



中山大學

SUN YAT-SEN UNIVERSITY

智慧城市建設原理與方法

Java 網絡計算類開發與應用

以北江流域為例

學號: _____

姓名: _____

一、设计目标

1. 任务要求分析

实验主要考察 Java 网络计算类的开发与应用，从数据、功能和性能三个方面提出了要求。

数据上，要求选择面积大于万平方公里的、网格数量达到百万级别的研究区域，和其在汛期至少一个月的降雨数据进行插值计算；功能上，需要完成填洼、流向、坡度、累积流的计算，用至少三种方法进行降雨插值；性能上，需要能适应网格大小不定的需求，采用 MySQL 数据库对数据进行管理，自动进行数据的读写，在方法中使用多线程技术，最终以图片格式输出计算结果。

2. 总体技术路线

针对以上任务要求，本实验选择合适的数据源，用 MySQL 数据库进行管理，使用数据流输入方式，以动态数组存储，在主要的计算方法中使用多线程技术，并封装为类独立调用，可视化后输出图片结果，整体的技术路线如下图所示。

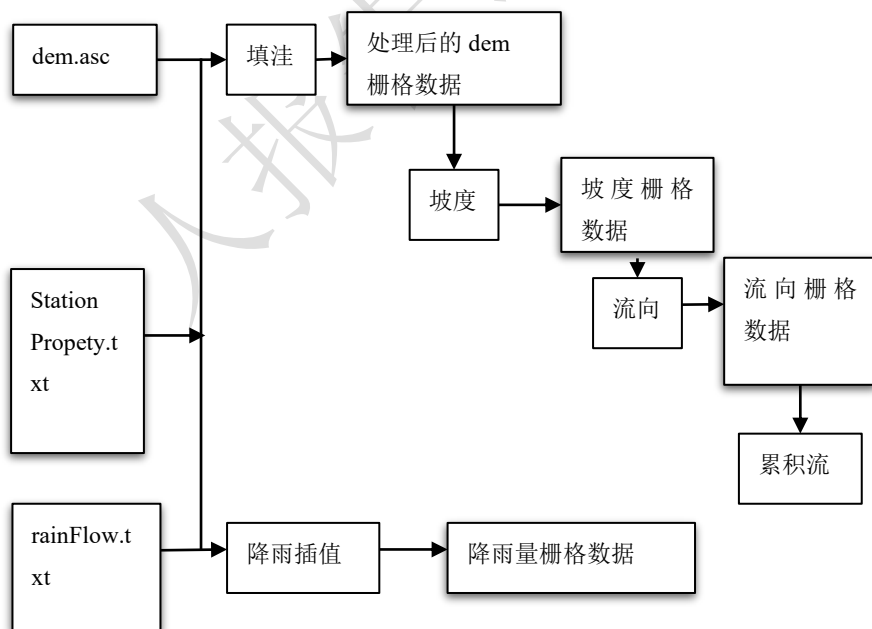


图 1 技术路线图

二、研究区域及数据来源

北江流域为珠江流域第二大支流,流域总面积为 46710km^2 ,干流总长 468km 。流域内大部分是山区和丘陵,呈现出北高南低的地势特征。北江流域年平均降水量为 1800mm ,汛期为 4~9 月。作为人口稠密的珠江三角洲上游的重要水系,研究北江流域的水文特征有重要的意义。

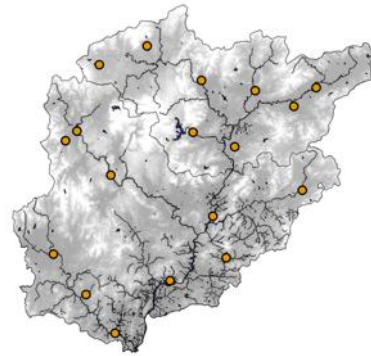


图 2 实验数据区域

本实验自欧洲空间局下载分辨率为 90m 的 DEM 数据,网格数量为 9736797 个;选择欧洲中期天气预报中心 (ECMWF) 2010 年 4~8 月 (汛期内) 18 个雨量站逐日的降水量数据进行插值计算。

三、研究方法及算法设计

1. 填洼: W&L 算法

DEM 数据时进行水文分析的基础数据,是比较光滑的地形表面的模拟,但是由于数据误差以及一些真实地形 (例如喀斯特地形) 的存在,使得 DEM 数据表面存在一些凹陷的区域。因为这些凹陷区域的存在,在进行水流方向计算时往往会得到不合理甚至错误的结果。因此,在进行水文分析之前,需要对原始 DEM 数据进行洼地填充,从而得到无洼地的无凹陷 DEM 数据。

本实验采用 L. Wang 和 H. Liu 在 2006 年提出的一种速度快、易于理解的算法 (以下简称 W&L 算法) 进行填洼。

W&L 算法引入“溢出高程”的概念,并在最小代价搜索框架内利用优先队列的数据结构来实现填洼处理。该方法从数字高程模型 (DEM) 的边界最低点出发,即从理论上可能是流域出口的位置开始向内搜索,逐步比较当前栅格与其相邻栅格的高程数据,最终实现整个 DEM 数据的更新。

由于 DEM 数据量较大，在填洼算法中加入并行处理的操作。具体来说算法的操作流程及相应的代码设计如下：

(1) 初始化

将所有边界上的最低点视为潜在的水流出口，加入到优先队列之中。

```
1. // 将非 NODATA_value 的边界网格插入优先队列
2. for (int i = startRow; i <= endRow; i++) {
3.     for (int j = startCol; j <= endCol; j++) {
4.         // 检查是否为边界并且不是 NODATA_value
5.         if ((i == startRow || i == endRow || j == startCol || j == endCol) && dem[i][j] != NODATA_value) {
6.             queue.add(new int[]{i, j, dem[i][j]});
7.         }
8.     }
9. }
```

(2) 迭代更新

从优先队列中选择具有最低溢出高程的栅格作为当前处理对象。

对于选定的栅格，检查其与周围邻接栅格之间的关系；如果某个邻接栅格的原始高程低于当前栅格的溢出高程，则更新该邻接栅格的高程值为当前栅格的溢出高程，并将其加入优先队列中继续处理。

当优先队列为空时，则说明没有栅格需要继续处理，则填洼过程结束。

```
1. // 主循环
2. while (!queue.isEmpty()) {
3.     int[] current = queue.poll(); // 取出队列中高度最低的栅格
4.     int row = current[0];
5.     int col = current[1];
6.     int height = current[2];
7.     // 遍历邻接网格
8.     for (int[] direction : getDirections(rows, cols)) {
9.         int newRow = row + direction[0];
10.        int newCol = col + direction[1];
11.        if (isValidCell(newRow, newCol, startRow, endRow, startCol, endCol) && dem[newRow][newCol] != NODATA_value) {
12.            // 计算新高度，取相邻栅格原始高度和当前栅格高度的最大值
13.            int newHeight = Math.max(dem[newRow][newCol], height);
14.            // 如果新高度大于当前相邻栅格的高度，则更新相邻栅格的高度，并将其加入优先队列
15.            if (dem[newRow][newCol] < newHeight) {
16.                dem[newRow][newCol] = newHeight;
```

```
17.         queue.add(new int[]{newRow, newCol, newHeight});
18.     }
19. }
20. }
21. }
```

(3) 并行处理

通过 Java 的 ForkJoinPool 框架和 RecursiveTask 类实现，将整个 DEM 分割成多个较小的子区域，对子区域进行并发处理。首先创建 ForkJoinPool 实例用于管理并行任务，再作为根任务提交给线程池执行，invoke() 启动任务。

```
1. ForkJoinPool pool = new ForkJoinPool();
2. FillTask task = new FillTask(filledDEM, 0, rows - 1, 0, cols - 1, rows, cols);
3. pool.invoke(task);
```

根据当前处理区域的大小决定是否进一步拆分任务，当子区域大小小于或等于 1000 个栅格时，直接处理该区域而不是继续分割。每个子任务完成自己的工作后，会自动返回给父任务，所有任务完成后整个填洼过程结束。FillTask 类继承自 Fork/Join 框架中的 RecursiveTask<Void>。

```
1. @Override
2. protected Void compute() {
3.     if ((endRow - startRow + 1) * (endCol - startCol + 1) <= 1000) { // 如果子区域足够小，则直接处理
4.         processRegion(startRow, endRow, startCol, endCol);
5.     } else {
6.         int midRow = (startRow + endRow) / 2;
7.         int midCol = (startCol + endCol) / 2;
8.
9.         invokeAll(
10.            new FillTask(dem, startRow, midRow, startCol, midCol, rows, cols),
11.            new FillTask(dem, startRow, midRow, midCol + 1, endCol, rows, cols),
12.            new FillTask(dem, midRow + 1, endRow, startCol, midCol, rows, cols),
13.            new FillTask(dem, midRow + 1, endRow, midCol + 1, endCol, rows, cols)
14.        );
15.     }
16.     return null;
17. }
```

个人报告请勿转载

3. 坡度：中心差分公式

对于一个二维函数 $f(x, y)$ ，在点 (x_i, y_i) 处沿 x 和 y 方向的梯度可以分别用以下中心差分公式近似：

东西方向：

$$\frac{\partial z}{\partial x} = \frac{f(x_{i+1}, y_j) - f(x_{i-1}, y_j)}{2\Delta x} \quad (1)$$

南北方向：

$$\frac{\partial z}{\partial y} = \frac{f(x_i, y_{j+1}) - f(x_i, y_{j-1})}{2\Delta y} \quad (2)$$

将两个方向的梯度合成：

$$slope = \sqrt{\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2} \quad (3)$$

具体实现过程中调用了 Java 的 Math 库执行以上的数学运算。

4. 流向：混合流向算法

在地表坡面径流过程中，坡度较大的区域，径流通常呈汇聚形式，表现为单方向流动，因此单流向算法能够较好地模拟这些区域的水流运动。在坡度平缓的地表上，水流往往会分流到多个方向，向周围较低高程处流动。在这种情况下，多流向算法相比于单流向算法更能准确地模拟水流向四周分配的现象，反映了水流在低洼区域的自然扩散特性。因此，本实验尝试利用混合的流向算法，对于陡峭区域施加单流向算法（D8），平缓区域施加多流向算法（MFD），以综合两种算法的优点。

（1）陡峭和平缓区域划分

设置坡度阈值，将坡度大于 20° 的栅格标记为陡峭。从而区分出陡峭和平缓的区域。

（2）陡峭区域：D8 算法

假设单个栅格中的水流只能流入与之相邻的 8 个栅格中。它用最陡坡度法来确定水流的方向，即在 3×3 的 DEM 栅格上，计算中心栅格与各相邻栅格间的距

离权落差（即栅格中心点落差除以栅格中心点之间的距离），取距离权落差最大的栅格为中心栅格的流出栅格。

用常量数组 DIRECTIONS 定义八个可能的方向（东、东南、南、西南、西、西北、北、东北），遍历查找具有最大高差的方向作为流向。

```

1. private int d8(int row, int col) {
2.     int maxDir = 0;
3.     double maxDiff = Double.NEGATIVE_INFINITY;
4.
5.     for (int dir = 0; dir < 8; dir++) {
6.         int r = row + DIRECTIONS[dir][0];
7.         int c = col + DIRECTIONS[dir][1];
8.
9.         if (isValid(r, c) && filledDEM[r][c] < filledDEM[row][col])
10.        {
11.            double diff = filledDEM[row][col] - filledDEM[r][c];
12.            if (diff > maxDiff) {
13.                maxDiff = diff;
14.                maxDir = D8_DIRECTIONS[dir];
15.            }
16.        }
17.
18.    return maxDir;
19. }

```

（3）平缓区域：MfD 算法

多流向算法的水流分配策略以坡度加权为基础，计算公式如下：

$$d_i = \frac{(\tan\beta_i)^p \times L_i}{\sum_{j=1}^8 (\tan\beta_j)^p \times L_j} \quad (4)$$

式中， d_i 为水流对第*i*号邻域像素的分配比例； $\tan\beta$ 为坡度； p 为水流分配权重； L_i 为对第*i*号邻域像素的等高线加权因子，定义为

$$L_i = \begin{cases} \frac{1}{2}, & \text{第 } i \text{ 号邻域像素高程低于中心像素，且位于水平或垂直方向} \\ \frac{\sqrt{2}}{4}, & \text{第 } i \text{ 号邻域像素高程低于中心像素，且位于对角线方向} \\ 0, & \text{第 } i \text{ 号邻域像素高程不低于中心像素} \end{cases} \quad (5)$$

5. 累积流

累积流是指在水文建模中，每个栅格点收集的上游水流总量，结合流向的计算结果进一步计算累积流。对于陡峭区域利用 D8 算法计算的流向更新累积流值，具体逻辑是找到最大高度差的方向，并将当前栅格点的流量加到该方向的目标栅格点上。多流向算法考虑了水流可以向多个方向流动的情况，并按式（4）权重分配流量。

采用 Java 8 引入的并行流的方式加速计算，从而利用多核处理器的优势来提高性能。

```
1. IntStream.range(0, nrows).parallel().forEach(i -> {
2.     IntStream.range(0, ncols).parallel().forEach(j -> {
3.         if (isValidCell(i, j)) {
4.             updateFlowAccumulation(flowAcc, i, j);
5.         }
6.     });
7. });
```

6. 降雨插值

本实验选择反距离权重法（IDW）、趋势面分析法和径向基函数法（RBF）进行插值。IDW 通过简单易懂的距离衰减模型，能够快速生成局部精度较高的插值结果，特别适合数据点密集且分布均匀的情况；趋势面分析利用全局多项式拟合，提供了平滑、连续的表面估计，适用于需要宏观趋势分析或背景场重建的应用；而 RBF 结合了局部和全局特性，能够在保持较高灵活性的同时提供精确的插值效果，尤其擅长处理复杂地形和不规则分布的数据点。

（1）坐标转换

由于插值过程都需要计算距离或空间坐标点，因此首先需要进行坐标转换。DEM 数据中各栅格的投影坐标系为 WGS 1984 49N，雨量站属性的经纬度则为地面直角坐标系。引入 GeoTools 库实现此功能。

坐标参考系统定义如下：

```
1. static final String SRC_CRS = "EPSG:4326"; // WGS 84 (Geographic)
2. static final String DST_CRS = "EPSG:32649"; // WGS 84 / UTM zone 49N
   (Projected)
3.
4. CoordinateReferenceSystem sourceCRS = CRS.decode(SRC_CRS);
5. CoordinateReferenceSystem targetCRS = CRS.decode(DST_CRS, true);
```

GeoTools 中 DirectPosition2D 类用于表示二维位置，构造函数接收坐标参考系统和经纬度或直角坐标值，执行坐标转换如下：

```
1. DirectPosition2D geoPosition = new DirectPosition2D(sourceCRS, station.  
    getLatitude(), station.getLongitude());  
2. DirectPosition2D utmPosition = new DirectPosition2D(targetCRS);  
3. InverseDist.transform.transform(geoPosition, utmPosition);
```

(2) 反距离权重

反距离加权法 (Inverse Distance Weighting, IDW) 是一种常见的空间插值方法。IDW 假设每个已知点对未知点的影响随着距离的增加而减少，具体影响程度由距离的倒数决定。通常还会引入一个幂参数以调整这种衰减的速度。其计算权重的公式如下：

$$w_i = \frac{1}{d_i^p} \quad (6)$$

(3) 趋势面

趋势面是一个光滑的数学曲面，它能够集中地反映空间数据在大范围内的变化趋势，它是揭示面状区域上连续分布的现象空间变化规律的理想工具，也是实际当中经常使用的描述空间趋势的主要方法。

实验中对于每个栅格点构建了一个设计矩阵 A ，它包含了所有参与插值计算的站点坐标 x 和 y 的项。这里假设了最简单的线性模型，即：

$$Z = a_0 + a_1x + a_2y \quad (7)$$

以每个站点的实际降水量为观测值向量 b 。

求解最小二乘问题 $Ax = b$ 以找到最佳拟合参数 x (即多项式的系数)。首先计算 $A^T A$ 和 $A^T b$ ，再对 $A^T A$ 进行 LU 分解，最后使用前向和后向替换来求解线性方程组。

(4) 径向基函数

在空间插值中，径向基函数可以用来衡量已知数据点对于未知位置的影响程度。通过将所有已知点的影响加权求和，可以估计出未知位置的属性值。对应的数学表达式为：

$$\varphi = \frac{1}{\sqrt{r^2 + \varepsilon^2}} \quad (8)$$

ε 是两点间的距离， r 是起平滑作用的参数。考虑到量纲，本实验中设置为10000。

7. 可视化

将不同类型的二维数组数据（整数、双精度浮点数和布尔值）转换为图像文件。这些方法根据输入的数据生成伪彩色或灰度图像。

`getGrayscaleColor` 和 `getPseudoColor` 方法分别为给定的归一化值提供灰度颜色和伪彩色。

```
1. static Color getGrayscaleColor(double normalizedValue) {  
2.     int grayLevel = (int) (normalizedValue * 255);  
3.     return new Color(grayLevel, grayLevel, grayLevel);  
4. }  
5. static Color getPseudoColor(double normalizedValue) {  
6.     // 使用一个简单的三段式颜色映射：蓝色 -> 绿色 -> 红色  
7.     int blue = (int) (normalizedValue * 255); // 低海拔区域为蓝色  
8.     int green = (int) (255 * Math.abs(normalizedValue - 0.5)); //  
        中间海拔区域为绿色  
9.     int red = (int) ((1 - normalizedValue) * 255); // 高海拔区域为红  
        色  
10.  
11.     return new Color(red, green, blue);  
12. }
```

四、 研究结果及评估

1. 填洼

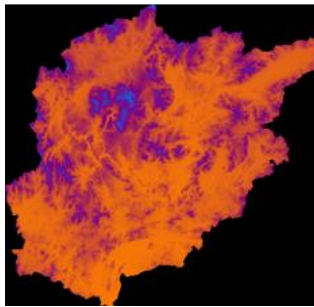


图 3

填洼前结果 (-51~1894m)

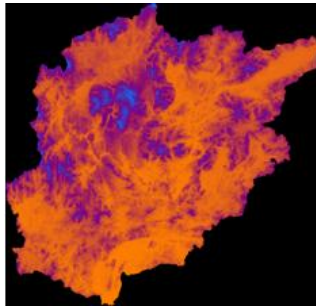


图 4

填洼后结果 (2~1894m)

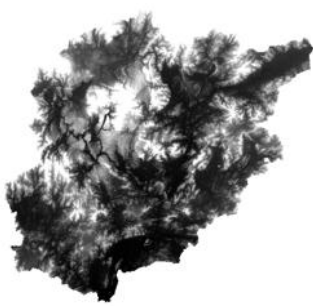


图 5

Arcmap 填洼 (1~1894m)

W&L 算法很好地执行了填洼，将低于 2m 的洼地填平，与 Arcmap 里的输出结果较为相近。

2. 坡度计算

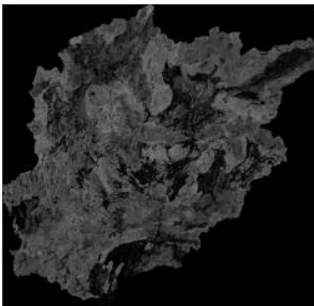


图 6

坡度计算结果



图 7

Arcmap 坡度计算结果

坡度计算与 Arcmap 的范围一致，均为 1.72~45。

3. 流向计算

(1) 单流向算法

对全局施加单流向 D8 算法，得到结果如上图所示，和 Arcmap 计算的结果基本一致。其中 1：东；2：东南；4 南；8：西南；16：西；32：西北；64：北；128：东北。

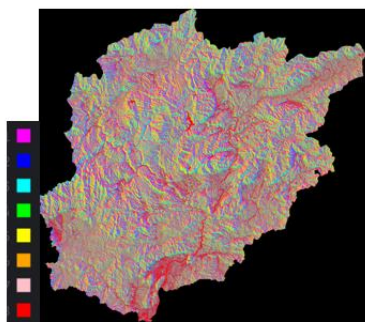


图 8
D8 算法

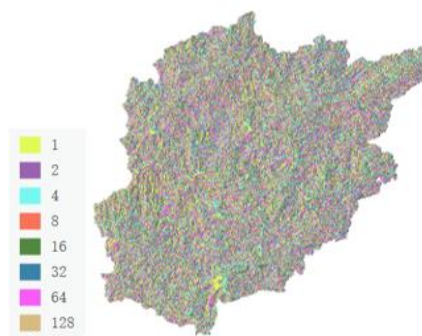


图 9
Arcmap 流向计算

(2) 地形分类

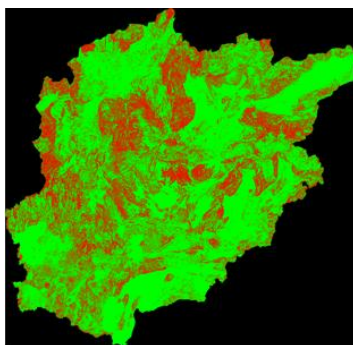


图 10 地形分类结果
红色为陡峭区域，绿色为平缓区域。

(3) 混合流向算法



图 11 混合流向算法

混合流向算法的可视化方式为各个流向颜色的叠加，在图上的显示效果不明显。

4. 累积流计算

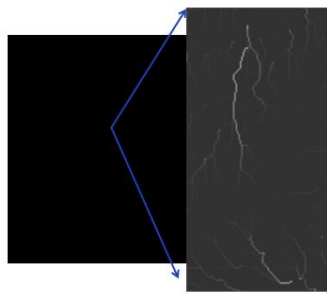


图 12 单流向算法累积流提取

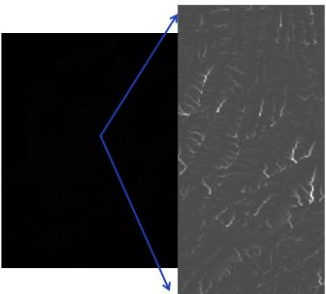


图 13 混合流向算法累积流提取

累积流的计算结果只提取到了一些细小水系，存在较大问题。

5. 降雨插值计算

三种插值方法为每一天都输出了一张插值结果，如下图所示。

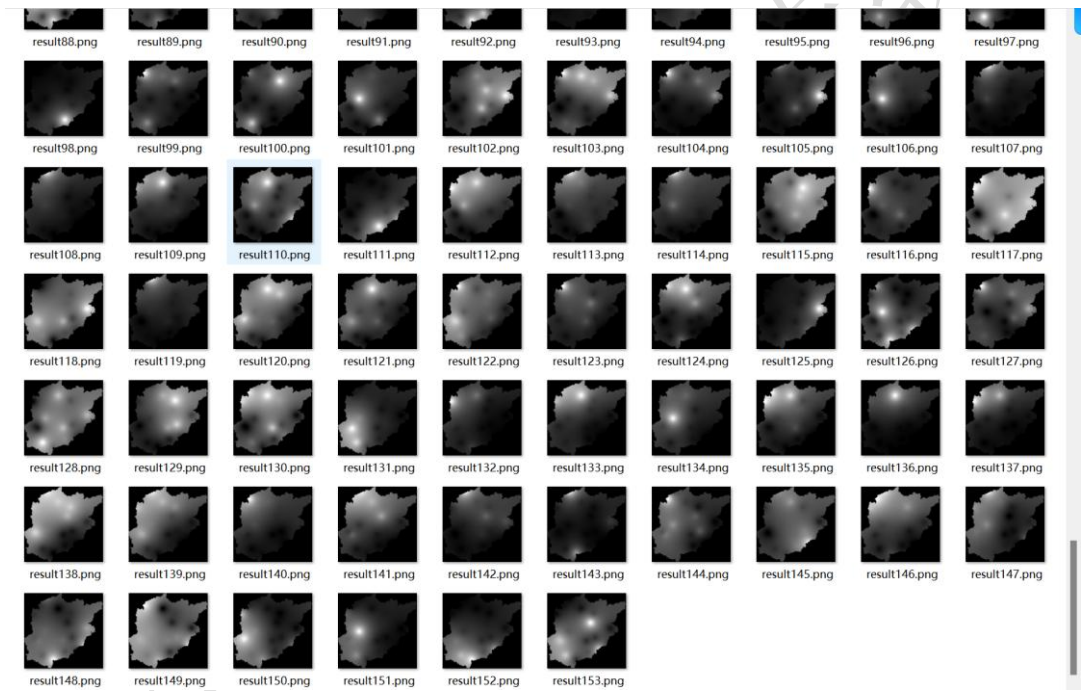


图 14 降雨插值结果一览

以 4 月 1 日的结果为例，三种插值方法结果如下。具体的分析请见第五部分讨论。



图 15 反距离权重插值结果



图 16 径向基函数插值结果



图 17 趋势面函数插值结果

五、 讨论

1. 插值方法对比

反距离权重法和径向基函数都有点状分布的痕迹，尤其是前者，因为这两种插值方法都基于距离计算权重，雨量站的分布会显著影响各栅格点的计算结果；趋势面插值法的计算结果最为平滑，但与实际情况出入较大。

2. 北江流域降雨特征分析

(1) 时间特征



图 18 RBF 插值结果

基于 RBF 的插值结果，从较长时序的尺度可以看出，北江流域在后汛期的降雨量更大，持续时间更长。

比如，result95~result101（7 月 4 日~7 月 10 日）和 result105~116（7 月 14 日至 7 月 25 日），北江流域的插值结果图都显示为深色，意味着降雨量较大，在两星期内密集地出现了大范围、高强度的降水。这段时间内北江流域的最高降雨量达到了 72.7mm，已经达到了暴雨级别的降水量。

这一方面是由于北江流域的季风气候特征比较明显，夏季是集成的降水季节。另一方面，2010 年发生了南方特大暴雨事件，全国有 40 多条较大河流发生超过

历史记录的特大洪水，65 座县级以上城市受淹，8 座小型水库垮坝。其中，江西中部偏南地区、广东西北部等地伴有短时雷雨大风等强对流天气，表现在了北江流域的降雨插值结果之中。

（2） 空间特征

基于反距离权重法的插值结果，对逐日的降雨量进行平均，得到北江流域 4~8 月的平均降雨插值结果，如下图所示。

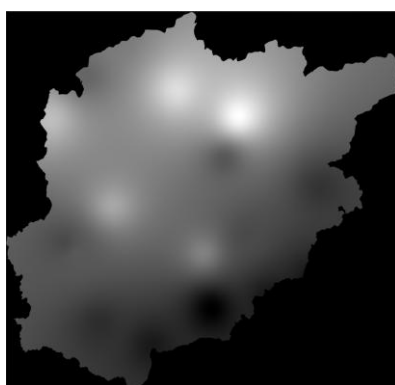


图 19 反距离权重插值平均结果

从图中可以读出，北江流域的降雨量在空间上呈现出自东南部向西北部递减的特征。

北江流域属于典型的东亚季风区，夏季时，东南季风携带大量水汽从海面吹向陆地，给东南沿海及附近区域带来丰富的降水。北江流域的东南部靠近海岸线，地势相对较低且平坦。相比之下，西北部则多为山地和丘陵地带，地势较高，表现出明显的迎风坡效应。当湿润空气从海洋吹向内陆时，遇到山脉等地阻挡，被迫抬升，在上升过程中气温下降，水汽凝结形成降雨。因此，位于迎风面的东南部地区往往会有更多的降水。

六、 结论

本实验基于 Java 网格计算类对北江流域进行了水文分析，完成了填洼、坡度、流向和累积流的计算，利用三种降雨插值方法对流域内降水进行了插值及可视化，得出了以下结论：北江流域的降水主要集中在后汛期，降雨量大、降水时间长；北江流域的降水呈现出自东南部向西北部递减的特征。

七、 不足与展望

实验对于混合流向算法的尝试结果并不如直接使用单流向算法的效果好。可能是由于北江流域的地形复杂度并没有达到需要分地形计算流向的程度，由此导致累积流对于水系的提取过于细小，可视化效果差。未来可以关注这一点改进流向和累积流的计算方式，并对降雨插值的结果进行定量的精度评估。

八、 心得体会

这门课学习到了很多 Java 的基本语法和水文分析的基本工具原理和操作，同时增进了对流溪河模型的了解，虽然以我的掌握程度还需要日后进一步的学习，但是这门课让我受益匪浅。提前祝老师新年快乐~~

个人报告请勿转载