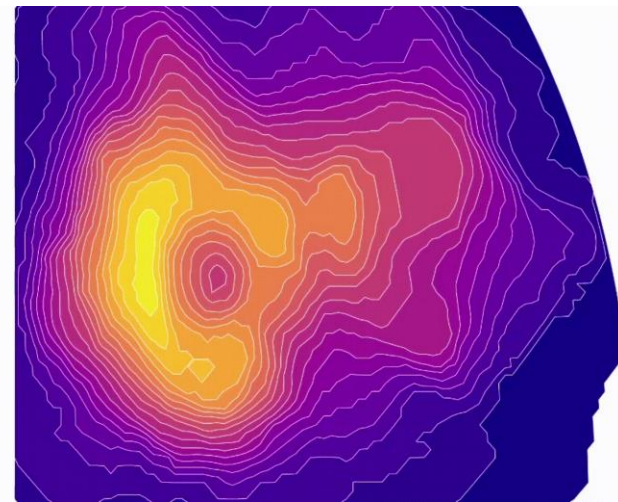
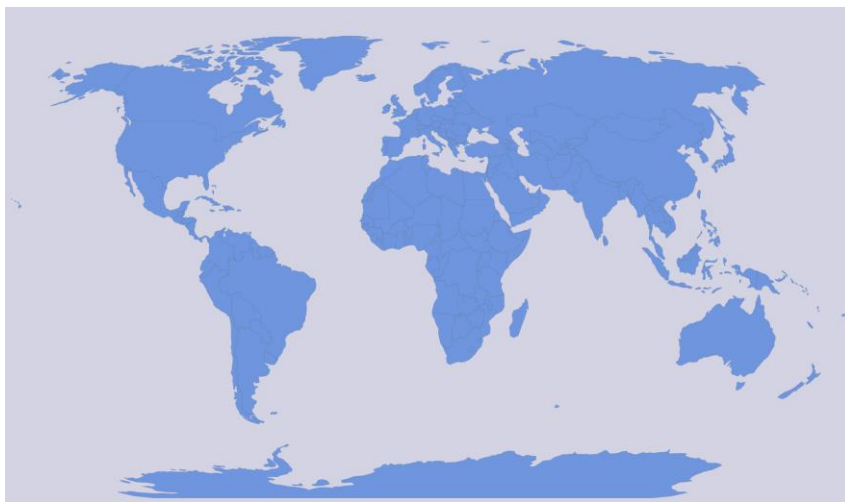


# D3.js

## 地图与地理数据可视化

张松海 张少魁

清华大学 可视媒体研究中心



# 概览

- JSON与geoJSON:
  - json的数据格式与读取。
  - geoJson的数据格式。
- d3-geo:
  - 地图数据(经纬度)到画布的映射(投影): `d3.geoPath().projection(...)`。
  - 'd'属性接口: `d3.geoPath()`。
- 基于d3-geo绘制北京市地图。
- d3-contour:
  - 等值线数据的输入与处理。
- 基于d3-contour与d3-geo绘制等值线图:
  - `d3.scaleSequential`: 基于插值的连续比例尺。

# Json

```
{  
  "firstName": "Shao-Kui",  
  "lastName": "Zhang",  
  "sex": "male",  
  "age": 25  
}
```

- Json: JavaScript Object Notation
  - 软件开发的常用数据格式，由JavaScript编程语言的核心开发者Douglas Crockford制定。
  - Json起源于JavaScript。
  - Json的数据格式本质上为纯文本，与编程语言无关。
  - 目前大量编程语言与框架均支持Json的读取与导出，工程上被广泛运用。
    - e.g., MongoDB, MineCraft, Qt, 3D-Front等。
- d3.json(...):
  - 读取一个.json文件，接受的参数为数据的路径。
  - 同d3.csv，要用.then( data => {...}) 的方式获得读取后的数据。
  - e.g., d3.json('static/data/sk.json').then( data => {...} );
  - （与d3.csv的调用方式完全一致。）

# Json

- Json的格式:

- ‘属性名’: ‘属性值’
- 属性值可以是字符串、整数、浮点数...
- 属性值可以是数组:
- 属性值可以是另一个对象:
- 属性值可以是另一个对象的数组:

```
"lastName": "Smith",  
"sex": "male",  
"age": 25,
```

```
"phoneNumber": [  
  "212 555-1234",  
  "646 555-4567"  
],
```

```
"address": {  
  "city": "New York",  
  "state": "NY",  
  "postalCode": "10021"  
},
```


```
"phoneNumber": [  
  {  
    "type": "home",  
    "number": "212 555-1234"  
  },  
  {  
    "type": "fax",  
    "number": "646 555-4567"  
  }  
]
```

# Json – 数据读取

```
d3.json('John.json').then( data => {  
  console.log(data);  
});
```

- 编程实例:

- 注意: 由于Json本身源于JavaScript对象, 因此读入后的数据结构就是原本.json文本的结构。

```
▼ Object   
  ▼ address:  
    city: "New York"  
    postalCode: "10021"  
    state: "NY"  
    streetAddress: "21 2nd Street"  
    ► __proto__: Object  
  age: 25  
  firstName: "John"  
  lastName: "Smith"  
  ▼ phoneNumber: Array(2)  
    ► 0: {type: "home", number: "212 555-1234"}  
    ► 1: {type: "fax", number: "646 555-4567"}  
    length: 2  
    ► __proto__: Array(0)  
  sex: "male"  
  ► __proto__: Object
```

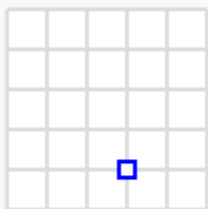
```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "sex": "male",  
  "age": 25,  
  "address": {  
    "streetAddress": "21 2nd Street",  
    "city": "New York",  
    "state": "NY",  
    "postalCode": "10021"  
  },  
  "phoneNumber": [  
    {  
      "type": "home",  
      "number": "212 555-1234"  
    },  
    {  
      "type": "fax",  
      "number": "646 555-4567"  
    }  
  ]  
}
```

# geoJson

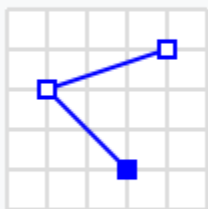
- geoJson:
  - 基于Json的数据格式，在Json的基础上进一步引入了规则与命名。
  - 记载经纬度信息，即地理信息。
  - 计算机中记载地理信息的标准格式（RFC 7946）：
    - 由2008年6月被制定并使用、于2016年被IETF(互联网标准制定组织)列入文档。
    - <https://tools.ietf.org/html/rfc7946>
- 一个geoJson包括如下内容：
  - 'type': 'FeatureCollection'
  - 'features', 一个**数组**包含若干个形状的**几何与形状的属性**:
    - type: 'Feature',
    - geometry: **几何**信息（下页），是一个对象。
    - properties: **属性**，是一个对象，如包含邮政编码、地区名称、人口、气候、栖息物种等。
    - 注意：properties通常为编程者所使用，其内部对象没有固定的规则；而geometry由D3.js使用，根据其几何绘制形状。

# geoJson

- <https://en.wikipedia.org/wiki/GeoJSON>



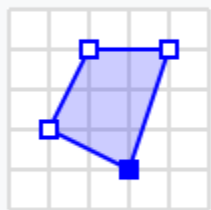
```
{  
  "type": "Point",  
  "coordinates": [30, 10]  
}
```



```
{  
  "type": "LineString",  
  "coordinates": [  
    [30, 10], [10, 30], [40, 40]  
  ]  
}
```

# geoJson

- <https://en.wikipedia.org/wiki/GeoJSON>

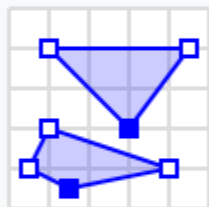


```
{  
  "type": "Polygon",  
  "coordinates": [  
    [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]]  
  ]  
}
```



# geoJson

- <https://en.wikipedia.org/wiki/GeoJSON>



```
{  
  "type": "MultiPolygon",  
  "coordinates": [  
    [  
      [[30, 20], [45, 40], [10, 40], [30, 20]]  
    ],  
    [  
      [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]]  
    ]  
  ]  
}
```

# geoJson

- 编程实例:

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "adcode": 110101,
        "name": "东城区"
      },
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [ ...
      ]
    },
    {
      "type": "Feature",
      "properties": {
        "adcode": 110102,
        "name": "西城区",
```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "adcode": 110101,
        "name": "东城区"
      },
      "geometry": {
        "type": "MultiPolygon",
        "coordinates": [
          [
            [
              116.38059,
              39.871148
            ],
            [
              116.399097,
              39.872205
            ],
            [
```

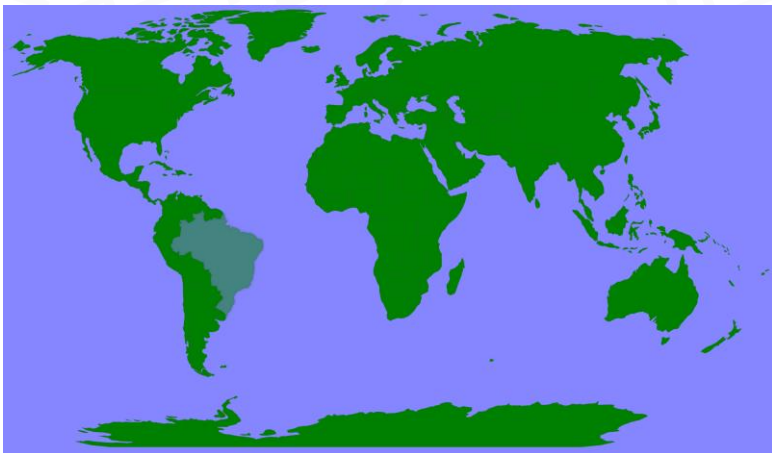
# d3-geo-projection

- geoJson中的几何数据(geometries)为经纬度:
  - 如何将地形的数据映射到画布上?
  - 需要一个‘比例尺’将经纬度映射到画布上的某个点, 即经纬度到画布的投影。
- **d3.geoNaturalEarth1()**:
  - 定义一个投影函数, 类比例尺的作用。
  - e.g., let proj = d3.geoNaturalEarth1();
- 返回的投影函数:
  - 输入: geoJson的一个点, 含经纬度, e.g., [116.418757, 39.917544]。
  - 输出: 画布的一个点, 含横纵坐标, e.g., [834.4, 565.1]。
  - e.g., proj([116.418757, 39.917544]) // 画布空间上的一点, 即[834.4, 565.1]。

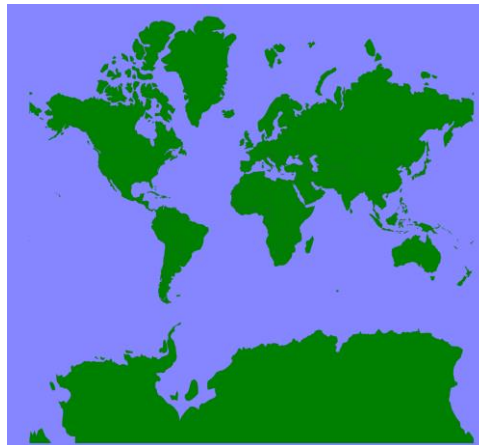
# d3-geo-projection

- d3.geoNaturalEarth1并不是唯一的投影方式，还有更多的接口。
- d3-geo-projection作为D3.js相对独立的模块，提供多种投影函数供编程者选择：
  - <https://github.com/d3/d3-geo-projection>
- e.g., 下述得到的proj均为投影函数，区别在于投影的方式：
  - let proj = d3.geoNaturalEarth1();
  - let proj = d3.geoMercator();

d3.geoNaturalEarth1()



d3.geoMercator()



d3.geoStereographic()



# Azimuthal Projections



d3.geoAzimuthalEquidistant: 方位角等距离投影



d3.geoStereographic: 球极平面投影（等角）



d3.geoOrthographic: 方位角正交投影



d3.geoAzimuthalEqualArea(): **Lambert**等面积方位角投影

# Conic Projections



d3.geoConicEqualArea: 亚尔勃斯等面积投影



d3.geoConicEquidistant: 圆锥等距投影



d3.geoHill(): 伪圆锥等面积投影



d3.geoConicConformal: **Lambert**圆锥等角投影

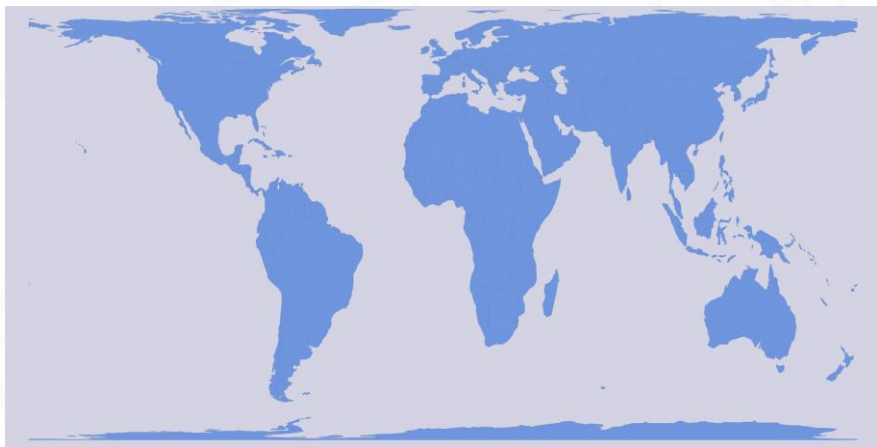
# Cylindrical Projections



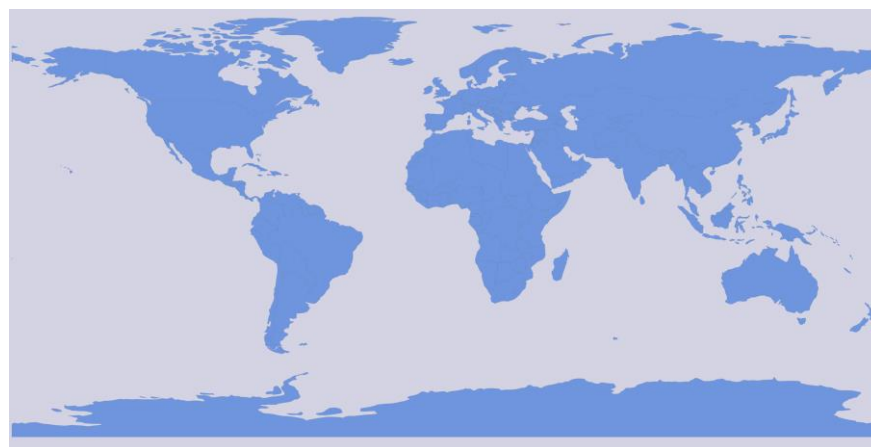
d3.geoCraster: 正弦曲线等面积伪圆柱投影



d3.geoMercator: 墨卡托投影



d3.geoCylindricalEqualArea: **Lambert**等面积圆柱投影



d3.geoEquirectangular: **Plate Carrée**投影 (等距)



# d3-geo-projection

- 投影函数同样需要‘定义域’与‘值域’。
- **proj.fitSize([width, height], geoJson):**
  - 类似“比例尺”，地图要画在多大的画布上？
  - 接受两个输入：画布的[宽, 高]与读取的geoJson(地图数据)。
  - e.g., proj.fitSize([1600, 800], data);
- 编程实例：

```
const projection = d3.geoNaturalEarth1();  
projection.fitSize([width, height], data);  
console.log(projection([116.418757, 39.917544]));
```



# d3.geoPath()

```
M713.617444890755,527.4596950259147L711.7481099475117,529.5650279344991L702.1988267574416,531.084182 beijing.html:26
0969617L700.959112806253,533.3582296345157L704.1245689653588,538.1003001950485L707.5222656879487,540.2626046974583L70
7.9100874674259,543.4143263033329L706.4463945516109,545.1185763067915L709.7350443978212,548.559758837946L707.87425892
69823,549.3478332859922L708.1183366309706,552.4521660434402L713.2883897515931,557.478538264193L714.7907818868334,556.
4168579236393L716.9139406264294,558.6772623989891L717.4712138522591,560.5191420993324L719.0570201489609,561.556596263
3336L723.095420595484,562.7553380877762L728.4975946518025,568.7011347291082L731.5360338477694,570.4070250680234L732.9
49233192543,573.1994041885191L736.786728896499,575.2674848074057L740.7483685524567,579.2517993577421L739.669272481049
1,580.1775063519781L736.3879339106643,579.1147494662691L735.3454878956836,581.577699702073L733.7673763954008,580.9661
656256321L728.277222177072,582.9546458362347L726.6619436484762,584.5967978407389L724.3191230749217,584.8303315435041L
722.2085910213646,586.9232376712898L719.6553435585156,587.2695829926452L719.8944634363652,589.105220388501L716.566792
5253802,589.0087377982309L713.7230811227128,590.4337148370832L709.3933803454638,588.3724483219994L706.6614763134494,5
89.2986804522516L705.9679780093575,590.6271758233815L708.1606311997675,593.2683522847728L709.8169918713975,596.586433
7007042L704.6692296526453,597.9555372476134L704.9173236813076,600.4280553124809L701.1564669691506,599.7516608633996L7
00.05854070637,600.9599687031987L696.4070316686921,600.4523006450145L695.7966578326741,601.6606115147042L692.68764670
94189,603.0460685720864L691.4654455884665,602.6190501827259L689.2639309527149,598.8090656246204L685.7361566652762,598
.6883344957714L681.0240274360258,603.2637838767914L676.4781224260732,598.8654728092588L675.8695008287177,593.55829937
```

- d3.geoPath():
  - <path>的'd'属性接口，输入输出类似d3.line、d3.arc。
  - 返回一个函数，输入geoJson的一个'**Feature**'，输出<path>的'd'属性。
  - e.g., let path = d3.geoPath();
  - path(geoJson.features[0]) // M713.617444890755,527.4596... ..
- path.projection(...):
  - d3.geoPath只负责生成'd'属性...
  - 设置geoJson中经纬度坐标到画布坐标的投影方式。
  - e.g., let proj = d3.geoNaturalEarth1().fitSize([1600, 800], data);
  - e.g., let path = d3.geoPath().projection(proj);
- 调用实例：基于d3-geo绘制北京地图。

# 基于d3-geo绘制地图

- 任务：北京市及其各个区的地图数据可视化。
- 数据来源：
  - <http://datav.aliyun.com/tools/atlas/>



# 基于d3-geo绘制地图

- 编程实例:

```
const svg = d3.select('svg');
const width = svg.attr('width');
const height = svg.attr('height');

d3.json('beijing.json').then(async data => {
  const projection = d3.geoNaturalEarth1();
  // const projection = d3.geoMercator();
  projection.fitSize([width, height], data);
  const path = d3.geoPath().projection(projection);
  svg.selectAll('path').data(data.features).join('path')
    .attr('stroke', 'black').attr('fill', 'none')
    .attr('d', path)
    .attr('id', d => d.properties.name);
});
```

# Tip: 如何为地图添加标签?

*PlanA与PlanB的代码均以注释的形式包含在beijing.html中。*

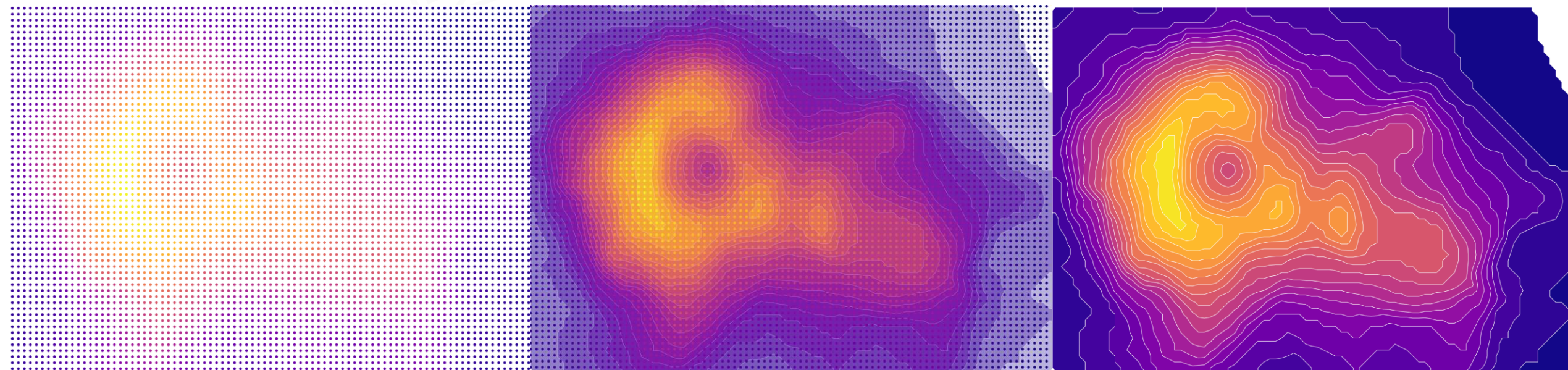
- PlanA:
  - 基于数据提供的中心(centroid)。
  - `.attr('transform', d => `translate(${projection([d.properties.centroid[0], d.properties.centroid[1]]))`)`
- PlanB:
  - 基于<path>勾勒中的某一个点。
  - `.attr('transform', d => {  
 let p = d3.select(`#${d.properties.name}`).node().getPointAtLength(0);  
 return `translate(${p.x}, ${p.y})`;`  
 })
- PlanC:
  - 基于交互。



# d3-contour

9	13	5	2
1	11	7	6
3	7	4	1
6	0	7	10

- D3.js中接受的等值线的数据类型为‘**矩阵**’。
  - 包括： 行数、列数与每个元素(element)的值(value)。
- D3.js的‘contour’模块做了什么？
  - Marching Square： 在输入‘矩阵’的元素(左)之间形成轮廓(中)将值(value)相近的元素包围在一起， 最终(拟合)得到每个轮廓的几何(右)。





# 等值线数据的输入与处理

- 轮廓线输入的数据格式：
  - 是JSON。
  - 矩阵的宽: `width`。
  - 矩阵的高: `height`。
  - 矩阵的值: 数值的数组（一阶）。
  - 行主序: 即 $(i, j)$ 表示数组中的第 $(i + j \times width)$ 个元素。

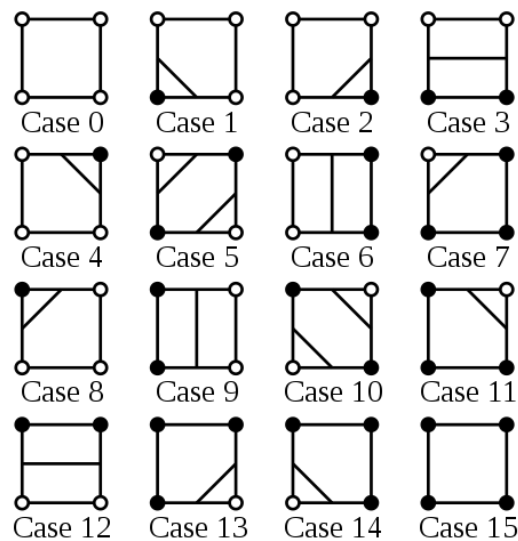
```
{
  "width": 87,
  "height": 61,
  "values": [
    103,
    104,
    104,
    105,
    105,
    106,
    106,
    106,
    107,
    107,
    106,
    106,
    105,
    105,
    104,
    104,
    104,
    104,
```

# 等值线数据的输入与处理



```
{
  "type": "MultiPolygon",
  "coordinates": [
    [
      [[30, 20], [45, 40], [10, 40], [30, 20]]
    ],
    [
      [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]]
    ]
  ]
}
```

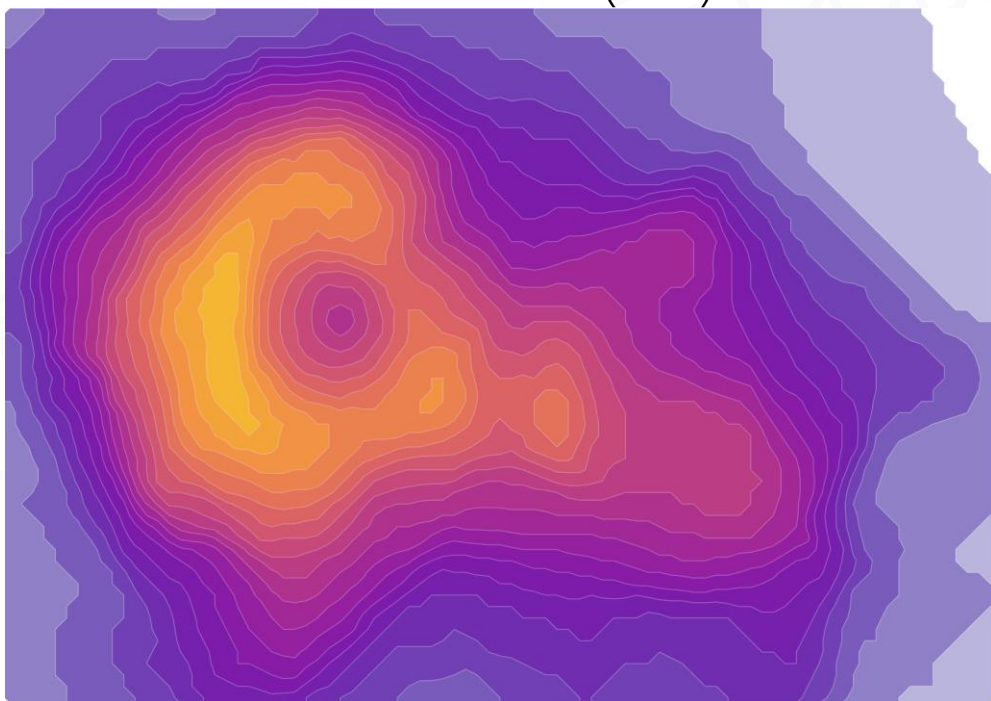
- `d3.contours()`:
  - 返回一个等值线生成函数，用于将矩阵转换为几何。
  - 函数的输入：数组，包含矩阵的全部元素，是一个数组，e.g., `[12,6,8,10]`;
  - 函数的输出：geoJson的geometry对象，e.g., ;
  - e.g., `let contour = d3.contours();` //定义轮廓线生成函数。
  - e.g., `const geoPolygons = contours(data.values);` // 返回geoJson。
- `contour.size([width, height])`:
  - 设置矩阵的行数与列数 (`[n, m]`)。
  - e.g., `contour.size([87, 61]);`
- `contour.thresholds( array )`:
  - 设置轮廓的阈值。
  - 阈值的数量等于轮廓的数量，即每个阈值会对应一个最终的轮廓。
  - 值(value)大于等于某一个阈值的元素会被包含在这个阈值的轮廓。
  - 相邻却属于不同阈值(轮廓)的元素间，使用marching squares分界。
  - e.g., `contour.thresholds([95, 100, 105, 110]);`



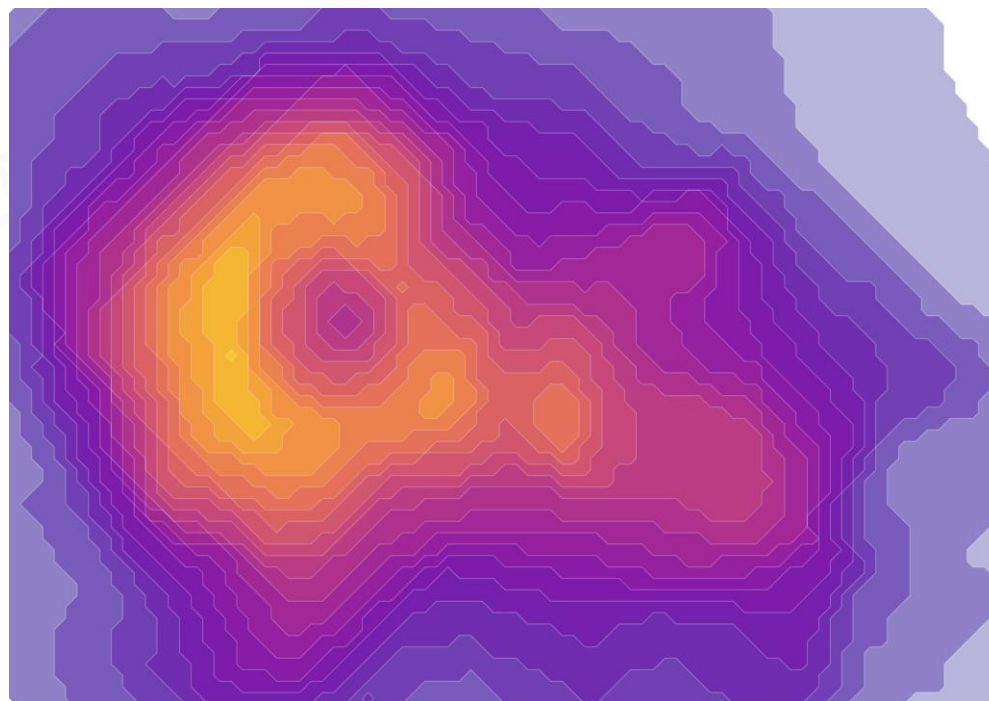
# 等值线数据的输入与处理

- `contour.smooth(true/false)`:
  - 是否平滑轮廓?
  - 由于输入数据为一个矩阵, 轮廓本质上是‘水平、竖直、斜’三种移动方式。

`contour.smooth(true)`



`contour.smooth(false)`





# d3-contour

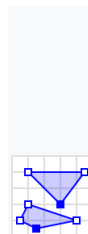
- 调用实例:

```
const contours = d3.contours()  
.size([data.width, data.height]).thresholds(thresholds).smooth(false);  
const geoPolygons = contours(data.values);
```

# d3-contour – 轮廓geoJson的投影

- d3.contour()返回的结果只是geoJson的几何数据：
  - 仍然需要投影(d3-geo-projection)做‘比例尺’映射。
  - 仍然需要d3.geoPath将geoJson转换为<path>的‘d’属性。
- 将d3.contour()导出的轮廓数据(geoJson)调整为画布的尺寸：
  - d3.contour()返回函数的输出不是一个完整的geoJson。
  - d3.contour()返回函数输出的是geoJson的几何(geometry)。
  - 若需要用d3-geo-projection做投影， 需要把geoJson补全：

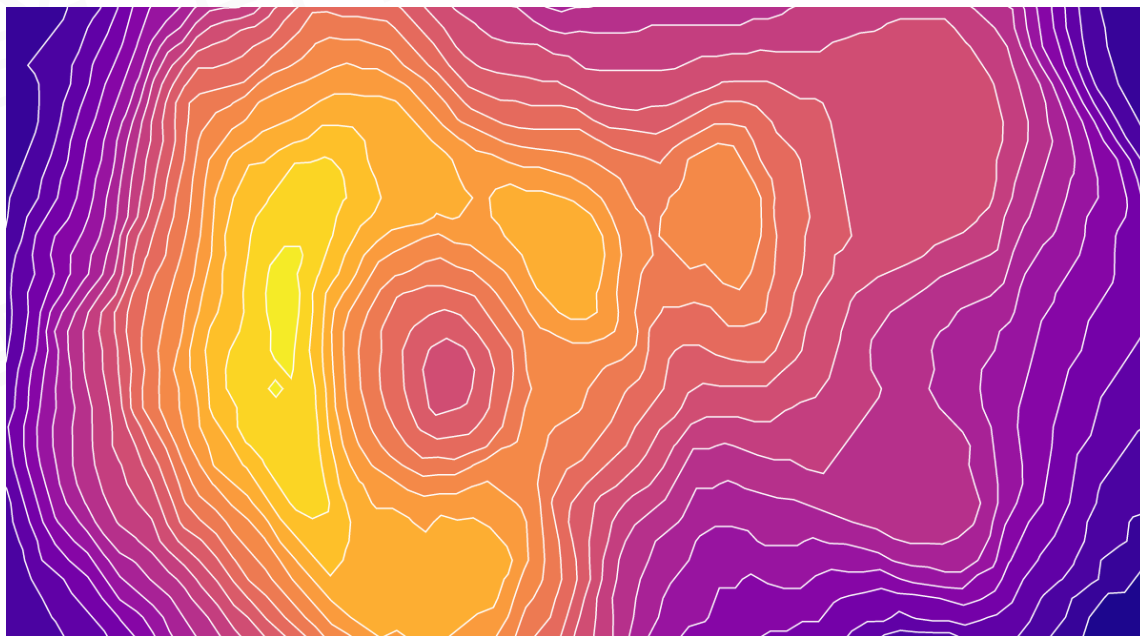
```
// 构造GeoJson并FitSize。  
const geoJson = { "type": "FeatureCollection", "features": [] };  
const path = d3.geoPath();  
const projection = d3.geoNaturalEarth1();  
for (const geoPolygon of geoPolygons) {  
  geoJson.features.push({ 'geometry': geoPolygon });  
}  
projection.fitSize([width, height], geoJson);  
path.projection(projection);
```



```
{  
  "type": "MultiPolygon",  
  "coordinates": [  
    [  
      [[30, 20], [45, 40], [10, 40], [30, 20]]  
    ],  
    [  
      [[15, 5], [40, 10], [10, 20], [5, 10], [15, 5]]  
    ]  
  ]  
}
```

# 基于d3-contour与d3-geo绘制等值线图

- 任务：火山地区温度可视化。
- 数据来源：
  - <https://observablehq.com/@d3/volcano-contours>
  - [https://en.wikipedia.org/wiki/Maungawhau\\_Mount\\_Eden](https://en.wikipedia.org/wiki/Maungawhau_Mount_Eden)



# 基于插值的连续比例尺

- `d3.scaleSequential( interpolator ).domain([min, max])`
  - 返回一个函数，将连续区间的某一个值映射到 插值函数 的某一个值。
  - 输入的插值器的范围是[0, 1]，定义域会自动映射到这个[0, 1]区间。
  - 映射到[0, 1]后的值，会被输入到插值函数。
  - 插值函数接受输入后会返回比例尺的结果。
- `d3.scaleSequential`与`d3.scaleLinear`的区别？
  - 通常可以互换，但对于已有插值函数的情况，`scaleSequential`更便捷。
  - `scaleSequential`更强调插值，故定义域之外的输入会返回插值的端点值。
  - `scaleSequential`的定义域与值域的设置均为一个数组，e.g., [min, max]。
  - `scaleSequential`的值域通常由插值函数代替。
- 调用实例：

```
let color = d3.scaleSequential(d3.interpolatePlasma).domain(d3.extent(data.values));
```

# 基于d3-contour与d3-geo绘制等值线图

- 编程实例:

```
// 构造等值线生成函数;
const contours = d3.contours().size([data.width, data.height]).thresholds(thresholds).smooth(true);
const geoPolygons = contours(data.values);
console.log(geoPolygons);

// 构造GeoJson并FitSize。
const geoJson = { "type": "FeatureCollection", "features": [] };
const path = d3.geoPath();
const projection = d3.geoNaturalEarth1();
for (const geoPolygon of geoPolygons) {
    geoJson.features.push({ 'geometry': geoPolygon });
}
projection.fitSize([width, height], geoJson);
path.projection(projection);

// Data-Join;
svg.selectAll('.myPath').data(geoPolygons).join('path')
.attr('class', 'myPath')
.attr("fill", d => color(d.value))
.attr("stroke", "white")
.attr("stroke-width", 1)
.attr('d', d => path(d));
```

# Tip: geoJson与topoJson?

- TopoJson

```
{
  "type": "Topology",
  "objects": {
    "countries": {
      "type": "GeometryCollection",
      "geometries": [
        {
          "type": "MultiPolygon",
          "arcs": [
            [
              [
                0
              ]
            ],
            [
              [
                1
              ]
            ]
          ],
          "id": "242",
          "properties": {
            "name": "Fiji"
          }
        }
      ]
    }
  }
}
```

## GeoJson

```
▼ {type: "FeatureCollection", features: Array(177)} ⓘ
  type: "FeatureCollection"
  ▼ features: Array(177)
    ▼ [0 ... 99]
      ► 0: {type: "Feature", id: "242", properties: {...}, geometry: {...}}
      ► 1: {type: "Feature", id: "834", properties: {...}, geometry: {...}}
      ► 2: {type: "Feature", id: "732", properties: {...}, geometry: {...}}
      ▼ 3:
        type: "Feature"
        id: "124"
        ▼ properties:
          name: "Canada"
          ► __proto__: Object
          ► geometry: {type: "MultiPolygon", coordinates: Array(30)}
          ► __proto__: Object
      ► 4: {type: "Feature", id: "840", properties: {...}, geometry: {...}}
      ► 5: {type: "Feature", id: "398", properties: {...}, geometry: {...}}
      ► 6: {type: "Feature", id: "860", properties: {...}, geometry: {...}}
      ► 7: {type: "Feature", id: "598", properties: {...}, geometry: {...}}
      ► 8: {type: "Feature", id: "360", properties: {...}, geometry: {...}}
```

# Tip: geoJson与topoJson?

## TopoJson

- 本质上是JSON格式。
- 对处理了GeoJson数据冗余的特点, 节约存储空间。
- 由D3的作者Mike Bostock制定。
- D3的geoPath使用GeoJson的格式, 因此需要转换:
  - <https://github.com/topojson/topojson>

## GeoJson

- 本质上是JSON格式。
- 官方: GeoJSON is a format for encoding a variety of geographic data structures。
- D3.js的geoPath使用GeoJson格式的地图数据:
- <https://www.jianshu.com/p/465702337744>

```
// convert topo-json to geo-json;  
worldmeta = topojson.feature(data, data.objects.countries);
```