Solution of 1D heat conduction equation with constant source using Spectral Methods

Ramkumar S.

The solution of 1D heat conduction equation with constant boundary temperatures and constant heat source was done using spectral methods. The results were compared with analytical solution for a range of number of terms in the basis function series and the observation was recorded. Modern Fortran code was developed for the solution of equations and a Python3 script was made for the post-processing of the results.

I. Problem definition

The present problem is described in the fig. 1.

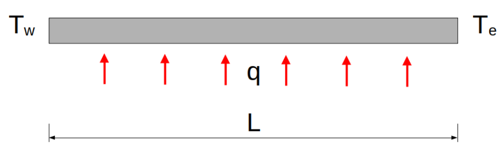


Fig. 1 problem definition

A rod of length L is taken with the constant temperatures at both ends as T_w and T_e, and with constant heat source, q. The heat conduction of the material is taken to be k. The governing equation for the present problem is given in eq. (1).

$$k\frac{\partial^2 T}{\partial x^2} + q = 0 \tag{1}$$

The analytical solution for the present problem is given in eq. (2) for $0 \leq x \leq L$.

$$T_x = -\frac{qx^2}{2k} + \frac{1}{L}\left(T_e - T_w + \frac{qL^2}{2k}\right)x + T_w \tag{2}$$

II. Solution methodology

The present problem was solved using spectral methods. The advantages of spectral methods over finite-difference can be found in the literature, at the same time, the disadvantage of spectral methods is its limited applicability to problems with simpler domain and boundary conditions. Moreover, the solution to the governing equation must be smooth, i.e. no discontinuities like shocks, for the spectral method to give better results. The assumed solution to the governing equation is given in eq. (3).

$$T_x = T_w + \frac{T_e - T_w}{L}x + \sum_{n=1}^{N} a_n sin\left(\frac{n\pi}{L}x\right) \tag{3}$$

Here, the basis function is taken as sine function, and the eq. (3) should be made such that it satisfies the boundary condition as given below.

$$\begin{aligned}\phi_n &= sin\left(\frac{n\pi}{L}x\right)\\ T_{(x=0)} &= T_w\\ T_{(x=L)} &= T_e\end{aligned}$$

Differentiating eq. (3) twice gives, eq. (4)

$$\frac{\partial^2 T}{\partial x^2} = -\sum_{n=1}^{N} a_n \left(\frac{n\pi}{L}\right)^2 sin\left(\frac{n\pi}{L}x\right) \tag{4}$$

Substituting the eq. (4) in eq. (1) gives the residual equation eq. (5) below.

$$r = -\sum_{n=1}^{N} a_n \left(\frac{n\pi}{L}\right)^2 sin\left(\frac{n\pi}{L}x\right) + q = 0 \tag{5}$$

Following Galerkin method for weighted residuals, the governing equation to determine the unknown coefficients a_n is given in eq. (6).

$$\int_0^L r\phi_i dx = 0 \tag{6}$$

$$\int_0^L \left(-\sum_{n=1}^{N} a_n \left(\frac{n\pi}{L}\right)^2 sin\left(\frac{n\pi}{L}x\right) + q\right) sin\left(\frac{i\pi}{L}x\right) = 0$$

Here the subscripts i and n are like nested loops with same range, hence taking advantage of orthogonal basis functions, the derived final expression for the coefficients is given in eq. (7).

$$a_n = \frac{2q}{k}\frac{L^2}{(n\pi)^3}\left(1-(-1)^n\right) \tag{7}$$

The eq. (7) gives the coefficient values till N series terms. The order of accuracy of the solution depends on the number of terms in the series taken. Substituting the computed a_n values in the eq. (3) will give the temperature distribution.

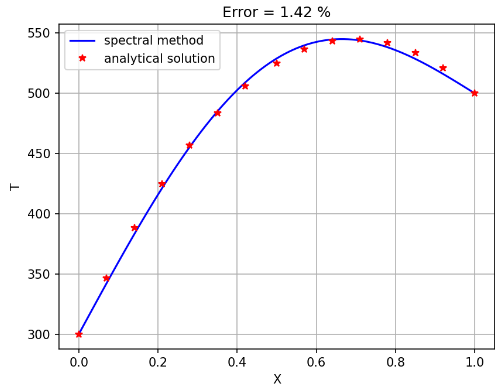
III. Results

The results obtained for different N series terms is given in fig. 2, along with the error percentage for each case. It can be seen that the error percentage of the results obtained in comparison with analytical solution, reduces as the number of terms in the sine series increases. The solution matched exactly with the analytical solution for the terms count of N = 40.

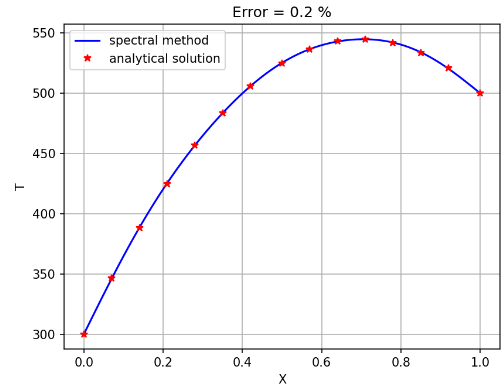
IV. Conclusion and future works

The solution to 1D heat conduction equation with constant source terms and boundary conditions was obtained successfully using the spectral methods. The present work will be further extended to 2D heat conduction and later to the solution of Euler/Navier Stokes equations for a driven cavity flow problem.

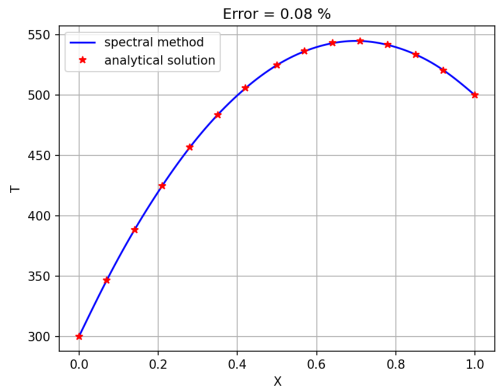
The FORTRAN program files and Python code of the present work is given in Section A.



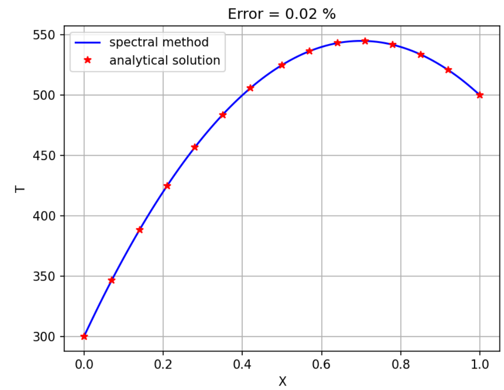
(a) N = 1



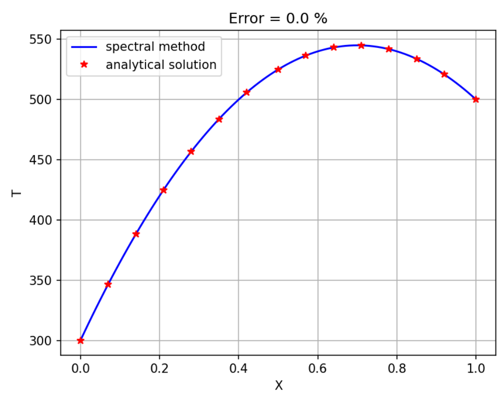
(b) N = 5



(c) N = 10



(d) N = 20



(e) N = 40

Fig. 2 variation of accuracy of the results computed using spectral methods

A. Appendix - FORTRAN and Python codes

This section contains the FORTRAN and Python codes used in the present work

Below is the main FORTRAN program that solves the present problem.

```fortran
!-----------------------------------------------------------------------
! Solutin of 1D heat conduction with source using spectral methods
!
! Main fortran file
!
! developed by Ramkumar S.
!-----------------------------------------------------------------------

! begin main program
program main

    ! using parameters and model_variables modules
    use parameters
    use model_vars

    ! explicit definition of variable types
    implicit none

    ! declaring needed variables
    integer(kind=ikd) :: i,n
    real(kind=rkd) :: sumVal = 0.0, an, A = 1.0/L*(Te-Tw + q*L**2/2.0/K)

    ! initializing the variables
    call initializer()

    print *, "Computing temperature at each node"

    ! main loop to compute temperature at each node point
    do i = 1,Nx
        ! initializing sum variable
        sumVal = 0.0
        ! sub loop running for summation of sine terms
        do n = 1,N_terms
           ! computing coefficient
           an = 2.0*Q/k*L**2/(n*pi)**3*(1.0-(-1.0)**n)

           ! adding to the sum variable
           sumVal = sumVal + an*sin(n*pi/L*X(i))

           ! computing the current temperature node value
           T(i) = Tw + (Te-Tw)*X(i)/L + sumVal

           ! computing analytical solution
           T_analytical(i) = -q*X(i)**2/2.0/k + A*X(i) + Tw
        end do
    end do

    ! writing data to the file
    call write()

end program main
```

Below is the parameters FORTRAN file containing modules of computation parameters

```fortran
!-----------------------------------------------------------------------
! Solutin of 1D heat conduction with source using spectral methods
!
! parameters and model variables module definition file
!-----------------------------------------------------------------------

! computation parameter variables definition
module parameters

    ! explicit definition of variable types
    implicit none

    ! defining precision and type for real and integer kind variables
```

```fortran
    integer, parameter :: ikd = selected_int_kind(8)
    integer, parameter :: rkd = selected_real_kind(8,8)

    ! number of terms to be taken in the sine series and number of vertex
    ! points to be taken in the x direction
    integer(kind=ikd), parameter :: N_terms = 40, Nx = 101

    ! length in x direction
    real(kind=rkd), parameter :: L = 1.0

    ! defining pi value
    real(kind=rkd), parameter :: PI = 4.0*atan(1.0)

    ! defining thermal conductivity and heat source values
    real(kind=rkd), parameter :: k = 1.0, q = 1000.0

    ! defining temperature values at west and east end of the domain
    real(kind=rkd), parameter :: Tw = 300.0, Te = 500.0

end module parameters

! model variables definition
module model_vars

    ! using parameters module
    use parameters

    ! explicit definition of variable types
    implicit none

    ! defining temperature and position arrays
    real(kind=rkd), dimension(Nx) :: T, X, T_analytical

end module model_vars
```

Below is the FORTRAN file containing the subroutines needed in the computations.

```fortran
!------------------------------------------------------------------------
! Solutin of 1D heat conduction with source using spectral methods
!
! subroutines definition file
!------------------------------------------------------------------------

! defining initializer subroutine
subroutine initializer()

    ! using parameters and model variables modules
    use parameters
    use model_vars

    ! explicit declaration of variable types
    implicit none

    ! declaring needed variables
    integer(kind=ikd) :: i
    real(kind=rkd) :: dx = L/float(Nx-1)

    print *,"Initializing the variables"

    ! looping through to initialize the temperature and position variables
    do i = 1,Nx
        T(i) = Tw
        X(i) = float(i-1)*dx
    end do

end subroutine initializer

! defining solution writer subroutine
```

```fortran
subroutine write()

    use parameters
    use model_vars

    implicit none

    ! declaring needed variables
    integer(kind=ikd) :: i

    ! declaring format to be followed in writing to the file
    50 format(f7.5,",",f9.5,",",f9.5)

    ! opening file to write the data
    open(unit=1, file="data.csv", status="replace")

    ! writing header line
    write(unit=1, fmt='(A)') "X,T,T_a"

    ! writing the data to file
    do i = 1,Nx
        write(unit=1, fmt=50) X(i),T(i),T_analytical(i)
    end do

    ! closing the file
    close(unit=1)

    print *,"data written to file"

end subroutine write
```

Below is the Makefile that is used to compile and execute the program.

```makefile
parameters.o: parameters.f90
	gfortran -c parameters.f90
subroutines1.o: subroutines1.f90
	gfortran -c subroutines1.f90
main.o: main.f90 parameters.mod
	gfortran -c main.f90
run.exe: main.o subroutines1.o
	gfortran *.o -o run.exe
run: run.exe
	./run.exe
	python script.py
clean:
	rm -f *.o *.mod *.exe
```

And below is the Python script used for plotting and error computation.

```python
#!/bin/python3
"""
1D heat conduction with source using spectral methods

plotting and error computation python script file
"""

# importing needed modules
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# reading the data file written by FORTRAN code
fid = pd.read_csv("data.csv")

# computing error percentage
error = np.max(np.abs(fid["T"]-fid["T_a"])/fid["T_a"])*100.0

# computing number of points to be displayed for analytical solution
```

```
N = int(fid.shape[0]*0.15)
Nval = np.linspace(0,fid.shape[0]-1,N, dtype=int)

# plotting graph
plt.figure()
plt.plot(fid["X"],fid["T"],'-b',label="spectral method")
plt.plot(fid["X"].iloc[Nval],fid["T_a"].iloc[Nval],'*r',label="analytical solution")
plt.grid()
plt.legend()
plt.xlabel("X")
plt.ylabel("T")
plt.title("Error = "+str(np.round(error,2))+" %")
plt.savefig("output.png", dpi = 150)

plt.show()
```

****