THREE LINEAR ALGEBRA APPLICATIONS

11/28/2018, MATH 121 GUEST LECTURE

Three Applications of Linear algebra

Rather than grinding through a laundry list of applications, we focus on three parts, where linear algebra plays a role. The first is an unsolved problem in **complexity theory** of arithmetic, the second is a short overview how **data structures** and data storage rely on notions put forward by linear algebra. The third is a **spectral problem** in graph theory which is related to **networks**. The lecture will conclude with a slide show showing off some applications without going into details.

1. Fast multiplication

1.1. An important unsolved problem in computer science is the question, how fast one can multiply two integers. The grade-school multiplication of two n-digit numbers uses about n^2 computation steps. Modern computer algebra frame works use more sophisticated methods like the GNU multiple precision arithmetic library. These methods chose between different methods.

1.2. The **Karatsuba method** uses the idea that the one can write $(a+b)(c+d) = u+(u+w-v)+w$ where $u = ac, w = bd, v = (b-a)(d-c)$. Instead of 4 multiplications, we need only 3. This cuts the $O(n^2)$ complexity to $O(n^{\log_2(3)}) = O(n^{1.5849})$ which is faster. For example, to multiply $x = 1234$ with $y = 5577$, build $u = 12 * 55 = 660$ and $w = 34 * 77 = 2618$, and $v = (33-12)(77-55) = 22 * 22 = 484$, then compute $u * 10000 + v100 + w = 6600000 + (660 + 2618 - 484) * 100 + 2618 = 6882018$.

1.3. I learned the following method from a talk of **Albrecht Beutelspacher**. If numbers are close to a round number like 1000, there is a neat trick to make the computation. Assume for example, we have two 3 digit numbers x, y so that the product of $1000-a$ and $1000-b$ is smaller than 1000. Then use $(1000-a)(1000-b) = 1000(1000-a-b)+ab = 1000(x-b)+ab$. For example, to compute $991 * 899$ build $x - b = 991 - 101 = 890$ and $ab = 9 * 101 = 909$. The result is 890909.

1.4. The fastest multiplication methods are based on linear algebra. Given again two integers like $x = 1234$ and $y = 5577$. If n is the number of digits in x and y, then we need about n^2 operations to get the product. It turns out that there is a faster way to do that. And this involves linear algebra. The idea is to write the numbers x, y as functions, then multiply the functions using **Fourier theory**. How does this work?

1.5. If x_k and y_k are the integer coefficients of x and y so that $x = \sum_{k=0}^{n-1} x_k 10^k$ and $y = \sum_{k=0}^{n-1} y_k 10^k$, we look at the functions $f(z) = \sum_{k=0}^{n-1} x_k z^k$ and $g(z) = \sum_{k=0}^{n} y_k z^k$. We assume n is the number of digits in xy and that x_k, y_k are zero for k larger than the number of digits of x or y. Now, if $h(z) = f(z)g(z)$, then $h(10)$ is the product. The point is that we can compute n data points of h with $n\log(n)$ operations and that we can get from n data points of a polynomial the function back in $n\log(n)$ operators. So, we can compute the product xy in $n\log(n)$ operations. How do we achieve this miracle?

1.6. Given a polynomial f, we can look at the periodic function $F(t) = f(e^{2\pi it})/\sqrt{n}$ and evaluate $X_k = F(2\pi k/n)$ for $k = 0, \dots, (n-1)$. This produces a linear map $U : (x_0, \dots, x_{n-1}) \to (X_0, \dots, X_{n-1})$ which is called the **discrete Fourier transform**. There is a **fast Fourier transform** implementation which does this in $n\log(n)$ computation steps. If $U(y_0, \dots, y_{n-1}) = (Y_0, \dots, Y_{n-1})$ then we have just to compute $Q = U^-1(XY)$ and get $xy = \sum_{k=0}^{n} Q_k 10^k$. Mathematica has the fast Fourier transform and its inverse implemented. Here is the code:

```
FastMultiplication[x_,y_]:=Module[{X,Y,U,V,n,Q},
  X = Reverse[IntegerDigits[x]]; Y = Reverse[IntegerDigits[y]];
  n =Length[X]+Length[Y]+1; X=PadRight[X,n];    Y=PadRight[Y,n];
  U=InverseFourier[X]; V=InverseFourier[Y];
  Q=Round[Re[Fourier[U*V]*Sqrt[n]]];
  Sum[Q[[k]] 10^(k-1), {k,n}]];
x0 = 11234; y0 = 52342; FastMultiplication[x0,y0]==x0*y0
```

The procedures Fourier and InverseFourier are implemented already in Mathematica Here are emulations showing what they do:

```
X={4,2,3,2,1}; n =Length[X]; InverseFourier[X]
f = Sum[X[[k]] t^(k-1),{k,n}];
N[Table[f /. t -> Exp[-I 2 Pi k/n],{k,0,n-1}]]/Sqrt[n]
N[(Table[Exp[I 2Pi k l/n],{k,0,n-1},{l,0,n-1}]/Sqrt[n]).X]
Fourier[X]
```

2. Data structures

2.1. Information is commonly represented in the form of **vectors** or **matrices**. This is sometimes not obvious as data can be complicated. Still, we usually get away with matrices. For example, if we have names linked to addresses and telephone numbers, we can make a **spread sheet** $A(i,j)$, where $A(i,1)$ is the name and $A(i,2)$ is the address and $A(i,3)$ is the telephone number. In a **relational database**, information is stored in **tables** as matrices containing **records** as rows and **attributes** as columns.

2.2. Data can also be stored more geometrically, especially in the form of a **graph**. A special case is the **hierarchical database** in which the graph is a tree, the Khipu knot encoding used in the **Incan empire** is an example of a hierarchical database. It is important to note however that also geometric information like a graph can be stored as part of a relational data base. A graph with nodes can be encoded with an **adjacency matrix** $A(i,j)$ defined as $A(i,j) = 1$ if i and j are connected and $A(i,j) = 0$ else.

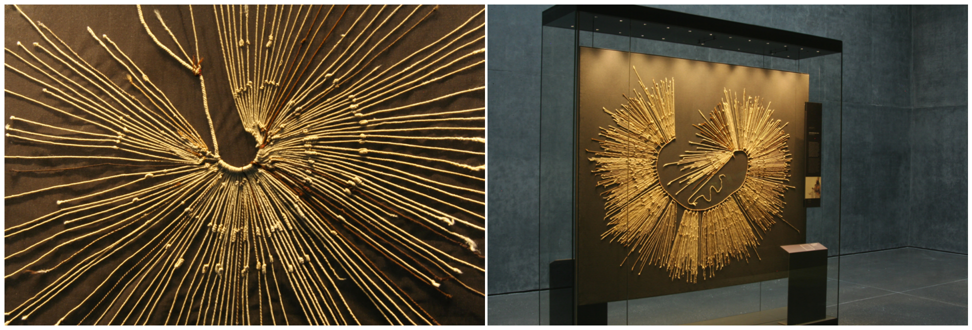

FIGURE 1. Khipu from the Inka period. Image Source: Museo Chileno De Arte Precolombino.

3. MEDIA

3.1. Multimedia like images, sound or movies are stored in vector or matrix form. An **image** is an array of pixels, a **sound** is an array of amplitudes, a **movie** is given by an array of pictures and a sound.

3.2. Let us look at a **color image** A: It is a matrix where each entry $A(i,j)$ contains three numbers (r,g,b), where r,g,b are red, green and blue color values. Even so this is a rectangular array of pixels, where each pixel is a vector of numbers, this can be written as a single vector.

3.3. A sound file consists of two vectors (l,r), where l is the left channel and r is the right channel. Each entry can be a signed integers from -127 to 128. A movie can be seen as a vector of pictures. It is a curve in the large space of all pictures.

FIGURE 2. 6 Frames from the Movie Samsara (2011). Each frame is a 360 x 854 matrix, where each pixel has a Red,Green and Blue value.

4. Networks

4.1. A graph $G = (V, E)$ consists of a collection of nodes V which are connected by edges collected in E. Graphs in which the direction of the edges matter are also called **digraphs**. If one says graph, one usually does not specify directions. A graph can be encoded as a matrix A, the **adjacency matrix** of A. It is defined as $A(i, j) = 1$ if i is connected to j and $A(i, j) = 0$ else. This matrix is **self-adjoint**, $A = A^T$. Therefore, the eigenvalues of A are real.

4.2. The analogue of the **Laplacian** in calculus is the matrix $L = B - A$, where $B = \mathrm{Diag}(d_1, \cdots, d_n)$ is the diagonal matrix containing the **vertex degrees** of B in the diagonal. The matrix L always has the eigenvalue 0, with eigenvector $[1, 1, \cdots, 1]$. If the graph is connected, the rest of the eigenvalues are positive. They are the **frequencies** one can hear, when one hits the graph. The eigenvectors tell something about the distribution of electrons if the graph represents a molecule. In the case of circular graphs, we need the same mathematics than for the fast multiplication of integers. The circle is closed.

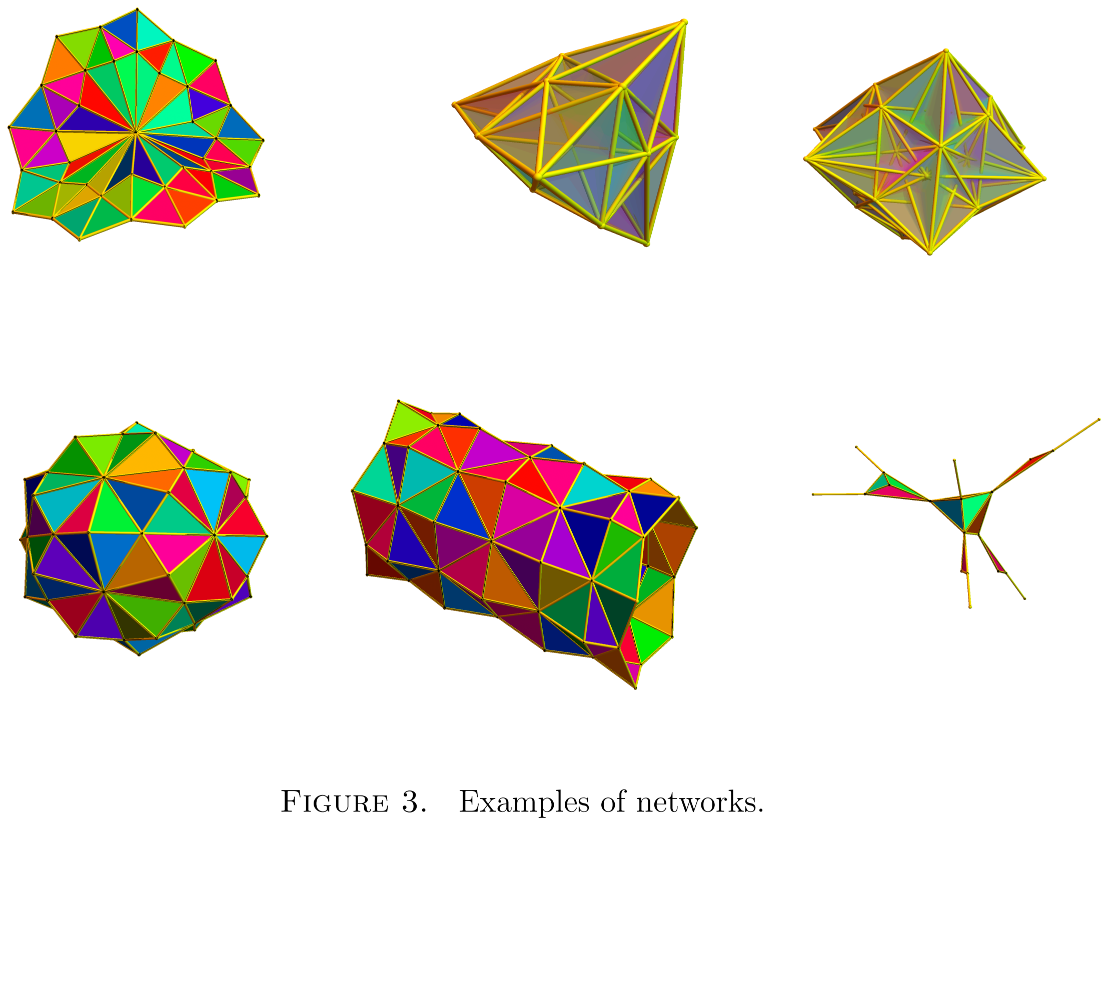

Figure 3. Examples of networks.