# gumath

*Release v0.2.0dev3*

**May 15, 2018**

# Contents

This package provides tools to dispatch functions towards the memory containers. These containers can be have a general structure or a Numpy-like container with a composable, generalized function concept.

CHAPTER 1

# Installation

To run gumathn xnd and ndtypes, your computer requires a Python interpreters, either version 2.7 or superior.

gumath can be installed using pip:

```
python3 -m pip install gumath
```

Or using anaconda package manager:

```
conda install -c xnd/label/dev gumath
```

gumath does not depend on third-party Python except for xnd and ndtypes (currently, these packages do not have any external dependensives themselves).

CHAPTER 2

# Index

## 2.1 Libgumath

C library.

### 2.1.1 libgumath

The libgumath library implements support for the function dispatch to the memory blocks defined using the xnd library.

This initial libgumath version displays a simple design. The goal is to determine whether the kernel signatures and the dispatch model are suitable for Numba.

Currently the only functions available are *sine* and *cosine*.

#### Gufunc

A gufunc is defined as a name and a collection of associated kernels. Gufunc structs are in a lookup table with their names as keys:

```
typedef struct {
    char *name;
    int nkernels;
    gm_kernel_t kernels[GM_MAX_KERNELS];
} gm_func_t;
```

Since each kernel has its own type signature, gufuncs are essentially multimethods.

#### Kernel

The kernel struct contains the type signature together with several (possibly optimized) kernel functions. Each of these functions may be NULL.

```
typedef void (* gm_c_kernel_t)(xnd_ndarray_t stack[]);
typedef void (* gm_fortran_kernel_t)(xnd_ndarray_t stack[]);
typedef void (* gm_strided_kernel_t)(xnd_ndarray_t stack[]);
typedef void (* gm_xnd_kernel_t)(xnd_t stack[]);

typedef struct {
    ndt_t *sig;

    gm_c_kernel_t C;
    gm_fortran_kernel_t Fortran;
    gm_strided_kernel_t Strided;
    gm_xnd_kernel_t Xnd;
} gm_kernel_t;
```

The idea is to have highly optimized kernels for contiguous C and Fortran arrays, a generic strided kernel for non-contiguous arrays and Xnd kernels for situations where variable arrays or optional values are needed.

For Numpy arrays the Xnd struct member may be NULL.

### Kernel application

The algorithm for gufunc application can be seen in the Python module.

1. Get the function name and the list of xnd function arguments.
2. Get the types of the function arguments.
3. Select the kernel:
   (a) Lookup the gufunc in the function table.
   (b) Iterate over the type signatures.
      i. If no match is found, return an error.
      ii. If a match is found, compute the return type(s) and the number of outer dimensions to be skipped.

      This stage should probably also do broadcasting, which is currently not implemented.
4. Allocate new xnd container(s) for the return values.
5. Input and output containers are pushed on a single stack. The types, which are available at any stage of the array traversal, keep track of the number of in/out args.
6. Call gm_map(), which orchestrates kernel application.
7. The actual kernel {C, Fortran, Strided, Xnd} is selected right before application (in this order of preference).

   If no kernel is found, an error is returned.

### More specialized kernel signatures

What to add to {C, Fortran, Strided, Xnd}?

MKL would be an obvious choice. Another idea is to support kernels with closure-like state and constr/destructor functions for the state.

#### Numba integration

The basic idea is that libgumath contains functions that allow inserting gufuncs and kernels into the lookup table.

Ideally, Numba would jit-compile specialized kernels and call the insertion function (must be on the C level for safety).

The function is then automatically available to be called on the Python level via the gumath Python module.

#### Obstacles

- If the datashape (ndt_t) signatures are given on the Python level (which is probably the only sane option), the jit-compiled kernel needs to be type-checked against the ndt_t type.

## 2.2 Gumath

Python module.

### 2.2.1 Mathematical operations

The gumath functions provide a python wrapper for the libgumath library. The operations currently available are *sine* and *cosine*.

#### Trigonometry functions

##### sin

Trigonometric sine, element-wise.

**param x** *(array_like)* Angle, in radians ($2\pi$ rad equals 360 degrees).

**returns** *(array_like)* The sine of each element of x.

#### Example

```
>>> import gumath as gm
>>> from xnd import xnd
>>> x = [0.0, 45 * 3.14159/180, 90 * 3.14159/180]
>>> gm.sin(xnd(x))
xnd([0.0, 0.70710, 0.99999], type='3 * float64')
```

##### cos

Trigonometric cosine, element-wise.

**param x** *(array_like)* Angle, in radians (2*pi rad equals 360 degrees).

**returns** *(array_like)* The cosine of each element of x.

#### Example

```
>>> import gumath as gm
>>> from xnd import xnd
>>> x = [0.0, 45 * 3.14159/180, 90 * 3.14159/180]
>>> gm.cos(xnd(x))
xnd([1.0, 0.70710, 6.12323e-17], type='3 * float64')
```

## 2.3 Releases

### 2.3.1 Releases

#### v0.2.0b2 (February 5th 2018)

The first version of libgumath has a relatively simple design. The goal is to determine whether the kernel signatures and the dispatch model are suitable for Numba.

Currently there is just one working test with sin().