```
int factorBasic(int n,  double *a, int LDA ){
  int k=0;
  double piv=a[0];
  const double minpiv=1e-6;
  while((fabs(piv)>minpiv) && (k<n-1))
  {
      for(int i=k+1; i<n; i++)
      {
          a[i+k*LDA]=a[i+k*LDA]/piv;
      }
      for (int i=k+1; i<n; i++)
      {
          for (int j=k+1; j<n; j++)
          {
              a[i+j*LDA] =a[i+j*LDA]-a[i+k*LDA]*a[k+j*LDA];
          }
      }
      k +=1;
      piv=a[k+k*LDA];
  }
  if(fabs(piv)<=minpiv)
  {
      std::cout<<"Null point in dLU1: "<<piv<<__FILE__<<":"<<__LINE__<<std::endl;
      return 0;
  }
  return 1;
```

```
int factorL2(int n,  double *a, int LDA ){
    int k=0;
    const double minpiv=1e-6;
    double piv =a[0];
    while((fabs(piv)>minpiv) && (k< n-1))
    {
        dscal_(n-(k+1),1./piv, a+(k+1+k*LDA),1);
        dger_(n-(k+1), n-(k+1), -1., a+(k+1+k*LDA), 1, a+(k+(k+1)*LDA), LDA, a+(k+1+(k+1)*LDA), LDA);
        k +=1;
        piv=a[k+k*LDA];
    }
    if (fabs(piv)<=minpiv)
    {
        std::cout<<"Null point in dLU2: "<<piv<<__FILE__<<":"<<__LINE__<<std::endl;
        return 0;
    }
    return 1;
};
```

```
int factorL3(int r, int n,  double *a, int LDA ){
  int l=0;
  while(l<n)
  {
     int m = std::min(n, l+r);
      int bsize = m-l;
      int success=factorL2(bsize, a+(l+l*LDA), LDA);
      if (!success)
      {
          std::cout<<"Can't fatorize one block" <<std::endl;
          return 0;
      }
      dtrsm_('L', 'L', 'N', 'U', bsize, n-m, 1.0, a+(l+l*LDA), LDA, a+l+m*LDA, LDA);
      dtrsm_('R', 'U', 'N', 'N', n-m, bsize, 1.0, a+(l+l*LDA), LDA, a+m+l*LDA, LDA);
      dgemm_('N','N', n-m, n-m, bsize, -1.0, a+m+l*LDA, LDA, a+l+m*LDA, LDA, 1.0, a+m+m*LDA, LDA);
      l=m;
```

```
  }
  return 1;
}
```

`computeBandwidthUp`

```
dmatrix_denseCM operator*(const dmatrix_denseCM &A, const dmatrix_denseCM &B){
  return muldgemm(A,B);
}

int computeBandwidthUp(const dmatrix_denseCM &A ){
    int start_i=0, max_i=0, min_j=0, m, UpperBandwidth=0;
    double sum=0.0, sumBuffer=0.0;
    m=A.getNbLines();
    int ceilrow=int(std::ceil(m));
    for (int j=0; j<ceilrow; j++)
    {
        sum=0.0;
        sumBuffer=0.0;
        int Test;
        for (int i=start_i; i<m; i++)
        {
            sum=sum+A(i,j);
            Test=(sum>sumBuffer);
            max_i=Test*i+!Test*max_i;
            min_j=Test*j+!Test*min_j;
            sumBuffer=sum;
            start_i=max_i;
        }
        start_i++;
        UpperBandwidth=std::max(UpperBandwidth, max_i-min_j);
    }
    std::cout<<"Calculated Upper Bandwith is: "<<UpperBandwidth<<"\n";
    return UpperBandwidth;
};
```

`computeBandwidthDown`

```
int  computeBandwidthDown(const dmatrix_denseCM &A ){
    int start_j=0, max_j=0, min_i=0, n, LowerBandwidth=0;
    double sum=0.0, sumBuffer=0.0;
    n=A.getNbColumns();
    int ceilcolumn=int(std::ceil(n));
    for (int i=0; i<ceilcolumn; i++)
    {
        sum=0.0;
        sumBuffer=0.0;
        int Test;
        for (int j=start_j; j<n; j++)
        {
            sum=sum+A(i,j);
            Test=(sum>sumBuffer);
            max_j=Test*j+!Test*max_j;
            min_i=Test*i+!Test*min_i;
            sumBuffer=sum;
            start_j=max_j;
        }
        start_j++;
        LowerBandwidth=std::max(LowerBandwidth, max_j-min_i);
    }
    //UpperBandwidth=max_i-min_j;
    std::cout<<"Calculated Lower Bandwith is: "<<LowerBandwidth<<"\n";
    return LowerBandwidth;
};
```

```
double& dsquarematrix_symband::operator()(int i, int j) {
    int formula=(lb-j+i)+j*(lb+1);
    return *(a+formula);
}
```

The results for different matrices for the second implementation can be seen as follows:

	Size	Factorisation time	Time	lbu	lbd	Error
data_band.mat	5	0.000232438	$2.937e-05$	2	2	0
bcsstk14.mtx	1806	0.531188	0.0121683	161	161	$1.25193e-11$
bcsstk15.mtx	3948	6.2573	0.130958	437	437	$7.14698e-10$

Table 1: The results for different matrices for Part 2