Numerical Analysis
Bachelor's Thesis

Approximating extrema of quadratic forms using Krylov subspaces

Simon Hallborn
Advisor: Gustaf Söderlind
27/09/2017

Contents

1 Abstract

In this thesis approximations to quadratic forms using Krylov subspaces are presented. These quantities are approximations of the norm and logarithmic norm of a matrix. In order to find these quantities an eigenvalue problem has to be solved, but because of limited storage and time, this is not feasible in practice with large matrices. Instead one can project this problem down to a problem of smaller size with a Krylov subspace and solve it there instead.

The original idea was to investigate, for a given matrix, if one could get sufficiently good estimates of the norms with only one subspace. This turned out to not be the case.

The span of the subspaces that was generated is good in approximating some vectors, but for others it is a poor choice. This is dependent on which norm we are trying to approximate but also on the matrix type. Hence we need different methods depending on what the specific conditions are for the problem.

The goal is to reach at least two digits of accuracy because an algorithm like the one that was constructed for this thesis could, if further developed, be used in conjunction with already existing methods. This is an approximation based method, that favours simplicity and speed. It is well suited for other methods that require the estimated quantities in multiple time steps or iterations.

To test the quality of these projections, different matrices that arise in practice are tested and their norms approximated. The test matrices have known norms and behaviour so the result can be interpreted.

Overall the results show that one can obtain two digit accuracy with a low dimension of the subspace, even for matrices with large dimensions, which is truly promising.

2 Background

In many applications in scientific computing, it is of interest to compute or at least estimate the norm and/or the logarithmic norm of a matrix A. This is needed for a wide variety of tasks, such as estimating convergence rates, stability constants, ellipticity etc. While today's software can easily compute these data for matrices of moderate size (say $n \leq 100$) the task becomes more or less prohibitive for large matrices. In addition, while these parameters can be estimated for linear operators, it often becomes a complicated task for the nonlinear case. Here we limit ourselves to the problem of computing these quantities for large matrices, with the aim of reducing the computational complexity using Krylov subspaces. Thus we intend to develop algorithms that are efficient also for large matrices, while making sure that we still obtain accurate results.

The reduction of the computational complexity using Krylov subspaces will be applied to solving both large linear systems of equations and large eigenvalue problems. The methods that would be used for these large problems vary depending on the type of the matrix, so a case by case basis is required to find the most efficient solution.

Specifically, this project will be based around approximating four quantities

$$L[A]^2 = \max_{x \neq 0} \frac{x^\top A^\top A x}{x^\top x} \tag{1}$$

$$M[A] = \max_{x \neq 0} \frac{x^\top A x}{x^\top x} \tag{2}$$

$$l[A]^2 = \min_{x \neq 0} \frac{x^\top A^\top A x}{x^\top x} \tag{3}$$

$$m[A] = \min_{x \neq 0} \frac{x^\top A x}{x^\top x}, \tag{4}$$

these are the extrema of the quadratic forms which we are interested in.

In order to estimate these quantities we begin with forming the Rayleigh Quotient

$$\underset{x \neq 0}{R(C,x)} = \frac{x^\top C x}{x^\top x},$$

and in our case $x \in \mathbb{R}^{n \times 1}$ is a nonzero vector and $C \in \mathbb{R}^{n \times n}$

As Equations 1 - 4 show we are looking at the stationary points of $R(C,x)$ for the cases when $C = A$ and $C = A^\top A$, to calculate the logarithmic norm and/or the norm of A respectively. In order to find the stationary points of of $R(C,x)$ we differentiate with respect to x and set the derivative to zero,

$$\frac{(x^\top C + x^\top C^\top) x^\top x - 2x^\top (x^\top C x)}{(x^\top x)^2} = 0.$$

Rearranging and transposing this we get the eigenvalue problem:

$$\frac{(C + C^\top)}{2} x = R(C, x)x.$$

Now let $C = A^\top A$ or $C = A$ and from this general problem we obtain two specific problems,

$$A^\top Ax = \frac{x^\top A^\top Ax}{x^\top x} x \tag{5}$$

$$\frac{(A^\top + A)}{2} x = \frac{x^\top Ax}{x^\top x} x. \tag{6}$$

This is done because Equation 5 gives us the solution to Equations 1 and 3 while Equation 6 gives us the solution to Equations 2 and 4.

Both of Equations 5 and 6 are two symmetric eigenvalue problems of the form

$$A^\top Ax = \sigma^2 x \tag{7}$$

$$\frac{(A^\top + A)}{2} x = \mu x, \tag{8}$$

where σ is the largest singular value of A, $\sigma^2 = ||A||_2^2 = L[A]^2$ and $\mu = M[A]$

When the matrix A is large and sparse a good idea for methods to solve Equations 5 and 6 are iterative ones. This is due to direct methods often requiring more work and because of iterative methods' ability to exploit the sparsity of the matrix.

In the realm of iterative methods a commonly used idea is to project an n-dimensional problem into a lower-dimensional Krylov subspace, with further reading in [2]. One way to solve large sparse symmetric eigenvalue problems as in Equations 5 and 6 is to use Lanczos iteration. But this thesis will be based on using a different method. This alternative method uses the same principle as Lanczos in the sense of creating a subspace and creating an orthogonal base from the subspace. After that point they diverge. The point of this thesis is not to compare the two, it is to test the result of the alternative method and see how effective it is.

There are a few reasons why this alternative method is promising.

The first is that it has a simple theory and implementation. Once the problem is projected down onto the subspace, obtaining the solution is simple. This is because the size of the problem is now small enough that direct methods can easily be applied.

The second reason is that it is a cheap and approximation based method, which is good for it to be implemented into other main methods. The quantities that are estimated here might be used frequently based on what kind of main method that uses them. The method remains cheap while the subspace's dimension is low, which will not be the case for every matrix. This because we might never reach two digits of accuracy, which is the minimum acceptable tolerance.

The third reason is that for some special matrices the solution is very accurate, even for low dimensional subspaces.

And the final reason is because one does not have to explicitly form A which takes unnecessary space on the memory. This is because all the operations with A can be obtained with black box methods as described in the next section.

2.1 Black box methods

The theory behind a black box is a procedure whose structure is hidden to the user but transmits a desired output [1]. An example could be how a black box method would be in our case for a given vector x. This method would return a linear mapping, $y = Ax$ of x. One of the strengths of this method is that A is never formed explicitly which saves space on the memory.

Exactly how this method is developed is not the primary focus, rather it is idea the behind it that is interesting. This is because black box methods that are used for applications today are complex because they vary from problem to problem. For example if there is a sparse matrix with its non-zero elements spread around in the matrix, then one can rearrange the rows so that all the non-zero elements are within a band. Inside this band there are both non-zero and zero elements, and on the outside it is just zeros. Then it is more compact and easier to only focus on the multiplications that affect the result.

For a dense matrix, with its only entries being non-zero elements, performing a matrix-vector multiplication costs $\mathcal{O}(n^2)$ operations. When n grows then this multiplication does become expensive.

However, for large and sparse matrices there is a certain structure that is desirable to exploit. If for example we have a sparse matrix with five non-zero elements per row and we perform a matrix-vector multiplication then it can be done in $\mathcal{O}(5n)$ calculations. This means that because of the sparsity we avoid a lot of unnecessary calculations.

So the main reasons that we want to use this procedure is because of the decreased amount of calculations that is necessary for the multiplication as well as the fact that A is never stored.

3 Theory

The idea now is to project down the large n-dimensional problem to a smaller one. The original idea was that we form one subspace only using power iteration as described in the next section. The assumption was that this subspace would, for a given matrix, be able to estimate its norms accurately.

This assumption turned out to be incorrect due to the fact that the span of the subspace that was generated failed to find certain vectors for certain cases. It failed to uphold the generality of the assumption, so instead we generate a subspace through either power or shifted inverse iteration (also described in the upcoming sections) depending on the situation. From either of these subspaces we approximate the solution to Equations 5 and 6 by using a subspace based on the structure of the matrix and what norm that is estimated. This case by case basis will grant us a better estimate than what only one of the methods could do on their own.

3.1 Power iteration

The first kind of Krylov matrix that is used, K_p, is defined below and it is obtained using power iteration [3] It is constructed by multiplying the matrix A with an initial vector b. These vectors will be the column vectors of K_p,

$$K_p = [b, Ab, A^2b, \dots, A^{N-1}b]. \tag{9}$$

If the next vector in the iteration equals the previous one then the iteration is stopped. For example if $A^{M-1}b = A^M b$ then the dimension of K_p will be $n \times M$, if this is the first time this has occurred in the iteration.

The operation cost depends on whether we use black box methods or direct computation, where A is formed explicitly. For the matrices in our experiments it suffices to use direct computation because the matrices are neither large nor sparse.

Two shortcomings with this iteration is that the algorithm only produces the largest eigenvalue of A in absolute value, and that the largest eigenvalue of A has to be significantly larger than the second largest eigenvalue.

3.2 Shifted inverse iteration

The second Krylov matrix will be created by using shifted inverse iteration. In a similar fashion as Equation 9 with the first Krylov subspace we now use $(A - sI)^{-1}$ instead of A, and s is a shift that helps us locate a certain eigenvalue. In order to not form the inverse we will transform the problem

$$(A - sI)^{-1}b = x,$$

to an easier one. This is done by multiplying both sides with $(A - sI)$ and we end up with the problem,

$$(A - sI)x = b.$$

This is a linear equation that when solved yields the desired vector x. With the matrices in our experiments it suffices to use the BACKSLASH OPERATOR in MATLAB that will in the general case solve this linear equation with an LU factorization. When we have larger and more sparse matrices this method would not be an optimal choice since the LU factorization ruins sparsity. This because the lower matrix L and the upper matrix U will be dense. Instead one needs to use iterative methods, but the method one would use depends on the structure of A and on the fact that A should preferably not be formed explicitly.

Which iterative method that should be used is entirely up to the conditions of the problem. If the matrix is symmetric and positive definite then there are methods like CG (Conjugate gradient) otherwise methods that work for more general matrices like GMRES (Generalized minimal residual) or BiCGSTAB (Biconjugate gradient stabilized) could be used [1].

Below is the definition of Krylov matrix, K_i, that is obtained by the inverse iteration,

$$K_i = [b, (A - sI)^{-1}b, ((A - sI)^{-1})^2 b, \ldots, ((A - sI)^{-1})^{N-1}b].$$

The dimension of K_i works the same as for K_p, because this is just a different form of power iteration. If the next vector in the iteration equals the previous one then the iteration stops. For example if $((A - sI)^{-1})^{M-1}b = ((A - sI)^{-1})^{M}b$ then the dimension of K_i will be $n \times M$, if this is the first time this has occurred in the iteration.

A shortcoming with using inverse iteration is that cases having two eigenvalues with almost the same magnitude causes difficulties with the spanning of the subspace. This is because we want the vectors to give as much information as possible and having two eigenvalues close to each other affects the how the vectors in this subspace are pointing.

The advantages of using inverse iteration are that instead of only finding the largest eigenvalue in absolute value, as is done with power iteration, one can find any eigenvalue. Also the rate of convergence can be controlled. This is possible if one can find a shift, s close enough to the eigenvalue of interest.

To find a suitable shift one can use the Gershgorin Circle Theorem that states that any eigenvalue of A can be found in a constructed circle centered at a_{ii} with radius R_i, where a_{ii} is the ith element on the diagonal of A and $R_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|$.

We can thus obtain a disc where the smallest and largest eigenvalues are contained. This narrows down the possible values of our shift. The disc that is created with the circle theorem can be relatively large depending on the matrix though. There might be ways to zoom in further to reduce the size of the disc of possible eigenvalues. However in our results as we will later see it suffices to set the shift to zero for our experiments. This is again because this iteration is only used in the experiments when we want to find the smallest eigenvalue in absolute value, which means some number relatively close to zero in our case. So without any methods of finding a better shift, zero is the best one for the experiments.

So what happens if $(A - sI)$ is singular?

The eigenvalues of $(A - sI)$ are $\lambda - s$, where λ are the eigenvalues of A. It is singular if the eigenvalue of interest and the shift are the same. The relevance of this is, as we will see later, when we try to approximate the eigenvalue equal to zero. An example of this could be the case when we have a singular positive semi definite matrix and we want to find the smallest eigenvalue in absolute value. To find a shift close to this eigenvalue of interest is to set the shift to zero which means $(A - sI)$ is singular. In these specific cases it is not really an issue because even though the algorithm will not return anything but a warning message, we know what the exact value is.

3.3 Orthogonalization

To combat the near linear dependence that we are creating with power iteration and inverse iteration, one needs to make sure that the vector space whose vectors are almost pointing in the same direction becomes orthogonal. This orthogonalization will be done with the QR factorization, from which we obtain an orthogonal base, Q.

As an example from using power iteration we obtain Q_p from,

$$K_p = Q_p R.$$

So the Krylov subspace using power iteration would have the following structure

$$K_N = \langle b, Ab, A^2 b, \ldots, A^{N-1} b \rangle = \langle q_1, q_2, q_3, \ldots, q_N \rangle \subseteq \mathbb{R}^n,$$

where $\langle q_1, q_2, q_3, \ldots, q_N \rangle$ are the column vectors of Q_p [1].

Let us now consider a vector $x = Qu$ that is represented as an element of the generated Krylov subspace, as a linear combination of the columns of Q.

3.4 Approximating $l[A]$ and $L[A]$

By setting $x = Qu$ in Equation 7 we get:

$$A^\top A Q u = \sigma^2 Q u.$$

We multiply both sides with $Q^\top$ from the left and we will end up with

$$(AQ)^\top A Q u = \sigma^2 u.$$

By setting $B_l = AQ$ we are left with the eigenvalue problem

$$B_l^\top B_l u = \hat{\sigma}^2 u, \tag{10}$$

where $\hat{\sigma}$ is an approximation to σ.

Even though B_l is a rectangular matrix with dimensions $(n \times N)$, this will not be a problem. This is because we form $B_l^\top B_l$ which makes it a square matrix and we can find the eigenvalues. The dimension will be $(N \times N)$ which is the desired result because $N << n$.

Also, because the matrix $B_l^\top B_l$ that we form is symmetric positive semi definite, we know that

$$0 \leq l[A] \leq L[A].$$

With this inequality in mind we want to use power iteration to find $L[A]$ and shifted inverse iteration to find $l[A]$, this is true for any non-singular matrix that we look at.

3.5 Approximating $m[A]$ and $M[A]$

By setting $x = Qu$ in Equation 8 and multiplying with $Q^\top$ from the left we get,

$$\frac{(A^\top + A)}{2} Qu = \mu Qu$$

$$\frac{Q^\top (A^\top + A) Q}{2} u = \mu u.$$

By setting $B_m = Q^\top A Q$ we get

$$\frac{(B_m^\top + B_m)}{2} u = \hat{\mu} u, \tag{11}$$

where $\hat{\mu}$ is an approximation to μ.

Here B_m is a square matrix instantly after the multiplications with dimension $(N \times N)$ which is again the desired result we are looking for because $N << n$.

One reason why it is important to obtain the values for $m[A]$ and $M[A]$ is because they are known as the lower and upper logarithmic norms of A, see [6] for an introduction to logarithmic norms. The logarithmic norm has many applications in differential equations, and a survey of its most important applications is found in [7].

One particular application is the recent link to stiffness. Thus stiffness can be assessed in terms of the average logarithmic norm

$$m_{avg} = \frac{(m[A] + M[A])}{2},$$

see [8] for details.

The average logarithmic norm requires that both $m[A]$ and $M[A]$ are computed. One of the motivations for this thesis is that it is generally expensive to compute $m[A]$ and $M[A]$ to full accuracy, as they technically require that we solve eigenvalue problems for matrices of the same dimension as A.

However, as stiffness does not call for more than an estimate of the average logarithmic norm, it is of interest to develop cheap techniques that allow $m[A]$ and $M[A]$ to be estimated to an accuracy of around two digits. If this can be achieved at low cost, stiffness can be assessed along the solution of an ODE, without causing the integration to slow down too much.

Now that the matrix is only symmetric instead of symmetric positive semi definite the inequality from the previous section changes to

$$-\infty < m[A] \leq M[A] < \infty.$$

This fact makes it so in order for the subspace to be accurate, a case by case basis of how the eigenvalues are located is necessary to have. This can be seen from the following examples:

If the initial matrix is positive definite then we have the same case as for $l[A]$ and $L[A]$, that being shifted inverse and power iteration respectively.

But if the matrix is negative definite then the roles reverse since $M[A]$ now is whatever eigenvalue is closest to zero and if we use power iteration to span that subspace then it will not be accurate.

The estimations we get from Equations 10 and 11 are the approximations to the exact values we are trying to obtain. These two eigenvalue problems can be solved by direct computation, this because of their reduced size. To get a view of how accurate these approximations truly are we compare the result of the approximate value $\hat{\sigma}$ to the exact value σ by forming the relative error

$$e = \frac{\hat{\sigma} - \sigma}{\sigma}, \tag{12}$$

and the same goes for μ and $\hat{\mu}$.

3.6 The symmetric case

Before any assumptions are made, consider the fact that the eigenvectors of a matrix A and its inverse A^{-1} are the same. To show this begin with an eigenvalue problem

$$Ax = \lambda x,$$

where A is invertible. By multiplying both sides with A^{-1} and $\frac{1}{\lambda}$, this works because we know that A^{-1} exists and that $\lambda \neq 0$, so we obtain

$$A^{-1}x = \frac{1}{\lambda}x.$$

As an example, assume now that the initial matrix is symmetric positive definite. This means that the quantities we are searching for are all positive. None of the eigenvalues can be equal to zero due to A being non-singular. Hence we can use both power and inverse iteration without any trouble.

With A being symmetric, finding the largest/smallest eigenvalues from the Equations 5 and 6 now changes. The left hand sides of both equations now simplify to A (Equation 6 simplifies because of the square root). So the new problem for this case is to approximate the largest/smallest eigenvalue of A which is exactly what our methods do, so the two eigenvalue problems coincide.

Also, the case of when the matrix is skew-symmetric works almost in the same way as the symmetric case except that Equation 6 is always the zero matrix.

So the subspaces we create in order to estimate the desired vectors are very accurate and this is shown in the result.

3.7 The indefinite case

To find estimates for $L[A]$ and $l[A]$ is no issue for the indefinite case. This because the matrix we form for these quantities will be symmetric positive semi definite so the eigenvalues will be real and non-negative. And we know from before we use power and shifted inverse iteration respectively to find the estimates.

But what happens for $M[A]$ and $m[A]$?

First, the matrix that we construct is at least symmetric so the eigenvalues are real. Imagine now that we have a matrix whose smallest eigenvalue is of the magnitude -10^{10} and largest eigenvalue of magnitude 10^5. If we use power iteration then the subspace will be based on the largest eigenvalue in absolute value, i.e $|-10^{10}|$.

The question remains, how does one find a good subspace to estimate the eigenvalue at 10^5. In order to find this eigenvalue we would need to use inverse iteration with a shift that is close to this eigenvalue of interest. If we do not get a shift that is close enough then the span of the subspace will not be accurate. Simply setting the shift to zero as is done for our experiments will not work in this case.

4 Discussion

The results of this thesis are very promising. The matrices that we observed had special properties, but it gives an idea of how it would work for general matrices. We saw what happens for positive/negative definite matrices, matrices with large entries, singular matrices and non-symmetric matrices. These matrices can be seen as test data, and we know the theory behind these matrices, thus making it possible to interpret the result and predict their behaviour. This makes it easier to understand the outcome and find any anomalies that do not fit the pattern.

Setting the tolerance to two correct digits might seem like a poor choice, but the idea behind this method is for it to function as fast and cheap as possible. Speed is preferred over accuracy. This is supposed to be a method that could be used before a complex main method initiates. The measuring of stiffness for differential equations is a good example, because the estimations of $m[A]$ and $M[A]$ determines whether the main method should be continued or stopped.

This method could be used multiple times for different main methods that require the estimations of the quantities in multiple time steps or iterations. And one normally only wants a rough idea of how stiff the differential equation is. If accuracy was the main purpose then it would require a lot of work for each time step or iteration which could possibly add up to be more work finding an estimate to these quantities than it would be to perform the main method itself.

As can be seen from the result, for some matrices we get both speed and accuracy. For very low dimensions of the subspaces, which do not require that much work, we obtain a surprisingly high accuracy. For an efficient and smooth algorithm more work has to be done, as we will talk about in the next section.

When the kind of matrix whose norms we are trying to estimate is unknown, and it happens to be symmetric or skew-symmetric, we know that this method is a good candidate to use. So instead of this method being used for general matrices, perhaps it should only be used on a subset containing symmetric and skew-symmetric matrices. Because it is here that the method shows its potential. The main purpose of this thesis was still to only test this method and see the result, not to compare to others.

The size of the selected problems that we have been working with is to test them on a smaller scale and then increase the dimensions, because if it is not working here then it will surely not work for larger dimensions. The largest dimension that we were working with was $n = 300$. For sparse matrices we could see a much higher dimension being possible. When we are working with $n = 300$ we can use direct methods without any problems because modern computers with multiple cores can compute it with ease. This enables us to verify and assess the algorithm.

Two already existing commands in MATLAB that were used are `eig` and `backslash operator`. The `backslash operator` will definitely not be used for larger problems due to it requiring a lot of work and ruining sparsity. The command `eig` is only used in the testing of this algorithm. This served as a comparison to see whether the output from the algorithm was close to the exact value. If the results were accurate, the algorithm can be initiated for larger problems without knowing the exact eigenvalues of the matrix A.

Something that has not been brought up yet, but still was developed in the algorithm is an adaptive variant of this method. It is rather basic but it shows how this could be used or combined with already existing main methods.

Right now a tolerance can be set and when the error is below this threshold the program terminates and returns when the threshold was achieved and at what accuracy. As an example, here is the output from the adaptive method for approximating $L[T]$ and $M[T]$ when the matrix T (see Results for its definition) has dimensions 300×300.

```
The given tolernace is tol = 1.000000e-04.
L has been approximated at N = 106
With the error -9.308016e-05.
M has been approximated at N = 2
With the error 3.629698e-05.
```

The reason not much time was spent on improving the adaptive method is because there is always room for improvement when dealing with adaptive methods and developing an advanced one was not the primary focus of this study.

4.1 Future work

In order for this method to be applicable to very large problems in the future, there are a few things that need to be researched and integrated for the method to function as efficiently as possible for any given matrix.

First and foremost a more general approach to finding the shift for inverse iteration of matrices without a known exact value of the eigenvalues of A is needed, to make sure that the subspaces that we generate are accurate. Take the example of the indefinite matrix as we saw before, whose smallest eigenvalue is of the magnitude -10^{10} and largest eigenvalue of magnitude 10^5. In order to find the eigenvalue 10^5 shifted inverse iteration must be used and for it to span a good subspace, then we need the shift to be close to the eigenvalue at 10^5. So with a method or a theorem, maybe in conjunction with Gershgorin Circle Theorem, finding a good shift would be required.

Secondly, general improvements to the structure of the code could be made. Right now it requires the user to manually type in the type of the matrix, instead of the algorithm determining this automatically. Also the algorithm only returns the quantities in a specific way. This made obtaining the plots of the results easy in our case. But this might have to be changed depending on what result the user is after.

Here is also where the adaptive method should be further researched. All it does currently is to check if the error is smaller than a given tolerance. We want the program to be able to look at the data the same way as we would look at a graph and make smart decisions about when the size of the Krylov space is sufficient and terminate the running algorithm and present the results.

When the threshold is reached one can take a set of data points surrounding this point of interest and compare the differences between them to see how the error behaves as we go forward in time but also as we move backwards to the previous data points. Here one can also check the tangent lines at these data points to get a better understanding of what the graphs curvature looks like and decide whether or not the algorithm should continue.

And the final point is to make sure A is never formed explicitly. In the algorithm we form A, because of the small size of the dimension of our experiments, which should be avoided for larger dimensions. So a black box method should be used whenever A is used in a operation, which can be done with a matrix-vector multiplication through all the steps were A is used within the algorithm.

Also it is worth mentioning that there are exceptional cases, such as nilpotent matrices, that have not been accounted for in the algorithm because of their special structure. There are more of these cases that could be an obstacle for the algorithm, but the focus was mainly on the structures that were observed.

5 Conclusion

The main purpose of this thesis was to investigate how viable this algorithm would be to estimate extreme values of quadratic forms. An accuracy of at least two digits is desired and from the results we see that this is indeed very possible. It all depends on the conditions of the certain problem.

For symmetric and skew-symmetric matrices we can definitely see a plausible chance that this method would be used in order to find the norms of a matrix instead of using different algorithms. A two digit accuracy could be achieved by using a subspace with two to six vectors. Both test matrices had the dimensions 300×300. And the small amount of work required to project this problem down to a small scale like that makes it very lucrative.

When the symmetry breaks, as happened with the case of the linear combination of T and S, it shows how even a small disturbance can affect the result. The plots using inverse iteration show that by just adding $0.1S$ to T, the amount of correct digits that the estimates could obtain went from twelve to six. Thus, the span of the subspace changed for the worse and failed to locate the vectors as well as the previous subspace did.

The result from the matrices K_0 (see figure 15) and K_{10} (see figure 16) are slightly odd though, the sudden decay when six vectors are used is both good and bad. The negative part is that six vectors are required to estimate a 20×20 problem. But these two matrices are very special, with their elements being very large and the lack of symmetry makes it difficult to quickly get an accurate subspace. The positive part is that when seven vectors are used the estimates are really accurate.

Even though we are looking at strict cases in this thesis, if further development and research were conducted on this topic, it could be used for general large problems. This method definitely has potential.

6 Results

First we have two matrices, S and T. These two Toeplitz matrices arise in Finite Difference methods, where S and T are both used in order to approximate the first and second derivative respectively. In differential equations originating from chemistry, the matrix T represents diffusion while S represents convection.

So let us begin with the matrix, S defined below as

$$S = \begin{bmatrix} 0 & 1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & -1 & 0 \end{bmatrix}.$$

S is a skew-symmetric matrix with interesting properties. If we consider the two matrices we are going to form, $S^\top S$ and $(S+S^\top)/2$, then $(S+S^\top)/2 = 0$ due to the fact that it is skew-symmetric. Thus making it redundant to bring up the results for $M[A]$ and $m[A]$. Another fact about S is that the non-zero eigenvalues are complex. This is because of its skew-symmetric structure and that all the elements in the matrix are real.

Assuming the dimension of A is n, where n is an even number, then there will be $\frac{n}{2}$ conjugate eigenvalue pairs. If A has dimension $n+1$ instead, this means that the matrix still has $\frac{n}{2}$ conjugate eigenvalue pairs, but there will be an additional eigenvalue without a pair that will be equal to zero. For $S^\top S$ to be singular then S has to be singular too. Thus we know that the matrices S and $S^\top S$ are singular when the dimension is odd and non-singular when the dimension is even.

Also, one does not have to fear about the imaginary part of the eigenvalues. This because the matrix $S^\top S$ is symmetric singular positive semi definite for odd dimensions and symmetric positive definite for even dimensions, so all the eigenvalues are real and non-negative.

The following two examples are when S has a dimension of 300×300 so it is non-singular. Otherwise we would have just obtained the result of $L[A]$ due to the fact there is an eigenvalue equal to zero, which is the value $l[A]$ would attain.

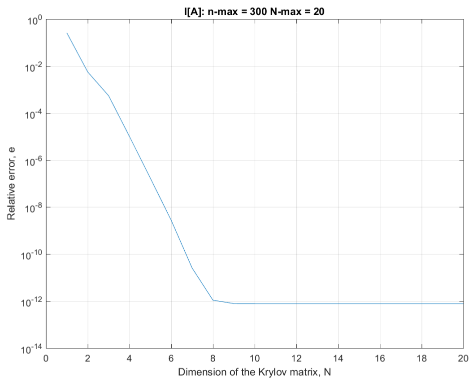

Figure 1: The relative error for approximating the constant $l[S]$ with dimension 300×300 based on the size of the Krylov space. The decay is rapid and already after two iterations we have two correct digits. Close to machine precision after $N = 8$. Shifted inverse was used here because we are trying to find the smallest eigenvalue of a positive definite matrix. The dimension was even so the matrix is non-singular. Again using shifted inverse iteration, here with the shift $= 0$ due to the non-singularity.

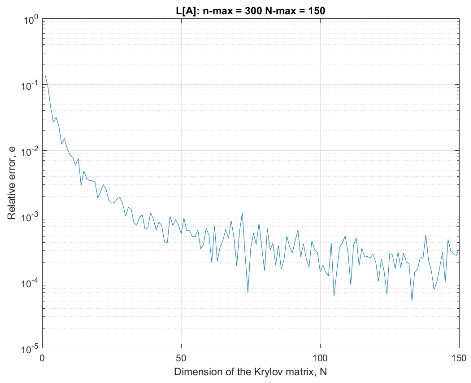

Figure 2: The relative error for approximating the constant $L[S]$ with dimension 300×300 based on the size of the Krylov space. The decrease in error is quick at the start and the but the speed almost evens out at around $N = 50$ with the error 10^{-4}. We have two correct digits after ten iterations. This was done with power iteration due to the fact that we are looking for the maximum value.

The matrix T is a symmetric negative definite matrix. This means that all the eigenvalues are real and non-positive.

$$T = \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}$$

Because T is negative definite and therefore non-singular, we see that shifted inverse iteration will yield a result instead of a warning message. T is now symmetric instead of being skew-symmetric so we have four different plots for T with dimension 300×300.

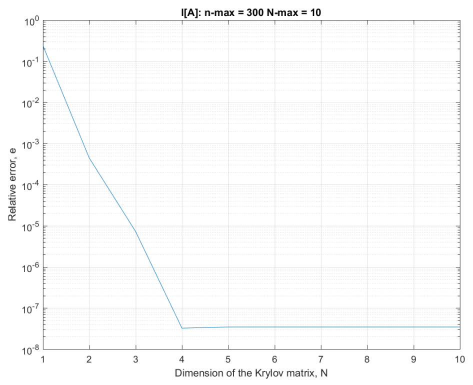

Figure 3: The relative error for approximating the constant $L[T]$ with dimension 300×300 based on the size of the Krylov space. Quick decrease until $N = 4$ when we obtain a constant error of the size 10^{-7}. Two digit accuracy after three iterations. Shifted inverse was used with the shift $= 0$, this was not an issue because T is not singular.

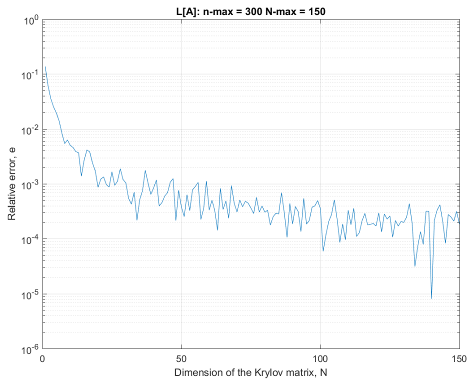

Figure 4: The relative error for approximating the constant $L[T]$ with dimension 300×300 based on the size of the Krylov space. Rapid decay until $N = 15$ when the speed evens out with an error of 10^{-3}. Two digit accuracy after six iterations. This was done with power iteration because we are looking for the highest value in magnitude.

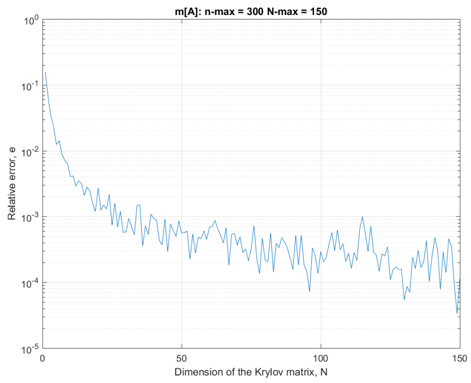

Figure 5: The relative error for approximating the constant $m[T]$ with dimension 300×300 based on the size of the Krylov space. Rapid decay until $N = 15$ when the speed evens out with an error of 10^{-3}. Two digit accuracy after six iterations. This was done with power iteration because we are looking for the lowest value in magnitude.

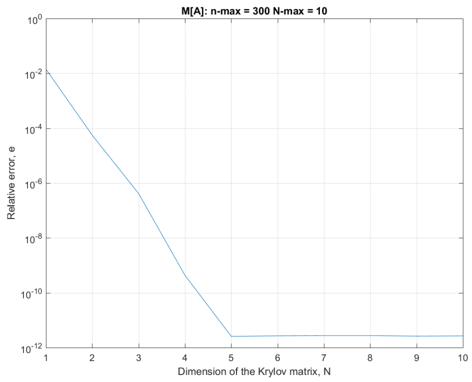

Figure 6: The relative error for approximating the constant $M[T]$ with dimension 300×300 based on the size of the Krylov space. Quick decay until $N = 5$ when we almost get machine precision. Two digit accuracy instantly. This was done with shifted inverse iteration because we are looking for the largest value of all the negative eigenvalues. A shift $= 0$ was used because it is not singular.

We can also form linear combinations out of these two matrices, $\gamma S + \beta T$. Linear combinations are important because they show what happens when symmetry breaks. An example of this is when we have both diffusion and convection in a system. We take a small γ to show the difference between the previous and the current result, in order to see how much it actually changes for a small disturbance.

$$T + 0.1S = \begin{bmatrix} -2 & 1.1 & & \\ 0.9 & \ddots & \ddots & \\ & \ddots & \ddots & 1.1 \\ & & 0.9 & -2 \end{bmatrix}$$

The following examples are created with the matrix $T + 0.1S$, which is still negative definite.

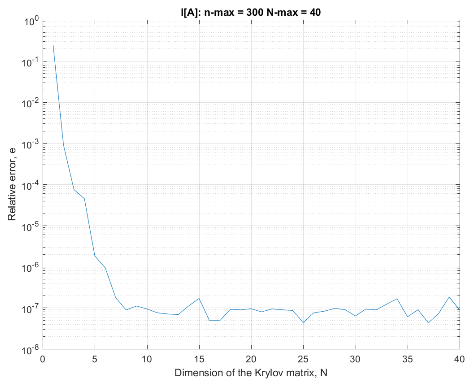

Figure 7: The relative error for approximating the constant $l[T+0.1S]$ based on the dimension of the Krylov space. Quick decay until the constant error of 10^{-7} has been achieved after seven iterations. We have a two digit accuracy after two iterations. shifted inverse iteration was used with a shift = 0.

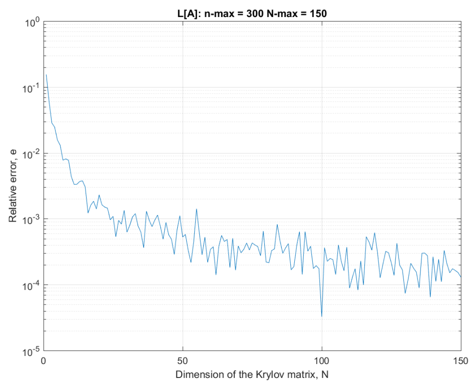

Figure 8: The relative error for approximating the constant $L[T + 0.1S]$ based on the dimension of the Krylov space. Rapid decay until $N = 20$ when the decay attains a more steadily decline wit the error of 10^{-3}. We have a two digit accuracy after seven iterations. power iteration was used to obtain the result.

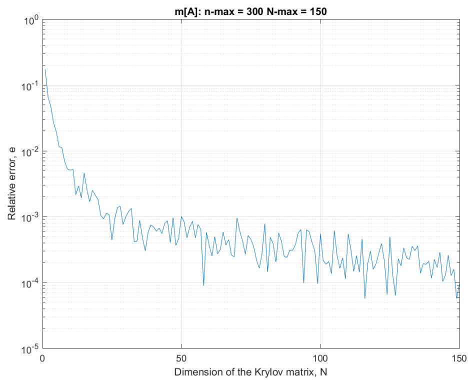

Figure 9: The relative error for approximating the constant $m[T + 0.1S]$ based on the dimension of the Krylov space. Quick decay until $N = 10$ where it stabilizes around three digit accuracy. Two digit accuracy after five iterations. power iteration was used to obtain the result.

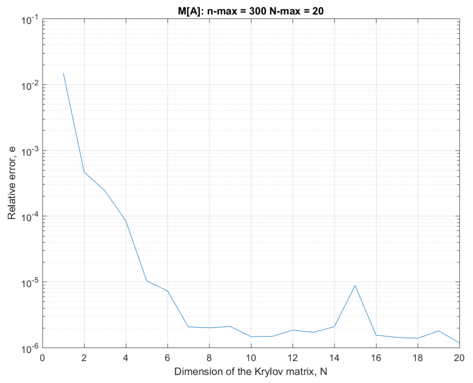

Figure 10: The relative error for approximating the constant $M[T + 0.1S]$ based on the dimension of the Krylov space. The decay is rapid and stables out at 10^{-6} where it becomes constant when $N = 7$. Two digit accuracy already after one iteration. inverse iteration with a shift $= 0$ was used.

We also consider Jan Verwer's air pollution model from the Bari Test Set [4], [5]. This is a large chemical reaction kinetics problem consisting of 20 equations with 25 widely varying reaction rate constants. The singular Jacobian can be written

$$J(x) = A \cdot K \cdot (R_0 + R_1 x),$$

where $A \in R^{20\times 25}$ is an incidence matrix of the reactions and $K \in R^{25\times 25}$ is a diagonal matrix of reaction rate constants. As the elemental reactions are of the form $r_k(x) = x_i$ or $r_k = x_i x_j$ (with $i \neq j$), it holds that

$$\mathrm{grad}_x r(x) = R_0 + R_1 x,$$

where $R_0 \in R^{25\times 20}$ is a constant matrix, and $R_1 \in R^{25\times 20\times 20}$ is a 3-tensor, implying that $R_1 x \in R^{25\times 20}$.
Thus the Jacobian is $J(x) = J(0) + (J(x) - J(0))$, where only $J(x) - J(0) = AKR_1x$ contributes to non-linearity. The Jacobian is dominated by $J(0) = AKR_0$.

We test the computation of the values for Equation 1 - 4 both for the initial condition for the solution vector x at time $t = 0$ (The matrix K_0, see figure 15) and at $t = 10$ (The matrix K_{10}, see figure 16).

Because of the law of mass conservation and the nature of our matrices, there will be an eigenvalue equal to zero. Since K_0 and K_{10} are singular then $l[K_0]$ and $l[K_{10}]$ will be equal to zero. For $L[K_0]$ and $L[K_{10}]$ we use power iteration because these quantities are the largest in absolute value. If we look at the matrix K_0, because no method for obtaining a shift was developed for this thesis, it is hard to find both of the values of $m[K_0]$ and $M[K_0]$. Power iteration will find the largest one in absolute value but the other one needs to be found with shifted inverse iteration, which is not possible with our algorithm. By using `eig` for the matrix $\frac{K_0^\top + K_0}{2}$ we found that the largest eigenvalue in absolute value was $m[K_0]$. The same thing applied to the matrix K_{10}.

Thus, only estimations for $L[A]$ and $m[A]$ are presented in the results, both obtained by using power iteration.

The following two examples are obtained when using the matrix K_0 from figure 15.

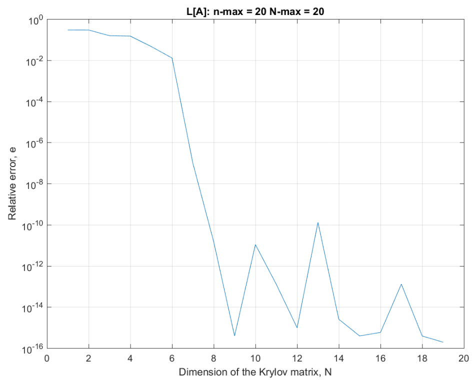

Figure 11: The relative error for approximating the constant $L[K_0]$ based on the dimension of the Krylov space. The decrease in error is slow in the beginning but after $N = 6$, when we have two digit accuracy, the decay drops substantially after, to machine precision in fact. Power iteration was used here.

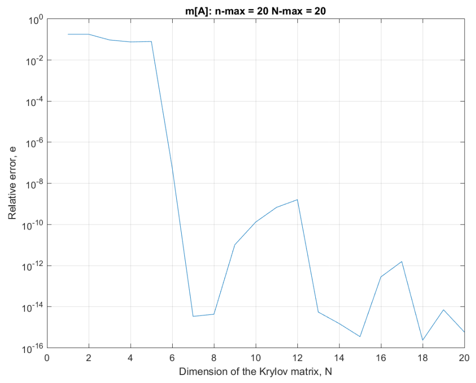

Figure 12: The relative error for approximating the constant $m[K_0]$ based on the dimension of the Krylov space. The decrease in error is slow in the beginning but after $N = 6$, when we have two digit accuracy, the decay drops substantially after, to machine precision again. Obtained using power iteration.

The following two examples are obtained when using the matrix K_{10} from figure 16.

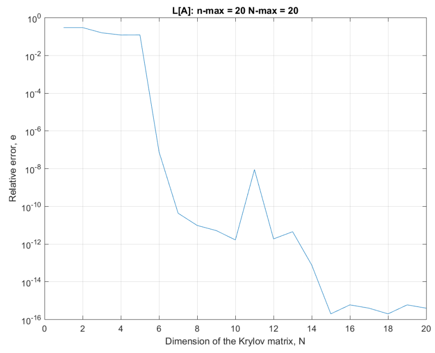

Figure 13: The relative error for approximating the constant $L[K_{10}]$ based on the dimension of the Krylov space. The decrease in error is slow in the beginning but after $N = 6$, when we have two digit accuracy, the decay drops substantially after, to machine precision again. Power iteration was used here.

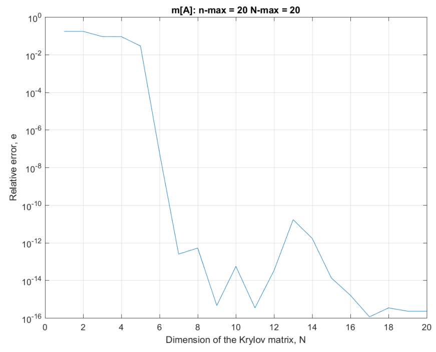

Figure 14: The relative error for approximating the constant $m[K_{10}]$ based on the dimension of the Krylov space. The decrease in error is slow in the beginning but after $N = 6$, when we have two digit accuracy, the decay drops substantially after, to machine precision again. Power iteration was used here.

7 Appendices

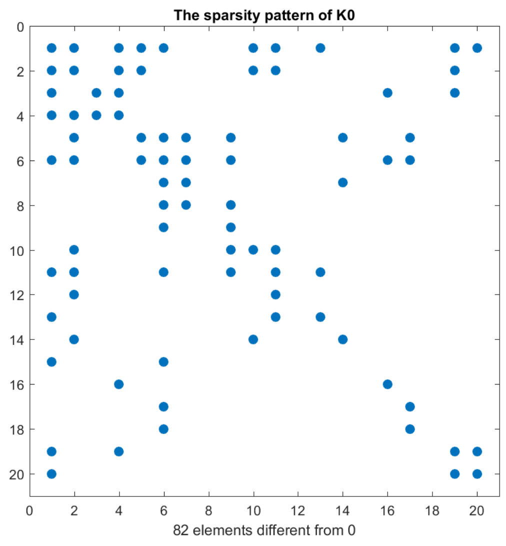

Figure 15: Displays the sparse pattern of the matrix, K_0 from Jan Verwer's air pollution model when $t = 0$. The non zero elements are very different in size, ranging from 10^{-3} to 10^{11}.

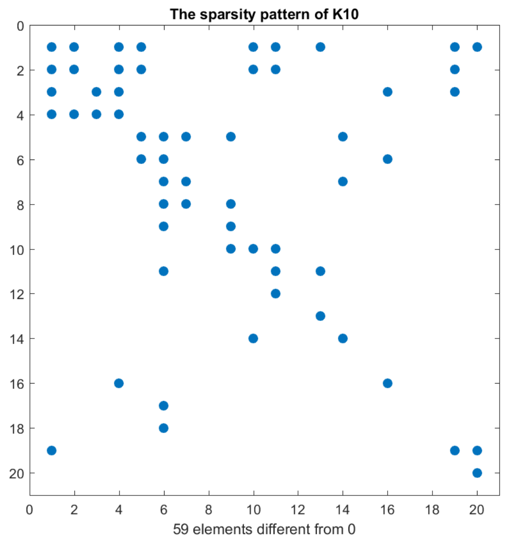

Figure 16: Displays the sparse pattern of the matrix, K_{10} from Jan Verwer's air pollution model $t = 10$. The non zero elements are very different in size, ranging from 10^{-3} to 10^{11}.

8 Code structure

This MATLAB code is constructed with six different m-files. Four of them are for the different quantities which we are trying to estimate. The two remaining files are for displaying information and to set certain parameters.

There were a few other m-files that were used to aid the six main m-files, but none of them will be presented in this thesis. 'Eigestimate.m' was one of these help files. Its purpose was to find an approximation to an eigenvalue for a matrix A. This was used to help with obtaining a good shift for shifted inverse iteration. It found an interval using the Gershgorin circle theorem where the desired eigenvalues would be contained in. Also there was an m-file called 'Matrix.m' that was used to initiate the rest of the m-files with a matrix. The matrix was either taken as input or constructed within.

These six m-files will be displayed down below, from the top of the hierarchy to the bottom.

This is the m-file 'Krylov.m', the execution file.

```


Program that calls Display and plots all the desired results. In here you
control what kind of output we wanna see with different parameters
specifying how Display should return the result.

Written (c) S Hallborn 21-05-2017
%}

%User can specify the desired dimensions, n dim of the input matrix and N
%the size of the krylov space.
n = 300;
N = 150;

%number of iterations, (graphs on the window)
iterations = 1;

% Typematrices: 1 = PD, -1 = ND, 0 = ID, specified by the use
Typematrix = -1;


% l[A]: lType = 0, L[A]: lType = 1, m[A]: mType = 0 or M[A]: mType = 1
%(values that we want to obtain)
lType = 1;
mType = 0;

% nType = 0 is plotting the error vs Size N of Krylov matrix
% nType = 1 is plotting the error vs Size n of A
% nType = 2 is gives the approximations for a given matrix A
%using adaptive methods.
nType = 0;

% tolerance for adaptive method.
tol = 0.01;

```

```
% Shift used in shifted inverse iteration
s = 0;

% Calculating and displaying Maverage

[a1,littlem] = msestimator(Typematrix,Matrix(n,n),rand(n,1),N,s);
[a2,bigm] = Mestimator(Typematrix,Matrix(n,n),rand(n,1),N,s);

mavg = (littlem + bigm)/2

% Cosmetic for the output

if lType == 0
    lfont = 'l[A]';
else
    lfont = 'L[A]';
end

if mType == 0
    mfont = 'm[A]';
else
    mfont = 'M[A]';
end

% Plotting begins

if nType == 1

for j=1:iterations
[x,l,k] = Display(Typematrix,nType,lType,mType,n,N,tol,s);
semilogy(x,abs(l))
hold on
grid on
title([lfont,': n-max = ',num2str(n),' N-max = ',num2str(N)])
ylabel('Relative error, e')
xlabel('Dimension of A, n')
end


figure
for b =1:iterations
[x,l,k] = Display(Typematrix,nType,lType,mType,n,N,tol,s);
semilogy(x,abs(k))
hold on
grid on
title([mfont,': n-max = ',num2str(n),' N-max = ',num2str(N)])
ylabel('Relative error, e')
xlabel('Dimension of A, n')
end

else if nType == 0

for j=1:iterations
[x,l,k] = Display(Typematrix,nType,lType,mType,n,N,tol,s);
semilogy(x,abs(l))
hold on
grid on
title([lfont,': n-max = ',num2str(n),' N-max = ',num2str(N)])
ylabel('absolute error, ')
```

```
xlabel('Dimension of the Krylov matrix, N')
end
figure
for b =1:iterations
[x,l,k] = Display(Typematrix,nType,lType,mType,n,N,tol,s);
semilogy(x,abs(k))
hold on
grid on
title([mfont,': n-max = ',num2str(n),' N-max = ',num2str(N)])
ylabel('Relative error, e')
xlabel('Dimension of the Krylov matrix, N')
end

    else
    [x,l,k,lada,mada] = Display(Typematrix,nType,lType,mType,n,N,tol,s);
    end

end
```

This is the m-file 'Display.m', the display file.

```

function [x,l,k,lada,mada] = Display(Typematrix,nType,lType,mType,n,N, tol,s)

%{
Function to display the results from the 4 estimator functions.

Inputs:

Typematrix - Type of A, -1 = negative definite, 1 = positive definite,
0 = indefinite.

nType - Changes the output layout;
nType = 0 is plotting the error vs Size N of Krylov matrix
nType = 1 is plotting the error vs Size n of A
nType = 2 is gives the approximations for a given matrix A
using ana adaptive method.

lType - The desired l/L [A] quantity (values that we want to obtain);
l[A]: lType = 0
L[A]: lType = 1

mType - The desired m/M [A] quantity (values that we want to obtain);
m[A]: mType = 0
M[A]: mType = 1

n - dimension of the initial matrix A.
N - Dimension of the krylov space
tol - Given tolerance to determine when the adaptive method should
terminate.
s - shift for inverse iteration

Outputs:

x - vector that is used as a x-axis for the plotting later
l - list of the errors from estimating l[A] or L[A] depending on the user
k - list of the errors from estimating m[A] or M[A] depending on the user
lada - The error from estimating l[A] or L[A] with the adaptive method.
mada - - The error from estimating m[A] or M[A] with the adaptive method.


Written (c) S Hallborn 21-05-2017
%}




l = [];
k = [];
%Typematrix only shows the name of the matrix in the plot
% This block activates if we want little n on the x-axis

if nType == 1

    % Displays the relative error based on the dimension of A

    x = 1:1:n;
    % n calculations are made
    for b = 1:n
    if lType == 1
       lada = Lestimator(Typematrix,Matrix(b,n),rand(b,1),N);
    else
       lada = lsestimator(Typematrix,Matrix(b,n),rand(b,1),N,s);
    end
    if mType == 1
       mada = Mestimator(Typematrix,Matrix(b,n),rand(b,1),N,s);
    else
       mada = msestimator(Typematrix,Matrix(b,n),rand(b,1),N,s);
    end
       l = [l;lada];
       k = [k;mada];
    end
else if nType == 0

 % Displays the relative error based on the dimension of the Krylov space

    x = 1:1:N;
    % N calculatins are made
    for b = 1:N
    if lType == 1
       lada = Lestimator(Typematrix,Matrix(n,n),rand(n,1),b);
    else
       lada = lsestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
    end
    if mType == 1
       mada = Mestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
    else
       mada = msestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
       %y1 = mestimator(Typematrix,Matrix(b,n),rand(b,1),N);
    end
       l = [l;lada];
       k = [k;mada];

    end

    else

    % Displays the result of the adaptive method, i.e when the tolernace
    % has been achieved.

    lada = 100;
    mada = 100;
    variable = 0;

    fprintf('The given tolernace is tol = %i.\n',tol)

    x = 1:1:N;
    % N calculatins are made
    for b = 1:N
        if abs(lada) >= tol

    if lType == 1
       lada = Lestimator(Typematrix,Matrix(n,n),rand(n,1),b);
       variable = 'L';
    else
       lada = lsestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
```

```
          variable = 'l';
     end
          else
          disp([num2str(variable),' has been approximated at N = ', num2str(b-1)])
          fprintf('With the error %i.\n',lada)
          break
          end
     end

     for b = 1:N
          if abs(mada) >= tol

     if mType == 1
         mada = Mestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
          variable = 'M';
     else
         mada = msestimator(Typematrix,Matrix(n,n),rand(n,1),b,s);
          variable = 'm';
     end

          else
          disp([num2str(variable),' has been approximated at N = ', num2str(b-1)])
          fprintf('With the error %i.\n',mada)
          break
          end
     end
end

end
```

This is the m-file 'lsestimate.m', the $l[A]$ estimator file.

```

function [lerr] = lsestimator(~,A,x0,N,s)
%{
Function to estimate the error between the theoretical and practical value
of the quantity l[A].

Inputs:

A - The input
x0 - randn(n,1) initial vector
N - Dimension of the krylov space
s - shift for inverse iteration

Outputs:

lerr - relative error between l[A] and the practical value


Written (c) S Hallborn 21-05-2017
%}



%theoretical value, the use of eig( ) of this big matrix can be justified
% by that it is only done in a test program like this. In the finished
% product we never even form m, because we will rely on the result of the
% code.
m = min(sqrt(abs(eig(A'*A))));
lex = m;

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(m)
    disp('Trying to approximate 0')
    lerr = 0;

else

%Practical shift, estimated with eigestimate)
[m,M] = eigestimate(A);
%Shift
L = s*m;
shift = -1;

% Shifting the matrix A and creating a new matrix A2.
A2 = A - shift*L*eye(size(A));

X = [];
xi = x0;
% A2 is used to form the Krylov matrix, X
% shifted inverse iteration is always used for l[A], for any A.
for i=1:N
    xi = A2\xi;
    xi = xi/norm(xi);
    X = [X xi];
end

```

```
% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix AQ is formed, with alot lower dimension than A.
AQ = A*Q;
aHa = AQ'*AQ;

% The practical value is obtained.
lprac = min(sqrt(abs(eig(aHa))));

% Relative error is obtained and is outputted.
lerr = (lprac - lex)/lex;
end
end
```

This is the m-file 'Lestimator', the $L[A]$ estimator file.

```

function [Lerr] = Lestimator(~,A,x0,N)

%{
Function to estimate the error between the theoretical and practical value
of the quantity L[A].

Inputs:

A - The input matrix
x0 - randn(n,1) initial vector
N - Dimension of the krylov space

Outputs:

Lerr - relative error between L[A] and the practical value


Written (c) S Hallborn 21-05-2017
%}


% Step 1 Create deterministic seed which is "irregular"
%{
D = size(A,1);
x = (1:D)'/D;
uno = ones(D,1);
y = x - uno/2;
x0 = uno - y.^2 - y + sin(7*pi*x);
%}

%theoretical value, the use of eig( ) of this big matrix can be justified
% by that it is only done in a test program like this. In the finished
% product we never even form m, because we will rely on the result of the
% code.
M = max(sqrt(eig(A'*A)));
Lex = M;

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(M)
    disp('Trying to approximate 0')
    Lerr = 0;

else


X = [];
xi = x0;
% power iteration is always used for l[A], for any A.
for i=1:N
    xi = A*xi;
    xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
```

```
[Q,R] = qr(X,0);

% Our new matrix AQ is formed, with alot lower dimension than A.
AQ = A*Q;
aHa = AQ'*AQ;

% The practical value is obtained.
Lprac = max(sqrt(eig(aHa)));

% Relative error is obtained and is outputted.
Lerr = (Lprac - Lex)/Lex;
end
end
```

This is the m-file 'msestimator.m', the $m[A]$ estimator file.

```
 function [merr, mprac] = msestimator(Typematrix,A,x0,N,s)
%{
Function to estimate the error and the practical estimation of
$m[A]$ depending on what matrix we have as input.

Inputs:

A - The input matrix
Typematrix - Type of A, -1 = negative definite, 1 = positive definite,
0 = indefinite.
x0 - randn(n,1) initial vector
N - Dimension of the krylov space
s - shift for inverse iteration

Outputs:

mprac - Estimated value of m[A]
merr - relative error between m[A] and mprac

Written (c) S Hallborn 21-05-2017
%}



% theoretical value, the use of eig( ) of this big matrix can be justified
% by that it is only done in a test program like this. In the finished
% product we never even form m, because we will rely on the result of the
% code.
mex = min(eig(He(A)));

% Choose shift (if needed)
X = 0;
shift = -1;
HeA = He(A);

%q = eig(A);
%mex = min(q);
%min(q);

if Typematrix == -1
%For negative definite matrices

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(mex)
    disp('Trying to approximate 0')
    merr = 0;
    mprac = 0;

else

X = [];
xi = x0;

% power iteration is used for negatove definite matrices.
for i=1:N
    xi = A*xi;
    xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);

% The practical value is obtained.
mprac = min(eig(QHeAQ));

% Relative error is obtained and is outputted.
merr = (mprac - mex)/mex;

end

else if Typematrix == 1

% For positivte definite matrices

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(mex)
    disp('Trying to approximate 0')
    merr = 0;
    mprac = 0;

else

%Practical shift, estimated with eigestimate)
[m,Minf] = eigestimate(A);
%Shift
L = s*m;

% Shifting the matrix A and creating a new matrix A2.
A2 = A + shift*L*eye(size(A));


X = [];
xi = x0;
% A2 is used to form the Krylov matrix, X
% shifted inverse iteration is used for positivte definite matrices
for i=1:N
    xi = A2\xi;
    xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);

% The practical value is obtained.
mprac = min(eig(QHeAQ));

% Relative error is obtained and is outputted.
```

```
merr = (mprac - mex)/mex;
end

 else
    % For indefinite matrices

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(mex)
     disp('Trying to approximate 0')
     merr = 0;
     mprac = 0;
else

% Step 2 Create Krylov subspace
X = [];
xi = x0;

% power iteration is used for indefinite matrices.
for i=1:N
     xi = A*xi;
     xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);

% The practical value is obtained.
mprac = min(eig(QHeAQ));

% Relative error is obtained and is outputted.
merr = (mprac - mex)/mex;
end
     end
end
 end
```

This is the m-file 'Mestimator.m', the $M[A]$ estimator file.

```

 function [Merr, Mprac] = Mestimator(Typematrix,A,x0,N,s)
%{
Function to estimate the error and the practical estimation of
$M[A]$ depending on what matrix we have as input.

Inputs:

A - The input matrix
Typematrix - Type of A, -1 = negative definite, 1 = positive definite,
0 = indefinite.
x0 - randn(n,1) initial vector
N - Dimension of the krylov space
s - shift for inverse iteration

Outputs:

Mprac - Estimated value of M[A]
Merr - relative error between M[A] and Mprac


Written (c) S Hallborn 21-05-2017
%}



%theoretical value, the use of eig( ) of this big matrix can be justified
% by that it is only done in a test program like this. In the finished
% product we never even form m, because we will rely on the result of the
% code.
M = max(eig(He(A)));

X = 0;
shift = -1;
HeA = He(A);


if Typematrix == 1
% For positive definite matrices
Mex = M;

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(Mex)
    disp('Trying to approximate 0')
    Merr = 0;
    Mprac = 0;
else


X = [];
xi = x0;

% power iteration is used for positive definite matrices.
for i=1:N
    xi = A*xi;
    xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);

% The practical value is obtained.
Mprac = max(eig(QHeAQ));

% Relative error is obtained and is outputted.
Merr = (Mprac - Mex)/Mex;
end

else if Typematrix == -1
    % For negative definite matrices


%theoretical value, the use of eig( ) of this big matrix can be justified
% by that it is only done in a test program like this. In the finished
% product we never even form m, because we will rely on the result of the
% code.
Mex = max(eig(He(A)));

% Failsafe so we can be assured that we don't try and approximate 0.
if abs(10^(-16)) >= abs(Mex)
     disp('Trying to approximate 0')
     Merr = 0;
     Mprac = 0;
else

%Practical shift, estimated with eigestimate)
[minf,M] = eigestimate(A);
%shift
L = -s*M;

% Shifting the matrix A and creating a new matrix A2.
A2 = A - shift*L*eye(size(A));


X = [];
xi = x0;

% A2 is used to form the Krylov matrix, X
% shifted inverse iteration is used for negative definite matrices.
for i=1:N
     xi = A2\xi;
     xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);
Mprac = max(eig(QHeAQ));
```

```

% The practical value is obtained.
Merr = (Mprac - Mex)/Mex;

% Relative error is obtained and is outputted.
M = Mprac;

end

    else

    % For indefinite matrices
    Mex = M;

    if abs(10^(-16)) >= abs(Mex)
    disp('Trying to approximate 0')
    Merr = 0;
    Mprac = 0;
    else

X = [];
xi = x0;

% power iteration is used for indefinite matrices.
for i=1:N
    xi = A*xi;
    xi = xi/norm(xi);
    X = [X xi];
end

% An orthogonal base, Q is obtained from the krylov matrix
[Q,R] = qr(X,0);

% Our new matrix QHeAQ is formed, with alot lower dimension than A.
QHeAQ = He(Q'*HeA*Q);

% The practical value is obtained.
Mprac = max(eig(QHeAQ));

% Relative error is obtained and is outputted.
Merr = (Mprac - Mex)/Mex;
end
    end

    end
 end
```

References

[1] L.N. Trefethen, D.B. Bau. *Numerical Linear Algebra, Pages 243-255* SIAM, 1997.

[2] Y.Saad. *Iterative Methods For Sparse Linear Systems.* 2nd edition, Section 6-7, SIAM, 2003.

[3] G.H.Golub, C.F.V.Loan. *Matrix Computations*, Pages 208-219, The Johns Hopkins University Press, Baltimore, Maryland, USA

[4] F. Mazzia, C. Magherini. *Test Set for Initial Value Problem Solvers*, Pages II-2-1 - II-2-9, , Universita degli studi di Bari, Italia, https://archimede.dm.uniba.it/ testset/report/pollu.pdf

[5] F Mazzia, J. R. Cash, and K. Soetaert. *A test set for stiff initial value problem solvers in the open source software R: Package deTestSet*, JCAM 236 (2012), Pages 4119-4131.

[6] *Logarithmic norms, Springer Encyclopedia of Applied and Computational Mathematics, Pages 826-830, (2015).*

[7] G. Söderlind. *The logarithmic norm. History and modern theory. BIT Numerical Mathematics* ***46*** *(2006) Pages: 631-652.*

[8] G. Söderlind, L. Jay, M. Calvo. *Stiffness 1952–2012. Sixty years in search of a definition. BIT Numerical Mathematics* ***55*** *(2015) Pages: 531–558, SpringerLink http://link.springer.com/article/10.1007/s10543-014-0503-3*