UMEÅ UNIVERSITY
Department of Computing Science
Assignment report, v.2

November 18, 2019

5DV005 - Project 1

Scientific Computing
Autumn 2018, 7.5 Credits

Robust and accurate root finding for real polynomials

Name Betty Törnkvist (`et16btt@cs.umu.se`)
Jonas Sjödin (`id16jsn@cs.umu.se`)
Emil Söderlind (`id15esd@cs.umu.se`)

Path `~et16btt/edu/tvb/5dv005ht18/`

Teacher
Carl Christian Kjelgaard Mikkelsen

Contents

1 Introduction

Finding and determining the roots of an equation is one of the oldest problems in mathematics. Every real polynomial P has either one or many roots, real numbers x_i, such that $P(x_i) = 0$. The Fundamental Theorem of Algebra tells us that a polynomial of degree n has n roots [1]. There are various ways of calculating these roots, and the methods have different strengths and weaknesses. When using computers to calculate these roots, errors, such as round-off, needs to be taken into consideration.

Our purpose with this project was to create a robust software for calculating both the roots and the errors that comes with it, prioritising accuracy before speed. We used a famous sequence of orthogonal polynomials, *Chebyshevs polynomials*, to build up test cases. Thereafter we created the root-finding software, using both Horners method and the bisection method. The software gave us an interval for the possible roots and the errors bounds, such as running error bound and a priori error bound. Lastly, we used our root-finding software to calculate the roots of the Chebyshev polynomials and discuss whether the results were reliable or not.

2 Chebyshev Polynomials

The Chebyshev polynomials are a series of orthogonal polynomials. They were first recognised by, and named after, Pafnuty Lvovich Chebyshev, in his publication *Théorie des mécanismes connus sous le nom de parallélogrammes* [2]. We use them to build up test cases since they are easily defined and easy to calculate by hand, so that we can run tests and without effort see if the results correlate with the computed results. The Chebyshev polynomials of the first kind are defined by the recurrence relation,

$$T_0(x) = 1,$$

$$T_1(x) = x,$$

$$T_{j+1} = 2xT_j(x) - T_{j-1}(x).$$

Given the definition above, the first six polynomials in the series can be expressed as

$$
\begin{aligned}
&T_0(x) = 1\\
&T_1(x) = x\\
&T_2(x) = 2x \times T_1 - T_0 = 2x^2 - 1\\
&T_3(x) = 2x \times T_2 - T_1 = 4x^3 - 3x\\
&T_4(x) = 2x \times T_3 - T_2 = 8x^4 - 8x^2 + 1\\
&T_5(x) = 2x \times T_4 - T_3 = 16x^5 - 20x^3 + 5x\\
&T_6(x) = 2x \times T_5 - T_4 = 32x^6 - 48x^4 + 18x^2 - 1
\end{aligned}
$$

2.1 Chebyshev Polynomial Calculation Software

Using Matlab, the program `MyChebyshev.m` was created to calculate any Chebyshev polynomial, see Appendix A. The program consists of the function `MyChebyshev(n,x)` that takes the arguments `n`, the number of polynomials, and `x`, a vector of length m containing the sample points. The function returns a matrix containing all the calculated polynomials, in order. See Figure 1. A minimal working example, `MyChebyshevMWE1`, was built to generate an illustration of the polynomials generated by the function, see Appendix B. A calculation of the six first polynomials in the series can be seen in Figure 2.

$$\text{y} = \begin{bmatrix} T_1(x_1) & T_2(x_1) & \cdots & T_n(x_1) \\ T_1(x_2) & T_2(x_2) & \cdots & T_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ T_1(x_m) & T_2(x_m) & \cdots & T_n(x_m) \end{bmatrix}$$

Figure 1: MyChebyshev.m returns an array of size $m \times n$.

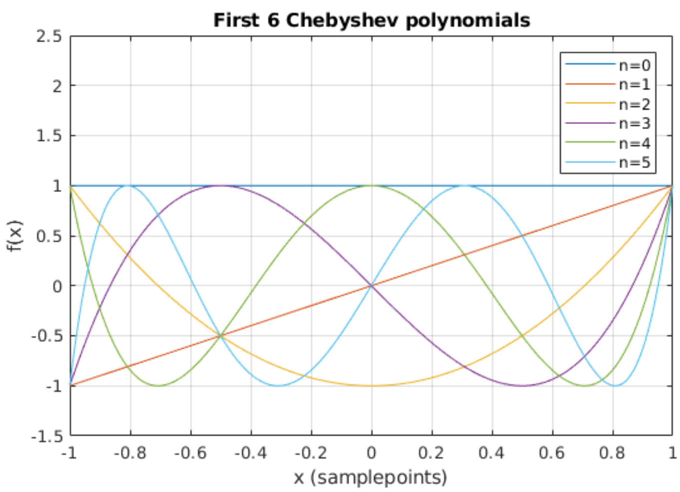

Figure 2: The first six Chebyshev Polynomials.

2.2 Induction Proof - T_n has Degree n

We start by defining the the function *deg* as below.

Definition 2.1. For a general real polynomial T,

$$T = \sum_{k=1}^{n} a_k x^k,$$

we let the function $deg(T)$ act as

$$deg(T) = n.$$

We now have the following theorem.

Theorem 2.1. Let T_n be a Chebyshev polynomial of the first kind, it then holds that

$$deg(T_n) = n, \tag{1}$$

i.e it gives us the degree of the polynomial.

Proof. By using induction and the definition of the Chebyshev polynomials, we want to prove equation (1). The axiom of induction is as follows,

$$\forall P(P(0) \land \forall k(P(k) \Rightarrow P(k+1)) \Rightarrow \forall n(P(n))).$$

We know that,

$$deg(A_n - Bm) = \max(deg(A_n), deg(B_m)), \tag{2}$$

if

$$deg(A_n) \neq deg(B_m).$$

Also, we have that

$$deg(A_n B_m) = deg(A_n) + deg(B_m). \tag{3}$$

With this information at hand we now perform the induction proof in three steps.

Step 1.
Show that equation (1) holds for T_0 and T_1. We know, per Chebyshev polynomial definition, that $t_0 = 1$ and $t_1 = x$. It follows that,

$$deg(T_0) = deg(1) = 0,$$

$$deg(T_1) = deg(x) = 1.$$

And so equation (1) is true for $n = 0$ and $n = 1$.

Step 2.
Assume equation (1) is true for case m and $m - 1$,

$$deg(T_m) = m,$$

$$deg(T_{m-1}) = m - 1.$$

Step 3.
Show that equation (1) is true for case $m + 1$. We get that

$$\begin{aligned} deg(T_{m+1}) &= deg(2xT_m - T_{m-1}) \\ &= max(deg(2xT_m), deg(T_{m-1})) \\ &= deg(2xT_m) = deg(2x) + deg(T_m) \\ &= 1 + deg(T_m) = 1 + m = m + 1 \end{aligned}$$

where the second equality above is due to equation (2) and the forth equality is due to equation (3).

Hence equation (1) holds for m+1, and so by Step 1,2 and 3 it is shown by induction to hold for all n. □

2.3 Proof - Chebyshev Roots

Theorem 2.2. The n roots of T_n are given by

$$x_k = \cos\left(\frac{(2k-1)\pi}{2n}\right), \quad k = 1, 2, 3, ..., n.$$

Proof. We now know that, according to Theorem 2.1,

$$deg(T_n) = n,$$

which implies, according to the Fundamental Theorem of Algebra [1], that T_n has n roots. We also know that the Chebyshev polynomials of the first kind satisfy the equation,

$$T_n(\cos\theta) = \cos(n\theta). \tag{4}$$

We start by looking for roots in the interval $x \in [-1, 1]$, which implies that

$$x = \cos\theta. \tag{5}$$

We then assume that x is a valid root of T_n, i.e.

$$0 = T_n(x) = T_n(\cos\theta) = \cos(n\theta),$$

where the second equality above is due to equation (4) and the third equality is due to Equation (5). From that it follows that

$$\cos(n\theta) = 0.$$

By the definition of cosine the equation above implies that

$$n\theta_k = \frac{(2k-1)\pi}{2}, \quad k = 1, 2, 3, ..., n$$

so,

$$\theta_k = \frac{(2k-1)\pi}{2n},$$

and the roots are

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

where we limit $k \leq n$, since we have at a maximum n roots, any $k > n$ would just give a previous root. We do not know that these correspond to unique roots though, but a quick look at the maximum angle, $\theta_{k=n}$, shows that

$$\theta_n = \frac{(2n-1)\pi}{2n} = \pi - \frac{\pi}{2n} \leq \pi < 2\pi,$$

which implies that

$$\theta_n < 2\pi,$$

and so all n roots are unique.

□

3 Root Finding Software

To be able to find a root of any real polynomial, we created a program `MyRoot.m` that uses both the bisection method and Horner's method. The software approximates a root within given interval $[\alpha, \beta]$, how many iterations needed to calculate, a flag signalling success or failure and a running error bound. A minimal working example was created to show how the program works, see Appendix D. The minimal working example finds all roots for the Chebyshev polynomial of degree 10, and compares the result by calculating the roots with the method in Theorem 2.2, the result can be seen in Figure **??**. The minimal working example also presents the relative error for each root, and the result is that the relative error is larger than 10^-13 in two cases, at the roots 5 and 6, the ones closest to 0.

3.1 Horner's Method

The Horner's Method is a technique for evaluating polynomials. The method require a given polynomials coefficients

$$C_P = a_1, a_2, ..., a_n$$

and sample points

$$x = s_0, s_1, s_2, \dots s_k$$

for some possible integer k, of which one is interested in calculating the corresponding value of.

The idea behind Horner's method is to repeatedly divide the polynomial into monomials of degree 1. Each monomial is then involved in a maximum of one multiplication or addition process each. By doing this repeatedly and collecting the result from each monomial evaluation the full polynomial evaluation can be performed. See Appendix E for our implementation of Horner's method.

3.2 Bisection Method

If the function f is continuous, and we know that the brackets $f(a)$ and $f(b)$ have opposite signs, then we also know that at least one root must lie in that interval, according to Bolzano's intermediate value theorem [3]. The bisection part of this function sets two brackets and halves the interval until it can find a root that is good enough. Such a root is defined by one of the following conditions:

- A predefined max iteration limit has been reached.
- The last bracket is smaller than the given delta, δ, i.e. $\delta > |\beta - \alpha|$.
- The last function value is bounded by epsilon (the error tolerance), ϵ, i.e. $\epsilon > |\frac{\beta-\alpha}{2}|$
- The sign of the root can not be trusted, i.e. $reb > |f(|\beta - \alpha|)|$ where reb is the running error bound.

By checking these values we know that the brackets are maintained and that the value of $\hat{y}$ has the same sign as y. We know that since we check the sign in each iteration of the bisection algorithm and also break if the running error bound is higher than the absolute value of y, since it otherwise creates a risk that the real value for $\hat{y}$ has changed sign, see Figure 3. This explains why it is crucial to maintain the brackets around a computed approximation.

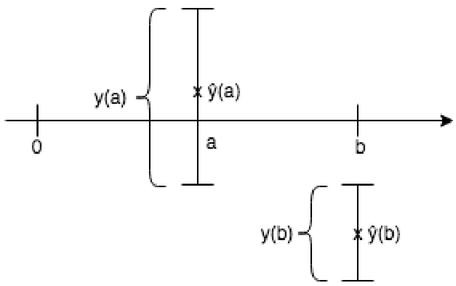

Figure 3: The range of the error bound shows that the value of (a) might have changed sign.

c

4 Discussion

Since we are using the bisection method with the help of a computer a floating point error will occur. This results in a possible wrong bracket, meaning that the root might not be within our bracket. To avoid this we only search for the root as long as our error is smaller than the difference between our brackets, see Subsection 3.2. By this we can conclude that our calculated roots can be trusted.

When looking at all roots of the Chebyshev polynomial of degree 10 we can conclude that our function `MyRoot.m` has 4 roots where $reb < 10^{-13}$.

During the project we realised how computers more often than not get an inaccurate value while evaluating arithmetical expressions. The answer the computer gives you is inaccurate, the question is how inaccurate the answer is. This is something that needs to be taken into account and clearly stated when presenting the result.

References

[1] A. Girard, *Invention nouvelle en l'algèbre.* Imprimé chez Muré frères, 1884.

[2] P. L. Tchebychev, *Théorie des mécanismes connus sous le nom de parallélogrammes.* Imprimerie de l'Académie impériale des sciences, 1853.

[3] B. Bolzano, *Rein analytischer Beweis des Lehrsatzes, das zwischen je zwey Werthen, die ein entgegengesetzes Resultat gewähren, wenigstens eine reelle Wurzel der Gleichung liege.* gedruckt bei Gottlieb Haase, 1817.

A MyChebyshev.m

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

% MyChebyshev Evaluates the first n Chebyshev polynomials
%
% CALL SEQUENCE: y=MyChebyshev(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.m

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen
%   2018-11-14 Skeleton extracted from working code
%   2018-11-25 Skeleton extracted and functonality added by
%                         Betty T rnkvist et16btt@cs.umu.se
%                         Jonas Sj din id16jsn@cs.umu.se
%                         Emil S derlind id15esd@cs.umu.se

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

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

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

% Initialize the first two columns of y
y(:,1:2)=[ones(m,1) x];

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

B MyChebyshevMWE1.m

```
% MyChebychevMWE1.m Minimal working example for MyChebyshev

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen
%   2018-11-14 Skeleton extracted from working code
%   2018-11-25 Skeleton extracted and functonality added by
%                          Betty T rnkvist et16btt@cs.umu.se
%                          Jonas Sj din id16jsn@cs.umu.se
%                          Emil S derlind id15esd@cs.umu.se
% Set number of polynomials
n = 6;
% Set number of sample points
x_size = 101;

% Define sample points
x = -1: 1/x_size:1;
% Generate function values
y = MyChebyshev(n, x);
% Plot all graphs with one command
plot(x, y)
% Adjust axis to make room for legend
axis([-1 1 -1.5 2.5]);
grid on;
% Set labels
xlabel('x␣(samplepoints)');
ylabel('f(x)');
title('First␣6␣Chebyshev␣polynomials');
% Construct and display legend
str=[];
for i=0:n-1
    str=[str strcat("n=",string(i))];
end
legend(str);
```

C 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
%
% INPUT:
%   p         array of coefficients used by my_horner
%   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 bada
%               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: Missing

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen (spock@cs.umu.se)
%   2018-11-14 Skeleton extracted from working code MyRoot
%   2018-11-25 Skeleton extracted and functonality added by Betty T rnkvist ,
%               Jonas Sj din and Emil S derlind

% Initialize the flag.
flag=0;

% Dummy initialization of *all* output arguments
x=NaN; it=0;
a=zeros(maxit,1);

b=zeros(maxit,1);
his=zeros(maxit,1);
y=zeros(maxit,1);
reb = zeros(maxit, 1);

% Initialize search bracket (alpha,beta) such that alpha <= beta
alpha = a0;
beta = b0;

% Compute fa=p(alpha) and fb= p(beta) and associated error bounds

[fa, ~, rebfa]=my_horner(p,alpha);
[fb, ~, rebfb]=my_horner(p,beta);

% Investigate if the flag should be -2 or -1
if sign(fa) == sign(fb)
    flag = -2;
end
if rebfa > abs(fa) || rebfb > abs(fb)
    flag = -1;
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]=my_horner(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 = 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 rebfc > abs(fc)
        flag = flag+4;
    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
% 1.000000005 ~ 1
a = a(1:j);
b = b(1:j);
his = his(1:j);
y = y(1:j);
reb = reb(1:j);

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

D MyRootMWE1.m

```
% A2F2 Minimal working example for MyChebyshev

% PROGRAMMING by Carl Christian Kjelgaard Mikkelsen
%   2018-11-14 Skeleton extracted from working code
%   2018-11-25 Skeleton extracted and functonality
%   added by Betty T rnkvist , Jonas Sj din and Emil S derlind

% Set Chebyshev degree
n = 10;

% Define sample points
it = zeros(n, 1);
flag = zeros(n, 1);
a = zeros(n, 1);
b = zeros(n, 1);
his = zeros(n, 1);
root = zeros(n, 1);
res = zeros(n, 1);
reb = zeros(n, 1);
x = zeros(n, 1);

% Chebyshev polynomial of degree 10
y = [-1, 0, 50, 0, -400, 0, 1120, 0, -1280, 0, 512];

m = 101;
lin = linspace(-1, 1, m);

inx = 1;
maxit = 100000;
delta = 2 * 10^-13;
epsilon = 10^-13;

for i=1:m-1
    % Calculate the x:th root of Chebyshev polynomial of degree n

    [retroot, retflag, retit, reta, retb, rethis, rety, retreb] =
    MyRoot(y, lin(i), lin(i+1), delta, epsilon, maxit);

    if isnan(retroot)
        continue
    end

    root(inx) = retroot;
    flag(inx) = retflag;
    it(inx) = retit;

    % Save values for printing later on
    a(inx) = reta(retit);
    b(inx) = retb(retit);
    his(inx) = rethis(retit);
    res(inx) = rety(retit);
    reb(inx) = retreb(retit);
    inx = inx + 1;
end

% Set if root can be trusted
trust = reb < res;

% Assign data that should be printed
data = [transpose(1:n), flag, it, a, b, root, res, reb, trust];
% Set headers for each column
colheaders = {'idx', 'flag', 'it', 'a', 'b', 'root', 'residual', 'reb', 'trust'};
% Set width of each column
width = [6, 4, 3, 12, 12, 12, 12, 12, 5];
% Format numbers
fms={'d','d','d','.5e','.5e', '.5e', '.5e', '.5e', 'd'};

displaytable(data, colheaders, width, fms);
```

```
T = flip(cos((2*(1:10) - 1) .* pi/(2*n)));
O = transpose(root);
R = abs((T - O) ./ T);

find(R > 10^-13)
```

E my_horner.m

```
function [y,aeb,reb]=my_horner(a,x)

% MY_HORNER   An implementation of Horner's method
%
% CALL SEQUENCE:
%
% [y]=my_horner(a,x)
%
% INPUT:
%  a      array of cofficients determining p
%  x      array of arguments to pass to p
%
% OUTPUT:
%  y      the computed value of the polynomial
%
% MINIMAL RUNNING EXAMPLE: my_horner

% Isolate the number of coefficients
m=numel(a);

% Isolate the degree of the polynomial
n=m-1;

% Both a and x must be in double precision or MATLAB works
% in single
if (strcmp(class(a),'double') && strcmp(class(x),'double'))
    % Set u to double precision unit roundoff
    u=2^-53;
else
    % Set u to single precision unit round off
    u=2^-24;
end

% Reshape the coefficient array as a row vector
aux=reshape(a,1,m);

% Determine the size of the input array x
sx=size(x);

% Initialize the output arrays
y=ones(sx)*aux(m);
pt=ones(sx)*abs(aux(m));

% Initialize running error bound
mu = zeros(sx);

% Main loop.
for j=1:n
    % Compute intermediate value
    z=y.*x;
    % Update polynomial p
    y=z+aux(m-j);
    % Update running error bound
    mu = mu.*abs(x) + abs(z) + abs(y);
    % Update polynomial pt
    pt=pt.*abs(x)+abs(aux(m-j));
end


% Compute the relvant gamma factor
gamma=(2*n*u)/(1-2*n*u);

% Compute the apriori error bound
aeb = gamma.*pt;

% Compute the running error bound
reb = mu.*u;
```