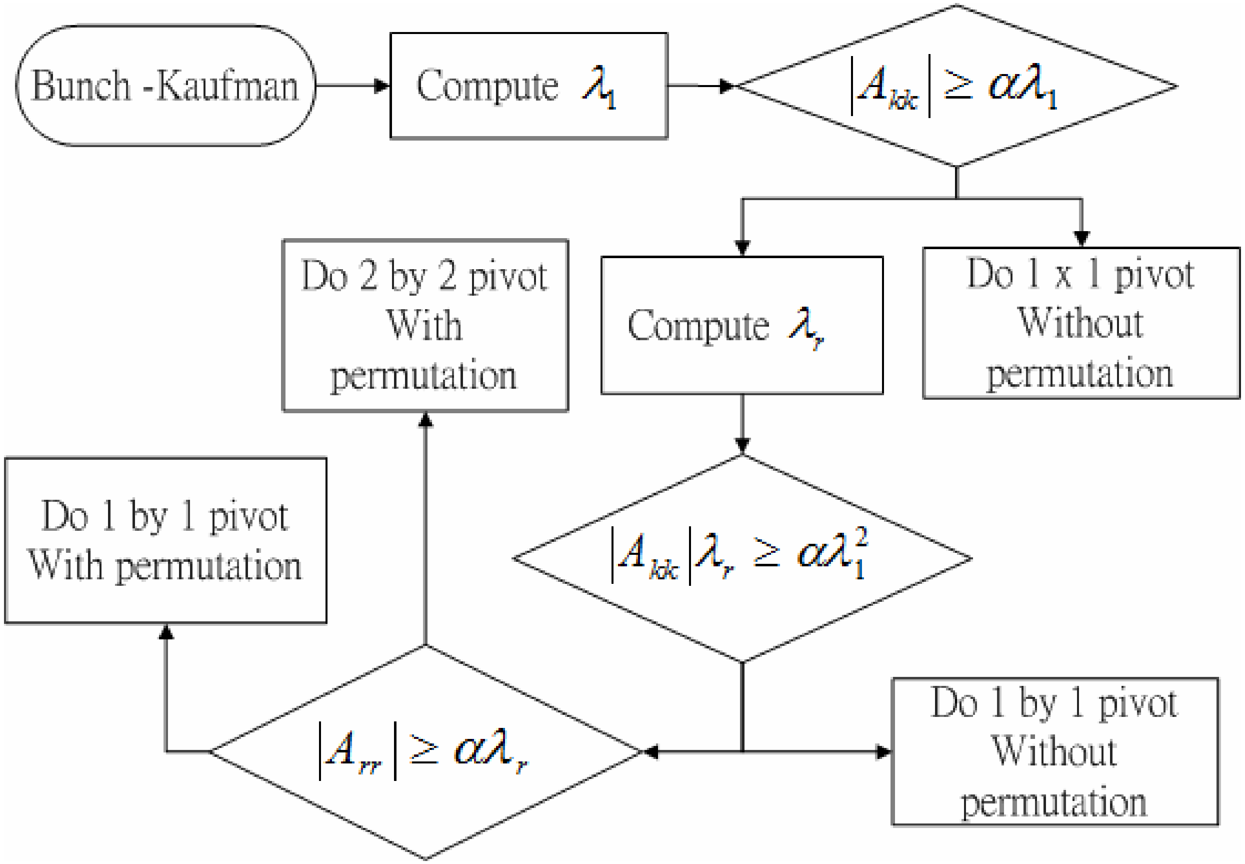
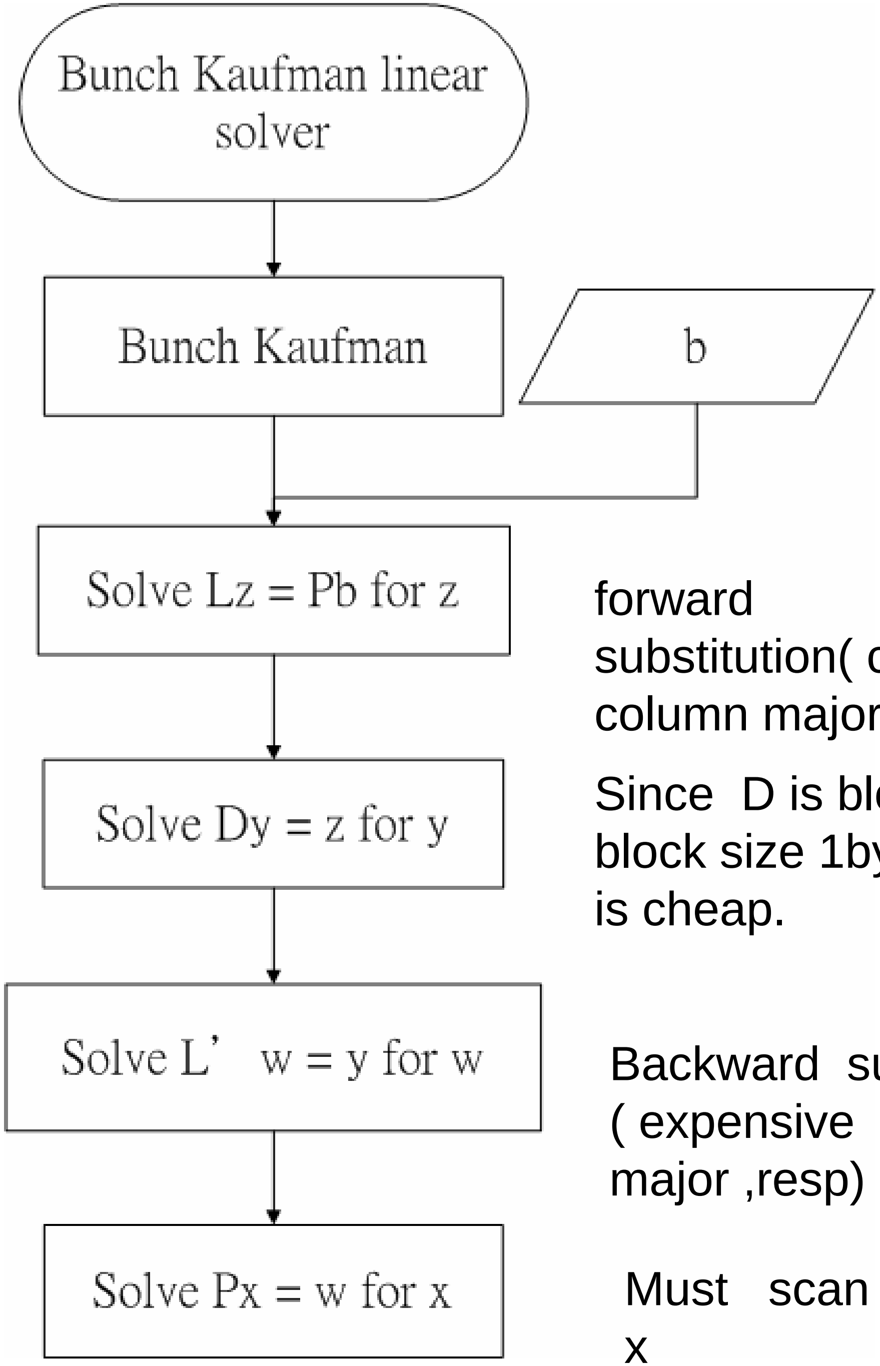
Report

- Implement PAP’ = LDL’ and linear solver in C/C++ (ok)
- describe program structure
- How to verify your program
- Comparison with MATLAB implementation
- Memory usage (do you need extra storage?)
- Speedup strategy

describe program structure


Bunch -Kaufman
Compute λ_1
$|A_{kk}| \geq \alpha\lambda_1$
Compute λ_r
Do 1 x 1 pivot Without permutation
Do 2 by 2 pivot With permutation
$|A_{kk}|\lambda_r \geq \alpha\lambda_1^2$
Do 1 by 1 pivot With permutation
$|A_{rr}| \geq \alpha\lambda_r$
Do 1 by 1 pivot Without permutation



forward substitution(cheap for column major,resp)

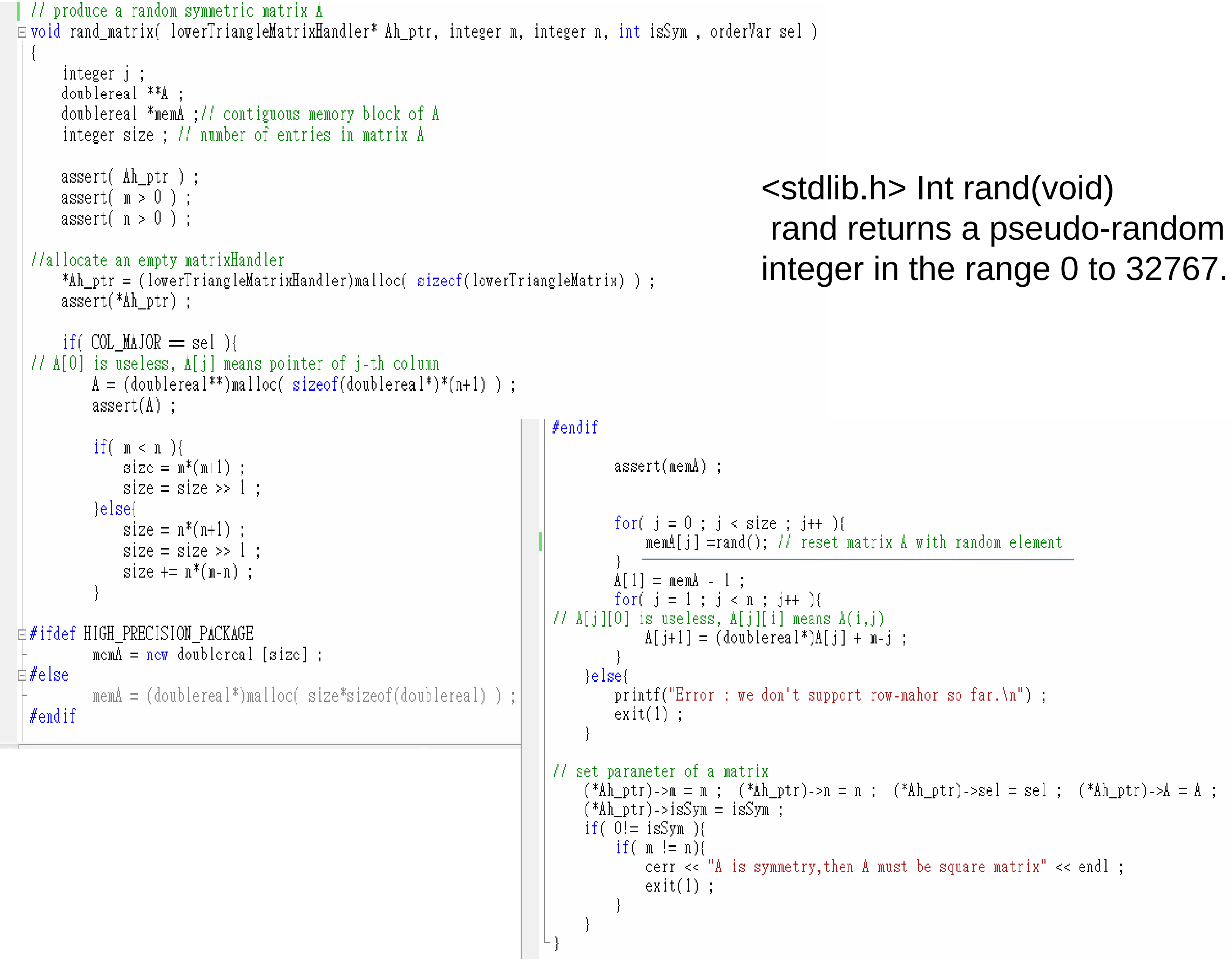
Since D is block diagnoal with block size 1by1 or 2 by2 , inv(D) is cheap.

Backward substitution (expensive for column major ,resp)

Must scan w to get x

How to verify your program

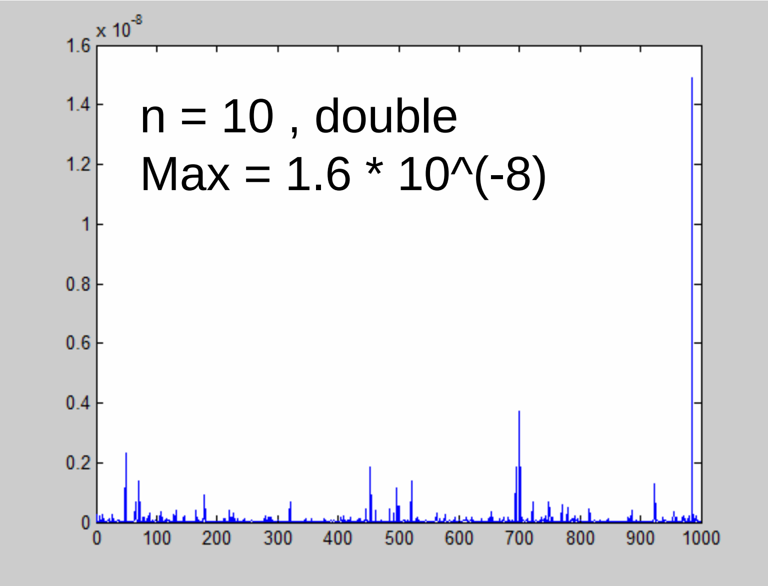
- I write a function named rand_matrix to produce "random" matrix . (In fact , it can only produce matrix with integer element between 0~32767)
- Use rand_matrix under double precision to test different matrix with size n = 10, 20, 30, 50, 100 . And plot the result 1000 time supnorm of residual r = b – Ax .
- Do the same thing above for different precision(double-double, quad-double, arbitrary precision) , then compare the reults .
- Since they are produced by psuedo-random, the test matrix are the same.



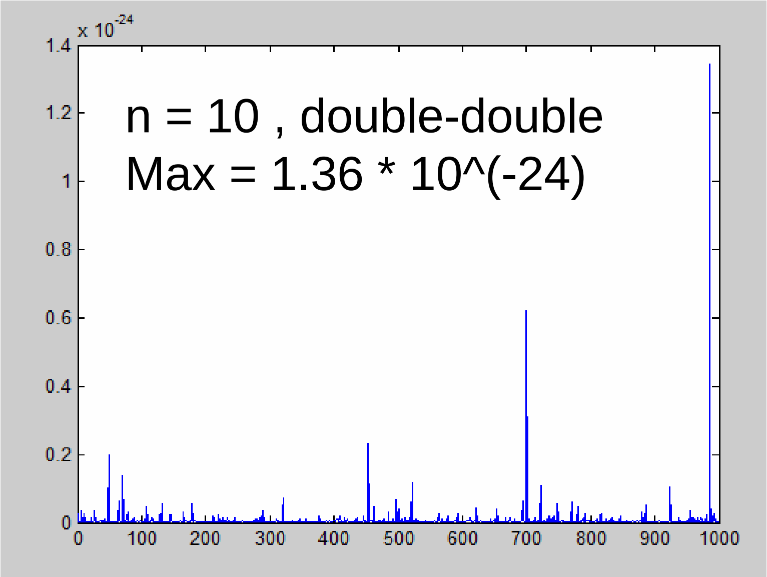
```
// produce a random symmetric matrix A
void rand_matrix( lowerTriangleMatrixHandler* Ah_ptr, integer m, integer n, int isSym , orderVar sel )
{
    integer j ;
    doublereal **A ;
    doublereal *memA ;// contiguous memory block of A
    integer size ; // number of entries in matrix A

    assert( Ah_ptr ) ;
    assert( m > 0 ) ;
    assert( n > 0 ) ;

//allocate an empty matrixHandler
    *Ah_ptr = (lowerTriangleMatrixHandler)malloc( sizeof(lowerTriangleMatrix) ) ;
    assert(*Ah_ptr) ;

    if( COL_MAJOR == sel ){
// A[0] is useless, A[j] means pointer of j-th column
        A = (doublereal**)malloc( sizeof(doublereal*)*(n+1) ) ;
        assert(A) ;

        if( m < n ){
            size = m*(m+1) ;
            size = size >> 1 ;
        }else{
            size = n*(n+1) ;
            size = size >> 1 ;
            size += n*(m-n) ;
        }

#ifdef HIGH_PRECISION_PACKAGE
        memA = new doublereal [size] ;
#else
        memA = (doublereal*)malloc( size*sizeof(doublereal) ) ;
#endif
```

```
#endif

        assert(memA) ;


        for( j = 0 ; j < size ; j++ ){
            memA[j] =rand(); // reset matrix A with random element
        }
        A[1] = memA - 1 ;
        for( j = 1 ; j < n ; j++ ){
// A[j][0] is useless, A[j][i] means A(i,j)
            A[j+1] = (doublereal*)A[j] + m-j ;
        }
    }else{
        printf("Error : we don't support row-mahor so far.\n") ;
        exit(1) ;
    }

// set parameter of a matrix
    (*Ah_ptr)->m = m ;  (*Ah_ptr)->n = n ;  (*Ah_ptr)->sel = sel ;  (*Ah_ptr)->A = A ;
    (*Ah_ptr)->isSym = isSym ;
    if( 0!= isSym ){
        if( m != n){
            cerr << "A is symmetry,then A must be square matrix" << endl ;
            exit(1) ;
        }
    }
}
```

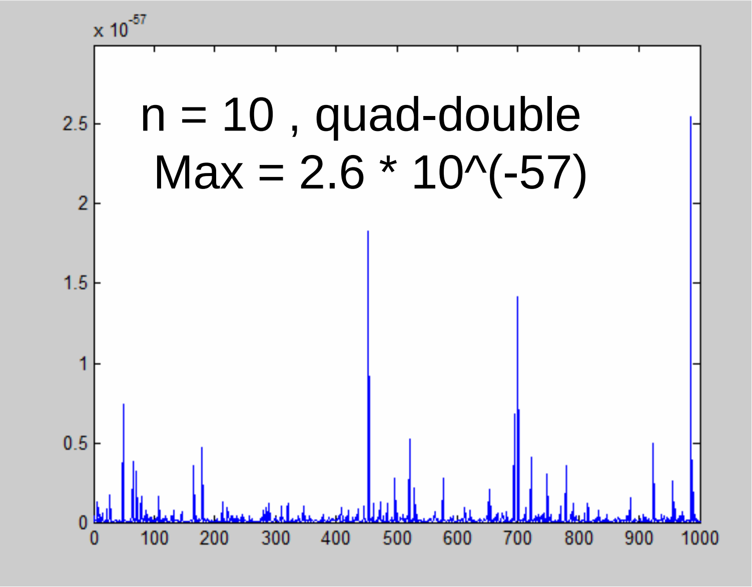
<stdlib.h> Int rand(void)
rand returns a pseudo-random integer in the range 0 to 32767.



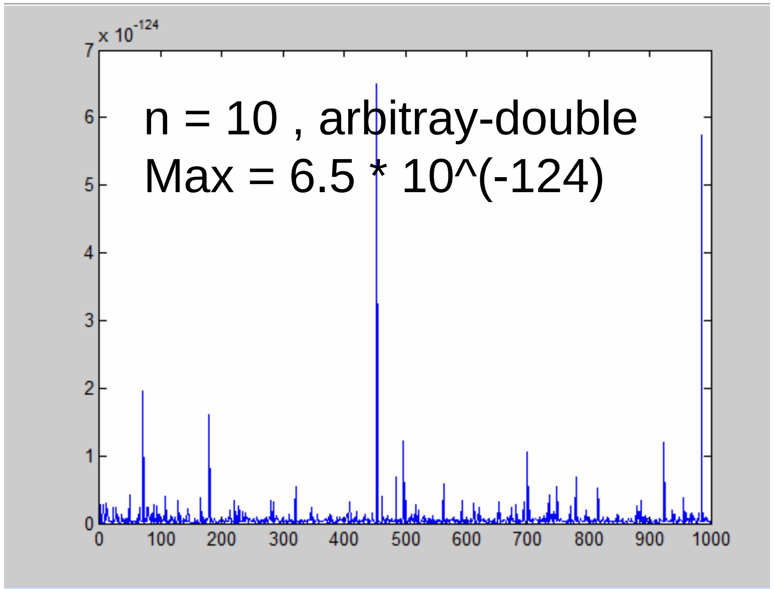
n = 10 , double
Max = 1.6 * 10^(-8)



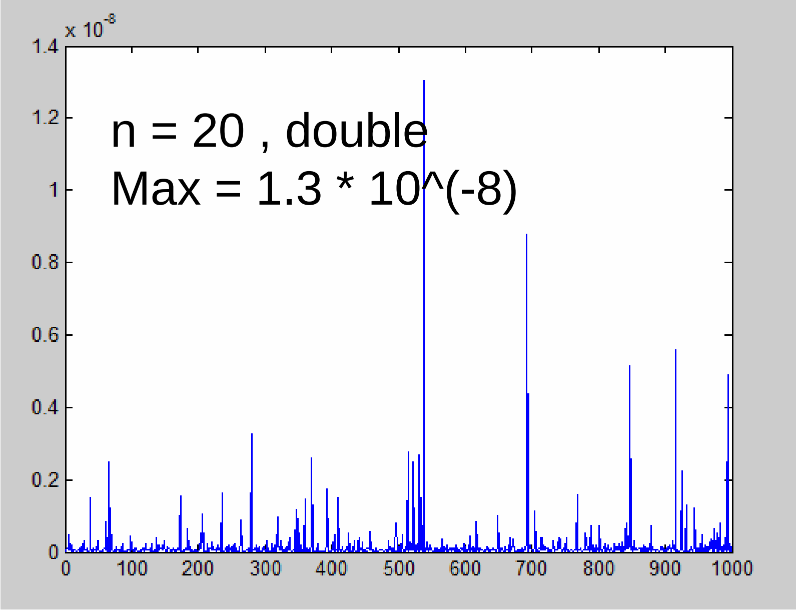
n = 10 , double-double
Max = 1.36 * 10^(-24)



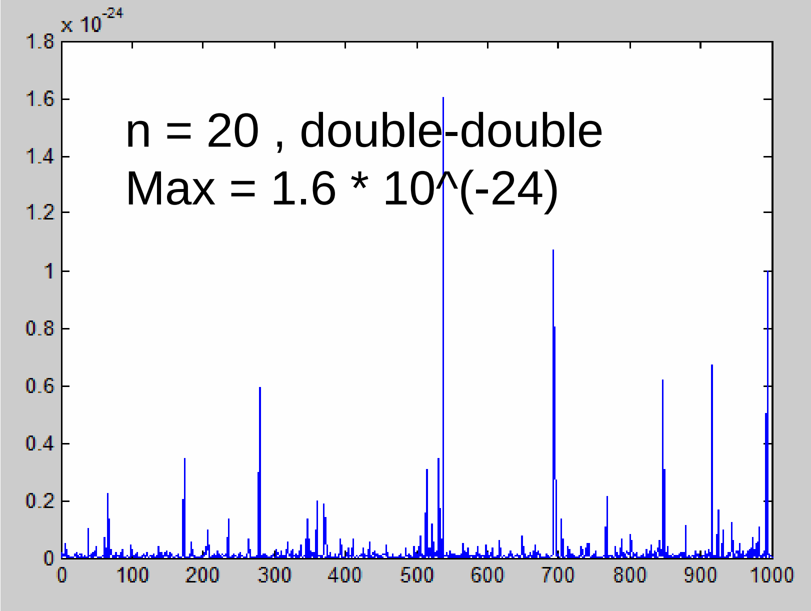
n = 10 , quad-double
Max = 2.6 * 10^(-57)



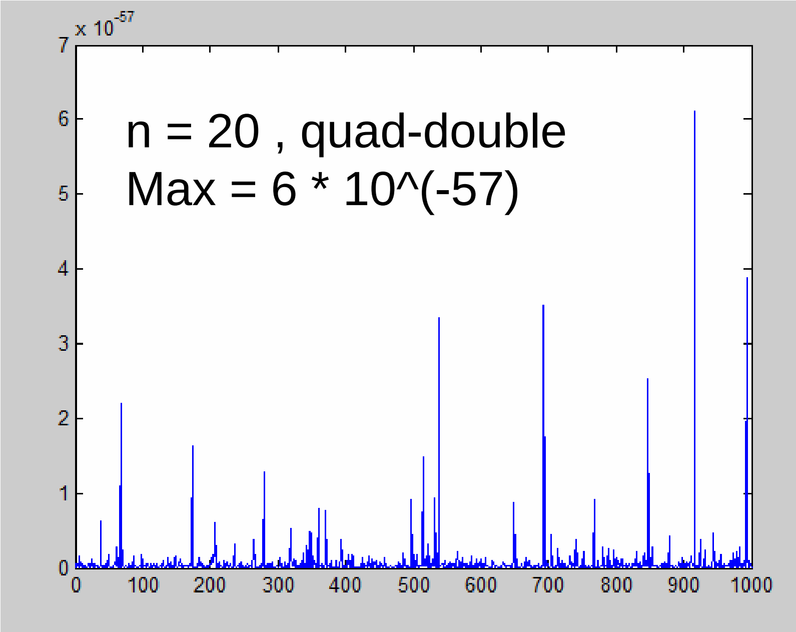
n = 10 , arbitray-double
Max = 6.5 * 10^(-124)



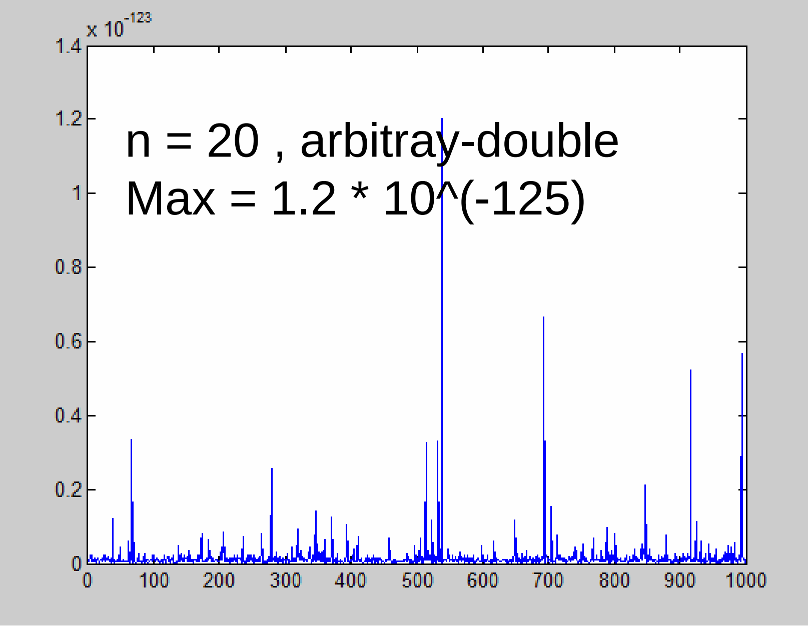
n = 20 , double
Max = 1.3 * 10^(-8)



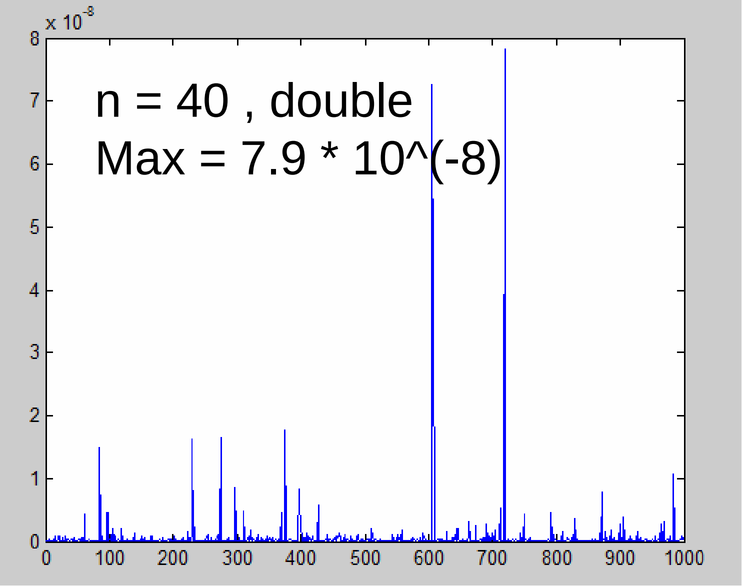
n = 20 , double-double
Max = 1.6 * 10^(-24)



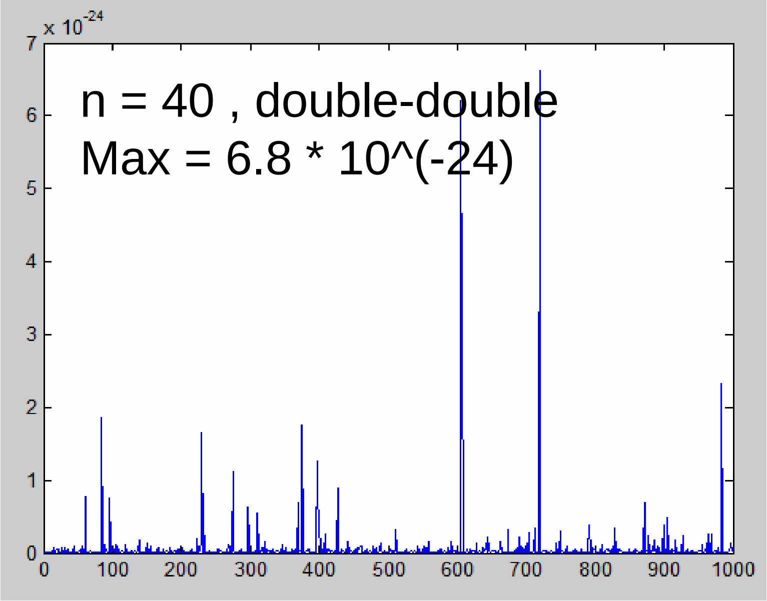
n = 20 , quad-double
Max = 6 * 10^(-57)



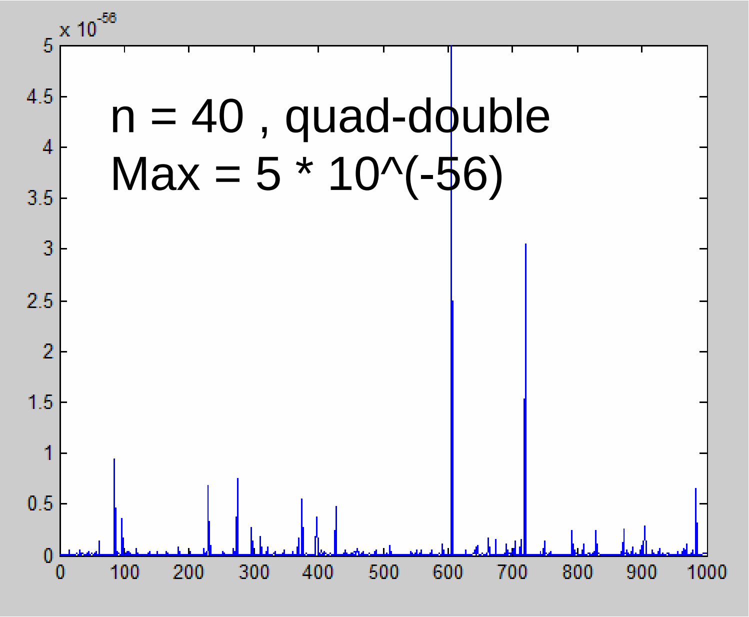
n = 20 , arbitray-double
Max = 1.2 * 10^(-125)



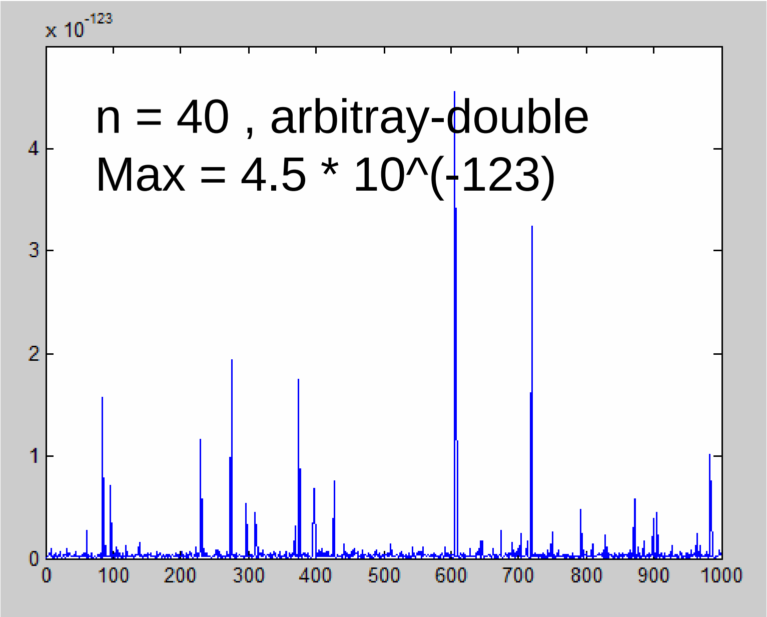
n = 40 , double
Max = 7.9 * 10^(-8)



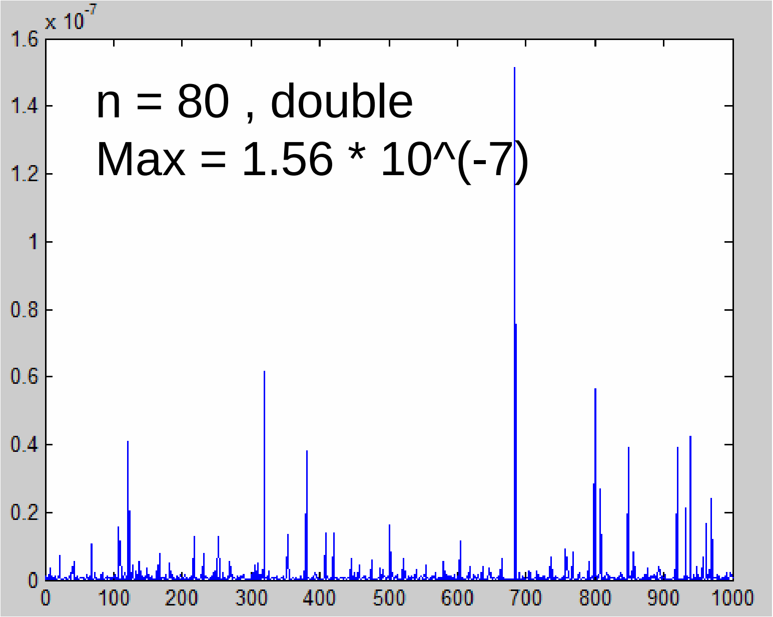
n = 40 , double-double
Max = 6.8 * 10^(-24)



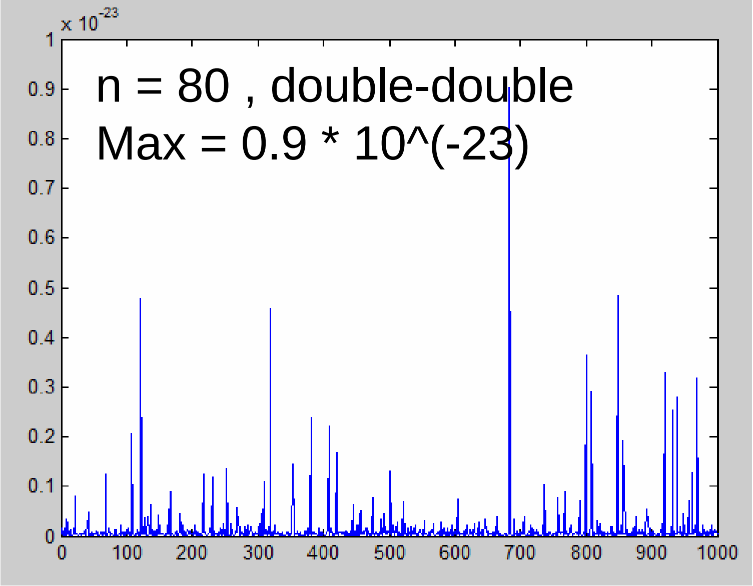
n = 40 , quad-double
Max = 5 * 10^(-56)



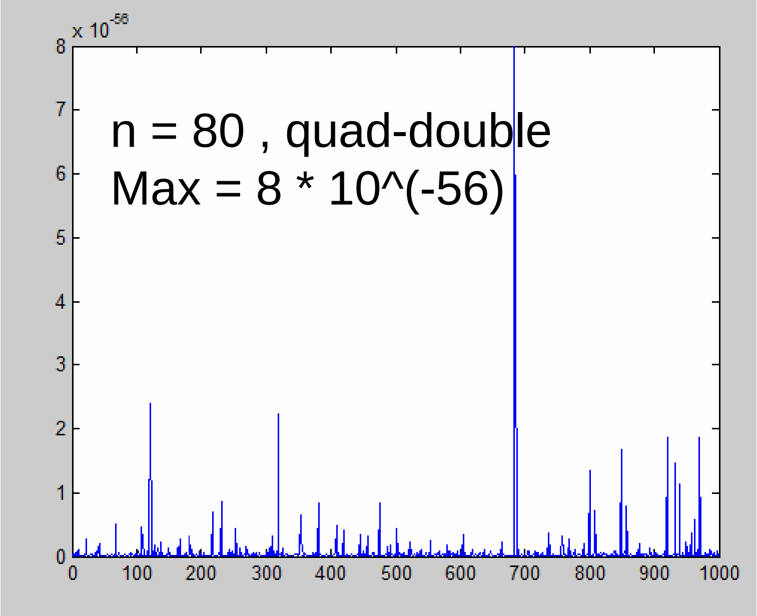
n = 40 , arbitray-double
Max = 4.5 * 10^(-123)


n = 80 , double

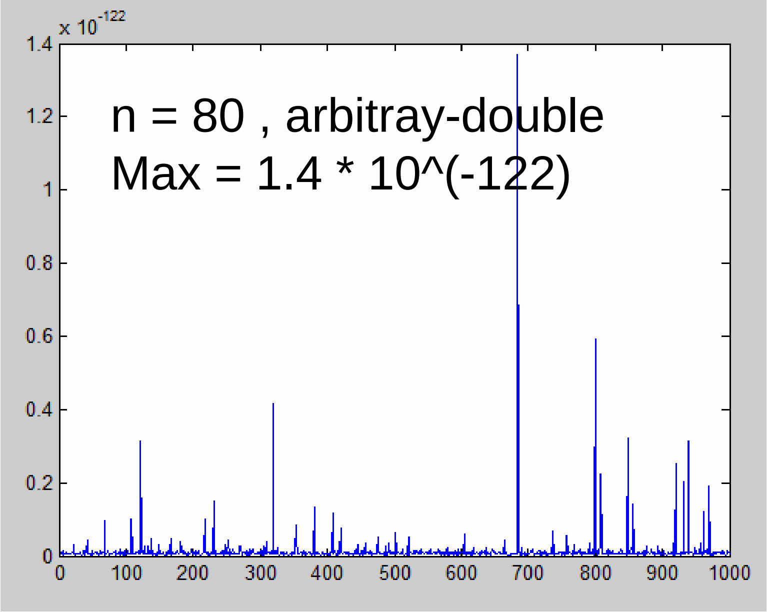
Max = 1.56 * 10^(-7)


n = 80 , double-double

Max = 0.9 * 10^(-23)


n = 80 , quad-double

Max = 8 * 10^(-56)


n = 80 , arbitray-double

Max = 1.4 * 10^(-122)

- Conclusion one :

  the code is deserve to trust, since with different precision (twice each time), the residual almost has the same magnitude improve .

- Conclusion two :

  when n large enough, the maximun residual will be increasing .(It is not true if you compare n = 10 with n = 20)

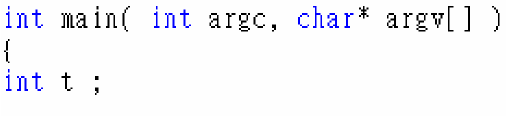
Comparison with MATLAB implementation

- In MATLAB, you don’t have to announce variable before you use it , but in C , you must to .
- In MATLAB, it much easier to find error(maybe just for me), since it is easy to show the thing you want to know on the screen . But in C , you need to write some code.
- In MATLAB ,it is very easy to create a matrix .Moreover, if you want to copy a vector , for example x which is a nx1 vector , you can wite   y(1:n ,1) = x(1:n,1) in stead of a “for loop “ .
- But in MATLAB , you can’t create a matrix only use storage of a lowertriangular. In C, you can create many kind of structure .
- And in C , you can use high-precision package to justify your code .

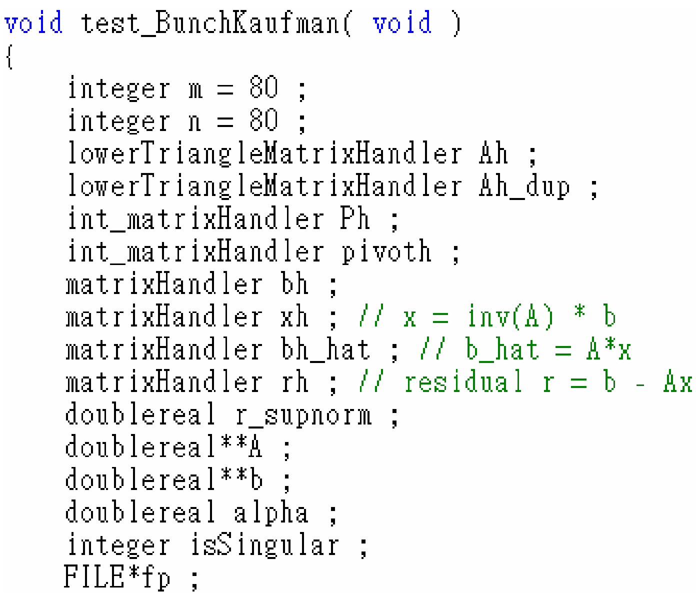
Memory usage (do you need extra storage?)

- When solving Ax = b, you can use only two variable rather than x,y,z,w in the original version .

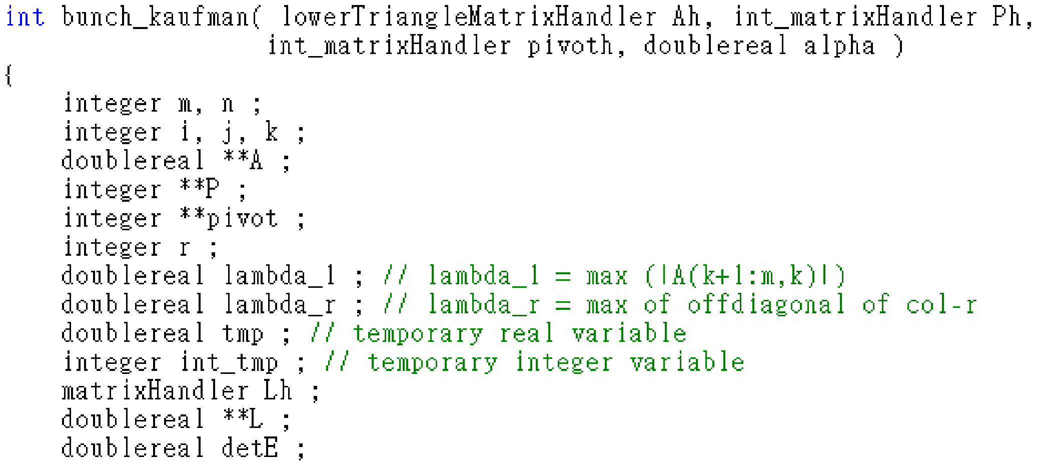
  Lz = Pb , Dy = z, L’z = y ,Px = w ➔ Ly = Pb , Dx = y, L’y = x, Px = y



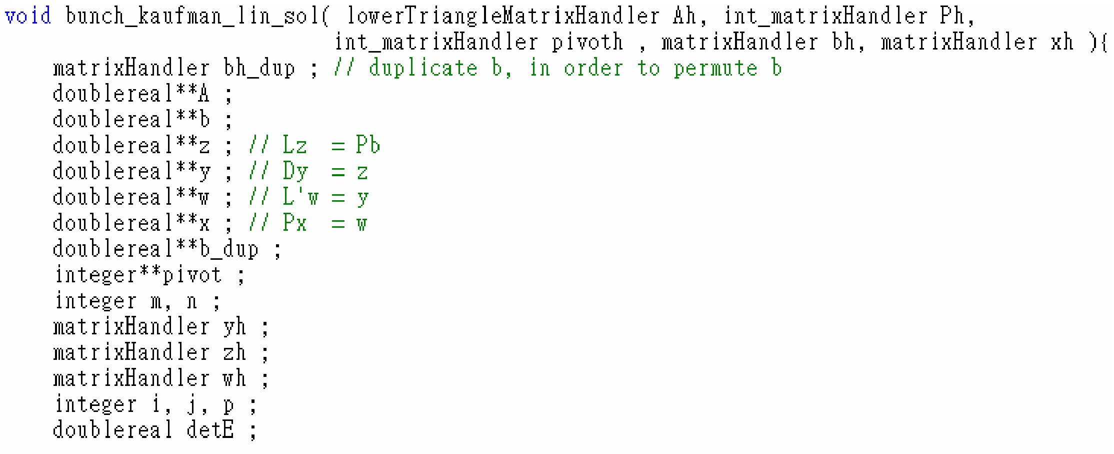
```
int main( int argc, char* argv[] )
{
int t ;
```



```
void test_BunchKaufman( void )
{
    integer m = 80 ;
    integer n = 80 ;
    lowerTriangleMatrixHandler Ah ;
    lowerTriangleMatrixHandler Ah_dup ;
    int_matrixHandler Ph ;
    int_matrixHandler pivoth ;
    matrixHandler bh ;
    matrixHandler xh ; // x = inv(A) * b
    matrixHandler bh_hat ; // b_hat = A*x
    matrixHandler rh ; // residual r = b - Ax
    doublereal r_supnorm ;
    doublereal**A ;
    doublereal**b ;
    doublereal alpha ;
    integer isSingular ;
    FILE*fp ;
```



```
int bunch_kaufman( lowerTriangleMatrixHandler Ah, int_matrixHandler Ph,
                   int_matrixHandler pivoth, doublereal alpha )
{
    integer m, n ;
    integer i, j, k ;
    doublereal **A ;
    integer **P ;
    integer **pivot ;
    integer r ;
    doublereal lambda_1 ; // lambda_1 = max (|A(k+1:m,k)|)
    doublereal lambda_r ; // lambda_r = max of offdiagonal of col-r
    doublereal tmp ; // temporary real variable
    integer int_tmp ; // temporary integer variable
    matrixHandler Lh ;
    doublereal **L ;
    doublereal detE ;
```



```
void bunch_kaufman_lin_sol( lowerTriangleMatrixHandler Ah, int_matrixHandler Ph,
                            int_matrixHandler pivoth , matrixHandler bh, matrixHandler xh ){
    matrixHandler bh_dup ; // duplicate b, in order to permute b
    doublereal**A ;
    doublereal**b ;
    doublereal**z ; // Lz  = Pb
    doublereal**y ; // Dy  = z
    doublereal**w ; // L'w = y
    doublereal**x ; // Px  = w
    doublereal**b_dup ;
    integer**pivot ;
    integer m, n ;
    matrixHandler yh ;
    matrixHandler zh ;
    matrixHandler wh ;
    integer i, j, p ;
    doublereal detE ;
```

Speedup strategy

- voi d *memcpy(s, ct, n)
- void *memset(s, c, n)