Amy Creel
May 2, 2019
MATH 361B
Final Project: F.8 Tridiagonal

# Tridiagonal Matrix Solver

## Algorithm

**Input:** tridiagonal matrix $A$, right-hand side vector $d$
**Output:** solution vector $x$
**Body:**

1. Create arrays for each of the three diagonals in the tridiagonal matrix $A$, where $a$ is the array of elements one below the diagonal, $b$ is the array of elements on the diagonal, and $c$ is the array of elements one above the diagonal.

2. Modify the coefficients of the arrays.

3. Pre-allocate the $x$-vector, and assign the last element of the $x$-vector.

4. Use backwards substitution to find an approximation to the solution at each unknown point for the $x$-vector.

# Differential Equations Solver

## Algorithm

**Input:** differential equation $f$, endpoints $aa$ and $bb$, boundary conditions $\alpha$ (corresponding to $a$) and $\beta$ (corresponding to $b$), number of unknowns $N$
**Output:** solution vector $u$
**Body:**

1. Set the step size $h$, pre-allocate $x$ values, set the right-hand side vector using the function $f$ and the pre-allocated $x$-values.

2. Create the tridiagonal matrix for the system using the centered difference formula $y'' \approx \frac{1}{h^2}(y_{i-1} - 2y_i + y_{i+1})$.

3. Create the arrays for each of the three diagonals in the tridiagonal matrix $A$, as seen in step 1 of the tridiagonal matrix solver.

4. Modify the coefficients of the matrix to use the tridiagonal system algorithm.

5. Create the solution vector and use the boundary conditions to set the first and last elements of the solution vector equal to $\alpha$ and $\beta$ respectively.

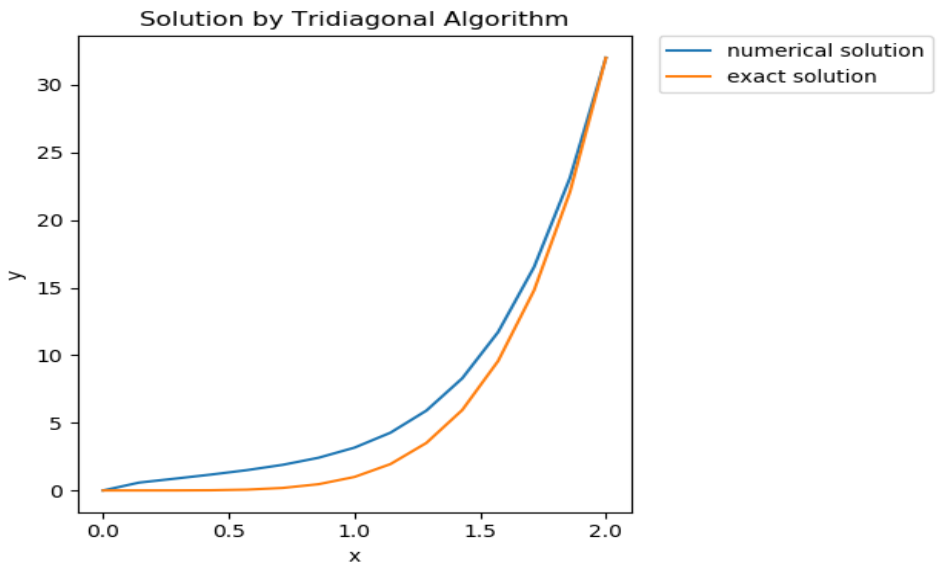
6. Loop through the pre-allocated solution vector and use backwards substitution to find an approximation to the solution at each unknown point.

## Examples

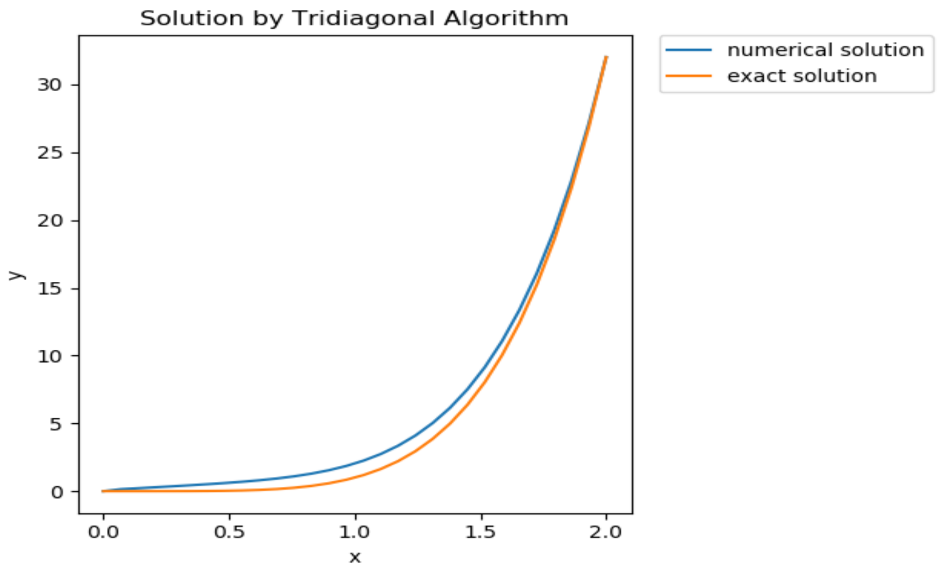
### Example 1

Let $f(x) = 20x^3$, with the domain $[0, 2]$. Then our differential equation is $u''(x) = 20x^3$, with boundary conditions $u(0) = 0$ and $u(2) = 52$. Note that the exact solution is $u(x) = x^5$.

Using the differential equations algorithm above, with $f$ as defined by $f(x)$, endpoints $a = 0$ and $b = 2$, boundary conditions $\alpha = 0$ and $\beta = 52$, and number of unknowns $N = 15$, we receive the following output:



Using the same algorithm and same inputs, but with a larger number of unknowns $N = 30$, we receive the following output:

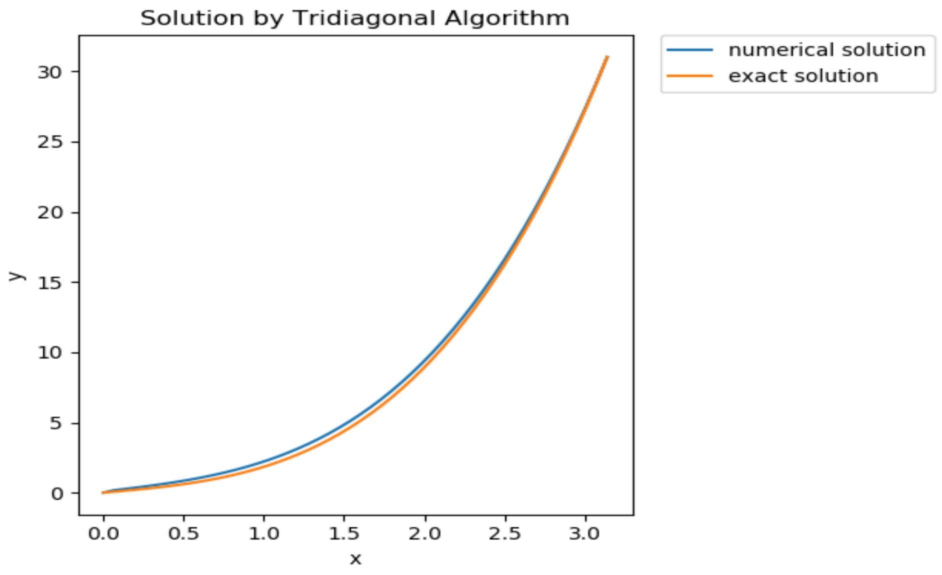


The algorithm does a fairly accurate job approximating the solution to the differential equation, especially more towards the endpoints, but the output with a larger number of unknowns was definitely more accurate, and as the number of unknowns increase, the output will become more and more accurate. The maximum error value for this example with $N = 15$ was approximately 2.392, while the maximum error value for this example with $N = 30$ was approximately 1.371. These errors are not extremely small with respect to this problem, but as the number of unknowns continue to increase, this maximum error value will decrease.

**Example 2**

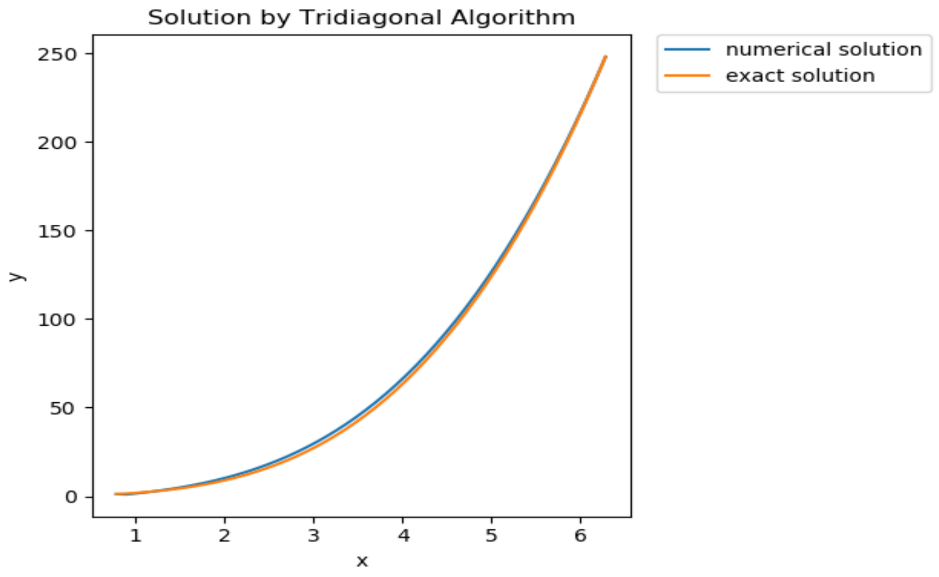
Let $f(x) = -\sin(x) + 6x$, with the domain $[0, \pi]$. Then our differential equation is $u''(x) = \sin(x) + x^3$, with boundary conditions $u(0) = 0$ and $u(\pi) = \pi^3$. Note that the exact solution is $u(x) = \sin(x) + x^3$.

Using the differential equation algorithm above, with $f$ as defined by $f(x)$, endpoints $a = 0$ and $b = \pi$, boundary conditions $\alpha = 0$ and $\beta = \pi^3$, and number of unknowns $N = 50$, we receive the following output:



Again, the approximation using this method is more accurate closer to the endpoints, which makes sense because we have the boundary conditions already set in place. The maximum error value for this example with $N = 50$ was approximately 0.4707.

Using the same algorithm, function, and step size but different endpoints and boundary conditions given by $a = \pi/4$, $b = 2\pi$, $\alpha = \sqrt{2}/2 + \pi^3/64$, and $\beta = 8\pi^3$, we obtain the following output:



The maximum error value for this example was approximately 2.957, which is higher than the previous output, but this is because the range of the function here is much larger than the range of the function in the first case, and so although the actual error value is larger, it seems that the error is relatively small in this case.

Overall, this method seems to provides accurate approximations to the solutions of a differential equation if a large enough step size is used.

## Pentadiagonal Matrix Solver

### Algorithm

**Input:** pentadiagonal matrix $A$, right-hand side vector $v$
**Output:** solution vector $x$
**Body:**

1. Create arrays for each of the five diagonals in the pentadiagonal matrix, where $d$ is the main diagonal, $g$ is the diagonal two below $d$, $h$ is the diagonal one below $d$, $e$ is the diagonal one above $d$, and $f$ is the diagonal two above $d$.

2. Pre-allocate the vectors $\alpha$, $\gamma$, $\delta$, and $\beta$ that will be used to factor the matrix $A$ into two separate matrices ($A = LR$), and then assign the first and second elements of each of these vectors.

3. Loop through the pre-allocated vectors mentioned in step 2 and assign the third through $(n-2)$ nd elements of the vectors.

4. Assign the $(n-1)$ st and $n$ th element of the necessary vectors.
5. Assign elements of the vector $c$, where $v = Lc$, using the right-hand side vector $v$ and the vectors mentioned in steps 2 through 4.
6. Use backwards substitution to solve the system $Rx = c$ for the solution vector $x$.

## Example

Define our pentadiagonal matrix $A$ and right-hand side vector $v$ to be:

$$A = \begin{pmatrix} 7 & 3 & 2 & 0 & 0 & 0 \\ 3 & 6 & 7 & 5 & 0 & 0 \\ 2 & 7 & 9 & 4 & 6 & 0 \\ 0 & 5 & 4 & 8 & 9 & 5 \\ 0 & 0 & 6 & 9 & 3 & 4 \\ 0 & 0 & 0 & 5 & 4 & 2 \end{pmatrix} \qquad v = \begin{pmatrix} 23 \\ 70 \\ 95 \\ 99 \\ 83 \\ 36 \end{pmatrix}$$

Using the pentadiagonal matrix solver with $A$ and $v$ as defined above, we get the following vector $x$ as our approximation to the solution:

$$x = \begin{pmatrix} 1 \\ 2 \\ 5 \\ 4 \\ 3 \\ 2 \end{pmatrix}$$

Note that the product of the pentadiagonal matrix $A$ and the approximation to the solution $x$ gives the right-hand side vector $v$ as expected,

$$\begin{pmatrix} 7 & 3 & 2 & 0 & 0 & 0 \\ 3 & 6 & 7 & 5 & 0 & 0 \\ 2 & 7 & 9 & 4 & 6 & 0 \\ 0 & 5 & 4 & 8 & 9 & 5 \\ 0 & 0 & 6 & 9 & 3 & 4 \\ 0 & 0 & 0 & 5 & 4 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 5 \\ 4 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 23 \\ 70 \\ 95 \\ 99 \\ 83 \\ 36 \end{pmatrix}$$

The maximum error value for the approximation to the solution vector in this example is approximately $4.26 \times 10^{-14}$, which is very small. Therefore it is reasonable to conclude that this algorithm provides accurate approximations to the solution vectors of pentadiagonal matrix systems.

## Discussion

While working on this project, I hit a lot of walls, especially when working on creating the pentadiagonal matrix solver. I thought it was going to be a lot easier to expand the tridiagonal solver to a pentadiagonal matrix, but I had to take a completely different route. Rather than just messing with the coefficients of the matrix in each of the diagonal's arrays, I had to create arrays that I could use to factor the matrix so that I could solve it one step at a time using the factorization I created. I also was still having some trouble adjusting to the 0 index that's used in Python, so there were many times that I had to search through my code to figure out if I was using the wrong index in my loops and my assignments to specific elements of the arrays. It turned out that most of the time when I was getting an error or a completely wrong answer from my code, it was because my indexing somewhere was wrong. As a result, I definitely think that this project has made me a lot more comfortable with the indexing with Python, which is extremely helpful.

My favorite part of this project was seeing visual results from the differential equations solver. Of course it was really satisfying to finally see the right numbers show up in the console and have everything match up well, but the actual plots with the approximation and exact solution took it to a completely different level, and I really enjoyed that aspect of it.