# Μαθηματικό Λογισμικό

Εργασία Εξαμήνου

Αναστάσιος Φραγκόπουλος Α.Μ: 58633, Υπογραφή: .

Διδάσκοντας: Γεώργιος Α. Γραββάνης

# Περιεχόμενα



# 1. Εργασία

## *1.1. Πρόλογος*

Αυτή η αναφορά είναι οργανωμένη με τέτοιο τρόπο ώστε κάθε πρόβλημα να αναλύεται και να εξηγείται μαζί με τον κώδικα **MATLAB** που χρησιμοποιείται για την επίλυση του κάθε προβλήματος. Επίσης παρουσιάζονται μια σαφής και συνοπτική επεξήγηση της μεθοδολογίας που χρησιμοποιήθηκε και των αποτελεσμάτων που προέκυψαν.

Ο φάκελος της εργασίας είναι οργανωμένος με τον ακόλουθο τρόπο.

```
.
├── ask1/         # Φάκελος της άσκησης 1
├── ask2/         # Φάκελος της άσκησης 2
├── ask3/         # Φάκελος της άσκησης 3
├── ask4/         # Φάκελος της άσκησης 4
├── ask5/         # Φάκελος της άσκησης 5
├── ask6/         # Φάκελος της άσκησης 6
├── ask7/         # Φάκελος της άσκησης 7
├── img/          # Φάκελος με όλες της εικόνες που χρησιμοποιήθηκαν στο έγγραφο
├── Report.pdf    # Η αναφορά σε μορφή pdf
├── README.odt    # Η αναφορά σε μορφή odt
└── README.org    # Η αναφορά σε μορφή org
```

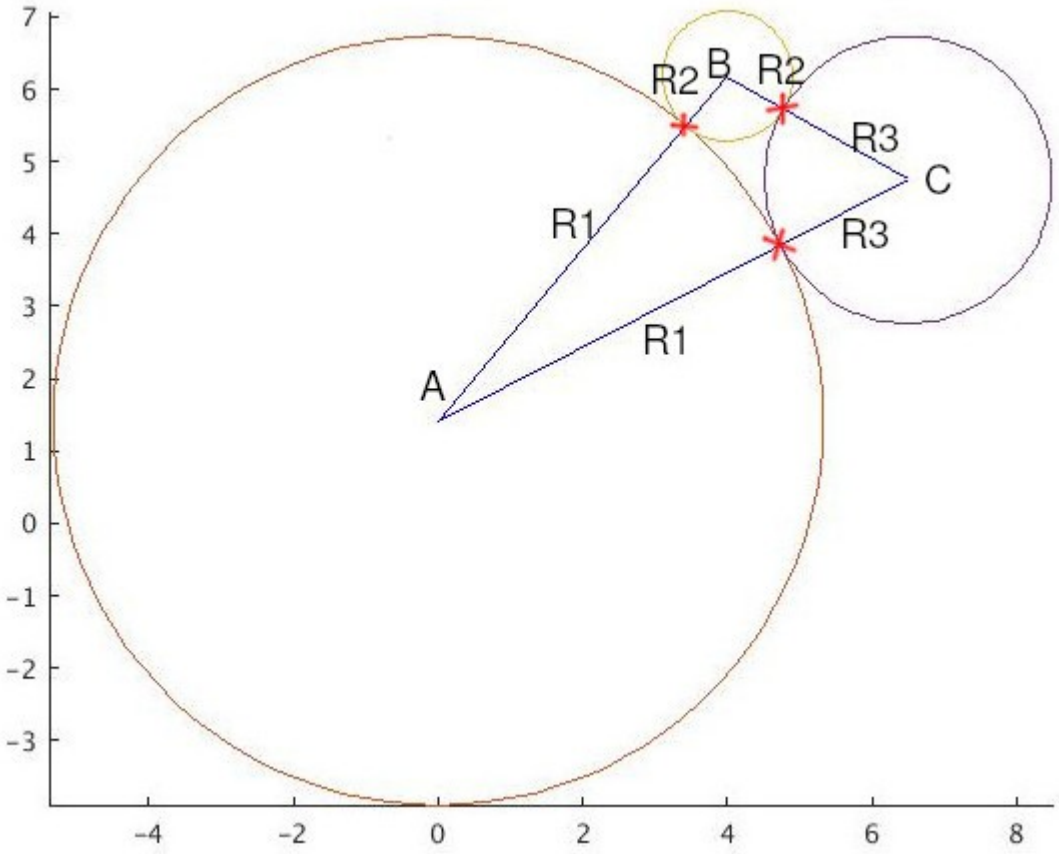
Στον κάθε φάκελο ask υπάρχουν όλα τα αρχεία με τον απαραίτητο κώδικα για την επίλυση της κάθε άσκησης.

Επίσης έχουμε τον φάκελο img που περιέχει όλες τις εικόνες που χρησιμοποιήθηκαν στο ακόλουθο έγγραφο.

## *1.2. Άσκηση 1*

### 1.2.1. Μέθοδος Επίλυσης

Για να λύσουμε την άσκηση θα πρέπει αρχικά να βρούμε τις ακτίνες κάθε κύκλου. Έστω κύκλοι A, B, C με κέντρα $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ και ακτίνες $R_1, R_2, R_3$ αντίστοιχα, όπως φαίνεται στην εικόνα παρακάτω.




Ξέροντας τα αυτά, η απόσταση μεταξύ κάθε κέντρου θα είναι:

$$AB = R_1 + R_2$$

$$AC = R_1 + R_3$$

$$BC = R_2 + R_3$$

Μπορούμε να λύσουμε για $R_1, R_2, R_3$ και να βρούμε ότι:

$$R_1 = \frac{AB+AC-BC}{2}$$

$$R_2 = \frac{AB-AC+BC}{2}$$

$$R_1 = \frac{-AB+AC+BC}{2}$$

Επιπλέων ξέρουμε ότι:

$$AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$AC = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2}$$

$$BC = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}$$

Έτσι υπολογίζοντας αρχικά τα $AB, AC, BC$ και κάνοντας τις πράξεις θα βρούμε τα $R_1, R_2, R_3$

### 1.2.2. Επίλυση στο Matlab

Η βασική επίλυση του προβλήματος γίνεται στο αρχείο `drawCircles.m` που έχει δύο συναρτήσεις, την `drawCircles(x, y)` και την `circle(x, y, r)`. Επίσης υπάρχει το αρχείο `example.m` που τρέχει την συνάρτηση `drawCircles` με x, y τις τιμές που δόθηκαν από την εκφώνηση ως παράδειγμα, την χρονομετρεί με την χρήση της συνάρτησης `timeit()` του Matlab και εκτυπώνει τον χρόνο.

Στη συνάρτηση `drawCircles(x, y)` αρχικά υπολογίζουμε τις τιμές AB, AC, BC.

```
AB = sqrt((x(2) - x(1))^2 + (y(2) - y(1))^2);
AC = sqrt((x(3) - x(1))^2 + (y(3) - y(1))^2);
BC = sqrt((x(3) - x(2))^2 + (y(3) - y(2))^2);
```

Ένας άλλος τρόπος να τις υπολογίσουμε θα ήταν να χρησιμοποιήσουμε την συνάρτηση `norm()`.

Στην συνέχεια φτιάχνουμε ένα διάνυσμα R, στο οποίο θα βάλουμε τις τιμές των $R_1, R_2, R_3$ που θα υπολογίσουμε.

```
R = [];
R(1) = ( AB + AC - BC)/2;
R(2) = ( AB - AC + BC)/2;
R(3) = (-AB + AC + BC)/2;
```

Τέλος κάνουμε plot τις ακμές του τριγώνου και τους κύκλους. Για τους κύκλους θα χρησιμοποιούμε την βοηθητική συνάρτηση `circle(x, y, r)` που βρίσκεται στο ίδιο αρχείο, η οποία παίρνει ως ορίσματα τις συντεταγμένες (x, y) του κέντρου του κύκλου και το μήκος της ακτίνας του r και κάνει plot τον κάθε κύκλο.

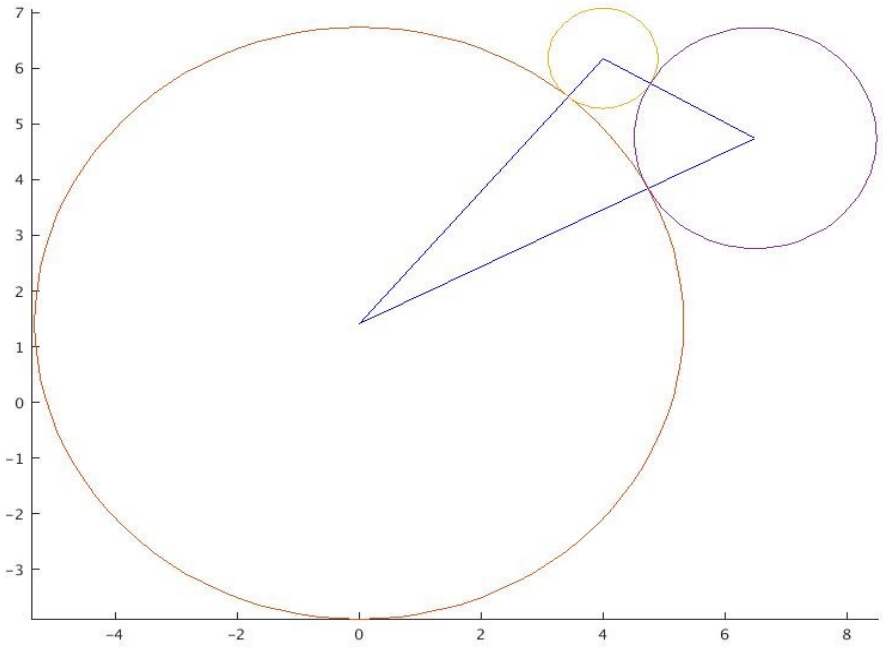
```
function circle(x, y, r)
% CIRCLE - This function plots a circle centered at (x, y) with radius r.
%   The input arguments x, y and r are the x and y-coordinates of the center of the
circle and its radius, respectively.
%   The function calculates the x and y-coordinates of 50 points evenly spaced
around the circumference of the circle,
%   using the parametric equation of a circle. The function then plots the circle
using the 'plot' function.

    th = 0:pi/50:2*pi;          % Theta range for 50 points
    xunit = r * cos(th) + x;    % x-coordinates of 50 points on circle
    yunit = r * sin(th) + y;    % y-coordinates of 50 points on circle
    plot(xunit, yunit);         % Plot the circle
end
```

Σε αυτή ορίζουμε το διάνυσμα th που έχει τις τιμές από 0 μέχρι $2\pi$ με βήμα $\frac{\pi}{50}$ και μετά υπολογίζουμε τις x και y συντεταγμένες του κύκλου στις xunit και yunit αντίστοιχα.

### 1.2.3. Αποτέλεσμα

Το αποτέλεσμα της συνάρτησης χρησιμοποιώντας τις τιμές που δίνονται από την εκφώνηση είναι το

γράφημα:




Και ο χρόνος που χρειάστηκε: 0.017586s

## *1.3. Άσκηση 2*

### 1.3.1. Επίλυση στο Matlab

Ο υπολογισμός του e με τον τύπο

$$e = \lim_{n\to\infty}(1 + \frac{1}{n})^n$$

γίνεται στο **matlab** με την ακόλουθη επανάληψη:

```
while(abs(em - e) > tol)
    e = (1 + 1/n)^n;      % approximate e using formula
    n = n * 2;            % double the value of n in each iteration
    count = count + 1;    % increment the iteration count
end
```

Η λούπα τρέχει όσο η απόλυτη διαφορά του e, που υπολογίζουμε, με του em (που είναι το e που δίνεται από το matlab `em = exp(1)`) είναι μεγαλύτερη του tol που είναι η ακρίβεια που θέλουμε και βγαίνει από `tol = 10^-(l + 1)`. Το n διπλασιάζεται σε κάθε, επανάληψη που είναι πολύ πιο αποτελεσματικό από την αύξηση του κατά ένα.

Αποτέλεσμα συνάρτησης με διπλασιασμό n:

```
>> approximate(9)
matlab's number:  2.718281828459
approximated number:    2.718281828380
accuracy to point:                ^
number of iterations:   35
```

Αποτέλεσμα συνάρτησης με αύξηση n κατα ένα:

```
>> approximate(9)
matlab's number:  2.718281828459
approximated number:    2.718281828360
accuracy to point:                ^
number of iterations:   66953800
```

### 1.3.2. Προβλήματα με την άσκηση

Το πρόβλημα με την συνάρτηση αυτή είναι ότι δεν δουλεύει για `l > 15`. Αυτό συμβαίνει γιατί το n γίνεται πολύ μεγάλο και έτσι το `1/n` είναι τόσο μικρό που το **Matlab** στην πρόσθεση του με το 1 αγνοεί τα δεκαδικά, με αποτέλεσμα το e να γίνει 1.

Μια λύση για να αντιμετωπίσουμε το πρόβλημα αυτό είναι να χρησιμοποιήσουμε την συνάρτηση `vpa()` του **Symbolic Math Toolbox** για να κάνουμε την πράξη. Βέβαια κάνει την συνάρτηση πολύ πιο αργή.

Ο πιο αποτελεσματικός τρόπος είναι να χρησιμοποιήσουμε τον τύπο:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

για να υλοποιήσουμε την συνάρτηση.

Στο αρχείο `fasterApproximation.m` υλοποιείται αυτός ο τύπος και μπορεί να υπολογίσει μέχρι 50 δεκαδικά του e με 19 επαναλήψεις.

## *1.4. Άσκηση 3*

### 1.4.1. Μέθοδος Επίλυσης

Για την λύση του προβλήματος αυτού θα χρειαστούμε να φτιάξουμε αρχικά ενα διάνυσμα 26 θέσεων. Κάθε θέση του διανύσματος θα αντιπροσωπεύει ένα γράμμα του λατινικού αλφαβήτου και θα περιέχει το πλήθος εμφάνισης του.

Θα το αρχικοποιήσουμε με μηδενικά

```
alphabet = zeros(1, 26);
```

Έπειτα θα διατρέξουμε κάθε χαρακτήρα της ακολουθίας χαρακτήρων και θα ελέγχουμε αν η τιμή ascii του είναι μεταξύ του 65 - 90 ή 97 - 122. Μεταξύ του 65 - 90 είναι οι τα κεφαλαία γράμματα και του 97 - 122 είναι τα μικρά. Έτσι διασφαλίζουμε ότι αγνοούμε όλους τους άλλους χαρακτήρες. Αφού ελέγξουμε τους χαρακτήρες κάνουμε την αναγκαία αφαίρεση για να βάλουμε τον κάθε χαρακτήρα στο διάνυσμα alphabet.

```
% Scan the input string `str`
for i = 1:strlength(str)

    % If a character in the string is between ASCII values 65 and 90 (upper case
letters),
    % its count is increased in the `alphabet` array at the corresponding index
(`str(i) - 64`).
    if(str(i) >= 65 && str(i) <= 90)
        alphabet(str(i) - 64) = alphabet(str(i) - 64) + 1;

    % If a character is between ASCII values 97 and 122 (lower case letters),
    % its count is increased in the `alphabet` array at the corresponding index
```

```
(`str(i) - 96`).
    elseif (str(i) >= 97 && str(i) <= 122)
        alphabet(str(i) - 96) = alphabet(str(i) - 96) + 1;
    end
end
```

Τέλος διατρέχουμε το διάνυσμα alphabet και εκτυπώνουμε τα γράμματα και πόσες φορές εμφανίζονται.

```
% Output the number of occurrences of each letter in the string
for i = 1:26
    if(alphabet(i) ~= 0)
        fprintf("Found %c: %d times \n", (i+96), alphabet(i));
    end
end
```

Παράδειγμα χρήσης συνάρτησης για ακολουθία χαρακτήρων 'Hello World':

```
>> findOccurances('Hello World')
Found d: 1 times
Found e: 1 times
Found h: 1 times
Found l: 3 times
Found o: 2 times
Found r: 1 times
Found w: 1 times
```

## *1.5. Άσκηση 4*

### 1.5.1. Μέθοδος Επίλυσης

Για να φτιάξουμε το αυτόματο μηχάνημα πρέπει να κάνουμε τρία βασικά πράγματα σε ολό το πρόγραμμα:

- Να εκτυπώσουμε στον χρήστη τις επιλογές που μπορεί να κάνει
- Να διαβάσουμε την επιλογή αυτή
- Να έχουμε κάποια λογική πάνω στις επιλογές του χρήστη

Οπότε θα χρειαστεί να τα εφαρμόσουμε για την επιλογή εισιτηρίου, την επιλογή αριθμού εισιτηρίων και την επιλογή κερμάτων για προσθήκη.

Αρχικά θα φτιάξουμε την επιλογή των εισιτηρίων. Θα έχουμε ένα `while loop` που θα τρέχει για πάντα και θα εκτυπώνει τα εισιτήρια που μπορεί να επιλέξει ο χρήστης, θα παίρνει την επιλογή και θα ελέγχει άμα είναι έγκυρη για να σταματήσει. Διαφορετικά θα εκτυπώνει μήνυμα στο χρήστη για μη έγκυρη επιλογή και θα ξανά τρέχει.

```
% Get user's choice of ticket
while(1)
    % Output ticket options
    fprintf("(1): Standar  1.40€.\n");
    fprintf("(2): Discount 0.60€.\n");
    fprintf("(3): Daily    4.50€.\n");
    fprintf("(4): Weekly   9.00€.\n");

    % Get user's ticket choise
    ticketChoice = str2double(input('Choose your ticket from the list.(1 - 4): ',
's'));

    % Check if valid ticket choise
    if(ticketChoice >= 1 & ticketChoice <= 4)
        break;
    else
        % Output error if choise not valid
        fprintf("\nWrong option. Please choose one of this options:\n");
```

```
    end
end
```

Όταν παίρνουμε την επιλογή από το χρήστη δίνουμε στην συνάρτηση `input()` ως δεύτερο όρισμα το 's' για να πάρει την είσοδο του χρήστη ως ακολουθία χαρακτήρων και να μην κάνει καμία πράξη με αυτή. Έπειτα χρησιμοποιούμε την `str2double()` για να κάνουμε την ακολουθία αριθμό.

Μετά θα κάνουμε κάτι παρόμοιο για να την επιλογή αριθμού εισιτηρίων.

```
% Get user's number of tickets
while(1)
    % Get user's choise
    numberOfTickets = str2double(input('How many tickets do you want? ', 's'));

    % Check if choise is valid
    if(numberOfTickets > 0)
        break
    else
        % Output error if choise not valid
        fprintf("\nNot a valid number of tickets. Please put a valid number.\n");
    end
end
```

Και παρόμοιο για την είσοδο των χρημάτων.

```
while(1)
    % Output price owed and available coins
    fprintf("You owe %.2f€. You can insert:\n", priceDifference);
    fprintf("\t(1): 10 euro note\n");
    fprintf("\t(2): 5 euro note\n");
    fprintf("\t(3): 2 euro coin\n");
    fprintf("\t(4): 1 euro coin\n");
    fprintf("\t(5): 50 cent euro coin\n");
    fprintf("\t(6): 20 cent euro coin\n");
    fprintf("\t(7): 10 cent euro coin\n");
    fprintf("\t(8): 5 cent euro coin\n");

    % Get user's inserted money
    moneyInserted = str2double(input('Choose money to insert.(1 - 8): ', 's'));

    % Check if choice is valid
    if(moneyInserted >= 1 & moneyInserted <= 8)
        priceDifference = round(priceDifference - money(moneyInserted), 2);

        if(~priceDifference)
            break;
        elseif(priceDifference < 0)
            fprintf("You inserted more money. Here's the difference:\n");
            priceDifference = - priceDifference;

            for i = 1:length(money)
                if(money(i) == priceDifference)
                    fprintf("\tYou got %.2f€.\n", money(i));
                    priceDifference = round(priceDifference - money(i), 2);
                    break;
                elseif(priceDifference > money(i))
                    fprintf("\tYou got %.2f€.\n", money(i));
                    priceDifference = round(priceDifference - money(i), 2);
                end
            end
            break;
        end
    else
        % Output error if choice not valid
        fprintf("\nWrong option. Please choose one of this options:\n");
    end
```

```
end
```

Πριν την επανάληψη αυτή, βρίσκουμε πόσα χρήματα χρωστάει ο χρήστης που βγαίνει από την τιμή του εισιτηρίου επί τον αριθμό τους. Τις τιμές των εισιτηρίων και τα επιτρεπόμενα χρήματα τα έχουμε σε ένα διάνυσμα.

```
% List of ticket prices
tickets = [1.40 0.6 4.5 9];
% List of accepted coins
money = [10 5 2 1 0.5 0.2 0.1 0.05];

% Find price user has to pay
priceDifference =  tickets(ticketChoice) * numberOfTickets;
```

Για την τελευταία επανάληψη ελέγχουμε αν τα λεφτά που δόθηκαν είναι ίσα με τα οφειλόμενα για να τελειώσουμε το πρόγραμμα. Αν είναι λιγότερα το πρόγραμμα ξανά τρέχει και ο χρήστης μπορεί να βάλει και άλλα χρήματα στο μηχάνημα. Αν βάλει περισσότερα τότε πρέπει το μηχάνημα να δώσει ρέστα. Οπότε διατρέχουμε το διάνυσμα με τα χρήματα από το μεγαλύτερο προς το μικρότερο. Αφαιρούμε από τα λεφτά που χρωστάει το μηχάνημα τα λεφτά της κάθε επανάληψης και ελέγχουμε αν είναι ίσα με το χρήμα της κάθε επανάληψης για να τελειώσει το πρόγραμμα ή αν είναι μεγαλύτερο για να συνεχίσει τις επαναλήψεις.

Είναι σημαντικό να αναφέρουμε πως κάθε φορά που κάνουμε αφαίρεση των χρημάτων χρησιμοποιούμε την συνάρτηση `round()` για στρογγυλοποιήσουμε στα δύο δεκαδικά γιατί οι πράξεις με δεκαδικούς στο **Matlab** δεν είναι ακριβείς και για αυτό δεν μπορούμε να πάρουμε μηδέν[1].

## *1.6. Άσκηση 5*

### 1.6.1. Επίλυση άσκησης

Για την άσκηση 5 έχουμε μία συνάρτηση, την `rlc()`, που υπολογίζει όλες τις τιμές που ζητούνται και εκτυπώνει τις τιμές και τα γραφήματα που χρειάζονται.

Αρχικά θα χρειαστεί να θέσουμε όλες τις παραμέτρους του κυκλώματος που δίνονται.

```
% Define the circuit parameters
R = 2500;
L = 5e-3;
C = 10e-6;
V = 310;

% Define angular frequency range
omega = [-20000*pi:20000*pi];
```

Επιπλέον θα χρειαστεί να ορίσουμε τις συναρτήσεις με τους τύπους που δίνονται από την εκφώνηση.

```
% Define functions with the formulas given for the Z
Zf = @(x) sqrt(R^2 + (x*L - 1./(x*C)).^2);
phif = @(x) atan((x*L - 1./(x*C))/ R);

% Define functions with the formulas given for the voltages
VRf = @(x) (V*R)./Zf(x);
VLf = @(x) (V*L*x)./Zf(x);
VCf = @(x) V.*(1./((x*C).*Zf(x)));
```

Έχοντας αυτά μπορούμε τώρα να βρούμε τις τιμές που ζητούνται.

Για την σύνθετη αντίσταση:

```
% Calculate the impedance magnitude and angle
Z = Zf(omega);
phi = phif(omega);
```

Για τις τάσεις:

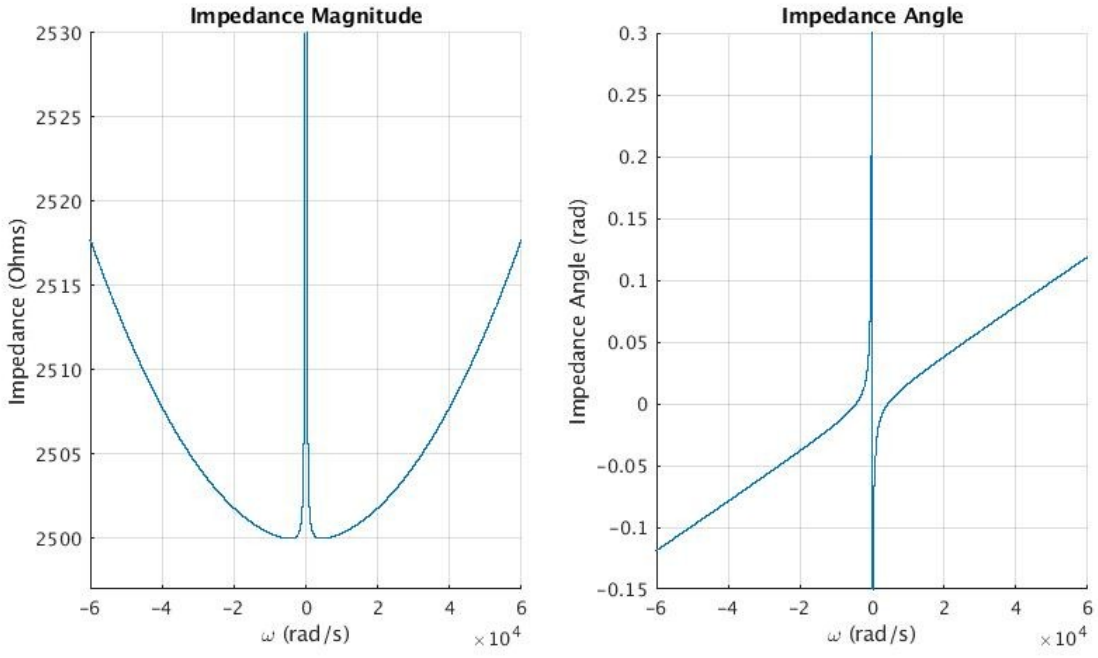
```
% Calculate the voltages across R, L and C
VR = VRf(omega);
VL = VLf(omega);
VC = VCf(omega);
```

Τώρα μπορούμε να κάνουμε τα γραφήματα για τις τιμές.

Οπότε για την σύνθετη αντίσταση:

```
% Plot impedance magnitude and angle
figure;
subplot(1,2,1);
hold on
ylim([2497 2530])
xlim([-6e4 6e4])
plot(omega, Z);
title('Impedance Magnitude');
xlabel('\omega (rad/s)');
ylabel('Impedance (Ohms)');
grid on;
hold off

subplot(1,2,2);
hold on
ylim([-0.15 0.3])
xlim([-6e4 6e4]);
plot(omega, phi);
title('Impedance Angle');
xlabel('\omega (rad/s)');
ylabel('Impedance Angle (rad)');
grid on;
hold off
```

έχουμε

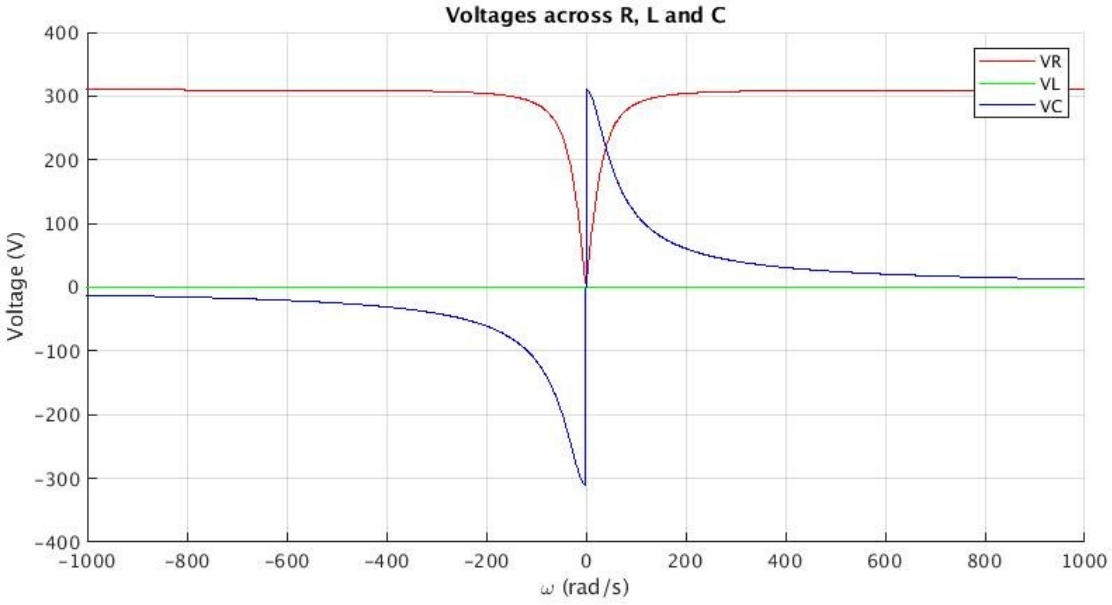



και για τις τάσεις:

```
% Plot voltages
figure;
hold on
ylim([-400 400]);
xlim([-1000 1000]);
plot(omega, VR, 'r', omega, VL, 'g', omega, VC, 'b');
```

```
legend('VR', 'VL', 'VC');
title('Voltages across R, L and C');
xlabel('\omega (rad/s)');
ylabel('Voltage (V)');
grid on;
hold off
```

έχουμε




Τέλος θα βρούμε την γωνία συντονισμού $\omega_r$. Αυτό θα το κάνουμε με βρίσκοντας τον δείκτη της VL όταν αυτή είναι μέγιστη με την συνάρτηση `max()` και χρησιμοποιώντας τον στο διάνυσμα του omega. Έπειτα μπορούμε να βρούμε τις τάσεις με τον δείκτη και θα εκτυπώσουμε τα αποτελέσματα.

```
% Find resonant frequency
[~, idx] = max(abs(VL));
omega_r = omega(idx);

% Calculate voltages at resonant frequency
Z_r = Z(idx);
VR_r = VR(idx);
VL_r = VL(idx);
VC_r = VC(idx);

% Display results
fprintf("Resonant Angular Frequency: %.4f rad/s\n", omega_r);
fprintf("Voltage Across R at Resonance: %.4f V\n", VR_r);
fprintf("Voltage Across L at Resonance: %.4f V\n", VL_r);
fprintf("Voltage Across C at Resonance: %.4f V\n", VC_r);
```

Αποτελέσματα:

```
>>rlc
Resonant Angular Frequency: -62831.8531 rad/s
Voltage Across R at Resonance: 307.6051 V
Voltage Across L at Resonance: -38.6548 V
Voltage Across C at Resonance: -0.1958 V
```

## *1.7. Άσκηση 6*

### 1.7.1. Μέθοδος Επίλυσης

Για να εφαρμόσουμε την μέθοδο ορθοκανονικοποιήσης Gram-Schmidt με κώδικα χρειάζεται απλά να εφαρμόσουμε τα βήματα του αλγορίθμου που δίνονται στην εκφώνηση και να υλοποιήσουμε τις μαθηματικές πράξεις που δίνονται.

Αρχικά θα χρειαστεί να αρχικοποιήσουμε τον πίνακα U, πού θα είναι το αποτέλεσμα της συνάρτησης, με μηδενικά. Οι διαστάσεις του θα είναι ίδιες με τον πίνακα V που παίρνουμε σαν είσοδο.

```
[m, n] = size(V);% Get the dimensions of the input matrix V
U = zeros(m, n);  % Initialize the output matrix U with zeros
```

Κάθε στήλη του πίνακα V αποτελεί ένα διάνυσμα $a_1, a_2, a_3, ..., a_n$. Το αντίστοιχο ισχύει για τον πίνακα U. Οπότε για το βήμα 1 του αλγορίθμου θα θέσουμε $u_1 = a_1$ έτσι:

```
U(:, 1) = V(:, 1);      % The first column of U is equal to the first column of V
```

Για το βήμα 2 θα έχουμε μία λούπα που σε κάθε επανάληψη θα υπολογίζει κάθε διάνυσμα. Θα τρέχει (n - 1) φορές γιατί έχουμε ήδη βρει το πρώτο διάνυσμα. Μέσα στην λούπα θα υλοποιήσουμε τον τύπο:

$$u_{i+1} = a_{i+1} - \sum_{k=1}^{i} \frac{\langle a_{i+1}, u_k \rangle}{\|u_k\|^2} u_k$$

**<u>Σημείωση</u>**:Υπάρχει λάθος στον τύπο της εκφώνησης. Μέσα στο άθροισμα όλα τα u θα έχουν δείκτη k και όχι i. Αλλιώς θα προσθέτουμε i φορές το ίδιο πράγμα[2].

Πρώτα θα υπολογίσουμε το εσωτερικό του αθροίσματος με την βοηθητική συνάρτηση `proj(v, u)`:

```
function [ res ] = proj(v, u)
% PROJ - This function calculates the projection of a vector 'v' onto another vector
'u'.
% Input:
%   v -> m x 1 vector in R^m
%   u -> m x 1 vector in R^m
%
% Output:
%   res - m x 1 vector, the projection of 'v' onto 'u'
%==========================================================================

    dotProduct = (v)'*u;      % Calculate the dot product of 'v' and 'u'
    normSquared = (u)'*u;     % Calculate the squared norm of 'u'

    % Calculate the projection of 'v' onto 'u'
    res = (dotProduct / normSquared) * u;
end
```

Για να υπολογίσουμε το εσωτερικό γινόμενο θα κάνουμε πολλαπλασιασμό ανάστροφου πίνακα με πίνακα, που είναι πολύ πιο αποτελεσματικό και γρήγορο από το να βρίσκαμε το άθροισμα όλων των γινομένων στοιχείων προς στοιχείο των δύο πινάκων. Αυτό γιατί οι ενσωματωμένοι τελεστές `'` και `*` πινάκων είναι πιο αποτελεσματικοί[3]. Για το τετράγωνο της νόρμας του $u_i$ άμα κάνουμε τις απλοποιήσεις μπορούμε να καταλήξουμε:

$$\|u_i\|^2 = \sqrt{u_1^2 + u_2^2 + ... + u_n^2}^2 = u_1^2 + u_2^2 + ... + u_n^2 = u \cdot u$$

Άρα μπορούμε να βρούμε το εσωτερικό γινόμενο του u με το u με τον ίδιο τρόπο. Τέλος στην συνάρτηση θα κάνουμε την διαίρεση μεταξύ τους και τον πολλαπλασιασμό με το u και θα το επιστρέψουμε.

Επιστρέφοντας στην βασική συνάρτηση θα βρούμε το άθροισμα μέσω μίας επανάληψης από k = 1 μέχρι i, όλο αυτο μέσα στην βασική επανάληψη που αναφέρθηκε παραπάνω.

```
% Loop through the remaining columns of V
for i=1:n-1
    s = 0; % Initialize the variable 's' to zero

    % Loop through the columns of U that have been computed so far
    for k = 1:i
        % Add the projection of V(:, i+1) onto U(:, k) to 's'
        s = s + proj(V(:, i+1), U(:, k));
    end

    % The next column of U is equal to the next column of V minus the projection
    U(:, i+1) = V(:, i+1) - s;
end
```

Στο τέλος κάθε επανάληψης θα κάνουμε την αφαίρεση του $a_{i+1}$ με το άθροισμα που υπολογίσαμε.

```
% The next column of U is equal to the next column of V minus the projection
U(:, i+1) = V(:, i+1) - s;
```

### 1.7.2. Αποτελέσματα

Εκτός από το αρχείο `gramSchmidt.m` της βασικής συνάρτησης υπάρχει το αρχείο `example.m` που τρέχει την συνάρτηση με δοκιμαστική βάση $R^3$ που δίνεται από την εκφώνηση, καθώς και υπολογίζει των χρόνο που χρειάστηκε για να τρέξει μέσω της συνάρτησης `timeit()`.

Αν το τρέξουμε λοιπόν παίρνουμε ως αποτέλεσμα:

```
>> example
The function took 0.000058s to run.

Ouput for base R3
     3    -4     0
     0     0     9
     4     3     0
```

## *1.8. Άσκηση 7*

### 1.8.1. Μέθοδος Επίλυσης

Οι σχέσεις x και y που δίνονται στην εκφώνηση βγαίνουν από την πράξη[5]

$$p_{n+1} = p_n - \frac{F}{J},$$

όπου p, F πίνακες $p = \begin{bmatrix} x & y \end{bmatrix}, F = \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix}$ και J ο Ιακωβιανός πίνακας του F

$$J = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial y} \\ \frac{\partial g(x,y)}{\partial x} & \frac{\partial g(x,y)}{\partial y} \end{bmatrix}$$

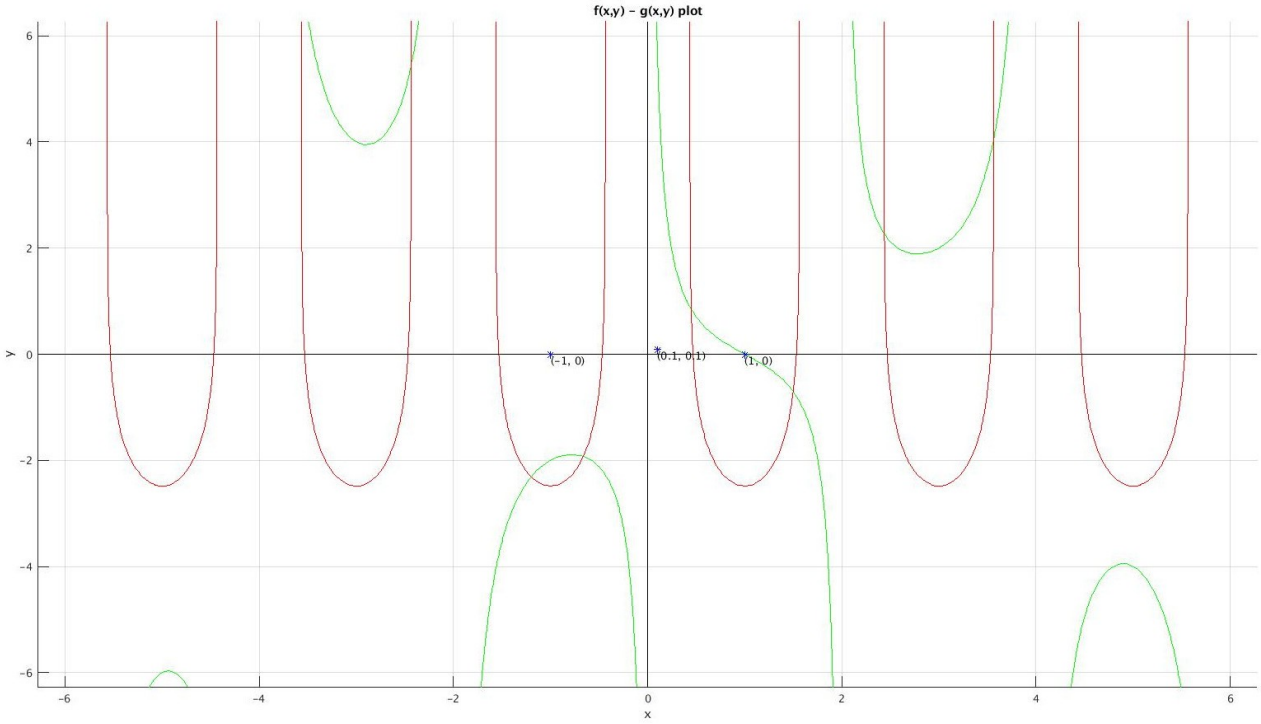
Αν κάνουμε τις πράξεις:

$$p_n - \frac{1}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \begin{bmatrix} \frac{\partial g}{\partial y} & -\frac{\partial f}{\partial y} \\ -\frac{\partial g}{\partial x} & \frac{\partial f}{\partial x} \end{bmatrix} \begin{bmatrix} f(x,y) \\ g(x,y) \end{bmatrix} = p_n - \frac{1}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \begin{bmatrix} f\frac{\partial g}{\partial y} - g\frac{\partial f}{\partial y} \\ g\frac{\partial f}{\partial x} - f\frac{\partial g}{\partial x} \end{bmatrix} \Rightarrow$$

$$p_{n+1} = \begin{bmatrix} x_n - \frac{f\frac{\partial g}{\partial y} - g\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \\ y_n - \frac{g\frac{\partial f}{\partial x} - f\frac{\partial g}{\partial x}}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \end{bmatrix} \Rightarrow \begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n - \frac{f\frac{\partial g}{\partial y} - g\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \\ y_n - \frac{g\frac{\partial f}{\partial x} - f\frac{\partial g}{\partial x}}{\frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial g}{\partial x}\frac{\partial f}{\partial y}} \end{bmatrix}$$

Θα χρησιμοποιήσουμε λοιπόν πίνακες συναρτήσεων αντί για ξεχωριστές συναρτήσεις γιατί θα είναι πιο εύκολο να τους χειριστούμε και να κάνουμε πράξεις με αυτούς στο matlab.

Επιπλέον για την μέθοδο θα χρειαστούμε μια αρχική προσέγγιση. Για να την βρούμε θα κάνουμε τα γραφήματα των δύο συναρτήσεων και θα βρούμε τιμές κοντά εκεί που τέμνονται οι παραστάσεις.




Στο γράφημα έχουν σημειωθεί μερικές αρχικές προσεγγίσεις. Οι (-1, 0), (0.1, 0.1) (1, 0).

## 1.8.2. Μέθοδος Επίλυσης στο Matlab

Η επίλυση του συγκεκριμένου προβλήματος θα γίνεται στο αρχείο `run.m` καθώς υπάρχει και αρχείο `newtonRaphson.m` που περιέχει την συνάρτηση που εφαρμόζει την μέθοδο Newton-Raphson. Ξεκινώντας θα θέσουμε λοιπόν τους πίνακες συναρτήσεων:

```
% Matix with functions f and g inside
f = @(x) [
    1-x(1)-x(2)*sin(pi*x(1)/2);   % f(x, y) - First equation
    exp(-x(2))+10*cos(pi*x(1))-2  % g(x, y) - Second equation
         ];

% Jacobian matrix of matrix f
df = @(x) [
    -1-x(2)*cos(pi*x(1)/2)*(pi/2),   -sin(pi*x(1)/2);
    -10*pi*sin(pi*x(1)),             -exp(-x(2))
          ];
```

Έπειτα θα θέσουμε τις αρχικές προσεγγίσεις, την ακρίβεια που θέλουμε και των αριθμό των επαναλήψεων.

```
tol = 1e-7; % Tolerance value for the Newton-Raphson method

% Different initial guess values and number of iterations.
x0 = [-1 0]; n = 16;
% x0 = [0.1 0.1]; n = 20;
% x0 = [1 0]; n = 18;
```

Ο χρήστης μπορεί να βάλει μια από αυτές τις προσεγγίσεις βγάζοντας την από τα σχόλια.

Μετά θα καλέσουμε την συνάρτηση `newtonRaphson()` και θα εκτυπώσουμε τα αποτελέσματα.

```
[res, times] = newtonRaphson(f, df, x0, n, tol); % Call to the Newton-Raphson method

fres = f(res); % Evaluate the function at the obtained result

% Display the result and the number of iterations.
fprintf("For f(x, y) = 0 and g(x, y) = 0 using the method Newton-Raphson that run %d
times, we found that\n\tx0 = %.6f,\n\ty0 = %.6f\n", times, res(1), res(2));
fprintf("f(x0, y0) = %.6f\ng(x0, y0) = %.6f\n", fres(1), fres(2));
```

### 1.8.3. Συνάρτηση newtonRaphson

Η συνάρτηση παίρνει ως ορίσματα τους πίνακες f, df, την αρχική προσέγγιση x0, τον αριθμό επαναλήψεων n και την ακρίβεια tol. Επιστρέφει τον πίνακα res που περιέχει τις ρίζες (x, y) και τον αριθμό των επαναλήψεων i που έκανε.

Αρχικοποιούμε την τον πίνακα res και θα κάνουμε την επανάληψη που θα τρέχει n φορές.

```
% Initialize result
res = [0 0];

% Perform Newton-Raphson iteration
for i = 1:n
    % Compute the inverse Jacobian and the residual of the system
    % invJ = inv(df(x0));
    invJ = inverse(df(x0));
    F = f(x0);
    res = x0 - (invJ*F)';

    % Check if the solution has converged
    if(abs(res - x0) < tol)
        break; % Solution has converged
    end

    % Update the guess for the next iteration
    x0 = res;
end
```

Στην αρχή τις επανάληψης υπολογίζουμε τον αντίστροφο πίνακα του `df(x0)` και τον πίνακα `f(x0)`. Κάνουμε την πράξη και την βάζουμε στο res. Χρειάζεται να πάρουμε τον ανάστροφο του πίνακα που προκύπτει από τον πολλαπλασιασμό του `injJ*F` για να έχει ίδιες διαστάσεις με το x0. Τέλος ελέγχουμε αν η διαφορά του res με το x0 είναι μικρότερη από το tol για να σταματήσουμε την επανάληψη και θέτουμε x0 ίσο με το res. Αν τελειώσει η επανάληψη και το i είναι ίσο με το n σημαίνει πως η μέθοδος απέτυχε να συγκλίνει οπότε θα το εκτυπώσουμε.

```
% Check if the Newton-Raphson method failed to converge
if(i == n)
    fprintf("Newton-Raphson method failed to converge.\n");
end
```

Επιπλέον έχουμε την βοηθητική μέθοδο `inverse()` που υπολογίζει το αντίστροφο πίνακα διαστάσεων 2x2 του πίνακα που του δίνεται.

```
function [B] = inverse(A)
% INVERSE - Find the inverse of a 2x2 matrix
```

```
% A: 2x2 matrix to find the inverse of
% B: Inverse of A

    detA = A(1, 1)*A(2, 2) - A(1, 2)*A(2, 1);  % Find the determinant of A
    adjA = [A(2, 2) -A(1, 2); -A(2, 1) A(1, 1)];  % Find the adjugate of A
    B = (1/detA) * adjA;                % Divide the adjugate of A by its
determinant to get the inverse of A
end
```

Υπολογίζει πρώτα την ορίζουσα του πίνακα A και τον συμπληρωματικό του πίνακα adj(A) και τέλος θέτει και επιστρέφει πίνακα B που έχει τις πράξεις.

### 1.8.4. Αποτέλεσμα

Εαν τρέξουμε το αρχείο `run.m` θα πάρουμε ως αποτέλεσμα:

για x0 = [-1 0]

```
>> run
For f(x, y) = 0 and g(x, y) = 0 using the method Newton-Raphson that run 15 times,
we found that
        x0 = 1.499230,
        y0 = -0.705166
f(x0, y0) = -0.000000
g(x0, y0) = 0.000000
The function took 0.000081s to run.
```

για x0 = [0.1 0.1]

```
>> run
For f(x, y) = 0 and g(x, y) = 0 using the method Newton-Raphson that run 7 times, we
found that
        x0 = 1.499230,
        y0 = -0.705166
f(x0, y0) = -0.000000
g(x0, y0) = 0.000000
The function took 0.000076s to run.
```

για x0 = [1 0]

```
>> run
For f(x, y) = 0 and g(x, y) = 0 using the method Newton-Raphson that run 17 times,
we found that
        x0 = 14.435906,
        y0 = 21.244032
f(x0, y0) = 0.000000
g(x0, y0) = -0.000000
The function took 0.000066s to run.
```

## *1.9. Αναφορές*


[1] Stiff, J., & Somani, S. (n.d.). FAQ | MATLAB Wiki | Fandom. MATLAB Wiki. Retrieved February 2, 2023, from https://matlab.fandom.com/wiki/FAQ#Why_is_0.3_-_0.2_-_0.1_(or_similar)_not_equal_to_zero?

[2] Harvey Mudd College. (2019). Gram-Schmidt Method. Gram-Schmidt Method – Calculus Tutorials. Retrieved February 6, 2023, from https://math.hmc.edu/calculus/hmc-mathematics-calculus-online-tutorials/linear-algebra/gram-schmidt-method/

[3] Mathworks. (2012, September 19). Vectorization - MATLAB & Simulink. MathWorks. Retrieved February 6, 2023, from https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

[4] Chakraborty, A. (2020, February 27). Nonlinear equations. Retrieved February 7, 2023, from https://www.isical.ac.in/~arnabc/numana/nonlin1.html