Quiz_2

November 20, 2024

1 Quiz 2

LIS MASc
Engaging Complexity

Access this notebook on GitHub
PDF generated using nbconvert

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use("fivethirtyeight")
plt.rcParams["figure.dpi"] = 300
```

1.1 1

1.1.1 (a)

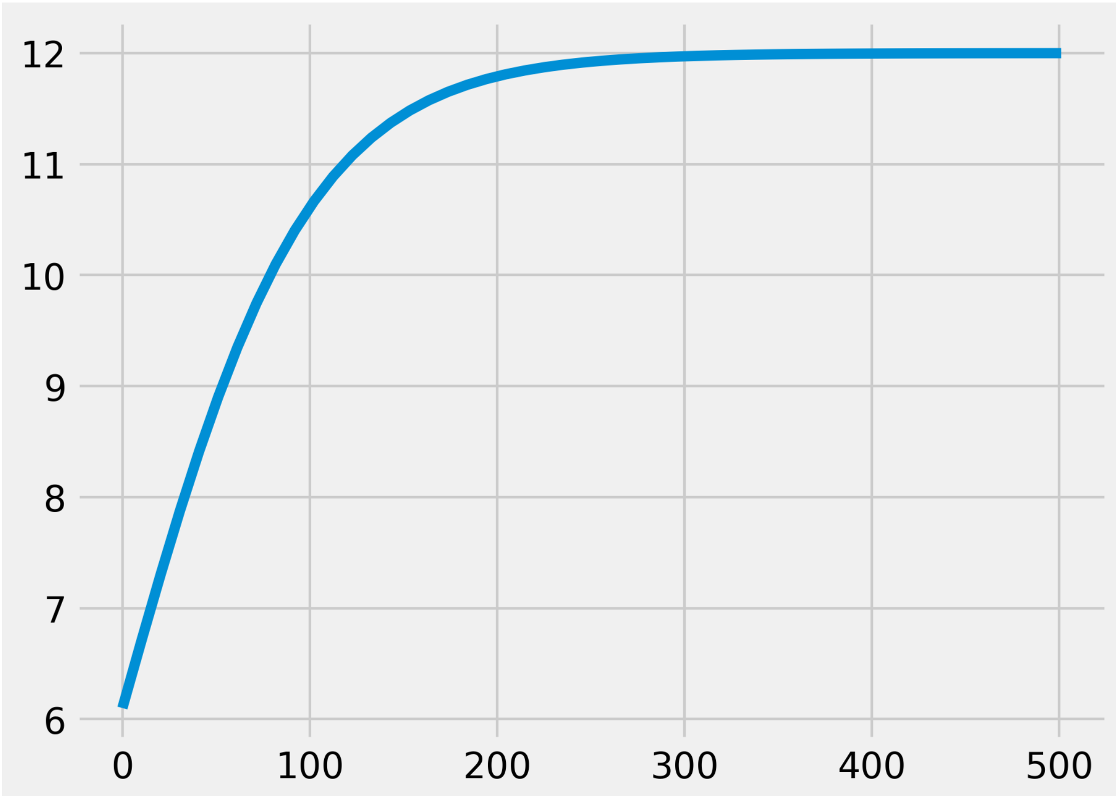
[ ]:
```
def population(t):
    return 73.2 / (6.1 + 5.9 * np.exp(-0.02 * t))

print(f'Year 2200: around {population(200):.2f} billion humans')
print(f'Year 2300: around {population(300):.2f} billion humans')
```

```
Year 2200: around 11.79 billion humans
Year 2300: around 11.97 billion humans
```

1.1.2 (b)

[14]:
```
years = np.linspace(0, 501)
plt.plot(years, population(years));
```



1.1.3 (c)

The population approaches as 12 billion as time goes on.

1.2 2

1.2.1 (a)

[15]:
```
def mass_remaining_after_radioactive_decay(time_in_days):
    return 13 * np.exp(-0.015 * time_in_days)
```

[18]:
```
mass_remaining_after_radioactive_decay(time_in_days=0)
```

[18]:
```
np.float64(13.0)
```

[19]:
```
mass_remaining_after_radioactive_decay(time_in_days=45)
```

[19]:
```
np.float64(6.61903346789814)
```

1.2.2 (b)

[21]:
```python
def mg_remaining(time_in_hours):
    return 50 * np.exp(-0.2 * time_in_hours)
```

[33]:
```python
for i in range(5):
    print(f'{i}h: {mg_remaining(i):.2f}mg remaining')
```

```
0h: 50.00mg remaining
1h: 40.94mg remaining
2h: 33.52mg remaining
3h: 27.44mg remaining
4h: 22.47mg remaining
```

If it starts at 50mg, the half life should be when the function returns 25mg, which would be between 3h and 4h.

$$D(t) = 50e^{-0.2t}$$

Assuming $D(t)$ is 25mg, we get:

$25 = 50e^{-0.2t}$

Solving for t:

$1/2 = e^{-0.2t}$

$\ln(0.5) = -0.2t$

$t = \frac{\ln(0.5)}{-0.2}$

[34]:
```python
np.log(0.5) / -0.2
```

[34]:
```
np.float64(3.465735902799726)
```

1.3 3

1.3.1 (a)

$$t = \frac{\log(N/50)}{log(2)}$$

$N = 1,000,000$

$t = \frac{\log(1{,}000{,}000/50)}{log(2)}$

$t = \frac{\log(20{,}000)}{log(2)}$

By the change of base formula

$\frac{\log(20{,}000)}{log(2)} = log_2(20,000)$

$t = log_2(20,000)$

[36]:
```
np.log2(20_000)
```

[36]: np.float64(14.287712379549449)

1.3.2 (b)

[38]:
```
def time_to_double_investment(compound_rate):
    return np.log(2) / compound_rate
```

[39]:
```
time_to_double_investment(0.06)
```

[39]: np.float64(11.552453009332423)

[40]:
```
time_to_double_investment(0.07)
```

[40]: np.float64(9.902102579427789)

[41]:
```
time_to_double_investment(0.08)
```

[41]: np.float64(8.664339756999317)

1.3.3 (c)

$t = -k\ln(1 - \frac{C}{C_0})$

$k = 0.25$

$C = 0.9C_0$

$t = -0.25\ln(1 - \frac{C}{C_0})$

$t = -0.25\ln(1 - \frac{0.9C_0}{C_0})$

$t = -0.25\ln(1 - 0.9)$

$t = -0.25\ln(0.1)$

[42]:
```
-0.25 * np.log(0.1)
```

[42]: np.float64(0.5756462732485114)

1.4 4

1.4.1 (a)

$$log_{10}(P) = log_{10}(c) - klog_{10}(W)$$

$log_{10}(P) = log_{10}(c/W^k)$

$P = c/W^k$

1.4.2 (b)

$k = 2.1$

$c = 8000$

$W = 2$

$P = c/W^k$

$P = 8000/2^{2.1}$

[44]:
```
8000/2**2.1
```

[44]: 1866.065983073615

[45]:
```
8000/10**2.1
```

[45]: 63.546258777942505