

# MATH 417: Numerical Methods

## Partial answers for homework assignment 11

**Problem 2 (Numerical solution of a vector-valued ODE).** The problem is stated as a differential equation of order 2, i.e. in the equation we have second derivatives of the distance variable  $d(t)$ . This is not according to the standard form, so we have to introduce an auxiliary variable. We define this variable, which is a velocity, through  $v(t) = d'(t)$ . The full ODE system in standard form then reads

$$\begin{aligned}d'(t) &= v(t), & d(0) &= 0, \\v'(t) &= f(t, d(t)), & v(0) &= 0,\end{aligned}$$

where  $f(t, d(t)) = \frac{F(t)}{m(t)}$ , with  $F(t)$  depending on the distance  $d(t)$  again.

The following short, and mostly self-explanatory program solves this problem:

```
#include <iostream>

double m(double t)
{
    return (t<600 ? 1.-0.9*t/600 : 0.1);
}

double T(double t)
{
    return (t<600 ? 12 : 0);
}

double G(double h, double t)
{
    return -6.371e6*6.371e6*10*m(t)/h/h;
}

double F(double t, double h)
{
    return T(t) + G(h,t);
}

double a(double t, double h)
```

```

{
  return F(t,h) / m(t);
}

int main ()
{
  double h = 6.371e6;
  double v = 0;

  double dt = .001;
  double t = 0;
  do
  {
    // compute f(x(k))
    const double f_v = a(t,h);
    const double f_h = v;

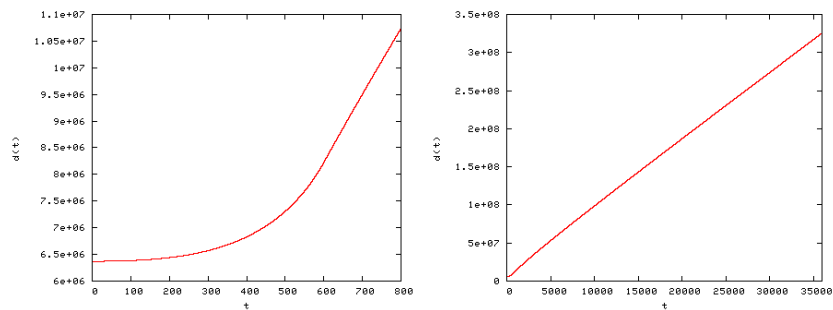
    // compute x(k+1)
    v = v+dt*f_v;
    h = h+dt*f_h;

    t += dt;

    std::cout << t << ' ' << h << ' ' << v << std::endl;
  }
  while (t<36000);
}

```

If we plot the distance from earth, we get the following graphs (left an inset up to time  $t = 800$ , at the right for the full time span):



What can be seen from these graphs is that the rocket really starts to gain height only as the time approaches  $t = 600$ . That is not particularly surprising: at the beginning, the thrust has to accelerate the entire rocket including all

the leftover fuel. As time progresses, the mass of the rocket becomes smaller and smaller as fuel is burnt; since the thrust stays the same, this means that the acceleration gets larger and larger. After the rocket's engines shut down at  $t = 600$ , the rocket flies on a ballistic trajectory, i.e. its path is changed only by gravity. Because it is already fast enough, it is slowed down by gravity but continues its journey upward (and will eventually escape earth's gravity field).

As for determining the altitude of the rocket at  $t = 36000$  accurately: with a time step of  $h = 1$  as used in the program shown above, the distance  $d(36000)$  from the earth core after 10 hours of flight is  $3.2534 \cdot 10^8$ , i.e. roughly 325,000km, or almost as far away from earth as the moon is. In order to determine how accurate this number is, let us compute it with a sequence of decreasingly smaller time steps:

$h$	$d(36000)$
100	$-5.29531 \cdot 10^9$
10	$2.91001 \cdot 10^8$
1	$3.21895 \cdot 10^8$
0.1	$3.24974 \cdot 10^8$
0.01	$3.25341 \cdot 10^8$
0.001	$3.25317 \cdot 10^8$

(The first line is obviously nonsensical, but so is the time step of 100 seconds.) For an accuracy of 100 meter, we need to know  $d(36000)$  up to 6 digits beyond the decimal point. This is obviously hard to achieve using this method, given that for  $h = 0.001$  we seem to have at best four digits beyond the decimal point accurately. Because we reduce the error by a factor of 10 each time we reduce  $h$  by a factor of 10 (this method is linearly convergent, after all), we gain one digit of accuracy in each such step; comparing with the accuracy we have with  $h = 0.001$ , this means that to determine  $d(36000)$  accurately to within 100 meters would require a step size  $h \approx 10^{-5}$ ; to use such a step size would make the program run for approximately 8 hours on my, already pretty fast, desktop – an exercise I leave to you. It may serve as an illustration that linearly convergent methods are usually not particularly fast, and that it is usually beneficial to use a higher order method; for example, the quadratically convergent Crank-Nicolson method gains 2 digits of accuracy with each reduction of  $h$  by a factor of 10, and the fourth order Runge-Kutta method gains a whole 4 digits each time. It is apparent, that we can get away with a significantly larger  $h$  for these methods than what is required for the explicit Euler method used here.

**Problem 3 (A modeling challenge).** To solve this challenge, we subdivide the time into two phases: for  $0 \leq t \leq t'$ , the turkey runs on the ground, at  $t'$  it achieves liftoff, and for  $t' < t \leq t''$  it is airborne. At  $t''$  it hits the ground again. For each of the two intervals, we can write down a set of ODEs.

Let's first look at the running phase: During this time, it is on the ground,

i.e. at  $h = 0$  and has no vertical velocity. That means that

$$\begin{aligned} h(0) &= 0, & u(0) &= 0, \\ h'(t) &= u(t), & u'(t) &= 0. \end{aligned}$$

That's a complete description for both variables in standard form. (We could have said  $h(t) = u(t) = 0$ , but that's not the form we want for the ODE.) On the other hand, the turkey is initially at rest at  $x = 0$ , and then accelerated linearly:

$$\begin{aligned} x(0) &= 0, & v(0) &= 0, \\ x'(t) &= v(t) & v'(t) &= 1.5. \end{aligned}$$

For the second phase, i.e.  $t > t'$ , we have that

$$\begin{aligned} x'(t) &= v(t), \\ v'(t) &= -\frac{v(t)^2}{10}, \\ h'(t) &= u(t), \\ u'(t) &= -4 + v(t). \end{aligned}$$

In summary, we get the following ODE system in standard form:

$$\begin{aligned} x'(t) &= v(t), & x(0) &= 0, \\ v'(t) &= \begin{cases} 1.5 & \text{for } t \leq t', \\ -\frac{v(t)^2}{10}, & \text{for } t > t', \end{cases} & v(0) &= 0, \\ h'(t) &= u(t), & h(0) &= 0, \\ u'(t) &= \begin{cases} 0 & \text{for } t \leq t', \\ -4 + v(t) & \text{for } t > t', \end{cases} & u(0) &= 0. \end{aligned}$$

Note that the real equations only concern the horizontal and vertical accelerations  $v', u'$ , and that the first and third equations only define the velocities in terms of the time derivatives of the displacements.

What we haven't specified yet is what that time  $t'$  is at which liftoff occurs. That's hard to do in a formula, but it's easy in a program: we simply record the first time step at which the horizontal velocity exceeds the threshold. In other words, we operate under the assumption that  $t < t'$  until we determine that the velocity is large enough – only then do we know what the value of  $t'$  is, and that for all times afterwards,  $t > t'$ .

We can solve this model numerically, for example using the forward (explicit) Euler method. The following code does exactly this:

```
#include <iostream>

// A flag that describes whether the turkey is airborne, i.e. t>t'
```

```

bool is_airborne = false;

// a function to return the horizontal acceleration
double horizontal_acceleration (const double v)
{
    if (is_airborne == false)
        return 1.5;
    else
        return -v*v/10;
}

// same for the vertical acceleration
double vertical_acceleration (const double v)
{
    if (is_airborne == false)
        return 0;
    else
        return -4+v;
}

// integrate the ODE using the explicit Euler method
int main ()
{
    // first define the variables and
    // their initial values
    double x=0, v=0, h=0, u=0;

    // then the (constant) time step and the time
    const double dt = 0.001;
    double t = 0;

    // then have variables to store
    // time and place of liftoff and
    // touchdown
    double liftoff_t, liftoff_x;
    double touchdown_t, touchdown_x;

    // next iterate until the stopping
    // criterion of touchdown is
    // satisfied (checked inside the
    // body of this loop)
    while (true)

```

```

{
    // compute f(x(k))
    const double f_x = v;
    const double f_v = horizontal_acceleration(v);
    const double f_h = u;
    const double f_u = vertical_acceleration(v);

    // compute x(k+1) = x(k)+hf(x(k))
    x = x+dt*f_x;
    v = v+dt*f_v;
    h = h+dt*f_h;
    u = u+dt*f_u;

    // increment the time variable as well
    t = t+dt;

    // output present solution
    std::cout << t << ' '
                << x << ' '
                << v << ' '
                << h << ' '
                << u << std::endl;

    // check whether the turkey was
    // already airborne and has now
    // touched down (its height
    // above ground has become
    // negative); if so, record
    // time and position and exit
    // the loop:
    if ((is_airborne == true) && (h<0))
    {
        touchdown_t = t;
        touchdown_x = x;
        break;
    }

    // alternatively, if it hasn't
    // been airborne yet, see if
    // the horizontal speed exceeds
    // the takeoff velocity. if so,
    // record time and position and
    // mark the turkey as airborne:
    if ((is_airborne == false) && (v>=8))
    {
        liftoff_t = t;

```

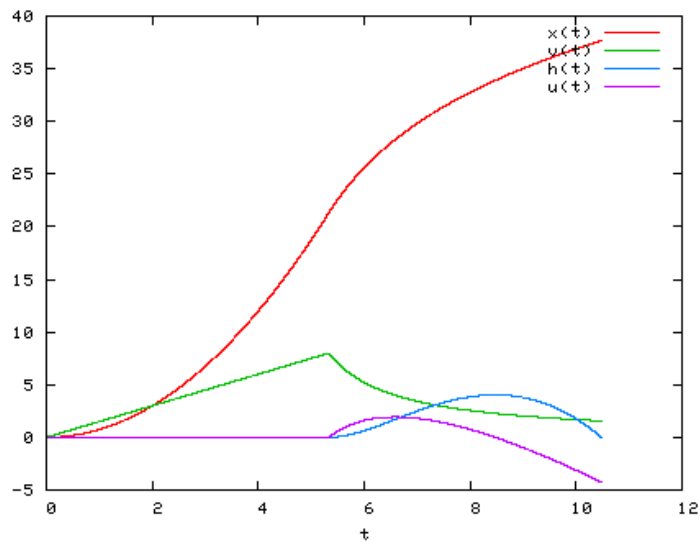
```

        liftoff_x = x;
        is_airborne = true;
    }
}

// at the end of the day, print
// when and where the turkey lifted
// off and touched down:
std::cout << "Turkey achieved liftoff at t=" << liftoff_t
<< ", x=" << liftoff_x << std::endl
<< "Turkey touched down again at t=" << touchdown_t
<< ", x=" << touchdown_x << std::endl;
}

```

With this program, we can compute the graph showing the trajectory of the turkey shown in the problem description, or we can plot all variables as functions of time together:



In addition, the program produces this output:

```

Turkey achieved liftoff at t=5.334, x=21.3347
Turkey touched down again at t=10.481, x=37.6619

```

In other words, the turkey was airborne for 5.147 seconds, during which it covered a distance of 16.33m. From the data the program produces, we can also infer that it's maximal height over ground was 4.03m, not so bad for a turkey!

As a sidenote: the time and distance before liftoff can, of course, be computed in an even simpler way. For this, realize that before liftoff, we have  $x''(t) = 1.5$  and  $h(t) = 0$ . This means, the turkey is on the ground, and the exact solution of

the first equation is  $x(t) = \frac{1.5}{2}t^2, v(t) = 1.5t$ . The liftoff speed is reached when  $1.5t = 8$ , i.e. at  $t = 16/3 = 5.3333$  and  $x = \frac{1.5 \cdot 16^2}{2 \cdot 3^2} = \frac{384}{18} = \frac{64}{3} = 21.33333$  (the numbers produced by our numerical integrator and stated above are therefore correct to at least three digits, even if not exact). Beyond liftoff time, however, the equation becomes nonlinear and there is little hope that we will be able to find an analytic solution to also compute touchdown time and distance exactly.