

Solution to the 1D Heat Equation

Advertisement No. CDS/KA/SERB-SRG/DEC2020/RA

A report submitted for the application of
Project Assistant Position

Sundaresan G
Scientist/Engineer
ISRO Propulsion Complex

B.Tech Mechanical Engineering
Batch of 2016, NITT

Contents

a. Analytical Solution	1
b. Numerical Scheme	3
c. Numerical simulations	4
d. Solution Plot for 128 grid points	5
e. Average Error in the Domain at each time step	6
A Appendix A: C code	7

a. Analytical Solution

Given

$$u_t = u_{xx} , \quad x \in [0, 1] \quad (1)$$

subject to periodic boundary conditions

$$u(0, t) = u(1, t) \quad \text{and} \quad u_x(0, t) = u_x(1, t)$$

and initial condition

$$u(x, 0) = \sin(2\pi x). \quad (2)$$

The given problem is a well posed homogeneous linear PDE and has a unique solution for the given boundary and initial conditions. The solution is derived as follows by separation of variables:

Let

$$u(x, t) = X(x) T(t). \quad (3)$$

Substituting into (1), we have

$$XT_t = X_{xx}T \implies \frac{X_{xx}}{X} = \frac{T_t}{T} = \alpha,$$

where α is a constant since the LHS is only a function of x and RHS is a function of t . We now examine three cases for α :

Case 1: $\alpha = 0$

$$\begin{aligned} X(x) &= ax + b, \quad T(t) = c, \\ \implies u^{\alpha=0} &= a_1x + a_2. \end{aligned} \tag{4}$$

Case 2: $\alpha = \beta^2 > 0$

$$\begin{aligned} X(x) &= ae^{\beta x} + be^{-\beta x}, \quad T(t) = ce^{\beta^2 t^2}, \\ \implies u^{\alpha>0} &= (b_1e^{\beta x} + b_2e^{-\beta x})e^{\beta^2 t^2}, \end{aligned} \tag{5}$$

where b_1 and b_2 are arbitrary constants depending on β .

Case 3: $\alpha = -\lambda^2 < 0$

$$\begin{aligned} X(x) &= a \cos(\lambda x) + b \sin(\lambda x), \quad T(t) = ce^{-\lambda^2 t^2}, \\ \implies u^{\alpha<0} &= (c_1 \cos(\lambda x) + c_2 \sin(\lambda x))e^{-\lambda^2 t^2}. \end{aligned} \tag{6}$$

where c_1 and c_2 are arbitrary constants depending on λ .

Due to the homogenous and linear nature of the given PDE, a finite sum of (4), (5) and (6) satisfies the PDE. It can be also proved that an infinite convergent sum consisting of (4), (5) and (6) satisfies the PDE. However we would not delve into this topic. For the given problem with the given initial condition, it is enough to consider case 3. Hence,

$$u(x, t) = (c_1 \cos(\lambda x) + c_2 \sin(\lambda x)) e^{-\lambda^2 t^2}. \tag{7}$$

Applying boundary condition $u(0, t) = u(1, t)$, we get

$$c_1(\cos(\lambda) - 1) + c_2 \sin(\lambda) = 0 \tag{8}$$

Applying boundary condition $u_x(0, t) = u_x(1, t)$ and the fact that $\lambda \neq 0$, we get

$$c_1 \sin(\lambda) + c_2(1 - \cos(\lambda)) = 0 \tag{9}$$

Solving equations (8) and (9), we get the following two equations

$$c_1 = c_2 = 0.$$

The above equation leads to a trivial solution and does not satisfy the given initial condition. Hence the following must hold

$$\sin(\lambda) = (1 - \cos(\lambda)) = 0,$$

which is satisfied if and only if

$$\lambda = n\pi, \quad \text{where } n \in \mathbb{Z} \setminus \{0\}$$

For the given initial condition (2), we get

$$c_1 = 0, \quad \lambda = 2\pi \quad \text{and} \quad c_2 = 1.$$

The analytical solution is therefore given by

$$u(x, t) = \sin(2\pi x) e^{-4\pi^2 t^2} \tag{10}$$

The above solution satisfies the PDE, initial and boundary conditions. Moreover due to well posed nature of the problem, the solution obtained is a unique one (by maximum energy principle).

b. Numerical Scheme

Discretizing (1) by the explicit Euler (time derivative) and second order central difference (space derivative) scheme, we have

$$u_i^{t+\Delta t} = u_i^t + \frac{\Delta t}{(\Delta x)^2} (u_{i+1}^t - 2u_i^t + u_{i-1}^t), \quad \text{where } i = 1, \dots, n-1.$$

For $i = 0, n$, the discretized equations, with periodic boundary conditions, yield

$$u_0^{t+\Delta t} = u_0^t + \frac{\Delta t}{(\Delta x)^2} (u_1^t - 2u_0^t + u_{n-1}^t) \quad \text{and} \quad u_n^{t+\Delta t} = u_n^t + \frac{\Delta t}{(\Delta x)^2} (u_n^t - 2u_n^t + u_0^t)$$

The convergence and stability criteria,

$$\frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

requires $n \leq 223$ for $\Delta t = 0.00001$.

c. Numerical simulations

A code is written in C language and given in Appendix A. It uses command line arguments for input of Number of grid points (n) and Time step at which solution is required. Please compile it using a suitable compiler (GCC) as follows

```
gcc code.c -o "output file name"  
"output file name".exe "n" "Time step(s)"
```

For example,

```
gcc code.c -o results  
results.exe 200 300 500 1000
```

The above example inputs 200 as the number of grid points and prints the numerical and analytical solutions at 300, 500 and 1000 time steps.

Note: The maximum number of grid points (default:128) the code accepts is 220 (for convergence) and the maximum time step (default: 0, 100, 500 and 1000) is 3000 and number of time step arguments (default:4) is 10.

d. Solution Plot for 128 grid points

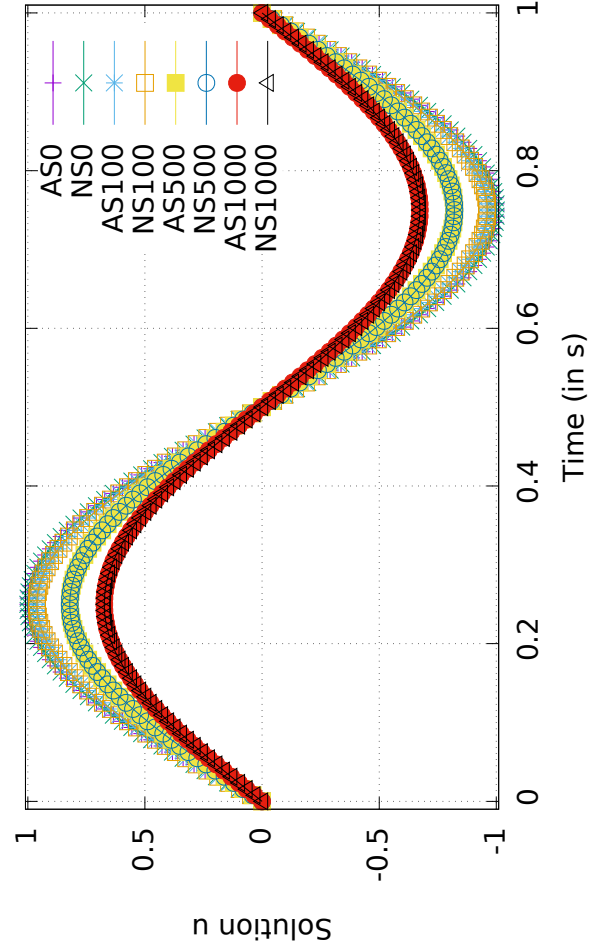


Figure 1: Analytical solution Vs Numerical solution at time steps 0, 100, 500 and 1000

- e. **Average Error in the Domain at each time step**

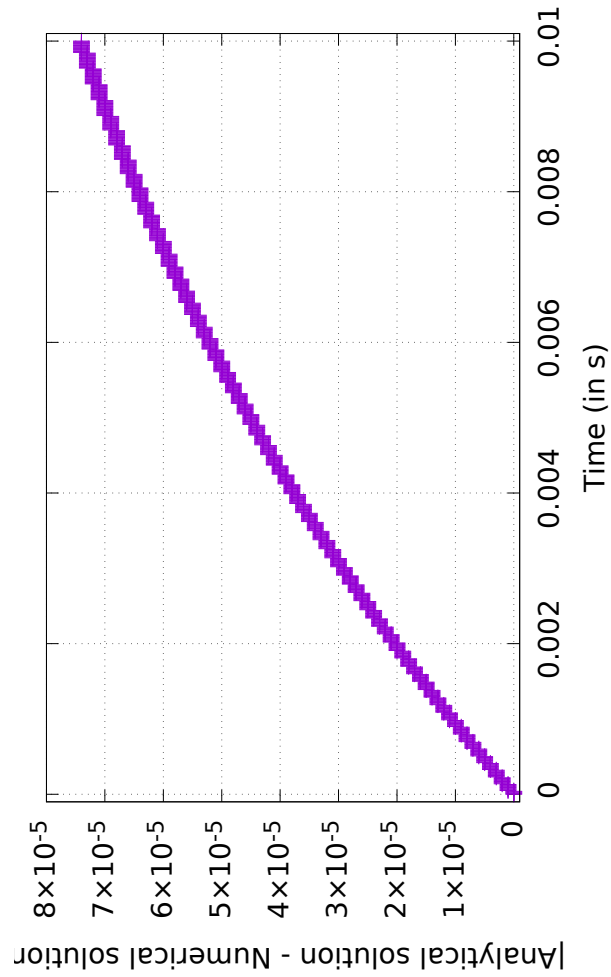


Figure 2: Average Error(AS-NS) vs Time for 128 grid points

A Appendix A: C code

```

1  #include<stdio.h>
2  #include<math.h>
3  #include<stdlib.h>
4  const double pi=acos(-1.0);
5  int max(int *h, int rows)
6  {
7      int s=0;
8      for(int i=0;i<rows;i++)
9      {
10         if(s<h[i])
11             s=h[i];
12     }
13     return s;
14 }
15 void analytical_method(double *as, double delta_t, int grid_points, int time_step,
16 double k, double interval_length)
17 {
18     for(int i=0;i<grid_points;i++)
19         as[i]=sin(2*pi*i*(interval_length/(grid_points-1)))*exp(pi*pi*(-4)*k*(time_step*delta_t));
20 }
21 void print_to_file(double *array,int x_points, double delta_x, char *s)
22 {
23     FILE *fptr=fopen(s,"w");
24     fprintf(fptr,"# %s\n", s);
25     for(int i=0;i<x_points;i++)
26         fprintf(fptr,"%lf \t %lf\n", i*delta_x, array[i]);
27     fclose(fptr);
28 }
29 void error_method(double *error_array, double *exact_solution, double *num_solution,
30 int grid_points)
31 {
32     for(int i=0;i<grid_points;i++)
33         error_array[i]=fabs(exact_solution[i]-num_solution[i]);
34 }
35 double error_sum_method(double *error_array, int grid_points)
36 {
37     double error_sum=0;
38     for(int i=0;i<grid_points;i++)
39     {
40         error_sum+=error_array[i];
41     }
42     return error_sum;
43 }
44 void initialization(double *array, int grid_points, double interval_length)
45 {
46     for(int i=0; i<grid_points;i++)
47         array[i]=sin(2*pi*i*(interval_length/(grid_points-1)));
48 }
49 void numerical_solution(double *u, double *uxx, int grid_points, double delta_t,
50 double interval_length, double k)
51 {
52     //array[i+1][columns-1]=array[i+1][0];
53     for(int j=0;j<grid_points;j++)
54     {
55         u[j]+=k*delta_t*uxx[j];
56     }
57 }
58 void uxx_calculate(double *uxx, double *u, int grid_points, double interval_length)
59 {
60     uxx[0]=
61     (u[1]-2*u[0]+u[grid_points-2])*(grid_points-1)*(grid_points-1)/(interval_length*interval_length);
62     uxx[grid_points-1]=uxx[0];
63     for(int i=1; i<grid_points-1;i++)
64     {
65         uxx[i]=
66         (u[i+1]-2*u[i]+u[i-1])*(grid_points-1)*(grid_points-1)/(interval_length*interval_length);
67     }
68 }
69 int main(int argc, char *argv[]) //argv[1] = number of grid points, argv[i] =
70 numerical and analytical solution printing time step
71 {

```



```

64     int grid_points=128;
65     int tc=1000;        //time counter parameter. default of 1000
66     double delta_t=0.00001;
67     double interval_length=1;
68     double k=1;
69     int print_time_step[20];
70
71     print_time_step[0]=0;
72     print_time_step[1]=100;
73     print_time_step[2]=500;
74     print_time_step[3]=1000;
75     int pts_length=4;           //print_time_step array length. Default is 3+1
76
77     if(argc>1)                 //for intializing printing time steps and grid
points based on command line inputs
78     {
79         grid_points=atoi(argv[1]);
80         if(argc>2)
81         {
82             for(int i=2; i<argc;i++)
83             {
84                 print_time_step[i-2]=atoi(argv[i]);
85                 if(print_time_step[i-2]==0 || print_time_step[i-2]>3000)
86                 {
87                     printf("\n Invalid time steps for printing. Please specify an
integer between 1 and 3000 for every printing time step");
88                     return -1;
89                 }
90             }
91             tc=max(print_time_step, argc-2);
92             pts_length=argc-2;
93         }
94     }
95
96
97
98     if(grid_points==0 || grid_points>220)        //the explicit scheme for the given
problem with delta_t=0.00001 converges only if the grid points are less than 223
points
99     {
100         printf("\n Invalid number of grid points. Please specify number of grid
points below 1000");
101         return -1;
102     }
103
104     grid_points+=1;    //to get even number of points since first and last point are
one and same
105
106     //double **analytical_solution= (double **)malloc(time_steps * sizeof(double *));
107     //double **num_solution= (double **)malloc(time_steps*sizeof(double *));
108     double **error_array= (double **)malloc(time_steps*sizeof(double *));
109     for(int i=0;i<time_steps;i++)
110     {
111         analytical_solution[i]=(double *)malloc(time_steps*sizeof(double));
112         num_solution[i]=(double *)malloc(time_steps*sizeof(double));
113         error_array[i]=(double *)malloc(time_steps*sizeof(double));
114     }
115     /*
116     double *u= (double *)malloc (grid_points*sizeof(double)); //solution
117     double *uxx= (double *)malloc (grid_points*sizeof(double)); //second derivative
with respect to space
118     double *as= (double *)malloc (grid_points*sizeof(double)); //analytical solution
119     double *error=(double *)malloc(grid_points*sizeof(double)); //for each grid
points
120     double *error_sum= (double *)malloc((tc+1)*sizeof(double)); //for sum of
errors of each grid points at various time steps
121
122     //printf("\n%p",&analytical_solution[0][0]);
123     //initialization(analytical_solution[0], grid_points, interval_length);
124     //analytical_method(analytical_solution, delta_t, time_steps, grid_points, k);
125     //printf("\n%f", analytical_solution[100][32]);
126
127     initialization(u, grid_points, interval_length);

```

```

128     analytical_method(as, delta_t, grid_points, 0, k, interval_length); //to
129     initialize analytical solution at 0
130     //printf("\n%f",u[33]);
131     error_sum[0]=0;
132     for(int j=0;j<pts_length;j++)
133         if(print_time_step[j]==0) //to dump the analytical and numerical solution
134             at 0th time step.
135             {
136                 char Numsol[1000], Asol[1000];
137                 sprintf(Numsol,"Numerical solution at %dth time step for %d grid
138                 points.txt", 0, grid_points-1);
139                 print_to_file(u, grid_points,
140                 (double)interval_length/(grid_points-1),Numsol);
141                 sprintf(Asol,"Analytical solution at %dth time step for %d grid
142                 points.txt", 0, grid_points-1);
143                 print_to_file(as, grid_points,
144                 (double)interval_length/(grid_points-1),Asol);
145             }
146     for(int i=1;i<=tc;i++)
147     {
148         uxx_calculate(uxx, u, grid_points, interval_length);
149         /*if(i==1)
150             printf("\n%f",uxx[32]);*/
151         numerical_solution(u, uxx, grid_points, delta_t, interval_length, k);
152         //updates u to the next time step
153         /*if(i==100)
154             printf("\n%f",u[32]);*/
155         analytical_method(as, delta_t, grid_points, i, k, interval_length);
156         /*if(i==100)
157             printf("\n%f",as[32]);*/
158         error_method(error, as, u, grid_points);
159         /*if(i==1000)
160             printf("\n%f",error[97]);*/
161         error_sum[i]= error_sum_method(error, grid_points);
162         /*if(i==1000)
163             printf("\n%f",error_sum[grid_points-3]);*/
164         for(int j=0;j<pts_length;j++)
165             if(print_time_step[j]==i) //to dump the analytical and numerical
166             solution.
167             {
168                 char Numsol[1000], Asol[1000];
169                 sprintf(Numsol,"Numerical solution at %dth time step for %d grid
170                 points.txt", i, grid_points-1);
171                 print_to_file(u, grid_points,
172                 (double)interval_length/(grid_points-1),Numsol);
173                 sprintf(Asol,"Analytical solution at %dth time step for %d grid
174                 points.txt", i, grid_points-1);
175                 print_to_file(as, grid_points,
176                 (double)interval_length/(grid_points-1),Asol);
177             }
178     }
179     char error_string[]="Sum of errors at each grid point for various time steps.txt";
180     print_to_file(error_sum, tc+1, delta_t, error_string);
181
182     free(error_sum);
183     free(error);
184     free(as);
185     free(uxx);
186     free(u);
187 }

```