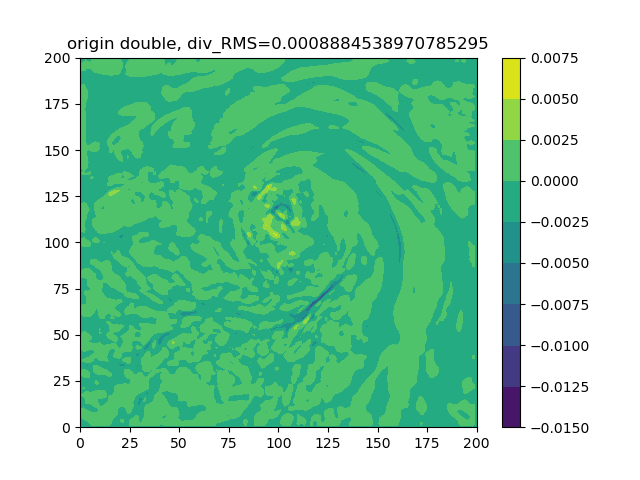
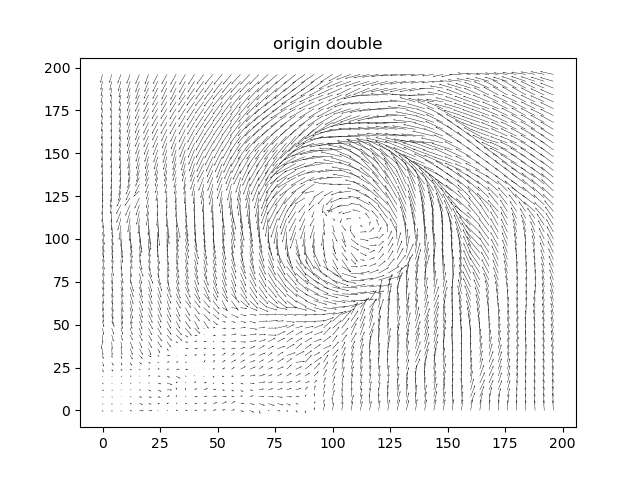
高等應用數學 HW5 黃展皇110621013

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

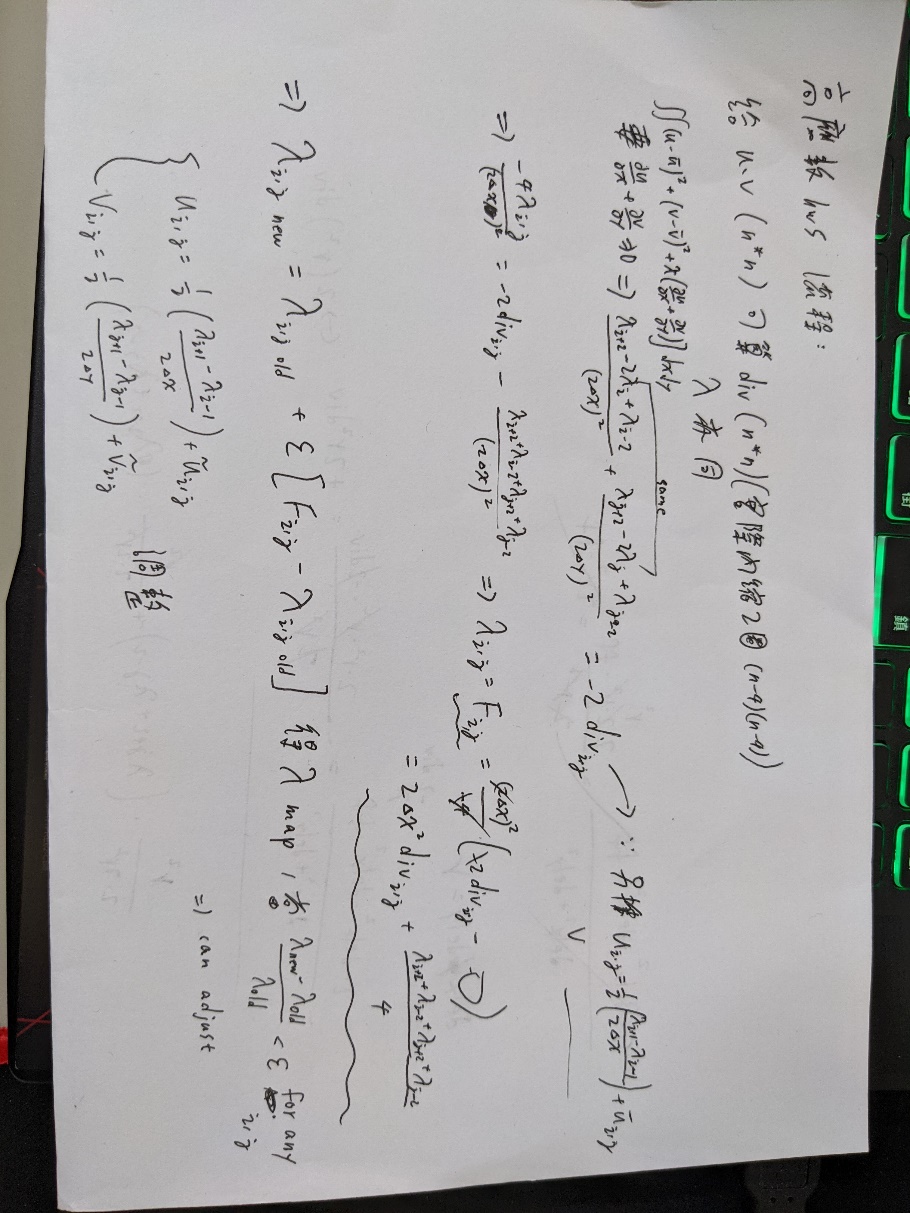


root-mean-square divergence is 8.8845\*10^-4

1. Plot the wind field.



1. 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.**



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.

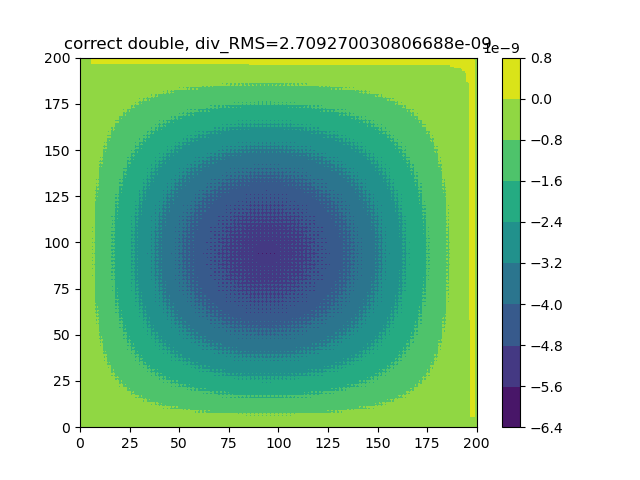
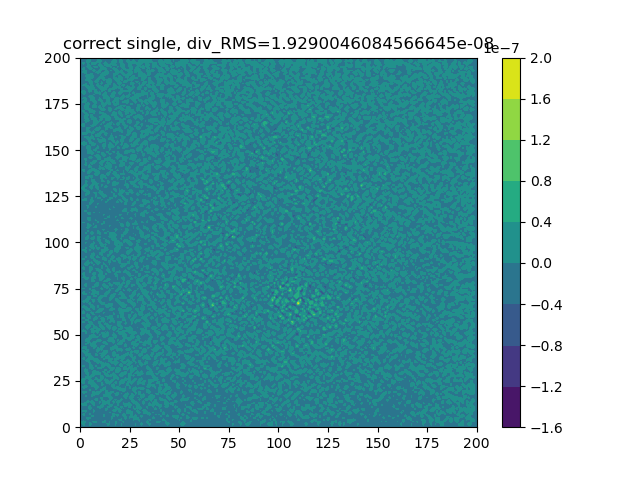
Key lambda renew process：

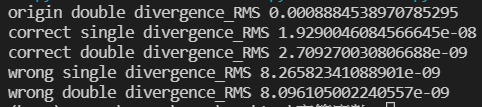
F = 2\*(resolution\*\*2)\*div[i][j] + (lamb[i+2][j]+lamb[i-2][j]+lamb[i][j+2]+lamb[i][j-2])/4

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

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

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



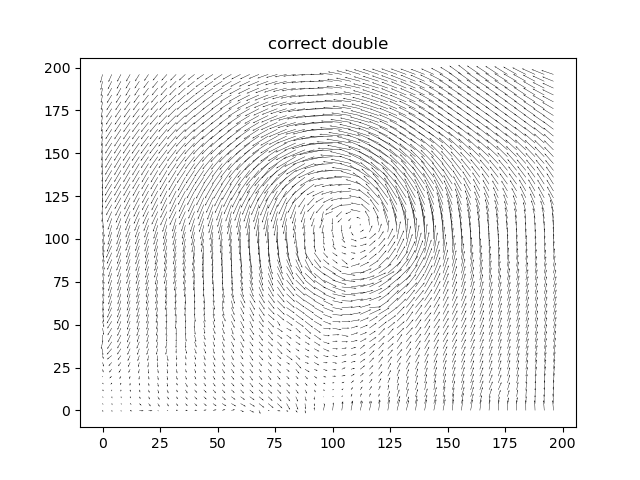
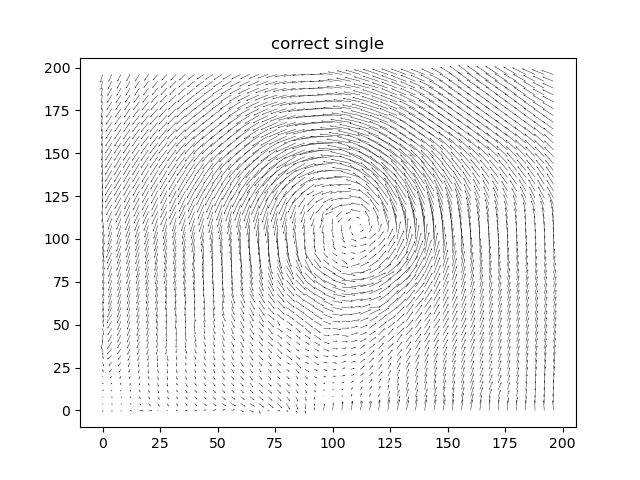


After adjustment, single-precision root-mean-square divergence is reduced to 1.92\*10\*\*-8, double-precision root-mean-square divergence is reduced to 2.71\*10\*\*-9.

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.

1. 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.

1. Repeat (3) to (5), but use the “wrong” and yet more accurate scheme.

(eps=1.5, max\_rela\_e < 0.0001)

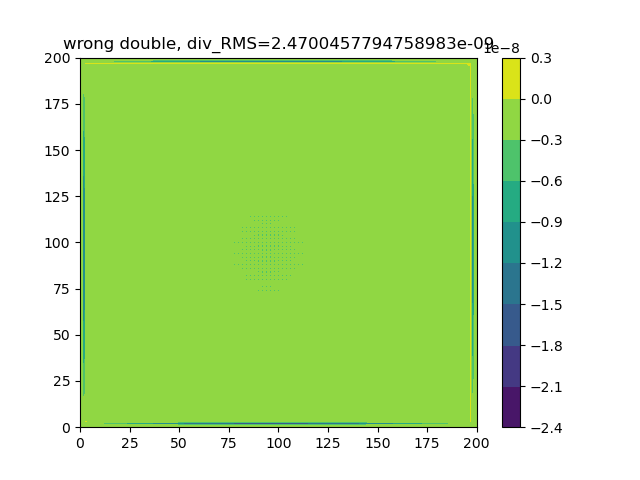
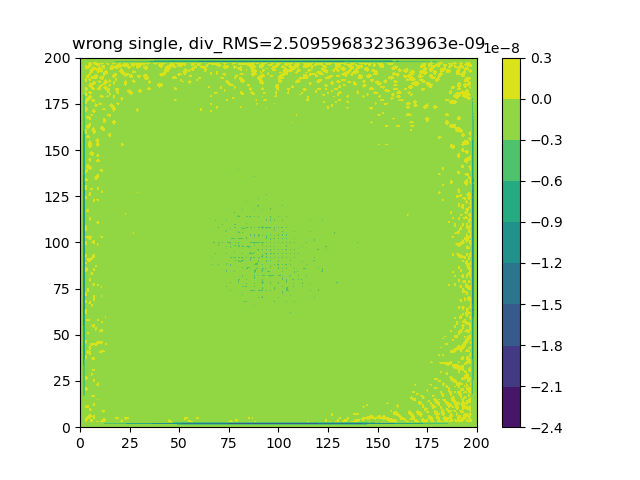
Same as above, but we consider lamb[+-1] Instead of lamb[+-2]

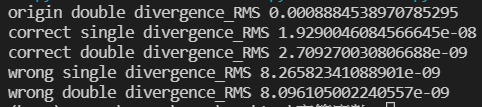
F = (resolution\*\*2)\*div[i][j]/2 + (lamb[i+1][j]+lamb[i-1][j]+lamb[i][j+1]+lamb[i][j-1])/4

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：**



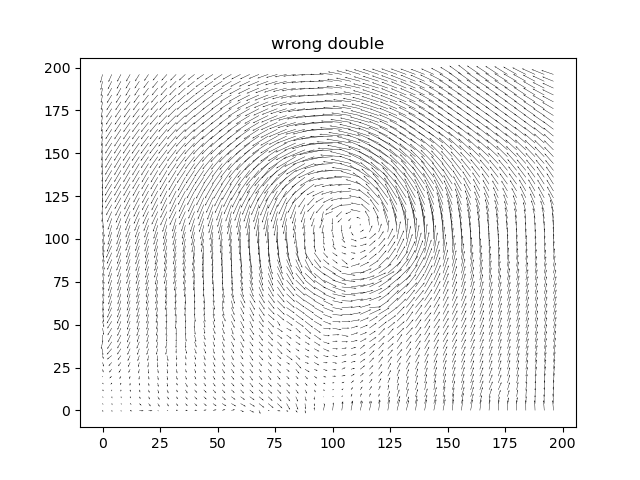
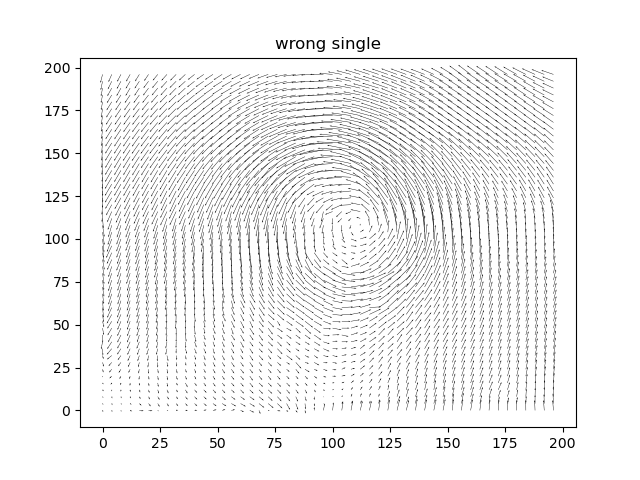


After adjustment, single-precision root-mean-square divergence is reduced to 8.27\*10\*\*-9, double-precision root-mean-square divergence is reduced to 8.10\*10\*\*-9.

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.

1. Plot the wind field after the adjustment.

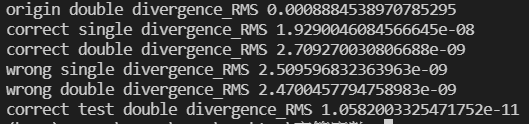
**single-precision： double-precision：**



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

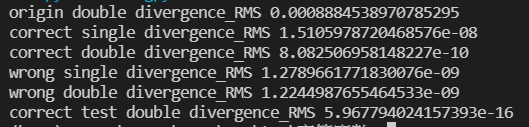
In summary, the variational method and numerical SOR can effectively achieve the constraints we want to complete

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:



**updated version**

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



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

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')