ΑΝΑΦΟΡΑ ΠΡΩΤΟΥ ΜΕΡΟΥΣ ΕΡΓΑΣΙΑΣ 2021-2022 ΕΠΙΣΤΗΜΟΝΙΚΟΥ ΥΠΟΛΟΓΙΣΜΟΥ

Συγγραφέας:
Αλέξανδρος Νατσολλάρη
Α.Μ.:1057769, Ετος:6ο

Διδάσκων Προσωπικό:
ΕυστράτιοςΓαλλόπουλος,Καθηγητής
Π.Χατζηδούκας,Αναπλ.Καθηγητής
Γ.Θανάσης,Υποψήφιος διδάκτορας

ΠΕΡΙΕΧΟΜΕΝΑ

Λίστα Εικόνων

Λίστα Πινάκων

Εισαγωγικά

1.1 Στοιχεία υπολογιστικού συστήματος

1. Ημερομηνία Έναρξης:16/11/2021, Ημερομηνία Λήξης: 15/12/2021
2. Χαρακτηριστικά του υπολογιστικού συστήματος:

i)

Table 1.1: Στοιχεία για τα πειράματα

Χαρακτηριστικό	Απάντηση
Έναρξη/λήξη εργασίας	16/11/2021-15/12/2021
Model	Προσωπικός Υπολογιστής:DESKTOP-4QOHGHP[1]
O/S	Microsoft Windows 10 Education 10.0.19042 Build 19042
processor name	6- Core Intel Core i5 10400[2]
processor speed	2.90 GHz (base)
number of processors	1
total cores	6
total theads	12
FMA instruction	yes
L1 cache	128KB Instruction, 128KB Data write Back
L2 cache	(per core) 256 KB, write-back
L3 cache	(shared) 12MB , write-back
Gflops/s	423.1[3]
Memory	16GB
Memory Bandwidth	41.6 GB/s
MATLAB Version	9.8.0.1323502 (R2020a)
BLAS	Intel(R) Math Kernel Library Version 2019.0.3 Product Build 20190125 for Intel(R) 64 architecture applications, CNR branch AVX2
LAPACK	Intel(R) Math Kernel Library Version 2019.0.3 Product Build 20190125 for Intel(R) 64 architecture applications, CNR branch AVX2 Linear Algebra PACKage Version 3.7.0

[1]Είναι σταθερός υπολογιστής φτιαγμένος κομμάτι-κομμάτι και δεν υπάρχει συγκεκριμένο μοντέλο.Μητρική: Asrock Steel Legend 8460,RAM: HyperX Fury DDR4 3200 MHZ HX432C16FB3K2/(2*8),SSD: Kingston SA2000M8/500G A200 M.2,PSU: EVGA 700 BQ

[2]https://ark.intel.com/content/www/us/en/ark/products/199271/intel-core-i510400-processor-12m-cache-up-to-4-30-ghz.html

[3]https://gadgetversus.com/processor/intel-core-i5-10400-specs/

ii)Έκδοση Matlab: 9.8.0.1323502 (R2020a) και οι πληροφορίες για τις βιβλιοθήκες που είναι εγκατεστημένες φαίνονται στην εικόνα 1 παρακάτω:

MATLAB	Version 9.8	(R2020a)
Simulink	Version 10.1	(R2020a)
Bioinformatics Toolbox	Version 4.14	(R2020a)
Communications Toolbox	Version 7.3	(R2020a)
Computer Vision Toolbox	Version 9.2	(R2020a)
Control System Toolbox	Version 10.8	(R2020a)
Curve Fitting Toolbox	Version 3.5.11	(R2020a)
DSP System Toolbox	Version 9.10	(R2020a)
Data Acquisition Toolbox	Version 4.1	(R2020a)
Database Toolbox	Version 9.2.1	(R2020a)
Deep Learning Toolbox	Version 14.0	(R2020a)
Embedded Coder	Version 7.4	(R2020a)
Fixed-Point Designer	Version 7.0	(R2020a)
Fuzzy Logic Toolbox	Version 2.7	(R2020a)
GPU Coder	Version 1.5	(R2020a)
Global Optimization Toolbox	Version 4.3	(R2020a)
Image Acquisition Toolbox	Version 6.2	(R2020a)
Image Processing Toolbox	Version 11.1	(R2020a)
Instrument Control Toolbox	Version 4.2	(R2020a)
LTE Toolbox	Version 3.3	(R2020a)
MATLAB Coder	Version 5.0	(R2020a)
MATLAB Compiler	Version 8.0	(R2020a)
MATLAB Compiler SDK	Version 6.8	(R2020a)
Mapping Toolbox	Version 4.10	(R2020a)
Model Predictive Control Toolbox	Version 6.4	(R2020a)
Optimization Toolbox	Version 8.5	(R2020a)
Parallel Computing Toolbox	Version 7.2	(R2020a)
Partial Differential Equation Toolbox	Version 3.4	(R2020a)
Robust Control Toolbox	Version 6.8	(R2020a)
Signal Processing Toolbox	Version 8.4	(R2020a)
SimEvents	Version 5.8	(R2020a)
Simscape	Version 4.8	(R2020a)
Simscape Electrical	Version 7.3	(R2020a)
Simscape Multibody	Version 7.1	(R2020a)
Simulink Coder	Version 9.3	(R2020a)
Simulink Real-Time	Version 6.12	(R2020a)
Stateflow	Version 10.2	(R2020a)
Statistics and Machine Learning Toolbox	Version 11.7	(R2020a)
Symbolic Math Toolbox	Version 8.5	(R2020a)
System Identification Toolbox	Version 9.12	(R2020a)
Wavelet Toolbox	Version 5.4	(R2020a)

Figure 1.1: Εγκαστεστημένες Βιβλιοθήκες

iii) Εκτελώντας την εντολή bench της Matlab λαμβάνονται τα αποτελέσματα που φαίνονται στο σχήμα 2:

Computer Type	LU	FFT	ODE	Sparse	2-D	3-D
Windows 10, AMD Ryzen Threadipper(TM) 3970x @ 3.50 GHz	0.2142	0.2931	0.3754	0.4758	0.1914	0.1960
Debian 9(R), AMD Ryzen Threadipper 3970x @ 3.50 GHz	0.2816	0.3059	0.3449	0.4187	0.2827	0.2478
This machine	0.4623	0.3269	0.4460	0.3350	0.3635	0.4258
iMac, macOS 10.14.5, Intel Core i9 @3.6 GHz	0.3951	0.4235	0.3112	0.2790	0.7170	0.3641
Windows 10, Intel Xeon(R) W-2133 @ 3.60 GHz	0.4195	0.5219	0.4805	0.5160	0.3000	0.3175
Windows 10, AMD Ryzen(TM) 7 1700 @ 3.00 GHz	0.7770	0.6159	0.5551	0.5734	0.3117	0.3043
Windows 10, Intel Xeon CPU E5-1650 v3 @ 3.50 GHz	0.7741	0.7450	0.5382	0.7103	0.3807	0.3963
Surface Pro 3, Windows(R) 10, Intel(R) Core i5-4300U @ 1.9 GHz	1.7664	1.1723	0.6900	0.6872	0.9974	0.8921
MacBook Pro, macOS 10.15.2, Intel Core i5 @ 2.6 GHz	1.5141	1.1708	0.5463	0.5959	2.0298	1.4466

Figure 1.2: Αποτελέσματα εκτέλεσης εντολής Bench

Αραιές αναπαραστάσεις και κατασκευές μητρώων

2.1 Κατασκευή συνάρτησης sp_mat2latex

Ο κώδικας βρίσκεται στο αρχείο sp_mat2latex.m, καθώς και στο τέλος της ενότητας αυτής. Για να τρέξει η συνάρτηση χρειάζεται και ένα αρχείο txt με όνομα sp_mat2latex.txt (το οποίο παρέχεται στον ίδιο φάκελο).

Επεξήγηση Αλγόριθμου που ακολουθείται στην συνάρτηση: Ξεκινώντας ανοίγει το αρχείο που θα γραφτεί ο κώδικας latex.Στην προκείμενη περίπτωση επιλέχθηκε αρχείο txt για ευκολία στον έλεγχο των αποτελεσμάτων. Αφού ανοίξει το αρχείο υπολογίζονται τα μη μηδενικά στοιχεία του μητρώου που δόθηκε, έτσι ώστε το val να έχει το κατάλληλο μέγεθος στην εντολή:

```
val = \\begin{tabular}{|l|l....|l|}\\hline
```

Έπειτα η συνάρτηση ελέγχει τον δοσμένο τύπο αραιής αναπαράστασης. Συγκεκριμένα ελέγχεται αν επιθυμούμε να έχουμε 'csr' ή 'csc'. Έστω ότι επιλέγεται 'csr', τότε τα στοιχεία της κάθε γραμμής 'ομαδοποιούνται' και αποθηκεύονται σε ένα κελί cell array όλα τα μη μηδενικά στοιχεία της κάθε γραμμής και ταυτόχρονα υπολογίζονται οι τιμές για το JA vector, οι οποίες θα τυπωθούν αργότερα στο αρχείο. Έπειτα το cell array μετατρέπεται σε matrix τύπου double και απο εκεί σε τύπου string. Τέλος χωρίζεται το πρώτο στοιχείο από τα υπόλοιπα προσωρινά ώστε να προστεθεί ο τελεστής & σε όλα τα στοιχεία εκτός του πρώτου και ενώνεται το τελικό αποτέλεσμα(το πρώτο στοιχείο με τα υπόλοιπα που έχουν τον τελεστή & μπροστά τους) και τυπώνεται το αποτέλεσμα στο αρχείο. Στην συνέχεια ξεκινάει η δημιουργία των εντολών latex για το IA vector. Ξεκινώντας για την δημιουργία του IA, αποθηκεύεται το μητρώο στην αραιή αναπαράστασή του με την εντολή sparse() της Matlab για πιο εύκολη διαχείριση, αφού εκτελώντας τις εντολές:

```
%Storing transpose A in sparse type in order to deal only
%with nonzero elements.
x=sparse(transpose(A));

[row2, columns2]=find(x);

%Storing column in wich the element is in original matrix, thus
%creating col_idx/IA vector to be printed to file.
col_idx = transpose(row2);
```

έχουμε δημιουργήσει το IA vector και το μόνο που μένει είναι να χωρίσουμε το πρώτο στοιχείο από τα υπόλοιπα προσωρινά ώστε να προστεθεί ο τελεστής & σε όλα τα στοιχεία εκτός του πρώτου και να ενώσουμε το τελικό αποτέλεσμα(το πρώτο στοιχείο με τα υπόλοιπα που έχουν τον τελεστή & μπροστά), ώστε να τυπώσουμε το αποτέλεσμα στο αρχείο. Μόλις τυπωθεί και το IA στο αρχείο ξεκινάει και η δημιουργία εντολών latex για το JA vector (το οποίο υπολογίζεται παράλληλα με το val). Εδώ το μόνο που γίνεται εφόσον το JA vector έχει υπολογιστεί είναι να να χωριστεί το πρώτο στοιχείο απο τα υπόλοιπα προσωρινά

ώστε να προστεθεί ο τελεστής & σε όλα τα στοιχεία εκτός του πρώτου και να ενωθεί το τελικό αποτέλεσμα(το πρώτο στοιχείο με τα υπόλοιπα που έχουν τον τελεστή & μπροστά), ώστε να τυπωθεί το αποτέλεσμα στο αρχείο.Στην συνέχεια ακολουθεί παράδειγμα εκτέλεσης και ο κώδικας της συνάρτησης.

Παράδειγμα εκτέλεσης:

Έστω μητρώο A που δίνεται στην εκφώνηση:

$$A = \begin{pmatrix} 0.3984 & 0.1895 & 0.8423 & 0.0000 \\ 0.0000 & 0.5458 & 0.0000 & 0.0000 \\ 0.9416 & 0.4122 & 0.1788 & 0.0000 \\ 0.0000 & 0.0000 & 0.7134 & 0.0000 \end{pmatrix}$$

Τότε το αποτέλεσμα της εντολής [val,row_ip,col_ip]=sp_mat2latex(A,'csr'); στο αρχείο sp_mat2latex.txt είναι:

```
$$val= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 0.3984 &0.1895 &0.8423 &0.5458 &0.9416 &0.4122 &0.1788 &0.7134 \\ \hline
\end{tabular}$$
$$IA= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 1 &2 &3 &2 &1 &2 &3 &3 \\ \hline
\end{tabular}$$
$$JA= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 1 &4 &5 &8 &9  \\ \hline
\end{tabular}$$
```

Αν τρέξουμε τον κώδικα αυτόν σαν κώδικα Latex το αποτέλεσμα που λαμβάνουμε είναι:

$val =$

0.3984	0.1895	0.8423	0.5458	0.9416	0.4122	0.1788	0.7134

$IA =$

1	2	3	2	1	2	3	3

$JA =$

1	4	5	8	9

Έστω τώρα ότι βρισκόμαστε σε 'csr' και το μητρώο A που δίνεται έχει κενή γραμμή(γραμμή 2):

$$A = \begin{pmatrix} 0.3984 & 0.1895 & 0.8423 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.9416 & 0.4122 & 0.1788 & 0.0000 \\ 0.0000 & 0.0000 & 0.7134 & 0.0000 \end{pmatrix}$$

Τότε με αυτή την είσοδο το αποτέλεσμα που λαμβάνουμε είναι:

$val =$

0.3984	0.1895	0.8423	0.9416	0.4122	0.1788	0.7134

$IA =$

1	2	3	1	2	3	3

$JA =$

1	4	4	7	8

Ενώ αν επιθυμούμε 'csc' [val,row_ip,col_ip]=sp_mat2latex(A,'csc'); και το αποτέλεσμα της εκτέλεσης βρίσκεται πάλι στο αρχείο sp_mat2latex.txt. Το αποτέλεσμα που λαμβάνουμε στο αρχείο sp_mat2latex.txt είναι:

```
$$val= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 0.3984 &0.9416 &0.1895 &0.5458 &0.4122 &0.8423 &0.1788 &0.7134 \\ \hline
\end{tabular}$$
$$IA= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 1 &3 &1 &2 &3 &1 &3 &4 \\ \hline
\end{tabular}$$
$$JA= \begin{tabular}{|l|l|l|l|l|l|l|l|}\hline
 1 &3 &6 &9 &9  \\ \hline
\end{tabular}$$
```

Αν τρέξουμε τον κώδικα αυτόν σαν κώδικα Latex το αποτέλεσμα που λαμβάνουμε είναι:

$val =$

0.3984	0.9416	0.1895	0.5458	0.4122	0.8423	0.1788	0.7134

$IA =$

1	3	1	2	3	1	3	4

$JA =$

1	3	6	9	9

Έστω τώρα ότι βρισκόμαστε σε 'csc' και το μητρώο A που δίνεται έχει κενή στήλη(στήλη 2):

$$A = \begin{pmatrix} 0.3984 & 0.0000 & 0.8423 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.9416 & 0.0000 & 0.1788 & 0.0000 \\ 0.0000 & 0.0000 & 0.7134 & 0.0000 \end{pmatrix}$$

Τότε με αυτή την είσοδο το αποτέλεσμα που λαμβάνουμε είναι:

$$val = \boxed{0.3984} \boxed{0.8423} \boxed{0.9416} \boxed{0.1788} \boxed{0.7134}$$

$$IA = \boxed{1} \boxed{3} \boxed{1} \boxed{3} \boxed{3}$$

$$JA = \boxed{1} \boxed{3} \boxed{3} \boxed{5} \boxed{6}$$

Ο κώδικας της συνάρτησης sp_matri2latex.m είναι:

```
function [val,row_ip,col_ip]= sp_mat2latex(A,sp_type)
% Author : ALEXANDROS NATSOLLARI , AM:1057769 , Date : 17/11/2021


    %open file to write the results,it can be whatever type.In this case
    %its a txt file.
    fid = fopen('sp_mat2latex.txt', 'w');

    %%%%%%%%%%%%%%%%%%%%%%%% val creation %%%%%%%%%%%%%%%%%%%%%%%%

    %In order for val to have the proper size each time based on the
    %given matrix we calculate nonzeros of given matrix and create
    %val = \\begin{tabular}{|l|l....|l|}\\hline based on that.
    k = nnz(A);
    val= zeros(1:k);%preallocation of val.
    val_str2 = zeros(1:k);
    val_str1 = '$$val= \begin{tabular}{|';
    val_str2 = repmat('l|', 1, k); %repmat is used instead of for loop.
    val_str3 = '}\hline';
    final_val_str = [val_str1 val_str2 val_str3];
    %The first  Latex line to be printed to file, is printed later to
    %to the file along with all the other lines in order to reduce
    %writing delays.

%Checking if sp_type is csr or csc.
if  strcmp(sp_type,'csr')

    [rows, columns] = size(A);
    CurrentRow = 1;

    %preallocation of NonZero Elements.
    NonZeroElements = {1:k};

    %preallocation of row_ptr.
    row_ptr = (1:rows+1);

    %We store nonzero elements of each line (since we are in csr sp_type)
    %cells.
    for i = 1:rows

        j=i+1;
        thisRow = A(i, :);
        nonzero = thisRow(thisRow ~= 0);

        %Calculating row_ptr/JA.
        row_ptr(1)= 1;
        row_ptr(j)= numel(nonzero) + row_ptr(i);

        %Store nonzero elements.
        if ~isempty(nonzero)
            NonZeroElements{CurrentRow} = nonzero;
            CurrentRow = CurrentRow+1;

        end

    end
```

```
%Converting cell arrays to matrix and then from double matrix to
%string in order to add & to desired numbers.
val = cell2mat(NonZeroElements);

%Split first element from others temporarly so it doesn't get the &
%like the others will.
col_idx_temp1 = val(1,1);
col_idx_tempstr1 = num2str(col_idx_temp1);
col_idx_temp2 = val(1,2:end);
col_idx_tempstr2 = string(col_idx_temp2);

%All elements from 2:end get & in front of them.
tempstr2_final = strcat('&',col_idx_tempstr2);


%%%%%%%%%%% Writing latex code for val to the file %%%%%%%%%%
fprintf(fid,'%s\n',final_val_str);
fprintf(fid,' %s %s',col_idx_tempstr1,tempstr2_final);
fprintf(fid," \\\\ \\hline\n");
fprintf(fid,"\\end{tabular}$$\n");



%%%%%%%%%%%%%%%%%%%%%%% col_idx/IA creation %%%%%%%%%%%%%%%%%%%%%%%
col_idx_str2 = zeros(1:k);

col_idx_str1 = '$$IA= \begin{tabular}{|';
%repmat is used instead of for loop.
col_idx_str2 = repmat('l|', 1, k);
col_idx_str3 = '}\hline';
final_col_idx_str = [col_idx_str1 col_idx_str2 col_idx_str3];

%Storing transpose A in sparse type in order to deal only
%with nonzero elements.
x=sparse(transpose(A));

[row2, columns2]=find(x);

%Storing column in wich the element is in original matrix, thus
%creating col_idx/IA vector to be printed to file.
col_idx = transpose(row2);

%Split first element from others temporarly so it doesn't get the &
%like the others will.
col_idx_temp1 = col_idx(1,1);
col_idx_tempstr1 = num2str(col_idx_temp1);
col_idx_temp2 = col_idx(1,2:end);
col_idx_tempstr2 = string(col_idx_temp2);

%All elements from 2:end get & in front of them.
tempstr2_final = strcat('&',col_idx_tempstr2);



%%%%%%%%%%% Writing latex code for col_idx to the file %%%%%%%%%%
fprintf(fid,"%s\n",final_col_idx_str);
fprintf(fid,' %s %s',col_idx_tempstr1,tempstr2_final);
fprintf(fid," \\\\ \\hline\n");
fprintf(fid,"\\end{tabular}$$\n");
```

```
    %%%%%%%%%%%%%%%%%%%%%%%% row_ptr/JA creation %%%%%%%%%%%%%%%%%%%%%%%%
    row_ptr_str2 = zeros(1:k);

    row_ptr_str1 = '$$JA= \begin{tabular}{|';
    %repmat is used instead of for loop.
    row_ptr_str2 = repmat('l|', 1, k);
    row_ptr_str3 = '}\hline';
    final_row_ptr_str = [row_ptr_str1 row_ptr_str2 row_ptr_str3];


    %Split first element from others temporarly so it doesn't get the &
    %like the others will.
    row_ptr_temp1 = row_ptr(1,1);
    row_ptr_tempstr1 = num2str(row_ptr_temp1);
    row_ptr_temp2 = row_ptr(1,2:end);
    row_ptr_tempstr2 = string(row_ptr_temp2);

    %All elements from 2:end get & in front of them.
    tempstr2_final = strcat('&',row_ptr_tempstr2);



    %%%%%%%%%%% Writing latex code for row_ptr to the file %%%%%%%%%%
    fprintf(fid,"%s\n",final_row_ptr_str);
    fprintf(fid,' %s %s',row_ptr_tempstr1,tempstr2_final);
    fprintf(fid," \\\\ \\hline\n");
    fprintf(fid,"\\end{tabular}$$\n");

    %saving IA to col_ip and JA to row_ip and closing file.
    row_ip = row_ptr;
    col_ip = col_idx;
    fclose(fid);

end

%Same as before but for 'csc' type this time
if  strcmp(sp_type,'csc')

    fid = fopen('sp_mat2latex.txt', 'w');
    val=(1:k);%preallocation of val

    %%%%%%%%%%%%%%%%%%%%%%%% val creation %%%%%%%%%%%%%%%%%%%%%%%%

    [rows, columns] = size(A);
    CurrentColumn = 1;

    %preallocation of NonZero Elements
    NonZeroElements = {1:k};

    %preallocation of col_ptr
    col_ptr = (1:columns+1);

    %We store nonzero elements of each column (since we
    %are in csc sp_type)to cells.
    for i = 1:columns
        j=i+1;
        thisColumn = A(:, i);
        nonzero = thisColumn(thisColumn ~= 0);

        %Calculate col_ptr/JA
        col_ptr(1)= 1;
        col_ptr(j)= numel(nonzero) + col_ptr(i);
```

```
    %Store nonzero elements.
    if ~isempty(nonzero)

        %We need nonzero' in order cell2mat to work properly.
        NonZeroElements{CurrentColumn} = nonzero';
        CurrentColumn = CurrentColumn+1;


    end

end


%Converting cell arrays to matrix and then from double matrix
%to string in order to add & to desired numbers.
val = cell2mat(NonZeroElements);

%Split first element from others temporarly so it doesn't get the &
%like the others will.
row_idx_temp1 = val(1,1);
row_idx_tempstr1 = num2str(row_idx_temp1);
row_idx_temp2 = val(1,2:end);
row_idx_tempstr2 = string(row_idx_temp2);

%All elements from 2:end get & in front of them.
tempstr2_final = strcat('&',row_idx_tempstr2);


%%%%%%%%%%% Writing latex code for val to the file %%%%%%%%%%
fprintf(fid,'%s\n',final_val_str);
fprintf(fid,' %s %s',row_idx_tempstr1,tempstr2_final);
fprintf(fid," \\\\ \\hline\n");
fprintf(fid,"\\end{tabular}$$\n");



%%%%%%%%%%%%%%%%%%%%%%% row_idx/IA creation %%%%%%%%%%%%%%%%%%%%%%%
row_idx_str2 = [];
row_idx_str1 = '$$IA= \begin{tabular}{|';
%repmat is used instead of for loop.
row_idx_str2 = repmat('l|', 1, k);
row_idx_str3 = '}\hline';
final_row_idx_str = [row_idx_str1 row_idx_str2 row_idx_str3];

%Storing A in sparse type in order to deal only with
%nonzero elements.
x=sparse(A);

[row2, columns2]=find(x);

row_idx = row2';
%Split first element from others temporarly so it doesn't get the &
%like the others will.
row_idx_temp1 = row_idx(1,1);
row_idx_tempstr1 = num2str(row_idx_temp1);
row_idx_temp2 = row_idx(1,2:end);
row_idx_tempstr2 = string(row_idx_temp2);

%All elements from 2:end get & in front of them.
tempstr2_final = strcat('&',row_idx_tempstr2);
```

```
    %%%%%%%%%%% Writing latex code for row_idx/IA to the file %%%%%%%%%%
    fprintf(fid,"%s\n",final_row_idx_str);
    fprintf(fid,' %s %s',row_idx_tempstr1,tempstr2_final);
    fprintf(fid," \\\\ \\hline\n");
    fprintf(fid,"\\end{tabular}$$\n");



    %%%%%%%%%%%%%%%%%%%%%%% col_ptr/JA creation %%%%%%%%%%%%%%%%%%%%%%%
    col_ptr_str2 = [];
    col_ptr_str1 = '$$JA= \begin{tabular}{|';
    %repmat is used instead of for loop.
    col_ptr_str2 = repmat('l|', 1, k);
    col_ptr_str3 = '}\hline';
    final_col_ptr_str = [col_ptr_str1 col_ptr_str2 col_ptr_str3];


    %Split first element from others temporarly so it doesn't get the &
    %like the others will.
    col_ptr_temp1 = col_ptr(1,1);
    col_ptr_tempstr1 = num2str(col_ptr_temp1);
    col_ptr_temp2 = col_ptr(1,2:end);
    col_ptr_tempstr2 = string(col_ptr_temp2);

    %All elements from 2:end get & in front of them.
    tempstr2_final = strcat('&',col_ptr_tempstr2);



    %%%%%%%%%%% Writing latex code for col_ptr/JA to the file %%%%%%%%%%
    fprintf(fid,"%s\n",final_col_ptr_str);
    fprintf(fid,' %s %s',col_ptr_tempstr1,tempstr2_final);
    fprintf(fid," \\\\ \\hline\n");
    fprintf(fid,"\\end{tabular}$$\n");

    %saving IA to row_ip and JA to column_ip and closing file.
    row_ip = row_idx;
    col_ip = col_ptr;
    fclose(fid);

end


if  strcmp(sp_type,'csr') ~= 1 && strcmp(sp_type,'csc') ~= 1
    fprintf('Worng type of sparse matrix.\n');
end



end
```

2.2 Κατασκευή συνάρτησης blkToeplitzTrid

Ο κώδικας βρίσκεται στο αρχείο blkToeplitzTrid.m καθώς και στο τέλος της ενότητας αυτής. Για την κατασκευή του block Toeplitz τριδιαγώνιου μητρώου χρησιμοποιούνται οι εντολές kron,diag,ones της Matlab και το μήκος του κώδικα είναι 1 γραμμή.

Επεξήγηση Αλγόριθμου που ακολουθείται στην συνάρτηση:
Για να φτιάξουμε το μπλοκ Toeplitz τριδιαγώνιο μητρώο σε αραιή μορφή χρησιμοποιούμε το γινόμενο Kronecker μεταξύ του μητρώου A και του μητρώου eye(n), (δηλαδή kron(eye(n),A)) φτιάχνοντας έτσι ένα μητρώο με διαστάσεις $n^2 \times n^2$, το οποίο έχει ως κύρια (μπλοκ) διαγώνιο το μητρώο A. Έπειτα σε αυτό το μητρώο προστίθεται ένα μητρώο το οποίο προκύπτει από το γινόμενο Kronecker μεταξύ του μητρώου B και της κάτω διαγώνιου του μητρώου ones(n-1,1), (δηλαδή kron(diag(ones(n-1,1), -1),B)), φτιάχνοντας έτσι την (μπλοκ) υπό-διαγώνιο του τελικού μητρώου. Τέλος προστίθεται και ένα μητρώο το οποίο προκύπτει από το γινόμενο Kronecker μεταξύ του μητρώου C και της πάνω διαγώνιου του μητρώου ones(n-1,1), (δηλαδή kron(diag(ones(n-1,1), 1),C)), φτιάχνοντας έτσι την (μπλοκ) υπερ-διαγώνιο του τελικού μητρώου. Τέλος το αποτέλεσμα αποθηκεύεται σε αραιή αναπαράσταση με την εντολή sparse() της Matlab. Παρακάτω ακολουθεί παράδειγμα εκτέλεσης και ο κώδικας της συνάρτησης.

Παράδειγμα εκτέλεσης:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix} C = \begin{pmatrix} 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \end{pmatrix}$$

Και για n=4 έχουμε τα εξής αποτελέσματα:

```
ans =

     1     1     1     1     3     3     3     3     0     0     0     0     0     0     0     0
     1     1     1     1     3     3     3     3     0     0     0     0     0     0     0     0
     1     1     1     1     3     3     3     3     0     0     0     0     0     0     0     0
     1     1     1     1     3     3     3     3     0     0     0     0     0     0     0     0
     2     2     2     2     1     1     1     1     3     3     3     3     0     0     0     0
     2     2     2     2     1     1     1     1     3     3     3     3     0     0     0     0
     2     2     2     2     1     1     1     1     3     3     3     3     0     0     0     0
     2     2     2     2     1     1     1     1     3     3     3     3     0     0     0     0
     0     0     0     0     2     2     2     2     1     1     1     1     3     3     3     3
     0     0     0     0     2     2     2     2     1     1     1     1     3     3     3     3
     0     0     0     0     2     2     2     2     1     1     1     1     3     3     3     3
     0     0     0     0     0     0     0     0     2     2     2     2     1     1     1     1
     0     0     0     0     0     0     0     0     2     2     2     2     1     1     1     1
     0     0     0     0     0     0     0     0     2     2     2     2     1     1     1     1
```

Figure 2.1: Αποτελέσματα εκτέλεσης Full(S)

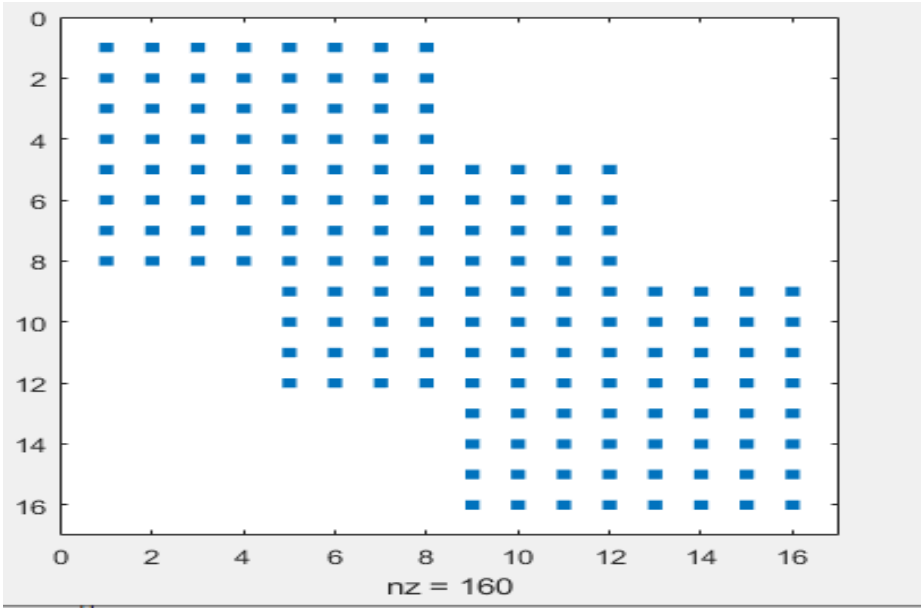


Figure 2.2: Αποτελέσματα εκτέλεσης spy(S)

Ο κώδικας της συνάρτησης blkToeplitzTrid.m είναι:

```
function S=blkToeplitzTrid(n,B,A,C)
% Author : ALEXANDROS NATSOLLARI , AM:1057769 , Date : 21/11/2021

S = sparse(kron(eye(n),A) + kron(diag(ones(n-1,1), -1),B)+ kron(diag(ones(n-1,1), 1),C));

%full(S)
%spy(S);


end
```

2.3 Κατασκευή συνάρτησης sp_mx2bccs

Επεξήγηση Αλγόριθμου που ακολουθείται στην συνάρτηση:
Ξεκινώντας σε κάθε επανάληψη καθώς διατρέχουμε μια στήλη από πάνω προς τα κάτω αυξάνεται το counter: brow_idx_counter (κατά ένα κάθε φορά), η τιμή του οποίου αποθηκεύεται στο array brow_idx, εφόσον το μπλοκ που ελέγχεται έχει τουλάχιστον ένα μη μηδενικό στοιχείο.Αν βρεθεί μη μηδενικό στοιχείο στο μπλοκ εκτός απο την τιμή του brow_idx_counter που αποθηκεύεται στο brow_idx.Αποθηκεύεται επίσης (στο val που στο τέλος μετατρέπεται σε matrix) και το μπλοκ σε ένα cell array σαν διάνυσμα $1 \times nbsq$,όπου το nbsq είναι το μέγεθος του μπλοκ στο τετράγωνο.Τέλος, αφού στην στήλη αυτή βρέθηκε μη μηδενικό μπλοκ αυξάνεται ο counter: bcol_ptr_counter (κατά ένα κάθε φορά).Έπειτα γίνεται έλεγχος αν η γραμμή μπλοκ της στήλης που βρισκόμαστε είναι η τελευταία.Αν είναι η τελευταία τότε ο brow_idx_counter μηδενίζεται καθώς αυτό σημαίνει ότι στην επόμενη επανάληψη θα αλλάξει και στήλη μπλοκ.Οπότε το μέτρημα των μπλοκ γραμμών πρέπει να ξανά αρχίσει από την αρχή.Στην συνέχεια ενεργοποιείται η συνθήκη αλλαγής στήλης (με switch_column = 1) και αποθηκεύεται το bcol_ptr_counter αυξημένο κατά ένα στο matrix bcol_ptr,ώστε να ξέρουμε δηλαδή σε ποιο στοιχείο γίνεται η αλλαγή στήλης.Αφού αποθηκευτεί η πληροφορία αυτή στο bcol_ptr απενεργοποιείται η συνθήκη αλλαγής στήλης (με switch_column = 0) και ξεκινάει η επόμενη επανάληψη μέχρι να εξαντληθούν όλα τα μπλοκ του μητρώου.Στο τέλος όλων των επαναλήψεων το cell array val={}, μετατρέπεται σε matrix με την εντολή «val=cell2mat(val);».Στην συνέχεια ακολουθεί παράδειγμα εκτέλεσης και ο κώδικας της συνάρτησης.

Παράδειγμα εκτέλεσης με το μητρώο που δίνεται στην εκφώνηση:

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \\ 6 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Για το μητρώο A και nb=2 το [val,browidx,bcol]= sp2bccs(A,nb) παράγει τα εξής αποτελέσματα:

$$val = \boxed{6}\boxed{3}\boxed{3}\boxed{0}\boxed{2}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{4}\boxed{2}\boxed{2}\boxed{2}\boxed{2}\boxed{1}\boxed{1}\boxed{1}$$

$$brow\_idx = \boxed{2}\boxed{3}\boxed{1}\boxed{1}\boxed{3}$$

$$bcol\_ptr = \boxed{1}\boxed{3}\boxed{4}\boxed{6}$$

$$B = \begin{pmatrix} 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 6 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Για το μητρώο B που έχει κενή μπλοκ στήλη και nb=2 το [val,browidx,bcol]= sp2bccs(A,nb) παράγει τα εξής αποτελέσματα:

$$val = \boxed{6}\boxed{3}\boxed{3}\boxed{0}\boxed{2}\boxed{1}\boxed{1}\boxed{1}\boxed{4}\boxed{2}\boxed{2}\boxed{2}\boxed{2}\boxed{1}\boxed{1}\boxed{1}$$

$$brow\_idx = \boxed{2}\boxed{3}\boxed{1}\boxed{3}$$

$$bcol\_ptr = \boxed{1}\boxed{3}\boxed{3}\boxed{5}$$

Ο κώδικας της συνάρτησης sp_mx2bccs.m είναι:

```
function [val,brow_idx,bcol_ptr]= sp_mx2bccs(A,nb)
% Author : ALEXANDROS NATSOLLARI , AM:1057769 , Date : 25/11/2021


%Computes number of rows in A.
num_rows = size(A,1);
%num_columns=num_rows since A is nxn.
num_columns = num_rows;
%block size square gia xrhsh sto reshape.
nbsq=nb*nb;


%val info.
%preallocation of val with worst case senario of all
%blocks being non zero.
non_zero_blocks = (num_rows*num_columns)/(nb*nb);
%total storage = number of non zero block*block size.
val={non_zero_blocks};
%aplh metavlhth poy leitourgei san cell position counter
%simple variable counter that points the correct cell position.
k=1;


%brow_idx info.
%preallocation of brow_idx with worst case senario of all
%blocks being non zero.
brow_idx = (non_zero_blocks);
%simple variable counter that points the correct array position.
m=1;
brow_idx_counter=0;


%bcol_ptr info.
%preallocation of bcol_ptr with worst case senario of all
%blocks being non zero
bcol_ptr = ((num_columns/nb)+1);
bcol_ptr(1) = 1;
%simple variable counter that points the correct array position.
p=2;
bcol_ptr_counter=0;

%simple variable that works like switch enabler for if statement
%line 82.
switch_column = 0;


for i=1:nb:(num_columns)

    for j=1:nb:(num_rows)

        brow_idx_counter = brow_idx_counter + 1;

        %Checking if the block has any non zero element.
        if any(A(j:j+nb-1,i:i+nb-1),'all')


            %If a non zero element is found
```

```
            %block is stored to cell array val.
            val{k}= reshape(A(j:j+nb-1,i:i+nb-1), [1 nbsq]);
            k=k+1;


            bcol_ptr_counter = bcol_ptr_counter + 1;
            brow_idx(m) = brow_idx_counter;
            m=m+1;

        end

        %Checking to see if loop reached in last block of
        %block column.If yes then counter of block row is set
        %to zero since we are changing block column.
        if brow_idx_counter == (num_rows/nb)

             brow_idx_counter = 0;

             %enable change of column since we parsed through
             %this block columns lines.
             switch_column=1;

        end

        if switch_column == 1

            %Since block column has changed store to bcol_ptr
            %in which element that change happened.
            bcol_ptr(p)= bcol_ptr_counter + 1;
            p=p+1;
            switch_column=0;

        end

    end

end

%Changing val from cell to mat.
val=cell2mat(val);
% brow_idx
% bcol_ptr

end
```

2.4 Κατασκευή συνάρτησης spmv_bccs

Επεξήγηση Αλγόριθμου που ακολουθείται στην συνάρτηση:
Ξεκινώντας ελέγχεται αν τα διανύσματα x,y έχουν τις κατάλληλες διαστάσεις ώστε να γίνουν οι πράξεις σωστά στην συνάρτηση.Αν δεν έχουν τις κατάλληλες διαστάσεις γίνονται οι απαραίτητες αναστροφές στα διανύσματα που χρειάζεται.Έπειτα σε κάθε επανάληψη λαμβάνονται τα μπλοκ στοιχεία της μπλοκ στήλης από το val, αντιστοιχίζονται στις κατάλληλες θέσεις του y και αφού γίνει ο πολλαπλασιασμός της κάθε στήλης(υπό-στήλης του μπλοκ) με το σωστό στοιχείο του x διανύσματος προστίθεται με τα στοιχεία που βρίσκονται στις θέσεις του y αυτής της επανάληψης και το αποτέλεσμα αποθηκεύεται σε αυτές τις θέσεις του y.Δηλαδή έστω ότι το μητρώο A και τα διανύσματα x,y[1] είναι αυτά που φαίνονται παρακάτω:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \\ 6 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Τότε μέσο της συνάρτησης sp_mx2bccs.m για nb=2 λαμβάνουμε τα:

$val =$

6	3	3	0	2	1	1	1	0	1	0	1	4	2	2	2	2	1	1	1

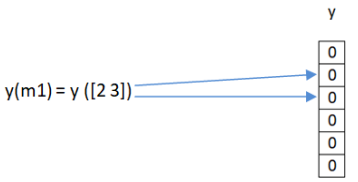
$brow\_idx =$

2	3	1	1	3

$bcol\_ptr =$

1	3	4	6

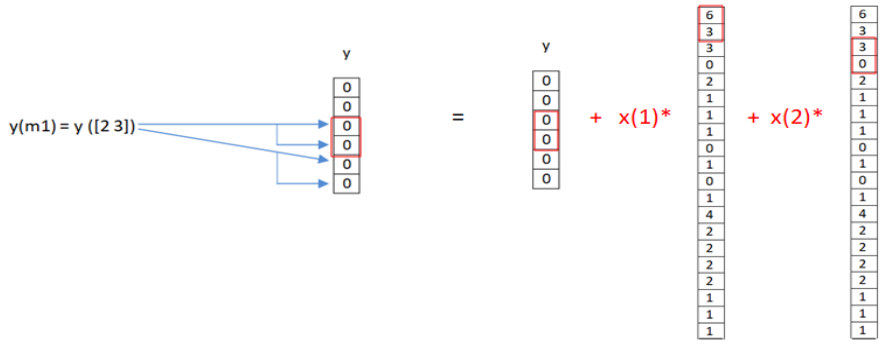
Τότε για την πρώτη επανάληψη ο αλγόριθμος δουλεύει ως εξής:
k1 = bcol_ptr(j) = 1
k2 = bcol_ptr(j+1) – 1 = 2
m1 = brow_idx (k1:k2)

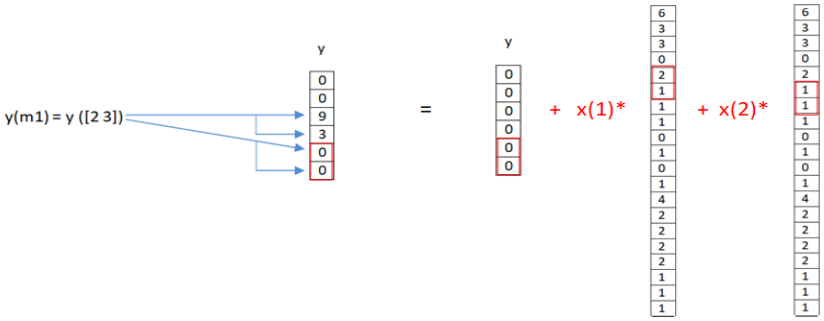


[1]!Προσοχή: Το y για λόγους ευ παρουσίασης στο παράδειγμα δίνεται σαν 6×1, δηλαδή σαν στήλη, ωστόσο στην συνάρτηση χειρίζεται ως 1×6,δηλαδή σαν γραμμή,ώστε να τρέχει σωστά η συνάρτηση, λόγο του ότι το val είναι γραμμή.

Αλλά επειδή έχουμε μπλοκ στοιχεία στο val και όχι απλά στοιχεία, σημαίνει ότι το y([2 3]) στην ουσία δείχνει μπλοκ γραμμές στο y, οπότε το σωστό είναι:

Οπότε για j=1,nb = 2 και για όσα μη μηδενικά μπλοκ έχει η μπλοκ στήλη (έστω 2 εδώ αφού non_zero_blocks = (k2-k1) + 1) έχουμε:





Οπότε το y στο τέλος αυτής της επανάληψης (για j=1)θα είναι:

$$y = \begin{bmatrix} 0 \\ 0 \\ 9 \\ 3 \\ 3 \\ 2 \end{bmatrix}$$

Παράδειγμα εκτέλεσης:
Για τα A,y,x που φαίνονται παρακάτω

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \\ 6 & 3 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 2 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Το αποτέλεσμα που λαμβάνουμε καλώντας την spmvbccs.m με τα παραπάνω δεδομένα είναι:

$$y = \begin{bmatrix} 6 & 6 & 9 & 3 & 6 & 4 \end{bmatrix}$$

Ο κώδικας της συνάρτησης spmv_bccs.m είναι:

```
function [y]= spmv_bccs(y,x,nb,val,brow_idx,bcol_ptr)
% Author : ALEXANDROS NATSOLLARI , AM:1057769 , Date : 10/12/2021


%check if y,x have the right dimensions in odrer the function to run
%properly, if not then transpose the vectors accordingly.
[dim1, dim2]=size(y);
[dim3, dim4]=size(x);

if (dim1 ~= 1 && dim2 == 1)

    y=transpose(y);

elseif (dim4 ~= 1 && dim3 == 1)

    x=transpose(x);

end

%Variable Initializations.

%Variable z is a counter that serves the purpose of multiplying each time
%the correct row of x vector with the correct column of A.
z=1;

%Variable start marks the proper position in which val vector will be
%traversed each time based on the last traverse.
start=1;

%length_of_associative is a variable that stores the length of vector y.
length_of_associative=length(y);
%astive is sort for associative.
astive=1:length_of_associative;

%associative_array is an array that given the block rows of block matrix
%and the size of block creates a cell array that asscociates the block row
%mumber with the row number that rows would have in the original matrix.
astive = mat2cell(astive,1, diff([0:nb:numel(astive)-1 numel(astive)]));


%simple variable that determines how many times will the main loop run
%based on the length of vector bcol_ptr.
loop_times = length(bcol_ptr)-1;

%main loop.
for j=1:loop_times

    %k1 and k2 variables point the block column in which this rounds block
    %elements are.
    k1=bcol_ptr(j);
    k2=bcol_ptr(j+1)-1;

    %m1 variable given the block column of this rounds block elements
    %calculates the rows associated with the block rows.
    m1=cell2mat(astive(brow_idx(k1:k2)));

    %non_zero_blocks stores the number of non zero blocks of this round.
    non_zero_blocks =(k2-k1)+1;
```

```
    %Variable h is a simple counter so x(row)*A(column) is stored in proper
    %position of y.
    h=1;

   %With this loop we traverse all block elements of this round.
   for k=1:non_zero_blocks

        %With this loop we calculate x(row)*A(column), add it with the next
        %x(row)*A(column) in line, so all sub-columns of block column
        %are multiplied with the proper x(row) and then added together.
        for p=1:nb

            y(m1(h:(h+nb)-1)) = y(m1(h:(h+nb)-1)) + x(z)*val(start:(start+nb)-1);

            start=start+nb;

            z=z+1;

        end

        h=h+nb;

        z=1;
   end

   z=j+nb-1;

end
%y
end
```

2.5 Κατασκευή συνάρτησης Askhsh5

Ο κώδικας βρίσκεται στο αρχείο Askhsh5.m καθώς και στο τέλος της ενότητας αυτής.

Επεξήγηση Αλγόριθμου που ακολουθείται στην συνάρτηση:
Η συνάρτηση αυτή δέχεται m,n και υπολογίζει την νόρμα 2 του διανύσματος y που προέρχεται από το y=y+A*x (με το A να είναι αποθηκευμένο όπως δίνεται στην πρωταρχική του μορφή) και την νόρμα 2 του διανύσματος y που προέρχεται από το y=y+A*x (με το A να είναι αποθηκευμένο σε block compressed sparse column αναπαράσταση).Τέλος υπολογίζεται η διαφορά των 2 νορμών και επιστρέφεται.Υπολογίζοντας την διαφορά νορμών για m=32; n=64; , m=64; n=128; και m=128; n=256; λαμβάνουμε αποτέλεσμα 0 που μαρτυρά ότι η υλοποίηση είναι έγκυρη εφόσον η διαφορά νορμών(της απλής υλοποίησης με την υλοποίηση της εργασίας) είναι 0(δηλαδή το σφάλμα είναι 0 ή πολύ μικρό).Παρακάτω ακολουθεί παράδειγμα εκτέλεσης και ο κώδικας της συνάρτησης.

Παράδειγμα εκτέλεσης:
Για m=32; n=64; , m=64; n=128; και m=128; n=256; η διαφορά των 2 νορμών είναι: norm_diff = 0

Ο κώδικας της συνάρτησης Askhsh5.m είναι:

```
function norm_diff = Askhsh5(m,n)
% Author : ALEXANDROS NATSOLLARI , AM:1057769 , Date : 12/12/2021


nb=m;
T=toeplitz([4,-1,zeros(1,m-2)]);
S=blkToeplitzTrid(n,inv(T),T^2,T);
y=eye(n*m,1);
x=(ones(n*m,1));
y1=y;

[val,brow_idx,bcol_ptr]= sp_mx2bccs(S,nb);

%function that calculates simple y=y+A*x and returns the result.
y1 = y1+S*x;

%function that calculates y=y+A*x with A stored in sparse blocked
%form and returns the result.
y2 = spmv_bccs(y,x,nb,val,brow_idx,bcol_ptr);

%Calculation of norm 2 of y from simple y=y+A*x.
simple_yAx_norm = norm(y1,2);

%Calculation of norm 2 of y from sparse blocked y=y+A*x.
spmv_yAx_norm = norm(y2,2);

%norm difference to check if result of function spmv_bccs is correct.
norm_diff = abs(simple_yAx_norm - spmv_yAx_norm);

end
```