Lab V : Solving partial differential equations, waves in strings

In this lab, you are going to code up a simulation in time of a traveling wave on an ideal string with fixed boundary conditions. As we saw in the lecture, the discrete equation that governs this simulation is

$$y(i, n+1) = r^2[y(i+1, n) + y(i-1, n)] + 2(1-r^2)y(i, n) - y(i, n-1)$$

where y is the vertical displacement of the i discrete segment, n is the time index and $r = c\Delta t/\Delta x$ with Δx being the spatial step and Δt the time step. c, here is the speed of the traveling waves on the string.

Our simulation code is going to perform the following tasks :

1. Initialize the profile
2. Evolve the profile in time in accordance with the equations of motion.
3. Plot the profile at every step.

It's a good idea to break these tasks into subroutines (functions) that e call from a main program. Let's call the main program `string_fixed.m`. We'll write two subrotines : one for setting up the initial profile (`initial_profile.m`) and the other for calculating the profile at the next step given the profiles for the present step and the previous step (`propagate.m`).

Let the initial profile be a Gaussian. A Gaussian usually has two degrees of freedom, its center and its width. `initial_profile`, which forms the initial profile takes in both of these in addition to an array of x-values and outputs the y-displacements in the following way :

```
## Sets up the initial profile as a Gaussian
## Usage : function y=initial_profile(x,x0,k)
## x is the array that contains values on the horizontal axis, x0 is the
## center of the Gaussian and k is a parameter that determines the width
## of Gaussian

function y=initial_profile(x,x0,k)

  y=exp(-k*(x-x0).^2);

endfunction
```

Here e need the `.` because `x` is an array and we need an element-by-element square.

`propagate` takes in the profiles as entire arrays from the present step and the previous step and calculates the profile in the next step.

```
## Takes in the previous and the present profiles and iterates to find
## the profile in the next time step.

function ynext=propagate(ynow,yprev,r)

  ## Quick and dirty way to fix boundary conditions -- for each step
  ## they are the same as the previous step.

  ynext=ynow;
  for i=2:length(ynow)-1
    ynext(i) = 2*(1-r^2)*ynow(i)-yprev(i)+r^2*(ynow(i+1)+ynow(i-1));
  endfor

endfunction
```

Let's analyze this code :

- This function takes two profiles (arrays), one corresponding to the time step n and the other corresponding to the time step $n-1$. Because y is characterized by to indices i and n, we could have stored all the time history of all the points on the string in a matrix, but this is unnecessary because we only need to consecutive steps at a time to calculate the next step. For this reason, we absorb the meaning of the time index n into names of arrays as in `ynext`, `yprev` and `ynow`.

- As seen in red, at every step of the `propagate` function, we assign all the elements of `ynext` to `ynow`. We are in fact doing this in order to fix the boundary conditions, i.e. at every time step, the first and last segments should remain the same. Becuase we are going to change the elements in the middle anyway, we can set the entire array `ynext` to `ynow` at the beginning. This also has the advantage of initializing the size of `ynext`. We could instead have done

```
ynext=zeros(size(ynow));
ynext(1)=ynow(1);
ynext(length(ynow))=ynow(length(ynow));
```

  which is only slightly more cumbersome than assigning the hole array.

- The loop of course starts from 2 and goes up to one segment before the last segment. This is obviously needed because `i-1` and `i+1` would be undefined for the first and last segments respectively. Besides the first and the last segments are already fixed by the boundary conditions.

We can finally put together all of this to write the final code as

```
## Script that simulates a traveling wave on an ideal string.

dx=1e-2;    ## Spatial increment [m]
c=300;      ## Speed of sound [m/sec]
dt=dx/c;    ## Time increment [sec]
r=c*dt/dx;  ## Dimensionless constant

x=-1:dx:1;
l=length(x);
x0=0.5;
k=1e2;
## Set up the initial profile
y=initial_profile(x,x0,k);
plot(x,y)
pause

## Impose the time boundary condition
ynow=y;
yprev=y;

Nsteps=2000;

for n=1:Nsteps

  ynext=propagate(ynow,yprev,r);
  plot(x,ynext,';;')
  axis([-1.05,1.05,-1.1,1.1])
  pause(0);

  yprev=ynow;
  ynow=ynext;

endfor
```

Let's now analyze this code, concentrating on the points higlighted in red :

- As we have discussed in lecture, `r`=1 is the best choice for our simulation. However, under certain circumstances, it might be necessary to use `r`s that are smaller than 1. In preparation for these cases, we define `r` in the formal way.
- Before starting the time iterations, we need to initialize `ynow` and `yprev`. Assigning both of them to the initial Gaussian profile is equivalent to keeping the string stationary in time for all times up until the time of the simulation.
- In the time loop, it is not sufficient to calculate the profile in the next step using `propagate`. We must also move forward `ynext` so that it becomes `ynow` in the next step. Similarly what we call `ynow` for the present step should become `yprev` for the next step. This circular shift has to be done ith a lot of care because if the order is wrong, you might overwrite data that you need.