27. Means and Medians

Three Instructive Problems:
- The Apportionment Problem
- The Polygon Averaging Problem
- The Median Filtering Problem

What?

The Apportionment Problem
How to fairly distribute 435 Congressional Districts among the 50 states.

<table>
<thead>
<tr>
<th>State</th>
<th>Pop</th>
<th>nDist</th>
<th>Pop/nDist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>4802982</td>
<td>7</td>
<td>686140</td>
</tr>
<tr>
<td>Alaska</td>
<td>721523</td>
<td>1</td>
<td>721523</td>
</tr>
<tr>
<td>Arizona</td>
<td>6412700</td>
<td>9</td>
<td>712522</td>
</tr>
<tr>
<td>Arkansas</td>
<td>2926229</td>
<td>4</td>
<td>731557</td>
</tr>
<tr>
<td>California</td>
<td>37541989</td>
<td>53</td>
<td>708339</td>
</tr>
<tr>
<td>Colorado</td>
<td>5044930</td>
<td>7</td>
<td>720704</td>
</tr>
<tr>
<td>Connecticut</td>
<td>3581628</td>
<td>5</td>
<td>716325</td>
</tr>
<tr>
<td>Delaware</td>
<td>900877</td>
<td>1</td>
<td>900877</td>
</tr>
<tr>
<td>Florida</td>
<td>18900773</td>
<td>27</td>
<td>700028</td>
</tr>
</tbody>
</table>

What?

The Polygon Averaging Problem
Given a polygon, connect the midpoints of the sides. This gives a new polygon. Repeat many times.

A Random Pentagon

The Side Midpoints
Connect the Midpoints

The Polygon Untangles Itself and Heads Towards an Ellipse

What?

The Median Filtering Problem

Visit each pixel in a picture and replace its value by the median value of its "neighbors".

A Picture With Dirt Specks

After Median Filtering is Applied

Why?

Each Problem has a couple of Python "nuggets" to practice with.

Each problem has something to "say" about averaging.

Each problem has a high-level "message"

A Nice Way to Wrap UP
The Apportionment Problem

How do you distribute 435 Congressional seats among the 50 states so that the ratio of population to delegation size is roughly the same from state to state?

Quite possibly one of the greatest division problems of all time!

Notation

Number of states: \( n \)

State populations: \( p[0], \ldots, p[n-1] \)

State delegation size: \( d[0], \ldots, d[n-1] \)

Total Population: \( P \)

Total number of seats: \( D \)

Ideal: Equal Representation

\[
\frac{P}{D} = \frac{p[0]}{d[0]} = \ldots = \frac{p[n]}{d[n]}
\]

More Realistic...

\[
\frac{P}{D} \approx \frac{p[0]}{d[0]} \approx \ldots \approx \frac{p[n]}{d[n]}
\]

But delegation size must be a whole number!!!
**Definition**

An **apportionment method** determines delegation sizes $d[0],...,d[49]$ that are whole numbers so that representation is approximately equal:

$$\frac{p[0]}{d[0]} \approx ... \approx \frac{p[49]}{d[49]}$$

**How it Is Done**

Think in terms of dealing cards.

You are the dealer.

You have 435 cards to deal to 50 people.

**At the Start**

Everybody gets one card...

```python
N = 435
d = [
for k in range(50):
    d.append(1)
    N = N - 1
```

Every state has at least one congressional district

**Dealing out the Rest...**

```python
while N > 0:
    k = the index of that state which is most deserving of an additional district.
    # Increase that state's delegation
    d[k] += 1
    # Decrease what's left to deal
    N = N - 1
```

**The Method of Small Divisors**

At this point in the "card game" deal a district to the state having the largest quotient

$$\frac{p[k]}{d[k]}$$

**Implementation**

```python
def smallDivisor(p,d):
    """returns an int j with the property that p[j]/d[j] is max."
    m = 0
    for k in range(50):
        if p[k]/d[k] >= m:
            m = p[k]/d[k]
            j = k
    return j
```

This is the old "look for a max" problem
Dealing out the Rest...

while $N > 0$:
  $k = \text{smallDivisors}(p,d)$
  # Increase that state's delegation
  $d[k] += 1$
  # Decrease what's left to deal
  $N = N - 1$

Several reasonable definitions of "most deserving."

The Method of Large Divisors

At this point in the "card game" deal a district to the state having the largest quotient

$$\frac{p(k)}{d(k) + 1}$$

Tends to favor small states.

Dealing out the Rest...

while $N > 0$:
  $k = \text{largeDivisors}(p,d)$
  # Increase that state's delegation
  $d[k] += 1$
  # Decrease what's left to deal
  $N = N - 1$

The Method of Major Fractions

At this point in the "card game" deal a district to the state having the largest value of

$$\frac{1}{2} \left( \frac{p(k)}{d(k)} + \frac{p(k)}{d(k) + 1} \right)$$

Several reasonable definitions of "most deserving."

Dealing out the Rest...

while $N > 0$:
  $k = \text{majorFractions}(p,d)$
  # Increase that state's delegation
  $d[k] += 1$
  # Decrease what's left to deal
  $N = N - 1$

The Method of Equal Proportions

At this point in the "card game" deal a district to the state having the largest value of

$$\sqrt[2]{\frac{p(k) * p(k)}{d(k) * (d(k) + 1)}}$$

This method is in use today.

Compromise via the Geometric Mean
Dealing out the Rest...

```
while N > 0:
    k = equalProportions(p,d)
    # Increase that state's delegation
    d[k] += 1
    # Decrease what's left to deal
    N = N-1
```

Four Different Ways to Compute "Most Deserving"

\[
\frac{p(k)}{d(k)} \quad \frac{p(k)}{d(k)+1}
\]

\[
\frac{1}{2} \left( \frac{p(k)}{d(k)} + \frac{p(k)}{d(k)+1} \right) \quad \sqrt{\frac{p(k)}{d(k)}} \cdot \frac{p(k)}{d(k)+1}
\]

And two different ways to compute an average

Takeaway: There is a Subjective Component to Math+Computing

One can design more equitable methods for apportionment, but they are complicated and cannot be "sold" to the lay public.

Another Division Problem

Gerrymandering:
The Art of drawing district boundaries so as to favor incumbents

Polygon Averaging

Connect the Midpoints
A Useful Class

class polygon:
    def __init__(self,x,y):
        self.x = x
        self.y = y

    x and y are numpy arrays that name
    the vertices of the polygon:
    (x[0],y[0]),…,(x[n-1],y[n-1])

The New Polygon

def newPoly(self):
    n = len(self.x); x = zeros(n); y = zeros(n)
    for k in range(n):
        # Get the next midpoint.
        j = (k+1)%n
        x[k] = (self.x[k]+self.x[j])/2
        y[k] = (self.y[k]+self.y[j])/2
    self.x = x
    self.y = y

Order From Chaos

1. Pick n, say n= 30
2. Generate random lists of floats x and y
3. P = polygon(x,y)
4. Then repeatedly replace P by with a new
   polygon obtained by connecting midpoints:

   for i in range(200):
       P.newPolygon()
       P.plotPoly()
Pictures as Arrays

A black and white picture can be encoded as a 2-dimensional array of numbers.

Typical:

\[ 0 \leq A[i,j] \leq 255 \]

(black) \quad (white)

Values in between correspond to different levels of grayness.

Just a Bunch of Numbers

1458-by-2084

150 149 152 153 152 155
151 150 153 154 153 156
153 152 153 154 155 158
154 153 156 157 156 159
156 154 158 159 158 161
157 156 159 160 159 162

Dirt!

1458-by-2084

Note how the "dirty pixels" look out of place.

Can We Filter Out the "Noise"?

1458-by-2084

Can We Filter Out the "Noise"?

1458-by-2084

Note how the "dirty pixels" look out of place.

Idea

1458-by-2084

Assign "typical" neighborhood gray values to "dirty pixels".

Getting Precise

"Typical neighborhood gray values"

Could use Median Or Mean

radius 1

radius 3

We'll look at "Median Filtering" first...
Median Filtering

Visit each pixel.
Replace its gray value by the median of the gray values in the “neighborhood”.

Using a radius 1 “Neighborhood”

How to Visit Every Pixel

for \( i \) in range(m):
    for \( j \) in range(n):
        Compute new gray value for pixel \((i,j)\).

Original:

Filtered:

Replace \( \times \) with the median of the values under the window.

Replace \( \times \) with the median of the values under the window.
What We Need...

(1) A function that computes the median value in a 2-dimensional array $C$:
$$m = \text{medVal}(C)$$

(2) A function that builds the filtered image by using median values of radius $r$ neighborhoods:
$$B = \text{medFilter}(A, r)$$

Medians vs Means

$$A =
\begin{array}{cccccc}
150 & 151 & 158 & 159 & 156 \\
153 & 151 & 156 & 155 & 151 \\
150 & 155 & \textbf{152} & 154 & 159 \\
156 & 154 & 152 & 158 & 152 \\
152 & 158 & 157 & 150 & 157 \\
\end{array}
$$

Median = 154  Mean = 154.2
**Medians vs Means**

A =

150 151 158 159 156  
153 151 156 155 151  
156 154 152 158 152  
152 158 157 150 157  

Median = 154  Mean = 148.2

**Back to Filtering...**

m = 9  
n = 18

for i in range(m):
    for j in range(n):
        Compute new gray value for pixel (i,j).
    end
end

**B = medFilter(A)**

**Original**

What About Using the Mean instead of the Median?

Replace each gray value with the average gray value in the radius r neighborhood.

**Mean Filter with r = 3**
Mean Filter with \( r = 10 \)

And Median Filters Leave Edges (Pretty Much) Alone

<table>
<thead>
<tr>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Why it Fails

<table>
<thead>
<tr>
<th>150</th>
<th>149</th>
<th>152</th>
<th>153</th>
<th>152</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>151</td>
<td>150</td>
<td>153</td>
<td>154</td>
<td>153</td>
<td>156</td>
</tr>
<tr>
<td>153</td>
<td>2</td>
<td>3</td>
<td>156</td>
<td>155</td>
<td>158</td>
</tr>
<tr>
<td>154</td>
<td>2</td>
<td>1</td>
<td>157</td>
<td>156</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>158</td>
<td>159</td>
<td>158</td>
<td>161</td>
</tr>
<tr>
<td>157</td>
<td>156</td>
<td>159</td>
<td>160</td>
<td>159</td>
<td>162</td>
</tr>
</tbody>
</table>

The mean does not capture representative values.

Takeaways

Image processing is all about operations on 2-dimensional arrays.

Simple operations on small patches are typically repeated again and again.

There is a profound difference between the median and the mean when filtering noise.