

FunkyYak (Functional Kayak)

Taking a few lessons from developing and working with Kayak, here is a stateless reverse-mode autodiff implementation that also offers higher-order derivatives.

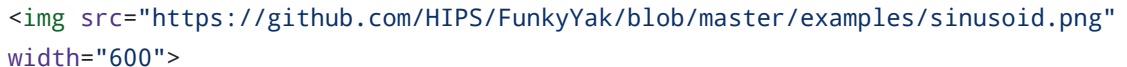
Example use:

```
```python
import numpy as np
import matplotlib.pyplot as plt
from funkyyak import grad

Define a function capable of taking `Node` objects
def fun(x):
 return np.sin(x)

d_fun = grad(fun) # First derivative
dd_fun = grad(d_fun) # Second derivative

x = np.linspace(-10, 10, 100)
plt.plot(x, map(fun, x), x, map(d_fun, x), x, map(dd_fun, x))
```


```

The function can even have control flow, which raises the prospect of differentiating through an iterative routine like an optimization. Here's a simple example.

```
```python
Taylor approximation to sin function
def fun(x):
 currterm = x
 ans = currterm
```

```

35 for i in xrange(1000):
36 currterm = - currterm * x ** 2 / ((2 * i + 3) * (2 * i + 2))
37 ans = ans + currterm
38 if np.abs(currterm) < 0.2: break # (Very generous tolerance!)
39
40 return ans
41
42 d_fun = grad(fun)
43 dd_fun = grad(d_fun)
44
45 x = np.linspace(-10, 10, 100)
46 plt.plot(x, map(fun, x), x, map(d_fun, x), x, map(dd_fun, x))
47 ```
48
49
52 We can take the derivative of the derivative automatically as well, as many
53 times as we like:
54 ```python
55 # Define the tanh function
56 def tanh(x):
57 return (1.0 - np.exp(-x)) / (1.0 + np.exp(-x))
58
59 d_fun = grad(tanh) # First derivative
60 dd_fun = grad(d_fun) # Second derivative
61 ddd_fun = grad(dd_fun) # Third derivative
62 dddd_fun = grad(ddd_fun) # Fourth derivative
63 ddddd_fun = grad(dddd_fun) # Fifth derivative
64 ddddddd_fun = grad(ddddd_fun) # Sixth derivative
65
66 x = np.linspace(-7, 7, 200)
67 plt.plot(x, map(tanh, x),
68 x, map(d_fun, x),
69 x, map(dd_fun, x),
70 x, map(ddd_fun, x),
71 x, map(dddd_fun, x),
72 x, map(ddddd_fun, x),
73 x, map(dddddd_fun, x))
74 ```
75
76 <img src="https://github.com/HIPS/FunkyYak/blob/master/examples/tanh.png"

```