There are many situations in numerical analysis where we deal with tridiagonal systems instead of a complete set of equations.
Therefore, using the conventional Gauss-Elimination algorithm leads to various useless operations that waste resources and computational time.
One can modify the algorithm, more specifically, just the loops for traversing the column to just run through the three diagonals. And that would help you save a lot of time and redundant operations due to so many 0s in the tridiagonal system.
Let's say if a loop in i is running through the rows, then we only need to worry about the i-1, i and i+1 columns, and the last column containing the right hand side values.

You can also notice that, I've commented out the code the code for partial pivoting, as I was not sure if it was needed. Will let you know once I find out.

CODE:

```
/************************************************
********SOLVING TRIDIAGONAL SYSTEMS WITH***********
*****************GAUSS ELIMINATION*****************
************************************************/
#include<stdio.h>
#include<math.h>
/*******
 Function that performs Gauss-Elimination on a Tridiagonal system and returns the Upper
triangular matrix and solution of equations:
There are two options to do this in C.
1. Pass the augmented matrix (a) as the parameter, and calculate and store the
upperTriangular(Gauss-Eliminated Matrix) in it.
2. Use malloc and make the function of pointer type and return the pointer.
This program uses the first option.
********/
void gaussEliminationTri(int m, int n, double a[m][n], double x[n-1]){
        int i,j,k;
        for(i=0;i<m-1;i++){
                /*//Partial Pivoting
                for(k=i+1;k<m;k++){
                        //If diagonal element(absolute vallue) is smaller than any of
the terms below it
                        if(fabs(a[i][i])<fabs(a[k][i])){
                                //Swap the rows
                                for(j=i-1;j<=i+1;j++){
                                        double temp;
                                        temp=a[i][j];
                                        a[i][j]=a[k][j];
                                        a[k][j]=temp;
                                }
                                double temp;
                                temp=a[i][n-1];
                                a[i][n-1]=a[k][n-1];
                                a[k][n-1]=temp;
                        }
                }*/
```

```
                //Begin Gauss Elimination
                for(k=i+1;k<m;k++){
                        double  term=a[k][i]/ a[i][i];
                        for(j=i-1;j<=i+1;j++){
                                a[k][j]=a[k][j]-term*a[i][j];
                        }
                        a[k][n-1]=a[k][n-1]-term*a[i][n-1];
                }
        }
        //Begin Back-substitution
        for(i=m-1;i>=0;i--){
                x[i]=a[i][n-1];
                j=i+1;
                x[i]=x[i]-a[i][j]*x[j];
                x[i]=x[i]/a[i][i];
        }
}
/*******
Function that reads the elements of a matrix row-wise
Parameters: rows(m),columns(n),matrix[m][n]
*******/
void readMatrix(int m, int n, double matrix[m][n]){
        int i,j;
        for(i=0;i<m;i++){
                for(j=0;j<n;j++){
                        scanf("%lf",&matrix[i][j]);
                }
        }
}
/*******
Function that prints the elements of a matrix row-wise
Parameters: rows(m),columns(n),matrix[m][n]
*******/
void printMatrix(int m, int n, double matrix[m][n]){
        int i,j;
        for(i=0;i<m;i++){
                for(j=0;j<n;j++){
                        printf("%lf\t",matrix[i][j]);
                }
                printf("\n");
        }
}
/*******
Function that copies the elements of a matrix to another matrix
Parameters: rows(m),columns(n),matrix1[m][n] , matrix2[m][n]
*******/
void copyMatrix(int m, int n, double matrix1[m][n], double matrix2[m][n]){
        int i,j;
        for(i=0;i<m;i++){
                for(j=0;j<n;j++){
                        matrix2[i][j]=matrix1[i][j];
                }
```

```
        }
}

int main(){
        int m,n,i,j;
        printf("Enter the size of the augmeted matrix:\nNo. of rows (m)\n");
        scanf("%d",&m);
        printf("No.of columns (n)\n");
        scanf("%d",&n);
        //Declare a matrix to store the user given matrix
        double a[m][n];
        //Declare another matrix to store the resultant matrix obtained after Gauss
Elimination
        double U[m][n];
        //Declare an array to store the solution of equations
        double x[m];
        printf("\nEnter the elements of matrix:\n");
        readMatrix(m,n,a);
        copyMatrix(m,n,a,U);
        //Perform Gauss Elimination
        gaussEliminationTri(m,n,U,x);
        printf("\nThe Upper Triangular matrix after Gauss Eliminiation is:\n\n");
        printMatrix(m,n,U);
        printf("\nThe solution of linear equations is:\n\n");
        for(i=0;i<n-1;i++){
                printf("x[%d]=\t%lf\n",i+1,x[i]);
        }
}
```

OUTPUT:

```
Enter the size of the augmeted matrix:
No. of rows (m)
4
No.of columns (n)
5

Enter the elements of matrix:
2       1       0       0       1
1       2       1       0       1
0       1       2       1       1
0       0       1       2       1

The Upper Triangular matrix after Gauss Eliminiation is:

2.000000        1.000000        0.000000        0.000000        1.000000

0.000000        1.500000        1.000000        0.000000        0.500000

0.000000        0.000000        1.333333        1.000000        0.666667

0.000000        0.000000        0.000000        1.250000        0.500000


The solution of linear equations is:

x[1]=   0.400000
x[2]=   0.200000
x[3]=   0.200000
x[4]=   0.400000
```

References and resources:

https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm

https://www.npmjs.com/package/tridiagonal-solve

Android Apps:

I've also created a few Android apps that perform various matrix operations and can come in handy to those taking a course on Numerical Methods.
Download: https://play.google.com/store/apps/details?id=com.bragitoff.numericalmethods
Download: https://play.google.com/store/apps/details?id=com.bragitoff.matrixcalculator

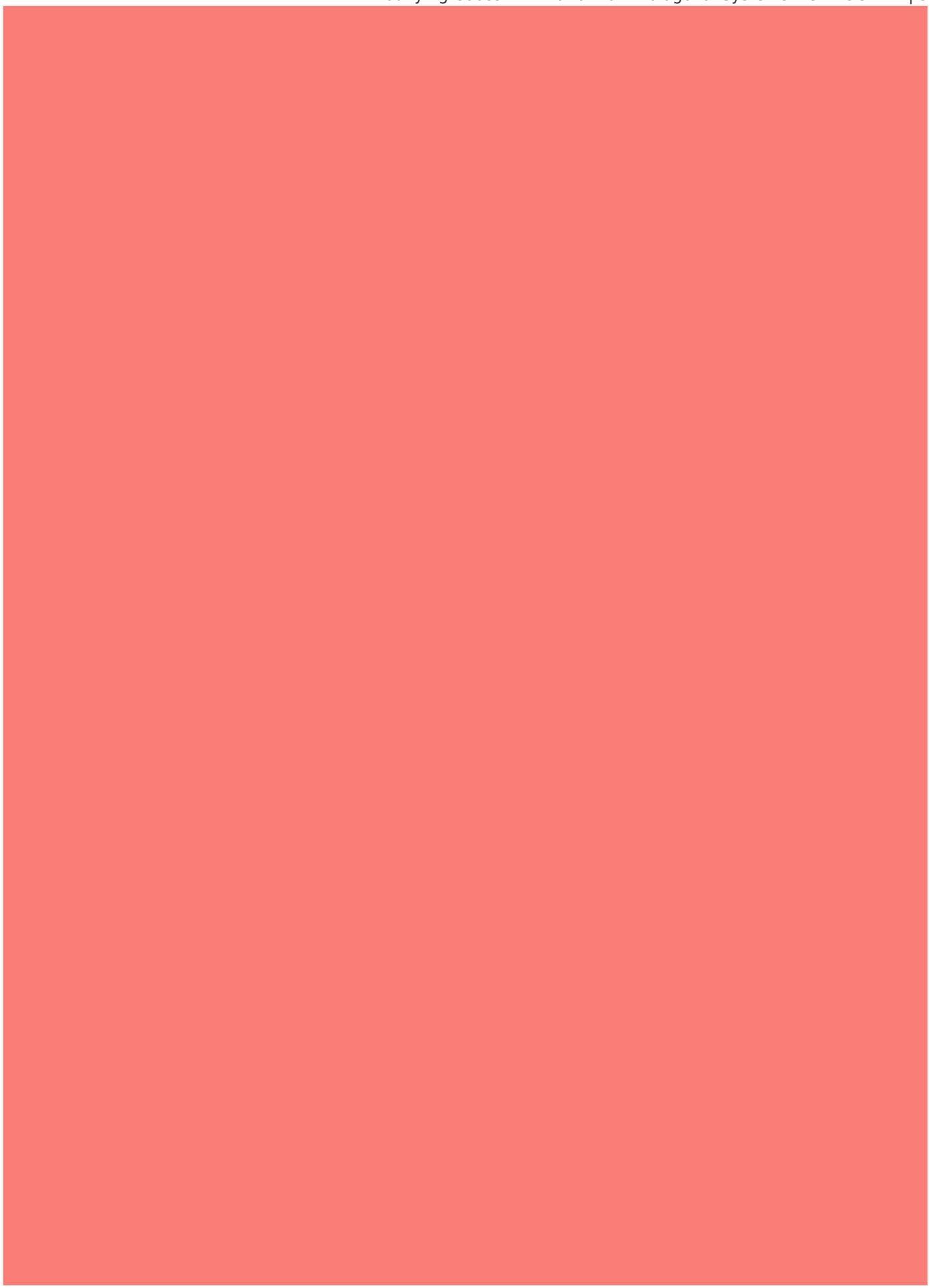
Well, that's it.
Hope you guys find it useful.

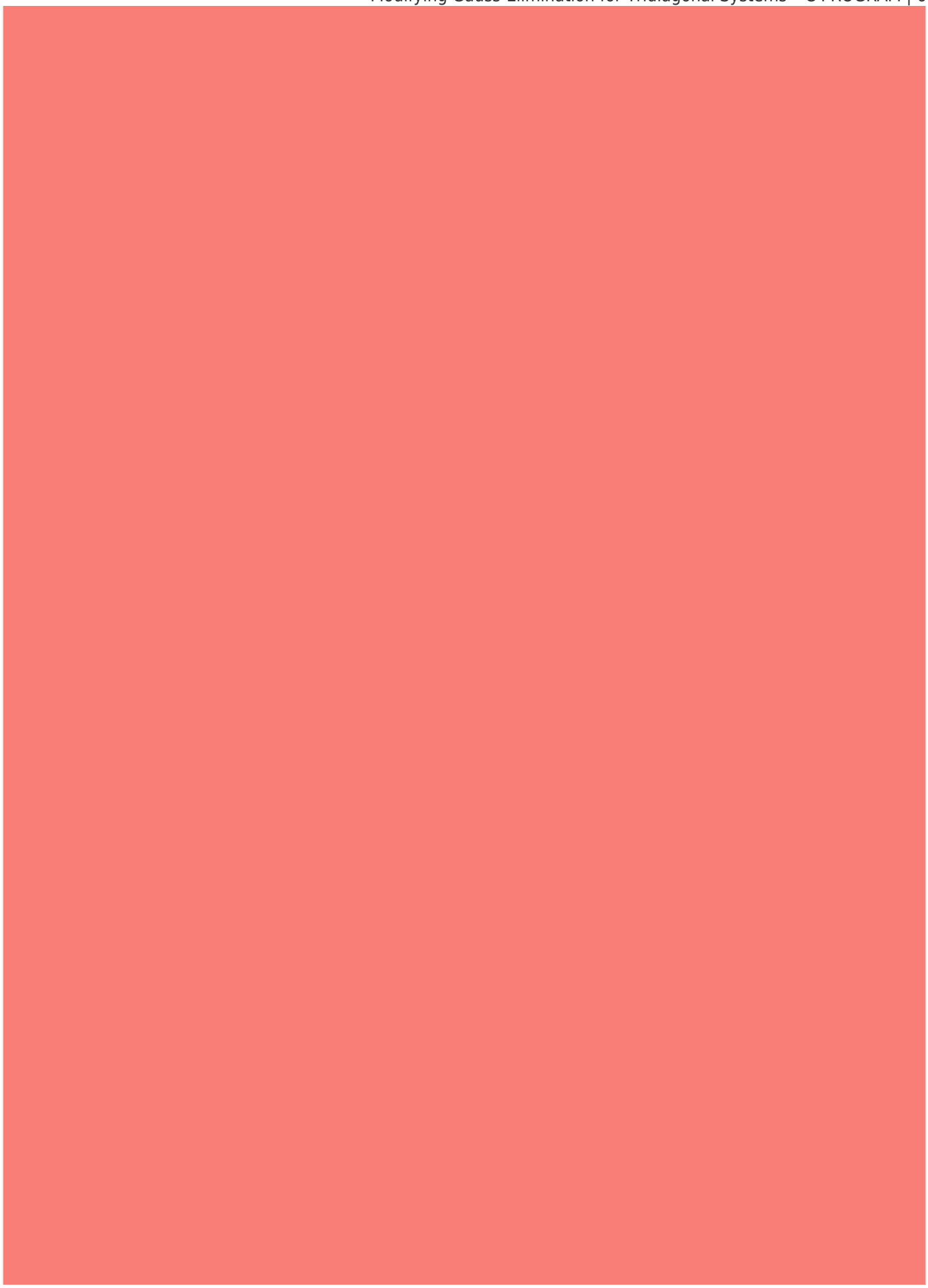
If you have any comments/questions/doubts/feedback/suggestions, leave them in the comments section down below.

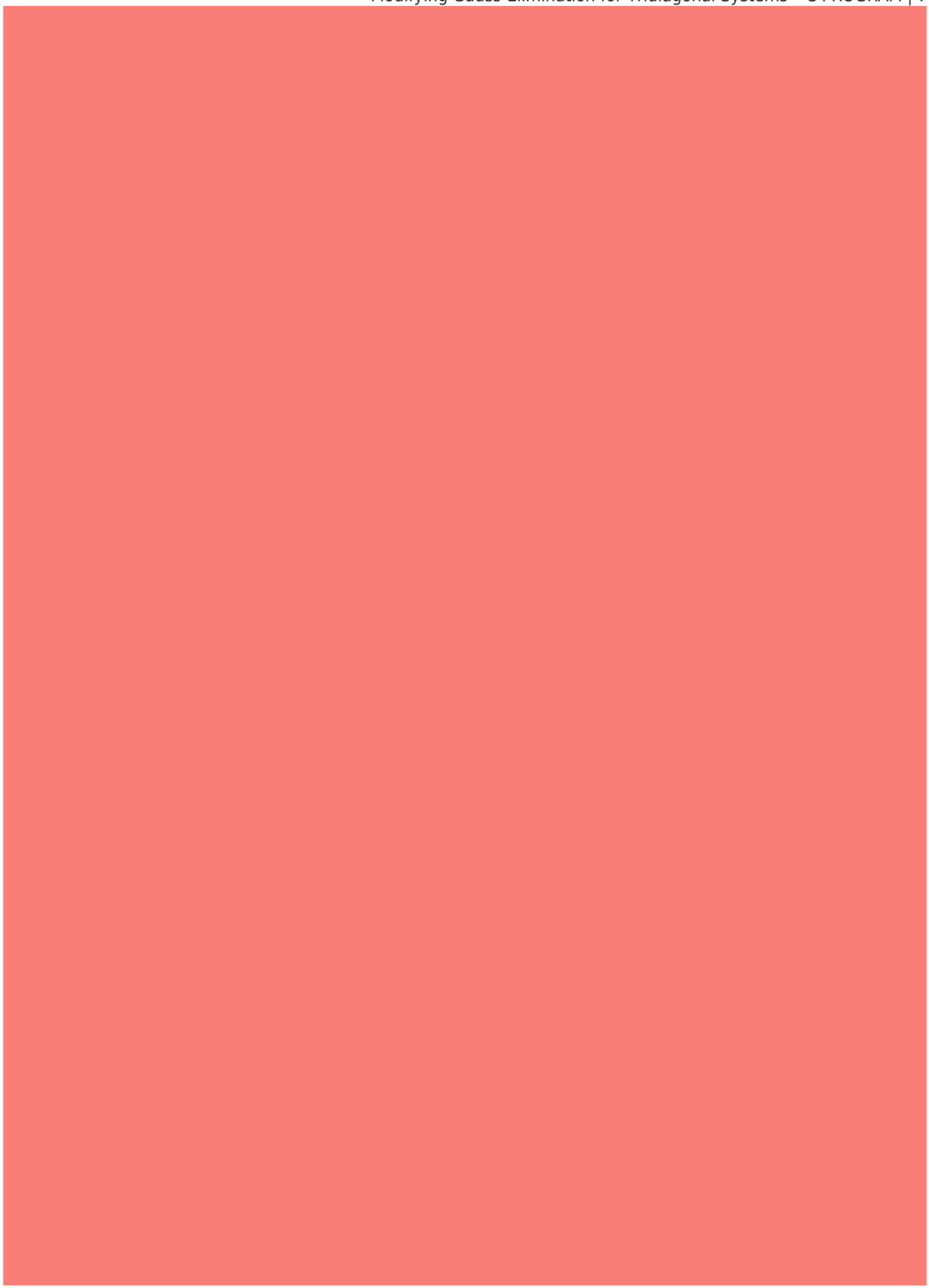
Manas Sharma

I'm a physicist specializing in computational material science with a PhD in Physics from Friedrich-Schiller University Jena, Germany. I write efficient codes for simulating light-matter interactions at atomic scales. I like to develop Physics, DFT, and Machine Learning related apps and software from time to time. Can code in most of the popular languages. I like to share my knowledge in Physics and applications using this Blog and a YouTube channel.

manas.bragitoff.com/







Share this:

Click to share on Facebook (Opens in new window)

Click to share on Twitter (Opens in new window)

Click to share on WhatsApp (Opens in new window)

Click to share on Pinterest (Opens in new window)

Click to share on Reddit (Opens in new window)

Click to share on LinkedIn (Opens in new window)

Click to email a link to a friend (Opens in new window)

Click to print (Opens in new window)

Click to share on Tumblr (Opens in new window)

Click to share on Pocket (Opens in new window)

Click to share on Telegram (Opens in new window)

[wpedon id="7041" align="center"]