数值天气预报课程作业

吴浩杰 20221170212

1 实习二

兰勃特投影的地图放大系数和科里奥利参数

1.1 实习目的

通过编程计算，使学生掌握兰勃特投影的地图放大系数和科里奥利参数的计算方法。

1.2 实习要求

编写并提交计算兰勃特投影的地图放大系数和科里奥利参数的 MATLAB 程序。要求学生在机房现场操作，参考沈桐立等 (2015) 关于兰勃特投影相关内容，撰写实习报告，教师进行随堂讲解和指导。

1.3 实习内容

试编写一个计算兰勃特投影地图放大系数 RM(i，j) 和科里奥利参数 f(i，j) 的子程序，输入平面正方形网格点坐标 (i，j)，输出 RM(i，j) 和 f(i，j)。参考点为北极点 (0，0)。

Listing 1: 兰伯特投影地图放大系数

```
1
2     import numpy as np
3     import matplotlib.pyplot as plt
4     import pandas as pd
5
6     # 基本参数
7     EARTH_RADIUS = 6371  # 地球半径
8     OMEGA = 7.292e-5  # 地球自转角速度
9
10    def set_chinese_font():
```

```
11          """设置matplotlib中文字体"""
12          plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文
                字体为黑体
13          plt.rcParams['axes.unicode_minus'] = False  # 解决负号显
                示问题
14
15      def set_parameters():
16          """设置投影参数"""
17          # 网格参数
18          M = 6  # 东西扩展
19          N = 7  # 南北扩展
20          d = 100  # 格点距（km）
21
22          # 投影参数
23          phy0 = 30  # 中央纬度
24          seita1 = 30  # 第一标准纬度
25          seita2 = 60  # 第二标准纬度
26          a = EARTH_RADIUS  # 地球半径
27
28          return M, N, d, phy0, seita1, seita2, a
29
30      def calculate_magnification_factor_and_coriolis_parameter(
            proj, In, Jn, d):
31          """计算投影的放大系数和科氏参数
32
33          Args:
34              proj (str): 投影方式 ('lambert', 'mercator', '
                    stereographic')
35              In (float): 东西方向的网格索引
36              Jn (float): 南北方向的网格索引
37              d (float): 网格距离
38
39          Returns:
40              tuple: (m, f) 其中m为放大系数，f为科氏参数
41          """
42          if proj == 'lambert':
43              k = 0.7156
44              le = 11423.37
45              l = np.sqrt((In**2 + Jn**2) * d**2)
46              m = k * l / EARTH_RADIUS / np.sqrt(1 - ((le**(2/k) -
                        l**(2/k)) / (le**(2/k) + l**(2/k)))**2)
47              f = 2 * OMEGA * (le**(2/k) - l**(2/k)) / (le**(2/k)
                    + l**(2/k))
```

```
48
49          elif proj == 'mercator':
50              m = np.sqrt((EARTH_RADIUS * np.cos(np.deg2rad(22.5))
                    )**2 + (Jn*d)**2) / EARTH_RADIUS
51              f = 2 * OMEGA * np.sin(Jn * d / np.sqrt((
                    EARTH_RADIUS * np.cos(np.deg2rad(22.5)))**2 + (Jn
                    *d)**2))
52
53          elif proj == 'stereographic':
54              le = 11888.45
55              l = np.sqrt((In**2 + Jn**2) * d**2)
56              m = (2 + np.sqrt(3)) / 2 / (1 + ((le**2 - l**2) / (
                    le**2 + l**2)))
57              f = 2 * OMEGA * ((le**2 - l**2) / (le**2 + l**2))
58
59          else:
60              raise ValueError('投影方式输入错误！')
61
62          return m, f
63
64      def calculate_projection_factors(M, N, d, phy0, seita1,
            seita2, a):
65          """计算投影因子"""
66          # 计算圆锥常数
67          k = (np.log(np.sin(np.deg2rad(seita1))) - np.log(np.sin(
                np.deg2rad(seita2)))) / \
68              (np.log(np.tan(np.deg2rad(seita1/2))) - np.log(np.
                    tan(np.deg2rad(seita2/2))))
69
70          # 计算映像平面上赤道到北极点的距离
71          le = a * np.sin(np.deg2rad(seita1)) / k * (1/np.tan(np.
                deg2rad(seita1/2)))**k
72
73          # 计算参考距离
74          l_ref = le * (np.cos(np.deg2rad(phy0))/(1+np.sin(np.
                deg2rad(phy0))))**k
75
76          # 初始化放大系数矩阵
77          m = np.zeros((2 * N + 1, 2 * M + 1))
78
79          # 计算每个网格点的放大系数
80          for In in range(-M, M+1):
81              for Jn in range(-N, N+1):
```

```
                l = np.sqrt((abs(In) * d)**2 + (l_ref-Jn*d)**2)
                m[Jn+N, In+M] = k*l/(a*np.sqrt(1 - \
                    ((le**(2/k) - l**(2/k))/(le**(2/k) + l**(2/k
                        )))**2))

        return np.flipud(m)

    def calculate_grid_coordinates(M, N, d, phy0):
        """计算网格点的经纬度坐标"""
        # 计算经纬度范围
        lon_range = np.linspace(-M*d/111, M*d/111, 2*M+1)  # 将
            距离转换为经度（大约111km/度）
        lat_range = np.linspace(phy0-N*d/111, phy0+N*d/111, 2*N
            +1)  # 将距离转换为纬度

        # 生成网格点坐标
        lon_grid, lat_grid = np.meshgrid(lon_range, lat_range)
        return lon_range, lat_range, lon_grid, lat_grid

    def plot_magnification_factor(m):
        """绘制放大系数等值线图"""
        # 获取网格参数
        M, N, d, phy0, seita1, seita2, a = set_parameters()

        # 计算网格点坐标
        lon_range, lat_range, lon_grid, lat_grid =
            calculate_grid_coordinates(M, N, d, phy0)

        plt.figure(figsize=(12, 10))

        # 绘制等值线
        levels = np.linspace(np.min(m), np.max(m), 15)  # 增加等
            值线级别
        contour = plt.contour(lon_grid, lat_grid, m, levels=
            levels, colors='k')
        plt.clabel(contour, inline=True, fontsize=8)  # 添加等值
            线标签

        # 添加填充等值线
        contourf = plt.contourf(lon_grid, lat_grid, m, levels=
            levels, cmap='RdYlBu_r', alpha=0.6)
        plt.colorbar(contourf, label='放大系数')

```

```
117         # 设置网格和标签
118         plt.grid(True, linestyle='--', alpha=0.5)
119         plt.title('兰伯特投影放大系数分布', pad=15, fontsize=14)
120         plt.xlabel('经度 (°)', fontsize=12)
121         plt.ylabel('纬度 (°)', fontsize=12)
122 
123         # 调整显示范围，确保所有网格点可见
124         plt.margins(0.1)
125         plt.tight_layout()
126 
127         # 设置刻度格式
128         plt.gca().xaxis.set_major_formatter(plt.FuncFormatter(
129             lambda x, p: f'{x:.1f}°E' if x > 0 else f'{-x:.1f}°W
                ' if x < 0 else '0°'))
130         plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(
131             lambda x, p: f'{x:.1f}°N' if x > 0 else f'{-x:.1f}°S
                ' if x < 0 else '0°'))
132 
133         plt.show()
134 
135     def save_to_excel(lon_range, lat_range, m, f):
136         """保存结果到Excel文件"""
137         # 创建结果数据
138         data = []
139         for i in range(len(lon_range)):
140             for j in range(len(lat_range)):
141                 data.append({
142                     '经度': f'{lon_range[i]:.2f}°E',
143                     '纬度': f'{lat_range[j]:.2f}°N',
144                     '地图系数': f'{m[j,i]:.4f}',
145                     '科里奥利参数': f'{f[j,i]:.4e}'
146                 })
147 
148         # 创建DataFrame并保存
149         df = pd.DataFrame(data)
150         df.to_excel('实习内容_地图系数.xlsx', index=False)
151         df.to_excel('实习内容_科里奥利参数.xlsx', index=False)
152 
153     def main():
154         # 设置中文字体
155         set_chinese_font()
156 
157         # 设置参数
```

```
158         M, N, d, phy0, seita1, seita2, a = set_parameters()
159
160         # 计算放大系数
161         m = calculate_projection_factors(M, N, d, phy0, seita1,
                seita2, a)
162
163         # 计算经纬度范围（中心点为100°E）
164         lon_range = np.linspace(100-M*d/111, 100+M*d/111, 2*M+1)
                # 将距离转换为经度
165         lat_range = np.linspace(phy0-N*d/111, phy0+N*d/111, 2*N
                +1)  # 将距离转换为纬度
166
167         # 计算科里奥利参数
168         f = np.zeros_like(m)
169         for i in range(2*M+1):
170             for j in range(2*N+1):
171                 In = i - M
172                 Jn = N - j  # 注意这里需要翻转j以匹配m矩阵
173                 _, f[j,i] =
                        calculate_magnification_factor_and_coriolis_parameter
                        ('lambert', In, Jn, d)
174
175         # 保存结果到Excel
176         save_to_excel(lon_range, lat_range, m, f)
177
178         # 绘制等值线图
179         plot_magnification_factor(m)
180
181     if __name__ == '__main__':
182         main()
```

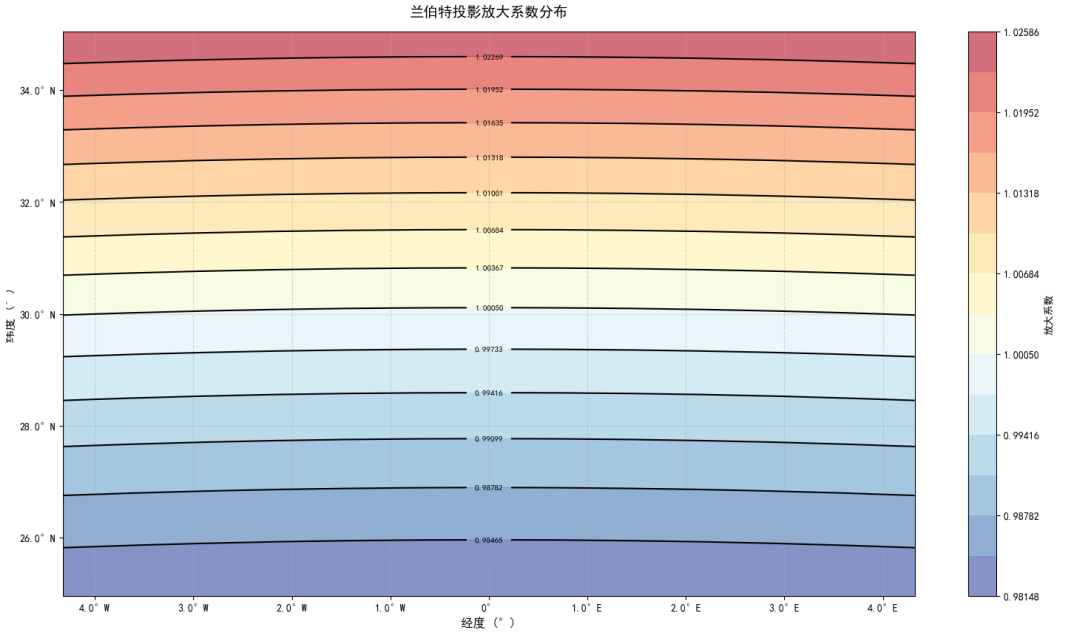

Figure 1: 中央经度 30 的兰伯特投影放大系数分布

2 习题

利用兰伯特投影，计算矩形网格上各点的地图放大系数和科里奥利参数。

2.1 习题一

在标准兰勃特投影图上，选取一矩形网格，矩形的 Y 轴与 100°E 平行，O 点为 (100°E，45°N)，网格距 80km，网格点向北 5 个，向东 6 个。计算网格上各点的地图放大系数和科里奥利参数。

Listing 2: 习题一

```
1
2     import numpy as np
3     import matplotlib.pyplot as plt
4     import pandas as pd
5
6     # 基本参数
7     EARTH_RADIUS = 6371  # 地球半径
8     OMEGA = 7.292e-5  # 地球自转角速度
9
10    def set_chinese_font():
11        """设置matplotlib中文字体"""
12        plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文
              字体为黑体
```

```
13          plt.rcParams['axes.unicode_minus'] = False  # 解决负号显
                示问题
14
15      def set_parameters():
16          """设置投影参数"""
17          # 网格参数
18          M = 6  # 东西扩展
19          N = 5  # 南北扩展
20          d = 80  # 格点距（km）
21
22          # 投影参数
23          phy0 = 45  # 中央纬度
24          seita1 = 30  # 第一标准纬度
25          seita2 = 60  # 第二标准纬度
26          a = EARTH_RADIUS  # 地球半径
27
28          return M, N, d, phy0, seita1, seita2, a
29
30      def calculate_magnification_factor_and_coriolis_parameter(
            proj, In, Jn, d):
31          """计算投影的放大系数和科氏参数
32
33          Args:
34              proj (str): 投影方式 ('lambert', 'mercator', '
                    stereographic')
35              In (float): 东西方向的网格索引
36              Jn (float): 南北方向的网格索引
37              d (float): 网格距离
38
39          Returns:
40              tuple: (m, f) 其中m为放大系数, f为科氏参数
41          """
42          if proj == 'lambert':
43              k = 0.7156
44              le = 11423.37
45              l = np.sqrt((In**2 + Jn**2) * d**2)
46              m = k * l / EARTH_RADIUS / np.sqrt(1 - ((le**(2/k) -
                        l**(2/k)) / (le**(2/k) + l**(2/k)))**2)
47              f = 2 * OMEGA * (le**(2/k) - l**(2/k)) / (le**(2/k)
                    + l**(2/k))
48
49          elif proj == 'mercator':
50              m = np.sqrt((EARTH_RADIUS * np.cos(np.deg2rad(22.5))
```

```
                )**2 + (Jn*d)**2) / EARTH_RADIUS
            f = 2 * OMEGA * np.sin(Jn * d / np.sqrt((
                EARTH_RADIUS * np.cos(np.deg2rad(22.5)))**2 + (Jn
                *d)**2))

        elif proj == 'stereographic':
            le = 11888.45
            l = np.sqrt((In**2 + Jn**2) * d**2)
            m = (2 + np.sqrt(3)) / 2 / (1 + ((le**2 - l**2) / (
                le**2 + l**2)))
            f = 2 * OMEGA * ((le**2 - l**2) / (le**2 + l**2))

        else:
            raise ValueError('投影方式输入错误！')

        return m, f

    def calculate_projection_factors(M, N, d, phy0, seita1,
        seita2, a):
        """计算投影因子"""
        # 计算圆锥常数
        k = (np.log(np.sin(np.deg2rad(seita1))) - np.log(np.sin(
            np.deg2rad(seita2)))) / \
            (np.log(np.tan(np.deg2rad(seita1/2))) - np.log(np.
                tan(np.deg2rad(seita2/2))))

        # 计算映像平面上赤道到北极点的距离
        le = a * np.sin(np.deg2rad(seita1)) / k * (1/np.tan(np.
            deg2rad(seita1/2)))**k

        # 计算参考距离
        l_ref = le * (np.cos(np.deg2rad(phy0))/(1+np.sin(np.
            deg2rad(phy0))))**k

        # 初始化放大系数矩阵
        m = np.zeros((2 * N + 1, 2 * M + 1))

        # 计算每个网格点的放大系数
        for In in range(-M, M+1):
            for Jn in range(-N, N+1):
                l = np.sqrt((abs(In) * d)**2 + (l_ref-Jn*d)**2)
                m[Jn+N, In+M] = k*l/(a*np.sqrt(1 - \
                    ((le**(2/k) - l**(2/k))/(le**(2/k) + l**(2/k
```

```
                    )))**2))
85
86          return np.flipud(m)
87
88      def calculate_grid_coordinates(M, N, d, phy0):
89          """计算网格点的经纬度坐标"""
90          # 计算经纬度范围
91          lon_range = np.linspace(-M*d/111, M*d/111, 2*M+1)  # 将
                距离转换为经度（大约111km/度）
92          lat_range = np.linspace(phy0-N*d/111, phy0+N*d/111, 2*N
                +1)  # 将距离转换为纬度
93
94          # 生成网格点坐标
95          lon_grid, lat_grid = np.meshgrid(lon_range, lat_range)
96          return lon_range, lat_range, lon_grid, lat_grid
97
98      def plot_magnification_factor(m):
99          """绘制放大系数等值线图"""
100         # 获取网格参数
101         M, N, d, phy0, seita1, seita2, a = set_parameters()
102
103         # 计算网格点坐标
104         lon_range, lat_range, lon_grid, lat_grid =
                calculate_grid_coordinates(M, N, d, phy0)
105
106         plt.figure(figsize=(12, 10))
107
108         # 绘制等值线
109         levels = np.linspace(np.min(m), np.max(m), 15)  # 增加等
                值线级别
110         contour = plt.contour(lon_grid, lat_grid, m, levels=
                levels, colors='k')
111         plt.clabel(contour, inline=True, fontsize=8)  # 添加等值
                线标签
112
113         # 添加填充等值线
114         contourf = plt.contourf(lon_grid, lat_grid, m, levels=
                levels, cmap='RdYlBu_r', alpha=0.6)
115         plt.colorbar(contourf, label='放大系数')
116
117         # 设置网格和标签
118         plt.grid(True, linestyle='--', alpha=0.5)
119         plt.title('兰伯特投影放大系数分布', pad=15, fontsize=14)
```

```
120         plt.xlabel('经度 (°)', fontsize=12)
121         plt.ylabel('纬度 (°)', fontsize=12)
122 
123         # 调整显示范围，确保所有网格点可见
124         plt.margins(0.1)
125         plt.tight_layout()
126 
127         # 设置刻度格式
128         plt.gca().xaxis.set_major_formatter(plt.FuncFormatter(
129             lambda x, p: f'{x:.1f}°E' if x > 0 else f'{-x:.1f}°W
                ' if x < 0 else '0°'))
130         plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(
131             lambda x, p: f'{x:.1f}°N' if x > 0 else f'{-x:.1f}°S
                ' if x < 0 else '0°'))
132 
133         plt.show()
134 
135     def save_to_excel(lon_range, lat_range, m, f):
136         """保存结果到Excel文件"""
137         # 创建结果数据
138         data = []
139         for i in range(len(lon_range)):
140             for j in range(len(lat_range)):
141                 data.append({
142                     '经度': f'{lon_range[i]:.2f}°E',
143                     '纬度': f'{lat_range[j]:.2f}°N',
144                     '地图系数': f'{m[j,i]:.4f}',
145                     '科里奥利参数': f'{f[j,i]:.4e}'
146                 })
147 
148         # 创建DataFrame并保存
149         df = pd.DataFrame(data)
150         df.to_excel('习题一_地图系数.xlsx', index=False)
151         df.to_excel('习题一_科里奥利参数.xlsx', index=False)
152 
153     def main():
154         # 设置中文字体
155         set_chinese_font()
156 
157         # 设置参数
158         M, N, d, phy0, seita1, seita2, a = set_parameters()
159 
160         # 计算放大系数
```

```
161         m = calculate_projection_factors(M, N, d, phy0, seita1,
                seita2, a)
162
163         # 计算经纬度范围（中心点为100°E）
164         lon_range = np.linspace(100-M*d/111, 100+M*d/111, 2*M+1)
                # 将距离转换为经度
165         lat_range = np.linspace(phy0-N*d/111, phy0+N*d/111, 2*N
                +1)  # 将距离转换为纬度
166
167         # 计算科里奥利参数
168         f = np.zeros_like(m)
169         for i in range(2*M+1):
170             for j in range(2*N+1):
171                 In = i - M
172                 Jn = N - j  # 注意这里需要翻转j以匹配m矩阵
173                 _, f[j,i] =
                        calculate_magnification_factor_and_coriolis_parameter
                        ('lambert', In, Jn, d)
174
175         # 保存结果到Excel
176         save_to_excel(lon_range, lat_range, m, f)
177
178         # 绘制等值线图
179         plot_magnification_factor(m)
180
181     if __name__ == '__main__':
182         main()
```

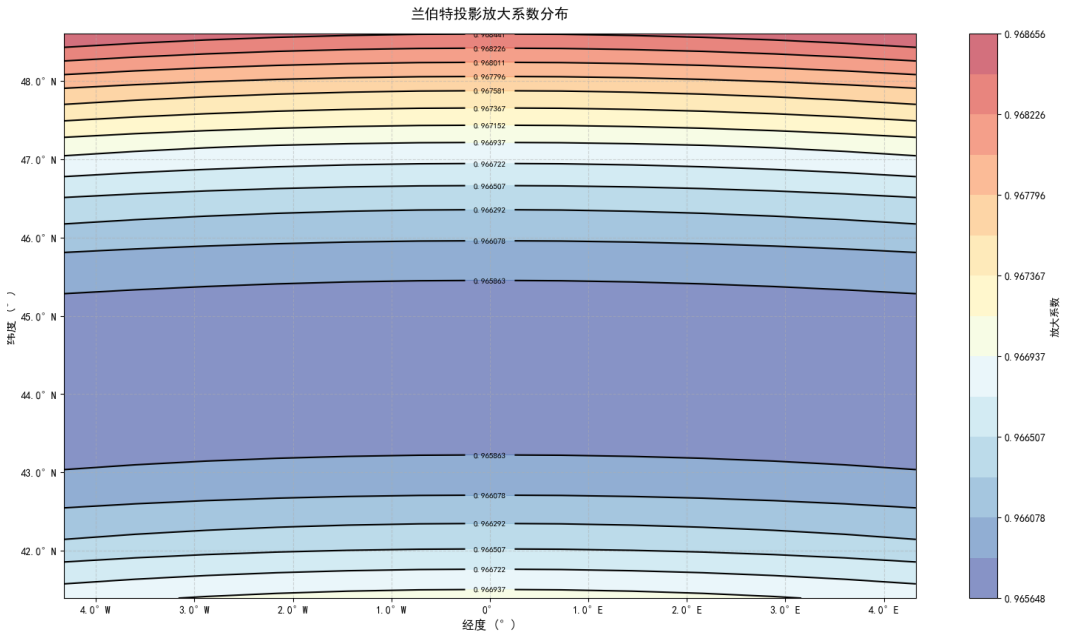

Figure 2: 中央经度为 45° 的兰伯特投影放大系数分布

2.2 习题二

中尺度天气预报模式 WRF 的兰勃特投影如图 2.5 所示，标准纬度 1 和标准纬度 2 也可根据模拟区域不同而有所调整，使模拟区域内地图放大系数尽可能接近 1，从而减小模式网格上的投影形变。设模拟区域为东亚地区 (100° 150°E，20° 60°N)，尝试编写程序，计算兰勃特投影在东亚地区的地图放大系数

Listing 3: 习题二

```
1
2     import numpy as np
3     import matplotlib.pyplot as plt
4     import pandas as pd
5
6     # 基本参数
7     EARTH_RADIUS = 6371  # 地球半径
8     OMEGA = 7.292e-5  # 地球自转角速度
9
10    def set_chinese_font():
11        """设置matplotlib中文字体"""
12        plt.rcParams['font.sans-serif'] = ['SimHei']  # 设置中文
              字体为黑体
13        plt.rcParams['axes.unicode_minus'] = False  # 解决负号显
              示问题
14
15    def set_parameters():
16        """设置投影参数"""
17        # 网格参数
18        M = 10  # 东西扩展（考虑到东亚地区的经度范围较大）
19        N = 8  # 南北扩展（考虑到纬度范围）
20        d = 500  # 格点距（km）
21
22        # 投影参数
23        phy0 = 40  # 中央纬度（东亚地区的中心纬度）
24        seita1 = 30  # 第一标准纬度
25        seita2 = 50  # 第二标准纬度（调整以优化东亚地区的投影）
26        a = EARTH_RADIUS  # 地球半径
27
28        return M, N, d, phy0, seita1, seita2, a
29
30    def calculate_projection_factors(M, N, d, phy0, seita1,
          seita2, a):
31        """计算投影因子"""
32        # 计算圆锥常数
```

```
33          k = (np.log(np.sin(np.deg2rad(seita1))) - np.log(np.sin(
                np.deg2rad(seita2)))) / \
34              (np.log(np.tan(np.deg2rad(seita1/2))) - np.log(np.
                    tan(np.deg2rad(seita2/2))))
35
36          # 计算映像平面上赤道到北极点的距离
37          le = a * np.sin(np.deg2rad(seita1)) / k * (1/np.tan(np.
                deg2rad(seita1/2)))**k
38
39          # 计算参考距离
40          l_ref = le * (np.cos(np.deg2rad(phy0))/(1+np.sin(np.
                deg2rad(phy0))))**k
41
42          # 初始化放大系数矩阵
43          m = np.zeros((2 * N + 1, 2 * M + 1))
44
45          # 计算每个网格点的放大系数
46          for In in range(-M, M+1):
47              for Jn in range(-N, N+1):
48                  l = np.sqrt((abs(In) * d)**2 + (l_ref-Jn*d)**2)
49                  m[Jn+N, In+M] = k*l/(a*np.sqrt(1 - \
50                      ((le**(2/k) - l**(2/k))/(le**(2/k) + l**(2/k
                            )))**2))
51
52          return np.flipud(m)
53
54      def plot_magnification_factor(m):
55          """绘制放大系数等值线图"""
56          # 设置经纬度范围（东亚地区：100°~150°E，20°~60°N）
57          lon_range = np.linspace(100, 150, 2*10+1)  # 经度范围
58          lat_range = np.linspace(20, 60, 2*8+1)     # 纬度范围
59
60          # 生成网格点坐标
61          lon_grid, lat_grid = np.meshgrid(lon_range, lat_range)
62
63          plt.figure(figsize=(12, 10))
64
65          # 绘制等值线
66          levels = np.linspace(np.min(m), np.max(m), 15)  # 增加等
                值线级别
67          contour = plt.contour(lon_grid, lat_grid, m, levels=
                levels, colors='k')
68          plt.clabel(contour, inline=True, fontsize=8)  # 添加等值
```

```
        线标签
69
70          # 添加填充等值线
71          contourf = plt.contourf(lon_grid, lat_grid, m, levels=
                levels, cmap='RdYlBu_r', alpha=0.6)
72          plt.colorbar(contourf, label='放大系数')
73
74          # 设置网格和标签
75          plt.grid(True, linestyle='--', alpha=0.5)
76          plt.title('东亚地区兰伯特投影放大系数分布', pad=15,
                fontsize=14)
77          plt.xlabel('经度 (°)', fontsize=12)
78          plt.ylabel('纬度 (°)', fontsize=12)
79
80          # 调整显示范围
81          plt.xlim(100, 150)
82          plt.ylim(20, 60)
83
84          # 设置刻度格式，强调东西南北方向
85          plt.gca().xaxis.set_major_formatter(plt.FuncFormatter(
86              lambda x, p: f'{x}°E'))
87          plt.gca().yaxis.set_major_formatter(plt.FuncFormatter(
88              lambda x, p: f'{x}°N'))
89
90          plt.tight_layout()
91          plt.show()
92
93      def save_to_excel(lon_range, lat_range, m, f):
94          """保存结果到Excel文件"""
95          # 创建结果数据
96          data = []
97          for i in range(len(lon_range)):
98              for j in range(len(lat_range)):
99                  data.append({
100                     '经度': f'{lon_range[i]:.2f}°E',
101                     '纬度': f'{lat_range[j]:.2f}°N',
102                     '地图系数': f'{m[j,i]:.4f}',
103                     '科里奥利参数': f'{f[j,i]:.4e}'
104                 })
105
106         # 创建DataFrame并保存
107         df = pd.DataFrame(data)
108         df.to_excel('习题二_地图系数.xlsx', index=False)
```

```
109         df.to_excel('习题二_科里奥利参数.xlsx', index=False)
110
111     def main():
112         # 设置中文字体
113         set_chinese_font()
114
115         # 设置参数
116         M, N, d, phy0, seita1, seita2, a = set_parameters()
117
118         # 计算放大系数
119         m = calculate_projection_factors(M, N, d, phy0, seita1,
                seita2, a)
120
121         # 计算经纬度范围
122         lon_range = np.linspace(100, 150, 2*M+1)
123         lat_range = np.linspace(20, 60, 2*N+1)
124
125         # 计算科里奥利参数
126         f = 2 * OMEGA * np.sin(np.deg2rad(lat_range))[:, np.
                newaxis] * np.ones_like(m)
127
128         # 保存结果到Excel
129         save_to_excel(lon_range, lat_range, m, f)
130
131         # 绘制等值线图
132         plot_magnification_factor(m)
133
134     if __name__ == '__main__':
135         main()
```

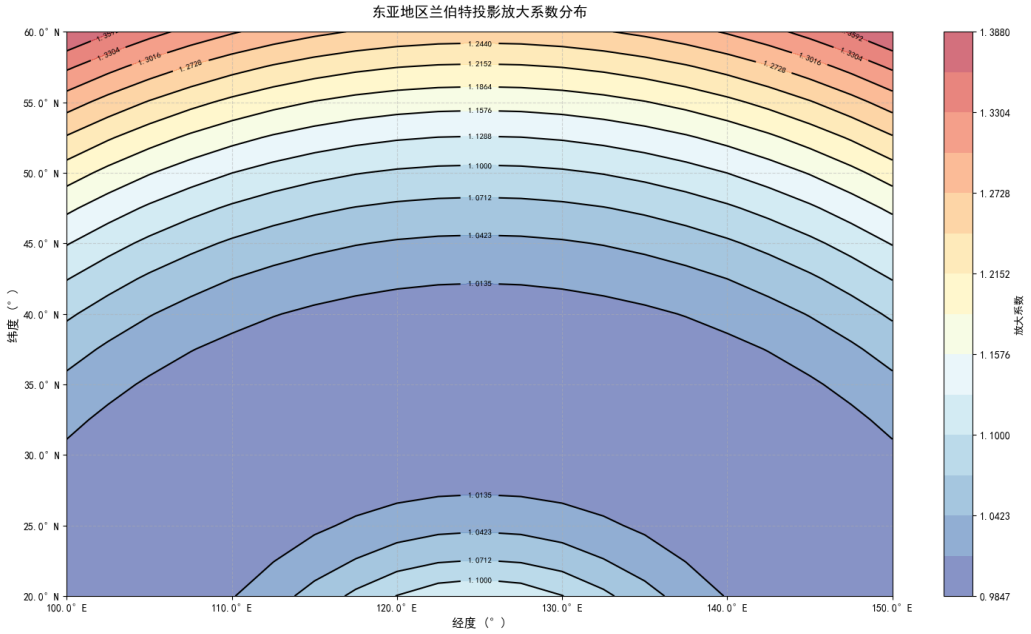

Figure 3: 兰伯特投影在东亚地区的放大系数分布