Numerical Computing - Notes

Matteo Alberici

January 2022

Contents

1 PageRank Algorithm

1.1 Eigenvectors

An **eigenvector** $v \in \mathbb{R}$ of a matrix A is a nonzero vector such that the relation

$$Av = \lambda v$$

holds, where λ is the **eigenvalue** associated with vector v.
An **eigenbasis** is a basis in which every vector is an eigenvector.
Given a vector v, the **Rayleigh quotient** computes the eigenvalue of v whether v is an eigenvector, or the eigenvalue associated with closest eigenvector of v otherwise:

$$\mu(v) = \frac{v^T A v}{v^T v}$$

1.2 PageRank Algorithm

The **PageRank** is determined by the structure of the World Wide Web. For any query, Google lists the matching Web pages in their PageRank order: a Web page has a high rank if other pages with a high rank link to it.
The algorithm is based on the **Random Surfer Model**, in which a user goes from one page to another by randomly choosing an outgoing link through **exploitation**. Since surfing randomly could lead to dead ends or cycles of Web pages, a random Web page is chosen through **exploration**. The random walk generated by exploitation and exploration is known as the **Markov chain**.
Let's introduce some definitions for the PageRank computation:

- n = number of Web pages
- G = $n-$by$-n$ connectivity matrix of a Web portion with:

$$g_{ij} = \begin{cases} 1 & \text{hyperlink from page } j \text{ to page } i \\ 0 & \text{otherwise} \end{cases}$$

- r_i = row sum of G and in-degree of $j-$th page:

$$r_i = \sum_j g_{ij}$$

- c_j = column sum of G and out-degree of $j-$th page:

$$c_j = \sum_i g_{ij}$$

- p = probability that the random walk follows a link
  - the typical value is: $p = 0.85$
  - the probability that some arbitrary page is chosen is: $1 - p$
  - the probability that a particular random page is chosen is:

$$\delta = \frac{1-p}{n}$$

- A = $n-$by$-n$ transition probability matrix of the Markov chain with:

$$a_{ij} = \begin{cases} p \cdot \frac{g_{ij}}{c_j} + \delta & c_j \neq 0 \\ \frac{1}{n} & c_j = 0 \end{cases}$$

All the elements in matrix A are strictly between 0 and 1 and all the column sums are equal to 1.
The **Perron-Frobenius theorem** states that a nonzero solution of the equation

$$x = Ax$$

exists and is unique to within a scaling factor. If this factor is chosen so that:

$$\sum_i x_i = 1,$$

then x is the **state vector** of the Markov chain and Google's PageRank. Its elements are strictly between 0 and 1 and it is the solution to the following linear system:

$$(I - A)x = 0$$

The best way to compute PageRank in MATLAB is defining matrix A as follows:

$$A = pGD + ez^T, \text{ where:}$$

- D = diagonal matrix formed from the out-degrees reciprocals:

$$d_{jj} = \begin{cases} 1/c_j & c_j \neq 0 \\ 0 & c_j = 0 \end{cases}$$

- e = $n-$vector of all ones
- z = vector with:

$$z_j = \begin{cases} \delta & c_j \neq 0 \\ 1/n & c_j = 0 \end{cases}$$

The assignment statement can now be written as follows:

$$(I - pGD)x = \lambda e,$$

where $\lambda = z^T x$ has the temporary value of 1.

1.3 Power Method

The **power method** is used to compute the dominant eigenvector λ_1 of a matrix A:

1. Start with an initial guess vector v_0:
2. Compute w in the following way:

$$w = Av_k$$

3. Set $v_{k+1} = ||w||$
4. Find the eigenvalue of v_{k+1} by using the Rayleigh quotient
5. Repeat from step 2 incrementing k by 1 until the difference between the previous and the current eigenvectors is lower than a certain threshold

In order to guarantee convergence, we assume that matrix A has an eigenvalue λ whose magnitude is strictly greater than the others and that vector x has a nonzero component in the direction of the eigenvector associated with λ.

1.4 Shift and Invert Method

Since the power method converges linearly and its error constant is $|\lambda_2/\lambda_1|$, the convergence could be too slow if $||\lambda_2||$ and $||\lambda_1||$ are close: the **shift and invert method** is used to improve the power method's converging rate.
Given a matrix A, we define matrix B as:

$$B = (A - \alpha I)^{-1}$$

The eigenvalues of B are defined as:

$$u_j = \frac{1}{\lambda_j - \alpha}.$$

By applying the power method to matrix B and assuming λ_2 is the closest-to-λ_1 eigenvalue of A, we obtain the following improved converging rate:

$$|\frac{u_2}{u_1}| = |\frac{\lambda_1 - \alpha}{\lambda_2 - \alpha}|$$

The cost of a power method iteration is given by a vector-matrix product, the cost of an iteration in the shift and invert method is given by a linear system.

2 Social Networks

2.1 Cholesky Factorization

The **Cholesky factorization** is the decomposition of a symmetric, positive-definite matrix A into the product of its lower triangular form L and its conjugate transpose L^T:

$$A = LL^T$$

It is twice as efficient as the LU decomposition.

2.2 Reverse Cuthill McKee Ordering

The **reverse Cuthill McKee ordering** is the permutation of a symmetric sparse matrix into a banded matrix with a smaller bandwidth. The resulting matrix has the non-zero elements closer to the diagonal.
In MATLAB, we can use the following implementation:

```
1) r = symrcm(A(2:end, 2:end));
2) prcm = [1 r+1];
```

This method is useful to reduce the number of fill-ins, making less expensive the Cholesky factorization.

2.3 Spectral Graph Partitioning

In **spectral graph partitioning**, we plant a random partition assigning values to the probabilities of intra-sets and inter-sets edges, then use the **Fiedler's vector**, which is the eigenvector v_2 associated with the second smallest eigenvalue λ_2, to find the partition.
In the Fiedler's vector, all indices of entries > 0 belong to one set and those of entries < 0 belong to the other. The partition minimizes the number of edges between the sets.

2.4 Degree Centrality

The **degree centrality** consists of ranking the number of incident links upon a node. For a given graph $G = (V, E)$ it is defined as the number of edges of a vertex v.
The **eigenvector centrality** measures the influence of a node in a network. The relationships between nodes with a high score contribute more to the node score.

3 Graph Partitioning

3.1 Graphs Matrices

3.1.1 Degree Matrix

A **degree matrix** D is a diagonal matrix containing the degree of each vertex. If the graph is weighted, the entries are the sum of all the weights of the connected edges.

3.1.2 Adjacency Matrix

An **adjacency matrix** is a square matrix representing the connections of an unweighted graph and its entries are defined as follows:

$$a_{ij} = \begin{cases} 1 & v_i \text{ connected to } v_j \\ 0 & \text{otherwise} \end{cases}$$

3.1.3 Weight Matrix

A **weight matrix** W is a square matrix representing the connections of a weighted graph and its entries are defined as follows:

$$w_{ij} = \begin{cases} w & v_i \text{ connected to } v_j \\ 0 & \text{otherwise} \end{cases}$$

3.1.4 Laplacian Matrix

The **Laplacian matrix** L is the matrix representation of a graph and is computed in the following ways:

- For an undirected and unweighted graph

$$L = D - A$$

- For an undirected and weighted graph

$$L = D - W$$

Matrix L only has real and non-negative eigenvalues and its eigenvectors are real and orthogonal.

3.2 Graph Partitioning

The **graph partitioning** problem is defined on a graph $G = (V, E)$ such that it is possible to partition G into smaller components with specific properties, generally with small cuts and equal-size partitions.

3.3 Spectral Bisection

Spectral bisection enables the decomposition of a symmetric matrix into eigenvalues within an orthonormal base of eigenvectors. The following procedure performs a spectral bisection:

1. Compute the Laplacian matrix L
2. Compute the Fiedler's vector w_2
3. Set a threshold to 0 or to the median of w_2
4. Choose $V_1 = \{v_i \in V | w_i < \text{threshold}\}$ and $V_2 = \{v_i \in V | w_i \geq \text{threshold}\}$
5. Return V_1 and V_2

Thresholding the values of w_2 around 0 results in two roughly equal-sized partitions with minimum edgecut, while thresholding around the median value of w_2 produces two strictly balanced partitions.

3.4 Inertial Bisection

Inertial bisection relies on the vertices' geometric coordinates: it finds the hyperplane running through the center of mass of the points.
In $2D$, the line is chosen such that the sum of squares of the distances of the nodes to the line is minimized.
The following procedure performs an inertial bisection:

1. Compute the center of mass of the points as follows:

$$x = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad y = \frac{1}{n}\sum_{i=1}^{n} y_i$$

2. Compute matrix M as follows:

$$S_{xx} = \sum_{i=1}^{n}(x_i - x)^2 \quad S_{yy} = \sum_{i=1}^{n}(y_i - y)^2 \quad S_{xy} = \sum_{i=1}^{n}((x_i - x)(y_i - y))$$

$$M = \begin{bmatrix} S_{xx} & S_{xy} \\ S_{xy} & S_{yy} \end{bmatrix}$$

3. Compute the smallest eigenvalue of matrix M and the associated eigenvector u
4. Minimize the distance of the nodes to the line:

$$u^T M u$$

5. Project each point to the line and compute the median to partition the nodes
6. Return V_1 and V_2

3.5 Recursive Bisection

Recursive bisection depends on the decisions made during the earlier stages of the process and lacks of global information, meaning that it may result in suboptimal partitions.

3.6 $K-$Way Partitioning

$K-$**way partition** starts with partitioning a small set of vertices, then projects it back towards the original set to refine it. The main difference with recursive bisection is that $k-$way partitioning stores global information about the graph.

3.7 Partitioning Metrics

The number of cut edges between partitions determines the result's quality. The size of an **edgecut** partitioning the graph into two complementary vertices subsets is defined as follows:

$$cut(V_1, V_2) = \sum_{i \in V_1, j \in V_2} w_{ij}$$

The **cardinality**, i.e. the number of nodes, of each subset is given by:

$$x^T x = ||x||_2^2 = |V_1|$$

Let $\tilde{V}$ be the average partition weight, then for a $k-$way partition the **load imbalance** b^k is defined as follows:

$$b^k = max_{|V_i^k|} \frac{|V_i^k|}{|\tilde{V}^k|}$$

$$b^k = 1 + b_r^k \geq 1$$

$$b_r^k \geq 0$$

It characterizes the deviation from obtaining a balanced partitioning.
The optimal value for b_r^k is 0 and implies that the k partitions contain the same number of nodes.

4 Spectral Graph Clustering

4.1 Graph Clustering

Given a set of data points $x_1, ..., x_n$ and some notion of similarity $s_{ij} \geq 0$, clustering extracts information by grouping similar data points.

4.2 Trees

A **tree** is an undirected graph in which any two vertices are connected by exactly one path. A **minimum spanning tree** is a subgraph including all the vertices of the graph, the minimum possible number of edges and the minimum possible total weight.

4.3 Similarity Graphs

In **similarity graphs** two vertices v_i and v_j are connected based on the similarity s_{ij} between the corresponding data points x_i and x_j. The edge between them is weighted by s_{ij}.
The goal is to model the neighborhood relationships between the data points.

4.3.1 $\epsilon-$Neighborhood Graph

In $\epsilon-$neighborhood graphs, we connect two vertices if their pairwise distances are smaller than ϵ. Weighting the edges would not incorporate more information about the data to the graph, thus they are unweighted.

4.3.2 $k-$Nearest Neighbor Graphs

In $k-$**nearest neighbor graphs**, we connect vertices v_i and v_j if the latter is among the $k-$nearest neighbors of the first. The graph is directed and we have two ways to make it undirected: either we ignore the directions obtaining a simple $k-$nearest neighbors graph, or we connect the vertices if each of them is among the $k-$nearest neighbors of the other, obtaining a **mutual** $k-$**nearest neighbors graph**.

4.3.3 Fully Connected Graph

In **fully connected graphs**, we connect two vertices if the similarity $s_{ij} > 0$. This construction is useful only if the similarity function models local neighborhoods.

4.4 Spectral Clustering

In **spectral clustering**, we create clusters with the same number of nodes through the following procedure:

1. Compute the minimum spanning tree to determine ϵ
2. Create the $\epsilon-$neighbourhood graph
3. Compute the adjacency matrix W as follows:

$$W = S \odot G$$

4. Compute the Laplacian matrix
5. Compute the $k-$smallest eigenvectors based on L
6. Use k-means algorithm to cluster the nodes
7. Use the eigenvector matrix in order to obtain the clusters

4.5 K-Means Clustering

The **k-means clustering** divides the clusters based on the distance between a node and a **centroid**, which represents the center of a cluster. The algorithm is the following:

1. Start with some random centroids
2. Assign each node to the nearest cluster based on its centroid
3. Recompute the centroids as the mean of the points in the cluster
4. Repeat from step 2 until no further improvement can be made

5 Image Deblurring and Conjugate Gradient

5.1 Blurred Image Problem Definition

Let matrix $B \in \mathbb{R}^{n^2 \times n^2}$ be a blurred image we want to deblur given the transformation and let $X \in \mathbb{R}^{n \times n}$ be the original square and greyscale image matrix where each entry corresponds to a pixel value. We can perform **vectorization** by converting matrix X into a column vector $x \in \mathbb{R}^{n^2}$ and matrix B into vector $b \in \mathbb{R}^{\ltimes^{\not\vDash}}$.
We can write the following equation:

$$Ax = b,$$

where $A \in \mathbb{R}^{n^2 \times n^2}$ indicates the transformation matrix coming from the repeated application of the image kernel. A blurred pixel is the weighted average of the surrounding ones, with the weights defined by the kernel matrix $K \in \mathbb{R}^{d \times d}$. The non-zero elements of each row of matrix A correspond to the values of matrix K. Matrix A ignores the elements outside the matrix borders:

$$(max(i, j) > n)$$

Matrix A is a d^2-banded matrix, where:

$$d << n,$$

meaning that matrix A is sparse.

5.2 Direct and Iterative Methods

Since the complexity of Gaussian elimination for large linear systems is too high, we need another approach: we use **direct methods** to compute the exact solution in n steps, while we use **iterative methods** with an arbitrary starting point to compute an approximate solution. Iterative methods converge after a few iterations, but they are less robust and not as general as direct ones.

5.3 Error and Residual

Let x be an exact solution and x^m be the computed point, then the **error** in step m is the deviation of the computed point from the exact solution:

$$e^m = x - x^m,$$

The error is not known during the iterations, otherwise we would know the solution.
The **residual** provides us with a measure of the real error:

$$r^m = b - Ax^m$$

5.4 Steepest Descent

The **steepest descent** algorithm is a precursor to the Conjugate Gradient algorithm that performs the following procedure:

1. Start with a random initial guess
2. Take the **gradient**, i.e. the direction of the deepest descent
3. Compute the minimum of the gradient
4. Take the new gradient such that it is orthogonal to the previous one
5. Repeat from step 2 until a certain convergence criterion is met

This algorithm has a low convergence rate, but it does not take optimal steps to find the next approximate solution.

5.5 Conjugate Gradient

The **conjugate gradient method** is used to solve equations of the type:

$$Ax = b,$$

where A is symmetric and positive-definite. It can be used as an iterative method since it provides monotonically improving approximations x_k to the exact solution after each iteration, with a number of iterations not larger than the size of the matrix, in the absence of round-off error. The improvement is typically linear and its speed is determined by the **condition number** $\kappa(A)$ of the matrix A:

$$\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}},$$

with σ indicating the singular values of A. For a real symmetric matrix, the values are equal to the eigenvalues absolute value:

$$\sigma = |\lambda|$$

The larger the condition number is, the slower the improvement.

The algorithm starts from an initial guess x_0 and applies a series of operations until the desired tolerance is reached:

1. $r = b - Ax_0$
2. $d = r$
3. $\rho_{old} =< r, r >$
4. $for\ i = 0, 1, ...\ do:$
    1. $s = A \cdot d_i$
    2. $\alpha = \rho_{old} / < d, s >$
    3. $x = x + \alpha \cdot d$
    4. $r = r - \alpha \cdot s$
    5. $\rho_{new} =< r, r >$
    6. $\beta = \rho_{new} / \rho_{old}$
    7. $d = r + \beta \cdot d$
    8. $\rho_{old} = \rho_{new}$
5. $end\ for$

Since matrix A is not positive-definite we must solve the following equations:

$$A^T Ax = A^T b \ \rightarrow \ \tilde{A}x = \tilde{b}$$

The pre-multiplication with A^T results in the positive-definite augmented transformation matrix $\tilde{A}$.
The condition number is the relation of the sensitivity of the solution x to changes in b: if small changes result in large changes in x, then the system is **ill-conditioned** and its condition number is large. The convergence rate of the algorithm is hindered when the system has a high condition number.

5.6 Preconditioned Conjugate Gradient

In the **preconditioned conjugate gradient method** (PCG), a symmetric and positive-definite **preconditioner** P is selected such that:

$$P^{-1}\tilde{A} \approx I$$

Furthermore, we can decompose the preconditioner such that:

$$P = LL^T,$$

where L is the Cholesky factor.

Now we have to solve the preconditioned augmented system:

$$P^{-1}\tilde{A}x = P^{-1}\tilde{b}$$

$$(L^{-1}\tilde{A}L^{-1})(Lx) = L^{-1}\tilde{b}$$

This is done to decrease the condition number and the range of the eigenvalues. The preconditioner should be computationally inexpensive to find. We can use the **incomplete Cholesky factorization** to compute the Cholesky factorization of the non-zero elements of $\tilde{A}$, returning the following preconditioner:

$$P = F^T F,$$

where F is the sparse incomplete Cholesky factor. Due to the fact that the routine fail since the existence of F is not guaranteed, a heuristic approach is used to apply a diagonal shift of P, enforcing positive-definiteness and making F computable.
PCG would be worth the added computational cost whether the conditional number $\kappa(A)$ is large, i.e. matrix A is ill-conditioned since with CG it will require a lot of iterations. While deblurring lots of images with PCG, the computational cost would decrease since it is possible to use the same precondition in order to save time.

6 Linear Programming and the Simplex Method

6.1 Linear Programming

Linear Programming is an optimization technique used to either maximize or minimize a linear objective function subject to linear equality and inequality constraints. An example of a linear program is the following:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{1,j} x_j \leq h_1 \\ & \vdots \\ & \sum_{j=1}^{n} a_{m,j} x_j \leq h_m \end{aligned}$$

Vector x_i must satisfy some constraints and the non-negativity condition.
Linear programming problems can be written in the following **standard form**:

$$\begin{array}{rl} \max & z = c^T x \\ \text{s.t.} & Ax \leq h \\ & x \geq 0 \end{array} \qquad \begin{array}{rl} \min & z = c^T x \\ \text{s.t.} & Ax \geq h \\ & x \geq 0 \end{array}$$

where z is the value of the objective function, $c \in \mathbb{R}^n$ is the coefficients vector, $x \in \mathbb{R}^n$ is the unknowns vector, $A \in \mathbb{R}^{m \times n}$ is the coefficients matrix, and $h \in \mathbb{R}^m$ is the vector of the constraints coefficients.
Given that the **feasible region** is generated by the vertices that satisfy the constraints, the optimal value can be found with the **Fundamental Theorem of Linear Programming**:

Theorem 1 *If a linear program admits a solution, it will lie on a vertex of the polytope defined by the feasible region. If two vertices are both maximizers or minimizers of the function, then all the points lying on the segment between them will represent the optimal solutions to the problem.*

6.2 The Simplex Method

Since computing the value of z at all the vertices of the feasible region could be expensive, we use the **simplex method**, which has an exponential worst-case complexity.

6.2.1 Slack and Surplus Variables

The method introduces **slack variables** for maximization and **surplus variables** for minimization to the standard form of the problem, denoting them with s_m as in the following example:

$$\begin{aligned} \max \quad & z = 3x + 2y \\ \text{s.t.} \quad & x + 2y + s_1 = 4 \\ & x - y + s_2 = 1 \\ & x, y \geq 0; \quad s_1, s_2 \geq 0 \end{aligned} \qquad \begin{aligned} \min \quad & z = 3x + 2y \\ \text{s.t.} \quad & x + 2y - s_1 = 4 \\ & x - y - s_2 = 1 \\ & x, y \geq 0; \quad s_1, s_2 \geq 0 \end{aligned}$$

6.2.2 Basic and Nonbasic Variables

We can swap the rows of matrix A as long as we swap the elements of vector h. Furthermore, we can swap the columns of matrix A as long as we swap the elements of vector x. Matrix A can be split in two submatrices:

$$A = [B \ D],$$

where matrix B contains the linearly independent columns of A, while matrix D contains the remaining ones. We can split vectors x and c in the same way:

$$x = \begin{bmatrix} x_B \\ x_D \end{bmatrix} \qquad c = \begin{bmatrix} c_B \\ c_D \end{bmatrix}$$

Finally, we obtain the following relation:

$$x_B = B^{-1}h - B^{-1}Dx_D$$

Vector x_B contains the **basic variables**, while vector x_D contains the **nonbasic variables**. By setting $x_D = 0$, we obtain the following equation:

$$x_B = B^{-1}h,$$

and if the non-negativity condition for x_B is satisfied, then this is a **basic solution** and corresponds to the feasible region. If x_B has some zero values, then the solution is **degenerate**.
The existence of a feasible solution implies the existence of a feasible basic solution, and the existence of an optimal solution implies the existence of an optimal basic solution.
The number of possible basic solutions grows exponentially with unknowns and constraints. The maximum possible number of iteration N is:

$$N = \frac{(m+n)!}{m!n!}$$

6.2.3 Optimality Condition

We must check if the solution satisfies the **optimality condition**, based on the basis of the **reduced cost coefficients**, at every iteration:

$$r_D = c_D^T - c_B^T B^{-1} D,$$

where c_B is the basic coefficient vector, c_D is the nonbasic coefficient vector, B is the basic matrix, and D is the nonbasic matrix.
The optimality condition for maximization is $r_D \leq 0$, while it is $r_D \geq 0$ for minimization. Both are satisfied when all the components of the reduced cost coefficients vectors are less or bigger than 0, respectively.

6.2.4 Iterative Rule

If the optimality condition is not met, we take the variable with the highest reduced cost coefficient, in the case of maximization, or the variable with the lowest cost coefficient, in the case of minimization, into the basis.
If two variables have the same value, we could end up swapping them over and over again creating a **cycle**. The **iterative rule** identifies the variables that must be taken out of the basis:

$$\frac{B^{-1}h}{B^{-1}D}$$

We select the ratio with the smallest positive value.

6.2.5 Simplex Method Procedure

The method procedure can be summarized as follows:

1. Write the problem in standard form
2. Add slack or surplus variables
3. Apply the iterative rule by exchanging basic with nonbasic variables
4. Repeat from step 3 until the optimality condition is met

6.3 The Auxiliary Problem

We could find a feasible initial basic solution by solving an **auxiliary problem**, which can be defined by introducing the **artificial variables** u_m:

$$\begin{aligned}
\max \quad & z_{\text{aux}} = \sum_{i=1}^{n} u_i \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{1,j} x_j + s_1 + u_1 = h_1 \\
& \vdots \\
& \sum_{j=1}^{n} a_{m,j} x_j + s_m + u_m = h_m \\
& x_i, ..., x_n \geq 0; \;\; s_1, ..., s_m \geq 0; \;\; u_1, ..., u_m \geq 0
\end{aligned}$$

The auxiliary problem aims at minimizing the sum of artificial variables and its optimal solution would be achieved when all the artificial variables are $= 0$. If $z_{aux} = 0$ is not achieved, then the auxiliary problem and the original problem do not admit a feasible solution.
In order to obtain a starting basic solution for the auxiliary problem, we can set the original and the slack variables to 0 and the artificial variables equal to the right-hand side.