**Review:** Methods for solving \( y'(t) = f(t, y) \)

* **Forward Euler:** \( y_{n+1} = y_n + dt \cdot f(t_n, y_n) \)
  
  \( \rightarrow \) Explicit, conditionally stable, 1st order accurate

* **Backward Euler:** \( y_{n+1} = y_n + dt \cdot f(t_{n+1}, y_{n+1}) \)
  
  \( \rightarrow \) Implicit, unconditionally stable, 1st order accurate

* **Trapezoidal Rule:** \( y_{n+1} = y_n + \frac{dt}{2} \left\{ f(t_n, y_n) + f(t_{n+1}, y_{n+1}) \right\} \)
  
  \( \rightarrow \) Implicit, unconditionally stable, 2nd order accurate

---

**IV. (The last property)** **OSCILLATORY BEHAVIOR**

(For B.E. & T.R. only)

Implicit methods allow us (in theory) to take arbitrarily large steps \( dt \gg 1 \). But, what happens if we do?

Check on model equation \( y' = -\lambda y \)

**B.E.**

\[
 y_{n+1} = y_n + \lambda dt \cdot y_{n+1} \Rightarrow y_{n+1} = \frac{1}{1 - \lambda dt} y_n \quad \text{as} \quad dt \to \infty
\]

**T.R.**

\[
 y_{n+1} = \frac{1 + \lambda dt/2}{1 - \lambda dt/2} y_n \quad \text{as} \quad dt \to \infty
\]

Thus, when using large timesteps, B.E. tends to quickly settle on the steady-state solution, while trapezoidal rule may oscillate for a prolonged period before settling.
Systems of ODE's

We often have systems of differential equations, with more than one unknown function, e.g.

\[
\begin{align*}
\frac{d}{dt} y_1(t) &= f_1(t, y_1(t), y_2(t), ..., y_n(t)) \\
\frac{d}{dt} y_2(t) &= f_2(t, y_1(t), y_2(t), ..., y_n(t)) \\
&\vdots \\
\frac{d}{dt} y_n(t) &= f_n(t, y_1(t), y_2(t), ..., y_n(t)).
\end{align*}
\]

A special case arises when each function \( f_i \) is linear in the unknown functions \( y_j(t) \), i.e. \( f_i = c_{i1} y_1(t) + c_{i2} y_2(t) + ... + c_{in} y_n(t) \), e.g.

\[
\begin{align*}
\frac{d}{dt} y_1(t) &= a_{11} y_1(t) + a_{12} y_2(t) + a_{13} y_3(t) \\
\frac{d}{dt} y_2(t) &= a_{21} y_1(t) + a_{22} y_2(t) + a_{23} y_3(t) \\
\frac{d}{dt} y_3(t) &= a_{31} y_1(t) + a_{32} y_2(t) + a_{33} y_3(t)
\end{align*}
\]

\[
\frac{d}{dt} \begin{pmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{pmatrix} = A \begin{pmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{pmatrix}.
\]

In each case, we can extend the concept of an integration method to a system of ODEs (linear or nonlinear) in the "obvious" fashion.
\[ y_1' = f_1(t, y_1, y_2) \]
\[ y_2' = f_2(t, y_1, y_2) \]

→ Using forward Euler
\[ y_1^{n+1} = y_1^n + dt f_1(t^n, y_1^n, y_2^n) \]
\[ y_2^{n+1} = y_2^n + dt f_2(t^n, y_1^n, y_2^n) \]

\[ y_1' = y_1^n + dt f_1(t^n, y_1^n, y_2^n) \]
\[ y_2' = y_2^n + dt f_2(t^n, y_1^n, y_2^n) \]

\[ y_1^{n+1} = y_1^n + dt f_1(t^n, y_1^n, y_2^n) \]
\[ y_2^{n+1} = y_2^n + dt f_2(t^n, y_1^n, y_2^n) \]

\[ y_1^{n+1} = y_1^n + dt f_1(t^n, y_1^n, y_2^n) \]
\[ y_2^{n+1} = y_2^n + dt f_2(t^n, y_1^n, y_2^n) \]

\[ \frac{dy}{dt} = A \hat{y} \quad \text{(or } \frac{dy}{dt} = Ay \text{ for brevity)} \]

\[ y \in \mathbb{R}^n, A \in \mathbb{R}^{n \times n} \]

**Forward Euler:**
\[ y^{n+1} = y^n + dt \cdot Ay^n \Rightarrow y^{n+1} = (I + dtA) y^n \]

**Backward Euler:**
\[ y^{n+1} = y^n + dt \cdot Ay^{n+1} \Rightarrow (I - dtA) y^{n+1} = y^n \]

(or \( y^{n+1} = (I - dtA)^{-1} y^n \)) = Need to solve a linear system.

**Trapezoidal Rule:**
\[ y^{n+1} = y^n + \frac{dt}{2} \left[ Ay^n + Ay^{n+1} \right] \]

\[ \Rightarrow (I - \frac{dt}{2} A) y^{n+1} = (I + \frac{dt}{2} A) y^n \]

\[ \text{Linear system to solve} \]
The properties of an integration rule mostly carry over from the scalar case to the case of ODE systems, i.e.

- If we use B.E. or T.R to solve a system of ODEs, the resulting method will be unconditionally stable.

- The order of accuracy in a system mirrors the order of accuracy observed on \( y' = \lambda y \).

Some questions remain more complicated, e.g.:

- An ODE \( y' = \lambda y \) had **stable solutions** when \( \lambda < 0 \) (i.e. the solutions exhibited exponential decay).

  What happens with systems?

**Answer:** This can be answered easily for the case of **linear systems**, i.e. \( y' = Ay \).

**Remember:** \( \lambda \) is an **eigenvalue** of \( A \), iff \( \text{det}(A - \lambda I) = 0 \).

Eigenvalues can be complex numbers, and found by solving the polynomial equation of degree \( n \) : \( \text{det}(A - \lambda I) = 0 \).

**Theorem:** If \( \text{Re}(\lambda) < 0 \) for all eigenvalues of \( A \), the solutions are **stable** (i.e. decay to zero).
An integration method for \( y' = Ay \) was ultimately written as \( y_{n+1} = k \cdot y_n \), and \( |k| < 1 \) was the condition for stability. What happens with systems?

**Answer:** An integration scheme for \( \tilde{y}' = A \tilde{y} \) is also ultimately written as \( \tilde{y}^{n+1} = k \tilde{y}^n \)

(i.e. for F.E. \( K = I + \delta t A \), for B.E. \( K = (I - \delta t A)^{-1} \)).

The stability condition translates to \( \| \lambda \| < 1 \) Complex magnitude for any eigenvalue \( \lambda_i \) of \( K \).

**Position / Velocity Systems**

We previously saw that a mass-spring system is governed by the 2nd order ODE

\[
f(t, x, v) = ma \quad \text{or} \quad f(t, x(t), x'(t)) = m \cdot x''(t)
\]

We then converted this equation to the 1st order system

\[
\begin{pmatrix}
x(t) \\
v(t)
\end{pmatrix}' =
\begin{pmatrix}
v(t) \\
\frac{1}{m} f(t, x(t), v(t))
\end{pmatrix} = F_1(t, x, v)
\]

\[
\begin{pmatrix}
x(t) \\
v(t)
\end{pmatrix} =
\begin{pmatrix}
v(t) \\
\frac{1}{m} f(t, x(t), v(t))
\end{pmatrix} = F_2(t, x, v)
\]
By directly mapping the previous methods to this system, we get, e.g.

**Forward Euler**

\[ x^{n+1} = x^n + dt \cdot v^n \]

\[ v^{n+1} = v^n + \frac{dt}{m} f(t, x^n, v^n) \]

*Easy implementation* — we already demonstrated it.

**Backward Euler**

\[ x^{n+1} = x^n + dt \cdot v^{n+1} \]

\[ v^{n+1} = v^n + \frac{dt}{m} f(t, x^n, v^{n+1}) \]

Somewhat unclear how to solve!

We will soon examine how the implicit system can be solved in practice; in the meantime, we examine yet another method which becomes an option for ODE systems:

→ We can design new integration methods by mixing/matching elements from different methods, for the distinct equations of the system.

*e.g.* \[ x^{n+1} = x^n + \frac{dt}{2} \left\{ v^n + v^{n+1} \right\} \]

*Trapezoidal-like*

\[ v^{n+1} = v^n + \frac{dt}{m} f(x^n, v^{n+1}) \]

*F.E.-like for \( x \)*

*B.E.-like for \( v \)*

These methods combine features of their "component" methods. For example, this method manages to be 2nd order overall, while the \( dt \) restriction depends only on \( k \) (Young's modulus) and not on the...
Additionally, this "hybrid" method will make it easier for us to describe a practical method for computing the values $x^{n+1}$ & $v^{n+1}$.

We start by revisiting the damping force incurred by a spring:

$$
\begin{align*}
\mathbf{f}_1 &= -b \mathbf{nn}^T (\mathbf{v}_1 - \mathbf{v}_2) \\
\mathbf{f}_2 &= -\mathbf{f}_1 \\
&= -b \mathbf{nn}^T (\mathbf{v}_1 - \mathbf{v}_2) \\
&= b \mathbf{nn}^T (\mathbf{v}_1 - \mathbf{v}_2) \\
&= b \mathbf{nn}^T \mathbf{v}_1 - b \mathbf{nn}^T \mathbf{v}_2
\end{align*}
$$

Let us manipulate this definition a bit:

$$
\begin{align*}
\mathbf{f}_1 &= -b \mathbf{nn}^T \mathbf{v}_1 + b \mathbf{nn}^T \mathbf{v}_2 \\
\mathbf{f}_2 &= b \mathbf{nn}^T \mathbf{v}_1 - b \mathbf{nn}^T \mathbf{v}_2
\end{align*}
$$

We observe that we can write this in matrix form:

$$
\begin{pmatrix}
\mathbf{f}_1 \\
\mathbf{f}_2 \\
\mathbf{v}_1 \\
\mathbf{v}_2
\end{pmatrix}
= 
\begin{pmatrix}
b \mathbf{nn}^T & b \mathbf{nn}^T \\
b \mathbf{nn}^T & -b \mathbf{nn}^T
\end{pmatrix}
\begin{pmatrix}
\mathbf{v}_1 \\
\mathbf{v}_2
\end{pmatrix}
$$
More generally, if this spring was connecting particles \( i \) & \( j \), the resulting damping force could be computed as:

\[
\begin{bmatrix}
f_1^d \\
f_2^d \\
\vdots \\
f_{i-1}^d \\
f_i^d \\
\vdots \\
f_{j-1}^d \\
f_j^d \\
f_{j+1}^d \\
f_{n+1}^d \\
\end{bmatrix}
= 
\begin{bmatrix}
& \ddots & \ddots & \ddots \\
& & \ddots & \ddots & \ddots \\
& & & \ddots & \ddots & \ddots \\
& & & & \ddots & \ddots \\
& & & & & \ddots \\
\end{bmatrix}

\begin{bmatrix}
G_{ij} \\
\vdots \\
G_{ji} \\
\vdots \\
G_{kn} \\
\end{bmatrix}

\]

where \( n_{ij} = (x_i - x_j) / \| x_i - x_j \| \)

By adding all the matrices from all springs we get:

\[
G = \sum_{(ij) \text{ is a spring}} G_{ij}
\]

and finally \( \mathbf{F}^d = G \cdot \mathbf{V} \) for all damping forces, collectively.

To be precise: \( G \) is not a constant; it depends on the positions \( \mathbf{X} \), via the normals \( n_{ij} \). Overall we can write:

\[
f^d(x, \mathbf{V}) = G(X) \cdot \mathbf{V}
\]
Adding the damping forces to the elastic forces \( f^{el} \) we get:

\[
f(x,v) = f(x) + f^{el}(x,v)
\]

\[
= f^{el}(x) + G(x) \cdot v
\]

Thus, the second equation of our "hybrid" integration rule becomes:

\[
u^{n+1} = u^n + \frac{dt}{m} \int f(x^n, v^{n+1})
\]

\[
= u^n + \frac{dt}{m} \left\{ f^{el}(x^n) + G(x^n) v^{n+1} \right\}
\]

\[
\Rightarrow \left( I - \frac{dt}{m} G(x^n) \right) v^{n+1} = u^n + \frac{dt}{m} f^{el}(x^n)
\]

After solving this linear system for \( v^{n+1} \), we substitute into

\[
x^{n+1} = x^n + \frac{dt}{2} \left\{ v_n + v_{n+1} \right\}
\]

Implementation demonstration = next class!