Math 128A Spring 2002 Handout # 26
Sergey Fomel April 30, 2002

Answers to Homework 10: Numerical Solution of ODE: One-Step Methods

---

1. (a) Which of the following functions satisfy the Lipschitz condition on y? For those that do, find the Lipschitz constant.

   i. $f(x,y) = \sqrt{x^2+y^2}$ for $x \in [-1,1]$
   ii. $f(x,y) = |y|$
   iii. $f(x,y) = \sqrt{|y|}$
   iv. $f(x,y) = |y|/x$ for $x \in [-1,1]$

   Answer:

   i. $f(x,y) = \sqrt{x^2+y^2}$ for $x \in [-1,1]$ satisfies the Lipschitz condition.
   It is proved by the following chain of equalities and inequalities:

$$\begin{aligned} |f(x,y_1) - f(x,y_2)| &= \left|\sqrt{x^2+y_1^2} - \sqrt{x^2+y_2^2}\right| = \frac{|y_1^2-y_2^2|}{\sqrt{x^2+y_1^2}+\sqrt{x^2+y_2^2}} \\ &= \frac{|y_1+y_2|}{\sqrt{x^2+y_1^2}+\sqrt{x^2+y_2^2}}\,|y_1-y_2| \\ &\le \frac{|y_1|+|y_2|}{\sqrt{x^2+y_1^2}+\sqrt{x^2+y_2^2}}\,|y_1-y_2| \le |y_1-y_2|\,. \end{aligned}$$

   The Lipschitz constant is 1.

   ii. $f(x,y) = |y|$ satisfies the Lipschitz condition.
   We have

$$f(x,y_1) - f(x,y_2) = |y_1| - |y_2|\,.$$

   From the equality

$$y_1 = y_2 + y_1 - y_2\,,$$

   it follows that

$$|y_1| \le |y_2| + |y_1 - y_2|$$

   and

$$|y_1| - |y_2| \le |y_1 - y_2|\,.$$

   From the equality

$$y_2 = y_1 + y_2 - y_1\,,$$

   it follows that

$$|y_2| \le |y_1| + |y_1 - y_2|$$

   and

$$-|y_1 - y_2| \le |y_1| - |y_2|\,.$$

Putting it together,

$$-|y_1 - y_2| \le |y_1| - |y_2| \le |y_1 - y_2|$$

or

$$||y_1| - |y_2|| \le |y_1 - y_2| \, .$$

This proves the Lipschitz condition. The Lipschitz constant is 1.

iii. $f(x,y) = \sqrt{|y|}$ does not satisfy the Lipschitz condition.
It is sufficient to consider a particular case $y_2 = 0$. Then

$$f(x,y_1) - f(x,y_2) = \sqrt{|y_1|}$$

and

$$\frac{|f(x,y_1) - f(x,y_2)|}{|y_1 - y_2|} = \frac{1}{\sqrt{|y_1|}}$$

The right-hand side is unbounded for y_1 approaching zero, which shows that the Lipschitz condition cannot be satisfied.

iv. $f(x,y) = |y|/x$ for $x \in [-1,1]$ does not satisfy the Lipschitz condition.
Again let us consider a particular case $y_2 = 0$. Then

$$\frac{|f(x,y_1) - f(x,y_2)|}{|y_1 - y_2|} = \frac{1}{|x|}$$

The right-hand side is unbounded for $x = 0$, which shows that the Lipschitz condition cannot be satisfied.

(b) Prove that the function $f(x,y) = -\sqrt{|1-y^2|}$ does not satisfy the Lipschitz condition and find two different solutions of the initial-value problem

$$\left\{ \begin{array}{rcl} y'(x) & = & -\sqrt{|1-y^2(x)|} \\ y(0) & = & 1 \end{array} \right. \tag{1}$$

on the interval $x \in [0,\pi]$.

Answer: To disprove the Lipschitz condition, let us consider the special case $y_2 = 1$. Then

$$f(x,y_1) - f(x,y_2) = -\sqrt{|1-y_1^2|}$$

and

$$\frac{|f(x,y_1) - f(x,y_2)|}{|y_1 - y_2|} = \frac{\sqrt{|1-y_1^2|}}{|1-y_1|} = \sqrt{\frac{|1+y_1|}{|1-y_1|}}$$

The right-hand side is unbounded for y_1 approaching 1, which shows that the Lipschitz condition cannot be satisfied.

Two different solutions of the initial-value problem are, for example,

$$y(x) = \cos x$$

and

$$y(x) = 1 \, .$$

2. Consider the initial-value problem

$$\begin{cases} y''(x) &= y(x) \\ y(0) &= y_0 \\ y'(0) &= y_1 \end{cases} \tag{2}$$

Write it as a system of two first-order differential equations with the appropriate initial conditions. Prove that Euler's method applied to this system can be unstable for a large step size.

Hint: Take the special case $y_1 = -y_0$.

Answer: Let us denote $y'(x)$ by $p(x)$. Then the initial-value problem takes the form of the system

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

with the initial conditions

$$\begin{cases} y(0) &= y_0 \\ p(0) &= y_1 \end{cases}$$

With application to this system, Euler's method is

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

or, in the matrix form,

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

In the special case $y_1 = -y_0$, the first step of Euler's method yields

$$\begin{bmatrix} y_1 \\ p_1 \end{bmatrix} = \begin{bmatrix} 1 & h \\ h & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix} = \begin{bmatrix} y_0 - h\, y_0 \\ h\, y_0 - y_0 \end{bmatrix} = (1-h) \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

Similarly, the next step is

$$\begin{bmatrix} y_2 \\ p_2 \end{bmatrix} = (1-h)^2 \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

and, after k steps, we will have

$$\begin{bmatrix} y_k \\ p_k \end{bmatrix} = (1-h)^k \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

The exact solution of the initial-value problem with $y_1 = -y_0$ is

$$\begin{bmatrix} y(x_k) \\ p(x_k) \end{bmatrix} = e^{-x_k} \begin{bmatrix} y_0 \\ -y_0 \end{bmatrix} = e^{-h\,k} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

While this solution is decaying at large x_k, Euler's solution will be unstable (increase in magnitude) if the step size $h > 2$, and $|1-h| > 1$. The stability region of Euler's method is $h \leq 2$.

3. Consider the initial-value problem

$$\begin{cases} y'(x) &= \lambda\, y(x) \\ y(0) &= y_0 \end{cases} \tag{3}$$

(a) Prove that the Taylor series method for this problem takes the form

$$y(x_{k+1}) \approx y_{k+1} = \left[1 + \lambda\, h + \frac{(\lambda\, h)^2}{2} + \cdots + \frac{(\lambda\, h)^n}{n!}\right] y_k\,, \tag{4}$$

where $h = x_{k+1} - x_k$, and n is the order of the method.

(b) Prove that every second-order Runge-Kutta method for this problem is equivalent to the second-order Taylor method.

(c) Prove that the second-order Taylor method can be unstable for large negative λ and find the stability region for the step size h.

Answer:

(a) Differentiating the equation directly, we obtain

$$\begin{aligned} y''(x) &= \lambda\, y'(x) = \lambda^2\, y(x)\,, \\ y'''(x) &= \lambda\, y''(x) = \lambda^3\, y(x)\,, \end{aligned}$$

and, by induction,

$$y^{(k)}(x) = \lambda^k\, y(x)\,.$$

The Taylor method of order n is

$$\begin{aligned} y_{k+1} &= y_k + h\, y_k'(x_k) + \frac{h^2}{2}\, y_k''(x_k) + \cdots + \frac{h^n}{n!}\, y_k^{(n)}(x_k) \\ &= y_k + \lambda\, h\, y_k + \frac{\lambda^2\, h^2}{2}\, y_k + \cdots + \frac{\lambda^n\, h^n}{n!}\, y_k = \left[1 + \lambda\, h + \frac{(\lambda\, h)^2}{2} + \cdots + \frac{(\lambda\, h)^n}{n!}\right] y_k\,. \end{aligned}$$

(b) We proved in class that the general form of the second-order Runge-Kutta method is

$$y_{k+1} = y_k + (1-B)\, h\, f(x_k, y_k) + B\, h\, f\left(x_k + \frac{h}{2\,B}, y_k + \frac{h}{2\,B}\, f(x_k, y_k)\right)\,,$$

where the constant B determines the particular method. For $f(x,y) = \lambda, y$, the method simplifies as follows:

$$\begin{aligned} y_{k+1} &= y_k + (1-B)\, h\, \lambda\, y_k + B\, h\, \lambda \left(y_k + \frac{\lambda\, h}{2\,B}\, y_k\right) \\ &= y_k + \lambda\, h\, y_k + \frac{(\lambda\, h)^2}{2}\, y_k = \left[1 + \lambda\, h + \frac{(\lambda\, h)^2}{2}\right] y_k\,. \end{aligned}$$

The last expression is equivalent to the previously found expression for the second-order Taylor method.

(c) The exact solution of the initial-value problem is

$$y(x_k) = y_0 \, e^{\lambda x_k} = y_0 \, e^{\lambda h k} \; .$$

It will decay for $\lambda h < 0$. The numerical second-order solution is

$$y(x_k) \approx y_k = \left[1 + \lambda h + \frac{(\lambda h)^2}{2}\right]^k y_0 \; .$$

The numerical solution will be unstable (increase in magnitude at each step) if

$$1 + \lambda h + \frac{(\lambda h)^2}{2} > 1$$

or

$$1 + \lambda h + \frac{(\lambda h)^2}{2} < -1 \; .$$

The second condition is never satisfied. The first condition is satisfied if $0 > \lambda h > -2$. Therefore, the stability condition for negative λ is

$$h \leq -\frac{2}{\lambda}$$

4. (Programming) The exact solution of the initial-value problem

$$\left\{ \begin{array}{rcl} y'(x) & = & f(x,y) = y^2(x) \, e^{-x} \\ y(0) & = & 1 \end{array} \right. \tag{5}$$

is

$$y(x) = e^x \; . \tag{6}$$

Solve the problem numerically on the interval $x \in [0,1]$ using

(a) Euler's method

$$y_{k+1} = y_k + h \; f(x_k, y_k) \tag{7}$$

(b) Second-order Taylor method

$$y_{k+1} = y_k + h \; f(x_k, y_k) + \frac{h^2}{2} \left[\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \; f(x_k, y_k)\right] \tag{8}$$

(c) Midpoint method

$$y_{k+1} = y_k + h \; f\left[x_k + \frac{h}{2}, y_k + \frac{h}{2} \; f(x_k, y_k)\right] \tag{9}$$

Take the step size $h = 0.1$ and output the error at all steps of the computation.

Answer:

x	Euler	Taylor	Midpoint
0.1	0.005171	0.000171	0.000298
0.2	0.011917	0.000397	0.000693
0.3	0.020605	0.000694	0.001208
0.4	0.031675	0.001077	0.001876
0.5	0.045656	0.001569	0.002732
0.6	0.063187	0.002195	0.003823
0.7	0.085027	0.002989	0.005205
0.8	0.112086	0.003991	0.006947
0.9	0.145447	0.005248	0.009134
1.0	0.186395	0.006822	0.011869

Solution:

C program:

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

/* example function */
double func(double x, double y)
{
    double f;
    f = y*y*exp(-x);
    return f;
}

/* main program */
int main (void)
{
    double x, euler, taylor, midpoint, exact;
    double h=0.1;
    int k, n=10;

    euler=taylor=midpoint=1.;
    for (k=0; k < n; k++) {
        x = k*h;
        exact = exp(x+h);
        euler += h*func(x,euler);
        taylor += h*func(x,taylor)*(1.+0.5*h*(2.*taylor*exp(-x)-1.));
        midpoint += h*func(x+0.5*h,midpoint+0.5*h*func(x,midpoint));

        printf("%f \t %f \t %f \t %f\n",x+h,
               exact-euler,exact-taylor,exact-midpoint);
    }

    return 0;
}
```

5. (Programming) In 1926, Volterra developed a mathematical model for predator-prey systems. If R is the population density of prey (rabbits), and F is the population density of predators (foxes), then Volterra's model for the population growth is the system of ordinary differential equations

$$
\begin{aligned}
R'(t) &= a\,R(t) - b\,R(t)\,F(t)\,; \qquad (10)\\
F'(t) &= d\,R(t)\,F(t) - c\,F(t)\,, \qquad (11)
\end{aligned}
$$

where t is time, a is the natural growth rate of rabbits, c is the natural death rate of foxes, b is the death rate of rabbits per one unit of the fox population, and d is the growth rate of foxes per one unit of the rabbit population.

Adopt the midpoint method for the solution of this system. Take $a = 0.03$, $b = 0.01$, c=0.01, and $d = 0.01$, the interval $t \in [0, 500]$, the step size $h = 1$ and the initial values

(a)

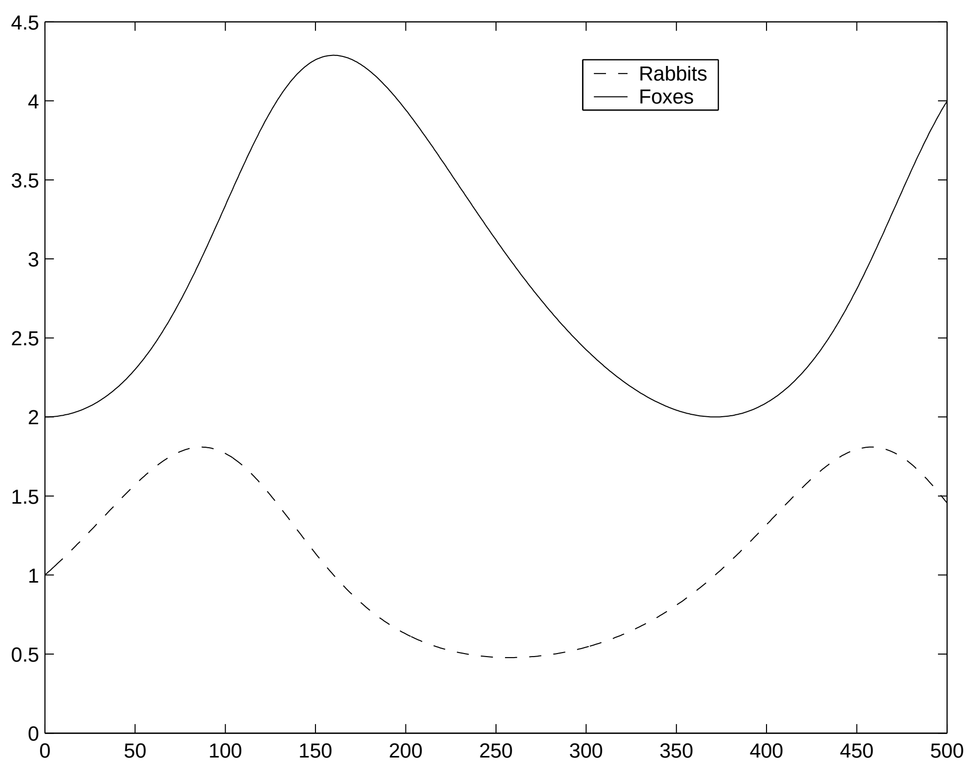
$$
\begin{aligned}
R(0) &= 1.0\,;\\
F(0) &= 2.0
\end{aligned}
$$

(b)

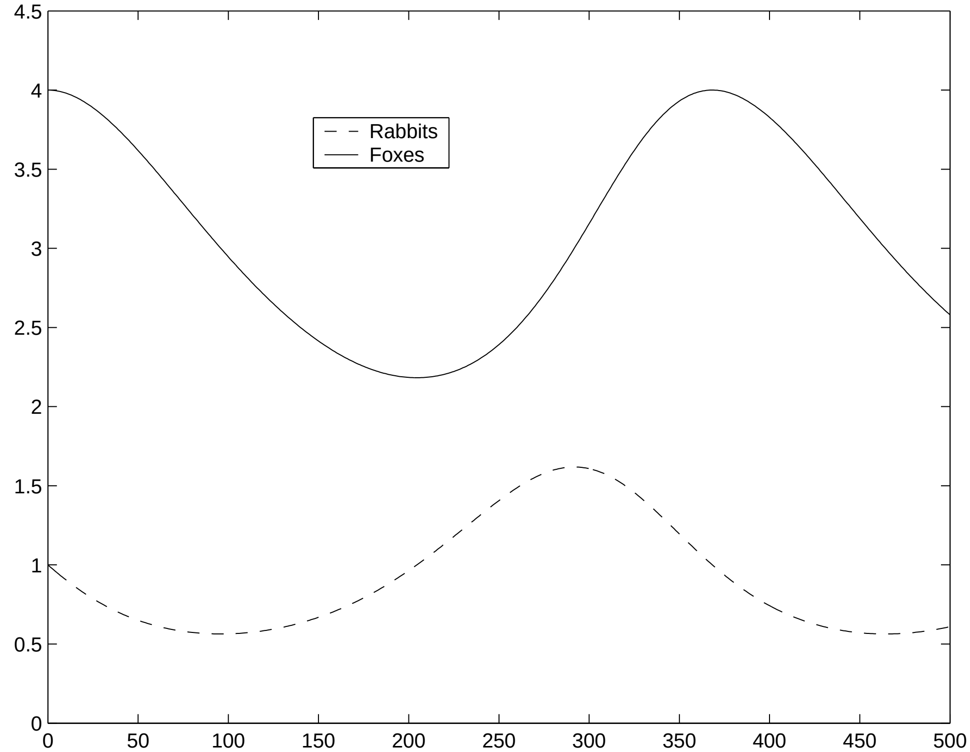
$$
\begin{aligned}
R(0) &= 1.0\,;\\
F(0) &= 4.0
\end{aligned}
$$

Plot the solution: functions $R(t)$ and $F(t)$.

Answer:



(a)



(b)

Solution:

C program:

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

/* Function: volterra
   ------------------
   Computes the right-hand side of Volterra’s equations
   rf[2] - rabbits and foxes
   f[2]  = the right-hand side
*/
void volterra (double t, const double* rf, double* f)
{
    static const double a=0.03, b=0.01, c=0.01, d=0.01;

    f[0] = a*rf[0] - b*rf[0]*rf[1];
    f[1] = d*rf[0]*rf[1] - c*rf[1];
}

/* 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
   t            - starting value of the variable
   y[n]         - function values
   f1[n], f2[n] - storage
   slope(t,y,f) - function pointer for the right-hand side
*/
void midpoint (int n, int nstep, double h, double t, double* y,
               double* f1, double* f2,
```

```
              void (*slope)(double t, const double* y, double* f))
{
    int k, step=0;

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

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

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

        slope(t+0.5*h,f1,f2);

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

/* main program */
int main (void) {
    int nstep=500;
    double h=1.;
    double y1[]={1.,2.}, y2[]={1.,4.};
    double f1[2], f2[2];

    midpoint(2, nstep, h, 0., y1, f1, f2, volterra);
    midpoint(2, nstep, h, 0., y2, f1, f2, volterra);

    return 0;
}
```