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

Given problem	$u_t = u_{xx}$
Domain	$x = [0, 1]$
Boundary condition	$u(0,t) = u(1,t),\ u_x(0,t) = u_x(1,t)$
Initial condition	$u(x,0) = sin(2\pi x)$

The given problem is a well posed homogenous linear PDE and has a unique solution for the given boundary (Neumann) and initial conditions.
The solution is derived as follows by variable separation:

Let $\quad u(x,t) = X(x) * T(t)$
$$u_t = u_{xx} \implies XT_t = X_{xx}T$$
$$\frac{X_{xx}}{X} = \frac{T_t}{T} = \alpha$$

(α is a constant since LHS is only a function of x and RHS is a function of t)

Case 1: $\alpha = 0$
$\implies \quad X = ax + b,\ T = c$
$\implies \quad u^{\alpha=0} = a_1 x + a_2$

Case 2: $\alpha = \beta^2 > 0$
$\implies \quad X = ae^{\beta x} + be^{-\beta x},\ T = ce^{\beta^2 t^2}$
$\implies \quad u^{\alpha>0} = (b_1 e^{\beta x} + b_2 e^{-\beta x})e^{\beta^2 t^2}$

where b_1 and b_2 are arbitrary constants depending on β

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

$\implies \quad X = acos(\lambda x) + bsin(\lambda x), \ T = ce^{-\lambda^2 t^2}$

$\implies \quad u^{\alpha<0} = (c_1 cos(\lambda x) + c_2 sin(\lambda x))e^{-\lambda^2 t^2}$

where c_1 and c_2 are arbitrary constants depending on λ

Due to the homogenous and linear nature of the given PDE, the solution is given by

$$
\begin{aligned}
u(x,t) \quad = \quad & a_1 x + a_2 + \sum_{\beta \in S_1}((b_1(\beta)e^{\beta x} + b_2(\beta)e^{-\beta x})e^{\beta^2 t^2}) \\
& + \sum_{\lambda \in S_2}((c_1(\lambda)cos(\lambda x) + c_2(\lambda)sin(\lambda x))e^{-\lambda^2 t^2})
\end{aligned}
$$

where S_1 and S_2 are countable subsets of non-zero Real numbers (since the summation should converge)

Now the solution is finite even for $t \to \infty$ (by Physics of the heat equation with no external energy source). Hence b_1 and b_2 should be 0 for any value of β.

$$u(x,t) = a_1 x + a_2 + \sum_{\lambda \in S_2}((c_1(\lambda)cos(\lambda x) + c_2(\lambda)sin(\lambda x))e^{-\lambda^2 t^2})$$

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

$$a_2 + \sum_{\lambda \in S_2}((c_1(\lambda))e^{-\lambda^2 t^2}) = a_1 + a_2 + \sum_{\lambda \in S_2}((c_1(\lambda)cos(\lambda) + c_2(\lambda)sin(\lambda))e^{-\lambda^2 t^2})$$

$$\implies a_1 + \sum_{\lambda \in S_2}([c_1(\lambda)(cos(\lambda) - 1) + c_2(\lambda)sin(\lambda)]e^{-\lambda^2 t^2}) = 0$$

Since the above equation is true for any value of t,

$$a_1 = 0 \qquad\qquad\qquad \text{and}$$

$$c_1(\lambda)(cos(\lambda) - 1) + c_2(\lambda)sin(\lambda) = 0 \qquad\qquad \text{for every } \lambda \in S_2 \qquad (1)$$

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

$$\sum_{\lambda \in S_2}(\lambda c_2(\lambda))e^{-\lambda^2 t^2}) = \sum_{\lambda \in S_2}((-\lambda c_1(\lambda)sin(\lambda) + \lambda c_2(\lambda)cos(\lambda))e^{-\lambda^2 t^2})$$

We know $\lambda \neq 0$ and the above equation is true for every t. Hence

$$c_1(\lambda)sin(\lambda) + c_2(\lambda)(1 - cos(\lambda)) = 0 \qquad\qquad \text{for every } \lambda \in S_2 \qquad (2)$$

2

Solving equations 1 and 2, we get the following two equations

$$c_1 = c_2 = 0 \tag{3}$$

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

$$sin(\lambda) = (1 - cos(\lambda)) = 0 \tag{4}$$

The above equation is satisfied if and only if,

$$\lambda = n\pi, \text{where n} \in Z \tag{5}$$

Substituting the results in the solution u(x,t) and combining the arbitrary constants,

$$u(x,t) = (\frac{A_0}{2} + \sum_{n=1}^{\infty}(A_n cos(n\pi x) + B_n sin(n\pi x)))e^{(-\lambda^2 t^2)} \tag{6}$$

Now for t=0,

$$u(x,0) = \frac{A_0}{2} + \sum_{n=1}^{\infty}(A_n cos(n\pi x) + B_n sin(n\pi x)) \tag{7}$$

The above equation is clearly in the form of Fourier series and hence the arbitrary constants are given by the following equations,

$$A_n = 2\int_0^1 u(x,0)cos(n\pi x)dx, \qquad \text{n=0,1,2,...} \tag{8}$$

$$B_n = 2\int_0^1 u(x,0)sin(n\pi x)dx, \qquad \text{n=1,2,...} \tag{9}$$

For the given initial condition $u(x,0) = sin(2\pi x)$, we get

$$A_n = 0 \qquad \text{n=0,1,2,...} \tag{10}$$

$$B_n = 0 \qquad \text{n} \neq 2 \tag{11}$$

$$B_2 = 1 \tag{12}$$

3

Thus the analytical solution is given by

$$u(x,t) = sin(2\pi x)e^{(-4\pi^2 t^2)} \tag{13}$$

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.

b. Numerical Scheme

$$u_t = u_{xx}$$

The following is the explicit Euler (time derivative) and second order central difference (space derivative) scheme

$$u_i^{t+1} = u_i^t + \frac{\Delta t}{(\Delta x)^2}(u_{i+1}^t - 2u_i^t + u_{i-1}^t) \tag{14}$$

The above equation can be used to obtain the value at next time step for every internal point. For the end points, the following equations are used due to periodic boundary conditions,

$$u_0^{t+1} = u_0^t + \frac{\Delta t}{(\Delta x)^2}(u_1^t - 2u_0^t + u_{endpoint-1}^t) \tag{15}$$

$$u_{endpoint}^{t+1} = u_0^{t+1} \tag{16}$$

Since $\Delta t = 0.00001$, the maximum number of grid points allowed is 223 due to convergence and stability criteria, $\Delta t \leq \frac{1}{2}(\Delta x)^2$

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 and Time step at which solution is required. Please compile it using a suitable compiler (GCC) as follows

4

gcc code.c -o "output file name"
**"output file name".exe "Number of grid points" "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

Analytical Vs Numerical solution for 128 grid points
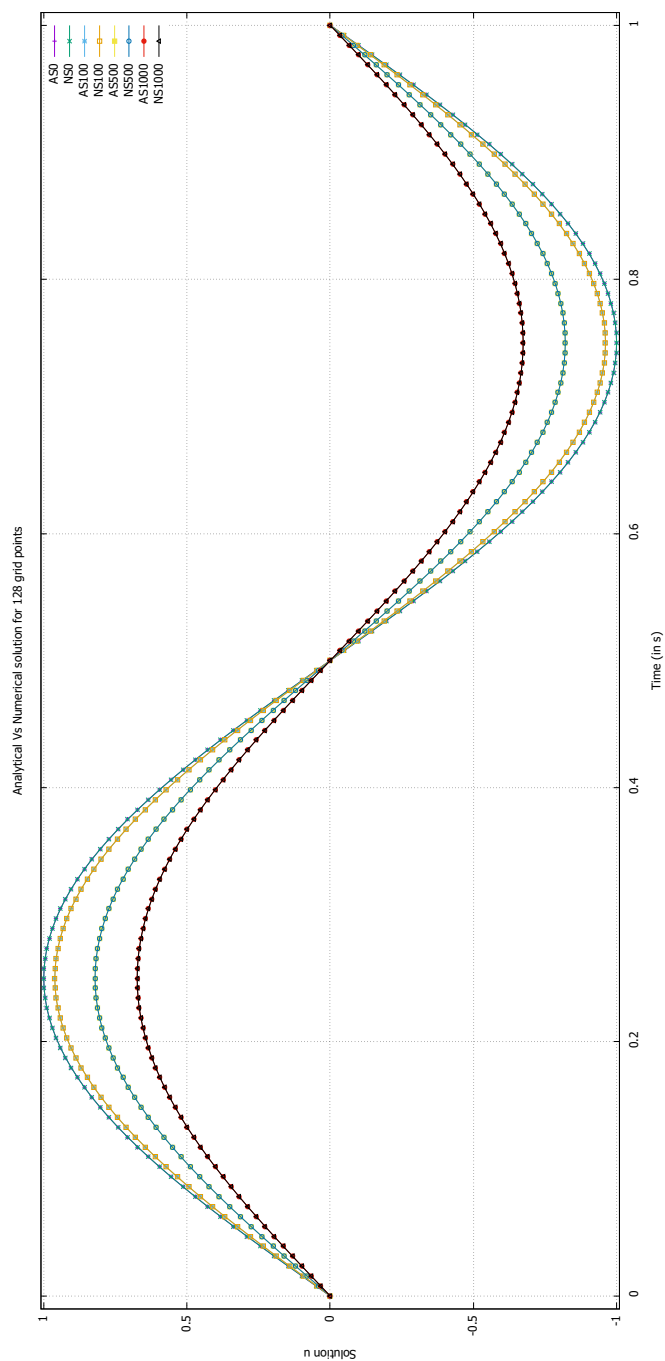
e.   **Average Error in the Domain at each time step**

Error vs Time for 128 grid points
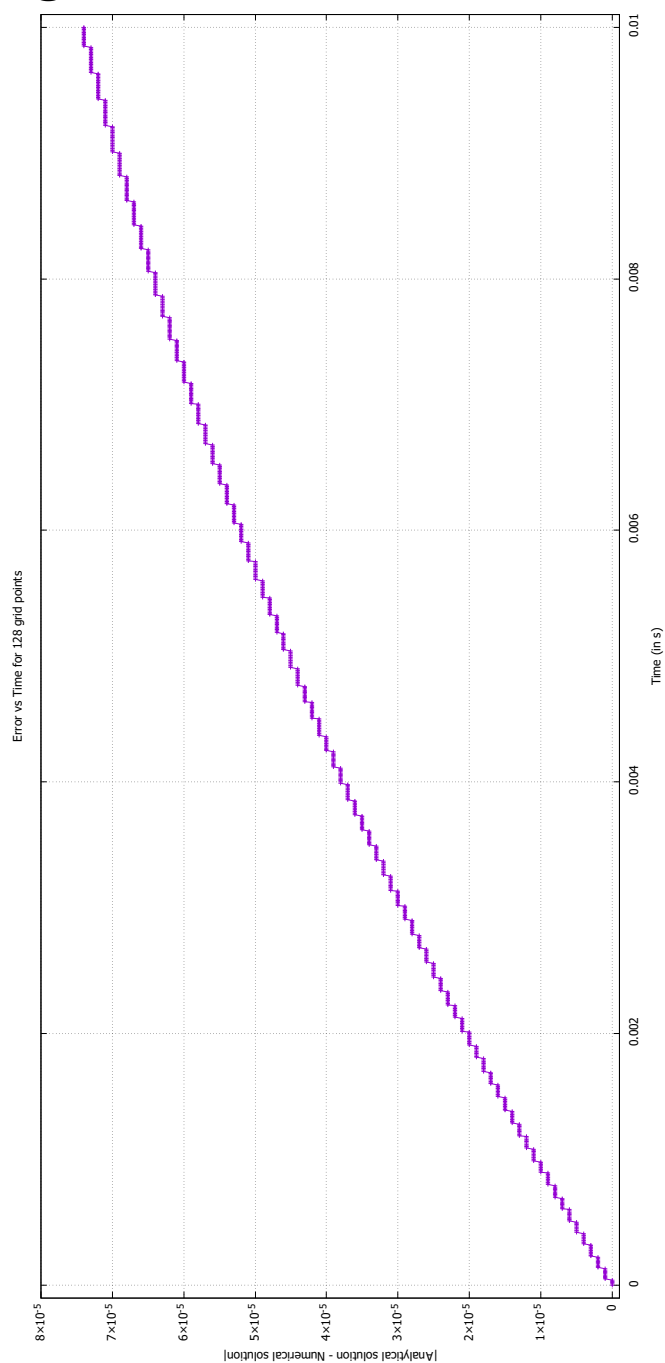
A   Appendix A: C code

```c
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,
      double k, double interval_length)
16    {
17        for(int i=0;i<grid_points;i++)
18
             as[i]=sin(2*pi*i*(interval_length/(grid_points-1)))*exp(pi*pi*(-4)*k*(time_ste
             p)*delta_t);
19    }
20    void print_to_file(double *array,int x_points, double delta_x, char *s)
21    {
22        FILE *fptr=fopen(s,"w");
23        fprintf(fptr,"# %s\n", s);
24        for(int i=0;i<x_points;i++)
25            fprintf(fptr,"%lf \t %lf\n", i*delta_x, array[i]);
26        fclose(fptr);
27    }
28    void error_method(double *error_array, double *exact_solution, double *num_solution,
      int grid_points)
29    {
30        for(int i=0;i<grid_points;i++)
31            error_array[i]=fabs(exact_solution[i]-num_solution[i]);
32    }
33    double error_sum_method(double *error_array, int grid_points)
34    {
35        double error_sum=0;
36        for(int i=0;i<grid_points;i++)
37        {
38            error_sum+=error_array[i];
39        }
40        return error_sum;
41    }
42    void initialization(double *array, int grid_points, double interval_length)
43    {
44        for(int i=0; i<grid_points;i++)
45            array[i]=sin(2*pi*i*(interval_length/(grid_points-1)));
46    }
47    void numerical_solution(double *u, double *uxx, int grid_points, double delta_t,
      double interval_length, double k)
48    {
49        //array[i+1][columns-1]=array[i+1][0];
50        for(int j=0;j<grid_points;j++)
51        {
52            u[j]+=k*delta_t*uxx[j];
53        }
54    }
55    void uxx_calculate(double *uxx, double *u, int grid_points, double interval_length)
56    {
57        uxx[0]=
           (u[1]-2*u[0]+u[grid_points-2])*(grid_points-1)*(grid_points-1)/(interval_length*in
           terval_length);
58        uxx[grid_points-1]=uxx[0];
59        for(int i=1; i<grid_points-1;i++)
60            uxx[i] =
           (u[i+1]-2*u[i]+u[i-1])*(grid_points-1)*(grid_points-1)/(interval_length*interv
           al_length);
61    }
62    int main(int argc, char *argv[])      //argv[1] = number of grid points, argv[i] =
      numerical and analytical solution printing time step
63    {
```

```c
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%lf", analytical_solution[100][32]);
126
127       initialization(u, grid_points, interval_length);
```

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