*** Two arrays must be declared:
!*** y (4) - an array containing the coordinates x = y (1), y=y(2) and projections of velocity vector
!*** Vx = y (3),Vy=y(4).
!*** The working array work (27) with a length to be calculated using the equation
!*** 3 + 6 * neqn (neqn- is number of equations)
!*** m_maa variable must also be declared as REAL. Here we take into account the Fortran
!*** feature. If variables are not declared at the beginning of the program by the REAL,
!*** INTEGER, etc. operators, the Fortran compiler can declare and create them by using the
!*** default rule: if the first character in the variable name is **i, j, k, l, m, n** then the variable must be
!*** INTEGER if it is not so then the variable must be REAL.
real y(2),work(15),m,k
!*** Declaration of additional integer work array iwork(5) with fixed length 5 (defined in subroutine rkf45)
integer iwork(5)
!*** Description the global variables which must be used in subroutine ″func″ to calculate
!*** derivatives of coordinate x,y and velocities projections Vx,Vy with respect to time
common omega2,beta
!*** external operator should be used to describe the variable “func” as a name of external subroutine
external func
!*** external operator should be used to describe the variable “func” as a name of external subroutine
open(10,file=″harm.dat″)
!*** elastic constant of the spring in N/m
k=5.
!*** mass of harmonic oscillator in kg
m=0.1
!*** frequency squared calculation
omega2=k/m
!*** coefficients of resistance force
alpha=0.1
beta=alpha/(2.*m)
*** Relative and absolute errors for calculation the coordinate and velocity this is a input parameter for
!*** subroutine rkf45
relerr=1.e-8
abserr=1.e-8
!*** Number of differential equations
neqn=2
!*** Initial value of time
t=0.
!*** iflag=1 is need to start the calculation, it means that we are starting a new simulation
iflag=1
!*** Initial conditions
!*** y(1)=0.1 m – x coordinate of point mass
!*** y(2)=0. m/s – velocity (x-projection) of point mass
y(1)=0.1
y(2)=0.

```
!*** Total number of time steps
nt=10000
!*** Time step in seconds
dt=0.001
!*** An additional parameter to save the value of the calculated work of the resistance force
!*** The general equation for calculating the work of any force looks like this:  A=\int_{x_1}^{x_2} \vec{F}\cdot\vec{dr}  , here
!***  \vec{F} - force vector and \vec{dr}  displacement vector  . For infinitely small displacement  \vec{dr}  we get equation
!***  dA=\vec{F}\cdot\vec{dr}=F_x\cdot dx+F_y\cdot dy+F_z\cdot dz  and for one dimensional motion
!***  dA=\vec{F}\cdot\vec{dr}=F_x\cdot dx  or  A=F\cdot\Delta x=F\cdot(x_2-x_1)  , here x2 and x1 are the end and start coordinates of the point
!***  mass
too=0.
!*** An additional parameter to save the value of the calculated work of the resistance force
x=y(1)
*** Cycle operator to perform the integration of the system of differential equations and
!*** calculation of coordinates and velocities at different time moments with timestep dt=0.001 second.
!*** The total time of simulation can be calculated as a product of variables  nt and dt ,
!*** in our  case it is equal to nt*dt=0.001*10000=10 seconds.
do i=1,nt
!*** Calculation of new value of time (should be done by hand)
tout=t+dt
*** Calculation of coordinate y(1) and velocitie y(2) at the next time moment tout=t+dt
call rkf45(func,neqn,y,t,tout,relerr,abserr,iflag,work,iwork)
!*** Just in case the control of the iflag value.  If iflag=2 the calculation was success and
!*** we can to continue
if (iflag.ne.2) then
iflag=2
endif
!*** calculation of displacement
dx=y(1)-x
!*** Calculation of the work of the resistance force
too=too-alpha*y(2)*dx
x=y(1)
!*** calculating of kinetic,potential and total energies
ekin=0.5*m*y(2)**2
epot=0.5*k*y(1)**2
etot=ekin+epot
!*** calculating of corrected total energy  E_{corr}=E_{tot}-A_{resist}  , we take into account the energy lost due to
!*** the force of resistance
ecorr=etot-too
!***Saving data in the following format
!*** time  x  Vx  Ekin Epot Etot Ecorr
write(10,*) tout,y,ekin,epot,etot,ecorr
```

```
enddo
!*** stop program
stop
!*** end of source code
end
!*** The most important part of the program. Here we must implement the differential equation.
!*** Now we  need to calculate the derivatives with respect to the coordinate and velocity
!*** with respect to time.
!*** Now the subroutine ″func″ must be created. Number of input parameters is fixed.
!*** t-time, y(4)-array consist the coordinate y(1),y(2) and velocity projections y(3),y(4) of sphere,
!*** yp(4)-array with derivatives so that:
!*** dy(1)=dy(1)/dt=velocity=y(2) and
!*** dy(2)=dy(2)/dt=acceleration=-omega2*y(1)-2.*beta*y(2)
!*** (theoretical background you can find in precis of lecture)
subroutine func(t,y,dy)
!*** Description the global variables which must be used in subroutine ″func″ to calculate
!*** derivatives of coordinate x and velocities projections  Vx with respect to time
common omega2,beta
!*** Declaration of arrays
real y(2),dy(2)
*** first derivative for coordinate x with respect to time
dy(1)=y(2)
!*** Second derivative, calculation of x-projection of acceleration with the second Newton′s low
dy(2)=-omega2*y(1)-2.*beta*y(2)
!*** Back to call operator in the main program
return
!*** End of source code for subroutine “func”
end
```