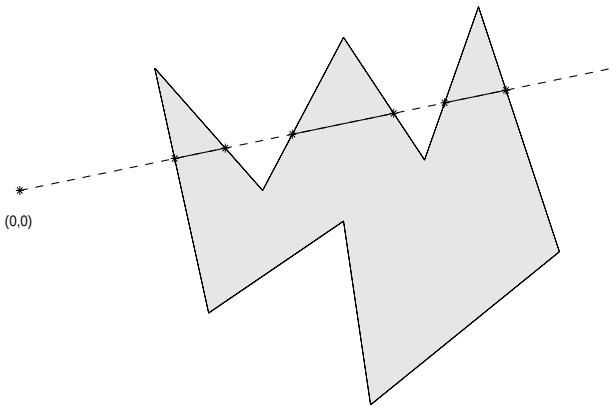
CS 4220: Assignment 2

Due: Wednesday, February 18, 2009 (In Lecture)

Scoring for each problem is on a 0-to-5 scale (5 = complete success, 4 = overlooked a small detail, 3 = good start, 2 = right idea, 1 = germ of the right idea, 0 = missed the point of the problem.) One point will be deducted for insufficiently commented code. Unless otherwise stated, you are expected to utilize fully MATLAB's vectorizing capability subject to the constraint of being flop-efficient. Test drivers and related material are posted on the course website `http://www.cs.cornell.edu/courses/cs4220/2009sp/`. For each problem submit output and a listing of all scripts/functions that *you* had to write in order to produce the output. You are allowed to discuss *background* issues with other students, but the codes you submit must be your own.

P1. (A Ray Through a Polygon

Here is a ray "going through" '



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`. 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.

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.

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. No loops are necessary for this part of the calculation as this process can be vectorized completely.

Run the test script `A2P1` to test your implementation of `InsideDist`. This script is available on the website. Submit the output and a listing of `InsideDist`.

P2. (A Markov Problem with Low rank Change)

Suppose $f, g \in \mathbb{R}^n$ and that $(f_1, g_1), \ldots, (f_n, g_n)$ are distinct. Define the matrix $C \in \mathbb{R}^{n \times n}$ by

$$c_{ij} = \begin{cases} 0 & \text{if } i = j \\ 1/((f_i - f_j)^2 + (g_i - g_j)^2) & \text{if } i \neq j \end{cases}$$

and the vector $d \in \mathbb{R}^n$ by

$$d_j = \sum_{i=1}^{n} c_{ij}$$

Three observations:

- If you move (f_1, g_1) then the C matrix changes by a rank-2 matrix.
- If $A = CD^{-1}$ where $D = \text{diag}(d_1, \ldots, d_n)$, then A is a stochastic matrix, i.e., it has unit columns sums and nonnegative entries.
- Think of a_{ij} as the probability that a flea hops from "landing point" (f_j, g_j) to landing point (f_i, g_i), with a kind of inverse square law underlying the hopping probabilities. (A flea is much more likely to hop to a nearby landing point.) To emphasize the dependence of A on the coordinates of the underlying landing points, we use the notation $A(f, g)$.

Develop a method for solving $Ax = b$ that involves the LU factorization of C, $PC = LU$. This will enable you to explore efficiently (via the Sherman-Morrison-Woodbury formula) how the solution changes if we move (f_1, g_1). Using these ideas, implement the following function so that it performs as specified.

```
  function [x0,X] = MarkovSol(f,g,u,v,b)
% f, g, and b are column n-vectors
% u and v are column q-vectors
% x0 = A(f,g)\b
% X is an n-by-q matrix with the property that if e1 is the first column of eye(n,n)
% then X(:,j) = A(f+u(j)*e1,g+v(j)*e1)\b for j=1:q
```

Test your implementation with the script `A2P2`.

P3. (Symmetric Semidefinite Systems)

We say that $A \in \mathbb{R}^{n\times n}$ is positive semidefinite if $x^T Ax \geq 0$ for all x. An easy way to generate a symmetric semidefinite matric is

```
V = randn(n,q)  % q < n
A = V*V';       % A will have n-p zero eigenvalues
```

Using MATLAB's help facility, read about the built-in function `ldl`. Examine what this function does when applied to a symmetric positive semidefinite matrix.

If $A \in \mathbb{R}^{n\times n}$ is symmetric positive semidefinite with rank p, then there exists a permutation P and a lower triangular matrix G such that

$$PAP^T = GG^T$$

where

$$G = \begin{bmatrix} G_{11} & 0 \\ G_{21} & 0 \end{bmatrix}$$

with $G_{11} \in \mathbb{R}^{q\times q}$ and $G_{21} \in \mathbb{R}^{n-q\times q}$. Implement a function of the form

```
[G11,G21,P] = PosSemiDefFact(A,tol)
```

that computes the factorization. Make effective use of `ldl`. The parameter `tol` should be a small positive tolerance that your code uses to identify zero submatrices. In particular, declare any submatrix of A to be zero if its 1-norm is less than `tol` times the 1-norm of A.

Run the test script `A2P3` and submit the output together with a listing of `PosSemiDefFact`.

P4. (Tridiagonal Systems)

Read §7.3 in the Chapter 7 handout and implement the following function:

```
  function uvals = TwoPtBVP(n,a,b,p,q,r,s)
%
% a and b are reals with a<b and n is an integer with >= 3.
% p, q, r, and s name functions defined on [a,b], the
% first two of which are positive on the interval.
%
% uvals is a column n-vector with the property that uvals(i) approximates
% the solution to the 2-point boundary value problem
%
%                -D[p(x)Du(x)] + s(x)u'(x) + q(x)u(x) = r(x)    a<=x<=b
%                 u(a) = u(b) = 0
%
% at x = a+(i-1)h where h = (b-a)/(n-1)
```

This will require the solution of an unsymmetric, positive definite, tridiagonal system. You may assume that pivoting is not required.

Test your implementation with the script `A2p4`. Submit output and listing.