Table of Contents

[INTRODUCTION 3](#_Toc446483566)

[Gauss Function 4](#_Toc446483567)

[Solving the given matrices 4](#_Toc446483568)

[Case I 4](#_Toc446483569)

[Case II 4](#_Toc446483570)

[Number of Additions/Subtractions, Multiplication and Divisions 5](#_Toc446483571)

[BGauss Function 6](#_Toc446483572)

[Solving the given matrices 6](#_Toc446483573)

[Case I 6](#_Toc446483574)

[Case II 6](#_Toc446483575)

[Number of Additions/Subtractions, Multiplication and Divisions 7](#_Toc446483576)

[Poisson’s Equation in 1D 8](#_Toc446483577)

[Gauss() 8](#_Toc446483578)

[BGauss() 8](#_Toc446483579)

[The Table that was asked for 9](#_Toc446483580)

[Information about the Table 10](#_Toc446483581)

[Analysis of Results 10](#_Toc446483582)

[Maximum Value of N 10](#_Toc446483583)

[Compiler Options 11](#_Toc446483584)

[Poisson’s Equation in 2D 12](#_Toc446483585)

[Table that was asked for 13](#_Toc446483586)

[Information about the Table 14](#_Toc446483587)

[Analysis of Results 15](#_Toc446483588)

[Maximum N 15](#_Toc446483589)

[Steps Taken to Maximize Execution Speed of Code 16](#_Toc446483590)

[-O3 Flag 16](#_Toc446483591)

[-fopenmp flag with #pragma omp commands 16](#_Toc446483592)

[Ignoring operations when ratio of row to be subtracted is 0 16](#_Toc446483593)

[Not changing values that are not accessed later 16](#_Toc446483594)

[Contour Plot 16](#_Toc446483595)

[Mastery Section: Poisson’s Equation in 3D 17](#_Toc446483596)

[Table for 3D 17](#_Toc446483597)

[Contour Plot 18](#_Toc446483598)

[Appendix 20](#_Toc446483599)

[Tables 20](#_Toc446483600)

[BGauss Poisson 1D 20](#_Toc446483601)

[Graphs 22](#_Toc446483602)

[Representative Code 23](#_Toc446483603)

[Gauss Function 23](#_Toc446483604)

[BGauss Function 24](#_Toc446483605)

[Required Matrix Functions for all Functions 25](#_Toc446483606)

[Poisson 1D 29](#_Toc446483607)

[Poisson 2D 31](#_Toc446483608)

[Poisson 3D 34](#_Toc446483609)

[Programs for Solving the given Matrices 39](#_Toc446483610)

All work done in this project is my own unless stated otherwise

INTRODUCTION

This project deals with solving linear systems of equations, .

In the first section I implement a function Guass() that uses Gaussian Elimination to turn an inputted matrix, **,** into a upper diagonal matrix through which I can then use backwards substitution to get the answer vector, .

In the second section I implement a BGauss() function which solves a banded linear system of equations. That is a system where the matrix, , has non-zero terms only on the diagonal and a certain number of diagonal bands removed from the central diagonal.

In section 3, I solve a certain case of Poisson’s equation for the 1D case using my Gauss() and BGauss() functions and compare the two methods.

In section 4, I solve a similar form of Poisson’s 1D equation from section 3 but for the 2D case instead and again compare the Gauss() function to the BGauss() function.

Finally, in the Mastery Section I solve Poisson’s Equation for the 3D case.

Gauss Function

The Gauss() function, with prototype

**double** **\***Gauss**(double** **\*\***A**,** **double** **\***y**,** **int** N**)**

solves a linear system of equations of the form , where is an matrix and is a vector of size . It does so using the Gaussian Elimination method. Here I treat and as one matrix of size and make it upper diagonal by performing row operations. Note for the purposes of this project, no pivoting has been implemented and so row operations are limited to arithmetic operations of addition, subtraction, multiplication and division of rows.

## Solving the given matrices

### Case I

Here I solve a matrix A of the form

which gives me the result

### Case II

Here I solve the given linear system of equations by and in Dr Moore’s Question Paper[[1]](#footnote-1).

I get the solution vector:

## Number of Additions/Subtractions, Multiplication and Divisions

Total number of operations for the forward elimination will be

and for the backward substitution will be

so I get a total number of floating point operations of

BGauss Function

The BGauss() function, with prototype

**double** **\***BGauss**(double** **\*\***A**,** **double** **\***y**,** **int** N**,** **int** B**)**

also solves a linear system of equations, form , where is an matrix and is a vector of size . Here only has non-zero values on a diagonal band of width .

## Solving the given matrices

### Case I

Here I change the inputted matrix into

and solve the system using my function to the desired output of

### Case II

Here I solve the same linear system of equations I did in Case II for the Gauss function. I transform the A vector into a banded matrix of the form

I get the solution vector (the same as before):

## Number of Additions/Subtractions, Multiplication and Divisions

The total number of iterations required for BGuass are:

The number of values requires for forward elimination, backward substitution and the extra values that are not looped over as j reaches the end bounds of the matrix.

This gives a total in terms of N and B of

Poisson’s Equation in 1D

In this Section we consider a 1D Poisson Equation of the form

We split up our 1D grid/line into N+1 points from 0..1 in increments of . So for . . This gives us the approximation for as

So this gives us a set of N-1 equations that can be solved using our Gauss() and BGauss() functions.

# Gauss()

For the Gauss function I use the A matrix that looks like

where the size of A is .

# BGauss()

In BGauss I use a matrix that has -2 down the middle and band width of 1. It then looks like

Its size is which is considerably smaller than the matrix created for the Gauss function for large N.

# The Table that was asked for

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | max rough | pos rough | max smooth | pos smooth | Wall Time Gauss | Speed | Wall Time BGauss | Speed |
| 8 | 5.781250 | 0.37500 | 4.062500 | 0.37500 | 0.005598 | 0.00002054 | 0.000001 | 0.0540000 |
| 16 | 4.960937 | 0.43750 | 4.062500 | 0.43750 | 0.000905 | 0.00054254 | 0.000002 | 0.0630000 |
| 32 | 4.541016 | 0.40625 | 4.101562 | 0.40625 | 0.001573 | 0.00127845 | 0.000003 | 0.0900000 |
| 64 | 4.321289 | 0.40625 | 4.101562 | 0.40625 | 0.003449 | 0.00235518 | 0.000007 | 0.0797143 |
| 128 | 4.211426 | 0.40625 | 4.101562 | 0.40625 | 0.013840 | 0.00235802 | 0.000006 | 0.1890000 |
| 256 | 4.156494 | 0.40625 | 4.101563 | 0.40625 | 0.012215 | 0.01070905 | 0.000012 | 0.1905000 |
| 512 | 4.129028 | 0.40625 | 4.101563 | 0.40625 | 0.026790 | 0.01955099 | 0.000022 | 0.2086364 |
| 1024 | 4.115295 | 0.40625 | 4.101562 | 0.40625 | 0.050985 | 0.04111254 | 0.000049 | 0.1877143 |
| 2048 | 4.108429 | 0.40625 | 4.101563 | 0.40625 | 0.101163 | 0.08290141 | 0.000096 | 0.1918125 |
| 4096 | 4.104996 | 0.40625 | 4.101563 | 0.40625 | 0.213360 | 0.15724752 | 0.002430 | 0.0151630 |
| 8192 | 4.103279 | 0.40625 | 4.101563 | 0.40625 | 0.451171 | 0.29746932 | 0.000286 | 0.2577273 |
| 16384 | 4.102421 | 0.40625 | 4.101563 | 0.40625 | 1.693254 | 0.31705493 | 0.000513 | 0.2874035 |
| 32768 | 4.101992 | 0.40625 | 4.101562 | 0.40625 | 7.529262 | 0.28521399 | 0.000902 | 0.3269335 |
| 65536 | 4.101777 | 0.40625 | 4.101562 | 0.40625 | 58.923778 | 0.14577933 | 0.001679 | 0.3512841 |
| 131072 | 4.101670 | 0.40625 | 4.101562 | 0.40625 |  |  | 0.003043 | 0.3876536 |
| 262144 | 4.101616 | 0.40625 | 4.101562 | 0.40625 |  |  | 0.006723 | 0.3509264 |
| 524288 | 4.101590 | 0.40625 | 4.101563 | 0.40625 |  |  | 0.017839 | 0.2645089 |
| 1048576 | 4.101579 | 0.40625 | 4.101565 | 0.40625 |  |  | 0.024390 | 0.3869277 |
| 2097152 | 4.101572 | 0.40625 | 4.101565 | 0.40625 |  |  | 0.053287 | 0.3542018 |
| 4194304 | 4.101568 | 0.40625 | 4.101564 | 0.40625 |  |  | 0.105457 | 0.3579537 |
| 8388608 | 4.101563 | 0.40625 | 4.101561 | 0.40625 |  |  | 0.204318 | 0.3695096 |
| 16777216 | 4.101517 | 0.40625 | 4.101516 | 0.40625 |  |  | 0.459274 | 0.3287687 |
| 33554432 | 4.100841 | 0.40625 | 4.100841 | 0.40625 |  |  | 0.823110 | 0.3668888 |
| 67108864 | 4.090507 | 0.40625 | 4.090507 | 0.40625 |  |  | 1.879811 | 0.3212981 |
| 134217728 | 3.923872 | 0.40181 | 3.923872 | 0.40181 |  |  | 3.314395 | 0.3644585 |
| 268435456 | 2.303370 | 0.37245 | 2.303370 | 0.37245 |  |  | 16.960100 | 0.1424472 |
| 536870912 | 0.965351 | 0.37494 | 0.965351 | 0.37494 |  |  | 88.737658 | 0.0544508 |

The rough and smooth solution’s maxval seem to be converging towards but once I reach about , inaccuracies start creeping in. This is due to me sending in where is smaller than DBL\_EPSILON. Also the Matrix A starts to get quite large and so during backward substitution we get inaccuracies. I tried to send in only and multiply the outputted vector by which theoretically gives the same solution but does not help solve the case of the inaccuracies which here are due to the large variation in sizes of A and Y at the end of the forward elimination. Simply stated, we are getting cases of underflow and are being unable to deal with them directly.

# Information about the Table

In the table produced above, I measure the time elapsed using wall time through the <sys/time.h>[[2]](#footnote-2) rather than measuring CPU time. CPU time gives larger values than actual elapsed time as CPU time gives the total amount of time all cores cumulatively have ‘worked’. A table the produces CPU time and wall time together for the BGauss and Gauss functions are included in the appendix where this can clearly be seen.

Secondly when calculating my values for the speed of my function, I implement a special case where I store the number of operations done by the function in and since that is returned I am able to compute speed exactly rather than using my formula. This is done because my Gauss and BGauss functions contain an if statement within the for loop which changes the value of count.

# Analysis of Results

## Maximum Value of N

Gauss stops functions for a lot smaller values of N as compared to the BGauss function. This is to be expected as for the 1D case Gauss allocates a whereas BGauss only allocates a matrix.

Furthermore, we know the sizeof a double is 8 bytes. This means that for the Gauss function the size of a matrix in Gigabytes is

so for powers of 2 from 13 onwards I get

Table 1: Size of Gauss Matrix again N=powers of 2

|  |  |
| --- | --- |
| Power of 2 | Size of Gauss Matrix (gb) |
| 13 | 0.499877937 |
| 14 | 1.999755867 |
| 15 | 7.999511726 |
| 16 | 31.99902344 |
| 17 | 127.9980469 |

For that is almost 128gb!! This amount is needed to store the A matrix for Gaussian elimination and thus is a severe limitation in terms of memory for the process. For even though around 32gb are needed for the process, the computer is able to write parts of the matrix that are non needed at the moment onto the disc and swap values later when they are needed. This is completely unfeasible for as a large part of the matrix is not stored on the RAM slowing down the process considerably due to the required constant reading and writing of data from the disc to the RAM.

In the case of BGauss() the size of the matrix needed for Gaussian elimination is

Table 2: Size of BGauss Matrix again N=powers of 2

|  |  |
| --- | --- |
| Powers of 2 | Size of Bgauss Matrix (gb) |
| 26 | 1.499999978 |
| 27 | 2.999999978 |
| 28 | 5.999999978 |
| 29 | 11.99999998 |
| 30 | 23.99999998 |

As we can see again for BGauss the function is able to use memory on the computer and “swapping” – that is writing partially onto the disc – to store the matrix but after this value is quite large but instead leads to issues in speed –due to how memory is accessed-and not so much memory allocation.

## Compiler Options

Firstly, in order to use OMP I use the flag

-fopenmp.

This makes considerable difference in the speed of my Gauss function which includes #pragma omp parallel for statements to parallelize the code. It does not make any difference for my BGauss function as parallelisation only kicks in for matrices with large Band sizes and in the 1D case the band size is 1 which simply runs in serial.

Secondly, I use compiling optimizations

-O3 (or equivalently –Ofast)

through which I get extreme speedup in my code, sometimes making it run almost 10 times faster!

Finally, using different compilers made no significant difference in the speed of executing my programs consistently. There is obviously a variation in results depending on the specifications of the machine but even when running code on different compilers on the same machine, it makes no significant difference to the general speed of the code. I ran my programs on GCC (predominantly), ICL, and Clang. On ICL however, I had to modify my code slightly (although not Gauss and BGauss) and was unable to run the code for N as large as in GCC and Clang.

Poisson’s Equation in 2D

In this section I consider a 2D Poisson Equation which can be described by splitting out domain into a grid of N points in the x direction and N points in the y direction.   
  
I then get the following equations that relate the points

In order to solve this system of equations I need be able to write it in the form , and so I arrange in the y lines. That is for N=4 I can write

My general matrix A then can be written as a composition of 2 matrices[[3]](#footnote-3)

A is then

For BGuass I then get a matrix A\_banded with a band of size , and total size of and looks like

For large N this is considerably smaller and thus allows for a lot more iterations before running out of memory.

# Table that was asked for

Below is the asked to be produced for Poisson 2D case.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | maxval rough | maxval pos x | maxval pos y | maxval smooth | maxval pos x | maxval pos y | Gauss WallTime | Speed | Bgauss WallTime | speed |
| 8 | 3.084332 | 0.37500000 | 0.37500000 | 1.670186 | 0.37500000 | 0.37500000 | 0.0025830 | 0.0070852 | 0.000008 | 0.76012500 |
| 12 | 2.579388 | 0.41666667 | 0.41666667 | 1.645431 | 0.41666667 | 0.41666667 | 0.0067550 | 0.0247433 | 0.000027 | 1.27388889 |
| 16 | 2.366303 | 0.37500000 | 0.37500000 | 1.701806 | 0.37500000 | 0.37500000 | 0.0121820 | 0.0636333 | 0.000075 | 1.52497333 |
| 20 | 2.229597 | 0.40000000 | 0.40000000 | 1.696736 | 0.40000000 | 0.40000000 | 0.0254220 | 0.0986807 | 0.000933 | 0.30813183 |
| 24 | 2.139713 | 0.37500000 | 0.37500000 | 1.708222 | 0.37500000 | 0.37500000 | 0.0336970 | 0.1927359 | 0.001758 | 0.34556143 |
| 32 | 2.029538 | 0.37500000 | 0.37500000 | 1.710522 | 0.37500000 | 0.37500000 | 0.0488720 | 0.5887506 | 0.001058 | 1.85698015 |
| 40 | 1.964529 | 0.37500000 | 0.37500000 | 1.711597 | 0.37500000 | 0.37500000 | 0.0712100 | 1.2710763 | 0.002025 | 2.40087753 |
| 48 | 1.921668 | 0.37500000 | 0.37500000 | 1.712184 | 0.37500000 | 0.37500000 | 0.1228770 | 1.8705963 | 0.004142 | 2.45563037 |
| 64 | 1.871698 | 0.39062500 | 0.39062500 | 1.714084 | 0.39062500 | 0.39062500 | 0.3588340 | 2.7691556 | 0.011202 | 2.90126736 |
| 80 | 1.841230 | 0.38750000 | 0.38750000 | 1.716023 | 0.38750000 | 0.38750000 | 0.9674900 | 3.1829690 | 0.036073 | 2.21392058 |
| 96 | 1.820595 | 0.38541667 | 0.38541667 | 1.716748 | 0.38541667 | 0.38541667 | 2.5294160 | 3.0608059 | 0.051639 | 3.22076285 |
| 128 | 1.794436 | 0.38281250 | 0.38281250 | 1.717014 | 0.38281250 | 0.38281250 | 11.3608940 | 2.9089728 | 0.150360 | 3.51462025 |
| 160 | 1.778546 | 0.38125000 | 0.38125000 | 1.716831 | 0.38125000 | 0.38125000 | 37.2975990 | 2.7251480 | 1.285858 | 1.00656396 |
| 192 | 1.768492 | 0.38541667 | 0.38541667 | 1.716998 | 0.38541667 | 0.38541667 | 97.2596490 | 2.6139264 | 1.958044 | 1.37358660 |
| 224 | 1.761210 | 0.38392857 | 0.38392857 | 1.717171 | 0.38392857 | 0.38392857 | 227.8470470 | 2.4206075 | 2.816770 | 1.77161429 |
| 256 | 1.755624 | 0.38281250 | 0.38281250 | 1.717155 | 0.38281250 | 0.38281250 | 466.2463080 | 2.3126971 | 3.639611 | 2.34165920 |
| 288 | 1.751278 | 0.38541667 | 0.38541667 | 1.717055 | 0.38194444 | 0.38194444 |  |  | 4.472066 | 3.05535286 |
| 320 | 1.747950 | 0.38437500 | 0.38437500 | 1.717180 | 0.38437500 | 0.38437500 |  |  | 5.716331 | 3.64574687 |
| 352 | 1.745154 | 0.38352273 | 0.38352273 | 1.717211 | 0.38352273 | 0.38352273 |  |  | 7.778198 | 3.92504296 |
| 384 | 1.742773 | 0.38281250 | 0.38281250 | 1.717181 | 0.38281250 | 0.38281250 |  |  | 9.524226 | 4.54208373 |
| 416 | 1.740818 | 0.38461538 | 0.38461538 | 1.717172 | 0.38461538 | 0.38461538 |  |  | 11.084334 | 5.37772819 |
| 448 | 1.739157 | 0.38392857 | 0.38392857 | 1.717217 | 0.38392857 | 0.38392857 |  |  | 11.827966 | 6.78089797 |
| 480 | 1.737682 | 0.38333333 | 0.38333333 | 1.717218 | 0.38333333 | 0.38333333 |  |  | 13.848252 | 7.63458150 |
| 512 | 1.736363 | 0.38281250 | 0.38281250 | 1.717190 | 0.38281250 | 0.38281250 |  |  | 20.807215 | 6.57952982 |
| 544 | 1.735271 | 0.38419118 | 0.38419118 | 1.717212 | 0.38419118 | 0.38419118 |  |  | 22.571478 | 7.73152002 |
| 576 | 1.734274 | 0.38368056 | 0.38368056 | 1.717227 | 0.38368056 | 0.38368056 |  |  | 28.041269 | 7.82367308 |
| 608 | 1.733362 | 0.38322368 | 0.38322368 | 1.717219 | 0.38322368 | 0.38322368 |  |  | 36.334646 | 7.49706963 |
| 640 | 1.732549 | 0.38437500 | 0.38437500 | 1.717203 | 0.38437500 | 0.38437500 |  |  | 41.278272 | 8.10341897 |
| 672 | 1.731834 | 0.38392857 | 0.38392857 | 1.717226 | 0.38392857 | 0.38392857 |  |  | 50.317568 | 8.08150517 |
| 704 | 1.731169 | 0.38352273 | 0.38352273 | 1.717230 | 0.38352273 | 0.38352273 |  |  | 55.031138 | 8.90176111 |
| 736 | 1.730547 | 0.38315217 | 0.38315217 | 1.717219 | 0.38315217 | 0.38315217 |  |  | 64.909616 | 9.01674646 |
| 768 | 1.730001 | 0.38411458 | 0.38411458 | 1.717221 | 0.38411458 | 0.38411458 |  |  | 75.250111 | 9.22220409 |
| 800 | 1.729496 | 0.38375000 | 0.38375000 | 1.717231 | 0.38375000 | 0.38375000 |  |  | 86.552141 | 9.44114188 |
| 832 | 1.729019 | 0.38341346 | 0.38341346 | 1.717229 | 0.38341346 | 0.38341346 |  |  | 104.258285 | 9.16995111 |
| 864 | 1.728573 | 0.38425926 | 0.38425926 | 1.717218 | 0.38310185 | 0.38310185 |  |  | 131.477092 | 8.45722653 |
| 896 | 1.728179 | 0.38392857 | 0.38392857 | 1.717229 | 0.383928571 | 0.38392857 |  |  | 147.043478 | 8.74672297 |
| 928 | 1.727802 | 0.38362069 | 0.38362069 | 1.717233 | 0.38362069 | 0.38362069 |  |  | 178.958844 | 8.27049241 |

# Information about the Table

Again in the table above I measure wall time rather than CPU time. I also again use the calculated number of operations done in my Gauss and BGauss functions by storing those values in the term.

As it can be seen my rough solution starts off with a much larger maxval than the smooth solution. The smooth solution converges to to its estimate of fairly quickly (by whereas the rough solution also seems to be converging but does to much more slowly. N in this case does not get large enough that we get cases of inaccuracies due to getting too small as in the 1D case.

# Analysis of Results

## Maximum N

In this case I again get limitations due to memory. As done in the 1D case, I can look at the approximate size of the matrices I create. For Guass I have that

Table 3: size of Gauss matrix against N

|  |  |
| --- | --- |
| N | Size of Gauss Matrix (gb) |
| 64 | 0.117368706 |
| 128 | 1.938228615 |
| 256 | 31.50292207 |
| 288 | 50.54959775 |
| 512 | 508.0117035 |

Which explains how I can only run my Gauss function up to as for I would need 50gb of memory!

For BGauss I have that

Table 4: size of BGauss matrix against N

|  |  |
| --- | --- |
| N | size of BGauss Matrix (gb) |
| **768** | 6.728050224 |
| **800** | 7.605576508 |
| **832** | 8.556268685 |
| **864** | 9.583056442 |
| **896** | 10.68886947 |
| **928** | 11.87663745 |

This shows that as N gets larger the amount of memory is getting quite large although it is not yet as large as in some of cases before. The Gaussian elimination on this matrix though takes a long time and slows down the code. Thus although I think it is possible to run the code for some larger values of N, I have omitted to do so due to the increase in time taken.

## Steps Taken to Maximize Execution Speed of Code

### -O3 Flag

A simple and extremely useful optimization.

### -fopenmp flag with #pragma omp commands

I use these for parallelizing my code. In the 2D case this method give me a significant speedup in elapsed time for Gauss and also for BGauss with large band values.

### Ignoring operations when ratio of row to be subtracted is 0

I have also included an if statement in my Guass and BGauss functions that stop it from doing row operations when the ratio of the terms to be deleted is zero. This considerably reduces the number of operations done my functions and as a lot of matrices inputted are sparse.

### Not changing values that are not accessed later

Furthermore, I have also stopped Gauss from subtracting the terms below the leading diagonal to zero as these values are never accessed after being used to compute the ratio of the row operation to be done. Nor are they used in the forward substitution. Similarly, in BGauss, when I am setting values diagonally down from the Central Band to zero, I don’t bother actually changing them to zero as they are never accessed later. This saves me a huge number of operations in a large matrix.

# Contour Plot

A copy of the solution contour plot for N=928 is

This is the solution for the smooth case – the rough looks identical for large N – and has been exported into excel in csv format in 33 by 33 matrix which increments x and y by

Mastery Section: Poisson’s Equation in 3D

In this section I do something very similar to what I did in the Poisson 2D case with treating the new dimension as composite matrix of block matricies.

If I call the matrix I have from Poisson 2D as then I can model the 3D case as

Here is an (N-1)^2 by (N-1)^2 matrix and is a (N-1)^2 by (N-1)^2 idetity matrix. In the matrix A, is on the diagonal (N-1) times.

So the size of A then is (N-1)^3 by (N-1)^3.

I can turn this into a block diagonal matrix by taking the band width to be is (N-1)^2.

I write the y vector now in z lines followed by the usual y lines followed by x lines.

# Table for 3D

Below is the table asked for 3D considering only the Smooth Solution

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | maxval | maxval pos x | maxval pos y | maxval pos z | wall time Gauss | speed | wall time Bgauss | speed |
| 5 | 1.767134 | 0.400000 | 0.400000 | 0.400000 | 0.0122000 | 0.005042 | 0.000112 | 0.294482 |
| 6 | 1.858112 | 0.333333 | 0.333333 | 0.333333 | 0.0061690 | 0.058282 | 0.000130 | 1.186523 |
| 8 | 1.744217 | 0.375000 | 0.375000 | 0.375000 | 0.0260050 | 0.203604 | 0.005265 | 0.304963 |
| 10 | 1.831862 | 0.400000 | 0.400000 | 0.400000 | 0.0368990 | 1.080226 | 0.003290 | 2.829189 |
| 12 | 1.669306 | 0.416667 | 0.416667 | 0.416667 | 0.0858730 | 2.331217 | 0.010748 | 3.531741 |
| 16 | 1.833507 | 0.375000 | 0.375000 | 0.375000 | 0.6426990 | 3.775032 | 0.181755 | 1.837385 |
| 20 | 1.798469 | 0.400000 | 0.400000 | 0.400000 | 4.6566430 | 3.485524 | 0.394554 | 4.442066 |
| 24 | 1.852803 | 0.375000 | 0.375000 | 0.375000 | 22.6540370 | 3.326131 | 0.739179 | 9.054424 |
| 28 | 1.835133 | 0.392857 | 0.392857 | 0.392857 | 97.5909710 | 2.798737 | 1.827984 | 11.271000 |
| 32 | 1.859844 | 0.375000 | 0.375000 | 0.375000 | 377.1358610 | 2.195604 | 5.615717 | 9.665319 |
| 36 | 1.850026 | 0.388889 | 0.388889 | 0.388889 |  |  | 16.045633 | 7.921019 |
| 40 | 1.863159 | 0.375000 | 0.375000 | 0.375000 |  |  | 37.535824 | 7.230064 |
| 48 | 1.864975 | 0.375000 | 0.375000 | 0.375000 |  |  | 136.741314 | 7.339383 |

I find that the maxval here is converging to approximately 1.865

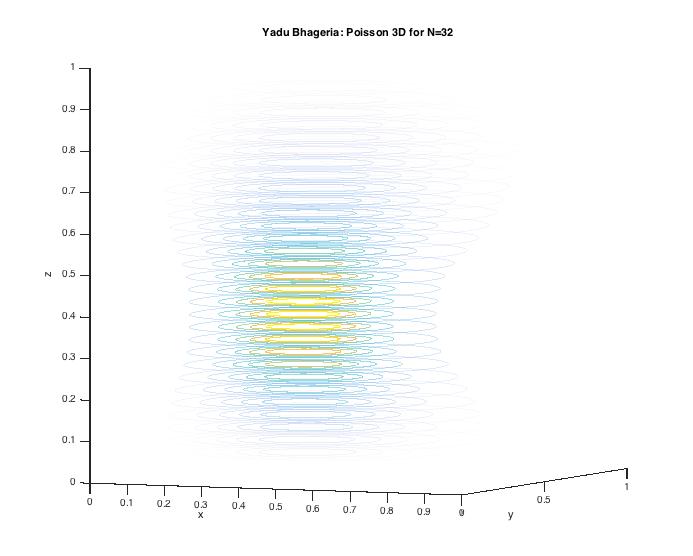
# Contour Plot

First I output my data in the format of (N-1)^2 grids one after another so I end up with a (N-1)^2 by (N-1) matrix. In MATLAB I can use the reshape function to turn this into a 3D matrix. I then use a package[[4]](#footnote-4) I found online that outputs a 3D contour plot for a given 3D vector with values for a given x,y,z axis.

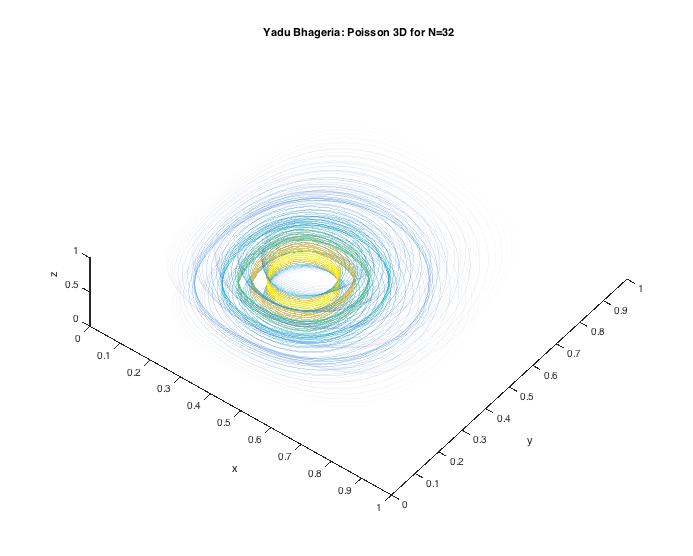
Below is a side angle (thinking of the X-Y plane as the floor) and a top view (almost making the Z axis invisible) to see the contour plots.

If I had not found this package another option would have been to produce multiple 2D contour plots for varying values of Z.

Side View:



Top View:



Appendix

# Tables

Here I have recreated some tables that I produced throughout this project.

## BGauss Poisson 1D

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ROUGH | | | | | | SMOOTH | | | | |
| 2 | N | maxval | maxval pos | cpu time | wall time | speed | maxval | maxval pos | cpu time | wall time | speed |
| 3 | 8 | 5.781250 | 3 | 0.0000060 | 0.0000020 | 0.0270000 | 4.062500 | 3 | 0.0000010 | 0.0000010 | 0.0540000 |
| 4 | 16 | 4.960937 | 7 | 0.0000010 | 0.0000020 | 0.0630000 | 4.062500 | 6 | 0.0000010 | 0.0000020 | 0.0630000 |
| 5 | 32 | 4.541016 | 13 | 0.0000020 | 0.0000020 | 0.1350000 | 4.101562 | 13 | 0.0000020 | 0.0000030 | 0.0900000 |
| 6 | 64 | 4.321289 | 26 | 0.0000030 | 0.0000040 | 0.1395000 | 4.101562 | 26 | 0.0000060 | 0.0000070 | 0.0797143 |
| 7 | 128 | 4.211426 | 52 | 0.0000060 | 0.0000060 | 0.1890000 | 4.101562 | 52 | 0.0000060 | 0.0000060 | 0.1890000 |
| 8 | 256 | 4.156494 | 104 | 0.0000140 | 0.0000140 | 0.1632857 | 4.101563 | 104 | 0.0000110 | 0.0000120 | 0.1905000 |
| 9 | 512 | 4.129028 | 208 | 0.0000250 | 0.0000240 | 0.1912500 | 4.101563 | 208 | 0.0000210 | 0.0000220 | 0.2086364 |
| 10 | 1024 | 4.115295 | 416 | 0.0000480 | 0.0000490 | 0.1877143 | 4.101562 | 416 | 0.0000490 | 0.0000490 | 0.1877143 |
| 11 | 2048 | 4.108429 | 832 | 0.0000960 | 0.0000950 | 0.1938316 | 4.101563 | 832 | 0.0000950 | 0.0000960 | 0.1918125 |
| 12 | 4096 | 4.104996 | 1664 | 0.0001920 | 0.0001930 | 0.1909119 | 4.101563 | 1664 | 0.0011030 | 0.0024300 | 0.0151630 |
| 13 | 8192 | 4.103279 | 3328 | 0.0003510 | 0.0003500 | 0.2106000 | 4.101563 | 3328 | 0.0002870 | 0.0002860 | 0.2577273 |
| 14 | 16384 | 4.102421 | 6656 | 0.0005050 | 0.0005060 | 0.2913795 | 4.101563 | 6656 | 0.0005130 | 0.0005130 | 0.2874035 |
| 15 | 32768 | 4.101992 | 13312 | 0.0010540 | 0.0010530 | 0.2800513 | 4.101562 | 13312 | 0.0009020 | 0.0009020 | 0.3269335 |
| 16 | 65536 | 4.101777 | 26624 | 0.0016730 | 0.0016720 | 0.3527548 | 4.101562 | 26624 | 0.0016790 | 0.0016790 | 0.3512841 |
| 17 | 131072 | 4.101670 | 53248 | 0.0032930 | 0.0038610 | 0.3055245 | 4.101562 | 53248 | 0.0030440 | 0.0030430 | 0.3876536 |
| 18 | 262144 | 4.101616 | 106496 | 0.0074800 | 0.0101530 | 0.2323725 | 4.101562 | 106496 | 0.0067240 | 0.0067230 | 0.3509264 |
| 19 | 524288 | 4.101590 | 212992 | 0.0161720 | 0.0161760 | 0.2917022 | 4.101563 | 212992 | 0.0145270 | 0.0178390 | 0.2645089 |
| 20 | 1048576 | 4.101579 | 425984 | 0.0264130 | 0.0297330 | 0.3173970 | 4.101565 | 425984 | 0.0243900 | 0.0243900 | 0.3869277 |
| 21 | 2097152 | 4.101572 | 851968 | 0.0488250 | 0.0488250 | 0.3865714 | 4.101565 | 851968 | 0.0532680 | 0.0532870 | 0.3542018 |
| 22 | 4194304 | 4.101568 | 1703936 | 0.1016840 | 0.1017160 | 0.3711188 | 4.101564 | 1703936 | 0.1054220 | 0.1054570 | 0.3579537 |
| 23 | 8388608 | 4.101563 | 3407872 | 0.2005660 | 0.2005640 | 0.3764258 | 4.101561 | 3407872 | 0.2043200 | 0.2043180 | 0.3695096 |
| 24 | 16777216 | 4.101517 | 6815714 | 0.4773880 | 0.4778520 | 0.3159868 | 4.101516 | 6815714 | 0.4589440 | 0.4592740 | 0.3287687 |
| 25 | 33554432 | 4.100841 | 13631372 | 0.9202720 | 0.9229430 | 0.3272032 | 4.100841 | 13631372 | 0.8230980 | 0.8231100 | 0.3668888 |
| 26 | 67108864 | 4.090507 | 27263101 | 1.6467740 | 1.6467700 | 0.3667663 | 4.090507 | 27263101 | 1.8798350 | 1.8798110 | 0.3212981 |
| 27 | 134217728 | 3.923872 | 53929750 | 3.2325200 | 3.2327580 | 0.3736622 | 3.923872 | 53929750 | 3.3144030 | 3.3143950 | 0.3644585 |
| 28 | 268435456 | 2.303370 | 99977615 | 23.9114370 | 28.8705990 | 0.0836810 | 2.303370 | 99977615 | 15.6768500 | 16.9601000 | 0.1424472 |
| 29 | 536870912 | 0.965351 | 201295359 | 63.9122660 | 85.7657210 | 0.0563376 | 0.965351 | 201295359 | 67.8584030 | 88.7376580 | 0.0544508 |

# Graphs

# Representative Code

## Gauss Function

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <omp.h>*

*/\* Yadu Bhageria, 00733164, M3SC \*/*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();**

**void** print\_vector**(double** **\*,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**void** free\_matrix**(double** **\*\*);**

*/\* -Functions-implemented-in-current-file------------------------------------- \*/*

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***Gauss**(double** **\*\***A**,** **double** **\***y**,** **int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,** singular **=** **0;**

**unsigned** **long** **long** **int** count **=** **0;**

**double** **\***x **=** allocate\_zero\_vector**(**N**);**

*/\**

*Decompose A along with y*

*\*/*

*#pragma omp parallel private(i)*

**{**

**for** **(**i**=1;** i**<**N **&&** singular**!=1;** i**++){**

**if** **(**A**[**i**][**i**]==0){**

printf**(**"ERROR| Zero on the diagonal of the matrix during Guassian Elimiation. Considering it as singular, exitting Guass(), and returning a zero vector\n"**);**

singular **=** **1;**

**}**

*#pragma omp for*

**for** **(**j**=**i**+1;** j**<**N**+1;** j**++){**

**if** **(**A**[**j**][**i**]!=0)**

**{**

**double** ratio **=** A**[**j**][**i**]/**A**[**i**][**i**];**

**for** **(int** k**=1+**i**;** k**<**N**+1;** k**++){**

A**[**j**][**k**]** **-=** ratio**\***A**[**i**][**k**];**

count **+=** **2;**

**}**

y**[**j**]** **-=** ratio**\***y**[**i**];**

count **+=** **3;**

**}**

**}**

**}**

**}**

*/\* Check if the matrix was considered singular and if so then return 0 \*/*

**if** **(**singular**==1){**

**return** x**;**

**}**

*/\**

*Substitute back in to get x*

*\*/*

x**[**N**]** **=** y**[**N**]/**A**[**N**][**N**];**

count**++;**

**for** **(**i**=**N**-1;** i**>0;** i**--){**

x**[**i**]** **+=** y**[**i**];**

**for** **(**j**=**N**;** j**>**i**;** j**--){**

x**[**i**]** **-=** A**[**i**][**j**]\***x**[**j**];**

count **+=** **2;**

**}**

x**[**i**]** **/=** A**[**i**][**i**];**

count **+=2;**

**}**

x**[0]** **=** **(double)** count**;**

**return** x**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

## BGauss Function

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <omp.h>*

*/\* Yadu Bhageria, 00733164, M3SC \*/*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();**

**double** maxval\_vec**(double** **\*,** **int);**

**void** print\_vector**(double** **\*,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

*/\* -Functions-implemented-in-current-file------------------------------------- \*/*

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***BGauss**(double** **\*\***A**,** **double** **\***y**,** **int** N**,** **int** B**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,**singular **=** **0;**

**unsigned** **long** **long** **int** count **=** **0;**

**double** **\***x **=** allocate\_zero\_vector**(**N**);**

*/\**

*Decompose A along wtih y*

*\*/*

*#pragma omp parallel private(i) if(B>128)*

**{**

**for** **(**i**=1;** i**<**N **&&** singular**!=1;** i**++){**

**if** **(**A**[**i**][**B**+1]==0.0){**

printf**(**"ERROR| Middle Band contains zero value. Condering matrix as singular, exitting BGuass(), and returning a zero vector.\n"**);**

singular **=** **1;**

**}**

**else{**

*#pragma omp for*

**for** **(**j**=1;** j**<**B**+1;** j**++){**

**if** **(**j**<**N**+1-**i**){**

**double** ratio **=** A**[**i**+**j**][**B**+1-**j**]/**A**[**i**][**B**+1];**

**for** **(int** k**=**B**+2-**j**;** k**<2\***B**+2-**j**;** k**++){**

A**[**i**+**j**][**k**]** **-=** ratio**\***A**[**i**][**j**+**k**];**

count **+=** **2;**

**}**

y**[**i**+**j**]** **-=** ratio**\***y**[**i**];**

count **+=** **3;**

**}**

**}**

**}**

**}**

**}**

*/\* Check if the matrix was considered singular and if so then return 0 \*/*

**if** **(**singular**==1){**

**return** x**;**

**}**

*/\**

*Substitute back in to get x*

*\*/*

x**[**N**]** **=** y**[**N**]/**A**[**N**][**B**+1];**

**for** **(**i**=**N**-1;** i**>0;** i**--){**

x**[**i**]** **+=** y**[**i**];**

**for** **(**j**=1;** j**<**B**+1;** j**++){**

**if** **(**j**<**N**-**i**+1){**

x**[**i**]** **-=** A**[**i**][**B**+1+**j**]\***x**[**i**+**j**];**

count **+=** **2;**

**}**

**}**

x**[**i**]** **/=** A**[**i**][**B**+1];**

count **+=** **2;**

**}**

x**[0]** **=** **(double)** count**;**

**return(**x**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

## Required Matrix Functions for all Functions

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <stdbool.h>*

*/\* --------------------------------------------------------------------------- \*/*

**void** print\_statements**(){**

*/\* Bhageria, Yadu, 00733164, M3SC \*/*

printf**(** " Name: Bhageria, Yadu"**);**

printf**(**"\n CID: 00733164"**);**

printf**(**"\n Course Code: 00733164, M3SC"**);**

printf**(**"\nEmail Address: yrb13@ic.ac.uk"**);**

printf**(**"\n Time: %s "**,**\_\_TIME\_\_**);**

printf**(**"\n Date: %s "**,**\_\_DATE\_\_**);**

printf**(**"\n \n"**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** print\_matrix**(double** **\*\***A**,** **int** N**,** **int** M**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

*/\**

*Extremely useful for debugging*

*\*/*

**for(int** i**=1;** i**<**N**+1;** i**++){**

**for(int** j**=1;** j**<**M**+1;** j**++){**

printf**(**"%3.1g "**,** A**[**i**][**j**]);**

**}**

printf**(**"\n"**);**

**}**

printf**(**"\n"**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** print\_vector**(double** **\***x**,** **int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**for** **(int** i**=1;** i**<**N**+1;** i**++){**

printf**(**"%3d | %12.4g\n"**,**i**,** x**[**i**]);**

**}**

printf**(**"\n"**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** print\_2dvector**(double** **\***x**,** **int** N**,** **int** M**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**for** **(int** j**=1;** j**<**M**+1;** j**++){**

**for** **(int** i**=1;** i**<**N**+1;** i**++){**

printf**(**"%6.3g, "**,** x**[(**N**)\*(**i**-1)+**j**]);**

**}**

printf**(**"\n"**);**

**}**

printf**(**"\n"**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\*\***allocate\_matrix**(int** N**,** **int** M**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

*//The safe approach; better sizes over 2^27 in size*

**double** **\*\*** A**;**

A**=(double** **\*\*)** malloc**((**N**+1)\*sizeof(double** **\*));**

**for(int** i**=1;** i**<**N**+1;** i**++){**

A**[**i**]=(double** **\*)** calloc**((**M**+1),sizeof(double));**

**}**

**return** A**;**

*/\**

*//The cautious approach:*

*double \*\*A; int i;*

*A = (double \*\*)malloc((N+1)\*sizeof(double \*));*

*A[0] = (double \*)calloc((N\*M+1),sizeof(double));*

*A[1] = A[0];*

*for (i=2; i<=N; i++) A[i] = A[i-1]+M;*

*return A;*

*\*/*

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***allocate\_zero\_vector**(int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**double** **\***X**;**

X **=** **(double** **\*)** calloc**((**N**+1),sizeof(double));**

**return** X**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**int** maxvalpos\_vec**(double** **\*** X**,** **int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**double** maxvalue**=**X**[1];**

**int** xpos **=** **1;**

**for(int** i**=2;** i**<**N**+1;** i**++){**

**if** **(**X**[**i**]** **>** maxvalue**){**

maxvalue **=** X**[**i**];**

xpos **=** i**;**

**}**

**}**

**return** xpos**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** free\_matrix**(double** **\*\***A**,** **int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

*//for the safe approach*

**for** **(int** i**=**N**;** i**>0;** i**--)** free**(**A**[**i**]);**

*/\**

*//for the cautious approach*

*free(A[0]); free(A);*

*\*/*

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** multiply\_vec**(double** **\***X**,** **int** N**,** **double** factor**){**

**int** i**;**

**for** **(**i**=1;** i**<**N**+1;** i**++){**

X**[**i**]** **\*=** factor**;**

**}**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***make\_Yvec1D**(int** N**,** **bool** smooth**,** **double** delta2**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**double** **\***Y **=** allocate\_zero\_vector**(**N**-1);**

**for** **(int** i**=1;** i**<**N**;** i**++){**

**if** **(** **(double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

**if** **(**smooth **==** true**){**

Y**[**i**]** **=** **-40.0;**

**}** **else{**

Y**[**i**]** **=** **-80.0;**

**}**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

Y**[**i**]** **=** **-80.0;**

**}** **else{**Y**[**i**]** **=** **0.0;}**

Y**[**i**]** **\*=** delta2**;**

**}**

**return** Y**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\*\***make\_AGauss1D**(int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**double** **\*\***A **=** allocate\_matrix**(**N**-1,**N**-1);**

**for** **(int** i**=1;** i**<=**N**-1;** i**++){**

A**[**i**][**i**]** **=** **-2.0;**

**if** **(**i**>1){**

A**[**i**][**i**-1]** **=** **1.0;**

**}**

**if** **(**i**<**N**-1){**

A**[**i**][**i**+1]** **=** **1.0;**

**}**

**}**

**return** A**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\*\***make\_ABGauss1D**(int** N**,** **int** B**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**double** **\*\***A **=** allocate\_matrix**(**N**-1,2\***B**+1);**

**for** **(int** i**=1;** i**<=**N**-1;** i**++){**

A**[**i**][**B**+1]** **=** **-2.0;**

A**[**i**][**B**+2]** **=** A**[**i**][**B**]** **=** **1.0;**

**}**

**return** A**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***make\_Yvec2D**(int** N**,** **bool** smooth**,** **double** delta**){**

**int** i**,**j**;**

**double** **\***F **=** allocate\_zero\_vector**(** **(**N**-1)\*(**N**-1)** **);**

**for** **(**j**=1;** j**<**N**;** j**++){**

**if** **((double)**j**/**N **==** **0.5** **||** **(double)**j**/**N **==** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

**if** **(**smooth **==** true**){**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-25.0\***delta**;**

**}** **else{**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

**if** **(**smooth **==** true**){**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-50.0\***delta**;**

**}** **else{**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}**

**}**

**}** **else** **if** **((double)**j**/**N **<** **0.5** **&&** **(double)**j**/**N **>** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

**if** **(**smooth **==** true**){**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-50.0\***delta**;**

**}** **else** **{**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

F**[(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}**

**}**

**}**

**return** F**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** contour\_print**(double** **\***x**,** **int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**;**

**int** nby32 **=** N**/32;**

**for** **(**i**=0;** i**<=32;** i**++){**

printf**(**" 0.0,"**);**

**}**

printf**(**"\n"**);**

**for** **(**i**=1;** i**<32;** i**++){**

printf**(**" 0.0,"**);**

**for** **(**j**=1;** j**<32;** j**++){**

printf**(**"%8.5f,"**,** x**[(**N**-1)\*(**i**\***nby32**-1)+**j**\***nby32**]);**

**}**

printf**(**" 0.0,\n"**);**

**}**

printf**(**"\n"**);**

**for** **(**i**=0;** i**<=32;** i**++){**

printf**(**" 0.0,"**);**

**}**

**}**

*/\* --------------------------------------------------------------------------- \*/*

## Poisson 1D

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*#include <stdbool.h>*

*#include <sys/time.h>*

**void** solve\_1d\_gauss**(int,** **bool);**

**void** solve\_1d\_bgauss**(int,** **bool);**

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**double** **\*\***make\_AGauss1D**(int);**

**double** **\*\***make\_ABGauss1D**(int,** **int);**

**double** **\***make\_Yvec1D**(int,** **bool,** **double);**

**void** print\_vector**(double** **\*,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**void** multiply\_vec**(double** **\*,** **int,** **double);**

**int** maxvalpos\_vec**(double** **\*,** **int);**

**void** free\_matrix**(double** **\*\*,** **int);**

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**i1**,**i2**,**N**,**gauss**;**

print\_statements**();**

printf**(**"Enter 0 for Gauss and anything else for BGuasss: "**);**

scanf**(**"%d"**,&**gauss**);**

printf**(**"\nEnter range of exponents of 2 to run for (i1 i2): "**);**

scanf**(**"%d %d"**,** **&**i1**,** **&**i2**);**

printf**(**"\nSolving using %s function\n"**,** gauss**==0** **?** "Gauss" **:** "BGauss"**);**

printf**(**" | ROUGH | SMOOTH\n"**);**

printf**(**" N| maxval, maxval pos, cpu time, wall time, speed| maxval, maxval pos, cpu time, wall time, speed|"**);**

**for** **(**i**=**i1**;** i**<=**i2**;** i**++){**

N **=** pow**(2,**i**);**

**if** **(**gauss **==** **0){**

printf**(**"\n%12d|"**,** N**);**

solve\_1d\_gauss**(**N**,**false**);**

solve\_1d\_gauss**(**N**,**true**);**

**}** **else{**

printf**(**"\n%12d,"**,** N**);**

solve\_1d\_bgauss**(**N**,**false**);**

solve\_1d\_bgauss**(**N**,**true**);**

**}**

**}**

printf**(**"\n"**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_1d\_bgauss**(int** N**,** **bool** smooth**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** B**=1;**

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***Y **=** make\_Yvec1D**(**N**,**smooth**,**delta**);**

**double** **\*\*** A\_banded **=** make\_ABGauss1D**(**N**,**B**);**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\*** x **=** BGauss**(**A\_banded**,**Y**,**N**-1,**B**);**

end **=** clock**();**

gettimeofday**(&**wt\_end**,**NULL**);**

free\_matrix**(**A\_banded**,** N**-1);**

free**(**Y**);**

**double** timetaken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**x**,** N**-1);**

*//multiply\_vec(x,N-1,delta);*

**double** speed **=** **1e-9** **\*** x**[0]** **/** wt\_timetaken**;**

printf**(**"%10.6f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** x**[**maxval\_position**]\***delta**,** **(double)**maxval\_position**/**N**,** timetaken**,** wt\_timetaken**,** speed**);**

free**(**x**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_1d\_gauss**(int** N**,** **bool** smooth**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***Y **=** make\_Yvec1D**(**N**,**smooth**,**delta**);**

**double** **\*\*** A **=** make\_AGauss1D**(**N**);**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\***x **=** Gauss**(**A**,**Y**,**N**-1);**

end **=** clock**();**

gettimeofday**(&**wt\_end**,**NULL**);**

free\_matrix**(**A**,** N**-1);**

free**(**Y**);**

**double** timetaken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**x**,** N**-1);**

*//multiply\_vec(x,N,delta);*

**double** speed **=** **1e-9** **\*** x**[0]** **/** wt\_timetaken**;**

*//printf("\n%f\n", x[0]);*

printf**(**"%10.6f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** x**[**maxval\_position**]\***delta**,** **(double)**maxval\_position**/**N**,** timetaken**,** wt\_timetaken**,** speed**);**

free**(**x**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

## Poisson 2D

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*#include <stdbool.h>*

*#include <sys/time.h>*

**double** **\***make\_Yvec2D**(int,** **bool,** **double);**

**void** solve\_2d\_gauss**(int,** **bool);**

**void** solve\_2d\_bgauss**(int,** **bool);**

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**void** print\_2dvector**(double** **\*,** **int,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**void** contour\_print**(double** **\*,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,int);**

**int** maxvalpos\_vec**(double** **\*,** **int);**

**void** free\_matrix**(double** **\*\*,** **int);**

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**N**,**gauss**;**

print\_statements**();**

printf**(**"Enter 0 for Gauss and anything else for BGauss: "**);**

scanf**(**"%d"**,&**gauss**);**

printf**(**"\nSolving using %s function\n"**,** gauss**==0** **?** "Gauss" **:** "BGauss"**);**

printf**(**" N| maxval, maxval pos x, maxval pos y, cpu time, wall time, speed| maxval, maxval pos x, maxval pos y, cpu time, wall time, speed|"**);**

**double** n\_val**[]** **=** **{8,12,16,20,24,32,40,48,64,80,96};**

**for** **(**i**=0;** i**<11;** i**++){**

N **=** n\_val**[**i**];**

**if** **(**gauss **==** **0){**

printf**(**"\n%12d|"**,** N**);**

solve\_2d\_gauss**(**N**,**false**);**

solve\_2d\_gauss**(**N**,**true**);**

**}** **else** **{**

printf**(**"\n%12d|"**,** N**);**

solve\_2d\_bgauss**(**N**,**false**);**

solve\_2d\_bgauss**(**N**,**true**);**

**}**

**}**

**for** **(**i**=1;** i**<=15;** i**++){**

N **=** **96+32\***i**;**

**if** **(**gauss **==** **0){**

printf**(**"\n%12d|"**,** N**);**

solve\_2d\_gauss**(**N**,**false**);**

solve\_2d\_gauss**(**N**,**true**);**

**}** **else** **{**

printf**(**"\n%12d|"**,** N**);**

solve\_2d\_bgauss**(**N**,**false**);**

solve\_2d\_bgauss**(**N**,**true**);**

**}**

**}**

*//solve\_2d\_bgauss(512,true);*

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_2d\_bgauss**(int** N**,** **bool** smooth**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,** B**=**N**-1,** grid\_size **=** **(**N**-1)\*(**N**-1);**

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***F **=** make\_Yvec2D**(**N**,**smooth**,**delta**\***delta**);**

**double** **\*\***A\_banded **=** allocate\_matrix**(**grid\_size**,2\***B**+1);**

**for** **(**i**=1;** i**<**N**;** i**++){**

**for** **(**j**=1;** j**<**N**;** j**++){**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][**B**+1]** **=** **-4.0;**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][1]** **=** **1.0;**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][2\***B**+1]** **=** **1.0;**

**if** **(**j**==1){**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][**B**+2]** **=** **1.0;**

**}** **else** **if** **(**j**==**N**-1){**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][**B**]** **=** **1.0;**

**}** **else** **{**

A\_banded**[(**N**-1)\*(**i**-1)+**j**][**B**+2]** **=** A\_banded**[(**N**-1)\*(**i**-1)+**j**][**B**]** **=** **1.0;**

**}**

**}**

**}**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\*** v **=** BGauss**(**A\_banded**,**F**,**grid\_size**,**B**);**

end **=** clock**();**

gettimeofday**(&**wt\_end**,**NULL**);**

free\_matrix**(**A\_banded**,** grid\_size**);**

free**(**F**);**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**v**,** grid\_size**);**

**int** y\_maxval **=** maxval\_position **%** **(**N**-1);**

**int** x\_maxval **=** **(**maxval\_position**-**y\_maxval**)/(**N**-1)** **+** **1;**

*//contour\_print(v,N);*

**double** speed **=** **1e-9** **\*** v**[0]** **/** wt\_timetaken**;**

printf**(**"%10.6f, %12.10f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** v**[**maxval\_position**],** **(double)**x\_maxval**/**N**,** **(double)**y\_maxval**/**N**,** time\_taken**,** wt\_timetaken**,** speed**);**

free**(**v**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_2d\_gauss**(int** N**,** **bool** smooth**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,**grid\_size **=** **(**N**-1)\*(**N**-1);**

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***F **=** make\_Yvec2D**(**N**,**smooth**,**delta**\***delta**);**

**double** **\*\***A **=** allocate\_matrix**(**grid\_size**,**grid\_size**);**

**for** **(**i**=1;** i**<**N**;** i**++){**

**for** **(**j**=1;** j**<**N**;** j**++){**

A**[(**N**-1)\*(**i**-1)+**j**][(**N**-1)\*(**i**-1)+**j**]** **=** **-4.0;**

**if** **((**N**-1)\*(**i**-1)+**j **<** grid\_size**){**

**if** **(**j**!=**N**-1){**

A**[(**N**-1)\*(**i**-1)+**j**][(**N**-1)\*(**i**-1)+**j**+1]** **=** **1.0;**

**}**

**}**

**if** **((**N**-1)\*(**i**-1)+**j **>** **1){**

**if** **(**j**!=1){**

A**[(**N**-1)\*(**i**-1)+**j**][(**N**-1)\*(**i**-1)+**j**-1]** **=** **1.0;**

**}**

**}**

**if** **((**N**-1)\*(**i**-1)+**j **>** N**-1){**

A**[(**N**-1)\*(**i**-1)+**j**][(**N**-1)\*(**i**-2)+**j**]** **=** **1.0;**

**}**

**if** **((**N**-1)\*(**i**-1)+**j **<=** grid\_size **-** **(**N**-1)){**

A**[(**N**-1)\*(**i**-1)+**j**][(**N**-1)\*(**i**)+**j**]** **=** **1.0;**

**}**

**}**

**}**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\*** v **=** Gauss**(**A**,**F**,**grid\_size**);**

gettimeofday**(&**wt\_end**,**NULL**);**

end **=** clock**();**

free\_matrix**(**A**,** grid\_size**);**

free**(**F**);**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**v**,** grid\_size**);**

**int** y\_maxval **=** maxval\_position **%** **(**N**-1);**

**int** x\_maxval **=** **(**maxval\_position**-**y\_maxval**)/(**N**-1)** **+** **1;**

**double** speed **=** **1e-9** **\*** v**[0]** **/** wt\_timetaken**;**

printf**(**"%10.6f, %12.10f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** v**[**maxval\_position**],** **(double)**x\_maxval**/**N**,** **(double)**y\_maxval**/**N**,** time\_taken**,** wt\_timetaken**,** speed**);**

free**(**v**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

## Poisson 3D

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*#include <stdbool.h>*

*#include <sys/time.h>*

**double** **\***make\_Yvec3D**(int,** **double);**

**void** solve\_3d\_gauss**(int);**

**void** solve\_3d\_bgauss**(int);**

**void** contour\_print3D**(double** **\*,** **int);**

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**void** print\_2dvector**(double** **\*,** **int,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**void** contour\_print**(double** **\*,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,int);**

**int** maxvalpos\_vec**(double** **\*,** **int);**

**void** free\_matrix**(double** **\*\*,** **int);**

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**N**,**gauss**;**

print\_statements**();**

*/\**

*Values of 2^n*

*2^30 = 1073741824*

*2^29 = 536870912*

*2^28 = 268435456*

*2^27 = 134217728*

*2^26 = 67108864*

*2^25 = 33554432*

*2^24 = 16777216*

*2^23 = 8388608*

*2^20 = 1048576*

*\*/*

printf**(**"Enter 0 for Gauss and anything else for BGauss: "**);**

scanf**(**"%d"**,&**gauss**);**

printf**(**"\nSolving using %s function\n"**,** gauss**==0** **?** "Gauss" **:** "BGauss"**);**

printf**(**" N| maxval, maxval pos x, maxval pos y, maxval pos z, cpu time, wall time, speed"**);**

**double** n\_val**[]** **=** **{5,6,8,10,12,16,20,24,28,32,36,40,48,64,80,96,128};**

**for** **(**i**=0;** i**<17;** i**++){**

N **=** n\_val**[**i**];**

**if** **(**gauss **==** **0){**

printf**(**"\n%12d|"**,** N**);**

solve\_3d\_gauss**(**N**);**

**}** **else** **{**

printf**(**"\n%12d|"**,** N**);**

solve\_3d\_bgauss**(**N**);**

**}**

**}**

printf**(**"\n"**);**

*//solve\_3d\_bgauss(32);*

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_3d\_bgauss**(int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,**k**,** B**=(**N**-1)\*(**N**-1),** grid\_size **=** **(**N**-1)\*(**N**-1);**

**int** cube\_size **=** **(**N**-1)\***grid\_size**;**

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***F **=** make\_Yvec3D**(**N**,**delta**\***delta**);**

**double** **\*\***A\_banded **=** allocate\_matrix**(**cube\_size**,2\***B**+1);**

**for** **(**k**=1;** k**<**N**;** k**++){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**for** **(**j**=1;** j**<**N**;** j**++){**

**int** current\_row **=** grid\_size**\*(**k**-1)+(**N**-1)\*(**i**-1)+**j**;**

A\_banded**[**current\_row**][**B**+1]** **=** **-6.0;**

A\_banded**[**current\_row**][1]** **=** **1.0;**

A\_banded**[**current\_row**][2\***B**+1]** **=** **1.0;**

**if** **(**i**!=1){**

A\_banded**[**current\_row**][**B**+1-(**N**-1)]** **=** **1.0;**

**}**

**if** **(**i**!=(**N**-1)){**

A\_banded**[**current\_row**][**B**+1+**N**-1]** **=** **1.0;**

**}**

**if** **(**j**==1){**

A\_banded**[**current\_row**][**B**+2]** **=** **1.0;**

**}** **else** **if** **(**j**==**N**-1){**

A\_banded**[**current\_row**][**B**]** **=** **1.0;**

**}** **else** **{**

A\_banded**[**current\_row**][**B**+2]** **=** A\_banded**[**current\_row**][**B**]** **=** **1.0;**

**}**

**}**

**}**

**}**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\*** v **=** BGauss**(**A\_banded**,**F**,**cube\_size**,**B**);**

end **=** clock**();**

gettimeofday**(&**wt\_end**,**NULL**);**

free\_matrix**(**A\_banded**,** cube\_size**);**

free**(**F**);**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**v**,** cube\_size**);**

**int** z\_maxval **=** maxval\_position **%** **(**N**-1);**

**int** y\_maxval **=** **(**maxval\_position**%(**N**-1)\*(**N**-1)-**z\_maxval**)/(**N**-1)** **+** **1;**

**int** x\_maxval **=** **(**maxval\_position**-(**N**-1)\*(**y\_maxval**-1)-**z\_maxval**)/(**N**-1)/(**N**-1)+** **1;**

*/\**

*printf("\n%d\n", maxval\_position);*

*printf("\n%d\n", z\_maxval);*

*printf("\n%d\n", y\_maxval);*

*printf("\n%d\n", x\_maxval);*

*\*/*

*//contour\_print3D(v,N);*

**double** speed **=** **1e-9** **\*** v**[0]** **/** wt\_timetaken**;**

printf**(**"%10.6f, %12.10f, %12.10f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** v**[**maxval\_position**],** **(double)**x\_maxval**/**N**,** **(double)**y\_maxval**/**N**,** **(double)**z\_maxval**/**N**,** time\_taken**,** wt\_timetaken**,** speed**);**

free**(**v**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** solve\_3d\_gauss**(int** N**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,**k**,**grid\_size **=** **(**N**-1)\*(**N**-1);**

**int** cube\_size **=** **(**N**-1)\***grid\_size**;**

**clock\_t** start**,** end**;**

**struct** timeval wt\_start**,** wt\_end**;**

**double** delta **=** **1.0/((double)**N**);**

**double** **\***F **=** make\_Yvec3D**(**N**,**delta**\***delta**);**

**double** **\*\***A **=** allocate\_matrix**(**cube\_size**,**cube\_size**);**

**for** **(**k**=1;** k**<**N**;** k**++){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**for** **(**j**=1;** j**<**N**;** j**++){**

**int** current\_row **=** grid\_size**\*(**k**-1)+(**N**-1)\*(**i**-1)+**j**;**

A**[**current\_row**][**current\_row**]** **=** **-6.0;**

**if** **(**j**!=1){**

A**[**current\_row**][**current\_row**-1]** **=** **1.0;**

**}**

**if** **(**j**!=(**N**-1)){**

A**[**current\_row**][**current\_row**+1]** **=** **1.0;**

**}**

**if** **(**i**!=1){**

A**[**current\_row**][**current\_row**-(**N**-1)]** **=** **1.0;**

**}**

**if** **(**i**!=(**N**-1)){**

A**[**current\_row**][**current\_row**+(**N**-1)]** **=** **1.0;**

**}**

**if** **(**k**!=1){**

A**[**current\_row**][**current\_row**-(**N**-1)\*(**N**-1)]** **=** **1.0;**

**}**

**if** **(**k**!=**N**-1){**

A**[**current\_row**][**current\_row**+(**N**-1)\*(**N**-1)]** **=** **1.0;**

**}**

**}**

**}**

**}**

start **=** clock**();**

gettimeofday**(&**wt\_start**,**NULL**);**

**double** **\*** v **=** Gauss**(**A**,**F**,**cube\_size**);**

gettimeofday**(&**wt\_end**,**NULL**);**

end **=** clock**();**

free\_matrix**(**A**,** cube\_size**);**

free**(**F**);**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

**double** wt\_timetaken **=** **(double)(**wt\_end**.**tv\_sec **-** wt\_start**.**tv\_sec **+** **(**wt\_end**.**tv\_usec **-** wt\_start**.**tv\_usec**)/1000000.0);**

**int** maxval\_position **=** maxvalpos\_vec**(**v**,** cube\_size**);**

**int** z\_maxval **=** maxval\_position **%** **(**N**-1);**

**int** y\_maxval **=** **(**maxval\_position**%(**N**-1)\*(**N**-1)-**z\_maxval**)/(**N**-1)** **+** **1;**

**int** x\_maxval **=** **(**maxval\_position**-(**N**-1)\*(**y\_maxval**-1)-**z\_maxval**)/(**N**-1)/(**N**-1)+** **1;**

*/\**

*printf("\n%d\n", maxval\_position);*

*printf("\n%d\n", z\_maxval);*

*printf("\n%d\n", y\_maxval);*

*printf("\n%d\n", x\_maxval);*

*\*/*

**double** speed **=** **1e-9** **\*** v**[0]** **/** wt\_timetaken**;**

printf**(**"%10.6f, %12.10f, %12.10f, %12.10f, %11.8f, %11.8f, %11.8f|"**,** v**[**maxval\_position**],** **(double)**x\_maxval**/**N**,** **(double)**y\_maxval**/**N**,** **(double)**z\_maxval**/**N**,** time\_taken**,** wt\_timetaken**,** speed**);**

free**(**v**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**double** **\***make\_Yvec3D**(int** N**,** **double** delta**){**

*/\* Yadu Bhageria, 00733164, M3SC \*/*

**int** i**,**j**,**k**;**

**double** **\***F **=** allocate\_zero\_vector**(** **(**N**-1)\*(**N**-1)\*(**N**-1)** **);**

**for** **(**k**=1;** k**<**N**;** k**++){**

**int** current\_grid **=** **(**N**-1)\*(**N**-1)\*(**k**-1);**

**if** **((double)**k**/**N **==** **0.5** **||** **(double)**k**/**N **==** **0.25){**

**for** **(**j**=1;** j**<**N**;** j**++){**

**if** **((double)**j**/**N **==** **0.5** **||** **(double)**j**/**N **==** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-25.0\***delta**;**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-50.0\***delta**;**

**}**

**}**

**}** **else** **if** **((double)**j**/**N **<** **0.5** **&&** **(double)**j**/**N **>** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-50.0\***delta**;**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}**

**}**

**}**

*/\* --- \*/*

**}** **else** **if** **((double)**k**/**N **<** **0.5** **&&** **(double)**k**/**N **>** **0.25){**

**for** **(**j**=1;** j**<**N**;** j**++){**

**if** **((double)**j**/**N **==** **0.5** **||** **(double)**j**/**N **==** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-50.0\***delta**;**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}**

**}**

**}** **else** **if** **((double)**j**/**N **<** **0.5** **&&** **(double)**j**/**N **>** **0.25){**

**for** **(**i**=1;** i**<**N**;** i**++){**

**if** **((double)**i**/**N **==** **0.5** **||** **(double)**i**/**N **==** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-100.0\***delta**;**

**}** **else** **if** **((double)**i**/**N **<=** **0.5** **&&** **(double)**i**/**N **>=** **0.25){**

F**[**current\_grid**+(**N**-1)\*(**j**-1)+**i**]** **=** **-200.0\***delta**;**

**}**

**}**

**}**

**}**

**}**

**}**

**return** F**;**

**}**

*/\* --------------------------------------------------------------------------- \*/*

**void** contour\_print3D**(double** **\***x**,** **int** N**){**

**int** i**,**j**;**

**for** **(**j**=0;** j**<=32;** j**++){**

**for** **(**i**=0;** i**<=32;** i**++){**

printf**(**" 0.0,"**);**

**}**

printf**(**"\n"**);**

**}**

**for** **(**i**=1;** i**<32;** i**++){**

contour\_print**(**x **+** **(**i**-1)\*(**N**-1)\*(**N**-1),** N**);**

printf**(**"\n"**);**

**}**

printf**(**"\n"**);**

**for** **(**j**=0;** j**<=32;** j**++){**

**for** **(**i**=0;** i**<=32;** i**++){**

printf**(**" 0.0,"**);**

**}**

printf**(**"\n"**);**

**}**

**}**

*/\* --------------------------------------------------------------------------- \*/*

### Programs for Solving the given Matrices

#### Case 1 Gauss:

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,** **int);**

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

**int** N **=** **3;**

**int** i**,**j**;**

**clock\_t** start**,** end**;**

print\_statements**();**

**double** **\***Y **=** allocate\_zero\_vector**(**N**);**

Y**[1]** **=** Y**[2]** **=** Y**[3]** **=** **2;**

**double** **\*\***A **=** allocate\_matrix**(**N**,**N**);**

**for** **(**i**=1;** i**<=**N**;** i**++){** *//print to see what it is*

**for** **(**j**=1;**j**<**N**+1;**j**++){**

A**[**i**][**j**]=0.0;**

**}**

**}**

**for** **(**i**=1;** i**<=**N**;** i**++){**

A**[**i**][**i**]=-2.0;**

**if** **(**i**!=**N**){**

A**[**i**][**i**+1]=1.0;**

A**[**i**+1][**i**]=1.0;**

**}**

**}**

start **=** clock**();**

**double** **\*** x **=** Gauss**(**A**,**Y**,**N**);**

end **=** clock**();**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

printf**(**"Time taken: %f\n"**,** time\_taken**);**

printf**(**"The returned vector is\n"**);**

print\_vector**(**x**,**N**);**

**}**

#### Case 2 Gauss

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,** **int);**

**double** **\***Gauss**(double** **\*\*,** **double** **\*,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

**int** N **=** **8;**

**int** i**,**j**;**

**clock\_t** start**,** end**;**

print\_statements**();**

**double** **\***Y **=** allocate\_zero\_vector**(**N**);**

**for** **(**i**=1;** i**<**N**+1;** i**++){**

Y**[**i**]** **=** i**;**

**}**

**double** **\*\***A **=** allocate\_matrix**(**N**,**N**);**

**for** **(**i**=1;** i**<=**N**;** i**++){** *//print to see what it is*

**for** **(**j**=1;**j**<**N**+1;**j**++){**

A**[**i**][**j**]=0.0;**

**}**

**}**

**for** **(**i**=1;** i**<=**N**;** i**++){**

A**[**i**][**i**]** **=** **6.0;**

**if** **(**i**!=**N**){**

A**[**i**][**i**+1]** **=** **-4.0;**

A**[**i**+1][**i**]** **=** **-4.0;**

**}**

**if** **(**i**<**N**-1){**

A**[**i**][**i**+2]** **=** **1.0;**

A**[**i**+2][**i**]** **=** **1.0;**

**}**

**}**

A**[1][1]** **=** **5.0;**

A**[7][7]** **=** **7.0;**

A**[8][8]** **=** **6.0;**

A**[8][7]** **=-8.0;**

A**[8][6]** **=** **2.0;**

start **=** clock**();**

**double** **\*** x **=** Gauss**(**A**,**Y**,**N**);**

end **=** clock**();**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

printf**(**"Time taken: %f\n"**,** time\_taken**);**

printf**(**"The returned vector is\n"**);**

print\_vector**(**x**,**N**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

#### Case 1 BGauss

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,** **int);**

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

**int** N **=** **3,** B **=** **1;**

**int** i**,**j**;**

**clock\_t** start**,** end**;**

print\_statements**();**

**double** **\***Y **=** allocate\_zero\_vector**(**N**);**

Y**[1]** **=** Y**[2]** **=** Y**[3]** **=** **2;**

**double** **\*\***A **=** allocate\_matrix**(**N**,2\***B**+1);**

**for** **(**i**=1;** i**<=**N**;** i**++){** *//print to see what it is*

**for** **(**j**=1;**j**<**N**+1;**j**++){**

A**[**i**][**j**]=0.0;**

**}**

**}**

A**[3][2]** **=** A**[2][2]** **=** A**[1][2]** **=** **-2.0;**

A**[2][3]** **=** A**[2][1]** **=** A**[3][1]** **=** A**[1][3]** **=** **1;**

start **=** clock**();**

**double** **\*** x **=** BGauss**(**A**,**Y**,**N**,**B**);**

end **=** clock**();**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

printf**(**"Time taken: %f\n"**,** time\_taken**);**

printf**(**"The returned vector is\n"**);**

print\_vector**(**x**,**N**);**

**}**

*/\* --------------------------------------------------------------------------- \*/*

#### Case 2 BGauss

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <math.h>*

*#include <time.h>*

*/\* -Functions-needed-from-other-files----------------------------------------- \*/*

**void** print\_statements**();** *//contained in the gauss.c file*

**void** print\_vector**(double** **\*,** **int);**

**void** print\_matrix**(double** **\*\*,** **int,** **int);**

**double** **\***allocate\_zero\_vector**(int);**

**double** **\*\***allocate\_matrix**(int,int);**

**double** **\***BGauss**(double** **\*\*,** **double** **\*,** **int,** **int);**

*/\* --------------------------------------------------------------------------- \*/*

**int** main**(void)** **{**

**int** N **=** **8,** B **=** **2;**

**int** i**,**j**;**

**clock\_t** start**,** end**;**

print\_statements**();**

**double** **\***Y **=** allocate\_zero\_vector**(**N**);**

**for** **(**i**=1;** i**<**N**+1;** i**++){**

Y**[**i**]** **=** i**;**

**}**

**double** **\*\***A **=** allocate\_matrix**(**N**,2\***B**+1);**

**for** **(**i**=1;** i**<=**N**;** i**++){** *//print to see what it is*

**for** **(**j**=1;**j**<2\*(**B**+1);**j**++){**

A**[**i**][**j**]=0.0;**

**}**

**}**

**for** **(**i**=1;** i**<=**N**;** i**++){**

A**[**i**][**B**+1]** **=** **6.0;**

A**[**i**][**B**+2]** **=** A**[**i**][**B**]** **=** **-4.0;**

A**[**i**][**B**-1]** **=** A**[**i**][**B**+3]** **=** **1.0;**

**}**

A**[1][**B**+1]** **=** **5.0;**

A**[7][**B**+1]** **=** **7.0;**

A**[8][**B**-1]** **=** **2.0;**

A**[8][**B**]** **=** **-8.0;**

start **=** clock**();**

**double** **\*** x **=** BGauss**(**A**,**Y**,**N**,**B**);**

end **=** clock**();**

**double** time\_taken **=** **((double)**end**-**start**)/**CLOCKS\_PER\_SEC**;**

printf**(**"Time taken: %f\n"**,** time\_taken**);**

printf**(**"The returned vector is\n"**);**

print\_vector**(**x**,**N**);**

**}**

1. http://wwwf.imperial.ac.uk/~drmii/M3SC\_2016/Exercise\_2\_16.pdf [↑](#footnote-ref-1)
2. http://www.cs.loyola.edu/~jglenn/702/S2008/Projects/P3/time.html [↑](#footnote-ref-2)
3. https://www.math.ust.hk/~mawang/teaching/math532/mgtut.pdf [↑](#footnote-ref-3)
4. https://www.mathworks.com/matlabcentral/fileexchange/4663-cont3d-m/content/cont3d\_examples.m [↑](#footnote-ref-4)