1 Problem Set I solving wave equation

$$\frac{\partial^2 \phi}{\partial^2 t} = c^2 \frac{\partial^2 \phi}{\partial^2 x} \tag{1}$$

1.1 fully first order formulation

$$\eta = \phi_{,t}, \quad \chi = \phi_{,x} \tag{2}$$

$$\eta(t,x)\chi(t,x)\vec{u}(\phi,\eta,\chi)$$

$$\vec{u}_{,t} + \mathbf{A}\vec{u}_{,x} = \vec{S} \tag{3}$$

1.2 initial condition

$$\phi(0,x) = e^{\sin^2\left(\frac{\pi x}{L}\right)} - 1, \quad 0 \leq x \leq L \tag{4}$$

with periodic condition:

$$\phi(t,x) = \phi(t, x \pm L) \tag{5}$$

2 Program

```
#include <cstdio>
#include <cmath>
#include <fstream>
#include <iostream>
using namespace std ;

void output(int ti, int xi, double t[], double x[], double phi[]);
void init(double t[], double x[], double phi[], double eta[], double chi[], int xSteps,  int dx, double L);
void boundaryCondition(double phi[], double eta[], double chi[], int xSteps);
double secondOrderSpatial(double funct[], int xi, double dx);
void forwardEulerMethod(double funct[], double funct2[], int ti, double dt, int xi, double dx, double factor);
void solvingWaveEquation(double phi[], double eta[], double chi[], double t[], double dt, double x[], double dx, const double CSpeed, const int xSteps);

void output(int ti, int xi, double t[], double x[], double phi[]){
    // t x phi
    cout << t[ti] << ' ' << x[xi] << ' ' <<  phi[xi] << endl;
};

void init(double t[], double x[], double phi[], double eta[], double chi[], int xSteps, double dx, double L){
    t[0]=0;
    //x[0]=0;
    for (int i = 0; i < xSteps; i=i+1) {
        phi[i] = exp(pow(sin(M_PI/L*(i*dx)),2))-1;
        eta[i] = phi[i];
        chi[i] = phi[i];
        x[i]=i*dx;
        output(0,i,t,x,phi);
        }
        x[xSteps]=xSteps*dx;
    boundaryCondition(phi, eta, chi, xSteps);
```

```
    output(0,xSteps,t,x,phi);
};

void boundaryCondition(double phi[], double eta[], double chi[], int xSteps){
    phi[xSteps] = phi[0];
    eta[xSteps] = eta[0];
    chi[xSteps] = chi[0];
};

double secondOrderSpatial(double funct[], int xi, double dx){
    return (funct[xi+1]-funct[xi-1])/(2*dx);
};

void forwardEulerMethod(double funct[], double funct2[], int ti, double dt, int xi,
↪  double dx, double factor){
    funct[ti+1]=funct[ti]+factor*dt*secondOrderSpatial(funct2, xi, dx);
};

void solvingWaveEquation(double phi[], double eta[], double chi[], double t[], double
↪  dt, double x[], double dx, const double CSpeed, const int xSteps){
    //for (int j = 0; j < tSteps; j=j+1) {
    for (int j = 0; j < 10; j=j+1) {
        for (int i = 0; i < xSteps; i=i+1) {
            forwardEulerMethod(phi, eta, j, dx, i, dt, 1);
            forwardEulerMethod(eta, chi, j, dx, i, dt, pow(CSpeed, 2));
            forwardEulerMethod(chi, eta, j, dx, i, dt, 1);
            output(j, i, t, x, phi);
        };
        t[j]=j*dt;
        boundaryCondition(phi, eta, chi, xSteps);
        output(j, xSteps, t, x, phi);
    };
};

int main(int argc, char** argv)
{
    const double CSpeed = 1;
    const double CMax = 1;
    const double dx = stod(argv[1]); //
    const double L = 1; // gridSpace
    const double timeLength = 1;
    const double dt = CMax*dx/CSpeed;
    const int xSteps = int( L / dx );
    const int tSteps = int (timeLength / dt );

    double //
    x[xSteps],
    t[tSteps],
    phi[xSteps],
    eta[xSteps],
    chi[xSteps]
    ;
    init(t, x, phi, eta, chi, xSteps, dx, L);

    // cases for solver
    solvingWaveEquation(phi, eta, chi, t, dt, x, dx, CSpeed, xSteps);
    //{{{solving wave equation}}
    //{{{second order spatial derivative}}
    //{{{forwad Euler method}}
    //{{{forth order spatial derivative}}
```

```
89      //{{Runge Kutter solver}}
90          return 0;
91  };
```