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

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

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[][2]);
void init(double t[], double x[], double phi[][2], double eta[][2], double chi[][2],
 ↪  int xSteps, double dx, double L);
void boundaryCondition(int xSteps, double phi[][2], double eta[][2], double
 ↪  chi[][2]);
double secondOrderSpatial(double funct2[][2], int xi, double dx);
void forwardEulerMethod(double funct[][2], double funct2[][2], int ti, double dt, int
 ↪  xi, double dx, double factor);
void solvingWaveEquation(int xSteps, double phi[][2], double eta[][2], double
 ↪  chi[][2], double t[], double dt, double x[], double dx, const double CSpeed);

void output(int ti, int xi, double t[], double x[], double phi[][2]){
    // t x phi
    cout << t[ti] << ' ' << x[xi] << ' ' <<  phi[xi][ti] << endl;
};

void init(double t[], double x[], double phi[][2], double eta[][2], double chi[][2],
 ↪  int xSteps, double dx, double L){
    t[0]=0;
    //x[0]=0;
    for (int i = 0; i < xSteps-4; i=i+1) {
        phi[i+2][0] = exp(pow(sin(M_PI/L*((i)*dx)),2))-1;
        chi[i+2][0] = exp(pow(sin(M_PI/L*(i*dx)),2))*cos(M_PI/L*(i*dx))*M_PI/L;
        eta[i+2][0] = chi[i+2][0];
        x[i]=i*dx;
        output(0, i, t, x, phi);
        }
        x[xSteps]=xSteps*dx;
```

```
    boundaryCondition(xSteps, phi, eta, chi);
    output( 0, xSteps-4, t, x, phi);
};

void boundaryCondition(int xSteps, double phi[][2], double eta[][2], double
↪  chi[][2]){
    phi[xSteps][0] = phi[0][0];
    eta[xSteps][0] = eta[0][0];
    chi[xSteps][0] = chi[0][0];
    phi[xSteps-1][0] = phi[1][0];
    eta[xSteps-1][0] = eta[1][0];
    chi[xSteps-1][0] = chi[1][0];
    phi[xSteps-2][0] = phi[2][0];
    eta[xSteps-2][0] = eta[2][0];
    chi[xSteps-2][0] = chi[2][0];
};

double secondOrderSpatial(int xSteps, double funct2[][2], int xi, double dx){
    return (funct2[xi+1][0]-funct2[xi-1][0])/(2*dx);
};

void forwardEulerMethod(double funct[][2], double funct2[][2], int ti, double dt, int
↪  xi, double dx, double factor){
    //funct[xi][1]=funct[xi][0]+factor*dt*secondOrderSpatial(funct2, xi, dx);
    funct[xi][1]=funct[xi][0]+factor*dt*(funct2[xi+1][0]-funct2[xi-1][0])/(2*dx);

};

void solvingWaveEquation(double phi[][2], double eta[][2], double chi[][2], double
↪  t[], double dt, double x[], double dx, const double CSpeed, int xSteps){
    //for (int j = 0; j < tSteps; j=j+1) {
    for (int j = 0; j < 10; j=j+1) {
        t[j]=j*dt;
        for (int i = 0; i < xSteps; i=i+1) {
            forwardEulerMethod(phi, eta, j, dt, i, dx, 1);
            forwardEulerMethod(eta, chi, j, dt, i, dx, pow(CSpeed, 2));
            forwardEulerMethod(chi, eta, j, dt, i, dx, 1);
            output(j, i, t, x, phi);
        };
        boundaryCondition(xSteps, phi, eta, chi);
        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 nGhosts = 4;
    const int xSteps = int( L / dx ) + nGhosts;
    const int tSteps = int (timeLength / dt );

    double //
    x[xSteps],
    t[tSteps],
    phi[xSteps][2],
```

```
    chi[xSteps][2],
    eta[xSteps][2]
    ;

    cout << "# parameters " << dx << ' ' << dt << ' ' <<  xSteps << endl;

    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}}
    //{{{Runge Kutter solver}}
        return 0;
};
```