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

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

with periodic condition:

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

2 Program

```cpp
#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 ti, 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], double dt, int xi, double dx, double factor);
void solvingWaveEquation(double phi[][2], double eta[][2], double chi[][2], double t, double dt, double x[], double dx, double CSpeed, int xSteps, int tSteps);
void updateFunc(int xSteps, double phi[][2], double eta[][2], double chi[][2]);
void gnuplot();

void output(int ti, int xi, double t, double x[], double phi[][2]){
    // t x phi
    //cout << t << ' ' << x[xi] << ' ' <<  phi[xi][ti] << endl;
    cout << 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;
    //x[0]=0;
    cout << "reset" << endl;
    cout << "set xrange [0:1]" << endl;
    cout << "set yrange [-10:10]" << endl;
//    cout << "set yrange [-2:2]" << endl;
    gnuplot();
```

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

void boundaryCondition(int ti, int xSteps, double phi[][2], double eta[][2], double chi[][2]){
    phi[0][ti] = phi[xSteps-4][ti];
    eta[0][ti] = eta[xSteps-4][ti];
    chi[0][ti] = chi[xSteps-4][ti];
    phi[1][ti] = phi[xSteps-3][ti];
    eta[1][ti] = eta[xSteps-3][ti];
    chi[1][ti] = chi[xSteps-3][ti];
    phi[xSteps-2][ti] = phi[2][ti];
    eta[xSteps-2][ti] = eta[2][ti];
    chi[xSteps-2][ti] = chi[2][ti];
    phi[xSteps-1][ti] = phi[3][ti];
    eta[xSteps-1][ti] = eta[3][ti];
    chi[xSteps-1][ti] = chi[3][ti];
    phi[xSteps][ti] = phi[4][ti];
    eta[xSteps][ti] = eta[4][ti];
    chi[xSteps][ti] = chi[4][ti];
};

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], 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, double CSpeed, int xSteps, int tSteps){
    for (int j = 1; j < tSteps; j=j+1) {
    //for (int j = 1; j < 40000; j=j+1) {
```

```
        t=j*dt;
        //cout << " " << endl;
        gnuplot();
        for (int i = 2; i < xSteps-2; i=i+1) {
            forwardEulerMethod(phi, eta, dt, i, dx, 1);
            forwardEulerMethod(eta, chi, dt, i, dx, pow(CSpeed, 2));
            forwardEulerMethod(chi, eta, dt, i, dx, 1);
            output(1, i, t, x, phi);
        };
        boundaryCondition(1, xSteps, phi, eta, chi);
        output(1, (xSteps-2), t, x, phi);
        cout << "elapsed time" << endl;
        //gnuplot();
        updateFunc(xSteps, phi, eta, chi);
        //cout << " " << endl;
    };
};

void updateFunc(int xSteps, double phi[][2], double eta[][2], double chi[][2]){
    for (int i = 0; i <= xSteps; i=i+1) {
        phi[i][0] = phi[i][1];
        chi[i][0] = chi[i][1];
        eta[i][0] = eta[i][1];
        }
};

void gnuplot(){
    //cout << "set term qt" << endl;
    cout << "plot '-' w l" << endl;
};

int main(int argc, char** argv)
{
    const double CSpeed = 1;
    const double CMax = 0.005;
    const double dx = stod(argv[1]); //
    const double L = 1; // gridSpace
    const double timeLength = 1;
    const double dt = CMax*dx/abs(CSpeed);
    const int nGhosts = 4;
    const int xSteps = int( L / dx ) + nGhosts;
    //const int tSteps = int (timeLength / dt );
    const int tSteps = int ( stod(argv[2]));

    double //
    x[xSteps],
    //t[tSteps],
    t=0,
    phi[xSteps][2],
    chi[xSteps][2],
    eta[xSteps][2]
    //phiGhost[nGhosts][2],
    //chiGhost[nGhosts][2],
    //etaGhost[nGhosts][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, tSteps);

    //{{solving wave equation}}
    //{{second order spatial derivative}}
    //{{forwad Euler method}}
    //{{forth order spatial derivative}}
    //{{Runge Kutter solver}}
        return 0;
};
```