Math Background for ROM Project

Angel Sarmiento - github.com/angel-sarmiento

12/5/2020

Note: this is just some of the math as an excerpt of a larger report. If you are looking for the rest of the report, it is here: github.com/angel-sarmiento/ROM-Projects/tree/master/1D-Heat_equation

The 1-Dimensional Heat/Diffusion Equation

The 1-Dimensional Heat/Diffusion Equation is defined here as:

$$\frac{\partial u}{\partial t} = \mu \frac{\partial^2 u}{\partial x^2}$$
$$u_t = \mu * u_{xx}$$

where μ is known as the *diffusivity* of the medium. The boundary conditions (Dirichlet and Neumann) are as defined below:

$$u_t = \mu * u_{xx} \qquad \text{for } 0 <= x <= 1 \tag{1}$$
$$u(0,t) = 1 \tag{2}$$
$$u_x(1,t) = 0 \tag{3}$$
$$u(x,0) = cos(3 * \pi * x/2) \tag{4}$$

Notice that this is a time-dependent Partial Differential Equation (PDE). In order to solve this, we implement a Backward in Time, Central in Space (BTCS) scheme for discretization. BTCS is chosen here since it is an implicit method without limitations on solutions like FTCS for stability. It also involves a smaller number of computations than the *Crank-Nicholson scheme*, making it simpler to program. For this investigation, importance is placed on the comparative advantage between two models, not on the accuracy of the numerical solution of the heat equation. The discretization is shown below:

$$\frac{u_{i,n} - u_{i,n-1}}{\Delta t} = \alpha \frac{u_{i-1,n} - 2u_{i,n} + u_{i+1,n}}{(\Delta x)^2}$$

where $u_{i,n} \approx u(x_i, t_n)$. Simplified, this yields,

$$u_{i,n} = (1 + 2\lambda)u_{i,n+1} - \lambda(u_{i+1,n+1} + u_{i-1,n+1})$$

for $i = 1, 2, ..., M-1$ and for $n = 1, 2, ..., N$, Solving for the system each time. As stated above, this method is *unconditionally stable* and convenient for the purposes listed here. The code used to implement this is available here

POD, SVD, and Snapshot Matrices

The Proper Orthogonal Decomposition (POD) is a method used to solve lower-order versions of reference models that is intended to be faster than solving the reference model fully each time. It is a method of projection onto subspaces where the basis consists of dimensions that are orthogonal to one another. The reference model in this case is described above as the 1-D heat equation. Solving the reference model initially yields snapshots for each *state* of the model, given in this case by u solved at each time t or $u(t_1), u(t_2)$.... These state vectors are collected and arranged as a snapshot matrix for which the *Singular Value Decomposition* (SVD), $X = U\Sigma V^T$ is applied to create a reduced basis.

The new basis state vectors are denoted by $x(t) \approx \Psi^x \phi^x(t)$ where Ψ^x is the reduced-order basis vectors and $\phi^x(t)$ is the coefficient vector of the same basis. Selecting the number of basis dimensions is vital to maintaining a larger proportion of variance of the full-order model (FOM). A lower number of dimensions in the reduced-basis state vectors creates a lower order differential algebraic system solution, which can be faster than solving the FOM in instances where the FOM is more complicated than the one described above. Projecting this reduced basis back to the full-dimensional representation involves performing a method like Galerkin's:

$$\Psi^T B \Psi \frac{d\phi}{dt}(t) = \Psi^T A \Psi \phi(t) + \Psi^T g(\Psi \phi(t))$$

or by another method, eventually solving a system of equations using the formula $Ax = b$ where A is a sparse, triangular matrix. The resulting triangular matrix is a matrix with $(1 + 2\lambda)$ in the center diagonal and $-\lambda$ in the outer diagonals (see above). Also where b is $u_{i-1,n-1}$. This can then be solved using LU Factorization for faster computation or by just solving $Ax = b$ in any solver. The Python code for this is implementation without Galerkin projection or LU Factorization is available here where this is implemented with $t = 1$ second.