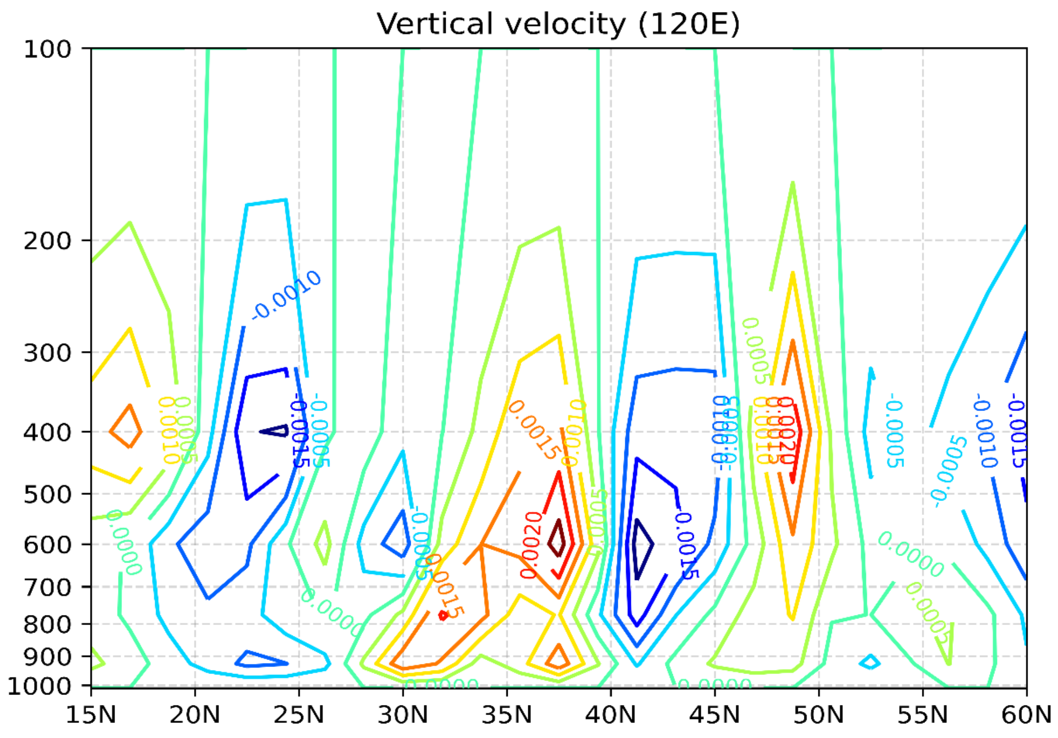
天氣學下 hw2　　　大氣 4A　黃展皇　　106601015

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：

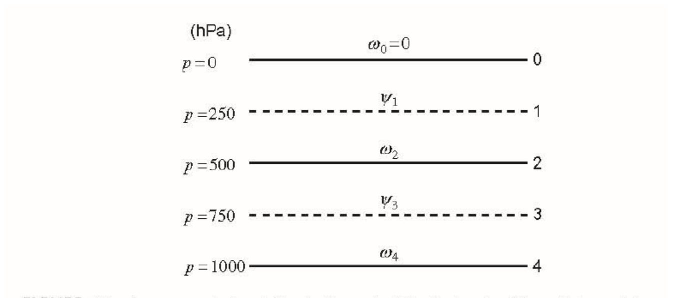


FIGURE 7.2 Arrangement of variables in the vertical for the two-level baroclinic model.

在 7.2 圖中，第 1, 3 層的ψ分別為ψ_1, ψ_3，$\delta p = 500\ \text{hPa}$。
先以中差法估計輻合輻散項：

$$\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)_p+\frac{\partial \omega}{\partial p}=0$$

$$\left(\frac{\partial \omega}{\partial p}\right)_1 \approx \frac{\omega_2-\omega_0}{\delta p}=\frac{\omega_2}{\delta p} \qquad (\omega_0=0, \text{在模式頂層 − 假設剛體})$$

$$\left(\frac{\partial \omega}{\partial p}\right)_3 \approx \frac{\omega_4-\omega_2}{\delta p}=-\frac{\omega_2}{\delta p} \qquad (\omega_4=0, \text{在模式表面 − 假設平坦})$$

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： $若\ J=0,\ \omega = S_p^{-1}\left(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}\right)$

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：

$$\underbrace{\left(\nabla^2 + \frac{f_0^2}{\sigma}\frac{\partial^2}{\partial p^2}\right)}_{A}\omega = \frac{f_0}{\sigma}\frac{\partial}{\partial p}\underbrace{\left[\mathbf{V}_g \cdot \nabla\left(\frac{1}{f_0}\nabla^2\Phi + f\right)\right]}_{B} + \underbrace{\frac{1}{\sigma}\nabla^2\left[\mathbf{V}_g \cdot \nabla\left(-\frac{\partial\Phi}{\partial p}\right)\right]}_{C} - \underbrace{\frac{\kappa}{\sigma p}\nabla^2 J}_{D} \qquad (6.34)$$

其中，B 項為渦度水平平流的垂直差異（differential vorticity advection），C 項為溫度水平平流的 Laplacian，D 項為非絕熱的 Laplacian。

$$\underbrace{\left(\nabla^2 + \frac{f_0^2}{\sigma}\frac{\partial^2}{\partial p^2}\right)}_{A}\omega \approx \frac{f_0}{\sigma}\left[\frac{\partial \mathbf{V}_g}{\partial p} \cdot \nabla\left(\frac{1}{f_0}\nabla^2\Phi + f\right)\right]$$

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

```python
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)/g
ap)+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)
```