3η Εργαστηριακή Άσκηση

Ονοματεπώνυμο: Παναγιώτης Ντενέζος

A.M: 5853

Εισαγωγικά

i) Υπολογιστικό Σύστημα:
   - Intel® Core™ i7-6700 CPU @ 3.40GHz
   - 3 επίπεδα κρυφής μνήμης
     (L1 cache 256KB, L2 cache 1,0MB και L3 cache 8,0MB)
   - DDR4 @ 16GB

ii) Το είδος της πολιτικής εγγραφής στην κρυφή μνήμη είναι write back

iii) Windows 10 Education και MATLAB 8.4.0.150421 (R2014b)

➢ Ερώτημα 1ο

Για την υλοποίηση αυτού του ερωτήματος κατασκευάστηκε η συνάρτηση *matrix_built_5853*, με είσοδο το μέγεθος (n) και την τιμή ή το διάνυσμα της διαγωνίου. Η συνάρτηση αυτή εσωτερικά δημιουργεί το ζητούμενο μητρώο με χρήση της εντολής sparse και μέσω μιας διπλής for τοποθετεί τους άσσους στο περίγραμμα και έπειτα κατασκευάζει τη διαγώνιο. Ο συγκεκριμένος κώδικας είναι ο εξής:

```
function [A] = matrix_build_5853(n,a)
A = sparse(n,n);
for i = 2:n
    A(1,i) = 1;
    A(n,i-1) = 1;
    A(i,1) = 1;
    A(i-1,n) = 1;
end

if length(a) == 1
    for i = 1:n
        A(i,i) = a;
```

```
    end
else
     for i = 1:n
        A(i,i) = a(i);
    end
end
end
```

- **Ερώτημα 2ο**

Για την υλοποίηση αυτού του ερωτήματος, καλείται η παραπάνω συνάρτηση με ορίσματα $n = 6$ και 5853 (δηλ. *matrix_built_5853(6,5853))* και έτσι δημιουργείται το μητρώο A. Για να γραφτεί το μητρώο A σε μορφή CSR χρησιμοποιούνται οι μεταβλητές VAL, COL και IA. Το μητρώο A σαρώνεται ανά γραμμές και VAL περιέχει τα μη μηδενικά στοιχεία του μητρώου A, η COL τη στήλη που βρίσκονται τα μηδικά αυτά στοιχεία και η IA τη θέση του πίνακα COL στην οποία αλλάζει γραμμή το μητρώο. Ο συγκεκριμένος κώδικας είναι ο εξής:

```
n = 6;
a = 5853*ones(n,1);
A = matrix_build_5853(n,a);

c = 1;
index = 2;
IA(1) = 1;

for i = 1:length(A)
    for j = 1:length(A)
        if A(i,j) ~= 0
            VAL(c,1) = A(i,j);
            COL(c,1) = j;
            c = c + 1;
        end
    end
    IA(index,1) = c;
    index = index + 1;
End
```

Και τα αποτελέσματα που προκύπτουν είναι τα παρακάτω:

VAL:

5853	1	1	1	1	1	1	5853	1	1	5853	1	1	5853	1	1	5853	1	1	1	1	1	5853

COL:

1	2	3	4	5	6	1	2	6	1	3	6	1	4	6	1	5	6	1	2	3	4	5	6

IA:

1	7	10	13	16	19	25

- **Ερώτημα 3ο**

Για την υλοποίηση αυτού του ερωτήματος, δημιουργείται ένα κάτω τριγωνικό μητρώο B με χρήση της συνάρτησης *tril*. Για να γραφτεί το μητρώο B σε μορφή CSC χρησιμοποιούνται οι μεταβλητές VAL, ROW, IA. Το μητρώο B σαρώνεται ανά στήλη και VAL περιέχει τα μη μηδενικά στοιχεία του μητρώου B, η COL τη στήλη που βρίσκονται τα μηδικά αυτά στοιχεία και η IA τη θέση του πίνακα COL στην οποία αλλάζει στήλη το μητρώο. Ο συγκεκριμένος κώδικας είναι ο εξής:

```
n = 6;
a = 5853*ones(n,1);
A = matrix_build_5853(n,a);
B = tril(A);


c = 1;
index = 2;
IA(1) = 1;
for i = 1:length(A)
    for j = 1:length(A)
        if A(i,j) ~= 0
            VAL(c,1) = A(i,j);
            ROW(c,1) = j;
            c = c + 1;
        end
    end
    IA(index,1) = c;
    index = index + 1;
end
```

Και τα αποτελέσματα που προκύπτουν είναι τα παρακάτω:

VAL:

5853	1	1	1	1	1	5853	1	5853	1	5853	1	5853	1	5853

ROW:

1	2	3	4	5	6	2	6	3	6	4	6	5	6	6

IA:

1	7	9	11	13	15	16

- **Ερώτημα 4ο**

Για τον υπολογισμό των μη μηδενικών στοιχείων του Α έγινε η εξής σκέψη. Το μητρώο Α στην πρώτη και στην τελευταία γραμμή (n-2) περιέχουν όλο άσσους, ενώ στις υπόλοιπες γραμμές περιέχουν άσσους στην πρώτη στήλη, στο στοιχείο της διαγωνίου και στην τελευταία στήλη. Συνεπώς το μητρώο Α έχει $2n + 3(n-2) = 5n - 6$ μη μηδενικά στοιχεία.

- **Ερώτημα 5ο**

Για την υλοποίηση αυτού του ερωτήματος κατασκευάστηκε η συνάρτηση *matrix_mv_5853*, με είσοδο ένα μητρώο Α και ένα διάνυσμα x. Η συνάρτηση αυτή εκτελεί τον πολλαπλασιασμό μεταξύ του μητρώου και του διανύσματος. Πιο συγκεκριμένα, με τη χρήση επαναληπτικής δομής προσπελαύνει το μητρώο Α κατά στήλες και στη γενική περίπτωση προσθέτει το πρώτο, το τελευταίο και το i-οστό στοιχείο του x πολλαπλασιασμένο επί το i-οστό στοιχείο της διαγωνίου του Α. Το πρώτο και το τελευταίο στοιχείο του διανύσματος εξόδου b, αποτελούν το εσωτερικό γινόμενο της εκάστοτε στήλης με το διάνυσμα x. Ο συγκεκριμένος κώδικας είναι ο εξής:

```
n = 6;
a = 5853;
x = ones(n,1);
b = matrix_mv_5853(a,x);
```

```
function [b] = matrix_mv_5853( a,x )
n = length(x);
b = zeros(n,1);

if length(a) == 1
    a = a*ones(n,1);
end

for j = 1:n
    if (j == 1 || j == n)
        for i = 1:n
            if (i == 1 && j == 1) || (i == n && j == n)
                b(j) = b(j) + a(i)*x(i);
            else
                b(j) = b(j) + x(i);
            end
        end
    else
        b(j) = x(1) + x(n) + x(j)*a(j);
    end
end
end
```

- **Ερώτημα 6ο**

Σε ένα μητρώο για να μπορεί να γίνει η παραγοντοποίηση Cholesky πρέπει να είναι συμμετρικό και θετικά ορισμένο. Είναι γνωστό, από την κατασκευή του, ότι το μητρώο είναι συμμετρικό. Μπορούμε να δείξουμε ότι το μητρώο είναι θετικά ορισμένο αν οι ιδιοτιμές του είναι θετικές. Αυτό μπορεί εύκολα να αποδειχθεί με χρήση του θεωρήματος Gerschgorin. Είναι:

$$|\lambda - \alpha_{ii}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$$

Είναι γνωστό ότι η πρώτη και η τελευταία γραμμή έχουν άθροισμα (χωρίς το στοιχείο της διαγωνίου) ίσο με 5, ενώ όλες οι υπόλοιπες ίσο με 2. Άρα

$$|\lambda - 6| \leq 2 \leftrightarrow -2 \leq \lambda - 6 \leq 2 \leftrightarrow 4 \leq \lambda \leq 8$$

και

$$|\lambda - 6| \leq 5 \leftrightarrow -5 \leq \lambda - 6 \leq 5 \leftrightarrow 1 \leq \lambda \leq 11$$

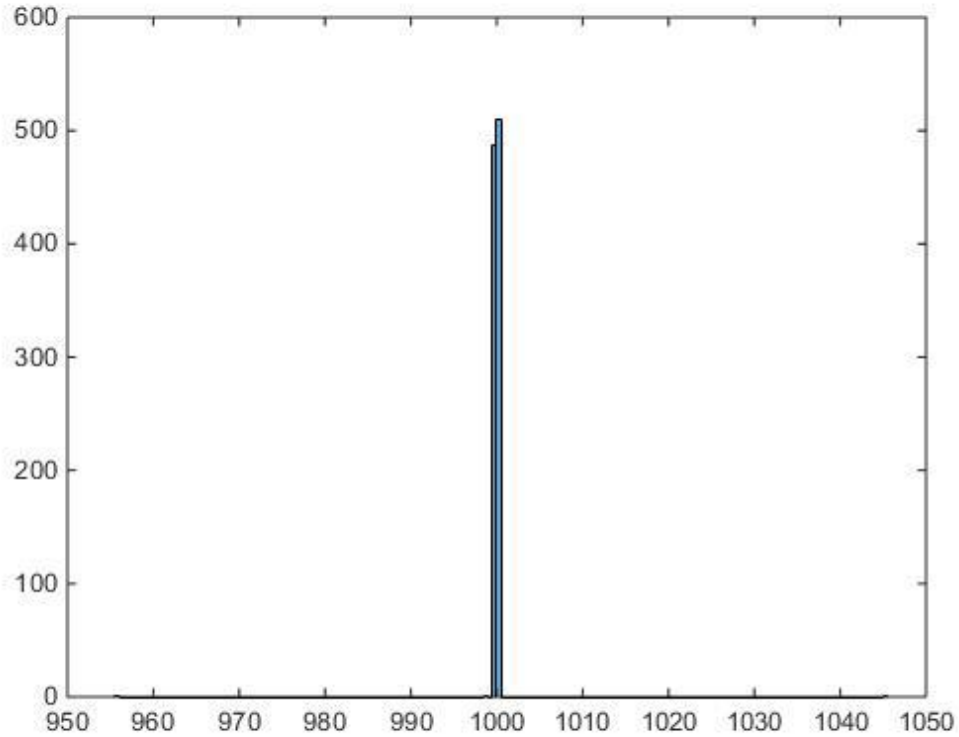
Άρα όλες οι ιδιοτιμές είναι θετικές και συνεπώς το μητρώο είναι θετικά ορισμένο με αποτέλεσμα να μπορεί να παραγοντοποιηθεί με τη μέθοδο Cholesky.

- **Ερώτημα 7ο**

Για την υλοποίηση αυτού του ερωτήματος κατασκευάστηκε ένα μητρώο μεγέθους 1000x1000. Στη διαγώνιο του τοποθετήθηκε η τιμή 100 και με χρήση της συνάρτησης *condest* έγινε μια εκτίμηση του δείκτη κατάστασης του μητρώου. Με χρήση της συνάρτησης *eig* βρέθηκαν οι ιδιοτιμές του μητρώου και με χρήση της συνάρτησης *sort* ταξινομήθηκαν. Ο συγκεκριμένος κώδικας είναι ο εξής:

```
n = 1000;
A = matrix_build_5853(n,n);
condition = condest(A);
s_eig = sort(eig(A));
histogram(s_eig);
```

Στο παρακάτω ιστόγραμμα εμφανίζονται ταξινομημένες.



- **Ερώτημα 8ο**

Για την υλοποίηση αυτού του ερωτήματος δημιουργήθηκε ο παρακάτω πίνακας.

	1	2	3	4	5	6	7
1	500	1.5448e-10	0.0027	9.3542e-12	2.0494e-04	6.8708e-12	9.3972e-05
2	1000	5.8911e-10	0.0196	2.5164e-11	2.8843e-04	3.4204e-11	1.9013e-04
3	5000	5.9252e-08	0.8148	1.2005e-09	0.0014	1.2476e-09	0.0010
4	10000	2.3689e-07	4.2966	4.5275e-10	0.0025	6.4234e-09	0.0021
5	20000	1.8950e-06	26.1646	1.7284e-08	0.0045	2.9964e-08	0.0037

Είναι $b = A * x$ και x το διάνυσμα με μονάδες.

- Η 1η στήλη περιέχει το μέγεθος n
- Η 2η στήλη περιέχει τη νόρμα 2 της $b - A * \hat{x}$, όπου το $\hat{x}$ υπολογίζεται κάνοντας παραγοντοποίηση Cholesky στο A και στην συνέχεια λύνοντας την $A * x = b$. Πιο συγκεκριμένα, ο υπολογισμός του $\hat{x}$ γίνεται ως εξής: $A * x = b \leftrightarrow x = A^{-1} * b \leftrightarrow x = (R^T * R)^{-1} * b \leftrightarrow x = R^{-1} * (R^T)^{-1} * b$, όπου στη Matlab λύνεται με τη χρήση της ανάποδης καθέτου.
- Η 3η στήλη περιέχει τους χρόνους επίλυσης της μεθόδου Cholesky.
- Η 4η στήλη περιέχει τη νόρμα 2 της $b - A * \hat{x}$, όπου το $\hat{x}$ υπολογίζεται μέσω της συνάρτησης *my_pcg* στην οποία έχουν γίνει κατάλληλες αλλαγές, ώστε να εκμεταλλευτούμε την ειδική μορφή του μητρώου.
- Η 5η στήλη περιέχει τους χρόνους επίλυσης της συνάρτησης *my_pcg*.
- Η 6η στήλη περιέχει τη νόρμα 2 της $b - A * \hat{x}$, όπου το $\hat{x}$ υπολογίζεται μέσω της συνάρτησης *calc*, ,όπου υλοποιεί την πράξη $\hat{x} = A\backslash b$.
- Η 7η στήλη περιέχει τους χρόνους επίλυσης της συνάρτησης *calc*.

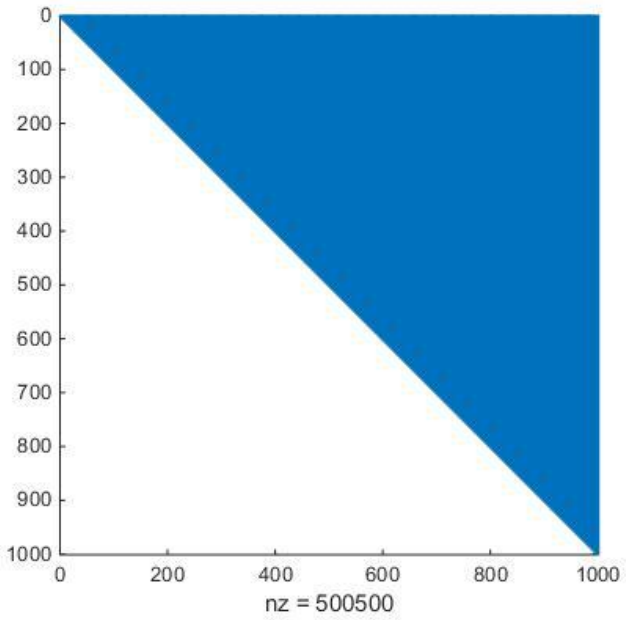
Όλοι οι χρόνοι υπολογίστηκαν με χρήση της μεθόδου *timeit*.

Ο αριθμός των επαναλήψεων που χρειάστηκε η *my_pcg* για την σύγκλιση, δίνεται στον παρακάτω πίνακα.

	1	2
1	500	2
2	1000	2
3	5000	2
4	10000	2
5	20000	2

- Η 1η στήλη περιέχει το μέγεθος n
- Η 2η στήλη περιέχει των αριθμό των επαναλήψεων.

Στην εικόνα παρακάτω φαίνεται η δομή του παράγοντα Cholesky για n = 1000



Οι κώδικες που υλοποιούν όλα τα παραπάνω είναι οι εξής:

Για την Ask8:

```
size = [500 1000 5000 10^4 2*10^4];
results = zeros(5,7);

for i = 1:length(size)
    n = size(i);
    results(i,1) = n;
    x = ones(n,1);
    A = matrix_build_5853(n,n);
    b = A*x;

    x_hat = myCholesky(A,b);
    results(i,2) = norm(b - A*x_hat,2);
    f = @()myCholesky(A,b);
    results(i,3) = timeit(f);

    [x_hat,flag,relres,iter(i,1)] = my_pcg(b,10^(-
8),1000,n,n);
    tmp = matrix_mv_5853(n,x_hat);
```

```
    results(i,4) = norm(b - tmp,2);
    f = @()my_pcg(b,10^(-8),1000,n,n);
    results(i,5) = timeit(f);

    x_hat = calc(A,b);
    results(i,6) = norm(b - A*x_hat,2);
    f = @()calc(A,b);
    results(i,7) = timeit(f);
end
```

Για την συνάρτησης *my_pcg:*

```
function [x,flag,relres,iter,resvec] =
my_pcg(b,tol,maxit,n_in,a)

if (nargin < 2)
    error(message('MATLAB:pcg:NotEnoughInputs'));
end

n = n_in;
existM1 = 0;
existM2 = 0;
n2b = norm(b);
x = zeros(n_in,1);

% Set up for the method
flag = 1;
xmin = x; %Iterate which has minimal residual so far
imin = 0; % Iteration at which xmin was computed
tolb = tol * n2b; % Relative tolerance
%r = b - iterapp('mtimes',afun,atype,afcnstr,x,varargin{:});
r = b - matrix_mv_5853(a,x);
normr = norm(r); % Norm of residual
normr_act = normr;

if (normr <= tolb) % Initial guess is a good enough solution
    flag = 0;
```

```
    relres = normr / n2b;
    iter = 0;
    resvec = normr;
    if (nargout < 2)
        itermsg('pcg',tol,maxit,0,flag,iter,relres);
    end
    return
end

resvec = zeros(maxit+1,1); % Preallocate vector for norm of
residuals
resvec(1,:) = normr; % resvec(1) = norm(b-A*x0)
normrmin = normr; % Norm of minimum residual
rho = 1;
stag = 0; % stagnation of the method
moresteps = 0;
maxmsteps = min([floor(n/50),5,n-maxit]);
maxstagsteps = 3;

% loop over maxit iterations (unless convergence or failure)

for ii = 1 : maxit
    if existM1
        y =
iterapp('mldivide',m1fun,m1type,m1fcnstr,r,varargin{:});
        if ~all(isfinite(y))
            flag = 2;
            break
        end
    else % no preconditioner
        y = r;
    end

    if existM2
        z =
iterapp('mldivide',m2fun,m2type,m2fcnstr,y,varargin{:});
        if ~all(isfinite(z))
```

```
            flag = 2;
            break
        end
    else % no preconditioner
        z = y;
    end

    rho1 = rho;
    rho = r' * z;
    if ((rho == 0) || isinf(rho))
        flag = 4;
        break
    end
    if (ii == 1)
        p = z;
    else
        beta = rho / rho1;
        if ((beta == 0) || isinf(beta))
            flag = 4;
            break
        end
        p = z + beta * p;
    end
    %q = iterapp('mtimes',afun,atype,afcnstr,p,varargin{:});
    q = matrix_mv_5853(a,p);
    pq = p' * q;
    if ((pq <= 0) || isinf(pq))
        flag = 4;
        break
    else
        alpha = rho / pq;
    end
    if isinf(alpha)
        flag = 4;
        break
    end
```

```
    % Check for stagnation of the method
    if (norm(p)*abs(alpha) < eps*norm(x))
        stag = stag + 1;
    else
        stag = 0;
    end

    x = x + alpha * p; % form new iterate
    r = r - alpha * q;
    normr = norm(r);
    normr_act = normr;
    resvec(ii+1,1) = normr;

    % check for convergence
    if (normr <= tolb || stag >= maxstagsteps || moresteps)
   %r = b -
   iterapp('mtimes',afun,atype,afcnstr,x,varargin{:});
        r = b - matrix_mv_5853(a,x);
        normr_act = norm(r);
        resvec(ii+1,1) = normr_act;
        if (normr_act <= tolb)
            flag = 0;
            iter = ii;
            break
        else
            if stag >= maxstagsteps && moresteps == 0
                stag = 0;
            end
            moresteps = moresteps + 1;
            if moresteps >= maxmsteps
                if ~warned

warning(message('MATLAB:pcg:tooSmallTolerance'));
                end
                flag = 3;
                iter = ii;
                break;
```

```
            end
        end
    end
    if (normr_act < normrmin) % update minimal norm
quantities
        normrmin = normr_act;
        xmin = x;
        imin = ii;
    end
    if stag >= maxstagsteps
        flag = 3;
        break;
    end
end % for ii = 1 : maxit

% returned solution is first with minimal residual
if (flag == 0)
    relres = normr_act / n2b;
else
%r_comp = b -
iterapp('mtimes',afun,atype,afcnstr,xmin,varargin{:});
    r_comp = b - matrix_mv_5853(a,xmin);
    if norm(r_comp) <= normr_act
        x = xmin;
        iter = imin;
        relres = norm(r_comp) / n2b;
    else
        iter = ii;
        relres = normr_act / n2b;
    end
end

% truncate the zeros from resvec
if ((flag <= 1) || (flag == 3))
    resvec = resvec(1:ii+1,:);
else
    resvec = resvec(1:ii,:);
```

```
end

% only display a message if the output flag is not used
```

Για την συνάρτησης *calc:*

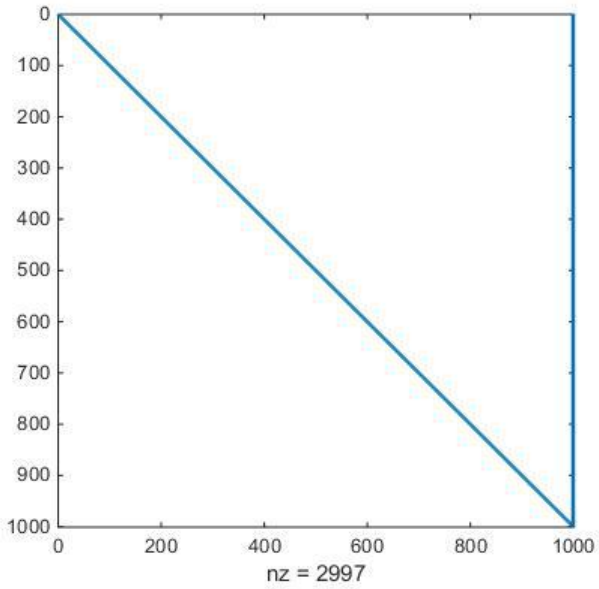
```
function [x] = calc(A,b)
x = A\b;
end
```

Για την συνάρτησης *myCholesky:*

```
function [x] = myCholesky(A,b )
 R = chol(A);
 spy(R)
 x = R\(R'\b);
end
```

- **Ερώτημα 9ο**

Για την υλοποίηση αυτού του ερωτήματος υπολογίζεται η μετάθεση, η μετάθεση αυτή φαίνεται στην εικόνα παρακάτω.



Παρατηρείται ότι ο αριθμός των μη μηδενικών στοιχείων είναι πολύ μικρότερος στη συγκεκριμένη περίπτωση, άρα θα γίνουν λιγότερες πράξεις για την επίλυση του συστήματος.

Ο συγκεκριμένος κώδικας που το κάνει αυτό είναι ο εξής:

```
n = 1000;
A = matrix_build_5853(n,n);
```

```
p = amd(A);
A = A(p,p);

R = chol(A);
spy(R);
```

- **Ερώτημα 10ο**

Για την υλοποίηση αυτού του ερωτήματος γίνεται χρήση της παραπάνω μετάθεσης και θεωρήθηκαν ένα μητρώο $B = P' * A * P$ και ένα διάνυσμα $\hat{b} = P' * b$ με χρήση της μεθόδου Cholesky υπολογίστηκε το αποτέλεσμα το οποίο στην πορεία μετατέθηκε κατά p, ώστε να βρεθεί το χ. Οι χρόνοι που μετρήθηκαν είναι οι εξής:

	1	2	3
1	500	6.8715e-12	7.4698e-05
2	1000	3.4126e-11	1.1179e-04
3	5000	1.2476e-09	4.8638e-04
4	10000	6.4234e-09	9.5133e-04
5	20000	2.9966e-08	0.0019

Ο συγκεκριμένος κώδικας είναι ο εξής:

```
size = [500 1000 5000 10^4 2*10^4];
results2 = zeros(5,3);

for i = 1:length(size)
    n = size(i);
    results2(i,1) = n;
    A = matrix_build_5853(n,n);
    x = ones(n,1);
    b = matrix_mv_5853(n,x);
    p = amd(A);
    B = A(p,p);
    b_hat = b(p);
    y = myCholesky(B,b_hat);
    x = y(p);

    results2(i,2) = norm(b - A*x,2);
    f = @()myCholesky(B,b_hat);
    results2(i,3) = timeit(f);
end
```

- **Ερώτημα 11ο**

  Από τα παραπάνω προκύπτει ότι ως προς την ταχύτητα η πιο γρήγορη είναι η ανάποδη κάθετος, μετά η *my_pcg* και τέλος η Cholesky. Αποτέλεσμα λογικό καθώς η ανάποδη κάθετος ως τελεστής της Matlab εκμεταλλεύεται πλήρως όλες τις ιδιότητες του μητρώου. Ακόμα παρατηρείται ότι η *my_pcg* έχει μικρούς χρόνους αποτέλεσμα επίσης λογικό, καθώς έχει παραμετροποιηθεί για να εκτελεί πιο γρήγορα πράξεις για τα αραιά μητρώα. Παρόλο που η Cholesky είναι η πιο αργή στην περίπτωση με τις μεταθέσεις τα αποτελέσματα που προκύπτουν από άποψη ακρίβειας είναι πολύ κοντά σε αυτά της ανάποδης καθέτου ενώ οι χρόνοι επίλυσης έχουν μικρή διαφοροποίηση για όλες τις τιμές του n .

- **Ερώτημα 12ο**

  Σε ένα μητρώο για να μπορεί να γίνει η παραγοντοποίηση Cholesky πρέπει να είναι συμμετρικό και θετικά ορισμένο. Είναι γνωστό, από την κατασκευή του, ότι το μητρώο είναι συμμετρικό και τα στοιχεία της διαγωνίου είναι θετικά ορισμένα. Για να ισχύει, ότι ένα μητρώο είναι διαγώνια κυρίαρχο, θα πρέπει να ισχύει: $|\alpha_{ii}| \geq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$. Μετά από πράξεις καταλήγουμε ότι το ζητούμενο a θα είναι 999.

  Ο συγκεκριμένος κώδικας είναι ο εξής:

```
a = 999;
n = 1000;
x = ones(n,1);
b = matrix_mv_5853(n,x);
A = matrix_build_5853(1000,a);
condition = condest(A);
[x_hat,flag,relres,iterations] = my_pcg(b,10^(-8),1000,n,a)
```

- **Ερώτημα 13ο**

  Για την διακριτοποίηση της διαφορικής εξίσωσης επιλέχθηκε πλέγμα n + 2 κόμβων για το διάστημα Δ = [0,1]. Είναι γνωστό ότι η απόσταση μεταξύ δυο κόμβων δίνεται από τον τύπο $h = \frac{1}{n+1}$. Κάθε κόμβος j απέχει από τον αρχικό κόμβο απόσταση ίση με $x_j = x_0 + j * h = j * h$.

  Από τη σειρά Taylor οι παράγωγοι προσεγγίζονται ως εξής:

$$U_{xx}(x) \approx \frac{u(x-h) - 2 * u(x) + u(x+h)}{h^2}$$

$$U_x\ (x) \approx \frac{u(x+h) - u(x-h)}{2 * h}$$

Θέτουμε $u(x_j) \rightarrow\ U_j$ και τα παραπάνω γίνονται:

$$U_{xx}\ (x) \approx \frac{U_{j-1} - 2 * U_j + U_{j+1}}{h^2}$$

$$U_x\ (x) \approx \frac{U_{j+1} - U_{j-1}}{2 * h}$$

Η αρχική μας διαφορική γίνεται :

$-(1 + x^2) * u_{xx} - 2 * x * u_x + 4 * u = 1 \rightarrow\ U_{j-1} * \left(-\frac{1}{h^2} - \frac{x_j^2}{h^2} + \frac{x_j}{h}\right) + U_j *$
$\left(\frac{2}{h^2} + \frac{2*x_j^2}{h^2} + 4\right) +\ U_{j+1} * \left(-\frac{1}{h^2} - \frac{x_j^2}{h^2} - \frac{x_j}{h}\right) = 1$

Ισχύει ότι $U_0\ = 0$ και $U_{n+1}\ = 1$ από τις συνοριακές συνθήκες , όπότε :

Για j =1 ισχύει $x_1 = \frac{1}{n+1}$:

$0 +\ U_1 * (2 * (n+1)^2 + 2 + 4\ ) +\ U_2 * (-(n+1)^2 - 1 - 1) = 1\ \rightarrow\ U_1 *$
$(2 * (n+1)^2 + 6) +\ U_2 * (-(n+1)^2 - 2) = 1$

Για j = k ισχύει $x_k = \frac{k}{n+1}$:

$-U_{k-1} * ((n+1)^2 +\ k^2 - k\ ) + U_k * (2 * (n+1)^2 +\ 2 * k^2 + 4) + U_{k+1} *$
$(-(n+1)^2 -\ k^2 - k\ ) = 1$

Για j = n ισχύει $x_n = \frac{n}{n+1}$:

$U_{n-1} * (-(n+1)^2 -\ n^2 + n\ ) + U_n * (2 * (n+1)^2 +\ 2 * n^2 + 4) + 1 *$
$(-(n+1)^2 -\ n^2 - n\ ) = 1$

- **Ερώτημα 14ο**

Για την υλοποίηση αυτού του ερωτήματος περιγράφουμε τις εξισώσεις του παραπάνω ερωτήματος στην μορφή $A * U = F$, όπου το A περιέχει τους συντελεστές του U και είναι τριδιαγώνιο και το F περιέχει τα δεύτερα μέλη των εξισώσεων. Ο αριθμός των iterations για αν επιτευχθεί η σύγκλιση είναι 993. Όμως παρατηρείται ότι ο κώδικας στο flag επιστρέφει την τιμή 1, που σημαίνει ότι τελικά η μέθοδος αποτυγχάνει.
Ο συγκεκριμένος κώδικας είναι ο εξής:

```
n = 1000;
A = sparse(n,n);
```

```
F = zeros(n,1);

for i = 1:n
    if i == 1
        A(1,1) = 2*(n+1)^2 + 6;
        A(1,2) = -(n+1)^2 - 2;
        F(1) = 1;
    elseif i == n
        A(n,n-1) = -(n+1)^2 - n^2 + n ;
        A(n,n) = 2*(n+1)^2 + 2*n^2 + 4;
        F(n) = 1 + (n+1)^2 + n^2 + n;
    else
        A(i,i-1) = - ((n+1)^2 + i^2 - i);
        A(i,i) = 2*(n+1)^2 + 2*i^2 + 4;
        A(i,i+1) = -(n+1)^2 - i^2 - i;
        F(i) = 1;
    end
end
[X,flag,relres,iterations] = pcg(A,F,[],1000);
```