science_notebook

email@allen.tools

June 29, 2019

0.1 Newton's Second Law of Motion

$y(t) = v_0 t - \frac{1}{2} g t^2$

[74]:
```
5 * 0.6 - 0.5 * 9.81 * 0.6 ** 2
```

[74]:
```
1.2342
```

0.2 Height of an object

$y(t) = v_0 t - \frac{1}{2} g t^2$

- v0 as initial velocity of objects
- g acceleration of gravity
- t as time

With y=0 as axis of object start when t=0 at initial time.
$v_0 t - \frac{1}{2} g t^2 = t(v_0 - \frac{1}{2} g t) = 0 \Rightarrow t = 0$ or $t = \frac{v_0}{g}$

- time to move up and return to y=0, return seconds is $\frac{2v_0}{g}$ and restricted to $t \in \left[0,\ \frac{2v_0}{g}\right]$

[75]:
```
# variables for newton's second law of motion
v0 = 5
g = 9.81
t = 0.6
y = v0*t - 0.5*g*t**2
print(y)
```

```
1.2342
```

[76]:
```
# or using good pythonic naming conventions
initial_velocity = 5
acceleration_of_gravity = 9.81
TIME = 0.6
VerticalPositionOfBall = initial_velocity*TIME - \
                         0.5*acceleration_of_gravity*TIME**2
print(VerticalPositionOfBall)
```

```
1.2342
```

0.3 Integral calculation

$$\int_{-\infty}^{1} e^{-x^2} dx \, .$$

[77]:
```
from numpy import *

def integrate(f, a, b, n=100):
    """
    Integrate f from a to b
    using the Trapezoildal rule with n intervals.
    """
    x = linspace(a, b, n+1)  # coords of intervals
    h = x[1] - x[0]
    I = h*(sum(f(x)) - 0.5*(f(a) + f(b)))
    return I

# define integrand
def my_function(x):
    return exp(-x**2)

minus_infinity = -20 # aprox for minus infinity
I = integrate(my_function, minus_infinity, 1, n=1000)
print("value of integral:", I)
```

```
value of integral: 1.6330240187288536
```

[78]:
```
# Celsius-Fahrenheit Conversion
C = 21
F = (9/5)*C + 32
print(F)
```

```
69.80000000000001
```

0.4 Time to reach height of y_c

$y_c = v_0 t - \frac{1}{2} g t^2$

Quadratic equation to solve.

$\frac{1}{2} g t^2 - v_0 t + y_c = 0$ $t_1 = \left(v_0 - \sqrt{v_0^2 - 2 g y_c} \right) / g$ up $(t = t_1)$ $t_2 = \left(v_0 + \sqrt{v_0^2 - 2 g y_c} \right) / g$ down $(t = t_2 > t_1)$

[79]:
```
v0 = 5
g = 9.81
yc = 0.2
import math
t1 = (v0 - math.sqrt(v0**2 - 2 * g * yc)) / g
```

```
t2 = (v0 + math.sqrt(v0**2 - 2 * g * yc)) / g
print('At t=%g s and %g s, the height is %g m.' % (t1, t2, yc))
```

```
At t=0.0417064 s and 0.977662 s, the height is 0.2 m.
```

0.5 The hyperbolic sine function $sinh(x) = \frac{1}{2}(e^x - e^{-x})$ and other math functions with right hand sides.

[80]:
```
from math import sinh, exp, e, pi
x = 2*pi
r1 = sinh(x)
r2 = 0.5*(exp(x) - exp(-x))
r3 = 0.5*(e**x - e**(-x))
print(r1, r2, r3) # with rounding errors
```

```
267.74489404101644 267.74489404101644 267.7448940410163
```

[81]:
```
# Math functions for complex numbers
from scipy import *

from cmath import sqrt
sqrt(-1)  # complex number with cmath

from numpy.lib.scimath import sqrt
a = 1; b = 2; c = 100
r1 = (-b + sqrt(b**2 - 4*a*c))/(2*a)
r2 = (-b - sqrt(b**2 - 4*a*c))/(2*a)
print("""
t1={r1:g}
t2={r2:g}""".format(r1=r1, r2=r2))
```

```
t1=-1+9.94987j
t2=-1-9.94987j
```

[82]:
```
# Symbolic computing
from sympy import (
    symbols,  # define symbols for symbolic math
    diff,  # differentiate expressions
    integrate,  # integrate expressions
    Rational,  # define rational numbers
    lambdify,  # turn symbolic expr. into python functions
    )

# declare symbolic variables
t, v0, g = symbols('t v0 g')
```

```python
# formula
y = v0*t - Rational(1,2)*g*t**2
dydt = diff(y ,t)
print("At time", dydt)
print("acceleration:", diff(y,t,t)) # 2nd derivative
y2 = integrate(dydt, t)
print("integration of dydt wrt t", y2)

# convert to python function
v = lambdify([t, v0, g],  # arguments in v
             dydt)  # symbolic expression
print("As a function compute y = %g" % v(t=0, v0=5, g=9.81))
```

```
At time -g*t + v0
acceleration: -g
integration of dydt wrt t -g*t**2/2 + t*v0
As a function compute y = 5
```

[83]:
```python
# equation solving for expression e=0, t unknown
from sympy import solve
roots = solve(y, t)  # e is y
print("""
If y = 0 for t then t solves y for [{},{}].

""".format(
            y.subs(t, roots[0]),
            y.subs(t, roots[1])
          ) )
```

```
If y = 0 for t then t solves y for [0,0].
```

$y(t) = v_0 t - \frac{1}{2} g t^2, t \in [0, \frac{2v_0}{g}]$

[84]:
```python
# Taylor series to the order n in a variable t around the point t0
from sympy import exp, sin, cos
f = exp(t)
f.series(t, 0, 3)
f_sin = exp(sin(t))
f_sin.series(t, 0, 8)
```

[84]:
$$1 + t + \frac{t^2}{2} - \frac{t^4}{8} - \frac{t^5}{15} - \frac{t^6}{240} + \frac{t^7}{90} + O\left(t^8\right)$$

0.6 Taylor Series Polynomial to approximate functions;

$1 + t + \frac{t^2}{2} - \frac{t^4}{8} - \frac{t^5}{15} - \frac{t^6}{240} + \frac{t^7}{90} + O\left(t^8\right)$

[85]:
```python
# expanding and simplifying expressions
from sympy import simplify, expand
x, y = symbols('x y')
f = -sin(x) * sin(y) + cos(x) * cos(y)
print(f)
print(simplify(f))
print(expand(sin(x + y), trig=True))  # expand as trig funct
```

```
-sin(x)*sin(y) + cos(x)*cos(y)
cos(x + y)
sin(x)*cos(y) + sin(y)*cos(x)
```

0.7 Trajectory of an object

$$f(x) = x tan\theta - \frac{1}{2v_0^2}\frac{gx^2}{cos^2\theta} + y_0$$

[86]:
```python
# Trajectory of an object
g = 9.81       # m/s**2
v0 = 15        # km/h
theta = 60     # degree
x = 0.5        # m
y0 = 1         # m

print("""\
v0      = %.1f km/h
theta   = %d degree
y0      = %.1f m
x       = %.1f m\
""" % (v0, theta, y0, x))

from math import pi, tan, cos
v0 = v0/3.6             # km/h 1000/1 to m/s 1/60
theta = theta*pi/180    # degree to radians

y = x*tan(theta) - 1/(2*v0**2)*g*x**2/((cos(theta))**2)+y0
print("y       = %.1f m" % y)
```

```
v0      = 15.0 km/h
theta   = 60 degree
y0      = 1.0 m
x       = 0.5 m
y       = 1.6 m
```

0.8 Conversion from meters to British units

[87]:
```python
# Convert meters to british length.
meters = 640
m = symbols('m')
in_m = m/(2.54)*100
ft_m = in_m / 12
yrd_m = ft_m / 3
bm_m = yrd_m / 1760

f_in_m = lambdify([m], in_m)
f_ft_m = lambdify([m], ft_m)
f_yrd_m = lambdify([m], yrd_m)
f_bm_m = lambdify([m], bm_m)

print("""
Given {meters:g} meters conversions for;
inches are {inches:.2f} in
feet are {feet:.2f} ft
yards are {yards:.2f} yd
miles are {miles:.3f} m
""".format(meters=meters,
           inches=f_in_m(meters),
           feet=f_ft_m(meters),
           yards=f_yrd_m(meters),
           miles=f_bm_m(meters)))
```

```
Given 640 meters conversions for;
inches are 25196.85 in
feet are 2099.74 ft
yards are 699.91 yd
miles are 0.398 m
```

0.9 Gaussian function

$$f(x) = \frac{1}{\sqrt{2\pi}s}\exp\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]$$

[88]:
```python
from sympy import pi, exp, sqrt, symbols, lambdify

s, x, m = symbols("s x m")
y = 1/ (sqrt(2*pi)*s) * exp(-0.5*((x-m)/s)**2)
gaus_d = lambdify([m, s, x], y)
gaus_d(m = 0, s = 2, x = 1)
```

[88]:
```
0.1760326633821498
```

0.10 Drag force due to air resistance on an object as the expression;

$$F_d = \frac{1}{2} C_D \varrho A V^2$$

Where * C_D drag coefficient (based on roughness and shape) * As 0.4 * ϱ is air density * Air density of air is ϱ = 1.2 kg/m^{-3} * V is velocity of the object * A is the cross-sectional area (normal to the velocity direction) * $A = \pi a^2$ for an object with a radius a * a = 11 cm

Gravity Force on an object with mass m is $F_g = mg$ Where * g = 9.81 m/s^{-2} * mass = 0.43kg

F_d and F_g results in a difference relationship between air resistance versus gravity at impact time

$\frac{kg}{m^{-3}}$ $\frac{m}{s^{-2}}$

[89]:
```
from sympy import (Rational, lambdify, symbols, pi)

g = 9.81  # gravity in m/s**(-2)
air_density = 1.2  # kg/m**(-3)
a = 11  # radius in cm
x_area = pi * a**2  # cross-sectional area
m = 0.43  # mass in kg
Fg = m * g  # gravity force
high_velocity = 120 / 3.6  # impact velocity in km/h
low_velocity = 30 / 3.6  # impact velocity in km/h

Cd, Q, A, V = symbols("Cd Q A V")
y = Rational(1, 2) * Cd * Q * A * V**2
drag_force = lambdify([Cd, Q, A, V], y)

Fd_low_impact = drag_force(Cd=0.4,
                       Q=air_density,
                       A=x_area,
                       V=low_velocity)

Fd_high_impact = drag_force(Cd=0.4,
                       Q=air_density,
                       A=x_area,
                       V=high_velocity)

print("ratio of drag force=%.1f and gravity force=%.1f: %.1f" % \
      (Fd_low_impact, Fg, float(Fd_low_impact/Fg)))

print("ratio of drag force=%.1f and gravity force=%.1f: %.1f" % \
      (Fd_high_impact, Fg, float(Fd_high_impact/Fg)))
```

```
ratio of drag force=6335.5 and gravity force=4.2: 1501.9
ratio of drag force=101368.7 and gravity force=4.2: 24030.7
```

$$t = \frac{M^{2/3} c \rho^{1/3}}{K \pi^2 (4\pi/3)^{2/3}} \log\left[0.76 \frac{(T_o - T_w)}{-T_w + T_y}\right]$$

[90]:
```python
def critical_temp(init_temp=4, final_temp=70, water_temp=100,
                  mass=47, density=1.038, heat_capacity=3.7,
                  thermal_conductivity=5.4*10**-3):
    """
    Heating to a temperature with prevention to exceeding critical
    points. Be defining critial temperature points based on
    composition, e.g., 63 degrees celcius outter and 70 degrees
    celcius inner we can express temperature and time as a
    function.


    Calculates the time for the center critical temp as a function
        of temperature of applied heat where exceeding passes a critical point.

    t = (M**(2/3)*c*rho**(1/3)/(K*pi**2*(4*pi/3)**(2/3)))*(ln(0.76*((To-Tw)/
↪(Ty-Tw))))

    Arguments:
        init_temp: initial temperature in C of object e.g., 4, 20
        final_temp: desired temperature in C of object e.g., 70
        water_temp: temp in C for boiling water as a conductive fluid e.g., 100
        mass: Mass in grams of an object, e.g., small: 47, large: 67
        density: rho in g cm**-3 of the object e.g., 1.038
        heat_capacity: c in J g**-1 K-1 e.g., 3.7
        thermal_conductivity: in W cm**-1 K**-1 e.g., 5.4*10**-3
    Returns: Time as a float in seconds to reach temperature Ty.
    """
    from sympy import symbols
    from sympy import lambdify
    from sympy import sympify
    from numpy import pi
    from math import log as ln # using ln to represent natural log

    # using non-pythonic math notation create variables
    M, c, rho, K, To, Tw, Ty = symbols("M c rho K To Tw Ty")
    # writing out the formula
    t = sympify('(M**(2/3)*c*rho**(1/3)/(K*pi**2*(4*pi/3)**(2/3)))*(ln(0.
↪76*((To-Tw)/(Ty-Tw))))')
    # using symbolic formula representation to create a function
    time_for_Ty = lambdify([M, c, rho, K, To, Tw, Ty], t)
    # return the computed value
    return time_for_Ty(M=mass, c=heat_capacity, rho=density,␣
↪K=thermal_conductivity,
                       To=init_temp, Tw=water_temp, Ty=final_temp)
```

[91]:
```python
critical_temp()
```

[91]: 313.09454902221626

[92]:
```
critical_temp(init_temp=20)
```

[92]: 248.86253747844728

[93]:
```
critical_temp(mass=70)
```

[93]: 408.3278117759983

[94]:
```
critical_temp(init_temp=20, mass=70)
```

[94]: 324.55849416396666

0.11 Newtons second law of motion in direction x and y, aka accelerations:

$F_x = ma_x$ is the sum of force, `m*a_x (mass * acceleration)`

$a_x = \frac{d^2x}{dt^2}$, `ax = (d**2*x)/(d*t**2)`

With gravity from F_x as 0 as $x(t)$ is in the horizontal position at time t

$F_y = ma_y$ is the sum of force, `m*a_y`

$a_y = \frac{d^2y}{dt^2}$, `ay = (d**2*y)/(d*t**2)`

With gravity from F_y as $-mg$ since $y(t)$ is in the veritcal postion at time t

Let coodinate $(x(t), y(t))$ be horizontal and verical positions to time t then we can integrate Newton's two components, $(x(t), t(t))$ using the second law twice with initial velocity and position with respect to t

$\frac{d}{dt}x(0) = v_0 cos\theta$

$\frac{d}{dt}y(0) = v_0 sin\theta$

$x(0) = 0$

$y(0) = y_0$

```
"""
Derive the trajectory of an object from basic physics.
    Newtons second law of motion in direction x and y, aka accelerations:
        F_x = ma_x is the sum of force, m*a_x (mass * acceleration)
        F_y = ma_y is the sum of force, m*a_y
    let coordinates (x(t), y(t)) be position horizontal and vertical to time t
    relations between acceleration, velocity, and position are derivatives of t

    $a_x = \frac {d^{2}x}{dt^{2}}$, ax = (d**2*x)/(d*t**2)
    $a_y = \frac {d^{2}y}{dt^{2}}$  ay = (d**2*y)/(d*t**2)

    With gravity and F_x = 0 and F_y = -mg

    integrate Newton's the two components, (x(t), y(t)) second law twice with
    initial velocity and position wrt t

    $\frac{d}{dt}x(0)=v_0 cos\theta$
    $\frac{d}{dt}y(0)=v_0 sin\theta$
    $x(0) = 0$
    $y(0) = y_0$

    Derivative(t)x(0) = v0*cos(theta) ; x(0) = 0
```

```
    Derivative(t)y(0) = v0*sin(theta) ; y(0) = y0

    from sympy import *
    diff(Symbol(v0)*cos(Symbol(theta)))
    diff(Symbol(v0)*sin(Symbol(theta)))

theta: some angle, e.g, pi/2 or 90

Return: relationship between x and y

# the expression for x(t) and y(t)



# if theta = pi/2 then motion is vertical e.g., the y position formula

# if t = 0, or is eliminated then x and y are the object coordinates

"""
```

there isn't any code to this, it just looks at newtons second law of motion

0.12 Sine function as a polynomial

$$sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots$$

[96]:
```python
x, N, k, sign = 1.2, 25, 1, 1.0
s = x
import math

while k < N:
    sign = - sign
    k = k + 2
    term = sign*x**x/math.factorial(k)
    s = s + term

print("sin(%g) = %g (approximation with %d terms)" % (x, s, N))
```

```
sin(1.2) = 1.0027 (approximation with 25 terms)
```

0.13 Print table using an approximate Fahrenheit-Celcius conversiion.

For the approximate formula $C \approx \hat{C} = (F - 30)/2$ farenheit to celcius conversions are calculated. Adds a third to conversation_table with an approximate value $\hat{C}$.

[97]:
```python
F=0; step=10; end=100 # declare
print('------------------')
while F <= end:
```

```
    C = F/(9.0/5) - 32
    C_approx = (F-30)/2
    print("{:>3} {:>5.1f} {:>3.0f}".format(F, C, C_approx))
    F = F + step
print('------------------')
```

```
------------------
  0 -32.0 -15
 10 -26.4 -10
 20 -20.9  -5
 30 -15.3   0
 40  -9.8   5
 50  -4.2  10
 60   1.3  15
 70   6.9  20
 80  12.4  25
 90  18.0  30
100  23.6  35
------------------
```

0.14 Create sequences of odds from 1 to any number.

[98]:
```
n = 9 # specify any number
c = 1
while 1 <= n:
    if c%2 == 1:
        print(c)

    c += 1
    n -= 1
```

```
1
3
5
7
9
```

0.15 Compute energy levels in an atom

Compute the n-th energy level for an electron in an atom, e.g., Hydrogen:

$$E_n = -\frac{m_e e^4}{8\epsilon_0^2 h^2} \cdot \frac{1}{n^2}$$

where:
$m_e = 9.1094 \cdot 10^{-31}$kg is the electron mass
$e = 1.6022 \cdot 10^{-19}$C is the elementary charge

$\epsilon_0 = 8.8542 \cdot 10^{-12} \mathrm{s}^2\mathrm{kg}^{-1}\mathrm{m}^{-3}$ is electrical permittivity of vacuum
$h = 6.6261 \cdot 10^{-34}$Js
Calculates energy level E_n for $n = 1, ..., 20$

[109]:
```python
# Symbolic computing
from sympy import (
    symbols,  # define symbols for symbolic math
    lambdify,  # turn symbolic expr. into python functions
    )

# declare symbolic variables
m_e, e, epsilon_0, h, n = symbols('m_e e epsilon_0 h n')
# formula
En = -(m_e*e**4)/(8*epsilon_0*h**2)*(1/n**2)

# convert to python function
y = lambdify([m_e, e, epsilon_0, h, n],  # arguments in En
             En)  # symbolic expression

def compute_atomic_energy(m_e=9.094E-34,
                          e=1.6022E-19,
                          epsilon_0=9.9542E-12,
                          h=6.6261E-34):

    En = 0 # energy level of an atom
    for n in range(1, 20): # Compute for 1,...,20
        En += y(m_e, e, epsilon_0, h, n)

    return En
```

[110]:
```python
compute_atomic_energy()
```

[110]: -2.7315307541142e-32

[ ]: