SUMMARY OF METHODS

General:

* Frequency domain methods (FDFD,MOM,FE)
* Time domain methods (FDTD)

* Frequency domain methods can be used for time domain analysis. Write a time-domain equation as a function of the nodal unknowns. Use the Freq. domain method to solve to the nodal unknowns at each time step (iteratively), with boundary conditions changing as a function of time step. This takes a lot of computer time, so is rarely done, except with FE-TD, which is gaining popularity.

* Time domain methods can be used for frequency domain analysis. Run a time-domain simulation to steady-state, and convert to the frequency domain using signal processing techniques. Fourier transform (DFT or FFT) is most commonly used. Desampling generally improves efficiency significantly. Several methods have been developed to improve efficiency for specific applications. See below.

FDFD:

* Start with frequency domain differential equation and boundary conditions, and discretize the problem into N nodes.
  Apply difference formula to convert differential equation to difference equation at each node. This gives you N equations and N unknowns.

* Obtain a sparse and banded matrix equation.

* Alternately (much more efficient storage) don't use a matrix equation; solve iteratively using SOR.

* Storage requirement: (SOR method) N nodal unknowns
  (matrix meth) NxN matrix
  could reduce with sparse storage methods

* cputime: roughly proportional to N
  more iterations required as N increases

* Accuracy:
  first order (5-point difference) equal-sized arms: h^2
  second order (9-point difference) any sized arms: h^2
  third order (9-point difference) any sized arms: h^3

* Good for closed-boundary problems. Open-region problems can be done, but boundary conditions are often a problem.

* Problems: May be slow to converge.

FDTD:

* Start with time domain differential equation, and initial conditions. Apply difference formula to convert to difference equation. Iterate to steady-state.

* Solved iteratively, no matrix.

* Storage: N nodal unknowns

* Cputime: roughly proportional to N
  more iterations generally required as N increases

* Accuracy:
  first order (square cells): h^2
  second order (square or rectangular cells): h_2
  (requires storage of n time step)
  (requires storage of n, n-1 time step)
  third order (sq. or rect. cells): h_3
  (requires storage or n, n-1, n-2 time steps)

* Good for open or closed regions. Open regions require nodes to be place throughout open region.

* Ideal for heterogeneous objects up to 10λ in size. This is the only method used to date to handle frequency-dispersive materials over a broad frequency band.

* Problems: (and some solutions)
May be slow to converge for high-Q objects. Signal Processing methods based on adaptive filters can be used to predict the steady-state without having to converge completely.

Error propagates in time. Numerical dispersion can cause errors, particularly when phase of interfering waves is important. Cell sizes small enough to reduce this can cause prohibitively large models.

Models are generally programmed with stair-stepped approximation. Contour modeling and subgridding can be used, but sometimes causes problems with stability.

Low-frequency modeling is difficult, because of large# of time steps per cycle. Partial cycles can be used, with steady-state values calculated from last few time steps.

FDTD must be rerun for every different excitation. If the broad-band response or response to several pulse-shapes are of interest, a broad-band "impulse" can be used to obtain the "impulse response", and then convolution can be used to obtain the response to any other desired excitation.

MOM:

* Choose basis function to interpolate unknown:

* Choose weight function to interpolate forcing function:

* Do integrals analytically or numerically to obtain a matrix equation (not sparse, and generally not symmetric). Numerical integration should be more accurate than the basis and weighting functions. Gaussian Quadrature method is commonly used.

* Solve for the unknowns using conventional methods or "conjugate gradient method", to speed matrix solution.

* Storage: N^2 matrix elements

* Cputime: conventional methods proportional to N^2
  conjugate gradient method proportional to NlogN
  Huge cputime may be required to calculate integrals to obtain matrix elements, particularly if g(r,r') is a complicated function.
  Additional large cputime is required to solve matrix eqn.

* Suitable for small to medium sized problems. Helped somewhat because nodes are required only where unknowns are located. Ideal for wire problems, medium to large irregularly-shaped metallic scatterers. (For large regularly-shaped scatterers, consider GTD or modal methods.)

* Accuracy: depends on basis and weight functions.
  Point matching is often "suspect" because of discontinuity of derivatives.

* Formulation of equation (particularly g(r,r')) can include the effects of nearby large physical properties, such as layer half-spaces, ground planes, or free space. Using this method, no extra nodes are required in these regions (unlike FDTD). g(r,r') may be difficult to calculate, and generally requires numerical integration.

* Self-terms must be calculated with great care, as they are usually the largest elements in the matrix.
* Large metallic or coated metallic objects can be modeled using surface patches, so no nodes are required on inside of object.
* Rotationally-symmetric objects are handled with a "body of revolution" method, where only one line of the object needs to be considered, and the problem is developed in cylindrical coordinates. (This BOR could be applied to other methods, but generally isn't.)

FE:

* Choose element and interpolating function

* Find shape functions, elemental C^e, gobal C, minimize energy, apply BCs to solve.

* Storage: N^2 matrix elements (sparse matrix, can be reduced)

* Cputime: proportional to N^2
(cputime is minimized because of simple interpolating functions, where integrals and derivatives can generally be done analytically for one element, and apply to all elements.)

* Accuracy: depends on element and interpolating function

* Irregular cells reduce model errors, but increase storage (must store nodal locations), and may make gridding very difficult. If grid elements vary too much in size or aspect ratio, the matrix can become ill-conditioned.

* Ideal for closed regions, has same difficulty as FDFD with open regions, but can be used.

* Ideal for small to medium (or large with uniform gridding), particularly where model shape is curved and critical.

* May be combined with FDTD to use better FDTD efficiency and better FETD modeling.

SUMMARY:

Least Memory ----------> Most Memory

FDTD FDFD-SOR FE-uniform grid FE-nonuniform grid MOM
MOM-wires
MOM-surface patch

Least cputime ----------> Most Cputime

FDTD FE MOM
FDFD-SOR
MOM-wires
MOM-patches

Special Advantages:
FDFD: low frequency
FDTD: time-domain, large heterogeneous objects
MOM: wires, metal patches, half-spaces
FE: modeling

Special Disadvantages:
FDFD: open regions, square modeling
FDTD: square modeling, low frequency difficult
MOM: memory, cputime, may be difficult to find g(r,r')
FE: memory,cputime, mesh generation, open regions

To increase accuracy:
ALL methods: increase model resolution (This may not help if basic physical principles are violated by initial assumptions, interpolation functions, etc.) .

FDFD: higher-order difference equation (lots more work)
FDTD: higer-order difference equation (lots more work, and storage)
MOM: higher-order basis and weighting functions (a little more work)
FE: higher-order elements (a lot more work, and storage)