מרכז ההדרכה 2000
תמיכה ועדכונים
עדכון מס' 55
יוני 2002

שימוש בספרייה המתמטית של C++ ליישומים מדעיים

מבוא

הספרייה המתמטית של C++ מספקת עצמת תיכנות ליישומים מדעיים שונים, תוך שמירה על מאפייני הכלליות (templates), היעילות והטיפוסיות החזקה (strong-typing) המקובלים בשפה.

הספרייה התקנית מספקת תמיכה בתכנות מתמטי מדעי ע"י מספר מרכיבים:

- **valarray** - template מחלקה המייצג מערך לחישובים וקטוריים אריתמטיים.
- **complex** - template של מחלקה לתמיכה במספרים מרוכבים.
- **numeric_limits** - template מחלקה הכולל הגדרות תחומים של טיפוסים מספריים.

יש לשים לב ש- C++ מספקת תמיכה מתמטית בנוסף לזו המסופקת ע"י ספרייה התקנית של C הכוללת פונקציות לטיפול בחזקות, לוגריתמים, טריגונומטריה, יצירת מספרים אקראיים וכו'.

במאמר זה נסקור את ה- valarray כמיכל אופטימלי לביצוע פעולות וקטוריות אריתמטיות. תוכן נושא זה מבוסס על פרק 14 בספר "***C++ - מדריך מקצועי***" בהוצאת "**מרכז ההדרכה 2000**".

valarray

valarray הוא termplate מחלקה המייצג מערך לחישובים וקטוריים אריתמטיים ביעילות גבוהה: ניתן לבצע פעולות מתמטיות פשוטות, פעולות בחזקות, פעולות לוגריתמיות, פעולות טריגונומטריות ועוד.

valarray שונה מ- vector בכך שהוא אינו מספק ממשק להכנסת איברים ולהוצאתם, וגם לא איטרטורים לתחילת הסדרה ולסופה. זאת משום שייעודו העיקרי הוא לביצוע פעולות וקטוריות על מערך נתון ולא תחזוקת הסדרה.

דוגמאות לשימוש ב valarray:

– שימוש ב valarray על שלמים:

```
int arr1[] = {1, 2, 3, 4};
valarray<int > v1(arr1, 4);          // v1 = {1, 2, 3, 4}

valarray<int> v2 = v1 ;              // v2 = {1, 2, 3, 4}
valarray<int> v3 = v1<<2 ;           // v3 = {4, 8, 12, 16}
valarray<int> v4 = -v1 ;             // v4 = {-1, -2, -3, -4}
```

– פעולות על ממשיים - חישוב ממוצע:

```
double arr2[5] = { 1.0, 2.0, 3.0, 4.0, 5.0 };
valarray<double> v(arr2, 5);
double avg = v.sum() / (double)v.size();          // avg = 3.0
```

המיכל **valarray** מוגדר בקובץ הספרייה <valarray> כך :

```
template<class T>
class valarray
{
public:
        typedef T value_type;

        // constructors:
        valarray();
        explicit valarray(size_t n);
        valarray(const T& val, size_t n));
        valarray(const T *p, size_t n);
        valarray(const slice_array<T>& sa);
        valarray(const gslice_array<T>& ga);
        valarray(const mask_array<T>& ma);
        valarray(const indirect_array<T>& ia);

        // assignment:
        valarray<T>&        operator=(const valarray<T>& va);
        valarray<T>&        operator=(const T& x);
        valarray<T>&        operator=(const slice_array<T>& sa);
        valarray<T>&        operator=(const gslice_array<T>& ga);
        valarray<T>&        operator=(const mask_array<T>& ma);
        valarray<T>&        operator=(const indirect_array<T>& ia);

        // subscript [ ] :
        T                   operator[](size_t n) const;
        T&                  operator[](size_t n);
        valarray<T>         operator[](slice sa) const;
        slice_array<T>      operator[](slice sa);
        valarray<T>         operator[](const gslice& ga) const;
        gslice_array<T>     operator[](const gslice& ga);
        valarray<T>         operator[](const valarray<bool>& ba) const;
        mask_array<T>       operator[](const valarray<bool>& ba);
        valarray<T>         operator[](const valarray<size_t>& xa) const;
        indirect_array<T>   operator[](const valarray<size_t>& xa);

        // arithmetic operators:
        valarray<T>         operator+();
        valarray<T>         operator-();
        valarray<T>         operator~();
        valarray<bool>      operator!();
        valarray<T>&        operator*=(const valarray<T>& x);
        valarray<T>&        operator*=(const T& x);
        valarray<T>&        operator/=(const valarray<T>& x);
        valarray<T>&        operator/=(const T& x);
        valarray<T>&        operator%=(const valarray<T>& x);
        valarray<T>&        operator%=(const T& x);
        valarray<T>&        operator+=(const valarray<T>& x);
        valarray<T>&        operator+=(const T& x);
        valarray<T>&        operator-=(const valarray<T>& x);
        valarray<T>&        operator-=(const T& x);
        valarray<T>&        operator^=(const valarray<T>& x);
        valarray<T>&        operator^=(const T& x);
        valarray<T>&        operator&=(const valarray<T>& x);
        valarray<T>&        operator&=(const T& x);
        valarray<T>&        operator|=(const valarray<T>& x);
        valarray<T>&        operator|=(const T& x);

        // bit shift operators:
        valarray<T>&        operator<<=(const valarray<T>& x);
        valarray<T>&        operator<<=(const T& x);
```

```
    valarray<T>&        operator>>=(const valarray<T>& x);
    valarray<T>&        operator>>=(const T& x);

    // element shift operators:
    valarray<T>         shift(int n) const;
    valarray<T>         cshift(int n) const;

    // size, sum, min and max:
    size_t              size() const;
    T                   sum() const;
    T                   max() const;
    T                   min() const;

    // other operations:
    valarray<T>         apply(T fn(T)) const;
    valarray<T>         apply(T fn(const T&)) const;
    void                fill(const T& val);
    void                free();
    void                resize(size_t n, const T& c = T());
};
```

בנוסף, מוגדרות הפונקציות הגלובליות הבאות על valarray:

- אופרטורים:

```
!=      %     &     &&    >
>>      >=    <     <<    <=
*       +     -     /     ==
^       |     ||
```

- פונקציות טריגונומטריות:

```
cos     cosh    acos    sin
sinh    asin    tan     tanh
atan    atan2
```

- מקסימום, מינימום, ערך מוחלט, חזקות ולוגריתמים:

```
max     min     abs     pow
sqrt    exp     log     log10
```

<u>דוגמאות לשימוש ב- valarray:</u>

1. הקוד הבא מאתחל עצמי valarray מטיפוס int, ומבצע עליהם פעולות:

```
int arr1[] = {1, 2, 3, 4};
valarray<int > v(arr1, 4);   // {1, 2, 3, 4}

valarray<int> v1 = v ;       // {1, 2, 3, 4}
valarray<int> v2 = v<<2 ; // {4, 8, 12, 16}
valarray<int> v3 = -v ;      // {-1, -2, -3, -4}
valarray<int> v4 = 3 * v ;  // {3, 6, 9, 12}
valarray<int> v5 = v + v4; // {4, 8, 12, 16}
```

2. פעולות הזזה על סיביות (<<,>>) ופעולות הזזה על איברים (shift, cshift):

```
int arr1[] = {1, 2, 3, 4};
valarray<int > v(arr1, 4);          // {1, 2, 3, 4}

valarray<int> v1 = v <<2;           // {4, 8, 12, 16}
valarray<int> v2 = v >> 1;          // {0, 1, 1, 2}
valarray<int> v3 = v.shift(2) ;     // {3, 4, 0, 0}
```

```
valarray<int> v4 = v.shift(-1);        // {0, 1, 2, 3}
valarray<int> v5 = v.cshift(2);        // {3, 4, 1, 2}
valarray<int> v6 = v.cshift(-1);       // {4, 1, 2, 3}
```

לצורך הדפסת valarrays ניתן להגדיר אופרטור >> גלובלי:

```
template <class T>
ostream & operator<<(ostream& out, const valarray<T>& v)
{
        for(int i=0; i<v.size(); i++)
                out << v[i] << " ";
        return out << endl;
}
```

וכעת ניתן להדפיס valarrays:

```
cout << "v1 = " << v1;
cout << "v2 = " << v2;
```

3. הגדרת מטריצת ממשיים כ- valarray של valarrays:

```
double arr1[] = {1, 2, 3, 4};
valarray<double> v1(arr1, 4);

valarray<valarray<double> >  mat1(5);        // 5*4 double matrix
mat1[0] = v1;

for(int i=1; i<mat1.size(); i++)
        mat1[i] = mat1[i-1].cshift(1);

cout << "v1 = " << v1;
cout << "mat1 = " << endl << mat1;

valarray<valarray<double> > mat2 = mat1 / v1;
cout << "mat2 = mat1 / v1 = " << endl << mat2;
```

הפלט:

```
v1 = 1 2 3 4
mat1 =
 1 2 3 4
 2 3 4 1
 3 4 1 2
 4 1 2 3
 1 2 3 4

mat2 = mat1 / v1 =
 1 1 1 1
 2 1.5 1.33333 0.25
 3 2 0.333333 0.5
 4 0.5 0.666667 0.75
 1 1 1 1
```

4. כנ"ל, עם מטריצת שלמים:

```
int arr1[] = {1, 2, 3, 4};
valarray<int> v1(arr1, 4);

valarray<valarray<int> >  mat1(5);  // 5*4 int matrix
mat1[0] = v1;

for(int i=1; i<mat1.size(); i++)
        mat1[i] = mat1[i-1].cshift(1);
```

```
cout << "mat1 = " << endl << mat1;

valarray<valarray<int> > mat2 = mat1 * v1;
cout << "mat2 = mat1*v1 = " << endl << mat2;

valarray<valarray<int> > mat3 = mat1 * mat2;
cout << "mat3 = mat1*mat2 = " << endl << mat3;
```

הפלט:

```
mat1 =
 1  2  3  4
 2  3  4  1
 3  4  1  2
 4  1  2  3
 1  2  3  4

mat2 = mat1*v1 =
 1  4  9  16
 2  6  12  4
 3  8  3  8
 4  2  6  12
 1  4  9  16

mat3 = mat1*mat2 =
 1  8  27  64
 4  18  48  4
 9  32  3  16
 16  2  12  36
 1  8  27  64
```

תכנית דוגמא: עיבוד אותות סיפרתיים (DSP)

כתכנית דוגמא ליכולת העיבוד המתמטי של הספרייה, נכתוב יישום לעיבוד אות בסיסי תוך שימוש במיכל valarray ובפעולות המוגדרות עליו.

ראשית, נגדיר מספר פונקציות עזר שישמשו אותנו בתכנית:

- אופרטורי קלט/פלט עבור valarray:

```
// valarray I/O
template <class T>
istream & operator>>(istream& in, valarray<T>& v)
{
        for(int i=0; i<v.size(); i++)
                in >> v[i];
        return in;
}

template <class T>
ostream & operator<<(ostream& out, const valarray<T>& v)
{
        for(int i=0; i<v.size(); i++)
                out << v[i] << " ";
        return out << endl;
}
```

- כמו כן נגדיר פונקציה בשם **plot** להצגת valarry באופן גרפי. הרעיון: נגדיר מטריצה בגודל שורות ועמודות המייצגות את המסך, ונכניס את ערכי המערך הנתון למטריצה כך שיוצגו כגרף:

```
template <class T>
void plot(const valarray<T>& v)
{
```

```
        const int W = 50;
        const int H = 20;

        valarray<valarray<char> > screen(H+1);     // matrix rep. screen
        valarray<char> row(' ', W);
        for(int i=0; i<=H; i++)
                screen[i] = row;

        // normalize the input array by shifting and scaling
        valarray<T> v_norm = v;
        T shift = v_norm.min();
        if(shift < 0)
                v_norm -= shift; // if has negative values, shift to 0

        double factor = ((double)H) / v_norm.max();
        v_norm *= factor;    // scale to 0..H

        // set matrix line in reverse order (bottom up)
        screen[H-abs(shift*factor)] = valarray<char>('_', W);
        for(int col=0; col<W; col++)
        {
                int row = (int) v_norm[col];
                screen[H-row][col] = 'x';
        }

        // display to screen
        cout << screen << endl;
}
```

הסבר: לפונקציה מספר שלבים -

- איתחול מטריצה של תווים, valarray<valarray<char> >, המייצגת את המסך.
- נירמול מערך הקלט כך שערכיו יהיו בתחום המסך: מבוצעת ראשית פעולת הזזה (כך שלא יהיו ערכים שליליים):

```
        // normalize the input array by shifting and scaling
        valarray<T> v_norm = v;
        T shift = v_norm.min();
        if(shift < 0)
                v_norm -= shift; // if has negative values, shift to 0
```

לאחר מכן מבוצע נירמול ע"י חלוקה ביחס שבין גובה המסך לערך המקסימלי:

```
        double factor = ((double)H) / v_norm.max();
        v_norm *= factor;    // scale to 0..H
```

- בשלב הבא ממלאים את תוכן המטריצה עפ"י ערכי המערך בסדר הפוך - כך ששורת הערכים המינימליים תודפס אחרונה, בתחתית המסך:

```
        screen[H-abs(shift*factor)] = valarray<char>('_', W);
        for(int col=0; col<W; col++)
        {
                int row = (int) v_norm[col];
                screen[H-row][col] = 'x';
        }
```

השורה הראשונה מציבה קו תחתי לאורך קו ה- 0.

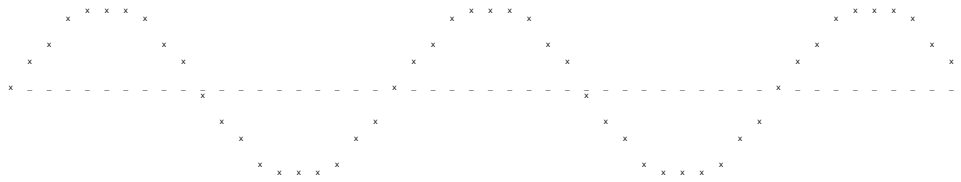
<u>הגדרת אות והצגתו</u>

כעת נכתוב תכנית היוצרת אות מסוים ומציגה אותו ע"י קריאה ל- plot():

```
#include <iostream>
#include <valarray>
```

```
#include <cmath>
#include <ctime>

using namespace std;

void test1()
{
	const double PI = 3.14159265359;
	const int SIZE = 50;		// sample size
	const double F1=5.0;			// frequencies
	const double dt = 0.01;
	valarray<double> t(SIZE);  // time axis
	valarray<double> sig(SIZE);		// signal

	// init time axis
	for(int i=0; i<t.size(); i++)
		t[i] = i * dt;
	sig = sin(2*PI*F1*t);		// create a sinus signal
	plot(sig);
}
```

הפלט:



<u>הסבר</u>: הפונקציה ראשית מגדירה מספר קבועים ושני מערכי valarray:

```
void test1()
{
	const double PI = 3.14159265359;
	const int SIZE = 50;		// sample size
	const double F1=5.0;		// frequencies
	const double dt = 0.01;
	valarray<double> t(SIZE);	// time axis
	valarray<double> sig(SIZE);	// signal
```

sig הוא מערך המייצג את האות על פני ציר הזמן, ו- t הוא מערך המייצג את נקודות הדגימה של האות על ציר הזמן. t מאותחל כך:

```
	for(int i=0; i<t.size(); i++)
		t[i] = i * dt;
```

כלומר, ערכי t יהיו {0, 0,01, 0.02, 0.03.....}.

בשלב הבא, מציבים לאות את הנוסחה sin(2 Π F t)

```
	sig = sin(2*PI*F1*t);
```

יש לשים לב שכאן בוצעה ההכפלה על כל מערך הזמנים t.

לבסוף האות מוצג למסך ע"י:

```
	plot(sig);
}
```

<u>עיבוד האות</u>

נוסיף כעת יכולת עיבוד לאות: ראשית, נגדיר פונקציה המוסיפה "רעש לבן" לאות, כלומר, מוסיפה ערכים אקראיים לאות:

```
template <class T>
void add_white_noise(valarray<T>& v)
```

```
{
	srand((unsigned)time(NULL));		// init random number generator seed

	for(int i=0; i<v.size(); i++)
		v[i] += (rand() % 256) / 128.0 - 1.0;
}
```

הפונקציה עושה שימוש במחולל המספרים האקראיים של הספרייה התקנית של C בכדי להוסיף ערכים אקראיים בתחום [-1..+1] למערך שבקלט.

כעת נכתוב פונקציה המבצעת עיבוד לאות, על מנת לשחזר את המידע המקורי, לפני הוספת הרעש: אנו נעשה שימוש בשיטת הממצע הנע (Moving Average) להגדרת מסנן מעביר נמוכים (Low-Pass Filter):

```
template <class T>
valarray<T> low_pass_filter(const valarray<T>& in)
{
	valarray<T> out = in;
	const int N = 5;
	double avg = 0;

	for(int i=0; i<N; i++)
		avg += in[i];
	avg /= N;

	for(i=N; i<in.size(); i++)
	{
		out[i] = avg;
		avg = avg - in[i-N]/N + in[i]/N;
	}
	return out;
}
```

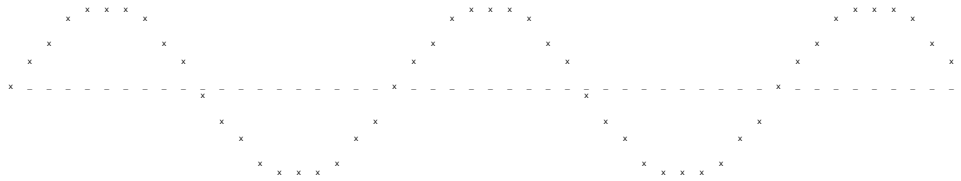
כפי שניתן לראות, הפונקציה low_pass_filter מסננת את הרעשים ע"י ממצע נע של 5 הקלטים האחרונים.

נוסיף כעת לפונקצית הבדיקה שתי פעולות: הכנסת רעש לאות המקורי, ושיחזורו לאחר מכן ע"י הפונקציות שכתבנו:

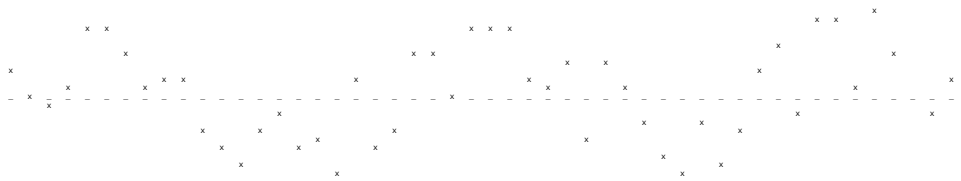
```
void test1()
{
	const double PI = 3.14159265359;
	const int SIZE = 50;		// sample size
	const double F1=5.0;		// frequencies
	const double dt = 0.01;
	valarray<double> t(SIZE);  // time axis
	valarray<double> sig(SIZE);		// signal
	valarray<double> sig_noise(SIZE);  // signal with noise
	valarray<double> sig_filtered(SIZE);		// filtered signal

	// init time axis
	for(int i=0; i<t.size(); i++)
		t[i] = i * dt;
	sig = sin(2*PI*F1*t);		// create a sinus signal

	sig_noise = sig;
	add_white_noise(sig_noise);		// add white noise
	plot(sig);
	plot(sig_noise);

	// filter
	sig_filtered = low_pass_filter(sig_noise);
	plot(sig_filtered);
}
```

פלט התכנית:

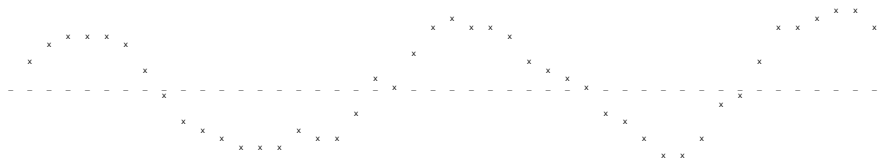
- האות המקורי:



- האות לאחר הוספת הרעש:



- האות המשוחזר:



כפי שניתן לראות, צורת האות שוחזרה ברמה סבירה.

<u>תרגול</u>

הגדר/י template למחלקת Signal שתכיל את כל הפונקציות הנ"ל כחברים בה. יש להגדיר כנתוני המחלקה את וקטור נתוני האות - valarray כללי - וכן את מערך ציר הזמן (valarray של ממשיים).

נסה/י לייעל את פעולת פונקצית עיבוד האות low_pass_filter() כך שיהיו מינימום פעולות חילוק. כמו כן, הגדר/י את N - גודל "חלון" הממוצע - כפרמטר לפונקציה.

הפעל/י את תכנית הבדיקה הבאה על המחלקה (מבצעת מה שביצעה התכנית הקודמת):

```
#include "signal.h"

void test2()
{
        const int SIZE = 50;                    // sample size
        const double F1=5.0;                    // frequency
        const double dt = 0.01;
        Signal<double> sig(SIZE, dt, F1);       // signal
        Signal<double> sig_noise;               // signal with noise
        Signal<double> sig_filtered;            // filtered signal

        sig_noise = sig;
        sig_noise.add_white_noise();            // add white noise
        sig.plot(50, 20);
        sig_noise.plot(50, 20);

        // filter
        sig_filtered = sig_noise.low_pass_filter(10);
        sig_filtered.plot(50,20);
}
```

סיכום

valarray הוא template של מיכל לתמיכה באריתמטיקה וקטורית. valarray תוכנן כמיכל אופטימלי לביצוע פעולות אלו, ולכן הוא בחירה טבעית ליישומים מדעיים כגון: עיבוד אות, פעולות על מטריצות, חישובי הסתברות ועוד.

התמיכה המובנית של valarray בפעולות הוקטוריות מספקת תחליף לשימוש בשפות עזר כמו Matlab, ומהווה יתרון בכך שהיא מובנית בשפת הפיתוח.

כל הזכויות שמורות © מאיר סלע

מרכז ההדרכה 2000