

PBMath

P. Baillehache

September 27, 2017

Contents

1	Interface	1
2	Code	3
3	Makefile	6
4	Usage	7

Introduction

PBMath is C library providing mathematical structures and functions.

The `Vec` structure and its function can be used to manipulate vectors of float values.

1 Interface

```
// ===== PBMath.H =====  
  
#ifndef PBMath_H  
#define PBMath_H  
  
// ===== Include =====  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <math.h>  
#include <string.h>  
#include <stdbool.h>
```

```

// ===== Define =====

#define PBMath_EPSILON 0.0000001

void VecTypeUnsupported(void*t, ...);
#define VecClone(V) _Generic((V), \
    VecFloat*: VecFloatClone, \
    default: VecTypeUnsupported)(V)
#define VecLoad(V, S) _Generic((V), \
    VecFloat*: VecFloatLoad, \
    default: VecTypeUnsupported)(V, S)
#define VecSave(V, S) _Generic((V), \
    VecFloat*: VecFloatSave, \
    default: VecTypeUnsupported)(V, S)
#define VecFree(V) _Generic((V), \
    VecFloat*: VecFloatFree, \
    default: VecTypeUnsupported)(V)
#define VecPrint(V, S) _Generic((V), \
    VecFloat*: VecFloatPrintDef, \
    default: VecTypeUnsupported)(V, S)
#define VecGet(V, I) _Generic((V), \
    VecFloat*: VecFloatGet, \
    default: VecTypeUnsupported)(V, I)
#define VecSet(V, I, VAL) _Generic((V), \
    VecFloat*: VecFloatSet, \
    default: VecTypeUnsupported)(V, I, VAL)
#define VecDim(V) _Generic((V), \
    VecFloat*: VecFloatDim, \
    default: VecTypeUnsupported)(V)

// ===== Data structure =====

// Vector of float values
typedef struct VecFloat {
    // Dimension
    int _dim;
    // Values
    float *_val;
} VecFloat;

// ===== Functions declaration =====

// Create a new VecFloat of dimension 'dim'
// Values are initialized to 0.0
// Return NULL if we couldn't create the VecFloat
VecFloat* VecFloatCreate(int dim);

// Clone the VecFloat
// Return NULL if we couldn't clone the VecFloat
VecFloat* VecFloatClone(VecFloat *that);

// Load the VecFloat from the stream
// If the VecFloat is already allocated, it is freed before loading
// Return 0 in case of success, or:
// 1: invalid arguments
// 2: can't allocate memory
// 3: invalid data
// 4: fscanf error
int VecFloatLoad(VecFloat **that, FILE *stream);

// Save the VecFloat to the stream

```

```

// Return 0 upon success, or
// 1: invalid arguments
// 2: fprintf error
int VecFloatSave(VecFloat *that, FILE *stream);

// Free the memory used by a VecFloat
// Do nothing if arguments are invalid
void VecFloatFree(VecFloat **that);

// Print the VecFloat on 'stream' with 'prec' digit precision
// Do nothing if arguments are invalid
void VecFloatPrint(VecFloat *that, FILE *stream, int prec);
void VecFloatPrintDef(VecFloat *that, FILE *stream);

// Return the i-th value of the VecFloat
// Index starts at 0
// Return 0.0 if arguments are invalid
float VecFloatGet(VecFloat *that, int i);

// Set the i-th value of the VecFloat to v
// Index starts at 0
// Do nothing if arguments are invalid
void VecFloatSet(VecFloat *that, int i, float v);

// Return the dimension of the VecFloat
// Return 0 if arguments are invalid
int VecFloatDim(VecFloat *that);

#endif

```

2 Code

```

// ===== PBMATH.C =====

// ===== Include =====

#include "pbmath.h"

// ===== Define =====

// ===== Functions implementation =====

// Create a new Vec of dimension 'dim'
// Values are initialized to 0.0
// Return NULL if we couldn't create the Vec
VecFloat* VecFloatCreate(int dim) {
    // Check argument
    if (dim <= 0)
        return NULL;
    // Allocate memory
    VecFloat *that = (VecFloat*)malloc(sizeof(VecFloat));
    // If we could allocate memory
    if (that != NULL) {
        // Allocate memory for values
        that->_val = (float*)malloc(sizeof(float) * dim);
        // If we couldn't allocate memory
        if (that->_val == NULL) {
            // Free memory
            free(that);

```

```

        // Stop here
        return NULL;
    }
    // Set the default values
    that->_dim = dim;
    for (int i = dim; i--;)
        that->_val[i] = 0.0;
}
// Return the new VecFloat
return that;
}

// Clone the VecFloat
// Return NULL if we couldn't clone the VecFloat
VecFloat* VecFloatClone(VecFloat *that) {
    // Check argument
    if (that == NULL)
        return NULL;
    // Create a clone
    VecFloat *clone = VecFloatCreate(that->_dim);
    // If we could create the clone
    if (clone != NULL) {
        // Clone the properties
        for (int i = that->_dim; i--;)
            clone->_val[i] = that->_val[i];
    }
    // Return the clone
    return clone;
}

// Load the VecFloat from the stream
// If the VecFloat is already allocated, it is freed before loading
// Return 0 in case of success, or:
// 1: invalid arguments
// 2: can't allocate memory
// 3: invalid data
// 4: fscanf error
int VecFloatLoad(VecFloat **that, FILE *stream) {
    // Check arguments
    if (that == NULL || stream == NULL)
        return 1;
    // If 'that' is already allocated
    if (*that != NULL) {
        // Free memory
        VecFloatFree(that);
    }
    // Read the number of dimension
    int dim;
    int ret = fscanf(stream, "%d", &dim);
    // If we couldn't fscanf
    if (ret == EOF)
        return 4;
    if (dim <= 0)
        return 3;
    // Allocate memory
    *that = VecFloatCreate(dim);
    // If we couldn't allocate memory
    if (*that == NULL) {
        return 2;
    }
    // Read the values
    for (int i = 0; i < dim; ++i) {

```

```

        fscanf(stream, "%f", (*that)->_val + i);
        // If we couldn't fscanf
        if (ret == EOF)
            return 4;
    }
    // Return success code
    return 0;
}

// Save the VecFloat to the stream
// Return 0 upon success, or:
// 1: invalid arguments
// 2: fprintf error
int VecFloatSave(VecFloat *that, FILE *stream) {
    // Check arguments
    if (that == NULL || stream == NULL)
        return 1;
    // Save the dimension
    int ret = fprintf(stream, "%d ", that->_dim);
    // If we couldn't fprintf
    if (ret < 0)
        return 2;
    // Save the values
    for (int i = 0; i < that->_dim; ++i) {
        ret = fprintf(stream, "%f ", that->_val[i]);
        // If we couldn't fprintf
        if (ret < 0)
            return 2;
    }
    fprintf(stream, "\n");
    // If we couldn't fprintf
    if (ret < 0)
        return 2;
    // Return success code
    return 0;
}

// Free the memory used by a VecFloat
// Do nothing if arguments are invalid
void VecFloatFree(VecFloat **that) {
    // Check argument
    if (that == NULL || *that == NULL)
        return;
    // Free memory
    free((*that)->_val);
    free(*that);
    *that = NULL;
}

// Print the VecFloat on 'stream' with 'prec' digit precision
// Do nothing if arguments are invalid
void VecFloatPrint(VecFloat *that, FILE *stream, int prec) {
    // Check arguments
    if (that == NULL || stream == NULL)
        return;
    // Create the format string
    char format[20] = {'\0'};
    sprintf(format, "%%.%df", prec);
    // Print the values
    fprintf(stream, "<");
    for (int i = 0; i < that->_dim; ++i) {
        fprintf(stream, format, that->_val[i]);
    }
}

```

```

        if (i < that->_dim - 1)
            fprintf(stream, ",");
    }
    fprintf(stream, ">");
}

void VecFloatPrintDef(VecFloat *that, FILE *stream) {
    VecFloatPrint(that, stream, 3);
}

// Return the i-th value of the VecFloat
// Index starts at 0
// Return 0.0 if arguments are invalid
float VecFloatGet(VecFloat *that, int i) {
    // Check argument
    if (that == NULL || i < 0 || i >= that->_dim)
        return 0.0;
    // Return the value
    return that->_val[i];
}

// Set the i-th value of the VecFloat to v
// Index starts at 0
// Do nothing if arguments are invalid
void VecFloatSet(VecFloat *that, int i, float v) {
    // Check argument
    if (that == NULL || i < 0 || i >= that->_dim)
        return;
    // Set the value
    that->_val[i] = v;
}

// Return the dimension of the VecFloat
// Return 0 if arguments are invalid
int VecFloatDim(VecFloat *that) {
    // Check argument
    if (that == NULL)
        return 0;
    // Return the dimension
    return that->_dim;
}

```

3 Makefile

```

OPTIONS_DEBUG=-ggdb -g3 -Wall
OPTIONS_RELEASE=-O3
OPTIONS=$(OPTIONS_DEBUG)

all : main

main: main.o pbmath.o Makefile
    gcc $(OPTIONS) main.o pbmath.o -o main -lm

main.o : main.c pbmath.h Makefile
    gcc $(OPTIONS) -c main.c

pbmath.o : pbmath.c pbmath.h Makefile
    gcc $(OPTIONS) -c pbmath.c

clean :
    rm -rf *.o main

```

4 Usage

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "pbmath.h"

int main(int argc, char **argv) {
    // Create a vector of dimension 3
    VecFloat *v = VecFloatCreate(3);
    // If we couldn't create the vector
    if (v == NULL) {
        fprintf(stderr, "VecCreate failed\n");
        return 1;
    }
    // Print the vector
    VecPrint(v, stdout);
    fprintf(stdout, "\n");
    // Set the 2nd value to 1.0
    VecSet(v, 1, 1.0);
    // Print the vector
    VecPrint(v, stdout);
    fprintf(stdout, "\n");
    // Save the vector
    FILE *f = fopen("./vec.txt", "w");
    if (f == NULL) {
        fprintf(stderr, "fopen failed\n");
        return 2;
    }
    int ret = VecSave(v, f);
    if (ret != 0) {
        fprintf(stderr, "VecSave failed (%d)\n", ret);
        return 3;
    }
    fclose(f);
    // Load the vector
    f = fopen("./vec.txt", "r");
    if (f == NULL) {
        fprintf(stderr, "fopen failed\n");
        return 4;
    }
    VecFloat *w = NULL;
    ret = VecLoad(&w, f);
    if (ret != 0) {
        fprintf(stderr, "VecLoad failed (%d)\n", ret);
        return 5;
    }
    fclose(f);
    // Get the dimension and values of the loaded vector
    fprintf(stdout, "%d ", VecDim(w));
    for (int i = 0; i < VecDim(w); ++i)
        fprintf(stdout, "%f ", VecGet(w, i));
    fprintf(stdout, "\n");
    // Free memory
    VecFree(&w);
    VecFree(&v);
    // Return success code
    return 0;
}
```

Output:

```
<0.000,0.000,0.000>  
<0.000,1.000,0.000>  
3 0.000000 1.000000 0.000000
```

vec.txt:

```
3 0.000000 1.000000 0.000000
```