Hypothesis

Team: September

Team Members:
Omnia Mohyee
Reham Hamdy
Zeinab Rabie
Hagar Haytham

Introduction

- Hypothesis is a Python library for creating unit tests which are simpler to write and more powerful when run, finding edge cases in your code you wouldn’t have thought to look for. It is stable, powerful and easy to add to any existing test suite.

- Property-based testing.

Example on property-based testing.

```python
@given(s=text())
@example(s="")
def test_decode_inverts_encode(s):
    assert decode(encode(s)) == s
```

Getting Started with Hypothesis

The main important feature of hypothesis is using **strategies** for **generating different inputs** meeting some input **specification**.

Strategy

Object with **methods** that describe **how to generate** and simplify certain kinds of values.

Strategy

Generating inputs of generic data types (integer , float , char , etc ….)

```
initial_y=draw (st.floats(min_value = 0 ,max_value = 10))

intervals=draw(st.integers(min_value = 2 ,max_value = 10))
```

Strategy - Lists

```
Xs = draw(st.lists(st.decimals(0,10,places=2), min_size=4, max_size=4, unique=True).map(sorted))
```

```
@st.composite
def generate_function(draw):
    Xs = []
    Ys = []
    p = draw(st.lists(st.floats(0,10), min_size=2, max_size=2))
    n = draw(st.integers(1,10))
    a = min(p[0], p[1])
```

Strategy - Regular Expressions

```
exp =draw(st.from_regex("([+-][1-9]{1,1}([m][x][p][1-9]{1,1})([m][c][o][s][(][x][)])){0,1}){1,2}",fullmatch = True)
```

Strategy - Regular Expressions

Example : generated expressions

```
5*x**5-5*x**5-5*y**5-5*y**5
1*x**8*cos(x)-3*x**2*cos(x)*sin(x)+4*y**7-2*y**2*sin(y)
9*x**9*sin(x)-9*x**9*cos(x)-9*y**9*cos(y)-9*y**9
4*x**7+7*x**4*cos(x)*sin(x)+7*y**4*cos(y)*sin(y)+4*y**4
```

On using numpy function

Hypothesis offers a number of strategies for NumPy testing, available in the **hypothesis.extra.numpy** package.

```
dim = draw( st.integers(min_value = 3 , max_value = 10) )

matrix_under_test= draw(numpy.arrays(dtype=np.integer,shape=(dim,dim), elements=st.integers(1,10),unique = True))
```

assume() function

Services as a filter for unwanted test inputs.

```
        assume(final_x > initial_x)

        assume(intervals > 0)
```

Composite decorator

- Appropriate for customized strategies.
- very useful for reusing the code.

```
@st.composite
def generate_list(draw):
    Xs = draw(st.lists(st.decimals(0,10,places=2), min_size=4, max_size=4, unique=True).map(sorted))
    Ys = draw(st.lists(st.decimals(0,10,places=2), min_size=4, max_size=4, unique=True).map(sorted))
    new_x = draw(st.floats(min(Xs),max(Xs)))
    Xs = [float(i) for i in Xs]
    Ys = [float(i) for i in Ys]
    return Xs, Ys, new_x
```

Composite decorator

How to use this function?

```
@given(l =generate_list())
def test_interpolationLagrange(l):
    Xs = l[0]
    Ys = l[1]
    new_x = l[2]
```

How failure appears?

Normally the output of a **failing test** will look something like this

```
Falsifying example: test_power_method(
    matrix_under_test=array([[1, 2, 3],
           [4, 5, 6],
           [7, 9, 8]]),
)
 eigen value diff 8.881784197001252e-15
error  1.1158062433918625e-16
Traceback (most recent call last):
  File "eigen_testing.py", line 29, in test_power_method
    assert abs(numpy_eigen_values[0] - eigen_value) <= error
AssertionError

Falsifying example: test_power_method(
    matrix_under_test=array([[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]),
)
 eigen value diff 0.0
error  0.0
 eigen vector value diff 0.5153201390469264
error  0.0
```

How failure appears?

Normally the output of a **failing test** will look something like:

```
  File "C:\Program Files\Python37\lib\site-packages\hypothesis\core.py", line 851, in run_engine
    % (len(self.falsifying_examples))
hypothesis.errors.MultipleFailures: Hypothesis found 2 distinct failures.
```

but this may not be enough!

* You may what to know the value of specific variable during the test.

* Using **note(value)** will report this **value** in the final execution.

```
note(" eigen vector value diff %r"%  abs(eigen_vector_value-eigen_vector[i]))

note("error  %r"% error)
```

Our Project

Numerical Analysis tool :

An integration of multiple programs that solve mathematical problems using numerical analysis techniques.

Project Main Functions

- Interpolation
- Integration (trapezoidal - simpsons)
- Eigenvalues & Eigenvectors (power method - deflate)
- Linear System of Equation
- Interpolation then differentiation
- Extrapolation (Richardson)
- Curve Fitting
- Partial differential Equations