

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

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 单文件脚本：读取 geo_197901.nc，做 12 小时积分预报，
5 使用固定边界条件、不随积分改变；每步后做空间平滑，
6 并在最终结果上做三点时间平滑 + 再空间平滑，
7 绘制初始场与平滑后预报场的 500 hPa 重力位势高度
8 """
9
10 import os
11 import numpy as np
12 import xarray as xr
13 from scipy.interpolate import griddata
```

```

14 from tqdm import tqdm
15 import matplotlib.pyplot as plt
16 import cartopy.crs as ccrs
17 from cartopy.util import add_cyclic_point
18
19 # — 中文与负号配置 —
20 plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
21 plt.rcParams['axes.unicode_minus'] = False
22
23 def cmf(d, clat, clon, m, n):
24     R = 6_371_000.0
25     = 2*np.pi/(23*3600+56*60+4)
26     = 30.0
27     kk = ((np.log(np.sin(np.deg2rad(30))) - np.log(np.sin(np.
28         deg2rad(60)))) /
29         (np.log(np.tan(np.deg2rad(30/2))) - np.log(np.tan(np.
30         deg2rad(60/2)))))
31
32     le = R*np.sin(np.deg2rad( ))/kk*(1/np.tan(np.deg2rad( /2)))
33         **kk
34     ll = le*(np.tan(np.deg2rad(clat/2)))**kk
35
36
37     rm = np.zeros((m,n))
38     f = np.zeros((m,n))
39     lmda = np.zeros((m,n))
40     phai = np.zeros((m,n))
41
42     for i in range(m):
43         for j in range(n):
44             II = i-(m-1)/2
45             JJ = ll/d + (n-1)/2 - j
46             L = np.hypot(II, JJ)*d
47             lt = (le**(2/kk) - L**(2/kk))/(le**(2/kk) + L**(2/kk
48                 ))
49             = np.rad2deg(np.arcsin(lt))
50             = np.rad2deg(np.arctan2(II, JJ))/kk + clon
51             rm[i, j] = (np.sin(np.deg2rad( ))
52                 /np.sin(np.deg2rad(90- )))
53                 *(np.tan(np.deg2rad((90- )/2))
54                 /np.tan(np.deg2rad( /2)))**kk)
55             f[i, j] = 2* *np.sin(np.deg2rad( ))
56             lmda[i, j], phai[i, j] = ,
57
58     return rm, f, lmda, phai

```

```

53
54 def cgw(za, rm, f, d, m, n):
55     c = 9.8/d
56     ua = np.zeros((m,n)); va = np.zeros((m,n))
57     for i in range(m):
58         ua[i,0] = -c*rm[i,0]*(za[i,1]-za[i,0])/f[i,0]
59         ua[i,n-1] = -c*rm[i,n-1]*(za[i,n-1]-za[i,n-2])/f[i,n-1]
60         for j in range(1,n-1):
61             ua[i,j] = -c*rm[i,j]*(za[i,j+1]-za[i,j])/f[i,j]
62     for j in range(n):
63         va[0,j] = c*rm[0,j]*(za[1,j]-za[0,j])/f[0,j]
64         va[m-1,j] = c*rm[m-1,j]*(za[m-1,j]-za[m-2,j])/f[m-1,j]
65         for i in range(1,m-1):
66             va[i,j] = c*rm[i,j]*(za[i+1,j]-za[i,j])/f[i,j]
67     return ua, va
68
69 def interp_proj_grid(u, v, z, lmda, phai, m, n, lon, lat):
70     Lon, Lat = np.meshgrid(lon, lat)
71     pts_src = np.column_stack((Lon.ravel(), Lat.ravel()))
72     pts_tgt = np.column_stack((lmda.ravel(), phai.ravel()))
73
74     ui_lin = griddata(pts_src, u.ravel(), pts_tgt, method='
75         linear')
76     ui_nn = griddata(pts_src, u.ravel(), pts_tgt, method='
77         nearest')
78     vi_lin = griddata(pts_src, v.ravel(), pts_tgt, method='
79         linear')
80     vi_nn = griddata(pts_src, v.ravel(), pts_tgt, method='
81         nearest')
82     zi_lin = griddata(pts_src, z.ravel(), pts_tgt, method='
83         linear')
84     zi_nn = griddata(pts_src, z.ravel(), pts_tgt, method='
85         nearest')
86
87     ui = ui_lin.copy(); vi = vi_lin.copy(); zi = zi_lin.copy()
88     mask = np.isnan(ui); ui[mask] = ui_nn[mask]
89     mask = np.isnan(vi); vi[mask] = vi_nn[mask]
90     mask = np.isnan(zi); zi[mask] = zi_nn[mask]
91
92     return ui.reshape(m,n), vi.reshape(m,n), zi.reshape(m,n)
93
94 def ti(ua, va, za, ub, vb, zb, rm, f, d, dt, zo, m, n):
95     c = 0.25/d

```

```

90     uc, vc, zc = np.zeros_like(ua), np.zeros_like(va), np.
        zeros_like(za)
91     m1, n1 = m-1, n-1
92     for i in range(1,m1):
93         for j in range(1,n1):
94             e = (-c*rm[i,j]*(
95                 (ub[i+1,j]+ub[i,j])*(ub[i+1,j]-ub[i,j]) +
96                 (ub[i,j]+ub[i-1,j])*(ub[i,j]-ub[i-1,j]) +
97                 (vb[i,j-1]+vb[i,j])*(ub[i,j]-ub[i,j-1]) +
98                 (vb[i,j]+vb[i,j+1])*(ub[i,j+1]-ub[i,j]) +
99                 19.6*(zb[i+1,j]-zb[i-1,j])
100             ) + f[i,j]*vb[i,j])
101             uc[i,j] = ua[i,j] + e*dt
102             g = (-c*rm[i,j]*(
103                 (ub[i+1,j]+ub[i,j])*(vb[i+1,j]-vb[i,j]) +
104                 (ub[i,j]+ub[i-1,j])*(vb[i,j]-vb[i-1,j]) +
105                 (vb[i,j-1]+vb[i,j])*(vb[i,j]-vb[i,j-1]) +
106                 (vb[i,j]+vb[i,j+1])*(vb[i,j+1]-vb[i,j]) +
107                 19.6*(zb[i,j+1]-zb[i,j-1])
108             ) - f[i,j]*ub[i,j])
109             vc[i,j] = va[i,j] + g*dt
110     for i in range(1,m1):
111         for j in range(1,n1):
112             h = (-c*rm[i,j]**2*(
113                 (ub[i+1,j]+ub[i,j])*(zb[i+1,j]/rm[i+1,j]-zb[i,
114                     j]/rm[i,j]) +
115                 (ub[i,j]+ub[i-1,j])*(zb[i,j]/rm[i,j]-zb[i-1,j
116                     ]/rm[i-1,j]) +
117                 (vb[i,j-1]+vb[i,j])*(zb[i,j]/rm[i,j]-zb[i,j
118                     -1]/rm[i,j-1]) +
119                 (vb[i,j]+vb[i,j+1])*(zb[i,j+1]/rm[i,j+1]-zb[i,
120                     j]/rm[i,j]) +
121                 2*(zb[i,j]-zo)/rm[i,j]*
122                 (ub[i+1,j]-ub[i-1,j]+vb[i,j+1]-vb[i,j-1])
123             ))
124             zc[i,j] = za[i,j] + h*dt
125     return uc, vc, zc
126
127 def ssbp(a, s, m, n):
128     w = a.copy()
129     m1, n1 = m-1, n-1
130     for i in range(1,m1):
131         for j in (1,n1-1):

```

```

128         w[i, j] = (a[i, j]
129                     +0.5*s*(1-s)*(a[i-1, j]+a[i+1, j]+a[i, j-1]+a
130                                   [i, j+1]-4*a[i, j])
131                     +0.25*s*s*(a[i-1, j-1]+a[i-1, j+1]+
132                                   a[i+1, j-1]+a[i+1, j+1]-4*a[i, j])
133                     )
134     for i in (1,m1-1):
135         for j in range(1,n1):
136             w[i, j] = (a[i, j]
137                         +0.5*s*(1-s)*(a[i-1, j]+a[i+1, j]+a[i, j-1]+a
138                                       [i, j+1]-4*a[i, j])
139                         +0.25*s*s*(a[i-1, j-1]+a[i-1, j+1]+
140                                       a[i+1, j-1]+a[i+1, j+1]-4*a[i, j])
141                         )
142     return w
143
144 def time_smooth(za, zb, zc, s=0.5):
145     zb2 = zb.copy()
146     zb2[1:-1, 1:-1] = zb[1:-1, 1:-1] + s*(za[1:-1, 1:-1] + zc
147         [1:-1, 1:-1] - 2*zb[1:-1, 1:-1]) / 2
148     return zb2
149
150 def plot_field(ax, data, lmda, phai, label):
151     ax.set_extent([60, 180, 20, 80], crs=ccrs.PlateCarree())
152     ax.coastlines('50m', linewidth=0.8)
153     gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
154                       linestyle='--', linewidth=0.5, color='gray',
155                       )
156     gl.top_labels = False; gl.right_labels = False
157     gl.xlocator = plt.FixedLocator(np.arange(60, 181, 30))
158     gl.ylocator = plt.FixedLocator(np.arange(20, 81, 10))
159     levels = np.arange(5000, 5751, 150)
160     cs = ax.contour(lmda, phai, data, levels=levels,
161                     colors='black', linewidths=1.2,
162                     transform=ccrs.PlateCarree())
163     ax.clabel(cs, fmt='%d', inline=True, fontsize=8)
164     ax.set_title(label, fontsize=12, pad=10)
165
166 def main():
167     nc = 'geo_197901.nc'
168     if not os.path.isfile(nc):
169         raise FileNotFoundError(f"找不到数据文件: {nc}")

```

```

165 ds = xr.open_dataset(nc, decode_times=False)
166 if 'pressure_level' in ds.dims:
167     ds = ds.sel(pressure_level=500)
168 z500 = ds['z']/9.8
169 u500 = ds['u']; v500 = ds['v']
170 lon = ds.longitude.values; lat = ds.latitude.values
171
172 # 初始场 t=0
173 u0 = u500.isel(valid_time=0).values
174 v0 = v500.isel(valid_time=0).values
175 z0 = z500.isel(valid_time=0).values
176
177 # 参数
178 m,n = 41,17
179 d, clat, clon = 300000.0, 45.0, 120.0
180 dt, zo, s = 150.0, 0.0, 0.5
181
182 # 投影 & 静力初始化 & 初始空间平滑
183 rm,f,lmda,phai = cmf(d,clat,clon,m,n)
184 ua,va,za = interp_proj_grid(u0,v0,z0,lmda,phai,m,n,lon,lat)
185 ua,va = cgw(za,rm,f,d,m,n)
186 za0 = ssbp(za.copy(),s,m,n)
187
188 # 记录固定边界
189 ua_bt, ua_bb = ua[0,:].copy(), ua[-1,:].copy()
190 ua_bl, ua_br = ua[:,0].copy(), ua[:, -1].copy()
191 va_bt, va_bb = va[0,:].copy(), va[-1,:].copy()
192 va_bl, va_br = va[:,0].copy(), va[:, -1].copy()
193 za_bt, za_bb = za[0,:].copy(), za[-1,:].copy()
194 za_bl, za_br = za[:,0].copy(), za[:, -1].copy()
195
196 # 初始半步场
197 ub, vb, zb = ua.copy(), va.copy(), za.copy()
198
199 zb_mid = None
200 for k in tqdm(range(6), desc="积分进度"):
201     # 半步预测
202     tu,tv,tz = ti(ua,va,za, ua,va,za, rm,f,d,dt,zo,m,n)
203     ub[1:-1,1:-1] = tu[1:-1,1:-1]
204     vb[1:-1,1:-1] = tv[1:-1,1:-1]
205     zb[1:-1,1:-1] = tz[1:-1,1:-1]
206     # 固定边界回置
207     ub[0,:], ub[-1,:] = ua_bt, ua_bb; ub[:,0], ub[:, -1] =

```

```

208         ua_bl, ua_br
        vb[0,:], vb[-1,:] = va_bt, va_bb; vb[:,0], vb[:, -1] =
        va_bl, va_br
209        zb[0,:], zb[-1,:] = za_bt, za_bb; zb[:,0], zb[:, -1] =
        za_bl, za_br
210        # 空间平滑
211        ub = ssbp(ub,s,m,n); vb = ssbp(vb,s,m,n); zb = ssbp(zb,s
        ,m,n)
212
213        # 半步修正
214        tu,tv,tz = ti(ua,va,za, ub,vb,zb, rm,f,d,dt,zo,m,n)
215        ua[1:-1,1:-1] = tu[1:-1,1:-1]
216        va[1:-1,1:-1] = tv[1:-1,1:-1]
217        za[1:-1,1:-1] = tz[1:-1,1:-1]
218        # 固定边界回置
219        ua[0,:], ua[-1,:] = ua_bt, ua_bb; ua[:,0], ua[:, -1] =
        ua_bl, ua_br
220        va[0,:], va[-1,:] = va_bt, va_bb; va[:,0], va[:, -1] =
        va_bl, va_br
221        za[0,:], za[-1,:] = za_bt, za_bb; za[:,0], za[:, -1] =
        za_bl, za_br
222        # 空间平滑
223        ua = ssbp(ua,s,m,n); va = ssbp(va,s,m,n); za = ssbp(za,s
        ,m,n)
224
225        if k == 2:
226            zb_mid = za.copy()
227
228        za12 = za
229
230        # 时间平滑 + 再空间平滑
231        zb_ts = time_smooth(za0, zb_mid, za12, s)
232        zb_ts = ssbp(zb_ts, s, m, n)
233
234        # 绘图
235        fig, (ax1, ax2) = plt.subplots(1,2,figsize=(15,6),
236            subplot_kw={'projection': ccrs.LambertConformal(
237                central_longitude=clon, central_latitude=clat,
238                standard_parallels=(30,60))})
239        plot_field(ax1, za0, lmda, phai, '(a) 原始场')
240        plot_field(ax2, zb_ts, lmda, phai, '(b) 平滑后 12h 场')
241
242        plt.suptitle('1979年1月10日 500hPa 重力位势高度场', fontsize

```

```

    =14, y=1.02)
243 plt.tight_layout()
244 plt.show()
245
246 if __name__ == '__main__':
247     main()

```

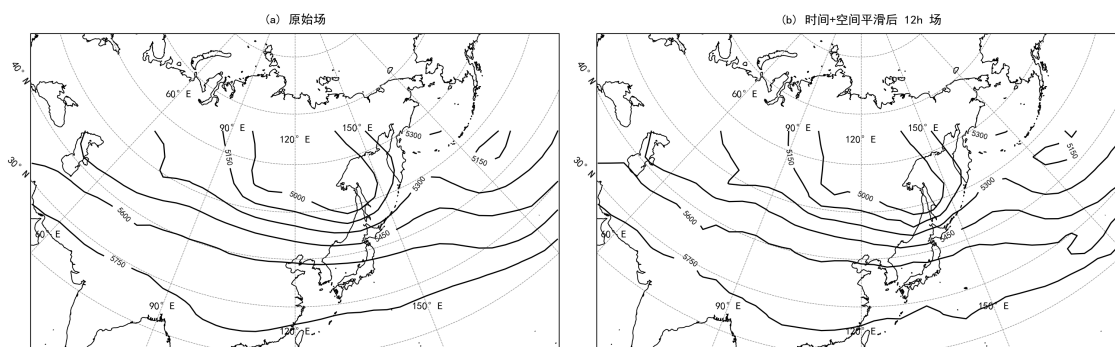


Figure 1: （实习内容）原始场与预报场对比

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

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

Listing 2: 习题一

```

1
2 # -*- coding: utf-8 -*-
3 """
4 Runge-Kutta 4阶积分方案实现正压原始方程模式
5 基于MATLAB代码solve.f90和integration.f90的Python实现
6 """
7

```



```

8 import os
9 import numpy as np
10 import xarray as xr
11 from scipy.interpolate import griddata
12 from tqdm import tqdm
13 import matplotlib.pyplot as plt
14 import cartopy.crs as ccrs
15 from cartopy.util import add_cyclic_point
16
17 # 中文与负号配置
18 plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
19 plt.rcParams['axes.unicode_minus'] = False
20
21 def cmf(d, clat, clon, m, n):
22     """兰伯特投影坐标变换"""
23     R = 6371000.0
24     = 2*np.pi/(23*3600+56*60+4)
25     = 30.0
26     kk = ((np.log(np.sin(np.deg2rad(30))) - np.log(np.sin(np.
27         deg2rad(60)))) /
28         (np.log(np.tan(np.deg2rad(30/2))) - np.log(np.tan(np.
29         deg2rad(60/2)))))
30
31     le = R*np.sin(np.deg2rad( ))/kk*(1/np.tan(np.deg2rad( /2)))
32     **kk
33
34     ll = le*(np.tan(np.deg2rad(clat/2)))*kk
35
36     rm = np.zeros((m,n))
37     f = np.zeros((m,n))
38     lmda = np.zeros((m,n))
39     phai = np.zeros((m,n))
40
41     for i in range(m):
42         for j in range(n):
43             II = i-(m-1)/2
44             JJ = ll/d + (n-1)/2 - j
45             L = np.hypot(II, JJ)*d
46             lt = (le**(2/kk) - L**(2/kk))/(le**(2/kk) + L**(2/kk
47                 ))
48             = np.rad2deg(np.arcsin(lt))
49             = np.rad2deg(np.arctan2(II, JJ))/kk + clon
50             rm[i, j] = (np.sin(np.deg2rad( ))
51                 /np.sin(np.deg2rad(90- )))
52                 *(np.tan(np.deg2rad((90- )/2))

```

```

47         /np.tan(np.deg2rad( /2)))**kk)
48     f[i,j] = 2* *np.sin(np.deg2rad( ))
49     lmda[i,j], phai[i,j] = ,
50
51     return rm, f, lmda, phai
52
53 def cgw(za, rm, f, d, m, n):
54     """地转风计算"""
55     c = 9.8/d
56     ua = np.zeros((m,n)); va = np.zeros((m,n))
57     for i in range(m):
58         ua[i,0] = -c*rm[i,0]*(za[i,1]-za[i,0])/f[i,0]
59         ua[i,n-1] = -c*rm[i,n-1]*(za[i,n-1]-za[i,n-2])/f[i,n-1]
60         for j in range(1,n-1):
61             ua[i,j] = -c*rm[i,j]*(za[i,j+1]-za[i,j])/f[i,j]
62     for j in range(n):
63         va[0,j] = c*rm[0,j]*(za[1,j]-za[0,j])/f[0,j]
64         va[m-1,j] = c*rm[m-1,j]*(za[m-1,j]-za[m-2,j])/f[m-1,j]
65         for i in range(1,m-1):
66             va[i,j] = c*rm[i,j]*(za[i+1,j]-za[i,j])/f[i,j]
67     return ua, va
68
69 def interp_proj_grid(u, v, z, lmda, phai, m, n, lon, lat):
70     """插值到投影网格"""
71     Lon, Lat = np.meshgrid(lon, lat)
72     pts_src = np.column_stack((Lon.ravel(), Lat.ravel()))
73     pts_tgt = np.column_stack((lmda.ravel(), phai.ravel()))
74
75     ui_lin = griddata(pts_src, u.ravel(), pts_tgt, method='
76         linear')
77     ui_nn = griddata(pts_src, u.ravel(), pts_tgt, method='
78         nearest')
79     vi_lin = griddata(pts_src, v.ravel(), pts_tgt, method='
80         linear')
81     vi_nn = griddata(pts_src, v.ravel(), pts_tgt, method='
82         nearest')
83     zi_lin = griddata(pts_src, z.ravel(), pts_tgt, method='
84         linear')
85     zi_nn = griddata(pts_src, z.ravel(), pts_tgt, method='
86         nearest')
87
88     ui = ui_lin.copy(); vi = vi_lin.copy(); zi = zi_lin.copy()
89     mask = np.isnan(ui); ui[mask] = ui_nn[mask]

```

```

84     mask = np.isnan(vi); vi[mask] = vi_nn[mask]
85     mask = np.isnan(zi); zi[mask] = zi_nn[mask]
86
87     return ui.reshape(m,n), vi.reshape(m,n), zi.reshape(m,n)
88
89 def tbv(ub, vb, zb, m, n):
90     """边界条件处理"""
91     ua, va, za = ub.copy(), vb.copy(), zb.copy()
92     ua[:,[0,n-1]] = ub[:,[0,n-1]]; va[:,[0,n-1]] = vb[:,[0,n-1]]
93     za[:,[0,n-1]] = zb[:,[0,n-1]]
94     ua[[0,m-1],:] = ub[[0,m-1],:]; va[[0,m-1],:] = vb[[0,m-1],:]
95     za[[0,m-1],:] = zb[[0,m-1],:]
96     return ua, va, za
97
98 def compute_tendency(u, v, z, rm, f, d, zo, m, n):
99     """计算时间倾向项 (基于正压原始方程)"""
100     c = 0.25/d
101     u_t = np.zeros_like(u)
102     v_t = np.zeros_like(v)
103     z_t = np.zeros_like(z)
104
105     m1, n1 = m-1, n-1
106
107     # u方程时间倾向
108     for i in range(1, m1):
109         for j in range(1, n1):
110             # 平流项
111             advection_u = (-c * rm[i,j] * (
112                 (u[i+1,j] + u[i,j]) * (u[i+1,j] - u[i,j]) +
113                 (u[i,j] + u[i-1,j]) * (u[i,j] - u[i-1,j]) +
114                 (v[i,j-1] + v[i,j]) * (u[i,j] - u[i,j-1]) +
115                 (v[i,j] + v[i,j+1]) * (u[i,j+1] - u[i,j]) +
116                 19.6 * (z[i+1,j] - z[i-1,j]))
117             ))
118             # 科里奥利力项
119             coriolis_u = f[i,j] * v[i,j]
120             u_t[i,j] = advection_u + coriolis_u
121
122     # v方程时间倾向
123     for i in range(1, m1):
124         for j in range(1, n1):
125             # 平流项
126             advection_v = (-c * rm[i,j] * (

```

```

125         (u[i+1,j] + u[i,j]) * (v[i+1,j] - v[i,j]) +
126         (u[i,j] + u[i-1,j]) * (v[i,j] - v[i-1,j]) +
127         (v[i,j-1] + v[i,j]) * (v[i,j] - v[i,j-1]) +
128         (v[i,j] + v[i,j+1]) * (v[i,j+1] - v[i,j]) +
129         19.6 * (z[i,j+1] - z[i,j-1])
130     ))
131     # 科里奥利力项
132     coriolis_v = -f[i,j] * u[i,j]
133     v_t[i,j] = advection_v + coriolis_v
134
135     # z方程时间倾向 (连续方程)
136     for i in range(1, m1):
137         for j in range(1, n1):
138             # 散度项
139             divergence = (-c * rm[i,j]**2 * (
140                 (u[i+1,j] + u[i,j]) * (z[i+1,j]/rm[i+1,j] - z[i,
141                     j]/rm[i,j]) +
142                 (u[i,j] + u[i-1,j]) * (z[i,j]/rm[i,j] - z[i-1,j
143                     ]/rm[i-1,j]) +
144                 (v[i,j-1] + v[i,j]) * (z[i,j]/rm[i,j] - z[i,j
145                     -1]/rm[i,j-1]) +
146                 (v[i,j] + v[i,j+1]) * (z[i,j+1]/rm[i,j+1] - z[i,
147                     j]/rm[i,j]) +
148                 2 * (z[i,j] - zo) / rm[i,j] *
149                 (u[i+1,j] - u[i-1,j] + v[i,j+1] - v[i,j-1])
150             ))
151             z_t[i,j] = divergence
152
153     return u_t, v_t, z_t
154
155 def runge_kutta_4(u, v, z, rm, f, d, dt, zo, m, n):
156     """Runge-Kutta 4阶积分方案
157
158     根据公式:
159      $y_{n+1} = y_n + (1/6)(k_1 + 2*k_2 + 2*k_3 + k_4)$ 
160
161     其中:
162      $k_1 = h*f(x_n, y_n)$ 
163      $k_2 = h*f(x_n + h/2, y_n + k_1/2)$ 
164      $k_3 = h*f(x_n + h/2, y_n + k_2/2)$ 
165      $k_4 = h*f(x_n + h, y_n + k_3)$ 
166     """

```

```

164 # 计算 k1
165 k1_u, k1_v, k1_z = compute_tendency(u, v, z, rm, f, d, zo, m
    , n)
166 k1_u *= dt
167 k1_v *= dt
168 k1_z *= dt
169
170 # 计算 k2
171 u_temp = u + 0.5 * k1_u
172 v_temp = v + 0.5 * k1_v
173 z_temp = z + 0.5 * k1_z
174 u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)
175
176 k2_u, k2_v, k2_z = compute_tendency(u_temp, v_temp, z_temp,
    rm, f, d, zo, m, n)
177 k2_u *= dt
178 k2_v *= dt
179 k2_z *= dt
180
181 # 计算 k3
182 u_temp = u + 0.5 * k2_u
183 v_temp = v + 0.5 * k2_v
184 z_temp = z + 0.5 * k2_z
185 u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)
186
187 k3_u, k3_v, k3_z = compute_tendency(u_temp, v_temp, z_temp,
    rm, f, d, zo, m, n)
188 k3_u *= dt
189 k3_v *= dt
190 k3_z *= dt
191
192 # 计算 k4
193 u_temp = u + k3_u
194 v_temp = v + k3_v
195 z_temp = z + k3_z
196 u_temp, v_temp, z_temp = tbv(u_temp, v_temp, z_temp, m, n)
197
198 k4_u, k4_v, k4_z = compute_tendency(u_temp, v_temp, z_temp,
    rm, f, d, zo, m, n)
199 k4_u *= dt
200 k4_v *= dt
201 k4_z *= dt
202

```

```

203     # RK4最终结果
204     u_new = u + (k1_u + 2*k2_u + 2*k3_u + k4_u) / 6.0
205     v_new = v + (k1_v + 2*k2_v + 2*k3_v + k4_v) / 6.0
206     z_new = z + (k1_z + 2*k2_z + 2*k3_z + k4_z) / 6.0
207
208     return u_new, v_new, z_new
209
210 def ssbp(a, s, m, n):
211     """空间平滑"""
212     w = a.copy()
213     m1, n1 = m-1, n-1
214     for i in range(1,m1):
215         for j in range(1,n1):
216             w[i,j] = (a[i,j]
217                       +0.5*s*(1-s)*(a[i-1,j]+a[i+1,j]+a[i,j-1]+a
218                                     [i,j+1]-4*a[i,j])
219                       +0.25*s*s*(a[i-1,j-1]+a[i-1,j+1]+a[i+1,j
220                                     -1]+a[i+1,j+1]-4*a[i,j]))
221
222     return w
223
224 def time_smooth(za, zb, zc, s=0.5):
225     """时间平滑"""
226     zb2 = zb.copy()
227     zb2[1:-1,1:-1] = zb[1:-1,1:-1] + s*(za[1:-1,1:-1] + zc
228       [1:-1,1:-1] - 2*zb[1:-1,1:-1])/2
229
230     return zb2
231
232 def plot_field(ax, data, lmda, phai, label):
233     """绘制等高线图"""
234     ax.set_extent([60,180,20,80], crs=ccrs.PlateCarree())
235     ax.coastlines('50m', linewidth=0.8)
236     gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True,
237                      linestyle='--', linewidth=0.5, color='gray',
238                      ')
239     gl.top_labels = False; gl.right_labels = False
240     gl.xlocator = plt.FixedLocator(np.arange(60,181,30))
241     gl.ylocator = plt.FixedLocator(np.arange(20,81,10))
242     levels = np.arange(5000,5751,150)
243     cs = ax.contour(lmda, phai, data, levels=levels,
244                    colors='black', linewidths=1.2,
245                    transform=ccrs.PlateCarree())
246     ax.clabel(cs, fmt='%d', inline=True, fontsize=8)
247     ax.set_title(label, fontsize=12, pad=10)

```

```

242
243 def main():
244     """主函数 - 使用RK4积分方案的正压原始方程模式"""
245     # 读取数据
246     nc = 'geo_197901.nc'
247     if not os.path.isfile(nc):
248         raise FileNotFoundError(f"找不到数据文件: {nc}")
249
250     ds = xr.open_dataset(nc, decode_times=False)
251     if 'pressure_level' in ds.dims:
252         ds = ds.sel(pressure_level=500)
253     z500 = ds['z']/9.8
254     u500 = ds['u']; v500 = ds['v']
255     lon = ds.longitude.values; lat = ds.latitude.values
256
257     # 初始场 t=0
258     nt0 = 0
259     u0 = u500.isel(valid_time=nt0).values
260     v0 = v500.isel(valid_time=nt0).values
261     z0 = z500.isel(valid_time=nt0).values
262
263     # 模式参数
264     m, n = 41, 17 # 网格点数
265     d, clat, clon = 300000.0, 45.0, 120.0 # 网格距、中心纬度、
        中心经度
266     dt, zo, s = 150.0, 0.0, 0.5 # 时间步长、参考高度、平滑系数
267
268     print("正在初始化模式...")
269
270     # 投影坐标变换和静力初始化
271     rm, f, lmnda, phai = cmf(d, clat, clon, m, n)
272     ua, va, za = interp_proj_grid(u0, v0, z0, lmnda, phai, m, n,
        lon, lat)
273     ua, va = cgw(za, rm, f, d, m, n) # 地转风初始化
274     za0 = ssbp(za.copy(), s, m, n) # 初始场空间平滑
275
276     # 边界条件初始化
277     ub, vb, zb = tbv(ua, va, za, m, n)
278
279     print("开始RK4积分...")
280
281     # 使用RK4进行时间积分 (6步 = 12小时)
282     n_steps = 6

```

```

283     zb_mid = None
284
285     for k in tqdm(range(n_steps), desc="RK4积分进度"):
286         # 使用RK4方案进行一个时间步的积分
287         ub_new, vb_new, zb_new = runge_kutta_4(ub, vb, zb, rm, f
288             , d, dt, zo, m, n)
289
290         # 更新变量
291         ub[1:-1, 1:-1] = ub_new[1:-1, 1:-1]
292         vb[1:-1, 1:-1] = vb_new[1:-1, 1:-1]
293         zb[1:-1, 1:-1] = zb_new[1:-1, 1:-1]
294
295         # 边界条件处理
296         ub, vb, zb = tbv(ub, vb, zb, m, n)
297
298         # 空间平滑
299         ub = ssbp(ub, s, m, n)
300         vb = ssbp(vb, s, m, n)
301         zb = ssbp(zb, s, m, n)
302
303         # 记录中间时刻的场
304         if k == 2: # 第3步, 即6小时
305             zb_mid = zb.copy()
306
307         za12 = zb # 12小时预报场
308
309         print("进行时间和空间平滑...")
310
311         # 时间平滑 + 再次空间平滑
312         zb_ts = time_smooth(za0, zb_mid, za12, s)
313         zb_ts = ssbp(zb_ts, s, m, n)
314
315         print("绘制结果...")
316
317         # 绘制对比图
318         fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6),
319             subplot_kw={'projection': ccrs.LambertConformal(
320                 central_longitude=clon, central_latitude=clat,
321                 standard_parallels=(30, 60))})
322
323         plot_field(ax1, za0, lmda, phai, '(a) 初始场 (RK4)')
324         plot_field(ax2, zb_ts, lmda, phai, '(b) RK4积分12h预报场')

```



```

325     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)
326     plt.tight_layout()
327     plt.show()
328
329     print("RK4积分完成！")
330
331     # 输出一些统计信息
332     print(f"初始场最大值: {np.max(za0):.2f} m")
333     print(f"初始场最小值: {np.min(za0):.2f} m")
334     print(f"12h预报场最大值: {np.max(zb_ts):.2f} m")
335     print(f"12h预报场最小值: {np.min(zb_ts):.2f} m")
336     print(f"最大变化: {np.max(np.abs(zb_ts - za0)):.2f} m")
337
338 if __name__ == '__main__':
339     main()

```

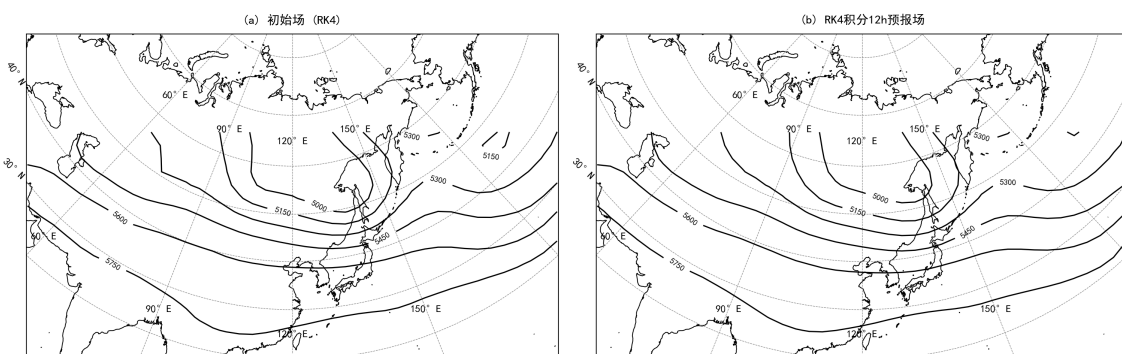


Figure 2: σ 习题一原始场与预报场对比

习题一参考了 BPDM 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。