UMEÅ UNIVERSITET
Department of Computer Science

December 8, 2021

5DV088

Assignment 1

Namn Mathias Hallberg
Gustaf Söderlund

Mail c19mhg@student.umu.se
et14gsd@student.umu.se

Version 2

Course
Scientific Computing
Handledare
Carl Christian Kjelgaard Mikkelsen Fredrik Petri

Contents

1 introduction

In this paper the development of the function myRoot is described. The paper is divided in the following sections to give the reader an easier read. Construction of test cases will give the reader the information of how to acquire simple polynomials with Chebyshev polynomials. In the myRoot section the matlab code is presented. In the section Calculations the result of the program is presented.

2 Construction Of Test Cases

2.1 Chebyshev poynomials

To construct the function *myRoot* polynomials are needed. In this project it's general formula for finding Chebyshev polynomials.

$$T_0(x) = 1 \tag{1}$$

$$T_1(x) = x \tag{2}$$

$$T_{j+1}(x) = 2xT_j(x) - T_{j-1}(x),\ j \ \epsilon \ \{1, 2, 3, ...\} \tag{3}$$

Find the explicit formula for T_n for n = 3, 4, 5, 6.

$$\begin{aligned} T_3(x) = 2xT_{j-1}(x) - T_{j-1}(x) = \\ 2x(2x^2 - 1) - x = \\ 4x^3 - 3x \end{aligned} \tag{4}$$

$$\begin{aligned} T_4(x) = 2xT_{j-1}(x) - T_{j-1}(x) = \\ 2x(4x^3 - 3x) - (2x^2 - 1) = \\ 8x^4 - 8x^2 + 1 \end{aligned} \tag{5}$$

$$\begin{aligned} T_5(x) = 2xT_{j-1}(x) - T_{j-1}(x) = \\ 2x(8x^4 - 8x^2 + 1) - (4x^3 - 3x) = \\ 16x^5 - 20x^3 + 5x \end{aligned} \tag{6}$$

$$\begin{aligned} T_6(x) = 2xT_{j-1}(x) - T_{j-1}(x) = \\ 2x(16x^5 - 20x^3 + 5x) - (8x^4 - 8x^2 + 1) = \\ 32x^6 - 48x^4 + 18x^2 - 1 \end{aligned} \tag{7}$$

2.2 Proof by induction

Show by induction that T_n has degree n.

Using the principle of mathematical induction from the pdf [1] chapter 2. We have the following, if $V \subseteq \mathbb{N}$ satisfies the following cases

1. $1 \in V$, and
2. $n \in V$ implies that $n+1 \in V$, then $V = \mathbb{N}$

We can now start doing the proof of induction. First of all we have to choose a set V such that the following hypothesis $n \in V$ can provide us with the information needed to prove that $n+1 \in V$. We set the V as the following

$$V = \{n \in \mathbb{N} : T_k \ has\ degree\ k\ for\ k \in \{n-1, n\}\} \tag{8}$$

We know that $T_0(x) = 0$ which implies that T_0 has degree 0 and $T_1(x) = x$ has degree 1. By letting $n \in V$ we need to prove that $n+1 \in V$ by showing that T_{n+1} has degree $n+1$.
Defined in the assigment we have the following definition of T_{n+1}

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x),\ n \in \{1, 2, 3, ..., n\} \tag{9}$$

Looking at what we knew from the beginning that $T_0(x)$ is of degree 0 and $T_1(x)$ is of degree 1. We can now expect that T_n has n degrees and T_{n-1} has $n-1$ degrees. Since $2x$ is of degree one it is expected that T_{n+1} is of degree $n+1$. By this conclusion it shows that $n+1 \in V$ and by the principle of mathematical induction that $V = \mathbb{N}$.

2.3 MyChebyshev.m

```
function y=MyChebyshev(n,x)

% A1F1 Evaluates the first n Chebyshev polynomials
%
% CALL SEQUENCE: y=a1f1(n,x)
%
% INPUT:
%   n     the number of polynomials
%   x     a vector of length m containing the sample points
%
% OUTPUT:
%   y      a matrix of dimension m by n such that y(i,j) = T(j,x(i))
%
% MINIMAL WORKING EXAMPLE: myChebyshevMWE1

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (spock@cs.ume.se)
%                    Mathias Hallberg (c19mhg@cs.umu.se)
%                    Gustaf S derlund (et14gsd@cs.umu.se)
%  2018-11-14 Skeleton extracted from working function
%  2020-11-04 Minor polishing ...
%  2021-11-28 Added calculations

% Determine number of element in x
m=numel(x);

% Reshape x as a column vector
x=reshape(x,m,1);

% Allocate space for output y
y=ones(m,n);

% Initialize the first two columns of y

y(:,1)=ones(m,1);
y(:,2)=y(:,2).*x;

% Calculate all remaining columns of y
for i=3:n
    y(:,i)=2.*x.*y(:,i-1)-y(:,i-2);
end
```

2.4 MyChebyshevMWE1.m

```
clc;clear;
% A1F2 Minimal working example for a1f1

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen
%                Mathias Hallberg (c19mhg@cs.umu.se)
%                Gustaf S derlund (et14gsd@cs.umu.se)
%   2018-11-14 Skeleton extracted from working code
%   2021-11-28 Added working plot

% Set number of polynomials
n=11;

% Set number of sample points
x=linspace(-1,1,1000);

% Generate function values
y=MyChebyshev(n, x)

% Plot all graphs with one command
plot(x, y(:,end));
grid on;

% Adjust axis to make room for legend
axis([-1 1 -1.5 2.5]);

% Construct and display legend
str=[];
for i=0:n-1
    str=[str strcat("n=",string(i))];
end
legend(str);
```

2.5 Proving n roots are given by T_n

Show that the roots of T_n are given by

$$x_k = \cos\left(\frac{(2k-1)\pi}{2n}\right),\ k = 1, 2, ..., n. \tag{10}$$

First of all we have to show that each x_k is a root and we have the following to start with

$$T_n(\cos(\theta)) = \cos(n\theta), \theta\ \in\ \mathbb{R} \tag{11}$$

And that is followed by

$$T_n(x_k) = T_n(\cos(\theta_k)) = \cos(n\theta_k) = \cos\left(\frac{2k-1}{2}\pi\right) = \cos(k\pi - \frac{\pi}{2}) = 0 \tag{12}$$

Further on we need to prove that there are distinct values generated by equation 9. Since the cosinus function is periodic we can't know for certain that numbers aren't repeated. $[\theta_k]_{k=1}^{n}$ is the set of n numbers. Further on to prove that the set of numbers contain distinct numbers, suppose that $\theta_i = \theta_j$, which gives us

$$\frac{(2i-1)\pi}{2n} = \frac{(2j-1)\pi}{2n} \Leftrightarrow 2i - 1 = 2j - 1 \Leftrightarrow i = j \tag{13}$$

Equation 12 then shows that the set contains n distinct numbers ranging from

$$0 < \theta_1 < ... < \theta_n < \pi \tag{14}$$

We now have the interval $(0, \pi)$ and since $\cos(\theta)$ is strictly decreasing in the interval between $(1, -1)$, it shows that

$$1 > \cos(\theta_1) > ... > \cos(\theta_n) > -1 \tag{15}$$

This presents that the Equation (9) only has n distinct real numbers in the interval $(-1, 1)$. Now according to the fundamental theorem of algebra, T_n has n roots. It is shown that we have n and there are no possible outcome for any other roots.

3 myRoot

3.1 myRoot.m

```
function [x, flag, it, a, b, his, y, reb]=myRoot(p,a0,b0,
    delta,eps,maxit)

% A3F3 Finds roots of polynomials using the bisection
    method
%
% CALL SEQUENCE: missing
%
% INPUT:
%   p        array of coefficients used by myHorner
%   a0, b0   the initial bracket
%   delta    return if current bracket is less than delta
```

```
%   eps       return if current residual is less than epsilon
%   maxit     return after maxit iterations
%
% OUTPUT:
%   x       final approximation of the root
%   flag    a flag signaling succes or failure,
%                flag  = -2  the initial bracket is bad
%                flag  = -1  the sign of f(a0) or f(b0) cannot be trusted
%                flag  =  0  maxit iterations completed without convergence
%                flag  >  0  then convergence has been achieved and if
%                     bit 0 set   then the last bracket is shorter than delta
%                     bit 1 set   then the last function value is bounded by eps
%                     bit 2 set   then the sign of the last function value cannot
%                                 be trusted
%   it      the number of iterations completed
%   a, b    a(j) and b(j) form the jth bracket around his(j)
%   his     a vector containing all computed approximations of the root
%   y       the computed values of y=p(his)
%   reb     the running error bounds for y
%
% MIMIMAL WORKING EXAMPLE: MyRootMWE1

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (spock@cs.umu.se)
%                   Mathias Hallberg (c19mhg@cs.umu.se)
%                   Gustaf S derlund (et14gsd@cs.umu.se)
%   2018-11-14 Skeleton extracted from working code myRoot
%   2020-11-04 Minor polishing
%   2021-11-28 Finished the skeleton

% Initialize the flag.
flag=0;

% Dummy initialization of *all* output arguments
x=NaN; it=0; flag=0;
a=zeros(1, maxit); b=zeros(1, maxit);
his=zeros(1,maxit); reb=zeros(1,maxit); y=zeros(1,maxit);
% Initialize search bracket (alpha,beta) such that alpha <= beta
if b0<a0
    alpha=b0; beta=a0;
else
    alpha=a0; beta=b0;
end
```

```
% Compute fa=p(alpha) and fb=p(beta) and associated error
    bounds
[fa, ~, rebfa]=myHorner(p, alpha);
[fb, ~, rebfb]=myHorner(p, beta);

% Investigate if the flag should be -2 or -1
if sign(fa)*sign(fb)>0
   flag=-1;
elseif abs(fa)<=rebfa || abs(fb)<=rebfb
    flag=-2;
end


if (flag<0)
    % The initial bracket is either bad or cannot be
        judged
    return
end

% Main loop
for j=1:maxit
    % Record the current search bracket
    a(j)=alpha; b(j)=beta;
    % Carefully compute the midpoint c of the current
        search bracket
    c=alpha+(beta-alpha)/2;
    % Evaluate fc = p(c) and the running error bound for
        fc
    [fc, ~, rebfc]=myHorner(p,c);
    % Save the current values
    x=c; his(j)=c; y(j)=fc; reb(j)=rebfc;

    % Check for small bracket
    if abs(beta-alpha)<=delta
        flag=1;
    end

    % Check for small residual
    if abs(fc)<=eps
        flag=flag+2;
    end

    % Check if the computed sign of the p(c) cannot be
        trusted
    if abs(y(j)) <=reb(j)
        flag=flag+4;
        break
    end

    % Check if we can break out of the loop
    if flag>0
        % Yes, there is no reason to continue
        break
    end
```

```

    % ------------------------------------------------------------------------
    % At this point we know that we need more iterations.
    % ------------------------------------------------------------------------

    % Rebracket the root and recycle the old function values
    if sign(fa)*sign(fc)==-1
        beta=c; fb=fc;
    else
        alpha=c; fa=fc;
    end
end

% Shrink the output to avoid tails of unnecessary zeros
a=a(1:j); b=b(1:j); his=his(1:j); reb=reb(1:j); y=y(1:j);

% Return the number of iterations
it=j;
```

3.2 myRootMWE1.m

```
% MyRoot Minimal working example for MyRoot

% PROGRAMMING by Gustaf S derlund (et14gsd@cs.umu.se)
%                Mathias Hallberg (c19mhg@cs.umu.se)
%
%   2021-11-28 completed the base for the minimal working example


clc;clear;
% T10 for computed values used by myRoot
p=[-1, 0, 50, 0, -400, 0, 1120, 0, -1280, 0, 512];
% Set brackets between -1 and 1 andd let max iterations be 101
a0=-1;b0=1;
maxit=101;
% Check size for array of polynomials
size = size(p);
% Set delta and eps.
delta=10^-10; eps=10^-10;
% Set brackets to check the roots
cosBrackets = linspace(0, pi, size(1,2));
brackets = -cos(cosBrackets);
% Initalize the output values to set size
x = zeros(1,size(1,2)-1); flag = zeros(1,size(1,2)-1);
it=zeros(1,size(1,2)-1); ab=zeros(1,size(1,2)-1);
bb=zeros(1,size(1,2)-1); yb=zeros(1,size(1,2)-1);
rebb=zeros(1,size(1,2)-1);
```

```
for i=1:size(1,2) - 1
    [x(i), flag(i), it(i), a, b, his, y, reb]=myRoot(p,
        brackets(i),brackets(i+1),delta,eps,maxit);
    ab(i)=a(it(i));
    bb(i)=b(it(i));
    yb(i)=y(it(i));
    rebb(i)=reb(it(i));
end

% Get the relative error using the algorithm
% (1/2)*abs(a-b)/min(abs(a),abs(b))
n=linspace(1,size(end) -1, size(end)-1);
relErr=zeros(1,size(end)-1);
for i=1:size(end)-1
    if sign(ab(1,i)) == sign(bb(1,i))

        relErr(1,i)=(0.5*abs(ab(1,i)-bb(1,i))/(min(abs(ab
            (1,i)), abs(bb(1,i)))));
    else
        relErr(1,i)=NaN;
    end
end

% Calculate trust
trust=abs(yb(1,:))>abs(rebb(1,:));

% Select the data for printing
data=[n(1,:)' flag(1,:)' it(1,:)' ab(1,:)' bb(1,:)' x
    (1,:)' yb(1,:)' rebb(1,:)' relErr(1,:)' trust'];

% Define the column headingsh
colheadings = {'Idx','flag', 'iter','a','b','root','
    residual', 'REB', 'relative error', 'Trust'};

% Set the widths of the columns
wids=[6 6 6 12 12 12 12 12 16 8];
% Define the format specification
fms={'d','d','d','.4e','.4e', '.4e','.4e','.4e', '.4e', '
    d'};

% Print the data nicely
displaytable(data,colheadings,wids,fms);
```

4 Calculations

From myRoot we get 10 different roots using T_{10} which are all approximations of the real root. We need to be assured that the approximation is close enough to the real root.

4.1 All roots for T_{10}

Idx	flag	iter	a	b	root	residual	REB	relative error	Trust
1	1	30	-9.8769e-01	-9.8769e-01	-9.8769e-01	-2.5741e-09	6.3407e-13	4.6150e-11	1
2	3	32	-8.9101e-01	-8.9101e-01	-8.9101e-01	-8.6401e-12	3.2028e-13	3.7117e-11	1
3	3	33	-7.0711e-01	-7.0711e-01	-7.0711e-01	1.7804e-11	7.8604e-14	3.6423e-11	1
4	3	33	-4.5399e-01	-4.5399e-01	-4.5399e-01	-4.1272e-11	8.9947e-15	7.1484e-11	1
5	1	33	-1.5643e-01	-1.5643e-01	-1.5643e-01	-2.1854e-10	5.5067e-16	2.2996e-10	1
6	1	33	1.5643e-01	1.5643e-01	1.5643e-01	-2.1854e-10	5.5067e-16	2.2996e-10	1
7	3	33	4.5399e-01	4.5399e-01	4.5399e-01	-4.1272e-11	8.9947e-15	7.1484e-11	1
8	3	33	7.0711e-01	7.0711e-01	7.0711e-01	1.7807e-11	7.8604e-14	3.6423e-11	1
9	3	32	8.9101e-01	8.9101e-01	8.9101e-01	-8.6401e-12	3.2028e-13	3.7117e-11	1
10	1	30	9.8769e-01	9.8769e-01	9.8769e-01	-2.5741e-09	6.3407e-13	4.6150e-11	1

Figure 1: Results using myRoot with T_{10}

4.2 Discussion

For each root the residual is larger than the running error bound, we can then trust every root. When the running error bound is larger than the residual we say that the root can not be trusted anymore and we can not complete another bisection. The reason why it can't be trusted is because we are calculating the running error bound of the computed residual and this would lead to the root to be less reliable. In the Figure 1 we can see that the Trust column is all 1, trusted, for each root. The reason is because the computed residual is larger than the running error bound.

References

[1] CARL CHRISTIAN KJELGAARD MIKKELSEN. *An Introduction to Scientific Computing*. Department of Computing Science, Umeå University.