EP 501 FA 2020 midterm examination

October 16, 2020

Instructions: This is a take-home exam (work ALL problems) to be turned in 48 hours from the time it is posted (see canvas due date), unless you have notified me of a medical issue, family emergency, or other extenuating circumstance. *Unexcused late exams will not be graded.*

I will be checking my email regularly over the next 48 hours to answer questions. If you do not understand a question, please ask for a clarification.

You MAY NOT work in groups or discuss details of the solution to these or similar problems with other students or faculty. You also MAY NOT use online "work this problem for me" resources (i.e. you must complete this problem without direct assistance of another human being). You ARE ALLOWED to use internet references, your notes, textbooks, and other resources such as physics books, differential equations books, integral tables, the TI-89, or mathematical software like Mathematica, Matlab, Maple, and similar. All normal University Honor Code rules apply, except as noted above (e.g. I am allowing use of books, notes, and internet resources).

Include citations, where appropriate, to results that you use for your solutions (e.g. "Jackson eqn. 3.70", "Wolfram website", "Maple software" and so on) and make sure that your work is completely described by your solution (i.e. it adequately developed and explained). Please start a new page for each problem. You must submit your completed midterm via CANVAS as a .zip file with Python or MATLAB codes and Word, Pages, or LaTeXtext.

Note that the problems are organized in order of *increasing* difficulty. You may complete any coding tasks using either Python or MATLAB.

You must sign (below) and attach this page to the front of your solution when you submit your solutions for this exam. Electronic signatures are acceptable and encouraged. If you are typing up your solution in LaTeX, please note that the source code for this document is included in the course repository.

I,  *, confirm that I read the instruction above and did not discuss the solution to these problems with anyone else.*

1. **This problem requires both written (or typeset) and coding components.**

   In class we compared the efficiency of Gaussian elimination vs. Jacobi iteration for linear systems of various size in the `benchmark` scripts (see `./linear_algebra/benchmark.(py,m)` in the course code repository for the language of your choice). In this problem we will explore a special method for solving tridiagonal systems that is even more efficient than these two options - the Thomas algorithm.

   (a) Write a MATLAB or Python function `tridiag()` that solves a tridiagonal system of equations using the Thomas algorithm. Verify your solution by applying it to the iterative test problem for HW2 in the EP501 Assignments repository in `./HW2/iterative_testproblem.mat` file (this problem is actually tridiagonal). If you are using Python, see the `./python_basics/load_matlab_file.py` script in the EP501 Python repository for an example of how to import MATLAB data into Python.

   (b) Produce a new version of the `benchmark.(py,m)` script that runs timed tests for all three of: Gaussian elimination, Jacobi iteration, and your new `tridiag` solver. Use the same tridiagonal problem for testing each of the solvers (as in the existing `benchmark.(py,m)` script). Store the solve times for each method and each system size in separate arrays and plot times vs. number of unknowns for each method on a single graph.

   (c) For what system sizes does Gaussian elimination perform better than Jacobi iteration (on your computer)?

   (d) Explain why the Thomas algorithm is more efficient for large numbers of unknowns. Try to be as quantitative as possible by discussing the number of operations requires to execute each algorithm/iteration. The textbook may be of use.

2. **This problem requires a substantial handwritten component which should be scanned or typeset and submitted with your midterm (the problem also requires coding).**

   In plasma physics we encounter the following homogeneous system of equations describing waves in a magnetohydrodynamic (MHD) plasma:

$$\begin{bmatrix} v^2 - C_s^2 \sin^2\theta - C_A^2 & 0 & -C_s^2 \sin\theta\cos\theta \\ 0 & v^2 - C_A^2\cos^2\theta & 0 \\ -C_s^2\sin\theta\cos\theta & 0 & v^2 - C_s^2\cos^2\theta \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = 0 \tag{1}$$

   represented compactly as:

$$\underline{\underline{A}}\,\underline{u} = 0 \tag{2}$$

   In this system v is the wave velocity, θ is the angle of propagation (with respect to the local magnetic field), $u_{x,y,z}$ are the components of the plasma drift. The remaining constants are the adiabatic sound speed C_s and the Alfvén speed C_A:

$$C_s \equiv \sqrt{\frac{\gamma p}{\rho}}; \qquad C_A \equiv \sqrt{\frac{B^2}{\mu_0 \rho}} \tag{3}$$

   where γ is the adiabatic index, p is the plasma thermal pressure, ρ is the plasma mass density, and B is the magnitude of the local magnetic field.

   (a) Under what condition does this homogeneous system of equations have a nontrivial solution (viz. $\underline{u} \neq 0$)?

   (b) Derive a polynomial for the unknown wave speed v using the condition you found in part a of this problem.

   (c) How many roots (known as wave modes) does your characteristic polynomial have?

μ_0: Vacuum Permeability
$\mu_0 = 4\pi \times 10^{-7}$ N/A²

(d) Two of the roots for this system are $v = \pm C_A \cos\theta$. Plug one of these roots back into the matrix equation and use it to determine which component(s) of drift, i.e. $u_{x,y,z}$, can be nonzero for this wave mode (the shear Alfvén mode).

(e) Given typical parameters for the solar wind plasma: $\gamma = 5/3$, $\rho = 1.67 \times 10^{-21}$ [kg/m^3], $p = 1.38 \times 10^{-11}$ [Pa], and $B = 10^{-9}$ [T], compute values for the sound and Alfvén speeds in the plasma.

(f) Use an *exact* Newton method (i.e. use analytically computed derivatives) to find numerical values for all roots using $\theta = \pi/4$ for the angle of propagation. Make sure you select a sensible convergence criteria given the coefficients for this problem and treat all parameters except for the unknown roots for v to be constant for purposes of developing derivatives needed to implement Newton's method.

3. **You will need to write a Python or MATLAB script to solve this question. Be sure to output all quantities requested by this problem and included written/typed responses explaining any formulae you have derived and used in your code.**

This problem concerns polynomial division and its use to develop root finding methods for high-order polynomials.

(a) Write a Python or MATLAB function that *analytically* solves a quadratic equation of the form:

$$ax^2 + bx + c = 0 \tag{4}$$

given the column vector of coefficients $[a, b, c]^\mathsf{T}$. Test your code on the quadratic:

$$2x^2 - 6x + 4 = 0, \tag{5}$$

and compare against roots that you find by hand via factorization or quadratic formula. Show your work.

(b) Write a polynomial division algorithm (the classic approach is discusssed on pgs. 194-195 of the textbook) capable of dividing a given polynomial $P_n(x)$ (of order n and defined by a set of coefficients) by a given divisor $(x - N)$. I.e. find $Q_{n-1}(x)$ such that:

$$P_n(x) = (x - N)Q_{n-1}(x) + R \tag{6}$$

Q_{n-1} is the polynomial left once you divide $(x - N)$ out of P_n. Test your code by using it to divide out a factor of $(x - 5)$ from the polynomial:

$$x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120 = 0 \tag{7}$$

Note: for the rest of this problem the remainder R ca be set to zero since we will be dividing out known roots.

(c) Use an *approximate* Newton's method (using an approximate derivative) to find *one* root of Eqn. 7

(d) Use your synthetic division algorithm to factor the root found in part c out of Eqn. 7 to produce a lower-order polynomial (i.e. $Q_{n-1}(x)$). The result will be a set of coefficients for a fourth order polynomial, denoted Q_4.

(d) Write a program to repeat the "deflation" process described above (parts c and d) to find all of roots of Eqn. 7. This can be done by taking Q_{n-1} (which has the first root you've found divided out) and using approximate Newton's method to find one of its roots. I.e. find a root N for Q_{n-1} and then use it to findQ_{n-2} such that: $Q_{n-1} = (x - N)Q_{n-2}$. Repeat this procedure until you have divided out three roots such that you are left with a second order polynomial Q_2. Once you have Q_2 use your second degree polynomial solver from part a to find the final two roots (this

procedure is referred to as polynomial deflation). You can code your script to work specifically for this fifth order polynomial example - it does not have to be general enough to work with a polynomial of arbitrary degree (this would require a bit more programming).

Hint/Suggestion: If you are using MATLAB you will likely find the function `polyval()` *to be useful in your Newton method since you can input coefficients for a polynomial of arbitrary degree. This prevents the need to hardcode a separate objective function for each Newton iteration needed. You can simply use* polyval *with your new set of coefficient each time you divide out a factor. If using Python the analogous function is* `numpy.poly1d()`*. It is probably easiest to do this problem with the approximate Newton's method since you do not need to manually recompute derivatives once you divide out a factor and produce a new polynomial.*

1(a) Thomas algorithm: tridiag( )

Contents


```
function [x,nit] = tridiag(A,b)
```

```
% Midterm Problem 1(a)
% A is a tridiagonal matrix
% Thomas algorithm
```

Check the size of A

```
n=size(A,1);  %system size
```

```
Not enough input arguments.

Error in tridiag (line 8)
n=size(A,1);  %system size
```

Store the tridiagonal elements in A'

```
    for i = 2 : n
        Ap(i,1) = A(i,i-1);
    end % for
    for i = 1 : n
        Ap(i,2) = A(i,i);
    end % for
    for i = 1 : n - 1
        Ap(i,3) = A(i,i+1);
    end % for
```

Elimination

```
nit = 0;
    for i = 2 : n
        em = Ap(i,1)/Ap(i-1,2); % Store the elimination multiplier
        Ap(i,2) = Ap(i,2) - em*Ap(i-1,3);
        b(i) = b(i) - em*b(i-1);
        nit = nit + 3;
    end % for
    x(n,1) = b(n)/Ap(n,2);
    nit = nit + 1;
    for i = n-1 : -1 : 1
        x(i,1) = (b(i) - Ap(i,3)*x(i+1))/Ap(i,2);
        nit = nit + 2;
    end %for
```

```
end %function
```

Published with MATLAB® R2020b

1(a) Test the tridiag() function

```
% Thomas algorithm
% Midterm Problem 1(a)

clear
clc
close all

load('iterative_testproblem.mat')

[xTA,iterations] = tridiag(Ait,bit);

disp('Solution with Thomas algorithm:  ')
disp(xTA);

xMAT = Ait\bit;
disp('Matlab built-in solution:  ')
disp(xMAT);

ERR = xMAT - xTA;
disp('Error: ')
disp(ERR)

disp('Number of multiplications required for Thomas algorithm:  ')
disp(iterations);

table(xTA, xMAT, ERR)
```

```
Solution with Thomas algorithm:
    0.0329
    0.1316
    0.2400
    0.3375
    0.4142
    0.4642
    0.4839
    0.4720
    0.4293
    0.3584
    0.2641
    0.1526
    0.0310
   -0.0926
   -0.2101
   -0.3138
   -0.3971
   -0.4544
   -0.4819
   -0.4780
   -0.4427
   -0.3785
   -0.2896
   -0.1817
   -0.0619
    0.0619
    0.1817
    0.2896
    0.3785
```

```
     0.4427
     0.4780
     0.4819
     0.4544
     0.3971
     0.3138
     0.2101
     0.0926
    -0.0310
    -0.1526
    -0.2641
    -0.3584
    -0.4293
    -0.4720
    -0.4839
    -0.4642
    -0.4142
    -0.3375
    -0.2400
    -0.1316
    -0.0329

Matlab built-in solution:
     0.0329
     0.1316
     0.2400
     0.3375
     0.4142
     0.4642
     0.4839
     0.4720
     0.4293
     0.3584
     0.2641
     0.1526
     0.0310
    -0.0926
    -0.2101
    -0.3138
    -0.3971
    -0.4544
    -0.4819
    -0.4780
    -0.4427
    -0.3785
    -0.2896
    -0.1817
    -0.0619
     0.0619
     0.1817
     0.2896
     0.3785
     0.4427
     0.4780
     0.4819
     0.4544
     0.3971
     0.3138
     0.2101
     0.0926
    -0.0310
```

```
   -0.1526
   -0.2641
   -0.3584
   -0.4293
   -0.4720
   -0.4839
   -0.4642
   -0.4142
   -0.3375
   -0.2400
   -0.1316
   -0.0329

Error:
   1.0e-15 *

   -0.0069
   -0.0278
   -0.0278
   -0.0555
         0
   -0.0555
   -0.0555
    0.0555
         0
   -0.0555
    0.0555
   -0.0278
   -0.0035
    0.0139
    0.0555
         0
    0.0555
    0.1110
    0.0555
    0.0555
    0.0555
    0.0555
         0
         0
    0.0278
   -0.0139
   -0.0555
   -0.1110
   -0.1110
   -0.1110
   -0.1110
   -0.1110
   -0.1110
         0
         0
         0
         0
    0.0035
         0
    0.0555
    0.0555
    0.0555
    0.0555
    0.0555
    0.0555
```

```
    0.0555
    0.0555
    0.0278
         0
    0.0069

Number of multiplications required for Thomas algorithm:
   246


ans =

  50×3 table
```

Thomas alg.

Matlab

Error btwn the two

xTA	xMAT	ERR
0.032907	0.032907	-6.9389e-18
0.13163	0.13163	-2.7756e-17
0.23995	0.23995	-2.7756e-17
0.33746	0.33746	-5.5511e-17
0.4142	0.4142	0
0.4642	0.4642	-5.5511e-17
0.48393	0.48393	-5.5511e-17
0.47203	0.47203	5.5511e-17
0.42926	0.42926	0
0.35842	0.35842	-5.5511e-17
0.26413	0.26413	5.5511e-17
0.15256	0.15256	-2.7756e-17
0.031021	0.031021	-3.4694e-18
-0.092552	-0.092552	1.3878e-17
-0.21007	-0.21007	5.5511e-17
-0.31385	-0.31385	0
-0.3971	-0.3971	5.5511e-17
-0.45437	-0.45437	1.1102e-16
-0.48193	-0.48193	5.5511e-17
-0.47796	-0.47796	5.5511e-17
-0.44273	-0.44273	5.5511e-17
-0.37854	-0.37854	5.5511e-17
-0.28958	-0.28958	0
-0.18169	-0.18169	0
-0.061914	-0.061914	2.7756e-17
0.061914	0.061914	-1.3878e-17
0.18169	0.18169	-5.5511e-17
0.28958	0.28958	-1.1102e-16
0.37854	0.37854	-1.1102e-16
0.44273	0.44273	-1.1102e-16
0.47796	0.47796	-1.1102e-16
0.48193	0.48193	-1.1102e-16
0.45437	0.45437	-1.1102e-16
0.3971	0.3971	0
0.31385	0.31385	0
0.21007	0.21007	0
0.092552	0.092552	0
-0.031021	-0.031021	3.4694e-18
-0.15256	-0.15256	0
-0.26413	-0.26413	5.5511e-17
-0.35842	-0.35842	5.5511e-17
-0.42926	-0.42926	5.5511e-17
-0.47203	-0.47203	5.5511e-17
-0.48393	-0.48393	5.5511e-17

The tridiag( ) function works very good!

Errors are very small.

-0.4642	-0.4642	5.5511e-17
-0.4142	-0.4142	5.5511e-17
-0.33746	-0.33746	5.5511e-17
-0.23995	-0.23995	2.7756e-17
-0.13163	-0.13163	0
-0.032907	-0.032907	6.9389e-18

Published with MATLAB® R2020b

1(b) benchmark

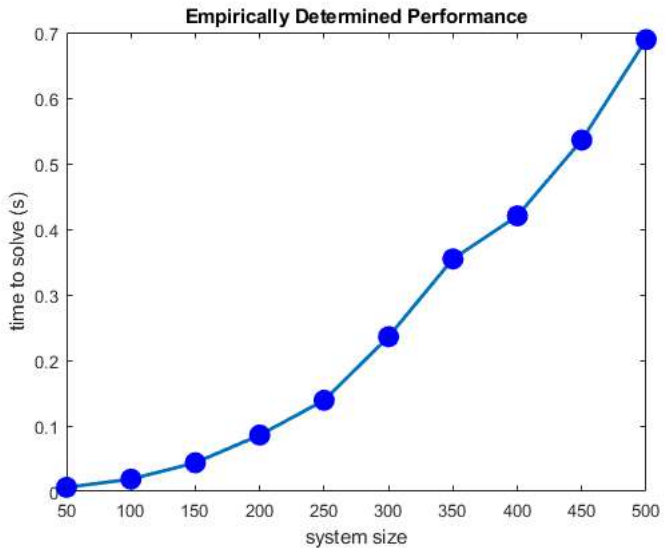
Contents


```
% Evaluate performance and scaling of Gaussian elimination, Jacobi
% iteration, and Thomas algorithm by solving systems of different
% size and timing the solves

clear
clc
close all

nvals=50:50:500;
testtimes=zeros(size(nvals));
lrep=10;     %how many times to repeat each test
```

Gaussian elimination

```
disp('Start of tests of Gaussian-elimination scaling');
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        [Blargemod,ordlarge]=Gauss_elim(Blarge,blarge);
        xlarge=backsub(Blargemod(ordlarge,:));
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' GE solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,1) = testtimes';

figure(1);
plot(nvals,testtimes,'-o','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','blue','MarkerFaceColor','blue')
xlabel('system size');
ylabel('time to solve (s)');
title('Empirically Determined Performance');
```

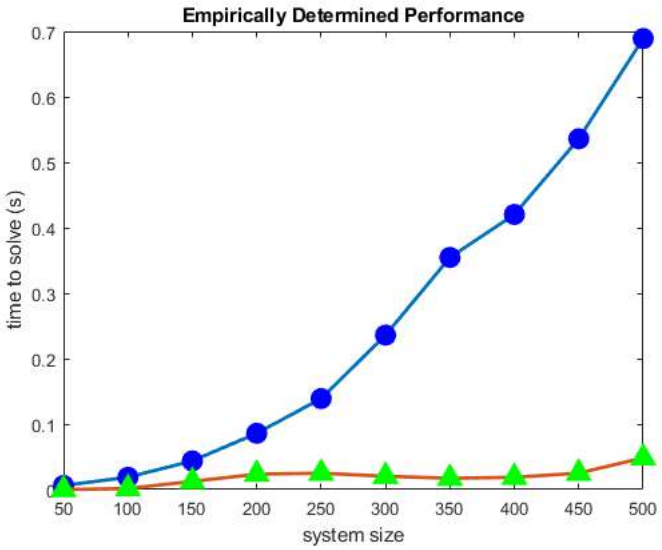
```
Start of tests of Gaussian-elimination scaling
 GE solution for system of size 50 takes average time 0.00625 s
 GE solution for system of size 100 takes average time 0.01875 s
 GE solution for system of size 150 takes average time 0.04375 s
 GE solution for system of size 200 takes average time 0.085938 s
 GE solution for system of size 250 takes average time 0.13906 s
 GE solution for system of size 300 takes average time 0.23594 s
 GE solution for system of size 350 takes average time 0.35469 s
 GE solution for system of size 400 takes average time 0.42031 s
 GE solution for system of size 450 takes average time 0.53594 s
 GE solution for system of size 500 takes average time 0.68906 s
```



Jacobi iteration

```
disp('Start of tests for Jacobi iteration');
tol=1e-9;
testtimes=zeros(size(nvals));
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        x0=randn(nlarge,1);
        [xit,iterations]=Jacobi(x0,Blarge,blarge,tol,false);
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' JI solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,2) = testtimes';

figure(1);
hold on
plot(nvals,testtimes,'-^','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','green','MarkerFaceColor','green')
```

```
Start of tests for Jacobi iteration
 JI solution for system of size 50 takes average time 0 s
 JI solution for system of size 100 takes average time 0.0015625 s
 JI solution for system of size 150 takes average time 0.0125 s
 JI solution for system of size 200 takes average time 0.023438 s
 JI solution for system of size 250 takes average time 0.025 s
 JI solution for system of size 300 takes average time 0.020313 s
 JI solution for system of size 350 takes average time 0.017187 s
 JI solution for system of size 400 takes average time 0.01875 s
 JI solution for system of size 450 takes average time 0.025 s
 JI solution for system of size 500 takes average time 0.048438 s
```



Thomas algorithm

```
disp('Start of tests for tridiag solver');
% tol=1e-9;
testtimes=zeros(size(nvals));
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        [xit,iterations]=tridiag(Blarge,blarge);
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' TA solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,3) = testtimes';
disp('Time required for GE, JI, & TA (s)')
disp(TT)

figure(1);
```

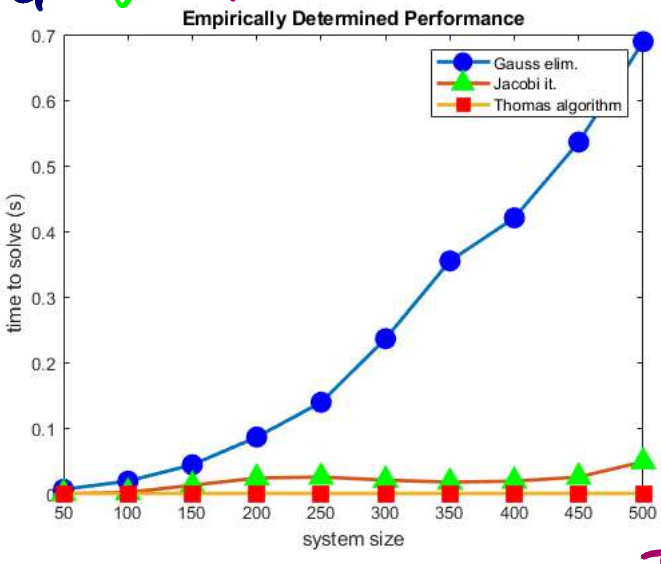
```
hold on
plot(nvals,testtimes,'-s','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','red','MarkerFaceColor','red')
legend('Gauss elim.','Jacobi it.','Thomas algorithm')
```

```
Start of tests for tridiag solver
 TA solution for system of size 50 takes average time 0 s
 TA solution for system of size 100 takes average time 0 s
 TA solution for system of size 150 takes average time 0 s
 TA solution for system of size 200 takes average time 0 s
 TA solution for system of size 250 takes average time 0 s
 TA solution for system of size 300 takes average time 0 s
 TA solution for system of size 350 takes average time 0 s
 TA solution for system of size 400 takes average time 0 s
 TA solution for system of size 450 takes average time 0 s
 TA solution for system of size 500 takes average time 0 s
Time required for GE, JI, & TA (s)
```

G	J	T
0.0063	0	0
0.0187	0.0016	0
0.0437	0.0125	0
0.0859	0.0234	0
0.1391	0.0250	0
0.2359	0.0203	0
0.3547	0.0172	0
0.4203	0.0187	0
0.5359	0.0250	0
0.6891	0.0484	0

Gaussian elim. doesn't do better than Jacobi for $50 \leq n \leq 500$.

Gaussian elim. escalates fast.



Jacobi inflates faster than Thomas.

Thomas barely increases in time for $n \sim 500$.

Published with MATLAB® R2020b

1(c) Gaussian vs. Jacobi

Tested relatively small sizes ($n \sim 50$) since for $50 \le n \le 500$, Jacobi seemed always faster than Gaussian.

Contents

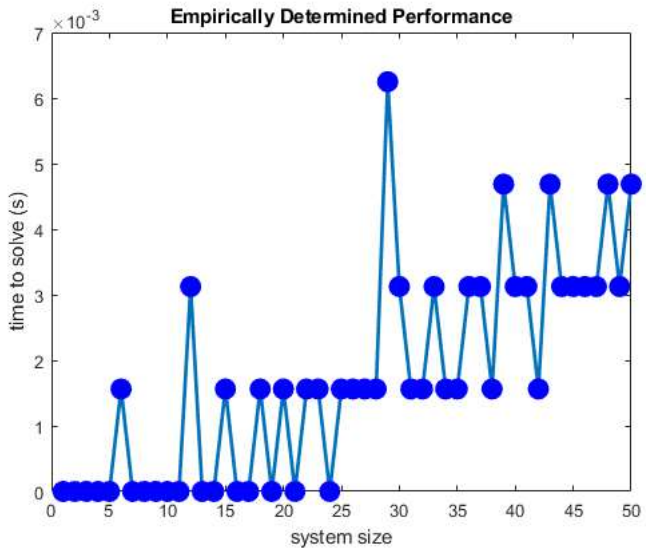

```
% Evaluate performance and scaling of Gaussian elimination, Jacobi
% iteration, and Thomas algorithm by solving systems of different
% size and timing the solves

clear
clc
close all

nvals=1:1:50;
testtimes=zeros(size(nvals));
lrep=10;     %how many times to repeat each test
```

Gaussian elimination

```
disp('Start of tests of Gaussian-elimination scaling');
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        [Blargemod,ordlarge]=Gauss_elim(Blarge,blarge);
        xlarge=backsub(Blargemod(ordlarge,:));
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' GE solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,1) = testtimes';

figure(1);
plot(nvals,testtimes,'-o','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','blue','MarkerFaceColor','blue')
xlabel('system size');
ylabel('time to solve (s)');
title('Empirically Determined Performance');
```

```
Start of tests of Gaussian-elimination scaling
 GE solution for system of size 1 takes average time 0 s
 GE solution for system of size 2 takes average time 0 s
 GE solution for system of size 3 takes average time 0 s
 GE solution for system of size 4 takes average time 0 s
 GE solution for system of size 5 takes average time 0 s
 GE solution for system of size 6 takes average time 0.0015625 s
 GE solution for system of size 7 takes average time 0 s
 GE solution for system of size 8 takes average time 0 s
 GE solution for system of size 9 takes average time 0 s
 GE solution for system of size 10 takes average time 0 s
 GE solution for system of size 11 takes average time 0 s
 GE solution for system of size 12 takes average time 0.003125 s
 GE solution for system of size 13 takes average time 0 s
 GE solution for system of size 14 takes average time 0 s
 GE solution for system of size 15 takes average time 0.0015625 s
 GE solution for system of size 16 takes average time 0 s
 GE solution for system of size 17 takes average time 0 s
 GE solution for system of size 18 takes average time 0.0015625 s
 GE solution for system of size 19 takes average time 0 s
 GE solution for system of size 20 takes average time 0.0015625 s
 GE solution for system of size 21 takes average time 0 s
 GE solution for system of size 22 takes average time 0.0015625 s
 GE solution for system of size 23 takes average time 0.0015625 s
 GE solution for system of size 24 takes average time 0 s
 GE solution for system of size 25 takes average time 0.0015625 s
 GE solution for system of size 26 takes average time 0.0015625 s
 GE solution for system of size 27 takes average time 0.0015625 s
 GE solution for system of size 28 takes average time 0.0015625 s
 GE solution for system of size 29 takes average time 0.00625 s
 GE solution for system of size 30 takes average time 0.003125 s
 GE solution for system of size 31 takes average time 0.0015625 s
 GE solution for system of size 32 takes average time 0.0015625 s
 GE solution for system of size 33 takes average time 0.003125 s
 GE solution for system of size 34 takes average time 0.0015625 s
 GE solution for system of size 35 takes average time 0.0015625 s
 GE solution for system of size 36 takes average time 0.003125 s
 GE solution for system of size 37 takes average time 0.003125 s
 GE solution for system of size 38 takes average time 0.0015625 s
 GE solution for system of size 39 takes average time 0.0046875 s
 GE solution for system of size 40 takes average time 0.003125 s
 GE solution for system of size 41 takes average time 0.003125 s
```

```
GE solution for system of size 42 takes average time 0.0015625 s
GE solution for system of size 43 takes average time 0.0046875 s
GE solution for system of size 44 takes average time 0.003125 s
GE solution for system of size 45 takes average time 0.003125 s
GE solution for system of size 46 takes average time 0.003125 s
GE solution for system of size 47 takes average time 0.003125 s
GE solution for system of size 48 takes average time 0.0046875 s
GE solution for system of size 49 takes average time 0.003125 s
GE solution for system of size 50 takes average time 0.0046875 s
```

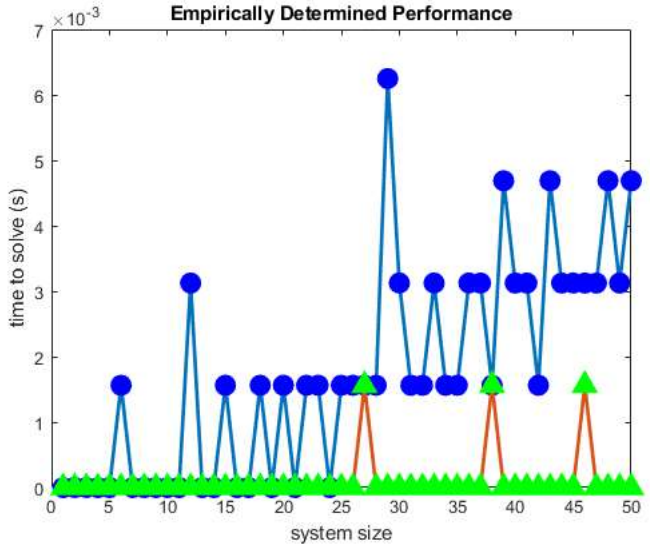


Jacobi iteration

```
disp('Start of tests for Jacobi iteration');
tol=1e-9;
testtimes=zeros(size(nvals));
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        x0=randn(nlarge,1);
        [xit,iterations]=Jacobi(x0,Blarge,blarge,tol,false);
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' JI solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,2) = testtimes';

figure(1);
hold on
plot(nvals,testtimes,'-^','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','green','MarkerFaceColor','green')
```

```
Start of tests for Jacobi iteration
 JI solution for system of size 1 takes average time 0 s
 JI solution for system of size 2 takes average time 0 s
 JI solution for system of size 3 takes average time 0 s
 JI solution for system of size 4 takes average time 0 s
 JI solution for system of size 5 takes average time 0 s
 JI solution for system of size 6 takes average time 0 s
 JI solution for system of size 7 takes average time 0 s
 JI solution for system of size 8 takes average time 0 s
 JI solution for system of size 9 takes average time 0 s
 JI solution for system of size 10 takes average time 0 s
 JI solution for system of size 11 takes average time 0 s
 JI solution for system of size 12 takes average time 0 s
 JI solution for system of size 13 takes average time 0 s
 JI solution for system of size 14 takes average time 0 s
 JI solution for system of size 15 takes average time 0 s
 JI solution for system of size 16 takes average time 0 s
 JI solution for system of size 17 takes average time 0 s
 JI solution for system of size 18 takes average time 0 s
 JI solution for system of size 19 takes average time 0 s
 JI solution for system of size 20 takes average time 0 s
 JI solution for system of size 21 takes average time 0 s
 JI solution for system of size 22 takes average time 0 s
 JI solution for system of size 23 takes average time 0 s
 JI solution for system of size 24 takes average time 0 s
```

```
 JI solution for system of size 25 takes average time 0 s
 JI solution for system of size 26 takes average time 0 s
 JI solution for system of size 27 takes average time 0.0015625 s
 JI solution for system of size 28 takes average time 0 s
 JI solution for system of size 29 takes average time 0 s
 JI solution for system of size 30 takes average time 0 s
 JI solution for system of size 31 takes average time 0 s
 JI solution for system of size 32 takes average time 0 s
 JI solution for system of size 33 takes average time 0 s
 JI solution for system of size 34 takes average time 0 s
 JI solution for system of size 35 takes average time 0 s
 JI solution for system of size 36 takes average time 0 s
 JI solution for system of size 37 takes average time 0 s
 JI solution for system of size 38 takes average time 0.0015625 s
 JI solution for system of size 39 takes average time 0 s
 JI solution for system of size 40 takes average time 0 s
 JI solution for system of size 41 takes average time 0 s
 JI solution for system of size 42 takes average time 0 s
 JI solution for system of size 43 takes average time 0 s
 JI solution for system of size 44 takes average time 0 s
 JI solution for system of size 45 takes average time 0 s
 JI solution for system of size 46 takes average time 0.0015625 s
 JI solution for system of size 47 takes average time 0 s
 JI solution for system of size 48 takes average time 0 s
 JI solution for system of size 49 takes average time 0 s
 JI solution for system of size 50 takes average time 0 s
```



Thomas algorithm

```
disp('Start of tests for tridiag solver');
% tol=1e-9;
testtimes=zeros(size(nvals));
for in=1:numel(nvals)
    nlarge=nvals(in);
    Blarge=diag(-1*ones(nlarge-1,1),-1)+diag(-1*ones(nlarge-1,1),1)+diag(4*ones(nlarge,1),0);    %this must be diagonally dominant or else the method won't converge
    blarge=ones(nlarge,1);

    for irep=1:lrep     %benchmark will repeat the same solution several times to eliminate random variations from CPU load, etc.
        tstart=cputime;
        [xit,iterations]=tridiag(Blarge,blarge);
        tend=cputime;
        testtimes(in)=testtimes(in)+(tend-tstart)/lrep;
    end %for
    disp([' TA solution for system of size ',num2str(nlarge),' takes average time ',num2str(testtimes(in)),' s']);
end %for

TT(:,3) = testtimes';
disp('Time required for GE, JI, & TA (s)')
disp(TT)

figure(1);
hold on
plot(nvals,testtimes,'-s','LineWidth',2,'MarkerSize',10,'MarkerEdgeColor','red','MarkerFaceColor','red')
legend('Gauss elim.','Jacobi it.','Thomas algorithm')
```

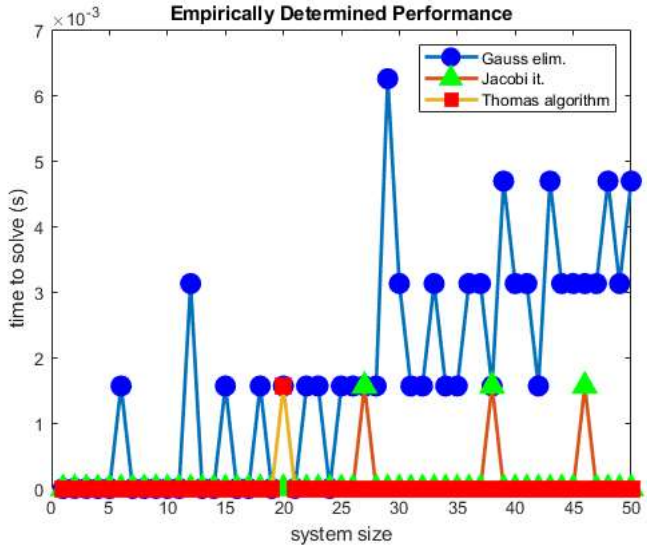
```
Start of tests for tridiag solver
 TA solution for system of size 1 takes average time 0 s
 TA solution for system of size 2 takes average time 0 s
 TA solution for system of size 3 takes average time 0 s
 TA solution for system of size 4 takes average time 0 s
 TA solution for system of size 5 takes average time 0 s
```

```
TA solution for system of size 6 takes average time 0 s
TA solution for system of size 7 takes average time 0 s
TA solution for system of size 8 takes average time 0 s
TA solution for system of size 9 takes average time 0 s
TA solution for system of size 10 takes average time 0 s
TA solution for system of size 11 takes average time 0 s
TA solution for system of size 12 takes average time 0 s
TA solution for system of size 13 takes average time 0 s
TA solution for system of size 14 takes average time 0 s
TA solution for system of size 15 takes average time 0 s
TA solution for system of size 16 takes average time 0 s
TA solution for system of size 17 takes average time 0 s
TA solution for system of size 18 takes average time 0 s
TA solution for system of size 19 takes average time 0 s
TA solution for system of size 20 takes average time 0.0015625 s
TA solution for system of size 21 takes average time 0 s
TA solution for system of size 22 takes average time 0 s
TA solution for system of size 23 takes average time 0 s
TA solution for system of size 24 takes average time 0 s
TA solution for system of size 25 takes average time 0 s
TA solution for system of size 26 takes average time 0 s
TA solution for system of size 27 takes average time 0 s
TA solution for system of size 28 takes average time 0 s
TA solution for system of size 29 takes average time 0 s
TA solution for system of size 30 takes average time 0 s
TA solution for system of size 31 takes average time 0 s
TA solution for system of size 32 takes average time 0 s
TA solution for system of size 33 takes average time 0 s
TA solution for system of size 34 takes average time 0 s
TA solution for system of size 35 takes average time 0 s
TA solution for system of size 36 takes average time 0 s
TA solution for system of size 37 takes average time 0 s
TA solution for system of size 38 takes average time 0 s
TA solution for system of size 39 takes average time 0 s
TA solution for system of size 40 takes average time 0 s
TA solution for system of size 41 takes average time 0 s
TA solution for system of size 42 takes average time 0 s
TA solution for system of size 43 takes average time 0 s
TA solution for system of size 44 takes average time 0 s
TA solution for system of size 45 takes average time 0 s
TA solution for system of size 46 takes average time 0 s
TA solution for system of size 47 takes average time 0 s
TA solution for system of size 48 takes average time 0 s
TA solution for system of size 49 takes average time 0 s
TA solution for system of size 50 takes average time 0 s
Time required for GE, JI, & TA (s)
```

G	J	T
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0.0016	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0.0031	0	0
0	0	0
0	0	0
0.0016	0	0
0	0	0
0	0	0
0.0016	0	0
0	0	0
0.0016	0	0.0016
0	0	0
0.0016	0	0
0.0016	0	0
0	0	0
0.0016	0	0
0.0016	0	0
0.0016	0.0016	0
0.0016	0	0
0.0063	0	0
0.0031	0	0
0.0016	0	0
0.0016	0	0
0.0031	0	0
0.0016	0	0
0.0016	0	0
0.0031	0	0
0.0031	0	0
0.0016	0.0016	0
0.0047	0	0
0.0031	0	0
0.0031	0	0
0.0016	0	0
0.0047	0	0
0.0031	0	0
0.0031	0	0

Seemingly, the Gaussian elim. didn't really beat the Jacobi it. if not never, for very small n, or at least the difference is too small to detect.
Gaussian may do equally good compared to Jacobi, but it does not do considerably faster on my computer.

0.0031	0.0016	0
0.0031	0	0
0.0047	0	0
0.0031	0	0
0.0047	0	0



Published with MATLAB® R2020b

1(d) The number of multiplications required by Gaussian Elimination is approximately

$$N_{GA} \approx (n^3/3 - n/3) + (n^2)$$

$$= \frac{n^3}{3} + n^2 - \frac{n}{3} \text{ times.}$$

(p.40 Ch. 1.3.3 of the textbook)

The Jacobi iterations calculate residuals and update x where both require n multiplications. The total number of multiplications required by the Jacobi it. is

$$N_{JI} \approx n^2 \text{ times.}$$

The Jacobi iteration, in general, is an order of magnitude faster than Gaussian Elim.

Now, this is the copy of the Thomas alg. from the textbook p. 50 (Ch.1 Sec.1.5).

em

$$a'_{1,2} = a'_{1,2} \tag{1.150a}$$

$$a'_{i,2} = a'_{i,2} - (a'_{i,1}/a'_{i-1,2})a'_{i-1,3} \qquad (i = 2, 3, \ldots, n) \tag{1.150b}$$

The **b** vector is modified as follows:

$$b_1 = b_1 \tag{1.151a}$$

$$b_i = b_i - (a'_{i,1}/a'_{i-1,2})b_{i-1} \qquad (i = 2, 3, \ldots, n) \tag{1.151b}$$

After $a'_{i,2}$ $(i = 2, 3, \ldots, n)$ and **b** are evaluated, the back substitution step is as follows:

$$x_n = b_n/a'_{n,2} \tag{1.152a}$$

$$x_i = (b_i - a'_{i,3}x_{i+1})/a'_{i,2} \qquad (i = n-1, n-2, \ldots, 1) \tag{1.152b}$$

The total number of multiplications that the Thomas algorithm requires is

Store 'em Elim. A' Elim. b

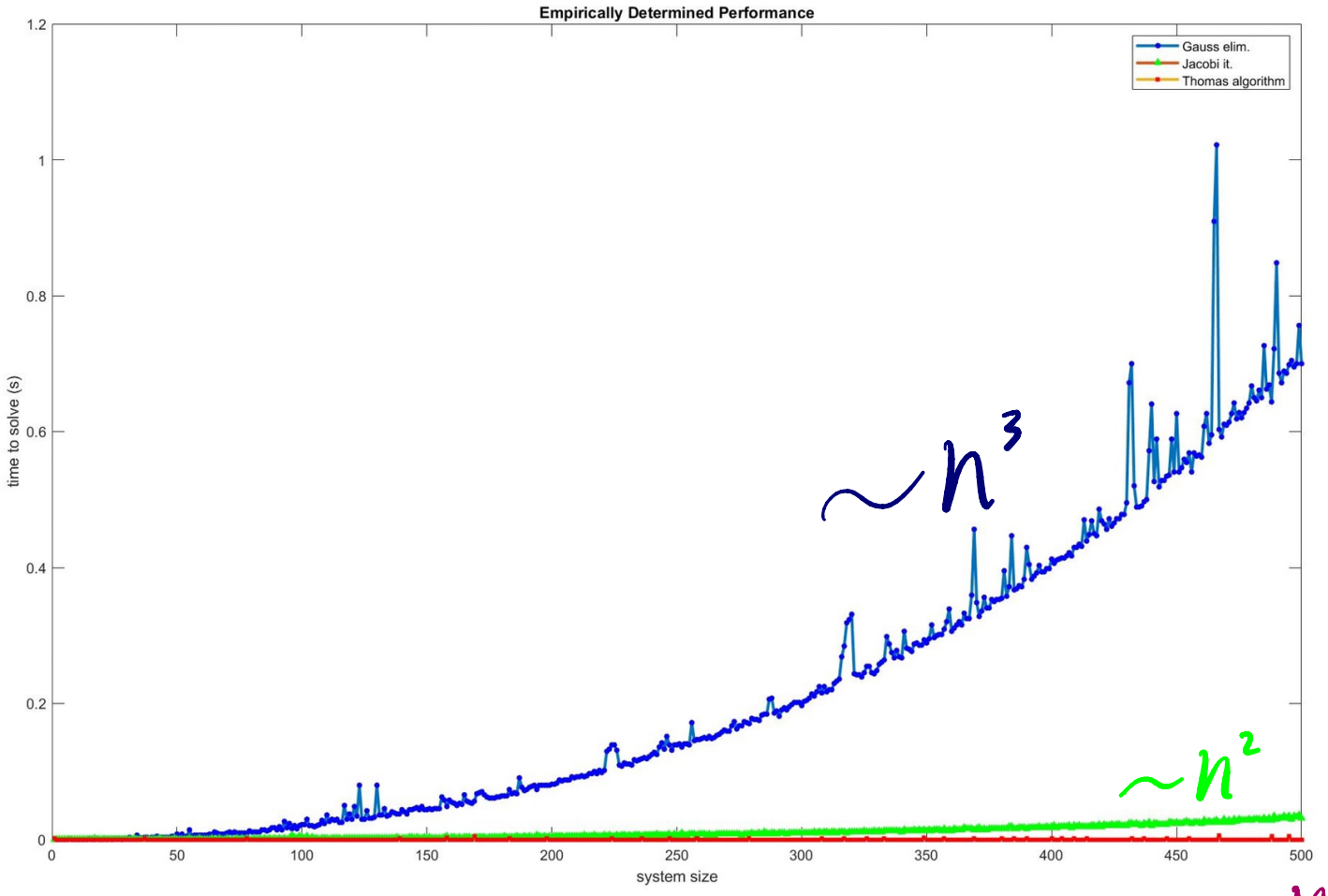
$$N_{TA} = (n-2+1)+(n-2+1)+(n-2+1)$$

$$+1+2(n-1-1+1) \leftarrow \text{Elim. } X$$

$$= 5n-4 \text{ times.}$$

This is even an order of magnitude less than what the Jacobi iteration requires.

As n increases, the Thomas increases linearly, whereas the Jacobi & Gaussian inflate at orders of n^2 & n^3. Hence, for a large n, the Thomas alg. gets comparably very efficient. ∎



↳ Rerun of the benchmark w/ increment of 1. For larger ns, the efficiency of the Thomas gets more attractive.

2. (a) $A\underline{u} = \underline{0}$

For $\underline{u}$ to be nontrivial, $\det(A) \neq 0$.

(b) $\det(A)$

$$= (V^2 - C_S^2 \sin^2\theta - C_A^2)(V^2 - C_A^2\cos^2\theta)(V^2 - C_S^2\cos^2\theta) - (C_S^2 \sin\theta\cos\theta)(C_S^2 \sin\theta\cos\theta)(V^2 - C_A^2\cos^2\theta)$$

$$= (V^2 - C_A^2\cos^2\theta)\left[V^4 - (C_S^2\sin^2\theta + C_S^2\cos^2\theta + C_A^2)V^2 + C_S^4\sin^2\theta\cos^2\theta + C_A^2C_S^2\cos^2\theta - C_S^4\sin^2\theta\cos^2\theta\right]$$

$$= (V^2 - C_A^2\cos^2\theta)\left[V^4 - (C_S^2\sin^2\theta + C_S^2\cos^2\theta + C_A^2)V^2 + C_A^2C_S^2\cos^2\theta\right]$$

$$= (V^2 - C_A^2\cos^2\theta)\left[V^4 - (C_S^2 + C_A^2)V^2 + C_A^2C_S^2\cos^2\theta\right]$$

$$= V^6 - (C_S^2 + C_A^2 + C_A^2\cos^2\theta)V^4 + (C_A^4\cos^2\theta + 2C_A^2C_S^2\cos^2\theta)V^2 - C_A^4C_S^2\cos^2\theta$$

This polynomial must not be zero.

(c) This is a polynomial of 6 degree, so it has 6 roots.

(d) Substituting $v = \pm C_A \cos\theta$ gives

$$A = \begin{bmatrix} C_A^2\cos^2\theta - C_S^2\sin^2\theta - C_A^2 & 0 & -C_S^2\sin\theta\cos\theta \\ 0 & 0 & 0 \\ -C_S^2\sin\theta\cos\theta & 0 & (C_A^2 - C_S^2)\cos^2\theta \end{bmatrix}$$

$$A\underline{u} = \begin{bmatrix} a_{11}u_x + a_{13}u_z \\ 0 \\ a_{31}u_x + a_{33}u_z \end{bmatrix} = \underline{0}$$

$\rightarrow$ u_y arbitrary

$$\therefore u_x = -\frac{a_{13}}{a_{11}}u_z$$

$$\therefore \underbrace{\left(a_{33} - \frac{a_{13}a_{31}}{a_{11}}\right)}_{0} u_z = \underbrace{0}_{\text{or } 0}$$

$$\therefore (C_A^2 - C_s^2)\cos^2\theta - \frac{C_s^4 \sin^2\theta\cos^2\theta}{C_A^2\cos^2\theta - C_s^2\sin^2\theta - C_A^2}$$

$$= \frac{\left(C_A^4\cos^4\theta - C_A^2C_s^2\sin^2\theta\cos^2\theta - C_A^4\cos^2\theta - C_A^2C_s^2\cos^4\theta + C_s^4\sin^2\theta\cos^2\theta + C_A^2C_s^2\cos^2\theta - C_s^4\sin^2\theta\cos^2\theta\right)}{C_A^2\cos^2\theta - C_s^2\sin^2\theta - C_A^2}$$

$$= \frac{C_A^2\cos^2\theta\,(C_A^2\cos^2\theta - C_s^2\sin^2\theta - C_A^2 - C_s^2\cos^2\theta + C_s^2)}{C_A^2(\cos^2\theta - 1) - C_s^2\sin^2\theta}$$

$$= \frac{C_A^2\cos^2\theta\left[C_A^2(\cos^2\theta - 1) - C_s^2(\sin^2\theta + \cos^2\theta) + C_s^2\right]}{-C_A^2\sin^2\theta - C_s^2\sin^2\theta}$$

$$= \frac{C_A^4\sin^2\theta\cos^2\theta}{(C_A^2 + C_s^2)\sin^2\theta}$$

$$= \frac{C_A^4\cos^2\theta}{C_A^2 + C_s^2}$$

$$= 0 \quad \therefore \cos^2\theta = 0 \quad \therefore \cos\theta = 0$$

$\cos\theta = 0 \longrightarrow u_x = 0$, u_z arbitrary

Therefore,

(i) When $\cos\theta \neq 0$,

u_y must be non-zero.

(u_y is arbitrary regardless of θ.)

(ii) When $\cos\theta = 0$, ($v = 0$)

either u_y or u_z must be non-zero.

for the wave mode $v = \pm C_A \cos\theta$.

(e)

Sound speed: $c_s = \sqrt{\frac{\gamma p}{\rho}} = \sqrt{\frac{\left(\frac{5}{3}\right)(1.38\times10^{-11}\ Pa)}{1.67\times10^{-21}\ kg/m^3}}$

$= 1.17\times10^{5}\ m/s$

Alfvén speed: $c_A = \sqrt{\frac{B^2}{\mu_0 \rho}} = \sqrt{\frac{(10^{-9}\ T)^2}{(4\pi\times10^{-7}\ N/A^2)(1.67\times10^{-21}\ kg/m^3)}}$

$= 2.18\times10^{4}\ m/s$

(f) $f(v)$

$$= v^6 - (C_s^2 + C_A^2 + C_A^2\cos^2\theta)v^4 + (C_A^4\cos^2\theta + 2C_A^2C_s^2\cos^2\theta)v^2 - C_A^4C_s^2\cos^4\theta$$

$f'(v)$

$$= 6v^5 - 4(C_s^2 + C_A^2 + C_A^2\cos^2\theta)v^3 + 2(C_A^4\cos^2\theta + 2C_A^2C_s^2\cos^2\theta)v$$

Matlab results on the following page.

Contents


```
% Midterm Problem 2(f)
% Find all roots using Newton's exact method

clear
clc
close all
```

Define the constants

```
gamma = 5/3; % adiabatic index
p = 1.38*1e-11; % plasma thermal pressure
rho = 1.67*1e-21; % plasma mass density
B = 1e-9; % magnitude of the local magnetic field
magcst = 4*pi*1e-7; % magnetic constant
theta = pi/4;   % plasma propagation angle
s = sin(theta);
ss = s^2;
c = cos(theta);
cs = c^2;
Cs = sqrt(gamma*p/rho); % sound speed
Css = Cs^2;
Ca = sqrt(B^2/(magcst*rho)); % Alfven speed
Cas = Ca^2;
```

Define the poly

```
o = 6; % Order of the poly
i = o;
A(:,1) = [1;0;-(Css+Cas*(1+cs));0;Cas*cs*(Cas+2*Css);0;-(Cas*cs)^2*Css]; % Coefficients
```

Newton's Method

```
maxit = 1000000;
tol = 1e-6;
v0 = 10000;    % Initial guess for root

for j = 1 : o
    if i == 6
        v0 = 10000;
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j) + A(4,j)*x^(o-2-j) + A(5,j)*x^(o-3-j) + A(6,j)*x^(o-4-j) + A(7,j)*x^(o-5-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j) + (o-j)*A(2,j)*x^(o-1-j) + (o-1-j)*A(3,j)*x^(o-2-j) + (o-2-j)*A(4,j)*x^(o-3-j) + (o-3-j)*A(5,j)*x^(o-4-j) + (o-4-j)*A(6
    elseif i == 5
        v0 = 20000;
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j) + A(4,j)*x^(o-2-j) + A(5,j)*x^(o-3-j) + A(6,j)*x^(o-4-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j) + (o-j)*A(2,j)*x^(o-1-j) + (o-1-j)*A(3,j)*x^(o-2-j) + (o-2-j)*A(4,j)*x^(o-3-j) + (o-3-j)*A(5,j)*x^(o-4-j);
    elseif i == 4
        v0 = -10000;
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j) + A(4,j)*x^(o-2-j) + A(5,j)*x^(o-3-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j) + (o-j)*A(2,j)*x^(o-1-j) + (o-1-j)*A(3,j)*x^(o-2-j) + (o-2-j)*A(4,j)*x^(o-3-j);
    elseif i == 3
        v0 = -20000;
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j) + A(4,j)*x^(o-2-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j) + (o-j)*A(2,j)*x^(o-1-j) + (o-1-j)*A(3,j)*x^(o-2-j);
    elseif i == 2
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j) + (o-j)*A(2,j)*x^(o-1-j);
    elseif i == 1
        F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j);
        Fprime = @(x) (o+1-j)*A(1,j)*x^(o-j);
    end % if

    [roots(j,1),it(j,1),success(j,1)] = newton_exact(F,Fprime,v0,maxit,tol);

    % Obtain next coefficients
    for k = 1 : o+1-j
        if k == 1
            A(k,j+1) = A(k,j);
        else
            A(k,j+1) = A(k,j) + roots(j,1)*A(k-1,j+1);
        end % if
    end % for

    i = i - 1;  % Update the order for the next poly

end % for

table(roots,it,success)
```

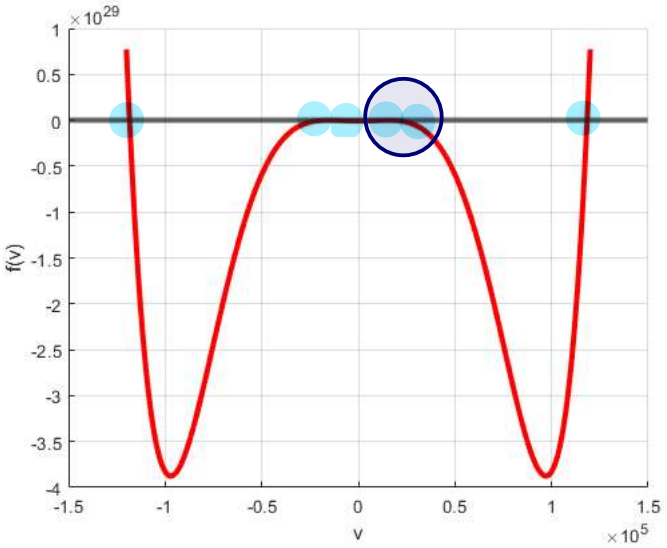
The roots are:

$v = \pm 1.53 \times 10^4$ m/s,
$\pm 1.54 \times 10^4$ m/s,
$\pm 1.18 \times 10^5$ m/s

```
ans =

  6×3 table

      roots       it    success
    __________    __    _______

         15301    10     true
         15436     5     true
        -15301    11     true
        -15436     4     true
    -1.1838e+05    7     true
     1.1838e+05    1     true
```

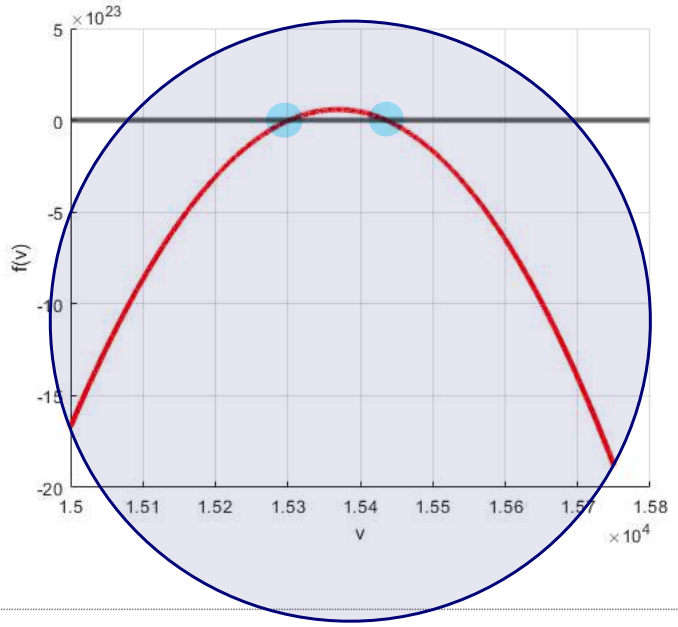
Plot the polynomial

```
figure
grid on
j = 1;
v = -120000:0.1:120000;
yline(0,'LineWidth',3)
hold on
plot(v,A(1,j)*v.^(o+1-j) + A(2,j)*v.^(o-j) + A(3,j)*v.^(o-1-j) + A(4,j)*v.^(o-2-j) + A(5,j)*v.^(o-3-j) + A(6,j)*v.^(o-4-j) + A(7,j)*v.^(o-5-j),'r','LineWidth',3);
xlabel('v');
ylabel('f(v)');
hold off

figure
grid on
v = 15000:0.1:15750;
yline(0,'LineWidth',3)
hold on
plot(v,A(1,j)*v.^(o+1-j) + A(2,j)*v.^(o-j) + A(3,j)*v.^(o-1-j) + A(4,j)*v.^(o-2-j) + A(5,j)*v.^(o-3-j) + A(6,j)*v.^(o-4-j) + A(7,j)*v.^(o-5-j),'r','LineWidth',3);
xlabel('v');
ylabel('f(v)');
hold off
```



$f(v)$

$-120000 \le v \le 120000$



← Close-up of $f(v)$

$-15000 \le v \le 15750$

Two roots are seen for sure.

Published with MATLAB® R2020b

3 (a) Solve $ax^2+bx+c=0$ analytically

```
function x = solveqe(A)

% Solves quadratic equation analitically

p = sqrt((A(2)^2/(4*A(1))-A(3))/A(1));
q = A(2)/(2*A(1));
x(1,:) = p - q;
x(2,:) = -p - q;

end % function
```

```
Not enough input arguments.

Error in solveqe (line 5)
p = sqrt((A(2)^2/(4*A(1))-A(3))/A(1));
```

Published with MATLAB® R2020b

$$ax^2+bx+c=0$$

$$\therefore a\left(x^2+\frac{b}{a}x\right)+c=0$$

$$\therefore a\left(x+\frac{b}{2a}\right)^2-\frac{b^2}{4a}+c=0$$

$$\therefore \left(x+\frac{b}{2a}\right)^2=\frac{\frac{b^2}{4a}-c}{a}$$

$$\therefore x=\pm\underbrace{\frac{b^2/4a-c}{a}}_{p}-\underbrace{\frac{b}{2a}}_{q}$$

3(a)

Contents


```
% Midterm Problem 3 (a)

clear
clc
close all
```

Coefficients of the quadratic equation

```
A = [2;-6;4];
```

Solve for roots

```
x = solveqe(A)
```

```
x =

     2
     1
```

Got x = 2, 1 using the quad. eq. Solver.

$$2x^2 - 6x + 4 = 0$$

$$\therefore 2(x^2 - 3x + 2) = 0$$

$$\therefore 2(x-2)(x-1) = 0$$

$$\therefore x = 2, 1$$

The solutions match!

Published with MATLAB® R2020b

3 (b)

Contents


```
% Midterm Problem 3 (b)
% Polynomial division algorithm

clear
clc
close all
```

Coefficients of the poly & divisor

```
o = 5; % Order of the poly
A(:,1) = [1;-15;85;-225;274;-120]; % Coefficients of the poly (Decreasing order)
N = 5; % Divide the poly by (x - N)
```

Division

Reordering the coefficients

```
for i = 1 : o+1
    Ap(i,1) = A(o-i+2,1);    % Increasing order
end % for

% Division
Qi(o+1,1) = Ap(o+1);
for i = o : -1 : 1
    Qi(i,1) = Ap(i) + N*Qi(i+1,1);    % Increasing order
end % for

R = Qi(1)   % Remainder

% Coefficients for Q
for i = 1 : o
    Q(i,1) = Qi(o-i+2,1);    % Decreasing order
end % for

disp('Qn-1 = ')
disp(Q)
```

Division algorithm

```
R =

     0

Qn-1 =
     1
   -10
    35
   -50
    24
```

Tested

$x^5 - 15x^4 + 85x^3 - 225x^2 + 274x - 120$ divided by $(x-5)$

$\rightarrow (x-5)(x^4 - 10x^3 + 35x^2 - 50x + 24)$

Q_{n-1}

Published with MATLAB® R2020b

3 (c) Find one root using Newton's approx. method

Contents

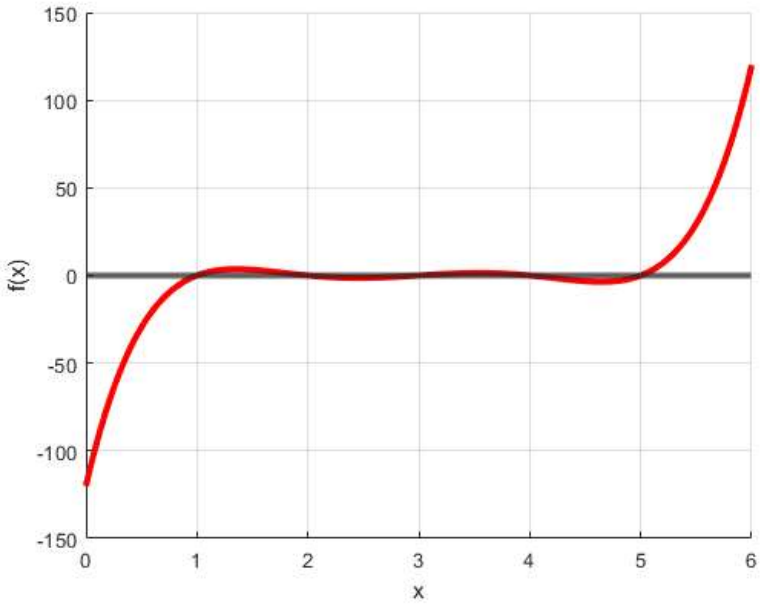

```
% Midterm Problem 3 (c)
% Find a root of the polynomial
% Use Newton's approx. method

clear
clc
close all
```

Coefficients of the poly

```
o = 5; % order of the poly
A(:,1) = [1;-15;85;-225;274;-120]; % Coefficients
```

plot

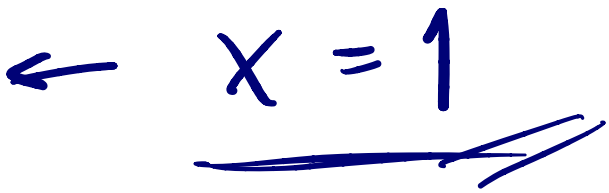
```
figure
grid on
j = 1;
x = 0:0.01:6;
yline(0,'LineWidth',3)
hold on
plot(x,A(1,j)*x.^(o+1-j) + A(2,j)*x.^(o-j) + A(3,j)*x.^(o-1-j) + A(4,j)*x.^(o-2-j) + A(5,j)*x.^(o-3-j) + A(6,j)*x.^(o-4-j),'r','LineWidth',3);
xlabel('x');
ylabel('f(x)');
```



Find the 1st root (domain = [-inf,inf))

```
F = @(x) A(1,j)*x^(o+1-j) + A(2,j)*x^(o-j) + A(3,j)*x^(o-1-j) + A(4,j)*x^(o-2-j) + A(5,j)*x^(o-3-j)+ A(6,j)*x^(o-4-j);
maxit = 10000;
tol = 1e-6;
x0 = 0.5;   % Initial guess for the root from the graph

[roots,it,success] = newton_approx(F,x0,maxit,tol);

table(roots,it,success)
```

```
ans =

  1×3 table

    roots    it    success
    _____    __    _______

      1      59     true
```



Published with MATLAB® R2020b

3 (d) $x^5-15x^4+85x^3-225x^2+274x-120$ divided by $(x-1)$

Contents


```
% Midterm Problem 3 (d)
% Polynomial division by the root

clear
clc
close all
```

Coefficients of the poly & divisor

```
o = 5; % Order of the poly
A(:,1) = [1;-15;85;-225;274;-120]; % Coefficients of the poly (Decreasing order)
N = 1; % Divide the poly by (x - N)
```

Division

Reordering the coefficients

```
for i = 1 : o+1
    Ap(i,1) = A(o-i+2,1);    % Increasing order
end % for

% Division
Qi(o+1,1) = Ap(o+1);
for i = o : -1 : 1
    Qi(i,1) = Ap(i) + N*Qi(i+1,1);    % Increasing order
end % for

R = Qi(1)   % Remainder

% Coefficients for Q
for i = 1 : o
    Q(i,1) = Qi(o-i+2,1);    % Decreasing order
end % for

disp('Q4 = ')
disp(Q)
```

Q4

```
R =

     0

Q4 =

     1
   -14
    71
  -154
   120
```

$(x-1)(x^4-14x^3+71x^2-154x+120)$

Published with MATLAB® R2020b

3(e)

Contents


```
% Midterm Problem 3 (e)
% Polynomial deflation
% This script works up to 6th order

clear
clc
close all
```

Order & Coefficients of the poly

```
o = 5; % Order of the poly
itn = o;
A(:,1) = [1;-15;85;-225;274;-120]; % Coefficients of the poly (Decreasing order)
```

Setup for iteration

```
maxit = 1000000;
tol = 1e-6;
x0 = 0.5;   % Initial guess for the root from the graph

if 2 < o && o < 7
```

```
    for k = 1 : o-2
```

Newton's approx. method to find the 1st root

```
        if itn == 6
            F = @(x) A(1,k)*x^(o+1-k) + A(2,k)*x^(o-k) + A(3,k)*x^(o-1-k) + A(4,k)*x^(o-2-k) + A(5,k)*x^(o-3-k) + A(6,k)*x^(o-4-k) + A(7,k)*x^(o-5-k);
        elseif itn == 5
            F = @(x) A(1,k)*x^(o+1-k) + A(2,k)*x^(o-k) + A(3,k)*x^(o-1-k) + A(4,k)*x^(o-2-k) + A(5,k)*x^(o-3-k) + A(6,k)*x^(o-4-k);
        elseif itn == 4
            F = @(x) A(1,k)*x^(o+1-k) + A(2,k)*x^(o-k) + A(3,k)*x^(o-1-k) + A(4,k)*x^(o-2-k) + A(5,k)*x^(o-3-k);
        elseif itn == 3
            F = @(x) A(1,k)*x^(o+1-k) + A(2,k)*x^(o-k) + A(3,k)*x^(o-1-k) + A(4,k)*x^(o-2-k);
        end % if
        [roots(k,1),it(k,1),success(k,1)] = newton_approx(F,x0,maxit,tol);
        table(roots,it,success)
```

```
ans =

  1×3 table

    roots    it    success
    _____    __    _______

      1      59     true


ans =

  2×3 table

    roots    it     success
    _____    ___    _______

      1       59     true
      2      216     true


ans =

  3×3 table
```

```
    roots    it     success
    _____    ___    _______

      1       59     true
      2      216     true
      3      237     true
```

Division

```
        N = roots(k,1)  % Difine the divisor
        for i = 1 : itn+1   % Reordering the coefficients
            Ap(i,k) = A(itn-i+2,k);    % Increasing order
        end % for
        % Division
        Qi(itn+1,k) = Ap(itn+1,k);
        for i = itn : -1 : 1
            Qi(i,k) = Ap(i,k) + N*Qi(i+1,k);    % Increasing order
        end % for
        R = Qi(1,k);   % Remainder
        % Coefficients for Q
        for i = 1 : itn
            Q(i,k) = Qi(itn-i+2,k);    % Decreasing order
        end % for
        for i = itn : -1 : 1
            A(i,k+1) = Q(i,k);
        end % for
        itn = itn - 1;  % Update for the order of Qn-1

disp('Qn-1 = ')
disp(Q)
```

```
N =

    1.0000

Qn-1 =
    1.0000
  -14.0000
   71.0000
 -154.0000
  120.0000
```

$x = 1$

$(x-1)(x^4-14x^3+71x^2-154x+120)$

```
N =

    2.0000

Qn-1 =
    1.0000    1.0000
  -14.0000  -12.0000
   71.0000   47.0000
 -154.0000  -60.0000
  120.0000         0
```

$x = 2$

$(x-1)(x-2)(x^3-12x^2+47x-60)$

```
N =

    3.0000

Qn-1 =
    1.0000    1.0000    1.0000
  -14.0000  -12.0000   -9.0000
   71.0000   47.0000   20.0000
 -154.0000  -60.0000         0
  120.0000         0         0
```

$x = 3$

$(x-1)(x-2)(x-3)(x^2-9x+20)$

```
    end % for
```

Coefficients of the quadratic equation

```
    P2 = [A(1,k+1);A(2,k+1);A(3,k+1)];
    x = solveqe(P2)
```

```
x =

    5.0000
    4.0000
```

$x = 5, 4$

```
elseif o == 2
    k = 2;
    x = solveqe(A)
elseif o == 1
    k = 1;
    [roots,it,success] = newton_approx(F,x0,maxit,tol)
end % if
```

Published with MATLAB® R2020b

$\therefore x = 1, 2, 3, 4, 5$