

**数值天气预报实习**

|  |  |
| --- | --- |
| 姓名 | 戴泽凯 |
| 学号 | 201983300439 |
| 班级 | 19大物1班 |
| 指导教师 | 宋耀明 |

**二Ｏ二二年5月**

南京信息工程大学 实验（实习）报告

实验（实习）名称 正压原始方程模式实习 实验（实习）日期 5.19 得分   
指导教师 宋耀明

系 大气物理 专业 大气科学(大气物理） 年级 2019 班次 1 姓名 戴泽凯   
学号 201983300439

1. 实验目的

通过正压原始方程模式的实习，加深理解该章所学的基本内容；掌握当今制作数值天气预报的一般方法和主要步骤；并在数值计算、编制程序和上机操作等方面得到进一步训练。

1. 实习要求

以1973年4月29日08时（北京时）我国东北、华北地区500hPa等压面位势高度场及地转风场作为初值，采用固定的水平侧边界条件，应用正压原始方程二次守恒平流格式的模式，制作未来24小时有限区域500hPa位势高度场和风场的预报，并写一份实习报告。

1. 实验内容

1.模式的性能介绍

正压原始方程模式描述的大气，既有缓慢移动的大气长波，又包含了快速移动的重力惯性波，因此，模式可以模拟出准地转演变和地转适应过程，比准地转模式能够更近似地描述实际大气中的物理过程。

2.模式的基本假定、基本方程、技术方法、计算流程等的介绍

（1）大气为正压状态：水平风速不随气压变化；

（2）大气为均匀不可压缩的流体：密度为一常数；

（3）大气处于静力平衡：水平气压梯度不随高度变化；

（4）大气上界为自由面：有水平辐合辐散；

（5）不考虑摩擦及非绝热作用。

3.给出计算地转风以及五点平滑的子程序（完整代码见附录）

|  |  |
| --- | --- |
| 地转风初值 | geo.py |
| def geo\_init(maps,f,path,dx,dy,outpath,flag=2):    #计算地转风初值      """计算地转风初值      maps :放大系数      f:科氏参数      path : 原始数据路径      dx,dy :分辨率 unit['m']      gradient ：函数,差分代替微分      flag : 差分格式 1:后差 2:中央差      """      #############读文件##############################################      geo\_meta = read\_meta(path)              #测试数据 n\*m 16\*20      #############计算初值############################################      if flag == 2: #中央差          ua\_0 = -maps\*g/f\*np.gradient(geo\_meta,dy,axis=0,edge\_order=1)          va\_0 = maps\*g/f\*np.gradient(geo\_meta,dx,axis=1,edge\_order=1) #axis=1,dz/dx      elif flag == 1: #前差          df\_dy\_end = geo\_meta[-1,:] - geo\_meta[-2,:] # 1\*20          df\_dy\_0 =  geo\_meta[1:,:] - geo\_meta[:-1,:] # 15\*20          df\_dy = np.vstack((df\_dy\_0,df\_dy\_end))/dx          df\_dx\_end = geo\_meta[:,-1] - geo\_meta[:,-2] # 16\*1          df\_dx\_0  = geo\_meta[:,1:] - geo\_meta[:,:-1] # 16\*19          df\_dx = np.hstack((df\_dx\_0,df\_dx\_end.reshape(-1,1)))/dx          ###########################################################          ua\_0 = -maps\*g/f\*df\_dy          va\_0 =  maps\*g/f\*df\_dx      #########存文件#############################################      data\_save(ua\_0,'ua\_0',outpath)      data\_save(va\_0,'va\_0',outpath)      return geo\_meta,ua\_0,va\_0 | |
| 五点平滑 | smooth.py |
| def N\_point\_smooth(F,N=5,flag = 0):    #空间平滑，这里先写五点平滑      """空间平滑：用来滤去短波，抑制非线性不稳定。      Args:          F (numpy.array): 第n时间层          N (int, optional): N点平滑，默认为5          flag(int) :0为只做正平滑      Returns:          F\_new(numpy.array):平滑后的数组      """      F\_new = F.copy()      if N == 5:          S = 0.5          F[1:-1,1:-1] = F[1:-1,1:-1] + S/4.0\*(F[2:,1:-1]+F[:-2,1:-1]+F[1:-1,2:]+F[1:-1,:-2]-4\*F[1:-1,1:-1])          if flag == 1: #做正逆平滑              S = - 0.5              F\_new[1:-1,1:-1] = F[1:-1,1:-1] + S/4.0\*(F[2:,1:-1]+F[:-2,1:-1]+F[1:-1,2:]+F[1:-1,:-2]-4\*F[1:-1,1:-1])          else :              F\_new = F      elif  N==9: #内边界平滑          S = 0.5          F\_new[[1,-2],1:-1] = F[[1,-2],1:-1] + S/2\*(1-S)\*(F[[1,-2],2:] + F[[2,-1],1:-1] + F[[1,-2],0:-2] + F[[0,-3],1:-1] \                                  - 4\*F[[1,-2],1:-1]) + S\*\*2/2\*(F[[2,-1],2:]+F[[2,-1],0:-2]+F[[0,-3],0:-2]+F[[0,-3],2:]-4\*F[[1,-2],1:-1])            F\_new[2:-2,[1,-2]] = F[2:-2,[1,-2]] + S/2\*(1-S)\*(F[2:-2,[2,-1]] + F[3:-1,[1,-2]] + F[2:-2,[0,-1]] + F[1:-3,[1,-2]] \                                  -4\*F[2:-2,[1,-2]]) + S\*\*2/2\*(F[3:-1,[2,-1]]+F[3:-1,[0,-1]]+F[1:-3,[0,-3]]+F[1:-3,[2,-1]]-4\*F[2:-2,[1,-2]])      return F\_new | |

4.试验结果及其分析

**（1）正平滑与正逆平滑**

|  |  |
| --- | --- |
|  |  |
|  |  |

从图上可以发现：都模拟出来槽脊的东移，强度减弱，气旋随时间逐渐衰弱（525dagpm->530dagpm）。只做正平滑的天气系统演变较快，可以看到，48h的正逆平滑的位势高度图和24只做正平滑的相近，表明只做正平滑的大气长波能量频散较快，正逆平滑的预报场对长波衰减较小。

**（2）地转风子程序中不同差分格式的数值试验；**

|  |  |
| --- | --- |
|  |  |

由图可知，由中央差格式与前差格式得出的初始风场作出的预报场差距并不明显。

**(3)是否做边界平滑和内点平滑的数值试验**

|  |  |
| --- | --- |
|  |  |
|  |  |

从图上可以发现：完全不做空间平滑对预报场影响十分剧烈，模式预报是出现大量频率短波。空间平滑分为边界9点平滑和内点5点平滑，内点平滑能滤掉大量短波，保证模式积分稳定。边界平滑同样也能抑制能量频散，但相比内点平滑影响较小。

**4）是否做时间平滑的数值试验。**

|  |  |
| --- | --- |
|  |  |
|  |  |

由上图可知，由有无时间平滑对24h乃至48h预报场差距并不明显。

四、小结与讨论

1. 只做正平滑，对长波衰减较大，导致槽脊发展和衰减较快。
2. 从前差、中央差的实验结果来看，初始场差分格式选取对预报结果的影响不大。
3. 边界平滑对正压原始方程模式的数值预报结果影响较大。内点平滑能滤掉大量短波，保证模式积分稳定。
4. 不做时间平滑的影响在短时间内很小。

五、所有程序附录

|  |  |
| --- | --- |
| Const.py | 储存常数项 |
| f0 = 7.292e-5  # 科里奥利力常数 [unit:'s^{-1}]  le = 11142.37e3  # 兰伯特投影面赤道到北极点的距离 [unit:'m']  k = 0.7156  # 兰伯特投影的放大系数  a = 6371e3  # 地球半径                        [unit:'m']  g = 9.81    # 重力加速度                       [unit:'m/s^2'] | |

|  |  |
| --- | --- |
| Fileio.py | 打开和储存文件 |
| import numpy as np  import os  import xarray as xr  def data\_save(u,filename,outpath:str,flag = 1):      """保存数据      Args:          u (numpy.array): 储存的文件          flag (int, [1-5]): 想要储存文件的格式 Defaults to 1.          flag = 1 : '.npy'          flag = 2 : '.txt'          flag = 3 : '.dat'          flag = 4 : '.nc'          flag = 5 : '.grib2'      """      os.chdir(outpath)      if flag == 1:          np.save(filename,u)  def read\_meta(path = r'C:\Users\dzk\Desktop\Barotropic\_primitive\_equation\_model\Input\za.dat'):      data = np.fromfile(path,sep = ' ')      return data.reshape((16,20),order='c') | |

|  |  |
| --- | --- |
| geo.py | 计算地图放大因子和地转风初值 |
| import itertools  import numpy as np  from Model.file\_io import data\_save  from Model.file\_io import read\_meta  from Model.const import le,f0,k,a,g  # np.random.seed(99) 测试数据  def cmf(theta:float,grids:list,dx:float,outpath): #计算地图放大系数和科氏参数      """计算地图放大系数和科氏参数      Args:          theta (float): 研究区域中心纬度          grids (list): 网格点数，第一个参数为纬向          dx (float)  : 网格距离,经纬方向一致 [unit:'m']          outpath (str): 保存计算的放大系数和科氏参数      Returns:          rm,f (tuple): 前一个为放大系数，后一个为科氏参数      """        ###############计算模式中心到北极的距离rl########################      theta = theta\*np.pi/180      th = 30\*np.pi/180      t = np.tan(theta/2)/np.tan(th/2)      rcent = a\*np.sin(th)/k\*t\*\*k      n,m = grids      #########确定网格坐标原点在地图坐标系中的位置#####################      xi0 = -(m-1)/2.0      yj0 = -(rcent / dx + (n-1)/2.0)      rm = np.empty((n,m))      f = np.empty((n,m))      ##求各格点至北极点的距离rl,(xj,yi)为模式各格点在地图坐标系中的位置#      for i, j in itertools.product(range(m), range(n)):          xi = xi0+i          yj = yj0+j          rl = dx\*np.sqrt(xi\*\*2+yj\*\*2)      ########## 求放大系数rm和柯氏参数f##############################          sin\_xy = (le\*\*(2.0/k)-rl\*\*(2.0/k))/(le\*\*(2.0/k)+rl\*\*(2.0/k))          cos\_xy = np.sqrt(1-sin\_xy\*\*2)          rm[j,i] = k\*rl/(a\*cos\_xy)          f[j,i] = 2\*f0\*sin\_xy        #################保存数据######################################      data\_save(rm,'maps\_Factor',outpath)      data\_save(f,'Coriolis',outpath)      return rm,f  def geo\_init(maps,f,path,dx,dy,outpath,flag=2):    #计算地转风初值      """计算地转风初值      maps :放大系数      f:科氏参数      path : 原始数据路径      dx,dy :分辨率 unit['m']      gradient ：函数,差分代替微分      flag : 差分格式 1:后差 2:中央差      """      #############读文件##############################################      geo\_meta = read\_meta(path)              #测试数据 n\*m 16\*20      #############计算初值############################################      if flag == 2: #中央差          ua\_0 = -maps\*g/f\*np.gradient(geo\_meta,dy,axis=0,edge\_order=1)          va\_0 = maps\*g/f\*np.gradient(geo\_meta,dx,axis=1,edge\_order=1) #axis=1,dz/dx      elif flag == 1: #前差          df\_dy\_end = geo\_meta[-1,:] - geo\_meta[-2,:] # 1\*20          df\_dy\_0 =  geo\_meta[1:,:] - geo\_meta[:-1,:] # 15\*20          df\_dy = np.vstack((df\_dy\_0,df\_dy\_end))/dx          df\_dx\_end = geo\_meta[:,-1] - geo\_meta[:,-2] # 16\*1          df\_dx\_0  = geo\_meta[:,1:] - geo\_meta[:,:-1] # 16\*19          df\_dx = np.hstack((df\_dx\_0,df\_dx\_end.reshape(-1,1)))/dx          ###########################################################          ua\_0 = -maps\*g/f\*df\_dy          va\_0 =  maps\*g/f\*df\_dx      #############存文件#############################################      data\_save(ua\_0,'ua\_0',outpath)      data\_save(va\_0,'va\_0',outpath)      return geo\_meta,ua\_0,va\_0 | |

|  |  |
| --- | --- |
| Smooth.py | 平滑算子 |
| def N\_point\_smooth(F,N=5,flag = 0):    #空间平滑，这里先写五点平滑      """空间平滑：用来滤去短波，抑制非线性不稳定。      Args:          F (numpy.array): 第n时间层          N (int, optional): N点平滑，默认为5          flag(int) :0为只做正平滑      Returns:          F\_new(numpy.array):平滑后的数组      """      F\_new = F.copy()      if N == 5:          S = 0.5          F[1:-1,1:-1] = F[1:-1,1:-1] + S/4.0\*(F[2:,1:-1]+F[:-2,1:-1]+F[1:-1,2:]+F[1:-1,:-2]-4\*F[1:-1,1:-1])          if flag == 1: #做正逆平滑              S = - 0.5              F\_new[1:-1,1:-1] = F[1:-1,1:-1] + S/4.0\*(F[2:,1:-1]+F[:-2,1:-1]+F[1:-1,2:]+F[1:-1,:-2]-4\*F[1:-1,1:-1])          else :              F\_new = F      elif  N==9: #内边界平滑          S = 0.5          F\_new[[1,-2],1:-1] = F[[1,-2],1:-1] + S/2\*(1-S)\*(F[[1,-2],2:] + F[[2,-1],1:-1] + F[[1,-2],0:-2] + F[[0,-3],1:-1] \                                  - 4\*F[[1,-2],1:-1]) + S\*\*2/2\*(F[[2,-1],2:]+F[[2,-1],0:-2]+F[[0,-3],0:-2]+F[[0,-3],2:]-4\*F[[1,-2],1:-1])            F\_new[2:-2,[1,-2]] = F[2:-2,[1,-2]] + S/2\*(1-S)\*(F[2:-2,[2,-1]] + F[3:-1,[1,-2]] + F[2:-2,[0,-1]] + F[1:-3,[1,-2]] \                                  -4\*F[2:-2,[1,-2]]) + S\*\*2/2\*(F[3:-1,[2,-1]]+F[3:-1,[0,-1]]+F[1:-3,[0,-3]]+F[1:-3,[2,-1]]-4\*F[2:-2,[1,-2]])      return F\_new  def time\_smooth(F,flag = 0):    #时间平滑      """时间平滑(3时间层)：用来阻尼高频振荡，抑制解的增长      Args:          F (3\*n\*m,):第一维为三个时间层，第二维度为纬度，第三为经度. 即：(t,y,x)          flag(int) :0为只做正平滑      Returns:          F\_new(numpy.array):平滑后的数组      """      S = 0.5      F[1,1:-1,1:-1] = (1-S)\*F[1,1:-1,1:-1] + S/2\*( F[2,1:-1,1:-1] + F[0,1:-1,1:-1] )      if flag != 1:          return F[1,1:-1,1:-1]      S = -0.5      return (1-S)\*F[1,1:-1,1:-1] + S/2\*( F[2,1:-1,1:-1] + F[0,1:-1,1:-1] ) | |

|  |  |
| --- | --- |
| Space\_diff.py | 空间积分格式 |
| import numpy as np  from Model.const import g  def L\_diff(u,v,f,dx,dy):  #空间差分算子L      """空间差分算子      math : \left [ \overline{\bar{u}^{x}()\_{x}}^{x} + \overline{\bar{v}^{y}( )\_{y} }^{y}  \right ]      Args:          u (numpy.array): 纬向风场          v (numpy.array): 经向风场          f (numpy.array): 某个矢量场          dx (float): x方向分辨率          dy (float): y方向分辨率      Returns:          Lu+Lv (numpy.array): 算子      """      Lu = u.copy()      Lv = v.copy()      Lu[1:-1,1:-1] = 1./(4.0\*dx) \* ( (u[1:-1,2:] + u[1:-1,1:-1])\*( f[1:-1,2:] - f[1:-1,1:-1])+( u[1:-1,:-2] + u[1:-1,1:-1])\*(f[1:-1,1:-1] - f[1:-1,:-2]) )      Lv[1:-1,1:-1] = 1./(4.0\*dy) \* ( (v[2:,1:-1] + v[1:-1,1:-1])\*( f[2:,1:-1] - f[1:-1,1:-1])+( v[:-2,1:-1] + v[1:-1,1:-1])\*(f[1:-1,1:-1] - f[:-2,1:-1]) )      return (Lu+Lv)[1:-1,1:-1]  def f\_plus(maps,f,u,v,dx,dy): #空间差分算子f\*      """变换后的科氏算子      math : f\_{i,j} = u\_{i,j}\overline{m\_{x}}^{x} - v\_{i,j}\overline{m\_{y}}^{y}      Args:          maps (array): 放大系数          f (array): 科氏参数          u (numpy.array): 纬向风场          v (numpy.array): 经向风场          dx (float): x方向分辨率          dy (float): y方向分辨率      Returns:          \_type\_: 变换后的科氏算子      """      m\_y,m\_x = np.gradient(maps,dy,axis=0)[1:-1,1:-1],np.gradient(maps,dx,axis=1)[1:-1,1:-1]      return (f[1:-1,1:-1] + u[1:-1,1:-1]\*m\_y - m\_x\*v[1:-1,1:-1] )  def adv(uu,vv,dx,dy): #空间差分算子      """空间差分平流算子      math : \overline{()\_{x}}^{x}+\overline{()\_{y}}^{y}      Args:          uu (numpy.array): 纬向风场          vv (numpy.array): 经向风场          dx (float): x方向分辨率          dy (float): y方向分辨率      Returns:          \_type\_: 算子      """      return (np.gradient(uu,dx,axis=1)[1:-1,1:-1]+np.gradient(vv,dy,axis=0)[1:-1,1:-1])  def advection\_equation\_2(maps,f,dx,dy,uu,vv,hh):      """二次守恒平流格式      Args:          maps (array): 放大系数          f (array): 科氏参数          u (numpy.array): 纬向风场          v (numpy.array): 经向风场          dx (float): x方向分辨率          dy (float): y方向分辨率          hh (\_type\_): 高度场      Returns:          E,G,H (tuple): 趋势场      """      ############################  du/dt   ##################################################      E = - maps[1:-1,1:-1]\*( L\_diff(uu,vv,uu,dx,dy)+g\*(np.gradient(hh,dx,axis=1)[1:-1,1:-1]) )+ f\_plus(maps,f,uu,vv,dx,dy)\*vv[1:-1,1:-1]        #############################  dv/vt  ##################################################        G = - maps[1:-1,1:-1]\*( L\_diff(uu,vv,vv,dx,dy) + g\*(np.gradient(hh,dy,axis=0)[1:-1,1:-1]) )-f\_plus(maps,f,uu,vv,dx,dy)\*uu[1:-1,1:-1]        #############################  dh/dt  ##################################################      H = - maps[1:-1,1:-1]\*\*2\*(L\_diff(uu,vv,hh/maps,dx,dy)+hh[1:-1,1:-1]/maps[1:-1,1:-1]\*adv(uu,vv,dx,dy))        return E,G,H | |

|  |  |
| --- | --- |
| Time\_diff.py | 时间积分格式 |
| from Model.space\_diff import advection\_equation\_2  def Eula\_past\_diff(u,v,h,map,f,dx,dy,dt):  #欧拉前差      """\_summary\_      Args:          u (\_type\_): \_description\_          v (\_type\_): \_description\_          h (\_type\_): \_description\_          map (\_type\_): \_description\_          f (\_type\_): \_description\_          dx (\_type\_): \_description\_          dy (\_type\_): \_description\_          dt (\_type\_): \_description\_      Returns:          \_type\_: \_description\_      """      E,G,H = advection\_equation\_2(map,f,dx,dy,u[1,...],v[1,...],h[1,...])# n时刻,注意(m-2)\*(n-2)      ######################u场更新 u[0,1:-1,1:-1] 作为中转层################      u[0,1:-1,1:-1] = u[1,1:-1,1:-1] + dt \* E      v[0,1:-1,1:-1] = v[1,1:-1,1:-1] + dt \* G      h[0,1:-1,1:-1] = h[1,1:-1,1:-1] + dt \* H      #######################################################################      E,G,H = advection\_equation\_2(map,f,dx,dy,u[0,...],v[0,...],h[0,...])      #######################################################################      u[-1,1:-1,1:-1] = u[1,1:-1,1:-1] + dt \* E      ###################### v场更新##########################################      v[-1,1:-1,1:-1] = v[1,1:-1,1:-1] + dt \* G      ###################### h场更新##########################################      h[-1,1:-1,1:-1] = h[1,1:-1,1:-1] + dt \* H      ########################################################################      u[1,...]= u[-1,...]      v[1,...]= v[-1,...]      h[1,...]= h[-1,...]        return u,v,h    def Central\_diff(u,v,h,map,f,dx,dy,dt):    # 时间中央差分      """时间中央差      Args:          u (\_type\_): \_description\_          v (\_type\_): \_description\_          h (\_type\_): \_description\_          map (\_type\_): \_description\_          f (\_type\_): \_description\_          dx (\_type\_): \_description\_          dy (\_type\_): \_description\_          dt (\_type\_): \_description\_      Returns:          \_type\_: \_description\_      """      ########################################################################      E,G,H = advection\_equation\_2(map,f,dx,dy,u[1,...],v[1,...],h[1,...])      ########################################################################      u[-1,1:-1,1:-1] = u[0,1:-1,1:-1] + 2.\* dt \* E      v[-1,1:-1,1:-1] = v[0,1:-1,1:-1] + 2.\* dt \* G      h[-1,1:-1,1:-1] = h[0,1:-1,1:-1] + 2.\* dt \* H      ########################################################################      u[:-1,...]= u[1:,...]      v[:-1,...]= v[1:,...]      h[:-1,...]= h[1:,...]      ########################################################################      return u,v,h  def three\_step(u,v,h,map,f,dx,dy,dt):   # 三步法起步预报      """\_summary\_      Args:          u (\_type\_): \_description\_          v (\_type\_): \_description\_          h (\_type\_): \_description\_          map (\_type\_): \_description\_          f (\_type\_): \_description\_          dx (\_type\_): \_description\_          dy (\_type\_): \_description\_          dt (\_type\_): \_description\_      Returns:          \_type\_: \_description\_      """        ##########半步前差#######################################################      # 得到边界      u\_1\_2 = u[1,...].copy()      v\_1\_2 = v[1,...].copy()      h\_1\_2 = h[1,...].copy()      ########################################################################      E,G,H = advection\_equation\_2(map,f,dx,dy,u[1,...],v[1,...],h[1,...])# 注意(m-2)\*(n-2)      ########################################################################      u\_1\_2[1:-1,1:-1] = u[1,1:-1,1:-1] +  dt \* E /2.0      v\_1\_2[1:-1,1:-1] = v[1,1:-1,1:-1] +  dt \* G /2.0      h\_1\_2[1:-1,1:-1] = h[1,1:-1,1:-1] +  dt \* H /2.0      ##########半步中央差######################################################      u\_1 = u[1,...].copy()      v\_1 = v[1,...].copy()      h\_1 = h[1,...].copy()      ########################################################################      E,G,H = advection\_equation\_2(map,f,dx,dy,u\_1\_2,v\_1\_2,h\_1\_2)      ########################################################################      u\_1[1:-1,1:-1] = u[1,1:-1,1:-1] +  dt \* E      v\_1[1:-1,1:-1] = v[1,1:-1,1:-1] +  dt \* G      h\_1[1:-1,1:-1] = h[1,1:-1,1:-1] +  dt \* H      ############ 存数据######################################################      u[0,1:-1,1:-1] = u\_1[1:-1,1:-1]      v[0,1:-1,1:-1] = v\_1[1:-1,1:-1]      h[0,1:-1,1:-1] = h\_1[1:-1,1:-1]      #############中央差分#####################################################      E,G,H = advection\_equation\_2(map,f,dx,dy,u\_1 ,v\_1 ,h\_1)      u[-1,1:-1,1:-1] = u[1,1:-1,1:-1] +  dt \* E \* 2.0      v[-1,1:-1,1:-1] = v[1,1:-1,1:-1] +  dt \* G \* 2.0      h[-1,1:-1,1:-1] = h[1,1:-1,1:-1] +  dt \* H \* 2.0      ########################################################################      u[1,...]= u[-1,...]      v[1,...]= v[-1,...]      h[1,...]= h[-1,...]      ########################################################################      return u,v,h | |

|  |  |
| --- | --- |
| Modelrun.py | 模式运行控制函数，注意！！！做敏感性实验要适当修改，注释该函数 |
| import numpy as np  from Model.time\_diff import \*  from Model.smooth import \*  from Model.geo import \*  from Model.file\_io import data\_save  def run():  # sourcery skip: for-index-underscore      """运行文件      """      #### 读取namelist，获得程序需要的参数      dt = 600 # unit['secend'] time\_step      ref\_lon = 51      grids = [16,20]      path = r'C:\Users\dzk\Desktop\实习报告\Barotropic\_primitive\_equation\_model\Input\za.dat'      outpath = r'C:\Users\dzk\Desktop\实习报告\Barotropic\_primitive\_equation\_model\Output'      dx = dy = 3e5      t\_sum = 48  #模拟时长 unit=['hour]      #######################################################################      time\_control = np.array([1,6,12,24],dtype=np.float32)\*6  # 时间控制参数 ，[unit = 'hour']这里规定 6h 做一次时间平滑，12h做空间平滑，24h结束积分      time\_num = int(time\_control[-1]\*3600/dt)    # 需要积分的时间步数      #########计算地图放大系数和科氏参数######################################      map,f = cmf(ref\_lon,grids,dx,outpath)        ##########初始化风场###################################################      h\_0,ua\_0,va\_0 = geo\_init(map,f,path,dx,dy,outpath,flag=2)        ###########初始化#####################################################      #######这里将数组设置为(时间，经度，纬度)三维数据 (3,n,m)################      #######这里直接默认赋固定边界值，因为后续操作都是内点(u[1:-1,1:-1])操作###      #####################################################################      u = np.tile(ua\_0,(3,1,1))      v = np.tile(va\_0,(3,1,1))      h = np.tile(h\_0,(3,1,1))      print('----------初始化成功------------\n')      ##########积分#######################################################      Na = 0      while Na < t\_sum/12 :          for i in range(int(time\_control[-2])):              #######前一小时 欧拉-后差#######################################              if i < time\_control[0]:                  u,v,h=Eula\_past\_diff(u,v,h,map,f,dx,dy,dt)              elif i == time\_control[0] :                  u,v,h= three\_step(u,v,h,map,f,dx,dy,dt)                  i = i+1  #总的是消耗两步              else :                  u,v,h=Central\_diff(u,v,h,map,f,dx,dy,dt)              if not ((i+1) % 6):                  u[1,...]=N\_point\_smooth(u[1,...],N = 5,flag = 1)                  v[1,...]=N\_point\_smooth(v[1,...],N = 5,flag = 1)                  h[1,...]=N\_point\_smooth(h[1,...],N = 5,flag = 1)                  print('--------------------已积分到第{0:f}小时---------------------------'.format((i+1)/6))              if i\*dt == 6\*3600 :                  pass                  # #############时间平滑#######################                  # u[1,1:-1,1:-1]=time\_smooth(u,flag = 1)                  # v[1,1:-1,1:-1]=time\_smooth(v,flag = 1)                  # h[1,1:-1,1:-1]=time\_smooth(h,flag = 1)      #############12h空间平滑################################################          u[1,...]=N\_point\_smooth(u[1,...],N = 9)          v[1,...]=N\_point\_smooth(v[1,...],N = 9)          h[1,...]=N\_point\_smooth(h[1,...],N = 9)          Na +=1      #############保存文件###################################################      data\_save(u,'u',outpath)      data\_save(v,'v',outpath)      data\_save(h,'h',outpath)      #############预报结束################################################### | |

|  |  |
| --- | --- |
| Main.py | 主程序 |
| import numpy as np  from Model.modelrun import run  #################################  print('------正压原始方程模式预报----------')  run()  print('------------预报成功！------------')  ############################################## | |

|  |  |
| --- | --- |
| Draw.py | 画图代码 |
| from turtle import position  import numpy as np  import matplotlib.pyplot as plt  import cartopy.crs  as ccrs  import cartopy.feature as cfeature  #########需要修改的量################################  title = '有正逆平滑'#'有平滑'  ###################################################  hgt = np.load('Output\h.npy')[1,...]  u\_wind= np.load('Output\\u.npy')[1,...]  v\_wind= np.load('Output\\v.npy')[1,...]  ###初始场路径##########################  # path = r'C:\Users\dzk\Desktop\实习报告\Barotropic\_primitive\_equation\_model\Input'  # hgt = np.fromfile(path+'\za.dat',sep = ' ').reshape((16,20),order='c')  # u\_wind= np.fromfile(path+'\\ua.dat',sep = ' ').reshape((16,20),order='c')  # v\_wind= np.fromfile(path+'\\va.dat',sep = ' ').reshape((16,20),order='c')  lon,lat = np.meshgrid(np.arange(85,155,3.5),np.arange(32.5,72.5,2.5)  fig = plt.figure(figsize=[8,6])  maps=ccrs.PlateCarree(central\_longitude=105)  ax = fig.add\_subplot(1,1,1, projection=maps)  plt.tick\_params(labelsize=12)   #设置坐标轴字体大小  plt.rcParams['font.sans-serif']='SimHei'    #设置默认字体  ax.set\_extent([85, 151,32.5, 67])  sh=ax.contour(lon,lat,hgt,np.arange(5100.,5900.,50.),colors='k',extend='both',transform=ccrs.PlateCarree())#cmap = 'RdBu\_r')  sf=ax.contourf(sh,transform=ccrs.PlateCarree(),cmap = 'RdBu\_r' )  ax.quiver(lon[1:-1,1:-1],lat[1:-1,1:-1],u\_wind[1:-1,1:-1],v\_wind[1:-1,1:-1],transform=ccrs.PlateCarree())  plt.clabel(sh, inline=True, fontsize=10)  ax.add\_feature(cfeature.COASTLINE,lw=0.5)  gl=ax.gridlines(draw\_labels=True, dms=True, x\_inline=False, y\_inline=False, color='gray', alpha=0.3, linestyle='--')  gl.top\_labels = False  gl.right\_labels = False  gl.rotate\_labels = False  plt.colorbar(sf,shrink=0.8)  ax.set\_title(title)  plt.savefig('Output\\'+title)  plt.show() | |