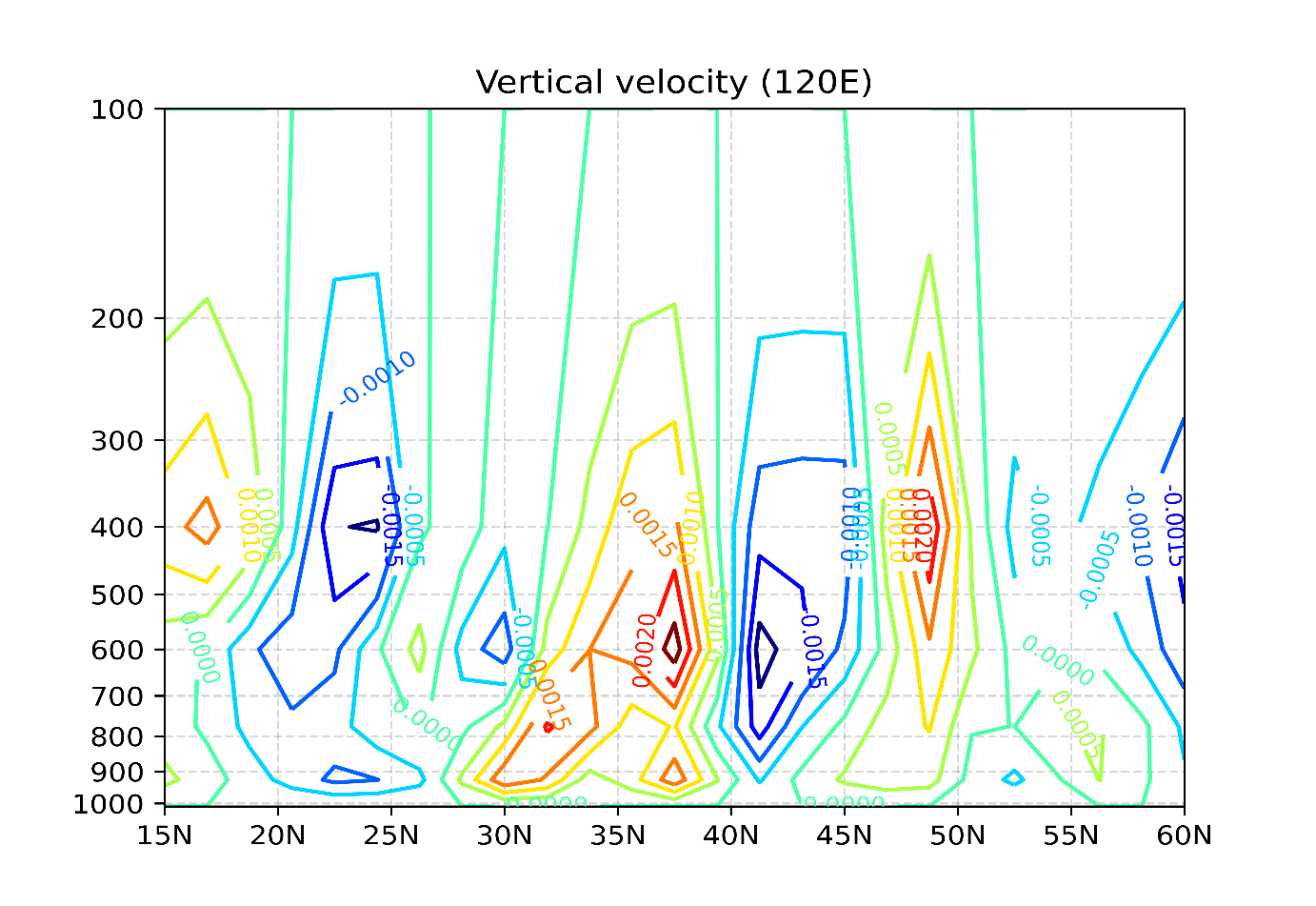
天氣學下hw2 大氣4A 黃展皇 106601015

工作環境：x64 windows10，conda 4.8.3，python3.6.10

Requestment：os、numpy、matplotlib、math、sys

(一)繪出120E，範圍15N-60N內垂直速度剖面圖(不須計算邊界)。



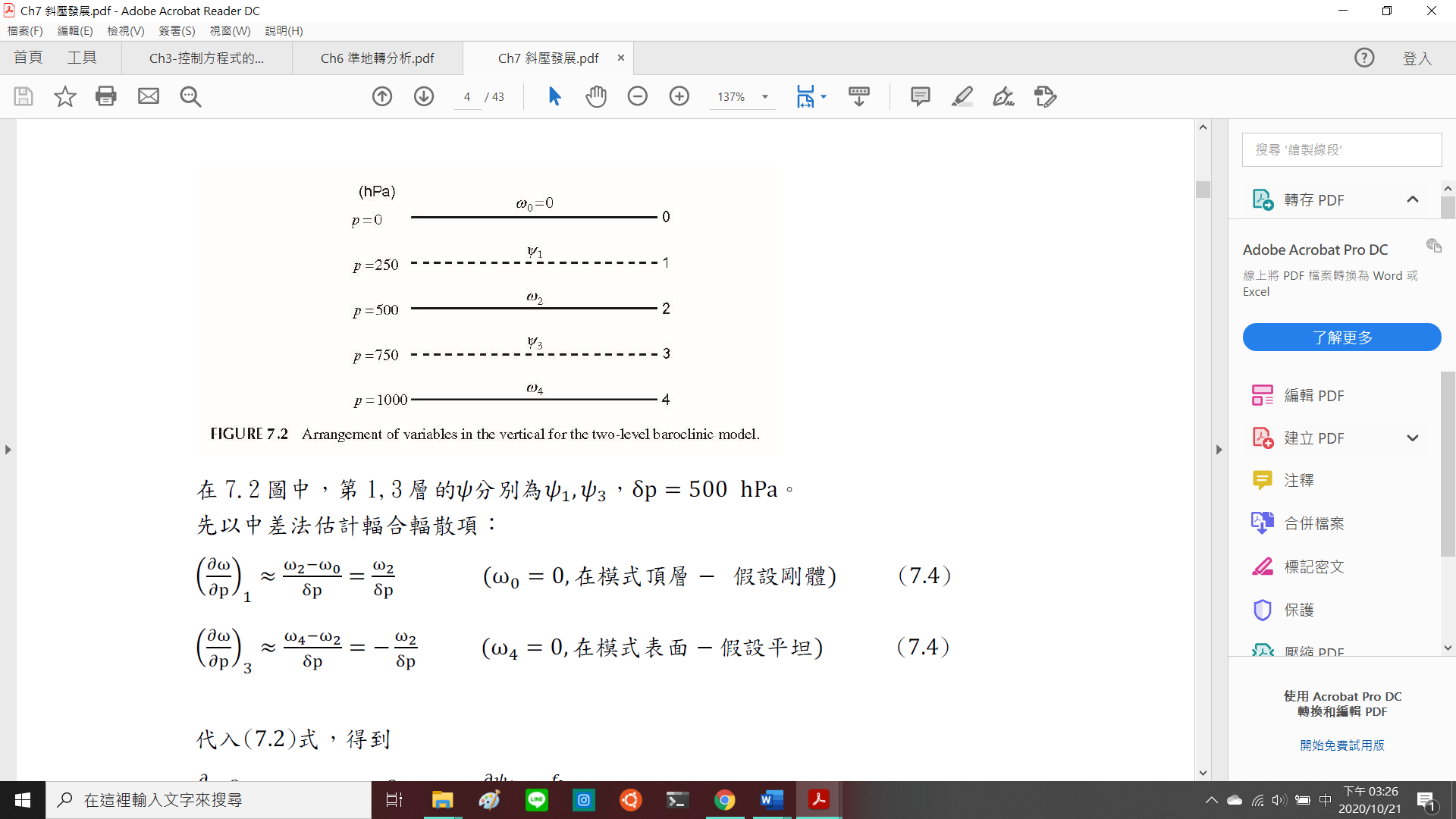
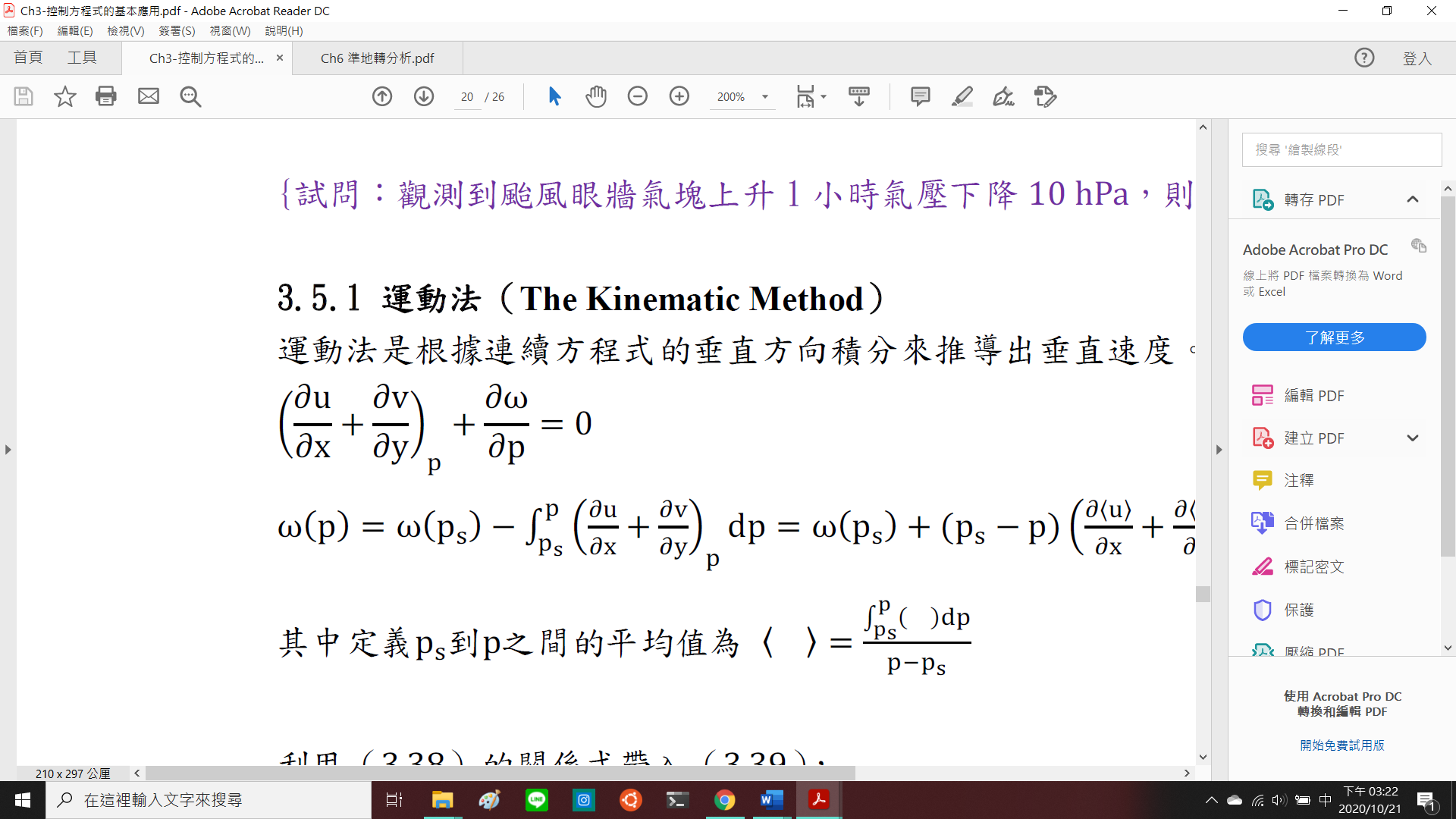
(二)問題討論：

(1)為何垂直速度要用計算的？計算出來後用途為何？

因為垂直速度w在綜觀尺度中大約為每秒數公分，但一般使用的探空儀器水平風速測量的精確度只到每秒一公尺，因此垂直速度需要由其他可實際測得的變數來計算。而計算出來的垂直速度其一可以直接表示當地的氣流是上升還是下沉氣流，尺度規模多大，可讓我們快速判斷當地的天氣狀況；其二大氣波動如淺水波也牽涉到垂直運動，了解垂直速度的分布有助於理解波動的傳播方式；其三根據continue eq垂直速度也對應到氣旋式與反氣旋式環流以及非地轉風的出流入流，因此了解垂直速度也有助於了解綜觀環流。

(2)此計算方法有何優缺點？

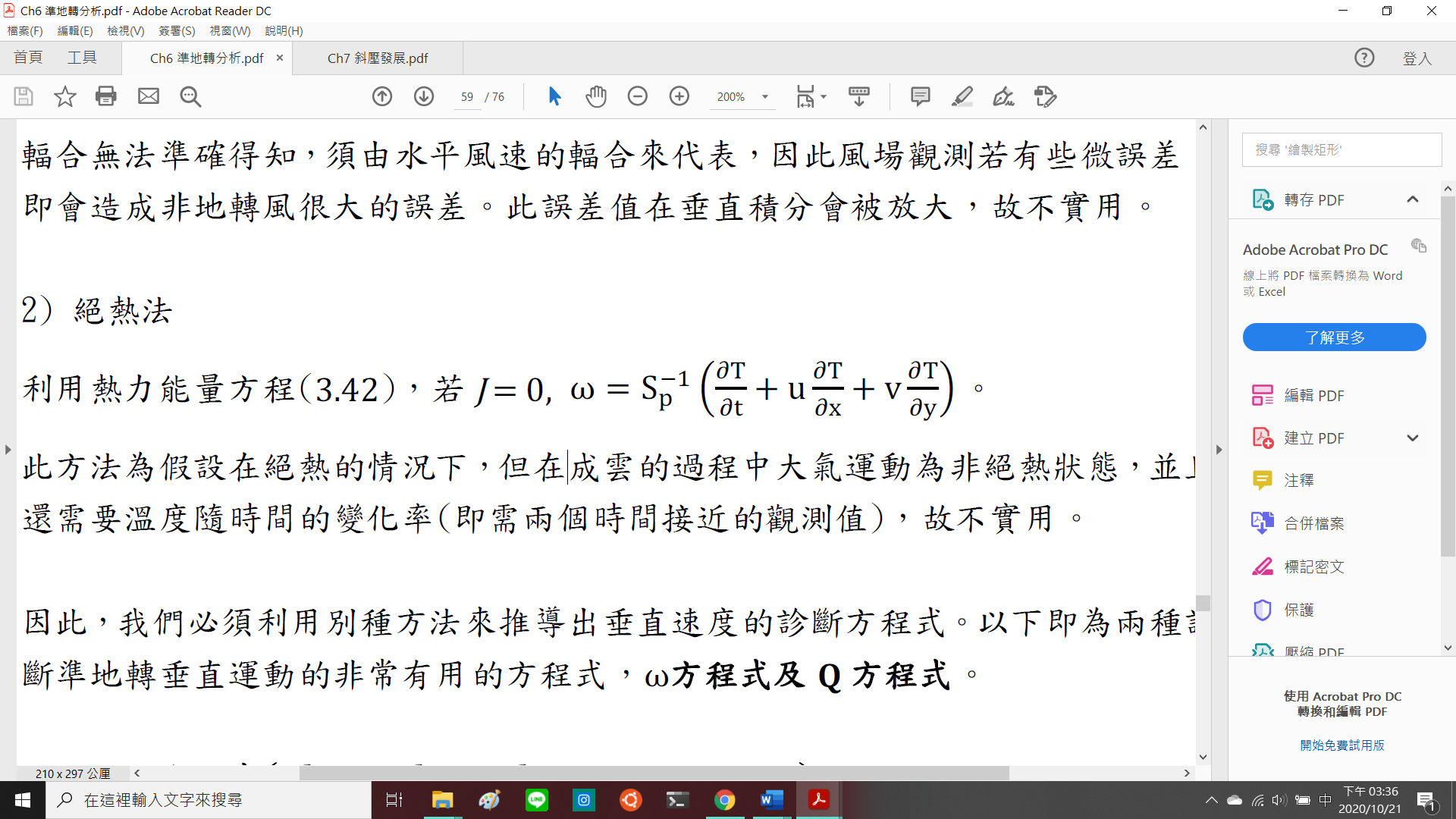
此方法類似於動力法與二層模式法的結合擴充版，原理如下圖：



此法假設綜觀尺度下continue eq成立且頂層與底層垂直速度=0，垂直速度即可由上下層水平風速的輻合輻散場來代表，因此風場觀測若有些微誤差即會造成很大的誤差，若u、v各誤差10%則w可能誤差100%。

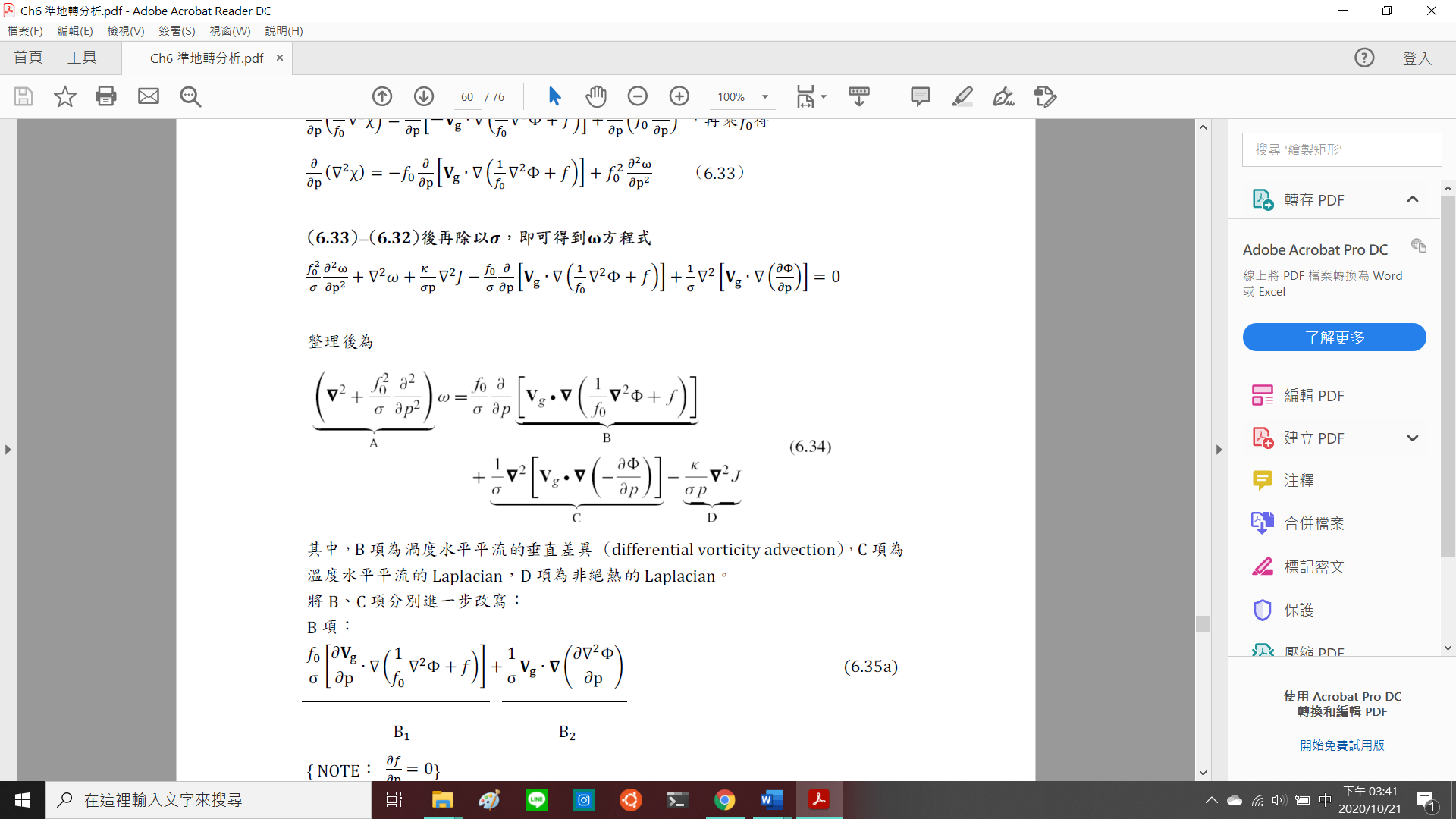
(3)其他計算垂直速度的方法及其優缺點？

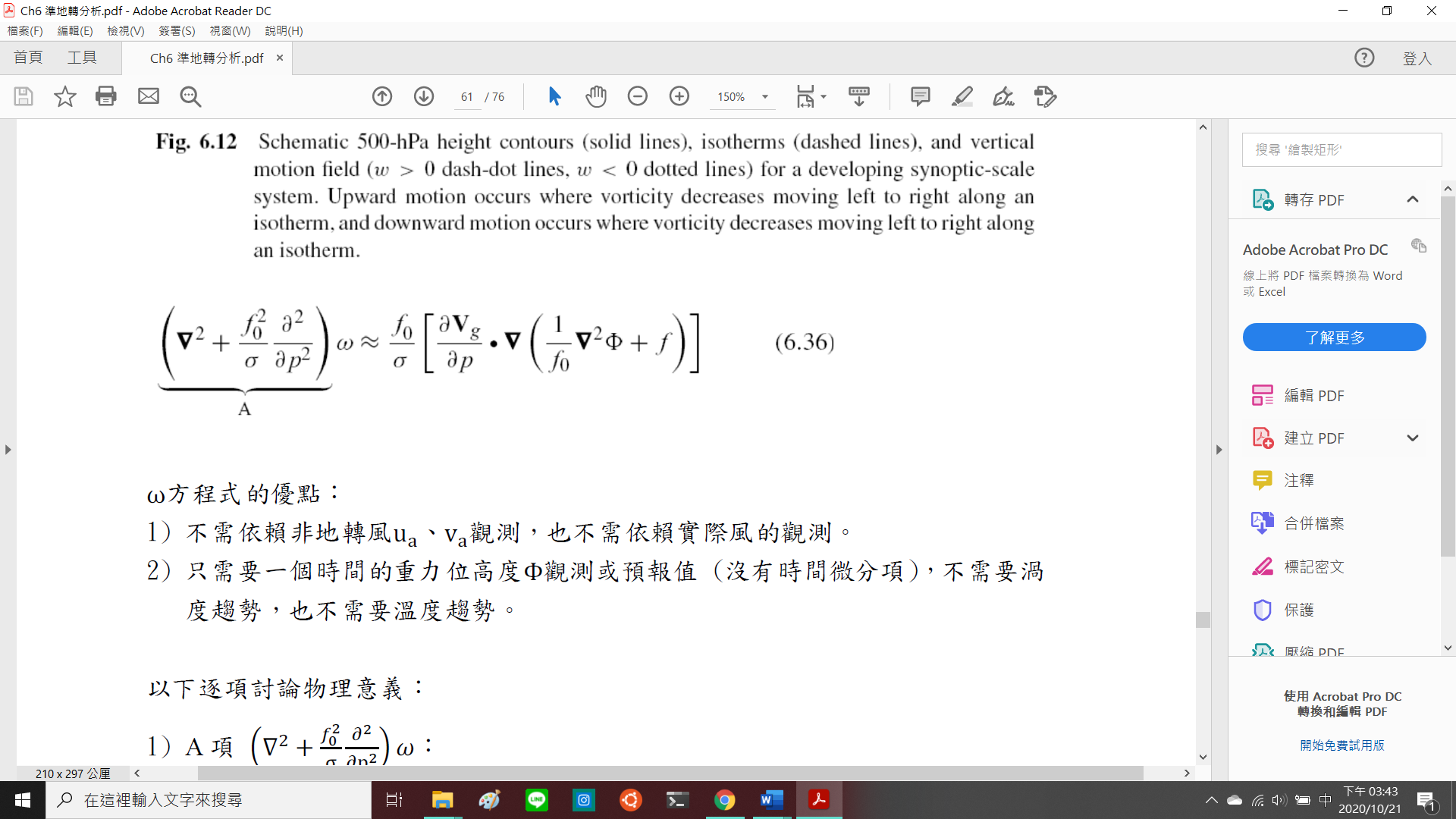
純動力法，如上介紹，但是純公式推導之後需要垂直積分，觀測上的誤差值會在垂直積分被放大，且仍有u、v各誤差10%則w可能誤差100%的問題，故計算出來的垂直速度值與觀測差距過大，不實用；

絕熱法，利用熱力能量方程：

假設在絕熱情況下若知道溫度隨時間的變化率、溫度場的分布以及u、v風場即可推導出垂直速度，但是溫度隨時間的變化率難以取得(高空對於溫度隨時間的變化率不夠密集)且成雲的過程中大氣運動為非絕熱狀態，故也有部分限制。

準地轉ω方程法，不需時間變化項的垂直速度，公式如下：





優點：不需要非地轉風ua、va觀測、實際風的觀測、渦度趨勢、溫度趨勢，只需要一個時間的重力位高度Φ觀測或預報值(無時間微分項)。

缺點：ω方程式的微分階數變高，運算精確度的難度增加。

(三)計算與繪圖程式碼 + 註解

import os

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.ticker as mticker

import math

import sys

print(sys.prefix) # show what virtual env I am in

# read binary data, analyze to 49x\*25y\*5(1000 850 700 500 300)\*4(H U V T) = 24500 np.array, return wanted plane data

def read\_bindata\_return\_wanted(hw1\_root\_path, filename, pressure, parameter):

    all\_data = np.fromfile(hw1\_root\_path+filename, dtype='<f4', count=-1, sep='')

    init\_lndex = 5\*x\_num\*y\_num\*parameter\_list.index(parameter) + x\_num\*y\_num\*pressure\_list.index(pressure)

    return all\_data[init\_lndex:init\_lndex+x\_num\*y\_num]

# Input y and output the corresponding latitude coordinates

def y\_to\_lat(y):

    return lat\_lower+y\*delta

def lon\_to\_x(lon):

    return int((lon-lon\_lower)/delta)

# Input pre, post, and d and output interpolation differential.

def median\_interpolation(front, behind, d):

    return (behind-front)/(2\*d)

# Input pre, here, and d and output the pre-interpolated differential.

def front\_interpolation(front, here, d):

    return (here-front)/d

# Input here value and post value and output post-interpolation differential.

def behind\_interpolation(here, behind, d):

    return (behind-here)/d

# return 120E 15~60N divergence

def get\_120E\_divergence(u, v):

    divergence = np.zeros([y\_num])

    dy = 6378000\*delta\*math.pi/180

    for y in range(y\_num):

        dx=dy\*math.cos(y\_to\_lat(y)\*(2\*math.pi/360))

        x = lon\_to\_x(120)

        dudx = median\_interpolation(u[y, x-1], u[y, x+1], dx)

        #print(u[y, x-1], u[y, x+1], dx, dy)

        # default try: median\_interpolation

        try:

            dvdy = median\_interpolation(v[y-1, x], v[y+1, x], dy)

        except IndexError:

            pass

        # y edge conditions

        if y == 0:dvdy = behind\_interpolation(v[y, x], v[y+1, x], dy)

        if y == y\_num-1:dvdy = front\_interpolation(v[y-1, x], v[y, x], dy)

        divergence[y] = dudx + dvdy

    #print(divergence)

    return divergence

# plot profile

def plot\_profile(w):

    print('plot init')

    #print(w\_pressure\_list)

    # set up y, z grid

    y = np.arange(lat\_lower, lat\_upper+delta, delta)

    z = np.array(w\_pressure\_list)

    Y, Z = np.meshgrid(y, z)

    # set clabel gap, y\_clabel\_list and plot contour

    gap = np.max(w)/8

    gap = 0.0005

    y\_clabel\_list = gap \* np.arange(int(np.min(w)/gap), int(np.max(w)/gap)+1)

    #print(y\_clabel\_list)

    cs = plt.contour(Y, Z, w, cmap='jet', levels=y\_clabel\_list)

    #set clabel, yscale=log , yticks, xlabel and xticks, ylim, grid, title, then savefig

    plt.clabel(cs, fontsize='small', fmt='%1.4f')

    plt.yscale('log')

    plt.yticks(np.linspace(100, 1000, 10), np.linspace(100, 1000, 10).astype('int32'))

    xlabel = np.linspace(15, 60, 10)

    plt.xticks(xlabel, ['{}N'.format(str(int(x))) for x in xlabel])

    plt.ylim(w\_pressure\_list[0], w\_pressure\_list[5])

    gl = plt.grid(color='gray', alpha=0.3, linestyle='--')

    plt.title('Vertical velocity (120E)')

    #plt.colorbar()

    #plt.savefig('1.png')

    plt.savefig(hw2\_root\_path+'1.png', dpi=800)

if \_\_name\_\_ == "\_\_main\_\_":

    # Definition of basic parameters

    print('init!!')

    names = locals()

    hw2\_root\_path = os.path.join(os.path.abspath('.'), 'hw2')+'\\'

    pressure\_list = ['1000', '850', '700', '500', '300']

    parameter\_list = ['H', 'U', 'V', 'T']

    x\_num, y\_num = 49, 25

    lon\_upper, lon\_lower, lat\_upper, lat\_lower = 180, 90, 60, 15

    delta = 1.875

    # Load u, v data + definate variable

    for parameter in parameter\_list:

        if parameter == 'H' or parameter == 'T':continue

        for pressure in pressure\_list:

            plane\_data\_name = parameter + pressure

            plane\_data = np.reshape(read\_bindata\_return\_wanted(hw2\_root\_path, filename='output.bin', pressure=pressure, parameter=parameter), (y\_num, x\_num))

            names[plane\_data\_name] = plane\_data

            #print(parameter+pressure, 'load ok')

    # init w0~5 as 0 array, and get every layers divergence

    w\_pressure\_list = [1010, 925, 775, 600, 400, 100]

    for i in range(len(w\_pressure\_list)):

        w\_name = 'w{}'.format(str(i))

        names[w\_name] = np.zeros(y\_num)

    for pressure in pressure\_list:

        names['D{}'.format(pressure)] = get\_120E\_divergence(names['U{}'.format(pressure)], names['V{}'.format(pressure)])

    # using old w to culculate every layers w

    for i in range(len(w\_pressure\_list)):

        if i == 0:w = np.copy(np.zeros(y\_num))

        else:

            w += names['D{}'.format(pressure\_list[i-1])] \* (w\_pressure\_list[i-1]-w\_pressure\_list[i])

            names['w{}'.format(str(i))] = np.copy(w)

    # using not ok w to fix D

    eps = (w5-w0)/(w\_pressure\_list[0]-w\_pressure\_list[5])

    for pressure in pressure\_list:

        names['D{}'.format(pressure)] = names['D{}'.format(pressure)] - eps

    # using fixed w to culculate every layers w

    for i in range(len(w\_pressure\_list)):

        if i == 0:w = np.copy(np.zeros(y\_num))

        else:

            w += names['D{}'.format(pressure\_list[i-1])] \* (w\_pressure\_list[i-1]-w\_pressure\_list[i])

            names['w{}'.format(str(i))] = np.copy(w)

    w = np.vstack([w0, w1, w2, w3, w4, w5])

    #print('w.shape', w.shape)

    plot\_profile(w)