

1 Problem Set I solving wave equation

$$\frac{\partial^2 \phi}{\partial t^2} = c^2 \frac{\partial^2 \phi}{\partial x^2} \quad (1)$$

1.1 fully first order formulation

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

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

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

1.2 initial condition

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

with periodic condition:

$$\phi(t, x) = \phi(t, x \pm L) \quad (5)$$

2 Program

```
1  #include <cstdio>
2  #include <cmath>
3  #include <fstream>
4  #include <iostream>
5  using namespace std ;
6
7  void output(int ti, int xi, double t, double x[], double phi[][2]);
8  void init(double t, double x[], double phi[][2], double eta[][2], double chi[][2],
9  → int xSteps, double dx, double L);
10 void boundaryCondition(int ti, int xSteps, double phi[][2], double eta[][2], double
11 → chi[][2]);
12 double secondOrderSpatial(double funct2[][2], int xi, double dx);
13 void forwardEulerMethod(double funct[][2], double funct2[][2], double dt, int xi,
14 → double dx, double factor, int deriv);
15 void solvingWaveEquation(double phi[][2], double eta[][2], double chi[][2], double t,
16 → double dt, double x[], double dx, double CSpeed, int xSteps, int tSteps);
17 void updateFunc(int xSteps, double phi[][2], double eta[][2], double chi[][2]);
18 void gnuplot();
19 void rungekuttaSolver(double y[], int n, double x, double h, void (derivs)(double,
20 → double[], double[]));
21 void waveDGL(double dt, double y[], double dydt[], double dx);
22
23 void output(int ti, int xi, double t, double x[], double phi[][2]){
24     // x phi
25     cout << x[xi] << ' ' << phi[xi][ti] << endl;
26 }
27
28 void init(double t, double x[], double phi[][2], double eta[][2], double chi[][2],
29 → int xSteps, double dx, double L){
30     cout << "reset" << endl;
31     cout << "set xrange [0:1]" << endl;
32     cout << "set yrange [-10:10]" << endl;
33     gnuplot();
34     for (int i = 0; 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] =
32         ↪ 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/
33         //chi[i][0] = (i-2)*dx;
34         //chi[i][0] = sin(M_PI/L*((i-2)*dx));
35         //chi[i][0] = 1;
36     eta[i][0] = -chi[i][0];
37     //eta[i][0] = 0;
38     //eta[i][0] = 1;
39     //eta[i][0] = pow(sin(M_PI/L*((i-2)*dx)),2);
40     ↪ 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/
41     x[i]=(i-2)*dx;
42     output(0, i, t, x, phi);
43     }
44     x[xSteps-2]=(xSteps-4)*dx;
45     boundaryCondition(0, xSteps, phi, eta, chi);
46     output(0, (xSteps-2), t, x, phi);
47 };
48
49 void boundaryCondition(int ti, int xSteps, double phi[][2], double eta[][2], double
50     ↪ chi[][2]){
51     phi[0][ti] = phi[xSteps-5][ti];
52     eta[0][ti] = eta[xSteps-5][ti];
53     chi[0][ti] = chi[xSteps-5][ti];
54     phi[1][ti] = phi[xSteps-4][ti];
55     eta[1][ti] = eta[xSteps-4][ti];
56     chi[1][ti] = chi[xSteps-4][ti];
57     phi[xSteps-3][ti] = phi[2][ti];
58     eta[xSteps-3][ti] = eta[2][ti];
59     chi[xSteps-3][ti] = chi[2][ti];
60     phi[xSteps-2][ti] = phi[3][ti];
61     eta[xSteps-2][ti] = eta[3][ti];
62     chi[xSteps-2][ti] = chi[3][ti];
63     phi[xSteps-1][ti] = phi[4][ti];
64     eta[xSteps-1][ti] = eta[4][ti];
65     chi[xSteps-1][ti] = chi[4][ti];
66 };
67
68 double secondOrderSpatial(double funct2[][2], int xi, double dx){
69     ↪ return (funct2[xi+1][0]-funct2[xi-1][0])/(2*dx);
70 };
71
72 void forwardEulerMethod(double funct[][2], double funct2[][2], double dt, int xi,
73     ↪ double dx, double factor, int deriv){
74     if (deriv == 0) {
75         funct[xi][1]=funct[xi][0]+factor*dt*funct2[xi][0];
76     } else {
77         funct[xi][1]=funct[xi][0]+factor*dt*secondOrderSpatial(funct2, xi, dx);
78     }
79 };
80
81 void solvingWaveEquation(double phi[][2], double eta[][2], double chi[][2], double t,
82     ↪ double dt, double x[], double dx, double CSpeed, int xSteps, int tSteps){
83     for (int j = 1; j <= tSteps; j=j+1) {
84         t=j*dt;
85         gnuplot();
86         for (int i = 2; i < xSteps-2; i=i+1) {

```

```

84         forwardEulerMethod(phi, eta, dt, i, dx, 1, 0);
85         forwardEulerMethod(eta, chi, dt, i, dx, pow(CSpeed, 2), 1);
86         forwardEulerMethod(chi, eta, dt, i, dx, 1, 1);
87         output(1, i, t, x, phi);
88     };
89     boundaryCondition(1, xSteps, phi, eta, chi);
90     output(1, (xSteps-2), t, x, phi);
91     cout << "elapsed time" << endl;
92     updateFunc(xSteps, phi, eta, chi);
93 };
94 };
95
96 void updateFunc(int xSteps, double phi[][2], double eta[][2], double chi[][2]){
97     for (int i = 0; i <= xSteps; i=i+1) {
98         phi[i][0] = phi[i][1];
99         chi[i][0] = chi[i][1];
100        eta[i][0] = eta[i][1];
101    }
102 };
103
104 void gnuplot(){
105     cout << "plot '-' w l" << endl;
106 };
107
108
109
110 void waveDGL(double t, double y[], double dydt[], double dx){
111     //dydt[0] =y[1];
112     //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)
113     ↪ ;
114     //dydt[2] =(-y[1][i+2]+8*y[1][i+1]-8*y[1][i-1]+y[1][i-2])/(12*dx) ;
115 }
116
117 int main(int argc, char** argv)
118 {
119     const double CSpeed = 1;
120     const double CMax = 0.005;
121     const double dx = stod(argv[1]); //
122     const double L = 1; // gridSpace
123     const double timeLength = 1;
124     const double dt = CMax*dx/abs(CSpeed);
125     const int nGhosts = 4;
126     const int xSteps = int( L / dx ) + nGhosts;
127     //const int tSteps = int (timeLength / dt );
128     const int tSteps = int ( stod(argv[2]));
129
130     double //
131     x[xSteps],
132     t=0,
133     phi[xSteps][2],
134     chi[xSteps][2],
135     eta[xSteps][2]
136     ;
137
138     cout << "# parameters dx=" << dx << " dt=" << dt << " xSteps=" << xSteps <<
139     ↪ endl;
140
141     init(t, x, phi, eta, chi, xSteps, dx, L);
142
143     // cases for solver

```

```
142  //{{solving wave equation}}  
143  solvingWaveEquation(phi, eta, chi, t, dt, x, dx, CSpeed, xSteps, tSteps);  
144  
145  //{{forth order spatial derivative}}  
146  //{{Runge Kutter solver}}  
147      return 0;  
148  };
```