MATLAB Solution of Flow and Heat Transfer through a Porous Cooling Channel and the Conjugate Heat Transfer in the Surrounding Wall

James Cherry, Mehmet Sözen
Grand Valley State University, cherryj1@gmail.com, sozenm@gvsu.edu

***Abstract* – A mathematical model was developed for simulating the transport phenomena in a porous cooling channel and the conjugate heat transfer in the surrounding wall. The numerical solution by the use of variable grid finite difference method was implemented by using MATLAB computing environment, a fourth generation programming language. Several case studies performed showed the behavior of the flow field in the porous cooling channel and the temperature distribution in the two domains of the problem. The use of MATLAB computing environment was shown to have several major advantages over the use of third generation programming languages such as FORTRAN and C. It was observed that when a numerical method involving extensive matrix operations was chosen, the built-in functions in MATLAB made the solution much simpler, requiring considerably less math background and programming efforts/skills. MATLAB also offered easy to use post-processing tools not available with other languages.**

Index Terms – Enhanced heat transfer, MATLAB, finite difference method, porous cooling channel

INTRODUCTION

In applications where cooling of structures by the use of conventional cooling channels proves to be inadequate, use of porous cooling channels may provide a viable solution to the thermal management problem. In this arrangement the conventional cooling channel is fitted with a porous insert, thereby yielding an effective thermal conductivity of the coolant-porous matrix that may be orders of magnitude larger than that of the coolant used alone. The mathematical model for the prevailing transport phenomena in the porous cooling channel and the conjugate heat transfer in the surrounding wall is well established. Several different forms of numerical solution methods may be used for obtaining a steady-state or unsteady solution to the problem. In particular for simple geometries, finite difference method offers one of the easiest ways of implementing a solution to the problem.

Third generation programming languages (3GL) such as BASIC, FORTRAN and later C were the most commonly used languages for implementing the finite solution. With the development of fourth generation languages (4GL) such as MATLAB, solving such problems became easier. Implementing such solution schemes does not require as much mathematical background and more importantly as much programming skills as in the past. In the current work, using MATLAB we implemented the steady state finite difference solution of the fluid flow and heat transfer through a porous rectangular cooling channel and the heat conduction through the surrounding walls of uniform thickness. The following sections of the paper will discuss in detail how this computing environment was used and the pros and cons of utilizing MATLAB over other traditional 3GL languages.

GOVERNING EQUATIONS

The schematic diagram of the rectangular channel that was considered is shown in Figure 1. It was 4 cm in height, 6 cm wide, 20 cm long, and a uniform wall thickness of 0.5cm. The solid walls were made of copper as were the porous mesh inserts. The porous inserts had a fiber diameter of 1 mm and a porosity of 40%. Water was used as the coolant, entering at a temperature of 25°C and a pressure of 80 bar. To ensure laminar flow, the water was given a Reynolds number of 600, resulting in a core velocity of 0.602 m/s. Water was modeled as an incompressible fluid and was assumed to have fully developed flow at the entrance region. The channel's walls were assumed to be subject to a constant surface temperature boundary condition of 500°C.

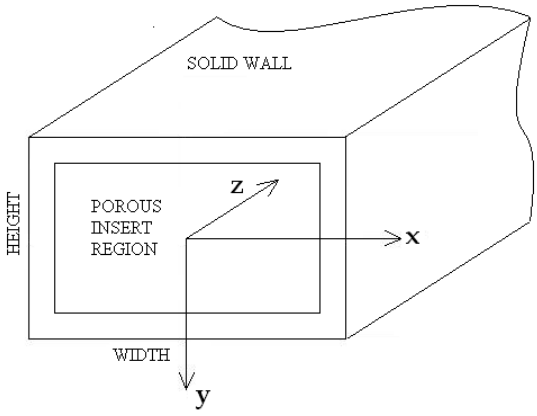

FIGURE 1
SCHEMATIC OF THE RECTANGULAR CHANNEL

Because of the incompressible fluid assumption, the momentum and energy equations in the porous channel domain were solved independently from each other. The momentum equations used was based on the Darcy–Forchhiemer–Brinkman formulation [1] and is presented in vector form as follows:

$$\nabla p = -\frac{\mu}{k}\vec{v} - \frac{c_f \rho |\vec{v}|}{\sqrt{k}}\vec{v} + \mu \nabla^2 \vec{v} \tag{1}$$

which takes the following scalar form with the assumption of fully developed flow and packed bed type porous medium for the porous matrix:

$$\frac{dp}{dz} = -\frac{150(1-\varepsilon)^2 \mu}{{d_p}^2 \varepsilon^3} u - \frac{1.75(1-\varepsilon)\rho}{d_p \varepsilon^3} u^2 + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2}$$

The energy equation in the porous domain which includes an advection term is given by

$$\left(\varepsilon \rho_f c_{p,f} + (1-\varepsilon)\rho_s c_{p,s}\right)\frac{\partial T}{\partial t} + \rho_f c_{p,f} u \frac{\partial T}{\partial z} = k_{eff} \nabla^2 T \tag{3}$$

The conjugate heat transfer in the surrounding solid wall is conduction that is governed by the heat diffusion equation.

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \nabla^2 T \tag{4}$$

Heat transfer at the interface between the porous insert region and the solid wall was determined by the continuity of heat fluxes using Fourier's law of conduction.

$$q_n^{"} = -k\frac{\partial T}{\partial n} \tag{5}$$

NUMERICAL METHOD

Finite difference methods were used to obtain solutions to the momentum and heat transfer equations. Due to the assumption of incompressible fluid, the momentum equation was solved independently from the energy equation of the porous domain. Moreover, for simpler implementation of the finite difference solution, explicit finite difference schemes were used. This is why we kept the transient term in the governing energy equations even though the solution we sought was the steady-state solution.

With the anticipation of high CPU time, the numerical model developed used two approaches to reduce computation time. First, the number of nodes required to be solved was reduced by identifying the symmetry conditions in the problem. This reduced the actual numerical domain to one quadrant of the rectangular channel and the surrounding wall due to symmetry about the *x* and *y* axes. The results could then be appropriately mirrored to produce nodal velocity values for the entire channel. The same concept was also employed for the channel's nodal temperatures. Figure 2 shows this representation and the *(m,n)* matrix used in laying out the discretized equations.

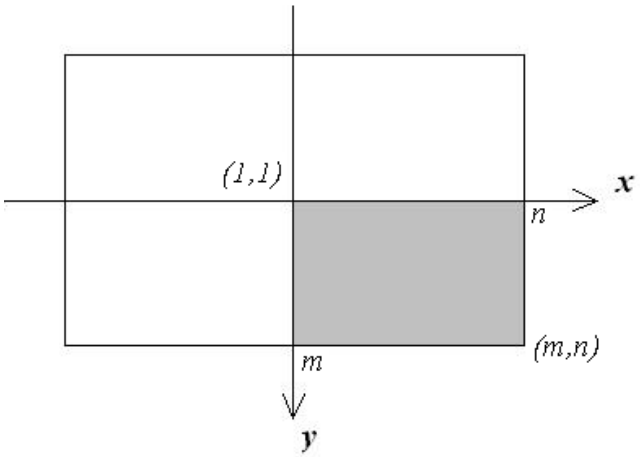

FIGURE 2
SIMPLIFIED REPRESENTATION OF FLOW AREA BEING SOLVED

The second technique used for reducing the CPU time was by using variable grid size. In order to capture the effects of the no-slip boundary condition at the walls and their vicinity and to relax the limitation on the grid size toward the core region of the porous channel, where there is much smaller velocity gradients, we decided to use the so called "compound interest" type grid sizing scheme, i.e., the size of the grids in the porous region was successively reduced moving from the center of the channel towards the walls. This allowed for a concentration of nodes near the walls where fluid velocity and temperature gradients would be significantly higher. Larger grid sizes would then be found in the middle of the channel where fluid velocity and temperature would vary little, if at all. CPU time was therefore reduced by systematically increasing the number of nodes only where they were needed.

The momentum equation seen in Eqn. (2) is a second order non-linear PDE, whose non-linearity is due to the inertial Forchhiemer term. In order to overcome this difficulty we used the Newton method of linearization [2] to solve Eqn. (2) numerically in an iterative procedure until a specified convergence criterion between two successive iterations was met. Using this method, Eqn. (2) was discretized by writing second order accurate finite difference approximations for all second derivative terms. These were derived by Taylor's expansion method. Central differencing was used for all nodes, including those on the axes of symmetry. This method resulted in a system of linear algebraic equations in terms of the fluid's velocity, *u*. Each iteration required matrix inversion to solve the linearized equations following the common form of $A\vec{x} = B$.

The energy equations for the two domains were disretized in similar manner using mostly second order accurate central difference approximations for derivative terms. One particular exception to this rule was the treatment of the advection term, $u(\partial T/\partial z)$, in which

upwind (backward) differencing was used from our experience [3] in order to avoid instabilities in the numerical scheme caused by the use of central differencing. The second derivative terms in Eqn. (3) were written using second order accurate central difference representations in all three dimensions. Interface nodes between the porous and solid regions used a modified version of Eqn. (5) in the *x* and *y* directions only. It was necessary to develop second order accurate finite difference approximations for the first derivative terms that appeared in the boundary condition at the interface due to Fourier's law of heat conduction. Lastly, the heat diffusion equation (4) was written for the solid domain in three dimensions with central differencing having second order accuracy. For the spatial derivative terms in the energy equation for the nodes on the exit plane it was necessary to use backward differencing. As mentioned previously, since an explicit finite difference scheme was used in the energy equations, the temporal derivative term was approximated by forward differencing.

IMPLEMENTATION OF NUMERICAL METHODS USING MATLAB

The approach to programming this particular numerical model in MATLAB was completed in two phases. First, an independently functioning and debugged numerical solution to the momentum equation was developed. This was followed by the numerical solution of the energy equations. The finite difference equations used in this study were initially organized and written out by hand before implementing any code in MATLAB. This was done by using generic variable declarations and generic "for" loops.

Due to the two dimensional nature of the velocity profile in the problem, the system did not yield a tri-diagonal system of linear equations often found in one dimensional cases. Instead a special penta-diagonal system of linear equations was developed, resulting in a very large coefficient matrix. The known values of a typical $A\vec{x} = B$ set of equations, found on the right hand side, was transformed from a two dimensional array to a single column vector. To take advantage of matrix inversion in MATLAB, the pseudo two-dimensional problem was transformed into a one-dimensional problem.

Setting up the numerical solution for the momentum equation in MATLAB allowed for an almost direct translation from the hand written discretized equations. All of the vectors and arrays were then populated using "for" and "while" loops. Solving the momentum equation for the velocity distribution required matrix inversion and multiplication. These tasks were performed very easily in MATLAB by the use of its built-in matrix inversion function followed by matrix multiplication. Regardless of which programming environment one chooses to numerically solve the system of equations that result from Eqn. (2), one must declare all necessary variables, program iterative subroutines, and populate the necessary arrays. However, solving the momentum equation using MATLAB requires less effort as the user does not have to develop subroutines for matrix inversion or other methods of solving a system of linear equations. This was done by simply using the built-in functions of MATLAB.

The second part of the study involved solving the necessary energy equations for each of the problem's two domains; porous and solid. This did not require any special matrix operations, but rather simple algebraic computations at each node to produce a new value as the solution marched in time with each successive time step. MATLAB was also used for this segment of the study even though it did not provide an advantage over other 3GL languages.

The complete numerical solution for the channel discussed used 60x80x40 nodes for the overall width, height, and length, respectively. The program, however, required 30x40x40 nodes due to the previously discussed symmetry in the width and height directions of the channel. Using the stated number of nodes and a time step of $5x10^{-4}$ seconds, CPU computation time was about 30 minutes.

POST PROCESSING AND GRAPHICAL RESULTS

Once a successful numerical simulation was complete, the results were ready for direct graphical analysis and presentation in MATLAB. Two things are of interest post simulation; the shape of the velocity profile and the temperature distribution throughout the channel and the surrounding wall so that heat transfer analysis can be performed.

In the first part of this study, the momentum equation was solved numerically, producing an array of the cooling fluid's velocity field throughout the channel. Figure 3 shows how the velocity of the fluid varies from zero at the wall (due to the no-slip boundary condition) to the channel's center velocity of 0.602 m/s chosen for the case study. Notice the quick transition from zero to centerline velocity, which occurs within approximately 0.3 mm from the wall. While the channel's complete 2-D profile or 3-D surface of the fluid's velocity could have been generated, these would offer little useful information due to the very small distance in which the velocity profile showed any transitioning. As a result, a complete plot would make the details along the channel's walls unrecognizable.

Once the overall numerical solution was complete, a three dimensional array of nodal temperatures for the entire channel was produced. MATLAB offers a variety of simple yet powerful commands to analyze and present these results. One way would be to use MATLAB's command *contour( )*, which could be used to plot a 2-D cross-section of the channel's temperature variation. Figure 4 depicts an example of such a contour plot. Recall that the walls of the channel are 0.5 cm thick (10 nodes used in this domain). From Figure 4 the increase in temperature along the channel's length in the direction of the fluid flow can be seen. A 3-D surface plot of Figure 4 can also be produced to show the same results, but in a different manner.

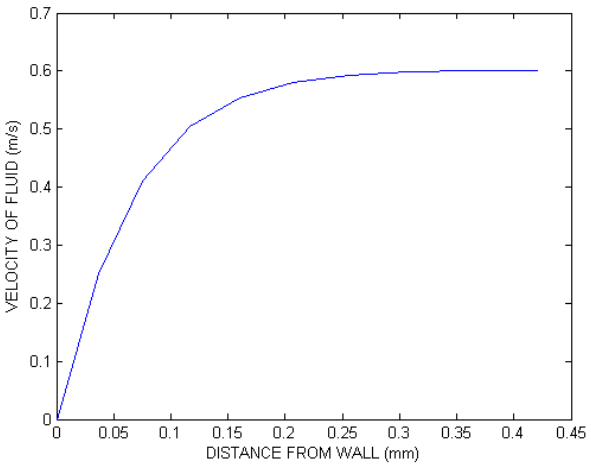

FIGURE 3
VELOCITY PROFILE OF COOLING FLUID AT WALL

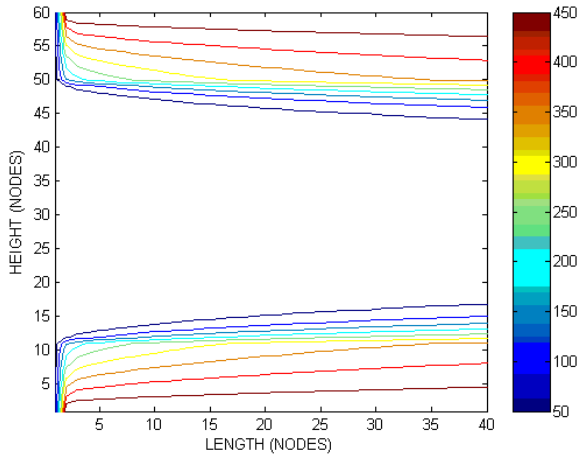

FIGURE 4
CROSS SECTION TEMPERATURE DISTRIBUTION ALONG CHANNEL LENGTH

By analyzing all of the exit nodes at the channel's end, one can gain an excellent understanding of how temperature varies between the solid and porous regions. A 3-D surface plot using MATLAB's command *surf( )* can be used to show such a temperature distribution, as shown in Figure 5. Looking at the edges of the rectangular shape the 500°C constant surface boundary condition can be noted. The remaining part of the solid region remains in the range of 500°C to 300°C, as temperatures decrease towards the inside of the channel where the flowing coolant is located and absorbs the heat transferred across the interface in a very efficient manner. Inside the porous region the temperature quickly drops down to 25°C, the value of the constant inlet temperature of the cooling fluid. The cooling fluid, water, remains in liquid phase near the wall's high temperatures due to a substantial inlet pressure of 80 bar placed on the flowing fluid.

From both Figures 4 and 5, the effectiveness of the porous cooling channels can be seen by observing how readily the cooling fluid removes heat from the high temperature region in the thermally conductive solid walls.

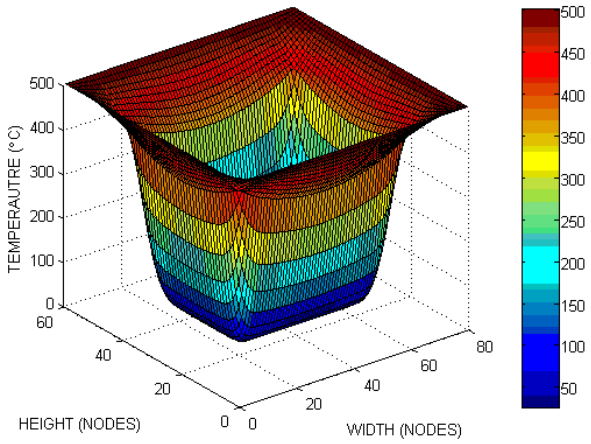

FIGURE 5
NODAL TEMPERATURE DISTRIBUTION OF WALL AND COOLING FLUID AT EXIT OF CHANNEL

DISCUSSION ON USING MATLAB TO SOLVE THIS NUMERICAL SYSTEM

The benefit to using MATLAB was quite evident when numerically solving this problem's particular momentum equation (2), which required matrix operations. In lower level programming environments such as C or FORTRAN, one would need to program subroutines that would multiply matrices or invert square matrices. Programming a subroutine to perform matrix inversion would be time consuming, require adequate linear algebra knowledge, require debugging, and or using external codes developed by a third party. MATLAB eliminates this extra programming by including useful tools that automatically perform tasks the user would otherwise have to program. The benefit in this case is that someone can perform a numerical simulation quickly, without having to know the necessary linear algebra or writing subprograms for mathematical operations required. The end result is a quicker and less prone to errors.

The second phase of the problem again used finite difference methods to solve the energy equations. These equations did not contain any advance matrix operations but rather simple algebraic calculations for each node of the three dimensional arrays. MATLAB or a 3GL programming language such as C or FORTRAN would have handled this portion of the problem as well. This portion of the simulation was accomplished with many nested "for" loops, something for which MATLAB is notoriously slow. Simulating the heat transfer portion of the study required tens of thousands of time steps before nodal temperatures converged to a steady state solution. In the form of the code used in this simulation with the many nested "for" loops, MATLAB has been cited to take longer than C or FORTRAN by at least 3 to 10 times. This is significant

when CPU time for this simulation varied dramatically depending on which parameters in the simulation were changed. For example, changing the value of the time step from $5x10^{-4}$ second to $5x10^{-5}$ second and increasing the number of nodes from 30×40 to 40×60 in one quadrant of the channel increased simulation time from 30 minutes to 14 hours. Therefore a reduction in computation time even by a factor of 3 would significantly reduce these large computation times. One way however around this "for loop" short fall of MATLAB is to vectorize as many arrays as possible – something that would require significant programming knowledge and experience.

In post-processing the results, MATLAB proved to be a powerful tool in producing 2-D and 3-D plots with little additional code or effort. The final 3-D temperature array contained hundreds of thousands of nodal temperatures, and yet it was straight forward in producing graphics that were meaningful. Other lower level programming languages do not have graphic capabilities, requiring that each array be exported into some other software, and then manipulated. One must then learn that new software as well and transfer over any arrays or values needed. This process of course is time consuming and more likely to introduce mistakes. Most importantly, many parameters of the simulation can be varied to study response. Altering grid size, channel size, channel material, boundary conditions, fluid flow velocity, etc., are all necessary to have a better understanding of how the system behaves. Being able to immediately and automatically graph results in MATLAB is extremely helpful and time effective.

CONCLUSION

Fluid flow and heat transfer in a rectangular porous cooling channel and the conjugate heat transfer in the surrounding wall were successfully simulated by the use of finite difference method. The solution was implemented in MATLAB. The results obtained from several case studies provided a clear understanding of the behavior of the investigated system by establishing the velocity field in the porous domain and the temperature distribution in the two domains. Superior heat transfer enhancement characteristics of porous cooling channel were observed.

Some advantages and disadvantages of using MATLAB for implementing the solution to the problem considered in the current work were established. It should be noted that developing the MATLAB programs for implementing the finite difference solution to the problem was carried out by a senior undergraduate engineering student, the first author of this paper, through an internal research grant at our institution. The student was provided with the mathematical model. He was guided through the course of the project as to how to implement the solution by finite difference method. This included providing him information about such aspects as explicit and implicit schemes, first and second order accurate finite difference approximations, Newton's method of linearization, upwind differencing, etc. Writing the MATLAB programs for solving the problem and post-processing the results obtained from the simulations was his responsibility, although suggestions on what to present in which way for easy interpretation of the results were provided.

It is clearly evident that the use of MATLAB in solving the governing momentum equation offered significant advantages over the use of 3GLs such as FORTRAN or C. The extensive matrix operations were easily handled by the built-in functions. These operations would have required development of subroutines or use of third party developed subroutines to perform these operations. That would require significant program development/debugging/integration efforts. Due to the choice of explicit finite difference schemes used in solving the energy equations, there were no matrix operations needed. In this case there was no particular advantage in using MATLAB over FORTRAN or C from a purely programming effort point of view. From a CPU time consideration it might have even exacted a penalty as our readings indicate that multiple nested loops are not executed in a time-efficient manner in MATLAB as compared to 3GLs. It should be kept in mind though that explicit finite difference method was chosen for its simplicity in implementation in any program. Had we used implicit finite difference schemes for solving the energy equations, the solution of the resulting system of linear algebraic equations would have required matrix operations. In that case using MATLAB, just as in the case of the momentum equation, would have offered the same compelling advantages over C or FORTRAN. Another advantage of using MATLAB would be in the phase of post-processing of the results from the numerical solutions obtained. Unlike 3GLs MATLAB has graphic capabilities, which makes presentation of data in the form of *xy* plots or 2-D or 3-D plots as easy as calling a built-in function such as *surf*( ). This is another significant advantage over using 3GLs in which case all the data would need to be transferred to third party post-processing software for manipulation.

This study showed that by the use of MATLAB, what would require considerable mathematical background and programming skills, and possibly graduate assistance, can be accomplished by using undergraduate assistance. This would also serve in engaging undergraduate students in research, which would be a highly desirable activity especially in predominantly undergraduate programs.

FUTURE WORK

The model and the program developed can be enhanced and modified to be used for investigating high heat flux applications that can be encountered in multiple areas of interest. Our future plans include conducting a comprehensive set of case studies for selected pairs of coolants and porous matrix materials for different thermal boundary conditions. For example, supercritical hydrogen and silicon carbide ceramic matrix composite material for transpiration cooling of liquid rocket propulsion engines.

Other classical boundary conditions such as constant heat flux and/or convective boundary conditions will be implemented. Generalized performance measures for the effectiveness of the heat transfer such as the variation of the local Nusselt number along the interface between the two domains will be determined in the program.

NOMENCLATURE

c_f = drag coefficient factor
c_p = specific heat (J/kg°C)
d_p = fiber diameter (m)
k = permeability of porous medium (m^2)
k_{eff} = effective thermal conductivity (W/m°C)
p = pressure (N/m^2)
T = temperature (°C)
t = time (s)
u = velocity component in z-direction (m/s)
$\vec{v}$ = velocity vector (m/s)

Greek symbols

α = thermal diffusivity (m^2/s)
ε = porosity (also known as void fraction)
μ = dynamic viscosity of fluid (kg/ms)
ρ = density of fluid (kg/m^3)

Subscripts

f = subscript for fluid phase substances
s = subscript for solid phase substances

ACKNOWLEDGMENT

The authors would like to acknowledge the R&D grant from the Center for Scholarly and Creative Excellence at Grand Valley State University in support of this work.

REFERENCES

[1] Kaviany, M., and Ling, F.F., *Principles of Heat Transfer in Porous Media, 2nd Edition*, Mechanical Engineering Series, Springer, 1999

[2] Nakamura, S., *Numerical Analysis and Graphic Visualization with MATLAB*, Prentice Hall, 2001

[3] Sözen, M., and Vafai, K., "Analysis of Oscillating Compressible Flow through a Packed Bed", *International Journal of Heat and Fluid Flow*, Vol. 12(2), pp. 130-136, 1991

AUTHOR INFORMATION

James Cherry is a 2009 graduate of Grand Valley State University, who completed a Bachelor of Science degree in Mechanical Engineering, cherryj1@gmail.com.

Mehmet Sözen is an Associate Professor of Engineering at the School of Engineering, Grand Valley State University, Grand Rapids, MI, 49504, sozenm@gvsu.edu.