Unit 20 – Ordinary Differential Equations (ODEs)

**Higher Order ODEs**

- Higher order ODEs, e.g., $y^{(n)} = f(t, y, y', y'', y''', \ldots, y^{(n-1)})$ reduce to a first order system

\[
\begin{pmatrix}
    y_1 \\
    y_2 \\
    y_3 \\
    \vdots \\
    y_{n-1} \\
    y_n
\end{pmatrix}
= \begin{pmatrix}
    y \\
    y' \\
    y'' \\
    \vdots \\
    y^{(n-2)} \\
    y^{(n-1)}
\end{pmatrix}
= \begin{pmatrix}
    y' \\
    y'' \\
    y''' \\
    \vdots \\
    y^{(n-1)} \\
    y^{(n)}
\end{pmatrix}
= \begin{pmatrix}
    y_1 \\
    y_2 \\
    y_3 \\
    \vdots \\
    y_{n-1} \\
    y_n
\end{pmatrix}
= \begin{pmatrix}
    y_2 \\
    y_3 \\
    y_4 \\
    \vdots \\
    y_n
\end{pmatrix}
\]

- Define $y_4 = y^{(n)}$ so that $y_4 = y^{(n)}$

- Thus, only need to consider first order systems of ODEs

- **Newton’s Second Law:** $F = ma$ or $a = x'' = F/m$ or \( \begin{pmatrix} x' \\ v \end{pmatrix} = \begin{pmatrix} v \\ F(x,v)/m \end{pmatrix} \)

**First Order Systems**

- $\ddot{y} = \ddot{f}(t, \dot{y})$ or with only one variable $y' = f(t, y)$

- **Families of Solutions** - e.g., $y' = y$ implies $\frac{dy}{dt} = y$, $\ln y - \ln y_o = t - t_o$, $y = y_o e^{t-t_o}$

- The specific solution depends on the initial condition $y_o = y(t_o)$
Well-Posedness
- Consider $y' = \lambda y$ with solution $y = y_0 e^{\lambda (t-t_0)}$
- When $\lambda > 0$, exponential growth, ill-posed
  - Small changes in initial conditions result in large solution differences
  - Small errors in solver result in large solution differences

- When $\lambda < 0$, exponential decay, well-posed
  - Small changes in initial conditions and solver errors are both damped out

- When $\lambda = 0$, constant solutions, linearly stable (borderline case, mildly ill-posed)
  - Small changes are maintained as is

- For a system of ODEs $\vec{y}' = \vec{f}(t, \vec{y})$, compute the Jacobian matrix $J = \frac{\partial \vec{f}}{\partial \vec{y}}$
- All eigenvalues of $J$ need to be non-positive for all time for well-posedness
- Shouldn’t solve ill-posed ODEs on the computer!

Stability and Accuracy
- For well-posed ODEs, we say a numerical approach is stable if the method does not overflow and produce NaNs
- For well posed ODEs, a stable numerical approach can be analyzed for order of accuracy to see how well it matches a solution relative to the step size
Forward Euler Method

- \( \frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \) or \( y_{k+1} = y_k + hf(t_k, y_k) \)

- Local Truncation Error: \( y_{k+1} = y_k + hf(t_k, y_k) + O(h^2) \)
  - An \( O(h^2) \) error is made at each step

- Global Truncation Error - integrating from \( t = t_o \) to \( t = t_{final} \) with \( n = O(1/h) \) steps gives a total error of \( O(nh^2) = O(h) \)
  - Euler’s method is 1st order accurate with \( \frac{y_{k+1} - y_k}{h} = f(t_k, y_k) + O(h) \)

- Stability
  - Consider the model equation \( y' = \lambda y \) with a well-posed \( \lambda < 0 \)
  - In general, \( \lambda \) is an eigenvalue of the Jacobian matrix
  - Euler’s method applied to the model equation is \( y_{k+1} = y_k + h\lambda y_k = (1 + h\lambda) y_k \)
  - So \( y_k = (1 + h\lambda)^k y_o \) and the error shrinks when \( |1 + h\lambda| < 1 \)
  - Thus, \(-2 < h\lambda < 0\) is needed for stability

- Example
  - Forward Euler on \( y' = -y \) for \( y_0 = 1, t_0 = 0 \). Stability is \( h < 2 \)
  - \( h = .5 \) is stable (left), but \( h = 3 \) is unstable (right)

Backward (Implicit) Euler

- \( \frac{y_{k+1} - y_k}{h} = f(t_{k+1}, y_{k+1}) \), which is 1st order accurate

- Backward Euler applied to \( y' = \lambda y \) is \( y_{k+1} = y_k + h\lambda y_{k+1} \), so \( y_{k+1} = (1 - h\lambda)^{-1} y_k \) and \( y_k = (1 - h\lambda)^{-k} y_o \), and the error shrinks when \( |(1 - h\lambda)^{-1}| < 1 \)

- Thus, \(-\infty < h\lambda < 0\) is needed for stability, i.e. unconditionally stable for any \( h \)

- Generally, need to solve a nonlinear equation to find \( y_{k+1} \)
  - Use Newton iteration, i.e. linearize, solve, linearize, solve, etc.
  - For some applications, only one linearize and solve cycle is used

- One can take very large time steps; however, it is not very accurate

- As \( h \to \infty \), \( y_{k+1} = y_k + h\lambda y_{k+1} \to 0 = 0 + h\lambda y_{k+1} \) or \( y_{k+1} = 0 \)

- This is the long time solution for \( y' = \lambda y \) with \( \lambda < 0 \), but we get there too fast (overdamped)

- Good for stiff problems, e.g. consider \( y = c1 \ y1 + c2 \ y2 \) where \( y2 \) requires a much smaller time step than \( y1 \), but \( c2 \) may also be very small (this sort of behavior is usually hidden in eigenvalues)
Trapezoidal Rule

- \( \frac{y_{k+1} - y_k}{h} = \frac{f(t_k, y_k) + f(t_{k+1}, y_{k+1})}{2} \), which is 2nd order accurate.

- Applied to the model equation: \( y_{k+1} = y_k + \frac{h}{2} \left( y_k + y_{k+1} \right) \), so \( y_{k+1} = (1 + h\mu/2)/(1 - h\mu/2) y_k \) and \( y_k = (1 + h\mu/2)^2/(1 - h\mu/2)^2 y_{k-1} \), and the error shrinks when \( |(1 + h\mu/2)/(1 - h\mu/2)| < 1 \).

- Thus, \( -\infty < h\mu < 0 \) is needed for stability, i.e. unconditionally stable.

- Generally, need to solve a nonlinear equation to find \( y_{k+1} \).

- As \( h \to \infty \), \( y_{k+1} = y_k + \frac{h}{2} \left( y_k + y_{k+1} \right) \to 0 = 0 + \frac{h\mu}{2} \left( y_k + y_{k+1} \right) \) or \( y_{k+1} = -y_k \).

- This is NOT the long time solution for \( y' = \lambda y \).

- Bad oscillatory behavior for too large time steps!

Runge Kutta Methods

- 1st order RK is Euler’s method \( \frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \).

- 2nd order RK \( \frac{y_{k+1} - y_k}{h} = \frac{k_1 + k_2}{2} \) with \( k_1 = f(t_k, y_k) \) and \( k_2 = f(t_k + h, y_k + hk_1) \).

- 4th order RK \( \frac{y_{k+1} - y_k}{h} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6} \) with \( k_1 = f(t_k, y_k) \), \( k_2 = f(t_k + h/2, y_k + hk_1/2) \), \( k_3 = f(t_k + h/2, y_k + hk_2/2) \), and \( k_4 = f(t_k + h, y_k + hk_3) \).

TVD Runge Kutta

- 1st order TVD RK is Euler’s method.

- 2nd order TVD RK is the same as 2nd order RK (above), and also known as the midpoint rule, the modified Euler method, and Heun’s predictor-corrector method:
  - Take two successive forward Euler steps, and average the initial and final state
    - \( \frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \) and \( \frac{y_{k+2} - y_{k+1}}{h} = f(t_{k+1}, y_{k+1}) \) and \( y_{k+1} = \frac{1}{2} y_k + \frac{1}{2} y_{k+1} \)
    - If each Euler step is well-behaved, then since averaging is well behaved, the final result is well behaved.

- 3rd order TVD RK
  - Take two successive forward Euler steps, and average the initial and final state
    - \( \frac{y_{k+1} - y_k}{h} = f(t_k, y_k) \) and \( \frac{y_{k+2} - y_{k+1}}{h} = f(t_{k+1}, y_{k+1}) \) and \( y_{k+1} = \frac{3}{4} y_k + \frac{1}{4} y_{k+1} \)
  - Take another Euler step, and then average yet again
    - \( \frac{y_{k+3/2} - y_{k+1/2}}{h} = f(t_{k+1/2}, y_{k+1/2}) \) and \( y_{k+1} = \frac{1}{3} y_k + \frac{2}{3} y_{k+3/2} \).
Multivalue Methods

- efficiently use lower accuracy on higher derivatives
- Taylor expansion $x^{n+1} = x^n + \Delta t x''^n + \frac{\Delta t^2}{2} x'''^n + O(\Delta t^3)$, e.g. with $x = v$, $x'' = a$ or $v = a$.
- If $x^n$ has $O(\Delta t^r)$ errors, than $x''^n$ can have $O(\Delta t^{r-1})$ errors without ruining the accuracy, and similarly $x'''^n$ can have $O(\Delta t^{r-2})$ errors.
- E.g., $3^{rd}$ order accurate $\dddot{x}$ can be obtained with a $2^{nd}$ order $\dddot{v}$ and $1^{st}$ order $\dddot{a}$
- Solving $\begin{pmatrix} \dddot{x} \\ \dddot{v} \end{pmatrix}$ as a standard system is overkill on $\dddot{v}$

Constant Acceleration Equations

- You have seen these in physics!
- $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \dddot{v}^n + \frac{\Delta t^2}{2} \dddot{a}^n$ quadratic position
- $\dddot{v}^{n+1} = \dddot{v}^n + \Delta t \dddot{a}^n$ linear velocity
- $\dddot{a}^{n+1} = \dddot{a}^n$ constant acceleration (that is constant from time $n$ to just before time $n+1$)

Newmark Methods

- Most used method in computational mechanics (actually, a lot of methods in disguise)
- $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \dddot{v}^n + \frac{\Delta t^2}{2} \left[(1 - 2\beta)\dddot{a}^n + 2\beta \dddot{a}^{n+1}\right]$ and $\dddot{v}^{n+1} = \dddot{v}^n + \Delta t \left[(1 - \gamma)\dddot{a}^n + \gamma \dddot{a}^{n+1}\right]$.
- Choice of $\beta, \gamma$ parameters makes a specific method
- $\beta = \gamma = 0$ - standard constant acceleration equations
- $\beta = 1/2$, $\gamma = 1$ - constant, implicit acceleration
  - $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \dddot{v}^n + \frac{\Delta t^2}{2} \dddot{a}^{n+1}$ and $\dddot{v}^{n+1} = \dddot{v}^n + \Delta t \dddot{a}^{n+1}$
  - Second equation is $1^{st}$ order accurate backward Euler
  - First equation is $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \left(\frac{\dddot{v}^n + \dddot{v}^{n+1}}{2}\right)$ which is the $2^{nd}$ order midpoint rule
  - Overall, still $1^{st}$ order accurate
- $\textbf{Theorem}: 2^{nd}$ order accuracy is obtained if and only if $\gamma = 1/2$
- $\beta = 1/4$, $\gamma = 1/2$ - Trapezoidal rule
  - $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \dddot{v}^n + \frac{\Delta t^2}{2} \left(\dddot{a}^n + \dddot{a}^{n+1}\right)$ and $\dddot{v}^{n+1} = \dddot{v}^n + \Delta t \left(\frac{\dddot{a}^n + \dddot{a}^{n+1}}{2}\right)$
  - first equation is equivalent to $\dddot{x}^{n+1} = \dddot{x}^n + \Delta t \left(\frac{\dddot{v}^n + \dddot{v}^{n+1}}{2}\right)$
Central Differencing

- $\beta = 0$, $\gamma = 1/2$ so $\ddot{x}^{n+1} = \ddot{x}^n + \Delta t \dot{v}^n + \frac{\Delta t^2}{2} a^n$ and $\ddot{v}^{n+1} = \ddot{v}^n + \Delta t \left( \frac{\dddot{a}^n + \dddot{a}^{n+1}}{2} \right)$

- Called central differencing because both the acceleration and the velocity can be expressed as centered finite differences

- Add $\ddot{x}^{n+1} = \ddot{x}^n + \Delta t \dot{v}^n + \frac{\Delta t^2}{2} a^n$ to $\ddot{x}^{n+2} = \ddot{x}^{n+1} + \Delta t \ddot{v}^{n+1} + \frac{\Delta t^2}{2} a^{n+1}$, and rearrange to obtain

$$\frac{\ddot{x}^{n+2} - \ddot{x}^n}{2\Delta t} = \frac{\ddot{v}^{n+1} + \frac{1}{2}\left( \ddot{v}^n + \Delta t \left( \frac{\dddot{a}^n + \dddot{a}^{n+1}}{2} \right) \right)}{2}$$

noting that the last term is identical to $\ddot{v}^{n+1}$, so that

$$\ddot{v}^{n+1} = \ddot{x}^{n+2} - \ddot{x}^n$$

- Subtract $\ddot{x}^{n+1} = \ddot{x}^n + \Delta t \dot{v}^n + \frac{\Delta t^2}{2} a^n$ from $\ddot{x}^{n+2} = \ddot{x}^{n+1} + \Delta t \ddot{v}^{n+1} + \frac{\Delta t^2}{2} a^{n+1}$, and reorganize to obtain

$$\frac{\ddot{x}^{n+2} - 2\ddot{x}^{n+1} + \ddot{x}^n}{\Delta t^2} = \left( \frac{\ddot{v}^{n+1} - \ddot{v}^n}{\Delta t} \right) + \left( \frac{\dddot{a}^n + \dddot{a}^{n+1}}{2} \right)$$

noting that $\frac{\ddot{v}^{n+1} - \ddot{v}^n}{\Delta t} = \frac{\dddot{a}^n + \dddot{a}^{n+1}}{2}$, so

$$\dddot{a}^{n+1} = \dddot{a}^n - 2\dddot{a}^{n+1} + \dddot{a}^n$$

Staggering Position and Velocity

- Define the velocity at the half grid points so that $\ddot{v}^{n+1/2} = \ddot{x}^{n+1/2} - \ddot{x}^n$ is second order accurate, where $\ddot{x}^n$ is still at grid points

- If we define $\ddot{v}^{n+1} = \ddot{v}^{n+1} + \ddot{v}^{n+3/2} \frac{2}{2\Delta t} = \ddot{x}^{n+2} - \ddot{x}^n$, then this is central differencing for velocity

- We can rewrite this as an update formula for position $\ddot{x}^{n+1} = \ddot{x}^n + \Delta t \ddot{v}^{n+1/2}$

- The acceleration is at the grid points, and $\ddot{v}^{n+3/2} - \ddot{v}^{n+1/2} = \dddot{a}^{n+1}$ is second order accurate, and

$$\frac{\Delta t}{\Delta t} = \dddot{a}^{n+1}$$

equivalent to

$$\frac{\ddot{x}^{n+2} - \ddot{x}^n}{\Delta t} - \left( \frac{\ddot{x}^{n+1} - \ddot{x}^n}{\Delta t} \right) = \dddot{a}^{n+1}$$

or

$$\dddot{a}^{n+1} = \dddot{a}^n - 2\dddot{a}^{n+1} + \dddot{a}^n$$

which is also central differencing

- We can rewrite this as an update formula for velocity $\ddot{v}^{n+3/2} = \ddot{v}^{n+1/2} + \Delta t \dddot{a}^{n+1}$

- The acceleration at grid points is evaluated via $\dddot{a}^{n+1} = \dddot{a}(\ddot{x}^{n+1}, \ddot{v}^{n+1}) = \dddot{a} \left( \ddot{x}^{n+1}, \ddot{v}^{n+1/2} + \ddot{v}^{n+3/2} \right)$
Central Differencing with Staggered Position/Velocit

- Given \( x^n \) and \( v^n \)

**Position Update:**

- By definition \( \tilde{v}^n = \frac{v^{n-1/2} + v^{n+1/2}}{2} \), so \( v^{n+1/2} = 2\tilde{v}^n - v^{n-1/2} = 2\tilde{v}^n - (\tilde{v}^{n+1/2} - \Delta t\tilde{a}^n) \) using \( \tilde{v}^{n+1/2} = \tilde{v}^{n-1/2} + \Delta t\tilde{a}^n \)

  - This can be rearranged to \( \tilde{v}^{n+1/2} = \tilde{v}^n + \frac{\Delta t}{2} \tilde{a}^n \) to get the half step velocity for advancing the position via \( x^{n+1} = x^n + \Delta t\tilde{v}^{n+1/2} \)

  - This is identical to \( \tilde{x}^{n+1} = \tilde{x}^n + \Delta t\tilde{v}^n + \frac{\Delta t^2}{2} \tilde{a}^n \)

**Velocity Update:**

- Then \( \tilde{v}^{n+3/2} = \tilde{v}^{n+1/2} + \Delta t\tilde{a}^n \left( \tilde{x}^{n+1}, \frac{\tilde{v}^{n+1/2} + \tilde{v}^{n+3/2}}{2} \right) \), and using \( \tilde{v}^{n+1} = \frac{\tilde{v}^{n+1/2} + \tilde{v}^{n+3/2}}{2} \) leads to

  \[
  2\tilde{v}^{n+1} - \tilde{v}^{n+1/2} = \tilde{v}^{n+1/2} + \Delta t\tilde{a}^n \left( \tilde{x}^{n+1}, \tilde{v}^{n+1} \right) \quad \text{or} \quad \tilde{v}^{n+1} = \tilde{v}^{n+1/2} + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^{n+1}, \tilde{v}^{n+1})
  \]

  - This last equation is implicit in the velocity (but fully explicit if acceleration doesn’t depend on velocity)

  - Otherwise iterate \( \tilde{u}^{k+1} = \tilde{v}^{n+1/2} + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^{n+1}, \tilde{u}^k) \) starting with \( \tilde{u}^0 = \tilde{v}^{n+1/2} \) noting that \( \tilde{u}^1 = \tilde{v}^{n+1/2} + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^{n+1}, \tilde{v}^{n+1/2}) \) is an explicit time step

- Often the dependence of acceleration on velocity is symmetric (e.g. damping forces), so one can use a fast Ax=b solver, e.g. PCG

- Overall, \( \tilde{v}^{n+1} = \tilde{v}^{n+1/2} + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^{n+1}, \tilde{v}^{n+1}) = \tilde{v}^n + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^n, \tilde{v}^n) + \frac{\Delta t}{2} \tilde{a}(\tilde{x}^{n+1}, \tilde{v}^{n+1}) \) which is the trapezoidal rule

**Stability/Accuracy:**

- No stability restriction on the time step from acceleration dependence on velocity
- Only time step stability restriction comes from the acceleration dependence on position
- Of course, bigger time steps are bad for the trapezoidal rule, so one could switch to backward Euler for the velocity, \( \tilde{v}^{n+1} = \tilde{v}^n + \Delta t\tilde{a}(\tilde{x}^{n+1}, \tilde{v}^{n+1}) \), while still using \( \tilde{x}^{n+1} = \tilde{x}^n + \Delta t\tilde{v}^n + \frac{\Delta t^2}{2} \tilde{a}^n \) for position (drops to 1st order accuracy)