Society for Science and Education United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 2, Issue 2
ISSN: 2054 - 7390

# Image Learning of Charge Motion in Electric and Magnetic Fields by Java Programming

**Masami Morooka[1], Suhua Qian[1], Midori Morooka[2]**
[1] *Department of Electrical Engineering, Fukuoka Inst. Tech,higashi-ku, Fukuoka 811-0295, Japan;*
[2] *Flash Design Center, Micron Japan, P. O. Box 57, Kamata 5-37-1, Ota-ku, Tokyo, Japan;*

**ABSTRACT**

Java programs in a GUI environment have been developed for an image learning of charge motions in electric and magnetic fields. Text fields of selected parameters for the numerical calculation of the differential equations of the charge motion, such as the electric field and magnetic field, are set on the display, and the calculation by Runge-Kutta method is begun by clicking the start button after inputting values into the text fields. The calculated results are plotted immediately after the completion of the calculation as a figure on the display, e.g., a charge locus for the simulation of charge motion. By changing the values in the text fields, new results can be represented immediately, and the charge motion under a new electric and magnetic fields can be easily simulated. These Java programs are useful in education applications for rapidly and accurately image learning for the phenomena expressed by ordinary differential equations.

**Keywords:** Image learning, charge motion in electric and magnetic fields, Java programming, ordinary differential equations.

## 1 INTRODUCTION

In many situations, it is often difficult to obtain the charge motion, e. g. the charge locus, in electric and magnetic fields, and it is even more difficult to visualize the variation of the motion due to the change in the fields. The charge motion can be expressed by ordinary differential equations which can be solved numerically by Runge-Kutta method. With the wide use of Java programming and the rapid development of personal computers, the author has recently directly solved the partial differential equations and has shown that the complicated diffusion of Au into Si can be simulated easily by the Java programming [1]. In this paper, the ordinary differential equations for a charge motion in electric and magnetic fields have been solved numerically, and the locus of the charge motion in the fields can be easily and accurately simulated using Java in a GUI (graphical user interface) environment.

**DOI:** 10.14738/tmlai.22.90
**Publication Date:** 11th April 2014
**URL:** http://dx.doi.org/10.14738/tmlai.22.90

## 2 NUMERICAL METHOD FOR CHARGE MOTION IN ELECTRIC AND MAGNETIC FIELDS

### 2.1 Basic Equations for Charge Motion in Electric and Magnetic Fields

We simulate the motion of a charge $q$ in an applied electric field **E** = ($E_x$, $E_y$, $E_z$) and an applied magnetic field B = ($B_x$, $B_y$, $B_z$). The charge motion is generally affected by the resistance force based on collisions and by the restoring Coulomb force in addition to the electric and electro-magnetic forces. The charge motion under these forces is given as

$$m\frac{d\mathbf{v}}{dt} = -a\mathbf{v} + q\mathbf{E} + q\mathbf{v}\times\mathbf{B} - b(\mathbf{r}-\mathbf{r}_{\mathrm{R0}})\,. \tag{1}$$

Here, $m$, v = ($v_x$, $v_y$, $v_z$), and r = ($x$, $y$, $z$) are the mass, velocity, and displacement of the charge, respectively. $a$ and $b$ are coefficients of the resistance and restoring forces. $r_{R0}$= ($x_{R0}$, $y_{R0}$, $z_{R0}$) is the restoring center, and

$$\frac{d\mathbf{r}}{dt} = \mathbf{v}\,. \tag{2}$$

By decomposing the vector equations (7) and (8) into the scalar equations in the $x$, $y$, and $z$ directions, we obtain

$$m\frac{dv_x}{dt} = -av_x + qE_x + q(v_yB_z - v_zB_y) - b(x - x_{\mathrm{R0}})\,, \tag{3}$$

$$m\frac{dv_y}{dt} = -av_y + qE_y + q(v_zB_x - v_xB_z) - b(y - y_{\mathrm{R0}})\,\square \tag{4}$$

$$m\frac{dv_z}{dt} = -av_z + qE_z + q(v_xB_y - v_yB_x) - b(z - z_{\mathrm{R0}})\,, \tag{5}$$

$$\frac{dx}{dt} = v_x\,, \tag{6}$$

$$\frac{dy}{dt} = v_y\,, \tag{7}$$

$$\frac{dz}{dt} = v_z\,. \tag{8}$$

In the case of the magnetic field having a fixed direction, for example, B = (0,0,$B_z$), we obtain the following equations, which are simpler than equations (3) - (5).

$$m\frac{dv_x}{dt} = -av_x + qE_x + qv_yB_z - b(x - x_{\mathrm{R0}})\,, \tag{9}$$

$$m\frac{dv_y}{dt} = -av_y + qE_y - qv_xB_z - b(y - y_{\mathrm{R0}})\,, \tag{10}$$

$$m\frac{dv_z}{dt} = -av_z + qE_z - b(z - z_{\mathrm{R0}})\,. \tag{11}$$

In this case, we can obtain the charge motion in the *x* and *y* directions by solving four ordinary differential equations ((6), (7), (9), and (10)) and the motion in the *z* direction by solving two ordinary differential equations ((8) and (11)).

## 2.2 Runge-Kutta Method

The ordinary differential equations can be solved numerically using the fourth-order Runge-Kutta method. When differential equations for dependent variables ($y_1(t)$, $y_2(t)$, •, •, •, $y_n(t)$) of the independent variable, *t*, are given, such as $dy_1(t)/dt = f_1(t, y_1(t), y_2(t), \bullet, \bullet, \bullet, y_n(t))$, $dy_2(t)/dt = f_2(t, y_1(t), y_2(t), \bullet, \bullet, \bullet, y_n(t))$, •, •, •, $dy_n(t)/dt = f_n(t, y_1(t), y_2(t), \bullet, \bullet, \bullet, y_n(t))$, the first-order increment functions are given as

$$k_1^{(1)} = f_1(t, y_1, y_2, \bullet, \bullet, \bullet, y_n)\,, \tag{12}$$

$$k_2^{(1)} = f_2(t, y_1, y_2, \bullet, \bullet, \bullet, y_n)\,, \tag{13}$$

•

•

$$k_n^{(1)} = f_n(t, y_1, y_2, \bullet, \bullet, \bullet, y_n)\,, \tag{14}$$

and the second-order increment functions as

$$k_1^{(2)} = f_1(t + \frac{h}{2}, y_1 + \frac{hk_1^{(1)}}{2}, y_2 + \frac{hk_2^{(1)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(1)}}{2})\,, \tag{15}$$

$$k_2^{(2)} = f_2(t + \frac{h}{2}, y_1 + \frac{hk_1^{(1)}}{2}, y_2 + \frac{hk_2^{(1)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(1)}}{2})\,, \tag{16}$$

•

•

$$k_n^{(2)} = f_n(t + \frac{h}{2}, y_1 + \frac{hk_1^{(1)}}{2}, y_2 + \frac{hk_2^{(1)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(1)}}{2})\,, \tag{17}$$

and the third-order increment functions as

$$k_1^{(3)} = f_1(t + \frac{h}{2}, y_1 + \frac{hk_1^{(2)}}{2}, y_2 + \frac{hk_2^{(2)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(2)}}{2})\,, \tag{18}$$

$$k_2^{(3)} = f_2(t + \frac{h}{2}, y_1 + \frac{hk_1^{(2)}}{2}, y_2 + \frac{hk_2^{(2)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(2)}}{2})\,, \tag{19}$$

•

•

$$k_n^{(3)} = f_n(t + \frac{h}{2}, y_1 + \frac{hk_1^{(2)}}{2}, y_2 + \frac{hk_2^{(2)}}{2}, \bullet, \bullet, \bullet, y_n + \frac{hk_n^{(2)}}{2})\,, \tag{20}$$

and the fourth-order increment functions as

$$k_1^{(4)} = f_1(t + h, y_1 + hk_1^{(3)}, y_2 + hk_2^{(3)}, \bullet, \bullet, \bullet, y_n + hk_n^{(3)})\,, \tag{21}$$

$$k_2^{(4)} = f_2(t+h, y_1 + hk_1^{(3)}, y_2 + hk_2^{(3)}, \bullet, \bullet, \bullet, y_n + hk_n^{(3)}), \tag{22}$$

$$\bullet$$

$$\bullet$$

$$k_n^{(4)} = f_n(t+h, y_1 + hk_1^{(3)}, y_2 + hk_2^{(3)}, \bullet, \bullet, \bullet, y_n + hk_n^{(3)}). \tag{23}$$

Here, $h$ is the increment of $t$. The variables at $t + h$ are given as

$$y_1(t+h) = y_1(t) + \frac{1}{6}(k_1^{(1)} + 2k_1^{(2)} + 2k_1^{(3)} + k_1^{(4)}), \tag{24}$$

$$y_2(t+h) = y_2(t) + \frac{1}{6}(k_2^{(1)} + 2k_2^{(2)} + 2k_2^{(3)} + k_2^{(4)}), \tag{25}$$

$$\bullet$$

$$\bullet$$

$$y_n(t+h) = y_n(t) + \frac{1}{6}(k_n^{(1)} + 2k_n^{(2)} + 2k_n^{(3)} + k_n^{(4)}). \tag{26}$$

If the variables at a $t$ are given, the numerical values at $t + h$ can be obtained from equations (24) – (26), then the values at $t + 2h$, at $t + 3h$, etc. by repeating the calculations.

## 2.3 Numerical Calculation of Charge Motion

For the charge motion in the $x$ and $y$ directions, the variables are $x(t)$, $y(t)$, $v_x(t)$, and $v_y(t)$, and the differential equations are equations (6), (7), (9), and (10). For the charge motion in the $z$ direction, the variables are $z(t)$ and $v_z(t)$, and the differential equations are equations (8) and (11). Here, we will discuss the case of motion in the $x$-$y$ plane only. We obtain the incremental functions for the charge motion

$$k_1^{(1)} = v_{x,j}, \tag{27}$$

$$k_2^{(1)} = v_{y,j}, \tag{28}$$

$$k_3^{(1)} = -c_1 v_{x.j} + c_3 E_x(t) + c_3 v_{y.j} B_z(t) - c_2(x_j - x_{\mathrm{R}0}), \tag{29}$$

$$k_4^{(1)} = -c_1 v_{y.j} + c_3 E_y(t) - c_3 v_{x.j} B_z(t) - c_2(y_j - y_{\mathrm{R}0}), \tag{30}$$

$$k_1^{(2)} = v_{x,j} + \frac{hk_3^{(1)}}{2}, \tag{31}$$

$$k_2^{(2)} = v_{y,j} + \frac{hk_4^{(1)}}{2}, \tag{32}$$

$$k_3^{(2)} = -c_1\left(v_{x.j} + \frac{hk_3^{(1)}}{2}\right) + c_3 E_x(t + \frac{h}{2}) + c_3\left(v_{y.j} + \frac{hk_4^{(1)}}{2}\right) B_z(t + \frac{h}{2}) - c_2\left(x_j + \frac{hk_1^{(1)}}{2} - x_{\mathrm{R}0}\right), \tag{33}$$

$$k_4^{(2)} = -c_1\left(v_{y.j} + \frac{hk_4^{(1)}}{2}\right) + c_3E_y(t+\frac{h}{2}) - c_3\left(v_{x.j} + \frac{hk_3^{(1)}}{2}\right)B_z(t+\frac{h}{2}) - c_2\left(y_j + \frac{hk_2^{(1)}}{2} - y_{R0}\right), \tag{34}$$

$$k_1^{(3)} = v_{x,j} + \frac{hk_3^{(2)}}{2}, \tag{35}$$

$$k_2^{(3)} = v_{y,j} + \frac{hk_4^{(2)}}{2}, \tag{36}$$

$$k_3^{(3)} = -c_1\left(v_{x.j} + \frac{hk_3^{(2)}}{2}\right) + c_3E_x(t+\frac{h}{2}) + c_3\left(v_{y.j} + \frac{hk_4^{(2)}}{2}\right)B_z(t+\frac{h}{2}) - c_2\left(x_j + \frac{hk_1^{(2)}}{2} - x_{R0}\right), \tag{37}$$

$$k_4^{(3)} = -c_1\left(v_{y.j} + \frac{hk_4^{(2)}}{2}\right) + c_3E_y(t+\frac{h}{2}) - c_3\left(v_{x.j} + \frac{hk_3^{(2)}}{2}\right)B_z(t+\frac{h}{2}) - c_2\left(y_j + \frac{hk_2^{(2)}}{2} - y_{R0}\right), \tag{38}$$

$$k_1^{(4)} = v_{x,j} + hk_3^{(3)}, \tag{39}$$

$$k_2^{(4)} = v_{y,j} + hk_4^{(3)}, \tag{40}$$

$$k_3^{(4)} = -c_1\left(v_{x.j} + hk_3^{(3)}\right) + c_3E_x(t+h) + c_3\left(v_{y.j} + hk_4^{(3)}\right)B_z(t+h) - c_2\left(x_j + hk_1^{(3)} - x_{R0}\right), \tag{41}$$

$$k_4^{(4)} = -c_1\left(v_{y.j} + hk_4^{(3)}\right) + c_3E_y(t+h) - c_3\left(v_{x.j} + hk_3^{(3)}\right)B_z(t+h) - c_2\left(y_j + hk_2^{(3)} - y_{R0}\right). \tag{42}$$

Ηερε, $\chi_1 = \alpha/\mu$, $\chi_2 = \beta/\mu$, ανδ $\chi_3 = \theta/\mu$. Τηε συβσχριπτ $\varphi$ ρεπρεσεντσ τηε κνοων ϖαριαβλεσ ατ $\tau$. Τηε υνκνοων ϖαριαβλεσ $\xi_{\varphi+1}$, $\psi_{\varphi+1}$, $\varpi_{\xi,\varphi+1}$, ανδ $\varpi_{\psi,\varphi+1}$ ατ $\tau + \eta$ αρε γιϖεν ασ

$$x_{j+1} = x_j + \frac{1}{6}(k_1^{(1)} + 2k_1^{(2)} + 2k_1^{(3)} + k_1^{(4)}), \tag{43}$$

$$y_{j+1} = y_j + \frac{1}{6}(k_2^{(1)} + 2k_2^{(2)} + 2k_2^{(3)} + k_2^{(4)}), \tag{44}$$

$$v_{x,j+1} = v_{x,j} + \frac{1}{6}(k_3^{(1)} + 2k_3^{(2)} + 2k_3^{(3)} + k_3^{(4)}), \tag{45}$$

$$v_{y,j+1} = v_{y,j} + \frac{1}{6}(k_4^{(1)} + 2k_4^{(2)} + 2k_4^{(3)} + k_4^{(4)}). \tag{46}$$

## 2.4 Reflection at a Boundary

When the charge reflects at a wall, the perpendicular component of the velocity against the wall changes into the inverse value and the parallel component does not change. That is, $v_y$ changes into $-v_y$ at $x$-plane and $v_x$ changes into $-v_x$ at $y$-plane. In the case of the reflection at $(x,y)$ on a cylindrical wall for $z$-direction, the perpendicular component of the velocity, $v_T$ , and the parallel component, $v_P$ , are

$$v_{\mathrm{T}} = v_x \frac{x-x_0}{R} + v_y \frac{y-y_0}{R}, \tag{47}$$

$$v_{\mathrm{P}} = v_x \frac{y-y_0}{R} - v_y \frac{x-x_0}{R}. \tag{48}$$

Here, *R* and ($x_0$,$y_0$) are the radius and center of the cylinder, respectively. If $\sqrt{(x-x_0)^2+(y-y_0)^2} \geq R$, $v_{\mathrm{T}}$ changes into $-v_{\mathrm{T}}$, and the new velocities after the reflection are obtained as

$$v_x = (-v_{\mathrm{T}}) \frac{x-x_0}{R} + v_{\mathrm{P}} \frac{y-y_0}{R}, \tag{49}$$

$$v_y = (-v_{\mathrm{T}}) \frac{y-y_0}{R} - v_{\mathrm{P}} \frac{x-x_0}{R}. \tag{50}$$

## 3 PROGRAMMING

In a numerical calculation, values alternating with time, such as electric field and magnetic field are more conveniently expressed as cosine functions instead of sine functions, for example $E_x = E_{x0}\cos(2\pi f_{\mathrm{E}} t)$ and $B_z = B_0\cos(2\pi f_{\mathrm{B}} t)$, for the calculation at $f_{\mathrm{E}} = 0$ and $f_{\mathrm{B}} = 0$. Here, $f_{\mathrm{E}}$ and $f_{\mathrm{B}}$ are frequencies of the electric and magnetic fields, respectively. If we use $E_x = E_{x0}\sin(2\pi f_{\mathrm{E}} t)$ and $B_z = B_0\sin(2\pi f_{\mathrm{B}} t)$, then $E_x = 0$ at $f_{\mathrm{E}} = 0$ and $B_z = 0$ at $f_{\mathrm{B}} = 0$ despite the existence of direct electric and magnetic fields.

In our basic numerical calculation, nine text fields for coefficient of the resistance force, *a*, coefficient of the restoring Coulomb force, *b*, absolute values of $E_x$, $E_y$, and $B_z$, that are $E_{x0}$, $E_{y0}$ and $B_0$, frequencies of electric and magnetic fields, $f_{\mathrm{E}}$ and $f_{\mathrm{B}}$, increment of time, *h*, and number of calculations, *n*, are set on the display. In addition them, more text fields are set for the motion in a boundary, such as the position and size of the boundary, for example, the radius and center of the cylinder, *R* and ($x_0$,$y_0$). Values of *a* and *b* in the text-fields are normalized by *q* due to same effective order with electric and magnetic fields.

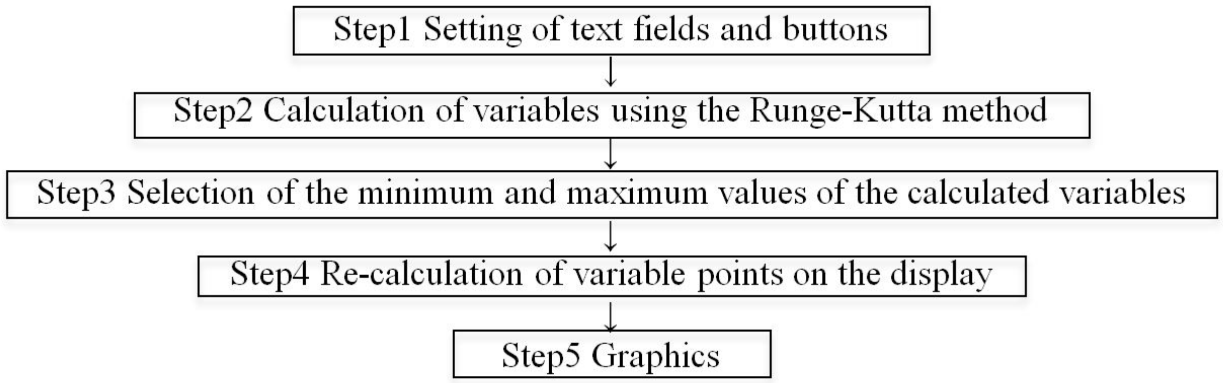
When the calculated results are presented on a display, the value of the *y*-direction is plotted on the lower part on the display; in other words, the *y*-direction on the display is opposite to the real *y*-direction. Therefore, to plot the point of the variables ($x(t)$, $y(t)$) within the area between ($X_{\min}$, $Y_{\min}$) and ($X_{\max}$, $Y_{\max}$) on the display, and to plot the point ($x_{\min}$, $y_{\max}$) at ($X_{\min}$, $Y_{\min}$), the points of the variables on the display, $x_{\mathrm{D}}(t)$ and $y_{\mathrm{D}}(t)$, should be re-calculated as follows:

$$x_{\mathrm{D}}(t) = X_{\min} + (X_{\max} - X_{\min}) \frac{x(t) - x_{\min}}{x_{\max} - x_{\min}}, \tag{51}$$

$$y_{\mathrm{D}}(t) = Y_{\min} + (Y_{\max} - Y_{\min}) \frac{y_{\max} - y(t)}{y_{\max} - y_{\min}}. \tag{52}$$

Here, the subscripts $_{\min}$ and $_{\max}$ represent the minimum and maximum values, respectively. In this case, the points $x_{\min}$ and $x_{\max}$ are plotted at $X_{\min}$ and $X_{\max}$, respectively, and the points $y_{\min}$ and $y_{\max}$ are plotted at $Y_{\max}$ and $Y_{\min}$, respectively. Therefore, the scale for $x$-direction, $(x_{\max} - x_{\min})/(X_{\max} - X_{\min})$ is different from that for $y$-direction, $(y_{\max} - y_{\min})/(Y_{\max} - Y_{\min})$, on the display. The charge motion can be plotted as a same scale for $x$ and $y$ directions by using their smaller value for $(X_{\max} - X_{\min})$ or $(Y_{\max} - Y_{\min})$ and using their larger value for $(x_{\max} - x_{\min})$ or $(y_{\max} - y_{\min})$ in Eqs. (51) and (52).

The main flow of the programming is shown below.

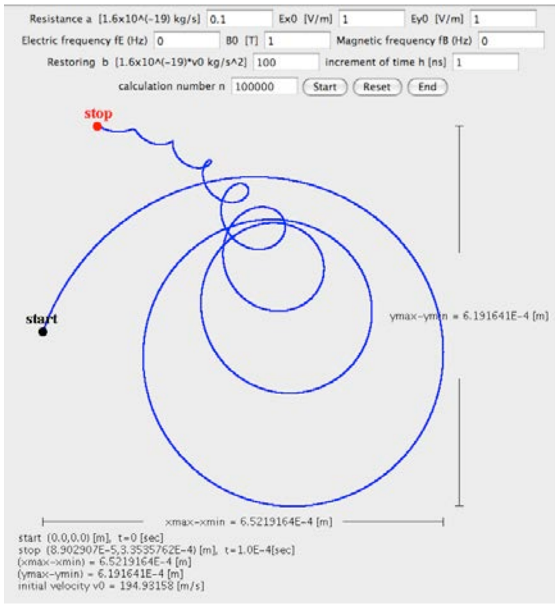

Step1 Setting of text fields and buttons
Step2 Calculation of variables using the Runge-Kutta method
Step3 Selection of the minimum and maximum values of the calculated variables
Step4 Re-calculation of variable points on the display
Step5 Graphics

## 4 RESULTS

The text fields are immediately presented on the display after the execution of the program by the applet viewer form of Java, and the calculation is initiated by clicking the start button after inputting values in the text fields. The charge motion is plotted immediately after the completion of the calculation. The time needed to display the charge motion after clicking the start button is usually less than several seconds depending on the number of calculations and the performance of computer. By changing the values in the text fields and clicking again the start button, new charge motion can be obtained immediately.

### 4.1 Charge Motion in No-Boundary

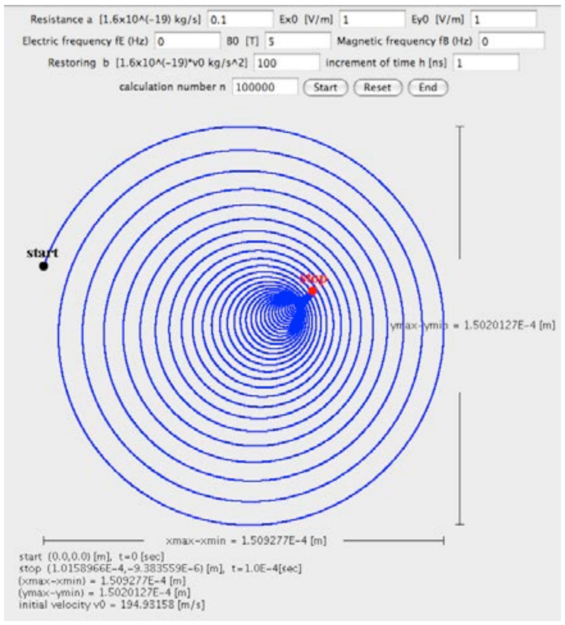
A typical ion motion in which the forces of the resistance, electric and magnetic fields. and restoring are all effective is shown in Fig. 1. The figure is shown as an image on the display. By increasing the value in the text field of $B_0$ by a factor of 5, a new locus in which the motion is strongly affected by the magnetic field, such as in Larmor motion, can be immediately obtained, as shown in Fig. 2. By increasing the value in the text field of $b$ by a factor of 10, a new locus in which the motion is strongly affected by the restoration can also be immediately obtained, as shown in Fig. 3. By increasing the value in the text field of $a$, $E_{x0}$, or $E_{y0}$ by a factor of 10, a new locus in which the motion is strongly affected by the resistance or each electric field can be immediately obtained also. The point of start and stop, the values of the motion range for $x$- and $y$-directions, the final time at the stop point, and the initial velocity of the charge are also shown on the display. By changing the values in the text fields and clicking again the start button, the change of the specific motion affected by each forces are recognized easily and correctly as an image of the locus of the motion on the display.



```
Resistance a [1.6x10^(-19) kg/s] 0.1     Ex0 [V/m] 1     Ey0 [V/m] 1
Electric frequency fE (Hz) 0     B0 [T] 1     Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 100     increment of time h (ns) 1
calculation number n 100000
```

```
xmax-xmin = 6.5219164E-4 [m]
ymax-ymin = 6.191641E-4 [m]
start (0.0,0.0) [m], t=0 [sec]
stop (8.902907E-5,3.3535762E-4) [m], t=1.0E-4[sec]
(xmax-xmin) = 6.5219164E-4 [m]
(ymax-ymin) = 6.191641E-4 [m]
initial velocity v0 = 194.93158 [m/s]
```

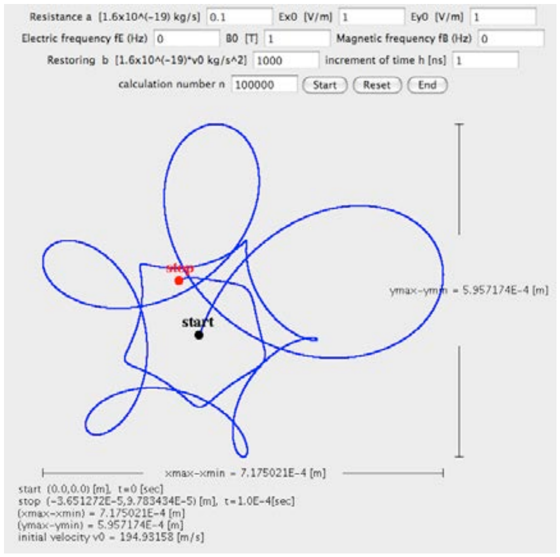
**Figure 1. Typical ion motion in which all forces are effective. The values used for the calculation are shown in the text fields. The distance, final time, and initial velocity are also shown in the figure.**



```
Resistance a [1.6x10^(-19) kg/s] 0.1     Ex0 [V/m] 1     Ey0 [V/m] 1
Electric frequency fE (Hz) 0     B0 [T] 5     Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 100     increment of time h (ns) 1
calculation number n 100000
```

```
xmax-xmin = 1.509277E-4 [m]
ymax-ymin = 1.5020127E-4 [m]
start (0.0,0.0) [m], t=0 [sec]
stop (1.0158966E-4,-9.383559E-6) [m], t=1.0E-4[sec]
(xmax-xmin) = 1.509277E-4 [m]
(ymax-ymin) = 1.5020127E-4 [m]
initial velocity v0 = 194.93158 [m/s]
```

**Figure 2. Updated motion affected strongly by the magnetic field $B_0$ after increasing by a factor of 5 relative to that in Fig. 1.**



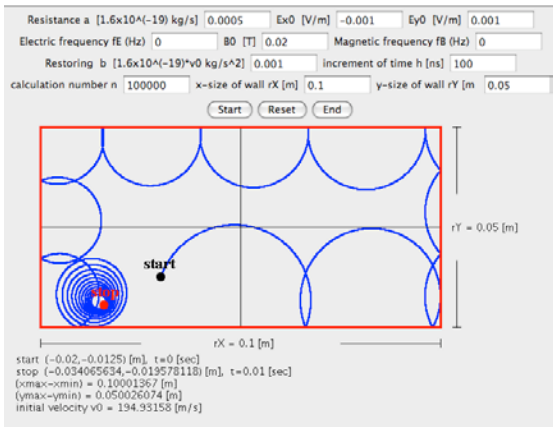
Resistance a [1.6x10^(-19) kg/s] 0.1 Ex0 [V/m] 1 Ey0 [V/m] 1
Electric frequency fE (Hz) 0 B0 [T] 1 Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 1000 increment of time h [ns] 1
calculation number n 100000

start (0.0,0.0) [m], t=0 [sec]
stop (-3.651272E-5,9.783434E-5) [m], t=1.0E-4[sec]
(xmax-xmin) = 7.175021E-4 [m]
(ymax-ymin) = 5.957174E-4 [m]
initial velocity v0 = 194.93158 [m/s]

**Figure 3. Updated motion affected strongly by the restoration *b* after increasing by a factor of 10 relative to that in Fig. 1.**

## 4.2 Charge Motion in Quadrilateral Boundary

A typical ion motion in a quadrilateral boundary, which size is 10 cm x 5 cm, is shown in Fig. 4. The reflection at the wall is clearly shown as an image on the display. By changing the values in the text fields, charge motions in the different size of the boundary under the different effect of each forces can be shown immediately and correctly as images on the display.



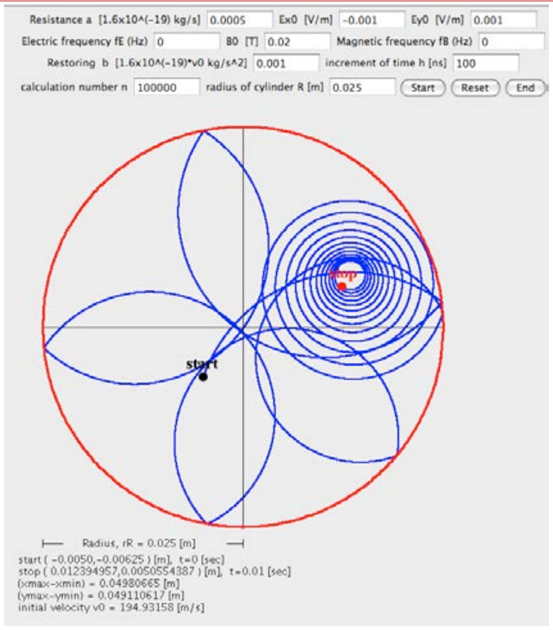
Resistance a [1.6x10^(-19) kg/s] 0.0005 Ex0 [V/m] -0.001 Ey0 [V/m] 0.001
Electric frequency fE (Hz) 0 B0 [T] 0.02 Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 0.001 increment of time h [ns] 100
calculation number n 100000 x-size of wall rX [m] 0.1 y-size of wall rY [m 0.05

start (-0.02,-0.0125) [m], t=0 [sec]
stop (-0.034065634,-0.019578118) [m], t=0.01 [sec]
(xmax-xmin) = 0.10001367 [m]
(ymax-ymin) = 0.050026074 [m]
initial velocity v0 = 194.93158 [m/s]

**Figure 4. Typical ion motion in a quadrilateral boundary, 10 cm x 5 cm.**

## 4.3 Charge Motion in Cylindrical Boundary

A typical ion motion in a cylindrical boundary for z-direction, which radius is 2.5 cm, is shown in Fig. 5. The values in the text fields except the boundary condition, that is the effects of each forces, are same as that in Fig. 4. The reflection at the cylindrical wall is clearly shown as an image on the display. By changing the values in the text fields, charge motions in the different size of the cylindrical boundary under the different effect of each forces can be also shown immediately and correctly as images on the display.



Resistance a [1.6x10^(-19) kg/s] 0.0005 Ex0 [V/m] -0.001 Ey0 [V/m] 0.001
Electric frequency fE (Hz) 0 B0 [T] 0.02 Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 0.001 increment of time h [ns] 100
calculation number n 100000 radius of cylinder R [m] 0.025

```
|——    Radius, rR = 0.025 [m]    ——|
start ( -0.0050,-0.00625 ) [m],  t=0 [sec]
stop ( 0.012394957,0.0050554387 ) [m],  t=0.01 [sec]
(xmax-xmin) = 0.04980665 [m]
(ymax-ymin) = 0.049110617 [m]
initial velocity v0 = 194.93158 [m/s]
```

**Figure 5. Typical ion motion in a cylinderical boundary, which radius is 2.5 cm. The values are same as that in Fig. 4 except the boundary condition.**

## 4.4 A sample of Program

One example of the program used here to get Fig. 5 is shown below. The function of the main part of program is written as a comment line.

```
/* Charge motion in cylindrical wall 2012.01, 2014.01 M. Morooka */
/*<applet code="IonAIVP30.class"width=700 height=800></applet>*/
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class IonAIVP30 extends Applet implements ActionListener{
        TextField txt1,txt2,txt3,txt4,txt5,txt6,txt7,txt8,txt9,txt10;
        Label lb1,lb2,lb3,lb4,lb5,lb6,lb7,lb8,lb9,lb10;
        Button btn1,btn2,btn3;
        String moji;
        double a,ex0,ey0,fE,b0,fB,b,h,rR;
        int n;
        public void init(){
                lb1=new Label("Resistance a  [1.6x10^(-19) kg/s]");
                add(lb1);
                txt1=new TextField(8);
                add(txt1);
                lb2=new Label("Ex0  [V/m]");
                add(lb2);
                txt2=new TextField(8);
                add(txt2);
                lb3=new Label("Ey0  [V/m]");
                add(lb3);
                txt3=new TextField(8);
                add(txt3);
```

```
		lb4=new Label("Electric frequency fE (Hz)");
		add(lb4);
		txt4=new TextField(8);
		add(txt4);
		lb5=new Label("B0  [T]");
		add(lb5);
		txt5=new TextField(8);
		add(txt5);
		lb6=new Label("Magnetic frequency fB (Hz)");
		add(lb6);
		txt6=new TextField(8);
		add(txt6);
		lb7=new Label("Restoring  b  [1.6x10^(-19)*v0 kg/s^2]");
		add(lb7);
		txt7=new TextField(8);
		add(txt7);
		lb8=new Label("increment of time h [ns]");
		add(lb8);
		txt8=new TextField(8);
		add(txt8);
		lb9=new Label("calculation number n");
		add(lb9);
		txt9=new TextField(8);
		add(txt9);
		lb10=new Label("radius of cylinder R [m]");
		add(lb10);
		txt10=new TextField(8);
		add(txt10);
		btn1=new Button("Start");
		btn1.addActionListener(this);
		add(btn1);
		btn2=new Button("Reset");
		btn2.addActionListener(this);
		add(btn2);
		btn3=new Button("End");
		btn3.addActionListener(this);
		add(btn3);
	}
	public void actionPerformed(ActionEvent e){
		moji=e.getActionCommand();
		repaint();
	}
	public void paint(Graphics g){
		if(moji=="Start"){
			lb1.setText("Resistance a  [1.6x10^(-19) kg/s]");
			lb2.setText("Ex0  [V/m]");
			lb3.setText("Ey0  [V/m]");
			lb4.setText("Electric frequency fE (Hz)");
			lb5.setText("B0  [T]");
			lb6.setText("Magnetic frequency fB (Hz)");
			lb7.setText("Restoring  b  [1.6x10^(-19) *v0 kg/s^2]");
			lb8.setText("increment of time h [ns]");
			lb9.setText("calculation number n");
```

```
				lb10.setText("radius of cylinder R [m]");
				try{
					a=Double.parseDouble(txt1.getText());
					ex0=Double.parseDouble(txt2.getText());
					ey0=Double.parseDouble(txt3.getText());
					fE=Double.parseDouble(txt4.getText());
					b0=Double.parseDouble(txt5.getText());
					fB=Double.parseDouble(txt6.getText());
					b=Double.parseDouble(txt7.getText());
					h=Double.parseDouble(txt8.getText());
					n=Integer.parseInt(txt9.getText());
					rR=Double.parseDouble(txt10.getText());
				}
				catch(NumberFormatException ex){
					lb1.setText("  error");
				}
				double[] t=new double[n+2];
				double[] x=new double[n+2];
				double[] vx=new double[n+2];
				double[] y=new double[n+2];
				double[] vy=new double[n+2];
		//constants
				double q,z,m,m0;
				a=a*1.6e-19;
				b=b*1.6e-19;
				h=0.000000001*h;
				z=1.0;
				q=z*1.6e-19;//charge q=z x 1.6e(-19) [C]
				m=9.11e-31;//mass of electron [kg]
				m0=197.0;//mass No. of ion (Gold)
				m=m0*1.66e-27;//mass of atom and ion
				double tmax;
				tmax = h*(double)n;// final time (sec)
				double v0,temp;
				temp=27.0;// temperature [C]
				v0=Math.sqrt(3.0*1.38e-23*(273.15+temp)/m);// thermal velocity [m/s]
				b=b*v0;
				double c1,c2,eEx0,eEy0,bB0;
				c1=a/m;
				c2=b/m;
				eEx0=ex0*q/m;
				eEy0=ey0*q/m;
				bB0=b0*q/m;
		//center of cylinder (x0,y0), minimum (xWmin,yWmin) and maximum (xWmax,yWmax) points of cylinder
				double x0,y0,xWmin,xWmax,yWmin,yWmax,r; // r: distance from the center = sqrt.((x-x0)^2+(y-
y0)^2)
				x0=0.0;
				y0=0.0;
				xWmin=x0-rR;
				xWmax=x0+rR;
				yWmin=y0-rR;
				yWmax=y0+rR;
```

```
                    double vT,vP; //vT:perpendicular component of v, vP:parallel component of v for wall
        //initial conditions
                    t[0]=0.0;
                    x[0]=x0-0.2*rR;// sqrt.((x[0]-x0)*(x[0]-x0)+(y[0]-y0)*(y[0]-y0)) < rR
                    y[0]=y0-0.25*rR;// sqrt.((x[0]-x0)*(x[0]-x0)+(y[0]-y0)*(y[0]-y0)) < rR
                    vx[0]=v0*1.0/3.0;//[m/s]
                    vy[0]=Math.sqrt(v0*v0-vx[0]*vx[0]);
                    double xR0,yR0;// center of restoring force (xR0,yR0)
                    xR0=x0;
                    yR0=y0;
        //Runge-Kutta method
                    double k1,k2,k3,k4,l1,l2,l3,l4,m1,m2,m3,m4,n1,n2,n3,n4;
                    int i;
                    for(i=0;i<=n;++i){
                        k1=vx[i];
                        l1=-
c1*vx[i]+eEx0*Math.cos(2.0*Math.PI*fE*t[i])+bB0*vy[i]*Math.cos(2.0*Math.PI*fB*t[i])-c2*(x[i]-xR0);
                        m1=vy[i];
                        n1=-c1*vy[i]+eEy0*Math.cos(2.0*Math.PI*fE*t[i])-
bB0*vx[i]*Math.cos(2.0*Math.PI*fB*t[i])-c2*(y[i]-yR0);
                        k2=vx[i]+h/2.0*l1;
                        l2=-
c1*(vx[i]+h/2.0*l1)+eEx0*Math.cos(2.0*Math.PI*fE*(t[i]+h/2.0))+bB0*(vy[i]+h/2.0*n1)*Math.cos(2.0*Math.PI*fB*(t[i]+h/2.0))-
c2*(x[i]+h/2.0*k1-xR0);
                        m2=vy[i]+h/2.0*n1;
                        n2=-c1*(vy[i]+h/2.0*n1)+eEy0*Math.cos(2.0*Math.PI*fE*(t[i]+h/2.0))-
bB0*(vx[i]+h/2.0*l1)*Math.cos(2.0*Math.PI*fB*(t[i]+h/2.0))-c2*(y[i]+h/2.0*m1-yR0);
                        k3=vx[i]+h/2.0*l2;
                        l3=-
c1*(vx[i]+h/2.0*l2)+eEx0*Math.cos(2.0*Math.PI*fE*(t[i]+h/2.0))+bB0*(vy[i]+h/2.0*n2)*Math.cos(2.0*Math.PI*fB*(t[i]+h/2.0))-
c2*(x[i]+h/2.0*k2-xR0);
                        m3=vy[i]+h/2.0*n2;
                        n3=-c1*(vy[i]+h/2.0*n2)+eEy0*Math.cos(2.0*Math.PI*fE*(t[i]+h/2.0))-
bB0*(vx[i]+h/2.0*l2)*Math.cos(2.0*Math.PI*fB*(t[i]+h/2.0))-c2*(y[i]+h/2.0*m2-yR0);
                        k4=vx[i]+h*l3;
                        l4=-
c1*(vx[i]+h*l3)+eEx0*Math.cos(2.0*Math.PI*fE*(t[i]+h))+bB0*(vy[i]+h*n3)*Math.cos(2.0*Math.PI*fB*(t[i]+h))-c2*(x[i]+h*k3-
xR0);
                        m4=vy[i]+h*n3;
                        n4=-c1*(vy[i]+h*n3)+eEy0*Math.cos(2.0*Math.PI*fE*(t[i]+h))-
bB0*(vx[i]+h*l3)*Math.cos(2.0*Math.PI*fB*(t[i]+h))-c2*(y[i]+h*m3-yR0);
                        t[i+1]=t[i]+h;
                        x[i+1]=x[i]+h/6.0*(k1+2.0*k2+2.0*k3+k4);
                        vx[i+1]=vx[i]+h/6.0*(l1+2.0*l2+2.0*l3+l4);
                        y[i+1]=y[i]+h/6.0*(m1+2.0*m2+2.0*m3+m4);
                        vy[i+1]=vy[i]+h/6.0*(n1+2.0*n2+2.0*n3+n4);
                        r = Math.sqrt((x[i+1]-x0)*(x[i+1]-x0)+(y[i+1]-y0)*(y[i+1]-y0));

                        if(r>=rR){
                            vT = (x[i+1]-x0)/rR*vx[i+1]+(y[i+1]-y0)/rR*vy[i+1];
                            vP = (y[i+1]-y0)/rR*vx[i+1]-(x[i+1]-x0)/rR*vy[i+1];
                            vT=-vT;
                            vx[i+1] = (x[i+1]-x0)/rR*vT+(y[i+1]-y0)/rR*vP;
```

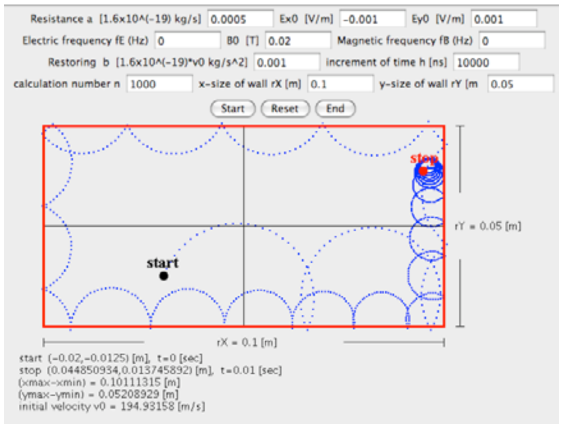
```
                                        vy[i+1] = (y[i+1]-y0)/rR*vT-(x[i+1]-x0)/rR*vP;
                              }
                    }
          //maximum and minimum values of x[i] and y[i]
                    double xmin,xmax,ymin,ymax;
                    xmin=x[0];
                    xmax=x[0];
                    ymin=y[0];
                    ymax=y[0];
                    for(i=0;i<=n;++i){
                              xmax=Math.max(x[i],xmax);//maximum of x
                              xmin=Math.min(x[i],xmin);//minimum of x
                              ymax=Math.max(y[i],ymax);//maximum of y
                              ymin=Math.min(y[i],ymin);//minimum of y
                    }
          // print range on display; from  (xDmin,yDmin) to (xDmax,yDmax)
                    int xDmin,xDmax,yDmin,yDmax;
                    xDmin=50;
                    xDmax=550;
                    yDmin=150;
                    yDmax=yDmin+(xDmax-xDmin);
          // point of cylinder on display
                    double xDa,yDa;
               //center (x0,y0) of cylinder
                    int x0D,y0D;
                    xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(x0-xWmin)/(2.0*rR);
                    yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(yWmax-y0)/(2.0*rR);
                    x0D=(int)xDa;
                    y0D=(int)yDa;
               // point of cylinder
                    int iwmax;
                    iwmax = 200;
                    int[] xDw=new int[iwmax+2];// point of x of cylinder
                    int[] ypDw=new int[iwmax+2];// point of +y of cylinder
                    int[] ynDw=new int[iwmax+2];// point of -y of cylinder
                    double xw,xDwa,ypw,ynw,ypDwa,ynDwa;
                    for(i=0;i<=iwmax;++i){
                              xw=xWmin+(xWmax-xWmin)*(double)i/(double)iwmax;
                              xDwa=(double)xDmin+(double)(xDmax-xDmin)*(xw-xWmin)/(xWmax-xWmin);
                              xDw[i]=(int)xDwa;
                              ypw=y0+Math.sqrt(rR*rR-(xw-x0)*(xw-x0));
                              ynw=y0-Math.sqrt(rR*rR-(xw-x0)*(xw-x0));
                              ypDwa=(double)yDmin+(double)(yDmax-yDmin)*(yWmax-ypw)/(yWmax-yWmin);
                              ypDw[i]=(int)ypDwa;
                              ynDwa=(double)yDmin+(double)(yDmax-yDmin)*(yWmax-ynw)/(yWmax-yWmin);
                              ynDw[i]=(int)ynDwa;
                    }
               // minimum (xWmin,yWmin) and maximum (xWmax,yWmax)) of cylinder
                    int xWminD,yWminD,xWmaxD,yWmaxD;
                    xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xWmin-
xWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                    xWminD=(int)xDa;
```

```
                              yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(yWmax-
yWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              yWminD=(int)yDa;
                              xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xWmax-
xWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              xWmaxD=(int)xDa;
                              yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(yWmax-
yWmax)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              yWmaxD=(int)yDa;
      // point of x[i] and y[i] on display (xDmin at x[i]=xWmin and xDmax at x[i]=xWmax, yDmin at y[i]=yWmax and
yDmax at y[i]=yWmin)
                              int[] xD=new int[n+2];// point of x[i] on display
                              int[] yD=new int[n+2];// point of y[i] on display
                              for(i=0;i<=n;++i){
                                    xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(x[i]-
xWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                                    yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(yWmax-
y[i])/Math.max(xWmax-xWmin,yWmax-yWmin);
                                    xD[i]=(int)xDa;
                                    yD[i]=(int)yDa;
                              }
      // point of xmin, xmax, ymin, and ymax on display
                              int xminD,yminD,xmaxD,ymaxD;
                              xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xmin-
xWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xWmax-
ymin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              xminD=(int)xDa;
                              yminD=(int)yDa;
                              xDa=(double)xDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xmax-
xWmin)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              yDa=(double)yDmin+(double)Math.min((xDmax-xDmin),(yDmax-yDmin))*(xWmax-
ymax)/Math.max(xWmax-xWmin,yWmax-yWmin);
                              xmaxD=(int)xDa;
                              ymaxD=(int)yDa;
      // print of start and stop points and motion range
                              double xW,yW;
                              xW=xmax-xmin;//[m]
                              yW=ymax-ymin;//[m]
                              float startx,starty,stopx,stopy,rRf,xWf,yWf;
                              startx=(float)x[0];
                              starty=(float)y[0];
                              stopx=(float)x[n];
                              stopy=(float)y[n];
                              rRf=(float)rR;
                              xWf=(float)xW;
                              yWf=(float)yW;
                              String sstartx,sstarty,sstopx,sstopy,srR,sxW,syW;
                              sstartx=Float.toString(startx);
                              sstarty=Float.toString(starty);
                              sstopx=Float.toString(stopx);
                              sstopy=Float.toString(stopy);
                              srR=Float.toString(rRf);
```

```
                    sxW=Float.toString(xWf);
                    syW=Float.toString(yWf);
                    float tmaxf;
                    tmaxf=(float)tmax;
                    String stmax;
                    stmax = Float.toString(tmaxf);
                    float v0f;
                    v0f=(float)v0;
                    String sv0;
                    sv0 = Float.toString(v0f);
                    g.drawString("Radius, rR = "+srR+" [m]",xWminD+50,yWminD+25);
                    g.drawLine(xWminD,yWminD+20-5,xWminD,yWminD+20+5);
                    g.drawLine(x0D,yWminD+20-5,x0D,yWminD+20+5);
                    g.drawLine(xWminD,yWminD+20,xWminD+25,yWminD+20);
                    g.drawLine(x0D-20,yWminD+20,x0D,yWminD+20);
                    g.drawString("start ( "+sstartx+","+sstarty+" ) [m],  t=0 [sec]",xWminD-30,yWminD+45);
                    g.drawString("stop ( "+sstopx+","+sstopy+" ) [m],  t="+stmax+" [sec]",xWminD-30,yWminD+60);
                    g.drawString("(xmax-xmin) = "+sxW+" [m]",xWminD-30,yWminD+75);
                    g.drawString("(ymax-ymin) = "+syW+" [m]",xWminD-30,yWminD+90);
                    g.drawString("initial velocity v0 = "+sv0+" [m/s]",xWminD-30,yWminD+105);
        // charge motion on display
                    for(i=0;i<=n;++i){
                            try{
                                    g.setColor(Color.blue);
                                    g.fillOval(xD[i],yD[i],1,1);
                            }
                            catch(ArrayIndexOutOfBoundsException ex){
                            }
                    }
                    g.setColor(Color.black);
                    g.drawLine(xWminD,(yWmaxD+yWminD)/2,xDmax,(yWmaxD+yWminD)/2);
                    g.drawLine((xWminD+xWmaxD)/2,yWmaxD,(xWminD+xWmaxD)/2,yWminD);
        // print of wall on display
                    g.setColor(Color.red);
                    BasicStroke stroke;
                    Graphics2D g2 = (Graphics2D)g;
                    stroke = new BasicStroke(2.0f); //thickness of wall
                    g2.setStroke(stroke);
                    for(i=0;i<=iwmax-1;++i){
                            try{
                                    g2.drawLine(xDw[i],ypDw[i],xDw[i+1],ypDw[i+1]);
                                    g2.drawLine(xDw[i],ynDw[i],xDw[i+1],ynDw[i+1]);
                            }
                            catch(ArrayIndexOutOfBoundsException ex){
                            }
                    }
        //selection of font
                    try {
                            Font f1=new Font("TIMES",Font.BOLD,20);
                            g.setFont (f1);
                            g.setColor(Color.black);
                            g.fillOval(xD[0]-5,yD[0]-5,10,10);
                    g.drawString("start",xD[0]-20,yD[0]-10);
                    g.setColor(Color.red);
                    g.fillOval(xD[n]-5,yD[n]-5,10,10);
                    g.drawString("stop",xD[n]-15,yD[n]-10);
                }
                catch (ArrayIndexOutOfBoundsException ex){
                }
            }
            else if(moji=="End"){
                System.exit(0);
            }
            if(moji=="Reset"){
                g.clearRect(0,0,701,500);
            }
        }
}
```

## 5 DISCUSSION

The accuracy of the calculation by this program depends on the value of the time increment, *h*. If we use a too large *h*, the calculation is not done accurately and the locus of the motion is shown non-continuously as shown in Fig. 6. In this case, the reflection at the wall slips out of the wall place, and the calculated point of the charge deviates more from the accurate point with increasing the time. The calculation is more accurate with use of smaller *h*, but a larger number of calculation, that is a longer calculation time, is needed to get a characteristic motion. It is better to chose the largest *h* value by tentative calculation with relative small calculation number *n*. Here, we used fourth-order Runge-Kutta method to solve numerically the ordinary differential equations, but the results such as Figs. 1-5 were completely same as that obtained by the improved  Runge-Kutta-Gill method [2] by using an appropriate value of *h*.



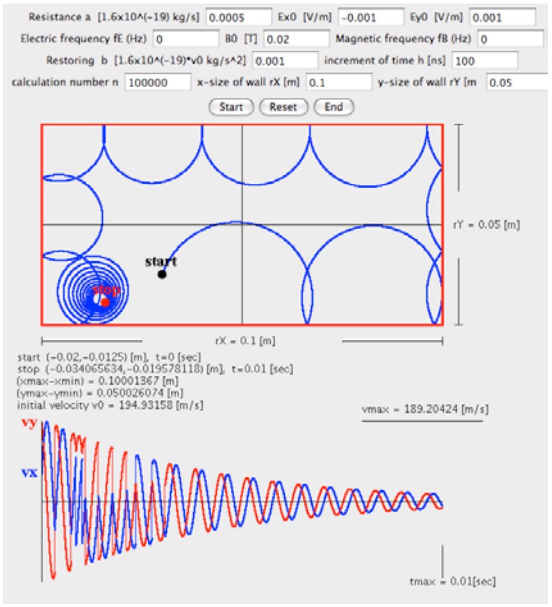
Resistance a [1.6x10^(-19) kg/s] 0.0005 Ex0 [V/m] -0.001 Ey0 [V/m] 0.001
Electric frequency fE (Hz) 0 B0 [T] 0.02 Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 0.001 increment of time h [ns] 10000
calculation number n 1000 x-size of wall rX [m] 0.1 y-size of wall rY [m 0.05

```
start (-0.02,-0.0125) [m], t=0 [sec]
stop (0.044850934,0.013745892) [m], t=0.01 [sec]
(xmax-xmin) = 0.10111315 [m]
(ymax-ymin) = 0.05208929 [m]
initial velocity v0 = 194.93158 [m/s]
```

Figure 6. A sample of an non-accurate calculation using a too large h.

In Figure 6, the values used for the calculation are same as that in Fig. 4, except *h* and *n*. The final time at the stop point, 0.05 sec, is also same as that in Fig. 4, but the stop point deviates largely from that in Fig. 4. The reflection at the wall slips out of the wall place, and the calculated point deviates more from that in Fig. 4 with increasing the time.

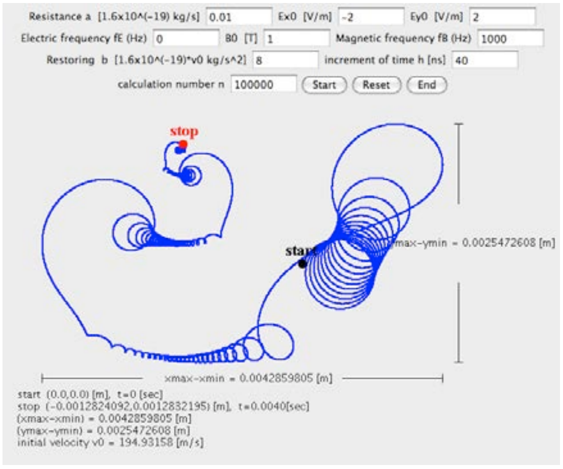
By plotting the change of the velocity with time in addition to the charge locus, as shown in Fig. 7, the charge motion can be understood more clearly in relation to the velocity, such as, the decrease of Larmor radius with time is caused by the decrease of the velocity due to the resistance force.


```
Resistance a [1.6x10^(-19) kg/s] 0.0005   Ex0 [V/m] -0.001   Ey0 [V/m] 0.001
Electric frequency fE (Hz) 0   B0 [T] 0.02   Magnetic frequency fB (Hz) 0
Restoring b [1.6x10^(-19)*v0 kg/s^2] 0.001   increment of time h [ns] 100
calculation number n 100000   x-size of wall rX [m] 0.1   y-size of wall rY [m 0.05

start (-0.02,-0.0125) [m], t=0 [sec]
stop (-0.034065634,-0.019578118) [m], t=0.01 [sec]
(xmax-xmin) = 0.10001367 [m]
(ymax-ymin) = 0.050026074 [m]
initial velocity v0 = 194.93158 [m/s]
vmax = 189.20424 [m/s]
```

**Figure 7. Change of the velocity with the time plotted together with the charge motion in Fig. 4. It is clearly understood that the decrease of Lamor radius is caused by the decrease of the velocity.**

The values in the text fields are used here from a point of view such that each force is effective or not effective. The used values for the electric and magnetic fields are reasonable, but that for the resistance and restoring change very much under the circumstances of the charge motion, for example, the value of resistance force, *a*, is very large in a solid but nearly equal to 0 in a vacuum. In this program, the values are used with no-considering actual media for the charge motion. In this view point, the charge shows a very mysterious motion under a particular condition, as shown in Fig. 8.



Resistance a [1.6x10^(-19) kg/s] 0.01 Ex0 [V/m] -2 Ey0 [V/m] 2
Electric frequency fE (Hz) 0 B0 [T] 1 Magnetic frequency fB (Hz) 1000
Restoring b [1.6x10^(-19)*v0 kg/s^2] 8 increment of time h [ns] 40
calculation number n 100000

```
start (0.0,0.0) [m], t=0 [sec]
stop (-0.0012824092,0.0012832195) [m], t=0.0040[sec]
(xmax-xmin) = 0.0042859805 [m]
(ymax-ymin) = 0.0025472608 [m]
initial velocity v0 = 194.93158 [m/s]
```

**Figure 8. A mysterious motion of charge under a particular condition.**

## 6 CONCLUSION

Java programs in a GUI environment have been developed for the simulations of charge motion in electric and magnetic fields. The values of the selected parameters for the numerical calculation are set using text fields on the display, and the calculation is initiated by clicking the start button after inputting these values. The calculated results are plotted immediately after the completion of the calculation as a locus of the charge motion on the display. By changing the values in the text fields and clicking again the start button, new results can be displayed immediately. The simulations of the charge motion depending on each force in the electric and magnetic fields can be obtained easily and accurately as a locus of the motion on the display. The time needed to simulate is very short and less than several seconds using an usual personal computer. These Java programs are useful for an image learning of the charge motion in education applications because of their ability to quickly provide accurate depictions of fundamentals of charge motion.

This Java program can be applied for rapidly and accurately image learning for the phenomena expressed by ordinary differential equations due to use Runge-Kutta method. By using Crank-Nicolson's implicit method and Gauss-Seidel's iteration method [1], these Java simulations are also useful in education applications for rapidly and accurately image learning for the more complicated phenomena expressed by partial differential equations such as diffusions of atoms and propagations of heat and wave, etc.

## REFERENCES

[1]. M. Morooka, Journal of Software Engineering and Applications, vol. 5 No. 10, pp. 764-776, (2012).

[2]. S. Gill, Proc. Cambridge Phil. Soc. , vol. 47, 96–108, (1951).