MAT301: Applied Mathematics &AI
Part 3 - Numerical solution of differential equations

Dr Craig Stark

Term 2, 2017

Aims and objectives

Aims: *You should*

1. Appreciate the need for numerical techniques to solve differential equations and their widespread application.
2. Understand the concept of the Runge-Kutta Method to solve ODEs.
3. Understand the concept of the finite difference method to solve PDEs.

Objectives: *You should be able to demonstrate that you can*

1. Numerically solve ODEs, including coupled sets of ODEs, using the Runga-Kutta method.
2. Solve simple PDEs using the finite difference method such as the wave equation in 1D.

1 Introduction

In the last section, we studied some of the many methods for finding analytical solutions for certain ordinary and partial differential equations. However, only the most elementary ordinary and partial differential equations have *closed-form* solutions, that is, solutions which can be expressed in terms of the fundamental functions of mathematics. Even when closed-form solutions are available, the value of such a solution can be questionable. In these scenarios, the only way forward is to solve the differential equations *numerically*. Numerical procedures to solve differential equations largely decompose into either techniques which aim to find the value of the solution at a finite number of nodes within the domain of definition of the equation and its boundary, or alternatively, techniques which expand the solution as a sum of basis functions defined on the domain of definition of the equation, and then aim to find the coefficients in this expansion. In this course we will focus on the former class of numerical procedures, and look at two powerful techniques: the Runge-Kutta method, for solving ordinary differential equations; and, the Finite Difference Method, for solving

partial differential equations. These numerical techniques are very popular and enjoy widespread use in many different areas to simulate phenomena in astronomy, physics, chemistry and biology, to name but a few. For us, numerical techniques are of importance for simulating physics in computer game physics engines; such as, the motion of objects described by Newton's second law (see Applied Mathematics 1 and 2); the rotation of rigid bodies described by Euler's equations (see Applied Mathematics 4 next year!); and the dynamics of deformable objects (soft-body dynamics) using the differential equation governing damped simple harmonic motion.

2 Runge-Kutta method

The Runge-Kutta method is a numerical technique used for solving ordinary differential equations. You came across this technique in Applied Mathematics 2, where you considered simple examples in which you worked out the numerical solution to ordinary differential equations by hand. Here, we are going to go a step further and use the Runge-Kutta methods in the context of computer simulations.

Consider the ordinary differential equation (ODE) of the form

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y). \tag{3.1}$$

To numerically solve this form of ODE, the fourth-order Runge-Kutta algorithm is

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h/2, y_n + k_1/2) \\ k_3 &= hf(x_n + h/2, y_n + k_2/2) \\ k_4 &= hf(x_n + h, y_n + k_3) \end{aligned}$$

and $h = x_{n+1} - x_n$; then,

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

This is the fourth-order Runge-Kutta method: in each step the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these derivatives the final function value (shown as a filled dot in Figure 1) is calculated.

The underlying idea behind the RK algorithm is to rewrite the dy's and dx's in Equation (3.1) as finite steps Δy and Δx, and multiply the equations by Δx: $\Delta y = \Delta x f(x,y)$. This gives algebraic formulas for the change in the functions when the independent variable x is "stepped" by one "stepsize" Δx. This means that the k_i's are in fact guesses of the solution of the ODE, y. Let's look at a couple of examples of the

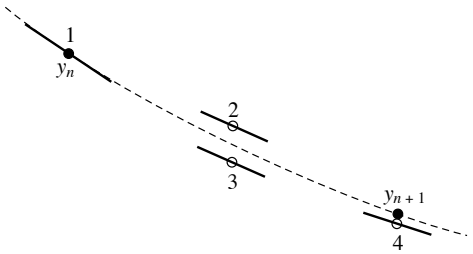

Figure 1: Fourth-order Runge-Kutta method (Numerical recipes in Fotran 77).

Ex.3.1
Ex.3.2
Ex.3.3
Ex.3.4

RK algorithm in action.

Example

A body of mass 2 kg is projected on a rough horizontal table that exerts a resistance given by $5v/2$ Newtons, where v is the velocity. Initially the velocity of the mass is 8 m/s. Find the velocity v of the body as a function of time t.

Solution

First, let's find the analytical solution to this problem so that we can compare it to the numerical solution to check if it is correct. The resistive force that opposes the motion of the body is given by

$$F(v) = -\frac{5v}{2}$$

Using Newton's second law gives us

$$\begin{aligned} F(v) &= m\frac{\mathrm{d}v}{\mathrm{d}t}, \\ -\frac{5v}{2} &= 2\frac{\mathrm{d}v}{\mathrm{d}t}, \end{aligned}$$

Using the separation of variables method,

$$\begin{aligned} \int \frac{\mathrm{d}v}{v} &= -\frac{5}{4}\int \mathrm{d}t, \\ \ln v &= -\frac{5}{4}t + c, \\ v &= Ae^{-\frac{5}{4}t}. \end{aligned}$$

The initial velocity of the body is 8 m/s, so our initial conditions are, $v(0) = 8$, and so $A = 8$; therefore,

$$v = 8e^{-\frac{5}{4}t}$$

How about the numerical solution? The governing ODE is

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\frac{5v}{4} = -av \quad \equiv \quad \frac{\mathrm{d}y}{\mathrm{d}x} = f(x,y).$$

where $a = 5/4$. This differential equation is of the same form as Equation (3.1), where we are using t instead of x, v instead of y and $f = -av$. Therefore, the fourth-order Runge-Kutta algorithm is

$$\begin{aligned} k_{v1} &= -hav_n \\ k_{v2} &= -ha(v_n + k_{v1}/2) \\ k_{v3} &= -ha(v_n + k_{v2}/2) \\ k_{v4} &= -ha(v_n + k_{v3}) \\ v_{n+1} &= v_n + \frac{k_{v1}}{6} + \frac{k_{v2}}{3} + \frac{k_{v3}}{3} + \frac{k_{v4}}{6} \end{aligned}$$

I have chosen to write my RK algorithm in MATLAB - you can use any programming language you like. In its simplest form, barring specific syntax, the RK algorithm will be very similar going from one programming language to another. The key feature of the code is the *for-loop* (or equivalently a *do-loop*), that repeats the numerical scheme for each step n until a maximum number of steps, $n_{\max}$, is reached. The stepsize is h and so for $n_{\max}$, the total time interval covered is $t_{\max} = n_{\max}h$. The RK scheme, takes the initial conditions of the problem, i.e. a value of v at $n = 1$, and then iteratively propagates the solution forward in time for increasing values of n until $n_{\max}$ is reached.

Example code using the RK algorithm to numerically solve the resistive motion problem is:

```
%total number of steps the code will march the solution through
nmax=500;
h=0.1;
a=5/4;
%defining the size of the solution array.
v = zeros(1,nmax);
%initial conditions: we need something to start calculating the ...
    solution from.
v(1) = 8;
% begin the iterative calculations until n_max.
for n = 1:nmax-1;
%
    kv1 = -h*a*v(n);
%
    kv2 = -h*a*(v(n)+0.5*kv1);
%
    kv3 = -h*a*(v(n)+0.5*kv2);
%
    kv4 = -h*a*(v(n)+kv3);
%
    v(n+1) = v(n) + kv1/6 +kv2/3 + kv3/3 +kv4/6;
%
end
plot([1:nmax]*h,v)
hold on
plot([1:nmax]*h,8*exp(-(5/4)*[1:nmax]*h),'r')
hold off
grid on
```

The resulting solution from the RK algorithm is shown in Figure 2. If we plot the analytical solution calculated previously on the same set of axes the two solutions would be indistinguishable (or at the very least within the expected error of the numerical solution).

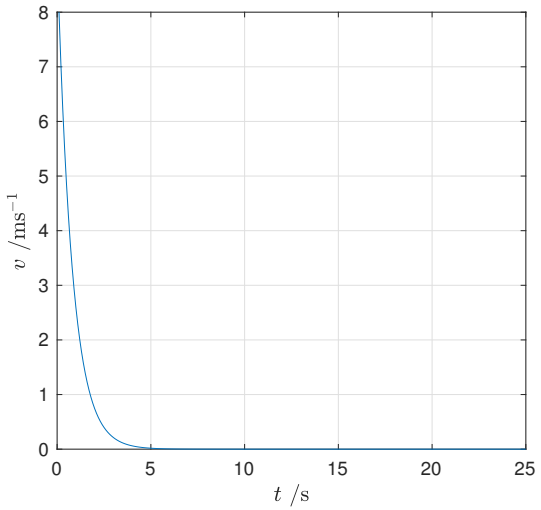

Figure 2: Resistive motion numerical solution using the RK algorithm.

2.1 Simple harmonic motion

Th RK algorithm as stated appears to only apply to first order differential equations but in fact the algorithm can be used to solve higher order ordinary differential equations. Consider the ordinary differential equation, governing SHM,

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx$$

This is a second order ODE: how do we use the RK algorithm to solve this numerically? Higher order equations should be reduced to a set of coupled first-order equations and solved in parallel. In the case of SHM, we set $v = \dot{x}$; therefore, we obtain the coupled set of ODEs that describe SHM:

$$\begin{aligned} \frac{\mathrm{d}v}{\mathrm{d}t} &= -\frac{k}{m}x = -bx \\ \frac{\mathrm{d}x}{\mathrm{d}t} &= v \end{aligned}$$

where $b = k/m$. Now that we have reduced the original second order ODE into two first order ODEs, we proceed as before and solve both using the RK algorithm in parallel. The critical part to notice is the cross-pollination of the k_{xi} values into the algorithm for the $\dot{v}$ equation; and the k_{vi} values into the algorithm for the $\dot{x}$ equation. Hence,

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v \qquad\qquad \frac{\mathrm{d}v}{\mathrm{d}t} = -bx$$

$$\begin{aligned} k_{x1} &= hv_n & k_{v1} &= -hbx_n \\ k_{x2} &= h(v_n + k_{v1}/2) & k_{v2} &= -hb(x_n + k_{x1}/2) \\ k_{x3} &= h(v_n + k_{v2}/2) & k_{v3} &= -hb(x_n + k_{x2}/2) \\ k_{x4} &= h(v_n + k_{v3}) & k_{v4} &= -hb(x_n + k_{x3}) \end{aligned}$$

and

$$\begin{aligned} x_{n+1} &= x_n + \frac{k_{x1}}{6} + \frac{k_{x2}}{3} + \frac{k_{x3}}{3} + \frac{k_{x4}}{6}, \\ v_{n+1} &= v_n + \frac{k_{v1}}{6} + \frac{k_{v2}}{3} + \frac{k_{v3}}{3} + \frac{k_{v4}}{6}. \end{aligned}$$

The following is example code of the implementation of the RK algorithm for SHM:

```
%SHM using RK4
nmax = 500;
h = 0.1;
b = 0.1;
%
x = zeros(1,nmax);
v = zeros(1,nmax);
%
x(1) = 10;
v(1) = 6;
%
for n = 1:nmax-1;
%
    kx1 = h*v(n);
    kv1 = -h*b*x(n);
%
    kx2 = h*( v(n)+0.5*kv1 );
    kv2 = -h*b*( x(n)+0.5*kx1 );
%
    kx3 = h*( v(n)+0.5*kv2 );
    kv3 = -h*b*( x(n)+0.5*kx2 );
%
    kx4 = h*( v(n)+kv3 );
    kv4 = -h*b*( x(n)+kx3 );
%
    x(n+1) = x(n) + kx1/6 +kx2/3 + kx3/3 + kx4/6;
    v(n+1) = v(n) + kv1/6 +kv2/3 + kv3/3 + kv4/6;
%
end
plot([1:nmax]*h,x)
hold on
plot([1:nmax]*h,v,'r')
```

```
33 hold off
34 grid on
```

For initial conditions $x(t=0) = 10$ m and $v(t=0) = 6$ ms^{-1}, the numerical solution is:

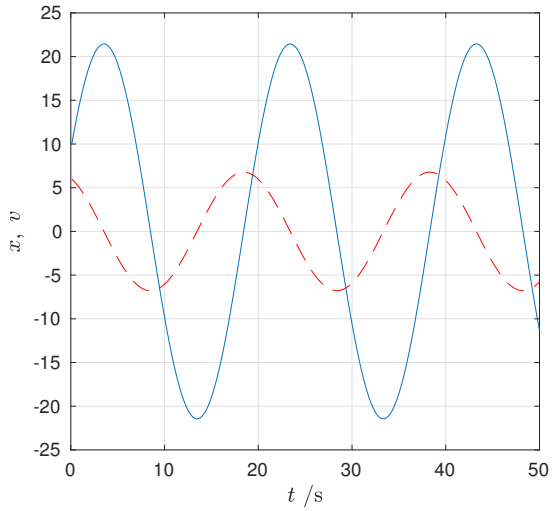

Figure 3: SHM: red-dashed curve - v; blue curve - x.

Have a go comparing the numerical solution to the analytical solution that you calculated in Applied Mathematics 2 for the equation of SHM.

3 Finite difference method

Finite difference methods approximate the derivatives appearing in a PDE by sums and differences of function values at a set of discrete points, usually uniformly spaced with respect to each independent variable. The idea is most effectively illustrated in one dimension. Let x and h be real numbers where h is usually regarded as a small positive step length, then

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x-h)}{2h} + O(h^2) \\ f''(x) &= \frac{f(x+h)-2f(x)+f(x-h)}{h^2} + O(h^2) \end{aligned}$$

are two expressions which are often used to compute the first and second order derivatives of f at x from knowledge of $f(x-h)$, $f(x)$ and $f(x+h)$. These results are obtained by calculating the Taylor Series expansion of f in the vicinity of x so that

$$\begin{aligned} f(x+h) &= f(x)+hf'(x)+\frac{h^2}{2}f''(x)+\frac{h^3}{6}f'''(x)+O(h^4) \\ f(x-h) &= f(x)-hf'(x)+\frac{h^2}{2}f''(x)-\frac{h^3}{6}f'''(x)+O(h^4) \end{aligned} \tag{3.2}$$

Then,

$$\begin{aligned} f(x+h)-f(x-h) &= 2hf'(x)-\frac{h^3}{3}f''' + O(h^5) \\ f(x+h)-2f(x)+f(x-h) &= 2h^2f'' + O(h^4) \end{aligned}$$

from which it follows that

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x-h)}{2h}+O(h^2) \\ f''(x) &= \frac{f(x+h)-2f(x)+f(x-h)}{h^2}+O(h^2) \end{aligned}$$

These are known as the central difference formulae for the first and second derivatives of f. The formulae are "second order" accurate by which we mean that the error incurred by approximating $f'(x)$ or $f''(x)$ by these formulae is proportional to h^2. This error is commonly called the truncation error in the sense that it is the mathematical error which is incurred by truncating the Taylor series to obtain the finite difference approximations. In fact, it is possible to find higher order central difference formulae but these will not be used in this lecture course. Using the same method as above, we can also calculate *forward* and *backward* difference formulae respectively,

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x)}{h} \\ f'(x) &= \frac{f(x)-f(x-h)}{h} \end{aligned}$$

Forward and backward difference formulae have a greater truncation error but can be useful in finding a stable numerical scheme. In some cases, direct substitution of the finite difference approximations into a PDE is sufficient, but in general care needs to be taken as the stability of the resulting algorithm can be in question. We will look at a particular method (the Lax-Wendroff method) for solving hyperbolic PDEs. For details about the precise algorithms for solving parabolic and elliptic PDEs, please refer to Mitchell & Griffiths (see booklist in the introduction to these notes).

Ex.3.5
Ex.3.6
Ex.3.7
Ex.3.8

3.1 Wave propagation in 1D

Consider the partial differential equation,

$$\partial_t u + a\partial_x u = 0, \tag{3.3}$$

where the partial derivative $\partial/\partial t$ is denoted by ∂_t; and, $\partial/\partial x$ is denoted by ∂_x. This PDE describes a wave travelling in the positive x-direction at constant speed a.

To obtain a finite difference solution of (3.3) the domain described by the PDE is defined in terms of a rectilinear grid, its edges parallel to the x and t axes. Defining discrete coordinates so that an arbitrary grid point is given by $(x,t) = (mh, nk)$ such that $m \in [0, m_{max}]$ and $n \in [0, n_{max}]$, where m, n are integers and h, k are the grid spacings in the x, t coordinates respectively. Writing $u(nk, mh) = u_m^n$ the finite difference formulae are obtained from the Taylor expansion of u about t in the neighbourhood of k, while holding x fixed,

$$u_m^{n+1} \approx \left(1 + k\partial_t + \frac{1}{2}k^2\partial_t^2\right) u_m^n \tag{3.4}$$

This is second-order accurate in time. The expression contains an inherent diffusion term, ∂_t^2, that helps dampen any numerical jitter and potential instabilities. For the case of the linear wave equation $\partial_t u_m^n$ is replaced by $-a\partial_x u_m^n$ and $\partial_t^2 u_m^n$ by $a^2\partial_x^2 u_m^n$ using (3.3). Therefore,

$$u_m^{n+1} = \left(1 - ka\partial_x + \frac{1}{2}k^2a^2\partial_x^2\right) u_m^n \tag{3.5}$$

Using the finite difference approximations:

$$\partial_x u_m^n = \frac{u_{m+1}^n - u_{m-1}^n}{2h} + O(h^2) \tag{3.6}$$

$$\partial_x^2 u_m^n = \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{h^2} + O(h^2) \tag{3.7}$$

this yields the Lax-Wendroff algorithm for the linear wave equation,

$$u_m^{n+1} = (1-p^2a^2)u_m^n - \frac{1}{2}pa(u_{m+1}^n - u_{m-1}^n) + \frac{1}{2}p^2a^2(u_{m+1}^n + u_{m-1}^n) \tag{3.8}$$

where $p = k/h$ is the mesh ratio. This algorithm is second-order accurate in space and time. This numerical scheme is stable provided $0 < p|a| \leq 1$ (the proof of this is beyond the scope of this course) i.e. as long as the product pa is less than 1, say $pa = 0.5$, then the algorithm will be stable. The Lax-Wendroff algorithm calculates the solution of the PDE u at each spatial point m at time $n+1$, by using the discretized solution at the previous time step n. It marches forward in time n in this way until it reaches the maximum time step $m_{\max}$.

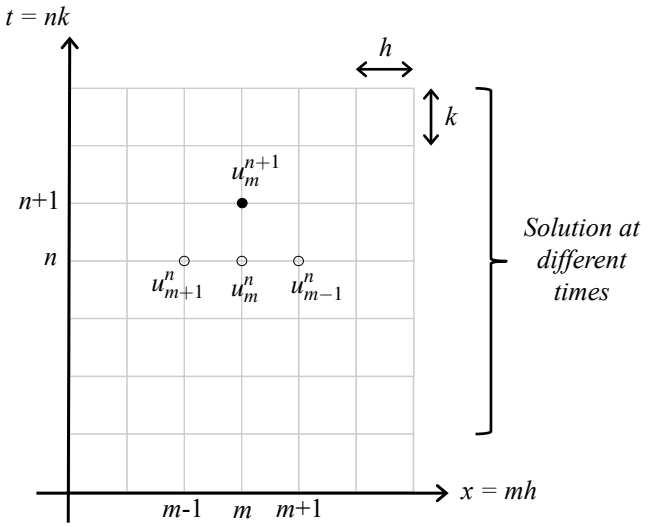

Figure 4: Discrete coordinates so that an arbitrary grid point is given by $(x,t) = (mh,nk)$ such that $m \in [0,m_{max}]$ and $n \in [0,n_{max}]$, where m, n are integers and h, k are the grid spacings in the x, t coordinates respectively.

The following example code solves the 1D wave equation using the Lax-Wendroff algorithm for the initial conditions,

$$u(n=0,m) = A_0 \exp(-s(m-m_0)^2) \qquad m \in [0,m_{\max}]$$

where $m_0 = \frac{1}{2}m_{\max}$, $s = 5\times10^4$, $A_0 = 0.5$ and $m_{\max} = 600$. Code:

```
%
p = 0.75;
a = 0.4;
nmax = 600;
mmax = 1000;
m0 = mmax/2;
s=5e-4;
A0=0.5;
u=zeros(nmax,mmax);
%
for m = 1:mmax
    u(1,m) = A0*exp(-s*(m-m0)^2 );
end
%
for n = 1:nmax-1
    for m = 2:mmax-1
        u(n+1,m) = (1-(p*a)^2)*u(n,m) ...
                    -0.5*p*a*(u(n,m+1)-u(n,m-1)) ...
```

```
                    +0.5*((p*a)^2)*(u(n,m+1)+u(n,m-1));
    end
end
for n=1:nmax
    plot(u(n,:));
    drawnow;
end
```

Figure 5 shows the initial conditions for the numerical solutions (solid line), and the numerical solution at later times n (dashed and dot-dash line respectively) exhibiting the evolution of the wave form and its propagation to the right.

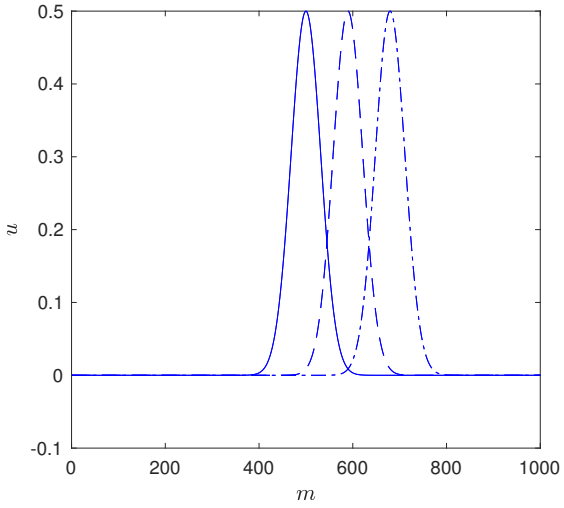

Figure 5: Wave propagation in 1D. Initial wave profile (solid line) moves to the right. Dash and dot-dash curves show wave at increasing values of time respectively.

Exercises

Exercise 3.1 (§ 2)

Using the Runge-Kutta method, numerically solve the following ordinary differential equation that describes damped simple harmonic,

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx - bv,$$

where $b/m = 0.1$, $k/m = 0.5$; and, with initial conditions $x(t = 0) = 10$ m and $v(t = 0) = 6\ \mathrm{ms}^{-1}$. Plot v and x as functions of time t. You may use a programming language of your choice.

Answer

Expected output:

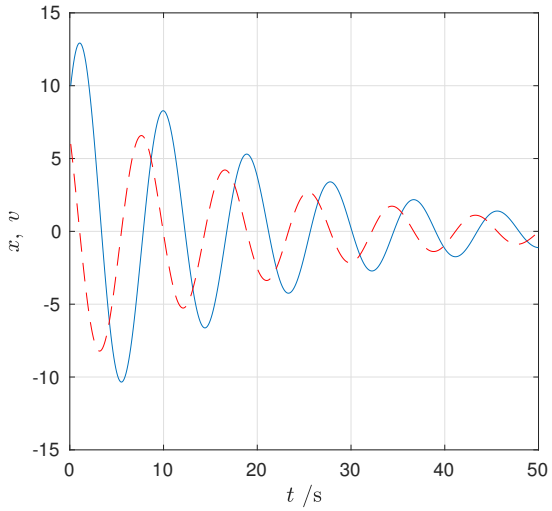

Figure 6: SHM with damping: red-dashed curve - v; blue curve - x.

Exercise 3.2 (§ 2)

Numerically solve the following differential equation governing the motion of a freely falling body near the surface of the Earth where $g = 9.8$ ms^{-2},

$$m\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -mg,$$

The initial conditions of the projectile are $x(t = 0) = 0$ m and $v(t = 0) = 20$ ms^{-1}. Plot v and x as functions of time t. You may use a programming language of your choice.

Answer

Expected output:

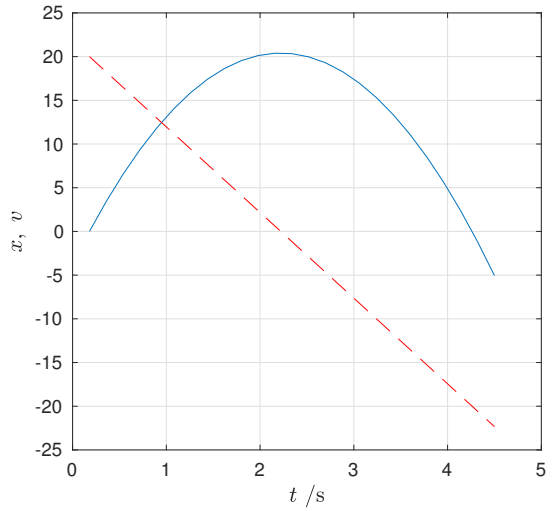

Figure 7: Freely falling body: red-dashed curve - v; blue curve - x.

Exercise 3.3 (§ 2)

Consider the ordinary differential equation that governs projectile motion in 2 dimensions,

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}(x,y) = -m(0,g)$$

The initial conditions of the projectile are $x(t=0) = 0$ m and $v_x(t=0) = v_y(t=0) = 10\ \mathrm{ms}^{-1}$. Plot the trajectory of the projectile. You may use a programming language of your choice.

Answer

Expected output:

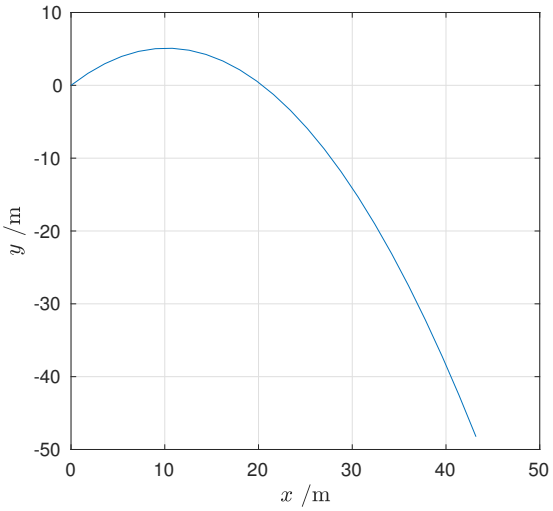

Figure 8: Projectile motion.

Exercise 3.4 (§ 2)

Consider the ordinary differential equation that governs projectile motion with air resistance in 2 dimensions,

$$m\frac{\mathrm{d}^2}{\mathrm{d}t^2}(x,y) = -m(0,g) - mk(v_x,v_y)$$

The initial conditions of the projectile are $x(t=0) = 0$ m and $v_x(t=0) = v_y(t=0) = 10\ \mathrm{ms}^{-1}$ and $k = 0.5$. Plot the trajectory of the projectile. You may use a programming language of your choice.

Answer

Expected output:

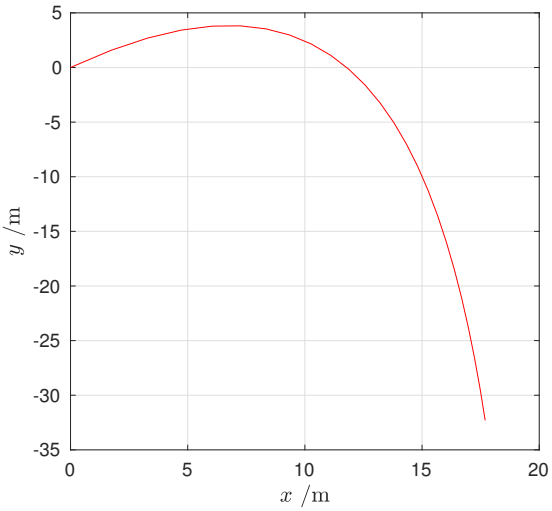

Figure 9: Projectile motion with air resistance.

Exercise 3.5 (§ 3)

Using the following finite difference expressions,

$$\begin{aligned}\partial_x^2 u_m^n &= \frac{u_{m+1}^n - 2u_m^n + u_{m-1}^n}{h^2} + O(h^2),\\ \partial_t^2 u_m^n &= \frac{u_m^{n+1} - 2u_m^n + u_m^{n-1}}{k^2} + O(k^2),\end{aligned}$$

numerically solve the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2},$$

where c is the wave speed. For the initial conditions use a sinusoidal wave profile of the form:

$$u(n=0,m) = A_0 \sin\left(\pi \frac{(m-m_0-\delta)}{\delta}\right) \exp\left(-\sigma \frac{(m-m_0)^2}{\delta}\right) \quad m \in [\frac{m_{\max}}{2} - \delta, \frac{m_{\max}}{2} + \delta]$$

where $A_0 = 0.5$, $m_0 = \frac{1}{2} m_{\max}$, $\delta = 0.1 m_{\max}$, $\sigma = 5 \times 10^{-1}$ and $m_{\max} = 1000$. Plot the wave solution as a function of n.

Answer

Sample Solution:

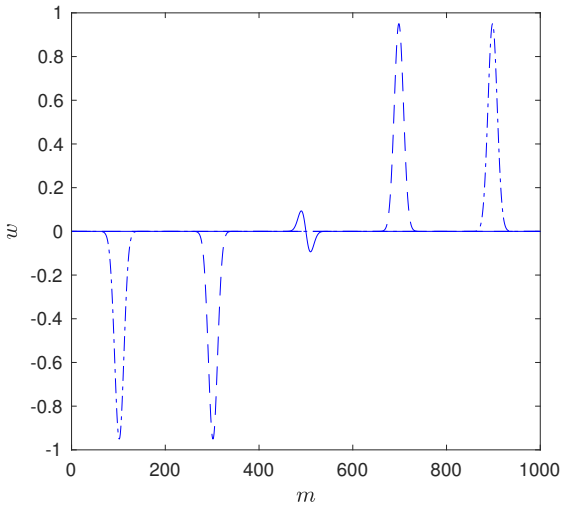

Figure 10: Wave equation. The initial wave profile (set at the centre of the domain $m_0 = \frac{1}{2}m_{\max}$) moves to the right and left. Dash and dot-dash curves show wave at increasing values of time respectively.

Exercise 3.6 (§ 3)

Following the approach to the numerical solution of wave propagation in 1D, numerically solve nonlinear wave equation in 1D:

$$\partial_t u + u\partial_x u = 0,$$

for the initial conditions:

$$u(n = 0, m) = A_0 \exp\left(-\sigma(m - m_0)^2\right),$$

where $A_0 = 0.5$, $\sigma = 1 \times 10^{-4}$, $m_{\max} = 1000$ and $m_0 = \frac{1}{2}m_{\max}$. Plot the solution u as a function of time n.

Answer

Sample Solution:

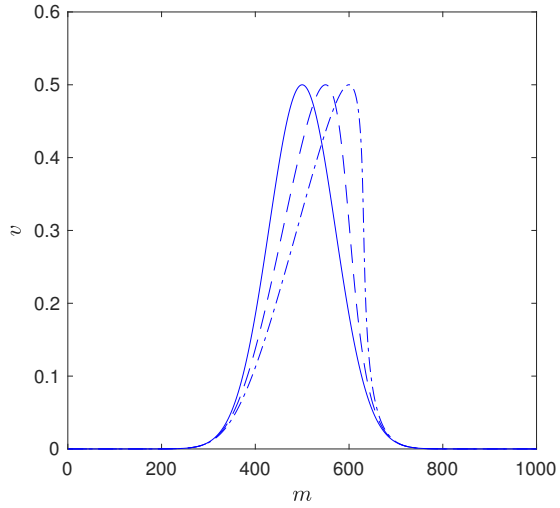

Figure 11: Nonlinear wave equation. Dash and dot-dash curves show wave at increasing values of time respectively.

Exercise 3.7 (§ 3)

Using the finite difference method, numerically solve the wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2},$$

where the central point in the spatial domain oscillates sinusoidally with time. Plot u as a function of time n.

Answer

Sample Solution:

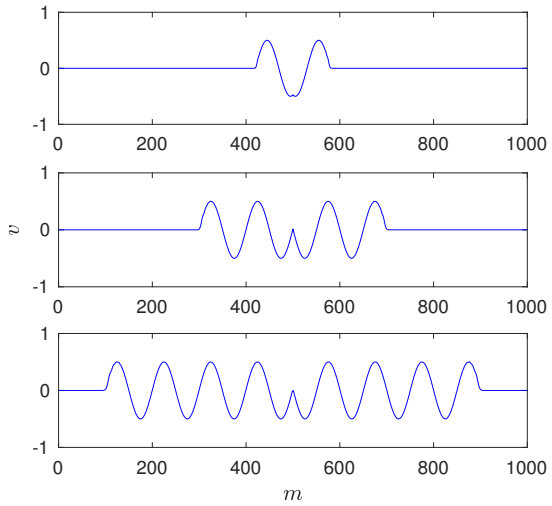

Figure 12: Numerical solution of the wave equation with oscillating central point. Going form the top panel to the bottom panel the time n increases.

Exercise 3.8 (§ 3)

Using the finite difference method, numerically solve the wave equation

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2},$$

where the boundaries of the domain are set to be always equal to zero and with the following initial condition:

$$u(t=0,m) = A_0 \sin\left(\frac{am\pi}{m_{\max}}\right) \quad m \in (0, m_{\max})$$

for arbitrary A_0 and $a \in [1,16]$ is an integer. Plot u as a function of time n.

Answer

Sample Solution:

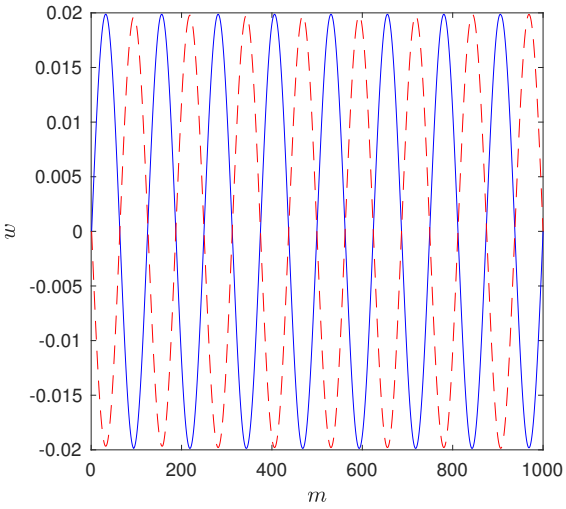

Figure 13: Numerical solution of the wave equation for standing waves. The two curves (dash and solid) show the extremities of the oscillating standing waves.