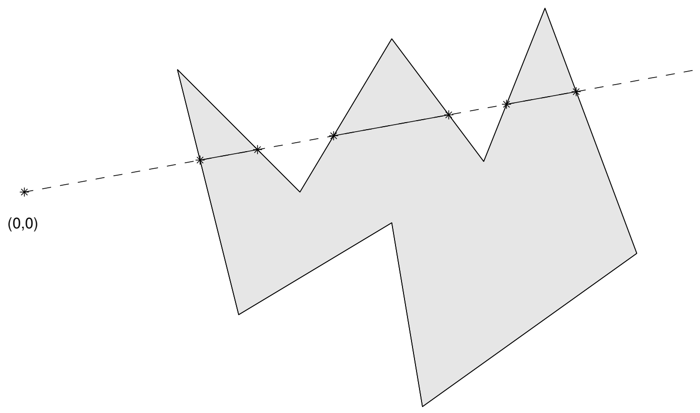
CS4220 Assignment 2 Due: 2/12/13 (Tue) at 11pm

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the solutions you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must first register as a group in CMS and then submit your work as a group. Each problem is worth 5 points. One point may be deducted for poor style.

Topics: Linear equation solving. Vectorization. Approximation. Heuristics.

1 A Ray Through a Polygon

Here is a ray "going through" a polygon:



We would like to compute the length of that part of the ray that is "inside" the polygon, i.e., the sum of the lengths of the solid line segments. To do this we need to compute the intersection points of the ray with the polygon edges and sum the appropriate point-to-point distances. It turns out that this involves solving a bunch of 2-by-2 linear systems, one for each edge of the polygon. The central part of the problem is to vectorize this process so that all these systems are solved "at the same time".

We start with a review of parametric equations for rays and line segments. A ray that "leaves" the origin making angle θ with the positive x-axis can be specified as follows:

$$\{\ (x(t), y(t)) \mid x(t) = \cos(\theta)t,\ y(t) = \sin(\theta)t,\ 0 \le t\ \}\ .$$

Likewise, a line segment that connects the points (α_1, β_1) and (α_2, β_2) can be specified as follows:

$$\{\ (x(t), y(t)) \mid x(t) = \alpha_1 + (\alpha_2 - \alpha_1)t,\ y(t) = \beta_1 + (\beta_2 - \beta_1)t,\ 0 \le t \le 1\ \}\ .$$

If we can find t_1 and t_2 that satisfy $0 \le t_1$ and $0 \le t_2 \le 1$ so that

$$\begin{aligned} \cos(\theta)t_1 &= \alpha_1 + (\alpha_2 - \alpha_1)t_2 \\ \sin(\theta)t_1 &= \beta_1 + (\beta_2 - \beta_1)t_2 \end{aligned}$$

then the ray and the line segment intersect. To that end we simply solve the 2-by-2 linear system

$$\begin{bmatrix} \cos(\theta) & (\alpha_1 - \alpha_2) \\ \sin(\theta) & (\beta_1 - \beta_2) \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix}$$

and check to see if $0 \le t_1$ and $0 \le t_2 \le 1$. If these conditions hold, then the ray and the line segment intersect and the point of intersection is $(\cos(\theta)t_1, \sin(\theta)t_1)$.

Returning to the ray-through polygon problem, suppose $(x_1, y_1), \ldots, (x_n, y_n)$ are the polygon's vertices and that the ray makes an angle of θ radians with the positive x-axis. Let $d(\theta)$ be the length of that part of the ray that is inside the polygon. Of course, if the ray fails to intersect the polygon then $d(\theta) = 0$. Otherwise, we need to determine the points where the ray intersects the polygon's edges. Suppose $(u_1, v_1), \ldots, (u_m, v_m)$ are these points of intersection, indexed in order of increasing distance from the origin. It follows that

$$d(\theta) = \sum_{i=1}^{m/2} \sqrt{(u_{2i-1} - u_{2i})^2 + (v_{2i-1} - v_{2i})^2}$$

Note that m, the number of intersection points, must be even because each time the ray "enters" the polygon it must "leave" the polygon.

Write a fully vectorized MATLAB function `[d,u,v] = InsideDist(x,y,theta)` that takes column n-vectors `x` and `y` that define the vertices of the given polygon and returns in `d` the value of $d(\theta)$ where θ is the value of the scalar `theta`. If the ray and the polygon intersect, then output parameters `u` and `v` should be column vectors that return the coordinates of the intersection points, indexed in order of increasing distance from the origin. If the ray and the polygon fail to intersect, then `u` and `v` should be empty vectors. Here is what `InsideDist` can assume about the polygon:

- $n \geq 3$.
- The origin is not inside the polygon.
- Every ray from the origin intersects no more than one polygon vertex. This guarantees that all the 2-by-2 systems are nonsingular.
- The polygons edges only meet at the vertices, i.e., no crossing edges.

For each polygon edge a 2-by-2 linear system must be solved to see if it intersects with the ray. These systems must be solved using the method of Gaussian elimination with partial pivoting. If the polygon has n sides, then we can assemble these linear systems in an n-by-4 array A, an n-by-2 array X, and an n-by-2 array B with the understanding that the kth linear system is

$$\begin{bmatrix} A(k,1) & A(k,2) \\ A(k,3) & A(k,4) \end{bmatrix} \begin{bmatrix} X(k,1) \\ X(k,2) \end{bmatrix} = \begin{bmatrix} B(k,1) \\ B(k,2) \end{bmatrix}$$

Here is one way to assemble the solutions in X using Gaussian elimination with partial pivoting

```
for k=1:n
   if |A(k,3)|>|A(k,1)|
   % interchange equations
      ???
   A11 = A(k,1); A12 = A(k,2); A21 = A(k,3); A22 = A(k,4);
   B1 = B(k,1); B2 = B(k,2);
   % L and U..
   U11 = A11; U12 = A12; L21 = A21/A11; U22 = A22 - L21*U12;
   % Solve "Ly = b"
   Y1  = B1; Y2  = B2  - L21*Y1
   % Solve "Ux = y"
   X2 = Y2/U22; X1 = (Y1 - U12*Y2)/U22;
   X(k,1) = X1; X(k,2) = X2
end
```

However, this can be vectorized. For example, if we ignore pivoting, then

$$L21 = A(:,3)./A(:,1);$$

assigns to `L21` the (2,1) entries of all the 2x2 L matrices. You will want to use the MATLAB `find` function to implement the vectorized pivot calculation. You may assume that all the 2x2 systems are nonsingular. (Not allowed if we were doing production graphics!) Submit `InsideDist` to CMS. A test script will be placed on the course website.

2 Approximate Exponential of an Intensity Matrix

The matrix exponential of $A \in \mathbb{R}^{n\times n}$ is given by

$$e^{At} \;=\; \sum_{k=0}^{\infty} \frac{(At)^k}{k}$$

This problem is about computing the exponential when A is an *intensity matrix.* The column sums of such a matrix are zero and its off-diagonal entries are positive, e.g.,

$$A \;=\; \begin{bmatrix} -.7 & .2 & .2 \\ .4 & -.8 & .3 \\ .3 & .6 & -.5 \end{bmatrix}$$

Implement the following function so that it performs as specified:

```
  function A = randIntensity(n)
% A is an n-by-n intensity matrix whose off diagonal entries are
% randomly selected from the interval [0,1/n].
```

We refer to an intensity matrix produced by this function as a *normalized intensity matrix.*

If A is an intensity matrix, then e^{At} is *stochastic* for all $t > 0$. That is, it has unit column sums and its entries are nonnegative. For the above matrix, if $t = 1$, then

$$e^{At} \;=\; \begin{bmatrix} 0.5384 & 0.1319 & 0.1319 \\ 0.2241 & 0.5201 & 0.1872 \\ 0.2375 & 0.3480 & 0.6809 \end{bmatrix}$$

Such exponentials arise in the conjunction with continuous Markov processes.

We now develop a framework for approximating the exponential of normalized intensity matrix. Just as

$$R(z) \;=\; \frac{1+z/2}{1-z/2}$$

approximates e^z for small z, so does

$$R(A) \;=\; (I - A/2)^{-1}(I + A/2) \qquad A \in \mathbb{R}^{n\times n}$$

approximate e^A for small $\| A \|$. Since $e^z = \left(e^{z/m}\right)^m$ we suspect that

$$R_m(z) \;=\; (R(z/m))^m$$

might be a better approximation provided m is large enough. Likewise, we suspect that

$$R_m(A) \;=\; (R(A/m))^m$$

might be a better approximation to e^A. Implement the following function so that it performs as specified:

```
  function F = MyExp(A,tol)
% A is an nxn  normalized intensity matrix, n<=1000
% tol satisfies  10^(-1) <= tol <= 10^(-12)
% F is a matrix that approximates F0 = e^A with
% |F(i,j) - F0(i,j)| <= tol for all i and j.
```

Your implementation should return $F = R_m(A)$ for a suitably chosen power m. Through experimentation, figure out how to determine m given `tol`. From the efficiency point of view the choice of m is important because work increases with m. You are allowed to solve linear systems use \\.

Submit `MyExp` to CMS together with a testing function `TestMyExp` of your own design that justifies your choosing-m strategy by running a suite of random examples. (Vary n and `tol`.) In your testing, you can use the MATLAB function `expm` for computing "true" matrix exponentials. `TestMyExp` should have `randomIntensity` as a subfunction. Since we will be running `TestMyExp` it should not take longer than 20 seconds to execute. Think of `TestMyExp` as experimental "proof" that your method for choosing m works most of the time. The code should be well written with nice comments. You are trying to convince "the jury"!

3 A Triangular Solver

In this problem you are to implement a function that can be used to solve the following equation for the matrix X:

$$SXT + X = B \tag{1}$$

We assume throughout that $S \in \mathbb{R}^{n\times n}$ is unit lower triangular, $T \in \mathbb{R}^{n\times n}$ is unit upper triangular, and $B \in \mathbb{R}^{n\times n}$. The algorithm you develop will compute X column by column and will exploit the *golden rule of matrix-vector multiplication*:

$$\boxed{\text{Matrix times Vector is a Linear Combination of the Matrix Columns}}$$

Let's apply the golden rule and determine $X(:,1)$. Comparing first columns in $B = SXT + X$ we conclude that

$$B(:,1) = S * (\text{first column of XT}) + X(:,1) = SX(:,1) + X(:,1) = (S+I)X(:,1)$$

Thus, $X(:,1) = (S+I)^{-1}B(:,1)$ can be found by solving a lower triangular linear system.

Now let's assume $k \geq 2$ and that we know $X(:,1),\ldots,X(:,k-1)$. We want compute $X(:,k)$. A key insight results by defining $Y = XT$ and using the golden rule to note that

$$Y(:,k) = XT(:,k) = T(1,k) * X(:,1) + T(2,k)X(:,2) + \cdots + T(k-1,k)X(:,k-1) + X(:,k). \tag{2}$$

Since

$$B = SXT + X \;=\; SY + X.$$

we also have

$$B(:,k) \;=\; SY(:,k) + X(:,k). \tag{3}$$

Manipulate (2) and (3) to show that $X(:,k)$ is the solution to a lower triangular linear system with a righthand side that depends on $B(:,k)$ and the previously computed $X(:,1),\ldots,X(:,k-1)$. Once you figure that out, you are all set to develop the following function:

```
  function X = TriMatSol(S,T,B)
% S is nxn and unit lower triangular
% T is nxn and unit upper triangular
% B is nxn
% X is nxn and xsatisfies SXT + X = B
```

To receive full credit, your implementation must be flop-efficient and it must obey the *brass rule of linear combinations*:

$$\boxed{\text{Any Linear Combination of Vectors is Secretly a Matrix-Vector Product}}$$

You are allowed to use \ when solving a triangular system. Submit `TrimatSol` to CMS. A test script `P3` is on the course website.