Assignment 4

Colin Turner: 250956100

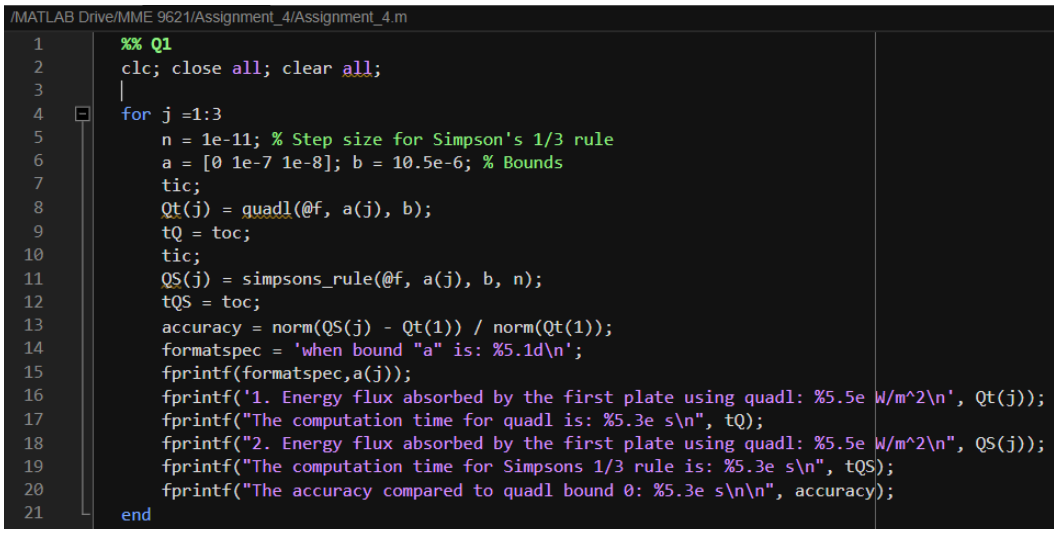
MME 9621: Computational Methods in Mechanical Engineering

Dr. Mohammad Z. Hossain

March 29, 2024

Q1.

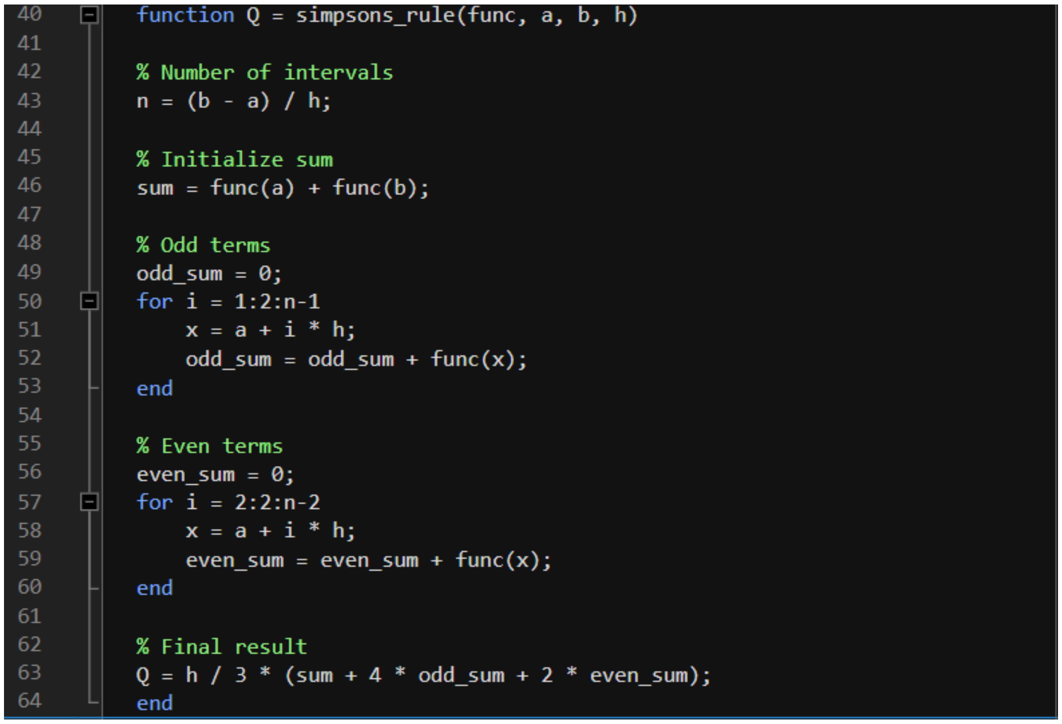
The following figure depicts the main body of the code, defining multiple initial bounds to test accuracy as we approach zero. This is necessary as there is a singularity at zero and results in NAN for Simpsons method if 0 is used.



/MATLAB Drive/MME 9621/Assignment_4/Assignment_4.m

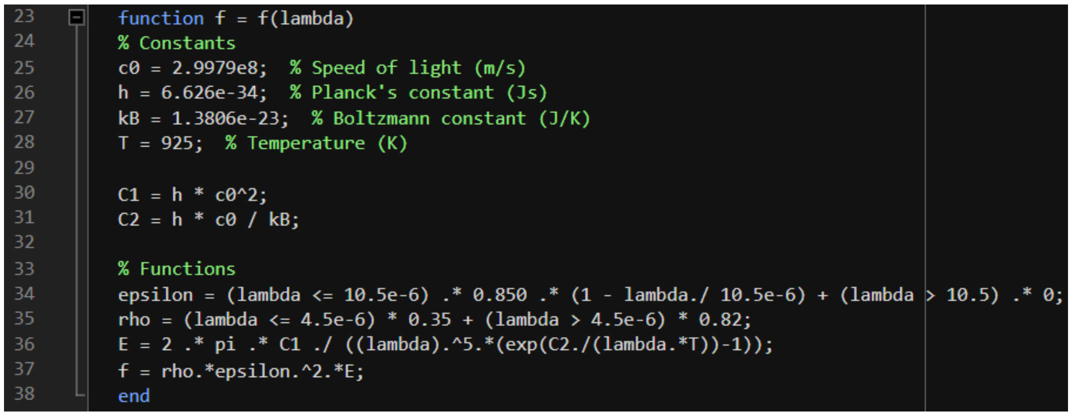
```matlab
%% Q1
clc; close all; clear all;

for j =1:3
    n = 1e-11; % Step size for Simpson's 1/3 rule
    a = [0 1e-7 1e-8]; b = 10.5e-6; % Bounds
    tic;
    Qt(j) = quadl(@f, a(j), b);
    tQ = toc;
    tic;
    QS(j) = simpsons_rule(@f, a(j), b, n);
    tQS = toc;
    accuracy = norm(QS(j) - Qt(1)) / norm(Qt(1));
    formatspec = 'when bound "a" is: %5.1d\n';
    fprintf(formatspec,a(j));
    fprintf('1. Energy flux absorbed by the first plate using quadl: %5.5e W/m^2\n', Qt(j));
    fprintf("The computation time for quadl is: %5.3e s\n", tQ);
    fprintf("2. Energy flux absorbed by the first plate using quadl: %5.5e W/m^2\n", QS(j));
    fprintf("The computation time for Simpsons 1/3 rule is: %5.3e s\n", tQS);
    fprintf("The accuracy compared to quadl bound 0: %5.3e s\n\n", accuracy);
end
```

The following code depicts the function used for simpsons 1/3 method where sum is F(0) + F(n), sum odd is F(1)+F(3)…+F(n-1) and sum even is F(2)+F(4)…+F(n-2).



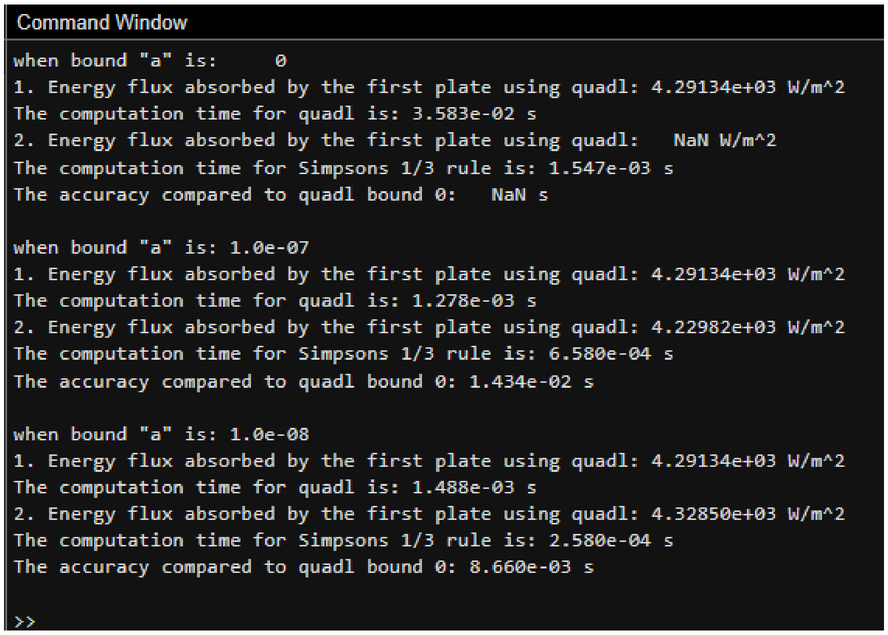
```matlab
function Q = simpsons_rule(func, a, b, h)

% Number of intervals
n = (b - a) / h;

% Initialize sum
sum = func(a) + func(b);

% Odd terms
odd_sum = 0;
for i = 1:2:n-1
    x = a + i * h;
    odd_sum = odd_sum + func(x);
end

% Even terms
even_sum = 0;
for i = 2:2:n-2
    x = a + i * h;
    even_sum = even_sum + func(x);
end

% Final result
Q = h / 3 * (sum + 4 * odd_sum + 2 * even_sum);
end
```

The following code depicts the equations' function which outlines the given equations and utilizing if statements follow the parameters of the problem.

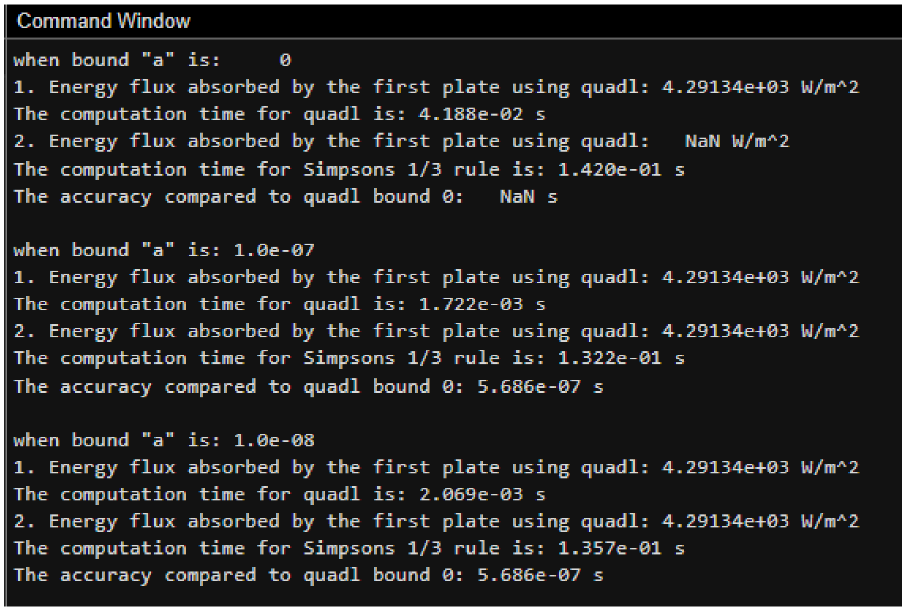


```matlab
function f = f(lambda)
% Constants
c0 = 2.9979e8;  % Speed of light (m/s)
h = 6.626e-34;  % Planck's constant (Js)
kB = 1.3806e-23;  % Boltzmann constant (J/K)
T = 925;  % Temperature (K)

C1 = h * c0^2;
C2 = h * c0 / kB;

% Functions
epsilon = (lambda <= 10.5e-6) .* 0.850 .* (1 - lambda./ 10.5e-6) + (lambda > 10.5) .* 0;
rho = (lambda <= 4.5e-6) * 0.35 + (lambda > 4.5e-6) * 0.82;
E = 2 .* pi .* C1 ./ ((lambda).^5.*(exp(C2./(lambda.*T))-1));
f = rho.*epsilon.^2.*E;
end
```

The following two command windows depict the outputs of this code when step size is 1e-6 and 1e-11 respectively. We can see that quadl can solve the problem when the initial bound is set to zero, however Simpsons method yields NAN as there is a singularity at zero. This results in additional simulations to be run with the initial bounds set close to zero instead of zero. The further from zero that is used the less accurate the solution becomes for both methods. Additionally, when step size is set to 1e-6 the solution becomes significantly less accurate yielding a 1e-2 and 1e-3 level of accuracy when compared to the baseline of quadl with initial bound 0. When step size 1e-11 is utilized, the solution becomes significantly more accurate yielding a 1e-7 level of accuracy when compared to the baseline. Finally, when comparing the computational time it can be seen that quadl yields significantly faster results when both step sizes are used and through all initial bounds defined. Given these results it is recommended that quadl be used over Simpsons method for problems similar to this given problem.



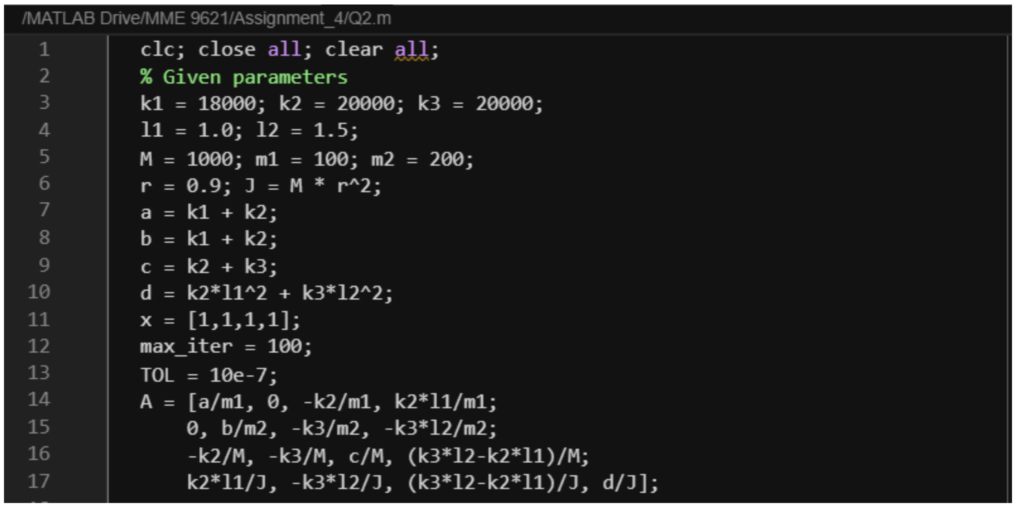
```
when bound "a" is:     0
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 3.583e-02 s
2. Energy flux absorbed by the first plate using quadl:   NaN W/m^2
The computation time for Simpsons 1/3 rule is: 1.547e-03 s
The accuracy compared to quadl bound 0:   NaN s

when bound "a" is: 1.0e-07
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 1.278e-03 s
2. Energy flux absorbed by the first plate using quadl: 4.22982e+03 W/m^2
The computation time for Simpsons 1/3 rule is: 6.580e-04 s
The accuracy compared to quadl bound 0: 1.434e-02 s

when bound "a" is: 1.0e-08
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 1.488e-03 s
2. Energy flux absorbed by the first plate using quadl: 4.32850e+03 W/m^2
The computation time for Simpsons 1/3 rule is: 2.580e-04 s
The accuracy compared to quadl bound 0: 8.660e-03 s

>>
```



```
when bound "a" is:     0
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 4.188e-02 s
2. Energy flux absorbed by the first plate using quadl:   NaN W/m^2
The computation time for Simpsons 1/3 rule is: 1.420e-01 s
The accuracy compared to quadl bound 0:   NaN s

when bound "a" is: 1.0e-07
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 1.722e-03 s
2. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for Simpsons 1/3 rule is: 1.322e-01 s
The accuracy compared to quadl bound 0: 5.686e-07 s

when bound "a" is: 1.0e-08
1. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for quadl is: 2.069e-03 s
2. Energy flux absorbed by the first plate using quadl: 4.29134e+03 W/m^2
The computation time for Simpsons 1/3 rule is: 1.357e-01 s
The accuracy compared to quadl bound 0: 5.686e-07 s
```

Q2.

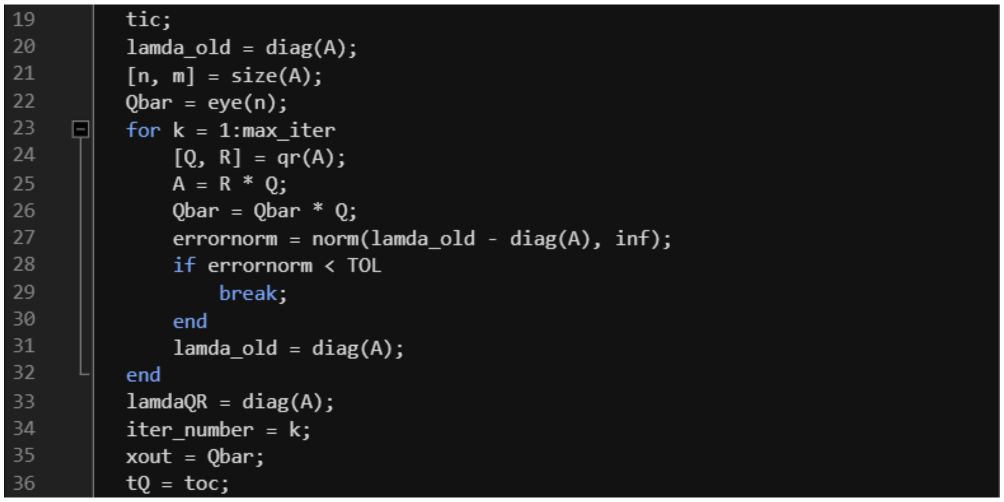
The following code depicts the initialization step of the code defining all given parameters and defining the matrix A. for this step each row was divided by its corresponding mass I.e. m1, m2, M, J.



/MATLAB Drive/MME 9621/Assignment_4/Q2.m

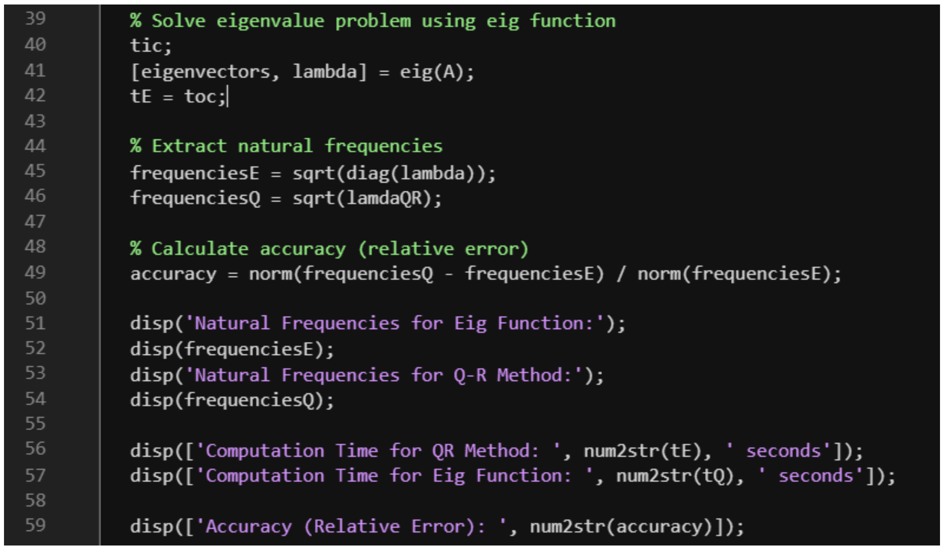
```
clc; close all; clear all;
% Given parameters
k1 = 18000; k2 = 20000; k3 = 20000;
l1 = 1.0; l2 = 1.5;
M = 1000; m1 = 100; m2 = 200;
r = 0.9; J = M * r^2;
a = k1 + k2;
b = k1 + k2;
c = k2 + k3;
d = k2*l1^2 + k3*l2^2;
x = [1,1,1,1];
max_iter = 100;
TOL = 10e-7;
A = [a/m1, 0, -k2/m1, k2*l1/m1;
    0, b/m2, -k3/m2, -k3*l2/m2;
    -k2/M, -k3/M, c/M, (k3*l2-k2*l1)/M;
    k2*l1/J, -k3*l2/J, (k3*l2-k2*l1)/J, d/J];
```

The following code depicts the Q-R function integrated into the main body of the code.



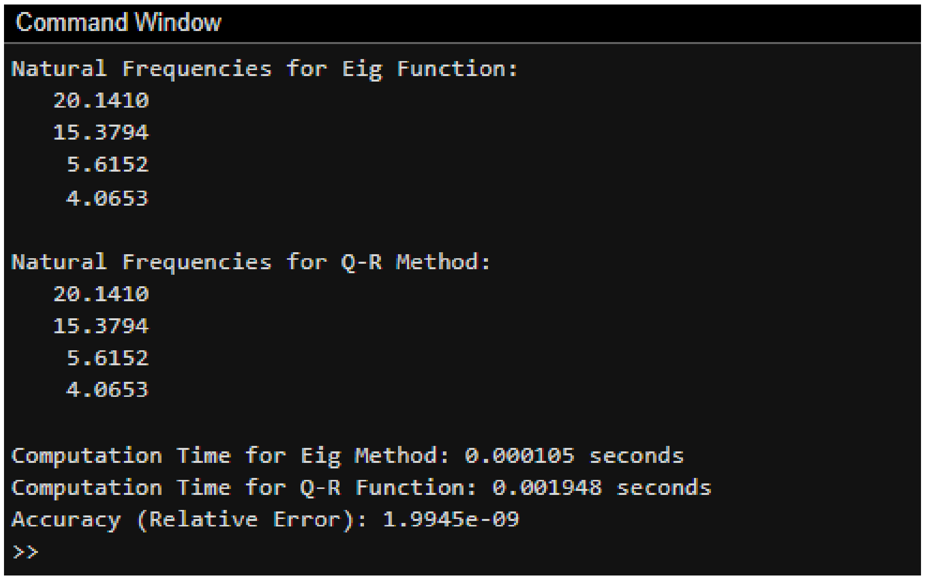
```
tic;
lamda_old = diag(A);
[n, m] = size(A);
Qbar = eye(n);
for k = 1:max_iter
    [Q, R] = qr(A);
    A = R * Q;
    Qbar = Qbar * Q;
    errornorm = norm(lamda_old - diag(A), inf);
    if errornorm < TOL
        break;
    end
    lamda_old = diag(A);
end
lamdaQR = diag(A);
iter_number = k;
xout = Qbar;
tQ = toc;
```

The following code depicts the resulting calculations and fprintfs to output all results. Note the eigenvectors and lambdas are also calculated however for this given problem only the natural frequencies will be output.



```
% Solve eigenvalue problem using eig function
tic;
[eigenvectors, lambda] = eig(A);
tE = toc;

% Extract natural frequencies
frequenciesE = sqrt(diag(lambda));
frequenciesQ = sqrt(lamdaQR);

% Calculate accuracy (relative error)
accuracy = norm(frequenciesQ - frequenciesE) / norm(frequenciesE);

disp('Natural Frequencies for Eig Function:');
disp(frequenciesE);
disp('Natural Frequencies for Q-R Method:');
disp(frequenciesQ);

disp(['Computation Time for QR Method: ', num2str(tE), ' seconds']);
disp(['Computation Time for Eig Function: ', num2str(tQ), ' seconds']);

disp(['Accuracy (Relative Error): ', num2str(accuracy)]);
```

The following command window depicts the outputs of the code listed above. It can be seen that both methods yield almost identical results with a relative accuracy of 1.9945e-9. This demonstrates that both methods can be used when accuracy is important. Additionally, it can be seen that Eig method yields significantly faster results than Q-R method. Therefore, for problems similar to this, it is recommended that Eig method is utilized.



```
Natural Frequencies for Eig Function:
   20.1410
   15.3794
    5.6152
    4.0653

Natural Frequencies for Q-R Method:
   20.1410
   15.3794
    5.6152
    4.0653

Computation Time for Eig Method: 0.000105 seconds
Computation Time for Q-R Function: 0.001948 seconds
Accuracy (Relative Error): 1.9945e-09
>>
```

Appendix

```
%% Q1
clc; close all; clear all;

for j =1:3
    n = 1e-11; % Step size for Simpson's 1/3 rule
    a = [0 1e-7 1e-8]; b = 10.5e-6; % Bounds
    tic;
    Qt(j) = quadl(@f, a(j), b);
    tQ = toc;
    tic;
    QS(j) = simpsons_rule(@f, a(j), b, n);
    tQS = toc;
    accuracy = norm(QS(j) - Qt(1)) / norm(Qt(1));
    formatspec = 'when bound "a" is: %5.1d\n';
    fprintf(formatspec,a(j));
    fprintf('1. Energy flux absorbed by the first plate using quadl: %5.5e W/m^2\n',
Qt(j));
    fprintf("The computation time for quadl is: %5.3e s\n", tQ);
    fprintf("2. Energy flux absorbed by the first plate using quadl: %5.5e W/m^2\n",
QS(j));
    fprintf("The computation time for Simpsons 1/3 rule is: %5.3e s\n", tQS);
    fprintf("The accuracy compared to quadl bound 0: %5.3e s\n\n", accuracy);
end

function f = f(lambda)
% Constants
c0 = 2.9979e8;  % Speed of light (m/s)
h = 6.626e-34;  % Planck's constant (Js)
kB = 1.3806e-23;  % Boltzmann constant (J/K)
T = 925;  % Temperature (K)

C1 = h * c0^2;
C2 = h * c0 / kB;

% Functions
epsilon = (lambda <= 10.5e-6) .* 0.850 .* (1 - lambda./ 10.5e-6) + (lambda > 10.5) .*
0;
rho = (lambda <= 4.5e-6) * 0.35 + (lambda > 4.5e-6) * 0.82;
E = 2 .* pi .* C1 ./ ((lambda).^5.*(exp(C2./(lambda.*T))-1));
f = rho.*epsilon.^2.*E;
end

function Q = simpsons_rule(func, a, b, h)

% Number of intervals
n = (b - a) / h;

% Initialize sum
sum = func(a) + func(b);

% Odd terms
odd_sum = 0;
```

```matlab
for i = 1:2:n-1
    x = a + i * h;
    odd_sum = odd_sum + func(x);
end

% Even terms
even_sum = 0;
for i = 2:2:n-2
    x = a + i * h;
    even_sum = even_sum + func(x);
end

% Final result
Q = h / 3 * (sum + 4 * odd_sum + 2 * even_sum);
end
%% Q2
clc; close all; clear all;
% Given parameters
k1 = 18000; k2 = 20000; k3 = 20000;
l1 = 1.0; l2 = 1.5;
M = 1000; m1 = 100; m2 = 200;
r = 0.9; J = M * r^2;
a = k1 + k2;
b = k1 + k2;
c = k2 + k3;
d = k2*l1^2 + k3*l2^2;
x = [1,1,1,1];
max_iter = 100;
TOL = 10e-7;
A = [a/m1, 0, -k2/m1, k2*l1/m1;
    0, b/m2, -k3/m2, -k3*l2/m2;
    -k2/M, -k3/M, c/M, (k3*l2-k2*l1)/M;
    k2*l1/J, -k3*l2/J, (k3*l2-k2*l1)/J, d/J];

tic;
lamda_old = diag(A);
[n, m] = size(A);
Qbar = eye(n);
for k = 1:max_iter
    [Q, R] = qr(A);
    A = R * Q;
    Qbar = Qbar * Q;
    errornorm = norm(lamda_old - diag(A), inf);
    if errornorm < TOL
        break;
    end
    lamda_old = diag(A);
end
lamdaQR = diag(A);
iter_number = k;
xout = Qbar;
tQ = toc;


% Solve eigenvalue problem using eig function
```

```
tic;
[eigenvectors, lambda] = eig(A);
tE = toc;

% Extract natural frequencies
frequenciesE = sqrt(diag(lambda));
frequenciesQ = sqrt(lamdaQR);

% Calculate accuracy (relative error)
accuracy = norm(frequenciesQ - frequenciesE) / norm(frequenciesE);

disp('Natural Frequencies for Eig Function:');
disp(frequenciesE);
disp('Natural Frequencies for Q-R Method:');
disp(frequenciesQ);

disp(['Computation Time for Eig Method: ', num2str(tE), ' seconds']);
disp(['Computation Time for Q-R Function: ', num2str(tQ), ' seconds']);

disp(['Accuracy (Relative Error): ', num2str(accuracy)]);
```