SORAY CENGİZ
161024077

HW05:

14.1.1 itpp-lu-decomp-cpp

14.1.1.1 lu_decomp_itpp_test.cc

NOT :

“it++” kütüphanesi “https://stackoverflow.com/questions/41077559/quick-and-hassle-free-installation-usage-of-it-library-on-linux-windows“ adresindeki stack overflow’daki yönergeler takip edilerek Linux’a kurulmuştur, kurulum yaklaşık 10dakika sürmüştür.

Çözülen Problemi Anlayabilmek için;

[1]“http://www.cfm.brown.edu/people/dobrush/cs52/Mathematica/Part2/PLU.html “

[2] “https://en.wikipedia.org/wiki/LU_decomposition”

[3]”https://ece.uwaterloo.ca/~dwharder/NumericalAnalysis/04LinearAlgebra/lup/”

vb. siteler ziyaret edilmiştir.

In C++:

```
 * For licensing information check the above url.
 * Please do not remove this header.
 * */

#include <iostream>
#include <iomanip>
#include <sstream>
#include <vector>
#include <tuple>
#include <cassert>
#include <cstdio>

#include <itpp/itstat.h>

namespace os_test {

  namespace itpp_ext {
    template
    <class T>
    itpp::Vec<T>
    get_diag_square
    (
      const itpp::Mat<T> & in_mat ,
      int                  K = 0
    )
    {
      assert( in_mat.rows() == in_mat.cols() );
      assert(
        ( K < +in_mat.rows() ) &&
        ( K > -in_mat.rows() ) );

      using namespace itpp;
      if ( K == 0 )
        return diag(in_mat);
      if ( K>0 )
        return diag( in_mat.get(
          0, -1-K+in_mat.rows(), K, -1+in_mat.cols()) );
      if ( K<0 )
        return diag( in_mat.get(
          -K, -1+in_mat.rows(), 0, -1+K+in_mat.cols()) );
    }
```

Bir template fonk. oluşumu yapılmış assert macrolar ile exception'lar atılmış matrix için şart olan kurallar işletilmiş bunlar satır sayısı == sütun sayısı olması şart ve satır veya sütun sayıları her ikisi'de sıfırdan büyük olmalıdır bunun check'i de yapılmış.

Bu fonksiyon (get_diag_square ve triu_square) içinde daha önce 4.sayfada bahsedilen iş yapılıyor sadece K==0 ve K<0 case'lerine girilebiliyor. Bu iki fonksiyon tam olarak anlaşılmamıştır fakat bir tahmin üzerine ilerleme yapılmıştır.

```
49  template
50  <class T>
51  itpp::Mat<T>
52  triu_square
53  (
54    const itpp::Mat<T> & in_mat,
55    int                  K = 0
56  )
57  {
58    assert( in_mat.rows() == in_mat.cols() );
59    using namespace itpp;
60
61    auto out_mat
62      = zeros(in_mat.rows(), in_mat.cols());
63    if ( K >=  0 + in_mat.rows() )
64      return out_mat;
65    if ( K <= +1 - in_mat.rows() )
66      return in_mat;
67
68    for ( int jj=K ; jj<in_mat.rows() ; ++jj )
69      out_mat +=
70        itpp::diag(
71          itpp_ext::get_diag_square(in_mat, jj) ,
72          jj );
73
74    return out_mat;
75  }
76
```

```
77  void
78  print_matrix
79  (
80    const itpp::mat & in_mat
81  )
82  {
83    for ( int ii=0 ; ii<in_mat.rows() ; ++ii )
84    {
85      for ( int jj=0 ; jj<in_mat.cols() ; ++jj )
86        printf( "  %11.6f" , in_mat(ii, jj) );
87      printf( "\n" );
88    }
89  }
90
91  void
92  print_ivec
93  (
94    const itpp::ivec & in_ivec
95  )
96  {
97    for ( int ii=0 ; ii<in_ivec.size() ; ++ii )
98    {
99      printf( "  %4d" , in_ivec(ii) );
100   }
101   printf( "\n" );
102 }
103 } // namespace
104
```

Print_matrix içerisinde gelen double matrix tipindeki matrix'in ekrana bastırılmasını sağlayan fonksiyon yazılmıştır, const referans olarak gelen matrix read-only'dir, değiştirimesi mümkün değil. Ekrana nasıl bastırmak istiyorsak o şekilde iç içe for yapısı kullanarak printf'ler yazılmıştır.

İn_mat(ii, jj) kullanımı python'dakine benzemektedir, C++'de 2D array'lerde array[ii][jj] yazıyorduk. ".rows ve .cols" ile ilgili matrix'in satır sütun sayılarına erişebiliyoruz. "%11.6f" ile basılacak ger bir double değişkenin ekrana hangi formatta virgülden sonra kaç basamak gösterelim ve bu gösterim kaçlık alanda yapılsın (setw(11)) gibi bir şey.

Print_ivec fonksiyonunda ise vector tipindeki bir değişkenin ekrana nasıl bastıralacağı tasarlanmıştır. Std::vector'de in_vec(ii) diyemiyorduk in_vec.at(ii) veya in_vec[ii] demek zorundaydık burada itpp::ivec değişkeni bu şekilde indexleniyor.

```
104
105 class MatrixDecomposer
106 {
107 public:
108   using lup_tuple_type
109     = std::tuple
110         <itpp::mat, itpp::mat, itpp::mat, itpp::ivec>;
111   using vec_of_lup_tuples_type
112     = std::vector<lup_tuple_type>;
113
```

MatrixDecomposer isimli bir class tanımı yapılmıştır, içerisinde birçok member tututalacaktır. PLU Factorization yapılacağından bir tuple içerisinde matrix'in kendisini, bu matrix'in U factor'ü, L factor'ü ve P factör'ü tutulacaktır. C++'da birbiri ile ilişkili birden fazla variable'lar bir tuple içerisinde tutulabilir.

108.satırda using keyword'ü aracılığı ile bir alias kullanımı söz konusudur, itpp::mat double'lardan oluşan bir matrix type'dır, itpp::ivec ise içersinde integer değerler tutan bir vektör tipidir. (EK 1'deki tablodan bakarak anlaşılmıştır sayfa 8).

110.satır neden öyle belirlendi çünkü A bir matrix, L ve U da bir matrix aslında P'de bir matrix ama her bir satırının ilgili sütününda 1 tane 1 bulunduran diğer elemanların hepsinin sıfır olduğu bir matrix'tir. P factor matrix olarak gösterimi tercih edilmemiş.

111.basit bir lup_tuple_type of vector tanımı yapılmıştır o tipten itemları tutacak bir vector tanımı yapılmış neden bu yapılmış çünkü birden fazla matrix için PLU Factor'leri hesaplanacak her biri vektörün bir index'inde tutulacak.

```
113
114 private:
115   int  no_entries_;
116   bool flag_symmetric_;
117
118   itpp::mat matrix_;
119
120   std::vector<itpp::mat>
121     vec_triu_matrices_;
122   vec_of_lup_tuples_type
123     vec_lup_;
```

Class'ın private memberlar'ı olarak kaç tane matrix için PLU factorleri bulunacaktır sorusunun cevabını bize no_entries verecektir. "flag_symmetric_" çok kullanılmamış ama bir boolean değer de tutulmak istenmiş. (Ekleme her bir matrix'in boyutu da bu no_entries'e bağlıdır)

İtpp::mat tipinde bir matrix tanımlanmıştır class içerisinde 1 tane matrix üretilecek ve bu matrix üzerinde değişiklikler ile 5 tane matrix belirlenecektir.

120.satırda triangular matrix'leri tutacak bir vektör tanımı görüyoruz, son olarak ta her şeyi (matrix ve PLU factors) tutacak bir tuple değişkeni görüyoruz.

```
125 public:
126   MatrixDecomposer
127   (
128     int  no_entries,
129     bool flag_symmetric = true
130   )
131   :
132   no_entries_(no_entries) ,
133   flag_symmetric_(flag_symmetric)
134   {
135     using namespace itpp;
136     matrix_ = itpp::randu(no_entries_, no_entries_);
137     if (flag_symmetric_)
138       matrix_ = (matrix_ + transpose(matrix_)) / 2.;
139
140     vec_triu_matrices_.clear();
141     vec_lup_.clear();
```

126.satıdan itibaren başlayan bir constructor yapısı vardır. Burada flag'e default değer ataması yapılmış ve matrix sayısı değişkeni bir diğer input parametresidir. 132, 133. private member'lara ilk değer kazandırıyoruz.

135.satırda itpp namespace'i koda çağırılmıştır muhtemelen beraberinde getirdiği birçok özellik kullanılacaktır.

136.satırda (0,1) arasında toplam 5x5 tane random sayı üretilmiştir burada bir tane 5x5 matrix oluşturulmuş ve double tipinde random elemanlara sahiptir. (EK 2 sayfa 8)

137.satırdaki if'e giriyoruz ve bu matrix üzerinde bir işlem yapılıyor. Buradaki transpose napıyor? Matrix'in transpose'sini alıyor ve transpose edilmiş halini return ediyor. (EK 3 sy. 9) büyük ihtimal ile itpp namespace'i içindeki fonksiyon kullanılıyor fakat itpp::mat<>::transpose şeklinde de bir kullanım varmış.

140 ve 141. satırda private member olarak tanımlanan ve triangular matrix'leri tutması gereken değişken ve tuple'ları tutması gereken değişkenler ".clear()" ile sıfırlanmış temizlenmiş içlerinde ne varsa silinmiş ama zaten içlerinin boş olması lazım burada garanti altına alınmış.

```
141       vec_lup_.clear();
142       for (int kk=0 ; kk>-matrix_.rows() ; --kk)
143       {
144         vec_triu_matrices_
145           .push_back(
146             itpp_ext::triu_square(matrix_, kk) );
147
148         mat  L, U;
149         ivec p;
150         itpp::lu(vec_triu_matrices_.back(), L, U, p);
151         vec_lup_
152           .emplace_back(vec_triu_matrices_.back(), L, U, p);
153       }
154     }
```

142.satırda 0'dan -5'e kadar yapılacak iterasyon gerçekleşecektir yani toplam 5 tane iterasyon olacak. 5 tane matrix oluşacağını ve bunların PLU factor'lerinin hesaplanacağını ve aynı zamanda her bir matrix'in 5x5 olacağını biliyorduk.

Triangular matrix lower ve upper olarak ikiye ayrılıyordu çıktı incelendiğinde Matrix 1 için tam bir upper matrix yapısı görüyoruz matrix 2 , 3, 4 ,5'e doğru gittikçe aslında upper'ın simetriğinin bir kopyası lower triangular matrix olarak yansıtılmış olduğunu görüyoruz matrix 5'de. Bu iteratif olarak gerçekleşmiştir, Matrix1 (1,5)'deki değeri Matrix5'de (5,1)'de görüyoruz.

```
Matrix 1:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.000000     0.929579     0.211344     0.729242     0.392897
     0.000000     0.000000     0.856167     0.794648     0.344741
     0.000000     0.000000     0.000000     0.616463     0.634687
     0.000000     0.000000     0.000000     0.000000     0.537693
```

```
Matrix 5:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.095850     0.729242     0.794648     0.616463     0.634687
     0.586983     0.392897     0.344741     0.634687     0.537693
```

Her bir matrix için örneğin ilk iterasyon için triu_square(matrix_, -1) gönderilmiş ve sonucunda yukarıdaki Matrix1 elde edilmiş.

148. ve 149. Satırlar nasıl öyle kullanılıyor? itpp::mat ve itpp::ivec yazılması beklenirdi namespace itpp bunu sağlıyor olabilir yani içerisinde bu gibi tipler için alias'lar yapılmış olabilir.

150.satırda itpp::lu ile giriş matrisi için LU factorization of real matrix bulunuyor (EK 4 sy. 9) input parametreleri mat tipinde giriş matrisi, mat tipinde Lower triangular matrix, mat tipinde Upper triangular matrix ve ivec tipinde Permutation Matrix'tir.

148 ve 149.satırlardaki değişkenler direkt olarak itpp::lu içine atıldı büyük ihtimal bu fonksiyon sonucunda bize bu değerler değer kazanmış olarak return edilecektir , yani bu işlem sonucunda ilgili matrix için Lower Upper ve Permutation matrix'ler bulunmuş ve return edilmiş olmalıdır.

151.satırda matrisi ve matrisin sahip olduğu her şeyi (PLU) tuple'a dolduruyoruz.

```
155
156     ~MatrixDecomposer() {}
157
158     const
159     std::vector<itpp::mat> &
160     get_vector_of_matrices
161     ()
162     const
163     {
164       return vec_triu_matrices_;
165     }
166
167     const
168     vec_of_lup_tuples_type &
169     get_vector_of_lup
170     ()
171     const
172     {
173       return vec_lup_;
174     }
175   };
176
177 } // namespace
```

156.satırda neden ‘ = default’ yazılmadı? Büyük ihtimalle dynamically allocated her hangi bir şey kodda kullanılmadı “heap” kullanılmamıştır.

158. ve 167.satırlarda triangular_matrislerin tutulduğu vector ve her şeyin tutulduğu tuple’ların getter’leri yazılmıştır. Neden getter yazıldı? Çünkü bunlar private koda çağırmak için getter lazım direkt erişemeyiz main’de. Getter’ların const yazılması daha iyidir private member’ın kesinlikle değişmeyeceği garanti altına alınıyor.

```
178
179 int
180 main (void)
181 {
182   using namespace os_test;
183   using namespace std;
184
185   MatrixDecomposer oa(5);
186   int cnt;
187
188   const auto & vec = oa.get_vector_of_matrices();
189   cnt = 0;
190   for ( const auto & item : vec )
191   {
192     cout << "Matrix " << ++cnt << ":" << endl;
193     itpp_ext::print_matrix(item);
194   }
195   cout << endl;
196
197   const auto & vec_lup = oa.get_vector_of_lup();
198   cnt = 0;
```

Main içinde kullanılacak namespace’ler çağırılıyor,

185.satırda class’ın instance’si oluşturuluyor ve constructor’una 5 yollanıyor (5 tane matrix oluşturulacak ve her matrix 5x5 boyutunda olacak).

188.satırda oluşturulan 5 matrisin tutulduğu private vektör değişkeninin getter’ı vec değişkenine atanmıştır. Const auto ref tercih edilmiş (read-only)

190 ile 194.satırlarda oluşan bu 5 matrix ekrana bastırılmıştır. “itpp_ext” içerisinde print_matrix() fonksiyonu kullanarak bu iş yapılmıştır.

190.satırda kullanılan for normalde range-based for’dur ancak burada cnt değişkeni bu for’u aynı zamanda index based olarak ta kullanmamıza olanak sağlamıştır.

197.satırda her şeyin(matrix and its PLU) tutulduğu tuple yine aynı yukarıda olduğu gibi const auto ref şekilde vec_lup değişkenine atanmıştır. Getter’i kullanarak main’e çağırabiliyoruz. “auto” keyword’ü ile bu getter’in return ettiği değişkenin ne olduğunu bilmeme gerek yok auto büyük bir zahmetten bizi kurtarıyor. “auto” kullanmak yerine başla ne kullanabilirdik gidip? 108.satırdaki using keyword’ü aracılığı ile oluşturulan alias’ı kullanmak gerekecekti.

“cnt” ile işimiz bitti onu tekrar sıfırladık muhtemelen başka bir yerde tekrar kullanacağız.

```
199   for ( const auto & item : vec_lup )
200   {
201     ++cnt;
202     std::stringstream ss;
203     ss << "Matrix " << cnt;
204
205     cout << ss.str() << ":" << endl;
206     itpp_ext::print_matrix( std::get<0>(item) );
207     cout << ss.str() << " (L factor):" << endl;
208     itpp_ext::print_matrix( std::get<1>(item) );
209     cout << ss.str() << " (U factor):" << endl;
210     itpp_ext::print_matrix( std::get<2>(item) );
211     cout << ss.str() << " (p factor):" << endl;
212     itpp_ext::print_ivec( std::get<3>(item) );
213
```

"cnt" kullanarak yine range-based for'u index-based olarak da kullanabiliyoruz. "stringstream" kullanılmış, << ss.str() yerine << "Matrix " << cnt' de kullanılabilirdi bu tercih edilmemiş.

Her şeyin tutulduğu tuple içinden std::get<>(tuple) kullanarak itemler'i çekiyoruz. Buradaki vec_lup bir vector içerisindeki her bir item bir tuple tutuyor, dolayısı ile for içinde kullanılan item bir tuple'a karşılık gelmektedir.

Her bir item yani her bir tuple içinden bu tuple içinde ne varsa tek tek std::get<>(tuple) kullanarak çekiyoruz ve ekrana bastırıyoruz, matrix tiplerindeki değişkenleri bastırmak için print_matrix, vec tipindeki değişkenleri bastırmak için ise print_ivec kullanılacaktır. Bu iki fonksiyon tek bir fonksiyon olarak yazılabilir miydi? Cevap evet burada std::get<x>(item) buradaki x'in ne olduğunu bilmek gerek biz bildiğimiz için x = 3 olduğunda hangi değişkenin bize return edileceğini bildiğimiz için print_ivec kullandık bunu bilmiyor da olabilirdik. Fakat bu iki fonksiyonu tek bir fonksiyonda yazmak uzun uğraş gerektirecektir.

```
214     auto P =
215       itpp::to_mat(
216         itpp::permutation_matrix( std::get<3>(item) ) );
217     cout << ss.str() << " (relative error):" << endl;
218     printf( "  %.6e\n" ,
219       itpp::norm(
220         std::get<0>(item)
221         - itpp::transpose(P)
222         * std::get<1>(item) * std::get<2>(item) )
223       /
224       itpp::norm(
225         std::get<0>(item) )
226     );
227   }
228
229   return 0;
230 }
231
```

214.satırdan başlayan yapıda yapılan şudur; itpp::permutation_matrix() (EK 6 sy. 9)fonksiyonunu kullanarak bizim ivec tipinde tuttuğumuz permutation vector'ünden, permutation matrix elde et (interchange) yer değiştirme yap bunun sonucunda elde edilen matrix itpp::to_mat ile matrix tipine cast edilmiştir. (EK 5 sy. 9)

219.satırdan 225.satıra kadar bu permutation matrix'inin norm hesabı iki farklı yoldan yapılmış ve bu iki farklı yoldan elde edilen değerlerin birbiri ile oranı relative error olarak bastırılmıştır. Burada lineer algebra'nın bize söylediği

$$\mathbf{X} = \mathbf{P}^T\mathbf{L}\mathbf{U},$$

Bu özellik test edilmiştir. 219.satıda zaten X-X gerçekleşmesini ve payın sıfır olmasını bekliyoruz. İtpp:norm(0)'ında sıfır vermesi gerekir.

Output:

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_5$ g++ lu_decomp_itpp_test.cc -li
tpp
lu_decomp_itpp_test.cc: In function ‘itpp::Vec<T> os_test::itpp_ext::get_dia
g_square(const itpp::Mat<T>&, int) [with T = double]’:
lu_decomp_itpp_test.cc:47:5: warning: control reaches end of non-void functi
on [-Wreturn-type]
   47 |     }
      |     ^
soray@soray-VirtualBox:~/Desktop/433_hw/hw_5$ ./a.out
Matrix 1:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.000000     0.929579     0.211344     0.729242     0.392897
     0.000000     0.000000     0.856167     0.794648     0.344741
     0.000000     0.000000     0.000000     0.616463     0.634687
     0.000000     0.000000     0.000000     0.000000     0.537693
Matrix 2:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.000000     0.211344     0.856167     0.794648     0.344741
     0.000000     0.000000     0.794648     0.616463     0.634687
     0.000000     0.000000     0.000000     0.634687     0.537693
Matrix 3:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.000000     0.729242     0.794648     0.616463     0.634687
     0.000000     0.000000     0.344741     0.634687     0.537693
Matrix 4:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.095850     0.729242     0.794648     0.616463     0.634687
     0.000000     0.392897     0.344741     0.634687     0.537693
Matrix 5:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.095850     0.729242     0.794648     0.616463     0.634687
     0.586983     0.392897     0.344741     0.634687     0.537693
```

```
Matrix 3:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.000000     0.729242     0.794648     0.616463     0.634687
     0.000000     0.000000     0.344741     0.634687     0.537693
Matrix 3 (L factor):
     1.000000     0.000000     0.000000     0.000000     0.000000
     0.000000     1.000000     0.000000     0.000000     0.000000
     0.348644    -0.022932     1.000000     0.000000     0.000000
     0.825031     0.534639    -0.567565     1.000000     0.000000
     0.000000     0.000000     0.443105     0.382722     1.000000
Matrix 3 (U factor):
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.000000     0.729242     0.794648     0.616463     0.634687
     0.000000     0.000000     0.778013     0.775368     0.154647
     0.000000     0.000000     0.000000     0.760650    -0.342938
     0.000000     0.000000     0.000000     0.000000     0.600417
Matrix 3 (p factor):
     0     3     2     3     4
Matrix 3 (relative error):
  0.000000e+00
```

```
Matrix 4:
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.654155     0.929579     0.211344     0.729242     0.392897
     0.276435     0.211344     0.856167     0.794648     0.344741
     0.095850     0.729242     0.794648     0.616463     0.634687
     0.000000     0.392897     0.344741     0.634687     0.537693
Matrix 4 (L factor):
     1.000000     0.000000     0.000000     0.000000     0.000000
     0.120887     1.000000     0.000000     0.000000     0.000000
     0.348644    -0.025721     1.000000     0.000000     0.000000
     0.825031     0.599666    -0.607167     1.000000     0.000000
     0.000000     0.604306    -0.147908     0.505941     1.000000
Matrix 4 (U factor):
     0.792885     0.654155     0.276435     0.095850     0.586983
     0.000000     0.650163     0.761231     0.604876     0.563728
     0.000000     0.000000     0.779369     0.776789     0.154593
     0.000000     0.000000     0.000000     0.759081    -0.335567
     0.000000     0.000000     0.000000     0.000000     0.389671
Matrix 4 (p factor):
     0     3     2     3     4
Matrix 4 (relative error):
  4.309947e-17
```

EKLER:

EK1:

Prepared Vector Types

A vector can in principle be of arbitrary type (that support addition, subtraction, multiplication and division), since the general vector class `Vec<TYPE>` is templated. However, the most commonly used vector types are predefined. These predefined vector types are:

- `vec`: Basic vector type containing `double`
- `cvec`: Vector type containing `complex<double>`
- `ivec`: Vector type containing `int`
- `bvec`: Vector type containing `bin`
- `svec`: Vector type containing `short`

The general vector class is used to define the specialized classes above. The `vec` class is actually a `Vec<double>`. We urge you to use these predefined classes instead of `Vec<TYPE>` when ever possible.

Prepared Matrix Types

The general matrix class is called `Mat<TYPE>`. These predefined matrix types are:

- `mat`: Basic matrix type containing double
- `cmat`: Matrix type containing `complex<double>`
- `imat`: Matrix type containing `int`
- `bmat`: Matrix type containing `bin`
- `smat`: Matrix type containing `short`

As with vector, the general matrix class is used to define the specialized classes above. The `mat` class is thus a `Mat<double>`. We urge you to use these predefined classes instead of `Mat<TYPE>` whenever possible.

EK2:

◆ randu() [5/5]

double itpp::randu (void) inline

Generates a random uniform (0,1) number.

Definition at line 804 of file random.h.

References itpp::Uniform_RNG::sample().

Referenced by itpp::LDPC_Generator_Systematic::construct(), itpp::LDPC_Parity_Unstructured::cycle_removal_MGW(), fpica(), itpp::LDPC_Parity_Unstructured::generate_random_H(), get_ms(), getSamples(), itpp::Packet_Channel::handle_input(), itpp::ACK_Channel::handle_input(), itpp::Packet_Channel::handle_nof_inputs(), itpp::Rice_Fading_Generator::init_MEDS(), main(), itpp::Sequence_Interleaver< double >::randomize_interleaver_sequence(), itpp::randu(), itpp::Sequence_Interleaver< double >::Sequence_Interleaver(), TEST(), and itpp::wcdma_turbo_interleaver_sequence().

EK3:

◆ transpose()

template<class Num_T >
Mat< Num_T > **itpp::Mat**< Num_T >::transpose ( ) const

Matrix transpose.

Definition at line **1169** of file **mat.h**.

Referenced by **itpp::SISO::mmsePIC()**, **itpp::Mat< bin >::T()**, and **itpp::SISO::zfPIC()**.

EK4:

◆ lu() [2/2]

ITPP_EXPORT bool itpp::lu (const mat & X,
mat & L,
mat & U,
ivec & p
)

LU factorisation of real matrix.

The LU factorization of the real matrix $\mathbf{X}$ of size $n \times n$ is given by

$$\mathbf{X} = \mathbf{P}^T\mathbf{L}\mathbf{U},$$

where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices and $\mathbf{P}$ is a permutation matrix.

The interchange permutation vector *p* is such that *k* and *p(k)* should be changed for all *k*. Given this vector a permutation matrix can be constructed using the function

```
bmat permutation_matrix(const ivec &p)
```

If *X* is an *n* by *n* matrix *lu(X,L,U,p)* computes the LU decomposition. *L* is a lower triangular, *U* an upper triangular matrix. *p* is the interchange permutation vector such that *k* and *p(k)* should be changed for all *k*.

Returns true is calculation succeeds. False otherwise.

Definition at line **117** of file **lu.cpp**.

References **it_error**.

Referenced by **itpp::det()**, and **TEST()**.

EK5:

◆ to_mat() [1/2]

template<class T >
mat to_mat (const **Mat**< T > & m) related

Converts a **Mat<T>** to mat.

Definition at line **216** of file **converters.h**.

◆ to_mat() [2/2]

template<class T >
mat to_mat (const **Mat**< T > & m)

Converts a **Mat<T>** to mat.

Definition at line **216** of file **converters.h**.

References **check_tests::i**, **j**, and **m**.

Referenced by **main()**, **itpp::operator*()**, **itpp::operator+()**, **itpp::operator-()**, **itpp::operator/()**, **itpp::rank()**, **TEST()**, and **itpp::unfix()**.

EK6:

◆ permutation_matrix()

ITPP_EXPORT **bmat** itpp::permutation_matrix (const **ivec** & p)

Make permutation matrix P from the interchange permutation vector p.

Definition at line **144** of file **lu.cpp**.

References **bvec**, **itpp::eye_b()**, **it_assert**, **n**, and **P**.

Referenced by **TEST()**.