CS 270 Algorithms **Spring 2021**

Lecture 5: Linear Algebraic Tools

Lecturer: Prasad Raghavendra Scribe: Zachary Golan-Strieb, Rohith Sajith, Jacob Krantz

5.1 Solving Linear Systems

Solve for vector $x \in \mathbb{R}^n$ given some system of equations $Ax = b$ where $A \in \mathbb{R}^{n \times n}$ is invertible.

Gaussian elimination

These work for every matrix and tend to have a fixed runtime. They have no dependence of eigenvalues or condition numbers and very high accuracy. The runtime of an algorithm for this technique is $O(n^3)$ and $O(n^2)$ space. We need $O(n^2)$ space to store the entries of the matrix. These methods are generally unable to effectively exploit sparsity. The following are examples of Gaussian elimination methods.

- Gaussian Elimination: Complete row reduction techniques in order to manipulate the matrix into a more desirable form, which presents the solution to the system of equations represented by the matrix.
- LU Decomposition: One can decompose a given matrix (which represents a system of equations) into the product of a lower triangular matrix and an upper triangular matrix. If the lower triangular and upper triangular matrices are called L and U, respectively, one can then write $Ax = b \iff LUx = b$ and use this decomposition to solve the system of equations with ease.

Iterative methods

One starts with an attempt at the solution and continues to improve it by modifying it. The drawback of using these methods is that they *are* dependent on certain properties of the matrices. For example, the runtime could be dependent on the condition number of the matrix. However, these algorithms only require $O(n)$ space (other than the original matrix A) and are well-equipped to exploit the sparsity of given matrices ("dense matrices never exist in real life"). The methods described below rely on condition number, which is somewhat troubling because this may not be in terms of given parameters. For example, if we had a system of linear equations on a graph, we would want to be able to describe the error in terms of graph parameters, but we would need to use the condition number too in this case. The following are examples of iterative methods.

- Steepest Descent: At each point in time, one performs gradient descent and chooses the step size, η, which goes to the minimum in the given direction, represented by the gradient. The error is $e^{-t/\kappa}$. We will discuss this algorithm below.

- Kaczmarz Method: A similar algorithm to steepest descent. We give a summary of the Kaczmarz method, as presented in chapter 14 of Vishnoi's book, *Laplacian Solvers and Their Algorithmic Applications*. We want to approximate a solution to $Ax = b$. Begin by initializing $x_0 = 0$. Then, for $t \geq 0$, where $Ax_t \neq b$, find a hyperplane, H, of the form $\langle a_i, x\rangle = b_i$ such that $x_t \notin H$ and a_i is the i-th row of A. We define x_{t+1} as the orthogonal projection onto H. One can also consider the Randomized Kaczmarz method, in which we project onto the hyperplane $\langle a_i, x\rangle = b_i$ with probability proportional to $\|a_i\|^2$.
- Conjugate Gradient Descent: A clever algorithm that is related to accelerated gradient descent and which has error $e^{-t/\sqrt{\kappa}}$. This algorithm can be found in chapter 16 of Vishnoi's book in greater depth, but we will see a brief summary of that material here. Let x_0 be the starting vector so that we may define $r_0 = A(x_* - x_0)$. We then note that, given $P = \{p_0, \ldots, p_t\}$, if for all $i \neq j$, we have $p_i^\top A p_j = 0$, we say that P is A-orthonormal. Thus, take a set of A-orthonormal vectors, P, as just defined, with span equal to $\mathcal{K}_t = \text{span}\{r_0, Ar_0, \ldots, A^{t-1}r_0\}$. One can then define

  $$\alpha_t = \frac{p_t^\top r_0}{p_t^\top A p_t}$$

  so that the update formula reads

  $$x_{t+1} = x_t + \alpha_t p_t.$$

  Note that the α_j are chosen so that $f(x_0 - \alpha p_j) - f(x_0)$ is minimized, where $f(x) = \frac{1}{2}x^\top Ax - b^\top x$.

Steepest descent

Assume A is a real symmetric PSD matrix. If A is not a symmetric PSD matrix, construct an $A_{\text{new}} = A^\top A$ and $b_{\text{new}} = A^\top b$ and solve $A_{\text{new}}x = b_{\text{new}}$.
The function f that we would like to minimize to solve the system is:

$$f(x) = \frac{1}{2}x^\top Ax - bx$$

f is convex because we assume $A \succeq 0$. Any x^* that minimizes f i.e. $\nabla f(x^*) = 0$ is a valid solution to our system $Ax = b$. The following is a proof:

$$\nabla f(x) = Ax - b$$
$$\nabla f(x^*) = 0$$
$$Ax^* - b = 0$$
$$Ax^* = b$$

A single step of gradient descent is:

$$x_{t+1} = x_t - \eta(Ax_t - b)$$

Let $\text{nnz}(A)$ be the number of non-zero elements of A. Ax_t can be computed in time $\text{nnz}(A)$. Note the following definition before continuing to the proof.

Definition 5.1. *(Various Norms) Let $T : V \to W$ be a linear map between normed vector spaces. One defines*

$$\|T\|_{\text{op}} = \sup\{\|Tx\|_W : \|x\|_V \leq 1\}.$$

Additionally, recall that

$$\|x\|_{\ell^2} = \left(\sum |x_i|^2\right)^{\frac{1}{2}} \text{ and } \|x\|_{\ell^p} = \left(\sum |x_i|^p\right)^{\frac{1}{p}}.$$

Finally, notice that if A is both real and symmetric, $\|A\|_{\text{op}} = \max_j\{|\lambda_j|\}$.

Proof of convergence

$$\begin{aligned} x_{t+1} - x^* &= (x_t - \eta \nabla f(x_t)) - x^* \\ &= (x_t - \eta(Ax_t - b)) - x^* \\ &= (x_t - \eta(Ax_t - Ax^*)) - x^* \\ &= (I - \eta A)(x_t - x^*) \end{aligned}$$

Let $\lambda_i(X)$ be the ith smallest eigenvalue of some matrix X, and let $\lambda_1, \cdots, \lambda_n$ be the eigenvalues of A in increasing order. We know that:

$$\lambda_i(I - \eta A) = 1 - \eta \lambda_i$$

Since $A \succeq 0$ i.e. $\lambda_i \geq 0, \forall i$, we can choose a small enough η such that:

$$|1 - \eta \lambda_i| < 1, \forall i$$

Therefore, the distance between x_t and the optimal solution x^* is bound by:

$$\begin{aligned} ||x_t - x^*|| &= ||(I - \eta A)^t (x_0 - x^*)|| \\ &\leq \rho^t ||x_0 - x^*|| \end{aligned}$$

In the above inequality, $\rho = ||1 - \eta A||_{\text{op}}$, which is known as the spectral radius in this case.

Let us choose $\eta = \frac{1}{\lambda_n}$. Then, we have:

$$\max_i \lambda(I - \eta A) = \max_i (1 - \eta \lambda_i) = 1 - \eta \lambda_1 = 1 - \frac{\lambda_1}{\lambda_n} = 1 - \frac{1}{\kappa}$$

In the above expression, κ is the condition number of A i.e. $\kappa = \frac{\lambda_n}{\lambda_1}$.

Choosing η

Let $h(\eta)$ be the function f along the direction of the gradient ∇f:

$$h(\eta) = f(x_t + \eta \nabla f(x_t))$$

To pick the optimal step size, we will choose the η which minimizes $h(\eta)$:

$$x_t + \eta \nabla f(x_t) = x_t - \eta(Ax_t - b)$$

$$\begin{aligned} h(\eta) &= f(x_t + \eta \nabla f(x_t)) \\ &= (x_t + \eta \nabla f(x_t))^\top A (x_t + \eta \nabla f(x_t)) - b^\top (x_t + \eta \nabla f(x_t)) \\ &= \frac{1}{2}(x_t - \eta(Ax_t - b))^\top A (x_t - \eta(Ax_t - b)) - b^\top (x_t - \eta(Ax_t - b)) \\ &= \frac{1}{2} x_t^\top A x_t - \eta x_t^\top A(Ax_t - b) + \frac{1}{2}\eta^2 (Ax_t - b)^\top A (Ax_t - b) - b^\top x_t + \eta b^\top (Ax_t - b) \\ &= \frac{1}{2} x_t^\top A x_t - b^\top x_t + \eta(-x_t^\top A(Ax_t - b) + b^\top (Ax_t - b)) + \frac{1}{2}\eta^2 (Ax_t - b)^\top A (Ax_t - b) \end{aligned}$$

$$\nabla h(\eta) = -x_t^\top A(Ax_t - b) + b^\top(Ax_t - b) + \eta(Ax_t - b)^\top A(Ax_t - b)$$
$$\nabla h(\eta^*) = 0$$
$$-x_t^\top A(Ax_t - b) + b^\top(Ax_t - b) + \eta^*(Ax_t - b)^\top A(Ax_t - b) = 0$$
$$\eta^* = \frac{x_t^\top A(Ax_t - b) - b^\top(Ax_t - b)}{(Ax_t - b)^\top A(Ax_t - b)}$$

Preconditioned gradient descent

We can can precondition A with a non-singular matrix L and solve the following system:

$$L^{-1}Ax = L^{-1}b$$

Our new single step of gradient descent is:

$$x_{t+1} = x_t - \eta L^{-1}(Ax_t - b)$$

The point of preconditioning is to find a matrix L so that $\kappa(L^{-1}A)$ is small and solving the system will require fewer gradient steps. We also want to be able to quickly multiply by L^{-1}. Computing $x = L^{-1}w$ is same as solving $Lx = w$, so solving $Ax = b$ can be reduced to solving $Lx = w$ proportional to $\kappa(L^{-1}A)$ times.

5.2 PCA

The goal of PCA is to find the best k-dimensional approximation of some data $A \in \mathbb{R}^{n \times d}$ where there are n data points of dimension d, where typically $k < d$:

$$\max_{W=\text{Span}(v_1,\cdots,v_k)} \sum_i ||\text{proj}_W(A_i)||^2$$

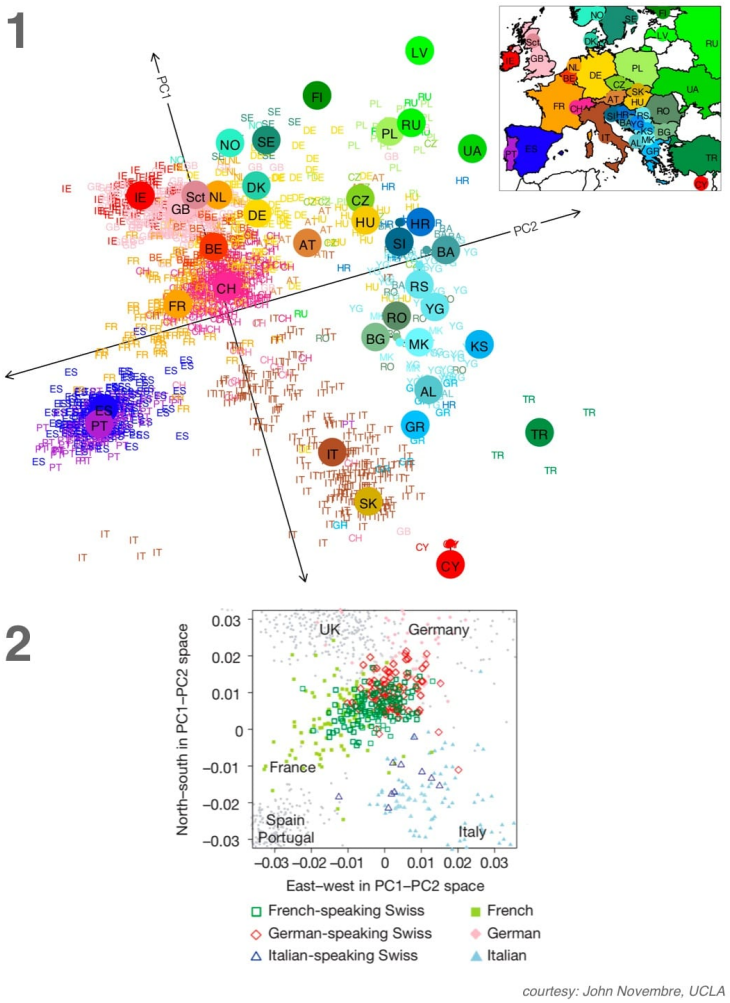

Figure 5.1: *
Plot of genetic data for people in Europe projected to two-dimensions using PCA. Interestingly, the plot resembles a map of Europe when data points are colored by that individual's country of origin.

Optimal greedy algorithm

It turns out that we can find the best low-dimensional subspace to project onto by iteratively finding the best direction to project onto, removing the components in that direction from our data, and repeating, storing these vectors as the basis for the subspace:

$$v_1 = \max_{||v||=1} \sum_i \langle A_i, v \rangle^2$$

$$v_2 = \max_{||v||=1, v \perp v_1} \sum_i \langle A_i, v \rangle^2$$

$$v_k = \max_{||v||=1, v \perp v_1, \cdots v_{k-1}} \sum_i \langle A_i, v \rangle^2$$

At any given iteration, we are attempting to solve the following optimization problem after removing the components of our data in the directions $v_1, \cdots, v_{i-1}$:

$$\begin{aligned} v_i &= \max_{||v||=1} \sum_i \langle A_i, v \rangle^2 \\ &= \max_{||v||=1} ||Av||_2^2 \\ &= \max_{||v||=1} v^\top A^\top A v \\ &= \lambda_{\max}(A^\top A) \end{aligned}$$

Therefore, the optimal choice of vector v_i is the eigenvector corresponding to the largest eigenvalue of $A^\top A$. It turns out since the eigenvectors of a real, symmetric matrix are orthogonal, the first k eigenvectors with largest eigenvalues make up the basis for the best k-dimensional subspace to project onto. We can perform QR-decomposition with runtime $O(n^3)$ to find the eigenvectors of $A^\top A$. However, for large matrices this is quite costly, so we often use an iterative algorithm called the power method to find the eigenvector with the largest eigenvalue.

5.2.1 Power method

The goal of the power method is to compute the largest eigenvector λ_n and its corresponding eigenvector v_n of $B = A^\top A$. Given a random vector x, we know it can be decomposed into a linear combination of $v_1, \cdots, v_n$ the eigenvectors of B.

$$x = \sum_{i=1}^{n} c_i v_i$$

Applying B to x will scale the components like so:

$$Bx = \sum_{i=1}^{n} c_i B v_i = \sum_{i=1}^{n} c_i \lambda_i v_i$$

Applying B to x a total of s times will give us:

$$B^s x = \sum_{i=1}^{n} c_i \lambda_i^s v_i$$

Let us assume without loss of generality the largest eigenvalue of B is $\lambda_n = 1$:

$$\lim_{s \to \infty} B^s x = \lim_{s \to \infty} c_i (1)^s v_i + \sum_{i=1}^{n-1} c_i \lambda_i^s v_i = c_i v_i$$

Therefore, in the limit that we keep applying B to x, a vector in the direction of the largest eigenvector will emerge.

In order to find the k largest eigenvectors, we can actually start with a set of orthogonal vectors $x_1, \cdots, x_k$, compute $Bx_1, \cdots, Bx_k$, othogonalize them, and repeat. One can also, instead, subtract the projection of A_i onto the most recent eigenvector from A_i itself and continue as described in the usual power method above.

If the eigenvalues of B are $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1} \leq 1 - \epsilon < \lambda_n = 1$, then the accuracy of our eigenvectors is:

$$\delta \leq (1 - \epsilon)^T d$$

Therefore, the runtime to achieve an accuracy of δ is:

$$T \geq \frac{\ln(\frac{d}{\delta})}{\epsilon}$$