Quiz-2 Review

November 13, 2006

- ***Singular Value Decomposition: SVD***

  Any rectangular $m \times n$ matrix A can be decomposed into a product of 3 matrices as shown in Equation 1, where $\Sigma_{m\times n}$ is a diagonal matrix containing singular values σ_i of matrix A such that $\sigma_i = \sqrt{\lambda_i}$, where λ_i are the eigenvalues of matrix AA^T. U and V are orthonormal matrices i.e. $V^T = V^{-1}$, $U^T = U^{-1}$. The columns of U and V are called left singular and right singular vectors of A respectively. This decomposition is called Singular Value Decomposition (SVD) of matrix A.

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n} \tag{1}$$

  . In matlab SVD of any matrix A can be performed using the command `svd`.

  The condition number of matrix is defined as $cond(A) = \sigma_{max}/\sigma_{min}$. SVD can be used to decouple a noisy signal into useful component and noise. Another useful application of SVD is in the solution of linear system of equations. The solution of a system of equation $A \cdot x = b$ is given in Equation 2, where we replace $\frac{1}{\sigma_i}$ with 0 if $\sigma_i \approx 0$. If there are more equations than unknowns then the equation gives a least squares solution of the overdetermined set of linear equations.

$$x = V \cdot \begin{pmatrix} \frac{1}{\sigma_1} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2} & \cdots & 0 & \cdots & 0 \\ \vdots & & & & & \\ 0 & 0 & \cdots & \frac{1}{\sigma_n} & \cdots & 0 \end{pmatrix} \cdot U^T \cdot b \tag{2}$$

- ***Ordinary Differential Equations: ODE and Differential Algebraic Equations: DAE***

  The general form of ODE is given in Equation 3, where $\underline{X}$ is a vector of length n and $\underline{F}$ is a set of n functions.

$$\frac{d\underline{X}}{dt} = \underline{F}\left(\underline{X}\right) \tag{3}$$

  If we have an n^{th} order ordinary differential equation, it can be converted into n ordinary differential equations. To do that we define $n-1$ new variables y_i's, where each $y_i = \frac{d^i y}{dt^i}$. To solve a set of ODEs we require the boundary conditions to be specified at $t = t_0$. If all the Boundary conditions are not specified at at the same boundary then we can either use shooting method or solve the system as a boundary value problem.

  The solution of Equation 3 can be obtained by performing numerical integration as shown in Equation 4.

$$\underline{X}(t) = \underline{X}(t_0) + \int_{t_0}^{t} \underline{F}\left(\underline{X}(t')\right) dt' \tag{4}$$

  If we are given the value of function $\underline{F}\left(\underline{X}(t')\right)$ at discrete points t_1, $t_2 ... t_3$, we can use Trapezoid rule or Simpson's rule to integrate the function. The local errors (error incurred per step) of Trapezoid rule and Simpson's rule are $O(\Delta t^3)$ and $O(\Delta t^5)$ respectively. We can increase the accuracy of the integration by using a more accurate method or decreasing Δt. Using too small of a time step leads to rounding off errors. To overcome this problem the concept of Richardson Extrapolation is used. When applied to trapezoid rule we get the result shown in Equation 5. Notice that the error properties of the result are the same as that of Simpson's rule.

$$\underline{X}_{true}(t_f) = \frac{4}{3}\underline{X}(t_f, \frac{\Delta t}{2}) - \frac{1}{3}\underline{X}(t_f, \Delta t) + O(\Delta t^4) \tag{5}$$

  ODEs can be solved using explicit methods. Euler's explicit method is the simplest explicit method but is inaccurate and unstable. More accurate methods include Runge-Kutta methods of variable orders. These methods require more number of functional evaluations per step but also have higher order accuracy. Adaptive time stepping can be implemented by using the Runge-Kutta methods. For example Runge-Kutta

method of fifth order require six function evaluations and the same six function evaluations can be used to get a fourth order method. The step size is varied until the difference between the fifth and fourth order Runge-Kutta methods are obtained to within some desired level of accuracy.

The explicit methods don't perform well when the system of differential equations is stiff. A system is said to be stiff when there are two or more very different time scales involved in the problem. Chemical kinetics problems are very often stiff. The differential equation has to take time steps which correspond to the fastest time scale (small Δt) but the total time that it has to integrate to is very large because of the presence of the slow time scale. Thus the total number of time steps that the method has to take is high. If an explicit method tries to take time steps larger than the one corresponding to the fastest time scale then it becomes unstable. For example for a set of linear differential equations given in Equation 6, the time step of explicit euler method $\Delta t < 2/\lambda_{max}$, where λ_{max} is the largest eigenvalue (fastest time scale) of matrix C. On the other hand the implicit euler method is always stable for this system.

$$\underline{X}' = -\underline{\underline{C}} \cdot \underline{X} \tag{6}$$

Higher order implicit method generally implement multi-step predictor-corrector type methods. In these methods we use the solutions at previous k steps ($\underline{X}_n$, $\underline{X}_{n-1}$, ..., $\underline{X}_{n-k+1}$) to estimate a solution at the n+1th step. This initial guess is used to calculate an exact value of $\underline{X}_{n+1}$ using a newton's type iterative method. Polynomials relating $\underline{X}_{n+1}$ to the previous values of $\underline{X}$ and their derivatives developed by Gear ensure stability for even stiff differential equations.

Differential Algebraic equations are of the general form

$$M(\underline{X})\dot{\underline{X}} = \underline{F}(\underline{X})$$

where $M(\underline{X})$ is called the mass matrix and is singular. A important concept related to DAE is its index number, which is defined as the number of differentiations that is required to convert it into an ODE. Index-1 differential equations can be solved using techniques similar to the ones developed for ODEs. `ode23s` and `ode23t` can solve problems with constant singular mass matrices. `ode15i` solves DAEs which are fully implicit $\underline{F}\left(\underline{X}, \dot{\underline{X}}, t\right) = 0$.

- ***Numerical Optimizations***

  Optmization problems can in general be posed as follows

$$\begin{aligned} \min_{\underline{x}} f(\underline{x}) \qquad & \text{s.t.} \\ h_i(\underline{x}) \geq 0 \qquad & i = 1 \cdots n_i \\ g_i(\underline{x}) = 0 \qquad & i = 1 \cdots n_e \end{aligned} \tag{7}$$

  If the inequality and equality constraints are not present then the problem is called unconstrained optimization problem. If gradient of the method is not provided then the problem can be solved using simplex method. This method is usually much slower than the gradient methods but for some problems is very robust. Matlab function `fminsearch` implements this algorithm. The gradient of the cost fuction is defined in Equation 8 and the negative of the gradient is the direction of steepest descent.

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{8}$$

  method of steepest descent makes use of this fact and moves sequqntially in the direction of the negative of the gradient. The size of the step depends on exact implementation of the algorithm. The steepest descent method makes fast progress initially but convergence of the method is slow when it reaches the bottom of the valley. In conjugate gradient method the step direction is not exactly the negative of the gradient but also has some element from previous search directions. This prevents the zig-zag trajectory which is a problem in the steepest descent method.

  Newtons method uses of the Hessian matrix which is provided in Equation 9.

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} \tag{9}$$

  The newton step can be written as

$$\underline{x}^{k+1} - \underline{x}^k = \Delta \underline{x}^k = -H^{-1}(\underline{x}^k) \cdot \nabla f(x^k)$$

The calculation of Hessian matrix is usually costly and in its place a approximate Hessian is used. A commonly used approximation to the Hessian matrix is BFGS (Broyden-Fletcher-Goldfarb-Shanno) update.

Method of Lagrangian multipliers is used for constrained minimizations with equality constraints. The lagrangian function L is defined as

$$L(\underline{x}, \underline{\lambda}) = f(\underline{x}) - \sum_{i}^{n_e} \lambda_i g_i(\underline{x})$$

At the constrained minimum the condition $\nabla f - \sum \lambda_i \nabla g_i = \nabla_{\underline{x}} L = 0$ and $\underline{g}(x) = \nabla_{\underline{\lambda}} L = 0$ is satisfied. Thus the problem of constrained minimization involves minimizing the lagrangian with respect to both $\underline{x}$ and $\underline{\lambda}$. Augmented lagrange methods add a quadratic penalty for straying away from the equality constraints. The augmented lagrangian is defined as

$$L(\underline{x}, \underline{\lambda}, \underline{\mu}) = f(\underline{x}) - \sum_{i}^{n_e} \left(\lambda_i g_i(\underline{x}) - \frac{1}{2\mu_i} [g_i(\underline{x})]^2 \right)$$

When inequality constraints are present then we can define a new lagrangian which includes these inequality constraints. This lagrangian should satisfy the Karush-Kuhn-Tucker (KKT) conditions given in Equation 10. These conditions imply that, for active constraints $h(\underline{x}_{min}) = 0$ and for inactive constraints $\kappa_j = 0$.

$$L(\underline{x}, \underline{\lambda}, \underline{\kappa}) = f(\underline{x}) - \sum_{i=1}^{n_e} \lambda_i g_i(\underline{x}) - \sum_{i=1}^{n_i} \kappa_i h_i(\underline{x})$$

$$\begin{aligned} \nabla L(\underline{x}_{min}) &= 0 \\ g_i(\underline{x}_{min}) &= 0 \\ h_j(\underline{x}_{min}) &\geq 0 \\ \kappa_j &\geq 0 \\ \kappa_j h_j(\underline{x}_{min}) &= 0 \end{aligned} \tag{10}$$

In Sequential Quadratic Programming we use the concept of slack variables. Each equality and inequality constraint is brough to a common ground by adding a slack variable to them.

- ***Boundary Value Problem***

  Very often we need to solve a conservation equation for some scalar quantity (ϕ) at steady state. The conservation equation takes the form

$$\underbrace{-\underline{\nabla}.(\phi\underline{v})}_{convection} + \underbrace{c\nabla^2\phi}_{diffusion} + \underbrace{S(\phi)}_{generation} = 0$$

  For cartesian coordinates the above equation becomes

$$v_x\frac{\partial\phi}{\partial x} + v_y\frac{\partial\phi}{\partial y} + v_z\frac{\partial\phi}{\partial z} = D_i\left[\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2}\right] + S(\phi) \qquad (11)$$

  If the velocity field is known at each point, then this problem can be solved using a basis function expansion i.e. representing $\phi = \sum c_i\psi_i$ and then calculating the values of c_i that satisfy the above equation. The c_i are usually cleverly chosen so that they satisfy the boundary conditions. The other approach is the finite difference approach for the solution of the above equation which involves dividing the 3-D space of (x, y, z) into discrete coordinates (x_i, y_j, z_k) and trying to calculate the value of $\phi(x_i, y_j, z_k)$ at each of those points. The discrete form of the partial differential equation can be written by using the following approximations

$$\left.\frac{\partial\phi}{\partial x}\right|_{(x_i,y_j,z_k)} = \frac{\phi(x_i, y_j, z_k) - \phi(x_{i-1}, y_j, z_k)}{\Delta x}$$

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{(x_i,y_j,z_k)} = \frac{\phi(x_{i-1}, y_j, z_k) - 2\phi(x_i, y_j, z_k) + \phi(x_{i+1}, y_j, z_k)}{\Delta x^2}$$

  Similar approximations can be used to for y and z directions. Notice the firt order partial derivative is approximated using upwind difference formulation which in many cases is known to give stable results. Let us assume we divide x, y and z directions in l, m and n discrete points. The number of unknowns in this sytem will be $l \times m \times n$, but the discretized form of Equation 11 will provide us with only $(l-2) \times (m-2) \times (n-2)$ set of equations. The rest of the equations will be provided by the boundary conditions of the problem. The commonly found boundary conditions will be of two kinds, namely Dirichlet and Neumann. Dirclet boundary conditions are of the form

$$\phi(x, y, z)|_{B.C.} = \phi_{B.C}$$

and directly give the remaining equations to solve the system uniquely. In Neumann boundary conditions the derivatives at the boundary are given. When we discretize the derivative at the boundary condition we get the following equation

$$\frac{-3\phi(x_1, y, z) + 4\phi(x_2, y, z) - \phi(x_3, y, z)}{2\Delta x} = \left.\frac{d\phi}{dx}\right|_{x=x_1}$$

The system of equation that results can be solved using any of the non-linear equation solvers. If the generation term is linear then the system of equations becomes linear and can be solved exactly using the \ command.

Method of lines is a convenient method for solving these equations when we know that there is stiffness present in one direction, say the x direction. Then the equation can be discretized in the remaining directions and in the x direction it is solved using a stiff ODE solver like `ode23s` or even `ode45` which is capable of doing adaptive time stepping. The only problem with method of lines is that the boundary conditions have to be known at the $x = x_1$, if this is not true then a shooting method is used in the x direction.