Hodgepodge

“Clumsy mixture of ingredients…”

---

CS112 Scientific Computation
Department of Computer Science
Wellesley College

---

The return of Peter Piper

function peterPiper (numReps)
% peterPiper(numReps)
% repeats a tongue twister “numReps” times

for count = 1:numReps
    disp('Peter Piper picked a peck of pickled peppers');
end

Can we make the numReps input optional?
Optional input arguments

Inside a user-defined function, nargin returns the number of inputs entered when the function was called.

```matlab
function peterPiper (numReps)
  % repeats a tongue twister multiple times
  % numReps input is optional
  if (nargin == 0)
    numReps = 5;
  end
  for count = 1:numReps
    disp('Peter Piper picked a peck of pickled peppers');
  end
```

A third form of the if statement

```matlab
if (cond1)
  ...
elseif (cond2)
  ...
elseif (cond3)
  ...
else
  ...
end
```
The elseif clause in action

```matlab
function drawCircle (radius, xcenter, ycenter, properties, width)
% drawCircle(radius, xcenter, ycenter, properties, width)
% draws a circle with specified radius, centered on (xcenter, ycenter)
% with the (optional) properties and width

t = linspace(0, 2*pi, 50);
xcoords = xcenter + radius * cos(t);
ycoords = ycenter + radius * sin(t);
if (nargin == 3)
    plot(xcoords, ycoords, 'b', 'LineWidth', 1);
elseif (nargin == 4)
    plot(xcoords, ycoords, properties, 'LineWidth', 1);
else
    plot(xcoords, ycoords, properties, 'LineWidth', width);
end
axis equal
```

Looping through a 2-D matrix

```matlab
count = 0;
for row = 1:5
    for col = 1:5
        if (nums(row,col) ~= 0)
            count = count + 1;
        end
    end
end
```

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>
```

nums

```
But why bother with nested loops here?!?

count = sum(sum(nums ~= 0))
```
Counting peaks

A peak is a value that is larger than its 4 neighbors*

two peaks

* Don’t bother checking locations around the border of the matrix

How many peaks?

function numPeaks = countPeaks (matrix)
% counts the number of peaks in a matrix of numbers, where
% a peak is a value that is larger than its 4 neighbors
[rows cols] = size(matrix);
numPeaks = 0;
for row = 2:rows-1
    for col = 2:cols-1
        val = matrix(row, col);
        if (val > matrix(row-1, col)) & ...
            (val > matrix(row+1, col)) & ...
            (val > matrix(row, col+1)) & ...
            (val > matrix(row, col-1))
            numPeaks = numPeaks + 1;
        end
    end
end
Simulating population growth

Goal: define a function that generates a figure with curves for different rates of population growth over multiple generations, using the logistic growth model for population growth:

\[ p_{t+1} = r \cdot p_t \cdot (K - p_t)/K \]

- \(p_t\): current population
- \(p_{t+1}\): population in the next generation
- \(r\): growth rate
- \(K\): carrying capacity

Guidelines & tips

Define a function named `popGrowth` with four inputs:
- vector of growth rates to simulate (default [1.2 1.4 1.6 1.8 2.0])
- initial population (default 2)
- number of generations (default 25)
- carrying capacity (default 1000)

For each growth rate:
- create a vector to store the populations for each generation and store the initial population in the first location of the vector
- for each new generation, apply the formula to calculate the new population size and store it in the vector
- plot the populations for this growth rate

Add figure embellishments at the end