A (Not So) Simple (Full) Matrix Class

April 5, 2016

Exercise

Reorganize the `fem1d` exercise code in the following way

- Allow the user to specify the problem and algorithm parameters on the command line rather than in a header
  - Use GetPot
- Refactor the code by implementing a separate library for matrices (and cloumn vectors implemented as 1-column matrices) based on STL containers such as std :: vector<> or std :: array<> with the following methods/functions
  - transpose : $A = A^T$
  - gauss-seidel : solve $Ax = b$ iteratively as in previous versions of the example
  - solve : solve $Ax = b$ by means of a direct method
  - operator* : matrix-matrix and matrix-vector multiplication

Solution I

- `matrix-0.1/matrix.{h,cpp}`
  - std :: array<> requires the size to be known at compile time, in order to make the matrix object configurable at run time we use std :: vector<>
  - matrices ar organized as *column major*, *i.e.* $A(i, j) =$ data[i + j * rows ()], conversion from 1d to 2d indexing is performed by the utility (private) function sub2ind
  - access to elements is implemented both in const and non-const versions, by overloading **operator**()
  - data is private, *getter methods* expose what is needed to the user, both const and non-const versions are provided
  - prototypes of fortran77 functions defined as **extern** "C" {...}, assume the compiler adds an underscore (this may change depending on the compiler), define upper and lower case versions

Solution II

- use precompiler conditionals to let user select at compile time which implementation to use
- naive implementation of matrix-matrix multiplication is slow because it has low *data locality* (see below more about memory hierarchy and cache optimized algorithms), simply transposing the left matrix factor improves performance significantly
- BLAS and LAPACK provide cache optimized linear algebra subroutines, optimizations are hardware dependent, the interface of BLAS and LAPACK is fixed, different implementations with different performance are available
  - ATLAS does self tuning via experiments
  - OpenBlas has hand-tuned versions for most common architectures
  - vendor-specific implementations (Intel MKL, Apple vecLib, . . .)
- **#include** <ctime> header provides timing utilities, tic () and toc (x) macros start and stop the timer (like in Matlab)

- `matrix-0.1/Makefile`

Solution III

- build static and/or shared library
- user provided flags are passed on to the preprocessor, compiler and linker

- `matrix-0.1/test_*.cpp`
  - for each feature a test to verify consistency and/or performance is provided
  - `run_test.m` builds `test_matrix_mult` with various options and compares performance of various implementations

- `fem1d-0.4/fem1d.h`
  - define help text using std::string
  - remove config.h

- `fem1d-0.4/fem1d.cpp`
  - define problem/algorithm constants using GetPot methods

Solution IV

- use matrix objects to define matrices and vectors

- `fem1d-0.4/Makefile`

  - include flags for matrix headers and libraries

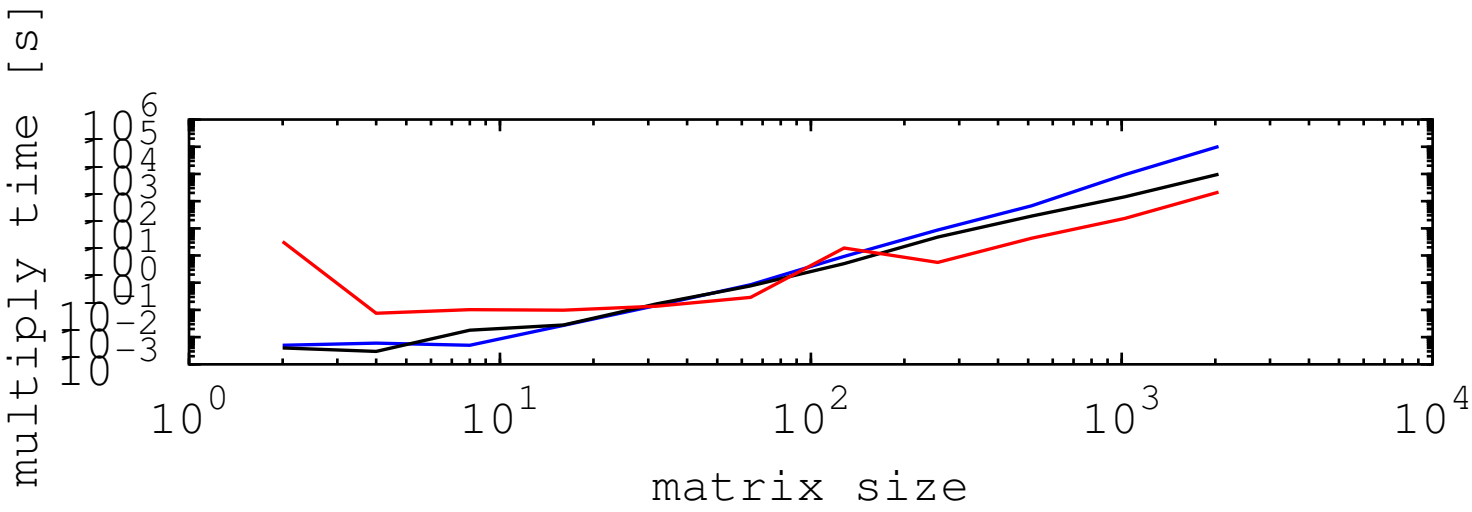

Figure: Performance of the different implementations of the matrix multiplication: 1) standard (blue), 2) with transpose of first factor (black), 3) DGEMM (red). For a matrix of size 2^{11} the algorithm 2) is 10 times faster than 1) and algorithm 3) is 64 times faster. The comparison was run on MacBook Pro with core2-duo CPU, code was compiled with g++ 4.8 and linked to OpenBlas.

Matrix Class (Header)

Lines 1–25 / 85

```
1  #ifndef HAVE_MATRIX_H
2  #define HAVE_MATRIX_H
3
4  #include <vector>
5  #include <iostream>
6
7  class
8  matrix
9  {
10
11 private :
12
13   std::vector<double> data;
14   const unsigned int rows;
15   const unsigned int cols;
16
17   inline
18   unsigned int
19   sub2ind (const unsigned int ir,
20            const unsigned int jc) const
21   { return (ir + jc * rows); };
22
23   double &
24   index (unsigned int irow, unsigned int jcol)
25   { return data[sub2ind (irow, jcol)]; };
```

Matrix Class (Header)

Lines 25–49 / 85

```
25      { return data[sub2ind (irow, jcol)]; };
26
27      const double &
28      const_index (unsigned int irow, unsigned int jcol) const
29      { return data[sub2ind (irow, jcol)]; };
30
31    public :
32
33      matrix (unsigned int size)
34        : rows (size), cols (size)
35      { data.resize (rows * cols, 0.0); };
36
37      matrix (unsigned int rows_,
38              unsigned int cols_)
39        : rows (rows_), cols (cols_)
40      { data.resize (rows * cols, 0.0); };
41
42      matrix (matrix const &) = default;
43
44      unsigned int
45      get_rows () const { return rows; }
46
47      unsigned int
48      get_cols () const { return cols; }
```

Matrix Class (Header)

Lines 49–73 / 85

```
49
50      double &
51      operator() (unsigned int irow, unsigned int jcol)
52      { return index (irow, jcol); };
53
54      const double &
55      operator()  (unsigned int irow, unsigned int jcol) const
56      { return const_index (irow, jcol); };
57
58      const double *
59      get_data () const { return &(data[0]); };
60
61      double *
62      get_data () { return &(data[0]); };
63
64      /// transposed matrix : B = A'
65      matrix
66      transpose () const;
67
68      void
69      solve (matrix &rhs);
70
71      void
72      gauss_seidel
73      (const matrix &f,
```

Matrix Class (Header)

Lines 73–97 / 85

```
73      (const matrix &f,
74       matrix &uh,
75       const unsigned int maxit = 100,
76       const double tol = 1.e-9);
77
78  };
79
80  /// matrix x matrix product : C = A * B
81  matrix
82  operator* (const matrix& A, const matrix& B);
83
84  #endif
```

Matrix Class (Implementation)

Lines 1–25 / 139

```
1  #include "matrix.h"
2  #include <cassert>
3  #include <algorithm>
4  #include <ctime>
5
6  #define dgesv dgesv_
7  #define DGESV dgesv_
8
9  #define dgemm dgemm_
10 #define DGEMM dgemm_
11
12 extern "C"
13 {
14   void
15   dgesv (const int *N, const int *NRHS, const double *A,
16          const int *LDA, int *IPIV, double *B, const int *LDB,
17          int *INFO);
18
19   void
20   dgemm (const char *TRANSA, const char *TRANSB, const int *M,
21          const int *N, const int *K, const double *ALPHA,
22          const double *A, const int *LDA, const double *B,
23          const int *LDB, const double *BETA, double *C,
24          const int *LDC);
25 }
```

Matrix Class (Implementation)

Lines 25–49 / 139

```
25  }
26
27
28  matrix
29  matrix::transpose () const
30  {
31    matrix retval (get_cols (), get_rows ());
32    unsigned int i, j;
33    for (i = 0; i < rows; ++i)
34      for (j = 0; j < cols; ++j)
35        retval (i, j) = const_index (j, i);
36    return (retval);
37  }
38
39  #ifdef USE_DGEMM
40  matrix
41  operator* (const matrix& A, const matrix& B)
42  {
43
44    int M = A.get_rows ();
45    int N = B.get_cols ();
46    int K = A.get_cols ();
47    assert (K == B.get_rows ());
48
49    char ntr = 'n';
```

Matrix Class (Implementation)

Lines 49–73 / 139

```
49      char ntr = 'n';
50      double one = 1.0;
51      double zero = 0.0;
52
53      matrix retval (M, N);
54
55      dgemm (&ntr, &ntr, &M, &N, &K,
56             &one, A.get_data (), &M,
57             B.get_data (), &K, &zero,
58             retval.get_data (), &M);
59
60      return (retval);
61    }
62  #elif defined (MAKE_TMP_TRANSP)
63  matrix
64  operator* (const matrix& A, const matrix& B)
65  {
66    assert (A.get_cols () == B.get_rows ());
67    matrix retval (A.get_rows (), B.get_cols ());
68    matrix tmp = A.transpose ();
69    for (unsigned int i = 0; i < retval.get_rows (); ++i)
70      for (unsigned int j = 0; j < retval.get_cols (); ++j)
71        for (unsigned int k = 0; k < A.get_cols (); ++k)
72          retval(i,j) += tmp(k,i) * B(k,j);
73    return (retval);
```

Matrix Class (Implementation)

Lines 73–97 / 139

```
73      return (retval);
74    }
75    #else
76    matrix
77    operator* (const matrix& A, const matrix& B)
78    {
79      assert (A.get_cols () == B.get_rows ());
80      matrix retval (A.get_rows (), B.get_cols ());
81      for (unsigned int i = 0; i < retval.get_rows (); ++i)
82        for (unsigned int j = 0; j < retval.get_cols (); ++j)
83          for (unsigned int k = 0; k < A.get_cols (); ++k)
84            retval(i,j) += A(i,k) * B(k,j);
85      return (retval);
86    }
87    #endif
88
89    void
90    matrix::solve (matrix &rhs)
91    {
92      int N = get_rows ();
93      int IPIV[N];
94      int NRHS = rhs.get_cols ();
95      int LDB = rhs.get_rows ();
96      int INFO = 0;
97      matrix tmp ((*this));
```

Matrix Class (Implementation)

Lines 97–121 / 139

```
97      matrix tmp ((*this));
98      dgesv (&N, &NRHS, tmp.get_data (), &N,
99             IPIV, rhs.get_data (), &LDB, &INFO);
100   };
101
102
103   void
104   matrix::gauss_seidel
105   (const matrix &f,
106    matrix &uh,
107    const unsigned int maxit,
108    const double tol)
109   {
110
111     assert (uh.get_rows () == f.get_rows ());
112     assert (uh.get_rows () == get_rows ());
113     assert (uh.get_cols () == 1);
114
115     double uh_new = 0;
116     double incrnorm = 0;
117
118     for (unsigned int ii = 0; ii < maxit; ++ii)
119       {
120         incrnorm = 0;
121         for (unsigned int jj = 0; jj < get_rows (); ++jj)
```

Matrix Class (Implementation)

Lines 121–145 / 139

```
121         for (unsigned int jj = 0; jj < get_rows (); ++jj)
122           {
123             double res = f(jj, 0);
124
125             for (unsigned int kk = 0; kk < get_cols (); ++kk)
126               if (kk != jj)
127                 res -= const_index(jj,kk) * uh(kk, 0);
128
129             uh_new = res / const_index(jj,jj);
130             incrnorm = std::abs (uh_new - uh(jj, 0)) > incrnorm ?
131               std::abs (uh_new - uh(jj, 0)) :
132               incrnorm;
133             uh(jj, 0) = uh_new;
134           }
135         if (incrnorm < tol)
136           break;
137       }
138   };
```

Matrix Class (Test Multiply)

Lines 1–24 / 24

```
1  #include "matrix.h"
2  #include "test_matrix_mult.h"
3  int main ()
4  {
5
6    matrix A(msize);
7    matrix B(msize);
8    for (unsigned int i = 0; i < msize; ++i)
9      {
10       A(i,i) = 10.0;
11       A(i,msize-1) = 30.0;
12       B(i,0) = 1.0;
13       B(i,i) = 3.0;
14     }
15
16   std::cout << "msize = " << msize << std::endl;
17
18   tic ();
19   matrix C = A * B;
20   toc ("multiply_time = ");
21
22   return 0;
23 }
```

Matrix Class (Test Gauß-Seidel)

Lines 1–25 / 37

```
1  #include "test_gauss_seidel.h"
2
3  int main ()
4  {
5
6    matrix A (10);
7    matrix rhs (10, 1);
8
9    for (unsigned int i = 0; i < 10; ++i)
10     {
11       rhs (i, 0) = 1.;
12       for (unsigned int j = 0; j < 10; ++j)
13         {
14           if (j+1 == i)
15             A(i, j) = -1.;
16           else if (j == i+1)
17             A(i, j) = -1.;
18           else if (i == j)
19             A(i, j) = 2.;
20         }
21     }
22
23   matrix uh (rhs);
24   A.gauss_seidel (rhs, uh, 1000);
```

Matrix Class (Test Gauß-Seidel)

Lines 25–49 / 37

```
25
26      matrix f = A * uh;
27
28      for (unsigned int i = 0; i < 10; ++i)
29        {
30          std::cout << "x ( " << i+1 << " ) = " << uh(i, 0) << ";"
31                    << "    y ( " << i+1 << " ) = " << f(i, 0) << ";"
32                    << "    b ( " << i+1 << " ) = " << rhs(i, 0) << ";"
33                    << std::endl;
34        }
35      return 0;
36    }
```

Matrix Class (Test Solve)

Lines 1–25 / 38

```
1  #include "test_solve.h"
2  #include <iostream>
3
4  int main ()
5  {
6
7    matrix A (10);
8    matrix rhs (10, 1);
9
10   for (unsigned int i = 0; i < 10; ++i)
11     {
12       rhs (i, 0) = 1.;
13       for (unsigned int j = 0; j < 10; ++j)
14         {
15           if (j+1 == i)
16             A(i, j) = -1.;
17           else if (j == i+1)
18             A(i, j) = -1.;
19           else if (i == j)
20             A(i, j) = 2.;
21         }
22     }
23
24   matrix uh (rhs);
25   A.solve (uh);
```

Matrix Class (Test Solve)

Lines 25–49 / 38

```
25      A.solve (uh);
26
27      matrix f = A * uh;
28
29      for (unsigned int i = 0; i < 10; ++i)
30        {
31          std::cout << "x␣(" << i+1 << ")␣=␣" << uh(i, 0) << ";"
32                    << "␣␣␣␣y␣(" << i+1 << ")␣=␣" << f(i, 0) << ";"
33                    << "␣␣␣␣b␣(" << i+1 << ")␣=␣" << rhs(i, 0) << ";"
34                    << std::endl;
35        }
36      return 0;
37    }
```

Matrix Class (run_test.m)

Lines 1–25 / 57

```
1
2
3   fmt = ...
4    'make␣CXX="g++"␣CPPFLAGS="-I.␣-Umsize␣-Dmsize=%d␣%s"␣distclean␣test_matrix_mult\
5
6   m   = [];
7   mm  = [];
8   mmm = [];
9   d   = [];
10
11  for ii = 1 : 11
12
13    printf (fmt, 2^ii, "");
14    [a, str] = system (sprintf (fmt, 2^ii, ""));
15    [a, str] = system ('./test_matrix_mult');
16    if (a != 0)
17      disp (str)
18      return
19    endif
20    eval (str);
21    d = [d; msize];
22    m = [m; multiply_time];
23
24    printf (fmt, 2^ii, "-DMAKE_TMP_TRANSP");
25    [a, str] = system (sprintf (fmt, 2^ii, "-DMAKE_TMP_TRANSP"));
```

Matrix Class (run_test.m)

Lines 25–49 / 57

```
25      [a, str] = system (sprintf (fmt, 2^ii, "-DMAKE_TMP_TRANSP"));
26      [a, str] = system ('./test_matrix_mult');
27      if (a != 0)
28        disp (str)
29        return
30      endif
31      eval (str);
32      mm = [mm; multiply_time];
33
34      printf (fmt, 2^ii, "-DUSE_DGEMM");
35      [a, str] = system (sprintf (fmt, 2^ii, "-DUSE_DGEMM"));
36      [a, str] = system ('./test_matrix_mult');
37      if (a != 0)
38        disp (str)
39        return
40      endif
41      eval (str);
42      mmm = [mmm; multiply_time];
43
44      loglog (d, m, 'linewidth', 5,
45              d, mm, 'linewidth', 5,
46              d, mmm, 'linewidth', 5);
47
48      set (gca (), "FontSize", 18)
49      legend ("without transpose", "with transpose", "whith dgemm",
```

Matrix Class (run_test.m)

Lines 49–73 / 57

```
49      legend ("without transpose", "with transpose", "whith dgemm",
50              "location", "southoutside")
51      xlabel ("matrix size", "FontSize", 18)
52      ylabel ("multiply time [ms]", "FontSize", 18)
53      axis tight
54      drawnow
55    endfor
56
57    print -dpdf speedup.pdf
```

Matrix Class Library Makefile

Lines 1–25 / 43

```
1  OBJS =  matrix.o
2  TEST_OBJS = test_matrix_mult.o test_gauss_seidel.o test_solve.o
3  TESTS = test_matrix_mult test_gauss_seidel test_solve
4  HEADERS = matrix.h
5  SHARED_EXT = so
6
7  .PHONY : tests
8
9  tests : $(TESTS)
10
11 libmatrix.a : $(OBJS)
12         $(AR) crv libmatrix.a $(OBJS)
13
14 libmatrix.$(SHARED_EXT) : $(OBJS)
15         $(CXX) -shared -L. $(LDFLAGS) -o $@ $(OBJS) $(LIBS)
16
17 test_matrix_mult : test_matrix_mult.o libmatrix.a
18         $(CXX) -L. $(LDFLAGS) -o $@  $< -lmatrix $(LIBS)
19
20 test_gauss_seidel : test_gauss_seidel.o libmatrix.a
21         $(CXX) -L. $(LDFLAGS) -o $@  $< -lmatrix $(LIBS)
22
23 test_solve : test_solve.o libmatrix.a
24         $(CXX) -L. $(LDFLAGS) -o $@ $< -lmatrix $(LIBS)
```

Matrix Class Library Makefile

Lines 25–49 / 43

```
25
26  speedup.pdf : run_test.m
27          octave $<
28
29  $(OBJS) : %.o : %.cpp %.h
30          $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
31
32  $(TEST_OBJS) : %.o : %.cpp %.h
33          $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
34
35  .PHONY: clean distclean
36
37  clean :
38          $(RM) $(OBJS) $(TEST_OBJS)
39
40  distclean : clean
41          $(RM) $(TESTS) \
42          speedup.pdf libmatrix.a libmatrix.$(SHARED_EXT)
```

fem1d main

Lines 1–25 / 79

```
1   #include "fem1d.h"
2   #include "matrix.h"
3   #include "GetPot"
4
5   int main (int argc, char **argv)
6   {
7
8     GetPot cl (argc, argv);
9
10    if (cl.search (2, "-h", "--help"))
11      {
12        std::cerr << help_text << std::endl;
13        return 0;
14      }
15    const double a = cl.follow (0.0, "-a");
16    const double b = cl.follow (1.0, "-b");
17    const unsigned int nnodes = cl.follow (100, 2, "-n", "-nnodes");
18
19    mesh m (a, b, nnodes);
20    matrix A(nnodes);
21
22    matrix mloc(2);
23    for (unsigned int iel = 0; iel < m.nels; ++iel)
24      {
```

fem1d main

Lines 25–49 / 79

```
25
26          std::fill (mloc.get_data (),
27                     mloc.get_data () + 4,
28                     0.0);
29
30          for (unsigned int inode = 0; inode < 2; ++inode)
31            {
32              double igrad = (inode == 0 ? 1.0 / m.h : -1.0 / m.h);
33              for (unsigned int jnode = 0; jnode < 2; ++jnode)
34                {
35                  double jgrad =  (jnode == 0 ? 1.0 / m.h : -1.0 / m.h);
36                  mloc(inode,jnode) = igrad * jgrad * m.h;
37                  A(m.elements[iel][inode],m.elements[iel][jnode]) +=
38                    mloc(inode,jnode);
39                }
40            }
41        }
42
43      matrix f(nnodes, 1);
44      matrix vloc(2, 1);
45
46      for (unsigned int iel = 0; iel < m.nels; ++iel)
47        {
48          std::fill (mloc.get_data (),
49                     mloc.get_data () + 2,
```

fem1d main

Lines 49–73 / 79

```
49                      mloc.get_data () + 2,
50                      0.0);
51
52          for (unsigned int inode = 0; inode < 2; ++inode)
53            {
54              vloc(inode, 0) = m.h / 2.0;
55              f(m.elements[iel][inode], 0) += vloc(inode, 0);
56            }
57        }
58
59      f(0, 0) = 0;
60      f(nnodes - 1, 0) = 0;
61
62      A(0,0) = 1.0;
63      A(nnodes-1,nnodes-1) = 1.0;
64      for (unsigned int ii = 1; ii < nnodes; ++ii)
65        {
66          A(0, ii) = 0.0;
67          A(nnodes-1, nnodes-1-ii) = 0.0;
68        }
69
70
71      matrix uh(f);
72      A.gauss_seidel (f, uh);
```

fem1d main

Lines 73–97 / 79

```
73
74      for (unsigned int ii = 0; ii < nnodes; ++ii)
75        std::cout << uh(ii, 0) << std::endl;
76
77      return 0;
78    };
```

fem1d Header

Lines 1–24 / 24

```
1  #ifndef HAVE_FEM1D_H
2  #define HAVE_FEM1D_H
3
4  #include <array>
5  #include <algorithm>
6  #include <cmath>
7  #include <iostream>
8  #include <string>
9
10 #include "mesh.h"
11
12
13 std::string help_text = std::string () +
14 "fem1d command line options\n" +
15   "-h, --help                print this text and exit\n" +
16   "-a <value>                first end of interval\n" +
17   "-b <value>                second end of interval\n" +
18   "-n, --nnodes <value>      number of triangulation nodes\n" +
19   "-m, --maxit <value>       number of iterations\n" +
20   "-t, --tol <value>         tolerance\n";
21
22
23 #endif
```

fem1d Executable Makefile

Lines 1–20 / 20

```
1  OBJS = fem1d.o mesh.o
2  HEADERS = config.h
3  CPPFLAGS ?= -I../matrix-0.1/ -I.
4  LDFLAGS ?= -L../matrix-0.1/
5  LIBS =
6
7  fem1d : $(OBJS)
8          $(CXX) $(LDFLAGS) $(OBJS) -o $@ -lmatrix $(LIBS)
9
10 $(OBJS) : %.o : %.cpp %.h $(HEADERS)
11         $(CXX) $(CPPFLAGS) $(CXXFLAGS) -c $<
12
13 .PHONY: clean distclean
14
15 clean :
16         $(RM) $(OBJS)
17
18 distclean : clean
19         $(RM) fem1d
```

Memory hierarchy and Cache Optimized Algorithms

Memory hierarchy and Cache Optimized Algorithms

insert pages 1-9 of file images/slides.pdf

BLAS and LAPACK

BLAS and LAPACK

insert pages 10-20 of file images/slides.pdf