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[][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, int deriv);
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 rungekuttaSolver(double y[],int n,double x,double h, void (derivs)(double,
    double[], double[]));
void waveDGL(double dt, double y[], double dydt[], double dx);

void output(int ti, int xi, double t, double x[], double phi[][2]){
    // x phi
    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){
    cout << "reset" << endl;
    cout << "set xrange [0:1]" << endl;
    cout << "set yrange [-10:10]" << 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] = 1;
        //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/
        x[i]=(i-2)*dx;
        output(0, i, t, x, phi);
        }
        x[xSteps-2]=(xSteps-4)*dx;
    boundaryCondition(0, xSteps, phi, eta, chi);
    output(0, (xSteps-2), t, x, phi);

};

void boundaryCondition(int ti, int xSteps, double phi[][2], double eta[][2], double
↪  chi[][2]){
    phi[0][ti] = phi[xSteps-5][ti];
    eta[0][ti] = eta[xSteps-5][ti];
    chi[0][ti] = chi[xSteps-5][ti];
    phi[1][ti] = phi[xSteps-4][ti];
    eta[1][ti] = eta[xSteps-4][ti];
    chi[1][ti] = chi[xSteps-4][ti];
    phi[xSteps-3][ti] = phi[2][ti];
    eta[xSteps-3][ti] = eta[2][ti];
    chi[xSteps-3][ti] = chi[2][ti];
    phi[xSteps-2][ti] = phi[3][ti];
    eta[xSteps-2][ti] = eta[3][ti];
    chi[xSteps-2][ti] = chi[3][ti];
    phi[xSteps-1][ti] = phi[4][ti];
    eta[xSteps-1][ti] = eta[4][ti];
    chi[xSteps-1][ti] = chi[4][ti];
};

double secondOrderSpatial(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, int deriv){
    if (deriv == 0) {
        funct[xi][1]=funct[xi][0]+factor*dt*funct2[xi][0];
    } else {
        funct[xi][1]=funct[xi][0]+factor*dt*secondOrderSpatial(funct2, xi, 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) {
        t=j*dt;
        gnuplot();
        for (int i = 2; i < xSteps-2; i=i+1) {
```

```
            forwardEulerMethod(phi, eta, dt, i, dx, 1, 0);
            forwardEulerMethod(eta, chi, dt, i, dx, pow(CSpeed, 2), 1);
            forwardEulerMethod(chi, eta, dt, i, dx, 1, 1);
            output(1, i, t, x, phi);
        };
        boundaryCondition(1, xSteps, phi, eta, chi);
        output(1, (xSteps-2), t, x, phi);
        cout << "elapsed time" << endl;
        updateFunc(xSteps, phi, eta, chi);
    };
};

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 << "plot '-' w l" << endl;
};

void rungekuttaSolver(double y[],int n, double t, double dt, double dx, void
↪  (derivs)(double, double[], double[])){
    int j,i;
    double dt2,dt6,tdt, tdtdt,y1[n],k1[n],k2[n],k3[n];
    dt2 = dt * 0.5;
    dt6 = dt / 6.;
    //Simpsonregel
    tdt2=t+dt*0.5;
    tdt=t+dt;
    derivs(t, y, k1);
    for(i=0;i<n;i++){
        y1[i]=y[i]+dt*0.5*k1[i];
    }
    derivs(tdt2, y1, k2);
    for(i=0;i<n;i++){
        y1[i]=y[i]-dt*k1[i]+2*dt*k2[i];
    }
    derivs(tdt, y1, k3);
    x+=h;
    for(i=0;i<n;i++){
        y[i]+=dt6*(k1[i]+4.*k2[i]+k3[i]);
    }
}

void waveDGL(double t, double y[], double dydt[], double dx){
    //dydt[0] =y[1];
    //dydt[1] =pow(CSpeed, 2)*(-y[2][i+2]+8*y[2][i+1]-8*y[2][i-1]+y[2][i-2])/(12*dx)
    ↪  ;
    //dydt[2] =(-y[1][i+2]+8*y[1][i+1]-8*y[1][i-1]+y[1][i-2])/(12*dx) ;
}

int main(int argc, char** argv)
{
    const double CSpeed = 1;
    const double CMax = 0.005;
    const double dx = stod(argv[1]); //
```

```
142     const double L = 1; // gridSpace
143     const double timeLength = 1;
144     const double dt = CMax*dx/abs(CSpeed);
145     const int nGhosts = 4;
146     const int xSteps = int( L / dx ) + nGhosts;
147     //const int tSteps = int (timeLength / dt );
148     const int tSteps = int ( stod(argv[2]));
149 
150     double //
151     x[xSteps],
152     t=0,
153     phi[xSteps][2],
154     chi[xSteps][2],
155     eta[xSteps][2]
156     ;
157 
158     cout << "# parameters dx=" << dx << " dt=" << dt << " xSteps=" <<  xSteps <<
        ↪  endl;
159 
160     init(t, x, phi, eta, chi, xSteps, dx, L);
161 
162     // cases for solver
163     //{{{solving wave equation}}
164     solvingWaveEquation(phi, eta, chi, t, dt, x, dx, CSpeed, xSteps, tSteps);
165 
166     //{{{forth order spatial derivative}}
167     //{{{Runge Kutter solver}}
168         return 0;
169 };
```