ANNOUNCEMENTS : MAKE UP CLASS
MIDTERM PRESENTATIONS

GRADES → HW 1b

BACTERIA PUZZLE

LAST TIME : INTEGRATING A FUNCTION

↳ we did just the simplest examples.

→ mostly to warm up.
now we start jogging

NOW : (ORDINARY) DIFFERENTIAL EQ.

philosophy: COMPUTATIONAL PHYSICS

solve the equations that we cannot solve by hand

PHYSICAL PRINCIPLE : Dynamics are local

laws of how system evolves

depend on nearby space & time points

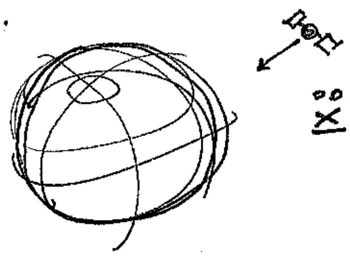
DERIVATIVES ; SMALL CHANGES IN TIME

observe : most laws are 1st or 2ND O DIFF. EQ.

one or two derivatives

why : DIMENSIONAL ANALYSIS
& PRINCIPLE OF LEAST ACTION

eg.



$$\ddot{\underline{x}} \sim \frac{-1}{\underline{x}(t)^2}$$

the acceleration depends on the position

SIMPLE ODE: want $x(t)$

$$\frac{dx}{dt} = \underbrace{\frac{2x}{t}}_{f(x,t)}$$

REMEMBER WHAT WE'D DO, PEN & PAPER:
SEPARATION OF VARIABLES

$$\frac{dx}{2x} = \frac{dt}{t}$$

definite integral

$$\frac{1}{2}\ln x - \frac{1}{2}\ln x_0 = \ln t - \ln t_0$$

$$x(t_0) = x_0$$

$$\underbrace{e^{\frac{1}{2}\ln x}}_{e^{\ln\sqrt{x}}} e^{-\frac{1}{2}\ln x_0} = \underbrace{e^{\ln t}}_{t} e^{-\ln t_0}$$

$$\sqrt{x} = e^{\ln \sqrt{x_0}/t_0}\, t$$

nb dimensional analysis

$$\boxed{x(t) = x_0 \frac{t^2}{t_0^2}} \leftarrow \frac{x_0}{t_0^2} t^2$$

CHECK: $\dot{x} = 2x_0 \, t/t_0^2 = \frac{2}{t}x(t)$ ✓

easy!

<u>HARDER</u> : $$\frac{dx}{dt} = \frac{2x}{t} + \boxed{\frac{3x^2}{t^3}}$$

btw: oughn't have x^3/t^3, right?

CAN NO LONGER SEPARATE VARIABLES

also: <u>nonlinear</u>

vs. <u>LINEAR</u> :

$$\frac{dx}{dt} = \frac{2x}{t}$$

$$\left(\frac{d}{dt} - \frac{2}{t}\right) x_1(t) = 0$$

$$\left(\frac{d}{dt} - \frac{2}{t}\right) x_2(t) = 0$$

$$\Rightarrow \left(\frac{d}{dt} - \frac{2}{t}\right) \underbrace{\left(a x_1(t) + b\, x_2(t)\right)}_{\text{lin. comb of solutions is also a solution}} = 0$$

*: for lin eq, only one "basis" solution, so this is a trivial statement.

So in a different class (P231), we talk all about milking <u>linearity</u>

the SWISS ARMY KNIFE of PHYSICS is the <u>Green's function</u>

given LINEAR DIFF. OPERATOR

$\mathcal{O} f = 0$, how to build a solution to $\mathcal{O} f = g$?

theory tool?
TAYLOR EXPAND VERY CAREFULLY (eg. Feynman Diagrams)

$\rightarrow$ BUT EVEN THIS METHOD FAILS FOR NONLINEAR EQ.

<u>computers</u>:

$$\boxed{\frac{dx}{dt} = f(x,t)}$$

if you can calculate this, then just step through.

$$\Delta x = f(x,t)\,\Delta t$$

in other words: (EULER METHOD)

$$\underbrace{x(t+\Delta t) = x(t) + \Delta t \frac{dx}{dt}}_{} + \mathcal{O}(\Delta t^2)$$

need <u>initial condition</u> $x_0(t_0)$

<u>useful tools</u>:

python array [ ]

↳ can use append( ) method
EVENTUALLY MAY WANT TO USE numpy arrays.

numpy.arange(t_{min}, t_{max}, Δt)

matplotlib.pyplot

"this is so easy" ⟶ can add more & more complicated <u>physics</u> into $f(x,t)$

let integration algorithm take care of it.

↳ just have to worry about approximation.
→ ITERATIVE ALGORITHM, errors can compound

By the way, how would you apply this to 2^{ND} Ⓞ ODE?

$$\ddot{x} = f(x,t) \leftarrow \left.\begin{matrix} x(t_0) \\ \dot{x}(t_0) \end{matrix}\right\} \text{initial data}$$

define $\dot{y}_1 = y_2 \leftarrow \dot{y}_1 = y_2 = \dot{x}$, $\dot{y}_2 = \ddot{x}$

$\dot{y}_2 = f(x_2, t) \leftarrow \ddot{x} = f(x,t)$

<u>SIMILARLY</u>:

$$\left.\begin{matrix} \dot{x} \to \dot{y}_1 = y_2 \\ \ddot{x} \to \dot{y}_2 = y_3 \\ \dddot{x} \to \dot{y}_3 = f(x,t) \end{matrix}\right\} \dddot{x} = f(x,t)$$

$\downarrow$

etc.

this is how you can prove that every system of ordinary differential eqns has a solution.

<u>ERROR ANALYSIS</u>

$$x(t+\Delta t) = x(t) + \Delta t \underset{\substack{\uparrow \\ f(x,t)}}{\frac{dx}{dt}} + \overset{O(\Delta t^2)}{\frac{1}{2}\Delta t^2} \underset{\substack{\uparrow \\ \boxed{\frac{d}{dt} f(x,t)}}}{\frac{d^2x}{dt}} + \cdots$$

have to estimate the derivative!?

HOW TO INCLUDE THIS?
... W/O DOING A LOT OF WORK?

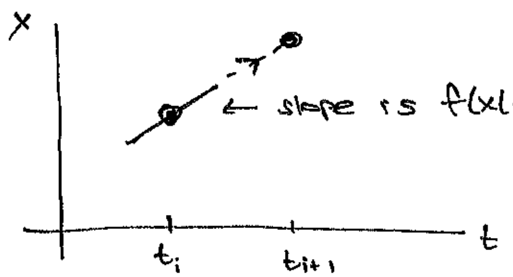
$$\sum_{k=0}^{N-1} \frac{1}{2}\Delta t^2 \frac{df}{dt} = \frac{1}{2}\Delta t \boxed{\sum \frac{df}{dt}\Delta t} \approx \frac{1}{2}\Delta t\Big(f(x(t_f), t_f) - f(x(t_0), t_0)\Big)$$

$\approx \int_{t_0}^{t_{fin}} \frac{df}{dt}\,dt$

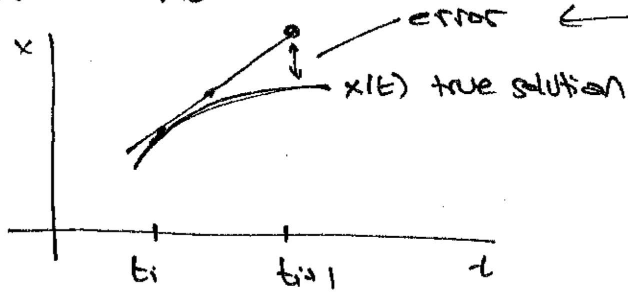
error is $O(\Delta t)$!

HERE'S A CLEVER WAY (Runge-Kutta)

EULER METHOD

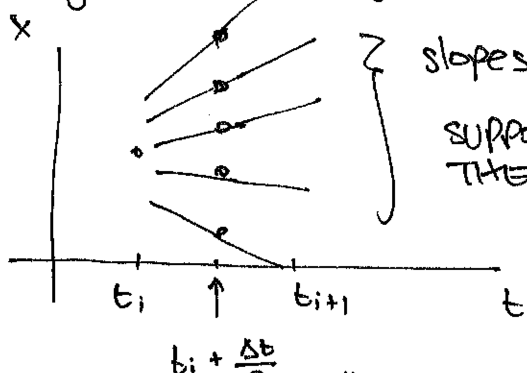


WORRY ABOUT:



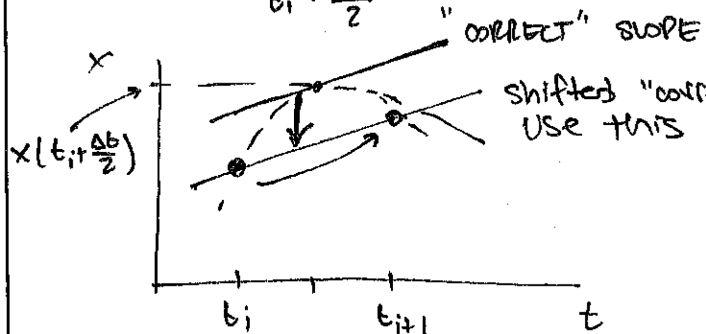
$(\Delta t)^2$ error ← accumulates ($N \sim \frac{1}{\Delta t}$) to $O(\Delta t)$ error

Runge-Kutta algorithm



slopes $f(x, t_i + \frac{\Delta t}{2})$

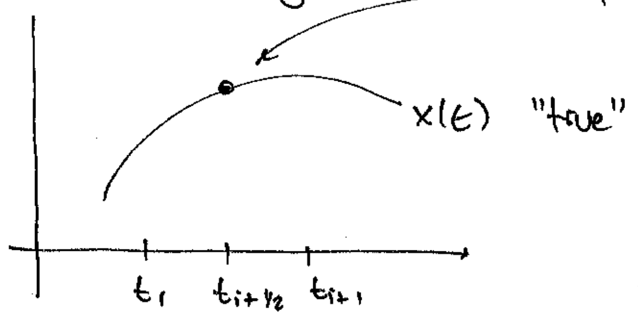
SUPPOSE WE KNEW THE "CORRECT" $x(t_i + \Delta t/2)$



shifted "correct slope"
use this to find $x(t_{i+1})$

<u>Mathematically</u>

taylor expand $x(t_i)$ & $x(t_{i+1})$ about $\boxed{t_i + \Delta t/2}$



$$x(t+\Delta t) = x(t+\Delta t/2) + \frac{\Delta t}{2}\frac{dx}{dt}\bigg|_{t+\Delta t/2} + \frac{1}{2}\left(\frac{\Delta t}{2}\right)^2 \frac{d^2x}{dt^2}\bigg| \ldots$$

$$x(t) = \underbrace{x(t+\Delta t/2)}_{\text{DUNNO THIS, DIDN'T SAMPLE}} - \frac{\Delta t}{2}\frac{dx}{dt}\bigg|_{t+\Delta t/2} + \underbrace{\frac{1}{2}\left(\frac{-\Delta t}{2}\right)^2 \frac{d^2x}{dt^2}\bigg| \ldots}_{\text{I CAN MAKE } O(\Delta t^2) \text{ CANCEL!}}$$

$$x(t+\Delta t) - x(t) = \Delta t \underbrace{\frac{dx}{dt}\bigg|_{t+\Delta t/2}}_{f\left(x\left(t+\frac{\Delta t}{2}\right),\ t+\frac{\Delta t}{2}\right)} + O(\Delta t^3)$$

We don't know this!
WELL ESTIMATE IT AS BEST WE CAN W/ WHAT WE DO KNOW (EULER)

$$x(t+\Delta t/2) = x(t) + \frac{\Delta t}{2} f(x(t), t) + \cdots$$

$$x(t+\Delta t) = x(t) + \underbrace{\Delta t\, f\Big[x(t) + \tfrac{1}{2}\underbrace{\Delta t\, f(x(t), t)}_{\equiv K_1 = \Delta t\, f(x,t)},\ t + \tfrac{\Delta t}{2}\Big]}_{\equiv K_2 = \Delta t\, f(x + \frac{1}{2}K_1,\ t + \frac{1}{2}\Delta t)} + O(\Delta t^3)$$

$$\boxed{x(t+\Delta t) = x(t) + K_2}$$

convenient.

SANITY CHECK

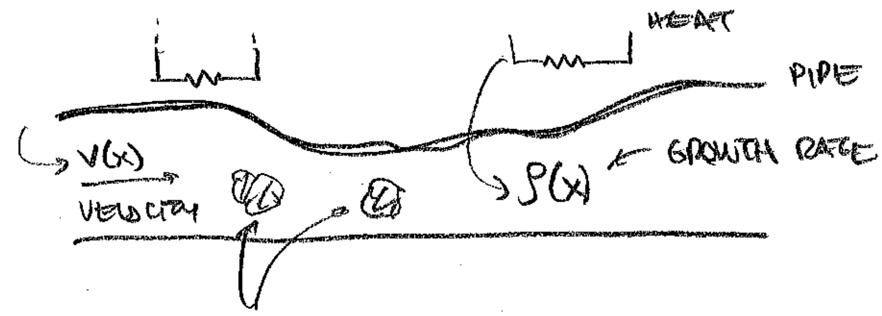
$$x(t+\Delta x) = x(t) + \Delta t\, f\Big(\underbrace{x(t) + \tfrac{1}{2}\Delta t\, f(x(t),t)}\,,\ t + \tfrac{\Delta t}{2}\Big)$$

APPROXIMATING $x(t + \Delta t/2)$

$$= \underbrace{x(t) + \tfrac{1}{2}\Delta t\, f(x,t)}_{\text{APPROX}} + \underbrace{O(\Delta t^2)}_{\text{ERROR}}$$

SO ERROR IS $O(\Delta t^3)$,
commensurate w/ procedure.

A PUZZLE:



$\longrightarrow x$

$D(x,t)$

↑ BACTERIA DENSITY

given $D(x, t_0)$, CAN YOU FIND $D(x,t)$?

HINT: PDE: $\left[\frac{\partial}{\partial t} + v(x)\frac{\partial}{\partial x} - \rho(x)\right] D(t,x) = 0$

SOLUTION: § 12.3 of An Introduction to Quantum Field Theory by Peskin & Schroeder.

Lecture 6 clean

April 24, 2018

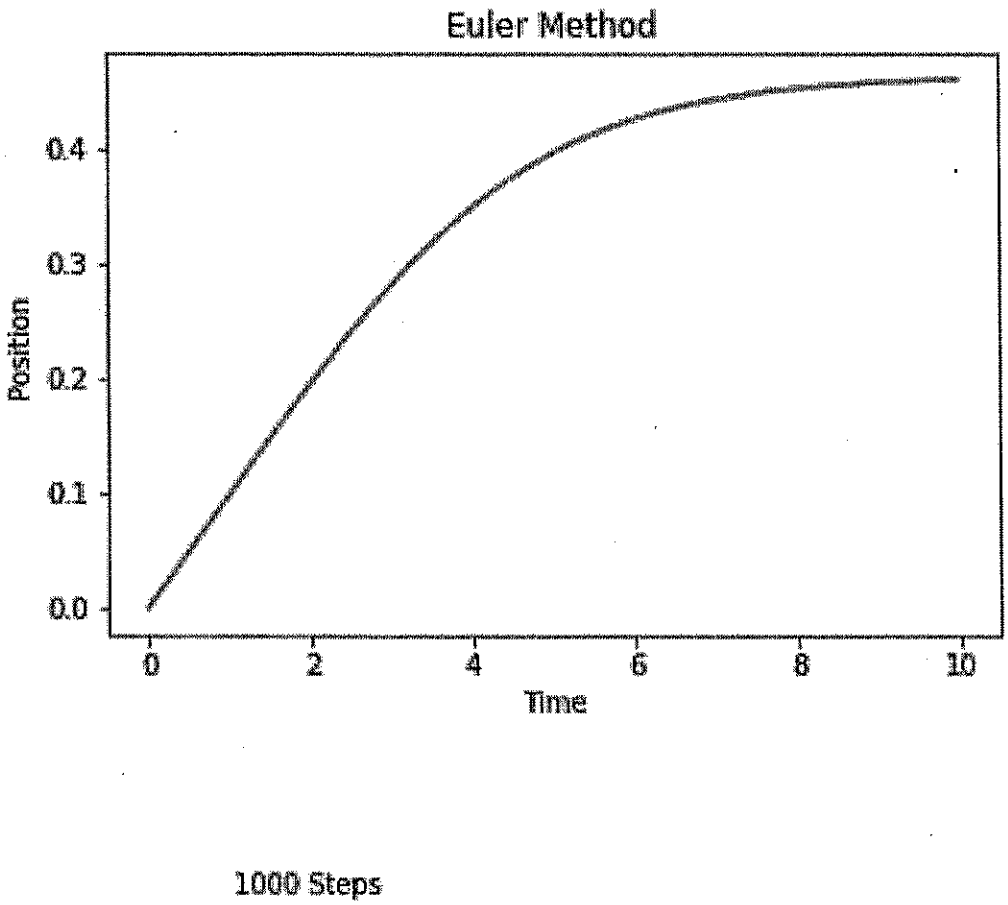
1 Lecture 6: Integration

1.1 Euler's Method, Plotting

In [15]:
```
# Homework assist

from numpy import arange


def f(x,t):
    return -x**3 + .1

# What do you expect?
# If we start at a small positive value, you eventually hit f=0

t_init = 0.0
t_fin = 10
N = 1000
delta_t = (t_fin - t_init)/N
x0 = 0

x = x0

tpoints = arange(t_init, t_fin, delta_t)
xpoints = []

for t in tpoints:
    xpoints.append(x)
    x += delta_t * f(x,t)
```

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(tpoints, xpoints)


plt.xlabel('Time')
plt.ylabel('Position')
plt.title('Euler Method')
```

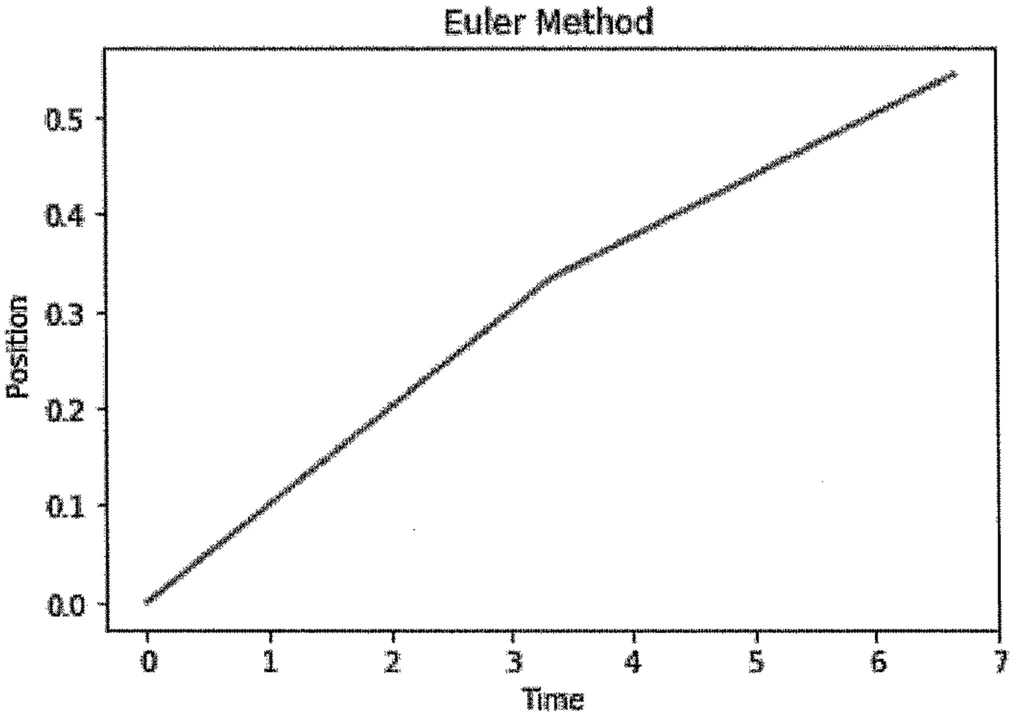
```
plt.text(1, -.25, '1000 Steps')

plt.show()
```




```
In [13]: help(arange)

Help on built-in function arange in module numpy.core.multiarray:

arange(...)
    arange([start,] stop[, step,], dtype=None)

    Return evenly spaced values within a given interval.

    Values are generated within the half-open interval ``[start, stop)``
    (in other words, the interval including `start` but excluding `stop`).
    For integer arguments the function is equivalent to the Python built-in
    `range <http://docs.python.org/lib/built-in-funcs.html>`_ function,
    but returns an ndarray rather than a list.
```

```
When using a non-integer step, such as 0.1, the results will often not
be consistent.  It is better to use ``linspace`` for these cases.

Parameters
----------
start : number, optional
    Start of interval.  The interval includes this value.  The default
    start value is 0.
stop : number
    End of interval.  The interval does not include this value, except
    in some cases where `step` is not an integer and floating point
    round-off affects the length of `out`.
step : number, optional
    Spacing between values.  For any output `out`, this is the distance
    between two adjacent values, ``out[i+1] - out[i]``.  The default
    step size is 1.  If `step` is specified as a position argument,
    `start` must also be given.
dtype : dtype
    The type of the output array.  If `dtype` is not given, infer the data
    type from the other input arguments.

Returns
-------
arange : ndarray
    Array of evenly spaced values.

    For floating point arguments, the length of the result is
    ``ceil((stop - start)/step)``.  Because of floating point overflow,
    this rule may result in the last element of `out` being greater
    than `stop`.

See Also
--------
linspace : Evenly spaced numbers with careful handling of endpoints.
ogrid: Arrays of evenly spaced numbers in N-dimensions.
mgrid: Grid-shaped arrays of evenly spaced numbers in N-dimensions.

Examples
--------
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

```
In [19]: def f(x,t):
             return -x**3 + .1

         # Runaway behavior if you overshoot

         t_init = 0.0
         t_fin = 10
         N = 3
         delta_t = (t_fin - t_init)/N
         x0 = 0

         x = x0

         tpoints_short = arange(t_init, t_fin, delta_t)
         xpoints_short = []

         for t in tpoints_short:
             xpoints_short.append(x)
             x += delta_t * f(x,t)


         %matplotlib inline
         import matplotlib.pyplot as plt
         plt.plot(tpoints, xpoints)
         plt.plot(tpoints_short, xpoints_short)


         plt.xlabel('Time')
         plt.ylabel('Position')
         plt.title('Euler Method')
         plt.text(1, -.25, '1000 Steps')

         plt.show()
```

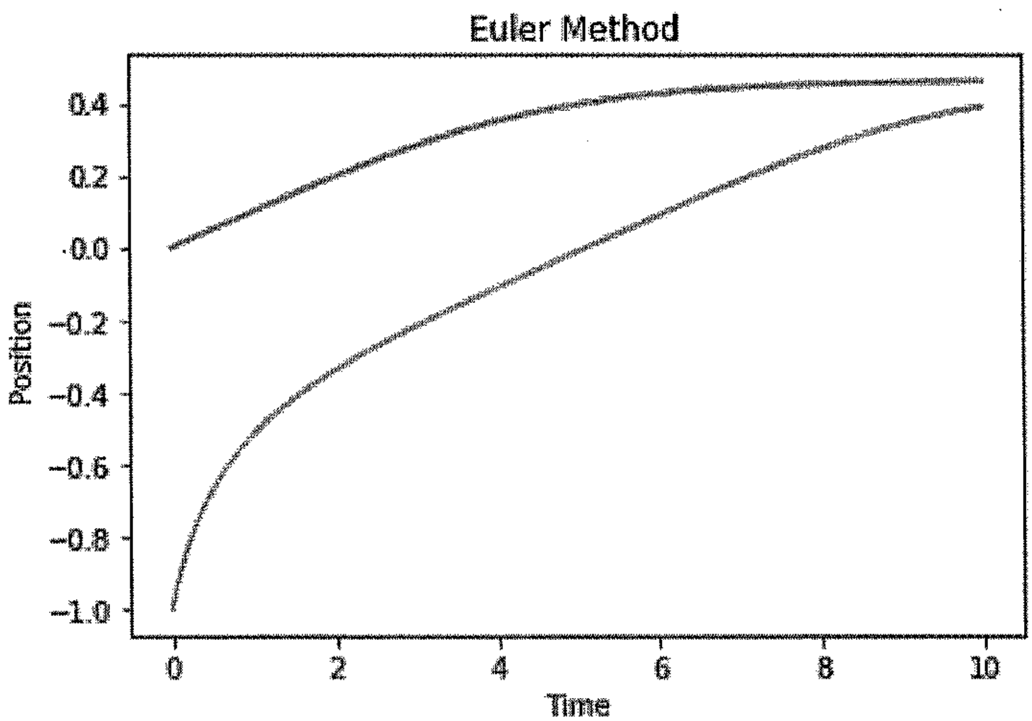


1000 Steps

In [30]:
```
# Homework assist

from numpy import arange


def f(x,t):
    return -x**3 + .1

# What do you expect?
# If we start at what about a big initial value of either sign?

t_init = 0.0
t_fin = 10
N = 1000
delta_t = (t_fin - t_init)/N
x0 = -1

x = x0

tpoints = arange(t_init, t_fin, delta_t)
```

```
        xpoints_neg = []

        for t in tpoints:
            xpoints_neg.append(x)
            x += delta_t * f(x,t)
```

```
In [31]: plt.plot(tpoints, xpoints)
         plt.plot(tpoints, xpoints_neg)


         plt.xlabel('Time')
         plt.ylabel('Position')
         plt.title('Euler Method')
         # plt.text(1, -.25, '1000 Steps')

         plt.show()
```



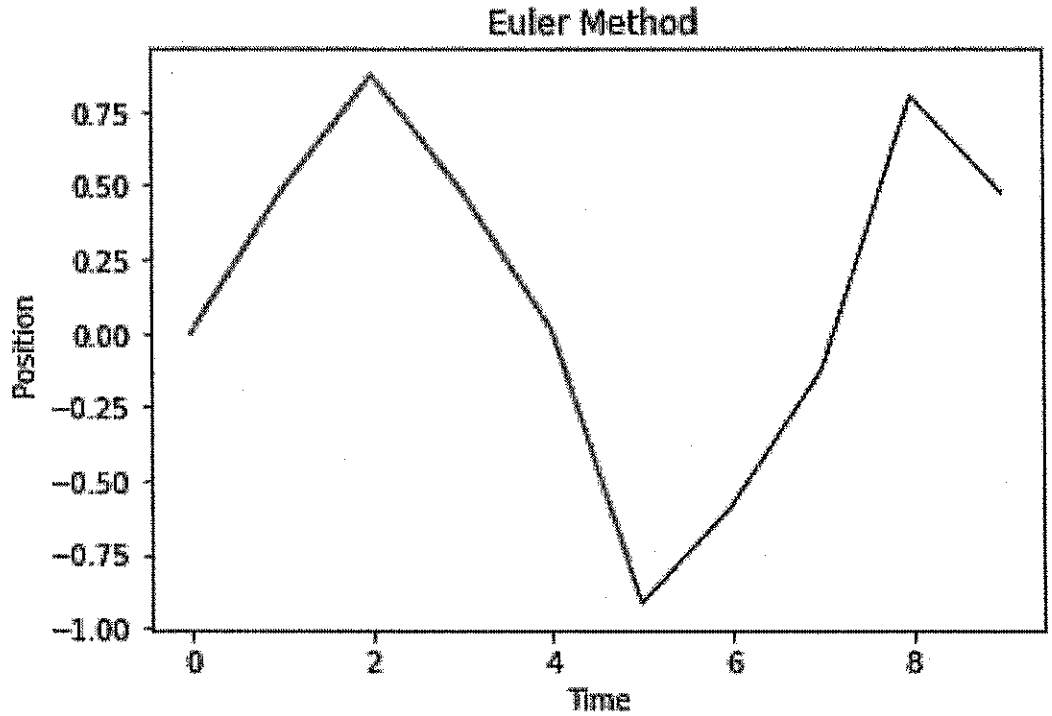
1.2 Runge Kutta

Second order, following Example 8.2 in Newman

```
In [33]: from math import sin
         from numpy import arange
```

```
def f(x,t):
    return -x**3 + sin(t)

t0 = 0
t1 = 10.
N = 10
dt = (t1-t0)/N

tpoints = arange(t0,t1,dt)
xpoints = []

# initial value
x = 0

for t in tpoints:
    xpoints.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x + 0.5*k1, t+ 0.5*dt)
    x += k2
```

```
In [34]: plt.plot(tpoints, xpoints)

         plt.xlabel('Time')
         plt.ylabel('Position')
         plt.title('Runge Kutta Method')
         # plt.text(1, -.25, '1000 Steps')

         plt.show()
```



In [35]:
```
# Try with higher resolution

t0 = 0
t1 = 10.
N = 100
dt = (t1-t0)/N

tpoints_100 = arange(t0,t1,dt)
xpoints_100 = []

# initial value
x = 0

for t in tpoints_100:
    xpoints_100.append(x)
    k1 = dt*f(x,t)
    k2 = dt*f(x + 0.5*k1, t+ 0.5*dt)
    x += k2

plt.plot(tpoints, xpoints)
plt.plot(tpoints_100, xpoints_100)

plt.xlabel('Time')
plt.ylabel('Position')
```

```
plt.title('Runge Kutta Method')
# plt.text(1, -.25, '1000 Steps')

plt.show()
```