An introduction to multi-precision linear algebra library MPLAPACK

Ge Baolai, Western University
SHARCNET | Compute Ontario
Digital Research Alliance of Canada | Alliance de recherche numérique du Canada

Outline

- What is and why MPLAPACK
- Availability
- Other numerical, multi-precision libraries
- Examples
- Parallel processing
- Limitations
- Reference

What is and why MPLAPACK

What is and why MPLAPACK

- MPLAPACK is a rewrite of the widely used linear algebra packages **BLAS** and **LAPACK**, with **multiple precision** support, by Japanese scientist 中田真秀 (NAKATA Maho). It keeps the same interfaces of BLAS and LAPACK routines.
- LAPACK is for solving system of linear equations, the least squares problems, finding eigenvalues/eigenvectors and computing singular value decomposition (SVD), etc using the building blocks provided in BLAS for dense matrices.
- LAPACK and BLAS were written in Fortran (FORTRAN 77), containing subroutines separately supported for IEEE **single** (~7 digit accuracy) and **double precision** (~15 digit accuracy) real and complex matrices. Same routine, same interface e.g. *x*GEMM, *x*GETRI, with different letter *x* for different types.
- There are needs from physics, chemistry, engineering and mathematics where higher precision, e.g. accuracy of hundreds of digits, is wanted.
- LAPACK does not support arbitrary precision. If one wants to solve ill-conditioned problem A*x = b or compute the eigenvalues, or solving positive semi-definite programming where higher precision is desired, there is a gap between the needs for LAPACK and the needs for higher precision.
- MPLAPACK bridges the gap.

What is and why MPLAPACK

Example 1: Solving an ill-conditioned problem A*x = b, where

$$A = \begin{bmatrix} 2 & 3.9998 \\ 1 & 2 \end{bmatrix}, \quad \text{let } x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b = Ax = \begin{bmatrix} 5.9998 \\ 3 \end{bmatrix}.$$

We construct the right hand side (RHS) b by multiplying matrix A by a unit vector.

We now make changes first in b and then in A and examine the solutions in MATLAB

1) Change b by less than 0.001, keep A the same

$$b' = \begin{bmatrix} 5.9998 \\ 3.0003 \end{bmatrix}, \quad \Longrightarrow x' = \begin{bmatrix} -5.7589 \\ 4.3796 \end{bmatrix}.$$

2) Change the elements in A by less than 0.001, keep b the same

$$A' = \begin{bmatrix} 2.0000 & 3.9999 \\ 1.0005 & 2.0002 \end{bmatrix}, \quad \Longrightarrow x' = \begin{bmatrix} -0.8036 \\ 1.9018 \end{bmatrix}.$$

Small changes in the input lead to large changes in the output.

What is and why MPLAPACK

Example 1: Solving an ill-conditioned problem A*x = b (cont'd)

$$A = \begin{bmatrix} 2 & 3.9998 \\ 1 & 2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b = Ax = \begin{bmatrix} 5.9998 \\ 3 \end{bmatrix}.$$

Small changes in b and A lead to large changes in the solution.

1) Change b by less than 0.001, keep A the same

$$b' = \begin{bmatrix} 5.9998 \\ 3.0003 \end{bmatrix}, \quad \Longrightarrow x' = \begin{bmatrix} -5.7589 \\ 4.3796 \end{bmatrix}.$$

2) Change the elements in A by less than 0.001, keep b the same

$$A' = \begin{bmatrix} 2.0000 & 3.9999 \\ 1.0005 & 2.0002 \end{bmatrix}, \quad \Longrightarrow x' = \begin{bmatrix} -0.8036 \\ 1.9018 \end{bmatrix}.$$

Things like this may confuse us in real life. This matrix is ill-conditioned.

What is and why MPLAPACK

Example 1: Solving an ill-conditioned problem A*x = b (cont'd)

$$A = \begin{bmatrix} 2 & 3.9998 \\ 1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 5.9998 \\ 3 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

This raises the questions, especially when we face real problems:

- How do we know the solution we have obtained is correct?
- How reliable is the computed solution, if we have a small change in the input and we see a large change in the output?
- etc.

What is and why MPLAPACK

Example 2: Solving system of linear equations H*x = b with Hilbert matrix, with

$$h_{ij} = 1/(i+j-1).$$

For example, for n=6*

$$H_6 = \begin{bmatrix} 1/1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{bmatrix}.$$

What is and why MPLAPACK

Example 2: Solving system of linear equations H*x = b with Hilbert matrix (cont'd). Hilbert matrix is ill-conditioned, solving the linear system with an ill-conditioned matrix becomes problematic when n is large.

$$H_6 = \begin{bmatrix} 1/1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{bmatrix}.$$

We construct the RHS b by multiplying H by a unit vector x, i.e. b = H*x.

We then solve H*x = b and see if we get the solution the same, or "close" to the the unit vector.

Note: One may see in the literature the advice that Hilbert matrix is not a good test matrix because it is ill-conditioned. We use it intentionally for the purpose of testing the features of MPLAPACK.

What is and why MPLAPACK

Example 2: Solving system of linear equations H*x = b with Hilbert matrix (cont'd). Hilbert matrix is ill-conditioned, solving the linear system via LU decomposition may become problematic when n becomes large.

```
Solving H * x = b of
size 10

Solution:
          [ 1 ]
              1
      0.9999999
       1.000003
      0.9999758
       1.000116
       0.999682
       1.000522
      0.9994947
       1.000266
      0.9999415
```

```
Solving H * x = b of
size 11

Solution:
          [ 1 ]
              1
      0.9999982
       1.000047
      0.9994746
       1.003154
      0.9888376
        1.02444
      0.9665197
       1.027928
      0.9870307
        1.00257
```

```
Solving H * x = b of
size 13

Solution:
          [ 1 ]
      0.9999999
       1.000013
      0.9994794
         1.0088
      0.9199938
       1.438536
     -0.5437174
        4.60748
      -4.656326
       6.882018
      -2.890713
       2.481753
      0.7526833
```

```
Solving H * x = b of
size 50

Solution:
          [ 1 ]
       1.000001
       0.999892
       1.004564
      0.9173155
       1.790654
      -3.380056
       15.46936
      -26.46469
       25.66145
       1.871251
       -15.6143
       15.50733
      -30.80703
       22.92981
       52.54445
      -54.27587
        ,,, ,,,
```

When n>=12, one will get warning: the matrix is singular to machine precision.

Availability

Availability

From https://github.com/nakatamaho/mplapack

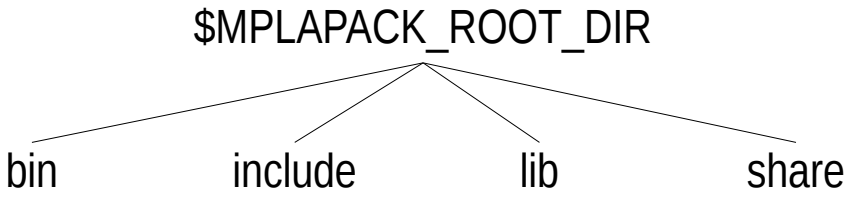
- Linux
- Mac OS
- Windows (mingw-64)

Two methods of having it on your computer:

- Using Docker, just follow the instruction; OR
- Building from tar-ball. Straightforward, requires **GMP** and **MPFR**, no hiccup, but may take more than an hour, keeping the fan humming.

Availability

The built package has a well organized structure



- It comes with a user manual and examples in share.
- The examples are easy to follow if one has used LAPACK or BLAS before.

Availability

Supported floating-point formats

- IEEE double precision (64-bit) ~15 digits.
- Extended (80-bit) double precision ~19 digits.
- Binary128 (128-bit) quadruple precision ~33 digits.
- Double-double ~32 digits and quad-double ~64 digits.
- GMP, but to be dropped in the future.
- MPFR for arbitrary precision.
- The precision can be set by environmental variables MPLAPACK_GMP_PRECISION, MPLAPACK_MPFR_PRECISION in binary digits (bits).

Other related tools and packages

Other related tools and packages

Related tools and packages

- Maple
- Mathematica
- GMP – GNU multiple precision arithmetic library.
- MPFR – GNU multiple precision floating-point reliable library
- MPFUN2020 https://www.davidhbailey.com/dhbsoftware/
- Arb – A C library for rigorous real and complex arithmetic with arbitrary precision
- Julia – with support through intrinsic data types by design.
- mpmath – A Python library for arbitrary-precision floating-point arithmetic, http://mpmath.org/.

Examples

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, ***revisited***. The RHS b is built by multiplying H by a unit vector x. The solution, however, deviates wildly when n gets bigger.

```
Solving H * x = b of size 10

Solution:
          [ 1 ]
              1
      0.9999999
       1.000003
      0.9999758
       1.000116
       0.999682
       1.000522
      0.9994947
       1.000266
      0.9999415
||dx||_2/||x||_2 = 0.00026777989621392767
```

```
Solving H * x = b of size 12

Solution:
          [ 1 ]
              1
      0.9999867
       1.000421
      0.9942459
       1.042286
      0.8138542
       1.519414
     0.05870534
       2.104559
      0.1904543
       1.336781
      0.9392925
||dx||_2/||x||_2 = 0.51517152763376584
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, ***revisited***. The RHS b is built by multiplying H by a unit vector x. The solution, however, deviates wildly when n gets bigger.

```
Solving H * x = b of size 13

Solution:
          [ 1 ]
      0.9999999
       1.000013
      0.9994794
         1.0088
      0.9199938
       1.438536
     -0.5437174
        4.60748
      -4.656326
       6.882018
      -2.890713
       2.481753
      0.7526833
||dx||_2/||x||_2 = 2.7677028033584072
```

```
Solving H * x = b of size 50

Solution:
          [ 1 ]
       1.000001
       0.999892
       1.004564
      0.9173155
       1.790654
      -3.380056
       15.46936
      -26.46469
       25.66145
       1.871251
       -15.6143
       15.50733
      -30.80703
       22.92981
       52.54445
      -54.27587
        ,,, ,,,
||dx||_2/||x||_2 = 27.025124940220273
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix (cont'd)

- The inverse of Hilbert matrix has a closed form containing integer values. But when n >= 12, some of its values fall beyond what can be represented exactly in IEEE double precision.
- This means one can't get accurate answer by the multiplication of the inverse matrix and the right RHS.

The following is output of the 9th to 13th columns of the inverse of an Hilbert matrix of order 13 in julia*

1891439550//1	-1849407560//1	1160082924//1	-421848336//1	67603900//1
-285985659960//1	282454972800//1	-178652770296//1	65418941952//1	-10546208400//1
10724462248500//1	-10680328659000//1	6802547792040//1	-2505779115840//1	406029023400//1
-174769014420000//1	175266931840000//1	-112296027043200//1	41577371996160//1	-6767150390000//1
1542672646515000//1	-1556276461740000//1	1002242041360560//1	-372734643481200//1	60904353510000//1
-8251094840788800//1	8366542258314240//1	-5412107023347024//1	2020660279013376//1	-331319683094400//1
28450997395460640//1	-28977867717598800//1	18818568211899456//1	-7050482856248832//1	1159618890830400//1
-65321167489578000//1	66791723910912000//1	-43525940081944320//1	16357726068203520//1	-2697888848054400//1
100863567447142500//1	-103492384705710000//1	67651337791837800//1	-25495049614114080//1	4215451325085000//1
-103492384705710000//1	106518477825760000//1	-69822862214785680//1	26379357625420800//1	-4371579151940000//1
67651337791837800//1	-69822862214785680//1	45883595169716304//1	-17374404181470336//1	2885242240280400//1
-25495049614114080//1	26379357625420800//1	-17374404181470336//1	6592659294154752//1	-1096868950850400//1
4215451325085000//1	-4371579151940000//1	2885242240280400//1	-1096868950850400//1	182811491808400//1

* This matrix is generated in julia with SpecialMatrices package. The values are represented in Rational format.

Examples

Example 3: Solving A*x = b by calling LAPACK

```
Int main()
{
    … …
    // Create Hilbert matrix of order n
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
            H[i + j * n] = one / ((i + 1) + (j + 1) - 1);
        x[i] = one; // True solution vector
    }

    // Creat the RHS b = H*x
    dgemm("N", "N", n, ione, n, one, H, n, x, n, zero, b, n);

    // Solve linear equation A * x = b, store solution in b
    dgesv(n, ione, H, n, ipiv, b, n, info);

    // Display the solution stored in b
    cout << "\nSolution:\n";
    printmat_pretty(b.data(),n,1,n,1,1);
}
```

Example 3: Solving A*x = b by calling MPLAPACK

```
#include <mpblas_mpfr.h>
#include <mplapack_mpfr.h>

int main()
{
    … …
    // Create Hilbert matrix of order n
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            mtmp = (i + 1) + (j + 1) - 1;
            H[i + j * n] = one / mtmp;
        }
        x[i] = one;
    }

    // Creat the RHS b = H*x
    Rgemm("N", "N", n, 1, n, one, H, n, x, n, zero, b, n);

    // Solve linear equation A * x = b, store solution in b
    Rgesv(n, (mplapackint)1, H, n, ipiv, b, n, info);

    // Display the solution stored in b
    printf("\nsol:\n"); printvec1(b.data(),n);
}
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=13 with precision MPLAPACK_MPFR_PRECISION=52 (~15 digits)

```
Solving H * x = b with size 13

sol:
+1.0000002291802703169310007069725543260574340820312500000000000000e+00
+9.9996529398826661250154756999108940362930297851562500000000000000e-01
+1.0013031652056785247850712039507925510406494140625000000000000000e+00
+9.7874093756185809311887169315014034509658813476562500000000000000e-01
+1.1878101491112316168141660455148667097091674804687500000000000000e+00
-5.1949976671743208073150910308868333231657743453979492187500000000e-03
+4.4679490417013827396885972120799124240875244140625000000000000000e+00
-6.9656401666254499360775298555381596088409423828125000000000000000e+00
+1.3304303953829702322764205746352672576904296875000000000000000000e+01
-1.1628704836171724679161343374289572238922119140625000000000000000e+01
+9.2571817643080045456827065208926796913146972656250000000000000000e+00
-2.1124192060706805662562146608252078294754028320312500000000000000e+00
+1.5147048314882933794933705939911305904388427734375000000000000000e+00
||dx||_2/||x||_2 = 5.9841935630246308
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=13 with precision set to 64 (~19 digists)

```
Solving H * x = b with size 13

sol:
+1.0000000000202592024102846157695978490664856508374214172363281250e+00
+9.9999999713765458229279906032793689973914297297596931457519531250e-01
+1.0000001011472766303082859717932251442107371985912322998046875000e+00
+9.9999843658530147822273773106083183392911450937390327453613281250e-01
+1.0000131584531067633573458985907222995592746883630752563476562500e+00
+9.9993259840260664269651416646844666047400096431374549865722656250e-01
+1.0002234095538461025945431437378374539548531174659729003906250000e+00
+9.9950535075265004806427393546641724242363125085830688476562500000e-01
+1.0007386362758243786167181021973249244183534756302833557128906250e+00
+9.9926527577780360181609886627285277427290566265583038330078125000e-01
+1.0004666128738118594963465990410611539118690416216850280761718750e+00
+9.9982882411360100036266246315008743295038584619760513305664062500e-01
+1.0000275989100892654395935643663051450857892632484436035156250000e+00
||dx||_2/||x||_2 = 3.543685575947061e-4
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=13 with precision set to 100 (~30 digists)

```
Solving H * x = b with size 13

sol:
+1.0000000000000000000019833737841857494393889254788727991192509581e+00
+9.9999999999999999969242948730919752968953432873454178776142717289e-01
+1.0000000000000000011773889486882940969002326333177844752469684563 9e+00
+9.9999999999999980482308732723047657239344617743593409759294762806e-01
+1.0000000000000001747800087572751223465981508226717093976653426279 4e+00
+9.9999999999999905360423615239651901080140474753238049009920404564 0e-01
+1.0000000000000329819971730344280395815656306243408600267644562032e+00
+9.9999999999992356977521657943478950632608591484138707528809414393e-01
+1.0000000000001189860591369017227118683853904575954407447490462912e+00
+9.9999999999987702381277727144025979229541132918715740627874782875e-01
+1.0000000000000809103848263016443933566861115827148172915256330029e+00
+9.9999999999996932995554366055196135877841182573019827780095789005e-01
+1.0000000000000050978832457269717364766462322147851017362400402599e+00
||dx||_2/||x||_2 = 5.8055680326398419e-14
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=50 with precision set to 64 (~19 digits)

```
Solving H * x = b with size 50

sol:
+9.999999832392049198726176328921866343080182559788227081298828125 0e-01
+1.000004095864469884423313128518628900565090589225292205810546875 0e+00
+9.997533585389530295158694306323354794585611671209335327148437500 0e-01
+1.006416890657216332643185408013408732585958205163478851318359375 0e+00
+9.101770936179730998199545466409432492582709528505802154541015625 0e-01
+1.754645726359263271454738686294660965359071269631385803222656250 0e+00
-3.025485577826932673088655700865956532652489840984344482421875000 0e+00
+1.492705377269517114970087678926802254864014685153961181640625000 0e+01
-2.990606924945096994732096629121542719076387584209442138671875000 0e+01
+4.276739684119682013280216992257010133471339941024780273437500000 0e+01
-3.069758402895030269599829786386635532835498452186584472656250000 0e+01
+2.211551147035071555706220447490295555326156318187713623046875000 0e+01
-4.139372050994576518467837900061567779630422592163085937500000000 0e+01
,,, ,,,
-6.263147497410995994532312280966834805440157651901245117187500000 0e+01
+5.791292167489029229024422029681318235816434025764465332031250000 0e+01
-1.378015067767715028882596906001367642602417618036270141601562500 0e+01
||dx||_2/||x||_2 = 38.78403993623718
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=50 with precision set to 128 (~38 digits)

```
Solving H * x = b with size 50

sol:
+1.0000000000000000380794158481129104890966356859773133508328736 88e+00
+9.9999999999999666750701030718839213402964044475822265094207334280e-01
+1.0000000000007255172703651693486401563253154143934208671130269567e+00
+9.9999999930388372725516212688099704162868239654423880846009464 50e-01
+1.0000000037115146126182340265678742155067632732281248938656716583e+00
+9.9999987538884445599075067331553907452807942009956243269657735589e-01
+1.0000028459629431368989744684343643758006993935196796059628634314e+00
+9.9995345392071606549291704153566672717231530818676983009051771465e-01
+1.0005648957967554605483907518990365568580425813599501768911185623e+00
+9.9478467201851628393711200786220385336335657458783827004246119611e-01
+1.0372503209164496010899653761339063290379157475001644945029049109e+00
+7.9209278251439380606377111389207802732849034139528473607318024525e-01
+1.9097837552464119470205639731269624580606586446574511656723177134e+00
,,, ,,,
+2.8721507790161390178564992472450103080885080664547374818894447845e+01
-6.4398157655023067179298982089858407629351504680616088425163990860e+00
+1.8268230699763150792418463987246657311249682987681419189201216862e+00
||dx||_2/||x||_2 = 27.676389927831689
```

Examples

Example 3: Solving system of linear equations H*x = b with Hilbert matrix, revisited (cont'd).

The solution using MPLAPACK for size n=50 with precision set to 256 (~77 digits)

```
Solving H * x = b with size 50

sol:
+9.99999999999999999999999999999999999998879492058103058929515211 3e-01
+1.00000000000000000000000000000000000027402472043574541594297153 20e+00
+9.99999999999999999999999999999999983260752379616163005546655839877e-01
+1.00000000000000000000000000000000453712747540174162611728149324056e+00
+9.99999999999999999999999999999309955434377779080325957231533638139e-01
+1.00000000000000000000000000006694614591449640070749677028466416456e+00
+9.99999999999999999999999995508169980385583839806043078800746094952e-01
+1.00000000000000000000000022033001762761836865691699629112669056610e+00
+9.99999999999999999999991773505619909749022226501850402206025537868e-01
+1.00000000000000000000241072212190904901968414255065708843491754 44e+00
+9.99999999999999999943209662520757994550399721362754230287390915 24e-01
+1.00000000000000000109630392523476677954585347135320388017854596 38e+00
+9.99999999999999982383508237462007591710183833393621877322555543 71e-01
,,, ,,,
+1.00000000077720387840342679860341486536920890311185043384768693 79e+00
+9.99999999346158125491937473337861078845521058043958457082881221 8e-01
+1.00000000002666441470378135169767244631086894663033342976716252 9e+00
||dx||_2/||x||_2 = 1.2307570889575885e-5
```

Examples

Example 4: Consider a different problem, computing the eigenvalues of a Wilkinson matrix with

$$\text{diag } d_i = \begin{cases} \dfrac{n-1}{2} - i + 1 & \text{if } n \text{ is odd} \\ \dfrac{n}{2} - i + \dfrac{1}{2} & \text{if } n \text{ is even} \end{cases}, \quad 1 \text{ on upper and lower diagonals.}$$

For example, for n=7

$$W_7 = \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 \end{bmatrix}.$$

Examples

Example 4: Compute the eigenvalues of Wilkinson matrices using LAPACK routine *x*STEV

Using **DSTEV** (15 digits)

```
-0.964132172690471
 0.212662842513101
 1.078164406734622
 1.654730092254141
 2.400771751580549
 2.613723558709661
 3.486274955522134
 3.517639311060091
 4.498968620932756
 4.501192469653734
 5.499953844648218
 5.500050238625574
 6.499998623277986
 6.500001455954134
 7.499999970478817
 7.500000030730065
 8.499999999521540
 8.500000000493207
 9.499999999993923
 9.500000000006223
10.499999999999936
10.500000000000064
11.499999999999996
11.499999999999998
12.499999999999991
12.500000000000004
13.499999999999998
13.500000000000009
14.500000000000005
14.500000000000007
15.500000000000622
15.500000000000627
16.500000000054484
16.500000000054484
17.500000003808122
17.500000003808132
18.500000205070435
18.500000205070439
19.500008158672937
19.500008158672941
20.500225680185167
20.500225680185174
21.503952002665361
21.503952002665361
22.538941119306440
22.538941119306443
23.710678647333040
23.710678647333047
25.246194182903356
25.246194182903363
```

Using MPLAPACK **Rstev** with ~24 digits

```
-9.641321726904788469204069728541104376297321554289965206407941877 8e-01
+2.126628425131107295702005952767162718242827113002135774877388030 3e-01
+1.078164406734610586751576715033333815109417574973349474021233618 3e+00
+1.654730092254134485579713510457807584819178714496956672519445419 3e+00
+2.400771751580549653471690953749546914609835912557400661171413958 1e+00
+2.613723558709657970251663198307746829168740632098888454493135213 9e+00
+3.486274955522127377316129938959147083893386209751952264923602342 6e+00
+3.517639311060085206913356779567678090867022255849860812304541468 6e+00
+4.498968620932769352208490799602784307800740215554924361640587449 1e+00
+4.501192469653747967059969564229768984597868097807804588228464126 6e+00
+5.499953844648219428838292654318823347858735672849377226875722408 3e+00
+5.500050238625574493614696119251695257540424321973659971263259649 3e+00
+6.499998623277988915591971050985216828629886798296411143383011221 9e+00
+6.500001455954136799897164317768892072627040512600160582223907113 1e+00
+7.499999970478811371657469582907069470187333415367447742028161883 4e+00
+7.500000030730059526457076085147310153672384203105139022227376699 4e+00
+8.499999999521540927978790156492702907720104832378638093359768390 7e+00
+8.500000000493206277760363584839809533016952158845924714114516973 5e+00
+9.499999999993923902705066262354199543849198050793347647413611412 0e+00
+9.500000000006222657447748142757145729808687661943622515536844730 4e+00
+1.049999999999993785719572442581814511117221933034215908264741301 5e+01
+1.050000000000006335023028659717974101980852985604997229529544711 1e+01
+2.150395200266536132836609750569037066703081961804855382069945335 4e+01
+2.150395200266536132836615044524957406080201138820484629832208156 6e+01
+1.350000000000000044214471337074236125669335351062727568205446005 e+01
+1.450000000000000574246363785412239051988980698126852075802162289 6e+01
+1.450000000000000574246386284724900494341737200443276378791779279 7e+01
+1.550000000000061978957208048449068708146691975002795516047626733 8e+01
+1.550000000000061978957212018916008962679531357764517451869323849 7e+01
+1.650000000005448810799040600120937260369968768713988538365811109 5e+01
+1.650000000005448810799053835010738108812766711253061657771468162 5e+01
+1.750000000380812688353690137905418686137259953738976037129759788 5e+01
+1.750000000380812688353692784883378855825819542246790661010891199 1e+01
+1.850000020507043780031862323734188208179674006714776623994112014 8e+01
+1.850000020507043780031862323734188208179674006714776623994112014 8e+01
+1.950000815867294501046400855354058696822239227230966207571327686 3e+01
+1.950000815867294501046406149309979036199358404246595455333590507 5e+01
+2.050022568018517034412522975027185802299189987252248101867735385 9e+01
+2.050022568018517034412528268983106141676309164267877349629998207 1e+01
+2.150395200266536132836609750569037066703081961804855382069945335 4e+01
+2.150395200266536132836615044524957406080201138820484629832208156 6e+01
+2.253894111930644088976742376876564837029359722464505466632544994 4e+01
+2.253894111930644088976745023854525006717919310972320090513676405 0e+01
+2.371067864733304648832766481722110138430359427275107009336352348 3e+01
+2.371067864733304648832766481722110138430359427275107009336352348 3e+01
+2.246194182903357570586825221289786584655256973519499297253787518 e+01
+2.524619418290335757058690463062859167531204462875393801368772983 6e+01
```

21.503952002665361
21.503952002665361

+2.150395200266536132836609750569037066703081961804855382069945335 4e+01
+2.150395200266536132836615044524957406080201138820484629832208156 6e+01

Compute Ontario colloquium: Introduction to MPLAPACK, Ge B, Feb 8, 2023

Examples

Example 5: Compute SVD of matrices. Take again the Hilbert matrix for example

$$h_{ij} = 1/(i+j-1).$$

For example, for n=6*

$$H_6 = \begin{bmatrix} 1/1 & 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 & 1/10 \\ 1/6 & 1/7 & 1/8 & 1/9 & 1/10 & 1/11 \end{bmatrix}.$$

Examples

Example 5: Compute SVD of matrices (cont'd). For an m-by-n real matrix A, the SVD decomposition of A is

$$A = USV^T$$

where U is an m-by-m matrix, V an n-by-n matrix and S is an m-by-n matrix with only nonzero "singular values" on the diagonal

$$S = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \text{ for } m \geq n, \quad S = \begin{bmatrix} \sigma_1 & & & \cdots & 0 \\ & \ddots & & & \\ & & \sigma_m & \cdots & 0 \end{bmatrix} \text{ for } m \leq n.$$

Among many things, one of the properties of singular values is it can tell us the rank and the condition number of a matrix, e.g. is it close to "singular"? that we've been looking at so far. We can compute the condition number in 2-norm by $\kappa_2 = \sigma_{\max}/\sigma_{\min}$ (see MATLAB).

Examples

Example 5: Compute SVD of matrices (cont'd) using LAPACK routine *x*GESVD, *x* being *S* for single and *D* for double. In MPLAPACK, the uniform interface is Rgesvd. The following is the code snippet

```
// Create/input the m x n matrix A
... ...
// Compute SVD of the matrix. Check the man page of DGESVD for details
Rgesvd("A", "A", m, n, A, m, S, U, m, VT, n, work, lwork, info);
```

except for the leading letter R, the interface is the same for both LAPACK and MPLAPACK

where

- First two letters indicating the options for computing U and VT respectively.
- A – m-by-n array, S – m-by-n matrix, U – m-by-m matrix, VT – n-by-n matrix.
- work – work space of length lwork.

Examples

Example 5: Compute SVD of matrices (cont'd). Results of SVD of Hilbert matrix of size 13, with 15 and 154 digits (roughly)

```
S:
        1.81383
      0.3968331
     0.04902942
    0.004348755
   0.0002951777
    1.56237e-05
   6.466419e-07
   2.076321e-08
   5.076552e-10
   9.141281e-12
   1.143512e-13
   8.904175e-16
   3.165503e-19
Cond(H(13): 5.7299894005078e+18
```

```
S:
+1.813830118796976885255477911117489376374210599057279124819451629 1e+00
+3.968330760176222057975561427960819783185430611782321816862252812 3e-01
+4.902941941980765767827361998487271515685307224639109059344121301 1e-02
+4.348755074641771156177581095282427432559134402820494218751870174 2e-03
+2.951777135329739951950983458023592163096224228498940993046551907 0e-04
+1.562370360405994755029729699158201275216357306527523176078985085 0e-05
+6.466418562935460488152284389492450724071822232117530373437647426 9e-07
+2.076321420615754876489400587291297692354694953864435311752933511 9e-08
+5.076551871014000227315722964240005183275518195695656629493934312 4e-10
+9.141282506528421554605980975406631942108362734918959680127159371 5e-12
+1.143496099807889110208147831635859967027554052692249969579603830 5e-13
+8.878210698585523744391679992205380050678019340646119282788949384 6e-16
+3.222901014860856593259237356034500456189101174864980513277669248 9e-18
Cond(H(13): +5.627942373760076502808898912295100827994507734827710854721485220 3e+17
```

We see the condition number is huge.

Parallel processing

Parallel processing

- Currently a few key routines in MPBLAS are optimized for acceleration via OpenMP
  - Raxpy – compute y = a*x + y.
  - Rcopy – copy A to B.
  - Rdot – dot-product of two vectors.
  - Rgemm – matrix-matrix multiplication C =alpha*A*B + beta*C
- Need to compile and link against the optimized MPBLAS library, e.g.

  -fopenmp -l**mpblas_mpfr_opt** -lmpfr -lmpc -lgmp

Limitations

Limitations

- Currently it only has the C/C++ (more C-style) interface. There is no Fortran interface (sorry!).
- Setting the precision (number of digits) is restricted by the underlying GMP/MPFR implementations.
- The data objects are opaque, handler functions for output are limited.
- Inter-language operations requires some coding work.

Remarks

- MPLAPACK is a multiple precision version of LAPACK for linear algebra operations.
- It is callable from C/C++, not readily from Fortran, but it has the potential being ported to other languages, thanks to its design philosophy – keep the interfaces identical to original BLAS and LAPACK.
- It is easy to install.
- It is quite straightforward to use, if one is familiar with LAPACK.
- The key building blocks in MPBLAS are optimized, capable of performing parallel processing with the support of OpenMP (on single computer).
- One needs some knowledge in numerical analysis as a guide line and needs to experiment with the accuracy by setting the number of digits.

References

[1] **MPLAPACK**, https://github.com/nakatamaho/mplapack.

[2] **LAPACK**, https://netlib.org/lapack/.

[3] Fredrik Johansson et al, **mpmath**: *A Python library for arbitrary-precision floating-point arithmetic*, http://mpmath.org/.

[4] Fredrik Johansson et al, **Arb**: *A C library for arbitrary-precision interval arithmetic*, https://github.com/fredrik-johansson/arb/.

[5] David Bailey, **MPFUN2020**, David Bailey's website: *High-Precision Software Directory*, https://www.davidhbailey.com/dhbsoftware/.

[6] **Eigen**: *A C++ template library for linear algebra,* https://eigen.tuxfamily.org/.

[7] **Elemental**: A new framework for distributed memory dense linear algebra, https://github.com/elemental/Elemental (halted).

[8] **Julia**, a language for scientific and high performance computing, https://docs.julialang.org/en/v1/.