```python
import numpy as np
import itertools as it
from funkyyak import grad
from copy import copy

def nd(f, *args):
    unary_f = lambda x : f(*x)
    return unary_nd(unary_f, args)

def unary_nd(f, x):
    eps = 1e-4
    if isinstance(x, np.ndarray):
        nd_grad = np.zeros(x.shape)
        for dims in it.product(*map(range, x.shape)):
            nd_grad[dims] = unary_nd(indexed_function(f, x, dims), x[dims])
        return nd_grad
    elif isinstance(x, tuple):
        return tuple([unary_nd(indexed_function(f, list(x), i), x[i])
                      for i in range(len(x))])
    elif isinstance(x, dict):
        return {k : unary_nd(indexed_function(f, x, k), v) for k, v in
x.items()}
    elif isinstance(x, list):
        return [unary_nd(indexed_function(f, x, i), v) for i, v in enumerate(x)]
    else:
        return (f(x + eps/2) - f(x - eps/2)) / eps

def indexed_function(fun, arg, index):
    local_arg = copy(arg)
    def partial_function(x):
        local_arg[index] = x
        return fun(local_arg)
    return partial_function
```

```python
def eq_class(dtype):
    return float if dtype == np.float64 else dtype

def check_equivalent(A, B):
    assert eq_class(type(A)) == eq_class(type(B)),\
        "Types are: {0} and {1}".format(eq_class(type(A)), eq_class(type(B)))
    if isinstance(A, (tuple, list)):
        for a, b in zip(A, B): check_equivalent(a, b)
    elif isinstance(A, dict):
        assert len(A) == len(B)
        for k in A: check_equivalent(A[k], B[k])
    else:
        if isinstance(A, np.ndarray):
            assert A.shape == B.shape, "Shapes are {0} and {1}".format(A.shape, B.shape)
        assert np.allclose(A, B, rtol=1e-4, atol=1e-6), "Diffs are:
{0}".format(A - B)
def check_grads(fun, *args):
    A = nd(fun, *args)
    B = tuple([grad(fun, i)(*args) for i in range(len(args))])
    check_equivalent(A, B)

def to_scalar(x):
    return np.sum(np.sin(x))
```