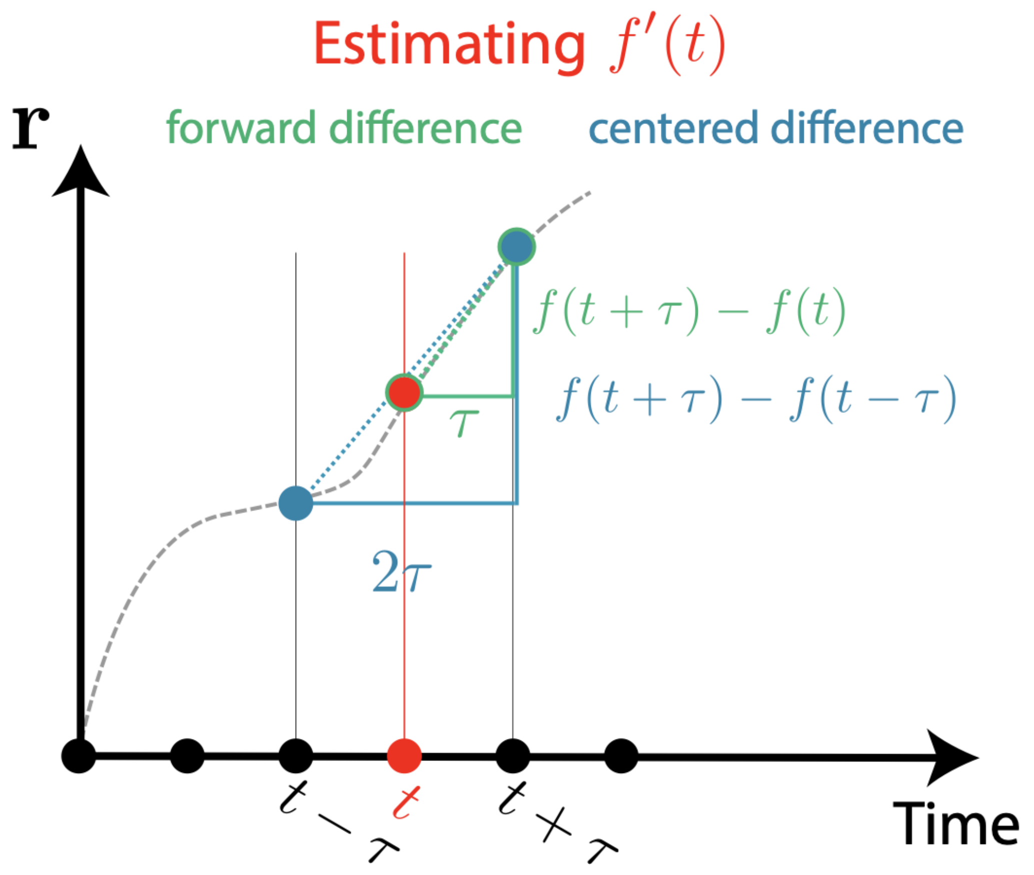
***Recap*: Week 2**

- Introduced the *Kepler problem*
  - *Non-dimensionalisation* simplified the formulation.
- Introduced the *centered-difference approximation* to overcome limitations of the scheme based on forward-difference approximation (Euler).
  - Yields a new method, *Verlet*, which exhibits breath-taking accuracy for Keplerian orbits with near-conservation of total energy

How to step forward in time, given a gradient dr/dt?

- *Euler*: use *forward difference* approximation
  - The velocity at the current time step to extrapolate forward τ in time.
- *Verlet* (for solving a dynamics problem): use a *centered difference* approximation for the second derivative
  - So can use the acceleration and two previous values to make the update.



Computational Physics: Lecture 3

- Our last lecture on ODEs (PDEs from next week!)
- *General form of ODEs* suitable for numerical solution.
  - Dynamics equations we've considered so far are a special case (Verlet).
- Introduce *Runge-Kutta (RK)* (Taylor series) methods:
  - Can be applied to the general form for ODEs.
  - Euler's method is the first-order RK scheme.
- Simulate motion of a *simple pendulum*.
  - Illustrate the general form and a vector ODE *right-hand side function*.

General form of a system of ODEs

- $\boxed{\dfrac{d\mathbf{x}}{dt} = \mathbf{f}\left[\mathbf{x}(t), t\right]}$.
    - N coupled first order ordinary differential equations (ODEs).
    - t is the *independent* variable (often time, but could be anything).
- $\mathbf{x}$ is a vector of the *dependent* variables, $\mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_N(t)]$.
- $\mathbf{f}$ is a vector function called the *right-hand side* (RHS):
    - $\mathbf{f}[\mathbf{x}(t), t] = \{f_1[\mathbf{x}(t), t], f_2[\mathbf{x}(t), t], \ldots, f_N[\mathbf{x}(t), t]\}$.
- General *Initial Value Problem* (IVP):
    - Given $\mathbf{x}(t_1)$, calculate $\mathbf{x}(t)$ for $t > t_1$.
- Writing ODEs in this form makes it easy for us to apply general numerical methods
    - Just need to specify the *RHS function* $\mathbf{f}\left[\mathbf{x}(t), t\right]$.

Writing higher-order ODEs in general form

- A similar trick can get any high-order ODE into the general form
    - ❓ Where did we do this already?
- Consider Nth-order linear ODE in y_1: $a_1 y_1 + a_2 \dfrac{dy_1}{dt} + a_3 \dfrac{d^2 y_1}{dt^2} + \cdots + a_{N+1} \dfrac{d^N y_1}{dt^N} = 0\,.$
- We can always *replace higher derivatives by new variables*, y_2, y_3, ...:
    - $y_2 = \dfrac{dy_1}{dt}\,, \quad y_3 = \dfrac{dy_2}{dt} = \dfrac{d^2 y_1}{dt^2}\,, \quad \cdots \quad y_N = \dfrac{dy_{N-1}}{dt} = \left(\dfrac{d}{dt}\right)^{N-1} y_1\,.$
- So now we can rewrite the Nth-order ODE equivalently in *general form*, $\dfrac{d\mathbf{x}}{dt} = \mathbf{f}$, by identifying:
    - $\mathbf{x} = (y_1, y_2, \ldots, y_N)$, and $\mathbf{f} = \left[y_2, y_3, \ldots, -\dfrac{1}{a_{N+1}}(a_1 y_1 + a_2 y_2 \cdots + a_N y_N)\right].$
- *You need to know how to apply this procedure*

Dynamics problems in general form

- Dynamics problems are a *special case*:
    - $\frac{d^2\mathbf{r}}{dt^2} = \mathbf{a}$ can be written $\frac{d\mathbf{r}}{dt} = \mathbf{v}\,,\ \frac{d\mathbf{v}}{dt} = \mathbf{a}$.
    - Position $\mathbf{r} = (x, y, z)$, velocity $\mathbf{v} = (v_x, v_y, v_z)$, and acceleration, $\mathbf{a} = (a_x, a_y, a_z)$.
- We can write them in the *general form*, $\frac{d\mathbf{x}}{dt} = \mathbf{f}\,(\mathbf{x}, t)$, by identifying $\mathbf{x} = (x, y, z, v_x, v_y, v_z)^T$ and $\mathbf{f} = (v_x, v_y, v_z, a_x, a_y, a_z)^T$:

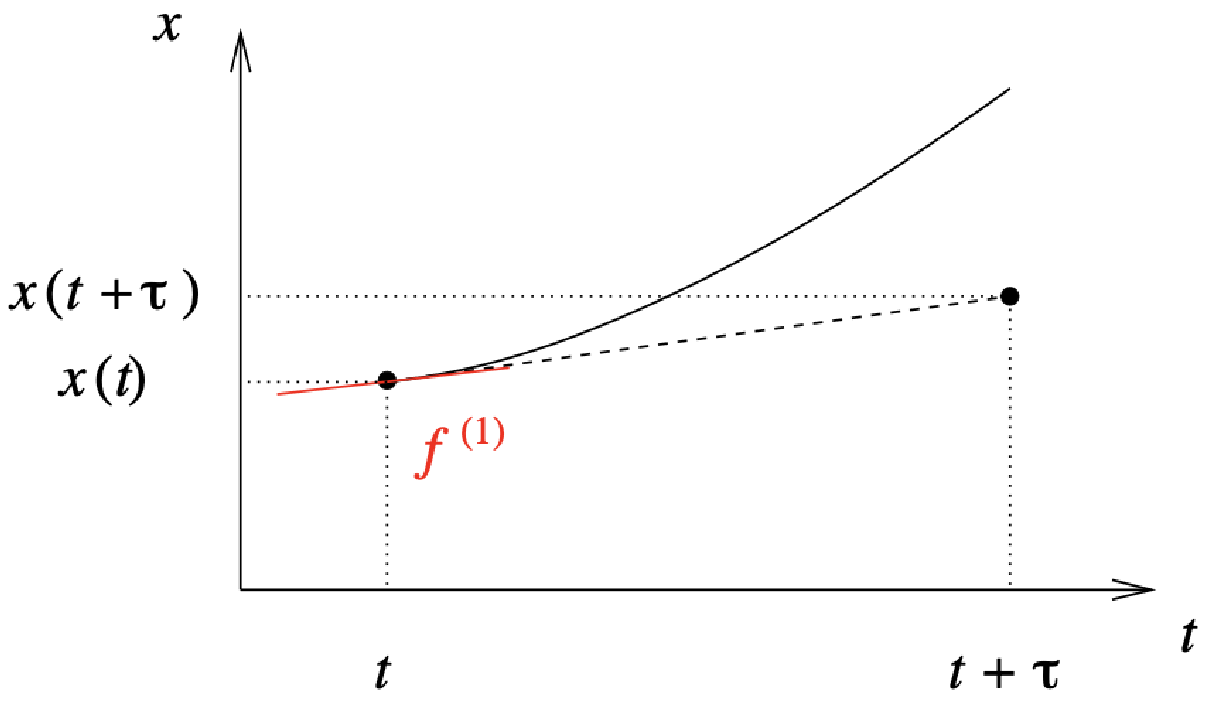
$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ a_x \\ a_y \\ a_z \end{pmatrix}.$$

Euler's method for the general form

- *Taylor-series* expansion (using $\frac{d\mathbf{x}}{dt} = \mathbf{f}[\mathbf{x}(t), t]$):
  - $$\begin{aligned} \mathbf{x}(t+\tau) &= \mathbf{x}(t) + \tau \frac{d\mathbf{x}}{dt} + O(\tau^2)\,, \\ &= \mathbf{x}(t) + \tau \mathbf{f}[\mathbf{x}(t), t] + O(\tau^2)\,. \end{aligned}$$
- *Euler's method* matches the Taylor series to $O(\tau)$:
  - $\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tau \mathbf{f}^{(1)}$, where $\mathbf{f}^{(1)} = \mathbf{f}[\mathbf{x}(t), t]$.
  - For Euler, $\mathbf{f}^{(1)}$ is the *right-hand side (RHS)* of the ODE.
- Euler for dynamics problems is a special case.

An Euler step in 1-D

- $x(t+\tau) = x(t) + \tau f^{(1)}$,
  - with $f^{(1)} = f[x(t), t] = \dfrac{dx}{dt}$.
- $f^{(1)}$ is $\dfrac{dx}{dt}$ evaluated at $(t, x(t))$:
  - the gradient of the tangent (red).
- Euler extrapolates forward τ along the tangent line.



Testing Euler on a simple exponential case

- *Simple ODE/IVP:* $\dfrac{dx}{dt} = x$ with $x(0) = 1$.
  - Analytic solution is $x(t) = e^t$ so integrating to $t = 1$ the final value should be $\approx 2.71828...$.
- `exp_euler` solves this using Euler's method by defining a function `rhs_exp(x)` that returns the same input, `x`.
  - ❓ Why is RHS $f^{(1)} = x$ here?
- *Testing* with `exp_euler` 😋
  - Let's compute the error for a 10-step integration (dividing the unit interval into 10 steps).
  - ❓ What's your prediction for a 20-step integration...?
- Euler works ok, but *can we do better than just taking little steps forward based on extrapolating the current derivative*?

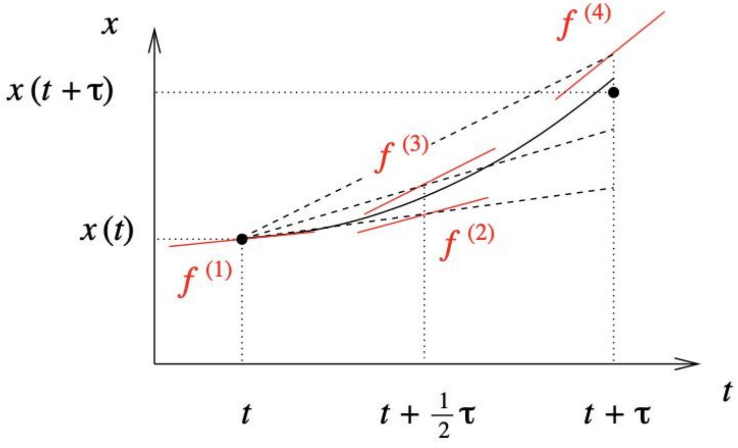
Pe

t	x
0	1
0.05	1.05
0.1	1.1025
0.15	1.1576
0.2	1.2155
0.25	1.2763
0.3	1.3401
0.35	1.4071
0.4	1.4775
0.45	1.5513
0.5	1.6289
0.55	1.7103
0.6	1.7959
0.65	1.8856
0.7	1.9799
0.75	2.0789
0.8	2.1829
0.85	2.292
0.9	2.4066
0.95	2.527
1	2.6533

```
Error: 2.39063%
```

Runge-Kutta

- Carl Runge (1856-1927) and Martin Kutta (1867-1944).
- In general, we can do better than naively marching forward with a single estimate of the derivative by combining information at fractions of a step.



Runge-Kutta Methods

- Runge-Kutta methods *match the Taylor series* to a given order.
  - Euler is 'first-order Runge-Kutta': $\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tau\mathbf{f}^{(1)}$.
- Higher-order schemes are more accurate.
- A *very popular* one: ⭐ ***Fourth-Order Runge-Kutta (RK4)*** ⭐:

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tfrac{1}{6}\tau\left[\mathbf{f}^{(1)} + 2\mathbf{f}^{(2)} + 2\mathbf{f}^{(3)} + \mathbf{f}^{(4)}\right],$$

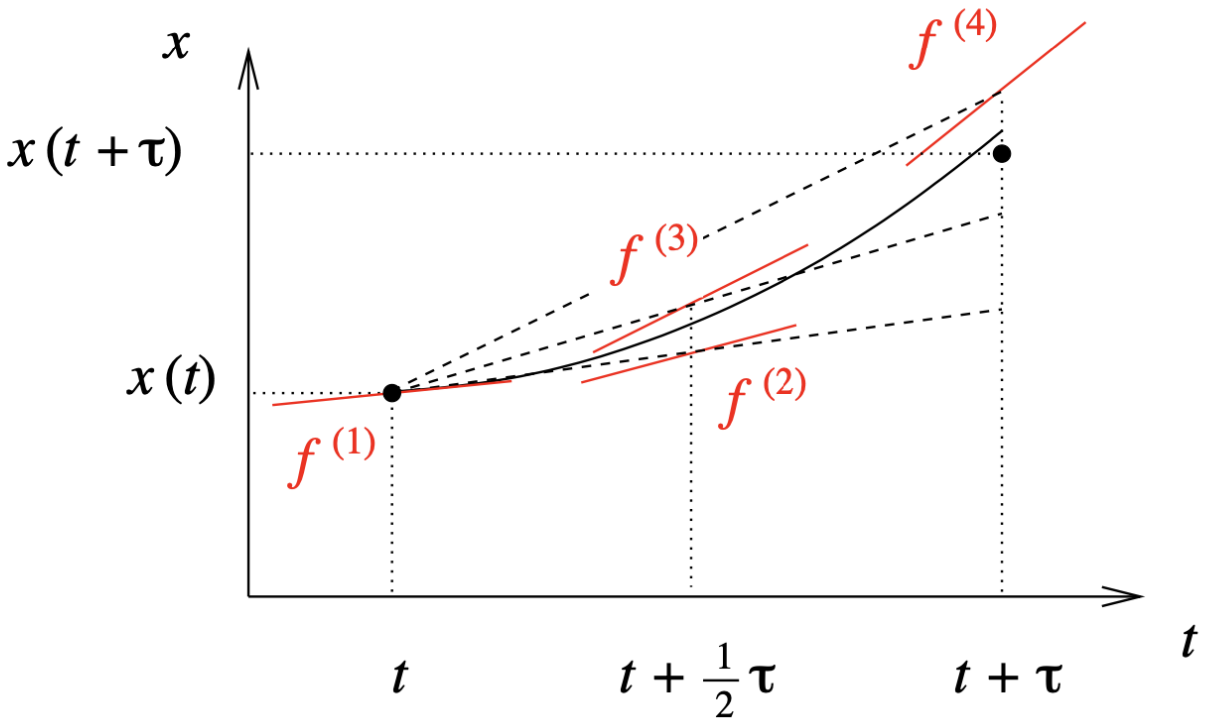
$$\begin{aligned}
\mathbf{f}^{(1)} &= \mathbf{f}[\mathbf{x}(t), t]\,, \\
\mathbf{f}^{(2)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(1)}, t + \tfrac{1}{2}\tau]\,, \\
\mathbf{f}^{(3)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(2)}, t + \tfrac{1}{2}\tau]\,, \\
\mathbf{f}^{(4)} &= \mathbf{f}[\mathbf{x}(t) + \tau\mathbf{f}^{(3)}, t + \tau]\,.
\end{aligned}$$

Fourth-Order Runge-Kutta (RK4)

- Four RHS/gradient evaluations:
  - two at a half step $(t + \frac{1}{2}\tau)$
  - two at a full step $(t + \tau)$.
- Extrapolates forward to $t + \tau$ along a complicated average gradient line.

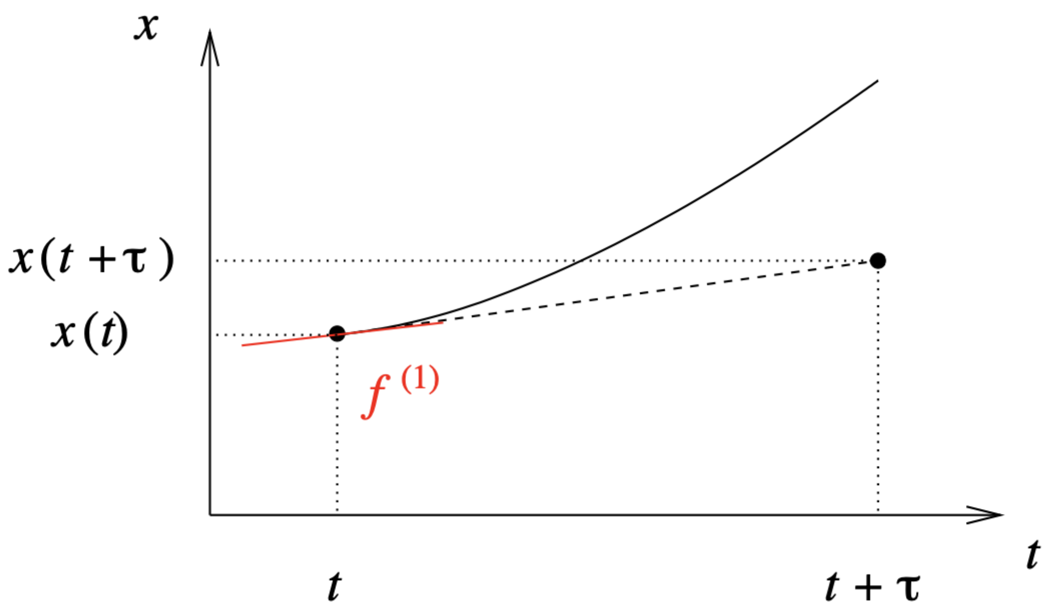
$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tfrac{1}{6}\tau\left[\mathbf{f}^{(1)} + 2\mathbf{f}^{(2)} + 2\mathbf{f}^{(3)} + \mathbf{f}^{(4)}\right],$$

$$\begin{aligned}
\mathbf{f}^{(1)} &= \mathbf{f}[\mathbf{x}(t), t]\,,\\
\mathbf{f}^{(2)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(1)}, t + \tfrac{1}{2}\tau]\,,\\
\mathbf{f}^{(3)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(2)}, t + \tfrac{1}{2}\tau]\,,\\
\mathbf{f}^{(4)} &= \mathbf{f}[\mathbf{x}(t) + \tau\mathbf{f}^{(3)}, t + \tau]\,.
\end{aligned}$$



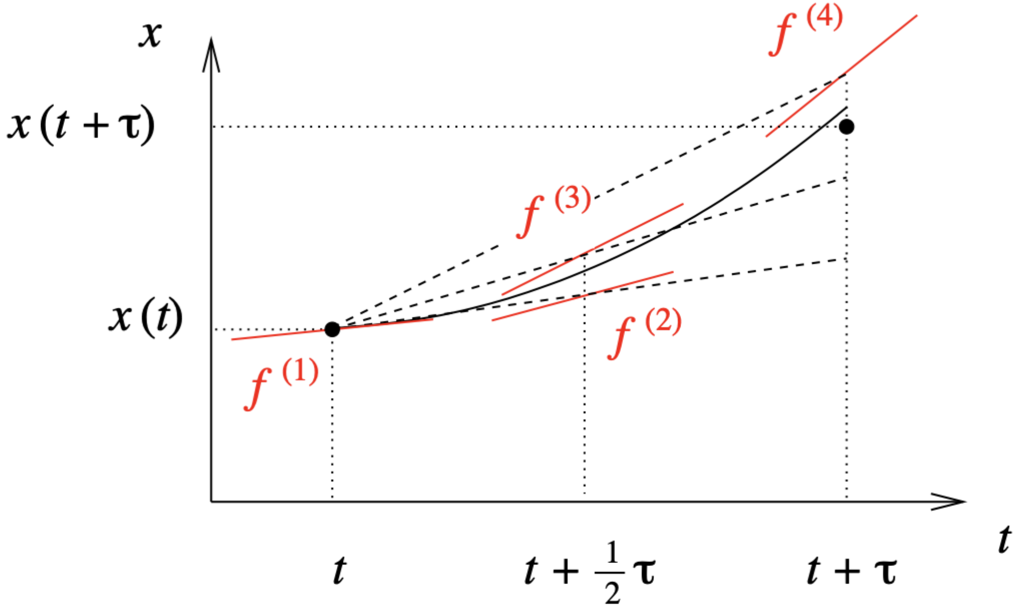
Euler (RK1)

$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tau \mathbf{f}^{(1)}.$



RK4

- $\mathbf{x}(t+\tau) = \mathbf{x}(t) + \frac{1}{6}\tau \left[\mathbf{f}^{(1)} + 2\mathbf{f}^{(2)} + 2\mathbf{f}^{(3)} + \mathbf{f}^{(4)}\right]$
  - $\mathbf{f}^{(1)} = \mathbf{f}[\mathbf{x}(t), t]$
  - $\mathbf{f}^{(2)} = \mathbf{f}[\mathbf{x}(t) + \frac{1}{2}\tau \mathbf{f}^{(1)}, t + \frac{1}{2}\tau]$
  - $\mathbf{f}^{(3)} = \mathbf{f}[\mathbf{x}(t) + \frac{1}{2}\tau \mathbf{f}^{(2)}, t + \frac{1}{2}\tau]$
  - $\mathbf{f}^{(4)} = \mathbf{f}[\mathbf{x}(t) + \tau \mathbf{f}^{(3)}, t + \tau]$



Understanding Runge-Kutta methods

- We're given a function, $\mathbf{f}\left[\mathbf{x}(t), t\right]$ and we have to try to work out how make a good step forward in time
- Consider a *Taylor-series expansion* of the dependent variables:
  - $\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tau \frac{d\mathbf{x}}{dt} + \frac{\tau^2}{2!}\frac{d^2\mathbf{x}}{dt^2} + \frac{\tau^3}{3!}\frac{d^3\mathbf{x}}{dt^3} + \dots$
- The Runge-Kutta approach is to *reproduce this series to a certain order*.
- *Euler matches to first order* (is 'RK1')
  - Uses RHS directly: $\frac{d\mathbf{x}}{dt} = \mathbf{f}\left[\mathbf{x}(t), t\right]$.
- Fourth-order Runge-Kutta (RK4) *matches the Taylor series to* $O(\tau^4)$.
  - i.e., local truncation error $O(\tau^5)$: *very accurate*
  - *Very commonly used* and can get you far: Matlab has RK4 `ode45` (but we will write our own version 😆)
- The derivation is complicated (cf. Appendix for RK2).

Code: Stepping with Euler

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tau \mathbf{f}^{(1)}$$

```
for n in range(numSteps):
    # One step of Explicit Euler
    f = rhs_exp(x)
    x = x + tau * f
    t = t + tau
```

Code: stepping with RK4

$$\mathbf{x}(t+\tau) = \mathbf{x}(t) + \tfrac{1}{6}\tau\left[\mathbf{f}^{(1)} + 2\mathbf{f}^{(2)} + 2\mathbf{f}^{(3)} + \mathbf{f}^{(4)}\right],$$

$$\begin{aligned}
\mathbf{f}^{(1)} &= \mathbf{f}[\mathbf{x}(t), t]\,, \\
\mathbf{f}^{(2)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(1)}, t + \tfrac{1}{2}\tau]\,, \\
\mathbf{f}^{(3)} &= \mathbf{f}[\mathbf{x}(t) + \tfrac{1}{2}\tau\mathbf{f}^{(2)}, t + \tfrac{1}{2}\tau]\,, \\
\mathbf{f}^{(4)} &= \mathbf{f}[\mathbf{x}(t) + \tau\mathbf{f}^{(3)}, t + \tau]\,.
\end{aligned}$$

```
for n in range(numSteps):
    # One step of RK4
    f1 = rhs_exp(x)
    f2 = rhs_exp(x + 0.5 * tau * f1)
    f3 = rhs_exp(x + 0.5 * tau * f2)
    f4 = rhs_exp(x + tau * f3)

    x = x + tau * (f1 + 2*f2 + 2*f3 + f4) /
    t = t + tau
```

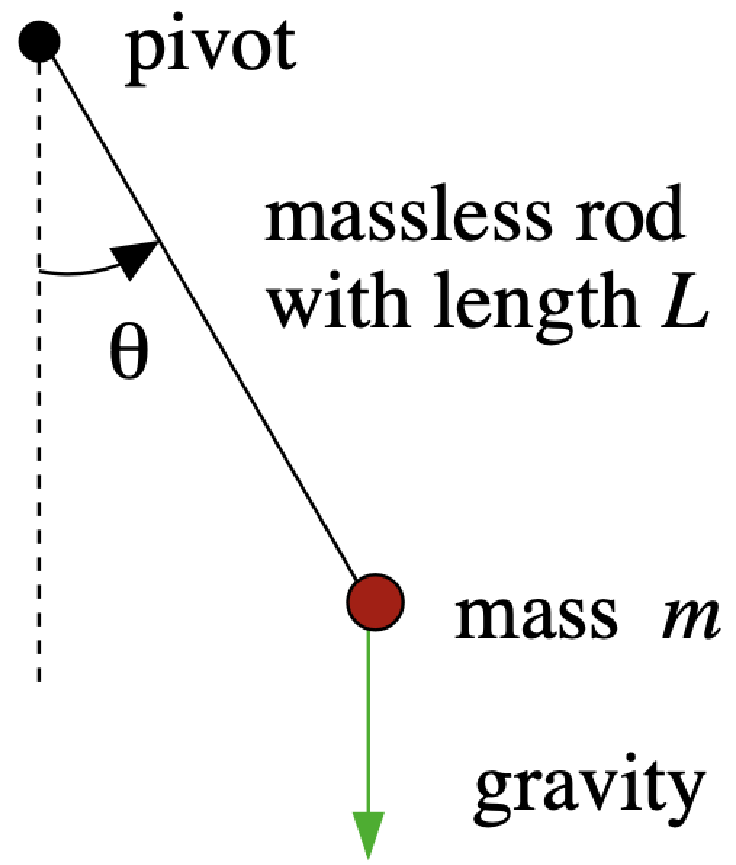
Euler VS RK4

Results!

- Percentage error in $x(1)$ after 10 steps:
    - Euler: 4.6% 😐
    - RK4: 0.000077% 😵
- With RK4, the final value is $\mathbf{2.7182797...}$ $(e = \mathbf{2.7182818...})$
    - Can we verify the local $O(\tau^5)$?
- ❓ Why is this kind of unfair?
    - 👀 😊 In each iteration, RK4 makes four RHS evaluations compared to Euler's 1...
    - *Euler*: 8 RHS calculations (8 steps): 5.6% error.
    - *RK4*: 8 RHS calculations (2 steps): 0.034% error.
- RK4 is 'smarter' at stepping forward using the same number of RHS calculations: *the increased accuracy at each step pays off!* 🐆💪🐆💪

This Week's Problem: The Simple Pendulum

- $\dfrac{d^2\theta}{dt^2} = -\dfrac{g}{L}\sin\theta.$
    - angle from the vertical θ, acceleration due to gravity g, length L.
- ❓: How might we get this second-order ODE into general form (coupled first-order)?
    - 👀 Same ole trick: introduce a new variable, ω:
      $$\frac{d\theta}{dt} = \omega\,, \qquad \frac{d\omega}{dt} = -\frac{g}{L}\sin\theta.$$
- Easy, huh? This is the *general form*, $\dfrac{d\mathbf{x}}{dt} = \mathbf{f}$
    - $\mathbf{x} = (\theta, \omega)$, $\mathbf{f} = \left(\omega, -\dfrac{g}{L}\sin\theta\right)$.



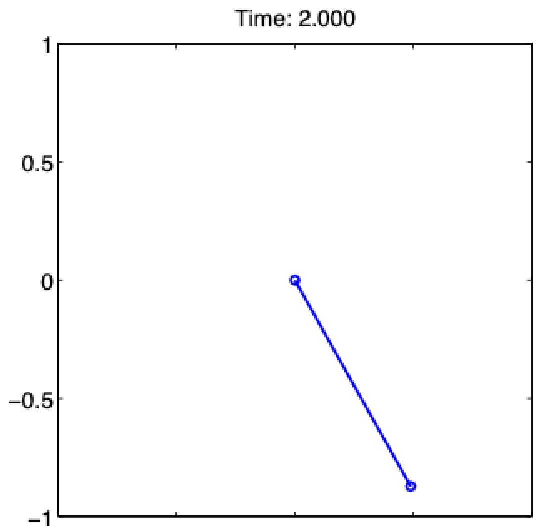
- *Initial Value Problem*: Starting from rest at an angle θ_1: $\theta(0) = \theta_1$ and $\omega(0) = \omega_1 = 0$.
- This is also a *dynamics* problem.
  - θ is angular position ($r \leftrightarrow \theta$).
  - ω is angular speed ($v \leftrightarrow \omega$).
- An *approximate* analytic solution (valid for $\theta_1 \ll 1$) is simple harmonic oscillation: $\theta = \theta_1 \cos(\Omega t)$,
  - with $\Omega^2 = \frac{g}{L}$, period $T_0 = \frac{2\pi}{\Omega} = 2\pi\sqrt{\frac{L}{g}}$, amplitude θ_1.
- For $\theta_1 \ll 1$, the period does not depend on amplitude.
  - But the general case is *nonlinear*, with no simple closed-form analytic solution 😵

Pendulum: non-dimensionalising

- θ is already non-dimensional
- For time, $\bar{t} = \dfrac{t}{t_s}$, we can set our characteristic timescale, $t_s = T_0 = 2\pi\sqrt{\dfrac{L}{g}}$:
  - $\dfrac{d^2\theta}{dt^2} = \dfrac{1}{T_0^2}\dfrac{d^2\theta}{d\bar{t}^2} = \dfrac{g}{4\pi^2 L}\dfrac{d^2\theta}{d\bar{t}^2}$.
  - $\dfrac{d^2\theta}{dt^2} = -\dfrac{g}{L}\sin\theta$ becomes $\boxed{\dfrac{d^2\theta}{d\bar{t}^2} = -4\pi^2\sin\theta}$ (no more g or L!)
- This natural timescale, T_0, yields a non-dimensional period (for $\theta \ll 1$), $\overline{T}_0 = \dfrac{T_0}{T_0} = 1$.
- In general form, $\dfrac{d\mathbf{x}}{dt} = \mathbf{f}$, we have $\mathbf{x} = \begin{pmatrix}\theta \\ \omega\end{pmatrix}$, and $\mathbf{f} = \begin{pmatrix}\omega \\ -4\pi^2\sin\theta\end{pmatrix}$.

Pendulum problem with RK4

- The general form advantage: we just write down our *RHS function* and let standard numerical methods (like RK) do the rest.
- `pend_rk4.m` applies RK4 to solve the pendulum initial value problem and animates the motion of a pendulum.
  - RHS defined in `rhs_pend.m`.
- Let's play with some different settings 😋
  - Low θ_1 (e.g., 50°) recovers oscillation?
  - ❓ predictions for $\theta_1 = 180°$...?!



❓ What's this doing?:

```
# One step of RK4
f1 = rhs_pend(x)
f2 = rhs_pend(x + 0.5*tau*f1)
f3 = rhs_pend(x + 0.5*tau*f2)
f4 = rhs_pend(x + tau*f3)
x = x + tau*(f1 + 2*f2 + 2*f3 + f4)/6
```

❓ What's this doing?:

```
func def rhs_pend(x):
    theta, omega = x
omega = x(2);
rhs(1) = omega;
rhs(2) = -4*pi^2*sin(theta);
```

❓ What's this doing?:

```
# Update the pendulum position:
# Co-ordinates of the pendulum bar
xPendArray[n+1] = [0, np.sin(x[0])]
yPendArray[n+1] = [0, -np.cos(x[0])]
```

In the tutorial

- You will play with the simple pendulum using `pend_rk4` and compare Runge–Kutta to Velocity–Verlet.
- You will adapt an implementation of RK4 to solve the Kepler problem from last week!
  - This may be your biggest challenge yet...
  - If you don't have much experience coding, you can get one-on-one help at the tutorial sessions!
- *Next week:* We move on to PDEs!