Numerical Linear Algebra

Numerical linear algebra forms the backbone of computer graphics, as many problems in computational geometry and physics based simulation reduce to 1) solving one or more linear systems of equations and 2) finding the eigenvalues and eigenvectors of a matrix.

1) Solving $Ax = b$ by directly inverting A is an $O(n^3)$ operation that is prohibitively expensive for large matrices. To compound the problem, A is often singular (or close to singular) and in some cases not even square. However, as A is generally sparse and symmetric positive definite in graphics applications, it is possible to compute x in a computationally feasible manner using direct or iterative solvers.

Direct solvers for linear systems decompose A into triangular or orthogonal matrices. Such decompositions provide a numerically stable way to solve a system of linear equations and to evaluate multiple right hand sides efficiently. Though direct solvers theoretically give the exact solution in a finite number of steps, this is often not the case in applications because of rounding errors; an error made in one step propagates in all the following steps. Commonly used decompositions include:

1. **Cholesky -** Decomposes a symmetric positive definite matrix A into a product of a unique lower triangular matrix L and its transpose, $A = LL^T$.
2. **LU -** Decomposes a square matrix A into a product of a lower triangular matrix L and an upper triangular matrix U, $A = LU$. The decomposition is not unique.
3. **QR -** Decomposes a general m by n matrix A into a product of an orthonormal matrix Q ($Q^TQ = I$) and an upper triangular matrix R, $A = QR$.

In contrast to direct solvers, iterative methods create a series of approximations x^k to the exact solution x such that $\text{Lim}_{k->\infty} x^k = x$. In practice, the vector x^k is updated until $||Ax^k - b|| < \varepsilon$. The main advantage of such solvers is that they are self correcting and provide insight into the system even after a few iterations. Commonly used iterative solvers include:

1. **Jacobi -** Given a diagonally dominant or symmetric and positive definite matrix A, the iteration formula $x_{i+1} = -D^{-1}(L + U)\, x_i + D^{-1}b$ for the Jacobi method converges for any starting vector x_0 as long as the max eigenvalue of $D^{-1}(L + U) < 1$.
2. **Gauss Seidal -** Given a diagonally dominant or symmetric and positive definite matrix A, the iteration formula $x_{i+1} = -(L + D)^{-1}U\, x_i + (L + D)^{-1}b$ for the Gauss Seidal method converges for any starting vector x_0 as long as the max eigenvalue of $(L + D)^{-1}U < 1$. The Gauss Seidal method converges faster than the Jacobi method and is more memory efficient as it uses one storage vector whose elements are overwritten as they are computed.

3. **Successive Over Relaxation -** The iteration formula $x_{i+1} = (D + wL)^{-1}[(1-w)D - wU] x_i + w(D + wL)^{-1}b$ of the successive over relaxation method is a refinement of the Gauss Seidal method. Values of $w > 1$ are used to speed up convergence while values of $w < 1$ are used to establish convergence of diverging or overshooting processes.
4. **Steepest Descent -** Given a symmetric positive definite matrix A, the steepest descent method tries to find the global minimum x of the quadratic form $Q(x) = (x^TAx - x^Tb) / 2$ by creating a series of iterations $x_{i+1} = x_i + (r_i^Tr_i / r_i^TAr_i) r_i$ where $r_i = b - Ax_i$, the gradient of Q(x) at x_i. Even though by definition it reduces Q(x) at each step, the steepest descent method suffers from slow convergence.
5. **Conjugate Gradient -** Instead of using r_i as a search direction to minimize Q(x), the conjugate gradient method constructs a series of linearly independent vectors v_i such that $v_i^TAv_j = 0$ for $i \neq j$ and $x_{i+1} = x_i + (v_i^Tr_i / v_i^TAv_i) v_i$. It reaches the exact solution in at most n steps (n being the number of rows and columns of A) after creating n linearly independent vectors v_i.

2) Finding the eigenvalues and eigenvectors of a nonsingular square matrix A is an $O(n^3)$ operation. However, many graphics applications only require the first few (smallest or largest) eigenvalues and eigenvectors of large matrices. Common techniques to compute them include:

1. **Power Method -** Constructs two series c_i and v_i that converge to the largest eigenvalue λ_1 and the corresponding eigenvector x_1 of A respectively by iteratively computing Av_i from a starting vector v_0 not orthogonal to x_1. c_{i+1} equals the norm of the matrix product while v_{i+1} equals the normalized vector. The power method can be extended to search for other eigenvalues and eigenvectors by constructing a new matrix B from A that has the same eigenvalues and eigenvectors except λ_1 and x_1.This is done by finding eigenvector h_1 of A^T corresponding to the eigenvalue λ_1 and setting $B = A - \lambda_1 h_1 h_1^T$. The process is repeated to find the remaining eigenvalues and eigenvectors.
2. **Inverse Iteration -** Given an initial approximation λ^* to an eigenvalue, the inverse iteration method finds the corresponding approximate eigenvector of the matrix A using the iteration $x_{i+1} = (A - \lambda^*I)^{-1}x_i$. The better the approximation λ^* to an eigenvalue, the faster the convergence. The inverse iteration method often complements methods for complete eigenanalysis.