpolation(v[i-1][j], v[i+1][j],
resolution)

            # iterate, until every lamb - old_lamb < threshold
            count = 0
            eps = np.array([1.5], dtype=np.float64)[0]
            mse_threshold = np.array([10**-2], dtype=np.float64)[0]
            while True:
                old_lamb = np.array(lamb)
                count += 1

                for i in range(2, n-2):
                    for j in range(2, n-2):
                        F = 2*(resolution**2)*div[i][j] +
(lamb[i+2][j]+lamb[i-2][j]+lamb[i][j+2]+lamb[i][j-2])/4
                        lamb[i][j] = lamb[i][j] + eps*(F-lamb[i][j])

                skip_flag = False
```

```
                mse = np.array([0.0], dtype=np.float64)[0]
                for i in range(2, n-2):
                    for j in range(2, n-2):
                        if old_lamb[i][j] == np.array([0.0],
dtype=np.float64)[0]:
                            mse = 10*mse_threshold # abs fail
                            skip_flag = True
                        if skip_flag == True:
                            break
                        mse += (lamb[i][j]-old_lamb[i][j])**2
                    if skip_flag == True:
                        break

                print(count, skip_flag, mse)

                if mse < mse_threshold:
                    break
                #if count > 30:
                #    break
            np.save('test_correct_double_lamb', lamb)
            print('test correct double count:', count)

        #plt.contourf(lamb)
        #plt.colorbar()
        #plt.title('lamb')
        #plt.show()

        # adjust
        for i in range(1, n-1):
            for j in range(1, n-1):
                u[i][j] += 0.5*((lamb[i][j+1]-lamb[i][j-
1])/(2*resolution))
                v[i][j] += 0.5*((lamb[i+1][j]-lamb[i-
1][j])/(2*resolution))
    else:
        print(':((((')
    return u, v
```

```python
if __name__ == '__main__':
    u, v = read_data()
    q1and4_check_averaged_divergence(u, v, 'origin double')
    q2and5_plot_wind_quiver(u, v, 4, 'origin double')

    cva_s_u, cva_s_v = q3_correct_variationally_adjust(u, v, 'single')
    q1and4_check_averaged_divergence(cva_s_u, cva_s_v, 'correct single')
    q2and5_plot_wind_quiver(cva_s_u, cva_s_v, 4, 'correct single')

    cva_d_u, cva_d_v = q3_correct_variationally_adjust(u, v, 'double')
    q1and4_check_averaged_divergence(cva_d_u, cva_d_v, 'correct double')
    q2and5_plot_wind_quiver(cva_d_u, cva_d_v, 4, 'correct double')

    wva_s_u, wva_s_v = q6_wrong_variationally_adjust(u, v, 'single')
    q1and4_check_averaged_divergence(wva_s_u, wva_s_v, 'wrong single')
    q2and5_plot_wind_quiver(wva_s_u, wva_s_v, 4, 'wrong single')

    wva_d_u, wva_d_v = q6_wrong_variationally_adjust(u, v, 'double')
    q1and4_check_averaged_divergence(wva_d_u, wva_d_v, 'wrong double')
    q2and5_plot_wind_quiver(wva_d_u, wva_d_v, 4, 'wrong double')



    cva_d_u_test, cva_d_v_test = q3_cva_test(u, v, 'double')
    q1and4_check_averaged_divergence(cva_d_u_test, cva_d_v_test,
'correct test double')
    q2and5_plot_wind_quiver(cva_d_u_test, cva_d_v_test, 4, 'correct test
double')
```