W1005
Intro to CS and Programming in MATLAB

Project Example

Instructor: Ilia Vovsha

http://www.cs.columbia.edu/~vovsha/w1005

The MATLAB Heist

- Suppose you are trying to steal a valuable object from a room. The room is protected by a laser grid which functions as a cloak of invisibility. Each laser dot is connected to every other dot, and each triple of dots ensures that the triangle defined by them is invisible
- You have a map which specifies the location of each laser dot in the room. Your goal is to turn off parts of the grid until the object becomes visible. To turn off a part, you need to "unravel" each dot in the triangle

The MATLAB Heist

- However, unraveling some dots requires more time than others. This is reflected by a value (in seconds) attached to each dot.
- Moreover, you need to unravel a given dot each time you wish to turn off a triangle for which it is a vertex
- Not to be misled, you should check whether the area you are searching is indeed visible
- You can stop when the object is found. That is, you may not need to turn off every triangle

Heist – Problem Formulation

- More specifically:
  - The room is defined by the 2D square [0.0,1.0],[0.0,1.0]
  - The grid is defined by a set of N laser dots ($N \geq 3$)
  - Each dot 'n', requires C(n) seconds to unravel
  - The location of each dot 'n' is specified by two coordinates $\{x_n, y_n\}$ Both coordinates are in the range [0.0,1.0]
  - To visualize the grid, we can connect every pair of dots with a straight line
- Given the complete grid, how do you choose which parts of the grid to turn off? How much time does it take to find the object?

Heist – Problem Formulation

- Assumptions:
  - M is an N-by-2 ($N \geq 3$) matrix specifying the grid
  - Columns {1,2} of M are the {x,y} coordinates of the dots respectively
  - C (time) is an N-by-1 (column) vector of positive real values
  - M consists of real values in the range [0.0,1.0]
- All the relevant info is given. That is, {M,N,C} must be known
- Note: no guarantee that the parameters are set correctly
- Note: there is no a priori information about the location of the object in the room

1st Step – Approach

- Approach:
  1. Collect and verify all parameters {M,N,C}
  2. Generate some plots: grid (2D), grid vs. time (3D), progress (2D triangles)
  3. Decide on a search strategy:
     I. Determine whether the starting point is in the convex hull for the grid.
     II. Choose which triangle to turn off (e.g. minimize effort or maximize triangle area)
     III. Check whether the area you are searching is visible
     IV. Update your strategy if necessary and keep track of progress
  4. Evaluate your strategy in hindsight

Convex Set

- *Definition:* a set of points C is convex, if the line segment between any two points in C lies in C
- Formally, if for any {x,y} in C and any $0 \leq w \leq 1$,
  $wx + (1-w)y$ is in C, then C is convex
- Implicitly assuming that C is a subset of the n-dimensional real space
- We can generalize the definition to hold for more than two points

Convex Hull

- *Definition:* we call a point 'p' of the form $p = w_1x_1 + w_2x_2 + \ldots + w_kx_k$, where $w_1 + w_2 + \ldots + w_k = 1$ and $w_i \geq 0$ ($\forall$ i), a *convex combination* of the points $\{x_1, \ldots, x_k\}$.
- *Convex combination*: weighted average of the points
- *Definition:* the set of all convex combinations of points in C is called the *convex hull* of C
- *Convex hull:* The smallest convex set that contains C
- These notions can be applied to infinite sums and non-Euclidean spaces as well

2nd Step – Concrete Tasks

- Problems & relevant functionality:
  1. Generate plots: grid (plot, triplot), grid vs. time (plot3)
  2. Compute convex hull: use grid data (convhull)
  3. Choose a triangle to turn off: maximize area (polyarea) or minimize effort by recording every triangle and value and then sorting (sortrows)
  4. Check if area is visible: verify whether a given location is inside a triangle (inpolygon)
  5. Keep track of progress: mark visible triangles (fill)

2^{nd} Step – Remark

- Built-in functionality simplifies work considerably
- The alternative is to define and solve linear equations (mldivide, linprog)

Plotting Grid & Triangles

- Recall that M is an N-by-2 ($N \geq 3$) matrix specifying the coordinates of the grid. Every pair of dots is connected
- To plot every triangle, we first need to create a list of triples. Each triple specifies three (non-collinear) dots on the grid
- Built-in function to plot triangles: triplot()
- Built-in function to compute triples without loops: combntns()

combntns()

- **Syntax**: combos = combntns(set, subset)
- **Action**: returns a matrix whose rows are the various combinations of elements from vector ‘set’. Each combo (row) is of length ‘subset’
- **Example**:
  - combos = combntns (1:3,2) % “3 choose 2”
  - combos:

    [1 2
      1 3
      2 3]

triplot()

- **Syntax**: triplot(TRI, x, y)
  - triplot(TRI, x, y, color)
  - triplot(TRI, x, y, ‘param’, ‘value’)
- **Action**: displays the triangles defined in the N-by-3 matrix TRI. A row of TRI contains indices into the vectors x,y that define a single triangle. The default line color is blue
- **Example**:
  - x = rand(5,1); y = rand(5,1);
  - combos = combntns (1:5,3);   % “5 choose 3”
  - triplot(combos, x, y);

3rd Step (Code) – Plots

- Code to generate plots:

```
1. X = M(:,1); Y = M(:,2);                      % grid dots
2. figure(1);                                   % Create figure
3. plot(X,Y, 'r.', 'MarkerSize', 20);           % plot dots
4. figure(2);
5. plot3(X,Y, M(:,3), 'r.');                    % plot dots vs. time
6. figure(1); hold on;                          % Reset to figure 1
7. LT = combntns(1:length(X), 3);               % List all triangles
8. triplot(LT, X, Y, 'black');                  % Plot triangles
```

Computing Convex Hull

- Given a set of points in N-D, computing the hull is not a trivial task. Several algorithms are available
- The optimal algorithm may depend on the properties of the points
- Built-in functions: convhull(), convhulln()
- The hull is specified by a set of 'outer boundary' points

convhull()

- **Syntax**: [CH,V] = convhull(X,Y)
- **Action**: returns the 2D convex hull CH of the points (X,Y), where X and Y are column vectors, and the corresponding area/volume V bounded by K. 'CH' is a vector of point indices arranged in a counter-clockwise cycle around the hull
- **Example**:
  - x = [0,1,0.5,1,0]'; y = [0,0,0.5,1,1]';
  - [CH, A] = convhull (x,y); % CH = [1,2,4,5,1]', A = 1.0 (area of square)
  - plot(x(CH),y(CH),'r-', x,y,'b+')

Polygon Interior Check

- *Polygon:* a plane shape consisting of straight lines that are joined together to form a circuit
- The convex hull for a set of 2D points is a polygon. A triangle is the simplest polygon
- To check whether a point is inside the hull/triangle we need to define the polygonal region and check if the point is in the interior
- Built-in function to check: inpolygon()
- Another approach: solve a set of linear equations using linprog()

inpolygon()

- **Syntax**: IN = inpolygon(X,Y, Px,Py)
  - [IN ON] = inpolygon(X,Y, Px,Py)
- **Action**: returns a 0/1 matrix 'IN' the same size as X & Y. IN(k) = 1 if {X(k), Y(k)} is inside the polygon or on the boundary. The polygon vertices are specified by the vectors {Px, Py}
- **Example**:
  - Px = [0,1,1,0]'; Py = [0,0,1,1]'; % Polygon vertices
  - X = [0.1,0.2,0.5,1]; Y = [0.1,0.2,1.1,0.8]; % Points to test
  - IN = inpolygon (X,Y,Px,Py); % IN = [1,1,0,1]

3rd Step (Code) – Interior Check

- Code to compute/plot convex hull and check interior:

1. X = M(:,1); Y = M(:,2); % grid dots
2. figure(1); hold on; prop = 'MarkerSize'; % create figure
3. plot(X,Y, 'r.', prop, 20); % plot dots
4. [CH, A] = convhull (X,Y); % compute hull
5. Px = X(CH); Py = Y(CH); % polygon vertices
6. plot(Px,Py, 'g-', X,Y, 'r.', prop, 20); % plot hull
7. IN_H = inpolygon (Ix,Iy,Px,Py); % {Ix,Iy} starting point
8. LT = combntns(1:length(X), 3); % LT is the list of TRG
9. Tx = X(LT(k,:)); Ty = Y(LT(k,:)); % triangle vertices
10. IN_T = inpolygon (Ix,Iy,Tx,Ty); % check if inside triangle

Hull Interior Check – SLE

- Notice that every point inside the convex hull is a convex combination (weighted avg.) of the points for which the hull was computed
- In other words, if pt = {ptx, pty} is inside the hull, then for some set of weights $\{w_1 \ldots w_k\}$:
  1. $ptx = w_1x_1 + w_2x_2 + \ldots + w_kx_k$
  2. $pty = w_1y_1 + w_2y_2 + \ldots + w_ky_k$
  3. $w_1 + w_2 + \ldots + w_k = 1$
  4. $\forall i, w_i \geq 0$
- Need to solve a system of linear equations (SLE) with lower bounds on the variables

Hull Interior Check – Notation

- Standard notation:
  1. $ptx = w_1x_1 + w_2x_2 + \ldots + w_kx_k$
  2. $pty = w_1y_1 + w_2y_2 + \ldots + w_ky_k$
  3. $w_1 + w_2 + \ldots + w_k = 1$
  4. $\forall i, w_i \geq 0$
- Vector notation:
  1. $X^TW = ptx$
  2. $Y^TW = pty$
  3. $1^TW = 1$
  4. $W \geq 0$
- Solver form:
  1. $[X; Y; 1]^TW = [ptx; pty; 1]$
  2. $0 \leq W$

Solving SLE

- Given a system of linear equations $Ax = b$, does it have a solution?
- Depends on the matrix A:
  1. If A is a square matrix: $x = A^{-1}b$
  2. Else we have an under/over-determined system
  3. Could have multiple/no solutions for either case
- To compute the inverse of A efficiently, can decompose the matrix. Multiple ways of doing this (LU,QR)
- When A is not square, can minimize norm(A*X – B) i.e., the length of the vector $AX - B$. This is the least squares solution

mldivide (\)

- **Syntax**: x = A\b
  - x = mldivide(A,b)
  - x = inv(A)*b
- **Action**: If A is a square matrix, A\b is roughly the same as inv(A)*B. Otherwise, x = A\b is the least squares solution. A warning message is displayed if A is badly scaled or nearly singular
- **Example**:
  - A = [X; Y; 1]^T; b = [ptx; pty; 1]; % Hull interior equations
  - x = A\b; % Solution to SLE

Choosing a Strategy

- We need a strategy (set of rules) to select which triangle to turn off next
- *Greedy algorithm* makes the decision that gives the maximum benefit in the immediate next step (locally optimal). This decision might not be the best considering more (all) steps (globally optimal)
- Greedy strategy 1: choose the largest triangle
  - Built-in function to compute area: polyarea()
- Greedy strategy 2: choose the least-effort triangle
  - Sum the time to unravel the vertices for each triangle

polyarea()

- **Syntax**: A = polyarea(X,Y)
- **Action**: returns the area of the polygon specified by the vertices in the vectors X and Y. If X and Y are matrices of the same size, then the area is computed for each column (polygon) of {X,Y}
- **Example**:
  - TSx = X(LT); TSy = Y(LT); % Vertices for every triangle
  - T_area = polyarea(TSx', Tsy'); % Area for every triangle

3rd Step (Code) – Strategy

- Code to implement strategy:

```
1. X = M(:,1); Y = M(:,2); T = M(:,3);            % grid dots & time
2. LT = combntns(1:length(X), 3);                 % LT is the list of TRG
3. TSx = X(LT); TSy = Y(LT);                      % vertices ∀TRG
4. T_area = polyarea(TSx', Tsy');                 % area ∀TRG
5. Tot_effort = sum(T(LT),2);                     % total effort ∀TRG
6. choice = [LT, T_area', Tot_effort];            % Combine into one mtx
7. top_choice = sortrows (choice,-4);             % sort rows by area
8. top_choice = sortrows (choice,[-4,5]);     % sort by area then by time
```

fill()

- **Syntax**: A = fill(X,Y,C)
- **Action**: fills the polygon whose vertices are specified in {X,Y} with the constant color specified in C (C can be a single character string chosen from the list {r,g,b,c,m,y,w,k} or an RGB row vector triple, [r g b])
- **Example**:
  - `fill(X(LT(1,:)),Y(LT(1,:)), 'm');` % fill one triangle
  - `fill(X(LT)',Y(LT)','m');` % fill every triangle

3rd Step (Code) – Tracking

- Code to track progress:

```
1. X = M(:,1); Y = M(:,2); T = M(:,3);          % grid dots & time
2. LT = combntns(1:length(X), 3);               % LT is the list of TRG
3. figure(1);  hold on;                         % create figure
4. fill(X(LT)',Y(LT)','m');                     % fill every triangle
5. triplot(LT, X, Y, 'black');                  % Plot triangles
6. [CH, A] = convhull (X,Y);                    % compute hull
7. plot(X(CH),Y(CH), ‘g-’, X,Y, ‘r.’, prop, 20);    % plot hull
```