Answers to Homework 11: Numerical Solution of ODE: Multistep Methods)

1. Derive a *variable-step* Adams-Bashforth method of the form

$$y_{k+1} = y_k + h_k\, A_k\, f(x_k, y_k) + h_k\, B_k\, f(x_{k-1}, y_{k-1})\,, \tag{1}$$

where $x_{k+1} = x_k + h_k$, $h_k = \alpha_k\, h_{k-1}$, and y_k approximates $y(x_k)$ – the solution of the initial-value problem

$$\left\{ \begin{array}{rcl} y'(x) & = & f(x,y) \\ y(x_0) & = & y_0 \end{array} \right. \tag{2}$$

Considering α_k and h_0 as given, derive appropriate expressions for the coefficients A_k and B_k and determine the order of the method.

Answer:

To derive this method, we can use linear interpolation to approximate the derivative of the function $y(x)$:

$$y'(x) \approx P(x) = \frac{f(x_k, y_k)(x - x_{k-1}) - f(x_{k-1}, y_{k-1})(x - x_k)}{h_{k-1}}\,.$$

The predicted value of $y(x_{k+1})$ is then approximated as follows:

$$\begin{aligned} y(x_{k+1}) &= y(x_k) + \int_{x_k}^{x_{k+1}} y'(x)\,dx \approx y_k + \int_{x_k}^{x_{k+1}} P(x)\,dx \\ &= y_k + \frac{f(x_k, y_k)}{h_{k-1}} \int_{x_k}^{x_{k+1}} (x - x_{k-1})\,dx - \frac{f(x_{k-1}, y_{k-1})}{h_{k-1}} \int_{x_k}^{x_{k+1}} (x - x_k)\,dx\,. \end{aligned}$$

We find that

$$A_k = \frac{1}{h_k\, h_{k-1}} \int_{x_k}^{x_{k+1}} (x - x_{k-1})\,dx = \frac{1}{h_k\, h_{k-1}} \int_{h_{k-1}}^{h_k + h_{k-1}} t\,dt = \frac{(h_k + h_{k-1})^2 - h_{k-1}^2}{2\, h_k\, h_{k-1}} = 1 + \frac{\alpha_k}{2}\,,$$

and

$$B_k = -\frac{1}{h_k\, h_{k-1}} \int_{x_k}^{x_{k+1}} (x - x_k)\,dx = -\frac{1}{h_k\, h_{k-1}} \int_0^{h_k} h_k t\,dt = -\frac{h_k^2}{2\, h_k\, h_{k-1}} = -\frac{\alpha_k}{2}\,.$$

To determine the local error of this method, we can integrate the error of the polynomial interpolation:

$$E = \int_{x_k}^{x_{k+1}} \frac{y'''(\xi_x)}{2} (x - x_{k-1})(x - x_k)\,dx\,,$$

where $y'''(x)$ is assumed to exist and ξ_x is some point between x_{k-1} and x_{k+1} (which depends on x). Since the factor $(x-x_{k-1})(x-x_k)$ does not change sign on the interval of integration, we can apply the weighted mean-value theorem for integrals to deduce that

$$\begin{aligned} E &= \frac{y'''(\xi)}{2}\int_{x_k}^{x_{k+1}}(x-x_{k-1})(x-x_k)\,dx = \frac{y'''(\xi)}{2}\int_0^{h_k}(t+h_{k-1})\,t\,dt \\ &= \frac{y'''(\xi)}{2}\left(\frac{h_k^3}{3}+\frac{h_k^2 h_{k-1}}{2}\right) = \frac{y'''(\xi)\,h_{k-1}^3\,\alpha_k^2\,(2\alpha_k+3)}{12}\,. \end{aligned}$$

where ξ is some point in $[x_{k-1}, x_{k+1}]$. We see that the local error has the order $O(h_0^3)$. The method order is 2.

2. Applying numerical differentiation to approximate the derivative $y'(x_{k+1})$ in the ordinary differential equation

$$y'(x) = f(x,y)\,, \tag{3}$$

we can arrive at the *backward differentiation formula* (BDF)

$$\sum_{i=0}^{n} A_i\, y_{k+1-i} \approx h\; f\,(x_{k+1}, y_{k+1})\;. \tag{4}$$

Derive a BDF method of the form

$$A\,y_{k+1} + B\,y_k + C\,y_{k-1} = h\; f\,(x_{k+1}, y_{k+1})\;, \tag{5}$$

find its error and compare it with the error of the Adams-Moulton method of the same order.

Answer:

We can find the coefficients, for example, with the Taylor method. Using the Taylor series expansions

$$y(x_k) = y(x_{k+1}) - h\,y'(x_{k+1}) + \frac{h^2}{2}\,y''(x_{k+1}) - \frac{h^3}{6}\,y'''(\xi_1)$$

and

$$y(x_{k-1}) = y(x_{k+1}) - 2h\,y'(x_{k+1}) + 2h^2\,y''(x_{k+1}) - \frac{(2h)^3}{6}\,y'''(\xi_2)\,,$$

where ξ_1 is a point in $[x_k, x_{k+1}]$ and ξ_2 is a point in $[x_{k-1}, x_{k+1}]$, we arrive at a system of linear equations for the coefficients of the approximation

$$h\,y'(x_{k+1}) \approx A\,y(x_{k+1}) + B\,y(x_k) + C\,y(x_{k-1})\,.$$

The equations are

$$\begin{aligned} A + B + C &= 0 \\ -B - 2C &= 1 \\ \frac{1}{2}B + 2C &= 0 \end{aligned}$$

and the solution is easily found to be $A = 3/2$, $B = -2$, and $C = 1/2$.

The error of the approximation is

$$E_{BDF} = -\frac{B}{A}\frac{h^3}{6}y'''(\xi_1) - \frac{C}{A}\frac{(2h)^3}{6}y'''(\xi_2) = \left[y'''(\xi_1) - 2y'''(\xi_2)\right]\frac{2h^3}{9}$$

Its magnitude is bounded by

$$|E_{BDF}| \le \left(|y'''(\xi_1)| + 2|y'''(\xi_2)|\right)\frac{2h^3}{9} \le \max\left(|y'''(\xi_1)|, |y'''(\xi_2)|\right)\frac{2h^3}{3}\,.$$

The local error of the second-order Adams-Moulton method is

$$E_{AM} = -y'''(\xi)\frac{h^3}{12}\,,$$

where ξ is a point in $[x_k, x_{k-1}]$, not necessarily equal to ξ_1. The bound on the magnitude of E_{AM} is approximately eight times smaller than the bound on $|E_{BDF}|$.

3. In Homework 10, we found that Euler's method can be unstable when applied to the initial-value problem

$$\begin{cases} y''(x) &= y(x) \\ y(0) &= y_0 \\ y'(0) &= -y_0 \end{cases} \tag{6}$$

Prove that both the backward Euler method (Adams-Moulton of order 1) and the trapezoidal method (Adams-Moulton of order 2) are unconditionally stable in this case for the positive step h.

Adams-Moulton methods have the general form

$$y_{k+1} = y_k + h\sum_{i=0}^{n} A_i\, f\left(x_{k+1-i}, y_{k+1-i}\right)\,. \tag{7}$$

Answer:

The first step is to rewrite the second-order equation as a system of two first-order equations. The system takes the form

$$\begin{cases} y'(x) &= p(x) \\ p'(x) &= y(x) \end{cases}$$

with the initial conditions

$$\begin{cases} y(0) &= y_0 \\ p(0) &= -y_0 \end{cases}$$

In the matrix form, the backward Euler method applied to this system is

$$\begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ p_k \end{bmatrix} + h\begin{bmatrix} p_{k+1} \\ y_{k+1} \end{bmatrix}$$

or, equivalently,

$$\begin{bmatrix} 1 & -h \\ -h & 1 \end{bmatrix}\begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ p_k \end{bmatrix}$$

Inverting the matrix, we can express Euler's method for this case in the explicit form

$$\begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & -h \\ -h & 1 \end{bmatrix}^{-1} \begin{bmatrix} y_k \\ p_k \end{bmatrix} = \frac{1}{1-h^2} \begin{bmatrix} 1 & h \\ h & 1 \end{bmatrix} \begin{bmatrix} y_k \\ p_k \end{bmatrix}$$

The first step yields

$$\begin{bmatrix} y_1 \\ p_1 \end{bmatrix} = \frac{1}{1-h^2} \begin{bmatrix} 1 & h \\ h & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix} = \frac{1}{1+h} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix}$$

By induction,

$$\begin{bmatrix} y_k \\ p_k \end{bmatrix} = \frac{1}{(1+h)^k} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix}$$

for all $k \geq 0$. For any $h > 0$, the numerical solution will decrease in magnitude at each step consistently with the exact solution $y(x) = y_0 e^{-x}$, $p(x) = -y_0 e^{-x}$. We need to select a stable solution in the special case $h = 1$, because, in this case, the original system is underdetermined and can admit infinitely many solutions.

The case of the trapezoidal method is analyzed analogously. The method takes the form

$$\begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} y_k \\ p_k \end{bmatrix} + \frac{h}{2} \begin{bmatrix} p_k \\ y_k \end{bmatrix} + \frac{h}{2} \begin{bmatrix} p_{k+1} \\ y_{k+1} \end{bmatrix}$$

or, equivalently,

$$\begin{bmatrix} 1 & -h/2 \\ -h/2 & 1 \end{bmatrix} \begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h/2 \\ h/2 & 1 \end{bmatrix} \begin{bmatrix} y_k \\ p_k \end{bmatrix}$$

Inverting the matrix, we can express the method in the explicit form

$$\begin{aligned} \begin{bmatrix} y_{k+1} \\ p_{k+1} \end{bmatrix} &= \begin{bmatrix} 1 & -h/2 \\ -h/2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & h/2 \\ h/2 & 1 \end{bmatrix} \begin{bmatrix} y_k \\ p_k \end{bmatrix} = \frac{1}{1-\frac{h^2}{4}} \begin{bmatrix} 1 & h/2 \\ h/2 & 1 \end{bmatrix}^2 \begin{bmatrix} y_k \\ p_k \end{bmatrix} \\ &= \frac{1}{1-\frac{h^2}{4}} \begin{bmatrix} 1+\frac{h^2}{4} & h \\ h & 1+\frac{h^2}{4} \end{bmatrix} \begin{bmatrix} y_k \\ p_k \end{bmatrix} \end{aligned}$$

The first step yields

$$\begin{bmatrix} y_1 \\ p_1 \end{bmatrix} = \frac{1}{1-\frac{h^2}{4}} \begin{bmatrix} 1+\frac{h^2}{4} & h \\ h & 1+\frac{h^2}{4} \end{bmatrix} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix} = \frac{1-h/2}{1+h/2} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix}$$

By induction,

$$\begin{bmatrix} y_k \\ p_k \end{bmatrix} = \left(\frac{1-h/2}{1+h/2}\right)^k \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix}$$

for all $k \geq 0$. For any value $h > 0$, the numerical solution will decrease in magnitude at each step consistently with the exact solution $y(x) = y_0 e^{-x}$, $p(x) = -y_0 e^{-x}$. We need to select a stable solution in the special case $h = 2$, because, in this case, the original system can admit infinitely many solutions.

4. (Programming) In Homework 9, we approximated the number π by computing the integral

$$\int_{-1}^{1} \frac{2\,dx}{1+x^2} \tag{8}$$

A similar solution can be obtained from the initial-value problem

$$\begin{cases} y''(x) &= -\left[y'(x)\right]^2 x \\ y(-1) &= 0 \\ y'(-1) &= 1 \end{cases} \tag{9}$$

Solve this problem numerically on the interval $x \in [-1,1]$ using the following methods:

(a) Adams-Bashforth of order 2

(b) Adams-Bashforth of order 2 as predictor followed by one correction step of Adams-Moulton of order 2

(c) Adams-Moulton of order 2 solving the quadratic equation for y_{k+1}

Initialize each of these methods with the midpoint step (Runge-Kutta of order 2).

Output the absolute error in approximating $\pi = y(1)$ with the step sizes $h = 0.1$, $h = 0.01$ and $h = 0.001$.

Answer:

Step size	Adams-Bashforth	Adams-Bashforth-Moulton	Adams-Moulton
0.100	2.78036×10^{-2}	5.03666×10^{-3}	7.46844×10^{-3}
0.010	3.14736×10^{-4}	6.20849×10^{-5}	6.53020×10^{-5}
0.001	3.18631×10^{-6}	6.36360×10^{-7}	6.39661×10^{-7}

Solution:

C program:

```
#include <stdio.h> /* for output */

#include "ode.h"

/* Function: midpoint
   ------------------
   Solves a system of ODEs by the midpoint method
   n            - number of equations in the system
   nstep        - number of steps
   h            - step size
   x            - starting value of the variable
   y[n], y2[n]  - function values
   f1[n], f2[n] - storage
   slope(x,y,f) - function pointer for the right-hand side
*/
void midpoint (int n, int nstep, double h, double x,
               double* y, double* y2, double* f1, double* f2,
               void (*slope)(double x, const double* y, double* f))
{
    int k, step=0;
```

```
    /* print initial conditions */
    printf("%f ",x);
    for (k=0; k < n; k++) {
        printf("%f ",y[k]);
    }
    printf("\n");

    for (step=0; step < nstep; step++) {
        slope(x,y,f1);

        for (k=0; k < n; k++) {
            y2[k] = y[k] + 0.5*h*f1[k]; /* predictor */
        }

        slope(x+0.5*h,y2,f2);
        x+=h;

        printf("%f ",x);
        for (k=0; k < n; k++) {
            y[k] += h*f2[k];           /* corrector */
            printf("%f ",y[k]);
        }
        printf("\n");
    }
}

/* Function: adams
   ---------------
   Solves a system of ODEs by second-order Adams-Bashforth-Moulton method
   n                   - number of equations in the system
   nstep               - number of steps
   ncorr               - number of correction steps
   h                   - step size
   x                   - starting value of the variable
   y[n], y2[n]         - function values
   f1[n], f2[n]        - storage
   slope(x,y,f)        - function pointer for the right-hand side
*/
void adams (int n, int nstep, int ncorr, double h, double x,
    double* y, double* y2, double* f1, double* f2,
            void (*slope)(double x, const double* y, double* f))
{
    int k, step, corr;

    for (step=0; step < nstep; step++) {
        slope(x,y,f2);

        for (k=0; k < n; k++) {                    /* Adams-Bashforth */
            y2[k] = y[k] + 0.5*h*(3.*f2[k]-f1[k]); /* predictor       */
        }

        x += h;

        for (corr=0; corr < ncorr; corr++) {
            slope (x,y2,f1);
            for (k=0; k < n; k++) {                  /* Adams-Moulton */
                y2[k] = y[k] + 0.5*h*(f2[k]+f1[k]);  /* corrector     */
            }
```

```
        }

        printf("%f ",x);
        for (k=0; k < n; k++) {
            y[k] = y2[k];
            f1[k] = f2[k];
            printf("%f ",y[k]);
        }
        printf("\n");
    }
}

#include <stdio.h> /* for output */
#include <math.h>  /* for math functions */

#include "ode.h"

/* right-hand side for the pi ODE system */
void pi_ode (double x, const double* y, double* f)
{
    f[0] = y[1];
    f[1] = -y[1]*y[1]*x;
}

/* Adams-Moulton specialized for the pi ODE system */
void pi_moulton(int nstep,  double h, double t, double* y)
{
    int step;
    double t2, y1;

    printf("%f %f %f\n",t,y[0],y[1]);
    for (step=0; step < nstep; step++) {
        t2 = t+h;
        y1 = y[1];

        /* solve the quadratic equation */
        y[1] = y1*(2.-h*t*y1)/(sqrt(1.+h*t2*y1*(2.-h*t*y1))+1.);
        y[0] += 0.5*h*(y1+y[1]);
        t = t2;
        printf("%f %f %f\n",t,y[0],y[1]);
    }
}

/* main program */
int main (void) {
    int k, nstep[]={20,200,2000};
    double pi, h[]={0.1,0.01,0.001}, y[2], y2[2], f1[2], f2[2];

    pi = acos(-1.);
    for (k=0; k < 3; k++) {
        y[0] = 0.; y[1] = 1.;
        midpoint(2, 1, h[k], -1., y, y2, f1, f2, pi_ode);
        adams(2, nstep[k]-1, 0, h[k], -1.+h[k], y, y2, f1, f2, pi_ode);
        fprintf(stderr,"AB error with h=%f: %g\n",h[k],fabs(pi-y[0]));

        y[0] = 0.; y[1] = 1.;
        midpoint(2, 1, h[k], -1., y, y2, f1, f2, pi_ode);
        adams(2, nstep[k]-1, 1, h[k], -1.+h[k], y, y2, f1, f2, pi_ode);
        fprintf(stderr,"ABM error with h=%f: %g\n",h[k],fabs(pi-y[0]));
```

```
        y[0] = 0.; y[1] = 1.;
        midpoint(2, 1, h[k], -1., y, y2, f1, f2, pi_ode);
        pi_moulton(nstep[k]-1, h[k], -1.+h[k], y);
        fprintf(stderr,"AM error with h=%f: %g\n",h[k],fabs(pi-y[0]));
    }

    return 0;
}
```

5. (Programming) In Homework 3, we studied the motion of a planet. A more direct approach to the same problem involves the equations of Newton's mechanics for the two-body problem:

$$x''(t) = -\frac{G\,M\,x(t)}{\left[x^2(t)+y^2(t)\right]^{3/2}}\,, \quad y''(t) = -\frac{G\,M\,y(t)}{\left[x^2(t)+y^2(t)\right]^{3/2}}\,, \tag{10}$$

where t is time, x and y are the planet coordinates with respect to the star, G is the gravitational constant, and M is the mass of the star. In terms of the orbit characteristics, $G\,M = \omega^2 a$, where ω is angular frequency, and a is the major semi-axis of the orbit. The appropriate initial conditions (for the planet position "in July") are

$$x(0) = a\,(1-\epsilon)\,, \quad x'(0) = 0 \tag{11}$$

$$y(0) = 0\,, \quad y'(0) = a\,\omega\sqrt{\frac{1+\epsilon}{1-\epsilon}}\,, \tag{12}$$

where ϵ is the orbit eccentricity.

Solve the initial-value problem numerically using the following methods:

(a) Adams-Bashforth of order 2

(b) Adams-Bashforth of order 2 as predictor followed by one correction step of Adams-Moulton of order 2

Take $\epsilon = 0.6$, $a = 1\,\text{AU}$, $\omega = 2\pi\,\frac{1}{\text{year}}$, the step size $h = \frac{1}{360}\,\text{year}$.

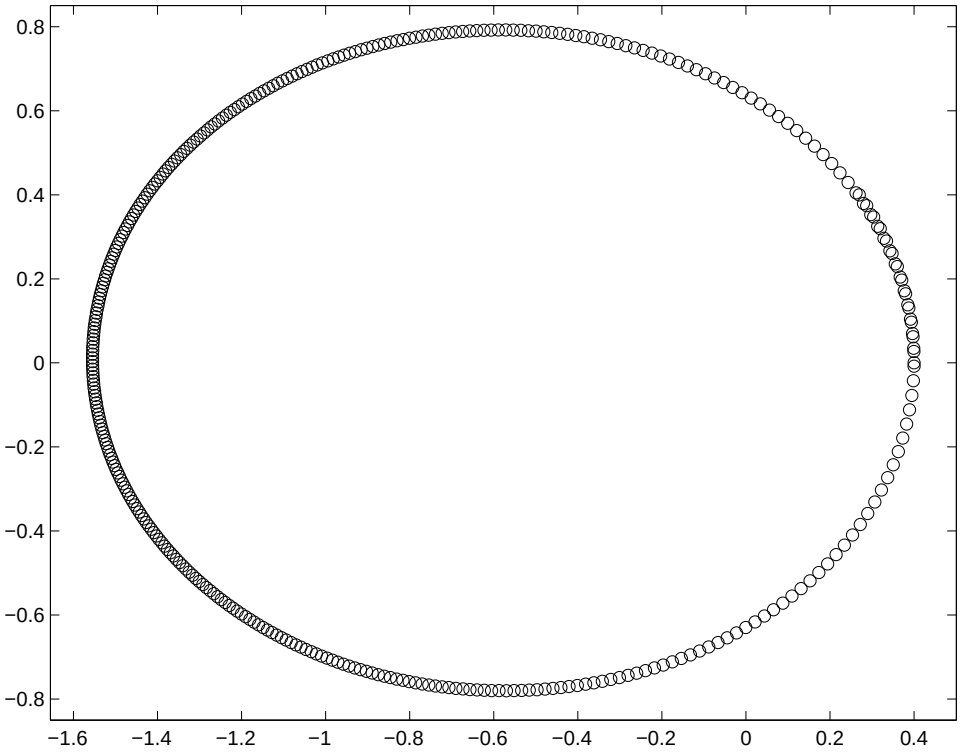
Plot the solution (planet trajectory) for $t \in [0,1]$. Does the periodicity hold? Output the error $x(k) - x(0)$ and $y(k) - y(0)$ for $k = 1,2,3$.

Answer:

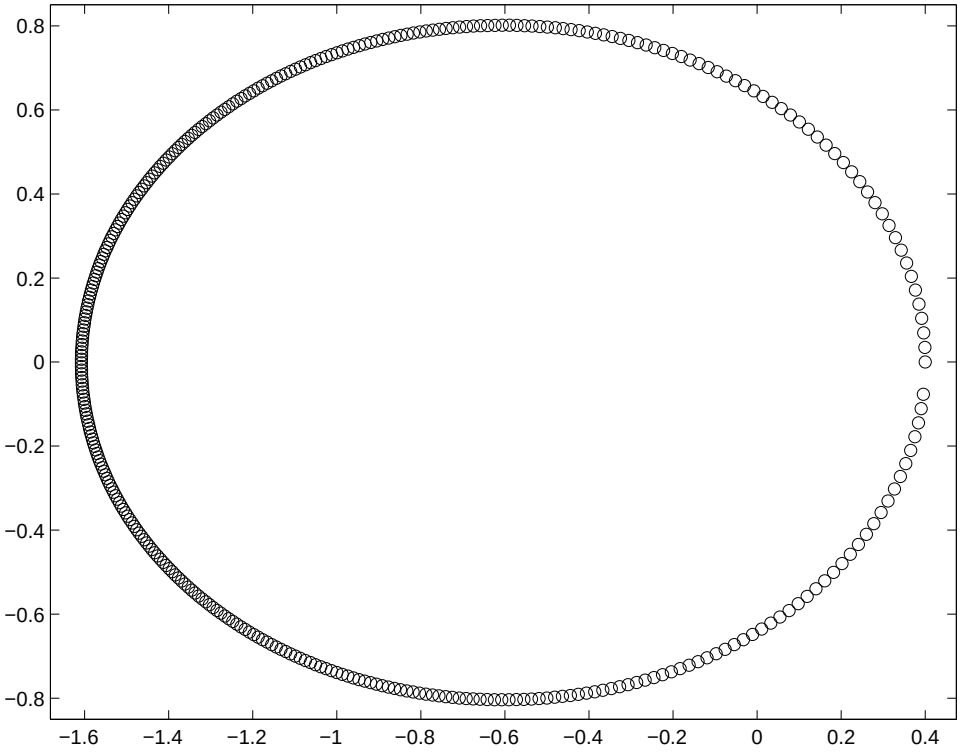
The period of the numerical solutions is erroneous compared to the exact period of one year. It is more stable in the case of the Adams-Bashforth-Moluton method.

	Adams-Bashforth		Adams-Bashforth-Moulton	
Time	x error	y error	x error	y error
1	-0.130388	0.399762	-0.00446057	-0.0766191
2	-0.360284	0.628241	-0.00954070	-0.112335
3	-0.572406	0.748445	-0.00870902	-0.108425

Adams-Bashforth:



Adams-Bashforth-Moulton:



Solution:

C program:

```
#include <stdio.h> /* for output */
#include <math.h>  /* for math functions */

#include "ode.h"

static const double a=1.;
static const double eps=0.6;
static double w;

/* right-hand side for the newton ODE system */
void newton (double x, const double* y, double* f)
{
    double r;

    f[0] = y[2];
    f[1] = y[3];

    r = y[0]*y[0]+y[1]*y[1];
    r *= sqrt(r);

    f[2] = - w*w*a*y[0]/r;
    f[3] = - w*w*a*y[1]/r;
}

/* main program */
int main (void) {
    int k, nstep=360, ncorr;
    double h, y0[4], y[4], y2[4], f1[4], f2[4];

    w = 2.*acos(-1.);
    h = 1./(double) nstep;
    y0[0] = a*(1.-eps);
    y0[1] = 0.;
    y0[2] = 0.;
    y0[3] = a*w*sqrt((1.+eps)/(1.-eps));

    for (ncorr=0; ncorr < 2; ncorr++) {
        for (k=0; k < 4; k++) {
            y[k] = y0[k];
        }
        midpoint(4, 1, h, 0., y, y2, f1, f2, newton);
        adams(4, nstep-1, ncorr, h, h, y, y2, f1, f2, newton);
        fprintf(stderr,"Error at t=1: %g %g\n",y[0]-y0[0],y[1]-y0[1]);
        for (k=1; k < 3; k++) {
            adams(4, nstep, ncorr, h, 0., y, y2, f1, f2, newton);
            fprintf(stderr,"Error at t=%d: %g %g\n",k+1,y[0]-y0[0],y[1]-y0[1]);
        }
    }

    return 0;
}
```