



تمرین کامپیوتری اول

محاسبات عددی دکتر آریانیان

سوال اول

تئوری:

برای باینری کردن هر عدد لازم است تا آن را مکررا تقسیم بر ۲ کنیم (جواب هر مرحله تقسیم را مجددا تقسیم بر ۲ کنیم). تا هنگامی که خارج قسمت برابر صفر شود و در هر مرحله تقسیم باقی مانده را که یا برابر صفر و یا برابر یک است را به ترتیب از سمت راست به چپ قرار دهیم.

همچنین برای xor کردن دو عدد لازم است تا باینری شده‌ی آن دو را زیر هم نوشته و اگر هر دو کاراکتر ستونی با هم برابر بودند زیر آن ستون صفر قرار داده و در غیر این صورت یک قرار دهیم در این صورت رشته‌ی صفر و یک به دست آمده جواب خواهد بود.

برای تبدیل عدد باینری به دسیمال نیز باید هر کاراکتر به از عدد باینری را به ترتیب از راست به چپ ضربدر دو به توان اندیسش کنیم که این اندیس‌ها از صفر شروع می‌شوند در هر مرحله عدد به دست آمده را با مجموع عدد های به دست آمده قبلی جمع کنیم.

برای مثال برای تبدیل عدد ۷ به باینری و مجددا تبدیل آن به دسیمال به صورت زیر عمل می‌کنیم:

$$6/2 = 3 \rightarrow 0$$

$$3/2 = 1 \rightarrow 1$$

$$1/2 = 0 \rightarrow 1$$

result : 110

110

$$0*2^0 + 1*2^1 + 1*2^2 = 2 + 4 = 6$$

توضیحات:

در صورتی که دنباله‌ی شما فقط یک عدد یکتا داشته باشد برنامه به شما یک پیغام خط‌نمایش می‌دهد زیرا باید حداقل دو عدد برای عمل xor وجود داشته باشد همچنین در صورتی که هیچ عدد یکتا نداشته باشد برنامه عدد صفر را به شما نشان خواهد داد.

همچنین فرمت نهایی جواب به صورت زیر خواهد بود:

finalResult = 3x6						
عدد اول	13	13	13	15	15	17
عدد دوم	15	17	6	17	6	6
نتیجه عملگر	2	28	11	30	9	23

توضیح کد:

خط ۱: در ابتدا تمامی پنجره‌ها بسته شده و تمامی دستورات و داده‌های پیشین پاک می‌شوند

خط ۳: طول دنباله‌ای که قرار است از کاربر دریافت شود، از کاربر، توسط `command window` دریافت شده و در متغیر `initialNumberSize` ذخیره می‌شود.

خط ۴: ماتریسی با ابعاد یک در (سایز دنباله) تعریف می شود (initial numbers) تا اعضای دنباله در آن نگهداری شوند.

خط ۵: متغیری برای نگهداری تعداد اعضای یکتای دنباله تعریف می شود از آن جا که هنوز این تعداد را نمی دانیم این عدد برابر صفر است

خط ۶: از آنجا که نیاز داریم اعداد یکتا را در ماتریس دیگری ذخیره کنیم ولی هم اکنون سایز این ماتریس برایمان مشخص نیست ماتریسی تعریف می کنیم با سایز دنباله ی اصلی تا بعدا اعضا یکتا را در یک ماتریس با سایز صحیح بریزیم طبیعی است که همیشه تمامی خانه های این ماتریس موقتی که در اینجا تعریف می شود پر نمی شود.

خطوط ۸ تا ۱۰: فرایند پر کردن خانه های ماتریس نگهدارنده ی اعضای دنباله با یک حلقه آغاز می شود اعضا از کاربر initial numbers توسط command window تحویل گرفته می شوند و به ترتیب درون ماتریس تعریف شده ی ذخیره می شوند.

خط ۱۲: حال ماتریس دنباله ی اصلی به کاربر نمایش داده می شود.

خطوط ۱۴ تا ۲۰: اعداد یکتای ماتریس اصلی توسط یک حلقه و شرط پیدا می شوند بدین صورت که در دور ۱ ام حلقه یک ماتریس با سایز ماتریس اصلی تشکیل شده که تمامی اعضای آن برابر عضو ۰ ام دنباله می باشد حال این ماتریس تولید شده با ماتریس دنباله برابر قرار داده می شود بنابراین ما با به تعدادی که از آن عضو در دنباله ی اصلی داریم ، عدد یک تحویل می گیریم حال با استفاده از تابع sum این یک ها جمع زده می شوند و اگر جمع آن ها بیشتر از یک شود که به معنای وجود چندین تا از آن عضو است به دور بعدی بعدی حلقه می رویم اما اگر مجموع آن ها یک باشد که بدین معنی است که آن عضو یک عضو یکتا بوده این عضو در ماتریسی که در ابتدا تحت عنوان اعداد تکرار نشده تعریف کردیم ذخیره می شود همچنین یکی به تعداد اعداد یکتا نیز اضافه می شود.

خط ۲۲: از آنجایی که هم اکنون تعداد اعداد یکتای دنباله را می دانیم و ماتریس اعداد غیرتکراری بی جهت بزرگ است (توضیح داده شد که چرا این تابع بی جهت بزرگ است. خط ۶) اعداد ماتریس اعداد غیر تکراری را به ماتریس با ابعاد یک در (تعداد اعداد یکتا) منتقل کرده و نام آن را ماتریس اعداد نهایی می گذاریم.

خطوط ۲۷ تا ۲۹: قرار است اگر دنباله فاقد عدد یکتا بود به کاربر عدد صفر نشان داده شود پس با یک شرط این موضوع را چک می کنیم که اگر ماتریس اعداد نهایی خالی بود عدد صفر نشان داده شود و برنامه تمام شود.

خطوط ۳۲ تا ۳۶: با یک شرط چک می شود که اگر ماتریس اعداد یکتا فقط یک عدد داشت پیغام خطایی که در بخش توضیحات به آن اشاره شد نمایش داده شود و برنامه تمام شود.

خطوط ۳۶ تا ۴۰: مراحل ۴۰ کردن که به ترتیب شامل تابع های باینری ساز ، تابع XOR کننده و تابع تبدیل باینری به عدد است اجرا شوند. در ذیل این توابع به صورت جداگانه توضیح داده شده اند. همچنین کد برنامه تا به اینجا در تصویر زیر قابل مشاهده است.

```

1 clc,clear,close all
2
3 initialNumbersSize = input("Please input the length of your sequence: ");
4 initialNumbers= zeros(1,initialNumbersSize);
5 finalNumbersSize=0;
6 notRepeated= zeros(1,initialNumbersSize);
7
8 for i= 1:initialNumbersSize
9     initialNumbers(1,i) = input(i+.number: " );
10 end
11
12 initialNumbers
13
14 for i= initialNumbers
15     checking= (i==initialNumbers);
16     if sum(checking) == 1
17         finalNumbersSize=finalNumbersSize+1 ;
18         notRepeated(1,finalNumbersSize)=i;
19     end
20 end
21
22 finalNumbers(1,1:finalNumbersSize)=notRepeated(1,1:finalNumbersSize)
23
24 if isempty(finalNumbers)
25     disp(0);
26     return
27 end
28
29 if size(finalNumbers,2)==1
30     disp('you have just one unrepeated number!!');
31     return
32 end
33
34 binary= binarymaker(finalNumbers)
35 resultAfterXor= xorMaker(binary)
36 finalResult=numbermaker(resultAfterXor,finalNumbers)

```

تابع باینری ساز، خطوط ۳۷ تا ۵۱: این تابع که یک ماتریس یک در n تحويل گرفته و یک ماتریس k در n تحويل می دهد که n تعداد اعداد دنباله‌ی یکتا و k تعداد کاراکتر لازم برای باینری کردن بزرگترین عدد دنباله است از آنجا که تعداد کاراکتر لازم برای باینری کردن هر عدد با لگاریتم در پایه ۲ آن عدد ارتباط دارد لگاریتم تمامی اعداد دنباله را گرفته و بزرگترین آن‌ها را به عنوان سایز باینری‌ها نام‌گذاری می‌کنیم که در واقع همان k است تعداد عدد‌های دنباله‌ی ارسالی را نیز با تابع سایز مطلب میابیم حال که ابعاد ماتریسی که قرار است برگردانده شود را می‌دانیم ماتریس را تشکیل داده و باینری می‌نامیم با استفاده از یک حلقه که اعداد دنباله را به ترتیب اندیس آن‌ها پیماش می‌کند اعداد را باینری کرده و در ردیف مربوط به خود می‌گذریم بدین صورت که ابتدا عنصر یکم از دنباله دریافتی را می‌گیریم و می‌دانیم باید در ردیف یکم از ماتریس جواب قرار گیرد حال با استفاده از یک حلقه‌ی while عنصر دریافتی را مکررا تقسیم بر ۲ کرده باقی مانده را در خانه‌ی ستون آخر ردیف اول قرار داده و به عقب میابیم تا خارج قسمت برابر صفر گردد و همچنین دیگر خانه‌ای بر عقب آمدن نباشد و سپس به دور بعد حلقه رفته و این کار را برای اندیس دوم انجام می‌دهیم. در نهایت ماتریس جواب که شامل عدد باینری شده در هر ردیف است برگردانده می‌شود

binary = 4x5

باینری عدد اول	0	1	1	0	1
باینری عدد سوم	0	1	1	1	1
	1	0	0	0	1
	0	0	1	1	0

تابع xor کننده، خطوط ۵۳ تا ۶۶ : در ابتدا ماتریس اعداد باینری شده به این تابع داده می شود. از آنجا که می دانیم عملگر باید دو به دو بر روی اعداد اعمال شود. پس ماتریسی که این اعداد تحويل می دهد باید ابعادی در اندازه (انتخاب ۲ از تعداد اعداد) در (تعداد کاراکتر لازم برای باینری کردن بزرگترین عدد) باشد. پس انتخاب را حساب کرده و این ماتریس را ساخته. در ادامه با استفاده از دو حلقه که حلقه اول یک ردیف ماتریس اعداد باینری (که در واقع یک عدد است) را انتخاب کرده و حلقه دوم ردیف های بعدی ماتریس تا انتهای را به ترتیب انتخاب می کند اعداد را دو به دو xor می کنیم. عملگر xor نیز به این صورت تعریف شده که ردیف اول انتخاب شده را با ردیف دوم انتخاب شده را متضاد (عملگر \sim) قرار می دهیم. در این صورت اگر هر کاراکتر از دو عدد باینری برابر باشند. صفر تولید شده و در غیر این صورت یک و در نهایت ماتریسی تک ردیفه ای که از این عملگر تولید شد در ردیف های ماتریس نهایی جواب قرار می گیرند.

```

37 function binary = binarymaker(numbers)
38
39     binarySize= max(floor(log2(numbers)+1));
40     numOfNumbers = size(numbers,2);
41     binary= zeros(numOfNumbers,binarySize);
42
43     for i= 1:numOfNumbers
44         number= numbers(1,i);
45         j=0;
46         while number ~= 0
47             [number,binary(i,binarySize-j)]= quorem(sym(number),sym(2));
48             j=j+1;
49         end
50     end
51 end
52
53 function result = xorMaker(binary)
54
55     resultSize= factorial(size(binary,1))/(2*factorial(size(binary,1)-2));
56     result= zeros(resultSize,size(binary,2));
57     resultSize=0;
58
59     for i=1:size(binary,1)
60         for j=i+1:size(binary,1)
61             resultSize=resultSize+1;
62             result(resultSize,1:size(binary,2))=
63             (binary(i,1:size(binary,2))~=binary(j,1:size(binary,2)));
64         end
65     end
66 end

```

تابع تبدیل باینری به عدد، خطوط ۶۸ تا ۸۸: این تابع ماتریس نتیجه عملگر که یک ماتریس باینری است را تحويل گرفته و سپس با استفاده از دو حلقه که اولی یک ردیف از این ماتریس را انتخاب کرده و دومی به ترتیب ۰ و ۱ های آن را ردیف را از انتهای آن بخواهد. همان ردیف به ترتیب ضربدر دو به توان شماره ستون خود شده و به مقدار هایی که از قبل برای همان ردیف به دست آمده بودند افزوده می شود. و در نهایت عدد کلی به دسته آمده از هر ردیف وارد یک ستون ماتریس نهایی جواب خواهد شد. ماتریس نهایی جواب سه ردیف خواهد داشت که ردیف آخر آن مخصوص نتایج است و دو ردیف بالایی مخصوص نشان دادن این که کدام دو عدد xor شده اند پس با استفاده از دو حلقه و همانند الگوریتم تابع xor کننده اعداد را دو به دو انتخاب کرده و در ردیف های اول و دوم هر ستون ماتریس نهایی می گذاریم برای اینکار متغیر شمارنده تعريف شده تا در هر مرحله عدد آن یکی بالاتر رفته و شماره ستون ما تغییر کند و نیازی به تعريف حلقه ی سوم نباشد. از آن جایی که تمامی توابع بر اساس ترتیب ماتریس اعداد یکتا عمل کرده اند طبیعی است که اینجا هم اعداد صحیح بالای جواب صحیح قرار می گیرند.

finalResult = 3x6						
عدد اول	13	13	13	15	15	17
عدد دوم	15	17	6	17	6	6
نتیجه عملگر	2	28	11	30	9	23

```

68 function finalresult = numbermaker(result,finalNumbers)
69
70     finalresult=zeros(3,size(result,1));
71
72     for j=1:size(result,1)
73         for i=0:(size(result,2)-1)
74             finalresult(3,j)= finalresult(3,j)+(result(j,size(result,2)-i)*(2^i));
75         end
76     end
77
78     counter=1;
79
80     for i=1:size(finalNumbers,2)
81         for j=i+1:size(finalNumbers,2)
82             finalresult(1,counter)=finalNumbers(1,i);
83             finalresult(2,counter)=finalNumbers(1,j);
84             counter=counter+1;
85         end
86     end
87 end

```

سوال دو^م

مقدمه (خطوط ۱ تا ۱۲) :

توضیحات: این بخش حاوی جدول داده‌ی صورت سوال است و توانایی تغییر دارد و همچنین این برنامه توانایی گرفتن هر جدول با هر تعداد داده‌ای را دارد و لازم نیست تا تعداد ردیف‌های جدول حتیماً ۳ یا ۴ تا باشد.

توضیح کد:

خط ۱ : در ابتداء تمامی پنجره‌ها بسته شده و تمامی دستورات و داده‌های پیشین پاک می‌شوند.

خط ۳ : کامنت جدا کننده

خط ۴ تا ۵ : تبدیل x به یک متغیر جبری به جای متغیر عددی و نمایش عنوان بخش

خط ۷ تا ۹ : ماتریسی که داده‌ی سوال و در واقع همان جدول صورت سوال است، تعریف می‌شود.

خط ۱۱ : از آن جا که ماتریس داده‌های می‌تواند هر تعداد ردیفی داشته باشد در ابتداء با استفاده از تابع سایز مطلب این عدد برای استفاده‌های بعدی پیدا می‌شود و آن را سایز داده می‌نامیم.

خط ۱۲ : از آن جا که داده‌های جدول باید وارد یک چند جمله‌ای درجه ۷ شود با تابعی به نام matrix maker که در ذیل توضیح داده شده داده‌های جدول را به صورت ضرایب چند جمله‌ای در آورده و سپس این ضرایب به همراه جواب معادله‌ها را وارد ماتریس می‌کنیم. نام این ماتریس را ماتریس ضرایب می‌نامیم.

تابع matrix maker ، خطوط ۹۰ تا ۹۹: این تابع جدول داده صورت سوال و سایز داده را تحويل می‌گیرد و در انتهای ماتریس ضرایب را تحويل می‌دهد. ابتداء یک ماتریس با ابعاد (سایز داده) در (سایز داده + ۱) می‌سازیم حال با استفاده از یک حلقه، ستون اول جدول داده‌ی سوال را به توان (سایزه داده - اندیس ستون ماتریس ضرایب) می‌رسانیم و در ستون‌های ۱ تا (سایزه داده) ماتریس ضرایب قرار می‌دهیم و در ستون آخر ماتریس ضرایب نیز ستون دوم جدول داده‌ی صورت سوال را قرار می‌دهیم حال ماتریس ضرایب که برای روش‌های گاوس و ژاکوبی استفاده می‌شود، تولید شد.

-----Information-----

data = 3x2

5.0000	106.8000
8.0000	177.2000
12.0000	279.2000

جدول داده‌ی سوال

matrix = 3x4

25.0000	5.0000	1.0000	106.8000
64.0000	8.0000	1.0000	177.2000
144.0000	12.0000	1.0000	279.2000

ماتریس ضرایب افزوده بر اساس این جدول

بخش الف (خطوط ۱۴ تا ۱۶) :

تئوری:

در روش حذفی گاوس پس از تشکیل ماتریس ضرایب افزوده که همان ماتریس ضرایب است با یک ستون اضافه تر که در این ستون جواب های معادله ها نوشته می شوند. حال ردیف اول را ضربدر (منفی مقدار درایه اول ردیف ا تقسیم بر مقدار درایه اول ردیف ۱) برای ۱ به ازای ۲ تا ۷ می کنیم و به ردیف ۲ ام اضافه می کنیم سپس برای ردیف دوم ، ردیف دوم را ضربدر (منفی مقدار درایه دوم ردیف ا تقسیم بر مقدار درایه دوم ردیف ۲) به ازای ۳ های ۳ تا ۷ می کنیم و به ردیف ۳ ام اضافه می کنیم و بدین ترتیب این کار را ادامه می دهیم تا ماتریس بالا مثلثی ساخته شود.

توضیح کد:

خط ۱۶ :تابع سرعت موشک برای اولین بار با استفاده از روش حذفی گاوس تولید می شود. این فانکشن، ماتریس ضرایب و سایز داده را تحويل گرفته و تابع سرعت موشک را تحويل می دهد. از این پس برای توابع مطلب کلمه ی (فانکشن) استفاده می شود. فانکشن حذفی گاوس در ذیل توضیح داده شده.

-----Gauss Method-----

144.0000	12.0000	1.0000	279.2000
0	2.9167	0.8264	58.3278
0	0	-0.2000	-0.2171

results for guass method in order:

0.2905 19.6905 1.0857

مقدار ضریب به ترتیب از چپ به راست

ماتریس بالا مثلثی گاوس

فانکشن حذفی گاوس، خطوط ۱۰۱ تا ۱۰۴ : در ابتدا X به عنوان متغیر جبری تعریف شده و اطلاعات ماتریس ضرایب در ماتریس دیگری به نام ماتریس گاوس ریخته می شود تا داده های ماتریس ضرایب برای باقی روش ها ، دست نخورد بماند.

فانکشن حذفی گاوس، خطوط ۱۰۵ تا ۱۱۹ : لازم است تا ابتدا ماتریس بالا مثلثی گاوس تولید شود بدین منظور با استفاده از یک حلقه یک ردیف انتخاب شده حال نیاز است تا محور گیری انجام شود با استفاده از تابع \max مطلب اندیس ردیفی که دارای بزرگترین مقدار در آن ستون (ستون با همان شماره ردیف انتخاب شده) است یافت می شود حال مقادیر آن ردیف را در یک متغیر موقتی ذخیره می کنیم و آن را با ردیفی که هم اکنون در ردیف شمارنده ی حلقه وجود دارد جا به جا می کند. سپس با ضرب ردیف انتخاب شده در یک ضریب مناسب که در بخش تئوری توضیح داده شد آن را از ردیف های پایین کم می کنیم تا ماتریس بالا مثلثی تشکیل شود.

فانکشن حذفی گاوس، خطوط ۱۲۱ تا ۱۳۹ : ماتریس بالا مثلثی گاوس تشکیل شده و به کاربر نمایش داده می شود حال با یک حلقه و انتخاب ردیف های ماتریس از ردیف سوم به بالا انتخاب کرده از آنجا که هر ردیف ماتریس یک

معادله است هر معادله را به این صورت حل می کنیم که مقادیر که باید کم شوند با استفاده از حلقه در یک متغیر به نام minus ریخته می شوند و از جواب معادله که ستون آخر ماتریس مثلثی است کم می شوند و در نهایت جواب تقسیم بر ضریب مجهول که همان شمارنده ی حلقه است و بنابراین مجهولات به ترتیب پیدا می شوند. در نهایت این ضرایب که در ماتریس result ذخیره شده اند به ترتیب ضربدر ایکس به توان (سایز داده - اندیس) شده وتابع نهایی را می سازند.

بخش ب (خطوط ۱۸ تا ۲۰):

توضیحات: از آنجا که ماتریس داده صورت سوال قطری اکید نیست پس پاسخ روش ژاکوبی همگرا نخواهد شد و جواب اشتباهی در نهایت نمایش داده می شود اما در این فانکشن این بخش دو آرگومان تحت عنوان های وجود دارند که با تغییر اولین آن ها به ماتریس افزوده دلخواه که محور گیری شده (ماتریس numOfData و matrix مربعی ضرایب با یک ستون بیشتر مخصوص جواب ها) و تغییر دومین آن ها به تعداد ردیف ماتریس افزوده دلخواه، جواب صحیح را از این تابع تحويل گرفت. همچنین آرگومان های سوم و چهارم ماتریس حدس های اولیه هر متغیر و تعداد دفعات تکرار روش ژاکوبی است که آن را نیز می توان به صورت دلخواه تغییر داد.



تئوری:

در روش ژاکوبی به تعداد معادلات دستگاه، فرمول برای پیدا کردن هر کدام از ضرایب تولید می شود و سپس با جایگذاری حدس های اولیه در این فرمول ها جواب تولید می شود و سپس جواب ها تولید شده مجددا در فرمول جاگذاری می شود تا به جواب واقعی همگرا شوند البته شرط همگرا شدن روش ژاکوبی آن است که ماتریس ضرایب دستگاه قطری مطلق باشد یعنی قدر مطلق هر درایه ی قطر اصلی از مجموع قدر مطلق سایر درایه های همان سطر به طور اکید بیشتر باشد در زیر روش تولید فرمول های روش ژاکوبی آمده است:

$$a_{11}x + a_{12}y + a_{13}z = b_1 \rightarrow x = \frac{1}{a_{11}}(b_1 - a_{12}y - a_{13}z)$$

$$a_{21}x + a_{22}y + a_{23}z = b_2 \rightarrow y = \frac{1}{a_{22}}(b_2 - a_{21}x - a_{23}z)$$

$$a_{31}x + a_{32}y + a_{33}z = b_3 \rightarrow z = \frac{1}{a_{33}}(b_3 - a_{31}x - a_{32}y)$$

توضیح کد:

خط ۲۰: آرگومان های فانکشن مانند ماتریس افزوده، ماتریس حدس های اولیه و تعداد تکرار ها تعیین می شوند و تابع سرعت بازگردانده می شود

تعداد تکرار حدس اولیه

with Initail numbers 0 0 0 and 25 repeats.
results for jacob method in order:

1.0e+14 *

0.1210	0.5630	7.2589
--	--	--

ضرایب تابع با روش ژاکوبی

فانکشن ژاکوبی، خطوط ۱۴۷ تا ۱۴۱: نمایش جبری کردن متغیر ایکس ساخت ماتریس جواب و قرار دادن ماتریس ضرایب اصلی در ماتریس دیگری به نام ژاکوبی تا اطلاعات ماتریس اصلی دست نخورده بماند.

فانکشن ژاکوبی، خطوط ۱۴۹ تا ۱۶۴: با استفاده از دو حلقه که اولی تعداد تکرار روش ژاکوبی و دومی برای پیاده سازی معادله های روش ژاکوبی است جواب ها را می یابیم با استفاده از حلقه دوم در هر مرحله تمامی ضرایب ضربدر مقادیر جواب نهایی می شوند (که در ابتداء همان حدس اولیه است) و با هم جمع می شوندو سپس مقدار داریه ایه که شماره آن با شماره ردیف انتخاب شده برابر است از این حاصل کم می شود و در نهایت این حاصل از درایه چهارم آن ردیف که همان جواب دستگاه است کم می شود و این حاصل تقسیم بر ضریب آن داریه ای می شود که شماره آن با ردیف انتخاب شده برابر است بدین ترتیب ضرایب محاسبه می شوند. و در نهایت همانند بخش الف تابع تولید می شود.

بخش ج (خطوط ۲۲ تا ۲۴):

تئوری:

برای تولید جمله ای درون یا $P(x)$ از فرمول زیر استفاده کرده که در آن f_i ها همان مقادیر تابع در نقاط جدول هستند و همچنین x ها همان نقاط داده شده ای جدول هستند همچنین تعریف L_i نیز در زیر آمده:

$$P(x) = L_0(x)f_0 + L_1(x)f_1 + L_2(x)f_2 + \dots + L_n(x)f_n$$

$$L_i(x) = \frac{(x - x_0)(x - x_1)\dots(x - x_{i-1})(x - x_{i+1})\dots(x - x_n)}{(x_i - x_0)(x_i - x_1)\dots(x_i - x_{i-1})(x_i - x_{i+1})\dots(x_i - x_n)}$$

توضیح کد:

خط ۲۴: ماتریس ضرایب و سایز داده به فانکشن داده می شود و تابع سرعت موشک با استفاده از روش درونیابی لاگرانژ تولید می شود. فانکشن لاگرانژ در ذیل توضیح داده شده.

تابع تولید شده ای لاگرانژ

-----Lagrange Method-----

$$\frac{349(x-5)(x-8)}{35} - \frac{443(x-5)(x-12)}{30} + \frac{178(x-8)(x-12)}{35}$$

$$\frac{61x^2}{210} + \frac{827x}{42} + \frac{38}{35}$$

ساده شده ای همان تابع

results for lagrange method in order:
0.2905 19.6905 1.0857

ضرایب همان تابع

فانکشن لاگرانژ خطوط ۱۶۶ تا ۱۷۱ : از آنجا که برایت تولید ضرایب L هر بار باید از یکی ضرایب پریید ابتدا ماتریسی با اعداد ۱ تا سایز داده می سازیم سپس ماتریس دیگری به نام اندیس نهایی می سازیم تا شماره اندیس ضرایب به جز ضریبی که باید از آن پریید ذخیره شود و در نهایت متغیر ایکس را جبری کرده وتابع جواب را تعریف می کنیم.

فانکشن لاگرانژ خطوط ۱۷۲ تا ۱۹۲ : حال برای تولید هر جمله از تابع نهایی از دو حلقه کمک می گیریم حلقه اول مشخص می کند کدام ضریب برای تولید L باید حذف شود بدین صورت که از ماتریس اندیس ضرایب که در ابتدا تعریف شد هر دفعه یک عدد را حذف کرده و باقی در ماتریس اندیس نهایی می ریزد حال حلقه دوم که شمارنده اش همان اعداد داخل ماتریس اندیس نهایی هستند مقادیر صورت L و مخرج آن را بر حسب ایکس و اعداد ماتریس ضرایب تولید می کند و بر هم تقسیم می کند تا L هر ردیف جدول داده صورت سوال ساخته شود در نهایت هر L ضربدر مقدار خود که همان ستون دوم جدول داده صورت سوال است شده و به مقادیر از تابع درونیابی که در دورهای قبل حلقه محاسبه شده بودند افزوده می شود حال تابع درونیابی شده آماده است ابتدا آن را نمایش داده سپس با فانکشن `simplify` ضرایب آن را ساده و تبدیل به عدد کرده و با استفاده از تابع `sym2pol` مطلب ضرایب این تابع استخراج شده و نمایش داده می شود و همچنین خود تابع نیز بازگردانده می شود.

بخش د(خطوط ۲۶ تا ۳۸):

تئوری:

تفاضلات تقسیم شده ی نیوتون مرتبع اول بین x_i و x_{i+1} به صورت زیر تعریف می شود.

$$f[x_i, x_{i+1}] = \frac{f_{i+1} - f_i}{x_{i+1} - x_i}$$

تفاضلات تقسیم شده ی مرتبه دوم نیز به این صورت تعریف می شوند:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_i, x_{i+1}] - f[x_{i+1}, x_{i+2}]}{x_i - x_{i+2}}$$

و تفاضلات تقسیم شده ی نیوتون مرتبه k به این صورت تعریف می شود:

$$f[x_{j-1}, x_j, \dots, x_{j+k}] = \frac{f[x_{j-1}, \dots, x_{j+k-1}] - f[x_j, \dots, x_{j+k}]}{x_{j-1} - x_{j+k}}$$

همچنین می توان تفاضلات تقسیم شده را برای زیبایی و خوانایی بیشتر بر اساس همین فرمول در یک جدول نمایش داد.

و فرمول چند جمله ای دورنیاب نیوتون به صورت زیر است:

$$P(x) = f_0 + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \dots + (x - x_0)(x - x_1)\dots(x - x_{n-1})f[x_0, x_1, \dots, x_n]$$

توضیح کد:

خط ۲۸ : این فانکشن ماتریس ضرایب و سایز داده را گرفته و تابع سرعت موشک با استفاده از روش درونیابی نیوتون تولید می کند. فانکشن نیوتون در ذیل توضیح داده شده.

Newton Method				
5.0000	106.8000	0	0	
0	0	23.4667	0	
8.0000	177.2000	0	0.2905	
0	0	25.5000	0	
12.0000	279.2000	0	0	

جدول تفاضلات تقسیم شده

$$\frac{352x}{15} + \frac{61(x-5)(x-8)}{210} - \frac{158}{15}$$

تابع تولید شده توسط روش نیوتن

$$\frac{61x^2}{210} + \frac{827x}{42} + \frac{38}{35}$$

ساده شده‌ی همان تابع

results for newton method in order:
0.2905 19.6905 1.0857

ضرایب همان تابع

فانکشن نیوتن خطوط ۱۹۴ تا ۱۹۹: ابتدا متغیر ایکس جبری شده و از آن جا که برای پیدا کردن تفاضلات تقسیم شده‌ی نیوتن رسم یک جدول که در ستون‌های اول دوم آن به ترتیب نقاط و مقدار تابع در آن نقاط وجود دارند و سپس در باقی ستون‌های تفاضلات تقسیم شده بر حسب ستون‌های قبلی خودشان تشکیل می‌شوند، نیاز است این جدول را نیز تحت عنوان جدول نیوتن تشکیل می‌دهیم همچنین برای زیبایی بصری بخش‌های خالی جدول را با صفر پر می‌کنیم بنابراین به جدولی با ابعاد (دو برابر سایز داده - ۱) در (۱ + سایز داده) نیاز داریم.

فانکشن نیوتن خطوط ۲۰۱ تا ۲۰۷: حال باید جدول نیوتن را پر کنیم برای این کار با استفاده از دو حلقه که اولی انتخاب کننده هر ستون از جدول و دومی انتخاب کننده‌ی ردیف از ستون انتخاب شده حال بر حسب فرمول تفاضلات تقسیم شده‌ی نیوتن و با استفاده از شمارنده‌ی انتخاب کننده‌ی ستون به ستون های قبل مورد نظر دسترسی پیدا می‌کنیم و محاسبات را انجام می‌دهیم و سپس مقدار محاسبه شده را در ستون و ردیف انتخاب شده می‌گذاریم. حال جدول پر شده را نمایش می‌دهیم.

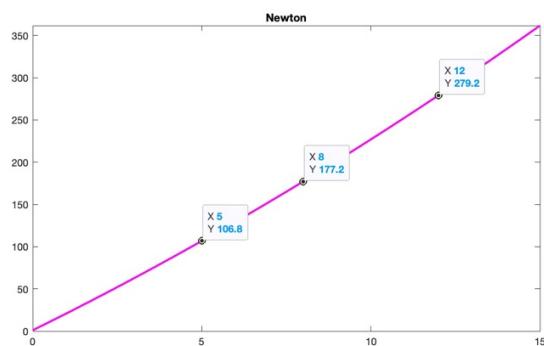
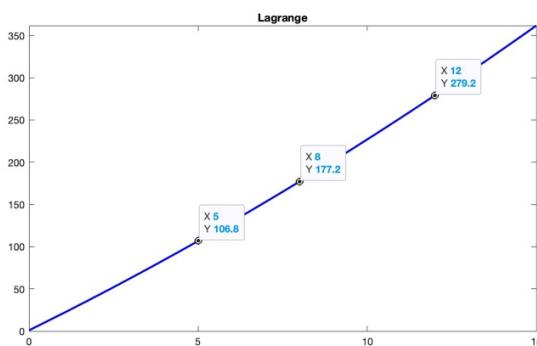
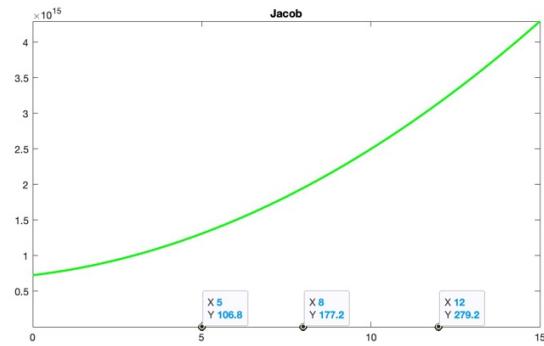
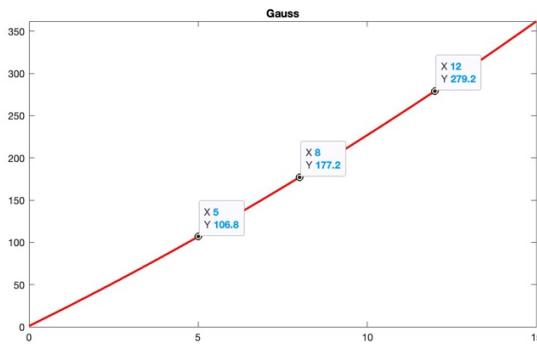
فانکشن نیوتن خطوط ۲۰۸ تا ۲۱۷: حال بر اساس فرمول چند جمله و با استفاده از دو حلقه که اولی نشان دهنده این است که در حال تولید کدام جمله‌ی تابع درون یا ب کلی هستیم و دومی برای ضرب کردن پرانتر های دارای ایکس در یکدیگر است تابع درون یا ب را تولید کرده بدین صورت که در هر مرحله یک بخش از تابع درون یا ب کلی را حساب کرده و با بخش‌های قبلی جمع می‌کنیم.

فانکشن نیوتن خطوط ۲۱۹ تا ۲۲۵: در نهایت تابع به دست آمده از بخش‌های قبل را با فانکشن simplify ساده سازی کرده با استفاده از فانکشن sym2pol مطلب ضرایب این ماتریس را استخراج کرده و نمایش می‌دهیم.

بخش ۵ (خطوط ۳۰ تا ۵۵): جواب سوال:

همانطور که در تصویر نمودارها مشاهد می‌شود سه روش حذفی گاووس، درونیابی لاگرانژ و درونیابی نیوتن به طور دقیقی یک تابع درجه ۲ را پیدا کرده‌اند و بر سه نقطه‌ی اطلاعات صورت سوال منطبق هستند و نمی‌توان تفاوتی میان دقت آن‌ها گذاشت و هیچ‌کدام دقت کمتری نسبت به دیگری ندارند. اما از آنجا که ماتریس ضرایب افزوده‌ی داده‌های سوال برای روش ژاکوبی قطری اکید نبودند روش لاگرانژ همگرا نشده است و تابع اشتباهی تولید کرده است که به هیچ عنوان بر نقاط تطابق ندارد.

نمودار های چهار روش:



توضیح کد:

خط ۳۱: یک فیگور (پنجره) ی جدید برای نمایش نمودار ها ایجاد شد

خط ۳۲ : مقدار ایکس نقاط جدول داده ی سوال در یک ماتریس ریخته شد

خط ۳۳ : مقدار ایگرگ نقاط جدول داده ی سوال در یک ماتریس ریخته شد

خط ۳۴ : مقدار ایگرگ نقاط جدول داده ی سوال در یک ماتریس ریخته شد

خطوط ۳۶ تا ۵۵ : در ابتدا با دستور subplot پنجره ی ایجاد شده به چهار قسمت تقسیم شده و بخش اول از این چهار قسمت انتخاب شده حال با استفاده از فانکشن fplot تابع ۱ که همانتابع روش حذفی گاووس است در بازه ۰ تا ۱۵ رسم شده سپس با دستور hold on و استفاده مجدد از دستور plot نقاط که پیش از این در دو ماتریس در تعریف شده بودند در همان نمودار حذفی گاووس کشیده می شوند در انتها نیز نام این نمودار مشخص می گردد به همین منوال نمودار چهار روش دیگر به همراه نقاط رسم می شوند.

بخش و (خطوط ۵۷ تا ۶۵):

توضیح کد:

خط ۵۹: تابع $f(x)$ به صورت یک تابع جبری که ورودی و خروجی دارد تعریف می شود.

خط ۶۰ : تمامی توابع درجه ۲ ای که تا به حال توسط چهار روش تولید شده اند در یک ماتریس ریخته می شوند.

خط ۶۲ تا ۶۵ : با استفاده از یک حلقه هر بار یکی از توابع تولید شده توسط چهار روش با $f(x)$ برابر قرار داده می شوند بدین معنا که حال می توانند ورودی و خروجی دلخواه بپذیرند. و سپس ورودی ۱۰ وارد هر کدام از آن ها می شود. و به کمک فانکشن vpa جواب نهایی ساده شده و به کاربر نمایش داده می شود.

speedIn10 = 227.0380952 حذفی گاوس

speedIn10 = 2.498609341e+15 ژاکوبی

speedIn10 = 227.0380952 درونیابی لاگرانژ

speedIn10 = 227.0380952 درونیابی نیوتن

سرعت در ثانیه ۱۰ تابع سرعت که توسط روش
های مختلف تولید شده است.

بخش ر (خطوط ۶۷ تا ۸۷):

توضیحات:

در این بخش جدول داده‌ی صورت سوال مجدداً تعریف می‌شود اما می‌توان بدون تعریف مجدداً این جدول داده و تغییر همان جدول داده‌ی ابتدایی در بخش مقدمه نیز این کار را انجام داد اما به جهت اینکه بتوان نتایج را هم برای حالت سه داده‌ای و هم برای حالت چهار داده‌ای به صورت همزمان دید این جدول و سایز داده‌ی آن مجدداً تعریف شدند.

توضیح کد:

خطوط ۶۹ تا ۷۵: همانند بخش مقدمه، جدول داده و سایز داده تعریف می‌شوند و ماتریس ضرایب با استفاده از فانکشن matrixMaker ساخته می‌شود

خطوط ۷۷ تا ۸۰: همانند بخش‌های الف تا د توابع جدید برای ماتریس ضرایب افزوده‌ی جدید بر اساس همان روش‌ها به همان ترتیب تولید شده و مراحل آن‌ها نمایش داده می‌شود.

خطوط ۸۲ تا ۸۷: همانند بخش و با استفاده از همان موارد و حلقه‌ها، مقدار توابع جدید در زمان ۱۰ ثانیه نمایش داده می‌شود.

```
results for guass method in order:  
-0.0071    0.4690    18.2905    4.5143
```

ضرایب تولید شده برای تابع درجه ۳ با داده‌های جدید

speedIn10 = 227.1809524 حذفی گاوس

speedIn10 = 6.396979362e+17 ژاکوبی

speedIn10 = 227.1809524 درونیابی لاگرانژ

speedIn10 = 227.1809524 درونیابی نیوتن

سرعت در ثانیه ۱۰ با توجه به تابع جدید

```

1 clc,clear, close all
2
3 %-----Information-----
4 disp("-----Information-----")
5 syms x;
6
7 data=[ 5 , 106.8 ;
8      8 , 177.2 ;
9      12, 279.2 ; ]
10
11 numOfData=size(data,1);
12 matrix= matrixMaker(data,numOfData)
13
14 %-----Part A - Gauss Method-----
15
16 finalFunction1= gaussSolver(matrix,numOfData);
17
18 %-----Part B - Jacob Method-----
19
20 finalFunction2=jacobSolver(matrix,numOfData,[0,0,0],25);
21
22 %-----Part C - Lagrange Method-----
23
24 finalFunction3 = lagrangeSolver(data,numOfData);
25
26 %-----Part D - Newton Method-----
27
28 finalFunction4=newtonSolver(data,numOfData);
29
30 %-----Part E - Ploting-----
31 f1=figure;
32
33 x_dots=reshape(data(1:numOfData,1),[1,numOfData])
34 y_dots=reshape(data(1:numOfData,2),[1,numOfData])
35
36 subplot(2,2,1)
37 fplot(finalFunction1,[0,15],'r','LineWidth',2)
38 hold on
39 plot(x_dots,y_dots,'o','color','black','LineWidth',2);
40 title('Gauss')
41 subplot(2,2,2)
42 fplot(finalFunction2,[0,15],'g','LineWidth',2)
43 hold on
44 plot(x_dots,y_dots,'o','color','black','LineWidth',2);
45 title('Jacob')
46 subplot(2,2,3)
47 fplot(finalFunction3,[0,15],'b','LineWidth',2)
48 hold on
49 plot(x_dots,y_dots,'o','color','black','LineWidth',2);
50 title('Lagrange')
51 subplot(2,2,4)
52 fplot(finalFunction4,[0,15],'m','LineWidth',2)
53 hold on
54 plot(x_dots,y_dots,'o','color','black','LineWidth',2);
55 title('Newton')
56

```

```

57 %-----Part F - Speed in 10s-----
58
59 syms f(x)
60 funcs = [finalFunction1,finalFunction2,finalFunction3,finalFunction4];
61
62 for i=1:4
63     f(x)=funcs(1,i);
64     speedIn10= vpa(f(10),10)
65 end
66
67 %-----Part G - New Data-----
68 disp("-----New Information-----")
69 data=[ 5 , 106.8 ;
70         8 , 177.2 ;
71         12, 279.2 ;
72         6 , 129.6 ;]
73
74 numOfData=size(data,1);
75 matrix= matrixMaker(data,numOfData)
76
77 finalFunction1= gaussSolver(matrix,numOfData);
78 finalFunction2= jacobSolver(matrix,numOfData,[0,0,0,0],25);
79 finalFunction3= lagrangeSolver(data,numOfData);
80 finalFunction4= newtonSolver(data,numOfData);
81
82 funcs = [finalFunction1,finalFunction2,finalFunction3,finalFunction4];
83
84 for i=1:4
85     f(x)=funcs(1,i);
86     speedIn10= vpa(f(10),10)
87 end
88
89
90 function matrix = matrixMaker(data,numOfData)
91
92     matrix= zeros(numOfData,numOfData+1);
93
94     for i=(numOfData-1):-1:0
95         matrix(1:numOfData,-i+numOfData)= (data(1:numOfData,1)).^(i);
96     end
97
98     matrix(1:numOfData,numOfData+1)= data(1:numOfData,2);
99 end
100

```

```

101 function finalFunction = gaussSolver(matrix,numOfData)
102     disp("-----Gauss Method-----")
103     syms x;
104     gauss = matrix;
105
106     for i=1:numOfData-1
107
108         [~,finalIndex]= max(gauss(i:numOfData,1:numOfData+1));
109         finalIndex=finalIndex(1,i)+i-1;
110         holder = gauss(i,1:numOfData+1);
111         gauss(i,1:numOfData+1)= gauss(finalIndex,1:numOfData+1);
112         gauss(finalIndex,1:numOfData+1)= holder;
113
114         for j=i+1:numOfData
115             gauss(j,1:numOfData+1)= gauss(j,1:numOfData+1)-
116             ((gauss(j,i)/gauss(i,i))*gauss(i,1:numOfData+1));
117         end
118     end
119
120     disp(gauss)
121
122     result= zeros(1,numOfData);
123
124     for i=numOfData:-1:1
125         minus1=0;
126         for j=1:(-i+numOfData)
127             minus1= minus1+(gauss(i,-j+numOfData+1)*result(1,-j+numOfData+1));
128         end
129         minus1= -minus1;
130         result(1,i)=(minus1+gauss(i,numOfData+1))/gauss(i,i);
131     end
132
133     finalFunction=0;
134     for i=1:numOfData
135         finalFunction= finalFunction+result(1,i)*(x^(-i+numOfData));
136     end
137
138     disp('results for guass method in order:');
139     disp(result);
140
141 function finalFunction = jacobSolver(matrix,numOfData,initialNumbers,repeats)
142     disp("-----Jacob Method-----")
143     disp(['with Initail numbers ',num2str(initialNumbers),' and ',num2str(repeats),' repeats.']);
144
145     syms x;
146     result=zeros(1,numOfData);
147     jacob= matrix;
148
149     for j=1:repeats
150         for i=1:numOfData
151             minus2=sum(jacob(i,1:numOfData).*initialNumbers)-(jacob(i,i)*initialNumbers(1,i));
152             result(1,i)=(jacob(i,numOfData+1)-minus2)/jacob(i,i);
153         end
154         initialNumbers=result;
155     end
156
157     finalFunction=0;
158     for i=1:numOfData
159         finalFunction= finalFunction+result(1,i)*(x^(-i+numOfData));
160     end
161
162     disp("results for jacob method in order:");
163     disp(result);
164 end

```

```

166 function finalFunction = lagrangeSolver(data,numOfData)
167     disp("-----Lagrange Method-----")
168     syms x;
169     index= reshape(1:numOfData,[1,numOfData]);
170     finalIndex= zeros(1,numOfData-1);
171     finalFunction=0;
172
173     for j= 1:numOfData
174         up=1;
175         down=1;
176         finalIndex(1,1:j-1)=index(1,1:j-1);
177         finalIndex(1,j:end)=index(1,j+1:end);
178
179         for i=finalIndex
180             up=up*(x-data(i,1));
181             down=down*(data(j,1)-data(i,1));
182         end
183         finalFunction=finalFunction+((up/down)*data(j,2));
184     end
185
186     disp(finalFunction);
187     finalFunction=simplify(finalFunction);
188     disp(finalFunction)
189     result= sym2poly(finalFunction);
190     disp("results for lagrange method in order:")
191     disp(result)
192 end
193
194 function finalFunction = newtonSolver(data,numOfData)
195     disp("-----Newton Method-----")
196     syms x;
197     newton=zeros(2*numOfData-1,1+numOfData);
198     newton(1:2:2*numOfData-1,2)=data(1:numOfData,2);
199     newton(1:2:2*numOfData-1,1)=data(1:numOfData,1);
200
201     for j=3:numOfData+1
202         for i= j-1:-1:-j+(2*numOfData+1)
203             newton(i,j)= (newton(i+1,j-1)-newton(i-1,j-1))/(newton(i+j-2,1)-newton(i-j+2,1));
204         end
205     end
206
207     disp(newton);
208     finalFunction=data(1,2);
209
210     for i=1:numOfData-1
211         cross=1;
212         for j=1:i
213             cross=cross*(x-data(j,1));
214         end
215
216         finalFunction= finalFunction+(cross*newton(i+1,2+i));
217     end
218
219     disp(finalFunction)
220     finalFunction=simplify(finalFunction);
221     disp(finalFunction)
222     result= sym2poly(finalFunction);
223     disp("results for newton method in order:")
224     disp(result);
225 end

```

سوال سوم

مقدمه (خطوط ۱ تا ۱۷):

توضیح کد:

خط ۱: تمامی داده های قبلی پاک و تمامی پنجره ها بسته می شوند.

خط ۲: متغیر ایکس به یک متغیر جبری تبدیل می شود.

خط ۳: دوتابع جبری newFunctionPrime و newFunction که تابع دوم قرار است مشتق تابع اول باشد تعریف می شوند.

خط ۶: یک ماتریس که قرار است حاوی تابع های جبری باشد با دستور sym مطلب تعریف می شود و ادامه قرار است تا توابع f_1 و f_2 های مختلف در آن قرار بگیرند. بنابراین باید ابعاد یک در هشت داشته باشد. نام این تابع mainFunctions است.

خط ۷: تابع f_1 به عنوان اولین تابع ماتریس mainFunctions قرار می گیرد.

خطوط ۹ تا ۱۷: با استفاده از دو حلقه که اولی برای انتخاب این است که کدام تابع از توابع f_2 بر حسب n شان ساخته شوند و حلقه داخلی برای پیاده سازی سیگما های هر کدام از این عبارت هاست. و پس از تولید هر تابع آن را درون ماتریس توابع اصلی قرار می دهیم. بنابراین هر هشت درایه ی ماتریس توابع اصلی پر می شود و در نهایت این ماتریس نمایش داده می شود.

$$\begin{aligned} \text{mainFunctions} = \\ \left(\frac{\sinh(x)}{x} \right) & \boxed{\frac{x^8}{362880} + \frac{x^6}{5040} + \frac{x^4}{120} + \frac{x^2}{6} + 1} & \boxed{\frac{x^{10}}{39916800} + \frac{x^8}{362880} + \frac{x^6}{5040} + \frac{x^4}{120} + \frac{x^2}{6} + 1} \\ f_1 & \quad f_2 \quad n=4 & f_2 \quad n=5 \end{aligned}$$

```

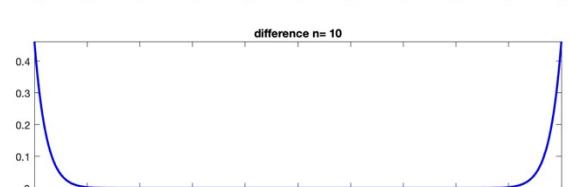
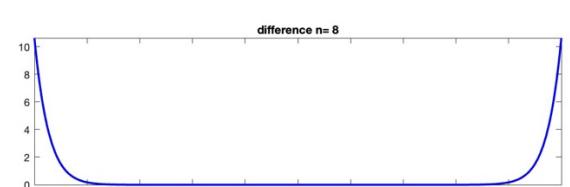
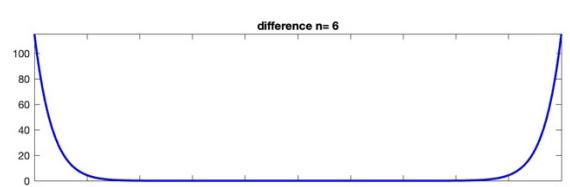
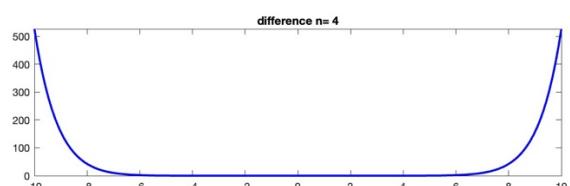
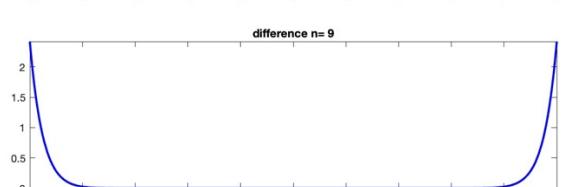
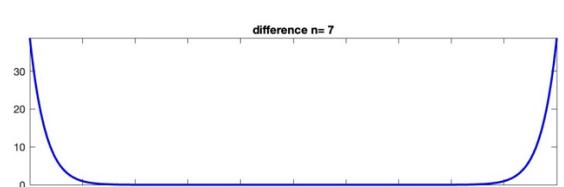
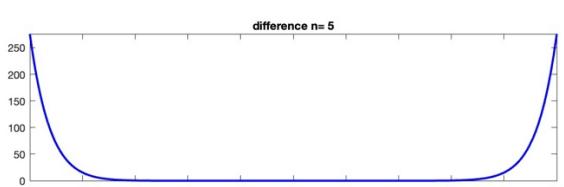
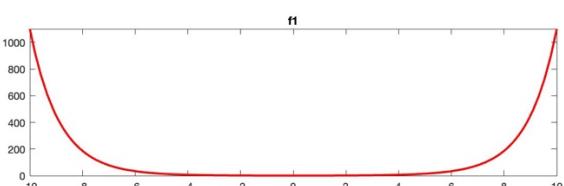
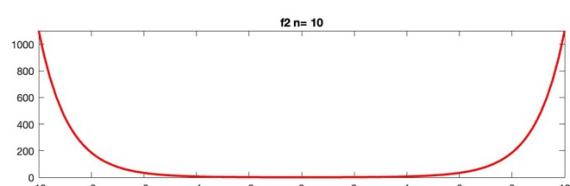
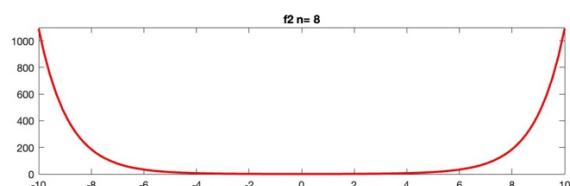
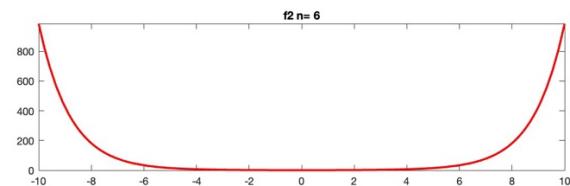
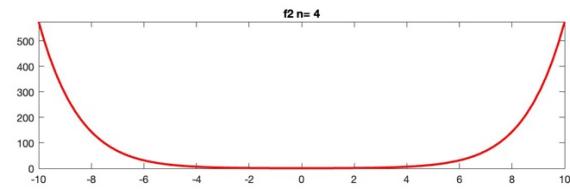
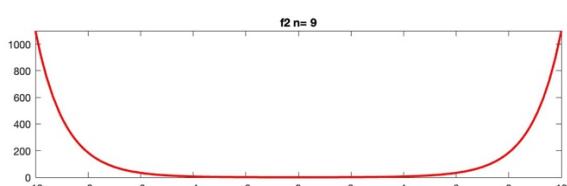
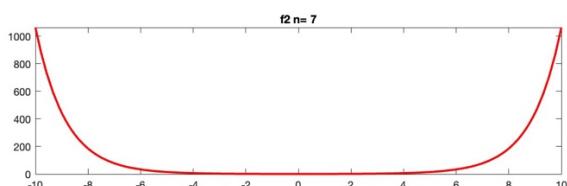
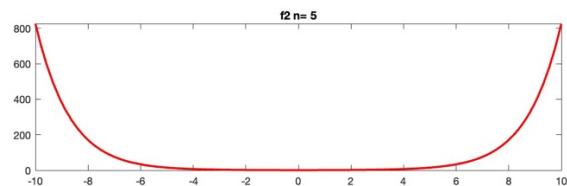
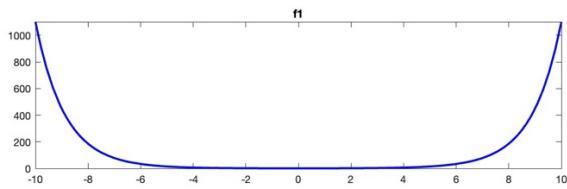
● ● ●
1 clc,clear,close all
2 syms x
3 syms newFunction(x) newFuncPrime(x)
4
5 %-----Finding Functions-----
6 mainFunctions=sym([1,8]);
7 mainFunctions(1,1)=sinh(x)/x;
8
9 for i=2:8
10    sigmaFunc=0;
11    for j=0:i+2
12       sigmaFunc= sigmaFunc+((x^(2*j))/factorial(2*j+1));
13    end
14    mainFunctions(1,i)=sigmaFunc;
15 end
16
17 mainFunctions

```

بخش الف (۱۹ تا ۴۳):

نمودار ها:

نمودار های توابع اصلی:

نمودار های تفاضل توابع از تابع f_1 :

توضیح کد:

خط ۲۰: بازه‌ای که نمودارها در آن مثبت منفی آن کشیده می‌شود برای مثال اگر مقدار ۱۰ داشته باشد نمودار در بازه‌ی ۱۰ تا ۱۵ کشیده می‌شود

خط ۲۲: یک فیگور (پنجره) ی جدید برای نمودارهای توابع باز می‌شود این پنجره لازم است تا نمودارها از نمودارهای اختلاف توابع جدا باشند.

خطوط ۲۳ تا ۲۵: با دستور subplot متلب پنجره ایجاد شده به هشت قسمت تقسیم می‌شود و در قسمت اول تابع جبری f1 کشیده می‌شود.

خطوط ۲۷ تا ۳۱: با استفاده از یک حلقه و همان دستور subplot باقی تقسیم بندی‌های پنجره‌ی ایجاد شده را با نمودار توابعی که در ماتریس (تابع اصلی) موجود است. پر می‌کنیم.

خطوط ۳۴ تا ۳۶: مجدداً یک پنجره‌ی جدید ایجاد می‌شود تا نمودارهای اختلاف توابع f2 با تابع f1 در این پنجره رسم شود. سپس پنجره بخش بندی می‌شود و تابع f1 در اولین بخش این پنجره رسم می‌شود.

خطوط ۳۸ تا ۴۳: مجدداً با یک حلقه هر کدام از توابع f2 از ماتریس (تابع اصلی) انتخاب می‌شوند و هر کدام از این توابع از تابع f1 کم می‌شوند و تابع به وجود آمده از حاصل این تفریق در یک متغیر به نام minus ریخته می‌شود و در نهایت در هر دور از حلقه minus در یک بخش از پنجره‌ی به وجود آمده کشیده می‌شود.

```

● ● ●

19 %-----Part A - Ploting -----
20 range=10;
21
22 f1=figure;
23 subplot(4,2,1);
24 fplot(mainFunctions(1,1),[-range,range],'b','LineWidth',2);
25 title('f1');
26
27 for i=2:8
28     subplot(4,2,i);
29     fplot(mainFunctions(1,i),[-range,range],'r','LineWidth',2);
30     title(['f2 n= ',num2str(i+2)]);
31 end
32
33 f2 =figure;
34 subplot(4,2,1);
35 fplot(mainFunctions(1,1),[-range,range],'r','LineWidth',2);
36 title('f1');
37
38 for i=2:8
39     subplot(4,2,i);
40     minus= mainFunctions(1,1)-mainFunctions(1,i);
41     fplot(minus,[-range,range],'b','LineWidth',2);
42     title(['difference n= ',num2str(i+2)]);
43 end

```

بخش ب (خطوط ۴۵ تا ۵۱):

توضیح کد:

خط ۴۶: یک ماتریس یک در هشت می سازیم تا حد آن هشت تابع تولید شده در آن نگهداری شود.

خطوط ۴۹ تا ۵۱: با استفاده از یک حلقه توابع درون ماتریس (تابع اصلی) را به ترتیب انتخاب کرده و به کمک فانکشن `limit` حد توابع را حساب کرده و در ماتریسی که برای نگهداری جواب ساختیم ذخیره می کنیم در نهایت این ماتریس جواب را نمایش می دهیم.

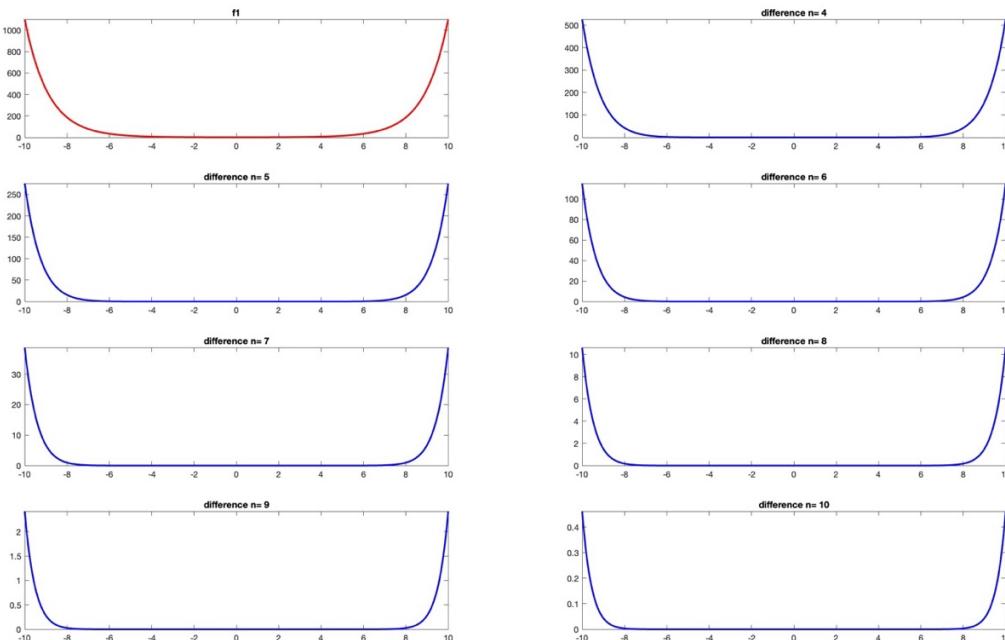
```
functionsLimit = 1x8
    1     1     1     1     1     1     1     1
```

حد توابع به ترتیب از چپ به راست در نقطه ۰

```
45 %-----Part B - Limits-----
46 functionsLimit= zeros(1,8);
47 for i= 1:8
48     functionsLimit(1,i)=limit(mainFunctions(1,i),x,0);
49 end
50
51 functionsLimit
```

بخش ج : پاسخ سوال:

با توجه به نمودار اختلاف توابع بر اساس n متوجه می شویم که هر چه n بالاتر می رود، تابع f_1 به تابع f_1 نزدیک می شوند. یعنی اختلاف این دو تابع کمتر می شود. برای مثال برای n مساوی ۵ مقدار اختلاف در نقطه ۰ برابر در مقیاس ۲۵۰ است اما برای n مساوی با ۱۰ مقدار این اختلاف در نقطه ۰ برابر ۰.۴ است.



بخش د (خطوط ۵۳ تا ۶۵):

توضیحات:

از آن جا که دقت اعشار ما ۵ رقم است (خواسته‌ی سوال ۴ رقم اعشار است) به محض این که جواب ما به این دقت برسد حلقه متوقف می‌شود بنابراین ممکن است جواب کاملاً دقیق نباشد طبیعتاً با افزایش دقت جواب دقیق تری به دست می‌آید.

تئوری:

فرمول روش نیوتون رافسون که در آن x_n جواب کنونی و x_{n+1} جواب بعدی و جواب دقیق‌تر است به صورت زیر می‌باشد و تعداد دفعات تکرار می‌تواند به تعداد دلخواه یا تا محقق شدن معیار توقف عملیات باشد:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

توضیح کد:

خط ۵۴: کدام n از سری توابع f برای برابر قرار داده شدن با f انتخاب شود در اینجا n برابر با ۴.

خط ۵۵: دقت ارقام اعشار را در متغیر error نگهداری می‌کنیم.

خط ۵۶: فانکشن جبری ای که در بخش مقدمه تعریف کرده بودیم را با با تفاضل تابع f انتخاب شده و f ، برابر قرار می‌دهیم تا بتوانیم به این تابع مقدار دهی کنیم.

خط ۵۷: تابع جبری دومی که در بخش مقدمه تعریف کردیم را نیز برابر به کمک فانکشن مشتق گیری متلب برابر با مشتق تابع تعریف شده بخش قبل، قرار می‌دهیم.

خط ۵۹: یک متغیر به نام نتیجه تعریف می‌شود که مقدار آن در ابتدا همان حدس اولیه برای جواب است. در نهایت نیز جواب نهایی در این متغیر ذخیره می‌شود.

خطوط ۶۱ تا ۶۵: با استفاده از حلقه `while` تا هنگامی که مقدار تابع به ازای نتیجه یافته شده کوچک‌تر از مقدار خطا نباشد دنباله یافتن جواب به روش نیوتون رافسون را ادامه می‌دهیم. توابع `vpa` برای ساده سازی کسرها استفاده می‌شود، و تعداد اعشار نیز بر اساس متغیر خطا تعیین می‌شود و در نهایت وقتی از حلقه خارج می‌شویم نتیجه نهایی را نشان می‌دهیم.

```

53 %-----Part D - Newton Method-----
54 n=4;
55 error = 10^(-5);
56 newFunction(x)= mainFunctions(1,1)-mainFunctions(1,n-2)
57 newFuncPrime(x)= diff(newFunction,x)
58
59 result=10;
60
61 while abs(vpa(newFunction(result))) > error
62     result= result-vpa(newFunction(result)/newFuncPrime(result),abs(log(error)/log(10))+1);
63 end
64
65 result = vpa(result,abs(log(error)/log(10))+1)

```

سوال چهارم

توضیحات: از آن جا که در این تابع قصد داریم با روش های تکراری ریشه‌ی ۹۱ تابع را با دقت ۱۶ رقم اعشار بیابیم طبیعی است که اجرا شدن برنامه زمان زیادی بگیرد لذا مدت زمان انتظار زیاد برای دریافت نتیجه نشانه از خرابی کد نیست.

مقدمه (خطوط ۱ تا ۲۱):

توضیح کد:

خط ۱: پاک کردن تمام داده‌های قبلی و بستن پنجره‌های باز

خطوط ۳ تا ۴: تعریف متغیر جبری d و تعریف تابع جبری `calculationFunc` برای مقدار دهی‌های بعدی

خط ۶: از آنجا که سوال دقت اعشار را ۱۶ رقم تعیین کرده با توجه جواب نهایی که تا مرتبه دهگان است پس دقت جواب را با استفاده از دستور `digits` تا ۱۸ رقم قرار داده ایم.

خط ۱۰: تمام سیگما اس‌ها در ماتریسی به همین نام با طول گام داده شده در سوال ذخیره می‌شوند.

خطوط ۱۱ تا ۱۳: سیگما اف، جی و سرعت که داده‌های سوال هستند نیز تعریف می‌شوند.

خط ۱۵: برای نگهداری تابع‌های جبری به ازای سیگما اس‌های مختلف با استفاده از دستور `sym` ماتریس جبری ای تعریف می‌کنیم و سایز آن را برابر با تعداد سیگما اس‌ها قرار می‌دهیم.

خطوط ۱۷ تا ۲۱: با استفاده از یک حلقه سیگما اس‌ها را به ترتیب انتخاب کرده و با استفاده از داده‌های سوال تابع‌های جبری مورد نیاز برای یافتن ریشه‌های را می‌سازیم همچنین با دستور `vpa` این توابع را ساده می‌کنیم، لازم به ذکر است که متغیر این توابع d می‌باشد و در نهایت تمامی این توابع را نشان می‌دهیم.

$$\text{mainFunctions} = \left(\sqrt{68.67d - 21.69} \right) \sqrt{69.651d - 21.69} \sqrt{70.632d - 21.69} \sqrt{71.613d - 21.69}$$

تعدادی از توابع تولید شده

```

1 clc,clear,close all
2
3 syms d
4 syms calculationFunc(d)
5
6 digits(18);
7
8 %-----Information-----
9
10 sigmaS= 5:0.05:9.5;
11 sigmaF=1.5;
12 g=9.81;
13 v=21.69;
14
15 mainFunctions=sym([1,size(sigmaS,2)]);
16
17 for i=1:size(sigmaS,2)
18     mainFunctions(1,i)= vpa(((3*g*d*(sigmaS(1,i)-sigmaF))/sigmaF).^(0.5))-v,14);
19 end
20
21 mainFunctions

```

بخش الف (خطوط ۲۳ تا ۵۷):

توضیحات:

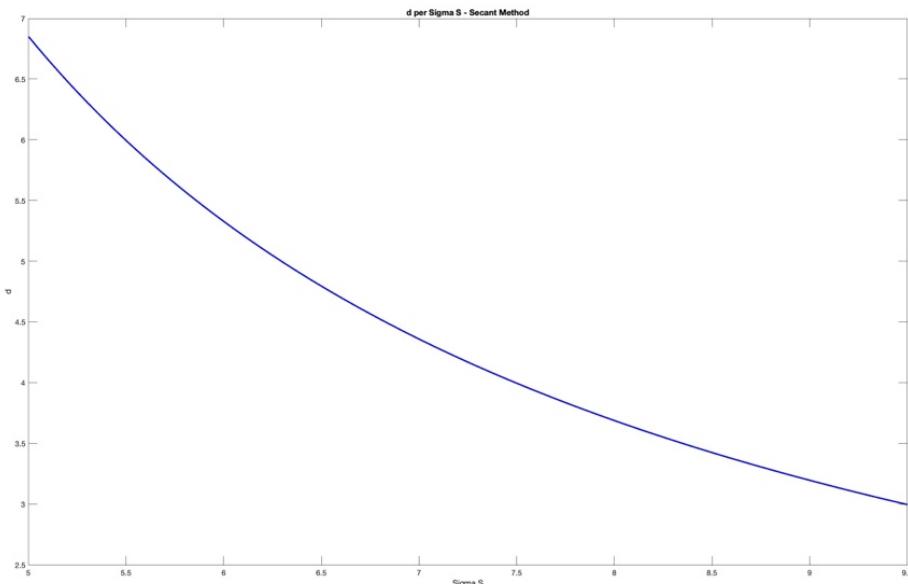
برای تغییر تعداد دفعات تکرار روش باید متغیر repeats موجود در خط ۲۹ را تعویض کرد و برای تغییر x_0 و x_1 باید به ترتیب متغیرهای preResult و middleResult در خطوط ۳۶ و ۳۷ را تغییر داد.

تئوری:

فرمول روش وتری برای یافتن ریشه به صورت زیر می باشد که این دنباله را تا تعداد دلخواه یا تا محقق شدن معیار توقف عملیات ادامه می دهیم:

$$x_{n+1} = \frac{(x_{n-1})f(x_n) - (x_n)f(x_{n-1})}{f(x_n) - f(x_{n-1})}$$

نمودار:



توضیح کد:

خطوط ۲۵ تا ۲۹: با توجه به ماهیت روش وتری که نیاز به مقدار تابع در مرحله قبلی و مرحله قبلش دارد و از آنجا که ۹۱ تابع مختلف داریم باید ۳ ماتریس جدگانه برای هرکدام از این مرحله ها که به ترتیب جمله های ایکس- n (او ایکس- n) و ایکس- $(n+1)$ هستند یک ماتریس تعریف می کنیم با نام های preResult ، middleResult و result و سپس تعداد تکرار روش وتری را معین می کنیم. نام ماتریس جواب نهایی result است.

خط ۳۱: از آنجا که مراحل پردازش ۹۱ تابع زمان می برد پیغامی تحت عنوان (صبر کنید...) به کاربر نمایش داده می شود.

خطوط ۳۳ تا ۵۱: از دو حلقه که اولی برای انجام روش وتری بر روی هر یک از ۹۱ تابع و دومی برای پیاده سازی خود روش وتری به تعداد دفعات تکرار است استفاده می کنیم در حلقه اول یکی از توابع را انتخاب کرده و حدس های اولیه را مشخص می کنیم سپس وارد حلقه ای دوم شده و با استفاده از تابع جبری ای که در بخش مقدمه تعریف کردیم و مقدار دهی آن صورت و مخرج کسر روش وتر را محاسبه کرده و از آنجا که سرعت همگرایی جواب بالا است و ممکن است با دقت اعشاری که ما در نظر گرفته ایم منجر به صفر شدن مخرج قبل از اتمام تعداد دفعات تکرار شود با استفاده از یک شرط این وضعیت را چک کرده و در صورت رخ دادن از حلقه خارج می شویم. سپس مقدار

جواب را حساب می کنیم و در نهایت هر مرحله از جواب را که همان $n-1$ و n هستند را یکی با یکی جلو تر از خود تعویض می کنیم. و این مراحل را به تعداد تکرار داده شده انجام می دهیم و در نهایت ماتریس جواب نهایی که همان $n+1$ است را نمایش می دهیم.

```
result1 =
(6.85096985583224115) 6.75447732265150536 6.6606651376146789
```

جواب های تولید شده توسط روش وتری به ترتیب از چپ به راست

خطوط ۵۳ تا ۵۶: با استفاده از دستور `plot` متلب نمودار خواسته شده در صورت سوال را رسم می کنیم.

```

23 %----- Part A - Secant Method -----
24
25 preResult=sym([1,size(sigmaS,2)]);
26 MiddleResult=sym([1,size(sigmaS,2)]);
27 result1=sym([1,size(sigmaS,2)]);
28
29 repeats= 30;
30
31 disp("Wait a while....")
32
33 for j=1:size(sigmaS,2)
34
35     calculationFunc(d)=mainFunctions(1,j);
36     preResult= 0;
37     MiddleResult= 15;
38
39     for i=1:repeats
40         up=((preResult*calculationFunc(MiddleResult))-(MiddleResult*calculationFunc(preResult)));
41         down=(calculationFunc(MiddleResult)-calculationFunc(preResult));
42         if down == 0
43             break
44         end
45         result1(1,j)= up/down;
46         preResult=MiddleResult;
47         MiddleResult=result1(1,j);
48     end
49 end
50
51 result1
52
53 plot(sigmaS,result1,'b','LineWidth',2);
54 title('d per Sigma S - Secant Method');
55 xlabel('Sigma S');
56 ylabel('d');

```

بخش ب (خطوط ۵۸ تا ۸۶): توضیحات:

برای تغییر تعداد دفعات تکرار روش متغیر `repeats` در خط ۶۱ و برای تغییر ابتدای بازه ی حدس متغیر `firstGuess` و برای تغییر انتهای بازه ی حدس متغیر `secondGuess` را تعویض می کنیم.

تئوری:

روش دوبخشی مراحل زیر است لازم به ذکر است که در ابتدای کار به ترتیب ابتدای بازه a و b را در ابتدای کار به ترتیب ابتدای بازه x_i حدد و انتهای بازه x_i جواب هستند و این مراحل را تا تعداد دلخواه یا تا محقق شدن معیار توقف عملیات ادامه می‌دهیم:

$$x_i = \frac{a+b}{2}$$

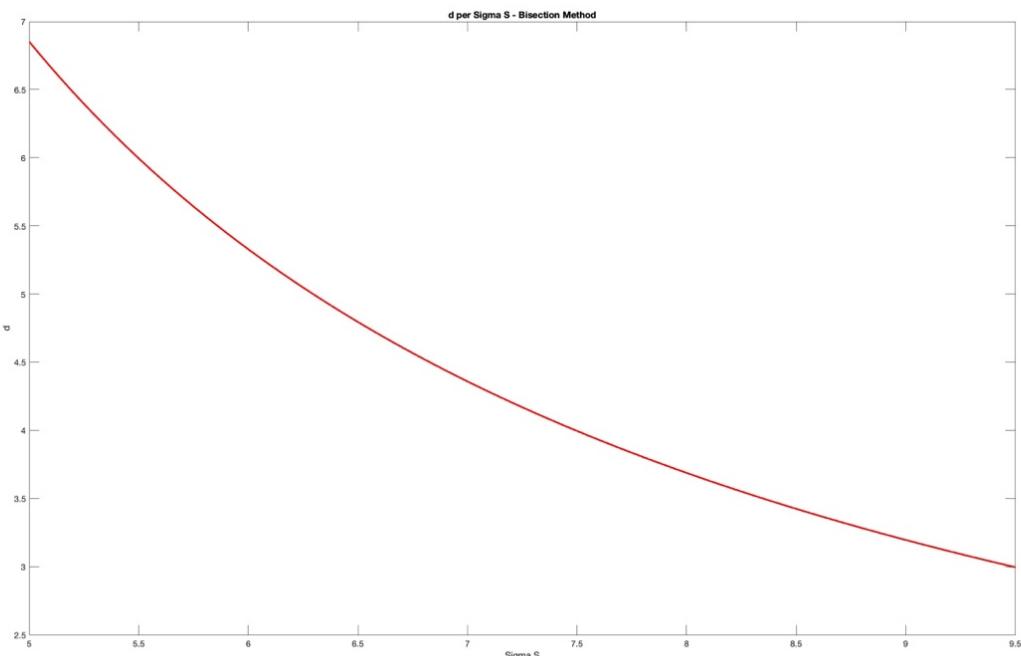
then

$$\text{if } f(x_i) = 0 \rightarrow x_i = \text{answer}$$

$$\text{if } f(a)f(x_i) < 0 \rightarrow b = x_i$$

$$\text{if } f(b)f(x_i) < 0 \rightarrow a = x_i$$

$$x_{i+1} = \frac{a+b}{2}$$

نمودار:**توضیح کد:**

خط ۶۰: یک ماتریس در ابعاد یک در (تعداد سیگما اس ها) به نام `result` برای نگهداری نتیجه به دست آمده توسط روش دو بخشی ساخته می شود.

خط ۶۱: تعداد تکرار روش دو بخشی معین می شود.

خط ۶۲: از آنجایی که مراحل انجام این پردازش زمان بر است پیغام (صبر کنید...) به کاربر نمایش داده می شود

خطوط ۶۴ تا ۸۰: از دو حلقه استفاده می کنیم که اولی برای انتخاب هر کدام از آن ۹۱ تابع است و دومی برای تعداد تکرار های روش دو بخشی در حلقه اول یکی از ۹۱ تابع انتخاب می شوند و مقدار ابتدا و انتهای بازه x_i جواب داده می شود سپس وارد حلقه دوم شده و میان بازه را به عنوان جواب در نظر می گیریم سپس با بررسی شرط روش

دوبخشی مجدداً انتها و ابتدای بازه را تنظیم می‌کنیم و این مراحل را به تعداد شمارندهٔ حلقه دوم تکرار می‌کنیم. و در نهایت ماتریس جواب را نمایش می‌دهیم.

```
result2 =
(6.85096985589552787) 6.75447732273823931 6.66066513758778456
```

جواب‌های تولید شده توسط روش دوبخشی به ترتیب از چپ به راست

خطوط ۸۲ تا ۸۵: با استفاده از دستور `plot` مطلب نمودار خواسته شده در صورت سوال را رسم می‌کنیم.

```

58 %----- Part B - Bisection Method -----
59
60 result2=sym([1,size(sigmaS,2)]);
61 repeats=40;
62 disp("Wait a while....");
63
64 for j=1:size(sigmaS,2)
65
66     calculationFunc(d)=mainFunctions(1,j);
67     firstGuess=0;
68     secondGuess=100;
69
70     for i=1:repeats
71         result2(1,j) = (firstGuess+secondGuess)/2;
72         if 0 > calculationFunc(firstGuess)*calculationFunc(result2(1,j))
73             secondGuess=result2(1,j);
74         else
75             firstGuess=result2(1,j);
76         end
77     end
78 end
79
80 result2= vpa(result2)
81
82 plot(sigmaS,result2,'r','LineWidth',2);
83 title('d per Sigma S - Bisection Method');
84 xlabel('Sigma S');
85 ylabel('d');
```

بخش ج (خطوط ۸۷ تا ۱۰۸):

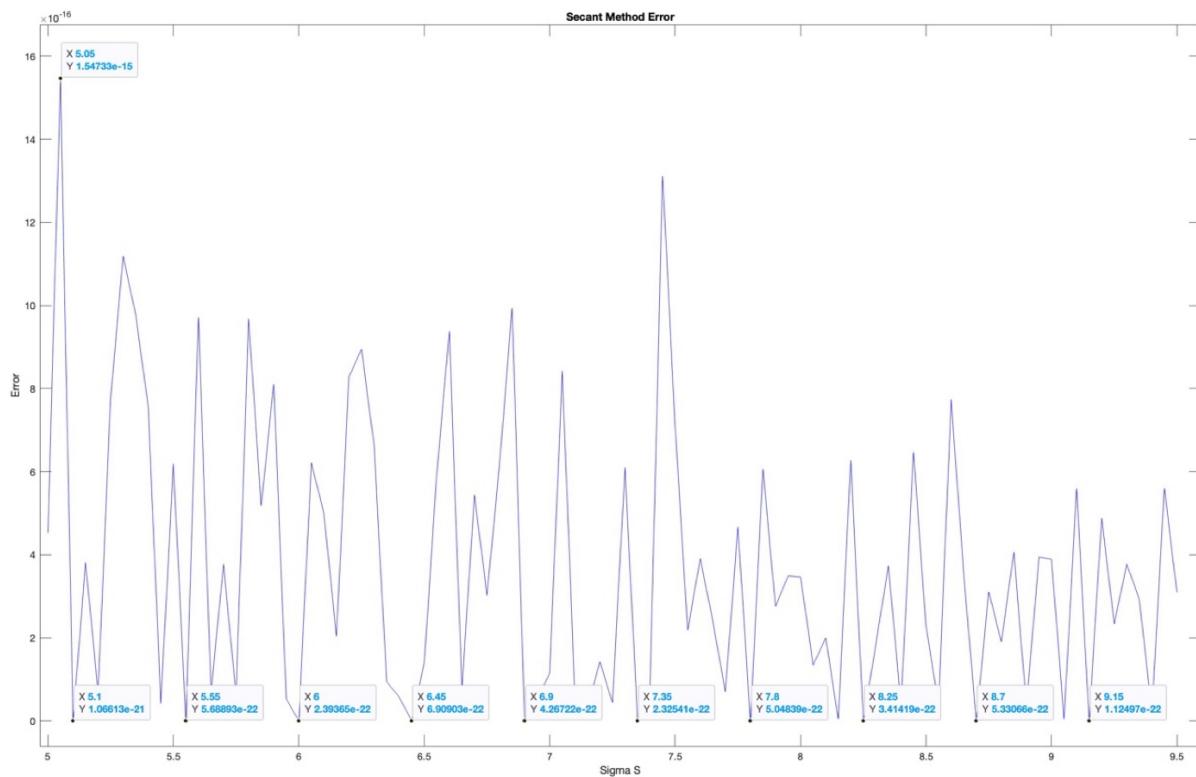
پاسخ سوال:

روش وتری: با توجه به نمودار خطای این روش می‌توان دریافت که حداقل خطا در تابع به ازای سیگمای اس ۵.۰۵ رخداده است و همچنین کمترین میزان خطا در تابع به ازای سیگمای اس ۹.۱۵ رخداده است. مقادیر خطا در این نقاط بر روی نمودار روش وتری قابل مشاهده هستند.

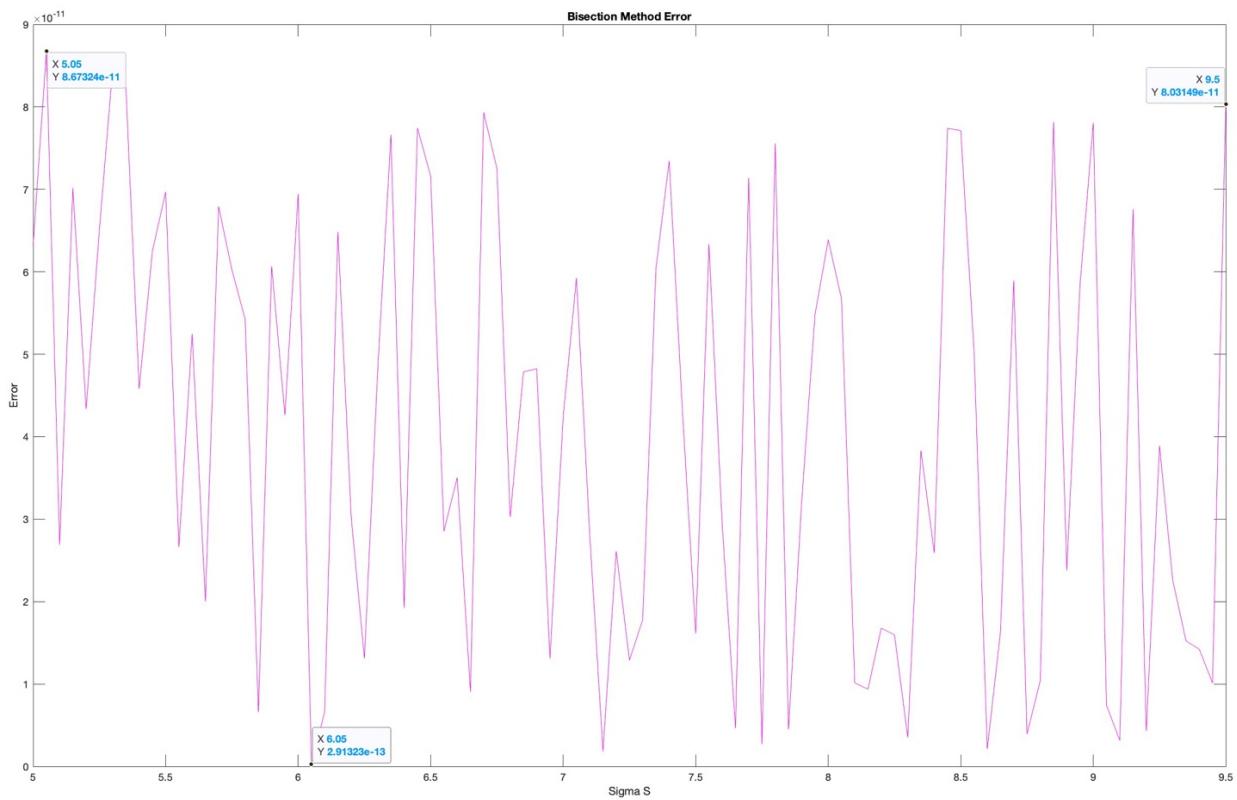
روش دو بخشی: با توجه به نمودار خطای این روش می‌توان دریافت که حداقل خطا در تابع به ازای سیگمای اس ۵.۰۵ رخداده است و همچنین کمترین میزان خطا در تابع به ازای سیگمای اس ۶.۰۵ رخداده است. مقادیر خطا در این نقاط بر روی نمودار روش دوبخشی قابل مشاهده هستند.

نمودار ها:

خطای روش وتری:



خطای روش دو بخشی:



توضیح کد:

خط ۸۹: ماتریس دیگری با ابعاد یک در (تعداد سیگما اس ها) و به نام `result3` می سازیم تا نتایج این قسمت را در آن ذخیره کنیم

خط ۹۱ تا ۹۳: با استفاده از یک حلقه هر یک از سیگما اس ها را انتخاب کرده و درونتابع بازنویسی شده که در صورت سوال گفته شده است قرار می دهیم و نتیجه را محاسبه و در ماتریس `result3` ذخیره می کنیم.

خط ۹۵: اعداد درون جواب را که به صورت کسری است ساده سازی می کنیم.

خط ۹۷: با استفاده از فانکشن قدر مطلق متلب قدر مطلق اختلاف ماتریس جواب سوم با ماتریس جواب اول را محاسبه می کنیم و در ماتریس `error 1` ذخیره می کنیم.

خط ۹۸: با استفاده از فانکشن قدر مطلق متلب قدر مطلق اختلاف ماتریس جواب سوم با ماتریس جواب دوم را محاسبه می کنیم و در ماتریس `error 2` ذخیره می کنیم.

```
result3 =
(6.85096985583224161   6.75447732265150691   6.6606651376146789
```

جواب های تولید شده از تابع بازنویسی شده به ترتیب از چپ به راست

خطوط ۱۰۰ تا ۱۰۹: با استفاده از تابع `plot` متلب ، نمودار های خواسته شده در صورت سوال را می کشیم.



```

87 %-----Part C - Rearranging and Errors -----
88
89 result3 = sym([1,size(sigmaS,2)]);
90
91 for i=1:size(sigmaS,2)
92     result3(1,i)= (sigmaF*(v^2))/(3*g*(sigmaS(1,i)-sigmaF));
93 end
94
95 result3 = vpa(result3)
96
97 error1= abs(result3-result1);
98 error2= abs(result3-result2);
99
100 plot(sigmaS,error1,'b');
101 title('Secant Method Error');
102 xlabel('Sigma S');
103 ylabel('Error');
104
105 plot(sigmaS,error2,'m');
106 title('Bisection Method Error');
107 xlabel('Sigma S');
108 ylabel('Error');

```

تمرین اول کامپیوتري

مجید صادقی نژاد

بهار ۱۴۰۲

