Applied and Computational Mathematics
2014; 3(3): 68-74
Published online May 30, 2014 (http://www.sciencepublishinggroup.com/j/acm)
doi: 10.11648/j.acm.20140303.11

A numerical algorithm for the resolution of scalar and matrix algebraic equations using Runge-Kutta method

Tahar Latreche

Doctorate student in Civil Engineering, B.P. 129 Salem Lalmi, 40003 Khenchela, Algeria

Email address:
latrache.tahar@yahoo.ca

To cite this article:
Tahar Latreche. A numerical Algorithm for the Resolution of Scalar and Matrix Algebraic Equations Using Runge-Kutta Method. *Applied and Computational Mathematics.* Vol. 3, No. 3, 2014, pp. 68-74. doi: 10.11648/j.acm.20140303.11

Abstract: The Runge-Kutta method is an interesting and precise method for the resolution of ordinary differential equations. Fortunately, when supposing the differentiation by any variable that the equation to solve is not variable of, and after iterations, the solution of this equation stretches to the algebraic roots of this equation. This feature of this algorithm, indeed, allows to solve precisely any scalar or matrix equation. The numerical algorithm proposed herein is an iterative procedure of the fourth-order Runge-Kutta method with an adopted precision tolerance of convergence. Also, a method to determine all the roots of the polynomial equations is presented. Some scalar and matrix algebraic equations are resolved using this proposed algorithm, and show how this algorithm featuring with an excellent precision, a good speed and a simplicity for programming to solve equations and deduct the roots.

Keywords: Algebraic Equations, Linear and Non-Linear Algebra, Elementary Equations, Polynomial Equations, Runge-Kutta Method

1. Introduction

Indeed, that there's a lack in mathematics and numerical analysis of theoretical methods or really good precision algorithms or techniques for solving scalar or matrix, as well as real or complex algebraic equations [1-10], especially when these equations being too complex, and when the theoretical mathematics stays be unable to propose a theoretical solution.

The Fourth-Order Runge-Kutta R-K method is a precise and a converged method for the resolution of the Ordinary Differential Equations, and is an effective, efficiency, and the most useful method in the science and engineering practices [1-10]. The special case when we would integrate the Ordinary Differential Equation ODE with any variable which the main equation is not function of, this case indeed, automatically implicate that the proposed ODE will becoming an Algebraic Equation AE, and the seeking solution being the AE root.

Moreover, the roots of the getting AE cannot be precisely occurring by the first resolution of this equation using the R-K method, because that these roots are unknown and; but the started roots of this AE don't be in general the exact roots needed. If we start our search of roots from a scalar zero or a null matrix or a vector, so we should need a sufficiently number of iterations of the R-K method, according to the precision sought, for the root we want getting of the proposed AE. Thus, for any so simple or complex elementary functions algebraic equations, it shall only to iterate the fourth order Runge-Kutta method till the convergence will be checked. For the Algebraic Polynomial Equations, an algorithm allows us to deduct all the roots of any degree of these equations is proposed.

In fact, this proposed algorithm is concerning any AEs that have real or complex roots, and when an Algebraic Equation has no real roots, this equation converge iteratively to the close point from zero, or the null vector or matrix, and then start diverging, so we can conclude that we should apply the complex equations technique.

Among that the proposed algorithm needs often, some experience for the regulations of the step of the variable integrate on and of the convergence tolerance for the reason to get the seeking roots with the precision needed; but in reality, is simple to use and programming and its offered precision doesn't ever compared with another existing algorithms such that the Newton method. The algorithm proposed to determine the roots of polynomial equations indeed, is a new proposal in this field and is very useful to resolve a large matrix and scalar problems such that the Eigen-Values and Eigen-Vectors problems, which present a universal problems in science and engineering. Some scalar and matrix AEs are presented and resolved using this algorithm, and the resulting illustrations of the iteration loops allow us to remark that this proposed numerical algorithm is effectively, simple, useful and very precise to solve any scalar or matrix AEs for the science and engineering practices.

2. The Elementary Functions Equations

The elementary functions algebraic equations represent a part of complex mathematical equations. For these algebraic equations, like all the algebraic equations, the iteration procedures of the Fourth-Order-Runge-Kutta method, preferred to starts with scalar zero or null vector or matrix with a so small step of the variable dividing of, and of the tolerance of convergence (the tolerance of the convergence can be controlled, by the way of the convergence of the equation to resolve or, the differences between root iterations). Anyway, the algorithm is very simple, such that it iterates on the Runge-Kutta method until the convergence is checked.

Although, we should indicate that some programming languages, don't contain the elementary functions of vectors or matrices, and so we have evaluate these kind of functions using Taylor series. We should to indicate herein too, that we shall avoid starting the roots with the absolute scalar zero or null vector or matrix, for some equations contains indefinite functions in these abscissas like the functions $Ln(x)$ or $Ctan(x)$.

The iterative algorithms, for the resolution of a scalar and matrix algebraic equations $f(x)$ and $F(X)$, are formulated as follows:

$$\begin{cases} x^1 = 0 \\ k_1 = DT\left(f(x^{i-1})\right) \\ k_2 = DT\left(f(x^{i-1} + 0.5k_1)\right) \\ k_3 = DT\left(f(x^{i-1} + 0.5k_2)\right) \\ k_4 = DT\left(f(x^{i-1} + k_3)\right) \\ x^i = x^{i-1} \pm (k_1 + 2k_2 + 2k_3 + k_4)/6 \end{cases} \quad (1)$$

And,

$$\begin{cases} X^1 = 0 \\ K_1 = DT\left(F(X^{i-1})\right) \\ K_2 = DT\left(F(X^{i-1} + 0.5K_1)\right) \\ K_3 = DT\left(F(X^{i-1} + 0.5K_2)\right) \\ K_4 = DT\left(F(X^{i-1} + K_3)\right) \\ X^i = X^{i-1} \pm (K_1 + 2K_2 + 2K_3 + K_4)/6 \end{cases} \quad (2)$$

Where, DT is the step of the variable divided by, x^k is the scalar root and X^k the matrix root for the k^{th} iteration of the elementary functions equations $f(x)$ and $F(X)$, respectively. k_l and K_l are the l^{th} scalar and matrix Runge-Kutta method coefficients.

The checking of the convergence is given by the logical expressions:

$$f(x) < conv \text{ and, } F(X)(i,j) < conv$$

Such that $conv$ is the convergence needed.

As an example, the FORTRAN algorithm for the resolution of the equation $f(x) = 5 - cos(x)e^x = 0$ is obtained by the instructions:

```
K = 0
X = 0.
DT = 0.0005
CONV = 0.00000000001
DO
K = K + 1
K1 = DT*(5. - COS(X)*EXP(X))
K2 = DT*(5. - COS(X+0.5*K1)*EXP(X+0.5*K1))
K3 = DT*(5. - COS(X+0.5*K2)*EXP(X+0.5*K2))
K4 = DT*(5. - COS(X+K3)*EXP(X+K3))
X = X + (K1+2.*K2+2.*K3+K4)/6.
EQA = 5. - COS(X)*EXP(X)
IF(ABS(EQA)<CONV)EXIT
ENDDO
WRITE(*,*)K,X,EQA
```

After execution, we get:

$K = 1687$

$X = 4.755426629255660$

$EQA = 9.504397269211040\text{E} - 012$

3. The Polynomial Equations

The polynomial equations have a special consideration in science and engineering, because of their considerable use and applicability in practice. The Eigen-values and Eigen-vectors problems or the Optimal Control Matrix Riccati equation are two examples of their applicability. The method proposed herein; present an algorithm to follow for the reason to find all the roots of any polynomial equation, using of course the iteration on the Runge-Kutta method to resolve.

Suppose that we have the arbitrary Polynomial Equation of degree n:

$$P_n(x) = x^n + {b_n}^1 x^{n-1} + {b_{n-1}}^1 x^{n-2} + \cdots + {b_1}^1 = 0 \quad (3)$$

Such that ${b_k}^1$ are the constant coefficients of this equation, and x represent the n roots of $P_n(x) = 0$, and such that the upper index "1" of b_k represent the first polynomial equation given $P_n(x) = 0$.

3.1. Viète Theorem

The theorem that will be formulated in fact cannot find the roots for any polynomial equation; but only formulates the relations between the roots and the constant coefficients of such equation. The Viète theorem is formulates as follows:

The roots of any polynomial equation, like the polynomial given by equation (1), and its coefficients shall be satisfying the following relations:

$$b_{n-k+1} = (-1)^k S_k \ \ (k = 1, 2, \dots n)$$

Where S_k are elementary symmetric functions of $x_1, x_2, \dots x_n$:

$$S_1 = \sum_{i=1}^{n} x_k$$

$$S_2 = \sum_{1 \le i < j}^{n} x_i x_j$$

$$S_3 = \sum_{1 \le i < j < k}^{n} x_i x_j x_k$$

$$\vdots$$

$$S_n = x_1 x_2 x_3 \dots x_n$$

3.2. The Method to Find the Roots

Suppose that we have the polynomial scalar equation of degree n given by (3) or the polynomial matrix equation:

$$P_n(X) = X^n + {B_n}^1 X^{n-1} + {B_{n-1}}^1 X^{n-2} + \cdots + {B_1}^1 = 0 \quad (4)$$

such that ${B_k}^1$ are the matrix constants coefficients, X are the n matrix or vector roots of the matrix equation (4) and n its degree. The upper index B_k has the same definition as b_k.

Suppose that, the scalar d or the matrix or vector D are given by:

$$d = sign({b_n}^1), \quad D = SIGN({B_n}^1) \quad (5)$$

Such that, $sign({b_k}^1)$ is the scalar sign of ${b_k}^1$, and $SIGN({B_k}^1)$ is the matrix sign of ${B_k}^1$, and for a constant coefficient ${B_m}^l$ the outer index l represent the incremental number, though the lower index m represent the number of the constant coefficients in the polynomial equation.

Thus, the first scalar root of $P_n(x)$ and $P_n(X)$, iterating on $i > 1$ until convergence, are given by:

$$\begin{cases} {x_1}^1 = 0 \\ k_1 = DT\left(P_n({x_1}^{i-1})\right) \\ k_2 = DT\left(P_n({x_1}^{i-1} + 0.5k_1)\right) \\ k_3 = DT\left(P_n({x_1}^{i-1} + 0.5k_2)\right) \\ k_4 = DT\left(P_n({x_1}^{i-1} + k_3)\right) \\ {x_1}^i = {x_1}^{i-1} + (-1)^{n+1} d\,(k_1 + 2k_2 + 2k_3 + k_4)/6 \end{cases} \quad (6)$$

And,

$$\begin{cases} {X_1}^1 = 0 \\ K_1 = DT\left(P_n({X_1}^{i-1})\right) \\ K_2 = DT\left(P_n({X_1}^{i-1} + 0.5K_1)\right) \\ K_3 = DT\left(P_n({X_1}^{i-1} + 0.5K_2)\right) \\ K_4 = DT\left(P_n({X_1}^{i-1} + K_3)\right) \\ {X_1}^i = {X_1}^{i-1} + (-1)^{n+1} D\,(K_1 + 2K_2 + 2K_3 + K_4)/6 \end{cases} \quad (7)$$

Where, $DT\ k_l$ and K_l are the same as declared in section 2. ${x_1}^i$ is the first scalar root and ${X_1}^i$ the first matrix or vector root for the i^{th} iteration of the polynomials equations $P_n(x)$ and $P_n(X)$, respectively.

The checking of the convergence is too, expressed in section 2. Replacing $f(x)$ by $P_n(x)$ and $F(x)$ by $P_n(X)$.

For the remaining scalar or matrix or vector roots, we shall to solve the getting polynomial equations, every time after decomposition and decline the preceding root. For the $(k+1)^{th}$ roots $(1 \le k \le n-1)$, the polynomial equations become:

$$P_{n-k}(x) = x^{n-k} + {b_{n-k}}^{k+1} x^{n-k-1} + {b_{n-k-1}}^{k+1} x^{n-k-2} + \cdots + {b_1}^{k+1} = 0 \quad (8)$$

$$P_{n-k}(X) = X^{n-k} + {B_{n-k}}^{k+1} X^{n-k-1} + {B_{n-k-1}}^{k+1} X^{n-k-2} + \cdots + {B_1}^{k+1} = 0 \quad (9)$$

Where,

$$\begin{cases} d = sign\left(x^{n-k} + {b_{n-k-1}}^k x^{n-k-2} + {b_{n-k-2}}^k x^{n-k-3} + \cdots + {b_1}^k\right) \\ {b_{n-k}}^{k+1} = {b_{n-k+1}}^k + x_k \\ {b_{n-k-1}}^{k+1} = {b_{n-k}}^k + {b_{n-k}}^{k+1} x_k \\ {b_{n-k-2}}^{k+1} = {b_{n-k-1}}^k + {b_{n-k-1}}^{k+1} x_k \\ \vdots \\ {b_1}^{k+1} = {b_2}^k + {b_2}^{k+1} x_k \end{cases} \quad (10)$$

We should then, follow the iterative Runge-Kutta algorithm:

$$\begin{cases} {x_{k+1}}^1 = 0 \\ k_1 = DT\left(P_n({x_{k+1}}^{i-1})\right) \\ k_2 = DT\left(P_n({x_{k+1}}^{i-1} + 0.5k_1)\right) \\ k_3 = DT\left(P_n({x_{k+1}}^{i-1} + 0.5k_2)\right) \\ k_4 = DT\left(P_n({x_{k+1}}^{i-1} + k_3)\right) \\ {x_{k+1}}^i = {x_{k+1}}^{i-1} + (-1)^{k+1} d\,(k_1 + 2k_2 + 2k_3 + k_4)/6 \quad for\ k \neq (n-1) \\ x_n = -{b_1}^n = -({b_1}^n + x_{n-1}) \quad for\ k = (n-1) \end{cases} \tag{11}$$

The same algorithm would be get, by changing small letters by capital ones, we obtain:

$$\begin{cases} D = SIGN\left(X^{n-k} + {B_{n-k}}^k X^{n-k-1} + {B_{n-k-1}}^k X^{n-k-2} + \cdots + {B_1}^k\right) \\ {B_{n-k}}^{k+1} = {B_{n-k+1}}^k + X_k \\ {B_{n-k-1}}^{k+1} = {B_{n-k}}^k + {B_{n-k}}^{k+1} X_k \\ {B_{n-k-2}}^{k+1} = {B_{n-k-1}}^k + {B_{n-k-1}}^{k+1} X_k \\ \vdots \\ {B_1}^{k+1} = {B_2}^k + {B_2}^{k+1} X_k \end{cases} \tag{12}$$

$$\begin{cases} {X_{k+1}}^1 = 0 \\ K_1 = DT\left(P_n({X_{k+1}}^{i-1})\right) \\ K_2 = DT\left(P_n({X_{k+1}}^{i-1} + 0.5K_1)\right) \\ K_3 = DT\left(P_n({X_{k+1}}^{i-1} + 0.5K_2)\right) \\ K_4 = DT\left(P_n({X_{k+1}}^{i-1} + K_3)\right) \\ {X_{k+1}}^i = {X_{k+1}}^{i-1} + (-1)^{k+1} d\,(K_1 + 2K_2 + 2K_3 + K_4)/6 \quad for\ k \neq (n-1) \\ X_n = -{B_1}^n = -({B_1}^n + X_{n-1}) \quad for\ k = (n-1) \end{cases} \tag{13}$$

The following FORTRAN listing, represent the algorithm to resolve the polynomial equation:

$$P_n(x) = x^6 - 32x^5 - 33x^4 - 2468x^3 + 9596x^2 - 2400x - 14400 = 0$$

```
N = 6
L = 6
DT = 0.0001
CONV = 0.00000000002
B(1,N) = -32.
B(1,N-1) = -33.
B(1,N-2) = -2468.
B(1,N-3) = 9596.
B(1,N-4) = -2400.
B(1,N-5) = -14400.
D = B1(1,N)

DO I = 1, N - 1

   K = 0
   X = 0.

   SIGNE = (-1)**(L+1)

   DO

      K = K + 1

      K1 = X**L
      DO LL = 1, L
         K1 = K1 + B(I,LL)*(X**(LL-1))
      ENDDO
      K1 = REAL(DT)*K1

      K2 = (X+0.5*K1)**L
      DO LL = 1, L
         K2 = K2 + B(I,LL)*((X+0.5*K1)**(LL-1))
      ENDDO
      K2 = REAL(DT)*K2

      K3 = (X+0.5*K2)**L
      DO LL = 1, L
         K3 = K3 + B(I,LL)*((X+0.5*K2)**(LL-1))
      ENDDO
      K3 = REAL(DT)*K3

      K4 = (X+K3)**L
      DO LL = 1, L
         K4 = K4 + B(I,LL)*((X+K3)**(LL-1))
      ENDDO
      K4 = REAL(DT)*K4

      X=X+SIGNE*SIGN(F,R)*(K1+2.*K2+2.*K3+K4)
/6.
```

```
      EQA = X**L
      DO LL = 1, L
         EQA = EQA + B(I,LL)*(X**(LL-1))
      ENDDO

      IF(ABS(EQA)<CONV)EXIT

   ENDDO

10 WRITE(*,*)K,X,EQA
   IF(K==0)EXIT
   L = L - 1

    D = X**L
    DO LL = 1, L-1
      D = D + A(I,LL)*(X**(LL-1))
    ENDDO

    B(I+1,L) = B(I,L+1) + X
    DO LL = L-1, 1, -1
       B(I+1,LL) = B(I,LL+1) + B(I+1,LL+1)*X
    ENDDO

    IF(I==N-1)THEN
      X = -B(I+1,L)
      K = 0
      EQA = X + B(I+1,L)
      GOTO 10
    ENDIF
  ENDDO
```

The output results of this equation are the following:

$$\begin{cases} K = 23, & X = \ \ 1.999999999999998, & EQA = -1.807620719773695E-011 \\ K = 62, & X = -1.000000000000001, & EQA = -1.942623839568114E-011 \\ K = 43, & X = -12.000000000000000, & EQA = \ \ 1.455191522836685E-011 \\ K = 8233, & X = \ \ 3.999999999999392, & EQA = \ \ 1.983835318242200E-011 \\ K = 8532, & X = \ \ 5.000000000000021, & EQA = -1.999467258428922E-011 \\ K = 0, & X = -29.999999999979420, & EQA = 0 \end{cases}$$

4. Complex Roots

Indeed, that there are be infinite equations that have no real roots; the complex solutions in this case represent the roots of such equations. As examples of these equations $f(x) = cos(x) \pm 2 = 0$, $f(x) = e^x + 1 = 0$, $f(x) = x^2 + 1 = 0$ and so on. The proposed algorithm is also available to solve these equations of complex roots. We have then to declare the equation $f(x)$ or $P_n(x)$, the roots x, the coefficients of the Runge-Kutta method and DT as complex variables, and the method is available for matrix equations or polynomial equations. The resolution of the equation $f(x) = e^x + 7 = 0$ in the FORTRAN code can be listing by the algorithm:

```
  K = 0
  X = 0.
  DT = (0.001,0.001)
  CONV = 0.00000000001

  DO
     K = K + 1
     K1 = DT*(7. + EXP(X))
     K2 = DT*(7. + EXP(X+0.5*K1))
     K3 = DT*(7. + EXP(X+0.5*K2))
     K4 = DT*(7. + EXP(X+K3))

     X = X + (K1+2.*K2+2.*K3+K4)/6.

     EQAC = 7. + EXP(X)

IF(ABS(REAL(EQAC))<CONV.AND.ABS(IMAG(EQAC
))<CONV)EXIT

  ENDDO
 WRITE(6,*)K,X,EQAC
```

The obtained results are:

$K = 4148$

$X = 1.945910149053894 + i3.141592653591149$

$EQA = 9.936940159605001E-012 + i - 9.486664654775595E-012$

5. Numerical Examples

- ***Example 1***

The first example extend and illustrates the variations of the matrix algebraic equation $P_2{}^1(X) = X^2 - A = 0$ and its roots, such that X represent the square-roots matrices of the matrix A. The matrix A is a (5×5) such that all the elements of this matrix $a_{ij} = 2$. The figure 1, Shows the converged variations of the elements of the matrix polynomial equations $P_2(X)$ and $P_1(X)$ and their roots elements versus the iterations numbers. The Table 1, offers the final converged results of $P_2(X)$ and $P_1(X)$ and their roots elements and the numbers of iterations needed for the convergence.

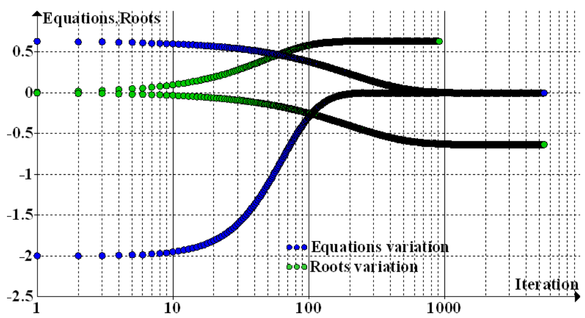

Figure 1. *The equations and roots variations vs. iterations (Example 1)*

- ***Example 2***

In this example we show illustratively and numerically the results of the resolution of the matrix polynomial equation:

$$P_5{}^2(X) = X^5 + B_5X^4 + B_4X^3 + B_3X^2 + B_2X + B_1 = 0$$

Such that all the composed matrices of this equation are (5×5) and, the constant matrices elements are given by:

$$b_{5_{ij}} = 9,\ b_{4_{ij}} = -515,\ b_{3_{ij}} = -11925,\ b_{2_{ij}} = 443250,\ b_{1_{ij}} = -2835000$$

Figure 2, shows the curves of the variations of the initial polynomial equation $P_5(X)$, and the derived polynomials equations elements, versus the numbers of iterations needed for the convergence; though, Figure 3, illustrates the variations of the roots elements. The Table 1, shows the final converged results.

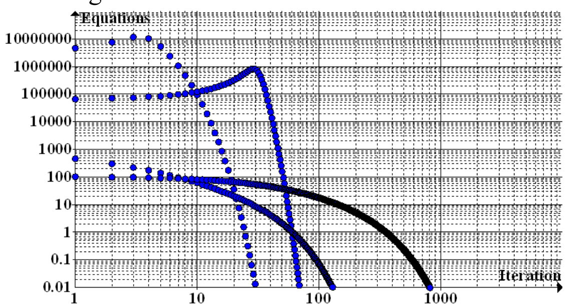

***Figure 2.** The variations of the polynomial matrix equation and the equations derived (Example 2)*

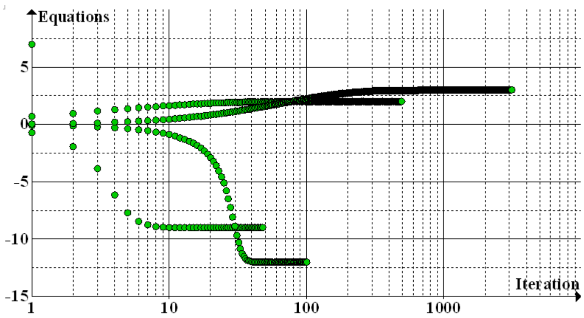

***Figure 3.** The variations of the polynomial matrix roots (Example 2)*

- ***Example 3***

In this example, we resolve the elementary scalar equation $f_1(x) = 5 - e^x cos(x) = 0$. Figure 4 and Table 1, show and illustrate the results.

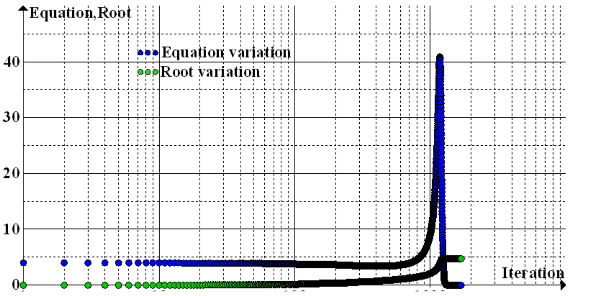

***Figure 4.** The variations of the equation and its root vs. iteration (Example 3)*

- ***Example 4***

We are occupied in this example to solve the scalar equation $f_2(x) = cos(x) - Ln(x) = 0$. The variations of $f(x)$ and x are shown by Figure 5. Although, the final converged results are given by Table 1.

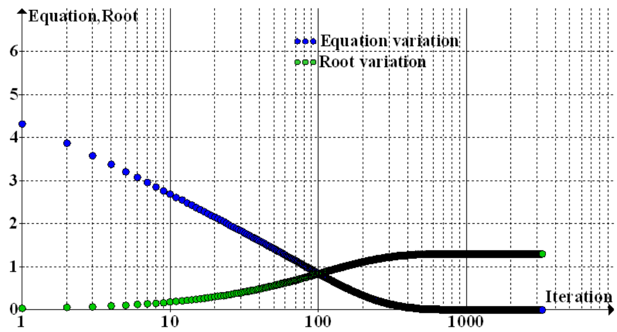

***Figure 5.** The variations of the equation and its root vs. iteration (Example 4)*

***Table 1.** The converged final results of the Examples 1, 2, 3 and 4*

Equations			Roots		Iterations
$P_2{}^1$	$P_2{}^1$	-1.006084104915317E-012	$X_{1_{ij}}$	6.324555320335168E-001	911
	$P_1{}^1$	1.001865257421741E-012	$X_{2_{ij}}$	-6.324555320325200E-001	5407
$P_5{}^2$	$P_5{}^2$	1.862645149230957E-008	$X_{1_{ij}}$	-8.999999999999998	48
	$P_4{}^2$	1.746229827404022E-009	$X_{2_{ij}}$	-12.000000000000000	100
	$P_3{}^2$	4.547473508864641E-013	$X_{3_{ij}}$	1.999999999999932	492
	$P_2{}^2$	1.008970684779342E-012	$X_{4_{ij}}$	3.000000000000030	3122
	$P_1{}^2$	0	$X_{5_{ij}}$	7.000000000000036	0
f_1		9.504397269211040E-012	x	4.755426629255660	1687
f_2		9.919781196552485E-013	x	1.302964001215440	3262

6. Results and Discussion

Indeed, that the results (the roots) of the examples taken of the different algebraic equations are much converged to the exact solutions. As it is shown by the different Figures and the Table 1, the convergence of the elementary functions equations or polynomials can be exceeds in general $1/10^{10}$, and their roots precision reach the order of $1/10^{14}$. As it is shown by Figure 1, and clear by Table 1, that the convergence of the equation approaches to $1/10^{12}$ and the sum of its total iteration number approaches to 6500 such that its two roots represent the square-roots of the matrix A, and the second root can be simply deducted without iterations, using the algorithm and the Fortran listing instructions of section 3. The matrix polynomial equation of degree 5 of the example 2, its convergence alternates from $1/10^{8}$ to $1/10^{12}$ and its total iteration number doesn't exceed 4000. Although, the elementary functions equations of examples 3 and 4, their convergence precision exceeds $1/10^{11}$ with a number of iterations less than 3500. According to the step DT adopted, the number of iterations doesn't exceed 10000, which can take a fraction of a second for the resolution. Moreover, the experience of this algorithm, can allow us to conclude that, when DT increases, the number of iterations decreases considerably. Also, when we would get good precision, we would of course decrease the tolerance of convergence, and this operation doesn't in fact, increase the number of iterations greatly. Although, when the tolerance of convergence decreasing, for the reason to get good precision, we have to take in our account probably that this change leads to decrease the step DT. As it is presented by Table 1, for the polynomials, the numbers of iterations increase considerably from the first equation (the original equation given) to the last equation deducted.

7. Conclusion

As we have indicated that the Runge-Kutta method is a good and precise method for the resolution of ordinary differential equations, and the remark that for constant equations coefficients, the iterative procedure of this method leads to these algebraic equations roots. Although, in the absent of precise algorithms to resolve every real or complex elementary functions or polynomial algebraic equations, the algorithm presented can resolve any algebraic equation with good precision that can regulated as we need such is demonstrated by the arbitrary examples presented (for all the examples presented, the tolerance of convergence at least $< 1/10^{8}$). The presented algorithm and the method presented in section 3, allow us to determine all the roots of any matrix or scalar polynomial equation like the scalar polynomials of the Eigen-values and the matrix equations of the Eigen-vectors. The algorithm presented too, is very simple for programming as presented in sections 2, 3 and 4 and some instructions lines can resolve any simple or complex equation. Moreover, with a personnel computer, any complex equation can be resolved with good precision and as possible, with a small number of iterations. As shown by the Figures presented, that the algorithm automatically refines its roots contributions considerably as possible only and, according to that the exact equation solution is considerably close, and according to the step DT and the convergence number adopted, the feature that make it very precise and economic algorithm as well in the time of solving and in the iterations numbers. Probably that the algorithm proposed as extended, needs some experience about the step DT and the tolerance of the convergence; but presents indeed, a new accurate numerical technique for solving any scalar or matrix algebraic equation with good precision and good execution speed, and the method proposed in section 3, for deducting all the remaining roots of polynomials, when the preceding are computed, is in fact a new idea and a new proposal for solving polynomial equations.

As a future development using this algorithm, we will start solving the problem of proper-frequencies (the Eigen-values) and the mode-shapes (the Eigen-vectors) for the semi-explicit solving of the dynamic equilibrium equation of structures subjected to dynamic arbitrary loading. The algorithm too, can be used to solve the two degree Riccati matrix algebraic equation for the optimal control of dynamic systems.

References

[1] Bird, J., (2010). Higher Engineering Mathematics. Sixth Edition, Elsevier, Ltd.

[2] Ciarlet, P. G., and Lions, J. L., (2000). Handbook of Numerical Analysis. Vol. 7. First Edition, Elsevier Science.

[3] Kiusalaas, J., (2005). Numerical Methods in Engineering with Python. First Edition, Cambridge University Press.

[4] Polyanin, A. D. and Manzhrov, A. V., (2007). Handbook of Mathematics for Engineers and Scientists. First Edition, Taylor and Francis Group, LLC.

[5] Press, W. H. et al., (2007). Numerical Recipes. Third Edition, Cambridge University Press.

[6] Press, W. H. et al., (1997). Numerical Recipes in Fortran 77. Second Edition, Cambridge University Press.

[7] Press, W. H. et al., (1997). Numerical Recipes in Fortran 90. Second Edition, Cambridge University Press.

[8] Riley, K. F. et al., (2006). Mathematical Methods for Physics and Engineering. Third Edition, Cambridge University Press.

[9] Soyeur, A. et al., (2011). Cours de Mathématiques. http://www.les-mathematiques.net

[10] Yang, W. Y., et al., (2005). Applied Numerical Methods using Matlab. First Edition, John-Wiley and Sons, Inc.