Zumpy

Generated by Doxygen 1.9.3

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 array Struct Reference

```
#include <zumpy.h>
```

Data Fields

- void ∗ data
- size_t ∗ arr_shape
- size_t shape_size
- size_t type_size
- size_t total_size
- type dtype

3.1.1 Detailed Description

Definition at line 13 of file zumpy.h.

3.1.2 Field Documentation

3.1.2.1 arr_shape

```
size_t∗ arr_shape
```

Definition at line 16 of file zumpy.h.

3.1.2.2 data

```
void* data
```

Definition at line 15 of file zumpy.h.

3.1.2.3 dtype

```
type dtype
```

Definition at line 20 of file zumpy.h.

3.1.2.4 shape_size

```
size_t shape_size
```

Definition at line 17 of file zumpy.h.

3.1.2.5 total_size

```
size_t total_size
```

Definition at line 19 of file zumpy.h.

3.1.2.6 type_size

```
size_t type_size
```

Definition at line 18 of file zumpy.h.

The documentation for this struct was generated from the following file:

- src/c/include/zumpy.h

Chapter 4

File Documentation

4.1 src/c/include/zumpy.h File Reference

```
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
```

Data Structures

- struct array

Enumerations

- enum type { INT32 , FLOAT }
- enum filter_type { ANY , ALL }

Functions

- void arr_init (array ∗arr, size_t ∗arr_shape, size_t shape_size, type dtype)

  Initialize an empty array of arbitrary shape.
- void arr_free (array ∗arr)

  Free up allocated memory taken by the array.
- void ∗ arr_at (array ∗arr, size_t ∗index)

  Access an element of the array by index.
- void arr_set (array ∗arr, size_t ∗index, void ∗value)

  Set a single value within the array.
- void arr_fill (array ∗arr, void ∗value)

  Fill an array with a constant value.
- float arr_sum (array ∗arr)

  Sum all elements in an array.

- void arr_slice (array ∗srcarray, size_t ∗∗sub_arr_idx, size_t ∗sub_arr_dims, size_t sub_arr_dims_len, array ∗subarray)

  Slice an array by specifying a jagged array indicating what indices to pull from which dimensions of a source array and store them into a target aray.
- void arr_print (array ∗arr)

  Print the contents of an array to the console.
- void arr_filter (array ∗arr, bool(∗filter)(void ∗), size_t ∗secondary_indices, size_t secondary_indices_size, filter_type, array ∗dest)

  Function to filter an array's rows and store results into a different array, dest. This is one of the more complicated functions so it may be best to check out the example to make sure you can understand.

4.1.1 Enumeration Type Documentation

4.1.1.1 filter_type

```
enum filter_type
```

Filter type used in arr_filter(array∗, bool (∗)(void∗), size_t∗, size_t, filter_type, array∗) to specify whether to check if ALL columns match the condition or if at least one does.

Enumerator

ANY	
ALL	

Definition at line 308 of file zumpy.h.

4.1.1.2 type

```
enum type
```

Enumerator

INT32	
FLOAT	

Definition at line 11 of file zumpy.h.

4.1.2 Function Documentation

4.1.2.1 arr_at()

```
void * arr_at (
             array * arr,
             size_t * index )
```

Access an element of the array by index.

Parameters

arr	Reference (pointer) to an array struct.
index	A size_t array (decayed to pointer) indicating the index to access.

```
array myarr;
size_t shape[] = {3, 3, 3};
arr_init(&myarr, shape, 3, INT32);
int32_t val = 10;
arr_fill(&myarr, &val);
// specify the index to access within the "3D" array
size_t index[] = { 2, 1, 1 };
printf("%d\n", *(int32_t*)arr_at(&myarr, index)); // 10
arr_free(&myarr);
```

4.1.2.2 arr_fill()

```
void arr_fill (
             array * arr,
             void * value )
```

Fill an array with a constant value.

Parameters

arr	Reference (pointer) to an array struct.
value	Value to fill the array with.

```
#include "zumpy.h"
// ... other code
array myarr;
size_t shape[] = {3, 3, 3};
arr_init(&myarr, shape, 3, INT32);
// fill "3D" array with values of 10
int32_t val = 10;
arr_fill(&myarr, &val);
// print the contents of the "3D" array
size_t index[] = { 0, 0, 0 };
for (size_t i = 0; i < shape[0]; ++i)
{
    index[0] = i;
    for (size_t j = 0; j < shape[1]; ++j)
    {
        index[1] = j;
        for (size_t k = 0; k < shape[2]; ++k)
        {
            index[2] = k;
            printf("%d ", *(int32_t*)arr_at(&myarr, index));
        }
        printf("\n");
    }
    printf("\n");
}
arr_free(&myarr);
```

Output:

```
10 10 10
10 10 10
10 10 10
10 10 10
10 10 10
10 10 10
10 10 10
10 10 10
10 10 10
```

4.1.2.3 arr_filter()

```
void arr_filter (
            array * arr,
            bool(*)(void *) filter,
            size_t * secondary_indices,
            size_t secondary_indices_size,
            filter_type ,
            array * dest )
```

Function to filter an array's rows and store results into a different array, dest. This is one of the more complicated functions so it may be best to check out the example to make sure you can understand.

Warning

It's best to initialize the "data" member for the "dest" array to NULL to avoid uninitialized errors. See examples below.

Note

If no rows are found, an "empty" array is created with {0, 0, ..., 0} shape.

Parameters

arr	Primary array to filter
filter	A boolean function pointer specifying your filter condition(s). Parameter must be a void pointer and you will have to cast your value to the desired data type.
secondary_indices	Optional parameter specifying specific column(s) to apply the filter to. If NULL is passed, all columns will be checked.
secondary_indices_size	The size of the previous parameter, secondary_indices. If NULL is passed, you can pass 0.
filter_type	An enum specifying the "type" of filter to apply. One of "ANY" or "ALL". For 1D arrays this has no effect. 1D arrays will just remove the values that don't match the condition regardless of which setting is used. For 2D arrays and higher, it will only keep the row if ANY of the specified columns match the condition, or if ALL values in the specified columns match the condition. See examples below for more detail.
dest	Destination array to store filtered results into. Memory will be allocated inside the function call so no need to initialize it beforehand.

The simplest example, filtering a 1D array of 5 elements. Note that using ANY or ALL would produce the exact same result. As mentioned above, this parameter has no effect on 1D arrays.

```
#include "zumpy.h"
#include <time.h>
bool filter(void* value)
```

```
{
    return *(int32_t*)value > 10;
}
int main()
{
    // initialize 3x3x3 array
    size_t shape[] = {5};
    array arr;
    arr_init(&arr, shape, 1, INT32);
    // fill array with crude random values between 0-49
    int32_t val;
    size_t idx[] = {0};
    srand(3331); // specific seed for reproducibility
    for (size_t r = 0; r < arr.arr_shape[0]; ++r)
    {
        idx[0] = r;
        val = rand() % 50;
        arr_set(&arr, idx, &val);
    }
    printf("BEFORE FILTERING:\n======================\n");
    arr_print(&arr);
    printf("\n");
    // it's best to initialize data to NULL to avoid uninitialized errors
    array filtered = {.data = NULL};
    // this filter checks for values that match the condition. Note that 1D arrays
    // are a bit of a special case. ANY or ALL actually has no effect. It will remove the values
    // that don't match regardless of which setting is used.
    arr_filter(&arr, &filter, NULL, 0, ANY, &filtered);
    printf("\nAFTER FILTERING:\n======================\n");
    arr_print(&filtered);
    // deallocate
    arr_free(&arr);
    arr_free(&filtered);
    return 0;
}
```

Output:

```
BEFORE FILTERING:
======================
43 8 25 26 13
\n
AFTER FILTERING:
======================
43 25 26 13
```

Here's a simple example of filtering a 3x2 array on one column. Next example shows a more complicated case of filtering a 3x3x3 array.

```
#include "zumpy.h"
#include <time.h>
bool filter(void* value)
{
    return *(int32_t*)value > 10;
}
int main()
{
    // initialize 3x2 array
    size_t shape[] = {3, 2};
    array arr;
    arr_init(&arr, shape, 2, INT32);
    // fill array with crude random values between 0-49
    int32_t val;
    size_t idx[] = {0, 0};
    srand(3331); // specific seed for reproducibility
    for (size_t r = 0; r < arr.arr_shape[0]; ++r)
    {
        idx[0] = r;
        for (size_t c = 0; c < arr.arr_shape[1]; ++c)
        {
            idx[1] = c;
            val = rand() % 50;
            arr_set(&arr, idx, &val);
        }
    }
    printf("BEFORE FILTERING:\n======================\n");
    arr_print(&arr);
    printf("\n");
    // it's best to initialize data to NULL to avoid uninitialized errors
    array filtered = {.data = NULL};
    // only apply filter to first and third column
    size_t secondary_idx[] = {1};
    // this filter checks if ANY columns in the second dimension are > 10 for column 1
    arr_filter(&arr, &filter, secondary_idx, 1, ANY, &filtered);
    printf("AFTER FILTERING:\n======================\n");
    arr_print(&filtered);
    // deallocate
    arr_free(&arr);
    arr_free(&filtered);
    return 0;
}
```

Output:
```
BEFORE FILTERING:
======================
43 8
25 26
13 44
\n
AFTER FILTERING:
======================
25 26
13 44
```

The below code is an example of filtering two columns from a 3x3x3 array. Three dimensions can be a little hard to visualize so hopefully this sheds insight into how it works for dimensions greater than two. Note that we have three rows of 3x3 arrays. We are checking column 0 and 2 in each 3x3 array. If ALL three rows in the 3x3 array meet the condition we keep it. Otherwise, we toss it. You can see that the third 3x3 block is excluded because the very last element is 1 which is in column 2 and is not greater than 10. But the other two blocks are kept because every value in column 0 and 2 is greater than 10.
```
#include "zumpy.h"
#include <time.h>
bool filter(void* value)
{
    return *(int32_t*)value > 10;
}
int main()
{
    // initialize 3x3x3 array
    size_t shape[] = {3, 3, 3};
    array arr;
    arr_init(&arr, shape, 3, INT32);
    // fill array with crude random values between 0-49
    int32_t val;
    size_t idx[] = {0, 0, 0};
    srand(3331); // specific seed for reproducibility
    for (size_t r = 0; r < arr.arr_shape[0]; ++r)
    {
        idx[0] = r;
        for (size_t c = 0; c < arr.arr_shape[1]; ++c)
        {
            idx[1] = c;
            for (size_t z = 0; z < arr.arr_shape[2]; ++z)
            {
                idx[2] = z;
                val = rand() % 50;
                arr_set(&arr, idx, &val);
            }
        }
    }
    printf("BEFORE FILTERING:\n======================\n");
    arr_print(&arr);
    printf("\n");
    // it's best to initialize data to NULL to avoid uninitialized errors
    array filtered = {.data = NULL};
    // only apply filter to first and third column
    size_t secondary_idx[] = {0,2};
    // this filter checks if ALL columns in the third dimension are > 10 for column 0 and 2
    arr_filter(&arr, &filter, secondary_idx, 2, ALL, &filtered);
    printf("AFTER FILTERING:\n======================\n");
    arr_print(&filtered);
    // deallocate
    arr_free(&arr);
    arr_free(&filtered);
    return 0;
}
```

Output:
```
BEFORE FILTERING:
======================
43 8 25
26 13 44
11 44 27
\n
26 20 40
44 19 17
40 49 47
\n
46 20 38
```

```
41 18 16
31 12 1
\n
\n
AFTER FILTERING:
======================
43 8 25
26 13 44
11 44 27
\n
26 20 40
44 19 17
40 49 47
```

4.1.2.4 arr_free()

```
void arr_free (
              array * arr )
```

Free up allocated memory taken by the array.

Parameters

arr	Reference (pointer) to an array struct.

```
#include "zumpy.h"
// ... other code
array myarr;
size_t shape[] = {3, 3, 3}; // create a 3x3x3 array
arr_init(&myarr, shape, 3, INT32); // allocates memory
printf("%d %d %d\n", myarr.shape[0], myarr.shape[1], myarr.shape[2]);
arr_free(&myarr); // free memory allocated by the array
```

4.1.2.5 arr_init()

```
void arr_init (
              array * arr,
              size_t * arr_shape,
              size_t shape_size,
              type dtype )
```

Initialize an empty array of arbitrary shape.

Parameters

arr	Reference (pointer) to an array struct.
arr_shape	A size_t array (decayed to a pointer) indicating the dimensions of the array.
shape_size	The length of the shape; i.e, the total number of dimensions.
dtype	Data type of the array; must be one of INT32 or FLOAT.

```
array myarr;
size_t shape[] = {3, 3, 3}; // create a 3x3x3 array
arr_init(&myarr, shape, 3, INT32); // allocates memory
printf("%d %d %d\n", myarr.shape[0], myarr.shape[1], myarr.shape[2]);
arr_free(&myarr); // free memory allocated by the array
```

4.1.2.6 arr_print()

```
void arr_print (
            array * arr )
```

Print the contents of an array to the console.

Parameters

arr	Reference (pointer) to an array struct.

```
#include "zumpy.h"
// ... other code
// initialize 3x2 array
size_t shape[] = {3, 2};
array arr;
arr_init(&arr, shape, 2, INT32);
// fill all cells with 10
int32_t val = 10;
arr_fill(&arr, &val);
// print array contents to the console
arr_print(&arr);
// deallocate
arr_free(&arr);
```

4.1.2.7 arr_set()

```
void arr_set (
            array * arr,
            size_t * index,
            void * value )
```

Set a single value within the array.

Parameters

arr	Reference (pointer) to an array struct.
index	Index to set the value at.
value	Value to set.

```
#include "zumpy.h"
// ... other code
array myarr;
size_t shape[] = {3, 3, 3};
arr_init(&myarr, shape, 3, INT32);
// fill array with values of 10
int32_t val = 10;
arr_fill(&myarr, &val);
// set index 2,1,1 to 20
size_t index[] = { 2, 1, 1 };
val = 20;
arr_set(&myarr, index, &val);
printf("%d\n", *(int32_t*)arr_at(&myarr, index)); // 20
arr_free(&myarr);
```

4.1.2.8 arr_slice()

```
void arr_slice (
            array * srcarray,
            size_t ** sub_arr_idx,
            size_t * sub_arr_dims,
            size_t sub_arr_dims_len,
            array * subarray )
```

Slice an array by specifying a jagged array indicating what indices to pull from which dimensions of a source array and store them into a target aray.

Note

For the sub array, you DO NOT need to initalize it as it will be initialized in the function for you. But you still must free it. See the example below for a full example.

Parameters

srcarray	Source array to slice from.
sub_arr_idx	A jagged array indicating the indices to pull from each dimension of srcarray. Index 0 will be an array of indices to extract from dimension 0 of the array and so on for higher indices.
sub_arr_dims	An array indicating the shape of the slice. E.g {3, 1} if your slice will produce a 3x1 array.
sub_arr_dims_len	A value indicating total dimensions that are being sliced.
subarray	Target array to store slices into.

```c
#include "zumpy.h"
// ... other code
// set up dimensions to slice
// this will take index 0-2 on dimension 0 and index 0 on dimension 1 (slicing one column)
size_t dims[2] = {3, 1}; // slice 3x1 array
size_t dim0[3] = {0, 1, 2}; // pull index 0-2 from dimension 0 (i.e all rows)
size_t dim1[1] = {0}; // pull index 0 from dimension 1 (i.e the first column)
size_t* sub_arr_index[2] = { dim0, dim1 };
// create example array: 3x3 filled with 10s
size_t shape[2] = {3, 3};
array arr, sub; // NOTE: DO NOT initialize sub here; it will be initialized for you
arr_init(&arr, shape, 2, INT32);
int32_t val = 10;
arr_fill(&arr, &val);
// slice the first column
arr_slice(&arr, sub_arr_index, dims, 2, &sub);
// print our sliced array (TODO: implement a generic method to do this; this is messy!!)
size_t idx[2] = {0,0};
for (size_t i = 0; i < sub.arr_shape[0]; ++i) {
   idx[0] = i;
   for (size_t j = 0; j < sub.arr_shape[1]; ++j) {
        idx[1] = j;
        printf("%d ", *(int32_t *)arr_at(&sub, idx));
    }
    printf("\n");
}
arr_free(&arr);
arr_free(&sub);
```

Output:

This slices a 3x1 array of 10s from the original 3x3 array (the first column)

```
10
10
10
```

4.1.2.9 arr_sum()

```
float arr_sum (
              array * arr )
```

Sum all elements in an array.

Note

For multi-dimensional arrays this will sum ALL cells. If you want to sum a specific row or column, check arr_sum_row(array∗) and arr_sum_column(array∗).

See also

arr_sum_row(array∗)

arr_sum_column(array∗)

Parameters

arr	Reference (pointer) to an array struct.

Returns

The sum of all cells as a float.

```
#include "zumpy.h"
// ... other code
array myarr;
size_t shape[] = {3, 3};
arr_init(&myarr, shape, 2, INT32);
// fill array with values of 10
int32_t val = 10;
arr_fill(&myarr, &val);
printf("%f\n", arr_sum(&myarr)); // 90.0
arr_free(&myarr);
```

4.2 zumpy.h

Go to the documentation of this file.

```
#ifndef ZUMPY_ZUMPY_H
#define ZUMPY_ZUMPY_H

#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

typedef enum { INT32, FLOAT } type;

typedef struct
{
    void *data;
    size_t *arr_shape;
    size_t shape_size;
    size_t type_size;
    size_t total_size;
    type dtype;
} array;

void arr_init(array* arr, size_t* arr_shape, size_t shape_size, type dtype);



void arr_free(array* arr);



void* arr_at(array* arr, size_t* index);



void arr_set(array* arr, size_t* index, void* value);


void arr_fill(array* arr, void* value);



float arr_sum(array* arr);



void arr_slice(array* srcarray, size_t** sub_arr_idx, size_t* sub_arr_dims, size_t sub_arr_dims_len,
 array* subarray);



void arr_print(array* arr);

typedef enum {ANY, ALL} filter_type;

void arr_filter(array* arr, bool (*filter)(void*), size_t* secondary_indices, size_t
 secondary_indices_size, filter_type, array* dest);
#endif //ZUMPY_ZUMPY_H
```

Index