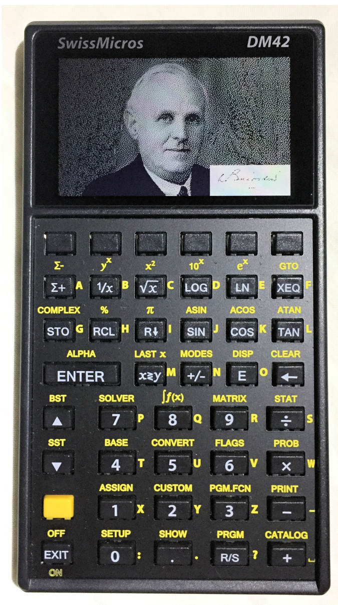
Lin-Bairstow polynomial roots finder algorithm for DM42

Enrico Bellincioni

Abstract

In mathematics, and especially in the applied sciences, it often happens that you have to find the roots of polynomials even high degree. In 1920 Professor Leonard Bairstow published the algorithm in the appendix of his book: *Applied Aerodynamics*. The great idea conceived by Bairstow was that of an algorithm that was able to determine two roots at a time instead of just one (like all the other algorithms invented before), in this way the calculations used could remain in real arithmetic, also considering the hardware resources available in those years ... none.



... the problem

Professor Leonard Bairstow wanting to solve example of calculation of the stability of an aeroplane when turning during horizontal flight had to solve the following polynomial equation[1]:

$$\lambda^8 + 20.4\lambda^7 + 151.3\lambda^6 + 490\lambda^5 + 687\lambda^4 + 719\lambda^3 + 150\lambda^2 + 109\lambda + 6.87 \;=\; 0$$

Advantages

The algorithm turns out to be very simple to implement, in fact it requires the repeated resolution of two recursive formulas and uses only real arithmetic for the calculations.

Disadvantages

The convergence order of the algorithm is 2 for distinct roots and drops to 1 for roots of multiplicity higher then 1. The algorithm may also not converge for this reason I have inserted a maximum limit of iterations equal to 80.

Derivation of the algorithm

The complete derivation of the algorithm requires several mathematical passages. To facilitate the reader to full understanding I preferred to divide into several sections:

1. Calculation of b_k, r, s
2. Linearization of the system
3. Calculation of r_p, s_p
4. Solution of linear system
5. Final steps and Algorithm

[1]For the solution obtained by Professor Leonard Bairstow (1920) compared with the DM42, see the final part of the article where some examples are presented.

1 Calculation of b_k, r, s

$$P_n(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + \alpha_2 x^{n-2} + \alpha_3 x^{n-3} + \cdots + \alpha_{n-2} x^2 + \alpha_{n-1} x + \alpha_n \quad (1)$$

Equation (1) represents the polynomial whose roots we want to determine (all at them). All coefficients of the polynomial are real numbers, this fact limits the typology of solutions to real numbers or complex conjugate pairs. In order to simplify the subsequent calculations, for the polynomial (1), lets suppose $\alpha_0 = 1$. If it is not already so, it is easy to divide all the coefficients by α_0.

$$P_n(x) = x^n + a_1 x^{n-1} + a_2 x^{n-2} + a_3 x^{n-3} + \cdots + a_{n-2} x^2 + a_{n-1} x + a_n \quad (2)$$

Where

$$a_1 = \frac{\alpha_1}{\alpha_0} \qquad a_2 = \frac{\alpha_2}{\alpha_0} \qquad \cdots \qquad a_n = \frac{\alpha_n}{\alpha_0}$$

Wanting to extract two roots from (2) we can divide $P_n(x)$ by quadratic $x^2 + px + q$ obtaining

$$P_n(x) = Q_{n-2}(x)(x^2 + px + q) + \underbrace{rx + s}_{reminder} \quad (3)$$

where

$$Q_{n-2}(x) = x^{n-2} + b_1 x^{n-3} + b_2 x^{n-4} + b_3 x^{n-5} + \cdots + b_{n-4} x^2 + b_{n-3} x + b_{n-2} \quad (4)$$

represents the reduced polynomial after extracting the quadratic factor $x^2 + px + q$ from $P_n(x)$ and coefficients r,s depend on p,q. If we want to extract two roots of $P_n(x)$ the quadratic $x^2 + px + q$ should divide $P_n(x)$ without reminder.

Substituting (2) and (4) in (3) and expanding

$$\begin{aligned}
&x^n + a_1 x^{n-1} + a_2 x^{n-2} + a_3 x^{n-3} + \cdots + a_{n-2} x^2 + a_{n-1} x + a_n = \\
&= (x^{n-2} + b_1 x^{n-3} + b_2 x^{n-4} + b_3 x^{n-5} + \cdots + b_{n-4} x^2 + b_{n-3} x + b_{n-2}) * \\
&\qquad * (x^2 + px + q) + rx + s = \\
&= x^n + px^{n-1} + qx^{n-2} + b_1 x^{n-1} + pb_1 x^{n-2} + qb_1 x^{n-3} + b_2 x^{n-2} + pb_2 x^{n-3} + qb_2 x^{n-4} + \cdots \\
&\cdots + b_{n-4} x^4 + pb_{n-4} x^3 + qb_{n-4} x^2 + b_{n-3} x^3 + pb_{n-3} x^2 + qb_{n-3} x + b_{n-2} x^2 + pb_{n-2} x + \cdots \\
&\qquad \cdots + qb_{n-2} + rx + s
\end{aligned}$$

then by comparing the polynomial coefficients we get

$$\begin{cases} a_1 = p + b_1 \\ a_2 = q + pb_1 + b_2 \\ a_3 = qb_1 + pb_2 + b_3 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ a_{n-3} = qb_{n-5} + pb_{n-4} + b_{n-3} \\ a_{n-2} = qb_{n-4} + pb_{n-3} + b_{n-2} \\ a_{n-1} = qb_{n-3} + pb_{n-2} + r \\ a_n = qb_{n-2} + s \end{cases} \tag{5}$$

from which

$$\begin{cases} b_1 = a_1 - p \\ b_2 = a_2 - pb_1 - q \\ b_3 = a_3 - pb_2 - qb_1 \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ b_{n-3} = a_{n-3} - pb_{n-4} - qb_{n-5} \\ b_{n-2} = a_{n-2} - pb_{n-3} - qb_{n-4} \\ r = a_{n-1} - pb_{n-2} - qb_{n-3} \\ s = a_n - qb_{n-2} \end{cases} \tag{6}$$

$$
\begin{cases}
b_1 = a_1 - p1 - q0 \\
b_2 = a_2 - pb_1 - q1 \\
b_3 = a_3 - pb_2 - qb_1 \\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\
b_{n-3} = a_{n-3} - pb_{n-4} - qb_{n-5} \\
b_{n-2} = a_{n-2} - pb_{n-3} - qb_{n-4} \\
r = \underbrace{a_{n-1} - pb_{n-2} - qb_{n-3}}_{b_{n-1}} \\
s - pb_{n-1} = \underbrace{a_n - pb_{n-1} - qb_{n-2}}_{b_n}
\end{cases}
\tag{7}
$$

$$
\begin{cases}
b_{-1} = 0 \\
b_0 = 1 \\
b_k = a_k - pb_{k-1} - qb_{k-2} \qquad k = 1, 2, ..., n
\end{cases}
\tag{8}
$$

$$
\begin{cases}
r \equiv b_{n-1} \\
s \equiv b_n + pb_{n-1}
\end{cases}
\tag{9}
$$

Equations (8) and (9) allow to automatically determine the coefficients b_k of the reduced polynomial and the remainder coefficients r, s, the a_k, p, q being known.

2 Linearization of the system

Our goal is to get zero reminder in (3), therefore

$$\begin{cases} r(p,q) = 0 \\ s(p,q) = 0 \end{cases} \tag{10}$$

The system (10) is non-linear, therefore the algorithm will solve it numerically using linear approximation of $r(p,q)$ and $s(p,q)$. Lets denote linear approximation of $r(p,q)$ and $s(p,q)$ as $\mathcal{L}r(p,q)$ and $\mathcal{L}s(p,q)$ respectively. Then linearized system (10) is

$$\begin{cases} \mathcal{L}r(p,q) = 0 \\ \mathcal{L}s(p,q) = 0 \end{cases} \tag{11}$$

The algorithm starts with initial values (p_0, q_0), solves the system (11) for linearization in (p_0, q_0) getting (p_1, q_1) as a solution. Then the pair (p_1, q_1) is used as a new p, q approximation and system is solved again, this time for (p_1, q_1). Repeating this process we get sequence $(p_0, q_0), (p_1, q_1)$, ..., $(p_k, q_k), (p_{k+1}, q_{k+1})$ and we can define

$$\begin{cases} D_p(k) = p_{k+1} - p_k \\ D_q(k) = q_{k+1} - q_k \end{cases} \tag{12}$$

thus

$$\begin{cases} p_{k+1} = p_k + D_p(k) \\ q_{k+1} = q_k + D_q(k) \end{cases} \tag{13}$$

Linearization $\mathcal{L}r(p,q)$, $\mathcal{L}s(p,q)$ in (p_k, q_k) can be expressed using first order Taylor series expansion in (p_k, q_k) as

$$\begin{cases} \mathcal{L}r(p,q) = r(p_k, q_k) + \dfrac{\partial r(p_k, q_k)}{\partial p}(p - p_k) + \dfrac{\partial r(p_k, q_k)}{\partial q}(q - q_k) \\ \\ \mathcal{L}s(p,q) = s(p_k, q_k) + \dfrac{\partial s(p_k, q_k)}{\partial p}(p - p_k) + \dfrac{\partial s(p_k, q_k)}{\partial q}(q - q_k) \end{cases} \tag{14}$$

Substituting (p_{k+1}, q_{k+1}) for (p, q) in (14) we get

$$\begin{cases} \mathcal{L}r(p_{k+1}, q_{k+1}) = r(p_k, q_k) + \dfrac{\partial r(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial r(p_k, q_k)}{\partial q}(q_{k+1} - q_k) \\ \\ \mathcal{L}s(p_{k+1}, q_{k+1}) = s(p_k, q_k) + \dfrac{\partial s(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial s(p_k, q_k)}{\partial q}(q_{k+1} - q_k) \end{cases} \tag{15}$$

We use linearization in (p_k, q_k), therefore the system (11) is solved for (p_{k+1}, q_{k+1}) (which follows from numerical algorithm described earlier), thus left sides are equal to zero

$$\begin{cases} 0 = r(p_k, q_k) + \dfrac{\partial r(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial r(p_k, q_k)}{\partial q}(q_{k+1} - q_k) \\ \\ 0 = s(p_k, q_k) + \dfrac{\partial s(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial s(p_k, q_k)}{\partial q}(q_{k+1} - q_k) \end{cases} \tag{16}$$

$$\begin{cases} \dfrac{\partial r(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial r(p_k, q_k)}{\partial q}(q_{k+1} - q_k) = -r(p_k, q_k) \\ \\ \dfrac{\partial s(p_k, q_k)}{\partial p}(p_{k+1} - p_k) + \dfrac{\partial s(p_k, q_k)}{\partial q}(q_{k+1} - q_k) = -s(p_k, q_k) \end{cases} \tag{17}$$

to simplify the reading of the system (17) making the calculations even more compact and clear it is better to write the four partial derivatives

$$
\begin{cases}
r_p = \dfrac{\partial r(p_k, q_k)}{\partial p} \\[2ex]
r_q = \dfrac{\partial r(p_k, q_k)}{\partial q} \\[2ex]
s_p = \dfrac{\partial s(p_k, q_k)}{\partial p} \\[2ex]
s_q = \dfrac{\partial s(p_k, q_k)}{\partial q}
\end{cases}
\tag{18}
$$

using this and the equations (9) and (12) the system (17) can be

$$
\begin{cases}
r_p \cdot D_p(k) \ + \ r_q \cdot D_q(k) \ = - \ b_{n-1} \\[2ex]
s_p \cdot D_p(k) \ + \ s_q \cdot D_q(k) \ = - \ b_n - p_k b_{n-1}
\end{cases}
\tag{19}
$$

We now begin to calculate two of the four partial derivatives, in particular r_q and s_q, differentiate the system (20) with respect to the variable q

$$
\begin{cases}
r = b_{n-1} \\
s = b_n + p_k b_{n-1}
\end{cases}
\tag{20}
$$

$$
\begin{cases}
r_q = \dfrac{\partial r(p_k, q_k)}{\partial q} = \dfrac{\partial b_{n-1}(p_k, q_k)}{\partial q} \\[2ex]
s_q = \dfrac{\partial s(p_k, q_k)}{\partial q} = \dfrac{\partial b_n(p_k, q_k)}{\partial q} + p_k \cdot \dfrac{\partial b_{n-1}(p_k, q_k)}{\partial q}
\end{cases}
\tag{21}
$$

and remembering that

$$\begin{cases} b_{-1} = 0 \\ b_0 = 1 \\ b_k = a_k - p_k b_{k-1} - q b_{k-2} \qquad k = 1, 2, ..., n \end{cases} \tag{22}$$

from which

$$\begin{cases} \dfrac{\partial b_{-1}(p_k, q_k)}{\partial q} = 0 \\ \\ \dfrac{\partial b_0(p_k, q_k)}{\partial q} = 0 \\ \\ \dfrac{\partial b_k(p_k, q_k)}{\partial q} = \underbrace{\dfrac{\partial a_k}{\partial q}}_{0} - p_k \cdot \dfrac{\partial b_{k-1}(p_k, q_k)}{\partial q} - q \cdot \dfrac{\partial b_{k-2}(p_k, q_k)}{\partial q} - b_{k-2} \qquad k = 1, 2, ..., n \end{cases} \tag{23}$$

by defining a new coefficient c_k

$$c_k = \; - \; \frac{\partial b_k(p_k, q_k)}{\partial q} \tag{24}$$

the system (23) can be rewritten

$$\begin{cases} c_{-1} = 0 \\ c_0 = 0 \\ c_k = b_{k-2} - p_k c_{k-1} - q_k c_{k-2} \qquad k = 1, 2, ..., n \end{cases} \tag{25}$$

$$\begin{cases} r_q = \dfrac{\partial r(p_k, q_k)}{\partial q} = \dfrac{\partial b_{n-1}(p_k, q_k)}{\partial q} = -\, c_{n-1} \\ \\ s_q = \dfrac{\partial s(p_k, q_k)}{\partial q} = \dfrac{\partial b_n(p_k, q_k)}{\partial q} + p_k \cdot \dfrac{\partial b_{n-1}(p_k, q_k)}{\partial q} = -\, c_n \,-\, p_k c_{n-1} \end{cases} \tag{26}$$

3 Calculation of r_p, s_p

The purpose of this section is to calculate the remaining two partial derivatives r_p, s_p and show how the four partial derivatives are related with each other. For convenience, rewritten the equation (3)

$$P_n(x) = Q_m(x)(x^2 + px + q) + rx + s \tag{27}$$

lets denote the polynomial $Q_{n-2}(x)$ as $Q_m(x)$ where (obviously) $m = n-2$, suppose we know the solutions of the quadratic factor

$$x^2 + px + q = 0 \tag{28}$$

and that these solutions are x_- and x_+.

Partially differentiate (27) with respect to the variables p and q we get

$$\begin{cases} \underbrace{\dfrac{\partial P_n(x)}{\partial p}}_{0} = \dfrac{\partial Q_m(x)}{\partial p} \cdot (x^2 + p_k x + q_k) + Q_m(x) \cdot x + x \cdot \dfrac{\partial r(p_k, q_k)}{\partial p} + \dfrac{\partial s(p_k, q_k)}{\partial p} \\ \\ \underbrace{\dfrac{\partial P_n(x)}{\partial q}}_{0} = \dfrac{\partial Q_m(x)}{\partial q} \cdot (x^2 + p_k x + q_k) + Q_m(x) \cdot 1 + x \cdot \dfrac{\partial r(p_k, q_k)}{\partial q} + \dfrac{\partial s(p_k, q_k)}{\partial q} \end{cases} \tag{29}$$

both partial derivatives found on the first member of the system are both zero because the polynomial $P_n(x)$, in particular the coefficients a_k do not depend on p, q. The system (29) can be evaluated for any value of the variable x, but if we calculate it for the values x_- and x_+ it can be simplified considerably. Let's start with $x = x_-$ using the (18)

$$\begin{cases} x_- \cdot r_p \,+\, s_p \;=\; -Q_m(x_-) \cdot x_- \\ x_- \cdot r_q \,+\, s_q \;=\; -Q_m(x_-) \end{cases} \tag{30}$$

multiplying the second equation of (30) by x_-

$$\begin{cases} x_- \cdot r_p \,+\, s_p \;=\; -Q_m(x_-) \cdot x_- \\ {x_-}^2 \cdot r_q \,+\, x_- \cdot s_q \;=\; -Q_m(x_-) \cdot x_- \end{cases} \tag{31}$$

from which

$$x_- \cdot r_p \,+\, s_p \;=\; x_-^2 \cdot r_q \,+\, x_- \cdot s_q \tag{32}$$

similarly for $x = x_+$ we get the system (33) from which the functional dependence of the four partial derivatives is evident, a linear system of two variabiles r_p,s_p that we can solve with the Cramer rule

$$\begin{cases} x_- \cdot r_p \,+\, s_p \;=\; x_-^2 \cdot r_q \,+\, x_- \cdot s_q \\ x_+ \cdot r_p \,+\, s_p \;=\; x_+^2 \cdot r_q \,+\, x_+ \cdot s_q \end{cases} \tag{33}$$

$$\Delta \;=\; \begin{vmatrix} x_- & 1 \\ x_+ & 1 \end{vmatrix} \;=\; (x_- \,-\, x_+) \tag{34}$$

$$\Delta_{r_p} \;=\; \begin{vmatrix} x_-^2 r_q + x_- s_q & 1 \\ x_+^2 r_q + x_+ s_q & 1 \end{vmatrix} \;=\; (x_-^2 - x_+^2) \cdot r_q \,+\, (x_- - x_+) \cdot s_q \tag{35}$$

$$\Delta_{s_p} \;=\; \begin{vmatrix} x_- & x_-^2 r_q + x_- s_q \\ x_+ & x_+^2 r_q + x_+ s_q \end{vmatrix} \;=\; -\, x_- \cdot x_+ \cdot (x_- - x_+) \cdot r_q \tag{36}$$

from which

$$r_p \;=\; \frac{\Delta_{r_p}}{\Delta} \;=\; (x_- + x_+) \cdot r_q \,+\, s_q \tag{37}$$

$$s_p \;=\; \frac{\Delta_{s_p}}{\Delta} \;=\; -\, x_- \cdot x_+ \cdot r_q \tag{38}$$

remembering now that x_- and x_+ are both solutions of (28), we can rewrite the equation in this way

$$x^2 + px + q \;=\; (x - x_-)(x - x_+) \;=\; x^2 \;-\; (x_- + x_+) \cdot x \;+\; x_- \cdot x_+$$

where it is evident that

$$p = -(x_- + x_+) \qquad and \qquad q = x_- \cdot x_+ \tag{39}$$

equations (37) and (38) using (39) become

$$r_p \;=\; s_q \;-\; p_k \cdot r_q \tag{40}$$

$$s_p \;=\; -\, q_k \cdot r_q \tag{41}$$

remembering the (26) the four partial derivatives can be thus written

$$\begin{cases} r_q \;=\; -\, c_{n-1} \\ s_q \;=\; -\, c_n \;-\; p_k \cdot c_{n-1} \\ r_p \;=\; s_q \;-\; p_k \cdot r_q \;=\; (-c_n - p_k c_{n-1}) - p_k \cdot (-c_{n-1}) \;=\; -\, c_n \\ s_p \;=\; -\, q_k \cdot r_q \;=\; -q_k \cdot (-c_{n-1}) \;=\; q_k \cdot c_{n-1} \end{cases} \tag{42}$$

4 Solution of linear system

substituting the expressions just calculated in the system (19) we obtain

$$\begin{cases} -c_n \cdot D_p(k) \ - \ c_{n-1} \cdot D_q(k) \ = \ - \ b_{n-1} \\ q_k c_{n-1} \cdot D_p(k) \ - \ (c_n + p_k c_{n-1}) \cdot D_q(k) \ = \ - \ b_n - p_k b_{n-1} \end{cases} \tag{43}$$

$$\begin{cases} c_n \cdot D_p(k) \ + \ c_{n-1} \cdot D_q(k) \ = \ b_{n-1} \\ - \ q_k c_{n-1} \cdot D_p(k) \ + \ (c_n + p_k c_{n-1}) \cdot D_q(k) \ = \ b_n + p_k b_{n-1} \end{cases} \tag{44}$$

subtracting from the second equation of (44) the first equation multiplied by p_k

$$\begin{cases} c_n \cdot D_p(k) \ + \ c_{n-1} \cdot D_q(k) \ = \ b_{n-1} \\ - \ (p_k c_n + q_k c_{n-1}) \cdot D_p(k) \ + \ c_n \cdot D_q(k) \ = \ b_n \end{cases} \tag{45}$$

the system (45) can be solved again with Cramer

$$D \ = \begin{vmatrix} c_n & c_{n-1} \\ -(p_k c_n + q_k c_{n-1}) & c_n \end{vmatrix} = \ c_n^2 + c_{n-1}(p_k c_n + q_k c_{n-1}) \tag{46}$$

$$\Delta_{D_p} \ = \begin{vmatrix} b_{n-1} & c_{n-1} \\ b_n & c_n \end{vmatrix} = \ b_{n-1} c_n \ - \ b_n c_{n-1} \tag{47}$$

$$\Delta_{D_q} \ = \begin{vmatrix} c_n & b_{n-1} \\ -(p_k c_n + q_k c_{n-1}) & b_n \end{vmatrix} = \ b_n c_n \ + \ b_{n-1}(p_k c_n + q_k c_{n-1}) \tag{48}$$

from which

$$D_p = \frac{\Delta_{D_p}}{D} = \frac{b_{n-1}c_n - b_n c_{n-1}}{c_n^2 + c_{n-1}(p_k c_n + q_k c_{n-1})} \tag{49}$$

$$D_q = \frac{\Delta_{D_q}}{D} = \frac{b_n c_n + b_{n-1}(p_k c_n + q_k c_{n-1})}{c_n^2 + c_{n-1}(p_k c_n + q_k c_{n-1})} \tag{50}$$

5 Final steps and Algorithm

The core of the algorithm was described at the beginning of section 2. Two things are still necessary and must be highlighted in order to solve the linear system (19), in particular:

- Initial values
- Terminating conditions

Initial values

- The initial values of p_0, q_0, if they are not known, can both be taken as one this choice from the tests carried out allows the convergence of the algorithm even in the presence of coincident roots and/or of some coefficients of the null polynomial.
- The initial value of the *error* which must necessarily be greater than the desired accuracy or tolerance *toll* (for example $error = 1$).
- The value of the iteration counter L = 0 (no iteration has yet been done)

Terminating conditions

Core termination conditions can be:

- The *error* reached is less than or equal to the desired accuracy *toll*
- The number of iterations L has exceeded the maximum value (set for example in 80). If this happens it means that the algorithm is not converging.

For each iteration we can update the error with the following formula:

$$error = max(|D_p|, |D_q|) \tag{51}$$

Algorithm

1. assigned n, α_k, *toll*
2. check that α_0 is equal to 1 if different divide all the coefficients α_k with the value of α_0
3. if $n > 2$ and fixed $p_0 = q_0 = 1$, $error = 1$ and L=0 are calculated b_k, c_k, D_p, D_q, updates $error$, $p_{k+1} = p_k + Dp$, $q_{k+1} = q_k + Dq$ until $error \leq toll$ or L > 80, show roots of quadratic factor $x^2 + px + q$ or exit with message error (if L > 80)
4. $n = n - 2$, replace a_k with b_k return to step 3
5. if $n = 2$ or $n = 1$ calculate the polynomial root(s)

Convergence of the algorithm

I have limited the maximum number of iterations of the algorithm L to 80 to understand if the algorithm is able to converge.

MATLAB® Bairstow Code

Before going into the technical details of drafting the algorithm code for DM42, I report the source code of the Bairstow algorithm that I made years ago in MATLAB® by which I was inspired for the recoding for the DM42

```
function rad = bairstow(a,toll,L)
if nargin == 1
    toll = 1e-6;
    L = 80;
elseif nargin == 2
    L = 80;
end
n = length(a) -1;
rad = [];

while n > 2
    p = 1;
    q = 1;
    [p,q,b,iter,error] = bairstkernel(a,p,q,toll,L);
    x1 = -0.5*(p+sqrt(p*p-4*q));
    x2 = -p-x1;
    rad = [rad x1 x2];
    a = b(2:n);
    n = n-2;
    disp(iter)
    disp(error)
end

if n == 2
    x1 = -0.5*(a(2)+sqrt(a(2)*a(2)-4*a(3)));
    x2 = -a(2)-x1;
    rad = [rad x1 x2];
elseif n == 1
    x1 = -a(2);
    rad = [rad x1];
end

rad = rad';

return
```

```
function [p,q,b,iter,error] = bairstkernel(a,p,q,toll,L)
if a(1) ≠ 1
    a = a/a(1);
end

n = length(a)-1;

a = a(2:n+1);

error = 1;
iter = 0;

while (error > toll)&&(iter ≤ L)
        b(1) = 0;
        b(2) = 1;
        for k = 1:n
            b(k+2) = a(k)-p*b(k+1)-q*b(k);
        end
        c(1) = 0;
        c(2) = 0;
        for k = 1:n
            c(k+2) = b(k)-p*c(k+1)-q*c(k);
        end

        D = c(n+2)*c(n+2)+c(n+1)*(p*c(n+2)+q*c(n+1));
        Dp = (b(n+1)*c(n+2)-b(n+2)*c(n+1))/D;
        Dq = (b(n+2)*c(n+2)+b(n+1)*(p*c(n+2)+q*c(n+1)))/D;

        error = max(abs(Dp),abs(Dq));

        p = p + Dp;
        q = q + Dq;
        iter = iter +1;
end

if (iter>L)
    disp('ATTENTION algorithm don''t converge')
    return
end
return
```

DM42 Resources Registers Used

REGISTER/S	SCOPE
R00 – R22	Polynomial coefficients $P_n(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + \cdots \alpha_n$
R30 – R52	Polynomial reduction and remainder $Q_{n-2}(x) = x^{n-2} + b_1 x^{n-3} + \cdots b_n$
R60 – R82	Coefficients c_k
R84	b_{n+1}
R85	b_{n+2}
R86	D_q
R87	D_p
R88	D
R89	k for loop index
R90	Maximum degree of polynomial $n \leq 22$
R91	Cycle indices and/or pointers
R92	Cycle indices and/or pointers or c_{n+1}
R93	Cycle indices and/or pointers or c_{n+2}
R94	Cycle indices and/or pointers
R95	p_k value on departure $p_0 = 0$
R96	q_k value on departure $q_0 = 0$
R97	error $= max(\lvert Dp \rvert, \lvert Dq \rvert)$ on departure error $= 1.0$
R98	Number of iterations m
R99	Desired tolerance / accuracy $toll \gg 1e - 33$

DM42 Resources Main Subroutines Used

NAME	SCOPE
A	Read the polynomial $P_n(x) = \alpha_0 x^n + \alpha_1 x^{n-1} + \cdots \alpha_n$ coefficients
B	Normalizes the polynomial coefficients if $a_0 \neq 1$
D	Kernel of the algorithm to determine $x^2 + px + q = 0$
d	Main while loop of algorithm
J	Copy $a_k \longleftarrow b_k$ and lower the polynomial degree $n \longleftarrow n - 2$
a	Solves $x^2 + px + q = 0$ and displays solutions
01 & c	Solves if polynomial degree $n = 1$ and display solution
02 & b	Solves if polynomial degree $n = 2$ and displays solutions
04	Check convergence if $m > 80 - -- > ERROR$

Print the solutions obtained to the text files

1. Shift + SETUP $\longrightarrow$ Printing set Text Print (X)
2. Shift + PRINT $\longrightarrow$ PON (enable) MAN (enable)

DM42 code

```
00 { 819-Byte Prgm }
01 LBL "BAI"
02 SIZE 100            @ Set 100 real registers
03 CLRG      @ Clear all registers
04 ALL      @ View all digits on the LCD
05 CLLCD      @ Clears the LCD display
06 CLST      @ Clears all registers of the X, Y, Z, T stack
07 "Polynomial Root"
08 AVIEW
09 PSE
10 PSE
11 PSE
12 PSE
13 "Finder n22"
14 AVIEW
15 PSE
16 PSE
17 PSE
18 PSE
19 "a0X↑n+...+an"
20 AVIEW
21 PSE
22 PSE
23 PSE
24 PSE
25 " n = ?"
26 PROMPT      @ Reads the degree of polynomial n
27 STO 90
28 STO 91
29 CLA
30 " n = "
31 AIP
32 AVIEW
33 PSE
34 CLST
```

```
35 CLA
36 RCL 91
37 1000
38 ÷
39 STO 92
40 LBL A            @ Reads all the polynomial coefficients
41 " X↑"            @ starting from the highest degree
42 RCL 91
43 AIP
44 AVIEW
45 PROMPT
46 STO IND 93
47 CLST
48 " = "               @ATTENTION look at the photo after the code !!!
49 ARCL IND 93
50 AVIEW
51 PSE
52 PSE
53 1
54 STO+ 93
55 STO- 91
56 RCL 92
57 ISG 92
58 GTO A
59 CLST
60 1
61 RCL 00        @ Check if the polynomial is monic in the case
62 XY?           @ATTENTION look at the photo after the code !!!
63 XEQ B        @ does not normalize it
64 " toll = ?"
65 PROMPT        @ Reads the tolerance required toll >> 1E-33
66 STO 99
67 CLA
68 " toll = "
69 ARCL 99
70 AVIEW
71 PSE
```

```
72 PSE
73 CLA
74 CLST
75 "... running"
76 AVIEW
77 LBL d   @ Main while loop of the Bairstow algorithm
78 3
79 STO 94
80 RCL 94
81 RCL 90
82 X<Y?
83 GTO 02  @ Check if n < 3 ( n = 2 or n = 1) jump to GTO 02
84 1       @ otherwise it initializes p = q = 1 and calls
85 STO 95  @ subroutine D find X↑2+pX+q = 0
86 STO 96  @ subroutine J copy An<-- Bn and lower polynomial degree n <--- n-2
87 XEQ D   @ subroutine a solves X↑2+pX+q = 0 and displays solutions
88 XEQ J
89 XEQ a
90 STOP
91 GTO d
92 LBL 02   @ Check if n = 2 calculates the solutions of the 2 degree trinomial
93 2        @ using the subroutine b
94 STO 94   @ if n = 1 jump to 01
95 RCL 94
96 RCL 90
97 XY?      @ATTENTION look at the photo after the code !!!
98 GTO 01
99 XEQ b
100 GTO 90
101 LBL 01 @ If n = 1 determines the real solution and displays the solution
102 XEQ c  @ using the subroutine c
103 LBL 90 @ All the solutions have been found STOP
104 CLA
105 " ... stop "
106 AVIEW
107 RTN
108 LBL B  @ Subroutine D to make the polynomial Pn(x) monic
```

```
109 1            @ Pointer register 93 to access the An
110 STO 93
111 RCL 90       @ Recalls the degree of polynomial n
112 1000             @ Initialize FOR loop using register 92 as index
113 ÷
114 STO 92
115 LBL C        @ Main FOR loop
116 RCL 00
117 STO÷ IND 93 @ Divide all the An / A0 coefficients using the pointer
118 1            @ increase pointer 93 (indirect access)
119 STO+ 93
120 RCL 92
121 ISG 92       @ Repeat on all the coefficients An except the first
122 GTO C
123 1            @ Initialize A0 <--- 1
124 STO 00
125 RTN
126 LBL D        @ Subroutine D kernel of Bairstow algorithm X↑2+pX+q = 0
127 1            @ Initialize ERROR <--- 1.0
128 STO 97
129 0            @ Initialize number of iterations m = 0
130 STO 98
131 LBL I        @ Subroutine I calculate the Bk coefficients using
132 1            @ the pointer register 91 to access the Ak
133 STO 91
134 RCL 90
135 1000
136 ÷
137 1
138 +
139 STO 89       @ Register 89 as an index of the FOR loop
140 30           @ registers location where b(k) are saved
141 STO 92       @ Pointer used to save / access b (k)
142 STO 93
143 STO 94
144 1
145 STO+ 93      @ Pointer used to save / access b (k + 1)
```

```
146 STO+ 94
147 STO+ 94         @ Pointer used to save / access b (k + 2)
148 0               @ Inizialize b(1) <--- 0
149 STO IND 92
150 1               @ Inizialize b(2) <--- 1
151 STO IND 93
152 LBL E           @ FOR loop to calculate all b (k)
153 RCL IND 91
154 STO IND 94
155 RCL IND 93
156 RCL 95
157 ×
158 +/-
159 STO+ IND 94
160 RCL IND 92
161 RCL 96
162 ×
163 +/-
164 STO+ IND 94
165 1
166 STO+ 91
167 STO+ 92
168 STO+ 93
169 STO+ 94
170 RCL 89
171 ISG 89
172 GTO E
173 RCL IND 92  @ Calls b (n + 1) and saves it in register 84
174 STO 84      @ for subsequent calculations D, Dp and Dq
175 RCL IND 93  @ Calls b (n + 2) and saves it in register 85
176 STO 85      @ for subsequent calculations D, Dp and Dq
177 30          @ Initialize pointer 91 where b (k) are located
178 STO 91
179 RCL 90      @ Initializes the index of the FOR loop
180 1000          @ and saves it in register 89
181 ÷
182 1
```

```
183 +
184 STO 89
185 60       @ registers location where c(k) are saved
186 STO 92   @ Pointer used to save / access c(k)
187 STO 93
188 STO 94
189 1
190 STO+ 93  @ Pointer used to save / access c(k+1)
191 STO+ 94
192 STO+ 94  @ Pointer used to save / access c(k+2)
193 0        @ Inizialize c(1) <--- 0
194 STO IND 92
195 0        @ Inizialize c(2) <--- 0
196 STO IND 93
197 LBL F    @ FOR loop to calculate all c (k)
198 RCL IND 91
199 STO IND 94
200 RCL IND 93
201 RCL 95
202 ×
203 +/-
204 STO+ IND 94
205 RCL IND 92
206 RCL 96
207 ×
208 +/-
209 STO+ IND 94
210 1
211 STO+ 91
212 STO+ 92
213 STO+ 93
214 STO+ 94
215 RCL 89
216 ISG 89
217 GTO F
218 RCL IND 92  @ Call c (n + 1) and save it in register 92
219 STO 92      @ for subsequent calculations D, Dp and Dq
```

```
220 RCL IND 93  @ Call c (n + 2) and save it in register 93
221 STO 93      @ for subsequent calculations D, Dp and Dq
222 CLST        @ Clears all registers of the X, Y, Z, T stack
223 RCL 95      @ Calculate D
224 RCL 93
225 ×
226 RCL 96
227 RCL 92
228 ×
229 +
230 RCL 92
231 ×
232 RCL 93
233 RCL 93
234 ×
235 +
236 STO 88
237 RCL 84      @ Calculate Dp
238 RCL 93
239 ×
240 RCL 85
241 RCL 92
242 ×
243 -
244 RCL 88
245 ÷
246 STO 87
247 RCL 95     @ Calculate Dq
248 RCL 93
249 ×
250 RCL 96
251 RCL 92
252 ×
253 +
254 RCL 84
255 ×
256 RCL 85
```

```
257 RCL 93
258 ×
259 +
260 RCL 88
261 ÷
262 STO 86
263 RCL 87    @ Calculate and save in the register 97 <--- max(|Dp|,!Dq|)
264 ABS
265 STO ST X
266 RCL 86
267 ABS
268 STO ST Y
269 X>Y?
270 GTO G
271 RCL 87
272 ABS
273 STO 97
274 GTO H
275 LBL G
276 RCL 86
277 ABS
278 STO 97
279 LBL H
280 RCL 87    @ Update p <--- p + Dp
281 STO+ 95
282 RCL 86    @ Update q <--- q + Dq
283 STO+ 96
284 1         @ Update the number of iterates m <--- m + 1
285 STO+ 98
286 80        @ Test if m > 80 ?
287 STO ST X
288 RCL 98
289 X>Y?
290 GTO 04
291 RCL 99     @ Test if max(|Dp|,!Dq|) < toll if you go out
292 RCL 97     @ otherwise continue to cycle
293 X>Y?
```

```
294 GTO I
295 RTN
296 LBL J    @ Polynomial degree n <--- n-2 & subroutine J copy Ak<-- Bk
297 0
298 STO 91
299 31
300 STO 92
301 RCL 90
302 1
303 -
304 STO 93
305 1000
306 ÷
307 STO 94
308 LBL 88
309 RCL IND 92
310 STO IND 91
311 1
312 STO+ 91
313 STO+ 92
314 RCL 94
315 ISG 94
316 GTO 88
317 RCL 90
318 2
319 -
320 STO 90
321 RTN
322 LBL a    @ subroutine a calculates and displays the
323 CLA      @ solutions of X↑2+pX+q = 0
324 CLST
325 "... m = "
326 ARCL 98
327 " n = "
328 ARCL 90
329 AVIEW
330 RCL 95
```

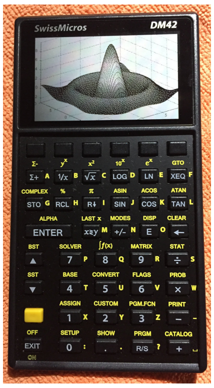
```
331 +/-
332 2
333 ÷
334 STO 91
335 X↑2
336 RCL 96
337 -
338 STO 92
339 CLST
340 RCL 91
341 RCL 92
342 SQRT
343 +
344 RCL 91
345 RCL 92
346 SQRT
347 -
348 PRSTK
349 RTN
350 LBL b     @ If n = 2 calculates and displays the solutions
351 CLA
352 CLST
353 "... continue"
354 AVIEW
355 RCL 01
356 +/-
357 2
358 ÷
359 STO 91
360 X↑2
361 RCL 02
362 -
363 STO 92
364 CLST
365 RCL 91
366 RCL 92
367 SQRT
```

```
368 +
369 RCL 91
370 RCL 92
371 SQRT
372 -
373 PRSTK
374 RTN
375 LBL c      @ If n = 1 calculate and visualize the real solution
376 CLA
377 CLST
378 "... continue"
379 AVIEW
380 RCL 01
381 +/-
382 STO 91
383 CLST
384 RCL 91
385 PRSTK
386 RTN
387 LBL 04      @ Test if m > 80 ?
388 CLA
389 "ERROR m > 80 ! "
390 AVIEW
391 CLST
392 STOP
393 RTN
```

ATTENTION **ATTENTION** **ATTENTION**

```
48 ├" = "

62 X≠Y?

97 X≠Y?
```

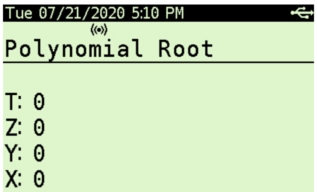
Examples and Comparisons

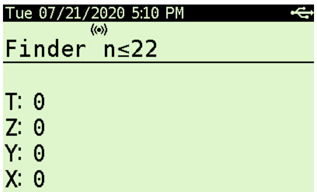


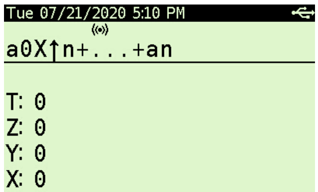
Example1

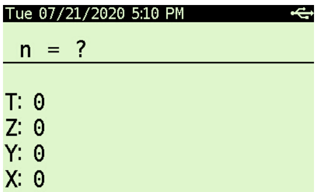
$$P_5(x) = 2(x-1)(x-2)(x-3)(x-4)(x-5) =$$
$$= 2x^5 - 30x^4 + 170x^3 - 450x^2 + 548x - 240 = 0$$

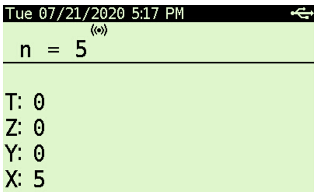
DM42

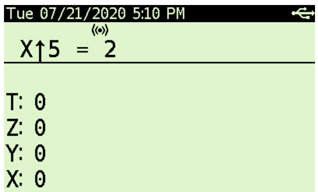


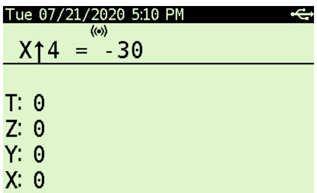


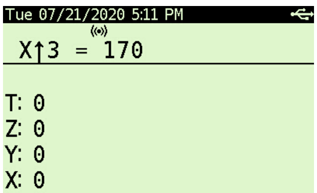


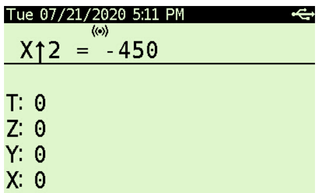


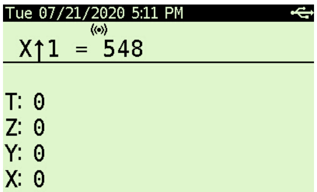


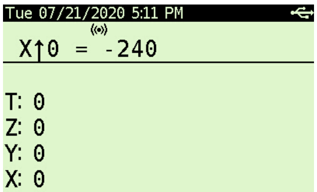


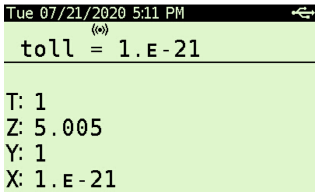


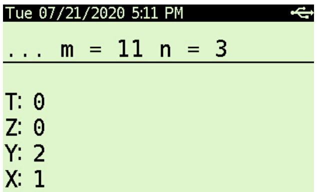


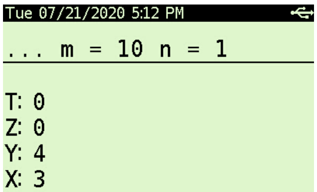


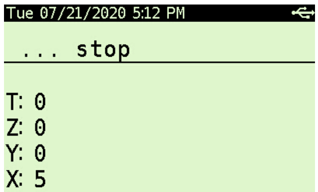












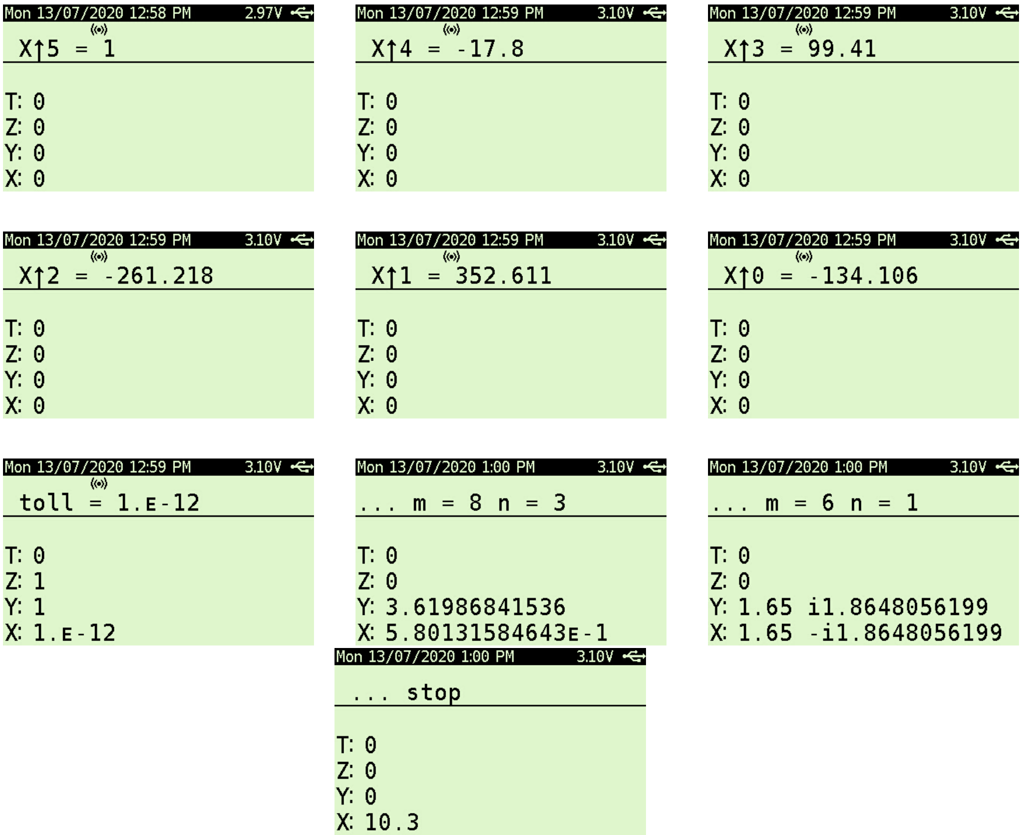

```
Polynomial Root
Finder n<22
a0X^n+...+an
 n = 5
 X^5
Polynomial Root
Finder n22
a0X^n+...+an
 n = 5
 X^5
 X^5 = 2
 X^4
 X^4 = -30
 X^3
 X^3 = 170
 X^2
 X^2 = -450
 X^1
 X^1 = 548
 X^0
 X^0 = -240
 toll = 1.-21
... running
```

```
... m = 11 n = 3

T=                      0
Z=                      0
Y=                      2
X=                      1
... m = 10 n = 1

T=                      0
Z=                      0
Y=                      4
X=                      3
... continue

T=                      0
Z=                      0
Y=                      0
X=                      5
 ... stop
```

Example2[2]

$$P_5(x) = x^5 - 17.8x^4 + 99.41x^3 - 261.218x^2 + 352.611x - 134.106 = 0$$

DM42




```
               XEQ "BAI"
n = ?
                   5    RUN
n = 5
X^5
X^5
                   1    RUN
```

[2]Example from TI-58/59 Module 11 (1978) Texas Instruments Incorporated.

```
 X^5 = 1
 X^4
 X^4
            -17.8    RUN
 X^4 = -17.8
 X^3
 X^3
            99.41    RUN
 X^3 = 99.41
 X^2
 X^2
         -261.218    RUN
 X^2 = -261.218
 X^1
 X^1
          352.611    RUN
 X^1 = 352.611
 X^0
 X^0
         -134.106    RUN
 X^0 = -134.106
 toll = ?
            1-12    RUN
 toll = 1.-12
... running
... m = 8 n = 3

T=                    0
Z=                    0
Y=         3.61986841536
X=      5.80131584643-1
                     RUN
... m = 6 n = 1

T=                    0
Z=                    0
Y=    1.65 i1.8648056199
```

```
X=   1.65 -i1.8648056199
                     RUN
... continue

T=                     0
Z=                     0
Y=                     0
X=                  10.3
 ... stop
```

Comparison of solutions with MATLAB®

```
>> p = [1 -17.8 99.41 -261.218 352.611 -134.106];
>> roots(p)

ans =

 10.299999999999999 + 0.000000000000000i
  3.619868415357074 + 0.000000000000000i
  1.649999999999998 + 1.864805619897151i
  1.649999999999998 - 1.864805619897151i
  0.580131584642934 + 0.000000000000000i

>> vpa(roots(p),50) % 50 digits of precision !!!

ans =

                                                        10.3
         3.6198684153570743760042205394711345434188842773438
 1.65 + 1.8648056198971516398554778633752676781690560441755i
 1.65 - 1.8648056198971516398554778633752676781690560441755i
        0.58013158464293379523724070168100297451019287109375
```

Example3

In numerical analysis, Wilkinson's polynomial is a specific polynomial which was used by James H. Wilkinson in 1963 to illustrate a difficulty when finding the root of a polynomial: the location of the roots can be very sensitive to perturbations in the coefficients of the polynomial.

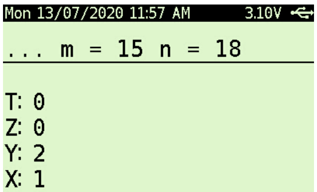
The polynomial is

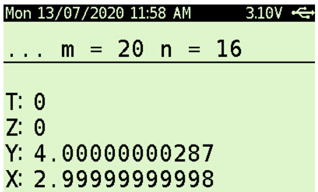
$$P_{20}(x) = \prod_{i=1}^{20}(x-i) = a_0x^{20} + a_1x^{19} + a_2x^{18} + \cdots + a_{20}$$

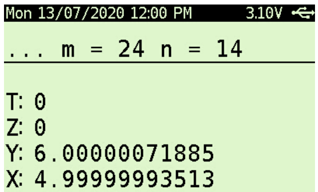
and $toll = 1 \cdot 10^{-12}$

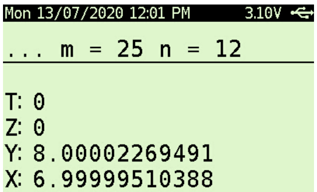
Coefficient	**Value**
a_0	1
a_1	-210
a_2	20615
a_3	-1256850
a_4	53327946
a_5	-1672280820
a_6	40171771630
a_7	-756111184500
a_8	11310276995381
a_9	-135585182899530
a_{10}	1307535010540395
a_{11}	-10142299865511450
a_{12}	63030812099294896
a_{13}	-311333643161390656
a_{14}	1206647803780373248
a_{15}	-3599979517947607040
a_{16}	8037811822645052416
a_{17}	-12870931245150988288
a_{18}	13803759753640704000
a_{19}	-8752948036761600000
a_{20}	2432902008176640000

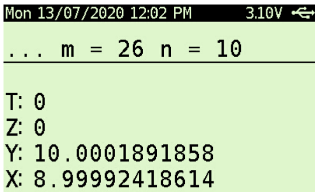
DM42

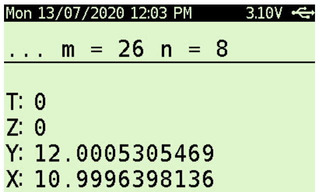


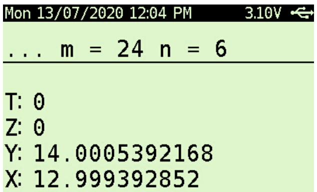


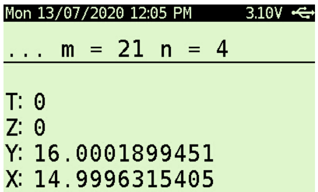


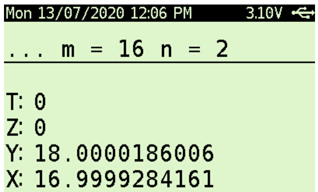


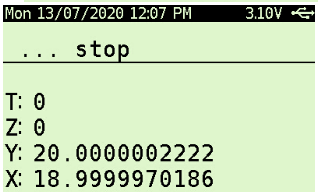













```
XEQ "BAI"
 n = ?
                 20     RUN
 n = 20
 X^20
 X^20
                  1     RUN
 X^20 = 1
 X^19
 X^19
               -210     RUN
 X^19 = -210
 X^18
 X^18
```

```
          20,615    RUN
 X^18 = 20,615
 X^17
 X^17
      -1,256,850    RUN
 X^17 = -1,256,850
 X^16
 X^16
      53,327,946    RUN
 X^16 = 53,327,946
 X^15
 X^15
  -1,672,280,820    RUN
 X^15 = -1,672,280,820
 X^14
 X^14
  40,171,771,630    RUN
 X^14 = 40,171,771,630
 X^13
 X^13
 -756,111,184,500    RUN
 X^13 = -756,111,184,500
 X^12
 X^12
11,310,276,995,381   RUN
 X^12 = 1.1310276995413
 X^11
 X^11
-135,585,182,899,530 RUN
 X^11 = -1.35585182914
 X^10
 X^10
1,307,535,010,540,395 RU
N
 X^10 = 1.3075350105415
 X^9
 X^9
```

```
-10,142,299,865,511,450
RUN
 X^9 = -1.0142299865516
 X^8
 X^8
63,030,812,099,294,896 R
UN
 X^8 = 6.3030812099316
 X^7
 X^7
-311,333,643,161,390,656
 RUN
 X^7 = -3.1133364316117
 X^6
 X^6
1,206,647,803,780,373,24
8                    RUN
 X^6 = 1.2066478037818
 X^5
 X^5
-3,599,979,517,947,607,0
40                   RUN
 X^5 = -3.5999795179518
 X^4
 X^4
8,037,811,822,645,052,41
6                    RUN
 X^4 = 8.0378118226518
 X^3
 X^3
-12,870,931,245,150,988,
288                  RUN
 X^3 = -1.2870931245219
 X^2
 X^2
13,803,759,753,640,704,0
00                   RUN
```

```
 X^2 = 1.3803759753619
 X^1
 X^1
-8,752,948,036,761,600,0
00                   RUN
 X^1 = -8.7529480367618
 X^0
 X^0
2,432,902,008,176,640,00
0                    RUN
 X^0 = 2.4329020081818
 toll = ?
            1-12    RUN
 toll = 1.-12
... running
... m = 15 n = 18

T=                     0
Z=                     0
Y=                     2
X=                     1
                     RUN
... m = 20 n = 16

T=                     0
Z=                     0
Y=         4.00000000287
X=         2.99999999998
                     RUN
... m = 24 n = 14

T=                     0
Z=                     0
Y=         6.00000071885
X=         4.99999993513
                     RUN
... m = 25 n = 12
```

```
T=                      0
Z=                      0
Y=          8.00002269491
X=          6.99999510388
                      RUN
... m = 26 n = 10

T=                      0
Z=                      0
Y=          10.0001891858
X=          8.99992418614
                      RUN
... m = 26 n = 8

T=                      0
Z=                      0
Y=          12.0005305469
X=          10.9996398136
                      RUN
... m = 24 n = 6

T=                      0
Z=                      0
Y=          14.0005392168
X=           12.999392852
                      RUN
... m = 21 n = 4

T=                      0
Z=                      0
Y=          16.0001899451
X=          14.9996315405
                      RUN
... m = 16 n = 2

T=                      0
```

```
Z=                    0
Y=         18.0000186006
X=         16.9999284161
                    RUN
... continue

T=                    0
Z=                    0
Y=         20.0000002222
X=         18.9999970186
 ... stop
```

Comparison of solutions with MATLAB®

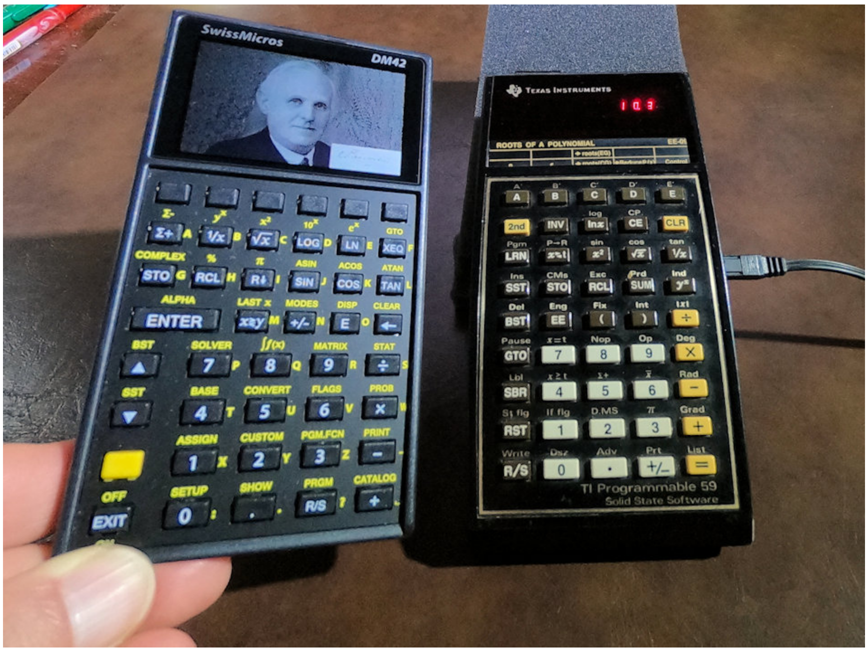
```
>> p = vpa(poly(1:20),50)'  %50 digits of precision !!!
p =
                      1
                   -210
                  20615
               -1256850
               53327946
            -1672280820
            40171771630
          -756111184500
         11310276995381
       -135585182899530
       1307535010540395
     -10142299865511450
      63030812099294896
    -311333643161390656
    1206647803780373248
   -3599979517947607040
    8037811822645052416
  -12870931245150988288
   13803759753640704000
```

```
  -8752948036761600000
   2432902008176640000

>> vpa(roots(p),50) %50 digits of precision !!!

ans =

 20.000000222199534868713599273462258466712127004597
 18.999997018587796499599382829761397076325793162443
 18.000018600605906061623674706363960053777913260240
 16.999928416017085118893055685779730825337982350020
 16.000189945470409472562242509352720428219240209614
 14.999631539779625744239784311608445427666843675867
 14.000539217936149354403824840292397964289943561236
 12.999392850542677085285165188054458878502693302973
 12.000530548412933592244708916464305835587941782724
 10.999639812328610607970362553159438217574647568529
 10.000189186679827908616860814451413349963023133105
 8.9999241856822158235050949809045751771845269373654
 8.0000226951019706281389263917095850048514492411022
 6.9999951038170559499797094372963554834034094545611
 6.0000007188589671560339023143964654294808075650932
 4.9999999351265723893873617834882112718161476762018
 4.0000000028712551058351832250815870668565313083984
 2.9999999999829963065286023953425383926195171558183
 1.99999999999984005932064258391586578920548494514000
 1.0000000000000097332321320038714977577746034770310
```

Comparison between DM42 and TI-59®

In this section I would like to compare the performance (speed and accuracy) between my first programmable calculator the TI-59 (1977) purchased when I was a young student in high school (1982) with the DM42 (2017) purchased in May 2020. Forty years exactly after the two calculators were released. The performances are obviously incomparable for several reasons, the main one is certainly the miniaturization of the transistors inside modern CPUs.

	TI-59®	**DM42**
CPU	TMC0501	STM32L476
Data Bus	4 bits	32 bits
$f_{clock}(max)$	230 kHz	80 MHz
Precision Digits	13	34
Display Digits	10	34
Registers Maximum	100	variable
Program Steps Maximum	960	
Magnetic Card Reader	YES	
Module ROM	YES	

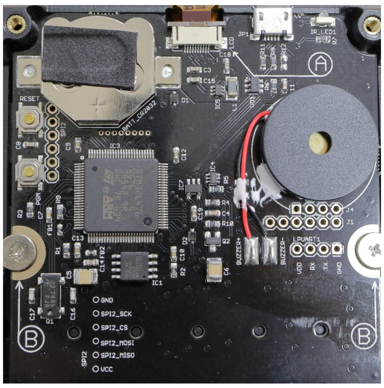


DM42 & TI-59®

TI-59® Front-Side

TI-59® Back-Side

TI-59® Hardware

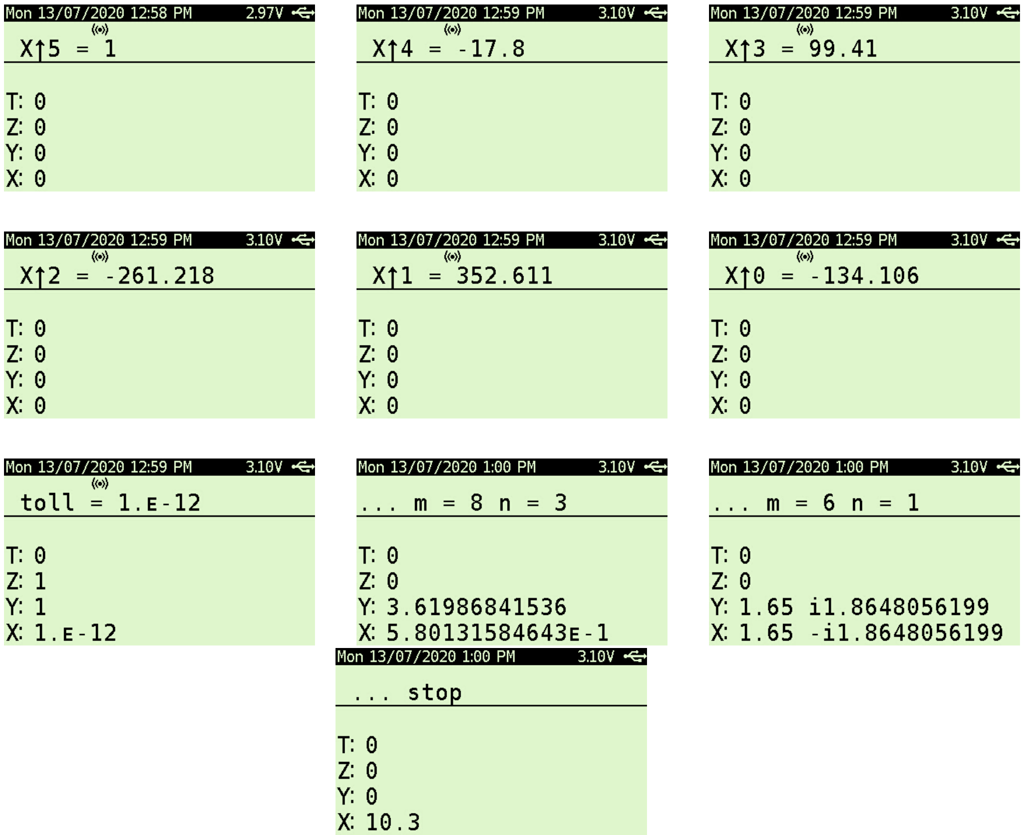


DM42 Hardware

Example2 with DM42 & TI-59® + Module EE11[3]

$$\mathrm{P}_5(x) \;=\; x^5 - 17.8x^4 + 99.41x^3 - 261.218x^2 + 352.611x - 134.106 \;=\; 0$$

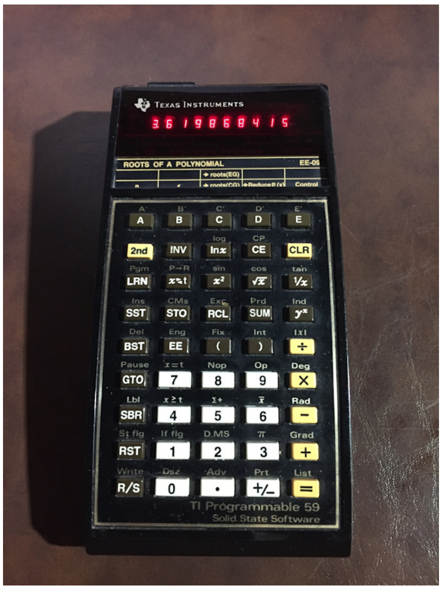
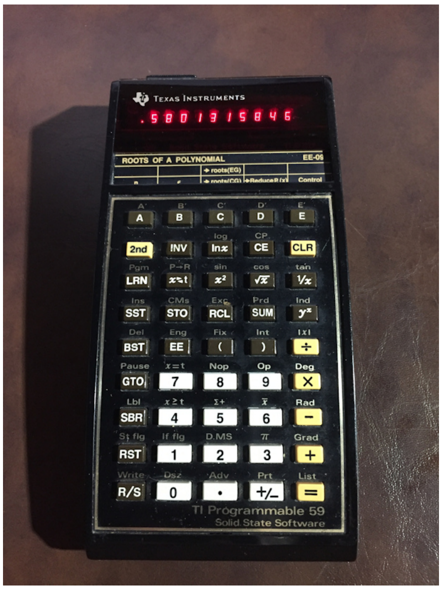
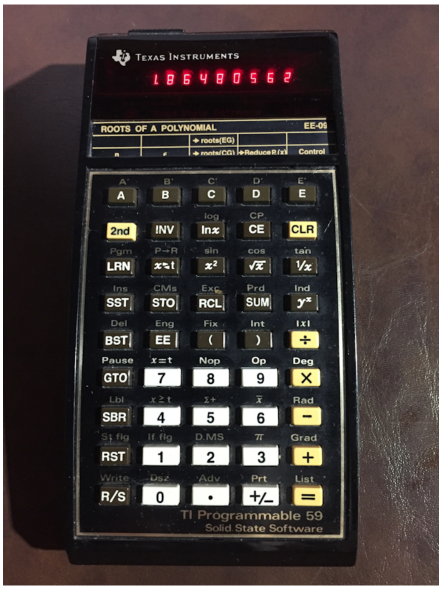
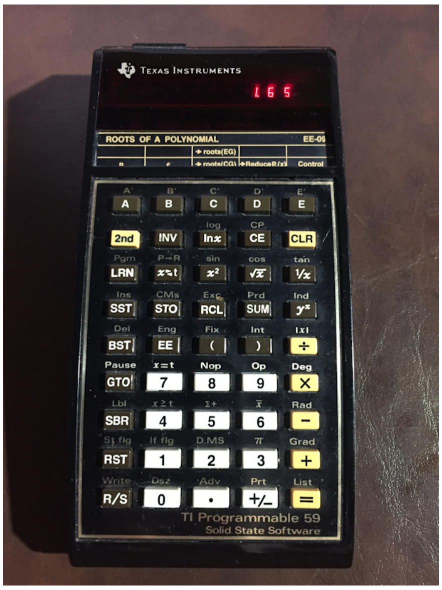
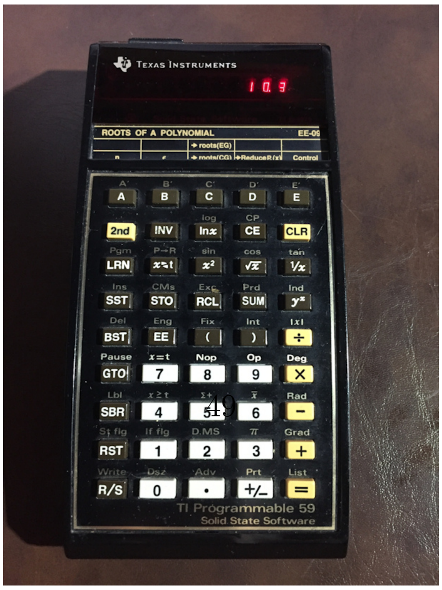
DM42



DM42 execution time $\ll$ 1 [s]

TI59® execution time = 150 [s]

[3]Example from TI-58/59 Module 11 (1978) Texas Instruments Incorporated.


Texas Instruments
ROOTS OF A POLYNOMIAL
TI Programmable 59
Solid State Software

Texas Instruments
ROOTS OF A POLYNOMIAL
TI Programmable 59
Solid State Software

Texas Instruments
ROOTS OF A POLYNOMIAL
TI Programmable 59
Solid State Software

Texas Instruments
ROOTS OF A POLYNOMIAL
TI Programmable 59
Solid State Software

Texas Instruments
ROOTS OF A POLYNOMIAL
TI Programmable 59
Solid State Software

Example4

In numerical analysis, Wilkinson's polynomial is a specific polynomial[4] which was used by James H. Wilkinson in 1963 to illustrate a difficulty when finding the root of a polynomial: the location of the roots can be very sensitive to perturbations in the coefficients of the polynomial.

The polynomial is

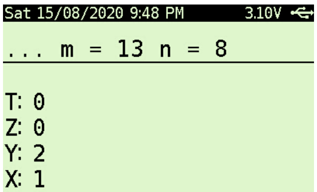
$$P_{10}(x) = \prod_{i=1}^{10}(x-i) = a_0x^{10} + a_1x^9 + a_2x^8 + \cdots + a_{10}$$

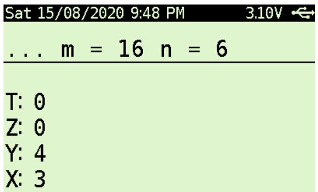
and $toll = 1 \cdot 10^{-12}$ for DM42 while $toll = 1 \cdot 10^{-9}$ for TI59®

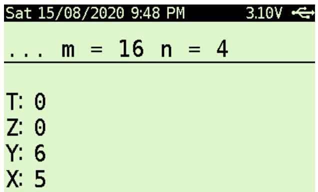
Coefficient	**Value**
a_0	1
a_1	-55
a_2	1320
a_3	-18150
a_4	157773
a_5	-902055
a_6	3416930
a_7	-8409500
a_8	12753576
a_9	-10628640
a_{10}	3628800

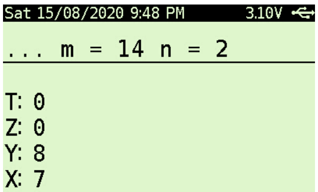
[4] Due to TI59's reduced ability to represent integers, I had to limit n = 10 instead of 20.

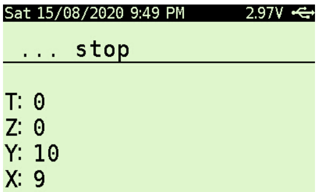
DM42





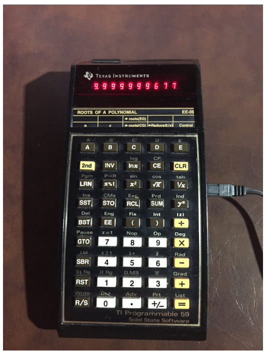


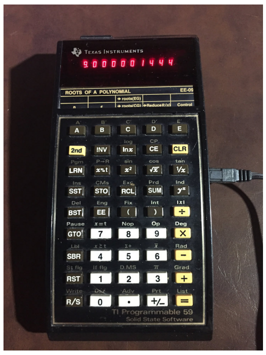
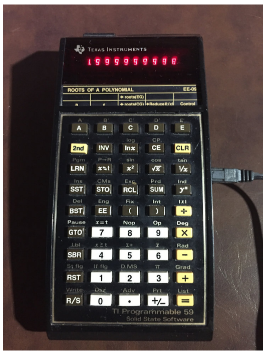
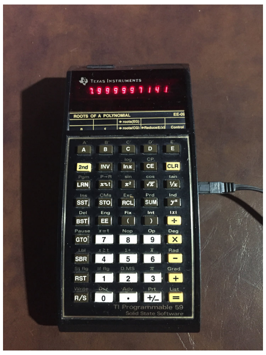


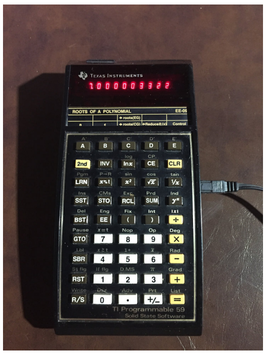


DM42 execution time = 4 [s] f_{clock} **= 80 MHz**

TI59® execution time = 8280 [s] !!!






Comparison between DM42, HP50G® and HP48G®

In this section I would like to compare the performance (speed and accuracy) between DM42 with HP48G® (1993) and HP50G® (2006).

	HP48G®	**HP50G®**	**DM42**
CPU	Saturn Yorke	ARM9	STM32L476
Data Bus	4 bits		32 bits
$f_{clock}(max)$	4 MHz	75 MHz	80 MHz
Precision Digits		15	34
ROM	512kB	2 MB	8 MB
RAM	32 kB	512 kB	75 kB

DM42 & HP50G® & HP48G®

Example3

In numerical analysis, Wilkinson's polynomial is a specific polynomial which was used by James H. Wilkinson in 1963 to illustrate a difficulty when finding the root of a polynomial: the location of the roots can be very sensitive to perturbations in the coefficients of the polynomial.

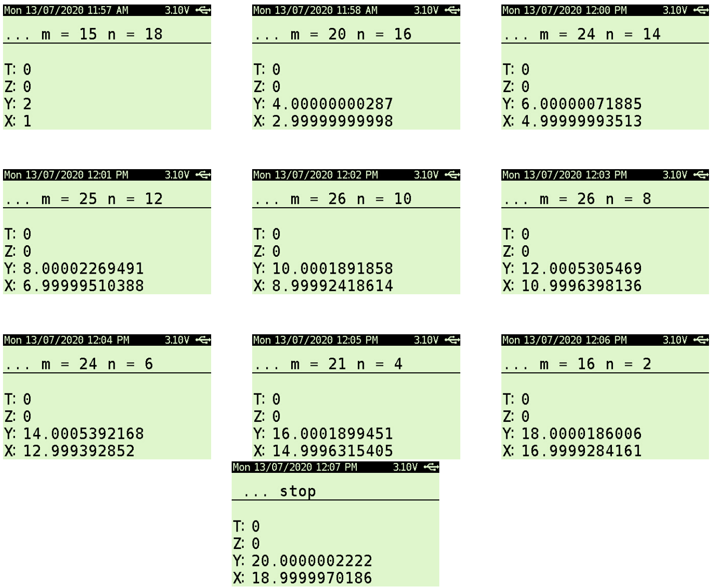
The polynomial is

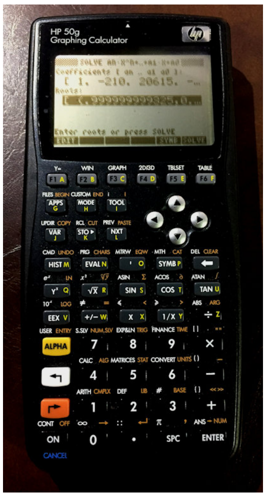
$$P_{20}(x) = \prod_{i=1}^{20}(x-i) = a_0x^{20} + a_1x^{19} + a_2x^{18} + \cdots + a_{20}$$

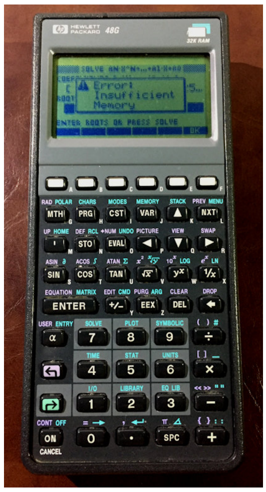
and $toll = 1 \cdot 10^{-12}$

Coefficient	**Value**
a_0	1
a_1	-210
a_2	20615
a_3	-1256850
a_4	53327946
a_5	-1672280820
a_6	40171771630
a_7	-756111184500
a_8	11310276995381
a_9	-135585182899530
a_{10}	1307535010540395
a_{11}	-10142299865511450
a_{12}	63030812099294896
a_{13}	-311333643161390656
a_{14}	1206647803780373248
a_{15}	-3599979517947607040
a_{16}	8037811822645052416
a_{17}	-12870931245150988288
a_{18}	13803759753640704000
a_{19}	-8752948036761600000
a_{20}	2432902008176640000

DM42







DM42 execution time = 10 [s] f_{clock} = 80 MHz

HP50G® execution time = 14 [s] f_{clock} = 75 MHz

HP48G® execution time = not evaluable

HP50G®	Roots of $P_{20}(x)$
x_1	0.999999999325
x_2	2.000000080220
x_3	2.99999843200
x_4	3.99999054543
x_5	5.00048951553
x_6	5.99635310588
x_7	6.98799614792
$x_{8,9}$	$8.17180636115 \pm i\ 0.43452021603$
$x_{10,11}$	$12.1018913985 \pm i\ 2.24809179333$
$x_{12,13}$	$14.6459622817 \pm i\ 2.44093568498$
$x_{14,15}$	$9.88718717922 \pm i\ 1.48403068110$
$x_{16,17}$	$17.1174681588 \pm i\ 1.89829970819$
x_{18}	20.0956132820
$x_{19,2}$	$19.0354640665 \pm i\ 0.811243731857$

Example4

In numerical analysis, Wilkinson's polynomial is a specific polynomial which was used by James H. Wilkinson in 1963 to illustrate a difficulty when finding the root of a polynomial: the location of the roots can be very sensitive to perturbations in the coefficients of the polynomial.

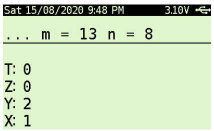
The polynomial is

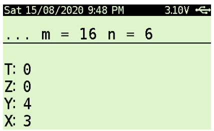
$$P_{10}(x) = \prod_{i=1}^{10}(x-i) = a_0x^{10} + a_1x^9 + a_2x^8 + \cdots + a_{10}$$

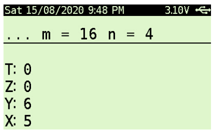
and $toll = 1 \cdot 10^{-12}$ for DM42

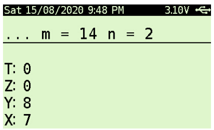
Coefficient	Value
a_0	1
a_1	-55
a_2	1320
a_3	-18150
a_4	157773
a_5	-902055
a_6	3416930
a_7	-8409500
a_8	12753576
a_9	-10628640
a_{10}	3628800

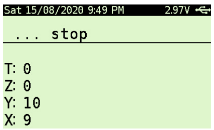
DM42

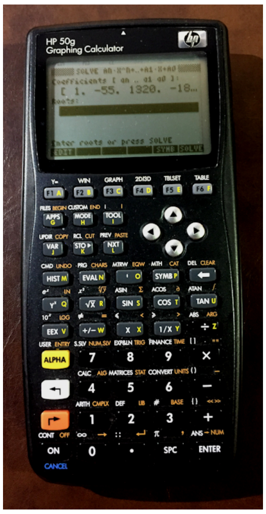


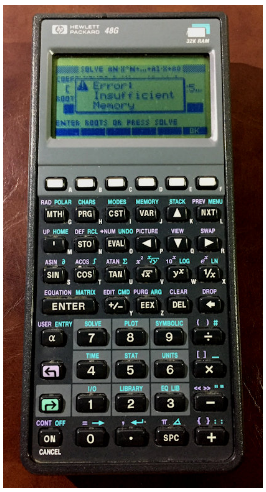












DM42 execution time = 4 [s] f_{clock} **= 80 MHz**

HP50G® execution time = 4 [s] f_{clock} **= 75 MHz**

HP48G® execution time = not evaluable

HP50G®	**Roots of $P_{10}(x)$**
x_1	0.999999999999
x_2	2.000000000010
x_3	3.000000000010
x_4	3.999999999480
x_5	5.000000003860
x_6	5.999999986310
x_7	7.000000026370
x_8	7.999999971800
x_9	9.000000015730
x_{10}	9.999999996430

... initial problem

$$\lambda^8 + 20.4\lambda^7 + 151.3\lambda^6 + 490\lambda^5 + 687\lambda^4 + 719\lambda^3 + 150\lambda^2 + 109\lambda + 6.87 \;=\; 0$$

the quadratic factors obtained by Professor Leonard Bairstow in 1920 were:

1. $\left(\lambda^2 + 11.25\lambda + 35.1\right)$
2. $\left(\lambda^2 - 0.006\lambda + 0.171\right)$
3. $\left(\lambda + 7.79\right)\left(\lambda + 0.067\right)$
4. $\left(\lambda^2 + 1.33\lambda + 2.19\right)$

while the solutions obtained with the DM42 with a $toll = 1 \cdot 10^{-21}$

1. $\left(\lambda^2 + 11.2170142414\lambda + 34.9705347691\right)$
2. $\left(\lambda^2 - 0.00566048716464\lambda + 0.1707972788\right)$
3. $\left(\lambda + 7.8575856905\right)\left(\lambda + 0.067381378159\right)$
4. $\left(\lambda^2 + 1.33550629852\lambda + 2.19246512844\right)$

Curiosity1

Very often Leonard Bairstow's algorithm is also called Lin-Bairstow, this is probably due to the fact that in 1943 a Chinese mathematician Lin Shih-Nge developed an algorithm similar to the one previously deduced by Bairstow. Many people over the years have tried to improve the algorithm, just to name a few in chronological order:

1. Bairstow, L., "Investigations Relating to the Stability of the Aeroplane", R and M 154 of Advisory Committee for Aeronautics (1914).
2. Lin, Shih-nge, "A Method for Finding Roots of Algebraic Equations", J. Math. and Phys. 22:60-77 (1943).
3. Friedman, B., "Note on Approximating Complex Zeros of a Polynomial", Comm. Pure Appl. Math ~:195-208 (1949).
4. Luke, Y, L., and Ufford, D., "On the Roots of Algebraic Equations", J. Math. and Phys. 30:94-101 (1951).
5 Kantorovich, L. V., "Functional Analysis and Applied Mathematics", N,B.S. Report 1509 (translation by C, D. Benster and edited by G. E. Forsythe).
6. Henrici, Peter, "Elements of Numerical Analysis", Wiley and Sons, New York (1964).
7. Forsythe, G. E., "Generation and the use of Orthogonal Polynomials for Data Fitting with a Digital Computer" J. Sot. Indust. Appl. Math. 5:74-88 (1957). -
8. Waltmann, W. Lo, and Lambert, R. J., "T-Algorithm for Tridiagonalization", J. Indust, Appl, Math. 13:1069-78 (1965).
9. G. H. Golub, T. N. Robertson, "A Generalized Bairstow Algorithm", Technical Report no. 54 January 13 (1967)

Curiosity2

If anyone was curious about the programmable calculators I purchased here is the list: TI59®, TI95®, HP48G®, HP50G® and last DM42 all are still fully functional.

Acknowledgments

I have always had the passion since I was a student of mathematics applied to science, in these 31 years of teaching I have tried to stimulate students to be curious, some of them have been, to mention only the latest Alessandro Moglia whom I wish to continue their university studies with satisfaction. Thanks also for my old student Dr. Eng. Samuele Becchia now my friend for reading the article and his suggestions.

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling and Brian P. Flannery (1992), *Numerical Recipes*, Cambridge University Press

[2] Anthony Ralston, Philip Rabinowitz (1978) *A First Curse In Numerical Analysis*, McGraw-Hill

[3] Electrical Engineering (1979) *TI Programmable 58/59 Module - 11*, Texas Instruments Incorporated

[4] L. Bairstow (1920), *Applied Aerodynamics*, Longmans, Green and Co