数值天气预报课程作业

吴浩杰 20221170212

1 实习十

正压原始方程模式的二次守恒格式

1.1 实习目的

通过编程设计正压原始方程模式二次守恒平流格式，使学生深入理解数值天气预报模式的动力框架，掌握二次守恒平流格式的基本原理及计算方法。

1.2 实习要求

编写运行正压原始方程模式二次守恒平流格式的 MATLAB 程序，撰写实习报告，分析程序流程和正压原始方程的积分结果。要求学生在机房现场操作，实习教师随堂讲解和指导。

1.3 实习内容

利用课堂的知识进行预报。

Listing 1: 实习内容

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
单文件脚本：读取 geo_197901.nc，做 12 小时积分预报，
使用固定边界条件、不随积分改变；每步后做空间平滑，
并在最终结果上做三点时间平滑 + 再空间平滑，
绘制初始场与平滑后预报场的 500 hPa 重力位势高度
"""

import os
import numpy as np
import xarray as xr
from scipy.interpolate import griddata
```

```python
from tqdm import tqdm
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from cartopy.util import add_cyclic_point

# — 中文与负号配置 —
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

def cmf(d, clat, clon, m, n):
    R = 6_371_000.0
     = 2*np.pi/(23*3600+56*60+4)
     = 30.0
    kk = ((np.log(np.sin(np.deg2rad(30))) - np.log(np.sin(np.deg2rad(60)))) /
          (np.log(np.tan(np.deg2rad(30/2))) - np.log(np.tan(np.deg2rad(60/2)))))
    le = R*np.sin(np.deg2rad( ))/kk*(1/np.tan(np.deg2rad( /2)))**kk
    l1 = le*(np.tan(np.deg2rad(clat/2)))**kk

    rm   = np.zeros((m,n))
    f    = np.zeros((m,n))
    lmda = np.zeros((m,n))
    phai = np.zeros((m,n))

    for i in range(m):
        for j in range(n):
            II = i-(m-1)/2
            JJ = l1/d + (n-1)/2 - j
            L  = np.hypot(II,JJ)*d
            lt = (le**(2/kk) - L**(2/kk))/(le**(2/kk) + L**(2/kk))
              = np.rad2deg(np.arcsin(lt))
              = np.rad2deg(np.arctan2(II,JJ))/kk + clon
            rm[i,j] = (np.sin(np.deg2rad( ))
                       /np.sin(np.deg2rad(90- ))
                       *(np.tan(np.deg2rad((90- )/2))
                         /np.tan(np.deg2rad( /2)))**kk)
            f[i,j]  = 2* *np.sin(np.deg2rad( ))
            lmda[i,j], phai[i,j] =  ,

    return rm, f, lmda, phai
```

```
def cgw(za, rm, f, d, m, n):
    c = 9.8/d
    ua = np.zeros((m,n)); va = np.zeros((m,n))
    for i in range(m):
        ua[i,0]   = -c*rm[i,0]*(za[i,1]-za[i,0])/f[i,0]
        ua[i,n-1] = -c*rm[i,n-1]*(za[i,n-1]-za[i,n-2])/f[i,n-1]
        for j in range(1,n-1):
            ua[i,j] = -c*rm[i,j]*(za[i,j+1]-za[i,j])/f[i,j]
    for j in range(n):
        va[0,j]   =  c*rm[0,j]*(za[1,j]-za[0,j])/f[0,j]
        va[m-1,j] =  c*rm[m-1,j]*(za[m-1,j]-za[m-2,j])/f[m-1,j]
        for i in range(1,m-1):
            va[i,j] = c*rm[i,j]*(za[i+1,j]-za[i,j])/f[i,j]
    return ua, va

def interp_proj_grid(u, v, z, lmda, phai, m, n, lon, lat):
    Lon, Lat = np.meshgrid(lon, lat)
    pts_src = np.column_stack((Lon.ravel(), Lat.ravel()))
    pts_tgt = np.column_stack((lmda.ravel(), phai.ravel()))

    ui_lin = griddata(pts_src, u.ravel(), pts_tgt, method='
        linear')
    ui_nn  = griddata(pts_src, u.ravel(), pts_tgt, method='
        nearest')
    vi_lin = griddata(pts_src, v.ravel(), pts_tgt, method='
        linear')
    vi_nn  = griddata(pts_src, v.ravel(), pts_tgt, method='
        nearest')
    zi_lin = griddata(pts_src, z.ravel(), pts_tgt, method='
        linear')
    zi_nn  = griddata(pts_src, z.ravel(), pts_tgt, method='
        nearest')

    ui = ui_lin.copy(); vi = vi_lin.copy(); zi = zi_lin.copy()
    mask = np.isnan(ui); ui[mask] = ui_nn[mask]
    mask = np.isnan(vi); vi[mask] = vi_nn[mask]
    mask = np.isnan(zi); zi[mask] = zi_nn[mask]

    return ui.reshape(m,n), vi.reshape(m,n), zi.reshape(m,n)

def ti(ua, va, za, ub, vb, zb, rm, f, d, dt, zo, m, n):
    c = 0.25/d
```

```
    uc, vc, zc = np.zeros_like(ua), np.zeros_like(va), np.zeros_like(za)
    m1, n1 = m-1, n-1
    for i in range(1,m1):
        for j in range(1,n1):
            e = (-c*rm[i,j]*(
                    (ub[i+1,j]+ub[i,j])*(ub[i+1,j]-ub[i,j]) +
                    (ub[i,j]+ub[i-1,j])*(ub[i,j]-ub[i-1,j]) +
                    (vb[i,j-1]+vb[i,j])*(ub[i,j]-ub[i,j-1]) +
                    (vb[i,j]+vb[i,j+1])*(ub[i,j+1]-ub[i,j]) +
                    19.6*(zb[i+1,j]-zb[i-1,j])
                ) + f[i,j]*vb[i,j])
            uc[i,j] = ua[i,j] + e*dt
            g = (-c*rm[i,j]*(
                    (ub[i+1,j]+ub[i,j])*(vb[i+1,j]-vb[i,j]) +
                    (ub[i,j]+ub[i-1,j])*(vb[i,j]-vb[i-1,j]) +
                    (vb[i,j-1]+vb[i,j])*(vb[i,j]-vb[i,j-1]) +
                    (vb[i,j]+vb[i,j+1])*(vb[i,j+1]-vb[i,j]) +
                    19.6*(zb[i,j+1]-zb[i,j-1])
                ) - f[i,j]*ub[i,j])
            vc[i,j] = va[i,j] + g*dt
    for i in range(1,m1):
        for j in range(1,n1):
            h = (-c*rm[i,j]**2*(
                    (ub[i+1,j]+ub[i,j])*(zb[i+1,j]/rm[i+1,j]-zb[i,j]/rm[i,j]) +
                    (ub[i,j]+ub[i-1,j])*(zb[i,j]/rm[i,j]-zb[i-1,j]/rm[i-1,j]) +
                    (vb[i,j-1]+vb[i,j])*(zb[i,j]/rm[i,j]-zb[i,j-1]/rm[i,j-1]) +
                    (vb[i,j]+vb[i,j+1])*(zb[i,j+1]/rm[i,j+1]-zb[i,j]/rm[i,j]) +
                    2*(zb[i,j]-zo)/rm[i,j]*
                    (ub[i+1,j]-ub[i-1,j]+vb[i,j+1]-vb[i,j-1])
                ))
            zc[i,j] = za[i,j] + h*dt
    return uc, vc, zc

def ssbp(a, s, m, n):
    w = a.copy()
    m1, n1 = m-1, n-1
    for i in range(1,m1):
        for j in (1,n1-1):
```

```
            w[i,j] = (a[i,j]
                      +0.5*s*(1-s)*(a[i-1,j]+a[i+1,j]+a[i,j-1]+a[i,j+1]-4*a[i,j])
                      +0.25*s*s*(a[i-1,j-1]+a[i-1,j+1]+
                                 a[i+1,j-1]+a[i+1,j+1]-4*a[i,j]))
    for i in (1,m1-1):
        for j in range(1,n1):
            w[i,j] = (a[i,j]
                      +0.5*s*(1-s)*(a[i-1,j]+a[i+1,j]+a[i,j-1]+a[i,j+1]-4*a[i,j])
                      +0.25*s*s*(a[i-1,j-1]+a[i-1,j+1]+
                                 a[i+1,j-1]+a[i+1,j+1]-4*a[i,j]))
    return w

def time_smooth(za, zb, zc, s=0.5):
    zb2 = zb.copy()
    zb2[1:-1,1:-1] = zb[1:-1,1:-1] + s*(za[1:-1,1:-1] + zc[1:-1,1:-1] - 2*zb[1:-1,1:-1]) / 2
    return zb2

def plot_field(ax, data, lmda, phai, label):
    ax.set_extent([60,180,20,80], crs=ccrs.PlateCarree())
    ax.coastlines('50m', linewidth=0.8)
    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                      linestyle='--', linewidth=0.5, color='gray')
    gl.top_labels = False; gl.right_labels = False
    gl.xlocator = plt.FixedLocator(np.arange(60,181,30))
    gl.ylocator = plt.FixedLocator(np.arange(20,81,10))
    levels = np.arange(5000,5751,150)
    cs = ax.contour(lmda, phai, data, levels=levels,
                    colors='black', linewidths=1.2,
                    transform=ccrs.PlateCarree())
    ax.clabel(cs, fmt='%d', inline=True, fontsize=8)
    ax.set_title(label, fontsize=12, pad=10)

def main():
    nc = 'geo_197901.nc'
    if not os.path.isfile(nc):
        raise FileNotFoundError(f"找不到数据文件: {nc}")

```

```python
    ds = xr.open_dataset(nc, decode_times=False)
    if 'pressure_level' in ds.dims:
        ds = ds.sel(pressure_level=500)
    z500 = ds['z']/9.8
    u500 = ds['u']; v500 = ds['v']
    lon = ds.longitude.values; lat = ds.latitude.values

    # 初始场 t=0
    u0 = u500.isel(valid_time=0).values
    v0 = v500.isel(valid_time=0).values
    z0 = z500.isel(valid_time=0).values

    # 参数
    m,n = 41,17
    d, clat, clon = 300000.0, 45.0, 120.0
    dt, zo, s = 150.0, 0.0, 0.5

    # 投影 & 静力初始化 & 初始空间平滑
    rm,f,lmda,phai = cmf(d,clat,clon,m,n)
    ua,va,za = interp_proj_grid(u0,v0,z0,lmda,phai,m,n,lon,lat)
    ua,va    = cgw(za,rm,f,d,m,n)
    za0 = ssbp(za.copy(),s,m,n)

    # 记录固定边界
    ua_bt, ua_bb = ua[0,:].copy(), ua[-1,:].copy()
    ua_bl, ua_br = ua[:,0].copy(), ua[:,-1].copy()
    va_bt, va_bb = va[0,:].copy(), va[-1,:].copy()
    va_bl, va_br = va[:,0].copy(), va[:,-1].copy()
    za_bt, za_bb = za[0,:].copy(), za[-1,:].copy()
    za_bl, za_br = za[:,0].copy(), za[:,-1].copy()

    # 初始半步场
    ub, vb, zb = ua.copy(), va.copy(), za.copy()

    zb_mid = None
    for k in tqdm(range(6), desc="积分进度"):
        # 半步预测
        tu,tv,tz = ti(ua,va,za, ua,va,za, rm,f,d,dt,zo,m,n)
        ub[1:-1,1:-1] = tu[1:-1,1:-1]
        vb[1:-1,1:-1] = tv[1:-1,1:-1]
        zb[1:-1,1:-1] = tz[1:-1,1:-1]
        # 固定边界回置
        ub[0,:], ub[-1,:] = ua_bt, ua_bb; ub[:,0], ub[:,-1] =
```

```
        ua_bl, ua_br
        vb[0,:], vb[-1,:] = va_bt, va_bb; vb[:,0], vb[:,-1] = va_bl, va_br
        zb[0,:], zb[-1,:] = za_bt, za_bb; zb[:,0], zb[:,-1] = za_bl, za_br
        # 空间平滑
        ub = ssbp(ub,s,m,n); vb = ssbp(vb,s,m,n); zb = ssbp(zb,s,m,n)

        # 半步修正
        tu,tv,tz = ti(ua,va,za, ub,vb,zb, rm,f,d,dt,zo,m,n)
        ua[1:-1,1:-1] = tu[1:-1,1:-1]
        va[1:-1,1:-1] = tv[1:-1,1:-1]
        za[1:-1,1:-1] = tz[1:-1,1:-1]
        # 固定边界回置
        ua[0,:], ua[-1,:] = ua_bt, ua_bb; ua[:,0], ua[:,-1] = ua_bl, ua_br
        va[0,:], va[-1,:] = va_bt, va_bb; va[:,0], va[:,-1] = va_bl, va_br
        za[0,:], za[-1,:] = za_bt, za_bb; za[:,0], za[:,-1] = za_bl, za_br
        # 空间平滑
        ua = ssbp(ua,s,m,n); va = ssbp(va,s,m,n); za = ssbp(za,s,m,n)

        if k == 2:
            zb_mid = za.copy()

    za12 = za

    # 时间平滑 + 再空间平滑
    zb_ts = time_smooth(za0, zb_mid, za12, s)
    zb_ts = ssbp(zb_ts, s, m, n)

    # 绘图
    fig, (ax1, ax2) = plt.subplots(1,2,figsize=(15,6),
        subplot_kw={'projection': ccrs.LambertConformal(
            central_longitude=clon, central_latitude=clat,
            standard_parallels=(30,60))})
    plot_field(ax1, za0,   lmda, phai, '(a) 原始场')
    plot_field(ax2, zb_ts, lmda, phai, '(b) 平滑后 12h 场')

    plt.suptitle('1979年1月10日 500hPa 重力位势高度场', fontsize
```

```python
            =14, y=1.02)
    plt.tight_layout()
    plt.show()

if __name__ == '__main__':
    main()
```

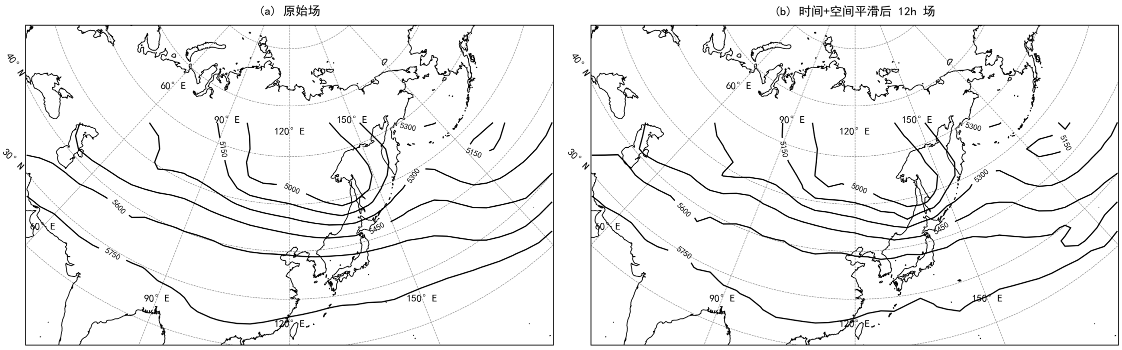

Figure 1: （实习内容）原始场与预报场对比

2 习题

2.1 习题一

试编程实现用 Runge Kutta 4 阶积分方案积分正压原始方程模式。Runge Kutta 4 阶积分公式通常可表示为

$$
\begin{cases}
y_{n+1} = y_n + \dfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\
k_1 = h\, f(x_n,\, y_n), \\
k_2 = h\, f\Big(x_n + \dfrac{h}{2},\, y_n + \dfrac{k_1}{2}\Big), \\
k_3 = h\, f\Big(x_n + \dfrac{h}{2},\, y_n + \dfrac{k_2}{2}\Big), \\
k_4 = h\, f\Big(x_n + h,\, y_n + k_3\Big).
\end{cases}
$$

Listing 2: 习题一

```python

# -*- coding: utf-8 -*-
"""
Runge-Kutta 4阶积分方案实现正压原始方程模式
基于MATLAB代码solve.f90和integration.f90的Python实现
"""

```

```python
import os
import numpy as np
import xarray as xr
from scipy.interpolate import griddata
from tqdm import tqdm
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
from cartopy.util import add_cyclic_point

# 中文与负号配置
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

def cmf(d, clat, clon, m, n):
    """兰伯特投影坐标变换"""
    R = 6371000.0
      = 2*np.pi/(23*3600+56*60+4)
      = 30.0
    kk = ((np.log(np.sin(np.deg2rad(30))) - np.log(np.sin(np.deg2rad(60)))) /
          (np.log(np.tan(np.deg2rad(30/2))) - np.log(np.tan(np.deg2rad(60/2)))))
    le = R*np.sin(np.deg2rad( ))/kk*(1/np.tan(np.deg2rad( /2)))**kk
    l1 = le*(np.tan(np.deg2rad(clat/2)))**kk

    rm   = np.zeros((m,n))
    f    = np.zeros((m,n))
    lmda = np.zeros((m,n))
    phai = np.zeros((m,n))

    for i in range(m):
        for j in range(n):
            II = i-(m-1)/2
            JJ = l1/d + (n-1)/2 - j
            L  = np.hypot(II,JJ)*d
            lt = (le**(2/kk) - L**(2/kk))/(le**(2/kk) + L**(2/kk))
              = np.rad2deg(np.arcsin(lt))
              = np.rad2deg(np.arctan2(II,JJ))/kk + clon
            rm[i,j] = (np.sin(np.deg2rad( ))
                      /np.sin(np.deg2rad(90- ))
                      *(np.tan(np.deg2rad((90- )/2))
```

```
                                /np.tan(np.deg2rad( /2)))**kk)
            f[i,j]   = 2* *np.sin(np.deg2rad( ))
            lmda[i,j], phai[i,j] =  ,

    return rm, f, lmda, phai

def cgw(za, rm, f, d, m, n):
    """地转风计算"""
    c = 9.8/d
    ua = np.zeros((m,n)); va = np.zeros((m,n))
    for i in range(m):
        ua[i,0]   = -c*rm[i,0]*(za[i,1]-za[i,0])/f[i,0]
        ua[i,n-1] = -c*rm[i,n-1]*(za[i,n-1]-za[i,n-2])/f[i,n-1]
        for j in range(1,n-1):
            ua[i,j] = -c*rm[i,j]*(za[i,j+1]-za[i,j])/f[i,j]
    for j in range(n):
        va[0,j]   =  c*rm[0,j]*(za[1,j]-za[0,j])/f[0,j]
        va[m-1,j] =  c*rm[m-1,j]*(za[m-1,j]-za[m-2,j])/f[m-1,j]
        for i in range(1,m-1):
            va[i,j] = c*rm[i,j]*(za[i+1,j]-za[i,j])/f[i,j]
    return ua, va

def interp_proj_grid(u, v, z, lmda, phai, m, n, lon, lat):
    """插值到投影网格"""
    Lon, Lat = np.meshgrid(lon, lat)
    pts_src = np.column_stack((Lon.ravel(), Lat.ravel()))
    pts_tgt = np.column_stack((lmda.ravel(), phai.ravel()))

    ui_lin = griddata(pts_src, u.ravel(), pts_tgt, method='linear')
    ui_nn  = griddata(pts_src, u.ravel(), pts_tgt, method='nearest')
    vi_lin = griddata(pts_src, v.ravel(), pts_tgt, method='linear')
    vi_nn  = griddata(pts_src, v.ravel(), pts_tgt, method='nearest')
    zi_lin = griddata(pts_src, z.ravel(), pts_tgt, method='linear')
    zi_nn  = griddata(pts_src, z.ravel(), pts_tgt, method='nearest')

    ui = ui_lin.copy(); vi = vi_lin.copy(); zi = zi_lin.copy()
    mask = np.isnan(ui); ui[mask] = ui_nn[mask]
```

```python
    mask = np.isnan(vi); vi[mask] = vi_nn[mask]
    mask = np.isnan(zi); zi[mask] = zi_nn[mask]

    return ui.reshape(m,n), vi.reshape(m,n), zi.reshape(m,n)

def tbv(ub, vb, zb, m, n):
    """边界条件处理"""
    ua, va, za = ub.copy(), vb.copy(), zb.copy()
    ua[:,[0,n-1]] = ub[:,[0,n-1]]; va[:,[0,n-1]] = vb[:,[0,n-1]]; za[:,[0,n-1]] = zb[:,[0,n-1]]
    ua[[0,m-1],:] = ub[[0,m-1],:]; va[[0,m-1],:] = vb[[0,m-1],:]; za[[0,m-1],:] = zb[[0,m-1],:]
    return ua, va, za

def compute_tendency(u, v, z, rm, f, d, zo, m, n):
    """计算时间倾向项（基于正压原始方程)"""
    c = 0.25/d
    u_t = np.zeros_like(u)
    v_t = np.zeros_like(v)
    z_t = np.zeros_like(z)

    m1, n1 = m-1, n-1

    # u方程时间倾向
    for i in range(1, m1):
        for j in range(1, n1):
            # 平流项
            advection_u = (-c * rm[i,j] * (
                (u[i+1,j] + u[i,j]) * (u[i+1,j] - u[i,j]) +
                (u[i,j] + u[i-1,j]) * (u[i,j] - u[i-1,j]) +
                (v[i,j-1] + v[i,j]) * (u[i,j] - u[i,j-1]) +
                (v[i,j] + v[i,j+1]) * (u[i,j+1] - u[i,j]) +
                19.6 * (z[i+1,j] - z[i-1,j])
            ))
            # 科里奥利力项
            coriolis_u = f[i,j] * v[i,j]
            u_t[i,j] = advection_u + coriolis_u

    # v方程时间倾向
    for i in range(1, m1):
        for j in range(1, n1):
            # 平流项
            advection_v = (-c * rm[i,j] * (
```

```python
                (u[i+1,j] + u[i,j]) * (v[i+1,j] - v[i,j]) +
                (u[i,j] + u[i-1,j]) * (v[i,j] - v[i-1,j]) +
                (v[i,j-1] + v[i,j]) * (v[i,j] - v[i,j-1]) +
                (v[i,j] + v[i,j+1]) * (v[i,j+1] - v[i,j]) +
                19.6 * (z[i,j+1] - z[i,j-1])
            ))
            # 科里奥利力项
            coriolis_v = -f[i,j] * u[i,j]
            v_t[i,j] = advection_v + coriolis_v

    # z方程时间倾向 (连续方程)
    for i in range(1, m1):
        for j in range(1, n1):
            # 散度项
            divergence = (-c * rm[i,j]**2 * (
                (u[i+1,j] + u[i,j]) * (z[i+1,j]/rm[i+1,j] - z[i,j]/rm[i,j]) +
                (u[i,j] + u[i-1,j]) * (z[i,j]/rm[i,j] - z[i-1,j]/rm[i-1,j]) +
                (v[i,j-1] + v[i,j]) * (z[i,j]/rm[i,j] - z[i,j-1]/rm[i,j-1]) +
                (v[i,j] + v[i,j+1]) * (z[i,j+1]/rm[i,j+1] - z[i,j]/rm[i,j]) +
                2 * (z[i,j] - zo) / rm[i,j] *
                (u[i+1,j] - u[i-1,j] + v[i,j+1] - v[i,j-1])
            ))
            z_t[i,j] = divergence

    return u_t, v_t, z_t

def runge_kutta_4(u, v, z, rm, f, d, dt, zo, m, n):
    """Runge-Kutta 4阶积分方案

    根据公式:
    y_{n+1} = y_n + (1/6)(k1 + 2*k2 + 2*k3 + k4)

    其中:
    k1 = h*f(x_n, y_n)
    k2 = h*f(x_n + h/2, y_n + k1/2)
    k3 = h*f(x_n + h/2, y_n + k2/2)
    k4 = h*f(x_n + h, y_n + k3)
    """

```

```python
    # 计算 k1
    k1_u, k1_v, k1_z = compute_tendency(u, v, z, rm, f, d, zo, m, n)
    k1_u *= dt
    k1_v *= dt
    k1_z *= dt

    # 计算 k2
    u_temp = u + 0.5 * k1_u
    v_temp = v + 0.5 * k1_v
    z_temp = z + 0.5 * k1_z
    u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)

    k2_u, k2_v, k2_z = compute_tendency(u_temp, v_temp, z_temp, rm, f, d, zo, m, n)
    k2_u *= dt
    k2_v *= dt
    k2_z *= dt

    # 计算 k3
    u_temp = u + 0.5 * k2_u
    v_temp = v + 0.5 * k2_v
    z_temp = z + 0.5 * k2_z
    u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)

    k3_u, k3_v, k3_z = compute_tendency(u_temp, v_temp, z_temp, rm, f, d, zo, m, n)
    k3_u *= dt
    k3_v *= dt
    k3_z *= dt

    # 计算 k4
    u_temp = u + k3_u
    v_temp = v + k3_v
    z_temp = z + k3_z
    u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)

    k4_u, k4_v, k4_z = compute_tendency(u_temp, v_temp, z_temp, rm, f, d, zo, m, n)
    k4_u *= dt
    k4_v *= dt
    k4_z *= dt
```

```
    # RK4最终结果
    u_new = u + (k1_u + 2*k2_u + 2*k3_u + k4_u) / 6.0
    v_new = v + (k1_v + 2*k2_v + 2*k3_v + k4_v) / 6.0
    z_new = z + (k1_z + 2*k2_z + 2*k3_z + k4_z) / 6.0

    return u_new, v_new, z_new

def ssbp(a, s, m, n):
    """空间平滑"""
    w = a.copy()
    m1, n1 = m-1, n-1
    for i in range(1,m1):
        for j in range(1,n1):
            w[i,j] = (a[i,j]
                      +0.5*s*(1-s)*(a[i-1,j]+a[i+1,j]+a[i,j-1]+a
                          [i,j+1]-4*a[i,j])
                      +0.25*s*s*(a[i-1,j-1]+a[i-1,j+1]+a[i+1,j
                          -1]+a[i+1,j+1]-4*a[i,j]))
    return w

def time_smooth(za, zb, zc, s=0.5):
    """时间平滑"""
    zb2 = zb.copy()
    zb2[1:-1,1:-1] = zb[1:-1,1:-1] + s*(za[1:-1,1:-1] + zc
        [1:-1,1:-1] - 2*zb[1:-1,1:-1])/2
    return zb2

def plot_field(ax, data, lmda, phai, label):
    """绘制等高线图"""
    ax.set_extent([60,180,20,80], crs=ccrs.PlateCarree())
    ax.coastlines('50m', linewidth=0.8)
    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
                      linestyle='--', linewidth=0.5, color='gray
                          ')
    gl.top_labels = False; gl.right_labels = False
    gl.xlocator = plt.FixedLocator(np.arange(60,181,30))
    gl.ylocator = plt.FixedLocator(np.arange(20,81,10))
    levels = np.arange(5000,5751,150)
    cs = ax.contour(lmda, phai, data, levels=levels,
                    colors='black', linewidths=1.2,
                    transform=ccrs.PlateCarree())
    ax.clabel(cs, fmt='%d', inline=True, fontsize=8)
    ax.set_title(label, fontsize=12, pad=10)
```

```python

def main():
    """主函数 - 使用RK4积分方案的正压原始方程模式"""
    # 读取数据
    nc = 'geo_197901.nc'
    if not os.path.isfile(nc):
        raise FileNotFoundError(f"找不到数据文件: {nc}")

    ds = xr.open_dataset(nc, decode_times=False)
    if 'pressure_level' in ds.dims:
        ds = ds.sel(pressure_level=500)
    z500 = ds['z']/9.8
    u500 = ds['u']; v500 = ds['v']
    lon = ds.longitude.values; lat = ds.latitude.values

    # 初始场 t=0
    nt0 = 0
    u0 = u500.isel(valid_time=nt0).values
    v0 = v500.isel(valid_time=nt0).values
    z0 = z500.isel(valid_time=nt0).values

    # 模式参数
    m, n = 41, 17  # 网格点数
    d, clat, clon = 300000.0, 45.0, 120.0  # 网格距、中心纬度、中心经度
    dt, zo, s = 150.0, 0.0, 0.5  # 时间步长、参考高度、平滑系数

    print("正在初始化模式...")

    # 投影坐标变换和静力初始化
    rm, f, lmda, phai = cmf(d, clat, clon, m, n)
    ua, va, za = interp_proj_grid(u0, v0, z0, lmda, phai, m, n, lon, lat)
    ua, va = cgw(za, rm, f, d, m, n)  # 地转风初始化
    za0 = ssbp(za.copy(), s, m, n)  # 初始场空间平滑

    # 边界条件初始化
    ub, vb, zb = tbv(ua, va, za, m, n)

    print("开始RK4积分...")

    # 使用RK4进行时间积分 (6步 = 12小时)
    n_steps = 6
```

```python
    zb_mid = None

    for k in tqdm(range(n_steps), desc="RK4积分进度"):
        # 使用RK4方案进行一个时间步的积分
        ub_new, vb_new, zb_new = runge_kutta_4(ub, vb, zb, rm, f
            , d, dt, zo, m, n)

        # 更新变量
        ub[1:-1, 1:-1] = ub_new[1:-1, 1:-1]
        vb[1:-1, 1:-1] = vb_new[1:-1, 1:-1]
        zb[1:-1, 1:-1] = zb_new[1:-1, 1:-1]

        # 边界条件处理
        ub, vb, zb = tbv(ub, vb, zb, m, n)

        # 空间平滑
        ub = ssbp(ub, s, m, n)
        vb = ssbp(vb, s, m, n)
        zb = ssbp(zb, s, m, n)

        # 记录中间时刻的场
        if k == 2:  # 第3步，即6小时
            zb_mid = zb.copy()

    za12 = zb  # 12小时预报场

    print("进行时间和空间平滑...")

    # 时间平滑 + 再次空间平滑
    zb_ts = time_smooth(za0, zb_mid, za12, s)
    zb_ts = ssbp(zb_ts, s, m, n)

    print("绘制结果...")

    # 绘制对比图
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6),
        subplot_kw={'projection': ccrs.LambertConformal(
            central_longitude=clon, central_latitude=clat,
            standard_parallels=(30, 60))})

    plot_field(ax1, za0, lmda, phai, '(a) 初始场 (RK4)')
    plot_field(ax2, zb_ts, lmda, phai, '(b) RK4积分12h预报场')

```

```python
    plt.suptitle('RK4积分方案 - 1979年1月10日 500hPa 重力位势高度场', fontsize=14, y=1.02)
    plt.tight_layout()
    plt.show()

    print("RK4积分完成！")

    # 输出一些统计信息
    print(f"初始场最大值: {np.max(za0):.2f} m")
    print(f"初始场最小值: {np.min(za0):.2f} m")
    print(f"12h预报场最大值: {np.max(zb_ts):.2f} m")
    print(f"12h预报场最小值: {np.min(zb_ts):.2f} m")
    print(f"最大变化: {np.max(np.abs(zb_ts - za0)):.2f} m")

if __name__ == '__main__':
    main()
```

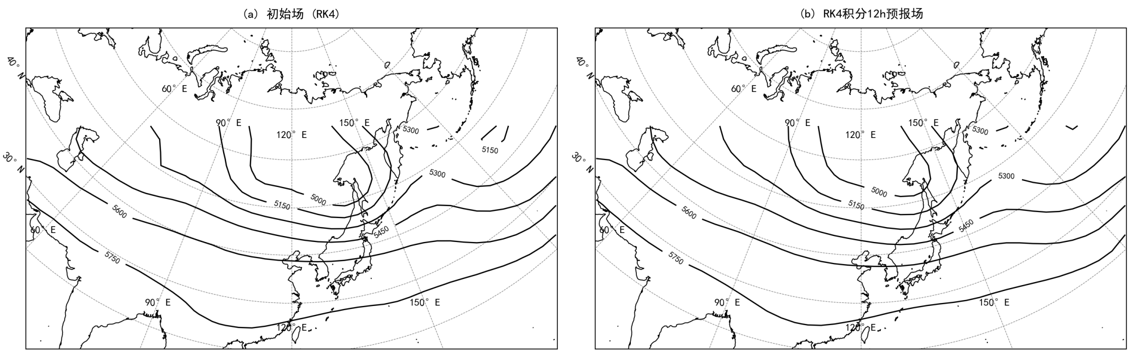

Figure 2: σ 习题一原始场与预报场对比

习题一参考了 BPEM 的相关代码 solve.f90 和 integration.f90，预报结果和原始场相比更加平滑，也有较大的变化。