Solver for the unsteady 2D heat equation

Mridul Mani Tripathi

German Research School for Simulation Sciences GmbH
Laboratory for Parallel Programming
mridul.tripathi@rwth-aachen.de

Abstract: The aim of this project is to solve the heat diffusion equation in 2D. This model also has an analytical solution to which the obtained numerical results can be compared. Mesh file is read first and its data such as number of elements, number of nodes, node coordinates, element connectivity, gauss quadrature points etc. are stored. Element matrices M, K, F are calculated after calculating and storing jacobian for each element. Dirichlet boundary condition is applied. For solving the given heat equation, explicit first order time integration scheme is used. Further global error and discretization error are calculated, latter being calculated from analytical solution.

1 Introduction

The aim of the project is to solve the heat diffusion equation in 2D to obtain the temperature distribution in a plane disk heated by a localized heat source. This heat source is assumed to be acting only on a subset of the domain represented by a small disk. Although analytical methods are still popular nowadays, they are usually only available for simple situations and conditions. In comparison to analytical methods, numerical methods could only provide results approximately within an acceptable error tolerance, but they are more flexible to deal with the complicated yet practical situations.

2 Description of problem

The plane disk is heated by a localized heat source that is shown by gray area in figure 1. The disk has radius R2 = 0.1m. A fixed temperature To = 300K is set on the disk boundary, and a heat source of total power P = 1W is applied on a small circular area of radius R1 = 0.01m centered at the origin.

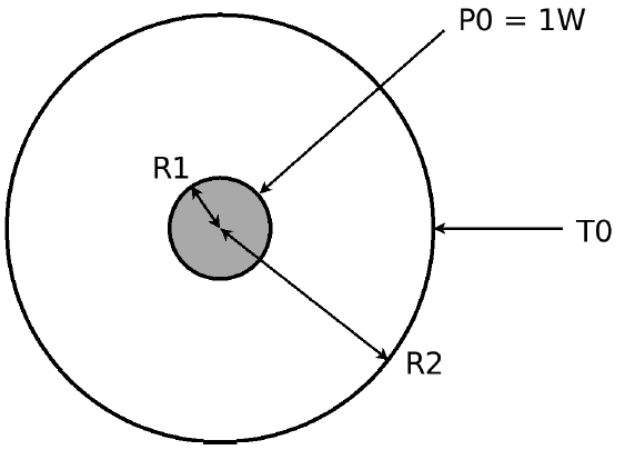

Figure 1: Geometry and boundary conditions

The formulation of the problem to solve is:

$$\begin{cases} \frac{\partial T}{\partial t} - \alpha \nabla^2 T = & f \ \ in\ the\ disk\ domain \\ T = To & on\ the\ disk\ boundary \end{cases}$$

Where T, α, and f are respectively the temperature, thermal diffusivity, and the heat source.

The heat source changes based on the location of the node:

$$f(r) = \begin{cases} \frac{Q}{\pi R_1^2} & if\ r < R_1 \\ 0 & if\ r > R_1 \end{cases}$$

$Q = P/d_z$ is the volumetric heat source. $d_z = 0.1m$ is the out of plane thickness. The analytical solution is also provided to compute the discretization error in the later stage. The analytical solution of the problem is:

$$T(r) = \begin{cases} To - \frac{Q}{2\pi\alpha}\left(\frac{1}{2}\left(\frac{r^2}{R_1^2} - 1\right) + ln\left(\frac{R_1}{R_2}\right)\right) & if\ \ r < R_1 \\ To - \frac{Q}{2\pi\alpha} ln\left(\frac{r}{R_2}\right) & if\ \ r \geq R_1 \end{cases}$$

3 Methodology

To begin with the problem, mesh file is converted to make mesh readable in paraview as a .pvtu file. This mesh file is read first and the data stored in it is obtained such as number of elements, number of nodes, node coordinates, element connectivity, gauss quadrature points etc. In solver file, initially jacobian of all elements is calculated and stored. Element matrices M, K, F are calculated next using the simple formulas mentioned below.

$$M_{ij}^e = \sum_{m=1}^{nGQP} w(m) S_i(\varepsilon_m, n_m) S_j(\varepsilon_m, n_m) |J_m|$$

$$K_{ij}^e = \sum_{m=1}^{nGQP} \alpha w(m) (S_{i,x}(\varepsilon_m, n_m) S_{j,x}(\varepsilon_m, n_m) + S_{i,y}(\varepsilon_m, n_m) S_{j,y}(\varepsilon_m, n_m)) |J_m|$$

$$F_i^e = \sum_{m=1}^{nGQP} f w(m) S_i(\varepsilon_m, n_m) |J_m|$$

w(m): Gauss weights; S: shape function in reference element; J_m: determinant of the jacobian

Since the mesh is unstructured, the isoparametric formulation is used. The idea is to define the shape functions and shape functions derivatives in a reference element and use a mapping from the reference element to an arbitrary deformed element. It is made sure that matrices are initialized and contributions from the previous quadrature points are not overwritten. The source term is applied conditionally based on the position of the node. Next the lumped mass matrix is computed from previously computed consistent mass matrix. The mass information is stored in two classes, i.e. vector and matrix. All the matrices are stored using helper functions.

Further, Dirichlet boundary condition is applied on the nodes at the boundary, i.e. nodes at distance of 0.1m from the center with tolerance of 0.1%. Temperature at boundary is initialized in global temperature array using helper function.

For solving the given heat equation, explicit first order time integration scheme is used.

$$\frac{dT}{dt}|_n = \frac{T^{n+1} - T^n}{\Delta t} + O(\Delta t)$$

This approximation is added to the matrix form with unknown quantities moved to left.

$$[M]\{T\}^{n+1} = [M]\{T\}^n + \Delta t\ (\{F\}^n - [K]\{T\}^n)$$

Using the above stated formulas, the new temperature is calculated for non-boundary nodes after each iteration. Global error is calculated, and residual is found using scaled global error.

For validation of the code, residual should be approximately equal to 10^{-7}. Now this solution is compared with analytical solution and discretization error is calculated, which shows how accurate our finite element code reproduces the correct solution.

Code excerpts are attached below.

```
void femSolver::calculateElementMatrices(const int e)
{
   // Add code here
  double M[nen][nen] = {0.0};
  double K[nen][nen] = {0.0};
  double F[nen] = {0.0};
  double a = settings->getD();    //thermal diffusivity
  double * xyz = mesh->getXyz();
  double f = settings->getSource();    //source term


  //computing M, K, F
  for(int k=0; k < nGQP; k++)    //loop over quadrature points
   {
    double w = mesh->getME(k)->getWeight();
    double J = mesh->getElem(e)->getDetJ(k);

    for(int i=0; i<nen; i++)
  {
   double Si = mesh->getME(k)->getS(i);
   double Six = mesh->getElem(e)->getDSdX(k,i);
   double Siy = mesh->getElem(e)->getDSdY(k,i);

   for(int j=0; j<nen; j++)
    {
     double Sj = mesh->getME(k)->getS(j);
     double Sjx = mesh->getElem(e)->getDSdX(k,j);
     double Sjy = mesh->getElem(e)->getDSdY(k,j);
     M[i][j] = M[i][j] + w*(Si*Sj)*J;
     K[i][j] = K[i][j] + a*w*(Six*Sjx + Siy*Sjy)*J;
    }

   int n = mesh->getElem(e)->getConn(i);    //global node number
   double x = xyz[n*nsd + xsd];
   double y = xyz[n*nsd + ysd];
   double r = sqrt(x*x + y*y);
   if(r<0.01)
    {
     F[i] = F[i] + w*Si*J*f;
    }
   else
    {
     F[i]=0;
    }
  }

   }

//Saving computed matrices

  double Me = 0;    //sum of all elements of M matrix
  double Mj = 0;    //sum of diagonal elements of M matrix
  double ML[nen];    //lumped mass matrix
```

```
 for(int i=0; i<nen; i++)
  {
   mesh->getElem(e)->setF(i,F[i]);    //set F vector
   Mj = Mj + M[i][i];
   for(int j=0; j<nen; j++)
 {
  Me = Me + M[i][j];
  mesh->getElem(e)->setK(i,j,K[i][j]);    //set K matrix
 }

  }

 //Mass lumping
 for(int i=0; i<nen; i++)
  {
   int n = mesh->getElem(e)->getConn(i);
   ML[i] = M[i][i]*(Me/Mj);
   mesh->getNode(n)->addMass(ML[i]);
   mesh->getElem(e)->setM(i,ML[i]);    //set M matrix
  }

  return;
}
```

Code 1: calculateElementMatrices(int e)

```
void femSolver::applyDrichletBC()
{
  // Add code here

  int nn = mesh->getNn();
  double * xyz = mesh->getXyz();
  double * T = mesh->getT();    //global temperature array
  double x,y;
  int d=0;    //number of dirichlet nodes
  for(int i=0; i<nn; i++)
   {
 x = xyz[i*nsd+xsd];    //coordinates for node i
 y = xyz[i*nsd+ysd];
 double r = sqrt(x*x + y*y);
 double tol = (r - 0.1)*100/0.1;    //0.1% tolerance
 if(tol >= -0.1 && tol<= 0.1)
  {
   d = d+1;
   mesh->getNode(i)->setBCtype(1);    //dirichlet boundary condition
   T[i] = settings->getBC(1)->getValue1();
  }

   }

  cout<<endl<<"Number of Dirichlet Nodes = "<<d<<endl;

  return;

}
```

Code 2: applyDirichletBC()

```
void femSolver::explicitSolver()
{
   // Add code here

 int iter = settings->getNIter();
 double dt = settings->getDt();
 int ne = mesh->getNe();
 int nn = mesh->getNn();
 double * MTnew = mesh->getMTnew();
 double * T = mesh->getT();
 double * massG = mesh->getMassG();
 double scale,res,gerror;
 double lim = pow(10,-7);
 int i;

 for(i=1; i<=iter; i++)     //loop over iterations
  {

    for(int j=0; j<ne; j++)     //loop over elements
 {
   double * M = mesh->getElem(j)->getMptr();
   double * K = mesh->getElem(j)->getKptr();
   double * F = mesh->getElem(j)->getFptr();

   for(int k=0; k<nen; k++)     //calculating MTnew
    {
      int n = mesh->getElem(j)->getConn(k);
      double KT = 0.0;
      double MT = 0.0;

      for(int l=0; l<nen; l++)
   {
     int p = mesh->getElem(j)->getConn(l);
     KT = KT + K[k*nen+l]*T[p];
   }

      MT = M[k]*T[n];
      MTnew[n] = MTnew[n]+ MT + dt*(F[k] - KT);

    }
 }

   gerror = 0.0;   //global error
   int nd = 0;     //non-dirichlet nodes

   for(int j=0; j<nn; j++)     //loop over nodes
 {
  int type = mesh->getNode(j)->getBCtype();
  if(type!=1)     //non-dirichlet node
   {
     nd = nd +1;
     double Tnew = MTnew[j]/massG[j];     //new temperature at current node
     double error = (Tnew-T[j])*(Tnew-T[j]);
     gerror = gerror + error;
     T[j] = Tnew;     //storing new temperature
   }

  MTnew[j]=0.0;
 }

   gerror = sqrt(gerror/nd);     //computing global error

   if(i==1)
 {
      scale = gerror;     //scaled value for first iteration
```

```
    cout<<"Number of Non-Dirichlet Nodes = "<<nd<<endl;
  }
    res = gerror/scale;    //rms error
    if(res<lim)
  break;
   }

  cout<<endl<<"Global Error = "<<gerror<<endl;
  cout<<"RMS Error = "<<res<<endl;
  cout<<"Iterations run = "<<i<<endl;

  return;
}
```

Code 3: explicitSolver()

```
void postProcessor::compareAnalytical()
{
  int nn = mesh->getNn();
  double * T = mesh->getT();
  double * xyz = mesh->getXyz();
  double x, y, radius, Ta;

  // Add code here

  double To=300, Q=10, pi=3.14, coeff, error=0;
  double a=settings->getD();
  x=0;
  y=0;
  radius=0;
  coeff=(double)(Q/(2*pi*a));

  for(int i=0;i<nn;i++)
     {
     x=xyz[i*nsd+xsd];
     y=xyz[i*nsd+ysd];
     radius=sqrt((x*x+y*y));
     Ta=0;

     if(radius<0.01)
        {
        Ta=To-(coeff)*(0.5*(((double)(radius*radius)/(0.01*0.01))-1)+log(0.1));
        }
     else if(radius>=0.01)
        {
        Ta=To-(coeff)*log(radius/0.1);
        }
     else
        cout<<"Error in Code"<<endl;

     error=error+((T[i]-Ta)*(T[i]-Ta));
     }
  error=sqrt((double)(error)/nn);
  cout<<endl<<"Discretization Error = "<<error<<endl;
  return;
}
```

Code 4: compareAnalytical()

4 Results

Meshes (coarse, fine, finest) with resulting temperature is obtained. The images are shown below. Distance is the diameter of the disk in the plots of Temp vs Distance.

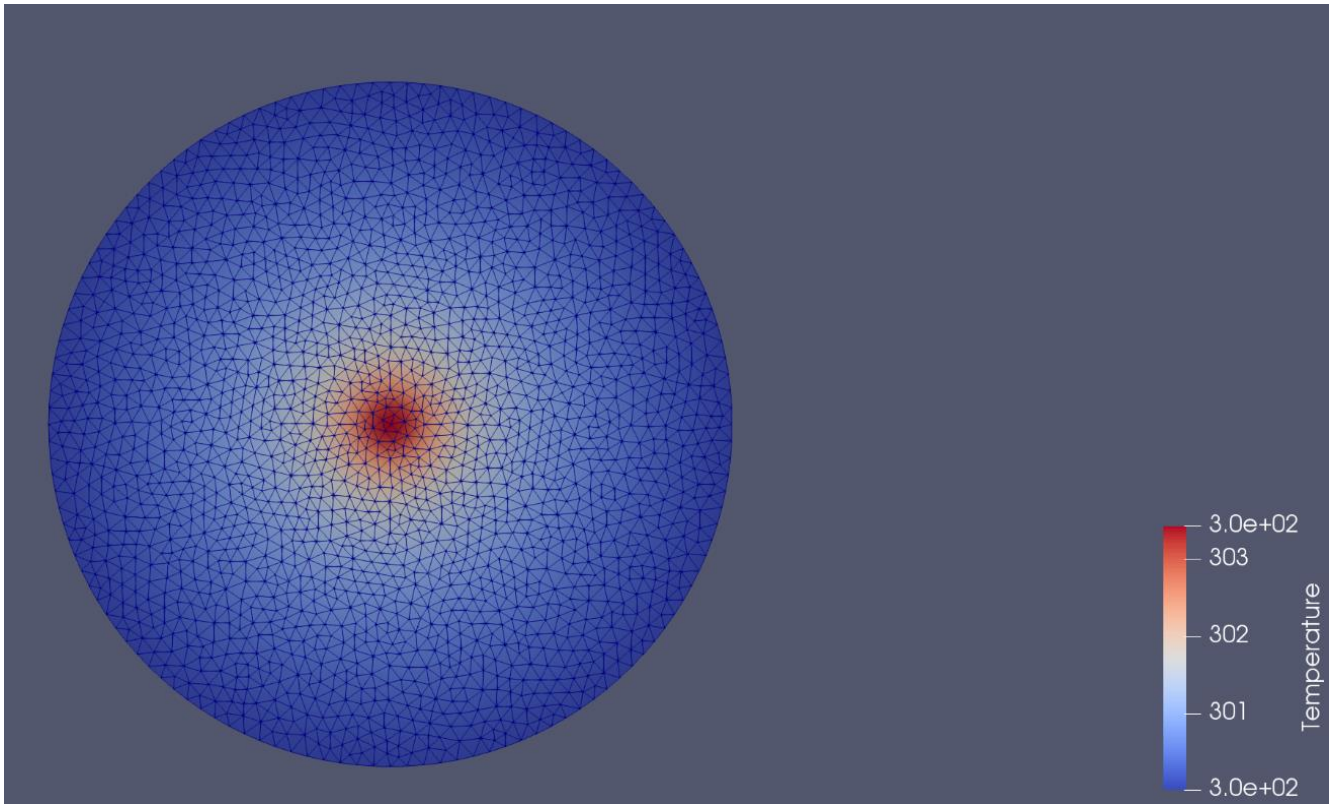

Figure 2: Coarse mesh with temperature distribution

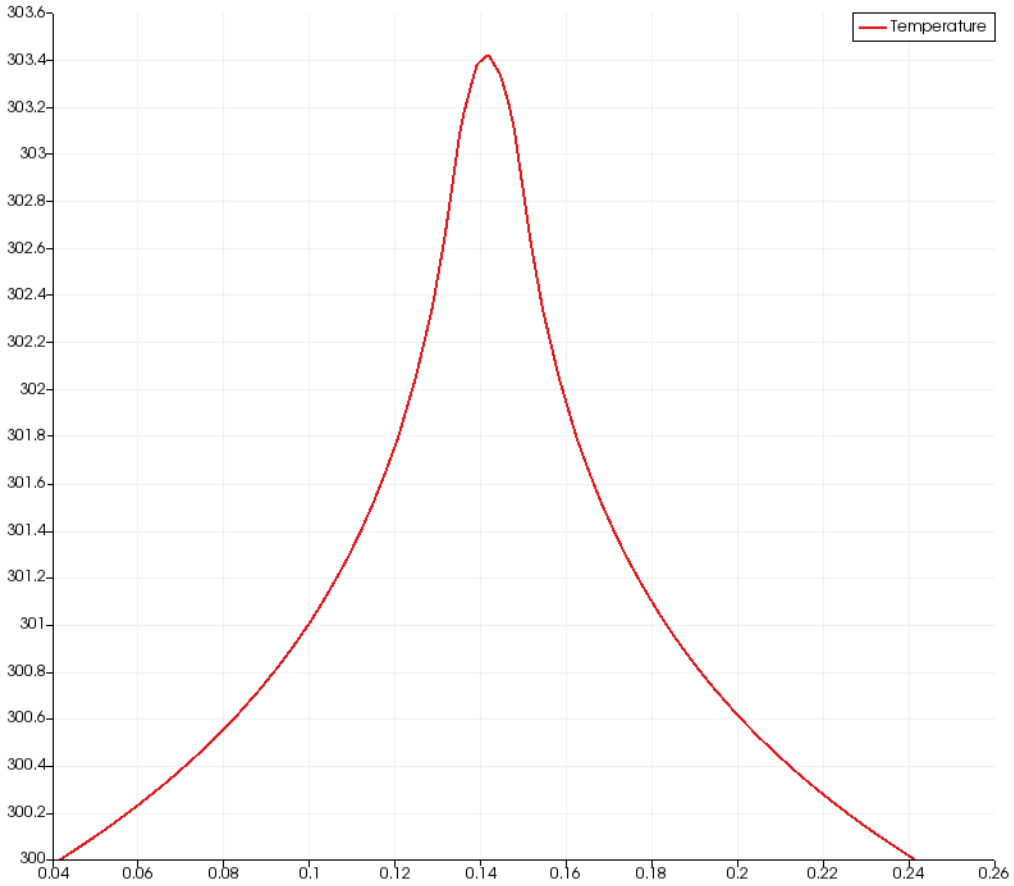

Figure 3: Temperature(K) vs Distance(m) for coarse mesh

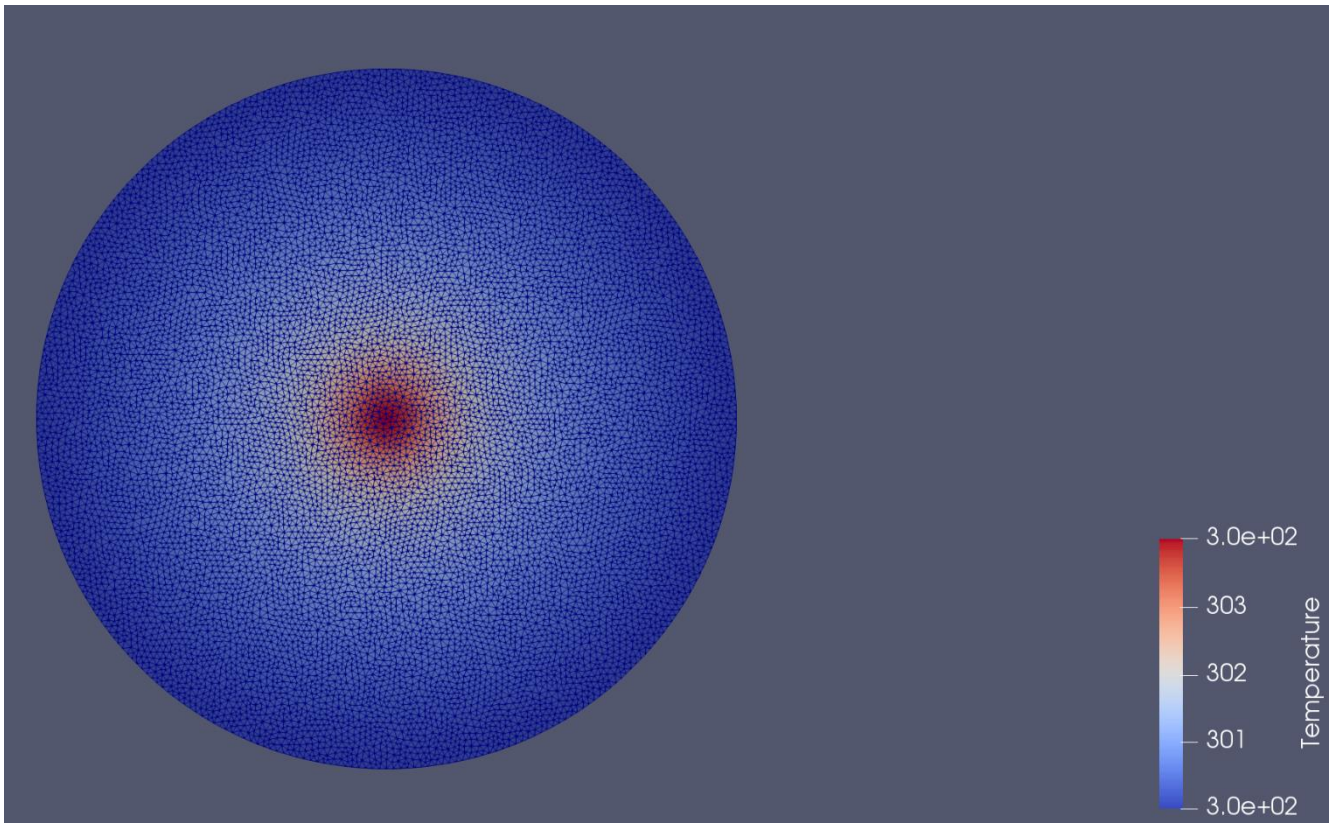

Figure 4: Fine mesh with temperature distribution

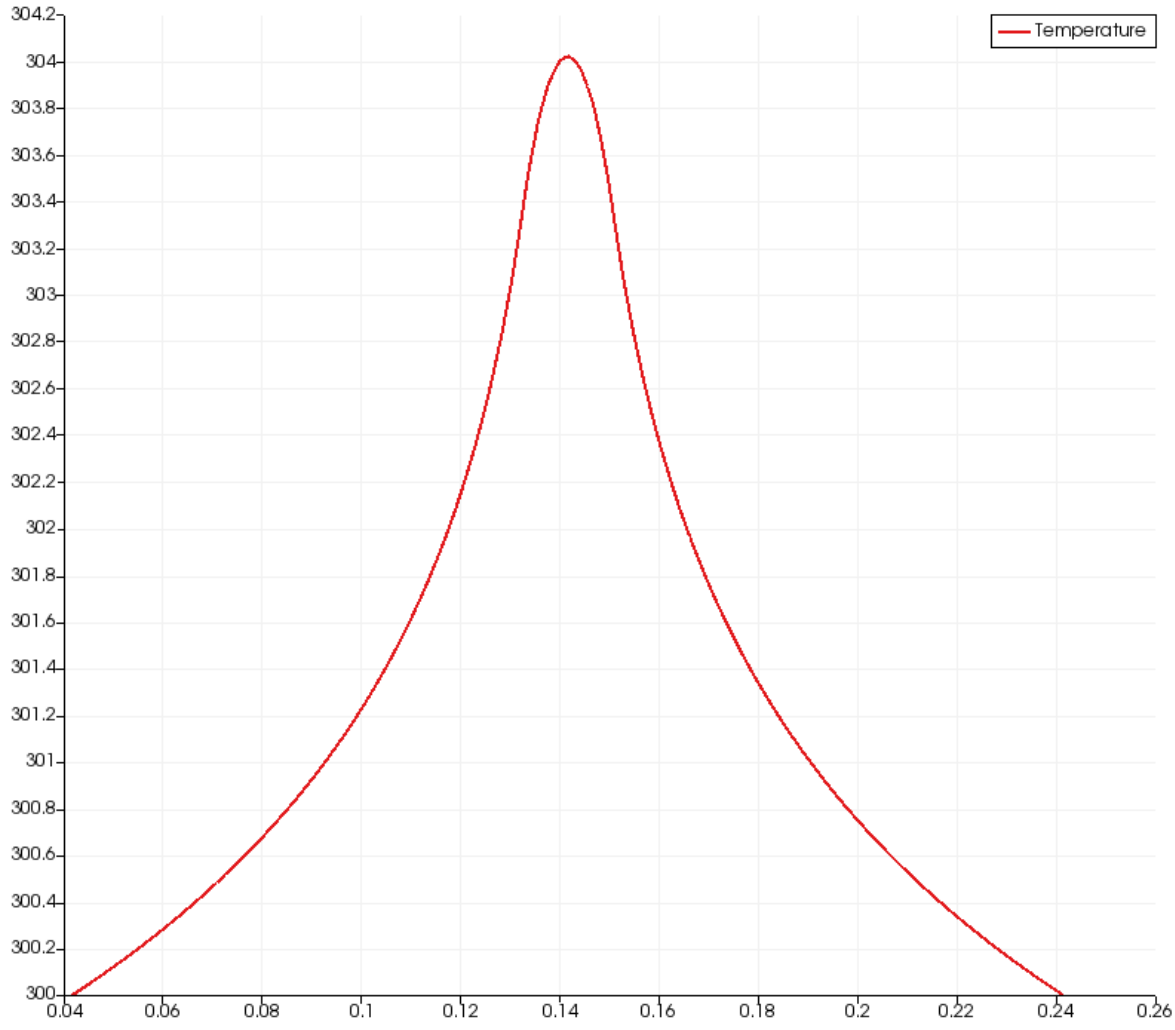

Figure 5: Temperature(K) vs Distance(m) for fine mesh

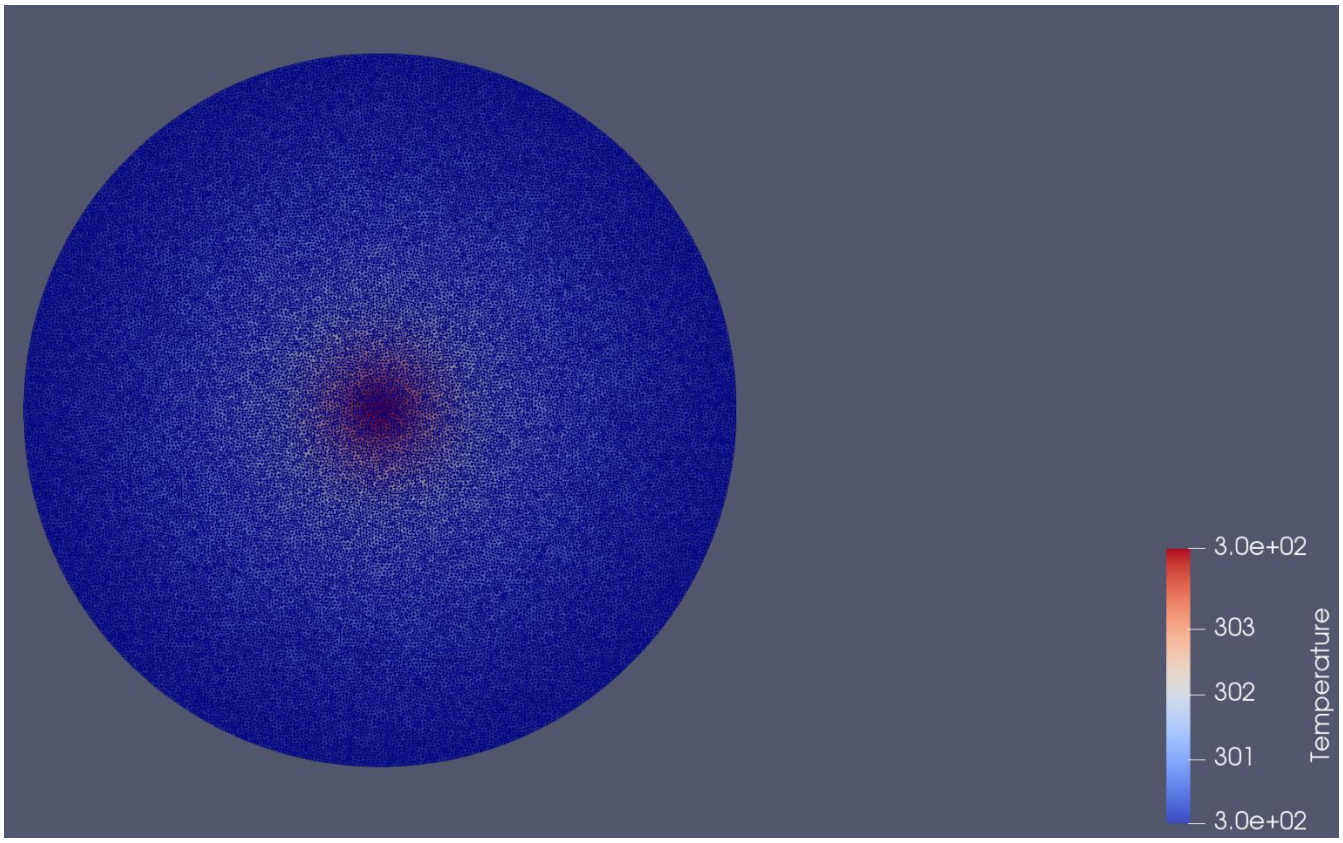

Figure 6: Finest mesh with temperature distribution

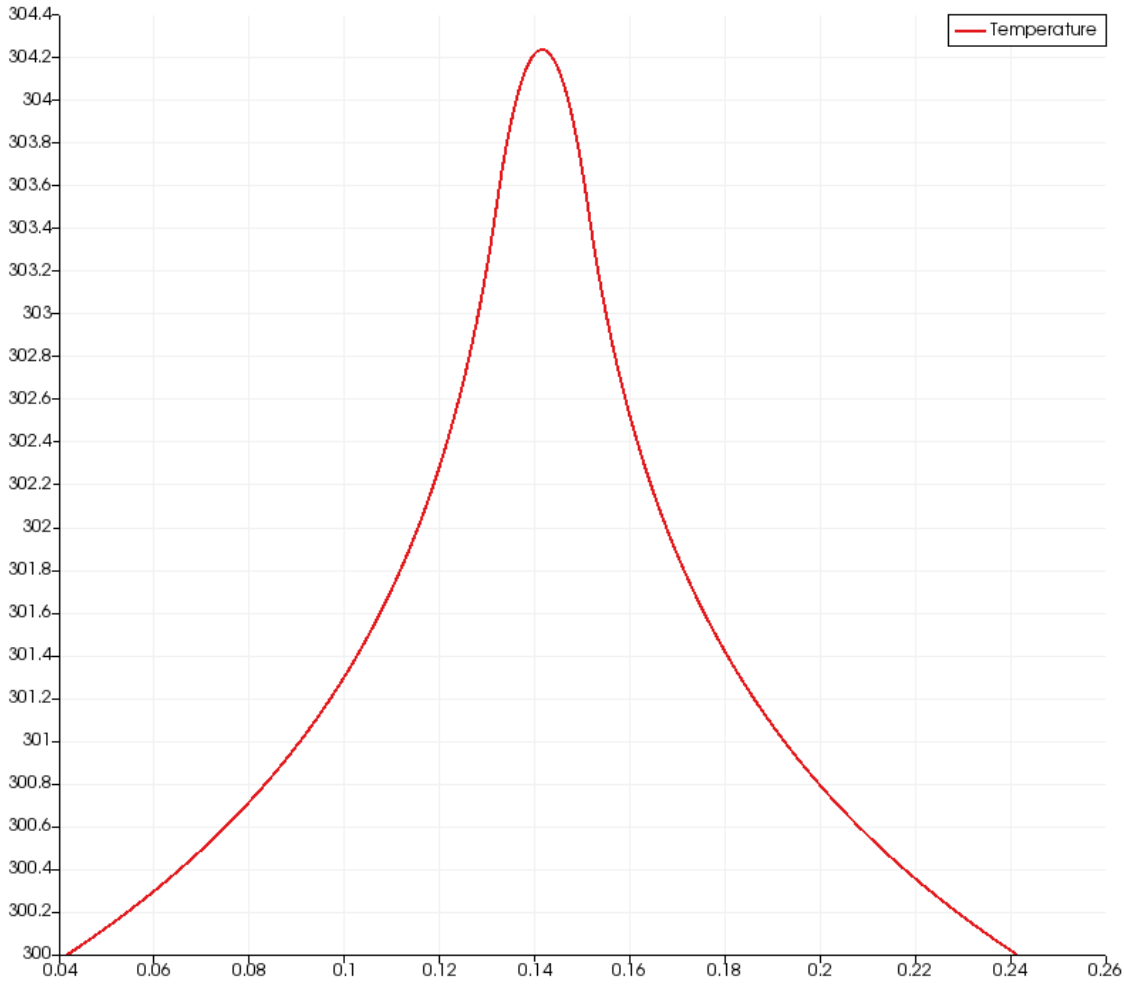

Figure 7: Temperature(K) vs Distance(m) for finest mesh

The errors i.e. global error and discretization error obtained are plotted against number of elements. This shows that magnitude of global error decreases significantly with increase in number of elements. Therefore, the global error is minimum for finest mesh. And for discretization error, which shows how accurate our numerical solution is, it is also minimum for finest mesh since large quantities of small elements result in better approximation. These errors help us understand about convergence of the solution.

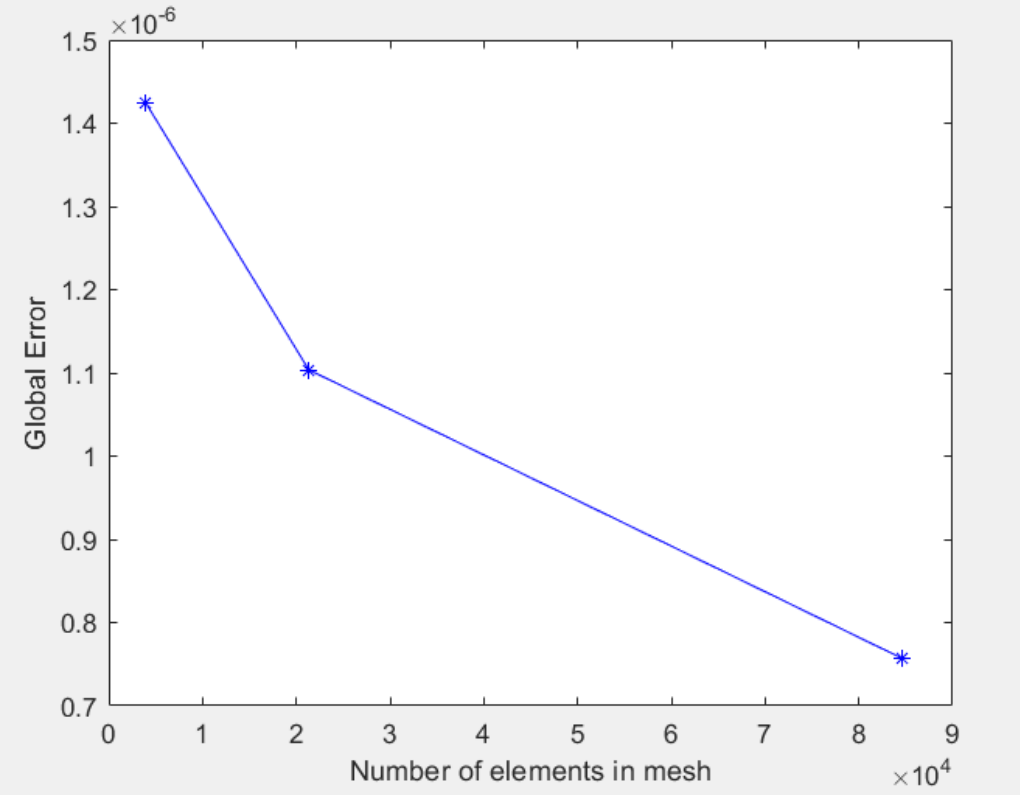

Figure 8: Global error vs number of elements in mesh

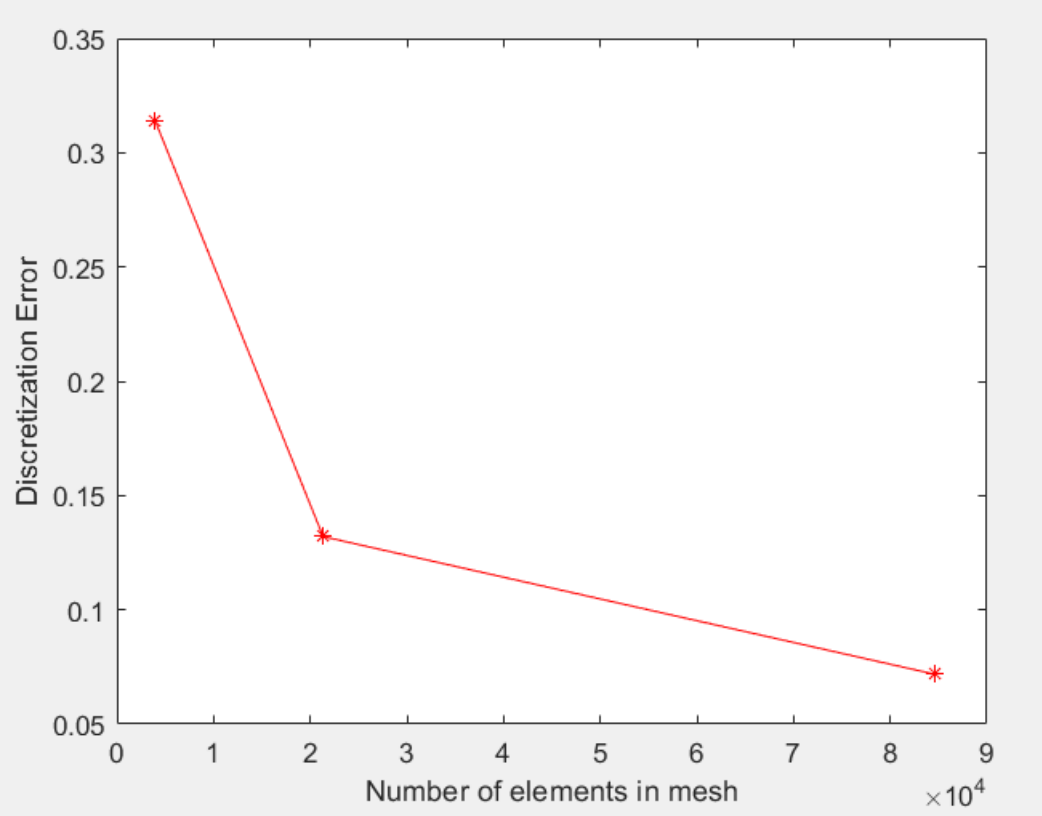

Figure 9: Discretization error vs number of elements in mesh

Few questions that arise:

1. Comparing results from different meshes, what can be said about convergence?

   As seen, error decreases with increase in number of elements in mesh. Thus, refined mesh is necessary for a more accurate converging solution. But for a given mesh, a converging solution is reached earlier if element density is low, but that solution need not be accurate.

2. To test the node position, can you compare directly the node distance from the centre with the theoretical radius of 0.1m? Is there any numerical limitation?

   We can directly compare node distance from centre only for nodes that are well within the boundary of the meshed geometry. But this method has its limitation for nodes at boundary. For nodes at boundary, a very small (generally 1% of element length) tolerance is taken for locating boundary nodes and to avoid mesh distortion.

3. What will happen if time step size is increased and why?

   If we increase time step size, the global error will increase. This happens because large values of time step require more iterations to get stable solution, thus if maximum number of iterations is limited, then the solver will fail. But it also depends on mesh density. The relation between mesh density and time step is given by COURANT NUMBER. For diffusion problem, this number should be low (much less than 1). For example, for coarse mesh, if time step increases by very small value then global error increases but number of iterations to get stable solution decreases but if time step increases by large value, then number of iterations to get convergence increases.

   Thus, if time step increases, global error will increase but number of iterations for convergence also depends on mesh density.

5 Conclusion

Basics of a explicit solver have been studied in this paper. The given 2D heat diffusion problem has been solved numerically and verified with its analytical solution. In this paper, effect of mesh density on errors has been discussed along with convergence of a solution i.e. increase in mesh density results in decrease in global error. Temperature distribution is obtained for different meshes and plotted with respect to distance along diameter of the disk.

Finally, solver is ready to compute the numerical solution which is approximately equal to analytical solution.

References

[1]https://www.researchgate.net/post/How_to_select_the_optimized_time_step_value_for_FEM_solution_of_transient_heat_transfer_problems

[2]https://www.rocscience.com/assets/resources/learning/papers/Effects-of-Time-Step-Size-on-the-Efficiency-of-Discontinuous-Deformation-Analysis.pdf

[3]https://www.researchgate.net/post/Are_the_results_of_an_implicit_solution_dependent_on_the_time_step_size

[4]https://books.google.de/books?id=UogDuM2q08AC&pg=PA46&lpg=PA46&dq=what+happens+if+step+size+is+increased+in+fem&source=bl&ots=hVrnQEps0i&sig=ACfU3U3aMoFMMa-dUGI93JV3RqqCeE6vcA&hl=en&sa=X&ved=2ahUKEwiQ0JvSjsXpAhVGUhoKHeWGA9QQ6AEwBXoECAsQAQ#v=onepage&q=what%20happens%20if%20step%20size%20is%20increased%20in%20fem&f=false

[5] https://core.ac.uk/download/pdf/41822299.pdf

[6] https://onlinelibrary.wiley.com/doi/pdf/10.1002/cnm.1640091103

[7] https://hal.archives-ouvertes.fr/hal-00678795/document

[8] https://mae.ufl.edu/nkim/egm6352/Chap5.pdf