

ETH Lectures 401-0663-00L Numerical Methods for Computer Science
401-2673-00L Numerical Methods for CSE

Homework Problems and Projects

Prof. R. Hiptmair, SAM, ETH Zurich

Autumn Term 2021, Version of October 2, 2022
(C) Seminar für Angewandte Mathematik, ETH Zürich

[Link](#) to the current version of this homework collection

0.1 General Information

0.1.1 Weekly Homework Assignments

All problems will be published in this single “.pdf” file. Every week, we publish a list of problem numbers on the lecture Moodle page. These are the current assignments for that week. Details are given on the lecture Moodle page.

0.1.2 Importance of Homework

Homework assignments are **not** mandatory. However, it is **very important** that you constantly exercise with the material you learn. “Solving” the homework assignments one week before the main exam by looking at the solutions will likely result in failure.

We provide hints and solutions from the beginning. It is your responsibility to look at the solutions only if you are stuck with a difficult problem and you tried to find a solution for a sufficient amount of time.

Most homework problems come with an estimate for the time required for its solution. These times are what we expect a bright and well-prepared student to require for that problem during an exam.

0.1.3 Corrections and Grading of Assignments

Homework assignments will not be graded.

However, you may submit your handwritten solutions by handing them to your tutor directly or uploading them via the Moodle upload interface, see the lecture Moodle page for instructions. The assistants will

Problem 2-19: A special Sylvester Equation

A **Sylvester equations** is a *linear matrix equation* of the form

$$\mathbf{A}\mathbf{X} + \mathbf{X}\mathbf{B} = \mathbf{C} \quad , \quad \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n,n} \quad ,$$

which yields a linear system of equations for the entries of the unknown matrix $\mathbf{X} \in \mathbb{R}^{n,n}$. In this exercise we consider a very special specimen of Sylvester equation and recast it for the sake of efficient direct solution with EIGEN.

Related to various topics from class and also Problem 2-8.

(2-19.a) □ (5 min.) Refresh your knowledge about s.p.d. matrices, see [Lecture → Def. 1.1.2.6] and [Lecture → Lemma 1.1.2.7]. ▲

(2-19.b) □ (5 min.) Recall the “vectorization” $\text{vec}(\mathbf{X}) \in \mathbb{R}^{n^2}$ of a matrix $\mathbf{X} \in \mathbb{R}^{n,n}$ from [Lecture → Rem. 1.2.3.4] and jot down $\text{vec}(\mathbf{X})$ for

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \in \mathbb{R}^{2,2} \quad .$$

SOLUTION of (2-19.b):

The vectorized matrix is obtained by stacking the columns of the matrix on top of each other:

$$\blacktriangleright \quad \text{vec}(\mathbf{X}) = [x_{11}, x_{21}, x_{12}, x_{22}]^\top \in \mathbb{R}^4 \quad .$$

(2-19.c) □ (15 min.) For $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{2,2}$ find the matrix $\mathbf{C} \in \mathbb{R}^{4,4}$ such that

$$\text{vec}(\mathbf{S}\mathbf{X}\mathbf{T}) = \mathbf{C} \text{vec}(\mathbf{X}) \quad \forall \mathbf{X} \in \mathbb{R}^{2,2} \quad ,$$

that is, express the entries of $\mathbf{C}$ in terms of the entries $(\mathbf{S})_{i,j}$ and $(\mathbf{T})_{i,j}$ of $\mathbf{S}$ and $\mathbf{T}$.

SOLUTION of (2-19.c):

We obtain the k -th column of $\mathbf{C}$ as

$$\mathbf{C}_{:,k} = \text{vec}(\mathbf{S} \text{vec}^{-1}(\mathbf{e}_k) \mathbf{T}) \quad ,$$

where $\mathbf{e}_k \in \mathbb{R}^4$ is the k -th unit vector, $k = 1, 2, 3, 4$. For instance

$$\begin{aligned} \mathbf{C}_{:,1} &= \text{vec} \left(\begin{bmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \right) = \text{vec} \left(\begin{bmatrix} s_{11} & 0 \\ s_{21} & 0 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix} \right) \\ &= \text{vec} \left(\begin{bmatrix} s_{11}t_{11} & s_{11}t_{12} \\ s_{21}t_{11} & s_{21}t_{12} \end{bmatrix} \right) = \begin{bmatrix} s_{11}t_{11} \\ s_{21}t_{11} \\ s_{11}t_{12} \\ s_{21}t_{12} \end{bmatrix} \quad . \end{aligned}$$

Thus we find

$$\mathbf{C} = \begin{bmatrix} s_{11}t_{11} & s_{12}t_{11} & s_{11}t_{21} & s_{12}t_{21} \\ s_{21}t_{11} & s_{22}t_{11} & s_{21}t_{21} & s_{22}t_{21} \\ s_{11}t_{12} & s_{12}t_{12} & s_{11}t_{22} & s_{12}t_{22} \\ s_{21}t_{12} & s_{22}t_{12} & s_{21}t_{22} & s_{22}t_{22} \end{bmatrix} = \mathbf{T}^\top \otimes \mathbf{S}.$$

▲

(2-19.d) 🎲 (20 min.) Based on the **Kronecker product** $\otimes$ from [Lecture → Def. 1.4.3.7] write down a matrix $\mathbf{C} \in \mathbb{R}^{n^2, n^2}$ depending on the given square matrices $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{n, n}$ such that

$$\text{vec}(\mathbf{SXT}) = \mathbf{C} \text{vec}(\mathbf{X}).$$

HINT 1 for (2-19.d): Read [LAN70].

┘

SOLUTION of (2-19.d):

By the rules of matrix multiplication for $\mathbf{S}, \mathbf{X}, \mathbf{T} \in \mathbb{R}^{n, n}$:

$$(\mathbf{AXT})_{i,j} = \sum_{\ell=1}^n \sum_{v=1}^n s_{i,\ell} x_{\ell,v} t_{v,j}, \quad (2.19.1)$$

where we abbreviated $s_{i,\ell} = (\mathbf{S})_{i,\ell}$, $x_{\ell,v} := (\mathbf{X})_{\ell,v}$, $t_{v,j} := (\mathbf{T})_{v,j}$.

We can extract a column of $\mathbf{C}$ by multiplying with a unit vector. Hence, we consider $\mathbf{E}_{k,m}$, the all-zero matrix except for an entry $= 1$ at position (k, m) . From (2.19.1) we find

$$(\mathbf{SE}_{k,m}\mathbf{T})_{i,j} = s_{i,k} t_{m,j}. \quad (2.19.2)$$

Write $P(i, j) \in \{1, \dots, n^2\}$ for the position of the matrix entry $(\mathbf{Z})_{i,j}$ in the vector $\text{vec}(\mathbf{Z})$, $\mathbf{Z} \in \mathbb{R}^{n, n}$, $i, j \in \{1, \dots, n\}$. Hence, from (2.19.2) we conclude

$$(\mathbf{C})_{P(i,j), P(k,m)} = s_{i,k} t_{m,j} = (\mathbf{T}^\top \otimes \mathbf{S})_{P(i,j), P(k,m)} \Leftrightarrow \mathbf{C} = \mathbf{T}^\top \otimes \mathbf{S}. \quad (2.19.3)$$

▲

Throughout the remainder of this problem let $\mathbf{A} \in \mathbb{R}^{n, n}$, $n \in \mathbb{N}$, stand for a **symmetric, positive definite** (s.p.d.) matrix, whose rows and columns contain at most five non-zero entries. Hence, for large n the matrix $\mathbf{A}$ can be considered **sparse** in the sense of [Lecture → Notion 2.7.0.1].

Then consider the special Sylvester equation:

$$\text{seek } \mathbf{X} \in \mathbb{R}^{n, n}: \quad \mathbf{XA}^{-1} + \mathbf{AX} = \mathbf{I}, \quad (2.19.4)$$

and the equivalent matrix equation:

$$\text{seek } \mathbf{X} \in \mathbb{R}^{n, n}: \quad \mathbf{X} + \mathbf{AXA} = \mathbf{A}. \quad (2.19.5)$$

(2-19.e) 🕒 (5 min.) Both (2.19.4) and (2.19.5) can be recast as an $n^2 \times n^2$ linear system of equations for the unknown vector $\text{vec}(\mathbf{X})$. Based on the result of Sub-problem (2-19.d) express their coefficient matrices by means of the Kronecker product [Lecture → Def. 1.4.3.7].

SOLUTION of (2-19.e):

According to (2.19.3) the coefficient matrices $\mathbf{C} \in \mathbb{R}^{n^2, n^2}$ are ($\mathbf{I} \in \mathbb{R}^{n, n}$ the identity matrix)

$$\text{For (2.19.4): } \mathbf{C}_4 = \mathbf{A}^{-1} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}, \quad (2.19.6)$$

$$\text{For (2.19.5): } \mathbf{C}_5 = \mathbf{I} \otimes \mathbf{I} + \mathbf{A} \otimes \mathbf{A}. \quad (2.19.7)$$

Here we have made use of the symmetry of $\mathbf{A}$.



(2-19.f) 🕒 (10 min.) Based on the result of Sub-problem (2-19.e) and in light of the observation made in [Lecture → Ex. 2.7.4.4] give a sharp upper bound for the number of non-zero entries of the coefficient matrices of those linear systems for (2.19.4) and (2.19.5).

HINT 1 for (2-19.f):

Inverses of regular sparse matrices can be densely populated!

$$\begin{bmatrix} 1 & -1 & 0 & \dots & \dots & 0 \\ 0 & 1 & -1 & 0 & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & 0 & 1 & -1 \\ 0 & \dots & \dots & 0 & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 1 & 1 & \dots & \dots & 1 \\ 0 & 1 & 1 & \dots & \dots & 1 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & \ddots & \ddots & \ddots & \vdots \\ & & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & 1 \end{bmatrix}$$



SOLUTION of (2-19.f):

By the properties of $\mathbf{A}$ we know

$$\text{nnz}(\mathbf{A}) \leq 5n.$$

However, as explained in the hint, $\text{nnz}(\mathbf{A}^{-1}) = n^2$ is possible. Moreover, for the Kronecker product we have the simple estimate

$$\text{nnz}(\mathbf{S} \otimes \mathbf{T}) \leq \text{nnz}(\mathbf{S}) \cdot \text{nnz}(\mathbf{T}). \quad (2.19.8)$$

Combining all these estimates gives us

$$\begin{aligned} \text{nnz}(\mathbf{A}^{-1} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{A}) &\leq n^3 + 5n^2, \\ \text{nnz}(\mathbf{I} \otimes \mathbf{I} + \mathbf{A} \otimes \mathbf{A}) &\leq 26n^2. \end{aligned}$$

Hence, (2.19.5) features a much sparser coefficient matrix than (2.19.4).



(2-19.g) 🕒 (15 min.) Show that both (2.19.4) and (2.19.5) give rise to linear systems of equations for the entries of $\mathbf{X}$ with *s.p.d.* coefficient matrices.

For your argument you can appeal to the following result:

Theorem 2.19.9. Eigenvalues of Kronecker-product matrices

Assume that both $\mathbf{S}, \mathbf{T} \in \mathbb{R}^{n,n}$ can be diagonalized and write $\sigma(\mathbf{S}), \sigma(\mathbf{T}) \subset \mathbb{C}$ for the sets of their eigenvalues. Then the set of eigenvalues of the Kronecker product is given by

$$\sigma(\mathbf{S} \otimes \mathbf{T}) = \{\lambda\mu : \lambda \in \sigma(\mathbf{S}), \mu \in \sigma(\mathbf{T})\}.$$

HINT 1 for (2-19.g): In linear algebra we learned that symmetric matrices can always be diagonalized.

HINT 2 for (2-19.g): Remember [Lecture $\rightarrow$ Lemma 1.1.2.7].

SOLUTION of (2-19.g):

By the very definition of the Kronecker product, both matrices $\mathbf{C}_4$ and $\mathbf{C}_5$ from (2.19.6) and (2.19.7), respectively, are symmetric. Since both $\mathbf{A}$ and $\mathbf{A}^{-1}$ are s.p.d. all of their eigenvalues are positive. Hence, by the formula of Thm. 2.19.9 this also holds for $\mathbf{C}_4$ and $\mathbf{C}_5$. Thus, by the third criterion from [Lecture $\rightarrow$ Lemma 1.1.2.7], these matrices are s.p.d.



(2-19.h) 🕒 (15 min.) Temporarily assume that $\mathbf{A}$ is a *diagonal* matrix. Implement a C++ function

```
template <typename Vector>
Eigen::SparseMatrix<double> solveDiagSylvesterEq(const Vector
&diagA);
```

that solves (2.19.4) with the diagonal of $\mathbf{A}$ passed through the vector argument `diagA`. The result $\mathbf{X}$ is returned as a sparse matrix. The type **Vector** must supply a `size()` method and component access through **operator []**.

SOLUTION of (2-19.h):

If $\mathbf{A}$ is s.p.d. and diagonal with diagonal entries $d_\ell = (\mathbf{A})_{\ell,\ell} > 0$, then (2.19.4) becomes

$$(d_i + d_j^{-1})(\mathbf{X})_{i,j} = \delta_{i,j}, \quad i, j \in \{1, \dots, n\}.$$

Hence, $\mathbf{X}$ will also be diagonal

$$(\mathbf{X})_{i,j} = \begin{cases} \frac{d_i}{d_i^2 + 1} & , \text{ if } i = j, \\ 0 & \text{ else.} \end{cases}$$

Implementation is straightforward, but care must be taken to initialize the result matrix efficiently. Reserve space for the single non-zero entry per row/column, see [Lecture → Code 2.7.2.1].

C++ code 2.19.10: Solving a “diagonal” Sylvester equation

```

2  template <typename Vector>
3  Eigen::SparseMatrix<double> solveDiagSylvesterEq(const Vector &diagA) {
4      int n = diagA.size();
5      Eigen::SparseMatrix<double> X(n, n);
6
7      // TODO: (2-19.h) Fill in the entries of X.
8      // Don't forget to use makeCompressed().
9
10     // START
11     // As X is a diagonal matrix, we reserve space
12     // for one non-zero entry per column.
13     X.reserve(Eigen::VectorXi::Constant(n, 1));
14     for (int j = 0; j < n; j++) {
15         // Set diagonal entry  $X_{j,j} = \frac{d_j}{d_j^2 + 1}$ 
16         X.insert(j, j) = diagA[j] / (diagA[j] * diagA[j] + 1.0);
17     }
18     X.makeCompressed();
19     // END
20
21     return X;
22 }

```

Get it on  GitLab (Sylvester.hpp).



(2-19.i)  (45 min.) Without using EIGEN's (beta-status) implementation of the Kronecker product, realize a C++ function

```

Eigen::SparseMatrix<double> sparseKron(const
    Eigen::SparseMatrix<double> &M);

```

that computes the Kronecker product $\mathbf{M} \otimes \mathbf{M}$ for the square sparse matrix $\mathbf{M} \in \mathbb{R}^{n,n}$ and returns the result in sparse-matrix format.

For the sake of efficient initialization you have to use the `reserve()` function and determine the maximum number of non-zero entries per column for $\mathbf{A}$ in advance. To do this use the member functions of **SparseMatrix** that allow direct access to the data vectors of the CCS format, cf. [Lecture → § 2.7.1.4]:

- **const double *valuePtr() const**: returns a pointer to the array of values of non-zero matrix entries,
- **const StorageIndex* innerIndexPtr() const**: returns a pointer to the array of row indices for every non-zero matrix entry,
- **const StorageIndex* outerIndexPtr() const**: returns a pointer to the vector of starting positions of columns.

These access methods also permit you to run through all non-zero entries of $\mathbf{A}$ and retrieve their locations.

SOLUTION of (2-19.i):

The comments in Code 2.19.11 explain the details of the algorithm.

C++ code 2.19.11: Computing $M \otimes M$ for a sparse matrix M

```

2  Eigen::SparseMatrix<double> sparseKron(const Eigen::SparseMatrix<double> &M) {
3      int n = M.rows();
4      assert(n == M.cols() && "Matrix M must be square.");
5
6      // JM is an array of size n+1, with
7      // JM[j] = number of non-zero entries in all columns 0,..., (j-1) of M.
8      const int *JM = M.outterIndexPtr();
9
10     // Define nnzColwiseM(j) = nnz( M.col(j) )
11     Eigen::VectorXi nnzColwiseM(n);
12     for (int j = 0; j < n; j++) {
13         nnzColwiseM(j) = JM[j + 1] - JM[j];
14     }
15
16     // B will be the Kronecker product of M with itself.
17     Eigen::SparseMatrix<double> B(n * n, n * n);
18
19     // TODO: (2-19.i) Fill in the entries of B.
20     // Hint: You can use nnzColwiseM when reserving space for B.
21     // Use M.valuePtr() and M.innerIndexPtr() to define arrays similar
22     // to JM. Use those arrays to access the non-zero entries of M.
23
24     // START
25
26     Eigen::VectorXi nnzColwiseB;
27     // We have that nnz(B.col(j*n+1)) = nnz(M.col(j)) * nnz(M.col(1)).
28     Eigen::MatrixXi tmp = nnzColwiseM * nnzColwiseM.transpose(); // O(n^2)
29     nnzColwiseB = Eigen::MatrixXi::Map(tmp.data(), n * n, 1);
30
31     // Reserve sufficient space.
32     B.reserve(nnzColwiseB);
33
34     // We have that B.block(i*n, j*n, n, n) = M(i, j) * M,
35     // i.e. B( i*n + k, j*n + l ) = M(i, j) * M(k, l).
36     // Hence, we loop over i, j, k, l.
37
38     // Row indices of non-zero entries (size = nnz):
39     const int *IM = M.innerIndexPtr();
40     // Values of non-zero entries (size = nnz):
41     const double *valM = M.valuePtr();
42
43     // Assuming nnz in each column is bounded by a constant, the below is O(n^2).
44     // Loop over the column index j:
45     for (int j = 0; j < n; j++) {
46         // Loop over the non-zero entries in column j:
47         for (int s = JM[j]; s < JM[j + 1]; s++) {
48             int i = IM[s]; // Row index of non-zero entry #s.
49             double Mij = valM[s]; // M(i, j) = Mij
50
51             // We now compute Mij * M.
52             // Loop over the column index l:
53             for (int l = 0; l < n; l++) {
54                 // Loop over the non-zero entries in column l:
55                 for (int t = JM[l]; t < JM[l + 1]; t++) {
56                     int k = IM[t]; // Row index of non-zero entry #t.

```

```

57     double Mkl = valM[t]; // M(k,l) = Mkl
58     // Insert entry (k,l) of block (i,j).
59     B.insert(i * n + k, j * n + l) = Mij * Mkl;
60 }
61 }
62 }
63 }
64 B.makeCompressed();
65 // END
66
67 return B;
68 }

```

Get it on  GitLab (Sylvester.hpp).



(2-19.j)  (20 min.) [depends on Sub-problem (2-19.i), Sub-problem (2-19.e)]

Write a C++ function

```

Eigen::MatrixXd solveSpecialSylvesterEq(
    const Eigen::SparseMatrix<double> &A);

```

that solves (2.19.4) and returns the solution X .

SOLUTION of (2-19.j):

C++ code 2.19.12: Solving Sylvester equation (2.19.4).

```

2  Eigen::MatrixXd solveSpecialSylvesterEq(const Eigen::SparseMatrix<double> &A) {
3      const int n = A.rows();
4      assert(n == A.cols() && "Matrix A must be square.");
5      Eigen::MatrixXd X(n, n);
6
7      // TODO: (2-19.j) Solve the equation  $X A^{-1} + A X = I$  .
8
9      // START
10     // The equation is equivalent to  $X + A X A = A$ ,
11     // which has the system matrix  $C = \text{kron}(I, I) + \text{kron}(A, A)$  .
12     // Define left hand side of  $C \text{Vec}(X) = b$ .
13     Eigen::SparseMatrix<double> C(n * n, n * n);
14     C.setIdentity();
15     // May involve expensive copying of data?
16     C += sparseKron(A);
17
18     // Define right hand side.
19     // Do not use a "sparse vector"!
20     Eigen::VectorXd b(n * n);
21     b.setZero();
22     const int *JA = A.outerIndexPtr();
23     const int *IA = A.innerIndexPtr();
24     const double *valA = A.valuePtr();
25     // Form b by stacking the columns of A.
26     // Iterate over columns of A:
27     for (int j = 0; j < n; j++) {
28         // Loop over the non-zero entries in column j:

```

```

29   for (int s = JA[j]; s < JA[j + 1]; s++) {
30       const int i = IA[s];    // Row index of non-zero entry #s.
31       b[j * n + i] = valA[s]; // b(j*n+i) = Aij
32   }
33   }
34   // Call sparse direct solver.
35   Eigen::SparseLU<Eigen::SparseMatrix<double>> solver;
36   // Since C is s.p.d., we could also use
37   // SimplicialLLT<Eigen::SparseMatrix<double>> solver;
38   solver.compute(C);
39   // x = Vec(X)
40   Eigen::VectorXd x;
41   x = solver.solve(b);
42   X = Eigen::MatrixXd::Map(x.data(), n, n);
43   // END
44
45   return X;
46 }

```

Get it on  GitLab (Sylvester.hpp).



End Problem 2-19 , 155 min.