# Ufuncs

Computation on NumPy arrays can be very fast, or it can be very slow. The key to making it fast is to use vectorized operations, generally implemented through NumPy's universal functions (ufuncs). **The Slowness of Loops** Python's default implementation (known as CPython) does some operations very slowly. This is in part due to the dynamic, interpreted nature of the language: the fact that types are flexible, so that sequences of operations cannot be compiled down to efficient machine code as in languages like C and Fortran. For many types of operations, NumPy provides a convenient interface into just this kind of statically typed, compiled routine. This is known as a vectorizedoperation. This can be accomplished by simply performing an operation on the array, which will then be applied to each element. This vectorized approach is designed to push the loop into the compiled layer that underlies NumPy, leading to much faster execution.

**import** **numpy** **as** **np**

np.random.seed(0)

**def** compute\_reciprocals(values):

output = np.empty(len(values))

**for** i **in** range(len(values)):

output[i] = 1.0 / values[i]

**return** output

big\_array = np.random.randint(1, 100, size=1000000)

%**timeit** compute\_reciprocals(big\_array)

1 loop, best of 3: 2.91 s per loop

%**timeit** (1.0 / big\_array)

100 loops, best of 3: 4.6 ms per loop

Most arithmatic, conditional, boolean, bitwise, trignometric, maths (abs, exp,log), scipy.special (gamma,gammaln,beta, erf,erfc,erfinv) are also available as numpy ufuncs. Advanced ufuncs include specifying output and aggregation and outer products. np.sum, np.prod, np.cumsum, np.cumprod and others are also Ufuncs.Likewise indexing operators.

**from** **scipy** **import** special

*# Gamma functions (generalized factorials) and related functions*

x = [1, 5, 10]

print("gamma(x) =", special.gamma(x))

print("ln|gamma(x)| =", special.gammaln(x))

print("beta(x, 2) =", special.beta(x, 2))

gamma(x) = [ 1.00000000e+00 2.40000000e+01 3.62880000e+05]

ln|gamma(x)| = [ 0. 3.17805383 12.80182748]

beta(x, 2) = [ 0.5 0.03333333 0.00909091]

*# Error function (integral of Gaussian)*

*# its complement, and its inverse*

x = np.array([0, 0.3, 0.7, 1.0])

print("erf(x) =", special.erf(x))

print("erfc(x) =", special.erfc(x))

print("erfinv(x) =", special.erfinv(x))

erf(x) = [ 0. 0.32862676 0.67780119 0.84270079]

erfc(x) = [ 1. 0.67137324 0.32219881 0.15729921]

erfinv(x) = [ 0. 0.27246271 0.73286908 inf]

x = np.arange(5)

y = np.empty(5)

np.multiply(x, 10, out=y)

print(y)

[ 0. 10. 20. 30. 40.]

y = np.zeros(10)

np.power(2, x, out=y[::2])

print(y)

[ 1. 0. 2. 0. 4. 0. 8. 0. 16. 0.]

x = np.arange(1, 6)

np.add.reduce(x)

15

np.multiply.reduce(x)

120

np.add.accumulate(x)

array([ 1, 3, 6, 10, 15])

np.multiply.accumulate(x)

array([ 1, 2, 6, 24, 120])

# Broadcasting

