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] \

+ phi[i,j+1] + phi[i,j-1])/4

# Jacobi method averages neighboring data points swapping phi and phiprime at each loop

phi[i,j] = ((1+omega)\*(phi[i+1,j] + phi[i-1,j] + phi[i,j+1] + phi[i,j-1])/4)-omega\*phi[i,j]

# Overrelaxation parameter omega is introduced 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)

A\_banded[0,:] = a2

A\_banded[1,:] = a1

A\_banded[2,:] = a2

# The Crank-Nicolson method involves solving a linear equation involving tridiagonal matrices. The code snippet shows how to define 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

y2[1:N] = y[1:]

y2[:N:-1] = -y[1:]

a = -np.imag(rfft(y2))[:N]

a[0] = 0.0

# Newman provided a user-defined function for the discrete sine transform. A similar one can also be defined for the inverse discrete sine transform.

return a

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