MathLab: Problem Set 2

Solutions

Mathematics Club IITM

Aditi Vaidya and Pranjal Varshney

June 2024

- The solutions to the 3 problems in Problem Set 2 can be found in this solution sheet.
- Feel free to reach out to us for doubts! Contact information of the solution-set creators:
  - Aditi Vaidya - +91 95451 49055
  - Pranjal Varshney - +91 77806 61602

Problem 1: A Spin to Euler's Method

We've walked you through Euler's method in the session. We even saw some slight modifications to Euler's method. Make a few of your own changes to Euler's method, list them and examine their errors by making substitutions in Taylor's series. Write a few points of analysis about this new method. Remember that it doesn't have to be a better method, it must simply be different.

Final Expected Submission: Description of ***your new variant*** of Euler's method, error analysis of ***your method*** and a few points analysing other features of ***your method***.

Solution:

While there are multiple approaches to this question, this is an example of what was expected. We are considering the differential equation to be function of only one variable.

Summary of Heun's Method:

Given the initial value problem

$$\frac{dy}{dt} = f(t), \quad y(t_0) = y_0,$$

we want to approximate the solution at $t_{n+1} = t_n + h$, where h is the step size.

1. Predictor Step (Euler's Method):

$$y_{n+1}^P = y_n + hf(t_n).$$

2. Corrector Step:

$$y_{n+1} = y_n + \frac{h}{2}\left[f(t_n) + f(t_{n+1})\right].$$

Our plan:

In Heun's method, we weigh the slopes at the initial point and the predicted point equally. Instead of doing so we can introduce a parameter α that let's us make any choice of weights.

Given the initial value problem

$$\frac{dy}{dt} = f(t), \quad y(t_0) = y_0,$$

we want to approximate the solution at $t_{n+1} = t_n + h$, where h is the step size, and α is a parameter that determines the weight of the corrector step.

1. Predictor Step (Euler's Method):

$$y_{n+1}^P = y_n + hf(t_n).$$

2. Corrector Step:

$$y_{n+1} = y_n + h\left[(1-\alpha)f(t_n) + \alpha f(t_{n+1})\right].$$

Error Analysis:

Local Truncation Error (LTE)

The local truncation error (LTE) is the error made in a single step of the numerical method.

To analyze the LTE, we consider the Taylor series expansion of the exact solution:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + O(h^3)$$

Since $y'(t) = f(t)$ and $y''(t) = f'(t)$:

$$y(t_{n+1}) = y(t_n) + hf(t_n) + \frac{h^2}{2}f'(t_n) + O(h^3)$$

Substituting the predictor value $y_{n+1}^P = y_n + hf(t_n)$ into the corrector step, we have:

$$\alpha f(t_{n+1}) = \alpha f(t_n + h)$$

Using a Taylor expansion:

$$f(t_n + h) = f(t_n) + hf'(t_n) + O(h^2)$$

Thus,

$$\alpha f(t_{n+1}) \approx \alpha\left(f(t_n) + hf'(t_n)\right)$$

The corrector step becomes:

$$y_{n+1} \approx y_n + h\left[(1-\alpha)f(t_n) + \alpha\left(f(t_n) + hf'(t_n)\right)\right]$$

Simplifying,

$$y_{n+1} \approx y_n + hf(t_n) + \alpha h^2 f'(t_n)$$

Comparing with the exact solution expansion, the LTE is:

$$LTE \approx \frac{h^2}{2} f'(t_n) - \alpha h^2 f'(t_n)$$

$$LTE \approx \left(\frac{1}{2} - \alpha\right) h^2 f'(t_n)$$

Note that at $\alpha = 0.5$ the LTE is proportional to h^3 since the h^2 term goes to zero.

Global Truncation Error (GTE)

The global truncation error (GTE) is the accumulation of the local truncation error over all steps. Assuming the LTE is proportional to h^2, the GTE over N steps is:

$$GTE \approx N \cdot LTE = \frac{T}{h} \cdot \left(\frac{1}{2} - \alpha\right) h^2 f'(t)$$

$$GTE \approx Th \left(\frac{1}{2} - \alpha\right) f'(t)$$

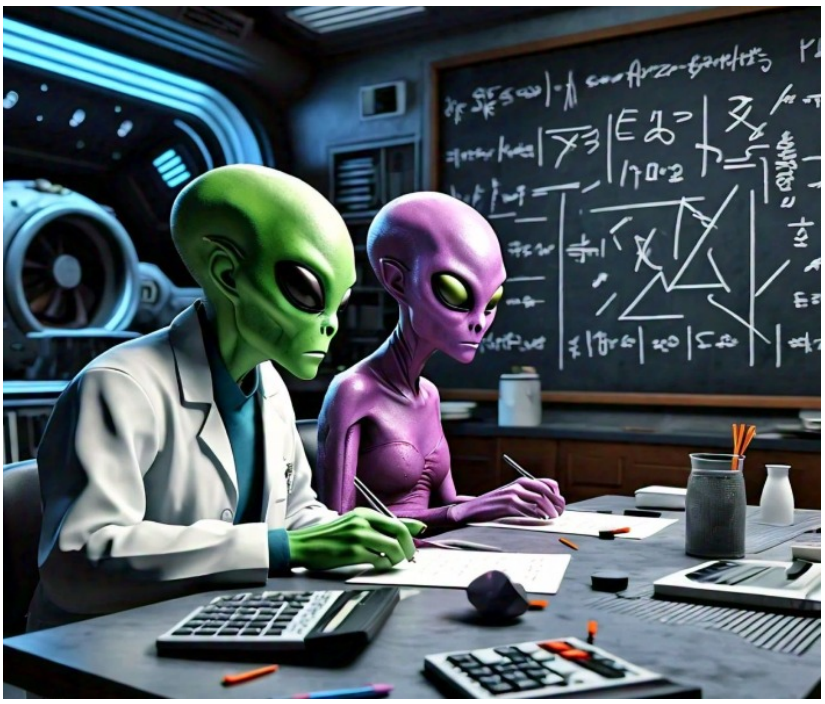
where T is the total time interval.

A Few Points of Analysis:

- A numerical method is stable if small errors introduced at any step (due to round-off errors or approximation errors) do not grow uncontrollably and lead to significant deviations from the true solution over time. Increasing α (closer to 1) can potentially improve stability by placing more emphasis on the corrector step, which uses a more accurate estimate of the future state.
- Changing the value of α for the same problem results in errors ranging from the order of h^2 (Euler's) to h^3 (Heun's) depending on the error we are comfortable with.

Problem 2: A New Approximation

The species of MathLand (some of which descended to Earth to conduct MathLab) are far too evolved, they believe that straight lines are elementary and outdated! They prefer neater splines to make approximations. Unfortunately they don't have any computational wisdom and are stuck! They need an algorithm that will help them integrate a function. Can you be their saviour and help them write the code to do so?



Hint: Be creative and try something new or Think Simpsons!

If you're developing your own method write out all the approximations and assumptions clearly, Remember: Taylor is always your friend and the approach always carries more importance than the final answer.

Final Expected Submission: A script written in `MATLAB` or `GNU Octave` that implements the following:

- Your own new algorithm for approximating integrals OR Simpson's method for calculating a definite integral.
- Tests out your algorithm on some integral (for example $\int_0^2 x^2 dx$ or any integral of your choice for that matter) and see if your integrator is working.

Note:

1. You are free to use any example function which has to be integrated in your code to demonstrate that your integrator works.
2. If you are submitting your code via a Google Drive link make sure to enable access for all, so that we will be able to access your submission.

Solution:

We will use the Taylor series approximation to find the integral.

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \mathcal{O}((x-a)^3)$$

Instead of truncating it at the first derivative, we add one more term resulting in a quadratic approximation.

`MATLAB/GNU Octave` code :

```
% This is a function used to integrate x^2
% If you want your integrator to to work on any function, use the package
↪  symbolic, it will find the derivatives for you.

function integral_approx = integrate_x2(lower_limit, upper_limit, step_size)
    f = @(x) x^2;

    f_prime = @(x) 2*x;
    f_double_prime = @(x) 2;

    % Initialize the integral approximation and the current x value
    integral_approx = 0;
    x_val = lower_limit;

    while x_val < upper_limit
        % Ensures that we don't go beyond the upper limit in the last step
        h = min(step_size, upper_limit - x_val);

        % Compute the Taylor series expansion up to the second derivative
        integral_approx += h * (f(x_val) + h / 2 * f_prime(x_val) + (h^2) / 6 *
        ↪  f_double_prime(x_val));

        % Update the current x value
        x_val += h;
    end
end
```

This is just one of the ways to compute an integral. If your code properly integrates a function using Simpson's or your own method you will get a good grade.

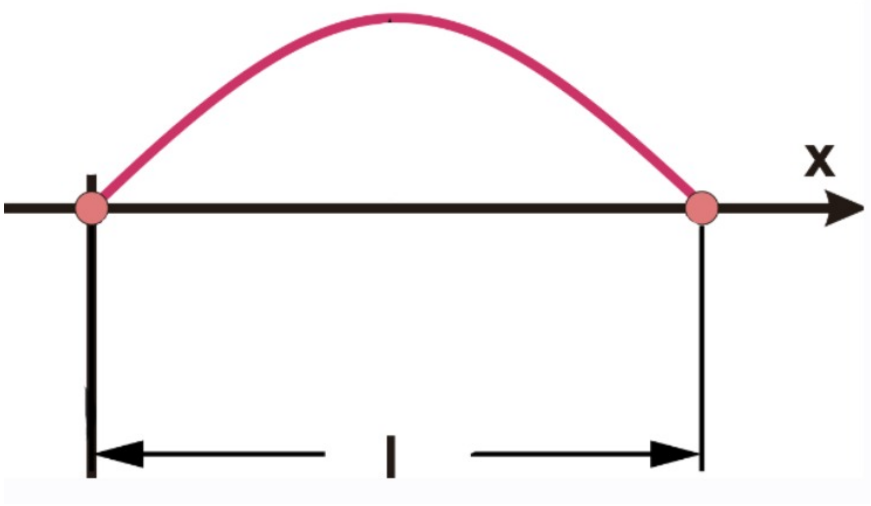
Problem 3: Satisfying Curiosity Numerically

Pranjal is playing with a string. He ties the ends of it between $x = 0$ and $x = L$. Let $u(x, t)$ represent the displacement of a point at a distance x at time t of the vibrating string. Pranjal is curious to know what the displacement of a point on the string is, at some time. Can you help him find out?

It has been found out that u satisfies the equation

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

It's given that $c = 1, L = 1$, and the initial condition is $u(x, 0) = 2x(1 - x)$ (that is the displacement of each point x when $t = 0$). The string is released from rest at $t = 0$.



Plot the displacement (u) vs. time (t) graph of the points on the string at $T = 1$ second, by writing a code on `GNU Octave` or `MATLAB`. Use $dx = 0.1$ units and $dt = 0.1$ seconds.

Submit your code and the plot you get.

Hint 1: Use the Method of Differences that has been taught to you in Part 2 of Session 2 to get an equation.

Hint 2: Its given that the string is released from rest, hence $\frac{\partial u}{\partial t} = 0$ at $t = 0$. Use Central difference approximation to get an equation.

Final Expected Submission: A script written in `MATLAB` or `GNU Octave` that implements the following:

- Solves the given problem using numerical methods in accordance with the given parameters.
- Plots the u vs. t graph as asked in the question.

and the plot that is generated by your script.

Note:
If you are submitting your code and plot via Google Drive links make sure to enable access for all, so that we will be able to access your submission.

Solution:

We have been given the PDE ($c = 1$),

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

Applying Central Difference Approximation to both terms :

$$\frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{(\Delta x)^2} = \frac{u(x,t+\Delta t) - 2u(x,t) + u(x,t-\Delta t)}{(\Delta t)^2}$$

$$u(x,t+\Delta t) = r^2[u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t] + 2u(x,t) - u(x,t-\Delta t),$$

where $r = \frac{\Delta x}{\Delta t}$ So,

$$u_{i,j+1} = r^2(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) + 2u_{i,j} - u_{i,j-1} \tag{1}$$

Observing this on a stencil, you require 2 rows of values of u to find all successive rows. Using the initial condition given to us $u_{i,0} = 2i\Delta x(1 - i\Delta x)$, we get u values of $j = 0$ row.

Putting in $j = 0$ in 1, we get,

$$u_{i,1} = r^2(u_{i+1,0} - 2u_{i,0} + u_{i-1,0}) + 2u_{i,0} - u_{i,-1} \tag{2}$$

What do we do about $u_{i,-1}$??
Since the string is just left, $\frac{\partial u}{\partial t} = 0$, at $t = 0$. Applying Central Difference Approximation to $\frac{\partial u}{\partial t}$, we get

$$\frac{u(x,t+\Delta t) - u(x,t-\Delta t)}{2\Delta t} = 0,$$

at $t = 0$, or simply, at $j = 0$,

$$u_{i,j+1} = u_{i,j-1} \Rightarrow u_{i,1} = u_{i,-1}$$

Substituting this in 2, we get

$$u_{i,1} = \frac{r^2}{2}(u_{i+1,0} - 2u_{i,0} + u_{i-1,0}) + u_{i,0} \tag{3}$$

Using this, we get u values of the $j = 1$ row.

Hence, we know our $j = 0$ and $j = 1$ rows, and using these 2 and equation 1, we can find all succeeding rows.

The `MATLAB` / `GNU Octave` code is given in the next page.

MATLAB/GNU Octave code :

```
% Initialize data
L = 1;         % length of rod
T = 1;         % the differential equation is solved till 1 sec elapses
dx = 0.1;
dt = 0.1;
N = L/dx;      % number of subintervals
M = T/dt;      % number of timesteps
r = dt/dx;
x = (0 : 1 : N) * dx;    % position array
u0 = 2*x.*(1-x);         % initial condition

% Partial Difference Equation, finding the j=1 row
for i = 2:N
  u1(i) = (r^2/2)*(u0(i+1) - 2*u0(i)+u0(i-1)) + u0(i);
end

% Setting boundary conditions
u1(1) = 0;
u1(N+1) = 0;

% Finding the next row using the 2 rows below
for j = 2:M
    for i = 2:N
        u2(i) = r^2*(u1(i+1) - 2*u1(i) + u1(i-1)) + 2*u1(i) - u0(i);
    end
    u2(1) = 0;    % Setting boundary conditions
    u2(N+1) = 0;
    u0 = u1;      % For finding succeeding rows
    u1 = u2;
end

% Plot solution
plot(x, u2);
xlabel("Position(x)", "fontsize", 17)
ylabel("Displacement (u)", "fontsize", 17)
title("Displacement (u) vs position (x) of points on the string at T = 1 sec",
↪  "fontsize", 20)
```

Output graph: