MTH343 Final Project - Lanczos Algorithm

Will Wolff-Myren (willw@pdx.edu)

2018-03-19

Overview

In this programming assignment, I will create a library of functions for operating on matrices stored in a Compressed Sparse Row format, as well as a presentation layer (web application), using Angular and TypeScript.

More specifically, this library will be able to:

- Read a matrix A from a CSR-format file
- Produce $B = A^T$, and store B in CSR format.
- Compute $C = AB$, check C for symmetry, and store C in CSR format.

This application will also:

- Implement the Lanczos algorithm, to generate the tridiagonal matrix $T = Q^T AQ$, with the option to stop at any step $m \geq 1$, where $Q = [q_1, q_2, ..., q_m]$ will consist of only the first m orthonormal vectors copmuted by the Lanczos algorithm.
- Compute the eigenvalues of the tridiagonal matrix T, for selected values of m.

In the subsequent sections of this report, I will briefly explain the algorithms used, as well as technical implementation details, and an analysis of the behavior of the eigenvalues of T for increasing values of m.

Technical Implementation

I decided to base my implementation around the Angular application framework, as a chance to utilize some of the experience that I already have with Angular and with TypeScript in general. I wanted to implement this assignment in such a way that it could be easily demonstrated without needing to be compiled/installed, and I thought it would be easier to visualize within a web browser.

In addition to the benefit of the inherent portability of constructing an application in a format that can be statically hosted (in other words, it does not require a server to run; all of the processing happens on the client, which in this case is the browser, or node), JavaScript and TypeScript have somewhat unique properties that I wanted to utilize in my matrix construction.

There are a variety of well-established numerical libraries available, including BLAS, EISPACK, LAPACK, and SciPy, but relatively few (that I am aware of) at the moment which are available for JavaScript, and even fewer available for TypeScript.

While planning my implementation approach, I investigated the following JavaScript/TypeScript libraries for use in this project: * Bluemath * Math.js * scijs

I first chose Bluemath (@bluemath/common, @bluemath/linalg), as it was directly compatible with TypeScript, but shifted to Math.js early on, mainly due to the fact that Math.js had a .transpose() method available. Unfortunately, the TypeScript definitions for the mathjs package are out of date, so I had to implement more workarounds than I would have liked, to add support for the new API methods that did not have corresponding definitions in the @types/mathjs package.

While all of the libraries that I investigated have similar core features, I found that both Bluemath and scijs use an NDArray implementation internally, which appears to be a better option for matrix manipulation than I found with Math.js' DenseMatrix and SparseMatrix implementations.

Analysis

I must admit, this was significantly more challenging than I had originally expected. I was hoping that my choice of programming language (TypeScript) would not be a significant roadblock, but I may have inadvertantly given myself a much harder problem to solve than I had planned.

I evaluated a number of available mathematics libraries for JavaScript, but settled on math.js early on, as it had *most* of the functions that I was looking for, it appeared to be well maintained, and it had existing type definitions for TypeScript. This is where my issues began.

The type definitions for mathjs (@types/mathjs) do not match the current API for the corresponding npm package (mathjs), so many of the new and particularly useful functions available from the mathjs library did pass type checks in my TypeScript applicaton. Rather than select a different library, I attempted to write a 'shim' class around the functionality that I was aware of, with the hopes of eventually swapping the underlying implementation of mathjs while keeping my interface intact.

For the most part, the 'shim' class approach actually led me to a nice implementation that is (now) well unit-tested, but took me a significantly longer amount of time than I had expected. Nonetheless, I pushed on, and not only fully implemented (as far as my needs were concerned) a Matrix class, but also implemented a functional Arnoldi algorithm in the corresponding Arnoldi class.

Again, I shot myself in the foot a bit here, as the end goal of the assignment is to implement and test a *Lanczos* algorithm implementation. I chose to approach the problem by constructing the comparatively 'complex' Arnoldi algorithm with the intention of extending it to implement the Lanczos algorithm, which is a simplified version of Arnoldi that requires a symmetric matrix instead of just an n x n matrix regardless of symmetry.

Despite all of this, I successfully implemented a Matrix class, and an Arnoldi class, but realized the fault in my choice of libraries when I finally came to the validation task of the assignment:

> ... Use existing software to compute the eigenvalues of the tridiagonal matrix T...

Eigenvalue Comparison (for selected values of m)

Math.js does not (currently) have a method to calculate the eigenvalues of a matrix. While I can attempt to write (yet another) helper class atop its existing functionality, I am unfortunately out of time, and need to choose an alternate route to determine and compare the eigenvalues of the current tridiagonal matrix results that I am able to obtain from Arnoldi.getTridiagonal().

Conclusion

I should have realized earlier in the process that I was quickly headed down a rabbit-hole and pulled myself up to the surface for some additional perspective. However, I am incredibly pleased with all that I've learned through my investigation for this assignment, and I have aquired a wealth of knowledge about numerical libraries, their implementations, and about linear algebra algorithms in general.