Assignment 5

Jennifer Tossell Mathilde Leuridan

March 2019

1 5A Root Finding

In this question, we will find the first positive root of the function $f(x) = 3xcos(x) - sin(x)$ using two different methods. When plotting the function, we find a rough value of it as 1.3.

a)

Firstly, we use the bisection method to find the exact root to an accuracy of 10^{-12}.

In order to do this, we write the m-file:

```
% This implements the bisection algorithm to find a root

format long;
f = @(x) 3*x*cos(x)-sin(x);% This is our given function

% This is our starting points because we saw on the plot
    that the root was
% between 1 and 2
a=1; % with f(a)>0
```

```
b=2; % with f(b)<0root=1;
root=1; % this will give the closest value to the root each iteration
counter=0; % this counts the number of iterations
while abs(a-b)>= 10^(-12) % While we are above the desired accuracy, we continue the algorithm
    x=(a+b)/2;
    if f(x)>0
        a=x;
        root=x
    end
    if f(x)<0
        b=x;
        root=x
    end
    if f(x)==0
        root=x;
        break;
    end
    counter=counter+1; % we increment the counter once we 've done one more iteration
end
approxroot=root % this gives the final value of the root found
counter % this gives the final number of iterations
```

which gives us the iterations:

root =

1.500000000000000

root =

1.250000000000000

root =

1.375000000000000

root =

1.312500000000000

root =

1.343750000000000

root =

1.328125000000000

root =

1.320312500000000

root =

1.324218750000000

root =

1.322265625000000

root =

1.323242187500000

root =

1.323730468750000

root =

1.323974609375000

root =

1.324096679687500

root =

1.324157714843750

root =

1.324188232421875

root =

1.324203491210938

root =

1.324195861816406

root =

1.324192047119141

root =

1.324193954467773

root =

1.324194908142090

```
root =

  1.324194431304932

root =

  1.324194669723511

root =

  1.324194550514221

root =

  1.324194490909576

root =

  1.324194461107254

root =

  1.324194446206093

root =

  1.324194453656673

root =

  1.324194449931383

root =

  1.324194448068738

root =

  1.324194449000061

root =

  1.324194449465722

root =

  1.324194449698552

root =

  1.324194449582137

root =

  1.324194449523930

root =

  1.324194449553033

root =

  1.324194449567585

root =

  1.324194449574861

root =

  1.324194449578499

root =

  1.324194449576680

root =

  1.324194449575771

approxroot =

  1.324194449575771

counter =

  40
```

thus the final approximation of the root within an accuracy of 10^{-12} is 1.324194449575771 which we get after 40 iterations.

b)

Now, we will find the root using the Newton-Raphson method.
In order to do this, we write the m-file:

```
% This implements the Newton−Raphson algorithm to find a root


format long;
f = @(x) 3*x*cos(x)−sin(x); % This is the given function
df= @(x) 2*cos(x)−3*x*sin(x);  % and its derivative
root=1; % This variable tracks of the current approximation of the root Xn
root2=0; % This variable tracks the last approximation of the root X(n−1)
counter=0; % This counts the numebr of iterations needed

while abs(root−root2)>= 10^(−12) % While we are not within the desired accuracy, we continue
    root2=root;
    root=root−(f(root)/df(root))
    counter= counter +1;
end
approxroot=root  % This gives back the final value of the root found
counter % This gives the final number of iterations.
```

which gives the successive iterations:

```
root =

  1.539847228854303

root =

  1.351814223269054

root =

  1.324820175290078

root =

  1.324194787748889

root =

  1.324194449575602

root =

  1.324194449575503

approxroot =

  1.324194449575503

counter =

   6
```

thus the final approximation of the root within an accuracy of 10^{-12} is 1.324194449575503 which we get after 6 iterations. We see that this method converges to the desired root much quicker than the bisection one.

c)

For the bisection method, we know that the interval $[a_n, b_n]$, in which the root is clamped in between, gets halved at each iteration.To estimate the root to within 10^{-M}, we need $b_n - a_n \leqslant 10^{-M}$.
Thus if we start with an interval of width $b_1 - a_1$, then $a_n - b_n = \frac{b_1-a_1}{2^n}$ and thus $b_n - a_n \leqslant 10^{-M} \iff \frac{b_1-a_1}{2^n} \leqslant 10^{-M} \iff (b_1-a_1)10^M \leqslant 2^n \iff n \geqslant \frac{Mlog(10)+log(b_1-a_1)}{log(2)}$.
That is , if $b_n - a_n \approx 10^{-M}$ then

$$n \approx \frac{Mlog(10)+log(b_1-a_1)}{log(2)}$$

So n approximately scales linearly to M. (where n is the number of iterations)

d)
The error term can be written, using Newtons scheme (as $x_n - r = e_n$):

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

as required.

Then using Taylors with the values b=r and a=x_n, we get :

$$f(r) = f(x_n) + (r - x_n)f'(x_n) + \frac{1}{2}(r - x_n)^2 f''(\xi)$$

which becomes

$$f(r) = f(x_n) - e_n f'(x_n) + \frac{1}{2}e_n^2 f''(\xi)$$

as $x_n - r = e_n$.

which becomes

$$e_n f'(x_n) - f(x_n) = \frac{1}{2}e_n^2 f''(\xi) - f(r)$$

which becomes

$$e_{n+1} = \frac{1}{2f'(x_n)}e_n^2 f''(\xi)$$

with $\xi \in [r, x_n]$

as f(r)=0 and $e_n f'(x_n) - f(x_n) = e_{n+1} f'(x_n)$.

which becomes

$$e_{n+1} = C_n e_n^2$$

with $C_n = \frac{1}{2f'(x_n)}f''(\xi)$ and $\xi \in [r, x_n]$

Then, as the errors converge to 0, we have $e_n \to 0$ ie $x_n \to r$ and $r \leqslant \xi \leqslant x_n$ so as $e_n \to 0, x_n \to r$ and thus, by the sandwich rule, $\xi \to r$, so

$$C_n = \frac{f''(\xi)}{2f'(x_n)} \to \frac{f''(r)}{2f'(r)} = C$$

because f' and f'' are continuous.

Now, we want to investigate the solution to $e_{n+1} = Ce_n^2$.

If $e_{n+1} = Ce_n^2$, then $Ce_{n+1} = C^2 e_n^2 = (Ce_n)^2$, but then, $Ce_n = C^2 e_{n-1}^2 = (Ce_{n-1})^2$, etc...

And thus by induction , we deduce that :

$$Ce_n = (Ce_0)^{2^n}$$

Then,

$$Ce_n \leqslant 10^{-M} \iff (Ce_0)^{2^n} \leqslant 10^{-M}$$

And taking logarithms on both sides, we get:

$$2^n log(Ce_0) \leqslant -Mlog(10)$$

Multiplying both sides by (-1):

$$-2^n log(Ce_0) \geqslant Mlog(10) > 1$$

And taking logarithms again:

$$nlog(2) + log(-log(Ce_0)) \geqslant log(M) + log(log(10))$$

which is :

$$n \geqslant \frac{log(M)+log(log(10))-log(-log(Ce_0))}{log(2)}$$

Thus if $Ce_n \approx 10^{-M}$,

$$n \approx \frac{log(M)+log(log(10))-log(-log(Ce_0))}{log(2)}$$

ie

$$n \approx \frac{log(M)}{log(2)}$$

ie

$$n \approx log(M)$$

so n approximately scales with log(M). (where n is the number of iterations) Indeed, we see that if we want $e_n \approx 10^{-M}$, that is $Ce_n \approx 10^{-M}$, then we just need to take logarithms twice on both sides to get $n \approx log(M)$.

2 5B Lattice Point Counting

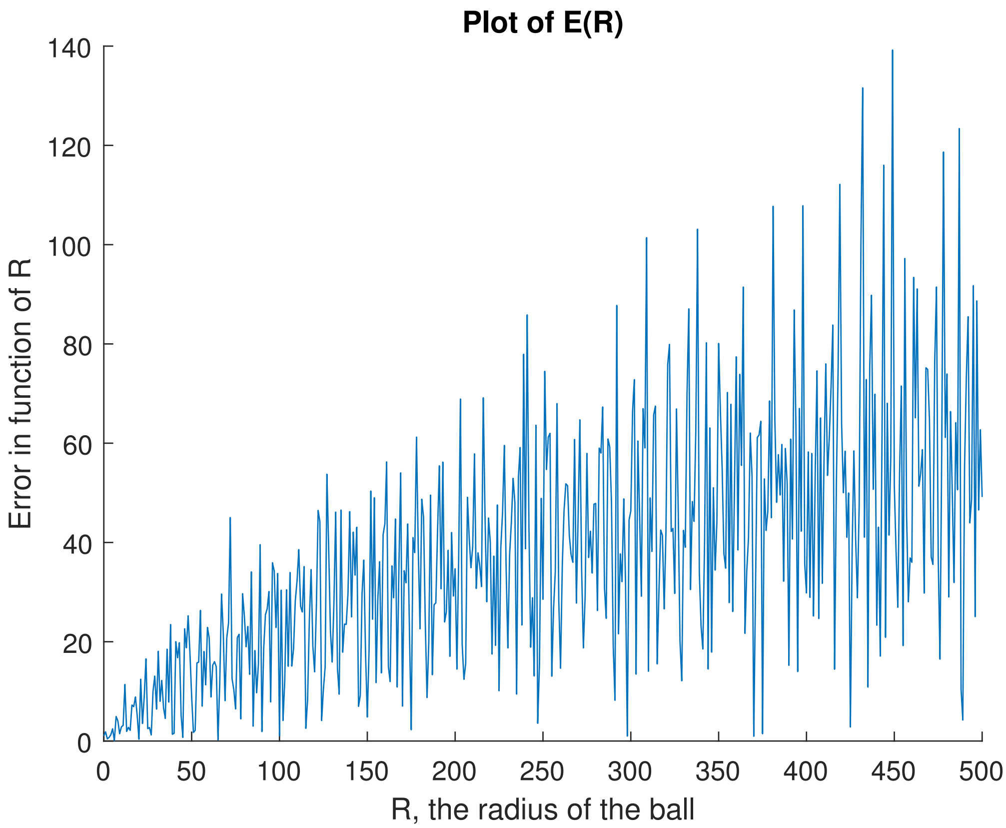
a)
To count the number of lattice points N(R) contained in the given ball, we write the m-file

```
%This file counts how many points (x,y) are in the ball described in the
%question.

function counter=counting(R)
    % Here we create a square of points which is bigger than the ball
    % described
    x= linspace(-R,R,2*R+1);
    y= linspace(-R,R,2*R+1);
    u=0; % Counts the lattice points inside the ball
    for i=1:2*R+1
        for j= 1:2*R+1
            if(sqrt(x(i)^2+y(j)^2)<=R) % checks the points are inside the ball
                u=u+1;
            end
        end
    end
counter=u;

end
```

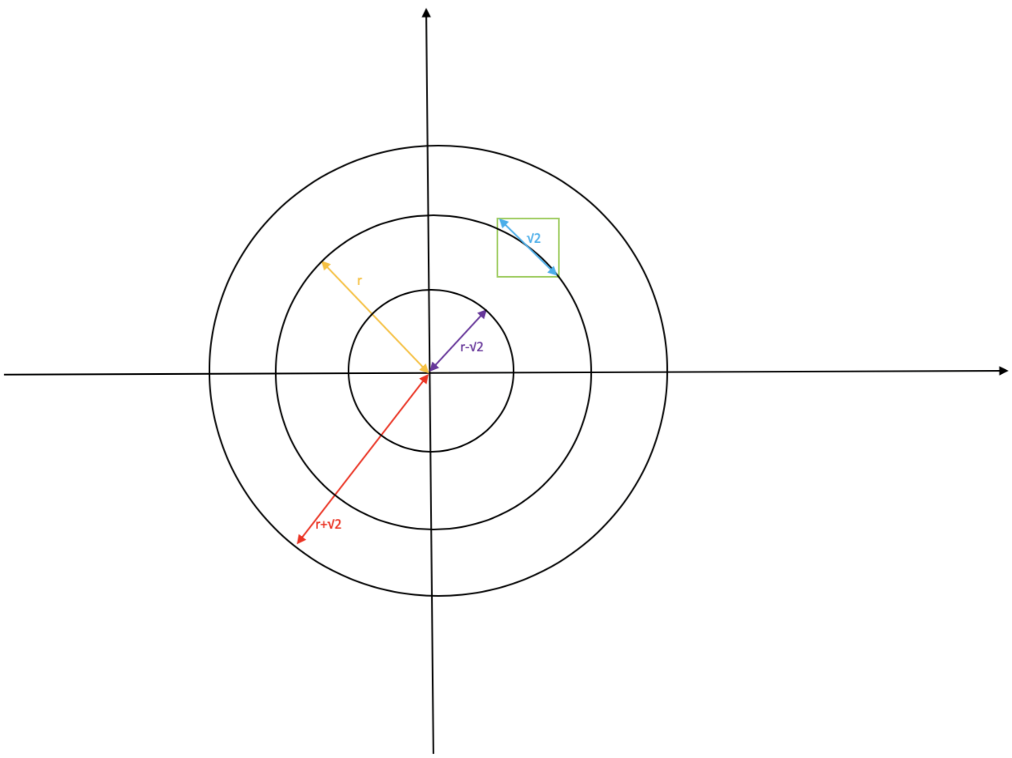
and then to plot the error E(R)=$|N(R)-\pi R|$ for $R\in[0,500]$, we write the m.file

```
% This file plots the error between the number of lattice
    points N(R) and
% the quantity PI*R

R=linspace(0,500,501); % We want it for all the points in
    [0,500].
Error=zeros(1,501); % This will contain all the errors,
    which are functions of R
for k= 1:501
    Error(k)= abs(counting(k-1)-pi*(k-1)*(k-1)) %This
        calculates the error
end

hold on
plot(R,Error)% this plots the error in function of R

plot(R,2*sqrt(2*pi)*R) % This plots the linear upper
    bound of the error

plot(R,3.1*R.^(63/100)) % This plots the more accurate
    upper bound of the error.
hold on
xlabel('R, the radius of the ball')
ylabel('Error in function of R')
title(Error E(R) and linear upper bound of E(R))
```

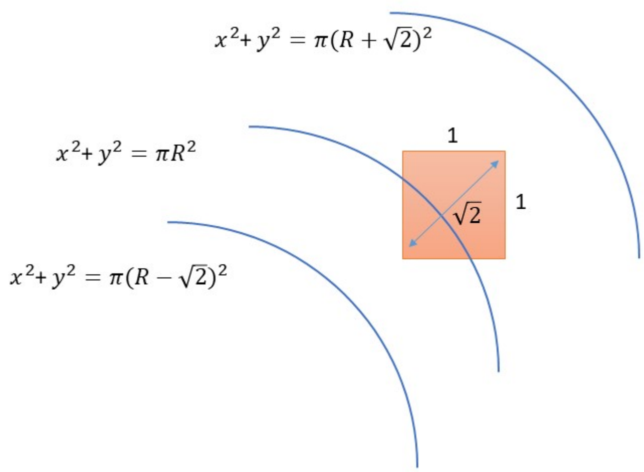
which gives the plot


Plot of E(R)
140
120
100
80
60
40
20
0
Error in function of R
0
50
100
150
200
250
300
350
400
450
500
R, the radius of the ball

b)
Let $R \geqslant 1$.
We can represent our problem as, where the lattice point is in the middle of a square of width 1:



or zooming in,



We do not know whether the lattice point is in the circle with radius R but we know for sure it will be in the circle of radius $R+\sqrt{2}$ and, we know for sure it will not be in the circle of radius $R-\sqrt{2}$.

Thus we deduce,

$$\pi(R-\sqrt{2})^2 \leqslant N(R) \leqslant \pi(R+\sqrt{2})^2$$

ie

$$2\pi - 2\sqrt{2}\pi R \leqslant N(R) - \pi R^2 \leqslant 2\pi + 2\sqrt{2}\pi R$$

ie

$$-2\pi R - 2\sqrt{2}\pi R \leqslant N(R) - \pi R^2 \leqslant 2\pi R + 2\sqrt{2}\pi R$$

since $R \geqslant 1$.
ie

$$-2\pi R(\sqrt{2}+1) \leqslant N(R) - \pi R^2 \leqslant 2\pi R(\sqrt{2}+1)$$

ie

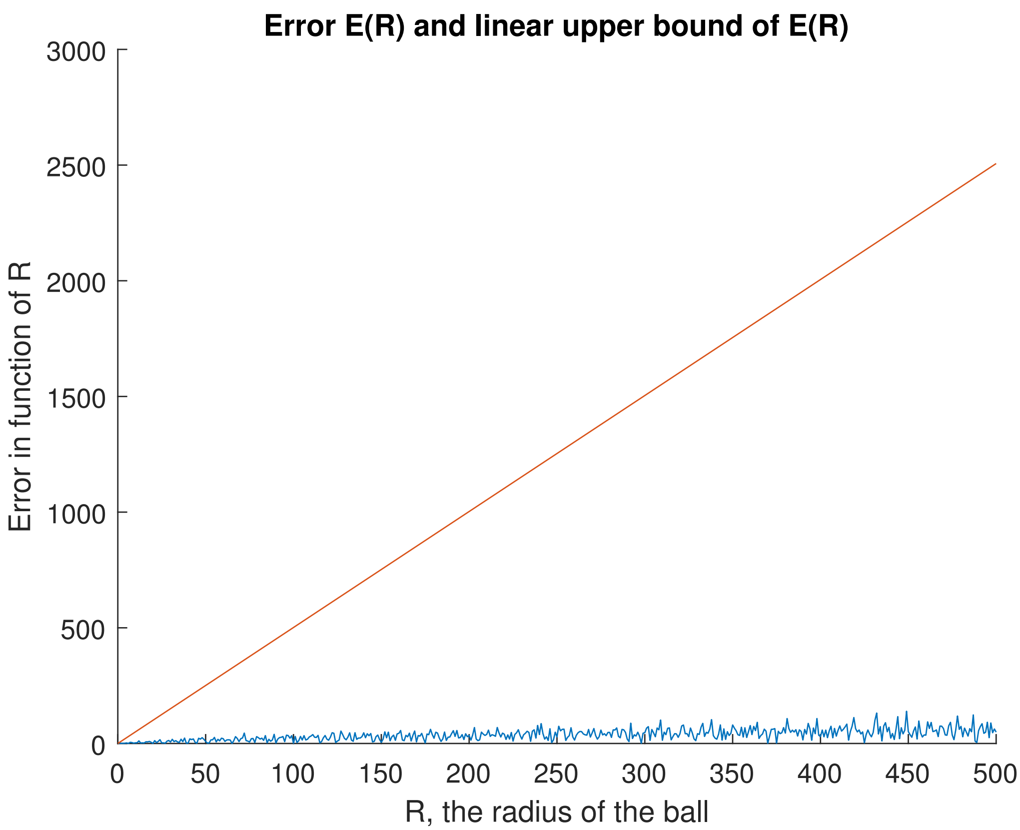
$$|N(R) - \pi R^2| \leqslant 2\pi R(\sqrt{2}+1)$$

so if C= $2\pi(\sqrt{2}+1)$, we have

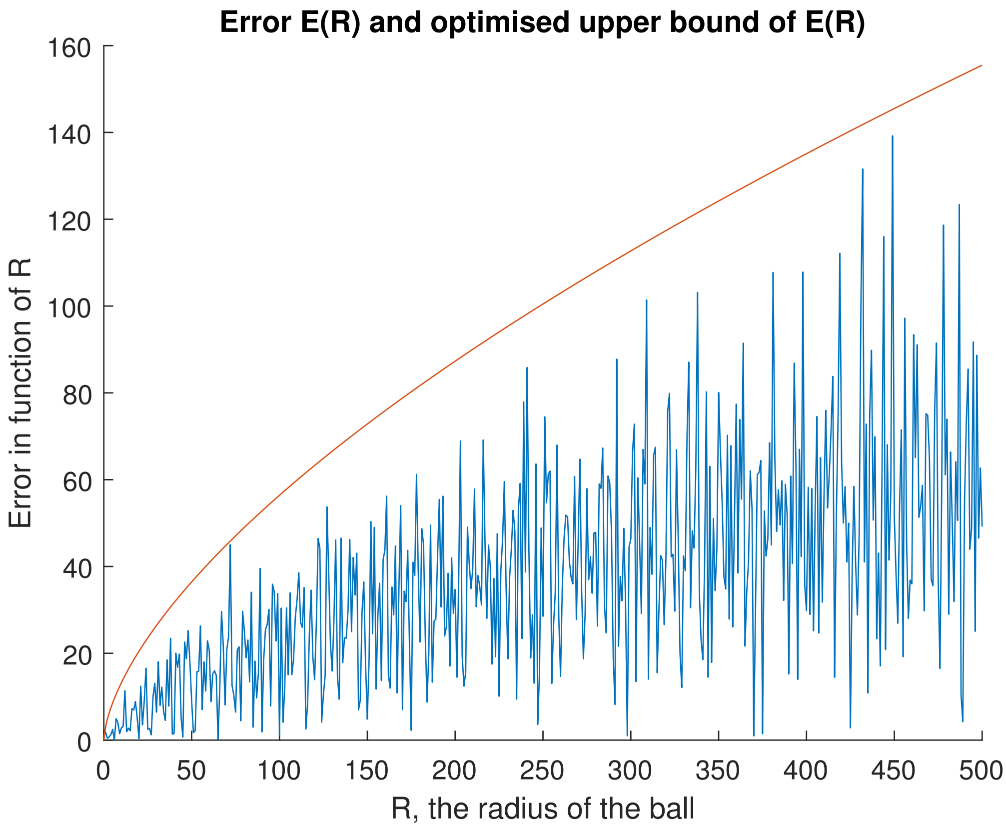
$$|N(R) - \pi R^2| \leqslant CR$$

as required.

c)
With the linear plot, using C= $2\sqrt{2\pi} \leqslant 2\pi(\sqrt{2}+1)$ (but CR is still an upper bound of |E(R)|), we get the plot:



and using the optimised upper bound with $\alpha = \frac{63}{100}$ and $C = 3.1$, we get $|E(R)| \leqslant CR^{\alpha}$ and the plot



.

Thus clearly, the second upper bound is much better and tighter than the first one.