LDL’ = PAP’

Speaker : 高崇閔

Program structure[1]

main.cpp

```
void test_matrix(void) ;
void test_Bunch_Kaufman(void) ;
void Pivot(lowerTriangleMatrixHandler Ah, integer k, int_matrixHandler pivoth);


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

#ifdef HIGH_PRECISION_PACKAGE
    unsigned int old_cw;
    fpu_fix_start(&old_cw);
#endif

#ifdef DO_ARPREC
    mp::mp_init(ARPREC_NDIGITS);
#endif

// main_code
    test_Bunch_Kaufman() ;
//main_code

#ifdef DO_ARPREC
    mp::mp_finalize();
#endif

#ifdef HIGH_PRECISION_PACKAGE
    fpu_fix_end(&old_cw);
#endif
    return 0;
}
```

1_1 pivot

Use high precision

Main function

<Note>
do 1_1 pivot in case_1 to case_3

test_matrix.cpp

```
void test_matrix(void){

    integer m = 4;
    integer n = 4;
    lowerTriangleMatrixHandler Ah;
    doublereal **A;

    zeros(&Ah, m, n, COL_MAJOR, 1);
    A = Ah->A;

    A[1][1] = 6.; A[1][2] = 12.; A[1][3] = 3.  ; A[1][4] = -6.;
                  A[2][2] = -8.; A[2][3] = -13.; A[2][4] = 4. ;
                                 A[3][3] = -7. ; A[3][4] = 1. ;
                                                 A[4][4] = 6. ;
    disp(Ah, cout);

    dealloc( Ah );

}
```

<Note>
test constructor and display

Program structure[2]

test_Bunch_Kaufman.cpp

```
void test_Bunch_Kaufman( void )

    integer m=4;
    integer n=4;
    lowerTriangleMatrixHandler Ah;
    lowerTriangleMatrixHandler Ah_dup;
    int_matrixHandler Ph;
    int_matrixHandler pivoth;
    matrixHandler bh;
    matrixHandler xh, yh;
    matrixHandler bh_hat;  //b_hat = A*x
    matrixHandler rh;  //result r = b - Ax
    doublereal r_supnorm;

    int isSingular;
    doublereal **A;
    doublereal **b;
    doublereal alpha;

    alpha = (1.0 + sqrt(17.0))  / 8.0 ;

    zeros(&Ah, m, n, COL_MAJOR, 1) ;
    A = Ah->A;

    A[1][1] = 6; A[1][2] = 12; A[1][3] = 3    ;  A[1][4] = -6  ;
                 A[2][2] = -8; A[2][3] = -13  ;  A[2][4] = 4   ;
                               A[3][3] = -7   ;  A[3][4] = 1   ;
                                                 A[4][4] = 6   ;
    cout <<"configuration of matrix A"<<endl ;
    disp( Ah, cout );

    zeros( &Ah_dup, m, n, COL_MAJOR, 1) ;
    duplicate( Ah, Ah_dup);

    zeros ( &Ph,     m, 1, COL_MAJOR);
    zeros ( &pivoth, m, 1, COL_MAJOR);
    zeros ( &yh    , m, 1, COL_MAJOR);
    zeros ( &xh    , m, 1, COL_MAJOR);
```

test_Bunch_Kaufman.cpp

```
//step2
    cout.precision(4) ;
    isSingular = bunch_kaufman(Ah, Ph, pivoth, alpha);

    cout <<"the matrix A"<<endl;
    disp(Ah, cout);
    cout <<"the permutation vector P"<<endl;
    disp(Ph, stdout);
    cout <<"the pivot vector pivot"<<endl;
    disp(pivoth, stdout);
}
```

<Note>
test LDL' = PAP'

Program structure[3]

Bunch_Kaufman.cpp

```
int bunch_kaufman(lowerTriangleMatrixHandler Ah, int_matrixHandler Ph, int_matrixHandler pivoth, doublereal alpha)
{
//initial
    integer n, m, r;
    integer i, j;
    integer k = 1;
    doublereal lambda_1 = 0.0;
    doublereal tmp;

    doublereal **A;
    integer **pivot;
    integer **Per, **P;
    Per = Ph->A;
```

Initial & declare

Bunch_Kaufman.cpp

```
//verify
    assert(Ah) ; assert(Ph) ; assert(pivoth);
    assert(COL_MAJOR == Ah->sel) ;
    assert(COL_MAJOR == Ph->sel) ;
    assert(COL_MAJOR == pivoth->sel);

    m = Ah->m; n = Ah->n;
    assert(m == n);
    assert(0 != Ah->A); //A must be symmetric

    assert(1 == Ph->n) ; assert(m <= Ph->m);
    assert(1 == pivoth->n); assert(m <= pivoth->m);

    assert ((0.0 < alpha) && (1.0 >= alpha));
```

Assert

Bunch_Kaufman.cpp

```
//construct lower triangle matrix L
    matrixHandler Lh;
    doublereal **L;
    zeros (&Lh, m, 2, COL_MAJOR);

    L=Lh->A;

    A = Ah->A; P = Ph->A; pivot = pivoth->A;

    for (i=1; i<=n; i++){
        P[1][i] = i ;
    }//initial permutation

    for (j=1; j<=n; j++){
        pivot[1][j] = 0;
    }


    doublereal a_kk;
    integer int_tmp;
```

Construct matrix L , pivot & permutation

Program structure[4]

Bunch_Kaufman.cpp

```
//CASE1
    a_kk = A[k][k];

    if (a_kk >= alpha*lambda_1){
        cout <<"case_1"<<endl;
    Pivot(Ah, k, pivoth) ;
      }//CASE1
```

<Note>
If a_kk >= $\alpha * \lambda$, then do Case_1

Bunch_Kaufman.cpp

```
//CASE2
    if (a_kk*lambda_r >= alpha*lambda_1*lambda_1){
        cout <<"case_2"<<endl<<endl;
        Pivot(Ah, k, pivoth);
        k++;
      }//CASE2
```

<Note>
If a_kk* λ_r >= $\alpha * \lambda$ _1^2 , then do Case_2

Bunch_Kaufman.cpp

```
void Pivot(lowerTriangleMatrixHandler Ah, integer k, int_matrixHandler pivoth){
//1-1pivot

    integer  i, j;
    doublereal **L, **A;
    integer **pivot;
    doublereal tmp, a_kk;
    integer n, m;

    n = Ah->n; m = Ah->m;

    matrixHandler Lh;
    zeros (&Lh, m, 2, COL_MAJOR);

    L=Lh->A;

    A = Ah->A;
    pivot = pivoth->A;

a_kk = A[k][k];
    //L(k+1:n, k) = A(k+1:n, k) / A(k,k)
        for (i=k+1; i<=n; i++){
            L[1][i] = A[k][i] / a_kk ;
        }
    //A(k+1:n,k+1:n) = A(k+1:n,k+1:n) - L(k+1:n,k) * A(k,k+1:n)
        for (j=k+1; j<=n; j++){
            tmp = A[k][j]; //A[k][j] = A(j,k)
            for (i=j; i<=n; i++){
                A[j][i] -= L[1][i] * tmp;
            }//for row
        }//for col
    //store L into A
        for (i=k+1; i<=n; i++){
            A[k][i] = L[1][i];
        }
    //update k = k+1, pivot(k) = 1
        pivot[1][k] = 1;
        k++;
}
```

1_1 Pivot

Program structure[5]

Bunch_Kaufman.cp

```
//CASE3
    a_rr = A[r][r];

    if (a_rr < 0){
        a_rr *= -1;
    }

    if(a_rr >= alpha*lambda_r){
     //update the permutation matrix
        int_tmp   = Per[1][k];
        Per[1][k] = Per[1][r];
        Per[1][r] = int_tmp  ;

     //do interchange of matrix A
        //(1) A[k][k] <--> A[r][r]
        tmp     = A[k][k];
        A[k][k] = A[r][r];
        A[r][r] = tmp;
        //(2) A (r+1:n,k)<-->A(r+1:n,r)
        for (i=r+1; i<=n; i++){
            tmp     = A[k][i];
            A[k][i] = A[r][i];
            A[r][i] = tmp;
        }
    //(3) A(k+1:r-1,k) <--> A(r,k+1:r-1)
        for (i=k+1; i<=r-1; i++){
            tmp     = A[k][i];
            A[k][i] = A[i][r];
            A[i][r] =tmp;
        }
     //update lower triangle matrix L
        if ( k>1 ){
            for (j=1; j<k; j++){
                tmp     = A[j][k] ;
                A[j][k] = A[j][r];
                A[j][r] = tmp;
            }
        }//if (k>1)
        Pivot(Ah, k, pivoth) ;
        k++;
    }//CASE3
```

If a_rr >= $\alpha * \lambda$_r , then do case_3

Update matrixes

do Pivot

Bunch_Kaufman.cp

```
//CASE4
    if (a_rr < alpha*lambda_r){
     //update permutation matrix per
        int_tmp     = Per[1][k+1] ;
        Per[1][k+1] = Per [1][r];
        Per[1][r]   = int_tmp;
        //(1) A(r,r) <--> A(k+1,k+1)
        tmp         = A[r][r];
        A[r][r]     = A[k+1][k+1];
        A[k+1][k+1] = tmp;
        //(2) A(r+1:n,k+1) <--> A(r+1:n,r)
        for (i=r+1; i<=n; i++){
            tmp       = A[k+1][i];
            A[k+1][i] = A[r][i];
            A[r][i]   = tmp;
        }
        //(3) A(k+1,k) <--> A(r,k)
        tmp       = A[k][k+1] ;
        A[k][k+1] = A[k][r];
        A[k][r]   = tmp;
        //(4) A(k+2:r-1,k+1) <--> A(r,k+2:r-1)
        for (i=k+2; i<=r-1; i++){
            tmp       = A[k+1][i];
            A[k+1][i] = A[i][r];
            A[i][r]   = tmp;
        }

     //update lower triangle matrix L  L(k+1,1:k-1) <--> L(r,1:k-1)
        if (k > 1){
            for (j=1; j<k; j++){
                tmp       = L[j][k+1];
                L[j][k+1] = L[j][r];
                L[j][r]   = tmp;
            }
        }//if (k>1)
```

If a_rr < $\alpha * \lambda$_r, do Case_

Program structure[6]

Bunch_Kaufman.cpp

```
    //compute detE
       doublereal detE;
       doublereal invE_11, invE_12, invE_21, invE_22;
       detE = A[k][k]*A[k+1][k+1] - A[k][k+1]*A[k][k+1];
       invE_11 = A[k+1][k+1] / detE;
       invE_22 = A[k][k] / detE;
       invE_21 = -A[k][k+1] / detE;
       invE_12 = invE_21 ;

    //L(k+2:n,k:k+1) = A(k+2:n,k:k+1) * invE
       for (i=k+2; i<=n; i++){
           L[1][i] = A[k][i] * invE_11 + A[k+1][i] * invE_21 ;
           L[2][i] = A[k][i] * invE_12 + A[k+1][i] * invE_22 ;
       }

       cout <<"L = " <<endl;
       disp(Lh,stdout) ;

//A(k+2:n,k+2:n) = A(k+2:n,k+2:n) - A(k+2:k+1,k:n)*L(k+2:n,k:k+1);
// warning : only update lower triangle part
       for (i=k+2; i<=n; i++){
           for (j=k+2; j<=i; j++){
               A[j][i] -= L[1][i] * A[k][j] +L[2][i] * A[k+1][j];
           }//for row
       }//for column
    //update pivot
       for (i=k+2; i<=n; i++){
           A[k][i]   = L[1][i];
           A[k+1][i] = L[2][i];
       }

       pivot[1][k] += 2;
       k += 2;
   }//CASE4
```

Compute determine

Compute matrix L

Update matrix A

Matlab[1]

- MatLab is more convenient in matrix computation.
- The code in matlab will be shorter.
- The code in matlab is easier to construct.
- C program is more difficult to construct, but may more fast . (?)

Matlab[2]

n = 100 time = 0.097959

n = 200 time = 0.197049

n = 400 time = 2.403642

n = 800 time = 21.268364

LDL' = PAP' composition

n = 100 time = 0.001295

n = 200 time = 0.002731

n = 400 time = 0.012545

n = 800 time = 0.028716

Linear solver in forward

n = 100 time = 0.087844

n = 200 time = 0.305122

n = 400 time = 1.568356

n = 800 time = 8.895769

Linear solver in backward

Conclusion :

(i) we spend most time in decomposition

(ii) forward is more faster than backward

Memory usage

- A lower triangle matrix to store the initial matrix A   [m*(m+1) / 2] * doublereal
- A column vector to store matrix pivot [(m*1)] * integer
- A column vector to store matrix Per(mutation) [(m*1)] *integer
- A temporary matrix to store matrix L [(m*2)] * doublereal
- Totally (1) [m(m+5) / 2] * doublereal   (2) (2m) * integer
- (this statistics is roughly and ignore other data which are not matrix date type)

Speedup strategy

- **Using mutilthread**
- **We will load lower triangle matrix A into graphical card at first.**
- **In case_4, we can use different thread to compute determine(because they're independent)**
- **When we finish decomposition , we must back to do linear solver, then we can use different thread to make computation fast. (even in backward or forward, we always do many computation into element multiple)**
- **However , we still not solve the major problem such that we have to spend our most time in composition .**