Beneath the surface of centrality

A computational approach to network theory

Anne Bomer

Supervised by dr. Sarah Gaaf

A thesis presented for the degree of
Bachelor of Mathematics

Department of Mathematics
Utrecht University
The Netherlands
July 2018

Contents

1 Introduction

A perceptive reader might ask him or herself upon seeing the title: what more could there be to a simple concept such as centrality? Everybody knows: Utrecht is the most central place in the Netherlands. And for the average person this intuitive approach is sufficient. But imagine a herring trader, who wants to transport his fish, using the fastest route to maintain absolute freshness. He might store his fish in Utrecht, because people told of its centrality. But the fisherman, who wants to make as much money as possible, has many questions. Utrecht may be in the middle of the country, but is it well connected to specific places in the Netherlands where most herring is sold? If I choose to store herring in Utrecht, is there a good connection from the North Sea going into Utrecht? If a main road leaving or going into Utrecht is closed, are there well-connected alternatives? And which of the above issues will have the most impact on my business? Especially in the Netherlands, with its complex and extensive infrastructure, these questions can quickly become mathematical problems. Meanwhile, in other large networks the question of centrality is also worth serious examination, such as in on- or offline social networks (who are the influencers?), public transport (scheduling, rescheduling, rerescheduling), search engines (links between websites) or the interaction of cells of the human body. The importance of these areas of life drove us to the following question: how can we examine network connectivity and compute this in an actual, real-world situation? It turns out the mathematical concept of centrality is very helpful in answering this, hence the emphasis in the title.

Now we have intuitively demonstrated the importance of the examination of centrality above, we can assure the reader that centrality is actually a well-defined mathematical concept. In this paper, we will first introduce this and related notions, and the mathematics it is embedded in. Thereafter we deal with questions of how these concepts and theory can be applied in a practical way, to proceed with some numerical experiments. Finally, we contrast our mathematical frame with a case wherein the centrality concept is used for determining port hierarchy, to conclude with an analysis of the implications of the differences found. Thereafter we will share our findings and conclusions regarding the research question.

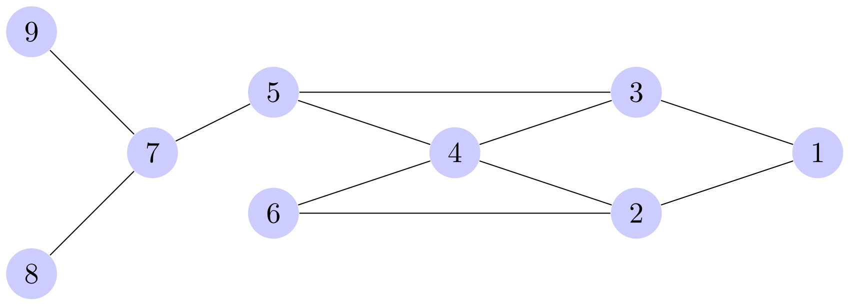
During the research process, we emphasized the exploration of the subject's practical and applied sides, in order to engage the reader's mind to the broader implications of a scientific discipline that can be extremely theoretical, and for our study to be educational and accessible for the reader that is a (relative) layman.

2 Network Properties

2.1 Introduction to network theory

In Applied Mathematics, it is often so that real-life problems have to be modeled in such a way that we can get more information about this problem. A particular branch of this modeling is network theory. Being a subdivision of graph theory, in network theory one studies graphs as a model of symmetric or asymmetric relations between discrete objects. To illustrate intuitively the general practice and value of such studies, we will start with an example.

Example Consider nine cities, connected with a two-way road (so you can travel in both directions) to at least one of the other cities. We can represent a possible network between these cities with a graph:



Intuitively we can understand how the different cities are linked to one another, and how you can get from one city to another. When a person from city 1 wants to do grocery shopping in city 4, and then return, he or she can travel via city 2 to 4 and then back to city 1 via 3. There are of course other options.

These types of networks, where different objects (in this case; cities) are in some way connected to each other (in this case; with a road), are abundant in the world. Other examples include:

- People connected through social networks
- Public transport routes
- Trade routes

Now we have introduced a basic notion of the subject we are dealing with, we progress to a formal mathematical introduction.

2.2 Graphs and Adjacency matrices

The following basic definitions are commonly used in network theory. These are derived from Handbook of Graph Theory [10].

Definition 2.1 A *graph* is an ordered pair $G = (V, E)$ consisting of a set V of nodes and a set E of lines. For nodes $\{x, y\} \subset V$ that are connected by a line, there is an element $(x, y) \in E$.

Under certain circumstances, a graph can be represented by an adjacency matrix.

Definition 2.2 When a graph $G = (V, E)$ is finite (it has a finite number of nodes), it can be represented by a square $|V| \times |V|$ matrix A where its elements $a_{ij} = 1$ if node i and j are connected with a line, and $a_{ij} = 0$ if not. This matrix is called an *adjacency matrix.* Note that for an adjacency matrix, it does not make a difference whether G has multiple (or an infinite amount of) lines between the same nodes.

The adjacency matrix B of our example is the following:

$$B = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

In our example, one can travel both ways. Thus, we have an undirected graph; a line (x, y) is not distinguished from (y, x). This means we have a symmetric adjacency matrix ($b_{ij} = b_{ji}$). Also, no city is connected to itself, this makes sure that our diagonal entries are zero. In the remainder of this paper, we will assume an adjacency matrix A is symmetric and has diagonal entries equal to zero, unless noted otherwise.

Definition 2.3 A *network* is an undirected graph with N nodes. It can be represented by adjacency matrix $A \in \mathbb{R}^{N \times N}$.

2.3 Centrality and the exponential matrix function

In graph theory, the importance of a node is determined by its *centrality*. Before defining centrality, we should ask: what characterizes an important node? In our example, should we choose the best-connected city (with the most roads connecting it to others), or the city that is indispensable (necessary to pass through)? The best-connected city is 4. City 5 has one connection less, but it's the only connection from 7, 8 and 9 to the other ones. So 4 is important for tourists who want to travel between different cities in a short time, where 5 is important for workers that work in 1 but live in 9. Because of these differing possibilities, different ways to define a centrality measure have emerged in the literature.

In recent times, Estrada and Higham[9] have defined an influential centrality measure. They first display a few centrality measures in the literature, after which they introduce their own. In this paragraph, I will closely follow their treatment of the subject, with the addition of my own explanations of how to interpret this in our example.

Definition 2.4 Given an adjacency matrix A of dimension N and the $N \times 1$ vector $\mathbf{e}$ with all values equal to one, the *degree* of node i is given by:

$$\deg_i = \sum_{k=1}^{N} a_{ik} = (A\mathbf{e})_i \tag{1}$$

This index corresponds with the number of lines that have i as an endpoint. In our example, $\deg_1 = 2$.

They proceed to look at the meaning of a squared adjacency matrix. Defined as $A^2_{ij} = \sum_{k=1}^{N} a_{ik}a_{kj}$, it counts the number of nodes that are connected to both i and j. In our example, city 2 and 3 are both connected to 1 and 4, so we get $B^2_{14} = 2$. To develop this idea, the notion of a walk is introduced.

Definition 2.5 A *walk* of length n is a list of nodes $i, k_1, k_2, \ldots, k_{n-1}, j$ such that $a_{i,k_1} = a_{k_1,k_2} = \ldots = a_{k_{n-1},j} = 1$, that is, successive nodes are connected. The values $k_1, k_2, \ldots, k_{n-1}$ do not have to be different, and can be equal to i or j. For the case that $i = j$, we have a *closed walk.*

Thus, A^2_{ij} can be interpreted as the number of walks of length two from i to j. Then what is the meaning of higher matrix powers? The authors introduce the following lemma as an answer(for a proof see [9]):

Lemma 2.1 The identity

$$A^n_{ij} = \sum_{k_1=1}^{n} \sum_{k_2=1}^{n} \cdots \sum_{k_{n-2}=1}^{n} \sum_{k_{n-1}=1}^{n} a_{i,k_1}a_{k_1,k_2} \cdots a_{k_{n-2},k_{n-1}}a_{k_{n-1},j}$$

counts the number of different walks ($i \neq j$) or closed walks ($i = j$) of length n between nodes i and j.

2.3.1 Centrality measures

Now we have the tools to define some centrality measures. The authors argue that the degree might not be a good option: *The degree, however, paints a very localized picture of a node's importance; it does not distinguish between edges that connect to well-connected or poorly connected nodes.* [Estrada and Higham, 2010, pp. 698] Note that $A_{ii}^2 = \sum_{k=1}^{N} a_{ik}a_{ki} = \deg_i$. Thus, the degree may be interpreted as a closed walk of length 2 in node i. They argue that longer closed walks also carry value; a better-connected node should also be able to form longer ones. With this reasoning, they consider $(A^n)_{ii}$ for all $n = 2, 3, \ldots$ of importance, although they argue that shorter walks may be more important; *information is passed more quickly and efficiently* [Estrada and Higham, 2010, pp. 698]. Thus they decide to scale closed walks based on length, choosing $1/(n!)$ to do so. This results in the following centrality measure:

$$\Big(\frac{A^2}{2!} + \frac{A^3}{3!} + \ldots + \frac{A^k}{k!} + \ldots\Big)_{ii}$$

Since $a_{ii} = 0$ (nodes are not self-connected) and the fact that relative ordering does not change by adding a constant (say 1), we choose

$$\Big(I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + \ldots + \frac{A^k}{k!} + \ldots\Big)_{ii},$$

where I is an identity matrix of size $N \times N$. The above is equivalent to

$$(\exp(A))_{ii}, \tag{2}$$

where $\exp(\cdot)$ is the matrix exponential function.

Definition 2.6 The *centrality* of node i is equivalent to $(\exp(A))_{ii}$.

2.3.2 Communicability

The idea of the importance of weighted walks can be extended to walks that are not closed. Walks of different length between some nodes i and j can provide knowledge of how quickly and efficiently information is passed through different parts of a network.

Definition 2.7 The *communicability* between node i and j is equivalent to

$$(\exp(A))_{ij}. \tag{3}$$

2.3.3 Betweenness

In our example network, we noted a situation wherein city 5 was more important than 4, since without it some cities couldn't be reached. This suggests it might be useful to analyze changes in communicability when a node is removed. If for a node r, we let $A - E(r)$ indicate the adjacency matrix of the network with r removed, where $E(r)$ is an $N \times N$ matrix with row and column r the same as A, and zeros elsewhere.

Definition 2.8 The *betweenness* of a node is equivalent to

$$\frac{1}{(N-1)^2 - (N-1)} \sum_{i \neq j, i \neq r, j \neq r} \sum \frac{(\exp(A)_{ij}) - \exp(A - E(r))_{ij}}{\exp(A)_{ij}}, \tag{4}$$

where $(N-1)^2 - (N-1)$ equals the number of terms in the sum, so we get a value between 0 and 1. Also, $N \geq 3$.

3 How to apply?

3.1 Example results

For our example, we can compute the above measures in MATLAB. Given adjacency matrix $A^* \in \mathbb{R}^{9\times 9}$ MATLAB computes (rounded to two decimal points):

$$\exp(B) = \begin{pmatrix} 2.61 & 2.39 & 2.40 & 2.32 & 1.48 & 1.40 & 0.39 & 0.08 & 0.08 \\ 2.39 & 3.92 & 2.40 & 3.80 & 1.85 & 2.93 & 0.47 & 0.09 & 0.09 \\ 2.40 & 2.40 & 4.00 & 3.86 & 3.26 & 1.77 & 1.16 & 0.30 & 0.30 \\ 2.32 & 3.80 & 3.86 & 5.48 & 3.63 & 3.31 & 1.25 & 0.31 & 0.31 \\ 1.48 & 1.85 & 3.26 & 3.63 & 3.92 & 1.56 & 2.21 & 0.77 & 0.77 \\ 1.40 & 2.93 & 1.77 & 3.31 & 1.56 & 3.01 & 0.41 & 0.08 & 0.08 \\ 0.39 & 0.47 & 1.16 & 1.25 & 2.21 & 0.40 & 3.06 & 1.61 & 1.61 \\ 0.08 & 0.09 & 0.30 & 0.31 & 0.77 & 0.08 & 1.61 & 1.64 & 0.64 \\ 0.08 & 0.09 & 0.30 & 0.31 & 0.77 & 0.08 & 1.61 & 0.64 & 1.64 \end{pmatrix}$$

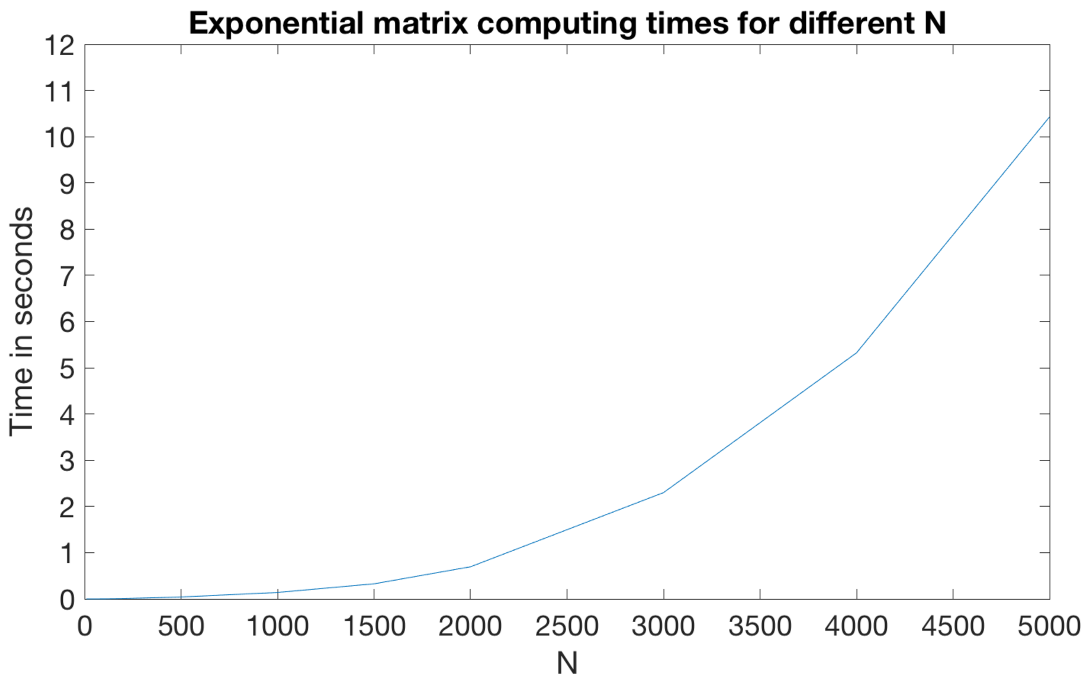
Betweenness is given by the following table:

Node	Betweenness
1	0.0923
2	0.1956
3	0.3333
4	0.5136
5	0.6158
6	0.0983
7	0.4798
8	0.0253
9	0.0253

In our example, we can see the highest centrality, communicability and betweenness indices are respectively given by $B_{44} = 5.48$, $B_{34} = B_{43} = 3.86$ and 0.62. This means that, according to our definitions, city 4 has the best-connected closed walks, city 3 and 4 are have the best connected routes between each other compared to routes between other cities, and city 5 is the most important for overall communicability between cities.

In many real situations, however, adjacency matrices may be of much bigger dimensions than our example matrix. This does not constitute a problem per se, but the command *expm* that is used to calculate the exponential matrix gets slower very rapidly when matrix dimensions rise. On the next page we included a figure that illustrates this for rising N.

Figure 1



We can see that the computing times of expm raise exponentially. Even for a relatively small matrix ($N = 5000$) the computing time is already 10.4 seconds. From this we conclude that for large matrices it is not feasible to calculate the matrix exponential using the expm command in MATLAB, due to the large computation times. One method to lower these is to approximate specific matrix elements, using a cheaper computational method, which is what we will be doing. An often used method in the literature is based on Gauss Quadrature and the Lanczos algorithm. The goal of this method is to produce suitably tight lower and upper bounds for the desired values, namely the entries of $f(A)$. We will give a description of this method in the rest of this chapter, closely based on the one in paragraphs 10.1 and 10.2 in Matrix Computations [6] by Golub and Van Loan. Therefore, only when we present a specific or crucial detail or deviate from this source we will use citations. Also, for reasons of simplicity, the notation in the book is kept the same in this paper.

3.2 Reformulation as an integral

Assume there is a large sparse symmetric positive definite matrix $A \in \mathbb{R}^{N\times N}$ with eigenvalues in some interval $[a, b]$. Then let $f(\lambda)$ be a given smooth function defined on this interval. Given some $u \in \mathbb{R}^N$, we can produce lower and upper bounds b and B such that:

$$b \leq u^T \cdot f(A) \cdot u \leq B. \tag{5}$$

When we choose $u = \mathbf{e}_i$ and $f(A) = \exp(A)$ we get $b \leq \exp(A)_{ii} \leq B$ from the equation above. However, with this choice of u we can only approximate the diagonal of $\exp(A)$.

For off-diagonal elements we can use the identity [12]

$$\mathbf{e}_i{}^T \cdot f(A) \cdot \mathbf{e}_j = \frac{1}{4}\Big[(\mathbf{e}_i + \mathbf{e}_j)^T \cdot f(A) \cdot (\mathbf{e}_i + \mathbf{e}_j) - (\mathbf{e}_i - \mathbf{e}_j)^T \cdot f(A) \cdot (\mathbf{e}_i - \mathbf{e}_j)\Big]$$

and choose $u_1 = \mathbf{e}_i + \mathbf{e}_j$, $u_2 = \mathbf{e}_i - \mathbf{e}_j$ to produce two different bounds, which we can substitute into the above equality.

Note that $\exp(A)$ is a smooth function defined on $\mathbb{R}$, and since we are dealing with large networks, it is reasonable to assume that A is sparse (most of its elements are zero). Also, A is symmetric, finite and its elements are nonnegative, wherefore it has N real eigenvalues contained in some interval $[a, b]$. We conclude that A satisfies the conditions given by Golub [6] wherefore it is possible to derive bounds for this situation.

To proceed with a method based on quadrature, we must regard $u^T f(A)u$ as a *Riemann-Stieltjes integral.* Given a real-valued integrand $f(x)$ and weight function $w(x)$, the Riemann-Stieltjes integral is denoted by

$$I(f) = \int_a^b f(x)dw(x) \tag{6}$$

and defined as a limit of sums of the form

$$S_K = \sum_{\mu=1}^{K} f(c_\mu)(w(x_\mu) - w(x_{\mu+1}))$$

where $a = x_K < \cdots < x_1 = b$ and $x_{\mu+1} \leq c_\mu \leq x_\mu$.

Now suppose $a = \lambda_N < \cdots < \lambda_1 = b$ and that

$$w(\lambda) = \begin{cases} w_{N+1}, & \text{if } \lambda < a \\ w_\mu, & \text{if } \lambda_\mu \leq \lambda < \lambda_{\mu-1}, \qquad \mu = 2, \ldots, n \\ w_1, & \text{if } b \leq \lambda \end{cases} \tag{7}$$

where $0 \leq w_{n+1} \leq \cdots \leq w_1$. When we observe S_K as $K \to \infty$, we find that

$$\int_a^b f(\lambda)dw(\lambda) = \sum_{\mu=1}^{N} (w_\mu - w_{\mu+1}) \cdot f(\lambda_\mu) \tag{8}$$

We will now illustrate that $u^T f(A)u$ can be considered as a Riemann-Stieltjes integral. When we let $A = X\Lambda X^T$, we get

$$u^T f(A)u = (X^T u)^T \cdot f(\Lambda) \cdot (X^T u) = \sum_{\mu=1}^{N} [X^T u]_\mu^2 \cdot f(\lambda_\mu). \tag{9}$$

When we choose weight function $w_\mu = [X^T u]_\mu^2 + \cdots + [X^T u]_N^2$ for $\mu = 1, \ldots, n+1$ and combine equation (8) and (9) we get

$$\int_a^b f(\lambda) dw(\lambda) = \sum_{\mu=1}^{N} [X^T u]_\mu^2 \cdot f(\lambda_\mu) = u^T f(A) u. \qquad (10)$$

We have now shown the desired equivalency. Now we know how to construe a Riemann-Stieltjes integral we can approximate for all our desired matrix elements, we will proceed to some Gauss approximation rules.

3.3 Gauss Quadrature Bounds

For the integral $I(f)$ a Gauss quadrature rule is an approximation of an integral computed by a weighted sum of f-values on its domain. The points (*nodes*) and weights are suitably chosen to give an exact solution for polynomials up to a degree related to accuracy parameter k. We will now introduce apposite rules for our research.

3.3.1 Gauss Rule

Compute weights $w_1, \ldots, w_k$ and nodes $t_1, \ldots, t_k$ such that

$$I_G(f) = \sum_{i=1}^{k} w_i f(t_i) = I(f) + R_G(f) \qquad (11)$$

where the error $R_G(f)$ is equal to

$$-\frac{f^{(2k)}(\eta)}{(2n)!} \int_a^b \Big[\prod_{i=1}^{k} (\lambda - t_i) \Big]^2 dw(\lambda), \qquad a < \eta < b.$$

Note that for all polynomials of degree $2k-1$ or less, the error is equal to zero.

3.3.2 Gauss-Radau Rules

Compute weights $w_a, w_b, w_1, \ldots, w_k$ and nodes $t_1, \ldots, t_k$ such that

$$I_{GR(a)}(f) = w_a f(a) + \sum_{i=1}^{k} w_i f(t_i) = I(f) + R_{GR(a)}(f) \qquad (12)$$

and

$$I_{GR(b)}(f) = w_b f(b) + \sum_{i=1}^{k} w_i f(t_i) = I(f) + R_{GR(b)}(f) \qquad (13)$$

where the errors $R_{GR(a)}(f)$ and $R_{GR(a)}(f)$ are respectively equal to

$$-\frac{f^{(2k+1)}(\eta)}{(2n+1)!} \int_a^b (\lambda - a) \Big[\prod_{i=1}^{k} (\lambda - t_i) \Big]^2 dw(\lambda), \qquad a < \eta < b,$$

and

$$-\frac{f^{(2k+1)}(\eta)}{(2n+1)!}\int_a^b(\lambda-b)\Big[\prod_{i=1}^{k}(\lambda-t_i)\Big]^2dw(\lambda), \qquad a<\eta<b.$$

Note that for all polynomials of degree $2k$ or less, the errors are equal to zero.

3.3.3 Bounds

It is known [4, 6] that the Gauss and Gauss-Radau(b) rules can be used to compute a lower bound, wheres the Gauss-Radau(a) rule can be used to compute an upper bound on $u^Tf(A)u$, if f is *strictly completely monotonic* (s.c.m.) on the interval containing the spectrum of A. A function is s.c.m. on an interval $[a,b]\in\mathbb{R}$ if $f^{(2k)}>0$ and $f^{(2k+1)}<0$ for all $k=0,1,2,\ldots$, where $f^{(0)}\equiv f$. Since we are interested in $\exp(A)$, which is not s.c.m., we can write $\exp(A)=\exp(-(-A))$, which is s.c.m. [4]. Thus, we can now regard the bounds b and B in equation (5) as

$$I_G(f)\le u^Tf(A)u\le I_{GR(a)}(f). \tag{14}$$

It is however not necessary to explicitly compute the Gauss nodes and weights, because we can generate these with the Lanczos algorithm, which will be explained in the next paragraph.

3.4 Gauss Quadrature via Lanczos

When we consider the Gauss Rule, the quantity we seek to compute has the form $\sum\limits_{i=1}^{k}w_if(t_i)$. This boils down to the computation of entries and spectral information on a certain tridiagonal matrix that we can generate with the Lanczos algorithm. Golub and Van Loan proceed immediately with a procedure to calculate this, so to understand this connection, we will first provide some background information based on the article Gaussian Quadrature and the Eigenvalue Problem [7] by John A. Gubner. We will provide a brief summary for background, and direct the interested reader to Gubner's article for a more thorough explanation.

3.4.1 Interpolation

Equation 10 shows that we have to calculate some integral $\int f(x)dw(x)$. Quadrature is the approximation of this integral by another integral $\int \hat{f}(x)dw(x)$ where $\hat{f}$ is a function close to f. We call $\hat{f}$ an interpolating polynomial. Interpolation is an estimation of a value within two known values in a sequence of values. Polynomial interpolation is the interpolation of a given data set by the polynomial of lowest possible degree that passes through the points of the dataset. [1, 2] Gubner shows that when an interpolating polynomial is of degree less than k, which means that $\hat{f}(x_i)=f(x_i)$ for $i=1,\ldots,k$, the second integral can be expressed as

$$\sum_{i=1}^{k}w_if(t_i),$$

where nodes t_i are in the range of integration and weights w_i can be calculated. Also, once arbitrary nodes are chosen in the integration interval, it is possible to choose weights such that for any polynomial f of degree less than k

$$\int f(x)dw(x) = \int \hat{f}(x)dw(x) = \sum_{i=1}^{k} w_i f(t_i). \tag{15}$$

With Gauss Quadrature the nodes are carefully chosen, which makes it possible to extend this equality to polynomials of degree $2k-1$ or less. How these nodes are chosen follows from the next section.

3.4.2 Extending the equality

Our goal is to show that there is a way to choose nodes such that equation 15 holds for polynomials up to degree $2k-1$. We will first remark that any polynomial f can be written as

$$f = q\zeta + r, \qquad\qquad \deg r < \deg \zeta = k, \tag{16}$$

where $\zeta(x) := (x-x_1)\cdots(x-x_k)$ is a k-degree interpolating polynomial, q is some polynomial and r is some polynomial with deg $r < k$. [7]

Since deg $r < k$ we get

$$\int r(x)dw(x) = \sum_{i=1}^{k} w_i r(x_i).$$

Also,

$$r(x_i) = f(x_i) - q(x_i)\zeta(x_i) = f(x_i),$$

since $\zeta(x_k) = 0$. Thus,

$$\int r(x)dw(x) = \sum_{i=1}^{k} w_i f(x_i).$$

So now we can write

$$\int f(x)dw(x) = \int q(x)\zeta(x)dw(x) + \sum_{i=1}^{k} w_i f(x_i).$$

Note that this last equation reduces to (15) if the first term on the right is zero. Thus, for f with deg $f \leq 2k-1$ we want to choose our nodes such that $\int q(x)\zeta(x)dw(x) = 0$.

Now suppose that (15) holds for all f of degree less than or equal to $2k-1$. Then, for $f = q\zeta$ this reduces to

$$\int q(x)\zeta(x)dw(x) = 0,$$

since $\zeta(x_i) = 0$ for $i = 1 \ldots k$. Gubner then provides the following theorem:

Theorem 3.1 Let f be any polynomial of degree $\leq 2k-1$. Then,

$$\int f(x)dw(x) = \sum_{i=1}^{k} w_i f(x_i) \tag{17}$$

if and only if

$$\int q(x)\zeta(x)dw(x) = 0 \tag{18}$$

holds for all polynomials q with degree smaller than k.

3.4.3 Choosing the nodes

Gubner proceeds to show that the polynomials constructed by the Gram-Schmidt method

$$\varphi_k(x) := x^k - \sum_{i=1}^{k-1} \frac{\langle x^k, \varphi_i \rangle}{\langle \varphi_i, \varphi_i \rangle} \varphi_i(x) \qquad \text{where } \varphi_0 = 1,$$

can be used to choose suitable nodes. Here, the inner product for polynomials p and q is defined as

$$\langle p, q \rangle := \int p(x)q(x)dw(x).$$

He shows φ_n is the ζ we need for (18) to hold. These polynomials have n distinct roots and φ_i and φ_j are orthogonal for $i \neq j$. Also they satisfy a three-term recurrence relation. Gubner states the following theorem and proves it in his paper:

Theorem 3.2 Suppose that $\varphi_0, \varphi_1, \ldots$ are orthogonal polynomials with deg $\varphi_k = k$ and leading coefficient one. For $k \geq 1$ we have the three-term recurrence

$$\varphi_{k+1}(x) = (x - a_k)\varphi_k(x) - b_k \varphi_{k-1}(x), \tag{19}$$

where

$$a_n := \frac{\langle x\varphi_k, \varphi_k \rangle}{\langle \varphi_k, \varphi_k \rangle}$$

and

$$b_k := \frac{\langle \varphi_k, x\varphi_{k-1} \rangle}{\langle \varphi_{k-1}, \varphi_{k-1} \rangle} > 0.$$

Now to obtain our weights and nodes, we need to determine the roots x_i of φ_n for our nodes t_i, and then proceed to calculate weights w_i. However, there is one other way, where the nodes and weights are calculated using a certain tridigiagonal matrix.

3.4.4 Three facts

Golub and Van Loan provide us three useful facts about orthogonal polynomials and Gauss Quadrature, to determine the nodes. These are quoted below.

> *Fact 1.* Given $[a, b]$ and $w(\lambda)$, there is a sequence of polynomials $p_0(\lambda), p1(\lambda), \ldots$ that satisfy
>
> $$\int_a^b p_i(\lambda) \cdot p_j(\lambda) \cdot dw(\lambda) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$
>
> with the property that the degree of $p_k(\cdot)$ is k for $k \geq 0$. The polynomials are unique up to a factor of +- 1 and they satisfy a three-term recurrence
>
> $$\gamma_k p_k(\lambda) = (\lambda - w_k) p_{k-1}(\lambda) - \gamma_{k-1} p_{k-2}(\lambda)$$
>
> where $p_{-1}(\lambda) \equiv 0$ and $p_0(\lambda) \equiv 1$.
>
> *Fact 2.* The zeros of $p_k(\lambda)$ are the eigenvalues of the tridiagonal matrix
>
> $$T_k = \begin{bmatrix} \omega_1 & \gamma_1 & 0 & \ldots & 0 \\ \gamma_1 & \omega_2 & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \omega_{k-1} & \gamma_{k-1} \\ 0 & \ldots & 0 & \gamma_{k-1} & \omega_k \end{bmatrix}.$$
>
> Since the γ_i are nonzero, it follows from Theorem 8.4.1 that the eigenvalues are distinct.
>
> *Fact 3.* If
>
> $$S^T T_k S = diag(\theta_1, \ldots, \theta_k)$$
>
> is a Schur decomposition of T_k, then the nodes and weights for the Gauss rule are given by $t_i = \theta_i$, and $w_i = s_{1i}^2$ for $i = 1 : k$. In other words,
>
> $$I_G(f) = \sum_{i=1}^{k} s_{1i}^2 f(\theta_i).$$
>
> [Golub & Van Loan, *Matrix Computations*, p. 559]

3.4.5 Lanczos Algorithm

From [7] we know that we can choose $p_i = \varphi_i$, $\omega_i = a_i$, $\gamma_i = \sqrt{b_i}$ for $i = 1, \ldots, k$, as in Theorem 3.2.

Now the only thing holding us back from the production of bounds is the construction of T_k. We can use the basic Lanczos Tridiagonalization algorithm for this, which can be shown to satisfy the same three-term recurrence as our polynomials. For the interested reader, this can be explored more deeply in [6] and [12]. The Lanczos algorithm is given below.

> Given a symmetric matrix $A \in \mathbb{R}^{N\times N}$ and a unit 2-norm vector $q_1 \in \mathbb{R}^N$, the following algorithm computes a matrix $Q_k = [q_1] \dots [q_k]$ with orthonormal columns and a tridiagonal matrix $T_k \in \mathbb{R}^{k\times k}$ such that $AQ_k = Q_k T_k$. The diagonal and superdiagonal entries of T_k are $\alpha_1, \dots, \alpha_k$ and $\beta_1, \dots, \beta_{k-1}$ respectively. The integer k satisfies $1 \le k \le N$.

```
$k = 0, \beta_0 = 1, q_0 = 0, r_0 = q_1$

**while** $k = 0$ or $\beta_k \neq 0$

    $q_{k+1} = r_k/\beta_k$

    $k = k + 1$

    $\alpha_k = q_k^T A q_k$

    $r_k = (A - \alpha_k I)q_k - \beta_{k-1} q_{k-1}$

    $\beta_k = ||r_k||_2$

**end**
```

> [Golub & Van Loan, *Matrix Computations*, p. 549]

The authors then proceed to show that if we choose $q_1 = u/||u||_2$ as our starting vector, we can calculate $I_G(f)$ with the tridiagonal matrix T_k that is generated. This is then done as follows:

1. The Lanczos algorithm with starting vector $q_1 = u/||u||_2$ is used to generate tridiagonal matrix T_k.
2. Schur decomposition $S^T T_k S = diag(\theta_1, \dots, \theta_k)$ is computed.
3. $I_G(f)$ is computed as $s_{11}^2 f(\theta_1) + \dots + s_{1k}^2 f(\theta_k)$.

For upper bound $I_{GR(a)}$ we need to adjust T_k in such a way that

$$I_{GR(a)} = s_{11}^2 f(\theta_1) + \dots + s_{1k}^2 f(\theta_k) + s_{1(k+1)}^2 f(\theta_{k+1}),$$

where

$$f(\theta_{k+1}) = f(a). \tag{20}$$

Thus, we extend T_k with one row and one column, establishing $(k+1)\times(k+1)$ tridiagonal matrix $\tilde{T}_{k+1}$ by including β_k (which we have already calculated) on the superdiagonal, and some $\tilde{\alpha}_{k+1}$ on the diagonal. Now, $\tilde{\alpha}_{k+1}$ must be chosen in such a way that a is in the spectrum of $\tilde{T}_{k+1}$, such that equation (14) holds. This is the case [6] for $\tilde{\alpha}_{k+1} = a + \beta_{k+1}^2 e_k^T (T_k - aI_k)^{-1} e_k$, so $I_{GR(a)}$ is calculated as follows:

1. The Lanczos algorithm with starting vector $q_1 = u/||u||_2$ is used to generate tridiagonal matrix T_k.

2. T_k is extended with one row and column by including β_k and $\tilde{\alpha}_{k+1}$, establishing $\tilde{T}_{k+1}$.

3. Schur decomposition $S^T \tilde{T}_{k+1} S = diag(\theta_1, \ldots, \theta_{k+1})$ is computed.

4. $I_{GR(a)}$ is computed as $s_{11}^2 f(\theta_1) + \cdots + s_{1(k+1)}^2 f(\theta_{k+1})$.

Note that eigenvalue a might not be known. In this case we can estimate it by choosing $a = -\max_i \deg_i$ [4].

Now we know everything to estimate centrality indices for large adjacency matrices, and we can run some experiments in MATLAB.

3.5 Implementation in MATLAB

We have made use of CONTEST: A Controllable Test Matrix Toolbox [8] to generate test matrices. Throughout all experiments we used two test matrices from the CONTEST Toolbox, generated by commands erdrey and pref. These are sparse, symmetric, non-weighted, non-directed and with a zero diagnal. Also, the estimation $a = -\max_i \deg_i$ is used throughout all experiments.

Now we need to decide when to stop the Lanczos algorithm. Logically, this should be when the errors between the bounds and the real value is sufficiently small. But note that we designed this method for a real-world case, in which the real values might not even be known. It turns out that there is a relation between the difference between the bounds and the real value, and the upper and lower bound themselves. This is illustrated below, where we calculated relative bound differences and errors for the estimation of $\exp(A)_{11}$ with $N = 250$.

Figure 2

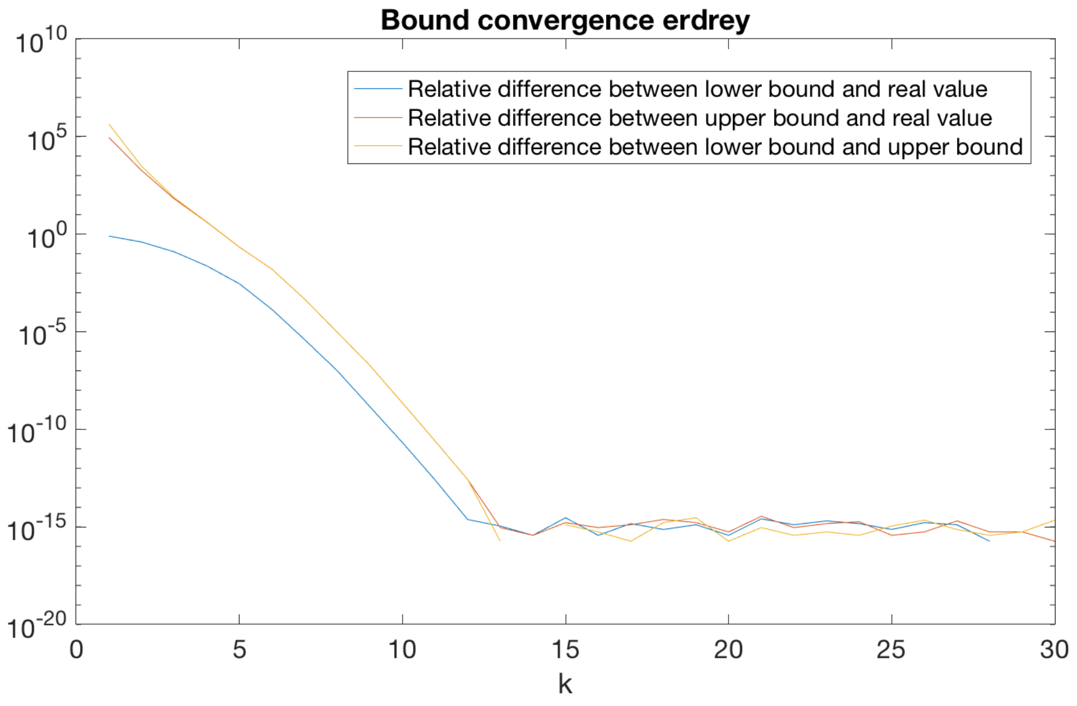
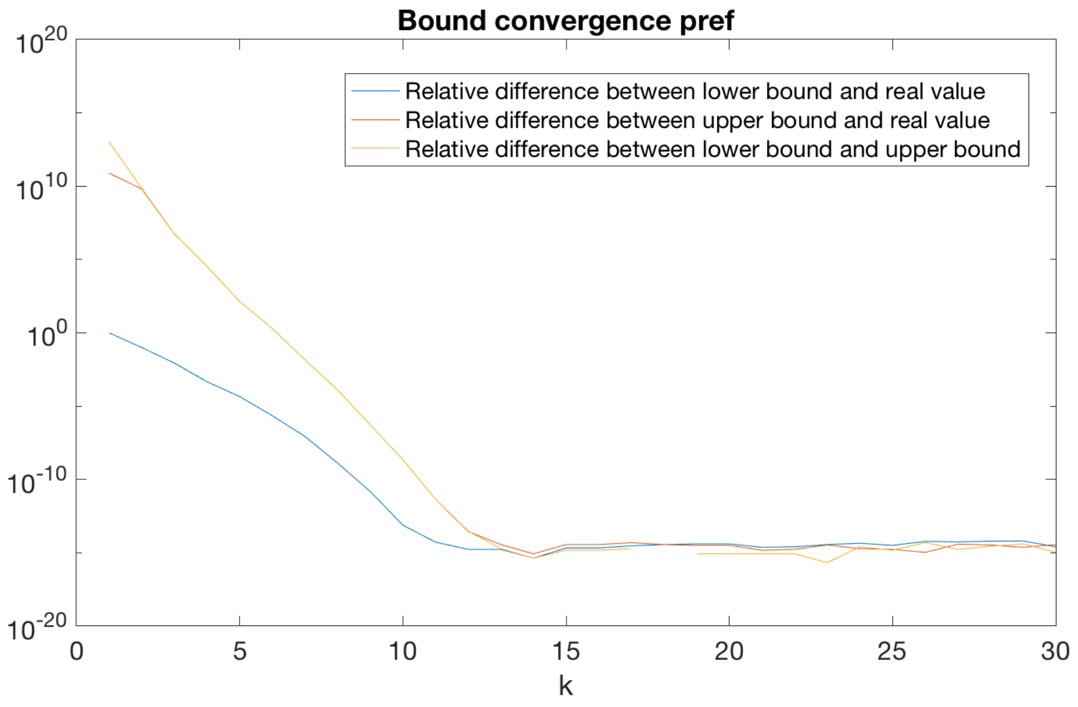


Figure 3



As we can see in the figures above, the errors seem to be approximately the same size for $k \geq 14$ as the difference between the upper and lower bound. Thus, if this last difference is sufficiently small, the errors are also. So we can redesign the basic Lanczos algorithm to stop when the upper and lower bound are sufficiently close to each other. What is sufficient, depends on the application we are dealing with, wherefore we will consider the relative error (or: relative difference between bounds). Thus we designed the algorithm to stop when the relative bound difference was smaller than $\epsilon = 10^{-10}$, with a maximum of $k = 30$ iterations. The MATLAB code we used is included at the end of this paper.

4 Numerical experiments

The primary reason for using an estimation method was finding a faster alternative for the expm command. So we will devise our experiments in order to answer the question: given accuracy $\epsilon = 10^{-10}$, how many iterations are needed to reach this accuracy and how long does this take to compute this in comparison to expm? Now, we will provide a few illustrations in computing times with our test matrices for different N and some incidental node.

4.1 Centrality computing times compared

To construct the following figures, we calculated $\exp(A)_{11}$ using expm and Gauss Quadrature via Lanczos for $N = 100, 250, 500, 1000, 1750, 2500, 3250, 4000, 5000$.

Figure 4

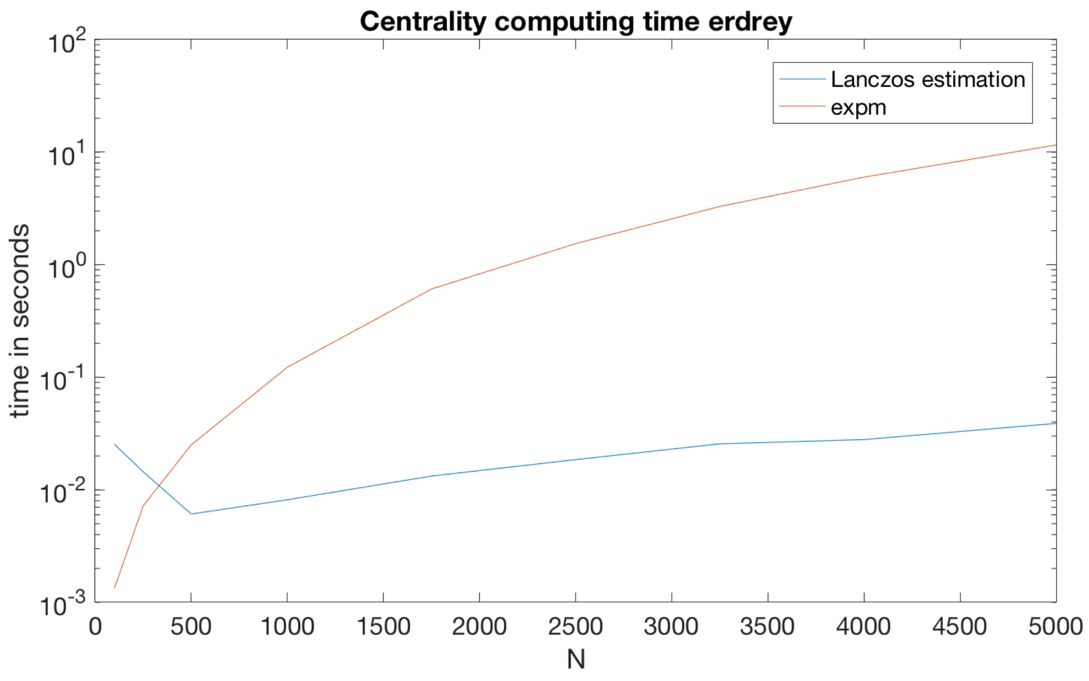
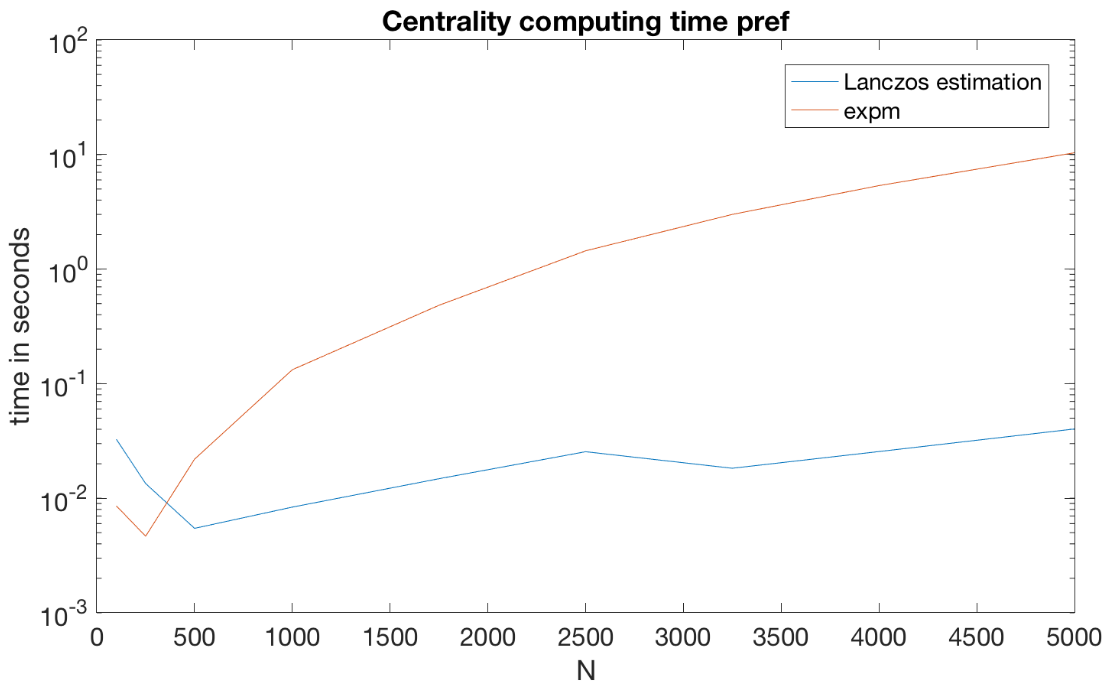


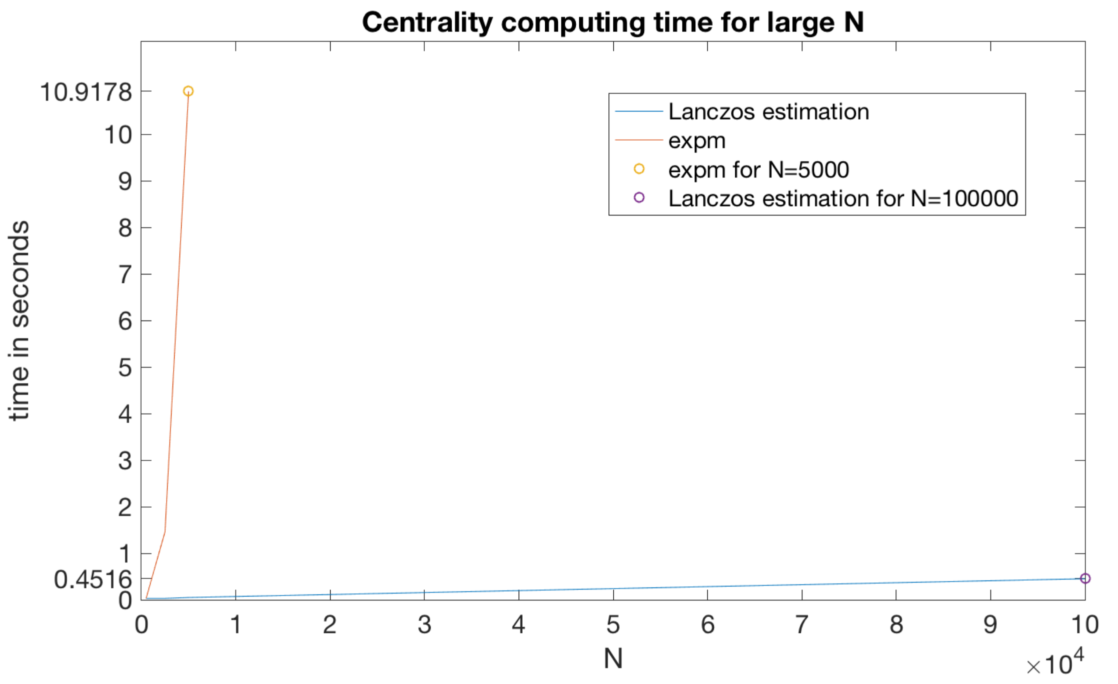
Figure 5



From both figures we can clearly see that while expm computing time rises exponentially, the Lanczos estimation remains constant around a computing time between 10^{-1} and 10^{-2} seconds. In all cases, we needed between 9 and 20 iterations to reach the desired accuracy.

To illustrate how our method compares to direct computation for larger N, we repeated the experiment, this time evaluating at $N = 500, 2500, 5000, 100000$. The results are provided on the next page.

Figure 6



In this figure it can be seen that our estimation method is clearly cheaper for large N. We needed between 13 and 17 iterations to reach the desired accuracy, and this took a maximum of 0.4516 seconds at $N = 100000$ to compute. For directly computing with expm, we stopped at $N = 5000$ because the computing time was already at 10.9178 seconds and rising.

Given its accuracy and computational speed up to large N, we conclude that our estimation method is viable in applications.

4.2 Communicability computing times compared

In this section we produced bounds for $\exp(A)_{12}$ by executing the Lanczos algorithm twice with starting vectors $(\mathbf{e}_1 + \mathbf{e}_2)$ and $(\mathbf{e}_1 - \mathbf{e}_2)$. We noted that the accuracy $\epsilon = 10^{-10}$ could not be met with 30 iterations, or even with 200, so we chose $\epsilon_* = 10^{-6}$ for communicability. The rest of the experiment remains the same.

The figures on the next page provide our results:

Figure 7

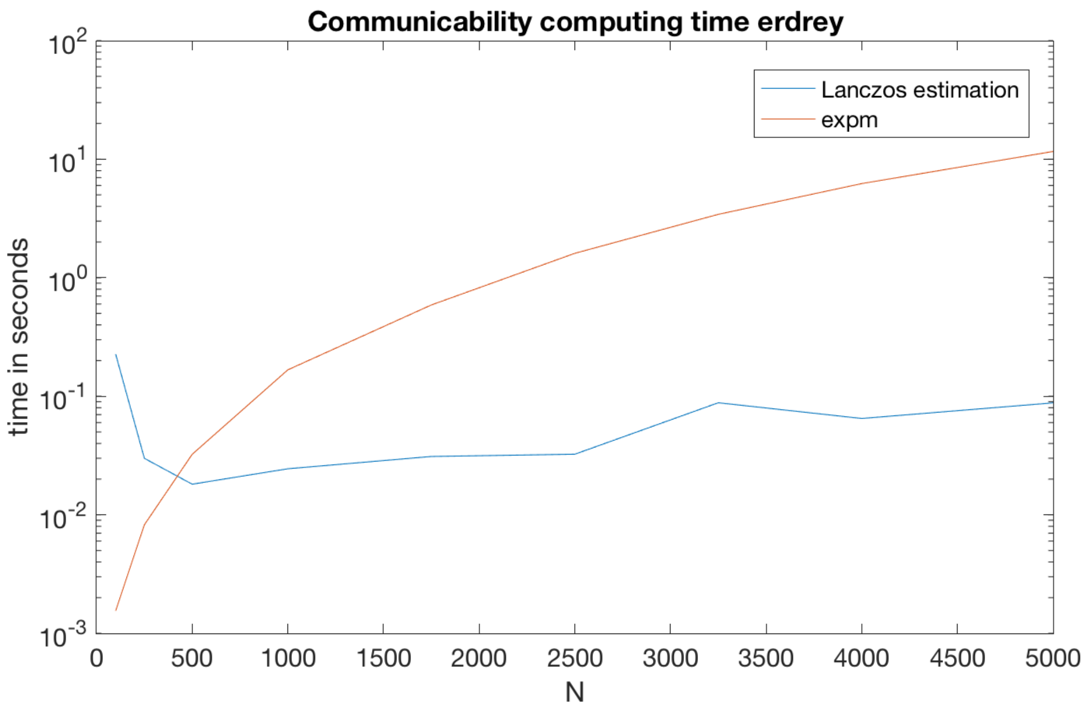
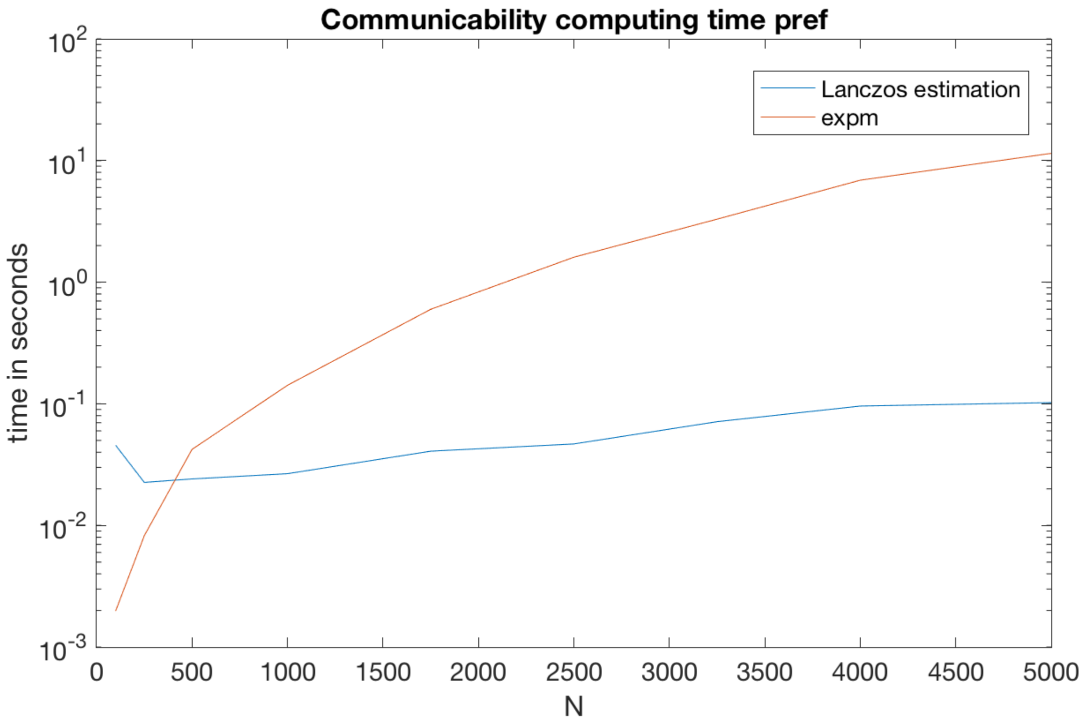


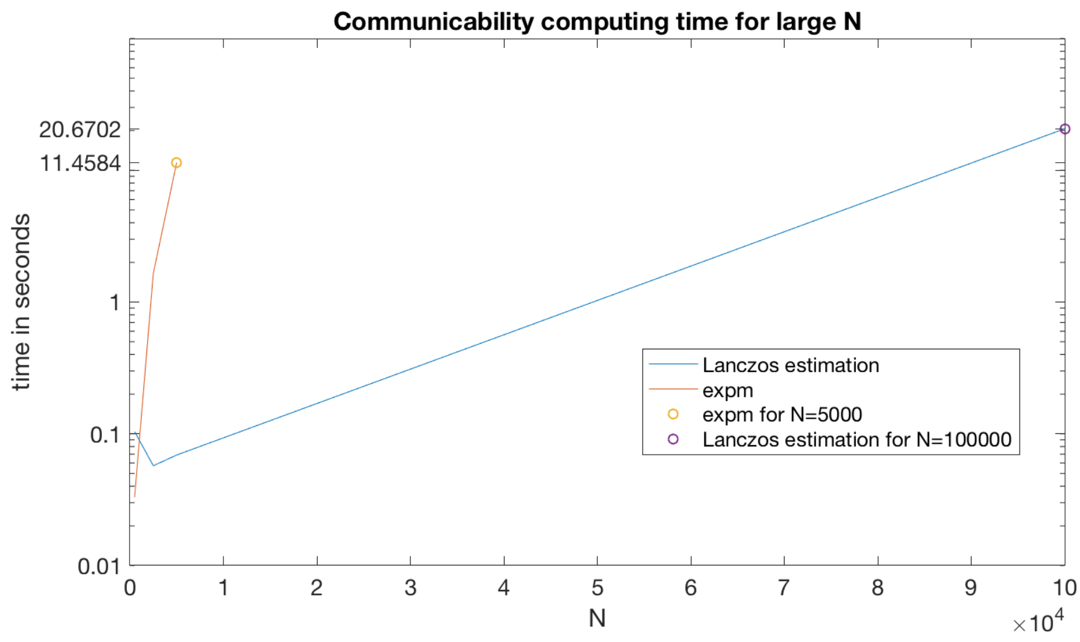
Figure 8



From both figures we can clearly see that while expm computing time rises exponentially,the Lanczos estimation remains constant around a computing time between 10^{-1} and 10^{-2} seconds. In all cases, we needed between 12 and 65 iterations to reach the desired accuracy.

To illustrate how our method compares to direct computation for larger N, we repeated the experiment, this time evaluating at $N = 500, 2500, 5000, 100000$.

Figure 9



In this figure it can be seen that our estimation method is clearly cheaper for large N. We needed between 12 and 35 iterations to reach the desired accuracy, and this took a maximum of 20.6702 seconds at $N = 100000$ to compute. For directly computing with expm, we stopped at $N = 5000$ because the computing time was already at 11.4584 seconds and rising.

Given its accuracy and computational speed up to large N, we conclude that our estimation method may be viable in applications. The computing time is longer than when computing centrality, but still much faster than direct computation.

4.3 Betweenness computing times compared

In this experiment it turned out to be impossible to calculate any estimate of betweenness with the desired accuracy in under 60 seconds. For every N, estimating betweenness takes much longer than calculating it directly, and the computation time rises exponentially. In order to explain this slowness, recall from definition 2.8 that to calculate betweenness, we need all elements of two different matrices to be known. This is in stark contrast with centrality and communicability, where we just estimate one matrix element. Thus, the Lanczos algorithm has to be executed N times for the diagonal elements, and $2N$ times for the off-diagonal elements, for two different matrices, instead of one or two times for one matrix. Evidently, this gets expensive very quickly.

4.4 Preliminary conclusions

We can conclude from the tests above that Gauss Quadrature via Lanczos is an excellent way to estimate centrality and communicability indices for large N, in an accurate and fast way. However, this is not the case for betweenness indices, because we have to execute the Lanczos algorithm repeatedly, such that the time gains in estimating a single matrix element are nullified.

5 Case study

We will now look at an application in the field of Transport Geography, concentrating on the degree and centrality of ports, and connect our theoretical approach to the article *Maritime degree, centrality and vulnerability: port hierarchies and emerging areas in containerized transport (2008–2010)* by Laxe, Seoane and Montes [5].

5.1 Comparison

The method the authors use differs from our method in three important ways:

1. They do (probably) not use an estimation method but directly compute matrix operations, due to the fact that their adjacency matrix is not very large.
2. Connections between nodes are not (always) two-way, thus their adjacency matrix is not symmetric.
3. Centrality is defined as
$$\sum_{i \neq j \neq r \neq r} \frac{\sigma_{ij}(r)}{\sigma_{ij}} \tag{21}$$
where $\sigma_{ij}(v)$ is the number of shortest walks from i to j through r and σ_{ij} is the total number of shortest walks from i to j.

5.2 Overcoming differences

As for the first difference, if a matrix is small enough to compute it directly, it is probably faster to do any computation directly than to undergo the time-consuming practice of studying and devising some estimation method that is cheaper. Therefore it is more interesting to consider the question: what if their adjacency matrix *was* large? Then we would have to use an estimation method, which leads to the other differences.

Note that the centrality measure of the authors looks similar to our definition of betweenness. $\sigma_{ij}(r)$ in (15) can also be written as $\sigma_{ij} - \sigma_{ij}(\bar{\mathrm{r}})$ where $\sigma_{ij}(\bar{\mathrm{r}})$ is equal to the number of shortest walks from i to j that do not pass through r. Suppose that the shortest walk from some i to j is equal to n_{ij}. Then, we can write (15) as:

$$\sum_{i \neq j, i \neq r, j \neq r}\sum \frac{A_{ij}^{n_{ij}} - (A - E(r))_{ij}^{n_{ij}}}{A_{ij}^{n_{ij}}} \tag{22}$$

where A is a non-symmetric adjacency matrix and $E(r)$ is the matrix defined in definition 2.8. This can be seen as a non-normalized version of betweenness with $f(A)_{ij} = A^{n_{ij}}$ instead of $f(A)_{ij} = \exp(A)_{ij}$.

In order to calculate appropriate bounds for the above value we can let $f(A) = A^n$ for some n in equation (5). Since A^n is not s.c.m. we can write $A^{-(-n)}$. The next step is to determine n_{ij} and calculate $A_{ij}^{n_{ij}}$ for all $i \neq j \neq r$. We are able to determine n_{ij} with the Dijkstra algorithm, but we cannot use the Lanczos Tridiagonalization Algorithm to calculate Gauss nodes and weights for bounds on $A^{n_{ij}}$ if A is not symmetric [6]. Lanczos Bidiagonalization and the Arnoldi iteration are options to be explored here [11] [3]. Also, since the computing time of betweenness with the Lanczos Tridiagonal Algorithm far exceeds the expm computing time, it is important to make sure this estimation method is considerably cheaper.

5.3 Preliminary conclusions

We have seen that when A is not symmetric, there are alternatives for the Lanczos Tridiagonalization Algorithm for calculating Gauss nodes and weights. Also, Dijkstra's algorithm can be used for determining the shortest path of length n_{ij} between two nodes $i \neq j$. Furter, we can choose $f(A) = A^{-(-n)}$ and adjust our betweenness definition to calculate the centrality measure that is used in [5].

6 Conclusion & Discussion

At this point our research is finished and we can conclude the following. We have seen that it is possible to represent graphs as adjacacency matrices, and under certain conditions, determine the importance of different nodes using various centrality measures. When a matrix is large, a faster option than directly calculating a centrality measure can be to use an estimation. In this paper we showed that Gauss Quadrature via the Lanczos Tridiagonalization algorithm is an estimation method that can be effectively used for these means. With some numerical experiments we showed that for estimating centrality and communicability this method was an accurate and fast one and can be easily applied in real-world cases. However, in estimating betweenness this method is slower than direct calculation and at best just as accurate, making it inefficient. The reason the betweenness estimate calculates slowly is because all exponential matrix elements need to be estimated, causing the Lanczos Algorithm to execute many times, which slows computation times. So as a rule this method is mainly useful for estimating single matrix elements. This means that if we want to efficiently use it, we already need to have an idea which nodes will be important. Thus, we need to use some method of prior analysis to target specific nodes. This may be an interesting subject for future research. Also, it may be useful to investigate different estimation methods to calculate all exponential matrix elements rapidly and accurately.

The case study showed a different approach to analyzing adjacency matrices and centrality measures, and that each application demands its own definitions. In the end, this thesis shows the many possibilities in analyzing a network. This branch of mathematics can be of real value in applied situations, and the end of this exploration is not yet in sight.

References

[1] URL: `https://whatis.techtarget.com/definition/extrapolation-and-interpolation`.

[2] URL: `https://archive.org/stream/IoNewsVolume1Number5#mode/2up`.

[3] David Bau. *Lecture 39: Biorthogonalization Methods*. 1997. URL: `https://www.globalspec.com/reference/72093/203279/lecture-39-biorthogonalization-methods` (visited on 12/07/2018).

[4] Michele Benzi & Paola Boito. "Quadrature rule-based bounds for functions of adjacency matrices". In: *Linear Algebra and its Applications* 433.3 (2010), pp. 637–652.

[5] Maria Jesus Freire Seoane Fernando González Laxe and Carlos Pais Montes. "Maritime degree, centrality and vulnerability: port hierarchies and emerging areas in containerized transport (2008–2010)". In: *Journal of Transport Geography* 24 (2012), pp. 33–44.

[6] Gene H. Golub and Charles F. van Loan. *Matrix Computations*. Johns Hopkins Series in the Mathematical Sciences 3. The Johns Hopkins University Press, 1989.

[7] John A. Gubner. *Gaussian Quadrature and the Eigenvalue Problem*. 2014. URL: `http://gubner.ece.wisc.edu/gaussquad.pdf` (visited on 12/11/2018).

[8] Desmond Higham. *CONTEST: A Controllable Test Matrix Toolbox for MATLAB*. `http://outreach.mathstat.strath.ac.uk/outreach/contest`.

[9] Ernesto Estrada & Desmond J. Higham. "Network Properties, revealed through Matrix Functions". In: *SIAM Review* 52.4 (2010), pp. 696–714.

[10] Ping Zhang Jonathan L. Gross Jay Yellen. *Handbook of Graph Theory*. 2. CRC Press, 2004.

[11] James V. Lambers. *A Crash Course on Matrices, Moments and Quadrature*. 2010. URL: `https://pdfs.semanticscholar.org/presentation/5a64/19754dd41cd3af3fe28b8aa02b94`
`pdf` (visited on 07/12/2018).

[12] Gene H. Golub & Gérard Meulant. *Matrices, Moments and Quadrature with Applications*. Princeton University Press, 2009.

Source UU Logo on the title page: http://www.uu.nl/organisatie/huisstijl/downloads/logo.