Math 110 - Fall 05 - Lectures notes # 33 - Nov 16 (Wednesday)

We continue chapter 5, concluding with a brief discussion of section 5.3 on Markov chains, including a discussion of how Google works, i.e. how they decide which web pages to display in what order when you do a search. We will not prove everything we tell you, but we will prove the basic results.

We will use the language of probability. Even if you have not had a course in probability before, our results will be intuitively meaningful.

Ex 1: Suppose there is a metropolitan area with city dwellers and suburb dwellers. For city planning purposes, it is important to understand whether people are tending to move from the city to the suburbs or vice versa, and what the eventual populations will be. To make it simple we will assume the total population is constant. The data available from surveys says that in any given year

   The probability that a city dweller stays in the city is .90
   The probability that a city dweller moves to the suburbs is .10

These two probabilities must be nonnegative and add up to 1.

   The probability that a suburbanite stays in the suburbs is .98
   The probability that a suburbanite moves to the city is .02

We can also represent this information as a labelled graph (picture)
We record this data in a probability matrix:

$$
\begin{array}{cc}
 & \begin{array}{cc} \text{city} & \text{suburb} \end{array} \\
P = & \begin{bmatrix} .90 & .02 \\ .10 & .98 \end{bmatrix}
\end{array}
\begin{array}{l} \text{city} \\ \text{suburb} \end{array}
$$

in which there is one column for each current location (city, suburb) and one row for each location next year (in the same order) so P_{ij} = probability of moving from location j to location i
Note that each column of P must add up to 1.

Suppose we know that this year the fraction of the whole population who are city dwellers is .7, and the remaining .3 are suburbanites.

Here are the questions we will ask and answer:

Q1: What will be the fractions of the population who are city dwellers and suburbanites after 1 year?
(Sneak preview: do a matrix-vector multiply)

Q2: What will be the fractions of the population who are city dwellers

and suburbanites after 5 years? 10 years?
(Sneak preview: do more matrix-vector multiplies)

Q3: What will be the fractions of the population who are city dwellers and suburbanites "eventually" (after n years, as n -> infinity)
(Sneak preview: compute an eigenvector)

Ex 2: Instead of classifying people by location, we classify by political party. Suppose there are 3 political parties, R, B and G. Data about party registration says what fraction of voters registered with each party either remain with the party or switch to another one. One can represent this data with a labelled graph (picture) or a matrix:

$$
P = \begin{array}{cc} & \begin{array}{ccc} R & B & G \end{array} \\ & \left[\begin{array}{ccc} .85 & .05 & .01 \\ .07 & .90 & .04 \\ .08 & .05 & .95 \end{array}\right] \begin{array}{c} R \\ B \\ G \end{array} \end{array}
$$

Again, each column of P must add up to 1.

Our 3 questions are similar to before (and the sneak previews are identical): How many voters will be registered in each party next year? After 5 years? 10 years? eventually?

Ex 3: Now our rows and column labels will not be locations, or political party affiliations, but what web pages you are looking at. Imagine an "average" web user who is looking at a web page. We'd like a model of what web pages they are most likely to look at. So we imagine that our "average" web user takes all the outward links from the current web page (say there are 10 such links), picks one at random (say each with probability .1), and clicks on it. On which web pages will such an "average" user spend most time? This is the basic model used by Google. The matrix P for this is just the transpose of the "incidence matrix of the Web" that we defined in Lecture 13, where we divide each column entry by the number of its nonzero entries so that each column adds up to 1.

P_{ij} = 1/m if there is a link on web page j pointing to web page i and there are a total of m such outward link on web page j
= 0 if there is no link from web page j to web page i

(There are variations on this, depending on whether one takes into account multiple links from web site i to web site j, but we will keep this presentation simple.)

The original algorithm, invented by the founders of Google as students, is called Pagerank. If you type "Pagerank" into Google there are over 22M hits, so it is easy to find out more.

Now for some formal definitions:

Def: An n by n probability matrix P, or stochastic matrix, is a square matrix with nonnegative entries where each column adds up to 1.

Let u be the vector each of whose entries is 1. Then the definition says that P^t * u = u, i.e. that u is an eigenvector of P^t with eigenvalue 1. Recall that the eigenvalues of P and P^t are the same. We will soon see that 1 is also the largest eigenvalue of P.

Now we imagine a process where at any step we have a population (say of people), each of which has an associated "state" (say location, or political party, or current web page). At each "step" of this process, each member of the population randomly picks a new state to be in at the next step of the process. If a member is in state j, the probability of picking next state = i is P_ij. This process is called a Markov process.

If the number of members now in state k is m_k, then the "expected value" of the population of state i at the next step is given as follows:

```
 next m_i =   m_1 * probability(switching from 1 to i)
            + m_2 * probability(switching from 2 to i)
            ...
            + m_n * probability(switching from n to i)
          = sum_{j=1 to n} P_ij * m_j
or
  next m = P * m
By dividing each m_i by the total population M,
to get m_i/M = fraction of population in state i, we get
    vector of next population fractions
         = P * vector of current population fractions
```

In other words, figuring out the expected fractions in each state after 1 step requires matrix-vector multiplication by P.
If we want the expected fractions after k years, we multiply by P k times:

```
    vector of fractions after k years
       = P*(P* ... k times ... *(P* vector of current fractions)
       = P^k * vector of current fractions)
```

This suggests that we can think of P^k as a probability matrix too,
corresponding to taking k steps. Indeed it is:

Lemma: If P is a probability matrix, so is P^k for any k

Proof: if P has nonnegative entries, obviously so does P^k.
and if P has column sums equal to 1, that is P^t * u = u,

```
then (P^k)^t * u = (P^t)^k * u
                 = P^t * P^t * ... * P^t * u
                 = P^t * (P^t * ... * (((P^t * u )))
                 = u    as desired
```

Ex: Consider the city/suburb population example:

```
   P     = [ .90  .02  ]      P^2 = [ .812 .038 ]
           [ .10  .98  ]            [ .188 .962 ]
   P^5   = [ .606 .079 ]     P^10 = [ .399 .120 ]
           [ .394 .921 ]            [ .601 .880 ]
   P^100 = [ .166 .166 ]  "P^inf" = [ 1/6  1/6 ]
           [ .833 .833 ]            [ 5/6  5/6 ]
```

Now if m is any initial population with total population M

```
  "P^inf"*m = [ 1/6 1/6 ]*[m_1] = [1/6*M]
              [ 5/6 5/6 ] [m_2]   [5/6*M]
```

In other words, the eventual distribution of populations
converges to the same answer independent of who lived where
in the beginning. To see how this final distribution is

```
related to P, set [m_1] = [1/6]
                  [m_2] = [5/6]
```

and note that P * m = m,
i.e. m is an eigenvector of P for the eigenvalue 1.

Thm 1: If P is a probability matrix where each entry of P^k
is positive for k large enough (i.e. you can get to any state
from any other, eventually), then as k->infinity,

(1) P^k converges to to a limiting probability matrix L where each column
    is equal to the same vector v of positive numbers (summing to 1).
(2) v is an eigenvector of P for the eigenvalue 1, scaled so its
    entries sum to 1
(3) For any m, P^k * m converges to v*M, where m = sum_i m_i

We will prove this only partially, assuming P is diagonalizable
(a complete proof would use ideas from Chapter 7, on Jordan

```
forms, which is an eigendecomposition for non diagonalizable
matrices). We will need a Theorem of independent interest:

Gershgorin's Theorem: Let A be an n by n matrix with entries from C
(the complex numbers). Then any eigenvalue lambda of A must satisfy the
following inequality for some i:
   | lambda - A_ii | <= sum_{j=1 to n except i} |A_ij|
Note: each such inequality defines a circle in the complex plane.
In other words, the eigenvalues must lie in a union of circles
(which might overlap).

Example of Gershgorin's Theorem:
  If A = [ 1 .1 .2 ]
         [ .1 4 .5 ]
         [ 1  4 20 ]
then each lambda of A must lie in one of 3 circles
    | lambda - 1 | <= .3
    | lambda - 4 | <= .6
    | lambda -20 | <= 5

Proof of Gershgorin's theorem: Suppose A*x = lambda*x, so that
x is an eigenvector. Divide x by its largest component in absolute
value. If this component was the i-th component then x_i is now = 1,
and the other components |x_j| <= 1. Then
  0 = (A*x - lambda*x)_i
    = (A_ii - lambda)*x_i + sum_{j=1 to n except i} A_ij*x_j
or
  (A_ii - lambda)*x_i = -sum_{j=1 to n except i} A_ij*x_j
Using x_i = 1, |x_j|<=1 and the triangle inequality we get
  |A_ii - lambda| <= sum_{j=1 to n except i} |A_ij*x_j|
                  <= sum_{j=1 to n except i} |A_ij|

Partial proof of Thm 1: We apply Gershgorin's Theorem to P^t
(which has the same eigenvalue as P) to get that any eigenvalue
lambda of P satisfies
   | lambda - P_ii | <= sum_{j=1 to n except i} |P_ji|
                      = sum_{j=1 to n except i} P_ji
                          ... since all P_ij >= 0
or
   | lambda | = | lambda - P_ii + P_ii |
             <= | lambda - P_ii | + | P_ii |
             <= sum_{j=1 to n except i} P_ji   +    P_ii     = 1
```

```
So all the eigenvalues of P are less than or equal to 1 in
magnitude. We know from before that one eigenvalue equals 1.
Assuming that
  (1) P is diagonalizable, and
  (2) all eigenvalues but one of P are actually < 1 in magnitude
then from diagonalizability we get
  P = Q * Lambda *Q^{-1}
where Lambda = diag( 1, lambda_2 , ... , lambda_n}
where | lambda_i | < 1 for i >= 2
so P^k = Q * Lambda^k * Q^{-1}
       = Q * diag(1, lambda_2^k , ... , lambda_n^k) * Q^{-1}
As k -> infinity, all the lambda_i^k -> 0, so
   P^k ->  Q * diag(1,0,...,0) * Q^{-1}
         = Q * ( e_1 * e_1^t) * Q^{-1}
         = (Q * e_1) * (e_1^t * Q^{-1})
         = (column 1 of Q) * (row 1 of Q^{-1})
         = (v) * (r^t)
         = L
v is an eigenvector of P corresponding to eigenvalue 1.
But note that any nonzero multiple of v is also an eigenvector,
so we haven't determined it uniquely yet. Nor have we determined r.

To determine r and then L, we note that since P^k has unit
column sums for all k, i.e. P^k*u = u, then
this is obviously true as k-> infinity, i.e. L*u=u.
In other words, L's column sums are 1 too. This is
all we need: Note that column i of L is v*r_i.
Since the sum of the entries of each v*r_i is 1,
we get that L = v*u^t, where v is the eigenvector
whose entries sum to 1, and the entries of u are all ones.

(Strictly speaking, we have assumed that there is only
one eigenvector of P corresponding the the eigenvalue 1,
except for constant multiples. To prove this we would need to
use our assumption that P^k has all positive entries for k
large enough.)

Now it is easy to prove the third part of Thm 1:
  P^k*m -> L*m = (v*u^t)*m = v*(u^t*m) = v * (sum_i m_i) = v*M

A complete proof of Thm 1 would use our assumption that each entry
```

of P^k is positive for large enough k to show that all eigenvalues of P except 1 are less than 1 in absolute value. We would also use the Jordan form to show that Lambda^k -> diag(1,0,...,0) even when P is not diagonalizable.

Now we discuss how Google works, roughly:
Once every few days:
  (1) Form the 8 billion by 8 billion matrix P corresponding to the web.
  (2) Compute (approximately!) the eigenvector v of P for eigenvalue 1. Thus v(i) assigns a number to each web page i, that corresponds to the chance than an "average" web user will look at it.
Every time an user types in a search request into Google:
  (1) Find all the pages that contain the search string (or a synonym, different spelling, etc).
  (2) Sort the pages in decreasing order of their values of v(i)
  (3) Display the pages in this order, ie the ones an "average user" would view most often coming first.

In Lecture 13, we said that multiplying P by itself would take millions of years, unless we took advantage of the fact that nearly all its entries are zeros: Most web pages have links to very few other web pages. If the average number of such links were, say, 10, then the number of nonzero matrix entries in P is 10*n where n is the dimension of P. Such a matrix, most of whose entries are zero, is called sparse. To multiply a sparse matrix times a vector, we only need to store and multiply by the nonzero entries. So instead of costing n^2 multiplies and n^2 additions to compute P*x, it costs only 10*n multiplies and 10*n additions, a factor of n/10 faster, or about .8 billion times faster.

So how does Google find the eigenvector v of P corresponding the eigenvalue 1? According to Theorem 1 (we assume P satisfies its conditions) if we pick an arbitrary starting vector x^(1), and repeatedly multiply it by P:

```
x^(2) = P * x^(1), x^(3) = P*x^(2) ,..., x^(k) = P*x^(k-1)
```

then x^(k) converges to a multiple of v. Once x^(k) stops changing very much, we assume it has converged well enough, and then set v = x^(k) / sum_i x^(k)_i.