Using Python for Mathematics

Solving ODE's

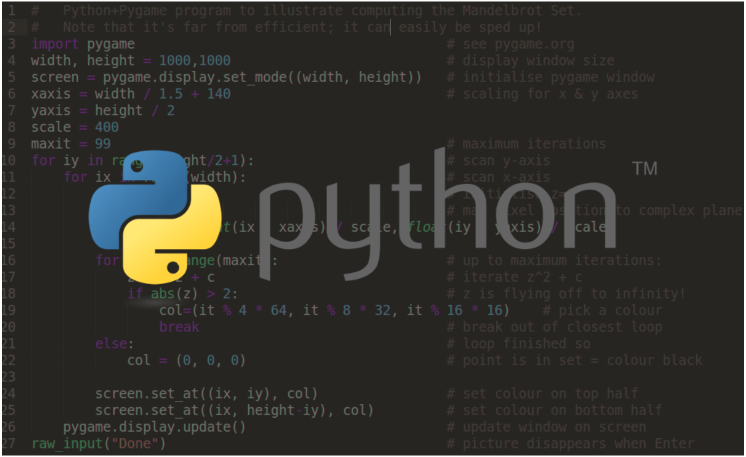
Michael Turner

12th December 2017

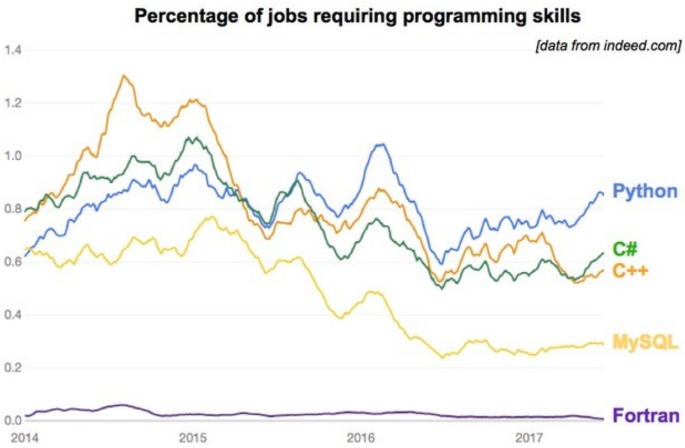
Presentation Outcomes:

To see the different ways in which Python can be applied to real mathematical problems and to demonstrate its power when used with situations where an analytical solution would be too difficult or impossible to find.

What Is Python?



Inspiration



Simple Problems

Integrating

If we consider the problem: $u'(t) = t^2 + 4$, $u(0) = 1$

Simple Problems

Integrating

If we consider the problem: $u'(t) = t^2 + 4$, $u(0) = 1$

This equation can be solved by just integrating both sides w.r.t x:

Simple Problems

Integrating

If we consider the problem: $u'(t) = t^2 + 4$, $u(0) = 1$

This equation can be solved by just integrating both sides w.r.t x:

$$u(t) = \int_0^t \left(\tau^2 + 4\right)\, d\tau + u(0) = \frac{t^3}{3} + 4t + u(0)$$

Simple Problems

Integrating

If we consider the problem: $u'(t) = t^2 + 4$, $u(0) = 1$

This equation can be solved by just integrating both sides w.r.t x:

$$u(t) = \int_0^t \left(\tau^2 + 4\right)\, d\tau + u(0) = \frac{t^3}{3} + 4t + u(0)$$

A more general form would be:

$$u(t) = u(0) + \int_0^T f(\tau)\, d\tau$$

Simple Problems

Integrating

If we consider the problem: $u'(t) = t^2 + 4$, $u(0) = 1$

This equation can be solved by just integrating both sides w.r.t x:

$$u(t) = \int_0^t \left(\tau^2 + 4\right)\, d\tau + u(0) = \frac{t^3}{3} + 4t + u(0)$$

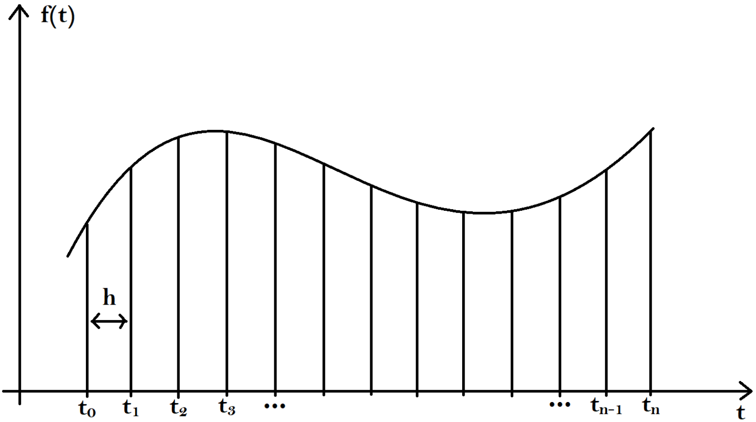
A more general form would be:

$$u(t) = u(0) + \int_0^T f(\tau)\, d\tau$$

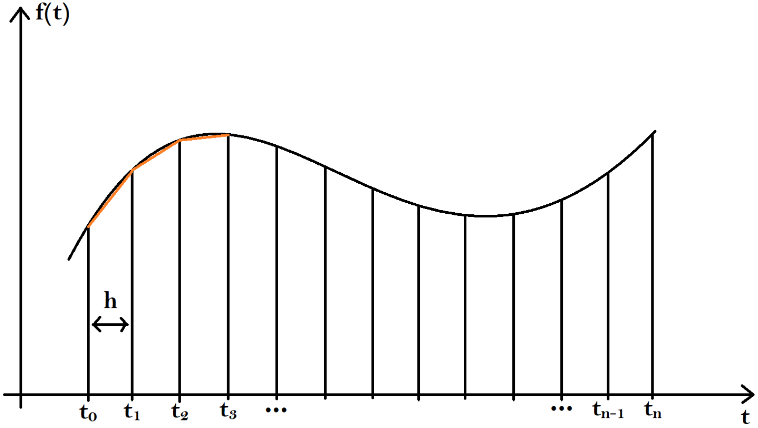
Trapezium Rule:

$$u(t) \approx u(0) + \frac{h}{2}\left[y_0 + 2\sum_{k=1}^{n-1} y_k + y_n\right]$$

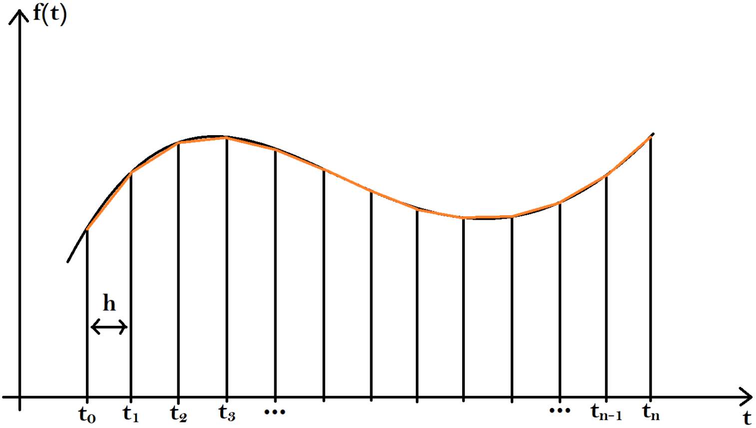
Integrating



Integrating



Integrating



Integrating

The Python code `The_Simplest_Case.py` finds a numerical solution where function f, time t, initial condition u_0 and time-steps n are inputs:

Integrating

The Python code The_Simplest_Case.py finds a numerical solution where function f, time t, initial condition u_0 and time-steps n are inputs:

```
import numpy as np

def f(t):
    return t*np.exp(t*t)

def integrate(f, T, n, u0):
    h = T/float(n)
    t = np.linspace(0, T, n+1) # From 0 to T with n strips
    I = f(t[0])
    for k in range(1, n):  # Goes from 1 to n-1
        I += 2*f(t[k]) # Sum of each element in f from 1 to n-1
    I += f(t[-1]) # Previous sum plus last element n
    I *= (h/2) # Multiply sum by h/2
    I += u0 # Previous sum plus the initial condition
    return float(I)
```

Simple Problems

The code can solve this equation: $u'(t) = te^{t^2}$, $u(0) = 0$ at time $T = 2$ with $n = 100, 1000$:

Simple Problems

The code can solve this equation: $u'(t) = te^{t^2}$, $u(0) = 0$ at time $T = 2$ with $n = 100, 1000$:

```
T = 2 # Set data
u0 = 0
n = 100
print("Numerical Solution of t*exp(t*t) is:",
integrate(f, T, n, u0))
```

Simple Problems

The code can solve this equation: $u'(t) = te^{t^2}$, $u(0) = 0$ at time $T = 2$ with $n = 100, 1000$:

```
T = 2 # Set data
u0 = 0
n = 100
print("Numerical Solution of t*exp(t*t) is:",
integrate(f, T, n, u0))
```

Terminal

```
Numerical Solution of t*exp(t*t) is:  26.8154183398632
```

Terminal

```
Numerical Solution of t*exp(t*t) is:  26.79923847740996
```

The exact solution is : $u(t) = \frac{1}{2}e^{2^2} + \frac{1}{2} \approx 27.7991$

Why Bother?

Scalar ODE's

Abstract Form

We'll work with ODE's written as: $u'(t) = f(u(t), t)$, $u(0) = u_0$

Scalar ODE's

Abstract Form

We'll work with ODE's written as: $u'(t) = f(u(t), t)$, $u(0) = u_0$

Assume that $u(t)$ is a **scalar function**.

Scalar ODE's

Abstract Form

We'll work with ODE's written as: $u'(t) = f(u(t), t)$, $u(0) = u_0$

Assume that $u(t)$ is a **scalar function**. Then we refer to the above as a **scalar differential equation**.

Scalar ODE's

Abstract Form

We'll work with ODE's written as: $u'(t) = f(u(t), t)$, $u(0) = u_0$

Assume that $u(t)$ is a **scalar function**. Then we refer to the above as a **scalar differential equation**.

Forward Euler Method

$$u_{k+1} = u_k + \Delta t f(u_k, t_k), \quad \text{where} \quad u(t_i) = u_i,\ i = 1, 2, \ldots, n.$$

Scalar ODE's

Abstract Form

We'll work with ODE's written as: $u'(t) = f(u(t), t)$, $u(0) = u_0$

Assume that $u(t)$ is a **scalar function**. Then we refer to the above as a **scalar differential equation**.

Forward Euler Method

$$u_{k+1} = u_k + \Delta t f(u_k, t_k), \quad \text{where} \quad u(t_i) = u_i,\ i = 1, 2, \dots, n.$$

Implementation

Example: $u' = u$, $u_0 = 1$, $\Delta t = 0.1$, $n = 10$

We can implement the **Forward Euler** method in a function in Python.

Scalar ODE's

```
import matplotlib.pyplot as plt
import numpy as np

def f(t):
    return u # f(u[t[k]], t[k])

def FEM(f, T, n, u0):
    t = np.zeros(n+1)
    u = np.zeros(n+1)
    u[0] = U0
    t[0] = 0
    dt = T/float(n)
    for k in range(0, n):
        t[k+1] = t[k] + dt
        u[k+1] = u[k] + dt*f(u[k], t[k])
    return u, t
```

Link to video

Other Methods

Heun's Method

$$
\begin{aligned}
u_* &= u_k + \Delta t f(u_k, t_k) \\
u_{k+1} &= u_k + \frac{1}{2}\Delta t f(u_k, t_k) + \frac{1}{2}\Delta t f(u_*, t_{k+1})
\end{aligned}
$$

Other Methods

Heun's Method

$$u_* = u_k + \Delta t f(u_k, t_k)$$
$$u_{k+1} = u_k + \frac{1}{2}\Delta t f(u_k, t_k) + \frac{1}{2}\Delta t f(u_*, t_{k+1})$$

4th-Order Runge-Kutta Method

$$u_{k+1} = u_k + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right)$$
$$K_1 = \Delta t f(u_k, t_k)$$
$$K_2 = \Delta t f\left(u_k + \frac{1}{2}K_1, t_k + \frac{1}{2}\Delta t\right)$$
$$K_3 = \Delta t f\left(u_k + \frac{1}{2}K_2, t_k + \frac{1}{2}\Delta t\right)$$
$$K_4 = \Delta t f(u_k + K_3, t_k + \Delta t), \text{ where } \Delta t = t_{k+1} - t_k$$

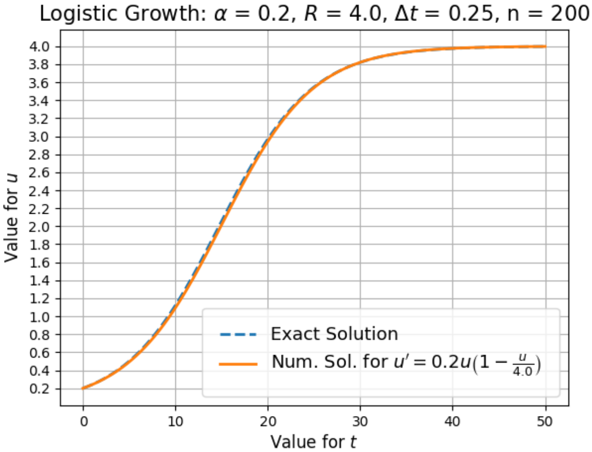
Practical Applications

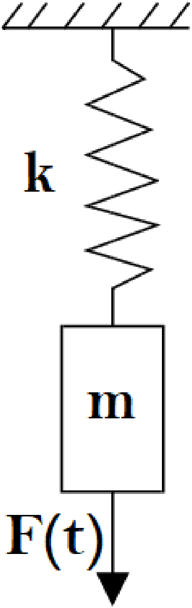
Logistic Growth

$$u'(t) = \alpha u(t) \left(1 - \frac{u(t)}{R}\right)$$

Logistic Growth

$$u'(t) = \alpha u(t)\left(1 - \frac{u(t)}{R}\right)$$




k
m
F(t)

System of ODE's
An oscillating spring-mass is governed by a second order ODE:

$$mu'' + \beta u' + ku = F(t), \quad u(0) = u_0,\, u'(0) = 0$$

System of ODE's
An oscillating spring-mass is governed by a second order ODE:

$$mu'' + \beta u' + ku = F(t), \quad u(0) = u_0,\, u'(0) = 0$$

$$u^{(0)}(t) = u(t), \quad u^{(1)}(t) = u'(t)$$

Practical Applications

System of ODE's

An oscillating spring-mass is governed by a second order ODE:

$$mu'' + \beta u' + ku = F(t), \quad u(0) = u_0,\, u'(0) = 0$$

$$u^{(0)}(t) = u(t), \quad u^{(1)}(t) = u'(t)$$

$$\frac{d}{dt}u^{(0)}(t) = u^{(1)}(t), \quad \frac{d}{dt}u^{(1)}(t) = \frac{1}{m}\left(F(t) - \beta u^{(1)} - ku^{(0)}\right)$$

System of ODE's
An oscillating spring-mass is governed by a second order ODE:

$$mu'' + \beta u' + ku = F(t), \quad u(0) = u_0,\, u'(0) = 0$$

$$u^{(0)}(t) = u(t), \quad u^{(1)}(t) = u'(t)$$

$$\frac{d}{dt}u^{(0)}(t) = u^{(1)}(t), \quad \frac{d}{dt}u^{(1)}(t) = \frac{1}{m}\left(F(t) - \beta u^{(1)} - ku^{(0)}\right)$$

$$u(t) = \left(u^{(0)}(t), u^{(1)}(t)\right)$$

$$f(t,u) = \left(u^{(1)}, \frac{1}{m}\left(F(t) - \beta u^{(1)} - ku^{(0)}\right)\right)$$

System of ODE's
An oscillating spring-mass is governed by a second order ODE:

$$mu'' + \beta u' + ku = F(t), \quad u(0) = u_0,\, u'(0) = 0$$

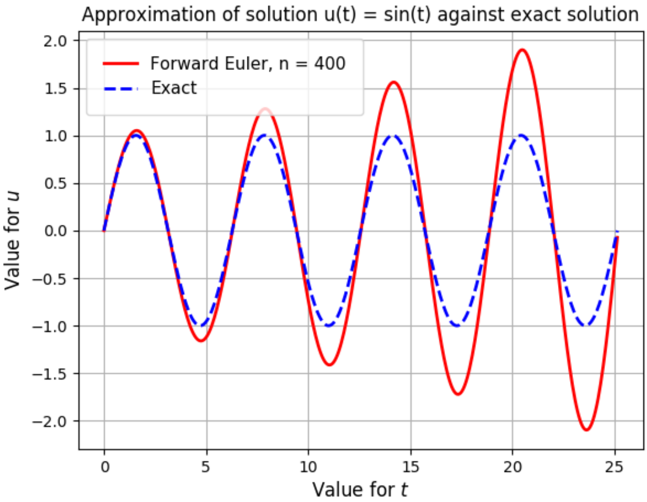
$$u^{(0)}(t) = u(t), \quad u^{(1)}(t) = u'(t)$$

$$\frac{d}{dt}u^{(0)}(t) = u^{(1)}(t), \quad \frac{d}{dt}u^{(1)}(t) = \frac{1}{m}\left(F(t) - \beta u^{(1)} - ku^{(0)}\right)$$

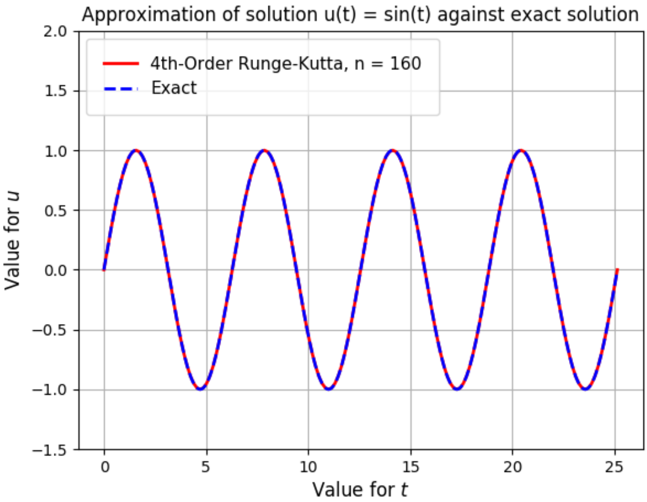
$$u(t) = \left(u^{(0)}(t), u^{(1)}(t)\right)$$

$$f(t, u) = \left(u^{(1)}, \frac{1}{m}\left(F(t) - \beta u^{(1)} - ku^{(0)}\right)\right)$$

Example
We shall test the code with another something simple:
$u'' + u = 0$, $u(0) = 0$, $u'(0) = 1$ with solution $u(t) = \sin(t)$

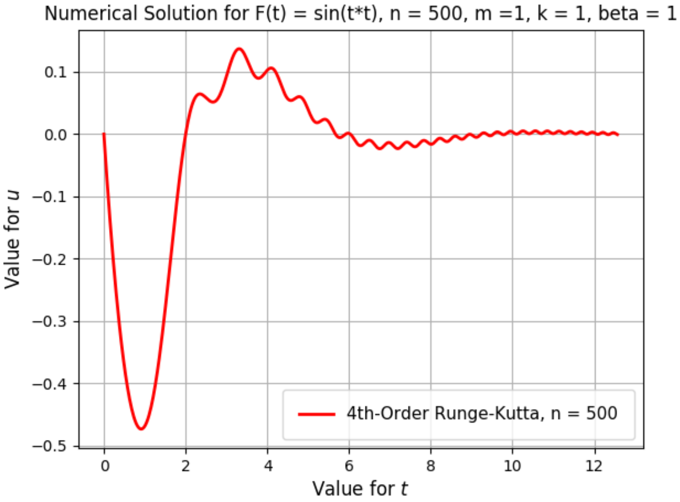

Approximation of solution u(t) = sin(t) against exact solution
Forward Euler, n = 400
Exact
Value for u
Value for t
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
5
10
15
20
25

Practical Applications



Advanced Problem

We have: $u'' + u' + u = \sin(t^2)$, $u(0) = 0$, $u'(0) = -1$



The Plan

- I would like to take what I've shown you here today to the deepest level. To continue working with ODE's with Python to learn even more about how we can use it to our advantage by the end of the semester.
- I would then want to begin looking at another topic for the second semester which is about discrete calculus. Finding ways for a computer to differentiate a function, approximating a function. Finding the best way to build a tool that can be used efficiently and accurately.

Conclusion

Summary

- This has all been done to see just how much we can accomplish with Python. How well it can be used for the mathematical problems we face in the real world.
- So far we have only scraped the surface. We have used Python to give us an answer to situations that would take hours to calculate numerically by hand in seconds.
- The world is becoming more reliant on technology than ever before. Having the skills to apply what we know through a computer language is in high demand. If you can't beat them then join them.

References

Book

Langtangen, H. P. (2012) *A Primer on Scientific Programming with Python*. Midtown Manhattan: Springer International Publishing.

Images

Geekboots, accessed December 2nd 2017 (https://www.geekboots.com/story/what-is-python-programming-and-why-you-should-learn-it)

Deviant Art, htf-lover12, accessed December 2nd 2017 (https://htf-lover12.deviantart.com/art/Keep-calm-doctor-who-trust-me-317868064)

Siri Chongchitnan, accessed December 2nd 2017