SP++3.0 User Guide

张明

2011-02

同心协力

共创开源

目录

0 SP++导引

0.1 SP++概述

SP++ (Signal Processing in C++) 是一个关于信号处理与数值计算的开源 C++程序库，该库提供了信号处理与数值计算中常用算法的 C++实现。

SP++中所有算法都以 C++类模板方法实现，以头文件形式组织而成，所以不需要用户进行本地编译，只要将相关的头文件包含在项目中即可使用。”XXX.h”表示声明文件，”XXX-impl.h”表示对应的实现文件。所有的函数和类均位于名字空间”splab”中，因此使用 SP++时要进行命名空间声明：”**using namespace splab**”。

SP++最早发表于开源中国社区，博客地址为：http://my.oschina.net/zmjerry。后来发布到Google Code，地址为：http://code.google.com/p/tspl/，因为名字冲突，所以项目名取为TSPL(Template Signal Processing Library)。

0.2 SP++安装（CodeBlocks）

1 将 SP++3.0 压缩包解压到某一路径下，如 D:\Program Files\SP++3.0；

2 打开 CodeBlocks，在 Settings->Compler and Debugger / Search directories / Compiler中加入D:\Program Files\SP++3.0\include，如图 0-1所示；

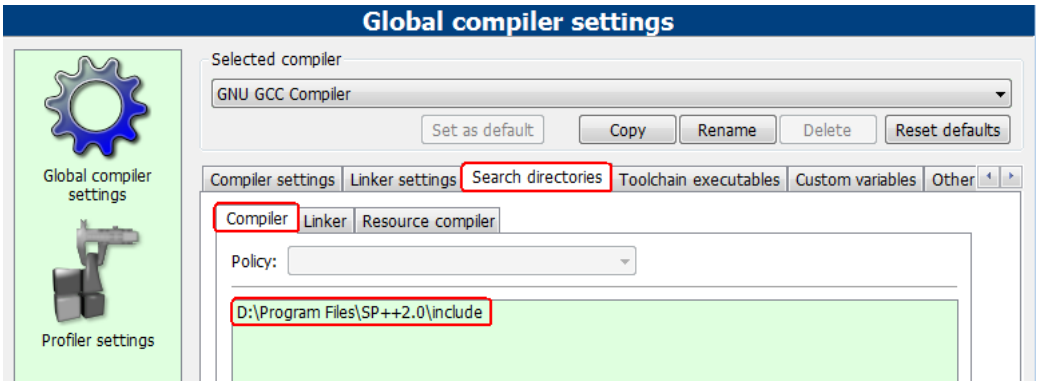

图 0-1

3 在 CodeBlocks 中建立 C++ 项目，例如 SP++3.0Test，代码如下：

```
1.  /*************************************************************************
2.   *                                fir_test.cpp
3.   *
```

```
4.    * FIR class testing.
5.    *
6.    * Zhang Ming, 2010-03, Xi'an Jiaotong University.
7.    ************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <fir.h>
14.
15.
16. using namespace std;
17. using namespace splab;
18.
19.
20. int main()
21. {
22.     string  wType = "Hamming";
23.
24. //    string  fType = "lowpass";
25. //    double  fs = 1000,
26. //            fpass = 200,
27. //            apass = -3,
28. //            fstop = 300,
29. //            astop = -20;
30. //    FIR fir( fType, wType );
31. //    fir.setParams( fs, fpass, apass, fstop, astop );
32.
33. //    string  fType = "highpass";
34. //    double  fs = 1000,
35. //            fstop = 200,
36. //            astop = -20,
37. //            fpass = 300,
38. //            apass = -3;
39. //    FIR fir( fType, wType );
40. //    fir.setParams( fs, fstop, astop, fpass, apass );
41.
42. //    string  fType = "bandpass";
43. //    double  fs = 1000,
44. //            fstop1 = 100,
45. //            astop1 = -20,
46. //            fpass1 = 200,
47. //            fpass2 = 300,
```

```
48. //            apass1 = -3,
49. //            fstop2 = 400,
50. //            astop2 = -20;
51. //    FIR fir( fType, wType );
52. //    fir.setParams( fs, fstop1, astop1, fpass1, fpass2, apass1, fstop2, astop2 );
53.
54.     string  fType = "bandstop";
55.     double  fs = 1000,
56.             fpass1 = 100,
57.             apass1 = -3,
58.             fstop1 = 200,
59.             fstop2 = 300,
60.             astop1 = -20,
61.             fpass2 = 400,
62.             apass2 = -3;
63.     FIR fir( fType, wType );
64.     fir.setParams( fs, fpass1, apass1, fstop1, fstop2, astop1, fpass2, apass2 );
65.
66.     fir.design();
67.     fir.dispInfo();
68.
69.     cout << endl;
70.     return 0;
71. }
```

运行结果：

```
1.                      Filter selectivity      :  bandstop
2.                      Window type             :  Hamming
3.                      Sampling Frequency (Hz) :  1000
4.                      Lower passband frequency (Hz) :  100
5.                      Lower passband gain      (dB) :  -3
6.                      Lower stopband frequency (Hz) :  200
7.                      Upper stopband frequency (Hz) :  300
8.                      Stopband gain            (dB) :  -20
9.                      Upper passband frequency (Hz) :  400
10.                     Upper passband gain      (dB) :  -3
11.
12.
13.                                 Filter Coefficients
14.
15.    N    [     N + 0           N + 1           N + 2           N + 3     ]
16.   ===   ================================================================
17.    0    -3.85192764e-003  8.42472981e-003  3.11153775e-003 -2.03549729e-003
```

```
    4    -3.55185234e-002  2.30171390e-002 -1.41331145e-001  2.61755439e-001
    8     2.88881794e-002  3.56158158e-001  3.56158158e-001  2.88881794e-002
   12     2.61755439e-001 -1.41331145e-001  2.30171390e-002 -3.55185234e-002
   16    -2.03549729e-003  3.11153775e-003  8.42472981e-003 -3.85192764e-003


          ==================== Edge Frequency Response ====================
              Mag(fp1) = -0.85449456(dB)       Mag(fp2) = -0.80635855(dB)
              Mag(fs1) = -21.347821(dB)       Mag(fs2) = -21.392825(dB)


Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```

0.3 SP++安装（VS2010）

1 将 SP++3.0 压缩包解压到某一路径下，如 D:\Program Files\SP++3.0；

2 建立VS2010 项目，在Project->Propertiesr的VC++Directories->Include Directories 中加入D:\Program Files\SP++3.0\include，如图 0-2所示；

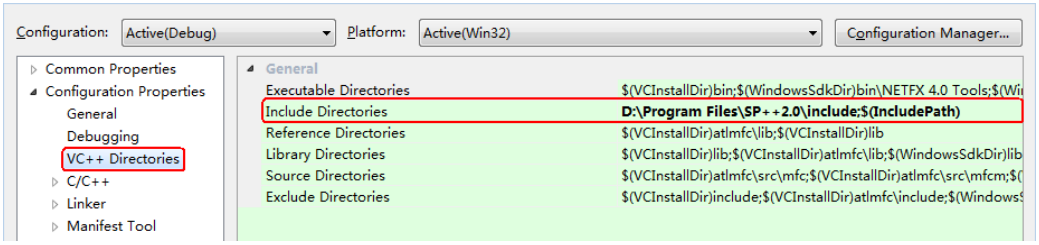

图 0-2

3 在 VS2010 中建立 C++ 项目，例如 SP++3.0Test，代码如下：

```
/*****************************************************************************
 *                              inverse_test.cpp
 *
 * Matrix inverse testing.
 *
 * Zhang Ming, 2010-08 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
```

```
13. #include <iomanip>
14. #include <inverse.h>
15. 
16. 
17. using namespace std;
18. using namespace splab;
19. 
20. 
21. typedef double  Type;
22. const   int     N = 3;
23. 
24. int main()
25. {
26.     Matrix<Type> A, invA, B(N,N);
27.     A.resize(3,3);
28.     A[0][0] = 1;    A[0][1] = 2;    A[0][2] = 1;
29.     A[1][0] = 2;    A[1][1] = 5;    A[1][2] = 4;
30.     A[2][0] = 1;    A[2][1] = 1;    A[2][2] = 0;
31. 
32.     cout << setiosflags(ios::fixed) << setprecision(4);
33.     cout << "The original matrix A is : " << A << endl;
34.     invA = inv(A);
35.     cout << "The inverse matrix of A (LUD) : " << invA << endl;
36.     invA = colPivInv(A);
37.     cout << "The inverse matrix of A (column pivot) : " << invA << endl;
38.     invA = cmpPivInv(A);
39.     cout << "The invese matrix of A (complete pivot) : " << invA << endl;
40.     cout << "The multiplication of A and its inverse: " << A*invA << endl;
41. 
42.     for( int i=1; i<=N; ++i )
43.     {
44.         for( int j=1; j<=N; ++j )
45.             if( i == j )
46.                 B(i,i) = i;
47.             else if( i < j )
48.                 B(i,j) = i;
49.             else
50.                 B(i,j) = j;
51.     }
52.     cout << "The original matrix B is : " << B << endl;
53.     invA = inv(B,"spd");
54.     cout << "The inverse matrix of B (Cholesky) : " << invA << endl;
55.     invA = colPivInv(B);
56.     cout << "The inverse matrix of B (column pivot) : " << invA << endl;
```

```
57.     invA = cmpPivInv(B);
58.     cout << "The inverse matrix of B (complete pivot) : " << invA << endl;
59.     cout << "The multiplication of B and its inverse: " << B*invA << endl;
60.
61.     cout << setiosflags(ios::fixed) << setprecision(3);
62.     Matrix<complex<Type> > cA(N,N), cIA;
63.     cA = complexMatrix( A, B );
64.
65.     cIA = inv(cA);
66.     cout << "The original complex matrix cA is: " << cA << endl;
67.     cout << "The inverse matrix of cA is (general method): "
68.          << inv(cA) << endl;
69.     cout << "The inverse matrix of cA is (column pivot): "
70.          << colPivInv(cA) << endl;
71.     cout << "The inverse matrix of cA is (complete pivot): "
72.          << cmpPivInv(cA) << endl;
73.     cout << "The inverse matrix of cA is (real inverse): "
74.          << cinv(cA) << endl;
75.     cout << "The multiplication of cA and its inverse: "
76.          << cA*cIA << endl;
77.
78.     return 0;
79. }
```

运行结果：

```
1.  The original matrix A is : size: 3 by 3
2.  1.0000  2.0000  1.0000
3.  2.0000  5.0000  4.0000
4.  1.0000  1.0000  0.0000
5.
6.  The inverse matrix of A (LUD) : size: 3 by 3
7.  -4.0000 1.0000  3.0000
8.  4.0000  -1.0000 -2.0000
9.  -3.0000 1.0000  1.0000
10.
11. The inverse matrix of A (column pivot) : size: 3 by 3
12. -4.0000 1.0000  3.0000
13. 4.0000  -1.0000 -2.0000
14. -3.0000 1.0000  1.0000
15.
16. The invese matrix of A (complete pivot) : size: 3 by 3
17. -4.0000 1.0000  3.0000
18. 4.0000  -1.0000 -2.0000
19. -3.0000 1.0000  1.0000
```

```
20.
21. The multiplication of A and its inverse: size: 3 by 3
22. 1.0000  0.0000  0.0000
23. 0.0000  1.0000  -0.0000
24. 0.0000  0.0000  1.0000
25.
26. The original matrix B is : size: 3 by 3
27. 1.0000  1.0000  1.0000
28. 1.0000  2.0000  2.0000
29. 1.0000  2.0000  3.0000
30.
31. The inverse matrix of B (Cholesky) : size: 3 by 3
32. 2.0000  -1.0000 0.0000
33. -1.0000 2.0000  -1.0000
34. 0.0000  -1.0000 1.0000
35.
36. The inverse matrix of B (column pivot) : size: 3 by 3
37. 2.0000  -1.0000 -0.0000
38. -1.0000 2.0000  -1.0000
39. 0.0000  -1.0000 1.0000
40.
41. The inverse matrix of B (complete pivot) : size: 3 by 3
42. 2.0000  -1.0000 0.0000
43. -1.0000 2.0000  -1.0000
44. -0.0000 -1.0000 1.0000
45.
46. The multiplication of B and its inverse: size: 3 by 3
47. 1.0000  0.0000  0.0000
48. 0.0000  1.0000  0.0000
49. 0.0000  0.0000  1.0000
50.
51. The original complex matrix cA is: size: 3 by 3
52. (1.000,1.000)   (2.000,1.000)   (1.000,1.000)
53. (2.000,1.000)   (5.000,2.000)   (4.000,2.000)
54. (1.000,1.000)   (1.000,2.000)   (0.000,3.000)
55.
56. The inverse matrix of cA is (general method): size: 3 by 3
57. (1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
58. (-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
59. (1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)
60.
61. The inverse matrix of cA is (column pivot): size: 3 by 3
62. (1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
63. (-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
```

```
64. (1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)
65.
66. The inverse matrix of cA is (complete pivot): size: 3 by 3
67. (1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
68. (-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
69. (1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)
70.
71. The inverse matrix of cA is (real inverse): size: 3 by 3
72. (1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
73. (-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
74. (1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)
75.
76. The multiplication of cA and its inverse: size: 3 by 3
77. (1.000,0.000)   (0.000,-0.000)  (-0.000,0.000)
78. (0.000,0.000)   (1.000,-0.000)  (0.000,0.000)
79. (0.000,0.000)   (0.000,-0.000)  (1.000,0.000)
80.
81. 请按任意键继续. . .
```

0.4 SP++与 Matlab 混合编程

1 建立C++项目，在项目管理窗口中项目名称上单击右键，选择最下边的Properties选项，如图 0-3所示；

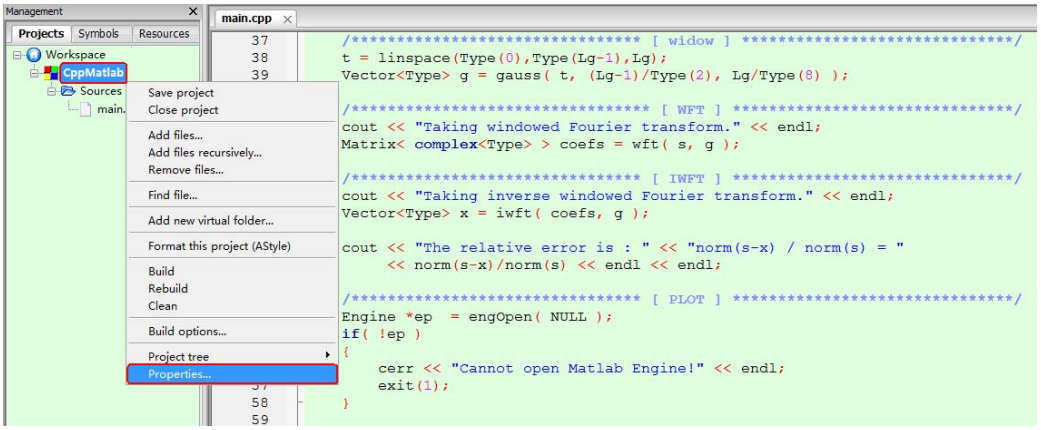

图 0-3

2 在打开的窗口中选择右下角的Project’s build options选项，如图 0-4所示；

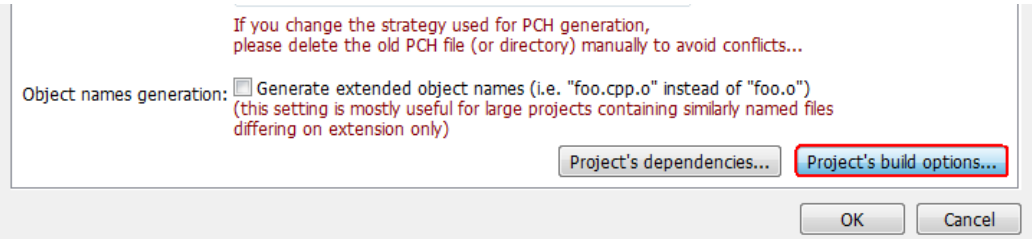

图 0-4

3 在打开的窗口中选择Search directories选项中的Compliler选项，在其中加入SP++3.0 与Matlab\extern中的include目录，如图 0-5所示，注意：此处必须先加载SP++3.0\include，然后再加载Matalb\extern\inlcude，因为SP++3.0 与Matlab中都有matrix.h头文件，如果加载顺序相反，则无法通过编译；

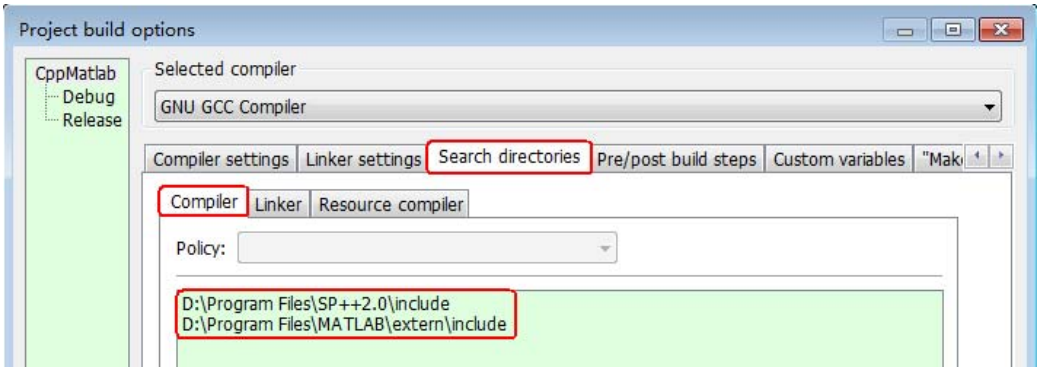

图 0-5

4 同上步，在打开的窗口中选择Search directories选项中的Linker选项，在其中加入Matlab中相关的lib目录，如图 0-6所示；

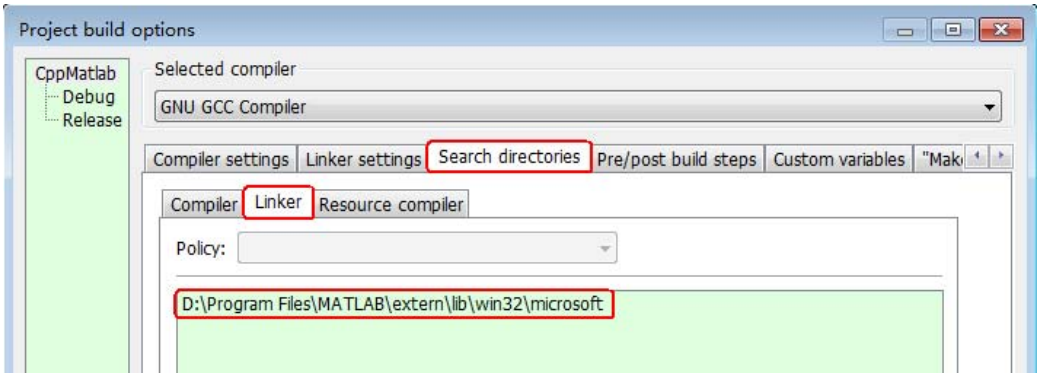

图 0-6

5 同上步，在打开的窗口中选择Linker setting选项，在其中加入Matlab中相关的lib文件，如图 0-7所示；

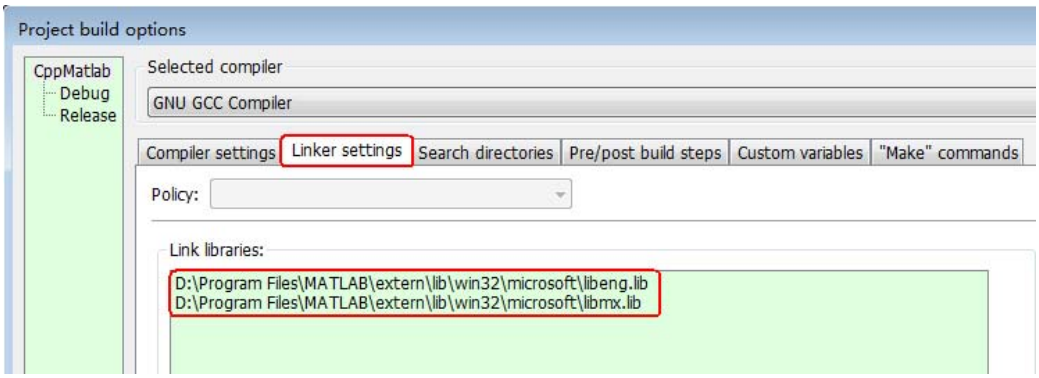

图 0-7

6　在项目中运行如下测试代码；

运行结果：

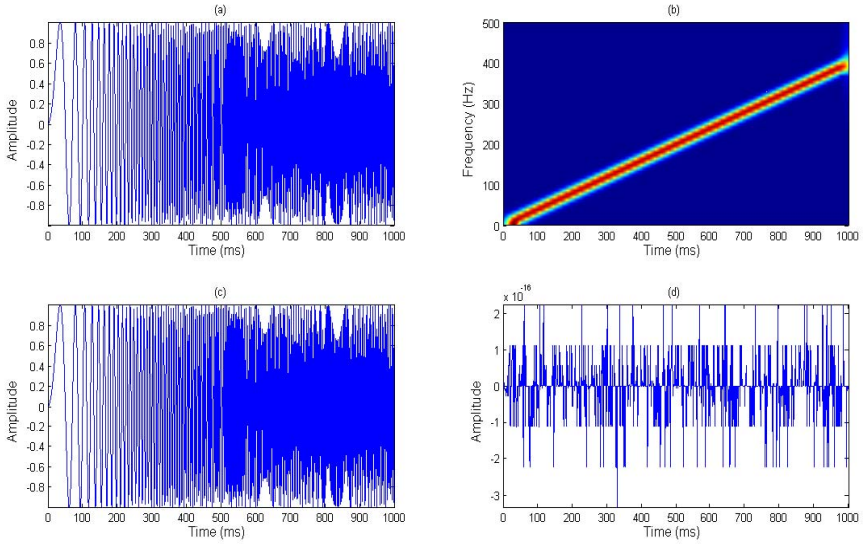

图 0-8

1 向量类模板

1.1 基本向量类

向量类模板Vector<Type>是专门为线性代数中向量运算而设计的一个模板类，支持实数向量和复数向量的各种运算。包含了向量的构造与析构，见表 1-1；向量类的基本属性提取，见表 1-2；向量常用计算的运算符重载，见表 1-3；以及一些其它常用函数，见表 1-4。

为了表示方便，表中所涉及的变量均未注明类型，可根据变量名称粗略地判断其类型，如习惯用 v 来表示向量，用 a 来表示数组，用 n 来表示元素个数等等。具体的函数声明与定义可以参见“vector.h”和“vector-impl.h”。

表 1-1 向量类的构造与析构函数

Operation	Effect
Vector<Type> v	创建一个空向量
Vecto<Type> v1(v2)	创建向量 v2 的拷贝 v1
Vector<Type> v(n,x)	创建常数向量
Vector<Type> v(n,a)	通过数组创建向量
v.~Vectro<Type>()	销毁向量并释放空间

表 1-2 向量类的属性获取

Operation	Effect
v.Type*()	向量到数组指针的类型转换
v.begin()	获取第一个元素的迭代器
v.end	获取最后一个元素下一位的迭代器
v.dim()	获取向量的维数
v.size()	获取向量的大小
v.resize(n)	重新设置向量的大小

表 1-3 向量类中重载的运算符

Operation	Effect
v1 = v2	向量对向量赋值
v = x	常数对向量赋值
v[i]	0 偏移下标访问
v(i)	1 偏移下标访问
-v	全部元素取反
v += x	向量自身加常数

v -= x	向量自身减常数
v *= x	向量自身乘常数
v /= x	向量自身除以常数
v1 += v2	向量自身加向量
v1 -= v2	向量自身减向量
v1 *= v2	向量自身乘向量(逐元素)
v1 /= v2	向量自身除以向量(逐元素)
v + x	向量与常数之和
x + v	常数与向量之和
v1 + v2	向量与向量之和
v - x	向量与常数之差
x - v	常数与向量之差
v1 - v2	向量与向量之差
v * x	向量与常数之积
x * v	常数与向量之积
v1 * v2	向量与向量之积(逐元素)
v / x	向量与常数之商
x / v	常数与向量之商
v1 / v2	向量与向量之商(逐元素)
>> v	输入向量
<< v	输出向量

表 1-4　向量类的其它函数

Operation	Effect
sum(v)	向量的元素之和
min(v)	求向量的最小元素
max(v)	求向量的最大元素
swap(v1,v2)	交换两向量的元素
norm(v)	向量的 L2 范数
dotProd(v1,v2)	向量的内积
linspace(a,b,n)	产生等差数列
abs(cv)	求复数向量的模值
arg(cv)	求复数向量的相角
real(cv)	求复数向量的实部
imag(cv)	求复数向量的虚部
complexVector(vr)	将实向量转为复向量
compltxVector(vr,vi)	通过实向量构造复向量

测试代码：

```
1.  /*****************************************************************************
2.   *                                vector_test.cpp
3.   *
4.   * Vector class testing.
```

```
5.    *
6.    * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
7.    ************************************************************************/
8.
9.
10.  #define BOUNDS_CHECK
11.
12.  #include <iostream>
13.  #include <iomanip>
14.  #include <complex>
15.  #include <vector.h>
16.
17.
18.  using namespace std;
19.  using namespace splab;
20.
21.
22.  const int M = 3;
23.
24.
25.  void display( const int *p, int length )
26.  {
27.      for( int i=0; i<length; ++i )
28.          cout << p[i] << "\t" ;
29.      cout << endl;
30.  }
31.
32.
33.  int main()
34.  {
35.      int k;
36.      int arrays[3] = {1,2,3};
37.
38.      Vector<int> v1( M,arrays );
39.      k = 1;
40.      Vector<int> v2( M,k );
41.
42.      Vector<int> v3( M );
43.      k = 0;
44.      v3 = k;
45.
46.      Vector<int> v4( v1 );
47.
48.      cout << "vector v1 : " << v1 << endl;
```

```
49.     cout << "vector v2 : " << v2 << endl;
50.     cout << "vector v3 : " << v3 << endl;
51.     cout << "vector v4 : " << v4 << endl;
52.
53.     display( v4, M );
54.     cout << endl;
55.
56.     v4.resize( 5 );
57.     Vector<int>::iterator itr = v4.begin();
58.     while( itr != v4.end() )
59.         *itr++ = 1;
60.     cout << "new vector v4 : " << v4 << endl;
61.     v4 = v1;
62.     cout << "new vector v4 : " << v4 << endl;
63.
64.     k = 2;
65.     v3 = k+v1;
66.     cout << "v3 = k + v1 : " << v3 << endl;
67.     v3 += k;
68.     cout << "v3 += k : " << v3 << endl;
69.
70.     v3 = v1-k;
71.     cout << "v3 = v1 - k : " << v3 << endl;
72.     v3 = k-v1;
73.     cout << "v3 = k - v1 : " << v3 << endl;
74.     v3 -= k;
75.     cout << "v3 -= k : " << v3 << endl;
76.
77.     v3 = k*v1;
78.     cout << "v3 = k * v1 : " << v3 << endl;
79.     v3 *= k;
80.     cout << "v3 *= k : " << v3 << endl;
81.
82.     v3 = v1/k;
83.     cout << "v3 = v1 / k : " << v3 << endl;
84.     v3 = k/v1;
85.     cout << "v3 = k / v1 : " << v3 << endl;
86.     v3 /= k;
87.     cout << "v3 /= k : " << v3 << endl;
88.
89.     v3 = v1+v2;
90.     cout << "v3 = v1 + v2 : " << v3 << endl;
91.     v3 += v1;
92.     cout << "v3 += v1 : " << v3 << endl;
```

```
93.
94.     v3 = v1-v2;
95.     cout << "v3 = v1 - v2 : " << v3 << endl;
96.     v3 -= v1;
97.     cout << "v3 -= v1 : " << v3 << endl;
98.
99.     v3 = v1*v2;
100.    cout << "v3 = v1 * v2 : " << v3 << endl;
101.    v3 *= v1;
102.    cout << "v3 *= v1 : " << v3 << endl;
103.
104.    v3 = v1/v2;
105.    cout << "v3 = v1 / v2 : " << v3 << endl;
106.    v3 /= v1;
107.    cout << "v3 /= v1 : " << v3 << endl;
108.
109.    cout << "minimum element of v1 :   " << min(v1) << endl << endl;
110.    cout << "maximum element of v1 :   " << max(v1) << endl << endl;
111.    cout << "L2 norm of v3 :   " << norm( v3 ) << endl << endl;
112.    cout << "inner product of v1 and v2 :   "
113.         << dotProd( v1, v2 ) << endl << endl;
114.
115.    complex<double> z = -1.0;
116.    Vector< complex<double> > v( M );
117.    v[0] = polar( 1.0,PI/4 );
118.    v[1] = polar( 1.0,PI );
119.    v[2] = complex<double>( 1.0,-1.0 );
120.    Vector< complex<double> > u = v*z;
121.
122.    cout << setiosflags(ios::fixed) << setprecision(4);
123.    cout << "convert from real to complex vector: "
124.         << complexVector(v3) << endl;
125.    cout << "convert from real to complex vector: "
126.         << complexVector(v3,-v1) << endl;
127.    cout << "complex vector v : " << v << endl;
128.    cout << "complex vector u = -v : " << u << endl;
129.    cout << "norm of coplex vector v : " << norm(v) << endl << endl;
130.    cout << "dot product of complex vector v and 1+u: "
131.         << dotProd(v,u-z) << endl << endl;
132.
133.    int N = 5;
134.    Vector<double> x = linspace( 0.0, TWOPI, N );
135.    Vector< complex<float> > cv(N);
136.    for( int i=0; i<N; ++i )
```

```
137.         cv[i] = complex<float>( float(sin(x[i])), float(cos(x[i])) );
138.     cout << "Complex vector vc : " << cv << endl;
139.     cout << "Absolute of vc : " << abs(cv) << endl;
140.     cout << "Angle of vc : " << arg(cv) << endl;
141.     cout << "Real part of vc : " << real(cv) << endl;
142.     cout << "Imaginary part of vc : " << imag(cv) << endl;
143.
144.     cout << resetiosflags(ios::fixed);
145.     Vector< Vector<double> > v2d1( M );
146.     for( int i=0; i<M; ++i )
147.     {
148.         v2d1[i].resize( M );
149.         for( int j=0; j<M; ++j )
150.             v2d1[i][j] = double( i+j );
151.     }
152.
153.     cout << "two dimension vector v2d1 : " << endl;
154.     Vector< Vector<double> >::const_iterator itrD2 = v2d1.begin();
155.     int rowNum = 0;
156.     while( itrD2 != v2d1.end() )
157.         cout << "the " << rowNum++ << "th row : " << *itrD2++ << endl;
158.
159.     Vector< Vector<double> > v2d2 = v2d1+v2d1;
160.     cout << "two dimension vector v2d2 = v2d1 + v2d1 : " << v2d2;
161.
162.
163.     return 0;
164. }
```

运行结果：

```
1.  vector v1 : size: 3 by 1
2.  1
3.  2
4.  3
5.
6.  vector v2 : size: 3 by 1
7.  1
8.  1
9.  1
10.
11. vector v3 : size: 3 by 1
12. 0
13. 0
14. 0
```

```
15.
16. vector v4 : size: 3 by 1
17. 1
18. 2
19. 3
20.
21. 1       2       3
22.
23. new vector v4 : size: 5 by 1
24. 1
25. 1
26. 1
27. 1
28. 1
29.
30. new vector v4 : size: 3 by 1
31. 1
32. 2
33. 3
34.
35. v3 = k + v1 : size: 3 by 1
36. 3
37. 4
38. 5
39.
40. v3 += k : size: 3 by 1
41. 5
42. 6
43. 7
44.
45. v3 = v1 - k : size: 3 by 1
46. -1
47. 0
48. 1
49.
50. v3 = k - v1 : size: 3 by 1
51. 1
52. 0
53. -1
54.
55. v3 -= k : size: 3 by 1
56. -1
57. -2
58. -3
```

```
59.
60. v3 = k * v1 : size: 3 by 1
61. 2
62. 4
63. 6
64.
65. v3 *= k : size: 3 by 1
66. 4
67. 8
68. 12
69.
70. v3 = v1 / k : size: 3 by 1
71. 0
72. 1
73. 1
74.
75. v3 = k / v1 : size: 3 by 1
76. 2
77. 1
78. 0
79.
80. v3 /= k : size: 3 by 1
81. 1
82. 0
83. 0
84.
85. v3 = v1 + v2 : size: 3 by 1
86. 2
87. 3
88. 4
89.
90. v3 += v1 : size: 3 by 1
91. 3
92. 5
93. 7
94.
95. v3 = v1 - v2 : size: 3 by 1
96. 0
97. 1
98. 2
99.
100. v3 -= v1 : size: 3 by 1
101. -1
102. -1
```

```
-1

v3 = v1 * v2 : size: 3 by 1
1
2
3

v3 *= v1 : size: 3 by 1
1
4
9

v3 = v1 / v2 : size: 3 by 1
1
2
3

v3 /= v1 : size: 3 by 1
1
1
1

minimum element of v1 :   1

maximum element of v1 :   3

L2 norm of v3 :   1

inner product of v1 and v2 :   6

convert from real to complex vector: size: 3 by 1
(1,0)
(1,0)
(1,0)

convert from real to complex vector: size: 3 by 1
(1,-1)
(1,-2)
(1,-3)

complex vector v : size: 3 by 1
(0.7071,0.7071)
(-1.0000,0.0000)
(1.0000,-1.0000)
```

```
147.
148. complex vector u = -v : size: 3 by 1
149. (-0.7071,-0.7071)
150. (1.0000,-0.0000)
151. (-1.0000,1.0000)
152.
153. norm of coplex vector v : 2.0000
154.
155. dot product of complex vector v and 1+u: (-3.2929,-0.2929)
156.
157. Complex vector vc : size: 5 by 1
158. (0.0000,1.0000)
159. (1.0000,0.0000)
160. (0.0000,-1.0000)
161. (-1.0000,-0.0000)
162. (-0.0000,1.0000)
163.
164. Absolute of vc : size: 5 by 1
165. 1.0000
166. 1.0000
167. 1.0000
168. 1.0000
169. 1.0000
170.
171. Angle of vc : size: 5 by 1
172. 1.5708
173. 0.0000
174. -1.5708
175. -3.1416
176. 1.5708
177.
178. Real part of vc : size: 5 by 1
179. 0.0000
180. 1.0000
181. 0.0000
182. -1.0000
183. -0.0000
184.
185. Imaginary part of vc : size: 5 by 1
186. 1.0000
187. 0.0000
188. -1.0000
189. -0.0000
190. 1.0000
```

```
191.
192. two dimension vector v2d1 :
193. the 0th row : size: 3 by 1
194. 0
195. 1
196. 2
197.
198. the 1th row : size: 3 by 1
199. 1
200. 2
201. 3
202.
203. the 2th row : size: 3 by 1
204. 2
205. 3
206. 4
207.
208. two dimension vector v2d2 = v2d1 + v2d1 : size: 3 by 1
209. size: 3 by 1
210. 0
211. 2
212. 4
213.
214. size: 3 by 1
215. 2
216. 4
217. 6
218.
219. size: 3 by 1
220. 4
221. 6
222. 8
223.
224.
225. Process returned 0 (0x0)   execution time : 0.151 s
226. Press any key to continue.
```

1.2 常用数学函数的向量版本

为了便于数值计算，将一些信号处理中经常使用的数学函数进行了重载，使之能够运用于向量对象，如表 1-5所示。

表 1-5　常用函数的向量版本（逐元素）

Operation	Effect
cos(v)	向量的余弦
sin (v)	向量的正弦
tan (v)	向量的正切
acos (v)	向量的反余弦
asin (v)	向量的反正弦
atan (v)	向量的反正切
exp (v)	向量的幂函数
log (v)	向量的自然对数函数
log10 (v)	向量以 10 为底的对数函数
sqrt (v)	向量的平方根
pow (v,v)	向量的向量幂函数
pow (v,p)	向量的常数幂函数
pow (b,v)	常数的向量幂函数
gauss (v,u,r)	向量的 Gauss 函数

测试代码：

```
1.  /*****************************************************************************
2.   *                              vectormath_test.cpp
3.   *
4.   * Math functions of vector testing.
5.   *
6.   * Zhang Ming, 2010-08, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <vectormath.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. int main()
22. {
23.
24.     int     N = 5;
25.     double  a = 0,
```

```
            b = 2*PI;
    Vector<double> x = linspace( a, b, N );

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << x << endl;
    cout << "sin of x : " << sin(x) << endl;
    cout << "cos of x : "<< cos(x) << endl;
    cout << "tan of x : " << tan(x) << endl;
    cout << "asin of x : "<< asin(x) << endl;
    cout << "acos of x : " << acos(x) << endl;
    cout << "atan of x : " << atan(x) << endl;

    cout << "exp of x : "<< exp(x) << endl;
    cout << "log of x : " << log(x) << endl;
    cout << "log10 of x : " << log10(x) << endl;

    a = 2.0;
    cout << "sqrt of x : "<< sqrt(x) << endl;
    cout << "pow of x : " << pow(x,x) << endl;
    cout << "pow of x : " << pow(x,a) << endl;
    cout << "pow of x : " << pow(a,x) << endl;

    cout << "The standard normal distribution : " << gauss( x, a, b ) << endl;

    return 0;
}
```

运行结果：

```
size: 5 by 1
0.0000
1.5708
3.1416
4.7124
6.2832

sin of x : size: 5 by 1
0.0000
1.0000
0.0000
-1.0000
-0.0000

cos of x : size: 5 by 1
1.0000
```

```
0.0000
-1.0000
-0.0000
1.0000

tan of x : size: 5 by 1
0.0000
16331778728383844.0000
-0.0000
5443926242794615.0000
-0.0000

asin of x : size: 5 by 1
0.0000
nan
nan
nan
nan

acos of x : size: 5 by 1
1.5708
nan
nan
nan
nan

atan of x : size: 5 by 1
0.0000
1.0039
1.2626
1.3617
1.4130

exp of x : size: 5 by 1
1.0000
4.8105
23.1407
111.3178
535.4917

log of x : size: 5 by 1
-inf
0.4516
1.1447
```

```
1.5502
1.8379

log10 of x : size: 5 by 1
-inf
0.1961
0.4971
0.6732
0.7982

sqrt of x : size: 5 by 1
0.0000
1.2533
1.7725
2.1708
2.5066

pow of x : size: 5 by 1
1.0000
2.0327
36.4622
1487.9012
103540.9204

pow of x : size: 5 by 1
0.0000
2.4674
9.8696
22.2066
39.4784

pow of x : size: 5 by 1
1.0000
2.9707
8.8250
26.2162
77.8802

The standard normal distribution : size: 5 by 1
0.0604
0.0633
0.0625
0.0578
0.0503
```

```
105.
106.
107. Process returned 0 (0x0)   execution time : 0.094 s
108. Press any key to continue.
```

1.3 常用的辅助函数

Matlab提供了非常丰富的工具箱，包含了众多研究领域中的常用函数，表 1-6列出了Matlab信号处理工具箱中经常使用的一些函数的C++实现。

表 1-6 Matlab 中常用函数

Operation	Effect
mod(m,n)	取 m 对 n 的非负模值
ceil(m,n)	对 m/n 进行向上取整
reverse(v)	向量反转
flip(v)	向量反转
shift(v,n)	补零移位
cirshift(v,n)	循环移位
fftshift(v,n)	FFT 移位
dyadUp(v,oe)	二进上采样
dyadDown(v,oe)	二进下采样
fftInterp(v,factor)	信号频域内插
wkeep(v,n,first)	提取向量的部分元素
wkeep(v,n,direct)	提取向量的部分元素
wextend(v,n,direct,mode)	向量延拓

测试代码：

```
1.  /*****************************************************************************
2.   *                                utilities_test.cpp
3.   *
4.   * Utilities testing.
5.   *
6.   * Zhang Ming, 2010-01, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <string>
```

```
#include <utilities.h>


using namespace std;
using namespace splab;


const int N = 5;


int main()
{

    Vector<int> v1(N);
    for( int i=1; i<=v1.dim(); i++ )
        v1(i) = i;
    cout << "vector v1 : " << v1 << endl;
    Vector<int> v2(N);
    for( int i=1; i<=v2.dim(); ++i )
        v2(i) = i+N;
    cout << "vector v2 : " << v2 << endl;

    int N = 11;
    double a = 0;
    double b = 1.0;
    Vector<double> x = linspace( a, b, N );
    cout << N << " points linearly spaced from 0 to 1.0"
         << x << endl;

    cout << "Flipping vector v1 from left to right : " << flip(v1) << endl;
    cout << "Shift vector v1 from left to right : " << shift(v1,2) << endl;
    cout << "Shift vector v1 from right to left : " << shift(v1,-2) << endl;
    cout << "Circle shift vector v1 from left to right : "
         << circshift(v1,2) << endl;
    cout << "Circle shift vector v1 from right to left : "
         << circshift(v1,-2) << endl;
    cout << "FFT shift of vector : " << fftshift(v1) << endl;

    cout << "Dyadic upsampling of vector v1 by zeros at the even position : "
         << dyadUp( v1,0 ) << endl;
    cout << "Dyadic upsampling of vector v1 by zeros at the odd position : "
         << dyadUp( v1,1 ) << endl;
    cout << "Dyadic downsampling of vector v1 by zeros at the even position : "
         << dyadDown( v1,0 ) << endl;
```

```
59.     cout << "Dyadic downsampling of vector v1 by zeros at the odd position : "
60.          << dyadDown( v1,1 ) << endl;
61.
62.     Vector<float> sn(N);
63.     Vector< complex<float> > cn(N);
64.     for( int i=0; i<N; ++i )
65.     {
66.         sn[i] = float(sin(i*TWOPI/N));
67.         cn[i] = complex<float>( float(sin(i*TWOPI/N)), float(cos(i*TWOPI/N)) );
68.     }
69.     cout << setiosflags(ios::fixed) << setprecision(4);
70.     cout << "real signal sn : " << sn << endl;
71.     cout << "FFT interpolation of sn by factor fo 2 : "
72.          << fftInterp( sn, 2 ) << endl;
73.     cout << "complex signal cn : " << cn << endl;
74.     cout << "FFT interpolation of sn by factor fo 2 : "
75.          << fftInterp( cn, 2 ) << endl;
76.     cout << resetiosflags(ios::fixed);
77.
78.     int n = 2;
79.     string dire = "left";
80.     string mode = "zpd";
81.     cout << "Extending vector v1 in left direction by zeros padding : "
82.          << wextend( v1,n,dire,mode ) << endl;
83.     mode = "ppd";
84.     cout << "Extending vector v1 in left direction by periodic mode : "
85.          << wextend( v1,n,dire,mode ) << endl;
86.     mode = "sym";
87.     cout << "Extending vector v1 in left direction by symmetric mode : "
88.          << wextend( v1,n,dire,mode ) << endl;
89.
90.     dire = "right";
91.     mode = "zpd";
92.     cout << "Extending vector v1 in right direction by zeros padding : "
93.          << wextend( v1,n,dire,mode ) << endl;
94.     mode = "ppd";
95.     cout << "Extending vector v1 in right direction by periodic mode : "
96.          << wextend( v1,n,dire,mode ) << endl;
97.     mode = "sym";
98.     cout << "Extending vector v1 in right direction by symmetric mode : "
99.          << wextend( v1,n,dire,mode ) << endl;
100.
101.    dire = "both";
102.    mode = "zpd";
```

```
103.     cout << "Extending vector v1 in both direction by zeros padding : "
104.          << wextend( v1,n,dire,mode ) << endl;
105.     mode = "ppd";
106.     cout << "Extending vector v1 in both direction by periodic mode : "
107.          << wextend( v1,n,dire,mode ) << endl;
108.     mode = "sym";
109.     cout << "Extending vector v1 in both direction by symmetric mode : "
110.          << wextend( v1,n,dire,mode ) << endl;
111.
112.     cout << "Keeping the center part of vector v1 : "
113.          << wkeep( v1,3,"center" ) << endl;
114.     cout << "Keeping the left part of vector v1 : "
115.          << wkeep( v1,3,"left" ) << endl;
116.     cout << "Keeping the right part of vector v1 : "
117.          << wkeep( v1,3,"right" ) << endl;
118.     cout << "Keeping the first(2) to first + L(3) elements of vector v1 : "
119.          << wkeep( v1,3,2 ) << endl;
120.
121.     cout << "The modulus of 2 divided by 5 is " << mod(2,5) << "." << endl;
122.     cout << "The modulus of -1 divided by 5 is " << mod(-1,5) << "." << endl;
123.     cout << endl;
124.     cout << "The nearest integer >= 10/2 is " << ceil(10,2) << "." << endl;
125.     cout << "The nearest integer >= 10/3 is " << ceil(10,3) << "." << endl;
126.
127.     cout << endl;
128.     cout << "The numbers can be represented by the integer power of 2 "
129.          << "from 0 to 1000 are : " << endl;
130.
131.     return 0;
132. }
```

运行结果：

```
1.  vector v1 : size: 5 by 1
2.  1
3.  2
4.  3
5.  4
6.  5
7.
8.  vector v2 : size: 5 by 1
9.  6
10. 7
11. 8
12. 9
```

```
10

11 points linearly spaced from 0 to 1.0size: 11 by 1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Flipping vector v1 from left to right : size: 5 by 1
5
4
3
2
1

Shift vector v1 from left to right : size: 5 by 1
0
0
1
2
3

Shift vector v1 from right to left : size: 5 by 1
3
4
5
0
0

Circle shift vector v1 from left to right : size: 5 by 1
4
5
1
2
3

Circle shift vector v1 from right to left : size: 5 by 1
```

```
3
4
5
1
2

FFT shift of vector : size: 5 by 1
4
5
1
2
3

Dyadic upsampling of vector v1 by zeros at the even position : size: 11 by 1
0
1
0
2
0
3
0
4
0
5
0

Dyadic upsampling of vector v1 by zeros at the odd position : size: 9 by 1
1
0
2
0
3
0
4
0
5

Dyadic downsampling of vector v1 by zeros at the even position : size: 3 by 1
1
3
5

Dyadic downsampling of vector v1 by zeros at the odd position : size: 2 by 1
2
```

```
4

real signal sn : size: 11 by 1
0.0000
0.5406
0.9096
0.9898
0.7557
0.2817
-0.2817
-0.7557
-0.9898
-0.9096
-0.5406

FFT interpolation of sn by factor fo 2 : size: 22 by 1
0.0000
0.2817
0.5406
0.7557
0.9096
0.9898
0.9898
0.9096
0.7557
0.5406
0.2817
-0.0000
-0.2817
-0.5406
-0.7557
-0.9096
-0.9898
-0.9898
-0.9096
-0.7557
-0.5406
-0.2817

complex signal cn : size: 11 by 1
(0.0000,1.0000)
(0.5406,0.8413)
(0.9096,0.4154)
(0.9898,-0.1423)
```

```
(0.7557,-0.6549)
(0.2817,-0.9595)
(-0.2817,-0.9595)
(-0.7557,-0.6549)
(-0.9898,-0.1423)
(-0.9096,0.4154)
(-0.5406,0.8413)

FFT interpolation of sn by factor fo 2 : size: 22 by 1
(0.0000,1.0000)
(0.2817,0.9595)
(0.5406,0.8413)
(0.7557,0.6549)
(0.9096,0.4154)
(0.9898,0.1423)
(0.9898,-0.1423)
(0.9096,-0.4154)
(0.7557,-0.6549)
(0.5406,-0.8413)
(0.2817,-0.9595)
(0.0000,-1.0000)
(-0.2817,-0.9595)
(-0.5406,-0.8413)
(-0.7557,-0.6549)
(-0.9096,-0.4154)
(-0.9898,-0.1423)
(-0.9898,0.1423)
(-0.9096,0.4154)
(-0.7558,0.6549)
(-0.5406,0.8413)
(-0.2817,0.9595)

Extending vector v1 in left direction by zeros padding : size: 7 by 1
0
0
1
2
3
4
5

Extending vector v1 in left direction by periodic mode : size: 7 by 1
4
5
```

```
1
2
3
4
5

Extending vector v1 in left direction by symmetric mode : size: 7 by 1
2
1
1
2
3
4
5

Extending vector v1 in right direction by zeros padding : size: 7 by 1
1
2
3
4
5
0
0

Extending vector v1 in right direction by periodic mode : size: 7 by 1
1
2
3
4
5
1
2

Extending vector v1 in right direction by symmetric mode : size: 7 by 1
1
2
3
4
5
5
4

Extending vector v1 in both direction by zeros padding : size: 9 by 1
0
```

```
0
1
2
3
4
5
0
0

Extending vector v1 in both direction by periodic mode : size: 9 by 1
4
5
1
2
3
4
5
1
2

Extending vector v1 in both direction by symmetric mode : size: 9 by 1
2
1
1
2
3
4
5
5
4

Keeping the center part of vector v1 : size: 3 by 1
2
3
4

Keeping the left part of vector v1 : size: 3 by 1
1
2
3

Keeping the right part of vector v1 : size: 3 by 1
3
4
```

```
5

Keeping the first(2) to first + L(3) elements of vector v1 : size: 3 by 1
3
4
5

The modulus of 2 divided by 5 is 2.
The modulus of -1 divided by 5 is 4.

The nearest integer >= 10/2 is 5.
The nearest integer >= 10/3 is 4.

The numbers can be represented by the integer power of 2 from 0 to 1000 are :

Process returned 0 (0x0)   execution time : 0.218 s
Press any key to continue.
```

1.4 简单计时器

为了方便测试程序的运行时间，SP++中提供了一个简单的计时器类，该类提供的接口如表 1-7所示。

表 1-7 计时器

Operation	Effect
Timing time	创建一个计时器
time.start()	开始计时
time.stop()	停止计时
time.read()	读取时间

测试代码：

```
/*****************************************************************************
 *                              time_test.cpp
 *
 * Timing function testing.
 *
 * Zhang Ming, 2010-01, Xi'an Jiaotong University.
 *****************************************************************************/

#include <iostream>
```

```
#include <cmath>
#include <timing.h>


using namespace std;
using namespace splab;


int main()
{
    double  var = 1;
    double  runTime = 0;
    Timing  time;

    time.start();
    for( int i=0; i<10000; ++i )
        for( int j=0; j<10000; ++j )
            var = sqrt(1.0*i*j);
    time.stop();
    runTime = time.read();

    cout << "The running time is : " << runTime << endl << endl;

    return 0;
}
```

运行结果：

```
The running time is : 7.254


Process returned 0 (0x0)   execution time : 7.301 s
Press any key to continue.
```

2 矩阵类模板

2.1 基本矩阵类

矩阵类模板Matrix<Type>是专门为线性代数中矩阵而设计的一个模板类，支持各种实数与复数矩阵的运算。包含矩阵的构造与析构，见表 2-1；矩阵类的基本属性提取，见表 2-2；矩阵常用计算的运算符重载，见表 2-3；以及其它常用的矩阵函数，见表 2-4。具体的函数声明与定义可以参见“matrix.h”和“ matrix -impl.h”。
注意：矩阵是以行优先存储的，第一维是行数，第二维是列数。

表 2-1 矩阵类的构造与析构函数

Operation	Effect
Matrix<Type> m	创建一个空矩阵
Matrix<Type> m2(m1)	创建矩阵 m2 的拷贝 m1
Matrix<Type> m(r,c,x)	创建常数矩阵
Matrix<Type> m(r,c,a)	通过数组创建矩阵
m.~Matrix<Type> ()	销毁矩阵并释放空间

表 2-2 矩阵类的属性获取

Operation	Effect
m.Type*()	矩阵到数组指针的类型转换
m.size()	矩阵元素总个数
m.dim(d)	矩阵第 d 维的维数
m.rows()	矩阵的行数
m.cols()	矩阵的列数
m.resize(r,c)	重新分配矩阵大小
m.getRow(r)	提取矩阵的第 r 行
m.getColumn(c)	提取矩阵的第 c 列
m.setRow(v)	设置矩阵的第 r 行
m.setColumn(v)	设置矩阵的第 c 列

表 2-3 矩阵类中重载的运算符

Operation	Effect
A1 = A2	矩阵对矩阵赋值
A = x	常数对矩阵赋值
A[i][j]	0 偏移下标访问
A(i,j)	1 偏移下标访问
-A	全部元素取反

A += x	矩阵自身加常数
A -= x	矩阵自身减常数
A *= x	矩阵自身乘常数
A /= x	矩阵自身除以常数
A1 += A2	矩阵自身加矩阵
A1 -= A2	矩阵自身减矩阵
A1 *= A2	矩阵自身乘矩阵(逐元素)
A1 /= A2	矩阵自身除以矩阵(逐元素)
x + A	常数与矩阵之和
A + x	矩阵与常数之和
A1 + A2	矩阵与矩阵之和
A - x	矩阵与常数之差
x - A	常数与矩阵之差
A1 - A2	矩阵与矩阵之差
A * x	矩阵与常数之积
x * A	常数与矩阵之积
A*v	矩阵乘以向量
A1 * A2	矩阵与矩阵之积
A / x	矩阵与常数之商
x / A	常数与矩阵之商
A1 / A2	矩阵与矩阵之商(逐元素)
>> A	输入矩阵
<< A	输出矩阵

表 2-4 矩阵类的其它函数

Operation	Effect
optMult (A,B,C)	优化版本的矩阵乘法
optMult (A,b,c)	优化版本的矩阵乘以向量
elemMult(A,B)	矩阵与矩阵逐元素相乘
elemMultEq(A,B)	矩阵与矩阵逐元素赋值相乘
elemDiv(A,B)	矩阵与矩阵逐元素相除
elemDivEq(A,B)	矩阵与矩阵逐元素赋值相除
trMult(A,B)	矩阵的转置乘以矩阵
trMult(A,b)	矩阵的转置乘以向量
multTr(A,B)	矩阵乘以矩阵的转置
multTr(A,b)	矩阵乘以向量的转置
multTr(b,c)	向量乘以向量的转置
trT(A)	矩阵的转置
trH(A)	矩阵的共轭转置
eye(n,x)	产生 n 阶单位矩阵
diag(A)	提取矩阵的对角线
diag(v)	通过向量生成对角矩阵
norm(A)	矩阵的 Frobenius 范数
swap(A,B)	交换两矩阵的元素

sum(A)	矩阵按列求和向量
min(A)	矩阵按列求最小值向量
max(A)	矩阵按列求最大值向量
mean(A)	矩阵按列求均值向量
abs(cA)	求复数矩阵的模值
arg(cA)	求复数矩阵的相角
real(cA)	求复数矩阵的实部
imag(cA)	求复数矩阵的虚部
complexMatrix(rM)	将实矩阵专为复数矩阵
complexMatrix(rM,iM)	通过实矩阵生成复矩阵

测试代码：

```
1.  /*****************************************************************************
2.   *                               matrix_test.cpp
3.   *
4.   * Matrix class testing.
5.   *
6.   * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <matrix.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. typedef double  Type;
22. const   int     M = 3;
23. const   int     N = 3;
24.
25.
26. int main()
27. {
28.     Type x;
29.
30.     Matrix<Type> m1;
31.     m1.resize( M, N );
```

```
32.     x = 1.0;
33.     m1 = x;
34.     cout << "matrix m1 : " << m1 << endl;
35.
36.     x = 2.0;
37.     Matrix<Type> m2( M, N, x );
38.     cout << "matrix m2 : " << m2 << endl;
39.
40.     Matrix<Type> m3 = m1;
41.     cout << "matrix m3 : " << m3 << endl;
42.     m3.resize( 3, 4 );
43.     for( int i=1; i<=3; ++i )
44.         for( int j=1; j<=4; ++j )
45.             m3(i,j) = Type(i*j);
46.
47. //    int row = m3.dim(1);
48. //  int column = m3.dim(2);
49. //  cout << "the row number of new matrix m3 : " << row << endl;
50. //  cout << "the column number of new matrix m3 : " << column << endl;
51. //  cout << "new matrix m3 : " << m3 << endl;
52. //
53. //  cout << "new matrix m3 : " << m3 << endl;
54. //  cout << "the diagonal matrix of m3 : " << diag( m3 ) << endl;
55. //  cout << "the transpose matrix of m3 : " << trT( m3 ) << endl;
56.
57.     cout << "\t\t\t\tmatrix-scalar operand" << endl << endl;
58.     cout << "scalar x = " << x << endl;
59.     cout << "m1 + x : " << m1+x << endl;
60.     cout << "x + m1 : " << x+m1 << endl;
61.     m1 += x;
62.     cout << "m1 += x : " << m1 << endl;
63.     cout << "m1 - x : " << m1-x << endl;
64.     cout << "x - m1 : " << x-m1 << endl;
65.     m1 -= x;
66.     cout << "m1 -= x : " << m1 << endl;
67.     cout << "m1 * x : " << m1*x << endl;
68.     cout << "x * m1 : " << x*m1 << endl;
69.     m1 *= x;
70.     cout << "m1 *= x : " << m1 << endl;
71.     cout << "m1 / x : " << m1/x << endl;
72.     cout << "x / m1 : " << x/m1 << endl;
73.     m1 /= x;
74.     cout << "m1 /= x : " << m1 << endl;
75.
```

```
76.     cout << "\t\t\telementwise matrix-matrix operand" << endl << endl;
77.     cout << "m1 + m2 : " << m1 + m2 << endl;
78.     m1 += m2;
79.     cout << "m1 += m2 : " << m1 << endl;
80.     cout << "m1 - m2 : " << m1-m2 << endl;
81.     m1 -= m2;
82.     cout << "m1 -= m2 : " << m1 << endl;
83.     m1 *= m2;
84.     cout << "m1 *= m2 : " << m1 << endl;
85.     m1 /= m2;
86.     cout << "m1 /= m2 : " << m1 << endl;
87.
88.     cout << "column minimum vector of m1 : " << min(m1) << endl;
89.     cout << "column maximum vector of m1 : " << max(m1) << endl;
90.     cout << "column sum vector of m1 : " << sum(m1) << endl;
91.     cout << "column mean vector of m1 : " << mean(m1) << endl;
92.
93.     cout << "\t\t\t\tmatrix-vector operand" << endl << endl;
94.     Vector<Type> v1( 3, 2 );
95.     cout << "vector v1 : " << v1 << endl;
96.     cout << "m1 * v1 : " << m1*v1 << endl;
97.     cout << "m1^T * v1 : " << trMult(m1, v1) << endl;
98.
99.     cout << "\t\t\t\tmatrix-matrix operand" << endl << endl;
100.    cout << "m1 * m2 : " << m1*m2 << endl;
101.    cout << "m3^T * m2 : " << trMult(m3, m2) << endl;
102.    cout << "m1 * m3^T^T : " << multTr(m1, trT(m3)) << endl;
103.
104.    Matrix<int> m4( 4, 5 );
105.    Vector<int> v2(5);
106.    v2[0] = 1;  v2[1] = 2;  v2[2] = 3;  v2[3] = 4;  v2[4] = 5;
107.    for( int i=0; i<4; ++i )
108.        m4.setRow( i*v2, i );
109.
110.    cout << "matrix m4 : " << m4 << endl;
111.    cout << "column vectors of m4 : " << endl;
112.    for( int j=0; j<5; ++j )
113.        cout << "the " << j << "th column" << m4.getColumn(j) << endl;
114.
115.    v2.resize(4);
116.    v2[0] = 1;  v2[1] = 2;  v2[2] = 3;  v2[3] = 4;
117.    for( int j=0; j<5; ++j )
118.        m4.setColumn( j*v2, j );
119.
```

```
120.    cout << "row vectors of m4 : " << endl;
121.    for( int i=0; i<4; ++i )
122.        cout << "the " << i << "th row" << m4.getRow(i) << endl;
123.
124.    cout << "\t\t\t\tcomplex matrix operand" << endl << endl;
125.    complex<Type> c = polar(1.0,PI/4);
126.    Matrix< complex<Type> > A( M, N ), B(M,N), C( M, N+1 );
127.    for( int i=0; i<M; i++ )
128.        for( int j=0; j<N; j++ )
129.        {
130.            A[i][j] = complex<Type>( Type(0.3*i+0.7*j), sin(Type(i+j)) );
131.            B[i][j] = complex<Type>( cos(Type(i+j)), Type(0.8*i+0.2*j) );
132.        }
133.    for( int i=0; i<N; ++i )
134.        C.setColumn( A.getColumn(i), i );
135.    C.setColumn( B.getColumn(0), N );
136.
137.    cout << setiosflags(ios::fixed) << setprecision(2);
138.    cout << "Matrix A:  " << A << endl;
139.    cout << "Matrix B:  " << B << endl;
140.    cout << "Matrix C:  " << C << endl;
141.    cout << "Absolute of C : " << abs(C) << endl;
142.    cout << "Angle of C : " << arg(C) << endl;
143.    cout << "Real part of C : " << real(C) << endl;
144.    cout << "Imaginary part of C : " << imag(C) << endl;
145.
146.    cout << "c*A - A*B + trH(B)  " << c*A - A*B + trH(B) << endl;
147.    cout << "A*C * C^H*B  " << A*C * trMult(C,B) << endl;
148.    cout << "(A.*B+C*C^H)) ./ A^T  "
149.         << elemDivd( elemMult(A,B)+multTr(C,C), trT(A) ) << endl;
150.    cout << "diag( A * diag(C) )  " << diag( A*diag(C)/c ) * c << endl;
151.    cout << "A .* ( B./(c*m1) )  "
152.         << elemMultEq( A, elemDivdEq(B,c*complexMatrix(m1)) ) << endl;
153.
154.    return 0;
155.}
```

运行结果：

```
1.  matrix m1 : size: 3 by 3
2.  1       1       1
3.  1       1       1
4.  1       1       1
5.
6.  matrix m2 : size: 3 by 3
```

```
2       2       2
2       2       2
2       2       2

matrix m3 : size: 3 by 3
1       1       1
1       1       1
1       1       1

                                matrix-scalar operand

scalar x = 2
m1 + x : size: 3 by 3
3       3       3
3       3       3
3       3       3

x + m1 : size: 3 by 3
3       3       3
3       3       3
3       3       3

m1 += x : size: 3 by 3
3       3       3
3       3       3
3       3       3

m1 - x : size: 3 by 3
1       1       1
1       1       1
1       1       1

x - m1 : size: 3 by 3
-1      -1      -1
-1      -1      -1
-1      -1      -1

m1 -= x : size: 3 by 3
1       1       1
1       1       1
1       1       1

m1 * x : size: 3 by 3
2       2       2
```

```
2       2       2
2       2       2

x * m1 : size: 3 by 3
2       2       2
2       2       2
2       2       2

m1 *= x : size: 3 by 3
2       2       2
2       2       2
2       2       2

m1 / x : size: 3 by 3
1       1       1
1       1       1
1       1       1

x / m1 : size: 3 by 3
1       1       1
1       1       1
1       1       1

m1 /= x : size: 3 by 3
1       1       1
1       1       1
1       1       1

                        elementwise matrix-matrix operand

m1 + m2 : size: 3 by 3
3       3       3
3       3       3
3       3       3

m1 += m2 : size: 3 by 3
3       3       3
3       3       3
3       3       3

m1 - m2 : size: 3 by 3
1       1       1
1       1       1
1       1       1
```

```
95.
96. m1 -= m2 : size: 3 by 3
97. 1       1       1
98. 1       1       1
99. 1       1       1
100.
101. m1 *= m2 : size: 3 by 3
102. 2       2       2
103. 2       2       2
104. 2       2       2
105.
106. m1 /= m2 : size: 3 by 3
107. 1       1       1
108. 1       1       1
109. 1       1       1
110.
111. column minimum vector of m1 : size: 3 by 1
112. 1
113. 1
114. 1
115.
116. column maximum vector of m1 : size: 3 by 1
117. 1
118. 1
119. 1
120.
121. column sum vector of m1 : size: 3 by 1
122. 3
123. 3
124. 3
125.
126. column mean vector of m1 : size: 3 by 1
127. 1
128. 1
129. 1
130.
131.                                 matrix-vector operand
132.
133. vector v1 : size: 3 by 1
134. 2
135. 2
136. 2
137.
138. m1 * v1 : size: 3 by 1
```

```
6
6
6

m1^T * v1 : size: 3 by 1
6
6
6

                                matrix-matrix operand

m1 * m2 : size: 3 by 3
6       6       6
6       6       6
6       6       6

m3^T * m2 : size: 4 by 3
12      12      12
24      24      24
36      36      36
48      48      48

m1 * m3^T^T : size: 3 by 4
6       12      18      24
6       12      18      24
6       12      18      24

matrix m4 : size: 4 by 5
0       0       0       0       0
1       2       3       4       5
2       4       6       8       10
3       6       9       12      15

column vectors of m4 :
the 0th columnsize: 4 by 1
0
1
2
3

the 1th columnsize: 4 by 1
0
2
4
```

```
6

the 2th columnsize: 4 by 1
0
3
6
9

the 3th columnsize: 4 by 1
0
4
8
12

the 4th columnsize: 4 by 1
0
5
10
15

row vectors of m4 :
the 0th rowsize: 5 by 1
0
1
2
3
4

the 1th rowsize: 5 by 1
0
2
4
6
8

the 2th rowsize: 5 by 1
0
3
6
9
12

the 3th rowsize: 5 by 1
0
```

```
4
8
12
16

                                complex matrix operand

Matrix A:  size: 3 by 3
(0.00,0.00)     (0.70,0.84)     (1.40,0.91)
(0.30,0.84)     (1.00,0.91)     (1.70,0.14)
(0.60,0.91)     (1.30,0.14)     (2.00,-0.76)

Matrix B:  size: 3 by 3
(1.00,0.00)     (0.54,0.20)     (-0.42,0.40)
(0.54,0.80)     (-0.42,1.00)    (-0.99,1.20)
(-0.42,1.60)    (-0.99,1.80)    (-0.65,2.00)

Matrix C:  size: 3 by 4
(0.00,0.00)     (0.70,0.84)     (1.40,0.91)     (1.00,0.00)
(0.30,0.84)     (1.00,0.91)     (1.70,0.14)     (0.54,0.80)
(0.60,0.91)     (1.30,0.14)     (2.00,-0.76)    (-0.42,1.60)

Absolute of C : size: 3 by 4
0.00    1.09    1.67    1.00
0.89    1.35    1.71    0.97
1.09    1.31    2.14    1.65

Angle of C : size: 3 by 4
0.00    0.88    0.58    0.00
1.23    0.74    0.08    0.98
0.99    0.11    -0.36   1.83

Real part of C : size: 3 by 4
0.00    0.70    1.40    1.00
0.30    1.00    1.70    0.54
0.60    1.30    2.00    -0.42

Imaginary part of C : size: 3 by 4
0.00    0.84    0.91    0.00
0.84    0.91    0.14    0.80
0.91    0.14    -0.76   1.60

c*A - A*B + trH(B)  size: 3 by 3
(3.33,-2.88)    (4.60,-1.68)    (4.37,-2.18)
```

```
(0.98,-4.19)    (2.92,-3.71)    (4.05,-3.88)
(-2.20,-4.87)   (0.99,-6.38)    (3.16,-6.90)

A*C * C^H*B  size: 3 by 3
(-22.45,41.39)  (-57.34,30.67)  (-75.11,36.86)
(-13.03,52.79)  (-53.59,46.63)  (-73.18,52.60)
(16.66,60.01)   (-25.93,77.12)  (-40.26,93.59)

(A.*B+C*C^H)) ./ A^T  size: 3 by 3
(inf,nan)       (3.20,-4.47)    (2.78,-0.12)
(2.30,-2.85)    (3.12,-2.22)    (3.99,2.29)
(-0.04,-1.18)   (3.05,0.07)     (3.81,3.69)

diag( A * diag(C) )  size: 3 by 3
(3.42,2.24)     (0.00,0.00)     (0.00,0.00)
(0.00,0.00)     (3.68,0.81)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (4.60,-1.70)

A .* ( B./(c*m1) )  size: 3 by 3
(0.00,0.00)     (0.57,0.27)     (-0.54,0.80)
(0.13,0.85)     (-0.50,1.38)    (0.03,2.65)
(-0.79,1.62)    (0.47,2.65)     (3.32,3.03)


Process returned 0 (0x0)   execution time : 0.218 s
Press any key to continue.
```

2.2 常用数学函数的矩阵版本

为了便于数值计算，将一些信号处理中经常使用的数学函数进行了重载，使之能够运用于矩阵对象，如表 2-5所示。

表 2-5 常用函数的矩阵版本（逐元素）

Operation	Effect
cos(m)	矩阵的余弦
sin (m)	矩阵的正弦
tan (m)	矩阵的正切
acos (m)	矩阵的反余弦
asin (m)	矩阵的反正弦
atan (m)	矩阵的反正切
exp (m)	矩阵的幂函数
log (m)	矩阵的自然对数函数

log10 (m)	矩阵以 10 为底的对数函数
sqrt (m)	矩阵的平方根
pow (m,m)	矩阵的向量幂函数
pow (m,p)	矩阵的常数幂函数
pow (b,m)	常数的矩阵幂函数

测试代码：

```
/*****************************************************************************
 *                              matrixmath_test.cpp
 *
 * Math functions of matrix testing.
 *
 * Zhang Ming, 2010-08, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <matrixmath.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
    int N = 9;
    Type  a = 0, b = 2*PI;
    Vector<Type> array = linspace( a, b, N );

    Matrix<Type> x( 3, 3, array );

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << x << endl;
    cout << "sin of x : " << sin(x) << endl;
    cout << "cos of x : "<< cos(x) << endl;
    cout << "tan of x : " << tan(x) << endl;
```

```
    cout << "asin of x : "<< asin(x) << endl;
    cout << "acos of x : " << acos(x) << endl;
    cout << "atan of x : " << atan(x) << endl;

    cout << "exp of x : "<< exp(x) << endl;
    cout << "log of x : " << log(x) << endl;
    cout << "log10 of x : " << log10(x) << endl;

    a = 2.0;
    cout << "sqrt of x : "<< sqrt(x) << endl;
    cout << "pow of x : " << pow(x,x) << endl;
    cout << "pow of x : " << pow(x,a) << endl;
    cout << "pow of x : " << pow(a,x) << endl;

    return 0;
}
```

运行结果：

```
  size: 3 by 3
0.0000  0.7854  1.5708
2.3562  3.1416  3.9270
4.7124  5.4978  6.2832

sin of x : size: 3 by 3
0.0000  0.7071  1.0000
0.7071  0.0000  -0.7071
-1.0000 -0.7071 -0.0000

cos of x : size: 3 by 3
1.0000  0.7071  0.0000
-0.7071 -1.0000 -0.7071
-0.0000 0.7071  1.0000

tan of x : size: 3 by 3
0.0000  1.0000  16331778728383844.0000
-1.0000 -0.0000 1.0000
5443926242794615.0000   -1.0000 -0.0000

asin of x : size: 3 by 3
0.0000  0.9033  nan
nan     nan     nan
nan     nan     nan

acos of x : size: 3 by 3
```

```
1.5708  0.6675  nan
nan     nan     nan
nan     nan     nan

atan of x : size: 3 by 3
0.0000  0.6658  1.0039
1.1694  1.2626  1.3214
1.3617  1.3909  1.4130

exp of x : size: 3 by 3
1.0000  2.1933  4.8105
10.5507 23.1407 50.7540
111.3178        244.1511        535.4917

log of x : size: 3 by 3
-inf    -0.2416 0.4516
0.8570  1.1447  1.3679
1.5502  1.7043  1.8379

log10 of x : size: 3 by 3
-inf    -0.1049 0.1961
0.3722  0.4971  0.5941
0.6732  0.7402  0.7982

sqrt of x : size: 3 by 3
0.0000  0.8862  1.2533
1.5350  1.7725  1.9817
2.1708  2.3447  2.5066

pow of x : size: 3 by 3
1.0000  0.8272  2.0327
7.5336  36.4622 215.2126
1487.9012       11732.6371      103540.9204

pow of x : size: 3 by 3
0.0000  0.6169  2.4674
5.5517  9.8696  15.4213
22.2066 30.2257 39.4784

pow of x : size: 3 by 3
1.0000  1.7236  2.9707
5.1202  8.8250  15.2104
26.2162 45.1855 77.8802

```

```
71.
72. Process returned 0 (0x0)   execution time : 0.078 s
73. Press any key to continue.
```

2.3 实矩阵与复矩阵的 Cholesky 分解

对于一个对称正定矩阵A，可以表示为A=L*L'，其中L为下三角矩阵，这就是对称正定矩阵的Cholesky分解。利用该分解可以解线性方程组，也可以求解矩阵方程，Cholesky类的具体功能见表 2-6。

表 2-6 实矩阵 Cholesky 分解

Operation	Effect
Cholesky<Real> cho	建立 Cholesky 类
cho.~Cholesky<Real>()	Cholesky 析构函数
cho.isSpd()	判断矩阵是否对称正定
cho.dec(A)	对实矩阵 A 进行 Cholesky 分解
cho.getL()	获取下三角矩阵 L
cho.solve(b)	解方程组 Ax = b
cho.solve(B)	解矩阵方程 AX = B

测试代码：

```
1.  /*****************************************************************************
2.   *                                cholesky_test.cpp
3.   *
4.   * Cholesky class testing.
5.   *
6.   * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <cholesky.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
```

```
typedef double  Type;
const   int     N = 5;


int main()
{
    Matrix<Type> A(N,N), L(N,N);
    Vector<Type> b(N);

    for( int i=1; i<N+1; ++i )
    {
        for( int j=1; j<N+1; ++j )
            if( i == j )
                A(i,i) = i;
            else
                if( i < j )
                    A(i,j) = i;
                else
                    A(i,j) = j;

        b(i) = i*(i+1)/2.0 + i*(N-i);
    }

    cout << setiosflags(ios::fixed) << setprecision(3);
    cout << "The original matrix A : " << A << endl;
    Cholesky<Type> cho;
    cho.dec(A);

    if( !cho.isSpd() )
        cout << "Factorization was not complete." << endl;
    else
    {
        L = cho.getL();
        cout << "The lower triangular matrix L is : " << L << endl;
        cout << "A - L*L^T is : " << A - L*trT(L) << endl;

        Vector<Type> x = cho.solve(b);
        cout << "The constant vector b : " << b << endl;
        cout << "The solution of Ax = b : " << x << endl;
        cout << "The Ax - b : " << A*x-b << endl;

        Matrix<Type> IA = cho.solve(eye(N,Type(1)));
        cout << "The inverse matrix of A : " << IA << endl;
        cout << "The product of  A*inv(A) : " << A*IA << endl;
```

```
    }

    return 0;
}
```

运行结果：

```
The original matrix A : size: 5 by 5
1.000   1.000   1.000   1.000   1.000
1.000   2.000   2.000   2.000   2.000
1.000   2.000   3.000   3.000   3.000
1.000   2.000   3.000   4.000   4.000
1.000   2.000   3.000   4.000   5.000

The lower triangular matrix L is : size: 5 by 5
1.000   0.000   0.000   0.000   0.000
1.000   1.000   0.000   0.000   0.000
1.000   1.000   1.000   0.000   0.000
1.000   1.000   1.000   1.000   0.000
1.000   1.000   1.000   1.000   1.000

A - L*L^T is : size: 5 by 5
0.000   0.000   0.000   0.000   0.000
0.000   0.000   0.000   0.000   0.000
0.000   0.000   0.000   0.000   0.000
0.000   0.000   0.000   0.000   0.000
0.000   0.000   0.000   0.000   0.000

The constant vector b : size: 5 by 1
5.000
9.000
12.000
14.000
15.000

The solution of Ax = b : size: 5 by 1
1.000
1.000
1.000
1.000
1.000

The Ax - b : size: 5 by 1
0.000
0.000
```

```
0.000
0.000
0.000

The inverse matrix of A : size: 5 by 5
2.000   -1.000  0.000   0.000   0.000
-1.000  2.000   -1.000  0.000   0.000
0.000   -1.000  2.000   -1.000  0.000
0.000   0.000   -1.000  2.000   -1.000
0.000   0.000   0.000   -1.000  1.000

The product of  A*inv(A) : size: 5 by 5
1.000   0.000   0.000   0.000   0.000
0.000   1.000   0.000   0.000   0.000
0.000   0.000   1.000   0.000   0.000
0.000   0.000   0.000   1.000   0.000
0.000   0.000   0.000   0.000   1.000


Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

复数矩阵的Cholesky分解如表 2-7所示。

表 2-7　复矩阵 Cholesky 分解

Operation	Effect
CCholesky<Real> cho	建立 CCholesky 类
cho.~CCholesky<Real>()	CCholesky 析构函数
cho.isSpd()	判断矩阵是否对称正定
cho.dec(cA)	对复矩阵 cA 进行 Cholesky 分解
cho.getL()	获取下三角矩阵 L
cho.solve(b)	解方程组 Ax = b
cho.solve(B)	解矩阵方程 AX = B

测试代码：

```
/*****************************************************************************
 *                            ccholesky_test.cpp
 *
 * Comnplex Cholesky class testing.
 *
 * Zhang Ming, 2010-12, Xi'an Jiaotong University.
 *****************************************************************************/

```

```
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <ccholesky.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. typedef double  Type;
22. const   int     N = 5;
23.
24.
25. int main()
26. {
27.     cout << setiosflags(ios::fixed) << setprecision(3);
28.     Matrix<complex<Type> > A(N,N), L(N,N);
29.     Vector<complex<Type> > b(N);
30.
31.     for( int i=1; i<N+1; ++i )
32.     {
33.         for( int j=1; j<N+1; ++j )
34.             if( i == j )
35.                 A(i,i) = complex<Type>(N+i,0);
36.             else
37.                 if( i < j )
38.                     A(i,j) = complex<Type>(Type(i),cos(Type(i)));
39.                 else
40.                     A(i,j) = complex<Type>(Type(j),-cos(Type(j)));
41.
42.         b(i) = i*(i+1)/2 + i*(N-i);
43.     }
44.
45.     cout << "The original matrix A : " << A << endl;
46.     CCholesky<complex<Type> > cho;
47.     cho.dec(A);
48.     if( !cho.isSpd() )
49.         cout << "Factorization was not complete." << endl;
50.     else
51.     {
52.         L = cho.getL();
```

```
        cout << "The lower triangular matrix L is : " << L << endl;
        cout << "A - L*L^H is : " << A - L*trH(L) << endl;

        Vector<complex<Type> > x = cho.solve(b);
        cout << "The constant vector b : " << b << endl;
        cout << "The solution of Ax = b : " << x << endl;
        cout << "The solution of Ax - b : " << A*x-b << endl;

        Matrix<complex<Type> > IA = cho.solve(eye(N,complex<Type>(1,0)));
        cout << "The inverse matrix of A : " << IA << endl;
        cout << "The product of  A*inv(A) : " << A*IA << endl;
    }

    return 0;
}
```

运行结果：

```
The original matrix A : size: 5 by 5
(6.000,0.000)   (1.000,0.540)   (1.000,0.540)   (1.000,0.540)   (1.000,0.540)

(1.000,-0.540)  (7.000,0.000)   (2.000,-0.416)  (2.000,-0.416)  (2.000,-0.416)

(1.000,-0.540)  (2.000,0.416)   (8.000,0.000)   (3.000,-0.990)  (3.000,-0.990)

(1.000,-0.540)  (2.000,0.416)   (3.000,0.990)   (9.000,0.000)   (4.000,-0.654)

(1.000,-0.540)  (2.000,0.416)   (3.000,0.990)   (4.000,0.654)   (10.000,0.000)


The lower triangular matrix L is : size: 5 by 5
(2.449,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

(0.408,-0.221)  (2.605,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

(0.408,-0.221)  (0.685,0.160)   (2.700,0.000)   (0.000,0.000)   (0.000,0.000)

(0.408,-0.221)  (0.685,0.160)   (0.848,0.367)   (2.727,0.000)   (0.000,0.000)

(0.408,-0.221)  (0.685,0.160)   (0.848,0.367)   (0.893,0.240)   (2.753,0.000)


A - L*L^H is : size: 5 by 5
(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

```

```
(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (-0.000,0.000)  (0.000,0.000)

(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)


The constant vector b : size: 5 by 1
(5.000,0.000)
(9.000,0.000)
(12.000,0.000)
(14.000,0.000)
(15.000,0.000)

The solution of Ax = b : size: 5 by 1
(0.356,-0.310)
(0.562,0.207)
(0.746,0.284)
(0.843,-0.037)
(0.842,-0.214)

The solution of Ax - b : size: 5 by 1
(0.000,-0.000)
(-0.000,0.000)
(-0.000,0.000)
(-0.000,0.000)
(0.000,0.000)

The inverse matrix of A : size: 5 by 5
(0.177,0.000)   (-0.018,-0.007) (-0.014,-0.004) (-0.008,-0.006) (-0.005,-0.007)

(-0.018,0.007)  (0.164,-0.000)  (-0.024,0.013)  (-0.020,0.003)  (-0.017,-0.002)

(-0.014,0.004)  (-0.024,-0.013) (0.160,-0.000)  (-0.032,0.018)  (-0.029,0.008)

(-0.008,0.006)  (-0.020,-0.003) (-0.032,-0.018) (0.150,0.000)   (-0.043,0.012)

(-0.005,0.007)  (-0.017,0.002)  (-0.029,-0.008) (-0.043,-0.012) (0.132,0.000)


The product of  A*inv(A) : size: 5 by 5
(1.000,0.000)   (-0.000,-0.000) (-0.000,-0.000) (-0.000,-0.000) (0.000,-0.000)
```

```
72.
73. (0.000,-0.000)  (1.000,0.000)   (-0.000,0.000)  (0.000,-0.000)  (0.000,-0.000)
74.
75. (0.000,-0.000)  (-0.000,-0.000) (1.000,0.000)   (-0.000,0.000)  (-0.000,-0.000)
76.
77. (0.000,-0.000)  (0.000,0.000)   (-0.000,-0.000) (1.000,0.000)   (0.000,0.000)
78.
79. (0.000,-0.000)  (0.000,0.000)   (-0.000,0.000)  (0.000,0.000)   (1.000,-0.000)
80.
81.
82.
83. Process returned 0 (0x0)   execution time : 0.125 s
84. Press any key to continue.
```

2.4 实矩阵与复矩阵的 LU 分解

对于一个n阶矩阵A，LU分解将其分为一个下三角阵L与一个上三角阵U的乘积，可以借助矩阵的LU分解来求解矩阵的行列式，线性方程组以及矩阵方程等，详见表2-8，该表所提供的函数对实矩阵与复矩阵均可适用。对于行数与列数不等的矩阵，LU同样可以分解成功。

表 2-8 LU 分解

Operation	Effect
LUD<Real> lu	建立 LUD 类
lu.~LUD<Real>()	LUD 析构函数
lu.dec(cA)	对复矩阵 cA 进行三角分解
lu.getL()	获取下三角矩阵 L
lu.getU()	获取上三角矩阵 U
lu.det()	计算矩阵的行列式
lu. isNonsingular()	判断矩阵是否非奇异
lu.solve(b)	解方程组 Ax = b
lu.solve(B)	矩阵方程 AX = B

测试代码：

```
1.  /*****************************************************************************
2.   *                              lud_test.cpp
3.   *
4.   * LUD class testing.
5.   *
6.   * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
7.   *****************************************************************************/
```

```
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <lud.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. typedef double  Type;
22. const   int     M = 3;
23. const   int     N = 4;
24.
25.
26. int main()
27. {
28.     Matrix<Type> A(3,3), B(3,3), L, U;
29.     Vector<Type> b(3);
30.
31.     A[0][0] = 1;    A[0][1] = 2;    A[0][2] = 1;
32.     A[1][0] = 2;    A[1][1] = 5;    A[1][2] = 4;
33.     A[2][0] = 1;    A[2][1] = 1;    A[2][2] = 0;
34.
35.     B[0][0] = 1;    B[0][1] = 0;    B[0][2] = 0;
36.     B[1][0] = 0;    B[1][1] = 1;    B[1][2] = 0;
37.     B[2][0] = 0;    B[2][1] = 0;    B[2][2] = 1;
38.
39.     b[0] = 1;   b[1] = 0;   b[2] = 1;
40.
41.     LUD<Type> lu;
42.     lu.dec(A);
43.     L = lu.getL();
44.     U = lu.getU();
45.     cout << setiosflags(ios::fixed) << setprecision(4);
46.     cout << "The original matrix A is : " << A << endl;
47.     cout << "The unit lower triangular matrix L is : " << L << endl;
48.     cout << "The upper triangular matrix U is : " << U << endl;
49.
50.     Vector<Type> x = lu.solve(b);
51.     cout << "The constant vector b : " << b << endl;
```

```
52.     cout << "The solution of A * x = b : " << x << endl;
53.     cout << "The solution of A * x - b : " << A*x - b << endl;
54.
55.     cout << "The inverse matrix of A : " << lu.solve(B) << endl;
56.     cout << "The A * inverse(A) : " << A*lu.solve(B) << endl;
57.     cout << "The determinant of A : " << endl;
58.     cout << lu.det() << endl << endl << endl;
59.
60.     Matrix<complex<Type> > cA(M,N), cPA(M,N), invcA, cL, cU;
61.     for( int i=0; i<M; i++ )
62.         for( int j=0; j<N; j++ )
63.             cA[i][j] = complex<Type>( Type(0.3*i+0.7*j), sin(Type(i+j)) );
64.     LUD<complex<Type> > clu;
65.
66.     clu.dec(cA);
67.     cL = clu.getL();
68.     cU = clu.getU();
69.     Vector<int> p = clu.getPivot();
70.     for( int i=0; i<M; ++i )
71.         cPA.setRow( cA.getRow(p[i]), i );
72.
73.     cout << setiosflags(ios::fixed) << setprecision(3);
74.     cout << "The original complex matrix cA is : " << cA << endl;
75.     cout << "The unit lower triangular matrix cL is : " << cL << endl;
76.     cout << "The upper triangular matrix cU is : " << cU << endl;
77.     cout << "cP*cA - cL*cU is : " << cPA - cL*cU << endl;
78.
79.     if( M == N )
80.     {
81.         invcA = clu.solve( eye( M, complex<Type>(1.0,0) ) );
82.         cout << "The inverse matrix of cA : " << invcA << endl;
83.         cout << "The cA * inverse(cA) : " << cA*invcA << endl;
84.     }
85.
86.     return 0;
87. }
```

运行结果：

```
1.  The original matrix A is : size: 3 by 3
2.  1.0000  2.0000  1.0000
3.  2.0000  5.0000  4.0000
4.  1.0000  1.0000  0.0000
5.
6.  The unit lower triangular matrix L is : size: 3 by 3
```

```
  1.0000  0.0000  0.0000
  0.5000  1.0000  0.0000
  0.5000  0.3333  1.0000

The upper triangular matrix U is : size: 3 by 3
2.0000  5.0000  4.0000
0.0000  -1.5000 -2.0000
0.0000  0.0000  -0.3333

The constant vector b : size: 3 by 1
1.0000
0.0000
1.0000

The solution of A * x = b : size: 3 by 1
-1.0000
2.0000
-2.0000

The solution of A * x - b : size: 3 by 1
-0.0000
0.0000
0.0000

The inverse matrix of A : size: 3 by 3
-4.0000 1.0000  3.0000
4.0000  -1.0000 -2.0000
-3.0000 1.0000  1.0000

The A * inverse(A) : size: 3 by 3
1.0000  0.0000  0.0000
0.0000  1.0000  0.0000
0.0000  0.0000  1.0000

The determinant of A :
1.0000


The original complex matrix cA is : size: 3 by 4
(0.000,0.000)   (0.700,0.841)   (1.400,0.909)   (2.100,0.141)
(0.300,0.841)   (1.000,0.909)   (1.700,0.141)   (2.400,-0.757)
(0.600,0.909)   (1.300,0.141)   (2.000,-0.757)  (2.700,-0.959)

The unit lower triangular matrix cL is : size: 3 by 3
```

```
(1.000,0.000)   (0.000,0.000)   (0.000,0.000)
(0.000,0.000)   (1.000,0.000)   (0.000,0.000)
(0.796,0.196)   (0.377,0.322)   (1.000,0.000)

The upper triangular matrix cU is : size: 3 by 4
(0.600,0.909)   (1.300,0.141)   (2.000,-0.757)  (2.700,-0.959)
(0.000,0.000)   (0.700,0.841)   (1.400,0.909)   (2.100,0.141)
(0.000,0.000)   (0.000,0.000)   (-0.275,-0.441) (-0.683,-1.252)

cP*cA - cL*cU is : size: 3 by 4
(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)
(0.000,0.000)   (0.000,0.000)   (0.000,0.000)   (0.000,0.000)
(0.000,0.000)   (0.000,0.000)   (0.000,-0.000)  (0.000,0.000)


Process returned 0 (0x0)   execution time : 0.059 s
Press any key to continue.
```

2.5 实矩阵与复矩阵的 QR 分解

对于一个m行n列的矩阵A，QR分解将分主一个m阶的正交矩阵Q和一个m行n列的上三角矩阵R，满足A=Q*R，QR分解总是存在的，即使对于秩亏矩阵。利用矩阵的QR分解可以求解超定线性方程组的最小二乘解与最小范数解等，详细内容参见下一章。QR分解的C++类见表 2-9。

表 2-9 实矩阵 QR 分解

Operation	Effect
QRD<Real> qr	建立 QRD 类
qr.~QRD<Real>()	QRD 析构函数
qr.dec(A)	对实矩阵 A 进行正交三角分解
qr.getQ()	获取正交矩阵 Q
qr.getR()	获取上三角矩阵 R
qr. isFullRank()	判断矩阵是否满秩
qr.solve(b)	线性方程组最小二乘解 Ax = b
qr.solve(B)	矩阵方程最小二乘解 AX = B

测试代码：

```
/*****************************************************************************
 *                              qrd_test.cpp
 *
 * QRD class testing.
```

```
5.    *
6.    * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
7.    ************************************************************************/
8.
9.
10.  #define BOUNDS_CHECK
11.
12.  #include <iostream>
13.  #include <iomanip>
14.  #include <qrd.h>
15.
16.
17.  using namespace std;
18.  using namespace splab;
19.
20.
21.  typedef double  Type;
22.  const   int     M = 4;
23.  const   int     N = 3;
24.
25.
26.  int main()
27.  {
28.      Matrix<Type> A(M,N), Q, R;
29.      A[0][0] = 1;    A[0][1] = 0;    A[0][2] = 0;
30.      A[1][0] = 1;    A[1][1] = 2;    A[1][2] = 4;
31.      A[2][0] = 1;    A[2][1] = 3;    A[2][2] = 9;
32.      A[3][0] = 1;    A[3][1] = 3;    A[3][2] = 9;
33.
34.      Matrix<Type> B = trT(A);
35.      QRD<Type> qr;
36.      qr.dec(B);
37.      Q = qr.getQ();
38.      R = qr.getR();
39.
40.      cout << setiosflags(ios::fixed) << setprecision(4);
41.      cout << "The original matrix B : " << B << endl;
42.      cout << "The orthogonal matrix Q  : " << Q << endl;
43.      cout << "The upper triangular matrix R : " << R << endl;
44.      cout << "B - Q*R : " << B - Q*R << endl;
45.
46.      Vector<Type> b(M);
47.      b[0]= 1;    b[1] = 0;   b[2] = 1, b[3] = 2;
48.
```

```
49.     qr.dec(A);
50.     if( qr.isFullRank() )
51.     {
52.         Vector<Type> x = qr.solve(b);
53.         cout << "The constant vector b : " << b << endl;
54.         cout << "The least squares solution of A * x = b : " << x << endl;
55.
56.         Matrix<Type> X = qr.solve(eye(M,Type(1)));
57.         cout << "The least squares solution of A * X = I : " << X << endl;
58.         cout << "The A * X: " << A*X << endl;
59.     }
60.     else
61.         cout << " The matrix is rank deficient! " << endl;
62.
63.     return 0;
64. }
```

运行结果：

```
1.  The original matrix B : size: 3 by 4
2.  1.0000  1.0000  1.0000  1.0000
3.  0.0000  2.0000  3.0000  3.0000
4.  0.0000  4.0000  9.0000  9.0000
5.
6.  The orthogonal matrix Q  : size: 3 by 3
7.  -1.0000 0.0000  0.0000
8.  0.0000  -0.4472 0.8944
9.  0.0000  -0.8944 -0.4472
10.
11. The upper triangular matrix R : size: 3 by 4
12. -1.0000 -1.0000 -1.0000 -1.0000
13. 0.0000  -4.4721 -9.3915 -9.3915
14. 0.0000  0.0000  -1.3416 -1.3416
15.
16. B - Q*R : size: 3 by 4
17. 0.0000  0.0000  0.0000  0.0000
18. 0.0000  0.0000  0.0000  0.0000
19. 0.0000  0.0000  0.0000  0.0000
20.
21. The constant vector b : size: 4 by 1
22. 1.0000
23. 0.0000
24. 1.0000
25. 2.0000
26.
```

```
The least squares solution of A * x = b : size: 3 by 1
1.0000
-1.8333
0.6667

The least squares solution of A * X = I : size: 3 by 4
1.0000  -0.0000 0.0000  -0.0000
-0.8333 1.5000  -0.3333 -0.3333
0.1667  -0.5000 0.1667  0.1667

The A * X: size: 4 by 4
1.0000  0.0000  0.0000  0.0000
0.0000  1.0000  -0.0000 0.0000
-0.0000 0.0000  0.5000  0.5000
-0.0000 0.0000  0.5000  0.5000


Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

复矩阵的QR分解如表 2-10所示。

表 2-10　复矩阵 QR 分解

Operation	Effect
CQRD<Real> qr	建立 CQRD 类
qr.~CQRD<Real>()	CQRD 析构函数
qr.dec(cA)	对复矩阵 cA 进行正交三角分解
qr.getQ()	获取正交矩阵 Q
qr.getR()	获取上三角矩阵 R
qr. isFullRank()	判断矩阵是否满秩
qr.solve(b)	线性方程组最小二乘解 Ax = b
qr.solve(B)	矩阵方程最小二乘解 AX = B

测试代码：

```
/*****************************************************************************
*                               cqrd_test.cpp
*
* CQRD class testing.
*
* Zhang Ming, 2010-12, Xi'an Jiaotong University.
*****************************************************************************/


```

```
#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <cqrd.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     M = 4;
const   int     N = 3;


int main()
{
    Matrix<Type> A(M,N);
    A[0][0] = 1;    A[0][1] = 2;    A[0][2] = 3;
    A[1][0] = 1;    A[1][1] = 2;    A[1][2] = 4;
    A[2][0] = 1;    A[2][1] = 9;    A[2][2] = 7;
    A[3][0] = 5;    A[3][1] = 6;    A[3][2] = 8;
//  A = trT(A);

    Matrix<complex<Type> > cA = complexMatrix( A, elemMult(A,A) );
    CQRD<Type> qr;
    qr.dec(cA);
    Matrix<complex<Type> > Q = qr.getQ();
    Matrix<complex<Type> > R = qr.getR();

    cout << setiosflags(ios::fixed) << setprecision(3);
    cout << "The original matrix cA : " << cA << endl;
    cout << "The orthogonal matrix Q  : " << Q << endl;
    cout << "The upper triangular matrix R : " << R << endl;
    cout << "Q^H * Q : " << trMult(Q,Q) << endl;
    cout << "cA - Q*R : " << cA - Q*R << endl;

    Vector<Type> b(M);
    b[0]= 1;    b[1] = 0;   b[2] = 1, b[3] = 2;
    Vector<complex<Type> > cb = complexVector(b);

    if( qr.isFullRank() )
    {
```

```
54.         Vector<complex<Type> > x = qr.solve(cb);
55.         cout << "The constant vector cb : " << cb << endl;
56.         cout << "The least squares solution of cA * x = cb : " << x << endl;
57.
58.         Matrix<complex<Type> > X = qr.solve( eye( M, complex<Type>(1) ) );
59.         cout << "The least squares solution of cA * X = I : " << X << endl;
60.         cout << "The cA * X: " << cA*X << endl;
61.     }
62.     else
63.         cout << " The matrix is rank deficient! " << endl;
64.
65.     return 0;
66. }
```

运行结果：

```
1.  The original matrix cA : size: 4 by 3
2.  (1.000,1.000)   (2.000,4.000)   (3.000,9.000)
3.  (1.000,1.000)   (2.000,4.000)   (4.000,16.000)
4.  (1.000,1.000)   (9.000,81.000)  (7.000,49.000)
5.  (5.000,25.000)  (6.000,36.000)  (8.000,64.000)
6.
7.  The orthogonal matrix Q  : size: 4 by 3
8.  (-0.055,0.000)  (-0.029,-0.000) (0.370,-0.040)
9.  (-0.055,0.000)  (-0.029,-0.000) (0.913,-0.143)
10. (-0.055,0.000)  (-0.985,-0.158) (-0.042,-0.000)
11. (-0.828,-0.552) (0.043,0.039)   (-0.063,-0.030)
12.
13. The upper triangular matrix R : size: 3 by 3
14. (-18.111,-18.111)       (-25.565,-31.418)       (-42.737,-52.676)
15. (0.000,0.000)   (-20.071,-77.269)       (-11.956,-45.432)
16. (0.000,0.000)   (0.000,0.000)   (-0.593,12.784)
17.
18. Q^H * Q : size: 3 by 3
19. (1.000,0.000)   (0.000,0.000)   (-0.000,-0.000)
20. (0.000,-0.000)  (1.000,0.000)   (-0.000,0.000)
21. (-0.000,0.000)  (-0.000,-0.000) (1.000,0.000)
22.
23. cA - Q*R : size: 4 by 3
24. (-0.000,-0.000) (-0.000,-0.000) (-0.000,-0.000)
25. (0.000,0.000)   (-0.000,-0.000) (-0.000,-0.000)
26. (0.000,0.000)   (0.000,0.000)   (0.000,-0.000)
27. (0.000,0.000)   (0.000,0.000)   (0.000,-0.000)
28.
29. The constant vector cb : size: 4 by 1
```

```
(1.000,0.000)
(0.000,0.000)
(1.000,0.000)
(2.000,0.000)

The least squares solution of cA * x = cb : size: 3 by 1
(-0.002,-0.035)
(-0.002,-0.002)
(0.007,-0.016)

The least squares solution of cA * X = I : size: 3 by 4
(-0.007,0.048)  (-0.025,0.120)  (-0.002,0.012)  (0.004,-0.048)
(-0.001,0.017)  (-0.004,0.042)  (0.001,-0.014)  (-0.001,-0.002)
(0.002,-0.029)  (0.008,-0.072)  (0.000,0.003)   (0.003,0.005)

The cA * X: size: 4 by 4
(0.143,-0.000)  (0.348,0.017)   (0.016,-0.003)  (0.022,-0.016)
(0.348,-0.017)  (0.858,0.000)   (-0.007,0.001)  (-0.009,0.007)
(0.016,0.003)   (-0.007,-0.001) (1.000,-0.000)  (-0.000,0.000)
(0.022,0.016)   (-0.009,-0.007) (-0.000,-0.000) (0.999,0.000)


Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

2.6 实矩阵与复矩阵的 SVD 分解

对于一个m行n列的矩阵A，SVD分解将分主一个m阶的正交矩阵U，一个m行n列的对角矩阵S和一个n阶的正交矩阵V，满足A=U*S*V’，矩阵的SVD分解总是存在的。利用矩阵的SVD分解可以求矩阵的秩，矩阵的 2 范数和条件数以及矩阵的广义逆等等，具体见表 2-11。

表 2-11 实矩阵 SVD 分解

Operation	Effect
SVD<Real> sv	建立 SVD 类
sv.~SVD<Real>()	SVD 析构函数
sv.dec(A)	对实矩阵 A 进行奇异值分解
sv.getU()	获取左奇异向量 U
sv.getV()	获取右奇异向量 V
sv.getSM()	获取奇异值矩阵
sv.getSV()	获取奇异值向量
sv.norm2()	计算矩阵的 2 范数（最大奇异值）

sv.cond(b)	计算矩阵的条件数
sv.rank(B)	计算矩阵的秩

测试代码：

```
/*****************************************************************************
 *                                svd_test.cpp
 *
 * SVD class testing.
 *
 * Zhang Ming, 2010-01 (revised 2010-08), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <svd.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
//    Matrix<Type> A(4,4);
//  A(1,1) = 16;    A(1,2) = 2;     A(1,3) = 3;     A(1,4) = 13;
//  A(2,1) = 5;     A(2,2) = 11;    A(2,3) = 10;    A(2,4) = 8;
//  A(3,1) = 9;     A(3,2) = 7;     A(3,3) = 6;     A(3,4) = 12;
//  A(4,1) = 4;     A(4,2) = 14;    A(4,3) = 15;    A(4,4) = 1;

//    Matrix<Type> A(4,2);
//  A(1,1) = 1;     A(1,2) = 2;
//  A(2,1) = 3;     A(2,2) = 4;
//  A(3,1) = 5;     A(3,2) = 6;
//  A(4,1) = 7;     A(4,2) = 8;

    Matrix<Type> A(2,4);
    A(1,1) = 1;     A(1,2) = 3;     A(1,3) = 5;     A(1,4) = 7;
```

```
    A(2,1) = 2;     A(2,2) = 4;     A(2,3) = 6;     A(2,4) = 8;

    SVD<Type> svd;
    svd.dec(A);

    Matrix<Type> U = svd.getU();
    Matrix<Type> V = svd.getV();
    Matrix<Type> S = svd.getSM();

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "Matrix--A: " << A << endl;
    cout << "Matrix--U: " << U << endl;
    cout << "Vector--S: " << S << endl;
    cout << "Matrix--V: " << V << endl;
    cout << "Matrix--A - U * S * V^T:  "
         << A- U*S*trT(V) << endl;
//         << A- U*multTr(S,V) << endl;

    cout << "The rank of A : " << svd.rank() << endl << endl;
    cout << "The condition number of A : " << svd.cond() << endl << endl;

    return 0;
}
```

运行结果：

```
Matrix--A: size: 2 by 4
1.0000  3.0000  5.0000  7.0000
2.0000  4.0000  6.0000  8.0000

Matrix--U: size: 2 by 2
0.6414  -0.7672
0.7672  0.6414

Vector--S: size: 2 by 2
14.2691 0.0000
0.0000  0.6268

Matrix--V: size: 4 by 2
0.1525  0.8226
0.3499  0.4214
0.5474  0.0201
0.7448  -0.3812

Matrix--A - U * S * V^T:  size: 2 by 4
```

```
-0.0000 0.0000  -0.0000 0.0000
-0.0000 -0.0000 -0.0000 -0.0000

The rank of A : 2

The condition number of A : 22.7640


Process returned 0 (0x0)   execution time : 0.024 s
Press any key to continue.
```

复矩阵的SVD分解如表 2-12所示。

表 2-12　复矩阵 SVD 分解

Operation	Effect
CSVD<Real> sv	建立 CSVD 类
sv.~CSVD<Real>()	CSVD 析构函数
sv.dec(cA)	对复矩阵 cA 进行奇异值分解
sv.getU()	获取左奇异向量 U
sv.getV()	获取右奇异向量 V
sv.getSM()	获取奇异值矩阵
sv.getSV()	获取奇异值向量
sv.norm2()	计算矩阵的 2 范数（最大奇异值）
sv.cond(b)	计算矩阵的条件数
sv.rank(B)	计算矩阵的秩

测试代码：

```
/*****************************************************************************
 *                                csvd_test.cpp
 *
 * CSVD class testing.
 *
 * Zhang Ming, 2010-12, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <csvd.h>

```

```
using namespace std;
using namespace splab;


typedef double  Type;
const   int     M = 4;
const   int     N = 5;


int main()
{
    Matrix< complex<Type> > A( M, N ), B(M,N);
    for( int i=0; i<M; i++ )
        for( int j=0; j<N; j++ )
            A[i][j] = complex<Type>( Type(0.3*i+0.7*j), sin(Type(i+j)) );

    CSVD<Type> svd;
    svd.dec(A);

    Matrix< complex<Type> > U = svd.getU();
    Matrix< complex<Type> > V = svd.getV();
    Matrix<Type> S = svd.getSM();
    Matrix< complex<Type> > CS = complexMatrix( S );
//  Vector<Type> S = svd.getSV();

    cout << setiosflags(ios::fixed) << setprecision(3);
    cout << "Matrix--------A: " << A << endl;
    cout << "Matrix--------U: " << U << endl;
    cout << "Vector--------S: " << S << endl;
    cout << "Matrix--------V: " << V << endl;
    cout << "Matrix--------A - U * S * V^H:  "
         << A - U*CS*trH(V) << endl;
//         << A - U*multTr(CS,V) << endl;

    cout << "The rank of A : " << svd.rank() << endl << endl;
    cout << "The condition number of A : " << svd.cond() << endl << endl;

    return 0;
}
```

运行结果：

```
Matrix--------A: size: 4 by 5
(0.000,0.000)   (0.700,0.841)   (1.400,0.909)   (2.100,0.141)   (2.800,-0.757)

```

```
(0.300,0.841)   (1.000,0.909)   (1.700,0.141)   (2.400,-0.757)  (3.100,-0.959)

(0.600,0.909)   (1.300,0.141)   (2.000,-0.757)  (2.700,-0.959)  (3.400,-0.279)

(0.900,0.141)   (1.600,-0.757)  (2.300,-0.959)  (3.000,-0.279)  (3.700,0.657)


Matrix--------U: size: 4 by 4
(0.394,0.000)   (-0.579,0.000)  (0.632,0.000)   (-0.331,0.000)
(0.466,-0.095)  (-0.431,0.137)  (-0.349,0.187)  (0.642,0.005)
(0.524,-0.121)  (0.130,0.031)   (-0.537,0.070)  (-0.629,-0.064)
(0.571,-0.060)  (0.644,-0.164)  (0.378,-0.085)  (0.275,0.053)

Vector--------S: size: 4 by 4
9.642   0.000   0.000   0.000
0.000   2.428   0.000   0.000
0.000   0.000   1.021   0.000
0.000   0.000   0.000   0.028

Matrix--------V: size: 5 by 4
(0.080,-0.114)  (0.267,0.027)   (0.120,0.734)   (0.561,0.118)
(0.236,-0.077)  (0.254,0.521)   (0.241,0.283)   (-0.451,-0.114)
(0.398,-0.002)  (0.145,0.504)   (0.141,-0.301)  (0.031,-0.058)
(0.547,0.024)   (-0.022,-0.007) (-0.008,-0.372) (0.555,0.074)
(0.677,-0.042)  (-0.156,-0.541) (0.009,0.243)   (-0.371,-0.001)

Matrix--------A - U * S * V^H:  size: 4 by 5
(0.000,0.000)   (0.000,-0.000)  (0.000,-0.000)  (0.000,0.000)   (0.000,0.000)

(0.000,-0.000)  (0.000,-0.000)  (-0.000,0.000)  (0.000,0.000)   (0.000,-0.000)

(0.000,0.000)   (0.000,-0.000)  (0.000,0.000)   (0.000,0.000)   (0.000,0.000)

(0.000,0.000)   (0.000,-0.000)  (-0.000,-0.000) (0.000,-0.000)  (0.000,0.000)


The rank of A : 4

The condition number of A : 350.515


Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

2.7 实矩阵与复矩阵的 EVD 分解

矩阵（或算子）的特征分解是一个非常重要的概念，对于一个矩阵A，存在正交矩阵V和对角矩阵D，满足AV=V*D，其中V的列向量是A的特征向量，D对角线上的元素为对应特征向量的特征值（有可能是复数），更多的说明见”evd.h”。EVD<Type>类的一些常用函数见表 2-13。

表 2-13 实矩阵 EVD 分解

Operation	Effect
EVD<Real> ev	建立 EVD 类
ev.~SVD<Real>()	EVD 析构函数
ev.dec(A)	对实矩阵 A 进行特征分解
isSymmetric()	判断矩阵是否对称
isComplex(tol)	判断矩阵特征值是否为复数
ev.getV()	获取实特征向量 V
ev.getCV()	获取复特征向量 CV
ev.getD()	获取实特征值 D
ev.getCD()	获取复特征值 CD

测试代码：

```
/*****************************************************************************
 *                                evd_test.cpp
 *
 * EVD class testing.
 *
 * Zhang Ming, 2010-01 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <evd.h>


using namespace std;
using namespace splab;


typedef double  Type;
```

```
22. const   int     N = 2;
23.
24.
25. int main()
26. {
27.     Matrix<Type> B(N,N);
28.     B[0][0] = 3.0;  B[0][1] = -2.0;
29.     B[1][0] = -2.0; B[1][1] = 4.0;
30.
31.     Matrix<Type> A(2*N,2*N), D, V;
32.     Matrix<complex<Type> > cA(2*N,2*N), cD, cV;
33.     for( int i=0; i<N; ++i )
34.         for( int j=0; j<N; ++j )
35.         {
36.             A[i][j]         = B[i][j];
37.             A[i][j+N]       = -B[j][i];
38.             A[i+N][j]       = B[j][i];
39.             A[i+N][j+N]     = B[i][j];
40.         }
41. //    A = B;
42.     cA = complexMatrix(A);
43.     cout << setiosflags(ios::fixed) << setprecision(2);
44.     cout << "The original matrix A : " << A << endl;
45.
46.     EVD<Type> eig;
47.     eig.dec(A);
48.     if( !eig.isComplex() )
49.     {
50.         V = eig.getV();
51.         D = diag(eig.getD());
52.         cout << "The eigenvectors matrix V : " << V << endl;
53.         cout << "The eigenvalue D : " << D << endl;
54.         cout << "The V'*V : " << trMult(V,V) << endl;
55.         cout << "The A*V - V*D : " << A*V - V*D << endl;
56.     }
57.     else
58.     {
59.         cV = eig.getCV();
60.         cD = diag(eig.getCD());
61.         cout << "The complex eigenvectors matrix V : " << cV << endl;
62.         cout << "The complex eigenvalue D : " << cD << endl;
63.         cout << "The A*V - V*D : " << cA*cV - cV*cD << endl;
64.     }
65.
```

```
    return 0;
}
```

运行结果：

```
The original matrix A : size: 4 by 4
3.00    -2.00   -3.00   2.00
-2.00   4.00    2.00    -4.00
3.00    -2.00   3.00    -2.00
-2.00   4.00    -2.00   4.00

The complex eigenvectors matrix V : size: 4 by 4
(-0.06,-0.61)   (-0.06,0.61)    (0.06,-0.79)    (0.06,0.79)
(0.08,0.78)     (0.08,-0.78)    (0.05,-0.61)    (0.05,0.61)
(-0.61,0.06)    (-0.61,-0.06)   (-0.79,-0.06)   (-0.79,0.06)
(0.78,-0.08)    (0.78,0.08)     (-0.61,-0.05)   (-0.61,0.05)

The complex eigenvalue D : size: 4 by 4
(5.56,5.56)     (0.00,0.00)     (0.00,0.00)     (0.00,0.00)
(0.00,0.00)     (5.56,-5.56)    (0.00,0.00)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (1.44,1.44)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (0.00,0.00)     (1.44,-1.44)

The A*V - V*D : size: 4 by 4
(0.00,-0.00)    (0.00,0.00)     (-0.00,0.00)    (-0.00,-0.00)
(-0.00,0.00)    (-0.00,-0.00)   (0.00,0.00)     (0.00,-0.00)
(0.00,-0.00)    (0.00,0.00)     (0.00,0.00)     (0.00,-0.00)
(-0.00,-0.00)   (-0.00,0.00)    (0.00,-0.00)    (0.00,0.00)


Process returned 0 (0x0)   execution time : 0.027 s
Press any key to continue.
```

复数矩阵的EVD分解如表 2-14所示。

表 2-14　复矩阵 EVD 分解

Operation	Effect
CEVD<Real> ev	建立 CEVD 类
ev.~CSVD<Real>()	CEVD 析构函数
ev.dec(cA)	对复矩阵 cA 进行特征分解
isHertimian()	判断矩阵是否 Hertimian 对称
ev.getV()	获取特征向量 V
ev.getD()	获取特征值 D

ev.getRD()	获取实特征值 RD

测试代码：

```
/*****************************************************************************
 *                              cevd_test.cpp
 *
 * CEVD class testing.
 *
 * Zhang Ming, 2010-12, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <cevd.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     N = 4;


int main()
{
    Matrix<Type> B(N,N);
    B[0][0] = 3.0;  B[0][1] = -2.0; B[0][2] = -0.9;     B[0][3] = 0.0;
    B[1][0] = -2.0; B[1][1] = 4.0;  B[1][2] = 1.0;      B[1][3] = 0.0;
    B[2][0] = 0.0;  B[2][1] = 0.0;  B[2][2] = -1.0;     B[2][3] = 0.0;
    B[3][0] = -0.5; B[3][1] = -0.5; B[3][2] = 0.1;      B[3][3] = 1.0;

    Matrix<complex<Type> > A = complexMatrix( B, elemMult(B,B) );
//  A = multTr(A,A);
    cout << setiosflags(ios::fixed) << setprecision(2);
    cout << "The original complex matrix A : " << A << endl;

    CEVD<Type> eig;
    eig.dec(A);

```

```
    if( eig.isHertimian() )
    {
        Matrix<complex<Type> > V = eig.getV();
        Vector<Type> D = eig.getRD();
        Matrix<complex<Type> > DM = diag( complexVector(D) );
        cout << "The eigenvectors matrix V is : " << V << endl;
        cout << "The eigenvalue D is : " << diag(D) << endl;
        cout << "The V'*V : " << trMult(V,V) << endl;
        cout << "The A*V - V*D : " << A*V - V*DM << endl;
    }
    else
    {
        Matrix<complex<Type> > V = eig.getV();
        Vector<complex<Type> > D = eig.getD();
        Matrix<complex<Type> > DM = diag( D );
        cout << "The complex eigenvectors matrix V : " << V << endl;
        cout << "The complex eigenvalue D : " << DM << endl;
        cout << "The A*V - V*D : " << A*V - V*DM << endl;
    }

    return 0;
}
```

运行结果：

```
The original complex matrix A : size: 4 by 4
(3.00,9.00)     (-2.00,4.00)    (-0.90,0.81)    (0.00,0.00)
(-2.00,4.00)    (4.00,16.00)    (1.00,1.00)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (-1.00,1.00)    (0.00,0.00)
(-0.50,0.25)    (-0.50,0.25)    (0.10,0.01)     (1.00,1.00)

The complex eigenvectors matrix V : size: 4 by 4
(-0.54,-0.75)   (1.50,-1.10)    (0.00,0.00)     (-0.19,-0.09)
(-1.49,-0.96)   (-0.94,0.24)    (-0.00,-0.00)   (0.05,0.23)
(0.00,0.00)     (0.00,0.00)     (-0.00,0.00)    (0.54,-1.92)
(0.03,-0.08)    (0.06,0.05)     (-1.11,1.67)    (-0.05,0.15)

The complex eigenvalue D : size: 4 by 4
(2.26,17.55)    (0.00,0.00)     (0.00,0.00)     (0.00,0.00)
(0.00,0.00)     (4.74,7.45)     (0.00,0.00)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (1.00,1.00)     (0.00,0.00)
(0.00,0.00)     (0.00,0.00)     (0.00,0.00)     (-1.00,1.00)

The A*V - V*D : size: 4 by 4
(-0.00,0.00)    (0.00,0.00)     (0.00,0.00)     (0.00,0.00)
```

```
(-0.00,0.00)    (-0.00,0.00)    (0.00,-0.00)    (0.00,0.00)
(0.00,-0.00)    (-0.00,-0.00)   (0.00,-0.00)    (-0.00,-0.00)
(-0.00,0.00)    (0.00,0.00)     (-0.00,0.00)    (-0.00,0.00)


Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.
```

2.8 矩阵的逆与广义逆

矩阵求逆运算是极其重要的算法，有着广泛的应用，但其运算量也非常大，一般是N的立方级。SP++中提供的矩阵求逆算法包括：列主元Gauss消元法，全主元Gauss消元法，LUD分解法和Cholesky分解法，其中Cholesky分解法针对对称正定矩阵，可以降低计算量，具体函数见表 2-15所示。

注意：很多矩阵求逆运算稍加推导可以转化为求解线性方程组问题，这样可以减少计算量，有关线性方程组的求解方法可参考下一章的相关内容。

表 2-15 矩阵求逆算法

Operation	Effect
inv(A,type)	普通实矩阵求逆算法
cinv(cA,type)	普通复矩阵求逆算法
colPivInv(A)	实矩阵与复矩阵列主元 Gauss 消元法
cmpPivInv(A)	实矩阵与复矩阵全主元 Gauss 消元法

测试代码：

```
/*****************************************************************************
 *                              inverse_test.cpp
 *
 * Matrix inverse testing.
 *
 * Zhang Ming, 2010-08 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <inverse.h>

```

```
17. using namespace std;
18. using namespace splab;
19. 
20. 
21. typedef double  Type;
22. const   int     N = 3;
23. 
24. int main()
25. {
26.     Matrix<Type> A, invA, B(N,N);
27.     A.resize(3,3);
28.     A[0][0] = 1;    A[0][1] = 2;    A[0][2] = 1;
29.     A[1][0] = 2;    A[1][1] = 5;    A[1][2] = 4;
30.     A[2][0] = 1;    A[2][1] = 1;    A[2][2] = 0;
31. 
32.     cout << setiosflags(ios::fixed) << setprecision(4);
33.     cout << "The original matrix A is : " << A << endl;
34.     invA = inv(A);
35.     cout << "The inverse matrix of A (LUD) : " << invA << endl;
36.     invA = colPivInv(A);
37.     cout << "The inverse matrix of A (column pivot) : " << invA << endl;
38.     invA = cmpPivInv(A);
39.     cout << "The invese matrix of A (complete pivot) : " << invA << endl;
40.     cout << "The multiplication of A and its inverse: " << A*invA << endl;
41. 
42.     for( int i=1; i<=N; ++i )
43.     {
44.         for( int j=1; j<=N; ++j )
45.             if( i == j )
46.                 B(i,i) = i;
47.             else if( i < j )
48.                 B(i,j) = i;
49.             else
50.                 B(i,j) = j;
51.     }
52.     cout << "The original matrix B is : " << B << endl;
53.     invA = inv(B,"spd");
54.     cout << "The inverse matrix of B (Cholesky) : " << invA << endl;
55.     invA = colPivInv(B);
56.     cout << "The inverse matrix of B (column pivot) : " << invA << endl;
57.     invA = cmpPivInv(B);
58.     cout << "The inverse matrix of B (complete pivot) : " << invA << endl;
59.     cout << "The multiplication of B and its inverse: " << B*invA << endl;
60. 
```

```
61.     cout << setiosflags(ios::fixed) << setprecision(3);
62.     Matrix<complex<Type> > cA(N,N), cIA;
63.     cA = complexMatrix( A, B );
64.
65.     cIA = inv(cA);
66.     cout << "The original complex matrix cA is: " << cA << endl;
67.     cout << "The inverse matrix of cA is (general method): "
68.          << inv(cA) << endl;
69.     cout << "The inverse matrix of cA is (column pivot): "
70.          << colPivInv(cA) << endl;
71.     cout << "The inverse matrix of cA is (complete pivot): "
72.          << cmpPivInv(cA) << endl;
73.     cout << "The inverse matrix of cA is (real inverse): "
74.          << cinv(cA) << endl;
75.     cout << "The multiplication of cA and its inverse: "
76.          << cA*cIA << endl;
77.
78.     return 0;
79. }
```

运行结果：

```
1.  The original matrix A is : size: 3 by 3
2.  1.0000  2.0000  1.0000
3.  2.0000  5.0000  4.0000
4.  1.0000  1.0000  0.0000
5.
6.  The inverse matrix of A (LUD) : size: 3 by 3
7.  -4.0000 1.0000  3.0000
8.  4.0000  -1.0000 -2.0000
9.  -3.0000 1.0000  1.0000
10.
11. The inverse matrix of A (column pivot) : size: 3 by 3
12. -4.0000 1.0000  3.0000
13. 4.0000  -1.0000 -2.0000
14. -3.0000 1.0000  1.0000
15.
16. The invese matrix of A (complete pivot) : size: 3 by 3
17. -4.0000 1.0000  3.0000
18. 4.0000  -1.0000 -2.0000
19. -3.0000 1.0000  1.0000
20.
21. The multiplication of A and its inverse: size: 3 by 3
22. 1.0000  0.0000  0.0000
23. 0.0000  1.0000  0.0000
```

```
0.0000  0.0000  1.0000

The original matrix B is : size: 3 by 3
1.0000  1.0000  1.0000
1.0000  2.0000  2.0000
1.0000  2.0000  3.0000

The inverse matrix of B (Cholesky) : size: 3 by 3
2.0000  -1.0000 0.0000
-1.0000 2.0000  -1.0000
0.0000  -1.0000 1.0000

The inverse matrix of B (column pivot) : size: 3 by 3
2.0000  -1.0000 -0.0000
-1.0000 2.0000  -1.0000
0.0000  -1.0000 1.0000

The inverse matrix of B (complete pivot) : size: 3 by 3
2.0000  -1.0000 -0.0000
-1.0000 2.0000  -1.0000
-0.0000 -1.0000 1.0000

The multiplication of B and its inverse: size: 3 by 3
1.0000  0.0000  0.0000
0.0000  1.0000  0.0000
0.0000  0.0000  1.0000

The original complex matrix cA is: size: 3 by 3
(1.000,1.000)   (2.000,1.000)   (1.000,1.000)
(2.000,1.000)   (5.000,2.000)   (4.000,2.000)
(1.000,1.000)   (1.000,2.000)   (0.000,3.000)

The inverse matrix of cA is (general method): size: 3 by 3
(1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
(-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
(1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)

The inverse matrix of cA is (column pivot): size: 3 by 3
(1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
(-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
(1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)

The inverse matrix of cA is (complete pivot): size: 3 by 3
(1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
```

```
(-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
(1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)

The inverse matrix of cA is (real inverse): size: 3 by 3
(1.400,-3.200)  (-0.200,1.600)  (-1.400,0.200)
(-2.000,1.000)  (1.000,-1.000)  (1.000,1.000)
(1.600,0.200)   (-0.800,0.400)  (-0.600,-1.200)

The multiplication of cA and its inverse: size: 3 by 3
(1.000,0.000)   (0.000,0.000)   (-0.000,-0.000)
(0.000,0.000)   (1.000,-0.000)  (0.000,0.000)
(0.000,0.000)   (0.000,0.000)   (1.000,-0.000)


Process returned 0 (0x0)   execution time : 0.168 s
Press any key to continue.
```

当矩阵不是方阵或矩阵非满秩时，就需要利用广义逆矩阵求解某些线性方程组，矩阵的广义逆可以通过通过SVD分解计算，SP++中提供了实矩阵与复矩阵的广义逆求解算法，如表 2-16所示。

注意：测试文件“pseudoinverse_test.cpp”在 GCC 编译器（CodeBlocks 环境）下没有错误，在 VS2010 环境下有错误，提示函数模板重载出现二义性，可以将复数矩阵的广义逆函数 pinv(cA,tol)改个名字即可，比如改为 cpinv。

表 2-16　矩阵的广义逆

Operation	Effect
pinv(A,tol)	实矩阵的广义逆
pinv(cA,tol)	复矩阵的广义逆

测试代码：

```
/*****************************************************************************
 *                            pseudoinverse_test.cpp
 *
 * Matrix pseudoinverse testing.
 *
 * Zhang Ming, 2010-08 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
```

```
#include <pseudoinverse.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
    Matrix<Type> A(8,6), invA;
    A[0][0]=64; A[0][1]=2;  A[0][2]=3;  A[0][3]=61; A[0][4]=60; A[0][5]=6;
    A[1][0]=9;  A[1][1]=55; A[1][2]=54; A[1][3]=12; A[1][4]=13; A[1][5]=51;
    A[2][0]=17; A[2][1]=47; A[2][2]=46; A[2][3]=20; A[2][4]=21; A[2][5]=43;
    A[3][0]=40; A[3][1]=26; A[3][2]=27; A[3][3]=37; A[3][4]=36; A[3][5]=30;
    A[4][0]=32; A[4][1]=34; A[4][2]=35; A[4][3]=29; A[4][4]=28; A[4][5]=38;
    A[5][0]=41; A[5][1]=23; A[5][2]=22; A[5][3]=44; A[5][4]=45; A[5][5]=19;
    A[6][0]=49; A[6][1]=15; A[6][2]=14; A[6][3]=52; A[6][4]=53; A[6][5]=11;
    A[7][0]=8;  A[7][1]=58; A[7][2]=59; A[7][3]=5;  A[7][4]=4;  A[7][5]=62;

    invA = pinv(A);
    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "The original matrix A is : " << A << endl;
    cout << "The pseudoinverse matrix of A is : " << invA << endl;
    cout << "The multiplication of A and its inverse is : " << A*invA << endl;

    Matrix<complex<Type> > cA = complexMatrix(A,-A);
    Matrix<complex<Type> > cPIA = pinv(cA);
    cout << "The original complex matrix A is : "
         << setprecision(0) << cA << endl;
    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "The real part of the pseudoinverse matrix of A is: "
         << real(cPIA) << endl;
    cout << "The real imaginary of the pseudoinverse matrix of A is: "
         << imag(cPIA) << endl;

    return 0;
}
```

运行结果：

```
The original matrix A is : size: 8 by 6
64.0000 2.0000  3.0000  61.0000 60.0000 6.0000
```

```
 9.0000  55.0000 54.0000 12.0000 13.0000 51.0000
 17.0000 47.0000 46.0000 20.0000 21.0000 43.0000
 40.0000 26.0000 27.0000 37.0000 36.0000 30.0000
 32.0000 34.0000 35.0000 29.0000 28.0000 38.0000
 41.0000 23.0000 22.0000 44.0000 45.0000 19.0000
 49.0000 15.0000 14.0000 52.0000 53.0000 11.0000
 8.0000  58.0000 59.0000 5.0000  4.0000  62.0000

The pseudoinverse matrix of A is : size: 6 by 8
0.0177  -0.0165 -0.0164 0.0174  0.0173  -0.0161 -0.0160 0.0170
-0.0121 0.0132  0.0130  -0.0114 -0.0112 0.0124  0.0122  -0.0106
-0.0055 0.0064  0.0060  -0.0043 -0.0040 0.0049  0.0045  -0.0028
-0.0020 0.0039  0.0046  -0.0038 -0.0044 0.0064  0.0070  -0.0063
-0.0086 0.0108  0.0115  -0.0109 -0.0117 0.0139  0.0147  -0.0141
0.0142  -0.0140 -0.0149 0.0169  0.0178  -0.0176 -0.0185 0.0205

The multiplication of A and its inverse is : size: 8 by 8
0.5417  -0.2083 -0.1250 0.2917  0.2083  0.1250  0.2083  -0.0417
-0.2083 0.3988  0.3393  -0.0298 0.0298  0.1607  0.1012  0.2083
-0.1250 0.3393  0.3036  -0.0179 0.0179  0.1964  0.1607  0.1250
0.2917  -0.0298 -0.0179 0.2560  0.2440  0.0179  0.0298  0.2083
0.2083  0.0298  0.0179  0.2440  0.2560  -0.0179 -0.0298 0.2917
0.1250  0.1607  0.1964  0.0179  -0.0179 0.3036  0.3393  -0.1250
0.2083  0.1012  0.1607  0.0298  -0.0298 0.3393  0.3988  -0.2083
-0.0417 0.2083  0.1250  0.2083  0.2917  -0.1250 -0.2083 0.5417

The original complex matrix A is : size: 8 by 6
(64,-64)        (2,-2)  (3,-3)  (61,-61)        (60,-60)        (6,-6)
(9,-9)  (55,-55)        (54,-54)        (12,-12)        (13,-13)        (51,-51)

(17,-17)        (47,-47)        (46,-46)        (20,-20)        (21,-21)
(43,-43)
(40,-40)        (26,-26)        (27,-27)        (37,-37)        (36,-36)
(30,-30)
(32,-32)        (34,-34)        (35,-35)        (29,-29)        (28,-28)
(38,-38)
(41,-41)        (23,-23)        (22,-22)        (44,-44)        (45,-45)
(19,-19)
(49,-49)        (15,-15)        (14,-14)        (52,-52)        (53,-53)
(11,-11)
(8,-8)  (58,-58)        (59,-59)        (5,-5)  (4,-4)  (62,-62)

The real part of the pseudoinverse matrix of A is: size: 6 by 8
0.0089  -0.0083 -0.0082 0.0087  0.0087  -0.0081 -0.0080 0.0085
```

```
-0.0060 0.0066  0.0065  -0.0057 -0.0056 0.0062  0.0061  -0.0053
-0.0027 0.0032  0.0030  -0.0022 -0.0020 0.0025  0.0023  -0.0014
-0.0010 0.0020  0.0023  -0.0019 -0.0022 0.0032  0.0035  -0.0031
-0.0043 0.0054  0.0058  -0.0055 -0.0059 0.0069  0.0073  -0.0070
0.0071  -0.0070 -0.0075 0.0085  0.0089  -0.0088 -0.0093 0.0103

The real imaginary of the pseudoinverse matrix of A is: size: 6 by 8
0.0089  -0.0083 -0.0082 0.0087  0.0087  -0.0081 -0.0080 0.0085
-0.0060 0.0066  0.0065  -0.0057 -0.0056 0.0062  0.0061  -0.0053
-0.0027 0.0032  0.0030  -0.0022 -0.0020 0.0025  0.0023  -0.0014
-0.0010 0.0020  0.0023  -0.0019 -0.0022 0.0032  0.0035  -0.0031
-0.0043 0.0054  0.0058  -0.0055 -0.0059 0.0069  0.0073  -0.0070
0.0071  -0.0070 -0.0075 0.0085  0.0089  -0.0088 -0.0093 0.0103


Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

3 线性方程组

3.1 常规线性方程组

常规线性方程组有诸多解法，若系数矩阵为一般的满秩矩阵，则可以用列主元Gauss消元法或LU分解法进行求解；若系数矩阵为对称正定矩阵，则可以用Cholesky分解法进行求解；若系数矩阵为三对角矩阵，则可以用追赶法进行快速求解。矩阵方程其实是求解多个线性方程组，故解法一致，具体函数见表 3-1，该表中提供的函数对实系数线性方程组和复系数线性方程组均可适用。

表 3-1 常规线性方程组求解方法

Operation	Effect
gaussSolver(A,B)	Gauss 消元法解矩阵方程
gaussSolver(A,b)	Gauss 消元法解线性方程组
luSolver(A,B)	LU 分解法解矩阵方程
luSolver(A,b)	LU 分解法解线性方程组
choleskySolver(A,B)	Cholesky 分解法解矩阵方程
choleskySolver(A,b)	Cholesky 分解法解线性方程组
utSolver(A,b)	求解上三角系数矩阵线性方程组
ltSolver(A,b)	求解下三角系数矩阵线性方程组
febsSolver(A,b)	追赶法解三对角方程组

测试代码：

```
1.  /*****************************************************************************
2.   *                              linequs1_test.cpp
3.   *
4.   * Deterministic Linear Equations testing.
5.   *
6.   * Zhang Ming, 2010-07 (revised 2010-12), Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10.  #define BOUNDS_CHECK
11.
12.  #include <iostream>
13.  #include <iomanip>
14.  #include <linequs1.h>
```

```
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. typedef float   Type;
22. const   int     M = 3;
23. const   int     N = 3;
24.
25.
26. int main()
27. {
28.     Matrix<Type> A(M,N), B;
29.     Vector<Type> b(N);
30.
31.     // ordinary linear equations
32.     A[0][0] = 1;    A[0][1] = 2;    A[0][2] = 1;
33.     A[1][0] = 2;    A[1][1] = 5;    A[1][2] = 4;
34.     A[2][0] = 1;    A[2][1] = 1;    A[2][2] = 0;
35.     B = eye( N, Type(1.0) );
36.     b[0] = 1;   b[1] = 0;   b[2] = 1;
37.     Matrix<Type> invA( A );
38.     Matrix<Type> X( B );
39.
40.     cout << setiosflags(ios::fixed) << setprecision(4) << endl;
41.     cout << "The original matrix A is : " << A << endl;
42.     gaussSolver( invA, X );
43.     cout << "The inverse of A is (Gauss Solver) : "
44.          << invA << endl;
45.     cout << "The inverse matrix of A is (LUD Solver) : "
46.          << luSolver( A, B ) << endl;
47.     cout << "The constant vector b : " << b << endl;
48.     cout << "The solution of A * x = b is (Gauss Solver) : "
49.          << gaussSolver( A, b ) << endl;
50.     cout << "The solution of A * x = b is (LUD Solver) : "
51.          << luSolver( A, b ) << endl << endl;
52.
53.     Matrix<complex<Type> > cA = complexMatrix( A, B );
54.     Vector<complex<Type> > cb = complexVector( b, b );
55.     cout << "The original complex matrix cA is : " << cA << endl;
56.     cout << "The constant complex vector cb : " << cb << endl;
57.     cout << "The solution of cA * cx = cb is (Gauss Solver) : "
58.          << gaussSolver( cA, cb ) << endl;
```

```
59.     cout << "The solution of cA * cx = cb is (LUD Solver) : "
60.          << luSolver( cA, cb ) << endl;
61.     cout << "The cA*cx - cb is : "<< cA*luSolver(cA,cb) - cb << endl << endl;
62.
63.     // linear equations with symmetric coefficient matrix
64.     for( int i=1; i<N+1; ++i )
65.     {
66.         for( int j=1; j<N+1; ++j )
67.             if( i == j )
68.                 A(i,i) = Type(i);
69.             else
70.                 if( i < j )
71.                     A(i,j) = Type(i);
72.                 else
73.                     A(i,j) = Type(j);
74.         b(i) = Type( i*(i+1)/2.0 + i*(N-i) );
75.     }
76.     cout << "The original matrix A : " << A << endl;
77.     cout << "The inverse matrix of A is (Cholesky Solver) : "
78.          << choleskySolver( A, B ) << endl;
79.     cout << "The constant vector b : " << b << endl;
80.     cout << "The solution of Ax = b is (Cholesky Solver) : "
81.          << choleskySolver( A, b ) << endl << endl;
82.
83.     cA = complexMatrix( A );
84.     cb = complexVector( b, b );
85.     cout << "The original complex matrix A : " << cA << endl;
86.     cout << "The constant complex vector b : " << cb << endl;
87.     cout << "The solution of Ax = b is (Cholesky Solver) : "
88.          << choleskySolver( cA, cb ) << endl << endl;
89.
90.     // upper and lower triangular system
91.     for( int i=0; i<N; ++i )
92.         for( int j=i; j<N; ++j )
93.             B[i][j] = A[i][j];
94.     cout << "The original matrix B : " << B << endl;
95.     cout << "The constant vector b : " << b << endl;
96.     cout << "The solution of Ax = b is (Upper Triangular Solver) : "
97.          << utSolver( B, b ) << endl << endl;
98.
99.     cA = complexMatrix( trT(B), -trT(B) );
100.    cout << "The original complex matrix A : " << cA << endl;
101.    cout << "The constant complex vector b : " << cb << endl;
102.    cout << "The solution of Ax = b is (Lower Triangular Solver) : "
```

```
103.         << ltSolver( cA, cb ) << endl << endl;
104.
105.     // Tridiagonal linear equations
106.     Vector<Type> aa( 3, -1 ), bb( 4, 4 ), cc( 3, -2 ), dd( 4 );
107.     dd(1) = 3;   dd(2) = 2;   dd(3) = 2;   dd(4) = 3;
108.     cout << "The elements below main diagonal is : " << aa << endl;
109.     cout << "The elements on main diagonal is : " << bb << endl;
110.     cout << "The elements above main diagonal is : " << cc << endl;
111.     cout << "The elements constant vector is : " << dd << endl;
112.     cout << "Teh solution is (Forward Elimination and Backward Substitution) : "
113.          << febsSolver( aa, bb, cc, dd ) << endl;
114.
115.     Vector<complex<Type> > caa = complexVector(aa);
116.     Vector<complex<Type> > cbb = complexVector(bb);
117.     Vector<complex<Type> > ccc = complexVector(cc);
118.     Vector<complex<Type> > cdd = complexVector(Vector<Type>(dd.dim()),dd);
119.     cout << "The elements below main diagonal is : " << caa << endl;
120.     cout << "The elements on main diagonal is : " << cbb << endl;
121.     cout << "The elements above main diagonal is : " << ccc << endl;
122.     cout << "The elements constant vector is : " << cdd << endl;
123.     cout << "Teh solution is (Forward Elimination and Backward Substitution) : "
124.          << febsSolver( caa, cbb, ccc, cdd ) << endl;
125.
126.     return 0;
127. }
```

运行结果：

```
1.  The original matrix A is : size: 3 by 3
2.  1.0000  2.0000  1.0000
3.  2.0000  5.0000  4.0000
4.  1.0000  1.0000  0.0000
5.
6.  The inverse of A is (Gauss Solver) : size: 3 by 3
7.  -4.0000 1.0000  3.0000
8.  4.0000  -1.0000 -2.0000
9.  -3.0000 1.0000  1.0000
10.
11. The inverse matrix of A is (LUD Solver) : size: 3 by 3
12. -4.0000 1.0000  3.0000
13. 4.0000  -1.0000 -2.0000
14. -3.0000 1.0000  1.0000
15.
16. The constant vector b : size: 3 by 1
17. 1.0000
```

```
0.0000
1.0000

The solution of A * x = b is (Gauss Solver) : size: 3 by 1
-1.0000
2.0000
-2.0000

The solution of A * x = b is (LUD Solver) : size: 3 by 1
-1.0000
2.0000
-2.0000


The original complex matrix cA is : size: 3 by 3
(1.0000,1.0000) (2.0000,0.0000) (1.0000,0.0000)
(2.0000,0.0000) (5.0000,1.0000) (4.0000,0.0000)
(1.0000,0.0000) (1.0000,0.0000) (0.0000,1.0000)

The constant complex vector cb : size: 3 by 1
(1.0000,1.0000)
(0.0000,0.0000)
(1.0000,1.0000)

The solution of cA * cx = cb is (Gauss Solver) : size: 3 by 1
(0.4000,-0.8000)
(-0.4000,1.2000)
(0.6000,-1.0000)

The solution of cA * cx = cb is (LUD Solver) : size: 3 by 1
(0.4000,-0.8000)
(-0.4000,1.2000)
(0.6000,-1.0000)

The cA*cx - cb is : size: 3 by 1
(-0.0000,0.0000)
(0.0000,0.0000)
(-0.0000,-0.0000)


The original matrix A : size: 3 by 3
1.0000  1.0000  1.0000
1.0000  2.0000  2.0000
1.0000  2.0000  3.0000
```

```
62.
63. The inverse matrix of A is (Cholesky Solver) : size: 3 by 3
64. 2.0000  -1.0000 0.0000
65. -1.0000 2.0000  -1.0000
66. 0.0000  -1.0000 1.0000
67.
68. The constant vector b : size: 3 by 1
69. 3.0000
70. 5.0000
71. 6.0000
72.
73. The solution of Ax = b is (Cholesky Solver) : size: 3 by 1
74. 1.0000
75. 1.0000
76. 1.0000
77.
78.
79. The original complex matrix A : size: 3 by 3
80. (1.0000,0.0000) (1.0000,0.0000) (1.0000,0.0000)
81. (1.0000,0.0000) (2.0000,0.0000) (2.0000,0.0000)
82. (1.0000,0.0000) (2.0000,0.0000) (3.0000,0.0000)
83.
84. The constant complex vector b : size: 3 by 1
85. (3.0000,3.0000)
86. (5.0000,5.0000)
87. (6.0000,6.0000)
88.
89. The solution of Ax = b is (Cholesky Solver) : size: 3 by 1
90. (1.0000,1.0000)
91. (1.0000,1.0000)
92. (1.0000,1.0000)
93.
94.
95. The original matrix B : size: 3 by 3
96. 1.0000  1.0000  1.0000
97. 0.0000  2.0000  2.0000
98. 0.0000  0.0000  3.0000
99.
100. The constant vector b : size: 3 by 1
101. 3.0000
102. 5.0000
103. 6.0000
104.
105. The solution of Ax = b is (Upper Triangular Solver) : size: 3 by 1
```

```
0.5000
0.5000
2.0000


The original complex matrix A : size: 3 by 3
(1.0000,-1.0000)        (0.0000,-0.0000)        (0.0000,-0.0000)
(1.0000,-1.0000)        (2.0000,-2.0000)        (0.0000,-0.0000)
(1.0000,-1.0000)        (2.0000,-2.0000)        (3.0000,-3.0000)

The constant complex vector b : size: 3 by 1
(3.0000,3.0000)
(5.0000,5.0000)
(6.0000,6.0000)

The solution of Ax = b is (Lower Triangular Solver) : size: 3 by 1
(0.0000,3.0000)
(0.0000,1.0000)
(0.0000,0.3333)


The elements below main diagonal is : size: 3 by 1
-1.0000
-1.0000
-1.0000

The elements on main diagonal is : size: 4 by 1
4.0000
4.0000
4.0000
4.0000

The elements above main diagonal is : size: 3 by 1
-2.0000
-2.0000
-2.0000

The elements constant vector is : size: 4 by 1
3.0000
2.0000
2.0000
3.0000

Teh solution is (Forward Elimination and Backward Substitution) : size: 4 by 1
```

```
1.5610
1.6220
1.4634
1.1159

The elements below main diagonal is : size: 3 by 1
(-1.0000,0.0000)
(-1.0000,0.0000)
(-1.0000,0.0000)

The elements on main diagonal is : size: 4 by 1
(4.0000,0.0000)
(4.0000,0.0000)
(4.0000,0.0000)
(4.0000,0.0000)

The elements above main diagonal is : size: 3 by 1
(-2.0000,0.0000)
(-2.0000,0.0000)
(-2.0000,0.0000)

The elements constant vector is : size: 4 by 1
(0.0000,3.0000)
(0.0000,2.0000)
(0.0000,2.0000)
(0.0000,3.0000)

Teh solution is (Forward Elimination and Backward Substitution) : size: 4 by 1
(0.0000,1.5610)
(0.0000,1.6220)
(0.0000,1.4634)
(0.0000,1.1159)


Process returned 0 (0x0)   execution time : 0.162 s
Press any key to continue.
```

3.2 超定与欠定线性方程组

若系数矩阵的行数大于列数，则称该方程组为超定线性方程组，这样的方程组不存在精确解，但可以求得与常数向量之间误差能量最小的解，即最小二乘解，主要的求解方法有最小二乘广义逆矩阵法和QR分解法。若系数矩阵的行数小于列数，

则称该方程组为欠定线性方程组，这样的方程组存在无穷多组解，但是可以找到这些解当中能量最小的一个，即极小范数解，主要的求解方法有最小范数广义逆矩阵法和QR分解法。超定与欠定线性方程组的具体函数调用形式详见表 3-2，该表中提供的函数对实系数线性方程组和复系数线性方程组均可适用。

表 3-2　超定与欠定线性方程组求解方法

Operation	Effect
lsSolver(A,b)	最小二乘广义逆解超定线性方程组
qrLsSolver(A,b)	QR 分解法解超定线性方程组
svdLsSolver(A,b)	SVD 分解法解超定线性方程组
lnSolver(A,b)	极小范数广义逆解欠定线性方程组
qrLnSolver(A,b)	QR 分解法解欠定线性方程组
svdLnSolver(A,b)	QR 分解法解欠定线性方程组

测试代码：

```
/*****************************************************************************
 *                              linequs2_test.cpp
 *
 * Undetermined Linear Equations testing.
 *
 * Zhang Ming, 2010-07 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <linequs2.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     M = 3;
const   int     N = 3;


int main()
{
    Matrix<Type> A(M,N), B(M,N);
```

```
29.     Vector<Type> b(N);
30.
31.     // overdetermined linear equations
32.     A.resize( 4, 3 );
33.     A[0][0] = 1;    A[0][1] = -1;   A[0][2] = 1;
34.     A[1][0] = 1;    A[1][1] = 2;    A[1][2] = 4;
35.     A[2][0] = 1;    A[2][1] = 3;    A[2][2] = 9;
36.     A[3][0] = 1;    A[3][1] = -4;   A[3][2] = 16;
37.     b.resize( 4 );
38.     b[0]= 1;    b[1] = 2;   b[2] = 3;   b[3] = 4;
39.
40.     cout << setiosflags(ios::fixed) << setprecision(3);
41.     cout << "The original matrix A : " << A << endl;
42.     cout << "The constant vector b : " << b << endl;
43.     cout << "The least square solution is (using generalized inverse) : "
44.          << lsSolver( A, b ) << endl;
45.     cout << "The least square solution is (using QR decomposition) : "
46.          << qrLsSolver( A, b ) << endl;
47.     cout << "The least square solution is (using SVD decomposition) : "
48.          << svdLsSolver( A, b ) << endl;
49.
50.     Matrix<complex<Type> > cA = complexMatrix( A, A );
51.     Vector<complex<Type> > cb = complexVector( b );
52.     cout << "The original complex matrix cA : " << cA << endl;
53.     cout << "The constant complex vector cb : " << cb << endl;
54.     cout << "The least square solution is (using generalized inverse) : "
55.          << lsSolver( cA, cb ) << endl;
56.     cout << "The least square solution is (using QR decomposition) : "
57.          << qrLsSolver( cA, cb ) << endl;
58.     cout << "The least square solution is (using SVD decomposition) : "
59.          << svdLsSolver( cA, cb ) << endl;
60.
61.     // undetermined linear equations
62.     Matrix<Type> At( trT( A ) );
63.     b.resize( 3 );
64.     b[0]= 1;    b[1] = 2;   b[2]= 3;
65.     cout << "The original matrix A : " << At << endl;
66.     cout << "The constant vector b : " << b << endl;
67.     cout << "The least norm solution is (using generalized inverse) : "
68.          << lnSolver( At, b ) << endl;
69.     cout << "The least norm solution is (using QR decomposition) : "
70.          << qrLnSolver( At, b ) << endl;
71.     cout << "The least norm solution is (using SVD decomposition) : "
72.          << svdLnSolver( At, b ) << endl;
```

```
73.
74.     cA = complexMatrix( At, -At );
75.     cb = complexVector( b );
76.     cout << "The original complex matrix cA : " << cA << endl;
77.     cout << "The constant complex vector cb : " << cb << endl;
78.     cout << "The least square solution is (using generalized inverse) : "
79.          << lnSolver( cA, cb ) << endl;
80.     cout << "The least square solution is (using QR decomposition) : "
81.          << qrLnSolver( cA, cb ) << endl;
82.     cout << "The least square solution is (using SVD decomposition) : "
83.          << svdLnSolver( cA, cb ) << endl;
84.
85.     return 0;
86. }
```

运行结果：

```
1.  The original matrix A : size: 4 by 3
2.  1.000   -1.000  1.000
3.  1.000   2.000   4.000
4.  1.000   3.000   9.000
5.  1.000   -4.000  16.000
6.
7.  The constant vector b : size: 4 by 1
8.  1.000
9.  2.000
10. 3.000
11. 4.000
12.
13. The least square solution is (using generalized inverse) : size: 3 by 1
14. 0.909
15. 0.079
16. 0.212
17.
18. The least square solution is (using QR decomposition) : size: 3 by 1
19. 0.909
20. 0.079
21. 0.212
22.
23. The least square solution is (using SVD decomposition) : size: 3 by 1
24. 0.909
25. 0.079
26. 0.212
27.
28. The original complex matrix cA : size: 4 by 3
```

```
(1.000,1.000)   (-1.000,-1.000) (1.000,1.000)
(1.000,1.000)   (2.000,2.000)   (4.000,4.000)
(1.000,1.000)   (3.000,3.000)   (9.000,9.000)
(1.000,1.000)   (-4.000,-4.000) (16.000,16.000)

The constant complex vector cb : size: 4 by 1
(1.000,0.000)
(2.000,0.000)
(3.000,0.000)
(4.000,0.000)

The least square solution is (using generalized inverse) : size: 3 by 1
(0.455,-0.455)
(0.039,-0.039)
(0.106,-0.106)

The least square solution is (using QR decomposition) : size: 3 by 1
(0.455,-0.455)
(0.039,-0.039)
(0.106,-0.106)

The least square solution is (using SVD decomposition) : size: 3 by 1
(0.455,-0.455)
(0.039,-0.039)
(0.106,-0.106)

The original matrix A : size: 3 by 4
1.000   1.000   1.000   1.000
-1.000  2.000   3.000   -4.000
1.000   4.000   9.000   16.000

The constant vector b : size: 3 by 1
1.000
2.000
3.000

The least norm solution is (using generalized inverse) : size: 4 by 1
0.373
0.421
0.336
-0.130

The least norm solution is (using QR decomposition) : size: 4 by 1
0.373
```

```
0.421
0.336
-0.130

The least norm solution is (using SVD decomposition) : size: 4 by 1
0.373
0.421
0.336
-0.130

The original complex matrix cA : size: 3 by 4
(1.000,-1.000)  (1.000,-1.000)  (1.000,-1.000)  (1.000,-1.000)
(-1.000,1.000)  (2.000,-2.000)  (3.000,-3.000)  (-4.000,4.000)
(1.000,-1.000)  (4.000,-4.000)  (9.000,-9.000)  (16.000,-16.000)

The constant complex vector cb : size: 3 by 1
(1.000,0.000)
(2.000,0.000)
(3.000,0.000)

The least square solution is (using generalized inverse) : size: 4 by 1
(0.186,0.186)
(0.211,0.211)
(0.168,0.168)
(-0.065,-0.065)

The least square solution is (using QR decomposition) : size: 4 by 1
(0.186,0.186)
(0.211,0.211)
(0.168,0.168)
(-0.065,-0.065)

The least square solution is (using SVD decomposition) : size: 4 by 1
(0.186,0.186)
(0.211,0.211)
(0.168,0.168)
(-0.065,-0.065)


Process returned 0 (0x0)   execution time : 0.108 s
Press any key to continue.
```

3.3 病态线性方程组

如果系数矩阵是秩亏矩阵，则称该方程组是病态方程组，这类方程组的求解方法是不稳定的，故不能用常规方法求解。为了提解的高稳定性，必须牺牲解的准确性，比较常用的解法有截断SVD法，阻尼SVD法和Tikhonov正则化方法等等。SP++中提供了这三类求解方法，具体函数调用格式见表 3-3，该表中提供的函数对实系数线性方程组和复系数线性方程组均可适用。

注意：测试文件“linequs3_test.cpp”在 GCC 编译器（CodeBlocks 环境）下没有错误，在 VS2010 环境下有错误，提示函数模板重载出现二义性，可以将函数改个名字即可，比如将求解复系数方程组的函数前加个字母’c’等。

表 3-3 病态线性方法组求解方法

Operation	Effect
tsvd(A,b,tol)	截断 SVD 法解秩亏线性方程组
dsvd(A,b,sigma)	阻尼 SVD 法解秩亏线性方程组
tikhonov(A,b,alpha)	Tikhonov 正则化法解秩亏线性方程组

测试代码：

```
/*****************************************************************************
 *                              linequs3_test.cpp
 *
 * Rank Defect Linear Equations testing.
 *
 * Zhang Ming, 2010-07 (revised 2010-12), Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <linequs3.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
```

```
26.     Matrix<Type> A(8,6);
27.     A[0][0]=64; A[0][1]=2;  A[0][2]=3;  A[0][3]=61; A[0][4]=60; A[0][5]=6;
28.     A[1][0]=9;  A[1][1]=55; A[1][2]=54; A[1][3]=12; A[1][4]=13; A[1][5]=51;
29.     A[2][0]=17; A[2][1]=47; A[2][2]=46; A[2][3]=20; A[2][4]=21; A[2][5]=43;
30.     A[3][0]=40; A[3][1]=26; A[3][2]=27; A[3][3]=37; A[3][4]=36; A[3][5]=30;
31.     A[4][0]=32; A[4][1]=34; A[4][2]=35; A[4][3]=29; A[4][4]=28; A[4][5]=38;
32.     A[5][0]=41; A[5][1]=23; A[5][2]=22; A[5][3]=44; A[5][4]=45; A[5][5]=19;
33.     A[6][0]=49; A[6][1]=15; A[6][2]=14; A[6][3]=52; A[6][4]=53; A[6][5]=11;
34.     A[7][0]=8;  A[7][1]=58; A[7][2]=59; A[7][3]=5;  A[7][4]=4;  A[7][5]=62;
35.
36.     Vector<Type> b(8);
37.     b[0]=260; b[1]=260; b[2]=260; b[3]=260;
38.     b[4]=260; b[5]=260; b[6]=260; b[7]=260;
39.
40.     Type alpha = 1.0e-006,
41.          sigma = 1.0e-009;
42.
43.     cout << setiosflags(ios::fixed) << setprecision(4);
44.     cout << "The original matrix A : " << A << endl;
45.     cout << "The constant vector b : " << b << endl;
46.     cout << "The solution of Truncated SVD (is consistent with Matlab) :"
47.          << endl << tsvd( A, b ) << endl;
48.     cout << "The solution of Damped SVD with alpha = "
49.          << sigma << " :" << endl << dsvd( A, b, sigma ) << endl;
50.     cout << "The solution of Tikhonov Regularization with alpha = "
51.          << alpha << " :" << endl << tikhonov( A, b, alpha ) << endl;
52.
53.     Matrix<complex<Type> > cA = complexMatrix( A, -A );
54.     Vector<complex<Type> > cb = complexVector( b );
55.
56.     cout << setiosflags(ios::fixed) << setprecision(4);
57.     cout << "The original complex matrix cA is: A - jA" << endl << endl;
58.     cout << "The constant complex vector cb is: b" << endl << endl;
59.     cout << "The solution of Truncated SVD (is consistent with Matlab) :"
60.          << endl << tsvd( cA, cb ) << endl;
61.     cout << "The solution of Damped SVD with alpha = "
62.          << sigma << " :" << endl << dsvd( cA, cb, sigma ) << endl;
63.     cout << "The solution of Tikhonov Regularization with alpha = "
64.          << alpha << " :" << endl << tikhonov( cA, cb, alpha ) << endl;
65.
66.     return 0;
67. }
```

运行结果：

```
The original matrix A : size: 8 by 6
64.0000 2.0000  3.0000  61.0000 60.0000 6.0000
9.0000  55.0000 54.0000 12.0000 13.0000 51.0000
17.0000 47.0000 46.0000 20.0000 21.0000 43.0000
40.0000 26.0000 27.0000 37.0000 36.0000 30.0000
32.0000 34.0000 35.0000 29.0000 28.0000 38.0000
41.0000 23.0000 22.0000 44.0000 45.0000 19.0000
49.0000 15.0000 14.0000 52.0000 53.0000 11.0000
8.0000  58.0000 59.0000 5.0000  4.0000  62.0000

The constant vector b : size: 8 by 1
260.0000
260.0000
260.0000
260.0000
260.0000
260.0000
260.0000
260.0000

The solution of Truncated SVD (is consistent with Matlab) :
size: 6 by 1
1.1538
1.4615
1.3846
1.3846
1.4615
1.1538

The solution of Damped SVD with alpha = 0.0000 :
size: 6 by 1
1.1539
1.4615
1.3847
1.3847
1.4615
1.1538

The solution of Tikhonov Regularization with alpha = 0.0000 :
size: 6 by 1
1.1538
1.4615
1.3846
1.3846
```

```
1.4615
1.1538

The original complex matrix cA is: A - jA

The constant complex vector cb is: b

The solution of Truncated SVD (is consistent with Matlab) :
size: 6 by 1
(0.5769,0.5769)
(0.7308,0.7308)
(0.6923,0.6923)
(0.6923,0.6923)
(0.7308,0.7308)
(0.5769,0.5769)

The solution of Damped SVD with alpha = 0.0000 :
size: 6 by 1
(0.5769,0.5769)
(0.7308,0.7308)
(0.6923,0.6923)
(0.6923,0.6923)
(0.7308,0.7308)
(0.5769,0.5769)

The solution of Tikhonov Regularization with alpha = 0.0000 :
size: 6 by 1
(0.5769,0.5769)
(0.7308,0.7308)
(0.6923,0.6923)
(0.6923,0.6923)
(0.7308,0.7308)
(0.5769,0.5769)


Process returned 0 (0x0)   execution time : 0.080 s
Press any key to continue.
```

4 非线性方程与方程组

4.1 非线性方程求根

工程应用中经常要计算非线性方程的根，一般情况下要得到解析解是很困难的，因此数值解就显得尤为重要。SP++中实现的 3 种常用的非线性方程求根方法，即二分法，Newton法和割线法，详见表 4-1。其中参数“f”为函数对象，代表非线性方程，在文件“nlfunc.h”中定义。

表 4-1 非线性方程求根

Operation	Effect
bisection (f,a,b,tol)	二分法非线性方程求根
newton (f,x0,tol,maxItr)	Newton 法非线性方程求根
secant (f,x1,x2,tol,maxItr)	割线法非线性方程求根

测试代码：

```
/*****************************************************************************
 *                                nle_test.cpp
 *
 * Rooting of nonlinear equation testing.
 *
 * Zhang Ming, 2010-04, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <iomanip>
#include <nleroot.h>


using namespace std;
using namespace splab;


typedef double  Type;

```

```
int main()
{
    Type x01, x02, x03;
    NLFunc<Type> f( 1.0, -3.0, 1.0 );

    cout << setiosflags(ios::fixed) << setprecision(8);
    x01 = bisection( f, 0.0, 2.0, 1.0e-6 );
    cout << "Bisection method :   " << x01 << endl <<endl;

    x02 = newton( f, 0.0, 1.0e-6, 100 );
    cout << "Newton method :      " << x02 << endl <<endl;

    x03 = secant( f, 1.0, 3.0, 1.0e-6, 100 );
    cout << "Secant method :      " << x03 << endl <<endl;

    return 0;
}
```

运行结果：

```
Bisection method :   0.38196611

Newton method :      0.38196601

Secant method :      2.61803406


Process returned 0 (0x0)   execution time : 0.061 s
Press any key to continue.
```

4.2 非线性方程组求根

非线性方程组的求解要比非线性方程的求解更为复杂，一般通过迭代法求解，比如Seidel迭代法和Newton迭代法，详见表 4-2。其中参数“G”和“F”为函数对象，代表非线性方程组，在文件“nlfuncs.h”中定义。

表 4-2 非线性方程组求根

Operation	Effect
seidel (G,X0 ,tol,maxItr)	Seidel 迭代法非线性方程组求根
newton (F,x0,tol,eps,maxItr)	Newton 迭代法非线性方程组求根

测试文件：

```
/*****************************************************************************
 *                                nle_test.cpp
 *
 * Rooting of nonlinear equations testing.
 *
 * Zhang Ming, 2010-10, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <iomanip>
#include <nleroots.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     N = 2;


int main()
{
    Vector<Type> X0(N);
    cout << setiosflags(ios::fixed) << setprecision(8);

    NLEqus<Type> G;
    X0(1) = 0;  X0(2) = 1;
    cout << "Seidel iteration method :   " << seidel( G, X0 ) << endl;

    NLFuncs<Type> F;
    X0(1) = 2;  X0(2) = 0;
    cout << "Newton iteration method :   " << newton( F, X0 ) << endl;

    return 0;
}
```

运行结果：

```
Seidel iteration method :   size: 2 by 1
-0.22221445
0.99380842

Newton iteration method :   size: 2 by 1
```

```
1.90067673
0.31121857


Process returned 0 (0x0)   execution time : 0.042 s
Press any key to continue.
```

4.3 Romberg 数值积分

SP++中提供了Romberg数值积分算法，如表 4-3所示。其中参数“f”为函数对象，代表被积函数，在文件“integrand”中定义。

表 4-3 Romberg 数值积分

Operation	Effect
romberg(f,a,b,tol)	Romberg 数值积分算法

测试代码：

```
/*****************************************************************************
 *                                integral_test.cpp
 *
 * Numerical integral testing.
 *
 * Zhang Ming, 2010-04, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <integral.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
    Type    p1 = 1,
            p2 = 2,
```

```
            lower = 0,
            upper = PI,
            I = 0;
    Func<Type> f( p1, p2 );

    I = romberg( f, lower, upper );
    cout << "The integral of function 'f' from " << lower <<
            " to " << upper << " is :   " << I << endl << endl;

    return 0;
}
```

运行结果：

```
The integral of function 'f' from 0 to 3.14159 is :   7.14159


Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

5 插值与拟合

5.1 Newton 插值

对于给定的 N+1 个点，可以通过 Newtow 插值法求取经过这些点的 N 次多项式。但该插值法主要针对底阶的插值多项式，因为高阶多项式一般会出现剧烈振荡，在插值点之间会引入非常大的误差。

SP++中Newton插值法类NewtonInterp<Type>的使用方法见表 5-1，并且与三次样条和最小二乘拟合共同继承了一般的插值类模板Interpolation<Type>，有关该类的声明可参见“interpolation.h”。

表 5-1 Newton 插值法

Operation	Effect
NewtonInterp<Type> intp(xi,yi)	创建 Newton 插值类
intp.~ NewtonInterp<Type>	析构 Newton 插值类
intp.calcCoefs()	计算插值多项式的系数
intp.evaluate(x)	计算给定坐标的函数值
intp.getCoefs()	获取插值多项式的系数

测试代码：

```
/*****************************************************************************
 *                                newtoninterp_test.cpp
 *
 * Newton interpolation testing.
 *
 * Zhang Ming, 2010-04, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <newtoninterp.h>


using namespace std;
```

```
18. using namespace splab;
19. 
20. 
21. typedef double  Type;
22. 
23. 
24. int main()
25. {
26.     Vector<Type> x(5),
27.                  y(5);
28.     x[0] = 0; x[1] = 30;  x[2] = 45;        x[3] = 60;        x[4] = 90;
29.     y[0] = 0; y[1] = 0.5; y[2] = sqrt(2.0)/2; y[3] = sqrt(3.0)/2; y[4] = 1;
30. 
31.     NewtonInterp<Type> poly(x,y);
32.     poly.calcCoefs();
33. 
34.     cout << setiosflags(ios::fixed) << setprecision(4);
35.     cout << "Coefficients of Newton interpolated polynomial:"
36.          << poly.getCoefs() << endl;
37. 
38.     cout << "The true and interpolated values:" << endl;
39.     cout << sin(10*D2R) << "   " << poly.evaluate(10) << endl
40.          << sin(20*D2R) << "   " << poly.evaluate(20) << endl
41.          << sin(50*D2R) << "   " << poly.evaluate(50) << endl << endl;
42. 
43.     return 0;
44. }
```

运行结果：

```
1.  Coefficients of Newton interpolated polynomial:size: 5 by 1
2.  0.0000
3.  0.0167
4.  -0.0001
5.  -0.0000
6.  0.0000
7.  
8.  The true and interpolated values:
9.  0.1736   0.1734
10. 0.3420   0.3419
11. 0.7660   0.7660
12. 
13. 
14. Process returned 0 (0x0)   execution time : 0.020 s
15. Press any key to continue.
```

5.2 三次样条插值

三次样条插值法对每个插值区间用三次多项式进行近似，并且保证区间端点插值多项式与其一阶导数均连续。因此所得的插值多项式有很好的性质，并且对于多点插值有很高的逼近程度。三次样条的使用方法与Newton相同，见表 5-2。

表 5-2 三次样条插值法

Operation	Effect
Spline3Interp<Type> intp(xi,yi,d2l,dwr)	创建三次样条插值类
intp.~ Spline3Interp<Type>	析构三次样条插值类
intp.calcCoefs()	计算插值多项式的系数
intp.evaluate(x)	计算给定坐标的函数值
intp.getCoefs()	获取插值多项式的系数

测试代码：

```
1.  /*****************************************************************************
2.   *                                spline3interp_test.cpp
3.   *
4.   * Spline3 interpolation testing.
5.   *
6.   * Zhang Ming, 2010-04, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <spline3interp.h>
15.
16.
17. using namespace std;
18. using namespace splab;
19.
20.
21. typedef double   Type;
22.
23.
24. int main()
25. {
26.     // f(x) = 1 / (1+25*x^2)   -1 <= x <= 1
```

```
    Type xi[11] = { -1.0, -0.8, -0.6, -0.4, -0.2,
                    0.0, 0.2, 0.4, 0.6, 0.8, 1.0 };
    Type yi[11] = { 0.0385, 0.0588, 0.1, 0.2, 0.5,
                    1.0, 0.5, 0.2, 0.1, 0.0588, 0.0385 };
    Type Ml = 0.2105, Mr = 0.2105;
    Vector<Type> x( 11, xi ), y( 11, yi );

    Spline3Interp<Type> poly( x, y, Ml, Mr );
    poly.calcCoefs();

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "Coefficients of cubic spline interpolated polynomial:   "
         << poly.getCoefs() << endl;

    cout << "The true and interpolated values:" << endl;
    cout << "0.0755" << "   " << poly.evaluate(-0.7) << endl
         << "0.3077" << "   " << poly.evaluate(0.3) << endl
         << "0.0471" << "   " << poly.evaluate(0.9) << endl << endl;

    return 0;
}
```

运行结果：

```
Coefficients of cubic spline interpolated polynomial:   size: 10 by 4
0.0385  0.0737  0.1052  0.1694
0.0588  0.1291  0.2069  0.8886
0.1000  0.3185  0.7400  0.8384
0.2000  0.7151  1.2431  13.4078
0.5000  2.8212  9.2877  -54.4695
1.0000  -0.0000 -23.3940        54.4704
0.5000  -2.8212 9.2882  -13.4120
0.2000  -0.7153 1.2410  -0.8225
0.1000  -0.3176 0.7476  -0.9482
0.0588  -0.1323 0.1787  -0.1224

The true and interpolated values:
0.0755   0.0747
0.3077   0.2974
0.0471   0.0472


Process returned 0 (0x0)   execution time : 0.057 s
Press any key to continue.
```

5.3 最小二乘拟合

给定观测值求其所遵从的函数关系时需要进行曲线拟合，最常用的一类拟合方法就是最小二乘拟合，该方法以观测数据与待拟函数之间的均方误差为准则求取拟合参数。实际中比较常用的是线性最小二乘拟合（许多非线性最小二乘拟合可以转化为线性最小二乘拟合），即待拟合的函数是已知函数簇的线性组合。

SP++中提供了线性最小二乘拟合类LSFitting<Type>，其中构造函数中“f”是已知的函数簇，其定义见“fitcurves.h”， 具体使用方法见表 5-3。

表 5-3 最小二乘拟合

Operation	Effect
LSFitting<Type> lsf(xi,yi,f)	创建最小二乘拟合类
lsf.~ LSFitting<Type>	析构最小二乘拟合类
lsf.calcCoefs()	计算拟合函数的系数
lsf.evaluate(x)	计算给定坐标的函数值
lsf.getCoefs()	获取拟合函数的系数

测试代码：

```
/*****************************************************************************
 *                               lsfit_test.cpp
 *
 * Least square fitting testing.
 *
 * Zhang Ming, 2010-04, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <lsfitting.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
```

```
{
    Type    A = 4.0,
            alpha = 1.0,
            beta = -2.0,
            gamma = -4.0,
            tmp = 0.0;

    int M = 100;
    Vector<Type> x = linspace( 0.01, 0.5*PI, M );
    Vector<Type> y(M);
    for( int i=0; i<M; ++i )
    {
        tmp = A * pow(x[i],alpha) * exp(beta*x[i]+gamma*x[i]*x[i]);
        y[i] = log(max(tmp,EPS));
    }

    Funcs<Type> phi;
    LSFitting<Type> lsf( x, y, phi );
    lsf.calcCoefs();
    Vector<Type> parms = lsf.getCoefs();
    parms(1) = exp(parms(1));

    cout << "The original parameters are:" << endl
         << A << endl << alpha << endl << beta << endl << gamma << endl << endl;
    cout << "The fitted parameters are:   "  << parms << endl;

    return 0;
}
```

运行结果：

```
The original parameters are:
4
1
-2
-4

The fitted parameters are:   size: 4 by 1
4
1
-2
-4


Process returned 0 (0x0)   execution time : 0.039 s
```

```
15.  Press any key to continue.
```

6 优化算法

6.1 一维线搜索

所有的基于下降方向的优化方法都需要通过一维搜索来确定步长，而下降步长直接影响到优化方法的收敛速率，因此一维搜索是优化算法中一个十分关键的步骤。SP++中提供了非精确一维搜索算法类LineSearch<DType,Ftype>，同时可以获得确定步长时目标函数的计算次数。具体函数见表 6-1。其中“func”是目标函数，定义参见“objfunc.h”头文件。

表 6-1 一维线搜索

Operation	Effect
LineSearch<DType,Ftype> ols	创建一维搜索类
ols.~ LineSearch <DType,Ftype>()	析构一维搜索类
ols. getStep (func, x0, dk, maxItr)	求取一维搜索步长
ols. getFuncNum ()	获取目标函数的计算次数
ols. isSuccess ()	判断一维搜索是否成功

6.2 最速下降法

最速下降法（即梯度法）是一种最古老和最简单的优化算法，其优点是稳定性比较高，缺点是收敛速率非常慢。因此对收敛速率要求不是很高的问题和高维稳定性比较差的问题，该方法仍然是一种非常实用的算法。

SP++中最速下降类SteepDesc<DType,Ftype>提供了求最优值、最小函数值、梯度模值和目标函数计算次数等函数，详见表 6-2。

表 6-2 最速下降法

Operation	Effect
SteepDesc<DType,Ftype> fmin	创建最速下降法类
fmin.~SteepDesc<DType,Ftype>()	析构最速下降法类
fmin. optimize(func, x0, tol, maxItr)	求指定参数函数的最小值
fmin.getOptValue()	获取自变量的最优值
fmin.getGradNorm()	获取迭代过程中梯度向量的模值
fmin.getFuncMin()	获取函数的最小值
fmin.getItrNum()	获取迭代次数

测试代码：

```
/*****************************************************************************
 *                                steepdesc_test.cpp
 *
 * Steepest descent method testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <objfunc.h>
#include <steepdesc.h>


using namespace std;
using namespace splab;


typedef float  Type;


int main()
{
    Type a = 1.0,
         b = -1.0,
         c = -1.0;
    ObjFunc<Type> f( a, b, c );
    Vector<Type> x0(2);
    x0(1) = Type(0.0);
    x0(2) = Type(0.0);

    Type tolErr = 1.0e-3;
    SteepDesc< Type, ObjFunc<Type> > steep;
    steep.optimize( f, x0, tolErr );
    if( steep.isSuccess() )
    {
        Vector<Type> xmin = steep.getOptValue();
        int N = steep.getItrNum();
        cout << "The iterative number is:   " << N << endl << endl;
        cout << "The number of function calculation is:   "
```

```
44.              << steep.getFuncNum() << endl << endl;
45.         cout << setiosflags(ios::fixed) << setprecision(4);
46.         cout << "The optimal value of x is:   " << xmin << endl;
47.         cout << "The minimum value of f(x) is:   " << f(xmin) << endl << endl;
48.         cout << "The gradient's norm at x is:   "
49.              << steep.getGradNorm()[N] << endl << endl;
50.     }
51.     else
52.         cout << "The optimal solution  can't be found!" << endl;
53.
54.     return 0;
55. }
```

运行结果：

```
1.  The iterative number is:   14
2.
3.  The number of function calculation is:   43
4.
5.  The optimal value of x is:   size: 2 by 1
6.  -0.7070
7.  0.0000
8.
9.  The minimum value of f(x) is:   -0.4289
10.
11. The gradient's norm at x is:   0.0006
12.
13.
14. Process returned 0 (0x0)   execution time : 0.039 s
15. Press any key to continue.
```

6.3 共轭梯度法

共轭梯度法通过生成共轭方向来确定每次迭代的搜索方向，具有二次终止性和较高的收敛速率，并且所需的存储空间也很少，因此对于一些高维的优化问题非常适用。但当目标函数不是二次函数（实际应用中目标函数往往不是二次的）时，不能通过 n 步迭代求得最优值，所以需要再开始技术重新确定下降方向。并且当目标函数不能用二次函数很好的逼近时，其收敛速率相对最速下降法并不具有明显的优势。

SP++中共轭梯度类ConjGrad <DType,Ftype>所提供的函数与最速下降法类相似，详见表 6-3，其中默认的再开始次数为目标函数的维数n。

表 6-3　共轭梯度法

Operation	Effect
ConjGrad <DType,Ftype> fmin	创建共轭梯度法类
fmin.~ ConjGrad <DType,Ftype>()	析构共轭梯度法类
fmin. optimize(func, x0, tol, maxItr)	求指定参数函数的最小值
fmin.getOptValue()	获取自变量的最优值
fmin.getGradNorm()	获取迭代过程中梯度向量的模值
fmin.getFuncMin()	获取函数的最小值
fmin.getItrNum()	获取迭代次数

测试代码：

```
/*****************************************************************************
 *                                conjgrad_test.cpp
 *
 * Conjugate gradient optimal method testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <objfunc.h>
#include <conjgrad.h>


using namespace std;
using namespace splab;


typedef double  Type;


int main()
{
    Type a = 1.0,
         b = -1.0,
         c = -1.0;
    ObjFunc<Type> f( a, b, c );
    Vector<Type> x0(2);
    x0(1) = 0.5;
```

```
33.     x0(2) = 0.5;
34.
35.     Type tolErr = 1.0e-6;
36.     ConjGrad< Type, ObjFunc<Type> > prp;
37.     prp.optimize( f, x0, x0.dim(), tolErr );
38.     if( prp.isSuccess() )
39.     {
40.         Vector<Type> xmin = prp.getOptValue();
41.         int N = prp.getItrNum();
42.         cout << "The iterative number is:   " << N << endl << endl;
43.         cout << "The number of function calculation is:   "
44.              << prp.getFuncNum() << endl << endl;
45.         cout << setiosflags(ios::fixed) << setprecision(4);
46.         cout << "The optimal value of x is:   " << xmin << endl;
47.         cout << "The minimum value of f(x) is:   " << f(xmin) << endl << endl;
48.         cout << "The gradient's norm at x is:   "
49.              << prp.getGradNorm()[N] << endl << endl;
50.     }
51.     else
52.         cout << "The optimal solution  cann't be found!" << endl;
53.
54.     return 0;
55. }
```

运行结果：

```
1.  The iterative number is:   29
2.
3.  The number of function calculation is:   207
4.
5.  The optimal value of x is:   size: 2 by 1
6.  -0.7071
7.  -0.0000
8.
9.  The minimum value of f(x) is:   -0.4289
10.
11. The gradient's norm at x is:   0.0000
12.
13.
14. Process returned 0 (0x0)   execution time : 0.036 s
15. Press any key to continue.
```

6.4 拟 Newton 法

拟 Newton 法通过梯度向量和校正公式来近似目标函数的 Hess 矩阵，同样具有二次终止性，并且相对最速下降法和共轭梯度法有很高的收敛速率。比较常用的两类拟 Newton 算法是 DFP 和 FBGS，是目前公认的求解非线性无约束优化问题的最好算法。但对于大规模的优化问题，拟 Newton 需要较大的存储空间，并且矩阵运算所耗费时间的弊端也突显出来。

SP++中提供了BFGS拟Newton算法，其调用格式与最速下降法以及共轭梯度法相同，详见表 6-4。

表 6-4 拟 Newton 法

Operation	Effect
BFGS <DType,Ftype> fmin	创建拟 Newton 法类
fmin.~ BFGS <DType,Ftype>()	析构拟 Newton 法类
fmin. optimize(func, x0, tol, maxItr)	求指定参数函数的最小值
fmin.getOptValue()	获取自变量的最优值
fmin.getGradNorm()	获取迭代过程中梯度向量的模值
fmin.getFuncMin()	获取函数的最小值
fmin.getItrNum()	获取迭代次数

测试代码：

```
1.  /*****************************************************************************
2.   *                                bfgs_test.cpp
3.   *
4.   * BFGS method testing.
5.   *
6.   * Zhang Ming, 2010-03, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <objfunc.h>
15. #include <bfgs.h>
16.
17.
18. using namespace std;
19. using namespace splab;
20.
21.
22. typedef double  Type;
```

```
23.
24.
25. int main()
26. {
27.     Type a = 1.0,
28.          b = -1.0,
29.          c = -1.0;
30.     ObjFunc<Type> f( a, b, c );
31.     Vector<Type> x0(2);
32.     x0(1) = Type(0.0);
33.     x0(2) = Type(0.0);
34.
35.     BFGS< Type, ObjFunc<Type> > bfgs;
36.     bfgs.optimize( f, x0 );
37.     if( bfgs.isSuccess() )
38.     {
39.         Vector<Type> xmin = bfgs.getOptValue();
40.         int N = bfgs.getItrNum();
41.         cout << "The iterative number is:   " << N << endl << endl;
42.         cout << "The number of function calculation is:   "
43.              << bfgs.getFuncNum() << endl << endl;
44.         cout << setiosflags(ios::fixed) << setprecision(4);
45.         cout << "The optimal value of x is:   " << xmin << endl;
46.         cout << "The minimum value of f(x) is:   " << f(xmin) << endl << endl;
47.         cout << "The gradient's norm at x is:   "
48.              << bfgs.getGradNorm()[N] << endl << endl;
49.     }
50.     else
51.         cout << "The optimal solution  cann't be found!" << endl;
52.
53.     return 0;
54. }
```

运行结果：

```
1.  The iterative number is:   7
2.
3.  The number of function calculation is:   16
4.
5.  The optimal value of x is:   size: 2 by 1
6.  -0.7071
7.  0.0000
8.
9.  The minimum value of f(x) is:   -0.4289
10.
```

```
The gradient's norm at x is:   0.0000


Process returned 0 (0x0)   execution time : 0.007 s
Press any key to continue.
```

7 Fourier 分析

7.1 2 的整次幂 FFT 算法

SP++中实现了长度为 2 的整次幂的FFT算法，FFTMR<Type>类，具体采用了基 8、基 4 和基 2 的混合基算法，并采用经济存储模式，见表 7-1。长度为 2 的整次幂信号的FFT计算效率非常高，故实际中很多应用都采取了该方式。

表 7-1 长度为 2 的幂次的 FFT 算法

Operation	Effect
FFTMR <Type> ft	创建 FFTMR 类
ft.~ FFTMR <Type>()	析构 FFTMR 类
ft.fft(cxn)	复信号的 Fourier 变换
ft.fft(rxn,Xk)	实信号的 Fourier 变换
ft.ifft(cXk)	复信号的逆 Fourier 变换
ft.ifft(Xk,rxn)	实信号的逆 Fourier 变换

测试代码：

```
/*****************************************************************************
 *                                fftmr_test.cpp
 *
 * Mixed Radix Algorithm FFT testing.
 *
 * Zhang Ming, 2010-04, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <iomanip>
#include <fftmr.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     LENGTH = 32;
```

```
21.
22.
23. int main()
24. {
25.     int     i, j, index, rows = LENGTH/4;
26.
27.     Vector<Type> xn(LENGTH);
28.     Vector< complex<Type> > yn(LENGTH),
29.                             Xk(LENGTH);
30.     FFTMR<Type> Fourier;
31.
32.     cout << "The original signal is: " << endl;
33.     for( i=0; i<rows; i++ )
34.     {
35.         cout << endl;
36.         for( j=0; j<3; j++ )
37.         {
38.             index = 3*i+j;
39.             xn[index] = i+j;
40.             cout << setiosflags(ios::fixed) << setprecision(6);
41.             cout << "\t" << xn[index];
42.         }
43.     }
44.     cout << endl << endl;
45.
46.     Fourier.fft( xn, Xk );
47.
48.     cout << "The Fourier transform of original signal is:" << endl;
49.     for( i=0; i<rows; i++ )
50.     {
51.         cout << endl;
52.         for( j=0; j<3; j++ )
53.         {
54.             index = 3*i+j;
55.             cout << setiosflags(ios::fixed) << setprecision(6);
56.             cout << "\t" << Xk[index];
57.         }
58.     }
59.     cout << endl << endl;
60.
61.     Fourier.ifft( Xk, xn );
62.     cout << "The inverse Fourier transform is" << endl;
63.     for( i=0; i<rows; i++ )
64.     {
```

```
65.         cout << endl;
66.         for( j=0; j<3; j++ )
67.         {
68.             index = 3*i+j;
69.             cout << setiosflags(ios::fixed) << setprecision(6);
70.             cout << "\t" << xn[index];
71.         }
72.     }
73.     cout << endl << endl;
74.
75.     cout << "The original signal is: " << endl;
76.     for( i=0; i<rows; i++ )
77.     {
78.         cout << endl;
79.         for( j=0; j<3; j++ )
80.         {
81.             index = 3*i+j;
82.             yn[index] = complex<double>(i,j);
83.             cout << setiosflags(ios::fixed) << setprecision(6);
84.             cout << "\t" << yn[index];
85.         }
86.     }
87.     cout << endl << endl;
88.
89.     Fourier.fft( yn );
90.     cout << "The Fourier transform of original signal is:" << endl;
91.     for( i=0; i<rows; i++ )
92.     {
93.         cout << endl;
94.         for( j=0; j<3; j++ )
95.         {
96.             index = 3*i+j;
97.             cout << setiosflags(ios::fixed) << setprecision(6);
98.             cout << "\t" << yn[index];
99.         }
100.    }
101.    cout << endl << endl;
102.
103.    Fourier.ifft( yn );
104.    cout << "The inverse Fourier transform is" << endl;
105.    for( i=0; i<rows; i++ )
106.    {
107.        cout << endl;
108.        for( j=0; j<3; j++ )
```

```
        {
            index = 3*i+j;
            cout << setiosflags(ios::fixed) << setprecision(6);
            cout << "\t" << yn[index];
        }
    }

    cout << endl << endl;
    return 0;
}
```

运行结果：

```
The original signal is:

        0.000000        1.000000        2.000000
        1.000000        2.000000        3.000000
        2.000000        3.000000        4.000000
        3.000000        4.000000        5.000000
        4.000000        5.000000        6.000000
        5.000000        6.000000        7.000000
        6.000000        7.000000        8.000000
        7.000000        8.000000        9.000000

The Fourier transform of original signal is:

        (108.000000,0.000000)   (-52.827446,1.358153)   (-0.613126,-23.640092)
        (13.310056,1.647944)    (-4.000000,9.656854)    (-8.902567,-4.353284)
        (3.082392,-7.681941)    (5.303682,2.676653)     (-4.000000,4.000000)
        (-5.314558,-4.659704)   (0.917608,-8.368233)    (0.220781,11.701395)
        (-4.000000,1.656854)    (-0.343999,-3.954231)   (4.613126,-0.326383)
        (0.554051,4.364941)     (-4.000000,-0.000000)   (0.554051,-4.364941)
        (4.613126,0.326383)     (-0.343999,3.954231)    (-4.000000,-1.656854)
        (0.220781,-11.701395)   (0.917608,8.368233)     (-5.314558,4.659704)

The inverse Fourier transform is

        0.000000        1.000000        2.000000
        1.000000        2.000000        3.000000
        2.000000        3.000000        4.000000
        3.000000        4.000000        5.000000
        4.000000        5.000000        6.000000
        5.000000        6.000000        7.000000
        6.000000        7.000000        8.000000
        7.000000        8.000000        9.000000
```

```
33.
34. The original signal is:
35.
36.         (0.000000,0.000000)     (0.000000,1.000000)     (0.000000,2.000000)
37.         (1.000000,0.000000)     (1.000000,1.000000)     (1.000000,2.000000)
38.         (2.000000,0.000000)     (2.000000,1.000000)     (2.000000,2.000000)
39.         (3.000000,0.000000)     (3.000000,1.000000)     (3.000000,2.000000)
40.         (4.000000,0.000000)     (4.000000,1.000000)     (4.000000,2.000000)
41.         (5.000000,0.000000)     (5.000000,1.000000)     (5.000000,2.000000)
42.         (6.000000,0.000000)     (6.000000,1.000000)     (6.000000,2.000000)
43.         (7.000000,0.000000)     (7.000000,1.000000)     (7.000000,2.000000)
44.
45. The Fourier transform of original signal is:
46.
47.         (84.000000,24.000000)   (-42.542528,1.020494)   (5.034128,-18.695144)
48.         (13.685581,5.361763)    (-4.000000,9.656854)    (-6.244002,-4.826961)
49.         (6.192124,-5.705118)    (6.023013,5.306175)     (-4.000000,4.000000)
50.         (-1.277237,-6.299534)   (13.722858,-4.086928)   (-8.776842,-2.726593)
51.         (-4.000000,1.656854)    (-1.716734,-2.445797)   (2.729646,0.149298)
52.         (-0.485914,4.057658)    (-4.000000,-0.000000)   (0.861334,-3.324976)
53.         (4.137445,2.209863)     (-1.852433,5.326967)    (-4.000000,-1.656854)
54.         (14.648770,-2.703773)   (-3.363697,-4.437017)   (-3.674728,0.622383)
55.
56. The inverse Fourier transform is
57.
58.         (-0.000000,0.000000)    (0.000000,1.000000)     (0.000000,2.000000)
59.         (1.000000,-0.000000)    (1.000000,1.000000)     (1.000000,2.000000)
60.         (2.000000,0.000000)     (2.000000,1.000000)     (2.000000,2.000000)
61.         (3.000000,-0.000000)    (3.000000,1.000000)     (3.000000,2.000000)
62.         (4.000000,-0.000000)    (4.000000,1.000000)     (4.000000,2.000000)
63.         (5.000000,-0.000000)    (5.000000,1.000000)     (5.000000,2.000000)
64.         (6.000000,-0.000000)    (6.000000,1.000000)     (6.000000,2.000000)
65.         (7.000000,0.000000)     (7.000000,1.000000)     (7.000000,2.000000)
66.
67.
68. Process returned 0 (0x0)   execution time : 0.092 s
69. Press any key to continue.
```

7.2 任意长度 FFT 算法

SP++中实现了任意长度的FFT算法，FFTPF<Type>类，该算法采用素数因子分解法计算任意长度的DFT，具体函数见表 7-2。如果需要大量计算FFT，建议调用

FFTW，以提高计算效率。

表 7-2　任意长度的 FFT 算法

Operation	Effect
FFTPF <Type> ft	创建 FFTPF 类
ft.~ FFTPF<Type>()	析构 FFTPF 类
ft.fft(rxn,Xk)	实信号的 Fourier 变换
ft.ifft(Xk,rxn)	实信号的逆 Fourier 变换
ft. fft(cxn,Xk)	复信号的 Fourier 变换
ft.ifft(Xk,cxn)	复信号的逆 Fourier 变换

测试代码：

```
/*****************************************************************************
 *                                fftpf_test.cpp
 *
 * Prime factor FFT test.
 *
 * Zhang Ming, 2010-09, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <cstdlib>
#include <vectormath.h>
#include <fftpf.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     MINLEN = 101;
const   int     MAXLEN = 1000;
const   int     STEP   = 10;


int main()
{
    Vector< complex<Type> >  sn, Rk, Sk, xn;
    Vector<Type> rn, tn;
    FFTPF<Type> Fourier;

```

```
32.     cout << "forward transform: Sk = fft(sn)  ( complex to complex )." << endl;
33.     cout << "inverse transform: xn = ifft(Sk) ( complex to complex )." << endl
34.          << endl;
35.     for( int len=MINLEN; len<MAXLEN; len+=STEP )
36.     {
37.         sn.resize(len);
38.         Sk.resize(len);
39.         xn.resize(len);
40.         for( int i=0; i<len; ++i )
41.             sn[i] = complex<Type>( rand()%10, rand()%10 );
42.
43.         Fourier.fft( sn, Sk );
44.         Fourier.ifft( Sk, xn );
45.         cout << "N = " << len << "\t\t" << "mean(abs((sn-xn)) = "
46.              << sum(abs(sn-xn))/len << endl;
47.     }
48.     cout << endl << endl;
49.
50.     cout << "forward transform: Rk = fft(rn)  ( real to complex )." << endl;
51.     cout << "inverse transform: tn = ifft(Rk) ( complex to real )." << endl
52.          << endl;
53.     for( int len=MINLEN; len<MAXLEN; len+=STEP )
54.     {
55.         rn.resize(len);
56.         Rk.resize(len);
57.         tn.resize(len);
58.         for( int i=0; i<len; ++i )
59.             rn[i] = rand()%10;
60.
61.         Fourier.fft( rn, Rk );
62.         Fourier.ifft( Rk, tn );
63.         cout << "N = " << len << "\t\t" << "mean(abs((rn-tn)) = "
64.              << sum(abs(sn-xn))/len << endl;
65.     }
66.     cout << endl;
67.
68.     return 0;
69. }
```

运行结果：

```
1.  forward transform: Sk = fft(sn)  ( complex to complex ).
2.  inverse transform: xn = ifft(Sk) ( complex to complex ).
3.
4.  N = 101         mean(abs((sn-xn)) = 9.56796e-015
```

```
  N = 111         mean(abs((sn-xn)) = 6.02908e-015
  N = 121         mean(abs((sn-xn)) = 3.68512e-015
  N = 131         mean(abs((sn-xn)) = 2.0032e-014
  N = 141         mean(abs((sn-xn)) = 8.53139e-015
  N = 151         mean(abs((sn-xn)) = 1.32487e-014
  N = 161         mean(abs((sn-xn)) = 6.01949e-015
  N = 171         mean(abs((sn-xn)) = 9.78613e-015
  N = 181         mean(abs((sn-xn)) = 4.90428e-015
  N = 191         mean(abs((sn-xn)) = 1.28095e-014
  N = 201         mean(abs((sn-xn)) = 7.66037e-015
  N = 211         mean(abs((sn-xn)) = 2.64879e-014
  N = 221         mean(abs((sn-xn)) = 1.23744e-014
  N = 231         mean(abs((sn-xn)) = 1.52028e-014
  N = 241         mean(abs((sn-xn)) = 1.48443e-014
  N = 251         mean(abs((sn-xn)) = 2.23183e-014
  N = 261         mean(abs((sn-xn)) = 1.19937e-014
  N = 271         mean(abs((sn-xn)) = 1.54789e-014
  N = 281         mean(abs((sn-xn)) = 6.37825e-014
  N = 291         mean(abs((sn-xn)) = 6.81017e-015
  N = 301         mean(abs((sn-xn)) = 3.67519e-014
  N = 311         mean(abs((sn-xn)) = 7.74232e-014
  N = 321         mean(abs((sn-xn)) = 2.06363e-014
  N = 331         mean(abs((sn-xn)) = 2.6944e-014
  N = 341         mean(abs((sn-xn)) = 2.32377e-014
  N = 351         mean(abs((sn-xn)) = 1.81853e-014
  N = 361         mean(abs((sn-xn)) = 2.70795e-014
  N = 371         mean(abs((sn-xn)) = 8.89281e-015
  N = 381         mean(abs((sn-xn)) = 1.1903e-014
  N = 391         mean(abs((sn-xn)) = 9.39787e-015
  N = 401         mean(abs((sn-xn)) = 6.26719e-014
  N = 411         mean(abs((sn-xn)) = 1.56459e-014
  N = 421         mean(abs((sn-xn)) = 5.49224e-014
  N = 431         mean(abs((sn-xn)) = 7.13155e-014
  N = 441         mean(abs((sn-xn)) = 7.9457e-015
  N = 451         mean(abs((sn-xn)) = 2.90133e-014
  N = 461         mean(abs((sn-xn)) = 5.84939e-014
  N = 471         mean(abs((sn-xn)) = 4.29465e-014
  N = 481         mean(abs((sn-xn)) = 4.52387e-014
  N = 491         mean(abs((sn-xn)) = 1.10306e-013
  N = 501         mean(abs((sn-xn)) = 3.37587e-014
  N = 511         mean(abs((sn-xn)) = 4.57228e-014
  N = 521         mean(abs((sn-xn)) = 1.11653e-013
  N = 531         mean(abs((sn-xn)) = 5.84668e-014
  N = 541         mean(abs((sn-xn)) = 1.27624e-013
```

```
N = 551         mean(abs((sn-xn)) = 3.54317e-014
N = 561         mean(abs((sn-xn)) = 5.29495e-014
N = 571         mean(abs((sn-xn)) = 3.74233e-014
N = 581         mean(abs((sn-xn)) = 3.8153e-014
N = 591         mean(abs((sn-xn)) = 3.24913e-014
N = 601         mean(abs((sn-xn)) = 1.32896e-013
N = 611         mean(abs((sn-xn)) = 2.92566e-014
N = 621         mean(abs((sn-xn)) = 4.35837e-014
N = 631         mean(abs((sn-xn)) = 1.35408e-013
N = 641         mean(abs((sn-xn)) = 1.35158e-013
N = 651         mean(abs((sn-xn)) = 1.38349e-014
N = 661         mean(abs((sn-xn)) = 1.30397e-013
N = 671         mean(abs((sn-xn)) = 2.18791e-014
N = 681         mean(abs((sn-xn)) = 7.84698e-014
N = 691         mean(abs((sn-xn)) = 1.38373e-013
N = 701         mean(abs((sn-xn)) = 1.66637e-013
N = 711         mean(abs((sn-xn)) = 2.22414e-014
N = 721         mean(abs((sn-xn)) = 3.17155e-014
N = 731         mean(abs((sn-xn)) = 4.88291e-014
N = 741         mean(abs((sn-xn)) = 6.79333e-014
N = 751         mean(abs((sn-xn)) = 6.76077e-014
N = 761         mean(abs((sn-xn)) = 1.4751e-013
N = 771         mean(abs((sn-xn)) = 3.765e-014
N = 781         mean(abs((sn-xn)) = 4.03454e-014
N = 791         mean(abs((sn-xn)) = 3.66948e-014
N = 801         mean(abs((sn-xn)) = 8.22744e-014
N = 811         mean(abs((sn-xn)) = 1.90965e-014
N = 821         mean(abs((sn-xn)) = 1.775e-013
N = 831         mean(abs((sn-xn)) = 1.52596e-014
N = 841         mean(abs((sn-xn)) = 9.06967e-014
N = 851         mean(abs((sn-xn)) = 8.87579e-014
N = 861         mean(abs((sn-xn)) = 6.5877e-015
N = 871         mean(abs((sn-xn)) = 2.90611e-014
N = 881         mean(abs((sn-xn)) = 1.11084e-013
N = 891         mean(abs((sn-xn)) = 2.36818e-014
N = 901         mean(abs((sn-xn)) = 8.32884e-014
N = 911         mean(abs((sn-xn)) = 1.32497e-013
N = 921         mean(abs((sn-xn)) = 5.2289e-014
N = 931         mean(abs((sn-xn)) = 6.6602e-014
N = 941         mean(abs((sn-xn)) = 4.79529e-014
N = 951         mean(abs((sn-xn)) = 5.12486e-014
N = 961         mean(abs((sn-xn)) = 1.3465e-013
N = 971         mean(abs((sn-xn)) = 1.80164e-013
N = 981         mean(abs((sn-xn)) = 8.43108e-014
```

```
 N = 991         mean(abs((sn-xn)) = 4.85968e-014

forward transform: Rk = fft(rn)  ( real to complex ).
inverse transform: tn = ifft(Rk) ( complex to real ).

 N = 101         mean(abs((rn-tn)) = 4.76826e-013
N = 111         mean(abs((rn-tn)) = 4.33869e-013
N = 121         mean(abs((rn-tn)) = 3.98012e-013
N = 131         mean(abs((rn-tn)) = 3.67629e-013
N = 141         mean(abs((rn-tn)) = 3.41556e-013
N = 151         mean(abs((rn-tn)) = 3.18937e-013
N = 161         mean(abs((rn-tn)) = 2.99127e-013
N = 171         mean(abs((rn-tn)) = 2.81634e-013
N = 181         mean(abs((rn-tn)) = 2.66074e-013
N = 191         mean(abs((rn-tn)) = 2.52144e-013
N = 201         mean(abs((rn-tn)) = 2.39599e-013
N = 211         mean(abs((rn-tn)) = 2.28244e-013
N = 221         mean(abs((rn-tn)) = 2.17916e-013
N = 231         mean(abs((rn-tn)) = 2.08482e-013
N = 241         mean(abs((rn-tn)) = 1.99832e-013
N = 251         mean(abs((rn-tn)) = 1.9187e-013
N = 261         mean(abs((rn-tn)) = 1.84519e-013
N = 271         mean(abs((rn-tn)) = 1.7771e-013
N = 281         mean(abs((rn-tn)) = 1.71386e-013
N = 291         mean(abs((rn-tn)) = 1.65496e-013
N = 301         mean(abs((rn-tn)) = 1.59998e-013
N = 311         mean(abs((rn-tn)) = 1.54853e-013
N = 321         mean(abs((rn-tn)) = 1.50029e-013
N = 331         mean(abs((rn-tn)) = 1.45497e-013
N = 341         mean(abs((rn-tn)) = 1.4123e-013
N = 351         mean(abs((rn-tn)) = 1.37206e-013
N = 361         mean(abs((rn-tn)) = 1.33406e-013
N = 371         mean(abs((rn-tn)) = 1.2981e-013
N = 381         mean(abs((rn-tn)) = 1.26403e-013
N = 391         mean(abs((rn-tn)) = 1.2317e-013
N = 401         mean(abs((rn-tn)) = 1.20098e-013
N = 411         mean(abs((rn-tn)) = 1.17176e-013
N = 421         mean(abs((rn-tn)) = 1.14393e-013
N = 431         mean(abs((rn-tn)) = 1.11739e-013
N = 441         mean(abs((rn-tn)) = 1.09205e-013
N = 451         mean(abs((rn-tn)) = 1.06784e-013
N = 461         mean(abs((rn-tn)) = 1.04467e-013
N = 471         mean(abs((rn-tn)) = 1.02249e-013
```

```
N = 481         mean(abs((rn-tn)) = 1.00124e-013
N = 491         mean(abs((rn-tn)) = 9.80844e-014
N = 501         mean(abs((rn-tn)) = 9.61266e-014
N = 511         mean(abs((rn-tn)) = 9.42455e-014
N = 521         mean(abs((rn-tn)) = 9.24365e-014
N = 531         mean(abs((rn-tn)) = 9.06957e-014
N = 541         mean(abs((rn-tn)) = 8.90193e-014
N = 551         mean(abs((rn-tn)) = 8.74037e-014
N = 561         mean(abs((rn-tn)) = 8.58457e-014
N = 571         mean(abs((rn-tn)) = 8.43423e-014
N = 581         mean(abs((rn-tn)) = 8.28906e-014
N = 591         mean(abs((rn-tn)) = 8.1488e-014
N = 601         mean(abs((rn-tn)) = 8.01322e-014
N = 611         mean(abs((rn-tn)) = 7.88207e-014
N = 621         mean(abs((rn-tn)) = 7.75514e-014
N = 631         mean(abs((rn-tn)) = 7.63224e-014
N = 641         mean(abs((rn-tn)) = 7.51317e-014
N = 651         mean(abs((rn-tn)) = 7.39776e-014
N = 661         mean(abs((rn-tn)) = 7.28584e-014
N = 671         mean(abs((rn-tn)) = 7.17726e-014
N = 681         mean(abs((rn-tn)) = 7.07187e-014
N = 691         mean(abs((rn-tn)) = 6.96953e-014
N = 701         mean(abs((rn-tn)) = 6.8701e-014
N = 711         mean(abs((rn-tn)) = 6.77348e-014
N = 721         mean(abs((rn-tn)) = 6.67953e-014
N = 731         mean(abs((rn-tn)) = 6.58816e-014
N = 741         mean(abs((rn-tn)) = 6.49925e-014
N = 751         mean(abs((rn-tn)) = 6.41271e-014
N = 761         mean(abs((rn-tn)) = 6.32844e-014
N = 771         mean(abs((rn-tn)) = 6.24636e-014
N = 781         mean(abs((rn-tn)) = 6.16638e-014
N = 791         mean(abs((rn-tn)) = 6.08842e-014
N = 801         mean(abs((rn-tn)) = 6.01241e-014
N = 811         mean(abs((rn-tn)) = 5.93828e-014
N = 821         mean(abs((rn-tn)) = 5.86595e-014
N = 831         mean(abs((rn-tn)) = 5.79536e-014
N = 841         mean(abs((rn-tn)) = 5.72645e-014
N = 851         mean(abs((rn-tn)) = 5.65916e-014
N = 861         mean(abs((rn-tn)) = 5.59343e-014
N = 871         mean(abs((rn-tn)) = 5.52921e-014
N = 881         mean(abs((rn-tn)) = 5.46645e-014
N = 891         mean(abs((rn-tn)) = 5.4051e-014
N = 901         mean(abs((rn-tn)) = 5.34511e-014
N = 911         mean(abs((rn-tn)) = 5.28644e-014
```

```
N = 921         mean(abs((rn-tn)) = 5.22904e-014
N = 931         mean(abs((rn-tn)) = 5.17287e-014
N = 941         mean(abs((rn-tn)) = 5.1179e-014
N = 951         mean(abs((rn-tn)) = 5.06408e-014
N = 961         mean(abs((rn-tn)) = 5.01139e-014
N = 971         mean(abs((rn-tn)) = 4.95978e-014
N = 981         mean(abs((rn-tn)) = 4.90922e-014
N = 991         mean(abs((rn-tn)) = 4.85968e-014


Process returned 0 (0x0)   execution time : 0.717 s
Press any key to continue.
```

7.3 普通信号 FFT 使用方法

为了方便使用，将 2 的整次幂与任意长度的FFT算法封装在了一起，当信号长度为 2 的整次幂时调用FFTMR类中的函数，提高计算效率；否则当信号长度不等于 2 的于整次幂时，调用FFTPF类的中函数。封装后的函数见表 7-3。

表 7-3 普通信号的 FFT 调用

Operation	Effect
fft(s)	计算实信号的离散 Fourier 变换
fft(cs)	计算复信号的离散 Fourier 变换
ifft (cS)	计算复信号的逆离散 Fourier 变换
fftr2c (s)	计算实信号的离散 Fourier 变换
fftc2c (cs)	计算复信号的离散 Fourier 变换
ifftc2r (cS)	计算复信号的实逆离散 Fourier 变换
ifftc2c (cS)	计算复信号的逆离散 Fourier 变换

测试代码：

```
/*************************************************************************
 *                             fft_test.cpp
 *
 * FFT test.
 *
 * Zhang Ming, 2010-09, Xi'an Jiaotong University.
 *************************************************************************/


#include <iostream>
#include <cstdlib>
```

```
12. #include <vectormath.h>
13. #include <fft.h>
14. 
15. 
16. using namespace std;
17. using namespace splab;
18. 
19. 
20. typedef double  Type;
21. const   int     MINLEN = 1;
22. const   int     MAXLEN = 1000;
23. const   int     STEP   = 10;
24. 
25. 
26. int main()
27. {
28.     Vector< complex<Type> >  sn, Rk, Sk, xn;
29.     Vector<Type> rn, tn;
30. 
31.     cout << "forward transform: complex to complex." << endl;
32.     cout << "inverse transform: complex to complex." << endl << endl;
33.     cout << "signal length" << "\t" << "mean(abs((sn-xn))" << endl;
34.     for( int len=MINLEN; len<MAXLEN; len+=STEP )
35.     {
36.         sn.resize(len);
37.         for( int i=0; i<len; ++i )
38.             sn[i] = complex<Type>( rand()%10, rand()%10 );
39. 
40.         Sk = fftc2c( sn );
41.         xn = ifftc2c( Sk );
42. //        Sk = fft( sn );
43. //        xn = ifft( Sk );
44.         cout << "    " << len << "\t\t" << "  " << sum(abs(sn-xn))/len << endl;
45.     }
46.     cout << endl << endl;
47. 
48.     cout << "forward transform: real to complex ." << endl;
49.     cout << "inverse transform: complex to real." << endl << endl;
50.     cout << "signal length" << "\t" << "mean(abs((rn-tn))" << endl;
51.     for( int len=MINLEN; len<MAXLEN; len+=STEP )
52.     {
53.         rn.resize(len);
54.         for( int i=0; i<len; ++i )
55.             rn[i] = rand()%10;
```

```
56.
57.         Rk = fftr2c( rn );
58.         tn = ifftc2r( Rk );
59. //        Rk = fft( rn );
60. //        tn = real( ifft(Rk) );
61.         cout << "    " << len << "\t\t" << "  " << sum(abs(rn-tn))/len << endl;
62.     }
63.     cout << endl;
64.
65.     return 0;
66. }
```

运行结果：

```
1.  forward transform: complex to complex.
2.  inverse transform: complex to complex.
3.
4.  signal length   mean(abs((sn-xn))
5.      1             0
6.      11            2.53071e-016
7.      21            2.04167e-015
8.      31            3.23943e-015
9.      41            2.72403e-015
10.     51            3.16559e-015
11.     61            2.59734e-015
12.     71            8.2923e-015
13.     81            6.68263e-015
14.     91            8.43078e-015
15.     101           1.00071e-014
16.     111           6.01269e-015
17.     121           3.93451e-015
18.     131           1.85043e-014
19.     141           8.70367e-015
20.     151           1.23613e-014
21.     161           6.08565e-015
22.     171           9.52577e-015
23.     181           4.67904e-015
24.     191           1.26907e-014
25.     201           7.17336e-015
26.     211           2.48636e-014
27.     221           1.16938e-014
28.     231           1.68039e-014
29.     241           1.37351e-014
30.     251           2.15957e-014
31.     261           1.34044e-014
```

```
271	1.57957e-014
281	5.82583e-014
291	6.69909e-015
301	3.8242e-014
311	7.39088e-014
321	2.20706e-014
331	2.58994e-014
341	2.24836e-014
351	1.86775e-014
361	2.74814e-014
371	9.27838e-015
381	1.14252e-014
391	1.0008e-014
401	5.94498e-014
411	1.53057e-014
421	5.47408e-014
431	7.25253e-014
441	7.87218e-015
451	2.92309e-014
461	5.81937e-014
471	4.23726e-014
481	4.51822e-014
491	1.10778e-013
501	3.52489e-014
511	4.42759e-014
521	1.18393e-013
531	5.78823e-014
541	1.26566e-013
551	3.69879e-014
561	5.15765e-014
571	3.83505e-014
581	3.7811e-014
591	3.06472e-014
601	1.39216e-013
611	2.93462e-014
621	4.50018e-014
631	1.35626e-013
641	1.4146e-013
651	1.36433e-014
661	1.35491e-013
671	2.22196e-014
681	7.82905e-014
691	1.41573e-013
701	1.63199e-013
```

```
     711           2.18095e-014
     721           3.11044e-014
     731           4.84485e-014
     741           6.47117e-014
     751           6.65676e-014
     761           1.50277e-013
     771           3.72186e-014
     781           4.02716e-014
     791           3.55064e-014
     801           8.17425e-014
     811           1.82512e-014
     821           1.78106e-013
     831           1.43936e-014
     841           8.8678e-014
     851           8.81647e-014
     861           6.26873e-015
     871           2.98626e-014
     881           1.08503e-013
     891           2.44492e-014
     901           8.29867e-014
     911           1.29968e-013
     921           5.25883e-014
     931           6.62704e-014
     941           4.87279e-014
     951           5.11132e-014
     961           1.33708e-013
     971           1.78372e-013
     981           8.49343e-014
     991           4.66578e-014


forward transform: real to complex .
inverse transform: complex to real.

signal length   mean(abs((rn-tn))
    1             0
    11            1.41301e-016
    21            1.11022e-015
    31            1.96513e-015
    41            1.3419e-015
    51            2.13333e-015
    61            1.88711e-015
    71            5.01281e-015
    81            4.06212e-015
```

```
91          4.96021e-015
101         6.08343e-015
111         3.27759e-015
121         2.21945e-015
131         1.2793e-014
141         4.77218e-015
151         8.30687e-015
161         3.41983e-015
171         5.91683e-015
181         3.50161e-015
191         8.77976e-015
201         4.62208e-015
211         1.66658e-014
221         7.22645e-015
231         1.01782e-014
241         9.79362e-015
251         1.55712e-014
261         7.40681e-015
271         1.01074e-014
281         3.81723e-014
291         3.8929e-015
301         2.22941e-014
311         4.76995e-014
321         1.36251e-014
331         1.78498e-014
341         1.3842e-014
351         1.16846e-014
361         1.50902e-014
371         4.81715e-015
381         7.45036e-015
391         5.79102e-015
401         3.68331e-014
411         8.45149e-015
421         3.5318e-014
431         4.64351e-014
441         4.5536e-015
451         1.82979e-014
461         4.11746e-014
471         2.67421e-014
481         2.76207e-014
491         7.49936e-014
501         2.24768e-014
511         2.8849e-014
521         7.29438e-014
```

```
164.    531         3.92363e-014
165.    541         8.14216e-014
166.    551         2.2449e-014
167.    561         3.26657e-014
168.    571         2.44778e-014
169.    581         2.23165e-014
170.    591         1.95616e-014
171.    601         8.50218e-014
172.    611         1.81843e-014
173.    621         2.89087e-014
174.    631         8.84136e-014
175.    641         9.02308e-014
176.    651         8.84008e-015
177.    661         8.58328e-014
178.    671         1.31807e-014
179.    681         4.8685e-014
180.    691         8.97514e-014
181.    701         1.07318e-013
182.    711         1.35663e-014
183.    721         1.82121e-014
184.    731         2.86341e-014
185.    741         4.07251e-014
186.    751         4.35763e-014
187.    761         9.37088e-014
188.    771         2.29587e-014
189.    781         2.29084e-014
190.    791         1.99687e-014
191.    801         5.27873e-014
192.    811         1.21722e-014
193.    821         1.06546e-013
194.    831         9.23176e-015
195.    841         5.27161e-014
196.    851         5.19815e-014
197.    861         3.93491e-015
198.    871         1.86002e-014
199.    881         7.19074e-014
200.    891         1.38322e-014
201.    901         4.85723e-014
202.    911         8.17142e-014
203.    921         3.32546e-014
204.    931         3.87686e-014
205.    941         3.16691e-014
206.    951         2.93188e-014
207.    961         8.21269e-014
```

```
208.     971           1.16589e-013
209.     981           5.44883e-014
210.     991           2.98482e-014
211.
212.
213. Process returned 0 (0x0)   execution time : 0.741 s
214. Press any key to continue.
```

7.4 FFTW 的 C++接口

FFTW是MIT学者用C语言编写的一个计算任意长度的一维或多维的离散Fourier变换程序库，该库的计算效率非常高，得到了广泛的应用，是信号处理中必不可少的一个C语言库函数。SP++对FFTW最新版本（3.2.2）中一维FFT提供了C++接口，如表 7-4所示。该表中的函数均是模板函数，支持float，double和long double三种数据类型。

表 7-4 FFTW 的 C++接口

Operation	Effect
fft(rxn,Xk)	实信号的 Fourier 变换
fft(cxn,Xk)	复信号的 Fourier 变换
ifft(Xk,rxn)	实信号的逆 Fourier 变换
ifft(Xk,cxn)	复信号的逆 Fourier 变换

测试代码：

```
1.  /*****************************************************************************
2.   *                              fftw_test.cpp
3.   *
4.   * FFTW interface testing.
5.   *
6.   * Zhang Ming, 2010-01, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <fftw.h>
15.
16.
17. using namespace std;
```

```
18. using namespace splab;
19. 
20. 
21. typedef float   Type;
22. const   int     N = 7;
23. 
24. 
25. int main()
26. {
27.     // complex to complex dft test...
28.     Vector< complex<Type> > xn( N );
29.     Vector< complex<Type> > yn( N );
30.     Vector< complex<Type> > Xk( N );
31. 
32.     for( int i=0; i<N; ++i )
33.     {
34.         Type theta = Type( 2*PI * i / N );
35.         xn[i] = complex<Type>( cos(theta), sin(theta) );
36.     }
37. 
38.     cout << setiosflags(ios::fixed) << setiosflags(ios::showpos) << setprecision(8
    );
39.     cout << "xn:   " << xn << endl << endl;
40. 
41.     fftw( xn, Xk );
42.     cout << "Xk=fft(xn):   " << Xk << endl << endl;
43.     ifftw( Xk, yn );
44.     cout << "xn-ifft(Xk):   " << xn-yn << endl << endl;
45. 
46.     // real to complex and complex to real dft test...
47.     Vector<Type> sn(N), tn(N);
48.     Vector< complex<Type> > Sk;
49.     for( int i=0; i<N; ++i )
50.     {
51.         Type theta = Type( 2*PI * i / N );
52.         sn[i] = sin(theta);
53.     }
54.     cout << "sn:   " << sn << endl;
55. 
56.     fftw( sn, Sk );
57.     cout << "Sk=fft(sn):   " << Sk << endl << endl;
58.     ifftw( Sk, tn );
59.     cout << "sn-ifft(Sk):   " << sn-tn << endl;
60. 
```

```
61.     return 0;
62. }
```

运行结果：

```
1.  xn:   size: +7 by 1
2.  (+1.00000000,+0.00000000)
3.  (+0.62348980,+0.78183150)
4.  (-0.22252095,+0.97492790)
5.  (-0.90096885,+0.43388382)
6.  (-0.90096885,-0.43388376)
7.  (-0.22252101,-0.97492790)
8.  (+0.62348968,-0.78183162)
9.
10.
11. Xk=fft(xn):   size: +7 by 1
12. (-0.00000018,-0.00000006)
13. (+7.00000000,-0.00000028)
14. (+0.00000021,-0.00000003)
15. (+0.00000018,+0.00000009)
16. (-0.00000003,+0.00000010)
17. (+0.00000005,+0.00000015)
18. (-0.00000028,+0.00000002)
19.
20.
21. xn-ifft(Xk):   size: +7 by 1
22. (+0.00000006,-0.00000000)
23. (+0.00000000,-0.00000006)
24. (-0.00000001,+0.00000000)
25. (+0.00000000,+0.00000006)
26. (+0.00000000,-0.00000003)
27. (-0.00000001,+0.00000000)
28. (+0.00000000,+0.00000000)
29.
30.
31. sn:   size: +7 by 1
32. +0.00000000
33. +0.78183150
34. +0.97492790
35. +0.43388382
36. -0.43388376
37. -0.97492790
38. -0.78183162
39.
40. Sk=fft(sn):   size: +4 by 1
```

```
(-0.00000006,+0.00000000)
(-0.00000013,-3.50000024)
(+0.00000006,-0.00000008)
(+0.00000009,-0.00000011)


sn-ifft(Sk):   size: +7 by 1
+0.00000000
-0.00000006
+0.00000000
+0.00000000
-0.00000003
+0.00000000
+0.00000006


Process returned 0 (0x0)   execution time : 0.116 s
Press any key to continue.
```

7.5 卷积与快速实现算法

卷积是信号处理中经常使用的运算，根据卷积定理可以将时域定义的卷积转化到频域进行计算，利用FFT算法可以将时间复杂度从N^2降低为$N\log_2 N$，极大提高了计算效率。SP++提供了普通的卷积算法与基于FFT的快速卷积算法，具体函数见表7-5。

表 7-5　卷积与相关的快速算法

Operation	Effect
conv(x,y)	计算 x 与 y 的卷积
convolution(s,f)	计算 s 与 f 的卷积（f 的长度小于 s）
fastConv(x,y)	计算 x 与 y 的卷积（快速算法）

测试代码：

```
/*****************************************************************************
 *                              convolution.cpp
 *
 * Convolution testing.
 *
 * Zhang Ming, 2010-01, Xi'an Jiaotong University.
 *****************************************************************************/

```

```
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <convolution.h>
14.
15.
16. using namespace std;
17. using namespace splab;
18.
19.
20. typedef double  Type;
21. const   int     M = 3;
22. const   int     N = 5;
23.
24.
25. int main()
26. {
27.     Vector<Type> xn( M ), yn( N );
28.     Vector<Type> zn;
29.
30.     for( int i=0; i<M; ++i )
31.         xn[i] = i;
32.     for( int i=0; i<N; ++i )
33.         yn[i] = i-N/2;
34.
35.     // convolution
36.     zn = conv( xn, yn );
37.     cout << "xn:  " << xn << endl << "yn:  " << yn << endl;
38.     cout << "convolution of xn and yn:   " << zn << endl;
39.     zn = fastConv( xn, yn );
40.     cout << "fast convolution of xn and yn:   " << zn << endl;
41.
42.     return 0;
43. }
```

运行结果：

```
1.  xn:  size: 3 by 1
2.  0
3.  1
4.  2
5.
6.  yn:  size: 5 by 1
7.  -2
```

```
-1
0
1
2

convolution of xn and yn:   size: 7 by 1
0
-2
-5
-2
1
4
4

fast convolution of xn and yn:   size: 7 by 1
-2.53765e-016
-2
-5
-2
1
4
4


Process returned 0 (0x0)   execution time : 0.059 s
Press any key to continue.
```

8 数字滤波器设计

8.1 常用窗函数

表 8-1列出了常用窗函数的调用形式，其中参数“amp”为窗函数的幅度，同时指定了窗函数的返回类型，其结果与Matlab中窗函数的结果相同。

表 8-1 常用的窗函数

Operation	Effect
window(wnName,N,amp)	产生类型为“wnName”的窗函数
window(wnName,N,alpha,amp)	产生类型为“wnName”的窗函数
rectangle(n,amp)	矩形窗
bartlett(n,amp)	Bartlett 窗
hanning(n,amp)	Hanning 窗
hamming(n,amp)	Hamming 窗
blackman(n,amp)	Blackman 窗
kaiser(n,alpha,amp)	Kaiser 窗
gauss(n,alpha,amp)	Gauss 窗

测试代码：

```
1.  /*****************************************************************************
2.   *                                window_test.cpp
3.   *
4.   * Windows function testing.
5.   *
6.   * Zhang Ming, 2010-01, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <window.h>
15.
16.
17. using namespace std;
```

```
18. using namespace splab;
19. 
20. 
21. typedef double  Type;
22. 
23. 
24. int main()
25. {
26.     int N = 5;
27.     Type A = 1.0;
28. 
29.     cout << setiosflags(ios::fixed) << setprecision(4);
30.     cout << "Rectangle window : " << rectangle(N,A) << endl;
31.     cout << "Bartlett window : " << bartlett(N,A) << endl;
32.     cout << "Hanning window : " << hanning(N,A) << endl;
33.     cout << "Hamming window : " << hamming(N,A) << endl;
34.     cout << "Blackman window : " << blackman(N,A) << endl;
35.     cout << "Kaiser window : " << kaiser(N,Type(8/PI),A) << endl;
36.     cout << "Gauss window : " << gauss(N,Type(2.5),A) << endl;
37. 
38.     cout << "Rectangle window : " << window("Rectangle",N,A) << endl;
39.     cout << "Bartlett window : " << window("Bartlett",N,A) << endl;
40.     cout << "Hanning window : " << window("Hanning",N,A) << endl;
41.     cout << "Hamming window : " << window("Hamming",N,A) << endl;
42.     cout << "Blackman window : " << window("Blackman",N,A) << endl;
43.     cout << "Kaiser window : " << window("Kaiser",N, Type(8/PI),A) << endl;
44.     cout << "Gauss window : " << window("Gauss",N,Type(2.5),A) << endl;
45. 
46.     return 0;
47. }
```

运行结果：

```
1.  Rectangle window : size: 5 by 1
2.  1.0000
3.  1.0000
4.  1.0000
5.  1.0000
6.  1.0000
7.  
8.  Bartlett window : size: 5 by 1
9.  0.0000
10. 0.5000
11. 1.0000
12. 0.5000
```

```
13. 0.0000
14.
15. Hanning window : size: 5 by 1
16. 0.0000
17. 0.5000
18. 1.0000
19. 0.5000
20. 0.0000
21.
22. Hamming window : size: 5 by 1
23. 0.0800
24. 0.5400
25. 1.0000
26. 0.5400
27. 0.0800
28.
29. Blackman window : size: 5 by 1
30. -0.0000
31. 0.3400
32. 1.0000
33. 0.3400
34. -0.0000
35.
36. Kaiser window : size: 5 by 1
37. 0.2933
38. 0.7742
39. 1.0000
40. 0.7742
41. 0.2933
42.
43. Gauss window : size: 5 by 1
44. 0.0439
45. 0.4578
46. 1.0000
47. 0.4578
48. 0.0439
49.
50. Rectangle window : size: 5 by 1
51. 1.0000
52. 1.0000
53. 1.0000
54. 1.0000
55. 1.0000
56.
```

```
Bartlett window : size: 5 by 1
0.0000
0.5000
1.0000
0.5000
0.0000

Hanning window : size: 5 by 1
0.0000
0.5000
1.0000
0.5000
0.0000

Hamming window : size: 5 by 1
0.0800
0.5400
1.0000
0.5400
0.0800

Blackman window : size: 5 by 1
-0.0000
0.3400
1.0000
0.3400
-0.0000

Kaiser window : size: 5 by 1
0.2933
0.7742
1.0000
0.7742
0.2933

Gauss window : size: 5 by 1
0.0439
0.4578
1.0000
0.4578
0.0439


Process returned 0 (0x0)   execution time : 0.089 s
```

```
Press any key to continue.
```

8.2 滤波器基类设计

所有滤波器都有其共性，如频率选择特性，通带频率，截止频率，增益等等，因此可以将这些共性抽取出来作为滤波器设计的一个基类，如表 8-2所示。

表 8-2 数字滤波器设计基类

Operation	Effect
DFD f(select)	创建 DFD 类
f.~ DFD <Type>()	析构 DFD 类
f. setParams(fs,f1,a1,f2,a2)	根据给定参数设计滤波器
f. setParams(fs,f1,a1,f2,f3,a2,f4,a3)	根据给定参数设计滤波器
f. dispInfo(fs,f1,a1,f2,f3,a2,f4,a3)	显示滤波器设计结果

8.3 FIR 数字滤波器设计

有限脉冲响应滤波器（FIR）具有线性相位特性，因此在一些对相位比较敏感的场合中得到了广泛应用。并且FIR没有反馈，所以稳定性比较好，当然其频率选择性也比较差。SP++中实现了基于窗函数法的FIR设计方法，具体见表 8-3。

表 8-3 FIR 数字滤波器设计

Operation	Effect
FIR<Type> f(select,win)	创建 FIR 类
FIR<Type> f(select,win,a)	创建 FIR 类(针对 Kaiser 与 Gauss 窗)
f.~FIR<Type>()	析构 FIR 类
f.design()	根据给定参数设计滤波器
f.dispInfo()	显示滤波器设计结果
f.getCoefs()	获取滤波器系数

测试代码：

```
/*****************************************************************************
 *                              fir_test.cpp
 *
 * FIR class testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/

```

```
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <fir.h>
14.
15.
16. using namespace std;
17. using namespace splab;
18.
19.
20. int main()
21. {
22.     string  wType = "Hamming";
23.
24. //    string  fType = "lowpass";
25. //    double  fs = 1000,
26. //            fpass = 200,
27. //            apass = -3,
28. //            fstop = 300,
29. //            astop = -20;
30. //    FIR fir( fType, wType );
31. //    fir.setParams( fs, fpass, apass, fstop, astop );
32.
33. //    string  fType = "highpass";
34. //    double  fs = 1000,
35. //            fstop = 200,
36. //            astop = -20,
37. //            fpass = 300,
38. //            apass = -3;
39. //    FIR fir( fType, wType );
40. //    fir.setParams( fs, fstop, astop, fpass, apass );
41.
42. //    string  fType = "bandpass";
43. //    double  fs = 1000,
44. //            fstop1 = 100,
45. //            astop1 = -20,
46. //            fpass1 = 200,
47. //            fpass2 = 300,
48. //            apass1 = -3,
49. //            fstop2 = 400,
50. //            astop2 = -20;
51. //    FIR fir( fType, wType );
```

```
52. //    fir.setParams( fs, fstop1, astop1, fpass1, fpass2, apass1, fstop2, astop2 );
53.
54.     string  fType = "bandstop";
55.     double  fs = 1000,
56.             fpass1 = 100,
57.             apass1 = -3,
58.             fstop1 = 200,
59.             fstop2 = 300,
60.             astop1 = -20,
61.             fpass2 = 400,
62.             apass2 = -3;
63.     FIR fir( fType, wType );
64.     fir.setParams( fs, fpass1, apass1, fstop1, fstop2, astop1, fpass2, apass2 );
65.
66.     fir.design();
67.     fir.dispInfo();
68.
69.     cout << endl;
70.     return 0;
71. }
```

运行结果：

```
1.                      Filter selectivity      :  bandstop
2.                      Window type             :  Hamming
3.                      Sampling Frequency (Hz) :  1000
4.                      Lower passband frequency (Hz) :  100
5.                      Lower passband gain      (dB) :  -3
6.                      Lower stopband frequency (Hz) :  200
7.                      Upper stopband frequency (Hz) :  300
8.                      Stopband gain            (dB) :  -20
9.                      Upper passband frequency (Hz) :  400
10.                     Upper passband gain      (dB) :  -3
11.
12.
13.                                 Filter Coefficients
14.
15.    N    [     N + 0           N + 1           N + 2           N + 3     ]
16.   ===   ================================================================
17.    0    -3.85192764e-003  8.42472981e-003  3.11153775e-003 -2.03549729e-003
18.    4    -3.55185234e-002  2.30171390e-002 -1.41331145e-001  2.61755439e-001
19.    8     2.88881794e-002  3.56158158e-001  3.56158158e-001  2.88881794e-002
20.   12     2.61755439e-001 -1.41331145e-001  2.30171390e-002 -3.55185234e-002
21.   16    -2.03549729e-003  3.11153775e-003  8.42472981e-003 -3.85192764e-003
```

```
22.
23.
24.          ==================== Edge Frequency Response ====================
25.              Mag(fp1) = -0.85449456(dB)       Mag(fp2) = -0.80635855(dB)
26.              Mag(fs1) = -21.347821(dB)       Mag(fs2) = -21.392825(dB)
27.
28.
29. Process returned 0 (0x0)   execution time : 0.025 s
30. Press any key to continue.
```

8.4 IIR 数字滤波器设计

无限脉冲响应（IIR）滤波器采用了反馈，因此具有很高的频率选择性，但同时也损失了线性相位特性，并且稳定性也不如FIR好。IIR可以借助模拟滤波器进行设计，而模拟滤波器有很成熟的设计方法，这给IIR设计带来了很大好处。SP++中实现的基于双线性变换的IIR设计方法，详见表 8-4。

表 8-4 IIR 数字滤波器设计

Operation	Effect
IIR<Type> f(select,method)	创建 IIR 类
f.~IIR<Type>()	析构 IIR 类
f.design()	根据给定参数设计滤波器
f.dispInfo()	显示滤波器设计结果
f.getNumCoefs()	获取滤波器的分子系数
f.getDenCoefs()	获取滤波器的分母系数

测试代码：

```
1.  /*****************************************************************************
2.   *                               iir_test.cpp
3.   *
4.   * IIR class testing.
5.   *
6.   * Zhang Ming, 2010-03, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iir.h>
14.
```

```
15.
16. using namespace std;
17. using namespace splab;
18.
19.
20. int main()
21. {
22. //    string  fType = "lowpass";
23. //    double  fs = 1000,
24. //            fpass = 200,
25. //            apass = -3,
26. //            fstop = 300,
27. //            astop = -20;
28. //    string  method = "Butterworth";
29. //    string  method = "Chebyshev";
30. //    string  method = "InvChebyshev";
31. //    string  method = "Elliptic";
32. //    IIR iir( fType, method );
33. //    iir.setParams( fs, fpass, apass, fstop, astop );
34.
35. //    string  fType = "highpass";
36. //    double  fs = 1000,
37. //            fstop = 200,
38. //            astop = -20,
39. //            fpass = 300,
40. //            apass = -3;
41. //    string  method = "Butterworth";
42. //    string  method = "Chebyshev";
43. //    string  method = "InvChebyshev";
44. //    string  method = "Elliptic";
45. //    IIR iir( fType, method );
46. //    iir.setParams( fs, fstop, astop, fpass, apass );
47.
48. //    string  fType = "bandpass";
49. //    double  fs = 1000,
50. //            fstop1 = 100,
51. //            astop1 = -20,
52. //            fpass1 = 200,
53. //            fpass2 = 300,
54. //            apass1 = -3,
55. //            fstop2 = 400,
56. //            astop2 = -20;
57. //    string  method = "Butterworth";
58. //    string  method = "Chebyshev";
```

```
59. //    string  method = "InvChebyshev";
60. //    string  method = "Elliptic";
61. //    IIR iir( fType, method );
62. //    iir.setParams( fs, fstop1, astop1, fpass1, fpass2, apass1, fstop2, astop2 );
63.
64.     string  fType = "bandstop";
65.     double  fs = 1000,
66.             fpass1 = 100,
67.             apass1 = -3,
68.             fstop1 = 200,
69.             fstop2 = 300,
70.             astop1 = -20,
71.             fpass2 = 400,
72.             apass2 = -3;
73. //    string  method = "Butterworth";
74. //    string  method = "Chebyshev";
75. //    string  method = "InvChebyshev";
76.     string  method = "Elliptic";
77.     IIR iir( fType, method );
78.     iir.setParams( fs, fpass1, apass1, fstop1, fstop2, astop1, fpass2, apass2 );
79.
80.     iir.design();
81.     iir.dispInfo();
82.
83.     cout << endl;
84.     return 0;
85. }
```

运行结果：

```
1.                      Filter selectivity      :  bandstop
2.                      Approximation method    :  Elliptic
3.                      Filter order            :  4
4.                      Overall gain            :  0.153272
5.                      Sampling Frequency (Hz) :  1000
6.                      Lower passband frequency (Hz) :  100
7.                      Lower passband gain      (dB) :  -3
8.                      Lower stopband frequency (Hz) :  200
9.                      Upper stopband frequency (Hz) :  300
10.                     Stopband gain            (dB) :  -20
11.                     Upper passband frequency (Hz) :  400
12.                     Upper passband gain      (dB) :  -3
13.
14.
```

```
            Numerator Coefficients                Denominator Coefficients

  N   [  1       z^-1           z^-2  ]     [  1       z^-1           z^-2  ]
 ===  ================================     ================================
  1    1.0   -0.45087426    1.00000000       1.0   -1.44482415    0.70572930
  2    1.0    0.45087426    1.00000000       1.0    1.44482415    0.70572930


          ==================== Edge Frequency Response ====================
              Mag(fp1) = -3.00000000(dB)       Mag(fp2) = -3.00000000(dB)
              Mag(fs1) = -36.87418446(dB)       Mag(fs2) = -36.87418446(dB)


Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.
```

9 随机信号处理

9.1 随机数生成器

在数值计算与信号处理中经常要产生一些给定分布的随机序列，以模拟实际信号与噪声，比如经常使用的Gauss白噪声。SP++中提供了常用分布的随机数与随机序列的生成函数，具体包括均匀分布、正态分布、指数分布、Rayleigh分布、Poisson分布、Bernoulli分布等，详见表 9-1。

表 9-1 随机数与随机序列产生器

Operation	Effect
randu(s,a,b)	均匀分布的随机数
randn (s,u,sigma)	正态分布的随机数
rande (s,beta)	指数分布的随机数
randr (s,sigma)	Rayleigh 分布的随机数
randp (s,lambda)	Poisson 分布的随机数
randb (s,p)	Bernoulli 分布的随机数
randu(s,a,b,N)	均匀分布的随机序列
randn (s,u,sigma,N)	正态分布的随机序列
rande (s,beta,N)	指数分布的随机序列
randr (s,sigma,N)	Rayleigh 分布的随机序列
randp (s,lambda,N)	Poisson 分布的随机序列
randb (s,p,N)	Bernoulli 分布的随机序列

测试代码：

```
1.  /*****************************************************************************
2.   *                              random_test.cpp
3.   *
4.   * Random number generator testing.
5.   *
6.   * Zhang Ming, 2010-10, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10.  #include <iostream>
11.  #include <iomanip>
12.  #include <random.h>
```

```
13. #include <statistics.h>
14. 
15. 
16. using namespace std;
17. using namespace splab;
18. 
19. 
20. typedef double  Type;
21. const   int     N = 100;
22. 
23. 
24. int main()
25. {
26.     int seed = 37;
27.     int low = 0, high = 100;
28.     Type mu = 0.0, sigma = 1.0;
29.     Type beta = 2.0;
30.     Type lambda = 4.0;
31.     Type p = 0.5;
32. 
33.     Vector<Type> rs(N);
34.     Vector<int> irs(N);
35.     Vector<double> tmp(N);
36. 
37. //    Random rand(seed);
38. //    cout << "Random Number Generator:" << endl;
39. //    for( int i=0; i<N; ++i )
40. //    {
41. //        cout << rand.random() << "\t";
42. //        if( !((i+1)%4) )
43. //            cout << endl;
44. //    }
45. //    cout << endl << endl;
46. 
47.     cout << "Uniform distribution: U ~ ( " << low << ", " << high << " )" << endl;

48.     for( int i=0; i<N; ++i )
49.         cout << randu( seed, low, high ) << "\t";
50.     cout << endl;
51.     irs = randu( seed, low, high, N );
52.     cout << "mean     (theoretical   generated):   "
53.          << mean(irs) << "\t" << (high-low)/2 << endl;
54.     cout << "variance (theoretical   generated):   "
55.          << var(irs) << "\t" << (high-low)*(high-low)/12
```

```
56.           << endl << endl << endl;
57.
58.     cout << "Normal distribution: N ~ ( " << mu << ", " << sigma << " )" << endl;

59.     cout << setiosflags(ios::fixed) << setprecision(4);
60.     for( int i=0; i<N; ++i )
61.         cout << randn( seed, mu, sigma ) << "\t\t";
62.     cout << endl;
63.     rs = randn( seed, mu, sigma, N );
64.     cout << "mean     (theoretical   generated):   "
65.          << mean(rs) << "\t" << mu << endl;
66.     cout << "variance (theoretical   generated):   "
67.          << var(rs) << "\t" << sigma*sigma
68.          << endl << endl << endl;
69.
70.     cout << "Exponential distribution: E ~ ( " << beta << " )" << endl;
71.     cout << setiosflags(ios::fixed) << setprecision(4);
72.     for( int i=0; i<N; ++i )
73.         cout << rande( seed, beta ) << "\t\t";
74.     cout << endl;
75.     rs = rande( seed, beta, N );
76.     cout << "mean     (theoretical   generated):   "
77.          << mean(rs) << "\t" << beta << endl;
78.     cout << "variance (theoretical   generated):   "
79.          << var(rs) << "\t" << beta*beta
80.          << endl << endl << endl;
81.
82.     cout << "Rayleigh distribution: R ~ ( " << sigma << " )" << endl;
83.     cout << setiosflags(ios::fixed) << setprecision(4);
84.     for( int i=0; i<N; ++i )
85.         cout << randr( seed, sigma ) << "\t\t";
86.     cout << endl;
87.     rs = randr( seed, sigma, N );
88.     cout << "mean     (theoretical   generated):   "
89.          << mean(rs) << "\t" << sigma*sqrt(PI/2.0) << endl;
90.     cout << "variance (theoretical   generated):   "
91.          << var(rs) << "\t" << (2-PI/2)*sigma*sigma
92.          << endl << endl << endl;
93.
94.     cout << "Poisson distribution: B ~ ( " << p << " )" << endl;
95.     for( int i=0; i<N; ++i )
96.         cout << randp( seed, lambda ) << "\t\t";
97.     cout << endl;
98.     irs = randp( seed, lambda, N );
```

```
    for( int i=0; i<N; ++i )
        tmp[i] = irs[i];
    cout << "mean     (theoretical   generated):   "
         << mean(tmp) << "\t" << lambda << endl;
    cout << "variance (theoretical   generated):   "
         << var(tmp) << "\t" << lambda
         << endl << endl << endl;

    cout << "Bernoulli distribution: B ~ ( " << p << " )" << endl;
    for( int i=0; i<N; ++i )
        cout << randb( seed, p ) << "\t\t";
    cout << endl;
    irs = randb( seed, p, N );
    for( int i=0; i<N; ++i )
        tmp[i] = irs[i];
    cout << "mean     (theoretical   generated):   "
         << mean(tmp) << "\t" << p << endl;
    cout << "variance (theoretical   generated):   "
         << var(tmp) << "\t" << p*(1-p)
         << endl << endl;

    return 0;
}
```

运行结果：

```
Uniform distribution: U ~ ( 0, 100 )
0       14      25      98      81      1       5       72      72      50
31      73      54      95      90      93      43      81      86      76
99      76      28      16      35      57      9       74      55      87
64      34      41      5       63      22      49      13      91      46
55      99      18      26      21      59      65      9       27      87
10      70      81      42      35      62      23      24      62      71
35      6       84      38      86      82      76      45      34      38
65      29      42      94      45      46      80      42      6       11
46      65      20      21      2       88      71      74      58      43
56      94      17      5       32      0       66      54      97      40

mean     (theoretical   generated):   49      50
variance (theoretical   generated):   840     833


Normal distribution: N ~ ( 0, 1 )
-0.7148         2.4179          -0.4947         -0.4264         0.0118
0.2321          -0.7548         -0.5397         -0.6533         -1.2043
```

```
 1.8746          0.7539          0.0210          0.3340          0.2786
-1.8528          -0.7200         1.1448          -0.7554         -1.0937
0.1583           -2.9330         0.3818          -0.0193         -1.3456
1.4424           1.8424          -0.6886         1.6361          0.9801
1.2555           0.1753          1.6852          0.4694          -0.2241
-2.2454          -1.3620         -0.2316         -2.3769         1.0066
-0.3988          0.5761          -0.2895         -0.2019         -0.6050
-0.4084          -0.4072         -1.2768         0.9203          -0.0252
-0.5463          -0.3924         -1.2157         -0.9195         -0.2086
0.0866           0.3005          0.2995          0.8217          1.5764
0.5863           2.6818          0.4002          -0.3112         -1.5317
-1.0913          0.4349          0.7805          1.3383          0.4580
-0.8948          0.1357          -0.1333         0.7235          0.7452
-0.2743          1.0441          -0.5881         -0.1807         1.8741
0.4193           -0.1397         0.7841          0.8995          1.7939
1.0729           -1.0523         -1.0604         0.1780          -1.8157
0.8338           -0.2278         0.7157          0.8513          0.7649
0.4494           -1.6238         0.5206          -0.0174         0.1020

mean     (theoretical   generated):   0.0480    0.0000
variance (theoretical   generated):   1.0853    1.0000


Exponential distribution: E ~ ( 2.0000 )
14.1841          3.8456          2.7722          0.0209          0.4107
7.9776           5.8336          0.6396          0.6362          1.3468
2.3214           0.6173          1.2231          0.0943          0.2085
0.1379           1.6627          0.4015          0.2789          0.5368
0.0145           0.5387          2.5359          3.6533          2.0613
1.1008           4.7209          0.5873          1.1839          0.2580
0.8894           2.1468          1.7691          5.9504          0.9212
3.0099           1.3934          3.9596          0.1688          1.5344
1.1923           0.0128          3.3930          2.6764          3.0453
1.0373           0.8570          4.6297          2.5929          0.2703
4.4738           0.6858          0.4023          1.6962          2.0739
0.9469           2.8776          2.8258          0.9276          0.6701
2.0931           5.4628          0.3361          1.8864          0.2925
0.3967           0.5345          1.5645          2.1240          1.9347
0.8492           2.4460          1.7180          0.1191          1.5695
1.5185           0.4214          1.7257          5.5630          4.2447
1.5262           0.8318          3.2001          3.0868          7.5201
0.2372           0.6579          0.5884          1.0777          1.6624
1.1570           0.1181          3.4733          5.8509          2.2530
9.7065           0.8140          1.1978          0.0423          1.8098
```

```
64.
65. mean     (theoretical   generated):   2.0445    2.0000
66. variance (theoretical   generated):   5.0310    4.0000
67.
68.
69. Rayleigh distribution: R ~ ( 1.0000 )
70. 3.7662          1.9610          1.6650          0.1445          0.6409
71. 2.8245          2.4153          0.7997          0.7976          1.1605
72. 1.5236          0.7857          1.1060          0.3070          0.4566
73. 0.3714          1.2895          0.6336          0.5281          0.7326
74. 0.1203          0.7340          1.5924          1.9114          1.4357
75. 1.0492          2.1728          0.7664          1.0881          0.5080
76. 0.9431          1.4652          1.3301          2.4393          0.9598
77. 1.7349          1.1804          1.9899          0.4108          1.2387
78. 1.0919          0.1133          1.8420          1.6360          1.7451
79. 1.0185          0.9257          2.1517          1.6103          0.5199
80. 2.1151          0.8281          0.6343          1.3024          1.4401
81. 0.9731          1.6964          1.6810          0.9631          0.8186
82. 1.4468          2.3373          0.5798          1.3735          0.5408
83. 0.6299          0.7311          1.2508          1.4574          1.3909
84. 0.9215          1.5640          1.3107          0.3452          1.2528
85. 1.2323          0.6492          1.3137          2.3586          2.0603
86. 1.2354          0.9120          1.7889          1.7569          2.7423
87. 0.4871          0.8111          0.7671          1.0381          1.2893
88. 1.0756          0.3436          1.8637          2.4189          1.5010
89. 3.1155          0.9022          1.0944          0.2056          1.3453
90.
91. mean     (theoretical   generated):   1.2553    1.2533
92. variance (theoretical   generated):   0.4735    0.4292
93.
94.
95. Poisson distribution: B ~ ( 0.5000 )
96. 0               4               3               11              2
97. 5               2               3               5               3
98. 3               5               3               7               5
99. 3               3               2               5               1
100. 0              6               1               4               3
101. 4              6               9               4               5
102. 6              5               4               5               4
103. 4              3               4               3               3
104. 1              8               4               6               3
105. 4              1               5               2               3
106. 6              2               3               1               2
107. 7              3               5               4               3
```

```
1 1 3 6 7 9
2 1 1 2 5 7
3 6 7 8 5 3
4 6 7 7 1 5
5 1 6 2 3 1
6 3 3 2 4 4
7 5 2 1 3 3
8 1 3 4 6 5
9
10 mean (theoretical generated): 3.9000 4.0000
11 variance (theoretical generated): 4.6566 4.0000
12
13
14 Bernoulli distribution: B ~ ( 0.5000 )
15 1 1 1 0 0
16 1 1 0 0 0
17 1 0 0 0 0
18 0 1 0 0 0
19 0 0 1 1 1
20 0 1 0 0 0
21 0 1 1 1 0
22 1 1 1 0 1
23 0 0 1 1 1
24 0 0 1 1 0
25 1 0 0 1 1
26 0 1 1 0 0
27 1 1 0 1 0
28 0 0 1 1 1
29 0 1 1 0 1
30 1 0 1 1 1
31 1 0 1 1 1
32 0 0 0 0 1
33 0 0 1 1 1
34 1 0 0 0 1
35
36 mean (theoretical generated): 0.5100 0.5000
37 variance (theoretical generated): 0.2524 0.2500
38
39
40 Process returned 0 (0x0) execution time : 0.203 s
41 Press any key to continue.
```

9.2 概率统计中的常用函数

“statistics.h”头文件中提供了随机信号处理中的一些常用算法，如随机变量的均值、中值、方差、偏度、峭度以及将随机变量标准化和概率密度函数估计算法等，详见表 9-2。

表 9-2 概率统计中的常用函数

Operation	Effect
mid(v)	取向量的中值
mean(v)	计算随机变量的均值
var(v)	计算随机变量的方差
stdVar(v)	计算随机变量的标准差
standard (v)	随机变量的标准化
skew(v)	计算随机变量的偏度
kurt(v)	计算随机变量的峭度
pdf(v,lambda)	估计随机变量的概率密度函数

测试代码：

```
/*****************************************************************************
 *                               statistics_test.cpp
 *
 * Statistics routines testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <random.h>
#include <statistics.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     N = 1000;

```

```
int main()
{
    Vector<Type> xn = randn( 37, Type(0.0), Type(1.0), N );

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "The minimum maximum and median value of the sequence." << endl;
    cout << min(xn) << endl << max(xn) << endl << mid(xn) <<endl << endl;
    cout << "The mean, variance and standard variance of the sequence." << endl;
    cout << mean(xn) << endl << var(xn) << endl << stdVar(xn) << endl << endl;

    Vector<Type> yn = xn - mean(xn);
    cout << "The skew and kurtosis of the sequence." << endl;
    cout << skew(yn) << endl << kurt(yn) << endl << endl;

    cout << "The PDF of the sequence." << endl;
    cout << pdf(yn) << endl;

    return 0;
}
```

运行结果：

```
The minimum maximum and median value of the sequence.
-3.6176
3.1326
0.0294

The mean, variance and standard variance of the sequence.
0.0167
1.0343
1.0170

The skew and kurtosis of the sequence.
-0.1354
-0.0274

The PDF of the sequence.
size: 14 by 1
0.0024
0.0026
0.0142
0.0406
0.0864
0.1595
0.2216
```

```
0.1322
0.1923
0.0903
0.0426
0.0127
0.0024
0.0003


Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

9.3 相关与快速实现算法

相关是随机信号处理中经常使用的运算，根据相关与卷积的关系可以将相关转化为卷积运算，进而可以利用卷积定理将时域的相关转化到频域进行计算，利用FFT算法可以将时间复杂度从N^2 降低为$N\log_2 N$，极大提高了计算效率。SP++中提供了自相关与互相关的函数，如表 9-3所示，计算结果与Matlab相同，可以通过参数”opt”选择普通相关运算，有偏相关估计和无偏相关估计。

表 9-3 卷积与相关的快速算法

Operation	Effect
corr (x,opt)	计算 x 的自相关
corr (x,y,opt)	计算 x 与 y 的互相关
fastCorr (x,opt)	通过 FFT 计算 x 的自相关
fastCorr (x,y,opt)	通过 FFT 计算 x 与 y 的互相关

测试代码：

```
/*****************************************************************************
 *                            correlation_test.cpp
 *
 * Correlation testing.
 *
 * Zhang Ming, 2010-10, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
```

```
#include <correlation.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     M = 3;
const   int     N = 5;


int main()
{
    Vector<Type> xn( M ), yn( N );

    for( int i=0; i<M; ++i )
        xn[i] = i;
    for( int i=0; i<N; ++i )
        yn[i] = i-N/2;

    cout << setiosflags(ios::fixed) << setprecision(4);
    cout << "xn:   " << xn << endl;
    cout << "yn:   " << yn << endl;

    // auto and cross correlation functions
    cout << "auto-correlation of xn:   " << corr(xn) << endl;
    cout << "biased auto-correlation of xn:   " << corr(xn,"biased") << endl;
    cout << "unbiased auto-correlation of xn:   " << corr(xn,"unbiased") << endl;

    cout << "cross-correlation of xn and yn:   " << corr(xn,yn) << endl;
    cout << "cross-correlation of yn and xn:   " << corr(yn,xn) << endl;

    cout << "biased cross-correlation of xn and yn:   "
         << corr(xn,yn,"biased") << endl;
    cout << "biased cross-correlation of yn and xn:   "
         << corr(yn,xn,"biased") << endl;

    cout << "unbiased cross-correlation of xn and yn:   "
         << corr(xn,yn,"unbiased") << endl;
    cout << "unbiased cross-correlation of yn and xn:   "
         << corr(yn,xn,"unbiased") << endl;

```

```
57.     // fast auto and cross correlation functions
58.     cout << "fast auto-correlation of xn:   "
59.          << fastCorr(xn) << endl;
60.     cout << "fast biased auto-correlation of xn:   "
61.          << fastCorr(xn,"biased") << endl;
62.     cout << "fast unbiased auto-correlation of xn:   "
63.          << fastCorr(xn,"unbiased") << endl;
64.
65.     cout << "fast cross-correlation of xn and yn:   " << fastCorr(xn,yn) << endl;
66.     cout << "fast cross-correlation of yn and xn:   " << fastCorr(yn,xn) << endl;
67.
68.     cout << "fast biased cross-correlation of xn and yn:   "
69.          << fastCorr(xn,yn,"biased") << endl;
70.     cout << "fast biased cross-correlation of yn and xn:   "
71.          << fastCorr(yn,xn,"biased") << endl;
72.
73.     cout << "fast unbiased cross-correlation of xn and yn:   "
74.          << fastCorr(xn,yn,"unbiased") << endl;
75.     cout << "fast unbiased cross-correlation of yn and xn:   "
76.          << fastCorr(yn,xn,"unbiased") << endl;
77.
78.     return 0;
79. }
```

运行结果：

```
1.  xn:   size: 3 by 1
2.  0.0000
3.  1.0000
4.  2.0000
5.
6.  yn:   size: 5 by 1
7.  -2.0000
8.  -1.0000
9.  0.0000
10. 1.0000
11. 2.0000
12.
13. auto-correlation of xn:   size: 5 by 1
14. 0.0000
15. 2.0000
16. 5.0000
17. 2.0000
```

```
0.0000

biased auto-correlation of xn:   size: 5 by 1
0.0000
0.6667
1.6667
0.6667
0.0000

unbiased auto-correlation of xn:   size: 5 by 1
0.0000
1.0000
1.6667
1.0000
0.0000

cross-correlation of xn and yn:   size: 9 by 1
0.0000
2.0000
5.0000
2.0000
-1.0000
-4.0000
-4.0000
0.0000
0.0000

cross-correlation of yn and xn:   size: 9 by 1
0.0000
0.0000
-4.0000
-4.0000
-1.0000
2.0000
5.0000
2.0000
0.0000

biased cross-correlation of xn and yn:   size: 9 by 1
0.0000
0.4000
1.0000
0.4000
-0.2000
```

```
-0.8000
-0.8000
0.0000
0.0000

biased cross-correlation of yn and xn:   size: 9 by 1
0.0000
0.0000
-0.8000
-0.8000
-0.2000
0.4000
1.0000
0.4000
0.0000

unbiased cross-correlation of xn and yn:   size: 9 by 1
0.0000
1.0000
1.6667
0.5000
-0.2000
-1.0000
-1.3333
0.0000
0.0000

unbiased cross-correlation of yn and xn:   size: 9 by 1
0.0000
0.0000
-1.3333
-1.0000
-0.2000
0.5000
1.6667
1.0000
0.0000

fast auto-correlation of xn:   size: 5 by 1
0.0000
2.0000
5.0000
2.0000
-0.0000
```

```
106.
107. fast biased auto-correlation of xn:   size: 5 by 1
108. 0.0000
109. 0.6667
110. 1.6667
111. 0.6667
112. -0.0000
113.
114. fast unbiased auto-correlation of xn:   size: 5 by 1
115. 0.0000
116. 1.0000
117. 1.6667
118. 1.0000
119. -0.0000
120.
121. fast cross-correlation of xn and yn:   size: 9 by 1
122. -0.0000
123. 2.0000
124. 5.0000
125. 2.0000
126. -1.0000
127. -4.0000
128. -4.0000
129. -0.0000
130. -0.0000
131.
132. fast cross-correlation of yn and xn:   size: 9 by 1
133. -0.0000
134. -0.0000
135. -4.0000
136. -4.0000
137. -1.0000
138. 2.0000
139. 5.0000
140. 2.0000
141. 0.0000
142.
143. fast biased cross-correlation of xn and yn:   size: 9 by 1
144. -0.0000
145. 0.4000
146. 1.0000
147. 0.4000
148. -0.2000
149. -0.8000
```

```
-0.8000
-0.0000
-0.0000

fast biased cross-correlation of yn and xn:   size: 9 by 1
-0.0000
-0.0000
-0.8000
-0.8000
-0.2000
0.4000
1.0000
0.4000
0.0000

fast unbiased cross-correlation of xn and yn:   size: 9 by 1
-0.0000
1.0000
1.6667
0.5000
-0.2000
-1.0000
-1.3333
-0.0000
-0.0000

fast unbiased cross-correlation of yn and xn:   size: 9 by 1
-0.0000
-0.0000
-1.3333
-1.0000
-0.2000
0.5000
1.6667
1.0000
0.0000


Process returned 0 (0x0)   execution time : 0.140 s
Press any key to continue.
```

10 功率谱估计

10.1 经典谱估计方法

通过有限的观测值来估计随机信号的功率在频域的分布即为随机信号的功率谱估计。SP++提供了经典的谱估计算法，包括相关函数法、周期图法与几种改进的周期图法，如表 10-1所示。这几类经典谱估计算法的特点是稳定性比较好，但频谱分辨率较低。

表 10-1　经典谱估计方法

Operation	Effect
correlogramPSE (xn,L)	相关函数法谱估计
periodogramPSE (xn,wn,L)	周期图法谱估计
bartlettPSE (xn,M,L)	Bartlett 法谱估计
welchPSE(xn,wn,K,L)	Welch 法谱估计
btPSE (xn,wn,L)	BT 法谱估计

测试代码：

```
/*****************************************************************************
 *                            classicalpse_test.cpp
 *
 * Classical power spectrum estimation testing.
 *
 * Zhang Ming, 2010-11, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <cstring>
#include <vectormath.h>
#include <classicalpse.h>
#include "engine.h"


using namespace std;
```

```
20. using namespace splab;
21. 
22. 
23. typedef double  Type;
24. const   int     N = 100;
25. const   int     K = 25;
26. const   int     M = 50;
27. const   int     L = 200;
28. 
29. 
30. int main()
31. {
32.     /****************************** [ signal ] ******************************/
33.     int mfn = L/2+1;
34.     Type amp1 = Type(1.0),
35.          amp2 = Type(0.5);
36.     Type f1 = Type(0.3),
37.          f2 = Type(0.4);
38.     Vector<Type> tn = linspace(Type(0), Type(N-1), N );
39.     Vector<Type> sn = amp1*sin(TWOPI*f1*tn) + amp2*sin(TWOPI*f2*tn);
40. 
41.     /****************************** [ widow ] ******************************/
42.     Vector<Type> wn = window( "Hamming", M, Type(1.0) );
43. 
44.     /****************************** [ PSD ] ******************************/
45. //  Vector<Type> Ps = correlogramPSE( sn, L );
46. //  Vector<Type> Ps = periodogramPSE( sn, wn, L );
47. //    Vector<Type> Ps = bartlettPSE( sn, M, L );
48.     Vector<Type> Ps = welchPSE( sn, wn, K, L );
49. //  Vector<Type> Ps = btPSE( sn, wn, L );
50. 
51.     /****************************** [ PLOT ] ******************************/
52.     Engine *ep  = engOpen( NULL );
53.     if( !ep )
54.     {
55.         cerr << "Cannot open Matlab Engine!" << endl;
56.         exit(1);
57.     }
58. 
59.     mxArray *msn = mxCreateDoubleMatrix( N, 1, mxREAL );
60.     mxArray *mPs = mxCreateDoubleMatrix( mfn, 1, mxREAL );
61.     memcpy( mxGetPr(msn), sn, N*sizeof(Type) );
62.     memcpy( mxGetPr(mPs), Ps, mfn*sizeof(Type) );
63.     engPutVariable( ep, "sn", msn );
```

```
64.     engPutVariable( ep, "Ps", mPs );
65.
66.     const char *mCmd =  " figure('name','Welch Method of Spectrum Estimation'); \

67.         N = length(sn); mfn = length(Ps); \
68.         subplot(2,1,1); \
69.             plot((0:N-1), sn); \
70.             axis([0,N,min(sn),max(sn)]); \
71.             title('(a)   Signal', 'FontSize',12); \
72.             xlabel('Samples', 'FontSize',12); \
73.             ylabel('Amplitude', 'FontSize',12); \
74.         subplot(2,1,2); \
75.             h = stem((0:mfn-1)/(mfn-1)/2, Ps); \
76.             axis([0,0.5,min(Ps),max(Ps)]); \
77.             set(h,'MarkerFaceColor','blue'); \
78.             set(gca, 'XTick', 0:0.1:0.5); \
79.             grid on; \
80.             title('(b)   Spectrum', 'FontSize',12); \
81.             xlabel('Normalized Frequency ( f / fs )', 'FontSize',12); \
82.             ylabel('Amplitude', 'FontSize',12); ";
83.     engEvalString( ep, mCmd );
84.
85.     mxDestroyArray( msn );
86.     mxDestroyArray( mPs );
87.     system( "pause" );
88.     engClose(ep);
89.
90.     return 0;
91. }
```

运行结果如图 10-1所示：

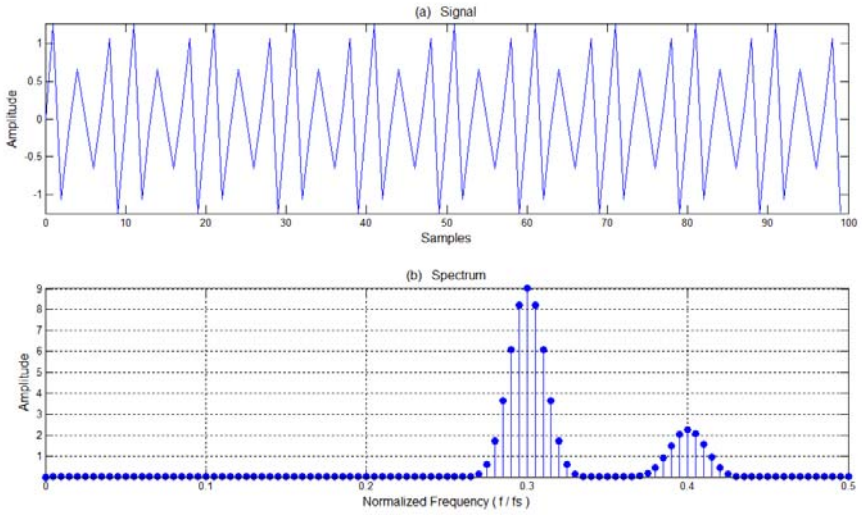

图 10-1　Welch 功率谱估计

10.2 参数化谱估计方法

参数化谱估计方法假定随机序列满足一定的结构，比如AR模型等等，然后从观测数据中估计出描述该结构的参数，进而通过所估计的参数确定随机序列的功率谱。此类方法具体很高的频率分辨率，但对模型结构要有充分的先验信息，否则会产生较大的估计误差。SP++提供了四类参数化谱估计方法，有AR模型和ARMA模型，具体如表 10-2所示。

表 10-2　参数化谱估计方法

Operation	Effect
yulewalkerPSE (xn,p,sigma)	Yule-Walker 谱估计方法
burgPSE (xn,p,sigma)	Burg 谱估计方法
fblplsPSE (xn,p,sigma)	正反向线性预测最小二乘谱估计方法
armaPSD (ak,bk,sigma,L)	ARMA 模型谱估计方法

测试代码：

```
/*****************************************************************************
 *                            parametricpse_test.cpp
 *
 * parametric power spectrum estimation testing.
 *
 * Zhang Ming, 2010-11, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <cstring>
#include <random.h>
#include <vectormath.h>
#include <parametricpse.h>
#include "engine.h"


using namespace std;
using namespace splab;

```

```
25. typedef double  Type;
26. const   int     N = 50;
27. const   int     L = 200;
28. const   int     yuleOrder = 4;
29. const   int     burgOrder = 4;
30. const   int     lplsOrder = 4;
31.
32.
33. int main()
34. {
35.     /****************************** [ signal ] ******************************/
36.     cout << setiosflags(ios::fixed) << setprecision(4);
37.     int mfn = L/2+1;
38.     Type amp1 = Type(1.0),
39.          amp2 = Type(1.0);
40.     Type f1 = Type(0.2),
41.          f2 = Type(0.4);
42.     Type sigma2, SNR;
43.
44.     Vector<Type> tn = linspace(Type(0), Type(N-1), N );
45.     Vector<Type> sn = amp1*sin(TWOPI*f1*tn) + amp2*sin(TWOPI*f2*tn);
46.     Vector<Type> wn = randn( 37, Type(0.0), Type(0.1), N );
47.     Vector<Type> xn = sn + wn;
48.     SNR = 20*log10(norm(sn)/norm(wn));
49.     cout << "The SNR = " << SNR << endl << endl;
50.
51.     /******************************** [ PSD ] ********************************/
52. //    Vector<Type> ak = yulewalkerPSE( xn, yuleOrder, sigma2 );
53. //    cout << "The estimated AR coefficients by Youle-Walker method are: "
54. //         << ak << endl;
55.
56. //    Vector<Type> ak = burgPSE( xn, burgOrder, sigma2 );
57. //    cout << "The estimated AR coefficients by Burg method are: "
58. //         << ak << endl;
59.
60.     Vector<Type> ak = fblplsPSE( xn, lplsOrder, sigma2 );
61.     cout << "The estimated AR coefficients by Youle-Walker method are: "
62.          << ak << endl;
63.
64.     cout << "The estimated variance is:   " << sigma2 << endl << endl;
65.
66.     Vector<Type> bk(1,Type(1.0));
67.     Vector<Type> Px = armaPSD( ak, bk, sigma2, L );
68.
```

```
69.     /****************************** [ PLOT ] ******************************/
70.     Engine *ep  = engOpen( NULL );
71.     if( !ep )
72.     {
73.         cerr << "Cannot open Matlab Engine!" << endl;
74.         exit(1);
75.     }
76.
77.     mxArray *mxn = mxCreateDoubleMatrix( N, 1, mxREAL );
78.     mxArray *mPx = mxCreateDoubleMatrix( mfn, 1, mxREAL );
79.     memcpy( mxGetPr(mxn), xn, N*sizeof(Type) );
80.     memcpy( mxGetPr(mPx), Px, mfn*sizeof(Type) );
81.     engPutVariable( ep, "xn", mxn );
82.     engPutVariable( ep, "Px", mPx );
83.
84.     const char *mCmd =  " figure('name','FBLPLS Method of Spectrum Estimation'); \

85.         N = length(xn); mfn = length(Px); \
86.         subplot(2,1,1); \
87.             plot((0:N-1), xn); \
88.             axis([0,N,min(xn),max(xn)]); \
89.             title('(a)   Signal', 'FontSize',12); \
90.             xlabel('Samples', 'FontSize',12); \
91.             ylabel('Amplitude', 'FontSize',12); \
92.         subplot(2,1,2); \
93.             h = stem((0:mfn-1)/(mfn-1)/2, Px); \
94.             axis([0,0.5,min(Px),max(Px)]); \
95.             set(h,'MarkerFaceColor','blue'); \
96.             set(gca, 'XTick', 0:0.1:0.5); \
97.             grid on; \
98.             title('(b)   Spectrum', 'FontSize',12); \
99.             xlabel('Normalized Frequency ( f / fs )', 'FontSize',12); \
100.            ylabel('Amplitude', 'FontSize',12); ";
101.    engEvalString( ep, mCmd );
102.
103.    mxDestroyArray( mxn );
104.    mxDestroyArray( mPx );
105.    system( "pause" );
106.    engClose(ep);
107.
108.    return 0;
109. }
```

运行结果如下及图 10-2所示：

```
The SNR = 18.9846

The estimated AR coefficients by Youle-Walker method are: size: 5 by 1
1.0000
0.9464
0.9341
0.9486
0.9411

The estimated variance is:   0.0404

请按任意键继续. . .
```

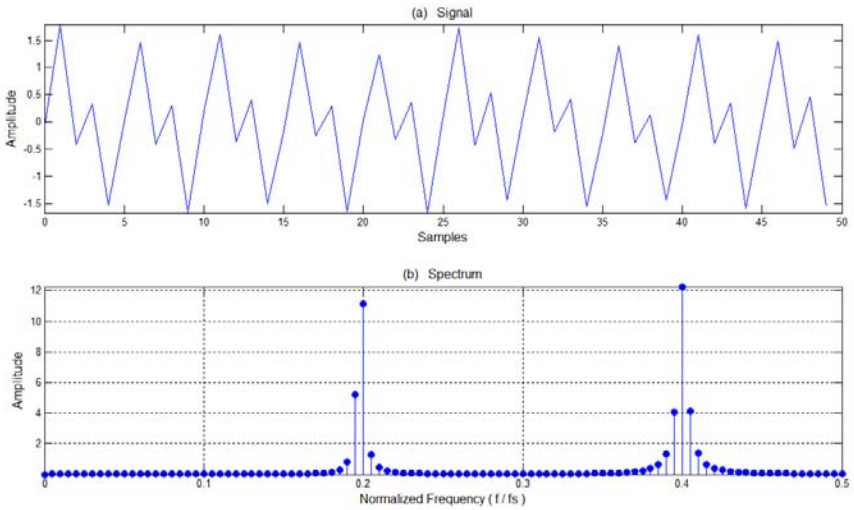

图 10-2　正反向线性预测最小二乘功率谱估计

10.3 特征分析谱估计方法

特征分析谱估计方法将随机信号的自相关矩阵分解为信号子空间和噪声子空间，然后利用二者的正交性估计随机信号的频率成分。该方法对被噪声污染的由几个正弦波叠加而成的信号非常有效，SP++提供了四类基于特征分析法的谱估计算法 ，如表 10-3所示。

表 10-3　特征分析谱估计方法

Operation	Effect
caponPSE (xn,M,L)	Capon 谱估计方法
musicPSE (xn,M,p,L)	MUSIC 谱估计方法
pisarenkoPSE (xn, M,p,L)	Pisarenko 谱估计方法
espritPSE (xn,M,p)	ESPRIT 谱估计方法

测试代码：

```
/*****************************************************************************
 *                            eigenanalysispse_test.cpp
 *
 * Eigenanalysis spectrum estimation testing.
 *
 * Zhang Ming, 2010-11, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <cstring>
#include <random.h>
#include <vectormath.h>
#include <eigenanalysispse.h>
#include "engine.h"


using namespace std;
using namespace splab;


typedef double  Type;
const   int     N = 200;
const   int     M = 20;
const   int     L = 200;


int main()
{
    /****************************** [ signal ] ******************************/
    cout << setiosflags(ios::fixed) << setprecision(4);
    int mfn = L/2+1;
    Type amp1 = Type(1.0),
         amp2 = Type(1.0);
    Type f1 = Type(0.2),
         f2 = Type(0.25);
    Type SNR;

    Vector<Type> tn = linspace(Type(0), Type(N-1), N );
    Vector<Type> sn = amp1*sin(Type(TWOPI)*f1*tn) + amp2*sin(Type(TWOPI)*f2*tn);
```

```
44.     Vector<Type> wn = randn( 37, Type(0.0), Type(0.2), N );
45.     Vector<Type> xn = sn + wn;
46.     SNR = 20*log10(norm(sn)/norm(wn));
47.     cout << "The SNR = " << SNR << endl << endl;
48.
49.     /******************************** [ PSD ] ******************************/
50.     int p = orderEst( xn, M );
51.     p = p/2*2;
52.     cout << "The model order that minimizes the MDL criterion is:   p = "
53.          << p << endl << endl;
54.
55. //    Vector<Type> Px = caponPSE( xn, M, L );
56.
57. //    Vector<Type> Px = pisarenkoPSE( xn, M, p, L );
58.
59. //    Vector<Type> Px = musicPSE( xn, M, p, L );
60.
61.     Vector<Type> fk = espritPSE( xn, M, p );
62.     Vector<Type> Px(mfn);
63.     for( int k=0; k<p; ++k )
64.     {
65.         int index = int( L*abs(fk[k]) + 0.5 );
66.         if( index != 0 && index != L/2 )
67.         Px[index] = Type(1.0);
68.     }
69.
70.     /******************************** [ PLOT ] ******************************/
71.     Engine *ep  = engOpen( NULL );
72.     if( !ep )
73.     {
74.         cerr << "Cannot open Matlab Engine!" << endl;
75.         exit(1);
76.     }
77.
78.     mxArray *mxn = mxCreateDoubleMatrix( N, 1, mxREAL );
79.     mxArray *mPx = mxCreateDoubleMatrix( mfn, 1, mxREAL );
80.     memcpy( mxGetPr(mxn), xn, N*sizeof(Type) );
81.     memcpy( mxGetPr(mPx), Px, mfn*sizeof(Type) );
82.     engPutVariable( ep, "xn", mxn );
83.     engPutVariable( ep, "Px", mPx );
84.
85.     const char *mCmd =  " figure('name','FBLPLS Method of Spectrum Estimation'); \

86.         N = length(xn); mfn = length(Px); \
```

```
        subplot(2,1,1); \
            plot((0:N-1), xn); \
            axis([0,N,min(xn),max(xn)]); \
            title('(a)   Signal', 'FontSize',12); \
            xlabel('Samples', 'FontSize',12); \
            ylabel('Amplitude', 'FontSize',12); \
        subplot(2,1,2); \
            h = stem((0:mfn-1)/(mfn-1)/2, Px); \
            axis([0,0.5,min(Px),max(Px)]); \
            set(h,'MarkerFaceColor','blue'); \
            set(gca, 'XTick', 0:0.05:0.5); \
            grid on; \
            title('(b)   Spectrum', 'FontSize',12); \
            xlabel('Normalized Frequency ( f / fs )', 'FontSize',12); \
            ylabel('Amplitude', 'FontSize',12); ";
    engEvalString( ep, mCmd );

    mxDestroyArray( mxn );
    mxDestroyArray( mPx );
    system( "pause" );
    engClose(ep);

    return 0;
}
```

运行结果如下及图 10-3所示：

```
The SNR = 14.3399

The model order that minimizes the MDL criterion is:   p = 4

请按任意键继续. . .
```

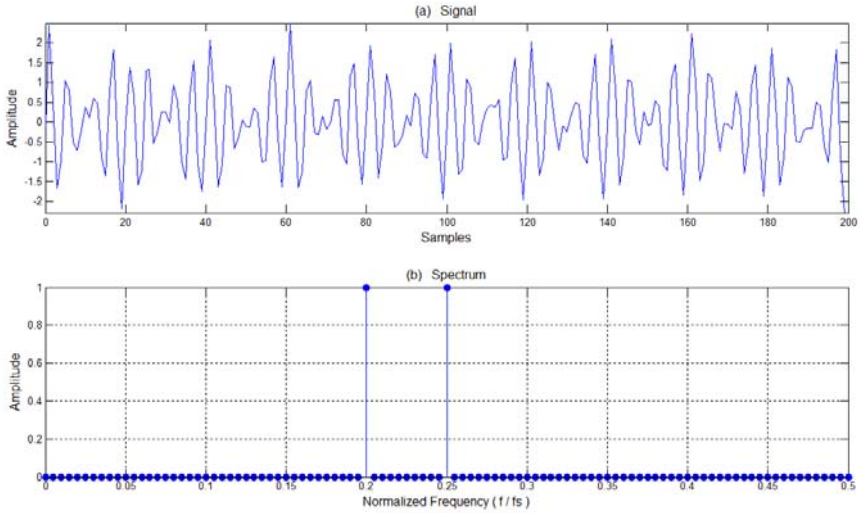

图 10-3　ESPRIT 功率谱估计

11 自适应滤波器

11.1 Wiener 滤波器

Wiener滤波器是最小均方误差意义下的最优线性滤波器，适用于平稳信号。可通过解救Wiener-Hopf方程（使用Levinson-Durbin快速递推算法）得到其时域解。在频域表现为：信噪比越大的频带衰减越小，信噪比越小的频带衰减越大，从而最大程度地滤除噪声，保留信号。SP++中提供了Wiener滤波器和Wiener预测器两种算法，同时包含了求解Toeplitz系数矩阵线性方程组的Levinson递推算法，如表 11-1所示。

表 11-1　Wiener 滤波器

Operation	Effect
wienerFilter (x,d,p)	P 阶 Wiener 滤波器
wienerPredictor (x,p)	P 阶 Wiener 预测器
levinson (t,b)	Levinson 算法求解 Toeplitz 方程组

测试代码：

```
1.  /*****************************************************************************
2.   *                                wiener_test.h
3.   *
4.   * Wiener filter testing.
5.   *
6.   * Zhang Ming, 2010-10, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <iomanip>
14. #include <wiener.h>
15. #include <random.h>
16. #include <vectormath.h>
17.
18.
19. using namespace std;
20. using namespace splab;
```

```
21.
22.
23. typedef double  Type;
24. const   int     N = 1024;
25. const   int     M = 12;
26. const   int     fOrder = 8;
27. const   int     pOrder = 3;
28.
29.
30. int main()
31. {
32.     Vector<Type> tn(N), dn(N), vn(N), xn(N), yn(N);
33.     tn = linspace( Type(0.0), Type(2*TWOPI), N );
34.     dn = sin(tn);
35.     vn = randn( 37, Type(0.0), Type(1.0), N );
36.     xn = dn + vn;
37.
38.     Vector<Type> hn = wienerFilter( xn, dn, fOrder );
39.     cout << "Wiener filter hn:   " << hn << endl;
40.     yn = wkeep( conv(xn,hn), N, "left" );
41.     cout << "original relative error:   " << norm(dn-xn)/norm(dn) << endl;
42.     cout << "filtered relative error:   " << norm(dn-yn)/norm(dn) << endl << endl;
43.
44.     Vector<Type> sn(M);
45.     for( int i=0; i<M; ++i )
46.         sn[i] = sin( Type(i*TWOPI/10) );
47.     Vector<Type> pn = wienerPredictor( sn, pOrder );
48.     Vector<Type> snPred = wkeep( conv(sn,pn), M, "left" );
49.     cout << "Wiener predictor pn:   " << pn << endl;
50.     cout << "original\tpredicted\terror" << endl;
51.     Type realValue = sin( Type(M*TWOPI/10) );
52.     cout << setiosflags(ios::fixed) << setprecision(4)
53.          << realValue << "\t\t" << snPred[M-1] << "\t\t"
54.          << abs(realValue-snPred[M-1]) << endl << endl;
55.
56.     return 0;
57. }
```

运行结果：

```
1.  Wiener filter hn:   size: 9 by 1
2.  0.0903079
3.  0.0858067
4.  0.0813221
```

```
0.0870775
0.0968708
0.0888659
0.0844214
0.0901495
0.0965093

original relative error:   1.43689
filtered relative error:   0.416294

Wiener predictor pn:   size: 3 by 1
0.606355
0.699992
-1.03413

original        predicted       error
0.9511          0.9643          0.0132


Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

11.2 Kalman 滤波器

Kalman滤波器同样是最小均方误差意义下的最优线性估计器，对平稳和非平衡过程均可适用，可以将其视为一种自适应的Wiener滤波器。其主要思想是利用“新息”与Kalman增益计算校正项，通过递推方法来估计线性动态系统的内部状态变量。SP++中提供了常规的Kalman滤波算法，如表 11-2所示。

表 11-2　Kalman 滤波器

Operation	Effect
kalman(A,C,Q,R,xp,y,V)	Kalman 滤波器, 详细说明参见代码注释

测试代码：

```
/*****************************************************************************
 *                                kalman.h
 *
 * Kalman filter testing.
 *
 * Zhang Ming, 2010-10, Xi'an Jiaotong University.
 *****************************************************************************/
```

```
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <kalman.h>
14.
15.
16. using namespace std;
17. using namespace splab;
18.
19.
20. typedef double  Type;
21. const   int     N = 2;
22. const   int     M = 2;
23. const   int     T = 20;
24.
25.
26. int main()
27. {
28.     Matrix<Type> A(N,N), C(M,N), Q(N,N), R(M,M);
29.     A = eye( N, Type(1.0) );    C = eye( N, Type(1.0) );
30.     Q = eye( N, Type(1.0) );    R = eye( N, Type(2.0) );
31.
32.     Vector<Type> x(N,Type(1.0)), y(M), ytInit(M);
33.     ytInit[0] = Type(0.5);  ytInit[1] = Type(2.0);
34.     Matrix<Type> yt(M,T);
35.     for( int t=0; t<T; ++t )
36.         yt.setColumn( ytInit, t );
37.
38.     Vector<Type> intV( N, Type(10.0) );
39.     for( int t=0; t<T; ++t )
40.     {
41.         y = yt.getColumn(t);
42.         x = kalman( A, C, Q, R, x, y, intV );
43.         cout << "Estimation of xt at the " << t << "th iteratin:   " << x << endl;

44.     }
45.
46.     cout << "The theoretical xt should converge to:   " << ytInit << endl;
47.
48.     return 0;
49. }
```

运行结果：

```
Estimation of xt at the 0th iteratin:   size: 2 by 1
0.576923
1.84615

Estimation of xt at the 1th iteratin:   size: 2 by 1
0.532787
1.93443

Estimation of xt at the 2th iteratin:   size: 2 by 1
0.51581
1.96838

Estimation of xt at the 3th iteratin:   size: 2 by 1
0.507835
1.98433

Estimation of xt at the 4th iteratin:   size: 2 by 1
0.503909
1.99218

Estimation of xt at the 5th iteratin:   size: 2 by 1
0.501953
1.99609

Estimation of xt at the 6th iteratin:   size: 2 by 1
0.500977
1.99805

Estimation of xt at the 7th iteratin:   size: 2 by 1
0.500488
1.99902

Estimation of xt at the 8th iteratin:   size: 2 by 1
0.500244
1.99951

Estimation of xt at the 9th iteratin:   size: 2 by 1
0.500122
1.99976

Estimation of xt at the 10th iteratin:   size: 2 by 1
0.500061
1.99988
```

```
44.
45. Estimation of xt at the 11th iteratin:   size: 2 by 1
46. 0.500031
47. 1.99994
48.
49. Estimation of xt at the 12th iteratin:   size: 2 by 1
50. 0.500015
51. 1.99997
52.
53. Estimation of xt at the 13th iteratin:   size: 2 by 1
54. 0.500008
55. 1.99998
56.
57. Estimation of xt at the 14th iteratin:   size: 2 by 1
58. 0.500004
59. 1.99999
60.
61. Estimation of xt at the 15th iteratin:   size: 2 by 1
62. 0.500002
63. 2
64.
65. Estimation of xt at the 16th iteratin:   size: 2 by 1
66. 0.500001
67. 2
68.
69. Estimation of xt at the 17th iteratin:   size: 2 by 1
70. 0.5
71. 2
72.
73. Estimation of xt at the 18th iteratin:   size: 2 by 1
74. 0.5
75. 2
76.
77. Estimation of xt at the 19th iteratin:   size: 2 by 1
78. 0.5
79. 2
80.
81. The theoretical xt should converge to:   size: 2 by 1
82. 0.5
83. 2
84.
85.
86. Process returned 0 (0x0)   execution time : 0.125 s
87. Press any key to continue.
```

11.3 LMS 自适应滤波器

LMS (Least Mean Squares) 是最常用的自适应滤波算法，它以当前的误差代替期望误差，通过梯度下降方法调整滤波器的系数，从而跟踪输入信号或系统的变化，达到自适应的目的。步长因子对算法性能有重要的影响，大的步长因子提供快速的跟踪特性，同时产生大的失调量以及不稳定性，反之易然。LMS有多种类型，SP++提供了 3 种常用的LMS算法，即普通LMS算法，LMS-Newton算法和归一化LMS算法，调用形式如表 11-3所示。

表 11-3 LMS 自适应滤波算法

Operation	Effect
lms (x,d,w,mu)	常规 LMS 算法
lmsNewton (x,d,w,mu,alpha,delta)	LMS-Newton 算法
lmsNormalize (x,d,w,rho,gamma)	归一化 LMS 算法

测试代码：

```
/*****************************************************************************
 *                                lms_test.cpp
 *
 * LMS adaptive filter testing.
 *
 * Zhang Ming, 2010-10, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <lms.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     N = 50;
const   int     order = 1;
const   int     dispNumber = 10;

```

```
26.
27. int main()
28. {
29.     Vector<Type> dn(N), xn(N), yn(N), wn(order+1);
30.     for( int k=0; k<N; ++k )
31.     {
32.         xn[k] = Type(cos(TWOPI*k/7));
33.         dn[k] = Type(sin(TWOPI*k/7));
34.     }
35.     int start = max(0,N-dispNumber);
36.     Type xnPow = dotProd(xn,xn)/N;
37.     Type mu = Type( 0.1 / ((order+1)*xnPow) );
38.     Type rho = Type(1.0), gamma = Type(1.0e-9), alpha = Type(0.08);
39.
40.     cout << "The last " << dispNumber
41.          << " iterations of Conventional-LMS:" << endl;
42.     cout << "observed" << "\t" << "desired" << "\t\t" << "output" << "\t\t"
43.          << "adaptive filter" << endl << endl;
44.     wn = 0;
45.     for( int k=0; k<start; ++k )
46.         yn[k] = lms( xn[k], dn[k], wn, mu );
47.     for( int k=start; k<N; ++k )
48.     {
49.         yn[k] = lms( xn[k], dn[k], wn, mu );
50.         cout << setiosflags(ios::fixed) << setprecision(4)
51.              << xn[k] << "\t\t" << dn[k] << "\t\t" << yn[k] << "\t\t";
52.         for( int i=0; i<=order; ++i )
53.             cout << wn[i] << "\t";
54.         cout << endl;
55.     }
56.     cout << endl << endl;
57.
58.     cout << "The last " << dispNumber << " iterations of LMS-Newton:" << endl;
59.     cout << "observed" << "\t" << "desired" << "\t\t" << "output" << "\t\t"
60.          << "adaptive filter" << endl << endl;
61.     wn = 0;
62.     for( int k=0; k<start; ++k )
63.         yn[k] = lmsNewton( xn[k], dn[k], wn, mu, alpha, xnPow );
64.     for( int k=start; k<N; ++k )
65.     {
66.         yn[k] = lmsNewton( xn[k], dn[k], wn, mu, alpha, xnPow );
67.         cout << setiosflags(ios::fixed) << setprecision(4)
68.              << xn[k] << "\t\t" << dn[k] << "\t\t" << yn[k] << "\t\t";
69.         for( int i=0; i<=order; ++i )
```

```
70.             cout << wn[i] << "\t";
71.         cout << endl;
72.     }
73.     cout << endl << endl;
74.
75.     cout << "The last " << dispNumber
76.          << " iterations of Normalized-LMS:" << endl;
77.     cout << "observed" << "\t" << "desired" << "\t\t" << "output" << "\t\t"
78.          << "adaptive filter" << endl << endl;
79.     wn = 0;
80.     for( int k=0; k<start; ++k )
81.         yn[k] = lmsNormalize( xn[k], dn[k], wn, rho, gamma );
82.     for( int k=start; k<N; ++k )
83.     {
84.         yn[k] = lmsNormalize( xn[k], dn[k], wn, rho, gamma );
85.         cout << setiosflags(ios::fixed) << setprecision(4)
86.              << xn[k] << "\t\t" << dn[k] << "\t\t" << yn[k] << "\t\t";
87.         for( int i=0; i<=order; ++i )
88.             cout << wn[i] << "\t";
89.         cout << endl;
90.     }
91.     cout << endl << endl;
92.
93.     cout << "The theoretical optimal filter is:\t\t" << "-0.7972\t1.2788"
94.          << endl << endl;
95.
96.     return 0;
97. }
```

运行结果：

```
1.  The last 10 iterations of Conventional-LMS:
2.  observed        desired         output          adaptive filter
3.
4.  -0.2225         -0.9749         -0.8216         -0.5683 1.0810
5.  0.6235          -0.7818         -0.5949         -0.5912 1.0892
6.  1.0000          -0.0000         0.0879          -0.6084 1.0784
7.  0.6235          0.7818          0.6991          -0.5983 1.0947
8.  -0.2225         0.9749          0.8156          -0.6053 1.1141
9.  -0.9010         0.4339          0.2974          -0.6294 1.1082
10. -0.9010         -0.4339         -0.4314         -0.6289 1.1086
11. -0.2225         -0.9749         -0.8589         -0.6239 1.1291
12. 0.6235          -0.7818         -0.6402         -0.6412 1.1353
13. 1.0000          -0.0000         0.0667          -0.6543 1.1271
14.
```

```
15.
16. The last 10 iterations of LMS-Newton:
17. observed        desired         output          adaptive filter
18.
19. -0.2225         -0.9749         -0.9739         -0.7958 1.2779
20. 0.6235          -0.7818         -0.7805         -0.7964 1.2783
21. 1.0000          -0.0000         0.0006          -0.7966 1.2783
22. 0.6235          0.7818          0.7816          -0.7966 1.2784
23. -0.2225         0.9749          0.9743          -0.7968 1.2787
24. -0.9010         0.4339          0.4334          -0.7971 1.2787
25. -0.9010         -0.4339         -0.4340         -0.7971 1.2787
26. -0.2225         -0.9749         -0.9747         -0.7971 1.2788
27. 0.6235          -0.7818         -0.7816         -0.7972 1.2789
28. 1.0000          -0.0000         0.0001          -0.7973 1.2789
29.
30.
31. The last 10 iterations of Normalized-LMS:
32. observed        desired         output          adaptive filter
33.
34. -0.2225         -0.9749         -0.9749         -0.7975 1.2790
35. 0.6235          -0.7818         -0.7818         -0.7975 1.2790
36. 1.0000          -0.0000         -0.0000         -0.7975 1.2790
37. 0.6235          0.7818          0.7818          -0.7975 1.2790
38. -0.2225         0.9749          0.9749          -0.7975 1.2790
39. -0.9010         0.4339          0.4339          -0.7975 1.2790
40. -0.9010         -0.4339         -0.4339         -0.7975 1.2790
41. -0.2225         -0.9749         -0.9749         -0.7975 1.2790
42. 0.6235          -0.7818         -0.7818         -0.7975 1.2790
43. 1.0000          -0.0000         -0.0000         -0.7975 1.2790
44.
45.
46. The theoretical optimal filter is:              -0.7972 1.2788
47.
48.
49. Process returned 0 (0x0)   execution time : 0.062 s
50. Press any key to continue.
```

11.4 RLS 自适应滤波器

RLS (Recursive Least Square) 算法与LMS算法不同，不是最小均方意义下的自适应滤波器，而是通过递归的方法求解以误差信号为代价函数的加权最小二乘问题，从而更新滤波器的系数。RLS算法的最大优点是收敛速度快，稳定性好，但这是以

计算量大和跟踪性能差为代价的(当然这可以通过遗忘因子进行调整，遗忘因子越小，收敛速度越快，跟踪能力越好，但失调量越大，稳定性越差)。SP++中提供了 5 类RLS自适应滤波算法，传统RLS算法，稳定快速横向RLS算法，普通格型RLS算法，误差反馈格型RLS算法和基于QR分解的RLS算法，如表 11-4所示，参数的具体意义可参考代码注释。

表 11-4 RLS 自适应滤波算法

Operation	Effect
rls (x,d,w,lambda,delta)	传统 RLS 算法
sftrls (x,d,w,lambda,epsilon,training)	稳定快速横向 RLS 算法
lrls (x,d,v,lambda,epsilon,training)	普通格型 RLS 算法
eflrls (x,d,v, lambda,epsilon,training)	误差反馈格型 RLS 算法
qrlms (x,d,w, lambdaSqrt,training)	基于 QR 分解的 RLS 算法

测试代码：

```
/*****************************************************************************
 *                                  rls_test.cpp
 *
 * RLS adaptive filter testing.
 *
 * Zhang Ming, 2010-10, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <convolution.h>
#include <vectormath.h>
#include <random.h>
#include <rls.h>


using namespace std;
using namespace splab;


typedef float   Type;
const   int     N = 1000;
const   int     orderRls = 1;
const   int     orderLrls = 16;
const   int     orderTrls = 12;
```

```
29. const   int     orderQrrls = 8;
30. const   int     sysLen = 8;
31. const   int     dispNumber = 10;
32.
33.
34. int main()
35. {
36.     int start = max(0,N-dispNumber);
37.     Vector<Type>    dn(N), xn(N), yn(N), sn(N), rn(N), en(N),
38.                     hn(sysLen+1), gn(orderLrls+1), wn(orderRls+1);
39.     Type lambda, delta, eps;
40.
41.
42.     cout << "/**************  Conventional RLS  <--->  Waveform Tracking \
43. *************/" << endl << endl;
44.     for( int k=0; k<N; ++k )
45.     {
46.         dn[k] = Type(sin(TWOPI*k/7));
47.         xn[k] = Type(cos(TWOPI*k/7));
48.     }
49.     lambda = Type(0.99);
50.     delta = dotProd(xn,xn)/N;
51.
52.     cout << "The last " << dispNumber << " iterations result:" << endl << endl;
53.     cout << "observed" << "\t" << "desired" << "\t\t" << "output" << "\t\t"
54.          << "adaptive filter" << endl << endl;
55.     for( int k=0; k<start; ++k )
56.         yn[k] = rls( xn[k], dn[k], wn, lambda, delta );
57.     for( int k=start; k<N; ++k )
58.     {
59.         yn[k] = rls( xn[k], dn[k], wn, lambda, delta );
60.         cout << setiosflags(ios::fixed) << setprecision(4)
61.              << xn[k] << "\t\t" << dn[k] << "\t\t" << yn[k] << "\t\t";
62.         for( int i=0; i<=orderRls; ++i )
63.             cout << wn[i] << "\t";
64.         cout << endl;
65.     }
66.     cout << endl << "The theoretical optimal filter is:\t\t" << "-0.7972\t1.2788"

67.          << endl << endl << endl;
68.
69.
70.     cout << "/**************  Lattice RLS  <--->  Channel Equalization \
71. ***************/" << endl << endl;
```

```
72.     for( int k=0; k<=sysLen; ++k )
73.         hn[k] = Type( 0.1 * pow(0.5,k) );
74.     dn = randn( 37, Type(0.0), Type(1.0), N );
75.     xn = wkeep( conv(dn,hn), N, "left" );
76.     lambda = Type(0.99), eps = Type(0.1);
77.
78.     wn.resize(orderLrls);
79.     wn = Type(0.0);
80.     for( int k=0; k<N; ++k )
81. //        yn[k] = lrls( xn[k], dn[k], wn, lambda, eps, "on" );
82.         yn[k] = eflrls( xn[k], dn[k], wn, lambda, eps, "on" );
83.     Vector<Type> Delta(orderLrls+1);
84.     Delta[(orderLrls+1)/2] = Type(1.0);
85.     for( int k=0; k<=orderLrls; ++k )
86. //        gn[k] = lrls( Delta[k], Type(0.0), wn, lambda, eps, "off" );
87.         gn[k] = eflrls( Delta[k], Type(0.0), wn, lambda, eps, "off" );
88.     cout << setiosflags(ios::fixed) << setprecision(4);
89.     cout << "The original system:   " << hn << endl;
90.     cout << "The inverse system:   " << gn << endl;
91.     cout << "The cascade system:   " << conv( gn, hn ) << endl << endl;
92. //
93.
94.     cout << "/************  Transversal RLS  <--->  System Identification \
95. ************/" << endl << endl;
96.     Vector<Type> sys(8);
97.     sys[0] = Type(0.1);   sys[1] = Type(0.3);
98.     sys[2] = Type(0.0);   sys[3] = Type(-0.2);
99.     sys[4] = Type(-0.4);  sys[5] = Type(-0.7);
100.    sys[6] = Type(-0.4);  sys[7] = Type(-0.2);
101.    xn = randn( 37, Type(0.0), Type(1.0), N );
102.    dn = wkeep( conv(xn,sys), N, "left" );
103.    lambda = Type(0.99), eps = Type(1.0);
104.    wn.resize(orderTrls);
105.    wn = Type(0.0);
106.
107.    for( int k=0; k<start; ++k )
108.        yn[k] = sftrls( xn[k], dn[k], wn, lambda, eps, "on" );
109.    cout << "The last " << dispNumber << " iterations result:" << endl << endl;
110.    cout << "input signal" << "    " << "original system output" << "    "
111.         << "identified system output" << endl << endl;
112.    for( int k=start; k<N; ++k )
113.    {
114.        yn[k] = sftrls( xn[k], Type(0.0), wn, lambda, eps, "off" );
115.        cout << setiosflags(ios::fixed) << setprecision(4)
```

```
116.             << xn[k] << "\t\t\t" << dn[k] << "\t\t\t" << yn[k] << endl;
117.     }
118.     cout << endl << "The unit impulse response of original system:   "
119.          << sys << endl;
120.     cout << "The unit impulse response of identified system:   "
121.          << wn << endl << endl;
122.
123.
124.     cout << "/***************  QR Based RLS  <--->  Signal Enhancement \
125. ***************/" << endl << endl;
126.     sn = sin( linspace( Type(0.0), Type(4*TWOPI), N ) );
127.     rn = randn( 37, Type(0.0), Type(1.0), N );
128.     dn = sn + rn;
129.     int delay = orderQrrls/2;
130.     for( int i=0; i<N-delay; ++i )
131.         xn[i] = rn[i+delay];
132.     for( int i=N-delay; i<N; ++i )
133.         xn[i] = 0;
134.     Type lambdaSqrt = Type(1.0);
135.     wn.resize(orderQrrls);
136.     wn = Type(0.0);
137.
138.     for( int k=0; k<N; ++k )
139.         yn[k] = qrrls( xn[k], dn[k], wn, lambdaSqrt, "on" );
140.     en = dn - yn;
141.     for( int i=0; i<N/2; ++i )
142.     {
143.         sn[i] = 0;
144.         rn[i] = 0;
145.         en[i] = 0;
146.     }
147.     cout << "The last " << dispNumber << " iterations result:" << endl << endl;
148.     cout << "noised signal\t\t" << "enhanced signal\t\t" << "original signal"
149.          << endl << endl;
150.     for( int k=start; k<N; ++k )
151.         cout << setiosflags(ios::fixed) << setprecision(4)
152.              << dn[k] << "\t\t\t" << en[k] << "\t\t\t" << sn[k] << endl;
153.     cout << endl << "The SNR before denoising is:   "
154.          << 20*(log10(norm(sn))/log10(norm(rn))) << " dB" << endl;
155.     cout << "The SNR after denoising is:    "
156.          << 20*(log10(norm(en))/log10(norm(sn-en))) << " dB" << endl << endl;
157.
158.
159.     return 0;
```

```
}
```

运行结果：

```
/**************  Conventional RLS  <--->  Waveform Tracking *************/

The last 10 iterations result:

observed        desired         output          adaptive filter

-0.9010         0.4339          0.4339          -0.7975 1.2790
-0.9010         -0.4339         -0.4339         -0.7975 1.2790
-0.2225         -0.9749         -0.9749         -0.7975 1.2790
0.6235          -0.7818         -0.7818         -0.7975 1.2790
1.0000          0.0000          0.0000          -0.7975 1.2790
0.6235          0.7818          0.7818          -0.7975 1.2790
-0.2225         0.9749          0.9749          -0.7975 1.2790
-0.9010         0.4339          0.4339          -0.7975 1.2790
-0.9010         -0.4339         -0.4339         -0.7975 1.2790
-0.2225         -0.9749         -0.9749         -0.7975 1.2790

The theoretical optimal filter is:              -0.7972 1.2788


/**************  Lattice RLS  <--->  Channel Equalization ***************/

The original system:   size: 9 by 1
0.1000
0.0500
0.0250
0.0125
0.0063
0.0031
0.0016
0.0008
0.0004

The inverse system:   size: 17 by 1
0.6038
-0.0026
-0.0018
-0.0012
0.0008
0.0011
0.0011
```

```
-0.0009
8.7646
-5.0002
0.0000
0.0015
-0.0007
-0.0004
0.0014
0.0027
0.0048

The cascade system:   size: 25 by 1
0.0604
0.0299
0.0148
0.0073
0.0037
0.0020
0.0011
0.0005
0.8767
-0.0618
-0.0309
-0.0153
-0.0077
-0.0039
-0.0018
-0.0006
0.0002
-0.0016
0.0002
0.0001
0.0000
0.0000
0.0000
0.0000
0.0000


/************  Transversal RLS  <--->  System Identification ************/

The last 10 iterations result:

input signal    original system output    identified system output
```

```
86.
87.  1.5173                  -1.7206                  -1.7206
88.  -0.9741                 -1.3146                  -1.3146
89.  -1.4056                 -2.1204                  -2.1204
90.  -0.9319                 -2.3165                  -2.3165
91.  -0.5763                 -2.1748                  -2.1748
92.  0.4665                  -1.2228                  -1.2228
93.  0.5959                  0.7623                   0.7623
94.  0.5354                  1.7904                   1.7904
95.  -0.4990                 1.6573                   1.6573
96.  -1.1519                 0.4866                   0.4866
97.
98.  The unit impulse response of original system:   size: 8 by 1
99.  0.1000
100. 0.3000
101. 0.0000
102. -0.2000
103. -0.4000
104. -0.7000
105. -0.4000
106. -0.2000
107.
108. The unit impulse response of identified system:   size: 12 by 1
109. 0.1000
110. 0.3000
111. 0.0000
112. -0.2000
113. -0.4000
114. -0.7000
115. -0.4000
116. -0.2000
117. 0.0000
118. 0.0000
119. -0.0000
120. -0.0000
121.
122.
123. /***************  QR Based RLS  <--->  Signal Enhancement  ***************/
124.
125. The last 10 iterations result:
126.
127. noised signal           enhanced signal         original signal
128.
129. 1.2928                  -0.2263                  -0.2245
```

```
-1.1740                 -0.1998                 -0.1999
-1.5808                 -0.1748                 -0.1752
-1.0823                 -0.1469                 -0.1504
-0.7017                 -0.1039                 -0.1255
0.3660                  -0.0745                 -0.1005
0.5205                  -0.0628                 -0.0754
0.4851                  -0.0439                 -0.0503
-0.5242                 -0.0220                 -0.0252
-1.1519                 0.0058                  0.0000

The SNR before denoising is:   17.5084 dB
The SNR after denoising is:    121.1680 dB


Process returned 0 (0x0)   execution time : 0.140 s
Press any key to continue.
```

12 时频分析

12.1 加窗 Fourier 变换

加窗 Fourier 变换（WFT）或短时 Fourier 变换（STFT）由于其思想直观，实现简单，成为了最常用的时频分析方法之一。WFT 通过对信号与时频原子作内积，将信号从时域信号变换到时频域，实现了时频局部化的分析特性，克服了 Fourier 变换全局性的缺点。

时频原子是通过对基本窗函数进行时移和调制所得，所以窗函数的选择对时频变换结果影响非常大，通常选用时频聚集性好的窗函数，如Gauss窗或Hamming等等。WFT的正反变换调用格式见表 12-1，其中带后缀“FFTW”的函数表示使用FFTW库计算DFT，使用这些函数需要安装FFTW。

表 12-1　加窗 Fourier 变换

Operation	Effect
wft(sn,gn,mod)	计算 sn 的加窗 Fourier 变换系数
iwft(coefs,gn)	计算 coefs 的加窗 Fourier 逆变换
wftFFTW(sn,gn,mod)	计算 sn 的加窗 Fourier 变换系数
iwftFFTW(coefs,gn)	计算 coefs 的加窗 Fourier 逆变换

测试代码：

```
1.  /**************************************************************************
2.   *                              wft_test.cpp
3.   *
4.   * Windowed Fourier transform testing.
5.   *
6.   * Zhang Ming, 2010-03, Xi'an Jiaotong University.
7.   *************************************************************************/
8.
9.
10. #define BOUNDS_CHECK
11.
12. #include <iostream>
13. #include <vectormath.h>
14. #include <timing.h>
15. #include <wft.h>
16.
```

```
17.
18. using namespace std;
19. using namespace splab;
20.
21.
22. typedef long double Type;
23. const   int         Lg = 100;
24. const   int         Ls = 1000;
25. const   int         Fs = 1000;
26.
27.
28. int main()
29. {
30.     /****************************** [ signal ] ******************************/
31.     Type a = 0;
32.     Type b = Ls-1;
33.     Vector<Type> t = linspace(a,b,Ls) / Type(Fs);
34.     Vector<Type> s = sin( Type(400*PI) * pow(t,Type(2.0)) );
35.
36.     /****************************** [ widow ] *******************************/
37.     a = 0;
38.     b = Type(Lg-1);
39.     Type u = (Lg-1)/Type(2);
40.     Type r = Lg/Type(8);
41.     t = linspace(a,b,Lg);
42.     Vector<Type> g = gauss(t,u,r);
43.     g = g/norm(g);
44.
45.     /****************************** [ WFT ] *******************************/
46.     Type runtime = 0;
47.     Timing cnt;
48.     cout << "Taking windowed Fourier transform." << endl;
49.     cnt.start();
50.     Matrix< complex<Type> > coefs = wft( s, g );
51.     cnt.stop();
52.     runtime = cnt.read();
53.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
54.
55.     /****************************** [ IWFT ] ******************************/
56.     cout << "Taking inverse windowed Fourier transform." << endl;
57.     cnt.start();
58.     Vector<Type> x = iwft( coefs, g );
59.     cnt.stop();
60.     runtime = cnt.read();
```

```
61.      cout << "The running time = " << runtime << " (ms)" << endl << endl;
62.
63.      cout << "The relative error is : " << "norm(s-x) / norm(s) = "
64.           << norm(s-x)/norm(s) << endl << endl;
65.
66.      return 0;
67. }
```

运行结果：

```
1.  Taking windowed Fourier transform.
2.  The running time = 0.031 (ms)
3.
4.  Taking inverse windowed Fourier transform.
5.  The running time = 0.047 (ms)
6.
7.  The relative error is : norm(s-x) / norm(s) = 7.013e-020
8.
9.
10. Process returned 0 (0x0)   execution time : 0.094 s
11. Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 12-1所示。

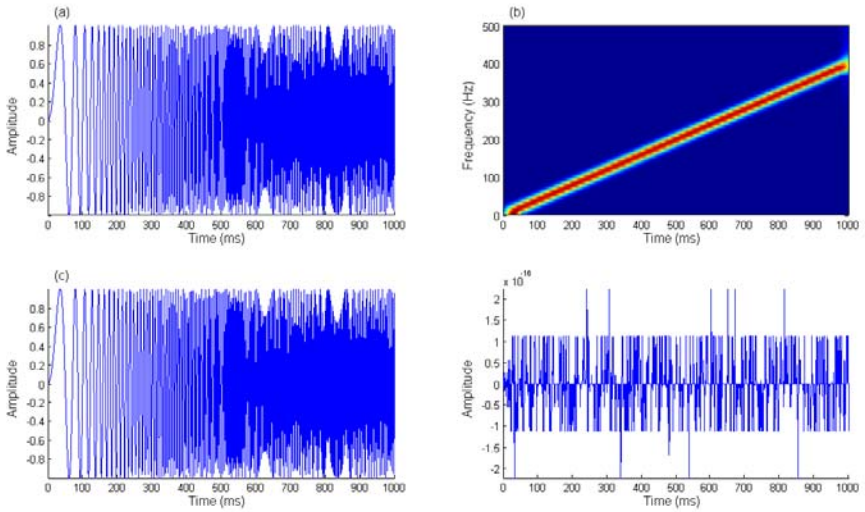

图 12-1　加窗 Fourier 变换

12.2 离散 Gabor 变换

离散 Gabor 变换（DGT）是用 Gabor 分析标架与综合标架对信号进行时频分解

与综合，DGT 可以看作是 WFT 在时频域的一种采样。其冗余度可能通过过采样率进行调整，并且同样可以借助 FFT 算法实现，因此计算效率相对 WFT 得到很大的提高。

DGT是一种标架运算，故其分析窗函数与综合窗函数一般是不相同的，而是要通过对偶窗函数进行计算。“dgt.h”头文件中提供了对偶函数以及离散Gabor正反变换的计算，具体见表 12-2，其中带后缀“FFTW”的函数表示使用FFTW库计算DFT，使用这些函数需要安装FFTW。

表 12-2　离散 Gabor 变换

Operation	Effect
dual(gn,N,dM)	计算 gn 的对偶窗函数
dgt(sn,gn,N,dM,mode)	计算 sn 的离散 Gabor 变换
idgt(coefs,gn,N,dM)	计算 coefs 的离散 Gabor 逆变换
dualFFTW(gn,N,dM)	计算 gn 的对偶窗函数
dgFFTWt(sn,gn,N,dM,mode)	计算 sn 的离散 Gabor 变换
idgtFFTW (coefs,gn,N,dM)	计算 coefs 的离散 Gabor 逆变换

测试代码：

```
/*****************************************************************************
 *                                dgt_test.cpp
 *
 * Discrete Gabor transform testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <vectormath.h>
#include <timing.h>
#include <dgt.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     Fs = 1000;
const   int     Ls = 10000;
const   int     Lg = 80;
```

```
26. const   int     N  = 40;
27. const   int     dM = 10;      // over sampling ratio is N/dM
28.
29.
30. int main()
31. {
32.
33.     /****************************** [ signal ] ******************************/
34.     Type a = 0;
35.     Type b = Ls-1;
36.     Vector<Type> t = linspace( a, b, Ls ) / Type(Fs);
37.     Vector<Type> st = cos( Type(400*PI) * pow(t,Type(2.0)) );
38.
39.     /****************************** [ widow ] ******************************/
40.     a = 0;
41.     b = Type( Lg-1 );
42.     Type r = sqrt( dM*N / Type(2*PI) );
43.     Type u = (Lg-1) / Type(2.0);
44.     t = linspace( a, b, Lg );
45.     Vector<Type> h = gauss( t, u, r );
46.     h = h / norm(h);
47.
48.     /***************************** [ daul function ] ************************/
49.     Type runtime = 0.0;
50.     Timing cnt;
51.     cout << "Compute daul function." << endl;
52.     cnt.start();
53.     Vector<Type> g = daul( h, N, dM );
54.     cnt.stop();
55.     runtime = cnt.read();
56.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
57.
58.     /******************************** [ DGT ] *******************************/
59.     cout << "Taking discrete Gabor transform." << endl;
60.     cnt.start();
61.     Matrix< complex<Type> > C = dgt( st, g, N, dM, "sym" );
62.     cnt.stop();
63.     runtime = cnt.read();
64.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
65.
66.     /******************************** [ IDGT ] *******************************/
67.     cout << "Taking inverse discrete Gabor transform." << endl;
68.     cnt.start();
69.     Vector<Type> xt = idgt( C, h, N, dM );
```

```
70.      cnt.stop();
71.      runtime = cnt.read();
72.      cout << "The running time = " << runtime << " (ms)" << endl << endl;
73.
74.      cout <<"The relative error : norm(s-x) / norm(s) = "
75.           << norm(st-xt)/norm(st) << endl << endl;
76.
77.      return 0;
78. }
```

运行结果：

```
1.  Compute daul function.
2.  The running time = 8.67362e-019 (ms)
3.
4.  Taking discrete Gabor transform.
5.  The running time = 0.063 (ms)
6.
7.  Taking inverse discrete Gabor transform.
8.  The running time = 0.031 (ms)
9.
10. The relative error : norm(s-x) / norm(s) = 5.79937e-012
11.
12.
13. Process returned 0 (0x0)   execution time : 0.109 s
14. Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 12-2与图 12-3所示。

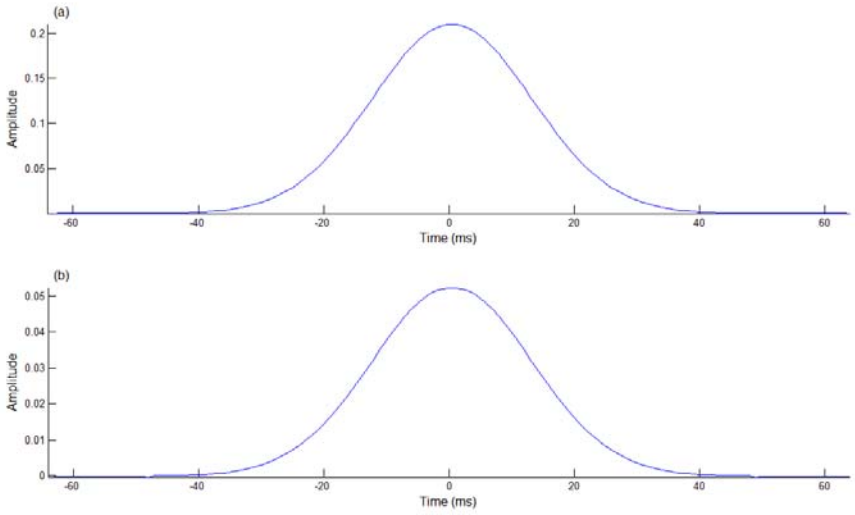

图 12-2　对偶函数

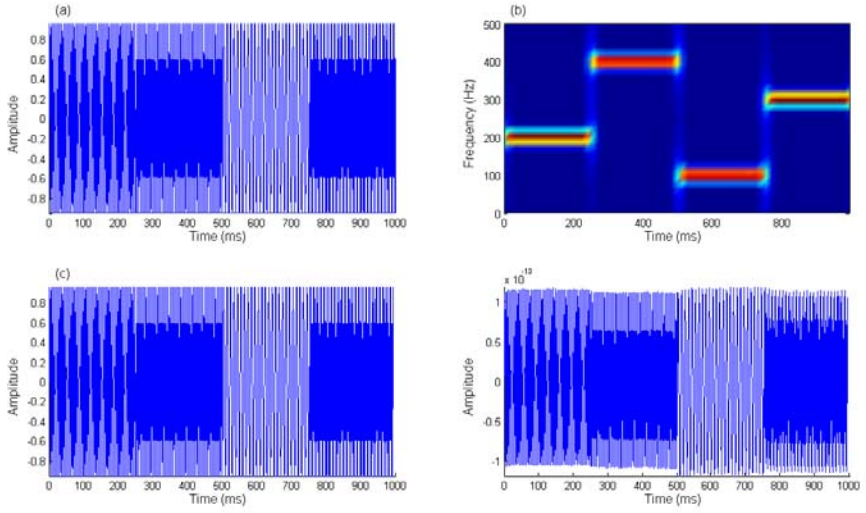

图 12-3　离散 Gabor 变换

12.3 Wigner-Wille 分布

加窗 Fourier 变换与小波变换都是用一族时频原子与信号做内积，因此它们的时频分辨率由于时频原子而受到限制。因此人们试图定义一种没有时频分辨率损失的参量分布，即这样的分布仅依赖于信号本身，并不受限于某一时频原子族。Wigner-Ville 分布（WVD）正是一种没有时频分辨率损失的时频能量人分布，它是通过信号本身的时间与频率平移计算所得。

WVD有许多优点，如WVD是一种时值分布，满足时间边沿条件和频率边沿条件，具有瞬时频率与群时延特性、时移与调制特性等等。但WVD有个致命缺点，既交叉项干扰。SP++中提供了实信号与复信号的WVD，调用格式相同，如表 12-3所示。

表 12-3　Wigner-Wille 分布

Operation	Effect
wvd(sn)	计算 sn 的 Wigner-Ville 分布

测试代码：

```
1.  /**************************************************************************
2.   *                              wvd_test.cpp
3.   *
4.   * Wigner-Ville distribution testing.
5.   *
6.   * Zhang Ming, 2010-10, Xi'an Jiaotong University.
7.   *************************************************************************/
8.
9.
```

```
#define BOUNDS_CHECK

#include <iostream>
#include <iomanip>
#include <timing.h>
#include <wvd.h>


using namespace std;
using namespace splab;


typedef double  Type;
const   int     Ls = 100;


int main()
{
    /****************************** [ signal ] ******************************/
    Vector<Type> t = linspace( Type(-0.5), Type(0.5-1.0/Ls), Ls );
    Vector< complex<Type> > sn(Ls);
    for( int i=0; i<Ls; ++i )
    {
        Type tip2 = t[i]*t[i],
             freq = Type(TWOPI*25);
        sn[i] = exp(-32*tip2) * polar(Type(1),freq*t[i]) * polar(Type(1),freq*tip2
);
    }
    cout << setiosflags(ios::fixed) << setprecision(4);

    /******************************** [ WVD ] ********************************/
    Type runtime = 0;
    Timing cnt;
    cout << "Computing Wigner-Wille distribution." << endl << endl;
    cnt.start();
    Matrix<Type> coefs = wvd( sn );
    cnt.stop();
    runtime = cnt.read();
    cout << "The running time = " << runtime << " (ms)" << endl << endl;

    Vector<Type> timeMarg = mean(coefs),
                 freqMarg = mean(trT(coefs));
    cout << "The time marginal condition is: " << timeMarg << endl;
    cout << "The frequency marginal condition is: " << freqMarg << endl;
```

```
53.
54.     return 0;
55. }
```

运行结果：

```
1.  Computing Wigner-Ville distribution.
2.
3.  The running time = 0.0000 (ms)
4.
5.  The time marginal condition is: size: 100 by 1
6.  0.0000
7.  0.0000
8.  0.0000
9.  0.0000
10. 0.0000
11. 0.0000
12. 0.0000
13. 0.0000
14. 0.0000
15. 0.0000
16. 0.0000
17. 0.0001
18. 0.0001
19. 0.0002
20. 0.0003
21. 0.0005
22. 0.0008
23. 0.0012
24. 0.0017
25. 0.0026
26. 0.0038
27. 0.0055
28. 0.0079
29. 0.0112
30. 0.0156
31. 0.0214
32. 0.0292
33. 0.0392
34. 0.0519
35. 0.0679
36. 0.0877
37. 0.1118
38. 0.1409
39. 0.1751
```

```
0.2148
0.2604
0.3115
0.3678
0.4290
0.4938
0.5612
0.6298
0.6978
0.7630
0.8239
0.8785
0.9246
0.9607
0.9857
0.9985
0.9984
0.9856
0.9608
0.9247
0.8784
0.8239
0.7630
0.6978
0.6298
0.5612
0.4937
0.4290
0.3679
0.3115
0.2604
0.2148
0.1751
0.1409
0.1119
0.0877
0.0679
0.0519
0.0391
0.0292
0.0215
0.0156
0.0112
0.0079
```

```
0.0055
0.0038
0.0026
0.0017
0.0012
0.0008
0.0005
0.0003
0.0002
0.0001
0.0001
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000

The frequency marginal condition is: size: 100 by 1
0.0000
0.0000
0.0000
0.0000
0.0000
0.0001
0.0003
0.0007
0.0016
0.0036
0.0078
0.0158
0.0308
0.0569
0.1001
0.1678
0.2677
0.4064
0.5877
0.8089
```

```
1.0600
1.3226
1.5707
1.7763
1.9121
1.9598
1.9121
1.7763
1.5707
1.3226
1.0600
0.8089
0.5877
0.4064
0.2677
0.1678
0.1001
0.0569
0.0308
0.0158
0.0078
0.0036
0.0016
0.0007
0.0003
0.0001
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
```

```
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000
0.0000


Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 12-4与图 12-5所示。

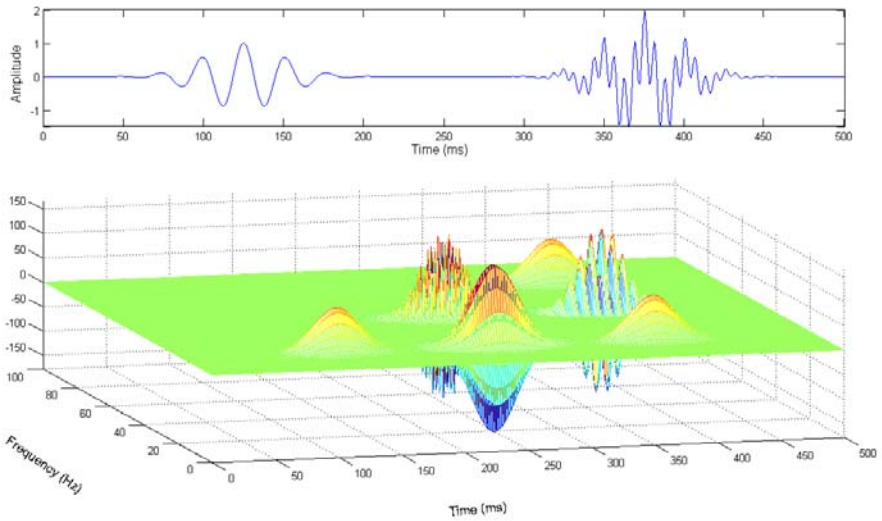

图 12-4　Wigner-Wille 分布

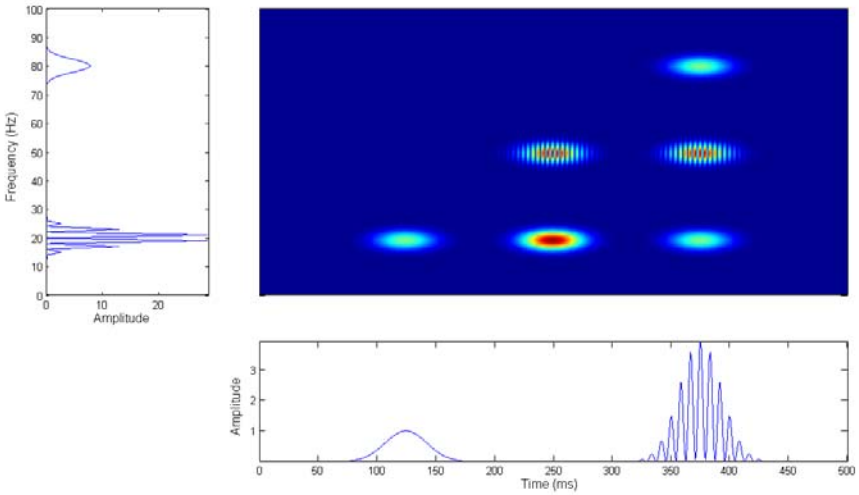

图 12-5　Wigner-Wille 分布

13 小波变换

13.1 连续小波变换

小波分析自上世纪 80 年代以来得到了飞速发展，并且在地震信号处理，生物医学信号处理，图像压缩与编码以及机械故障诊断等领域中取得了非常成功的应用。现已成为了最常用的一种信号分析手段。

连续小波变换（CWT）具有很高的冗余度，但其所含信息丰富，经常用于信号的属性提取；同时CWT具有很好的稳定性，在去噪中也得到了广泛使用。SP++中提供了实连续小波变换（以Mexico帽小波为例）和复连续小波变换（以Morlet小波为例）的正反变换程序，调用格式见表 13-1，其中带后缀“FFTW”的函数表示使用FFTW库计算DFT，使用这些函数需要安装FFTW。逆变换的重构精度可以通过尺度参数进行调整，精度越高，计算量就越大，默认参数的重构误差大致在 10^{-4} 至 10^{-8} 之间。

表 13-1　连续小波变换

Operation	Effect
CWT<Type> wt(wname)	创建 CWT 类
wt.~CWT<Type>()	析构 CWT 类
CWTFFTW<Type> wt(wname)	创建 CWTFFTW 类
wt.~ CWTFFTW<Type>()	析构 CWTFFTW 类
wt.setScales(fs,fmin,fmax,dj)	设置尺度参数
wt.cwrR(sn)	计算 sn 的实小波变换
wt.icwrR(coefs)	计算 coefs 的实小波逆变换
wt.cwrC(sn)	计算 sn 的复小波变换
wt.icwrC(coefs)	计算 coefs 的复小波逆变换

测试代码：

```
1.  /****************************************************************************
2.   *                              cwt_test.cpp
3.   *
4.   * Continuous wavelet transform testing.
5.   *
6.   * Zhang Ming, 2010-03, Xi'an Jiaotong University.
7.   ****************************************************************************/
8.
9.
10.  #define BOUNDS_CHECK
```

```
11.
12. #include <iostream>
13. #include <vectormath.h>
14. #include <statistics.h>
15. #include <timing.h>
16. #include <cwt.h>
17.
18.
19. using namespace std;
20. using namespace splab;
21.
22.
23. const   int     Ls = 1000;
24. const   double  fs = 1000.0;
25.
26.
27. int main()
28. {
29.
30.     /****************************** [ signal ] ******************************/
31.     Vector<double> t = linspace( 0.0, (Ls-1)/fs, Ls );
32.     Vector<double> st = sin( 200*PI*pow(t,2.0) );
33.     st = st-mean(st);
34.
35.     /****************************** [ CWT ] ******************************/
36.     Matrix< complex<double> > coefs;
37.     CWT<double> wavelet("morlet");
38.     wavelet.setScales( fs, fs/Ls, fs/2 );
39.     Timing cnt;
40.     double runtime = 0.0;
41.     cout << "Taking continuous wavelet transform(Morlet)." << endl;
42.     cnt.start();
43.     coefs = wavelet.cwtC(st);
44.     cnt.stop();
45.     runtime = cnt.read();
46.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
47.
48.     /****************************** [ ICWT ] ******************************/
49.     cout << "Taking inverse continuous wavelet transform." << endl;
50.     cnt.start();
51.     Vector<double> xt = wavelet.icwtC(coefs);
52.     cnt.stop();
53.     runtime = cnt.read();
54.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
```

```
55.
56.     cout << "The relative error is : " << endl;
57.     cout << "norm(st-xt) / norm(st) = " << norm(st-xt)/norm(st) << endl;
58.     cout << endl << endl;
59.
60.
61.     /****************************** [ signal ] ******************************/
62.     Vector<float> tf = linspace( float(0.0), (Ls-1)/float(fs), Ls );
63.     Vector<float> stf = sin( float(200*PI) * pow(tf,float(2.0) ) );
64.     stf = stf-mean(stf);
65.
66.     /****************************** [ CWT ] ******************************/
67.     CWT<float> waveletf("mexiHat");
68.     waveletf.setScales( float(fs), float(fs/Ls), float(fs/2), float(0.25) );
69.     runtime = 0.0;
70.     cout << "Taking continuous wavelet transform(Mexican Hat)." << endl;
71.     cnt.start();
72.     Matrix<float> coefsf = waveletf.cwtR(stf);
73.     cnt.stop();
74.     runtime = cnt.read();
75.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
76.
77.     /****************************** [ ICWT ] ******************************/
78.     cout << "Taking inverse continuous wavelet transform." << endl;
79.     cnt.start();
80.     Vector<float> xtf = waveletf.icwtR(coefsf);
81.     cnt.stop();
82.     runtime = cnt.read();
83.     cout << "The running time = " << runtime << " (ms)" << endl << endl;
84.
85.     cout << "The relative error is : " << endl;
86.     cout << "norm(st-xt) / norm(st) = " << norm(stf-xtf)/norm(stf) << endl << endl
    ;
87.
88.     return 0;
89. }
```

运行结果：

```
1.  Taking continuous wavelet transform(Morlet).
2.  The running time = 0.034 (ms)
3.
4.  Taking inverse continuous wavelet transform.
5.  The running time = 0.004 (ms)
6.
```

```
The relative error is :
norm(st-xt) / norm(st) = 0.000604041


Taking continuous wavelet transform(Mexican Hat).
The running time = 0.037 (ms)

Taking inverse continuous wavelet transform.
The running time = 0.002 (ms)

The relative error is :
norm(st-xt) / norm(st) = 0.00157322


Process returned 0 (0x0)   execution time : 0.085 s
Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 13-1所示。

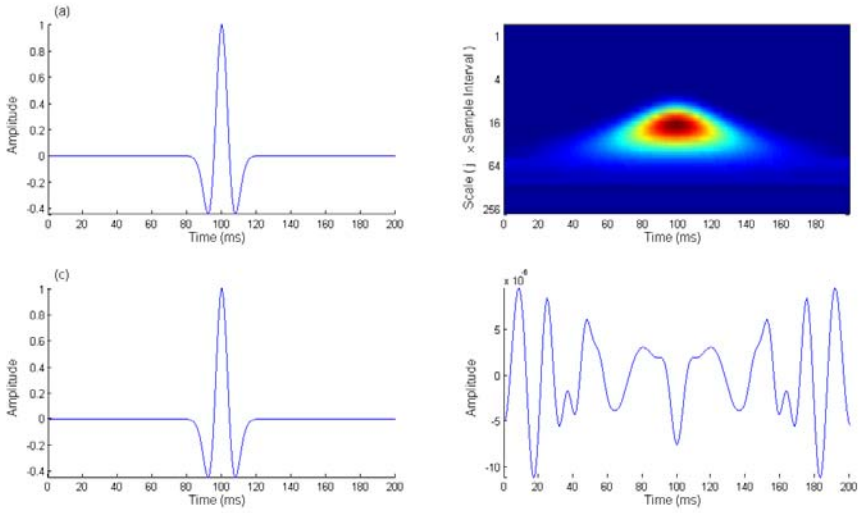

图 13-1　连续小波变换

13.2 二进小波变换

连续小波变换的尺度与平移参数都是连续取值的，所以冗余度很高，如果对尺度参数进行离散化而保持平移参数连续，就得到二进小波变换。二进小波变换即减少了冗余，又能够保证小波变换的平移不变性，因此得到了广泛的应用。

SP++中提供了基于二次样条的二进小波变换程序，如表 13-2所示。对信号进行J级的二进小波变换后可得到 1 级到J级的细节系数和J级的平滑系数，如果想得到j级的平滑系数，则可对小波变换系数进行J-j级的重构即可。

表 13-2　二进小波变换

Operation	Effect
bwt(sn,J)	对 sn 进行 J 级的二进小波变换
ibwt(coefs,j)	对 coefs 进行 j 级的二进小波重构

测试代码：

```
/*****************************************************************************
 *                                bwt_test.cpp
 *
 * Dyadic wavelet transform testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <timing.h>
#include <bwt.h>


using namespace std;
using namespace splab;


const int Ls = 100;


int main()
{

    /************************** [ signal ] ************************/
    Vector<double> s(Ls);
    for(int i=0; i<Ls; i++)
    {
        if(i<Ls/4)
            s[i] = 0.0;
        else if(i<2*Ls/4)
            s[i] = 1.0;
        else if(i<3*Ls/4)
            s[i] = 3.0;
```

```
        else
            s[i] = 0.0;
    }

    /************************** [ BWT ] **************************/
    int level = 3;
    Timing cnt;
    double runtime = 0.0;
    cout << "Taking dyadic wavelet transform." << endl;
    cnt.start();
    Vector< Vector<double> > coefs = bwt( s, level );
    cnt.stop();
    runtime = cnt.read();
    cout << "The running time = " << runtime << " (s)" << endl << endl;

    /************************** [ IBWT ] **************************/
    cout << "Taking inverse dyadic wavelet transform." << endl;
    cnt.start();
    Vector<double> x = ibwt( coefs, level );
    cnt.stop();
    runtime = cnt.read();
    cout << "The running time = " << runtime << " (s)" << endl << endl;

    cout << "The relative error is : norm(s-x) / norm(s) = "
         << norm(s-x)/norm(s) << endl << endl;

    return 0;
}
```

运行结果：

```
Taking dyadic wavelet transform.
The running time = 0 (s)

Taking inverse dyadic wavelet transform.
The running time = 0 (s)

The relative error is : norm(s-x) / norm(s) = 3.76025e-016


Process returned 0 (0x0)   execution time : 0.005 s
Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 13-2与图 13-3所示。

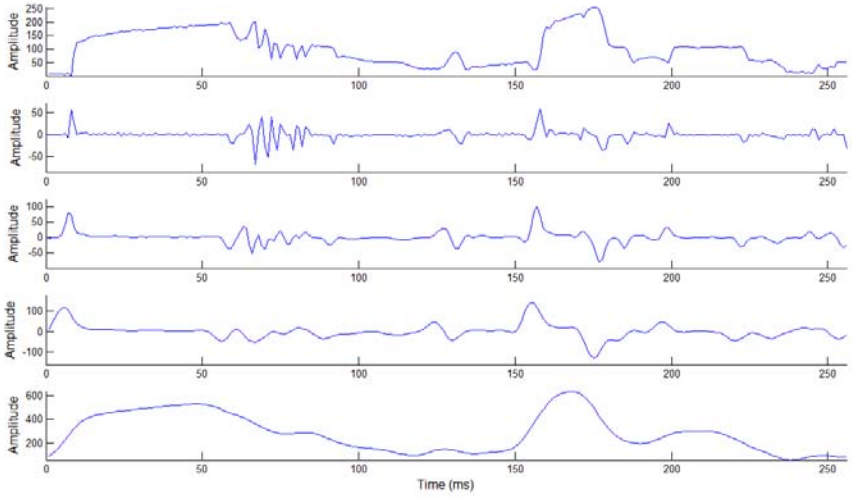

图 13-2　二进小波变换

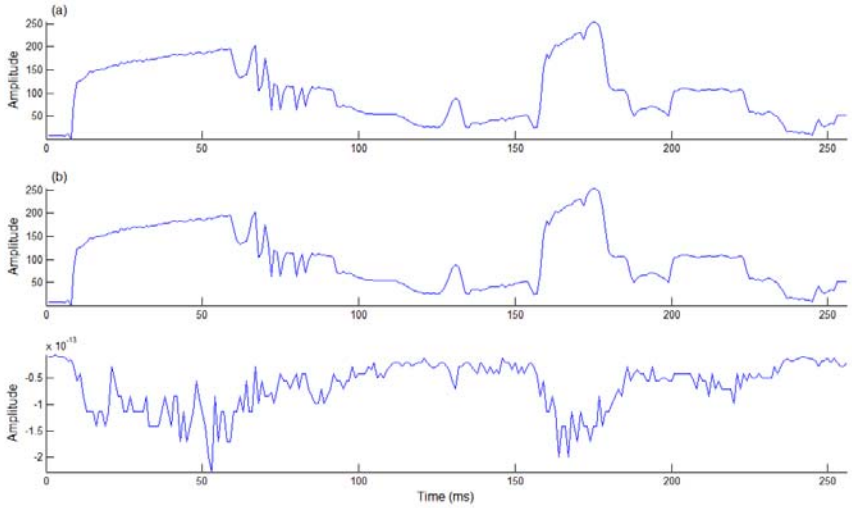

图 13-3　二进小波变换误差分析

13.3 离散小波变换

如果对连续小波变换的尺度参数与平移参数均进行二进离散化，即为离散小波变换。离散小波变换实际上是信号在正交小波基上的投影，故没有冗余，并且在 Mallat 塔式快速算法，计算效率非常高。所以离散小波变换在图像压缩与编码中得到了非常成功的应用。

SP++中提供了离散小波变换的一般算法，默认的滤波器组为”db4”小波，用户根据自己的需要进行修改。为了节约存储空间，小波变换系数存放于一个一维数组中，可以通过一个指标数组访问不同的细节系数和逼近系数，但是这些实现细节已封装在类的内部，用户不必关心。具体的常用操作见表 13-3。

表 13-3　离散小波变换

Operation	Effect
DWT<Type> wt(wname)	创建 DWT 类
wt.~DWT<Type>()	析构 DWT 类
wt.dwt(sn,J)	对 sn 进行 J 级的离散小波变换
wt.idwt(coefs,j)	对 coefs 进行 j 级的离散小波重构
wt.getApprox(ceofs)	获取离散小波变换的平滑系数
wt. getDetial (ceofs,j)	获取离散小波变换的 j 级细节系数
wt.setApprox(a,ceofs)	设置离散小波变换的平滑系数
wt. setDetial (d,ceofs,j)	设置离散小波变换的 j 级细节系数

测试代码：

```
/*****************************************************************************
 *                                dwt_test.cpp
 *
 * Discrete wavelet transform testing.
 *
 * Zhang Ming, 2010-03, Xi'an Jiaotong University.
 *****************************************************************************/


#define BOUNDS_CHECK

#include <iostream>
#include <dwt.h>
#include <timing.h>


using namespace std;
using namespace splab;


typedef float   Type;
const   int     Ls = 1000;


int main()
{
    /****************************** [ signal ] ****************************/
    Vector<Type> s(Ls);
    for(int i=0; i<Ls; i++)
    {
        if(i<Ls/4)
```

```
            s[i] = 0.0;
        else if(i<2*Ls/4)
            s[i] = 1.0;
        else if(i<3*Ls/4)
            s[i] = 3.0;
        else
            s[i] = 0.0;
    }

    /****************************** [ DWT ] ******************************/
    int level = 3;
    DWT<Type> discreteWT("db4");
    Timing cnt;
    double runtime = 0.0;
    cout << "Taking discrete wavelet transform." << endl;
    cnt.start();
    Vector<Type> coefs = discreteWT.dwt( s, level );
    cnt.stop();
    runtime = cnt.read();
    cout << "The running time = " << runtime << " (s)" << endl << endl;

    /****************************** [ IDWT ] ******************************/
    level = 0;
    cout << "Taking inverse discrete wavelet transform." << endl;
    cnt.start();
    Vector<Type> x = discreteWT.idwt( coefs, level );
    cnt.stop();
    runtime = cnt.read();
    cout << "The running time = " << runtime << " (s)" << endl << endl;

    cout << "The relative error is : norm(s-x) / norm(s) = "
         << norm(s-x)/norm(s) << endl << endl;

    return 0;
}
```

运行结果：

```
Taking discrete wavelet transform.
first
The running time = 0.001 (s)

Taking inverse discrete wavelet transform.
The running time = 0.001 (s)

```

```
8.  The relative error is : norm(s-x) / norm(s) = 9.26354e-008
9.
10.
11. Process returned 0 (0x0)   execution time : 0.007 s
12. Press any key to continue.
```

为了更加直观，下面给出一个Matlab的仿真结果，如图 13-4与图 13-5所示。

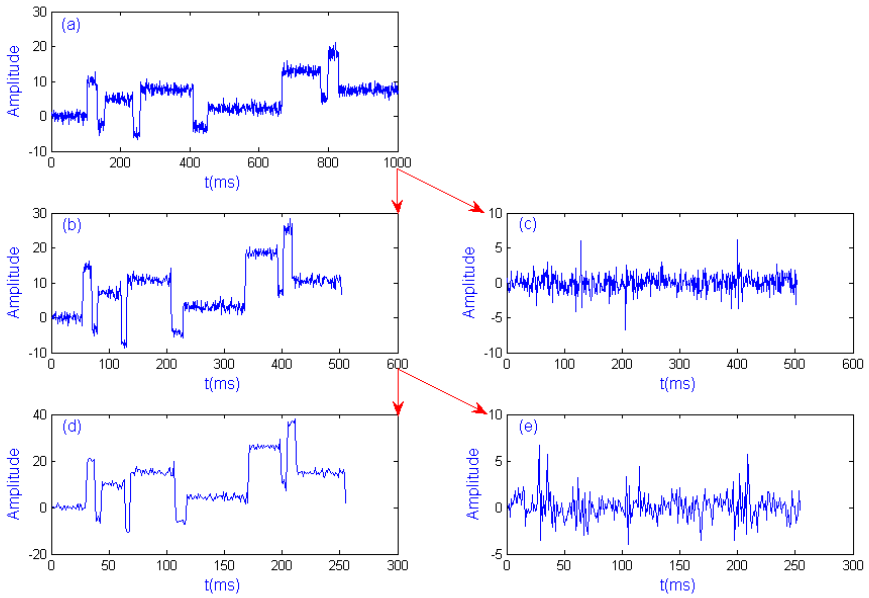

图 13-4　离散小波分解

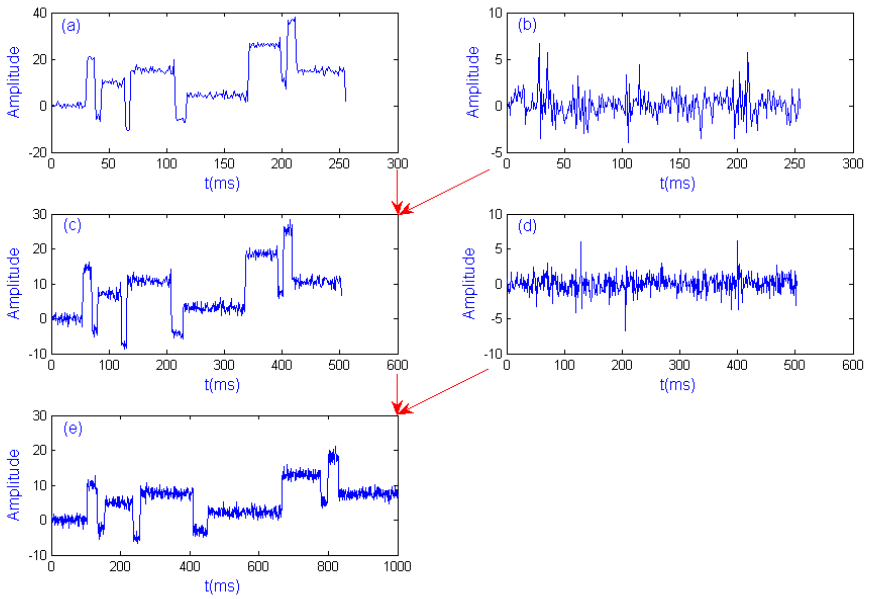

图 13-5　离散小波重构

14 查找与排序

14.1 二叉查找树

二叉树是一类极为重要的数据结构，有着广泛的应用。SP++中实现了二叉树的常用操作，包括前序、中序和后序遍历，插入，删除，查找，最大项与最小项等等，所有算法都以非递归的形式实现，实现过程中用到栈基本数据结构，可参见“stack.h”文件，具体函数见表 14-1。

表 14-1　二叉查找树

Operation	Effect
BSTree bst	创建二叉查找树
bst.~BSTree()	析构二叉查找树
bst.isEmpty()	判断树是否为空
bst.makeEmpty()	将树置空
bst. preTraverse ()	前序遍历
bst. inTraverse ()	中序遍历
bst. postTraverse ()	后序遍历
bst. insert (x)	插入元素
bst.remove(k)	删除元素
bst.search(k)	查找元素
bst.maxItem()	查找最大项
bst.minItem()	查找最小项

测试代码：

```
1.  /*****************************************************************************
2.   *                              bstree_test.cpp
3.   *
4.   * Binary search tree class testing.
5.   *
6.   * Zhang Ming, 2010-07, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #include <iostream>
11. #include <student.h>
12. #include <bstree.h>
```

```
13.
14.
15. using namespace std;
16. using namespace splab;
17.
18.
19. int main()
20. {
21.     const int N = 8,
22.               M = 4;
23.
24.     int x[N] = { 3, 1, 2, 7, 5, 8, 4, 6 };
25.     int y[N] = { 3, 2, 5, 7, 7, 9, 5, 0 };
26.     int z[M] = { 3, 2, 1, 7 };
27.     Student stu;
28.     BSNode<Student> *pNode = NULL;
29.     BSTree<Student, int> stuTree;
30.
31.     for( int i=0; i<N; ++i )
32.     {
33.         stu.key = x[i];
34.         if( stuTree.insert( stu ) )
35.             cout << "Insert " << x[i] << " success." << endl;
36.         else
37.             cout << "Insert failing." << endl;
38.     }
39.     cout << endl;
40.
41.     cout << "Preorder Travesal: " << endl;
42.     stuTree.preTraverse();
43.     cout << endl << "Inorder Travesal: " << endl;
44.     stuTree.inTraverse();
45.     cout << endl  << "Postorder Travesal: " << endl;
46.     stuTree.postTraverse();
47.     cout << endl;
48.
49.     for( int i=0; i<N; ++i )
50.     {
51.         if( stuTree.remove( y[i] ) )
52.         {
53.             cout << "Remove " << y[i] <<" sucess. Preorder Travesal: "<< endl;
54. //            stuTree.preTraverse();
55.             stuTree.inTraverse();
56. //            stuTree.postTraverse();
```

```
57.         }
58.         else
59.             cout << "No such item (key=" << y[i] << ") in the tree!" << endl;
60.     }
61.
62.     cout << endl;
63.     for( int i=0; i<M; ++i)
64.     {
65.         pNode = stuTree.search( z[i] );
66.         if( pNode )
67.             cout << "Finding the element: " << pNode->data;
68.         else
69.             cout << "No such item (key=" << z[i] << ") in the tree!" << endl;
70.     }
71.     cout << endl;
72.
73.     stuTree.makeEmpty();
74.     if( stuTree.isEmpty() )
75.         cout << "The tree is empty." << endl;
76.     else
77.         cerr << "Making tree empty failing!" << endl;
78.
79.     cout << endl;
80.     return 0;
81. }
```

运行结果：

```
1.  Insert 3 success.
2.  Insert 1 success.
3.  Insert 2 success.
4.  Insert 7 success.
5.  Insert 5 success.
6.  Insert 8 success.
7.  Insert 4 success.
8.  Insert 6 success.
9.
10. Preorder Travesal:
11. 3       Zhang Ming
12. 1       Zhang Ming
13. 2       Zhang Ming
14. 7       Zhang Ming
15. 5       Zhang Ming
16. 4       Zhang Ming
17. 6       Zhang Ming
```

```
8       Zhang Ming

Inorder Travesal:
1       Zhang Ming
2       Zhang Ming
3       Zhang Ming
4       Zhang Ming
5       Zhang Ming
6       Zhang Ming
7       Zhang Ming
8       Zhang Ming

Postorder Travesal:
2       Zhang Ming
1       Zhang Ming
4       Zhang Ming
6       Zhang Ming
5       Zhang Ming
8       Zhang Ming
7       Zhang Ming
3       Zhang Ming

Remove 3 sucess. Preorder Travesal:
1       Zhang Ming
2       Zhang Ming
4       Zhang Ming
5       Zhang Ming
6       Zhang Ming
7       Zhang Ming
8       Zhang Ming
Remove 2 sucess. Preorder Travesal:
1       Zhang Ming
4       Zhang Ming
5       Zhang Ming
6       Zhang Ming
7       Zhang Ming
8       Zhang Ming
Remove 5 sucess. Preorder Travesal:
1       Zhang Ming
4       Zhang Ming
6       Zhang Ming
7       Zhang Ming
8       Zhang Ming
Remove 7 sucess. Preorder Travesal:
```

```
1       Zhang Ming
4       Zhang Ming
6       Zhang Ming
8       Zhang Ming
No such item (key=7) in the tree!
No such item (key=9) in the tree!
No such item (key=5) in the tree!
No such item (key=0) in the tree!

No such item (key=3) in the tree!
No such item (key=2) in the tree!
Finding the element: 1  Zhang Ming
No such item (key=7) in the tree!

The tree is empty.


Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

14.2 平衡二叉树

二叉树可将时间复杂度为N的运算降为$\log_2 N$，这是以二叉树比较平衡为前提的，如果树不平衡，最坏情况下复杂度仍然为N，而且要浪费多余的存储空间。一种解决方案就是通过旋转技术使二叉树保持平衡，这种二叉树称之为平衡二叉树，或AVL树。SP++中实现了AVL树的常用操作，包括遍历，查找，插入，删除等等，实现过程中用到栈基本数据结构，可参见“stack.h”文件，具体函数见表 14-2。

表 14-2　平衡二叉树

Operation	Effect
AVLTree avlt	创建 AVL 树
avlt.~ AVLTree()	析构 AVL 树
avlt.isEmpty()	判断树是否为空
avlt.makeEmpty()	将树置空
avlt. print (mode)	树的遍历，包括前序、中序、后序
avlt. height()	计算树的高度
avlt. search(k)	查找关键字 k
avlt. insert(x)	插入元素
avlt. remove(k,x)	删除元素

测试代码：

```
/*****************************************************************************
 *                                avltree_test.cpp
 *
 * AVL tree class testing.
 *
 * Zhang Ming, 2009-10, Xi'an Jiaotong University.
 *****************************************************************************/


#include <iostream>
#include <student.h>
#include <avltree.h>


using namespace std;
using namespace splab;


int main()
{
    int x[16] = { 3, 2, 1, 4, 5, 6, 7, 16, 15, 14, 13, 12, 11, 10, 8, 9 };
    int y[16] = { 9, 4, 12, 6, 5, 2, 3, 15, 14, 7, 8, 1, 1, 3, 20, 12 };
    int z[4] = { 10, 13, 5, 1 };

    Student stu;
    AVLNode<Student, int> *pNode;
    AVLTree<Student, int> stuTree;

    for( int i=0; i<16; ++i )
    {
        stu.key = x[i];
        stuTree.insert( stu );
    }

    cout << "Preorder Travesal: " << endl;
    stuTree.print( "preorder" );
    cout << endl << endl;

    for( int i=0; i<16; ++i )
    {
        if( stuTree.remove( y[i], stu ) )
        {
            cout << "The removed item is:  "<< stu;
            cout << "Preorder Travesal: " << endl;
```

```
45.             stuTree.print( "preorder" );
46.         }
47.         else
48.         {
49.             cout << "No such item (key=" << y[i] << ") in the tree!";
50.         }
51.         cout << endl << endl;
52.
53.     }
54.
55.     cout << endl;
56.     for( int i=0; i<4; ++i)
57.     {
58.         pNode = stuTree.search( z[i] );
59.         if( pNode )
60.             cout << "Have finding the element: " << pNode->data;
61.         else
62.             cout << "No such item (key=" << z[i] << ") in the tree!";
63.         cout << endl;
64.     }
65.     cout << endl;
66.
67.     stuTree.makeEmpty();
68.     pNode = stuTree.search( 10 );
69.
70.     return 0;
71. }
```

运行结果：

```
1.  Preorder Travesal:
2.    1     7       Zhang Ming
3.    0     4       Zhang Ming
4.    0     2       Zhang Ming
5.    0     1       Zhang Ming
6.    0     3       Zhang Ming
7.    -1    6       Zhang Ming
8.    0     5       Zhang Ming
9.    -1    13      Zhang Ming
10.   -1    11      Zhang Ming
11.   0     9       Zhang Ming
12.   0     8       Zhang Ming
13.   0     10      Zhang Ming
14.   0     12      Zhang Ming
15.   0     15      Zhang Ming
```

```
  0     14      Zhang Ming
  0     16      Zhang Ming


The removed item is:  9 Zhang Ming
Preorder Travesal:
  1     7       Zhang Ming
  0     4       Zhang Ming
  0     2       Zhang Ming
  0     1       Zhang Ming
  0     3       Zhang Ming
  -1    6       Zhang Ming
  0     5       Zhang Ming
  -1    13      Zhang Ming
  -1    11      Zhang Ming
  1     8       Zhang Ming
  0     10      Zhang Ming
  0     12      Zhang Ming
  0     15      Zhang Ming
  0     14      Zhang Ming
  0     16      Zhang Ming


The removed item is:  4 Zhang Ming
Preorder Travesal:
  1     7       Zhang Ming
  0     3       Zhang Ming
  -1    2       Zhang Ming
  0     1       Zhang Ming
  -1    6       Zhang Ming
  0     5       Zhang Ming
  -1    13      Zhang Ming
  -1    11      Zhang Ming
  1     8       Zhang Ming
  0     10      Zhang Ming
  0     12      Zhang Ming
  0     15      Zhang Ming
  0     14      Zhang Ming
  0     16      Zhang Ming


The removed item is:  12        Zhang Ming
Preorder Travesal:
  0     7       Zhang Ming
```

```
  0     3       Zhang Ming
  -1    2       Zhang Ming
  0     1       Zhang Ming
  -1    6       Zhang Ming
  0     5       Zhang Ming
  0     13      Zhang Ming
  0     10      Zhang Ming
  0     8       Zhang Ming
  0     11      Zhang Ming
  0     15      Zhang Ming
  0     14      Zhang Ming
  0     16      Zhang Ming


The removed item is:  6 Zhang Ming
Preorder Travesal:
  0     7       Zhang Ming
  -1    3       Zhang Ming
  -1    2       Zhang Ming
  0     1       Zhang Ming
  0     5       Zhang Ming
  0     13      Zhang Ming
  0     10      Zhang Ming
  0     8       Zhang Ming
  0     11      Zhang Ming
  0     15      Zhang Ming
  0     14      Zhang Ming
  0     16      Zhang Ming


The removed item is:  5 Zhang Ming
Preorder Travesal:
  1     7       Zhang Ming
  0     2       Zhang Ming
  0     1       Zhang Ming
  0     3       Zhang Ming
  0     13      Zhang Ming
  0     10      Zhang Ming
  0     8       Zhang Ming
  0     11      Zhang Ming
  0     15      Zhang Ming
  0     14      Zhang Ming
  0     16      Zhang Ming

```

```
104.
105. The removed item is:  2 Zhang Ming
106. Preorder Travesal:
107.   1     7       Zhang Ming
108.   1     1       Zhang Ming
109.   0     3       Zhang Ming
110.   0     13      Zhang Ming
111.   0     10      Zhang Ming
112.   0     8       Zhang Ming
113.   0     11      Zhang Ming
114.   0     15      Zhang Ming
115.   0     14      Zhang Ming
116.   0     16      Zhang Ming
117.
118.
119. The removed item is:  3 Zhang Ming
120. Preorder Travesal:
121.   -1    13      Zhang Ming
122.   1     7       Zhang Ming
123.   0     1       Zhang Ming
124.   0     10      Zhang Ming
125.   0     8       Zhang Ming
126.   0     11      Zhang Ming
127.   0     15      Zhang Ming
128.   0     14      Zhang Ming
129.   0     16      Zhang Ming
130.
131.
132. The removed item is:  15        Zhang Ming
133. Preorder Travesal:
134.   -1    13      Zhang Ming
135.   1     7       Zhang Ming
136.   0     1       Zhang Ming
137.   0     10      Zhang Ming
138.   0     8       Zhang Ming
139.   0     11      Zhang Ming
140.   1     14      Zhang Ming
141.   0     16      Zhang Ming
142.
143.
144. The removed item is:  14        Zhang Ming
145. Preorder Travesal:
146.   0     10      Zhang Ming
147.   0     7       Zhang Ming
```

```
  0     1       Zhang Ming
  0     8       Zhang Ming
  0     13      Zhang Ming
  0     11      Zhang Ming
  0     16      Zhang Ming


The removed item is:  7 Zhang Ming
Preorder Travesal:
  0     10      Zhang Ming
  1     1       Zhang Ming
  0     8       Zhang Ming
  0     13      Zhang Ming
  0     11      Zhang Ming
  0     16      Zhang Ming


The removed item is:  8 Zhang Ming
Preorder Travesal:
  1     10      Zhang Ming
  0     1       Zhang Ming
  0     13      Zhang Ming
  0     11      Zhang Ming
  0     16      Zhang Ming


The removed item is:  1 Zhang Ming
Preorder Travesal:
  -1    13      Zhang Ming
  1     10      Zhang Ming
  0     11      Zhang Ming
  0     16      Zhang Ming


No such item (key=1) in the tree!

No such item (key=3) in the tree!

No such item (key=20) in the tree!

No such item (key=12) in the tree!


Have finding the element: 10    Zhang Ming
```

```
192.
193. Have finding the element: 13    Zhang Ming
194.
195. No such item (key=5) in the tree!
196. No such item (key=1) in the tree!
197.
198. The tree is empty!
199.
200.
201. Process returned 1 (0x1)   execution time : 0.156 s
202. Press any key to continue.
```

14.3 基本排序算法

排序是一类非常重要而又有广泛应用的算法，SP++中提供了常用的 6 类排序算法，时间复杂度与空间复杂度分析可参考数据结构的书籍，在此不再赘述。具体函数见表 14-3。

表 14-3　排序算法

Operation	Effect
bubbleSort(v,left,right)	Bubble 排序算法
selectSort (v,left,right)	选择排序算法
insertSort (v,left,right)	插入排序算法
quickSort (v,left,right)	快速排序算法
mergSort (v,left,right)	归并排序算法
heapSort (v,left,right)	堆排序算法

测试代码：

```
1.  /**************************************************************************
2.   *                                sort_test.cpp
3.   *
4.   * Sorting algorithm testing.
5.   *
6.   * Zhang Ming, 2010-07, Xi'an Jiaotong University.
7.   **************************************************************************/
8.
9.
10. #include <iostream>
11. #include <random.h>
12. #include <sort.h>
13.
```

```
14.
15. using namespace std;
16. using namespace splab;
17.
18.
19. const int SIZE = 10;
20.
21.
22. template <typename Type>
23. void printVector( const Vector<Type> &a )
24. {
25.     Vector<int>::const_iterator itr = a.begin();
26.     while( itr != a.end() )
27.         cout << *itr++ << "\t";
28.
29.     cout << endl;
30. }
31.
32.
33. int main()
34. {
35.     Vector<int> a( SIZE );
36.
37.     cout << "Unsorted Numbers : " << endl;
38.     a = randu( 127, 1, 10*SIZE, SIZE );
39.     printVector( a );
40.     cout << "Bubble Sorted Numbers : " << endl;
41.     bubbleSort( a, 0, a.size()-1 );
42.     printVector( a );
43.     cout << endl;
44.
45.     cout << "Unsorted Numbers : " << endl;
46.     a = randu( 13579, 1, 10*SIZE, SIZE );
47.     printVector( a );
48.     cout << "Select Sorted Numbers : " << endl;
49.     selectSort( a, 0, a.size()-1 );
50.     printVector( a );
51.     cout << endl;
52.
53.     cout << "Unsorted Numbers : " << endl;
54.     a = randu( 37, 1, 10*SIZE, SIZE );
55.     printVector( a );
56.     cout << "Insert Sorted Numbers : " << endl;
57.     insertSort( a, 0, a.size()-1 );
```

```
58.     printVector( a );
59.     cout << endl;
60.
61.     cout << "Unsorted Numbers : " << endl;
62.     for( int i=0; i<a.size(); ++i )
63.         a[i] = randu( 127, 1, 10*SIZE );
64.     printVector( a );
65.     cout << "Quick Sorted Numbers : " << endl;
66.     quickSort( a, 0, a.size()-1 );
67.     printVector( a );
68.     cout << endl;
69.
70.     cout << "Unsorted Numbers : " << endl;
71.     for( int i=0; i<a.size(); ++i )
72.         a[i] = randu( 127, 1, 10*SIZE );
73.     printVector( a );
74.     cout << "Merg Sorted Numbers : " << endl;
75.     mergSort( a, 0, a.size()-1 );
76.     printVector( a );
77.     cout << endl;
78.
79.     cout << "Unsorted Numbers : " << endl;
80.     for( int i=0; i<a.size(); ++i )
81.         a[i] = randu( 127, 1, 10*SIZE );
82.     printVector( a );
83.     cout << "Heap Sorted Numbers : " << endl;
84.     heapSort( a, 0, a.size()-1 );
85.     printVector( a );
86.     cout << endl;
87.
88.     return 0;
89. }
```

运行结果：

```
1.  Unsorted Numbers :
2.  1       80      37      24      93      9       40      55      39      43
3.
4.  Bubble Sorted Numbers :
5.  1       9       24      37      39      40      43      55      80      93
6.
7.
8.  Unsorted Numbers :
9.  31      65      76      19      87      79      85      56      1       16
10.
```

```
Select Sorted Numbers :
1       16      19      31      56      65      76      79      85      87


Unsorted Numbers :
1       15      25      98      81      2       6       72      73      51

Insert Sorted Numbers :
1       2       6       15      25      51      72      73      81      98


Unsorted Numbers :
1       80      37      24      93      9       40      55      39      43

Quick Sorted Numbers :
1       9       24      37      39      40      43      55      80      93


Unsorted Numbers :
37      17      94      81      18      98      33      38      23      76

Merg Sorted Numbers :
17      18      23      33      37      38      76      81      94      98


Unsorted Numbers :
73      11      24      93      49      12      91      96      17      34

Heap Sorted Numbers :
11      12      17      24      34      49      73      91      93      96


Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

14.4 Huffman 编码

Huffman编码是根据数据的统计规律实现的一种无损压缩编码，可以通过Huffman树实现编码与解码过程。具体函数见表 14-4。实现过程中用到最小二叉堆基本数据结构，可参见“binaryheap.h”文件，并且设计了节点指针类以实现通过指针指向内容的大小比较指针的大小。

表 14-4 Huffman 编码

Operation	Effect
HuffmanTree ht	创建 Huffman 树
ht.~ HuffmanTree()	析构 Huffman 树
ht.code(codeArray,lenght)	Huffman 编码
ht.decode(bits,length,decodeword)	Huffman 解码
ht.printCode(bits,length)	打印码字
ht.printCodeTable()	打印码表

测试代码：

```
1.  /*****************************************************************************
2.   *                              huffmancode_test.cpp
3.   *
4.   * Huffman Code class testing.
5.   *
6.   * Zhang Ming, 2010-07, Xi'an Jiaotong University.
7.   *****************************************************************************/
8.
9.
10. #include <iostream>
11. #include <string>
12. #include <huffmancode.h>
13.
14.
15. using namespace std;
16. using namespace splab;
17.
18.
19. const int CHARNUM = 128;
20. template <typename Object, typename Weight>
21. void creatCodeInfo( string &str, CodeObject<Object,Weight> *&codeArr, int &length
    );
22. template <typename Object, typename Weight>
23. void printCodeInfo( CodeObject<Object,Weight> *&codeArr, int length );
24.
25.
26. int main()
27. {
28.     int length = 0;
29.     CodeObject<char, int> *codeArr = NULL;
30.     string str = "AAAABBCD";
31.     creatCodeInfo( str, codeArr, length );
```

```
32.     printCodeInfo( codeArr, length );
33.
34.     HuffmanTree<char, int> ht;
35.     ht.code( codeArr, length );
36.     ht.printCodeTable();
37.
38.     char decodedResult;
39.     unsigned char codeword[CODESIZE];
40.     cout << "Coding\t" << "Decodeding" << endl;
41.     codeword[0] = 0xC0;
42.     if( ht.decode( codeword, 3, decodedResult ) )
43.     {
44.         ht.printCode( codeword, 3 );
45.         cout << "\t" << decodedResult << endl;
46.     }
47.     else
48.         cerr << "invalid codeword! " << endl;
49.     codeword[0] = 0xC0;
50.     if( ht.decode( codeword, 4, decodedResult ) )
51.     {
52.         ht.printCode( codeword, 3 );
53.         cout << "\t" << decodedResult << endl;
54.     }
55.     else
56.     {
57.         ht.printCode( codeword, 4 );
58.         cerr << "\t" << "invalid codeword! " << endl;
59.     }
60.     codeword[0] = 0x40;
61.     if( ht.decode( codeword, 3, decodedResult ) )
62.     {
63.         ht.printCode( codeword, 3 );
64.         cout << "\t" << decodedResult << endl;
65.     }
66.     else
67.     {
68.         ht.printCode( codeword, 3 );
69.         cerr << "\t" << "invalid codeword! " << endl;
70.     }
71.
72.     cout << endl;
73.     return 0;
74. }
75.
```

```
76.
77. /**
78.  * Generate the array for counting character frequency.
79.  */
80. template <typename Object, typename Weight>
81. void creatCodeInfo( string &str, CodeObject<Object,Weight> *&codeArr, int &length
    )
82. {
83.     char charFreq[CHARNUM];
84.     int index[CHARNUM];
85.     for( int i=0; i<CHARNUM; ++i )
86.     {
87.         charFreq[i] = 0;
88.         index[i] = -1;
89.     }
90.
91.     length = 0;
92.     for( unsigned int i=0; i<str.size(); ++i )
93.         charFreq[int(str[i])] += 1;
94.
95.     for( int i=0; i<CHARNUM; ++i )
96.         if( charFreq[i] )
97.             index[length++] = i;
98.
99.
100.    codeArr = new CodeObject<Object,Weight>[length];
101.    for( int i=0; i<length; ++i )
102.    {
103.        codeArr[i].data = index[i];
104.        codeArr[i].cost = charFreq[index[i]];
105.    }
106. }
107.
108.
109. /**
110.  * Print the characters and there frequency.
111.  */
112. template <typename Object, typename Weight>
113. void printCodeInfo( CodeObject<Object,Weight> *&codeArr, int length )
114. {
115.     cout << "Object\tFrequency" << endl;
116.     for( int i=0; i<length; ++i )
117.         cout << codeArr[i].data << "\t" << codeArr[i].cost << endl;
118.     cout << endl;
```

```
}
```

运行结果：

```
Object  Frequency
A       4
B       2
C       1
D       1

Object  Code    Size
A       0       1
B       10      2
C       110     3
D       111     3

Coding  Decodeding
110     C
1100    invalid codeword!
010     invalid codeword!


Process returned 0 (0x0)   execution time : 0.000 s
Press any key to continue.
```

15 参考文献

15.1 书籍

[1] John H. Mathews, Kurtis D. Fink. Numerical Methods Using MATLAB, Fourier Edition. Beijing, Publishing House of Electronics Industry.

[2] Shie Qian. Introduction to Time-Frequency Analysis and Wavelet Transforms. Beijing, China Machine Press, 2005.

[3] Stephane Mallat. A Wavelet Tour of Signal Processing. Second Edition. Beijing, China Machine Press, 2006.

[4] B. Widrow, S. D. Stearns. Adaptive Signal Processing. Beijing, China Machine Press, 2008.

[5] Paulo S.R. Diniz. Adaptive Filtering Algorithms and Practical Implementation, Third Edition. Springer Science+Business Media, LLC, 2008.

[6] Richard C. Aster, Brian Borchers, and Clifford Thurber. Parameter Estimation and Inverse Problems. 2004.

[7] T. M. Cover, J. A. Thomas. Elements of Information Theory, Second Edition. 阮吉寿,张华译.北京,机械工业出版社,2008.

[8] Ingrid Daubechies. 小波十讲.李建平，杨万年译. 北京，国防工业出版社, 2004.

[9] 葛哲学，沙威. 小波分析理论与 MATLABR2007 实现. 北京，电子工业出版社, 2007.

[10] 邹理合. 数字信号处理(上册). 西安，西安交通大学教材科印, 2006.

[11] Alen V. Oppenheim, Alen S. Willsky and With S. Hamid Nawab. 信号与系统，第二版,刘树棠译. 西安，西安交通大学出版社, 2005.

[12] 姚泽清，苏晓冰等. 应用泛函分析，北京，科学出版社, 2007.

[13] 郭懋正. 实变函数与泛函分析. 北京，北京大学出版社, 2007.

[14] 魏战线. 线性代数与解析几何. 高等教育出版社，2006.

[15] 邓建中，刘之行. 计算方法. 西安交通大学, 2001.

[16] 胡茂林. 矩阵计算与应用. 北京，科学出版社, 2007.

[17] 高西全,丁玉美. 数字信号处理，第三版. 西安，西安电子科技大学出版社, 2008.

[18] 叶中付. 统计信号处理. 安徽，中国科技大学出版社, 2009.

[19] 解可新，韩健，林友联. 最优化方法，修定版. 天津，天津大学出版社, 2008.

[20] Mark Allen Weiss. Data Structures and Algorithm Analysis in C++. Third Edition. Beijing, Posts & Telecom Press, 2007.

[21] Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. C++ Primer, Fourth Edition. Addison Wesley Professional, 2005.

[22] Scott Meyers. Effective C++ 3th, 侯捷译. 北京，电子工业出版社, 2006.

[23] Nicolai M. Josuttis. C++ Standard Library A Tutorial and Reference. Addison

Wesley Longman, 1999.
[24] Kernighan, Ritchie. The C Programming Language, 2nd. 徐宝文, 李志译. 北京, 电子工业出版社, 2009.
[25] 谭浩强. C++ 面向对象程序设计. 北京, 清华大学出版社，2006.
[26] 严蔚敏, 吴伟民. 数据结构(C 语言版). 清华大学出版社, 2006.
[27] 殷人昆. 数据结构(面向对象方法与 C++语言描述), 第二版. 北京, 清华大学出版社，2007.
[28] 林锐. 高质量 C/C++编程指南. 2001.

15.2 文章

[29] Christopher Torrence, Gilbert P. Compo. A Practical Guide to Wavelet Analysis. Bulletin of the American Meteorological Society,1998,79(1): 61~78.
[30] Shie Qian, Dapang Chen. Discrete Gabor Transform. IEEE Transactions on Signal Processing, 1993, 41(7): 2429~2438.
[31] Stephane Mallat. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1989, 11(7): 674~693.
[32] Stephane Mallat, Sifen Zhong. Characterization of Signals from Multiscale Edges.IEEE Transactions on Pattern Analysis and Machine Intelligence. 1992, 14(7):710~732.
[33] Ingrid Daubechies.The Wavelet Transform, Time-Frequency Localization and Signal Analysis. IEEE Transactions on Information Theory. 1990, 36(5): 961~1005.
[34] R. G. Stockwell, L. Mansinha, and R. P. Lowe Localization of the Complex Spectrum: The S Transform, IEEE Transactions on Signal Processing, 1996, 44(4): 998~1001.
[35] Pham DT. Fast algorithms for mutual information based independent component analysis. IEEE Transactions on Signal Processing, 2004, 52(10): 2690～2700.

15.3 网站

[36] FFTW http://www.fftw.org/
[37] Blitz++ http://www.oonumerics.org/blitz/
[38] GSL http://www.gnu.org/software/gsl/
[39] Numerical Recipes http://www.nr.com/
[40] Template Numerical Toolkit http://math.nist.gov/tnt/
[41] IT++ http://sourceforge.net/apps/wordpress/itpp/
[42] SGI STL http://www.sgi.com/tech/stl/
[43] Boost http://www.boost.org/

16 有感于 SP++

16.1 心血来潮

SP++最早的工作始于 2008 年，而动机则成于 2009 年。

2008 年全年和 2009 年上半年正值本科毕业设计和研一上课，在此期间甚是无聊，于是就想用 C++写点程序。本人从事信号处理专业，因此就把信号处理中的常用算法用 C++写了一遍，但这些程序各自为阵，不成体系。

2009 年下半年做项目时要实现一个很复杂的算法，Matlab 程序中调用了许多内部函数，转 C 语言时遇到了一些困难，因为该算法在频域实现，所以各种函数都是基于复数形式。为此上网查了一些 C 和 C++的开源库，结果有两惊人的发现，一个令人高兴，而另一个则让人失望：高兴的是网上有成千上万的关于数值计算和信号处理的程序库（我最终选用了 GSL）；失望的是这么多的库几乎没有哪个是中国人写的（唉，其实原因也很简单：在这个浮躁和功利的学术氛围下，谁会愿意做这种“义务劳动”的事情呢？）。

经过一年的“烟酒僧”生活，我有过一些思考：三年研究生应当怎样度过呢？我没有保尔柯察金那样的豪迈气魄，但还是觉得应该做点有意义的事，于是萌生了一个想法：写一个关于信号处理的 C++程序库。然而由于信号处理算法中要经常用到矩阵分解，解线性、非线性方程组、曲线插值与拟合等等数值计算的内容，因此 SP++实际上可分为两部分：数值计算与信号处理。

16.2 艰苦历程

想着容易做着难啊！

在时间、精力、Money 都有限的情况下，仅凭我一人完成这样一项工作谈何容易。我又有个毛病：做什么事都不甘心，既然动手做了，就一定尽我所能把它做好！于是从理论学习，算法设计，代码编写、调试、验证，到最终文档撰写，维护与更新，一步一步做下去。

首先是时间问题，平时要忙项目和硕士论文，所以这项工作只能在晚上和周末有时间的情况下做，还是鲁迅说的对：“时间就像海面里的水，要挤还是有的”。其次是精力问题，现在精力大不如前了，想当初前一天包夜第二天考试，而如今熬个通宵几天都缓不过来。最后就是钱的问题了，做学生难，在中国做学生更难，穷得叮当响，买一本几十块钱的书都要犹豫好长时间，真是杯具呀！

最艰难的莫过于调程序——推公式——调程序——推公式……要写出正确高效的代码，必须对算法的理论有深入理解。浅尝辄止、似是而非是远远不够的；在已有的基础上生搬硬套、照猫画虎，那就有可能贻笑大方了。

我坚信“有付出就会有回报”，经过 3 年努力，SP++版本已经更新 3 次，目前实现的算法有：（数值计算部分）向量与矩阵基本运算，矩阵分解，常规、欠定、超定线性方程组求解，非线性方程求根，曲线插值与拟合，非线性无约束优化算法等等；（信号处理部分）Fourier 变换，数字滤波器设计，随机信号处理，自适应信号处理，时频分析，小波变换等等。

一个人的力量毕竟有限，如果仅凭一个人把一件事做得尽善尽美，那么此人乃天才也！然而我不相信这个世界存在天才，我们生来都是人才，不同的是经过中国式的教育培养，有些人保持着人才的特性（还好，我属于这一类），而有些人则被教育成了蠢才。所以 SP++中必然存在着已经发现的和尚未发现的诸多不足，已发现的力争及时改正，未发现的只好争取尽早发现再做改正。

16.3 有得有失

我这人有点傻，事实证明我确实有些傻，不仅毫无回报地把几乎所有空闲时间都花在了这个“义务劳动”上，而且自己还倒贴了六七百元买书。

“有得有失，有失有得”，我感觉这是老祖宗给我们留下的最大智慧（至少是“之一”）。由于 SP++而失去和错过的东西太多，在此没有心情去讲了，还是谈谈所得吧，以勉励他人。有时候在想我付出这么多到底获得了什么？其一是网友的感谢。其二是水平的提高？其三是帮我找到了一份工作，其四，唉，实在没有了！

最直接的回报当数网友的致谢、赞美和建议、批评了。致谢和赞美固然好，让人有飘飘然的感觉；不过建议和批评也必不可少，使你改正错误，更进一步。

至于水平吗，不敢说有多少提高，只能是在某些方面加强吧了。水平，或者能力的面太广了，学识渊博是一种能力，为人处事也是一种能力，只是在不同领域中、不同场合下能力的各种方面的权重不一样而已。而且某一方面能力的加强，必然会导致另一方面的减少，姑且称之为“能力守恒定律”吧，就看你的选择是什么了。

我们课题组承担都是国家的项目，什么“863 项目”，“科技部重大专项”等，这些玩意儿很玄乎，都偏重于理念研究，平时听起来很牛，可是到了找工作就杯具了！唯一的去处可能就是研究所吧（高校是不要小硕滴），可惜国企我一概不考虑（本人一没背景，二没靠山，进去了怎么混啊？）。各类公司都更加注重技术，关心的是做出一些实用的东西来，而不是高深莫测的理论，因为公司毕竟是要赚钱的，要不然就没法活下去了。还好，从 SP++的开发中学到了一点点技术，应付个笔试和面试还行，看来“付出就有回报”这句话还是有道理的。

看过一本书里讲学习的四种境界：学会——会学——会用——被用。真是有一种找到知己的感觉，英雄所见略同啊（大言不惭地自称一下英雄，过过瘾，哈哈）！学会很容易，我们从小学（甚至幼儿园）到大学，经过千锤百炼，身经百战，可谓是过五关斩六将的勇士，该学的也都学会了，至少也学得差不多了。会学可就难了，学会与会学就像鱼与渔的关系，可惜据我观察，会学的人不多。会用也相对容易些，但也有好坏之分，用得好，用得恰如其分就能达到事半功倍的效果。最难的要数被用了，做到这一点实属不易，要得到别人的认可必须要有过人之处。扯远了，赶快回来！

总而言之，最大的收获，或是对我最大的鼓励莫过于“被用”了，这也是我一直将 SP++进行到底的动力所在。

16.4 见仁见智

SP++到底是个什么水平？

答：比起老外们写的成熟的库相差甚远，比起网上传的“散兵游勇”式程序要高出那么一点点。毕竟 SP++有自己的体系框架，很多算法在项目中经过了实践的检验，而且有些算法根据网上高人指点和好友建议做了不少改进，在此表示忠心感谢！

不过对于一些建议我还是固执己见，没做修改（同样表示忠心感谢），比如某些明显的效率问题。原因有二：其一，既然明显，编译器会帮助我们优化，所以不必担心；其二，部分缺陷是出于开发效率和运行效率的折中，运行速度是效率，开发速度也是效率，如果能够方便使用，牺牲一些运行效率还是值得的。

SP++对我来说到底有什么意义？

答：意义深远，不敢说是比硕士学位重要（当然不敢，以后还得靠这吃饭了），至少比那个本本有意义。一个硕士学位证到底有多少含金量？没上过的不知道，上过的吓一跳——真没多含金量，含水量倒不少。试想，一个硕士论文能被多少人引用？毕业后还有多少人能继续关注曾经被他研究过的东西？不言自明，少之又少，因为我们处在一个文凭与水平毫无关系的时代。所以 SP++过万的访问量对我来说确实比那张学位证书更有意义。

是为感

张明，2011 年 02 月于西安