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

```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, 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) {
```

```
29        phi[i][0] = exp(pow(sin(M_PI/L*((i-2)*dx)),2))-1;
30        //chi[i][0] = phi[i][0];
31        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/
32        //chi[i][0] = (i-2)*dx;
33        //chi[i][0] = sin(M_PI/L*((i-2)*dx));
34        //chi[i][0] = 1;
35        eta[i][0] = -chi[i][0];
36        //eta[i][0] = 0;
37        //eta[i][0] = 1;
38        //eta[i][0] = pow(sin(M_PI/L*((i-2)*dx)),2);
39        //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/
40        x[i]=(i-2)*dx;
41        output(0, i, t, x, phi);
42        }
43        x[xSteps-2]=(xSteps-4)*dx;
44    boundaryCondition(0, xSteps, phi, eta, chi);
45    output(0, (xSteps-2), t, x, phi);
46
47 };
48
49 void boundaryCondition(int ti, int xSteps, double phi[][2], double eta[][2], double
   ↪  chi[][2]){
50    phi[0][ti] = phi[xSteps-5][ti];
51    eta[0][ti] = eta[xSteps-5][ti];
52    chi[0][ti] = chi[xSteps-5][ti];
53    phi[1][ti] = phi[xSteps-4][ti];
54    eta[1][ti] = eta[xSteps-4][ti];
55    chi[1][ti] = chi[xSteps-4][ti];
56    phi[xSteps-3][ti] = phi[2][ti];
57    eta[xSteps-3][ti] = eta[2][ti];
58    chi[xSteps-3][ti] = chi[2][ti];
59    phi[xSteps-2][ti] = phi[3][ti];
60    eta[xSteps-2][ti] = eta[3][ti];
61    chi[xSteps-2][ti] = chi[3][ti];
62    phi[xSteps-1][ti] = phi[4][ti];
63    eta[xSteps-1][ti] = eta[4][ti];
64    chi[xSteps-1][ti] = chi[4][ti];
65 };
66
67 double secondOrderSpatial(double funct2[][2], int xi, double dx){
68    return (funct2[xi+1][0]-funct2[xi-1][0])/(2*dx);
69 };
70
71 void forwardEulerMethod(double funct[][2], double funct2[][2], double dt, int xi,
   ↪  double dx, double factor, int deriv){
72    if (deriv == 0) {
73        funct[xi][1]=funct[xi][0]+factor*dt*funct2[xi][0];
74    } else {
75        funct[xi][1]=funct[xi][0]+factor*dt*secondOrderSpatial(funct2, xi, dx);
76    }
77 };
78
79 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){
80    for (int j = 1; j <= tSteps; j=j+1) {
81        t=j*dt;
82        gnuplot();
83        for (int i = 2; i < xSteps-2; i=i+1) {
```

```cpp
                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 x,double h, void (derivs)(double,
      double[], double[])){
        int j,i;
        double h2,h6,xh, xhh,y1[n],k1[n],k2[n],k3[n];
        h2=h*0.5;    h6=h/6.;
        //Simpsonregel
        xh=x+h2;          xhh=x+h;
        (derivs)(x,y,k1);
        for(i=0;i<n;i++) y1[i]=y[i]+h2*k1[i];
        (derivs)(xh,y1,k2);
        for(i=0;i<n;i++) y1[i]=y[i]-h*k1[i]+2*h*k2[i];
        (derivs)(xhh,y1,k3);
        x+=h;
        for(i=0;i<n;i++) y[i]+=h6*(k1[i]+4.*k2[i]+k3[i]);
    }

    void waveDGL(double dt, 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]); //
        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]));

```

```cpp
142        double //
143        x[xSteps],
144        t=0,
145        phi[xSteps][2],
146        chi[xSteps][2],
147        eta[xSteps][2]
148        ;
149
150        cout << "# parameters dx=" << dx << " dt=" << dt << " xSteps=" <<  xSteps <<
     ↪  endl;
151
152        init(t, x, phi, eta, chi, xSteps, dx, L);
153
154        // cases for solver
155        //{{solving wave equation}}
156        solvingWaveEquation(phi, eta, chi, t, dt, x, dx, CSpeed, xSteps, tSteps);
157
158        //{{forth order spatial derivative}}
159        //{{Runge Kutter solver}}
160            return 0;
161    };
```