# Homework 6

Yash Ganatra

April 11, 2016

## Problem 0: Homework checklist

Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.

- Zhehuan Xu, Nate Majid Kesto, Emily Zimovan,Woi Sok Oh
- Coding Language: Matlab
- $\epsilon = 2.24 \times 10^{-16}$ (from Matlab command **eps(1)**)
- To caculate time taken to run a function,*timeit* function is used from *Mathworks*

## Problem 1: Gautschi Exercise 4.1

The rate of convergence of *order p* is given by Equation 1

$$\lim_{n\to\infty} \frac{x_{n+1} - \alpha}{(x_n - \alpha)^p} \quad \text{where } \alpha = \lim_{n\to\infty} x_k = 0 \tag{1}$$

Equation 2 shows the limits. Since they are indeterminate form they can be computed using L'Hospitals Rule

$$\begin{aligned}
&\lim_{n\to\infty} \frac{(n+1)^{-10}-\alpha}{(n)^{-10}-\alpha} = 1 \\
&\lim_{n\to\infty} \frac{(10)^{-(n+1)}-\alpha}{(10)^{-n}-\alpha} = \frac{1}{10} \\
&\lim_{n\to\infty} \frac{(10)^{-(n+1)^2}-\alpha}{(10)^{-(n)^2}-\alpha} = \lim_{n\to\infty} 10^{-2\,n-1} = 0 \qquad (2) \\
&\lim_{n\to\infty} \frac{(n+1)^{10}\,(3)^{-(n+1)}-\alpha}{(n)^{10}\,(3)^{-n}-\alpha} = \frac{1}{3} \\
&\lim_{n\to\infty} \frac{(10)^{-3)\cdot 2^{(n+1)}}-\alpha}{(10^{-3\cdot 2^n}-\alpha)^2} = 1
\end{aligned}$$

Figures 1, 2 show the rates of convergence whose orders are shown in Table 1

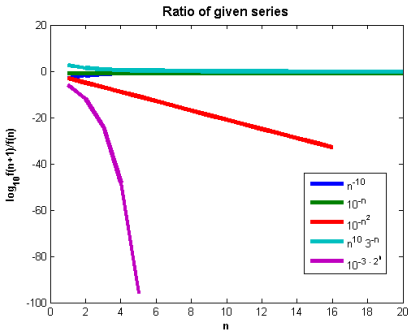


Figure 1: Rate of convergence for given series

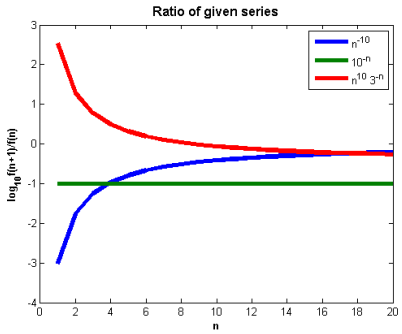


Figure 2: Rate of convergence for given series

.

Table 1: Rates of convergence for given sequences

| Sequence | Limit | Rate of Convergence |
|---|---|---|
| $n^{-10}$ | 1 | Sublinear |
| $10^{-n}$ | $\frac{1}{10}$ | Linear |
| $10^{-n^2}$ | $10^{-2\,n-1} = 0$ as n | Superlinear |
| $n^{10}\,3^{-n}$ | 3 | Linear |
| $10^{-3\cdot 2^n}$ | 1 | Quadratic |

Code is in Appendix 1

## Problem 2: Implementing a core routine

*Comments*

- Newtons method is used for root finding $f(x) = x^2 - c = 0$ to find$\sqrt{c}$. Codes in Appendix 2,3,
- This method assumes that $x > 0$ & $x \in \mathbb{R}$
- From Figure 3, the error between value computed using built in ($sqrt$) function and via Newton's method is of the order of $\approx 10^{-15}$ which indicates a good fit. The markers are closely spaced closer to 0 due to the spacing of floating point numbers.
- In Newton's method when $f\prime(x_k) \to 0$,

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f\prime(x_k)} \\ x_{k+1} &= \frac{1}{2}\left(x_k + \frac{c}{x_k}\right) \end{aligned} \tag{3}$$

- For $c = 0$, $\sqrt{c} = \epsilon$ where $\epsilon =$ Machine Precison. From Equation 3, the method reduces to $x_{k+1} = \frac{x_k}{2} \neq 0$. This may cause ***severe cancellation errors during subtraction and division***

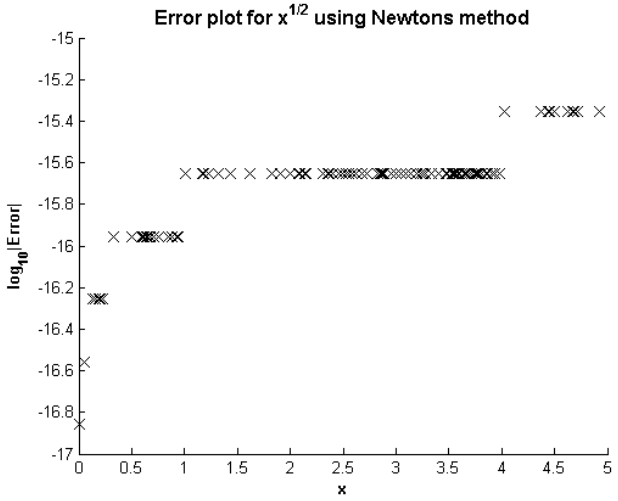


Figure 3: Error plot for $\sqrt{x}$ calculated using Newton's method and built-in MATLAB command

## Problem 3: Comparing the algorithms (Gautschi Machine Exercise 4.2)

Convergence can be defined as

$$\begin{aligned} &|f(x)| < tol \\ &|x_n - x_{n-1}| < tol(usedhere) \\ &tol = ||x_n||\,\epsilon_r + \epsilon_a \\ &\epsilon_r, \epsilon_a \text{are Relative and absolute tolerances} \end{aligned} \tag{4}$$

Codes are in Appendix 4,5,6,7,8

Table 2: Convergence and Peformance evaluation of rootfinding algorithms for $(f(x) = (1/2)x - \sin x = 0$

| **Relative Tolerance** | **Absolute Tolerance** | **Bisection** | **False Position** | **Secant** | **Newton** |
|---|---|---|---|---|---|
| $\epsilon$ | $\epsilon$ | 53 | 39 | 12 | 6 |
| $1e-7$ | $1e-7$ | 24 | 20 | 11 | 5 |
| **Function Evaluations (per iteration)** | | 2 | 3 | 1 | 2 |

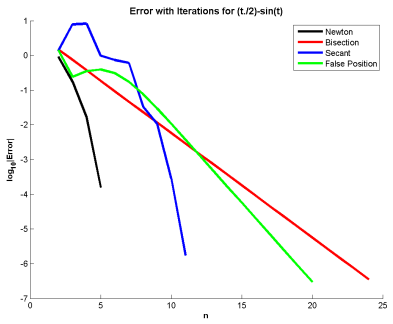


Figure 4: Error plots for 7 digit precision

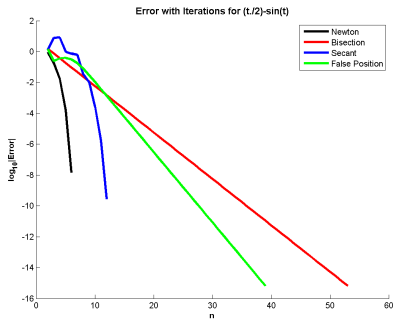


Figure 5: Error plots for full machine (32-bit) precision

*Comments*

- The bisection method has linear convergence and the number of steps can be calculated by

$$iterations = \frac{log\left(\frac{b-a}{tol}\right)}{log\,2}$$

  which is consistent with the iterations obtained from the code. Bisection method gurantees convergence.

- The false position method has linear convergence with asymptotic constant as $k \to \infty$.

- The secant method has convergence of order $p = 1.61803\ldots$ which implies its faster than Bisection and False Position
- Newton's method has quadratic convergence and is the fastest amongst these. The choice of initial guess affects the convergence behaviour in this method. Also computing the derivative computationally expensive if the function is complicated.
- **Secant method require less time per iteration whereas Newton's method requires less iterations**

## Appendix

Listing 1: Rates of convergence for given series

```
clc;close all;clear all;
n = 1:1:20;
k1 = @(n) n.^-10;
k2 = @(n) 10.^-n;
k3 = @(n) 10.^(-n.^2);
k4 = @(n) n.^10.*3.^-n;
k5 = @(n) 10.^(-3.*2.^n);
hold on
figure(1)
plot(n,log10(k1(n)),n,log10(k2(n)),n,log10(k3(n)),n,log10
    (k4(n)),...
     n,log10(k5(n)),'LineWidth',4)
legend('n^{-10}','10^{-n}','10^{-n^2}','n^{10} 3^{-n}','
    10^{-3 \cdot 2^n}')
figure(2)
plot(n,log10(k1(n+1)./k1(n)),n,log10(k2(n+1)./k2(n)),n,
    log10(k3(n+1)./k3(n))...
     ,n,log10(k4(n+1)./k4(n)),n,log10(k5(n+1)./k5(n)),'
         LineWidth',4)
legend('n^{-10}','10^{-n}','10^{-n^2}','n^{10} 3^{-n}','
    10^{-3 \cdot 2^n}')
xlabel('n','FontWeight','bold','FontSize',10)
ylabel('log_{10}f(n+1)/f(n)','FontWeight','bold','
    FontSize',10)
title('Ratio of given series'...
     ,'FontWeight','bold','FontSize',12)
figure(3)
plot(n,log10(k1(n+1)./k1(n)),n,log10(k2(n+1)./k2(n)),...
  n,log10(k4(n+1)./k4(n)),'LineWidth',4)
legend('n^{-10}','10^{-n}','n^{10} 3^{-n}')
xlabel('n','FontWeight','bold','FontSize',10)
ylabel('log_{10}f(n+1)/f(n)','FontWeight','bold','
    FontSize',10)
title('Ratio of given series'...
     ,'FontWeight','bold','FontSize',12)
```

Listing 2: Newton-Raphson method for $\sqrt{c}$ - Main

```
clc;close all;
clear all;
%% Newtons Method for sqrt(c)
%for c = 0:1e-2:5
c =3;
```

```
initial_guess = 1;% Initial value
%%
syms t
f = @(t) t.^2-c;
g=diff(f(t));
f_der  = matlabFunction(g);
%%
error_rel = 0;
error_abs = eps(0);
% [x,c_sq_new] = p2_newton_func(f,f_der,initial_guess,
    error_rel,error_abs);
[x,c_sq_new] = p2_newton_func2(c,initial_guess,error_rel,
    error_abs);
%%
c_sq_new
x = x(1:c_sq_new)
% figure(1)
% plot(1:c_sq_new,abs(sqrt(c)-log10(x)),'kx')
% figure(2)
% hold on
% plot(c,log10(abs(sqrt(c)-(x(end)))),'kx','MarkerSize
    ',10)
% %plot(c,c_sq_new,'bo')
% xlabel('x','FontWeight','bold','FontSize',10)
% ylabel('log_{10}|Error|','FontWeight','bold','FontSize
    ',10)
% title('Error plot for x^{1/2} using Newtons method'...
%     ,'FontWeight','bold','FontSize',12)
%end
%% Timing
 g = @() p2_newton_func(f,f_der,initial_guess,error_rel,
     error_abs);
 h = @() sqrt(c);
 diffRunTime = timeit(g)-timeit(h)
```

Listing 3: Newton-Raphson method for $\sqrt{c}$ - Function

```
function [x,count,error_new] = p2_newton_func2(c,x_0,
    error_rel,error_abs)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
error_new = zeros([1,maxiter]);
x = zeros([1,maxiter]);
while(error > tol)
```

```
    count = count+1;
    if count==1
    %x(count) = x_0 - f(x_0)./f_der(x_0);
    x(count) = 0.5.*(x_0 + c./x_0);
    error = abs(x(count)-x_0);
    else
%     x(count) = x(count-1) - f(x(count-1))./f_der(x(
    count-1));
    x(count) = 0.5.*(x(count-1) + c./x(count-1));
    error = abs(x(count)-x(count-1));
    end
    tol =  abs(x(count)).*error_rel + error_abs;
    %tol =  error_abs;
    if count > maxiter
        display(' No Conver Reached - Newtons');
        break
    end
error_new(count)= error;
end
end
```

Listing 4: Root finding algorithms for $f(x) = (1/2)x - \sin x = 0$ - Main

```
clc;close all;
clear all;

initial_guess = 3;% Initial value
%%
syms t
f = @(t) (t./2)-sin(t);
g=diff(f(t));
f_der  = matlabFunction(g);
% range = [2.1,3.5];
range = [1/(2*pi),pi];
%%
error_rel = eps;
error_abs = eps;
%% Bisection
%[x_bi_ext,c_bi] = p3_bisection2(f,range);
[x_bi_ext,c_bi,error_bi,tol_bi] = p3_bisection(f,range,
    error_rel,error_abs);
x_bi = x_bi_ext(1:c_bi);
display('B')
x_bi(end)
n_bi = ceil(log(abs(range(2)-range(1))./tol_bi)./log(2))
    ; % Iterations needed for Bisection
```

```
c_bi
%% False Position
[x_fp_ext,c_fp,error_fp] = p3_false_pos(f,range,error_rel
    ,error_abs);
x_fp = x_fp_ext(1:c_fp);
display('FP')
x_fp(end)
c_fp
%% Secant
[x_sec_ext,c_sec,error_sec] = p3_secant(f,range,error_rel
    ,error_abs);
x_sec = x_sec_ext(1:c_sec);
display('S')
x_sec(end)
c_sec
%% Newtons
[x_new_ext,c_new,error_new] = p2_newton_func(f,f_der,
    initial_guess,error_rel,error_abs);
x_new = x_new_ext(1:c_new);
display('N')
x_new(end)
c_new
%% Error Plots
% figure(2)
% hold on
% plot(2:c_new,log10(error_new(1:c_new-1)),'k','LineWidth
    ',3)
% plot(2:c_bi,log10(error_bi(1:c_bi-1)),'r','LineWidth
    ',3)
% plot(2:c_sec,log10(error_sec(1:c_sec-1)),'b','LineWidth
    ',3)
% plot(2:c_fp,log10(error_fp(1:c_fp-1)),'g','LineWidth
    ',3)
% legend('Newton','Bisection','Secant','False Position')
% xlabel('n','FontWeight','bold','FontSize',10)
% ylabel('log_{10}|Error|','FontWeight','bold','FontSize
    ',10)
% title('Error with Iterations for (t./2)-sin(t)','
    FontWeight','bold','FontSize',12)
%%% Saving file
% saveas(gcf,'D:\Courses\Spring 2016\Numerical analysis\
    HW\HW6\Results\p3_Error_r_0_a_eps',...
%     'png')
% file_path = 'D:\Courses\Spring 2016\Numerical analysis\
    HW\HW6\Results';
% file_name = strcat(file_path,'\p3_r_', num2str(
```

```
    error_rel,'%1.0e'),'_a_',num2str(error_abs,'%1.0e'));
% saveas(gcf,file_name,'png');
%%
%%
%% Plot Function
% figure(1)
% x_grid =linspace(range(1),range(2),20);
% plot(x_grid,f(x_grid))
% hline = refline([0 0]);
% hline.Color = 'r';
% axis('tight')
% xlabel('x','FontWeight','bold','FontSize',10)
% ylabel('(x/2)-sin(x)','FontWeight','bold','FontSize
    ',10)
% title('Function plot','FontWeight','bold','FontSize
    ',12)
```

Listing 5: Bisection method - Function

```
function [x,count,error_bi,tol] = p3_bisection(f,range,
    error_rel,error_abs)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
error_bi = zeros([1,maxiter]);% Pre-Allocation
x = zeros([1,maxiter]); % Pre-Allocation
while(error>tol)
    count = count+1;
    x(count) = sum(range)./2;
    if f(x(count))*f(range(1))<0
        range = [range(1),x(count)];
        error = abs(x(count)-range(1));
    %elseif f(x(count))*f(range(1))>0
    else
        range = [x(count),range(2)];
        error = abs(x(count)-range(2));
    end
    tol =  abs(x(count)).*error_rel + error_abs;
    if count > maxiter
        display(' No Conver Reached - Bisection');
        break
    end
    error_bi(count)= error;
end
end
```

Listing 6: False Position - Function

```
function [x,count,error_fp] = p3_false_pos(f,range,
    error_rel,error_abs)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
error_fp = zeros([1,maxiter]);
x = zeros([1,maxiter]);
while(error > tol)
    if range(1) > range(2)
        display('Range(1) < Range(2) needed')
        break
    end
    count = count+1;
  x(count) = range(1) - ((range(1)-range(2)).*f(range(1))
     ...
            ./(f(range(1))-f(range(2))));
    if f(x(count))*f(range(1))<0
        range = [range(1),x(count)];
    else
        range = [x(count),range(2)];
    end
    if(count==1)
        error = abs(x(count)-sum(range)./2);
    else
        error = abs(x(count)-x(count-1));
    end
    tol =  abs(x(count)).*error_rel + error_abs;
    if count > maxiter
        display(' No Conver Reached - False Position');
        break
    end
    error_fp(count)= error;
end

end
```

Listing 7: Secant method - Function

```
function [x,count,error_sec] = p3_secant(f,range,
    error_rel,error_abs)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
```

```
error_sec = zeros([1,maxiter]);
x = zeros([1,maxiter]);

while(error > tol)
    if range(1) > range(2)
        display('Range(1) < Range(2) needed')
        break
    end
    count = count+1;
    if count==1
        x(count) = sum(range)./2; % Arbitary --> Midpoint
            of range
        x(count+1) = range(2) - ((range(2)-range(1)).*f(
            range(2))...
            ./(f(range(2))-f(range(1))));
    else
        x(count+1) = x(count) - ((x(count)-x(count-1)).*f
            (x(count))...
            ./(f(x(count))-f(x(count-1))));
    end
    error = abs(x(count+1)-x(count));
    tol =  abs(x(count)).*error_rel + error_abs;
    if count > maxiter
        display(' No Conver Reached - Secant');
        break
    end
    error_sec(count)= error;
end
end
```

Listing 8: Newton method - Function

```
function [x,count,error_new] = p2_newton_func(f,f_der,x_0
    ,error_rel,error_abs)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
error_new = zeros([1,maxiter]);
x = zeros([1,maxiter]);
while(error > tol)
    count = count+1;
    if count==1
    x(count) = x_0 - f(x_0)./f_der(x_0);
    error = abs(x(count)-x_0);
    else
```

```
    x(count) = x(count-1) - f(x(count-1))./f_der(x(count
        -1));
    error = abs(x(count)-x(count-1));
    end
    tol =  abs(x(count)).*error_rel + error_abs;
    if count > maxiter
        display('No Conver Reached');
        break
    end
error_new(count)= error;
end
end
```