There exist several methods for solving ODEs such as the time-symmetric Leapfrog method or the half-point Verlet method. However, the classic algorithm is the fourth-order Runge-Kutta (RK4) method.

Gauss-Seidel-method:~# Gauss-Seidel method is an iterative algorithm using an overrelaxed modified Jacobi method to solve a BVP. The modification uses a singular constantly updating array instead of a double loopwise-updating one.

```
phiprime[i,j] = (phi[i+1,j] + phi[i-1,j] \          # Jacobi method averages neighboring data
+ phi[i,j+1] + phi[i,j-1])/4                        points swapping phi and phiprime at each loop

phi[i,j] = ((1+omega)*(phi[i+1,j] + phi[i-1,j]      # Overrelaxation parameter omega is introduced
+ phi[i,j+1] + phi[i,j-1])/4)-omega*phi[i,j]        and loop uses a single 2D array
```

! Pitfall: Choose omega wisely as it dictates the nature of stability. Also, non-modified overrelaxed Jacobi method is always unstable.

Crank-Nicolson-method:~# Crank-Nicolson method solves an IVP using Neumann stability analysis to force a system to be neutrally stable straddling between the decaying implicit method and the unstable FTCS method.

```
A_banded = np.zeros([3, N+1], complex)              #  The Crank-Nicolson method involves solving
A_banded[0,:] = a2                                  a linear equation involving tridiagonal
A_banded[1,:] = a1                                  matrices. The code snippet shows how to define
A_banded[2,:] = a2                                  the tridiagonal evolution operator.
```

! Pitfall: While Crank-Nicolson is numerically stable, it is still slower that FTCS method. Also, while Crank-Nicolson is faster than spectral method, the former needs to calculate all steps iteratively to desired step.

spectral-method:~# Spectral method solves an IVP by decomposing the solution into a Fourier sine series, solving the coefficients by executing FFT (or FST) to the initial condition, then stitching and inverting back by IFST to form a complete solution.

```
def dst(y):
    N = len(y)
    y2 = np.empty(2*N,float)
    y2[0] = y2[N] = 0.0                             #  Newman provided a user-defined function for
    y2[1:N] = y[1:]                                 the discrete sine transform. A similar one can
    y2[:N:-1] = -y[1:]                              also be defined for the inverse discrete sine
    a = -np.imag(rfft(y2))[:N]                      transform.
    a[0] = 0.0
    return a
```

! Pitfall: Spectral method only works for simple boundaries such as vanishing ones, simply shaped regions such as a box, and linear PDEs.