STUDIA UNIVERSITATIS BABEŞ-BOLYAI
Engineering 67(1) 2022 DOI: 10.24193/subbeng.2022.1.20

Simulation of electric field lines produced by electric point charges

Marius-Florian Predus, Cristian-Mircea Muscai*, Cornel Hatiegan

Abstract. *The paper proposes o method of solving differential equations using Runge-Kutta method and presents an application made in a Visual programming language that solves two differential equations step by step drawing the graph obtained for two electrically charged particles that interact by their electrical fields.*

Keywords*: differential equations, electrical field, simulation*

1. Introduction

Numerical methods are those techniques that allow the transformation of mathematical models into numerical models (operating on finite spaces) and assume algorithms that can be easily transformed into source codes, using different classical programming languages and, through them, solve real problems with the help of the computer [1], [2].

Numerical calculation methods often require retaining a very large number of terms and developing them in a Taylor series in order to obtain the best accuracy of the results. This is not always possible due to the large number of calculations required for each of the required points, especially if the integration step is constant and of very small value, these limitations being imposed by the memory space and the execution speed of these algorithm calculations [1], [2], [3].

Partial differential equations are often used in mathematically oriented scientific fields such as physics and engineering, aiding in the scientific understanding of phenomena in the fields of electrostatics, electrodynamics, thermodynamics and other physics-related phenomena modeling applications [1], [2], [4].

The integration of differential equations by numerical methods requires the creation of a program that will process the entered data and obtain a discrete string of values that represent an approximation of the exact solution requested. The accuracy of this solution depends on the solving method used but also on the chosen form of the equations subjected to numerical processing [3], [4].

The difficulty of creating computer programs that perform the integration of differential equations is to find a critical ratio between the processing precision and the speed of the program's execution, excluding the limit points where the defined functions do not make sense.

In order to obtain an algorithm as stable as possible, the time interval dt must be small but large enough for the method to be efficient from the point of view of the duration of the calculation [2], [3].

In general, integration methods are divided into two large categories: step-by-step methods and complex methods that take into account several previous values, not just one, as is the case with step-by-step methods [4].

2. The Runge-Kutta method

Runge-Kutta methods are iterative methods for solving ordinary differential equations used in setting initial value problems where we are given a differential equation y′(t)=f(t,y(t)) over a time interval [t0, t1] with a starting point y(t0)=y0.

The Runge-Kutta method of the fourth degree performs four successive approximations, and the time interval dt must be chosen so that the integration is as precise as possible, being the most frequently used in practice [5], [6], [7].

Unlike Euler's Method, which calculates one slope at an interval, Runge-Kutta calculates four different slopes and uses them as weighted averages to better approximate the actual slope, the velocity, of the monitored body [6], [7].

These slopes are commonly named k1, k2, k3 and k4, and must be calculated at each time step. Its position is then calculated using this new slope at each iteration. The Runge-Kutta's formula is defined as [4], [5], [6]:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{1}$$

Where each term is defined as:

$$k_1 = f(x_i, y_i) \tag{2}$$

$$k_2 = f(x_1 + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h) \tag{3}$$

$$k_3 = f(x_i + \frac{1}{2}h, y_i + k_3 h) \tag{4}$$

$$k_4 = f(x_i + h, y_i + k_3 h) \tag{5}$$

Each of the Runge-Kutta terms has a well-defined role in the calculation method used: k1 is the increment based on the slope at the beginning of the interval, k2 is the increment based on the slope at the midpoint of the interval, k3 is the increment based on the midpoint slope and k4 is the increment based on the slope at the end of the interval [5], [6].

The method used has the advantage that during the integration, the step size is adapted so that the estimated error remains below a threshold defined by the user, thus solving the two distinct cases that may appear during the program run, i.e. if the calculated error is too large, it is repeated with a step of a smaller size, and if the error is much too small, the step size is increased to save processing time, thus helping the user who no longer has to identify the size suitable for processing [3], [5], [6].

3. Electric field line modeling application

The developed application calculates the steps required for the successive plotting of the solution for the electrostatic equations for the 3 distinct cases using the Runge-Kutta method of order 4, the standard variant of the method, and 13 necessary constants.

For the case of the two electrostatically charged particles, their modeling is done with the help of two differential equations [8] which have as boundary conditions x(s0) = x0 and y(s0) = y0 :

$$\frac{dx}{ds} = \frac{-E_y}{\sqrt{E_x^{\,2} + E_y^{\,2}}} \tag{6}$$

$$\frac{dx}{ds} = \frac{-E_y}{\sqrt{E_x^{\,2} + E_y^{\,2}}} \tag{7}$$

The first case is presented in Figure 1 where we considered two electrostatically charged point particles with charges of opposite signs, in which case the equipotential field lines and the field lines that close between the two electrostatically charged particles are drawn.

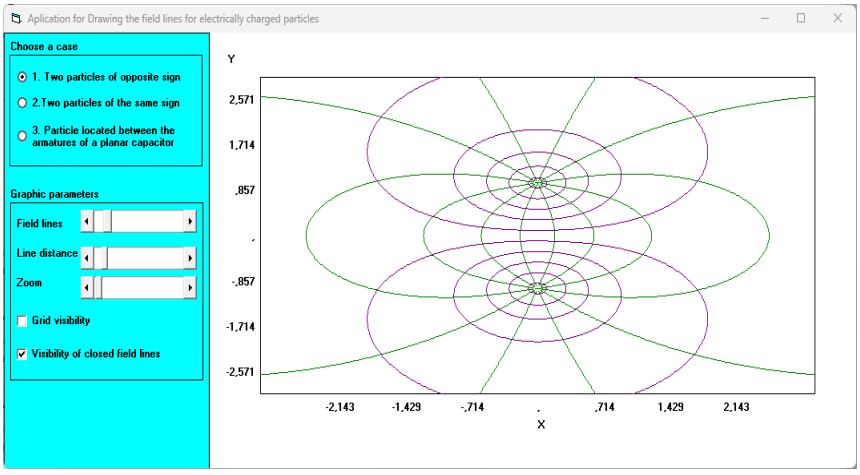

Figure 1. The two particles have opposite signs

The second case is somewhat similar, but the potentials of the two particles differ in the sense that they are now of the same sign, which can also be seen in Figure 2 where their field lines repel each other.

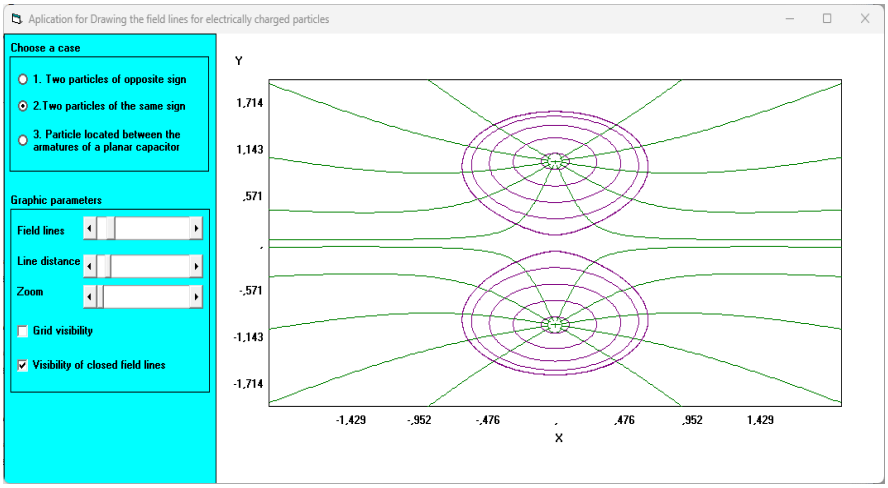

Figure 2. The two particles have same signs

The third case requires placing a single electrostatically charged particle between the armatures of a planar capacitor, as seen in Figure 3, and the field lines are drawn for this case, with the user being able to change the distance of the particle from the capacitor arms, represented as two parallel lines in the graphic.

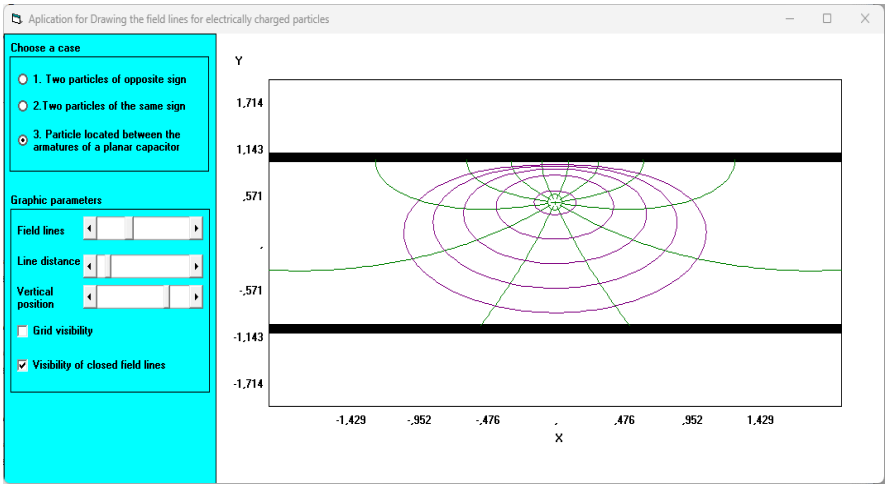

Figure 3. The particle placed between capacitor's armatures

The application allows changing the number of equipotential lines displayed, the size of the drawn plane, and the distance between successive field lines. For better visibility, the application window can be resized and the lines on the display grid can be hidden.

We often come across simulation applications made for didactic purposes using common programming languages in different technical fields [9], [10].

The program is made in the Visual Studio development environment, with the help of the Visual Basic programming language, in the form of a standard executable that allows it to run under Windows operating systems [11], [12]. The application consists of a module containing the graphics routines and a main window named Form1, on which several graphics controls are placed, including a PictureBox control that represents the display surface for graphic functions, and several HscrollBar controls for setting various display parameters like zoom or number of lines drawn.

To calculate the grid lines and the values displayed on the graph axes, we use a graphic routine called "Draw" which receives as values the limits, minimum and maximum values of the axes and the Form window that contains the graphic control on which the respective lines are drawn [13], [14].

The three cases are selected using a set of three RadioButton controls that allow individual selection of the drawn cases.

The starting point for the application is the main form. From the main form, based on our selected option, one of the two routines, "Afișez" or "Afisez1", is called, allowing the calculation of the derivatives and the point-by-point plotting of the graphs resulting from different scaling for each type of curve drawn on the screen [15].

```
Sub rezolv(ByRef hr As Double, ByRef x As Double, ByRef y_par() As Double)
Dim rk1(1 To 20) As Double
Dim rk2(1 To 20) As Double
Dim rk3(1 To 20) As Double
Dim rk4(1 To 20) As Double
Dim ryl(1 To 20) As Double

Form1.sist y_par(), rk1()
For j = 1 To 2
    rk1(j) = hr * rk1(j)
    ryl(j) = y_par(j) + rk1(j) / 2
Next j

Form1.sist ryl(), rk2()
For j = 1 To 2
    rk2(j) = hr * rk2(j)
    ryl(j) = y_par(j) + rk2(j) / 2
Next j

Form1.sist ryl(), rk3()
For j = 1 To 2
    rk3(j) = hr * rk3(j)
    ryl(j) = y_par(j) + rk3(j)
Next j

Form1.sist ryl(), rk4()
For j = 1 To 2
    rk4(j) = hr * rk4(j)
    y_par(j) = y_par(j) + (rk1(j) + 2 * (rk2(j) + rk3(j)) + rk4(j)) / 6
Next j
```

Figure 4. The routine that implements the Runge-Kutta method

The "Rezolv" subroutine shown in Figure 4 is the one that helps to implement the 4th order Runge-Kutta algorithm, which successively calculates k1, k2, k3 and k4, returning the new values to the calling routine with the help of the parameters passed by reference to it. To avoid getting stuck, the algorithm calculates, at each step, a reference value that represents the sum of the squared dx and dy. If this value is lower than a very small imposed limit, it stops the program because the desired precision has been reached or it has reached a point of singularity from where the created algorithm can no longer exit. The program can calculate only in one direction, based on the values already memorized, so a dead lock point can't be exited otherwise [11], [12].

The types of the variables used in the program are Double precision to keep the accuracy of the processed data within reasonable limits due to the small quantities used to model the two charged particles [10], [11], [12].

4. Conclusions

The advantage of implicit Runge-Kutta methods over explicit methods is their greater stability, especially when stiff equations are applied.

The application allows drawing of all existing field lines between electrostatically charged particles, both of the equipotential type that always have a closed contour, and of the others that originate in the vicinity of positive electric charges and close to negative electric charges.

The proposed application has a simple and intuitive interface and the results obtained can represent the basis of an understanding of the way of drawing the field lines and the way of interaction between electrically charged particles.

References

[1] Demsoreanu B., *Metode numerice cu aplicații in fizică*, Editura Academiei-Române, 2005

[2] Sorea, D., Lungoci, C., Scutaru G., *Metode numerice cu aplicații în ingineria electrică. Curs aplicativ*, Editura Universității Transilvania Braşov, 2009.

[3] Burrage K., *Parallel and Sequential Methods for Ordinary Differential Equations, Numer. Math. Sci. Comput*., Oxford University, New York, 1995.

[4] Butcher J.C., *Numerical Methods for Ordinary Differential Equations*, 2nd ed., John Wiley & Sons, Chichester, 2003.

[5] Shampine L.F., *Numerical Solution of Ordinary Differential Equations*, Chapman & Hall, New York, 1994.

[6] Ketcheson D.I., BinWaheed U., A comparison of high-order explicit Runge-Kutta, extrapolation, and deferred correction methods in serial and parallel, *Communications in applied mathematics and computational science*, 9(2), 2014, pp.175 – 200.

[7] Scutaru G., *Metode numerice*, Editura Universității Transilvania Braşov, 2003.
[8] Haţiegan C., Suciu L., *Fizică Tehnologică. Teorie şi aplicaţii*, Editura Eftimie Murgu, Resita, 2010/
[9] Hatiegan C., Stroia M.-D., Popescu C., Muscai C.-M., Application for Simulating and Analysis of a Serial R-L-C Circuit, *Analele Universităţii Constantin Brâncuşi din Târgu-Jiu - Seria Inginerie*, 3, 2020.
[10] Stroia M.-D., Hatiegan C., Muscai C.-M., Simulating an improved algorithm for propagation of transverse oscillations through a string, *Studia Universitatis Babeş-Bolyai Engineering*, 65(1), 2020.
[11] Schneider D., *An Introduction to Programming Using Visual Basic*, 9th Edition, Pearson, 2013, USA.
[12] Newsome B., *Beginning Visual Basic 2015*, Wrox, USA, 2015.
[13] Alexander R., Diagonally implicit Runge–Kutta methods for stiff O.D.E.s, *SIAM J. Numer. Anal*. 14(6), 1977, pp. 1006–1021.
[14] Zlatev Z., Dimov I., Faragó I., Havasi Á., *Richardson Extrapolation: Practical Aspects and Applications, De Gruyter Ser. Appl. Numer. Math. 2*, De Gruyter, Berlin, 2018.
[15] Durham Jr. H.L., Francis Jr. O.B., Gallaher L.J., Hale Jr. H.G., Perlin I.E., *Final Report: Study of methods for the numerical solution of ordinary differential equations:* Final report. 9 Nov. 1963 - 8 Nov. 1964. NASA, Huntsville, AL, Georgia Institute of Technology, Engineering Experiment Station, Atlanta, Georgia, 1965.

Adresses:

- Dr. Eng. Marius-Florian Predus, Babeş-Bolyai University, Faculty of Engineering, Piaţa Traian Vuia, nr. 1-4, 320085, Reşiţa, Romania
  marius.predus@ubbcluj.ro
- PhD. Stud. Eng. Cristian-Mircea Muscai, Polytechnic University of Bucharest, Faculty of Electrical Engineering, Splaiul Independenței 313, 060042, București, Romania
  muscaicristian@yahoo.com
  (**corresponding author*)
- Lect. Dr. Eng Cornel Hatiegan, Babeș-Bolyai University, Faculty of Engineering, Piaţa Traian Vuia, nr. 1-4, 320085, Reşiţa, Romania
  cornel.hatiegan@ubbcluj.ro