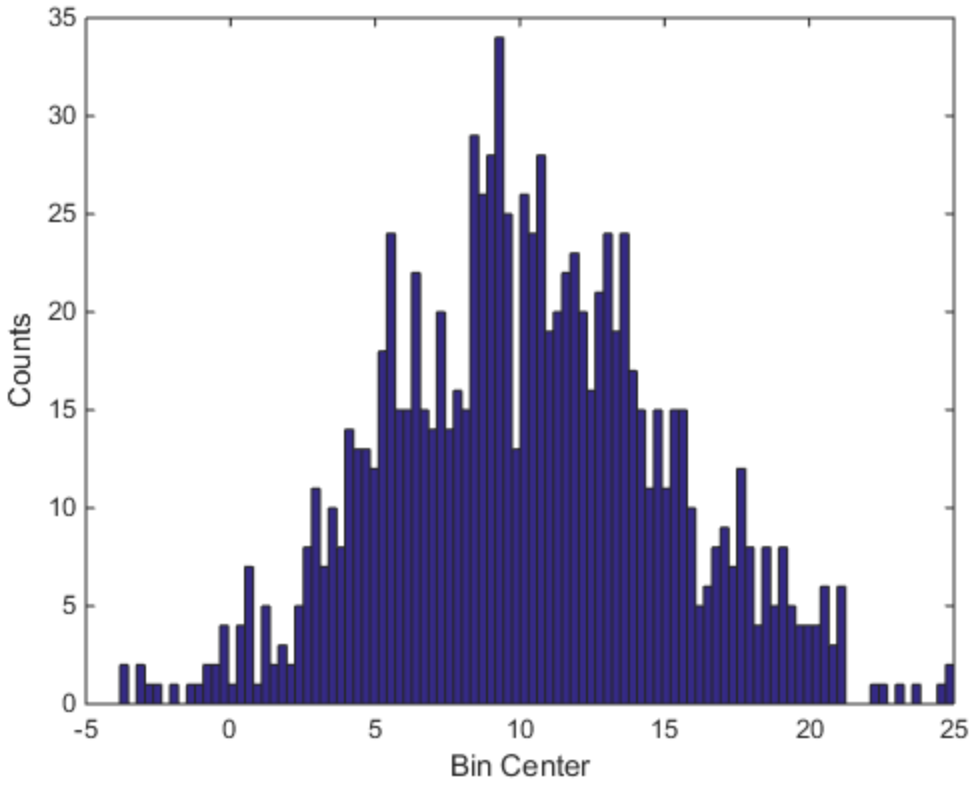
Table of Contents

Problem 1

```
%-----------------------------------------------------------------
% This is the code block for problem 1.  See the results below.
clc
fprintf('Problem 1 Results\n\n')

m=10;       % Mean of the distribution
s=5;        % Standard deviation

%Generate random numbers from a normal distribution with mean of zero,
%standard deviation of 1. Then correct to what I want.
a=5*randn(1000,1)+10;

% Check the properties of my vector.
fprintf('The mean of the vector is: %g\n',mean(a))
fprintf('The standard deviation of the vector is: %g\n',std(a))

fprintf('\nThey are close to the distribution mean and standard deviation\n')
fprintf('but are not exact because it is a finite sized vector\n')

fprintf('\nHere is the histogram showing the distribution of our vector\n')

% I use 100 to have more accurate histogram
figure
hist(a,100);  % You can also use 'histogram'
xlabel('Bin Center')
ylabel('Counts')
%-----------------------------------------------------------------
```

```
Problem 1 Results

The mean of the vector is: 10.2648
The standard deviation of the vector is: 4.89943

They are close to the distribution mean and standard deviation
but are not exact because it is a finite sized vector

Here is the histogram showing the distribution of our vector
```



Problem 2

```
%-----------------------------------------------------------------
% This is the code block for problem 2.  See the results below.
clc
fprintf('Problem 2 Results\n\n')

%Create function handle. 'a' is the vector created in problem 1.
f=@(x) sum((x-a).^2);

%Plot the function to take a look at where minimum might be


%Function doesn't work if x is a vector, so calculate one at a time

%Initialize variable (not necessary but good practice)
y=zeros(size(Xvec));

%Calculate function values over a range.
Xvec=linspace(-20,20,100);
for ii=1:length(Xvec)
    y(ii)=f(Xvec(ii));
end

%Plot
```

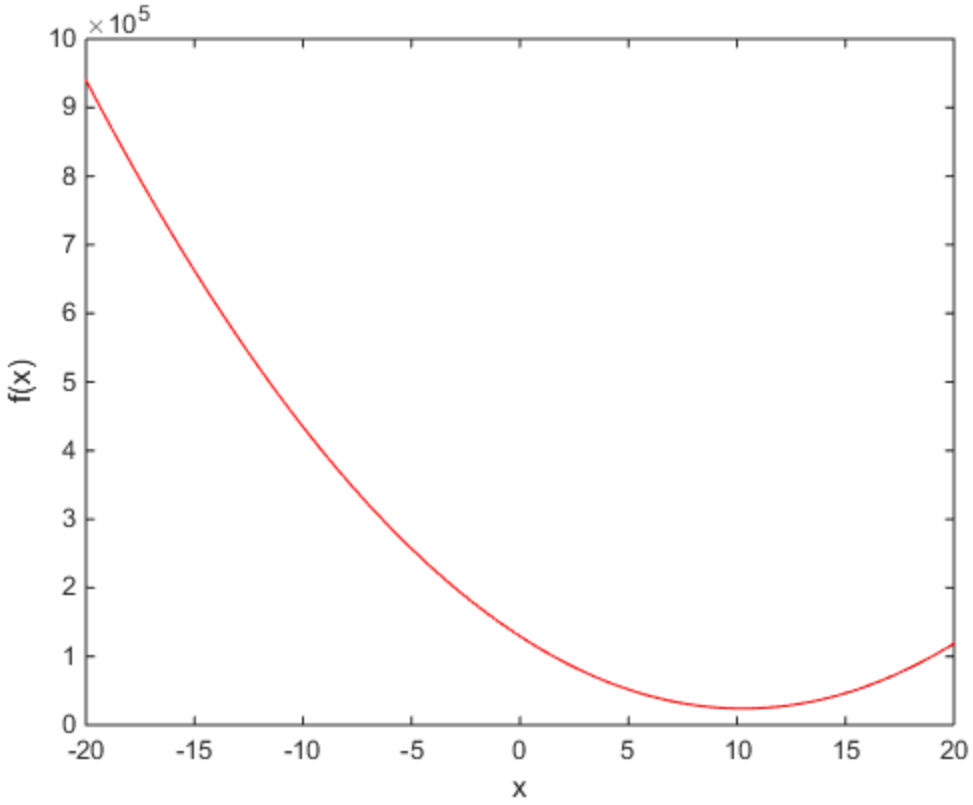
```
figure
plot (Xvec, y,'r')
xlabel('x')     %Always label!
ylabel('f(x)')

fprintf('Looks like minimum is between 0 and 20 so I use that as bounds.\n')

%Create the starting values. A---X0--B
A=0;
B=20;
r=(1+sqrt(5))/2;
X0=(B-A)*r+A;

%Starting X interval positions and function values at those points
X=[A X0 B];
FX=[f(X(1)) f(X(2)) f(X(3))];

%Create object and set properties
GR=HW7_Solution_GoldenSection();
GR.F=f;
GR.X=[A X0 B];
GR.TolX=1e-16;

%Run the method that finds the minimum
GR.findMin()

%Note: We could watch the search finding using 'GR.plotUpdateGS()'
%
%Display this property (Set by finMin() )

fprintf('\nThe found minimum is %g.  I show this on the plot below\n',GR.Found)

%Plot
figure
plot (Xvec, y,'r')
hold on
plot ([GR.Found,GR.Found], [min(y) max(y)],'k--')
xlabel('x')     %Always label!
ylabel('f(x)')
legend('Function','Found Minimum')

fprintf('\nI notice the found minimum: %g is the same as the mean of a:%g\n',GR.Fo

%Is this a coincidence?

%-----------------------------------------------------------------
```

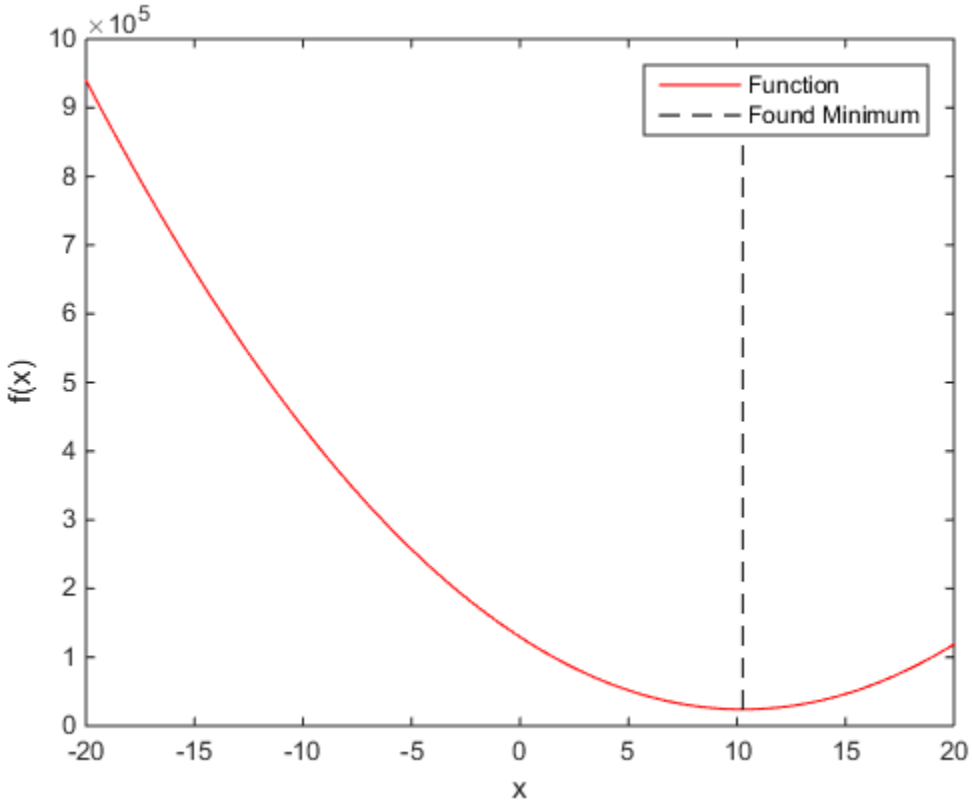
Problem 2 Results

Looks like minimum is between 0 and 20 so I use that as bounds.

The found minimum is 10.2648. I show this on the plot below

I notice the found minimum: 10.2648 is the same as the mean of a:10.2648


×10^5
10
9
8
7
6
5
4
3
2
1
0
f(x)
-20
-15
-10
-5
0
5
10
15
20
x


×10^5
Function
Found Minimum
10
9
8
7
6
5
4
3
2
1
0
f(x)
-20
-15
-10
-5
0
5
10
15
20
x

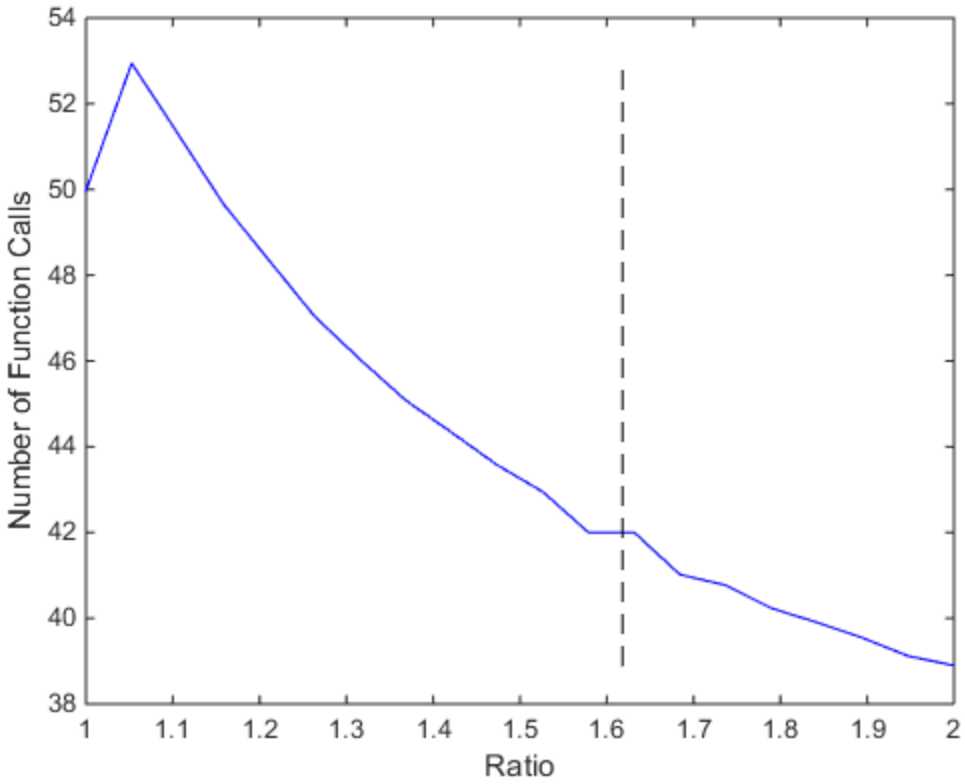
Problem 3

```
%-----------------------------------------------------------------
% This is the code block for problem 2.  See the results below.
clc
fprintf('Problem 3 Results\n\n')

%Number of averages over the same ratio.  Function minimums will vary.
Naverages=1000;

%Number of different ratios to try between 0 and 1
Nratios=20;

%Create array of ratios to test.
Ratios=linspace(1,2,Nratios);

%Intialize matrix for output
FunctionCalls=zeros(Nratios,Naverages);

%Loop over Ratios
for jj=1:Nratios

    %Uncomment this line to show progress
    %fprintf('Calculating for Ratio: %g\n',Ratios(jj))

    %Setup for this ratio
    R=Ratios(jj);
    A=0;
    B=1;
    r=R/(1+R);
    X0=(B-A)*r+A;

    %Create object and set properties
    GR=HW7_Solution_GoldenSection();
    GR.R=R;
    GR.X=[A X0 B];
    GR.TolX=1e-8;

    %'Nested' loop over functions with different (random) minimums
    for ii=1:Naverages

        %Create new function with random minimum
        a=rand();
        f = @(x) (x-a).^2;

        %Change object property to new function handle
        GR.F=f;

        %Reset Bounds
        GR.X=[A X0 B];

        %Run method to find minimum
        GR.findMin()
```

```
        %Store our result in the matrix
        FunctionCalls(jj,ii)=GR.NFunctionCalls;
    end
end

%Plot the results

%Take average along the second dimension
AverageFC=mean(FunctionCalls,2);

%This is our main result:
fprintf('\nThe plot below is our main result.\n')

figure
plot(Ratios,AverageFC,'b')
hold on
%Plot the position of the Golden Ratio
plot([1.618 1.618],[min(AverageFC) max(AverageFC)],'k--')
xlabel('Ratio')
ylabel('Number of Function Calls')


fprintf('Interesting- the golden ratio is not optimal!\n')

%-----------------------------------------------------------------
```

Problem 3 Results

The plot below is our main result.
Interesting- the golden ratio is not optimal!



Problem 4

```
% Notice the many 'fprintf' staments used to print text next to figures.
%
% I changed the output file format to 'PDF' and hit publish
%
% I'll also print out the class definition for the class I am using:

type HW7_Solution_GoldenSection


classdef HW7_Solution_GoldenSection < handle
    %HW7_GoldenRatio Find minimum of fucntion using Golden Section search
    %   Detailed explanation goes here

    properties
        F;                  %Function Handlde
        X;                  %Current intervals (3 x 1);
        FX;                 %Function Evalulated at X
        TolX=.001;          %Tolerance for stopping
        R=(1+sqrt(5))/2;    %Default Ratio (Golden)
        Found;              %Found minimum
        NFunctionCalls;     %Number of function calls required to find minimum
    end

    methods
```

```
function findMin(obj)

    %Interval size used to check convergence
    dX=obj.X(3)-obj.X(1);

    %Calculate the function at the initial X values
    obj.FX(1)=obj.F(obj.X(1));
    obj.FX(2)=obj.F(obj.X(2));
    obj.FX(3)=obj.F(obj.X(3));

    %Number of function calls (Three from above)
    N=3;

    %Loop until interval size is below tolerance.
    while dX>obj.TolX

        %Get new interval
        [ X_out,FX_out ] = obj.updateGRInterval(obj.X,obj.FX);

        %Calculate new interval size
        dX=X_out(3)-X_out(1);

        %update X and Function values
        obj.X=X_out;
        obj.FX=FX_out;

        %update number of function calls.
        N=N+1;
    end

    %Set these object properties
    obj.Found=(obj.X(3)+obj.X(1))/2;
    obj.NFunctionCalls=N;

end

function [ X_out,FX_out ] = updateGRInterval(obj,X,FX)

    f=obj.F;

    %Ratio of lengths of interval (a/b)
    % |-----a-----|--b--|
    R=obj.R;

    % a/(a+b) is:
    r=R/(1+R);

    %Find if the large interval is on right or left
    if (X(2)-X(1)) > (X(3)-X(2))
        %X1-----X2---X3
        IntSide='Left';
    else
        %X1---X2-----X3
```

```
    IntSide='Right';
end

switch IntSide
    case 'Left'
        %X1---Xnew--X2---X3

        %Find new X position
        IntSize=X(2)-X(1); %large interval size
        Xnew=X(1)+IntSize*r;
        %Evaulate function at Xnew
        F_Xnew=f(Xnew);

        %Return new interval X and FX depending on F_Xnew
        if F_Xnew<FX(2)  %use left interval
            X_out(1)=X(1);
            X_out(2)=Xnew;
            X_out(3)=X(2);
            FX_out(1)=FX(1);
            FX_out(2)=F_Xnew;
            FX_out(3)=FX(2);
        else    %use right interval
            X_out(1)=Xnew;
            X_out(2)=X(2);
            X_out(3)=X(3);
            FX_out(1)=F_Xnew;
            FX_out(2)=FX(2);
            FX_out(3)=FX(3);
        end

    case 'Right'

        %X1---X2--Xnew---X3

        %Find new X position
        IntSize=X(3)-X(2); %large interval size
        Xnew=X(2)+IntSize*(1-r);
        %Evaulate function at Xnew
        F_Xnew=f(Xnew);

        %Return new interval X and FX depending on F_Xnew
        if F_Xnew<FX(2)  %use right interval
            X_out(1)=X(2);
            X_out(2)=Xnew;
            X_out(3)=X(3);
            FX_out(1)=FX(2);
            FX_out(2)=F_Xnew;
            FX_out(3)=FX(3);
        else    %use left interval
            X_out(1)=X(1);
            X_out(2)=X(2);
            X_out(3)=Xnew;
            FX_out(1)=FX(1);
            FX_out(2)=FX(2);
```

```
                FX_out(3)=F_Xnew;
            end
        end
    end


    function plotUpdateGS(obj,PauseTime)


        %This is similar to findMin() but shows updates.

        if nargin<2
            PauseTime=1;
        end

        dX=obj.X(3)-obj.X(1);
        obj.FX(1)=obj.F(obj.X(1));
        obj.FX(2)=obj.F(obj.X(2));
        obj.FX(3)=obj.F(obj.X(3));

        GrayOut=0;

        %Plot function over interval
        Xvec=linspace(obj.X(1),obj.X(3),100);
        Yvec=obj.F(Xvec);

        figure
        plot(Xvec,Yvec,'r')
        hold on
        xlabel('x')
        ylabel('f(x)')

        while dX>obj.TolX

            if GrayOut %so it doesn't try to do this the first time
                for jj=1:3
                    PobjVec(jj).Color=[.75,.75,.75];
                end
            else
                GrayOut=1;
            end

            %Plot the new intervals
            for jj=1:3
                PobjVec(jj)=plot([obj.X(jj) obj.X(jj)],[min(Yvec) max(Yvec)],'
            end

            %Update the intervals
            [ X_out,FX_out ] = obj.updateGRInterval(obj.X,obj.FX);
            dX=X_out(3)-X_out(1);
            obj.X=X_out;
            obj.FX=FX_out;

            %pause so we can see update
```

```
                pause(PauseTime)
            end
        end


    end

end
```

Published with MATLAB® R2014b