NumCSE Mock Exam, HS 2018

1. *Estimating point locations from distances* [14 pts.]

Let $n \in \mathbb{N}$, $n > 2$ points be located on the real axis. The leftmost point is fixed at origin, whereas the other points are at unknown locations:

$$\begin{aligned} x_i &\in \mathbb{R}, \quad \text{for } i = 1, 2, \ldots, n, \\ x_{i+1} &> x_i, \\ x_1 &= 0. \end{aligned}$$

The distances $d_{i,j} := |x_i - x_j|$, $\forall i, j \in \{1, 2, \ldots, n\}$, $i > j$ are measured and arranged in a vector

$$\mathbf{d} := [d_{2,1}, d_{3,1}, \ldots, d_{n,1}, d_{3,2}, d_{4,2}, \ldots, d_{n,n-1}]^\top \in \mathbb{R}^m,$$

where $m = n(n-1)/2$. Assume that there are no measurement errors.

Some templates are provided in `estimatePositions.cpp`, write your code in the template coresponding to the instructions in the tasks below.

(a) To determine the unknown point locations using the distance measurements, formulate a linear least squares problem

$$\mathbf{z}^* = \underset{\mathbf{z} \in \mathbb{R}^{n-1}}{\arg\min} \|\mathbf{A}\mathbf{z} - \mathbf{b}\| . \tag{1}$$

[2 pts.]

Solution:

We find that

$$x_i - x_j = d_{ij}, 1 \le j < i \le n.$$

This can be written as

$$\begin{bmatrix} -1 & 1 & 0 & \dots & & & 0 \\ -1 & 0 & 1 & 0 & & & \\ \vdots & & \ddots & \ddots & & & \\ & & & & & & \\ -1 & \dots & & & & 0 & 1 \\ 0 & -1 & 1 & 0 & \dots & & 0 \\ 0 & -1 & 0 & 1 & 0 & & \vdots \\ \vdots & & & & & & \\ & 0 & -1 & 1 & 0 & & \\ & & & & & & \\ 0 & \dots & & & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_{2,1} \\ d_{3,1} \\ \vdots \\ \\ d_{n,1} \\ d_{3,2} \\ d_{4,2} \\ \vdots \\ d_{4,3} \\ \vdots \\ d_{n,n-1} \end{bmatrix}. \tag{2}$$

Setting $x_1 := 0$ amounts to dropping the first column of the system matrix. The remaining matrix is the matrix $\mathbf{A}$ from (1), which is of the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_{n-1} \\ * \end{bmatrix} \in \mathbb{R}^{m,n-1}.$$

Since the top $(n-1) \times (n-1)$ block is the identity matrix, $\mathbf{A}$ must have full rank.

(b) Write an EIGEN based C++ implementation

```
using namespace Eigen;

SparseMatrix<double> buildDistanceLSQMatrix(int n);
```

which initializes the system matrix $\mathbf{A}$ from (1) in an efficient manner for large n. [2 pts.]

Solution:

The matrix $\mathbf{A}$ is sparse with $2m - (n-1) = (n-1)^2 < \frac{n(n-1)^2}{2}$ non-zero entries. The signature of the function `buildDistanceLSQMatrix` already imposes the usage of sparse matrix data formats. There are two alternative methods that guarantee an efficient implementation.

- Matrix assembly via intermediate triplet format:
  i. A vector of triplets is preallocated.
     This is possible, because we know that $\mathbf{A}$ has a total of $2m - (n-1) = (n-1)^2$ non-zero entries. The vector is then filled with triplets.
  ii. Initialization via an intermediate triplet (COO) format and `EIGEN`'s method `setFromTriplets()`.
- Direct entry specification via `SparseMatrix<T>::insert` (also `SparseMatrix<T>::coeffRef` is accepted). To avoid unnecessary memory reallocations, `SparseMatrix<T>::reserve` must be called with an appropriate estimate.

```
SparseMatrix<double> buildDistanceLSQMatrix(int n) {
    SparseMatrix<double> A(n*(n-1)/2, n-1);

    // Assembly
    std::vector<Triplet<double>> triplets; // List of non-zeros
        coefficients
    triplets.reserve((n-1)*(n-1)); // Two non-zeros per row (at most),
        first n-1 rows only one entry
    // --> (n-1)^2 total non-zero entries

    // Loops over vertical blocks
    int row = 0; // Current row counter
    for(int i = 0; i < n-1; ++i) { // Block with same "-1" column
        for(int j = i; j < n-1; ++j) { // Loop over block
            triplets.push_back(Triplet<double>(row, j, 1));
            if(i > 0) { // Remove first column
                triplets.push_back(Triplet<double>(row, i-1, -1));
            }
            row++; // Next row
        }
    }

    // Build matrix
    A.setFromTriplets(triplets.begin(), triplets.end());

    A.makeCompressed();
    return A;
}
```

(c) Give explicit formulas for the entries of the system matrix $\mathbf{M}$ of the *normal equations* corresponding to the system (1). [2 pts.]

Solution:

The entries of matrix $\mathbf{M} = \mathbf{A}^\top\mathbf{A}$ can be expressed as inner products of two different columns of $\mathbf{A}$:

$$(\mathbf{A}^\top\mathbf{A})_{i,j} = (\mathbf{A})_{:,i}^\top(\mathbf{A})_{:,j}.$$

Two columns of $\mathbf{A}$ have both non-zero entries, ±1 of opposite sign, only in a single position, hence $(\mathbf{M})_{i,j} = -1$ for $i \neq j$. The diagonal entries of $\mathbf{M}$ are the squares of the Euclidean norms of the columns of $\mathbf{A}$. Every column of $\mathbf{A}$ has exactly $n-1$ entries with value ±1, which means $(\mathbf{M})_{i,i} = n-1$.

(d) Show that the system matrix $\mathbf{M}$ obtained in the previous step can be written as a rank-1 perturbation of a diagonal matrix. [2 pts.]

Solution:

As

$$(\mathbf{M})_{i,j} = \begin{cases} -1 & \text{, if } i \neq j, \\ n-1 & \text{, if } i = j \end{cases} \quad , \quad 1 \leq i,j \leq n-1, \tag{3}$$

we have that

$$\mathbf{M} = n\mathbf{I}_{n-1} - \mathbf{1}\cdot\mathbf{1}^\top, \quad \mathbf{1} = [1,\dots,1]^\top \in \mathbb{R}^{n-1}. \tag{4}$$

The tensor product matrix $\mathbf{1}\cdot\mathbf{1}^\top$ has rank 1.

(e) Write an EIGEN based C++ implementation

```
VectorXd estimatePointsPositions(const MatrixXd& D);
```

which solves the linear least squares problem (1) using normal equations method. Here $\mathbf{D} \in \mathbb{R}^{n\times n}$

$$(\mathbf{D})_{i,j} = \begin{cases} d_{i,j} & \text{if } i > j, \\ 0 & \text{if } i = j, \\ -d_{i,j} & \text{if } i < j. \end{cases}$$

Use the observations from the previous step. [5 pts.]

Solution:

We apply the Sherman-Morrison-Woodbury formula to the normal equations

$$(n\mathbf{I}_{n-1} - \mathbf{1}\cdot\mathbf{1}^\top)\,\mathbf{x} = \mathbf{A}^\top\mathbf{d}.$$

This yields

$$\mathbf{x} = \tfrac{1}{n}\mathbf{b} + \frac{\tfrac{1}{n}\mathbf{1}\cdot\mathbf{1}^\top\mathbf{b}}{n - \mathbf{1}^\top\mathbf{1}} = \tfrac{1}{n}\left(\mathbf{b} + \mathbf{1}\cdot\mathbf{1}^\top\mathbf{b}\right), \quad \mathbf{b} := \mathbf{A}^\top\mathbf{d}. \tag{5}$$

Note that the entries of the vector $\mathbf{b} \in \mathbb{R}^{n-1}$ can be computed by summing the entries of the last $n-1$ rows of $\mathbf{D}$ (the intermediate points of the distances cancel each other out)

```
VectorXd estimatePointsPositions(const MatrixXd& D) {

    VectorXd x;

    // Vector of sum of columns of A
    ArrayXd b = D.rowwise().sum().tail(D.cols()-1);
    // Vector 1
    ArrayXd one = ArrayXd::Constant(D.cols()-1, 1);
    // Apply SMW formula
    x =  (b + one * b.sum()) / D.cols();

    return x;
}
```

(f) What is the asymptotic complexity of the function `estimatePointPositions` implemented in subproblem (e) for $n \to \infty$? [1 pt.]

Solution:

An implementation of (5) involves SAXPY operations and inner products for vectors of length $n-1$, all of which can be carried out with asymptotic complexity $O(n)$.

However, forming the vector $\mathbf{b}$ has to access all distances and involves computational cost $O(n^2)$, which dominates the total asymptotic complexity.

2. *Solving an eigenvalue problem with Newton method* [14 pts.]

Given a symmetric positive-definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, solving

$$\begin{aligned} \mathbf{F}(\mathbf{z}) = \mathbf{0}, &\quad \mathbf{z} = (\mathbf{x}, \lambda)^\top, \\ \text{for} \quad \mathbf{F}(\mathbf{z}) &:= \begin{pmatrix} \mathbf{A}\mathbf{x} - \lambda\mathbf{x} \\ 1 - \frac{1}{2}\|\mathbf{x}\|^2 \end{pmatrix}, \end{aligned} \tag{6}$$

is equivalent to finding an eigenvector $\mathbf{x}$ and associated eigenvalue λ for $\mathbf{A}$.

Therefore, a possible numerical method for computing one eigenvalue/eigenvector of $\mathbf{A}$ is the application of Newton's method to find a zero of the vector-valued function $\mathbf{F} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ defined in (6).

(a) Compute the Jacobian of $\mathbf{F}$ at $\begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} \in \mathbb{R}^{n+1}$. [2 pts.]

Solution:

$$\mathbf{DF}(\mathbf{x}) = \begin{pmatrix} \mathbf{A}\mathbf{x} - \lambda\mathbf{I} & -\mathbf{x} \\ -\mathbf{x}^\top & 0 \end{pmatrix}.$$

(b) Devise an iteration of the Newton method to solve $\mathbf{F}(\mathbf{z}) = \mathbf{0}$. [4 pts.]

Solution:

$$\begin{pmatrix} x^{(k+1)} \\ \lambda^{(k+1)} \end{pmatrix} = \begin{pmatrix} x^{(k)} \\ \lambda^{(k)} \end{pmatrix} - \begin{pmatrix} \mathbf{A} - \lambda\mathbf{I} & -\mathbf{x}^{(k)} \\ -(\mathbf{x}^{(k)})^\top & 0 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{A}\mathbf{x}^{(k)} - \lambda^{(k)}\mathbf{x}^{(k)} \\ 1 - \frac{1}{2}\left\|\mathbf{x}^{(k)}\right\|^2 \end{pmatrix}.$$

(c) Write an EIGEN based C++ implementation for the Newton method devised in the previous step:

```
void eigNewton(const MatrixXd &A, double atol, int
    maxItr, VectorXd &z);
```

which, given the matrix `A`, tolerance `tol` and initial guess `z`, returns the solution in `z`. [8 pts.]

Hint: For the initial guess: choose $\mathbf{x}$, then evaluate $\lambda = \dfrac{\mathbf{x}^\top\mathbf{A}\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}$.

Hint: Test your code with some small matrix $\mathbf{A}$.

Solution:

```
void eigNewton(const Eigen::MatrixXd &A, double tol, int maxItr,
    Eigen::VectorXd &z) {
    int m = z.size();
    int n = m - 1;

    Eigen::MatrixXd DF(m, m);
    Eigen::VectorXd F(m);
    Eigen::VectorXd F_old(m);

    for (int i = 0; i < maxItr; ++i) {
        Eigen::VectorXd x = z.head(n);
        F.head(n) = A * x - z(n) * x;
        F(n) = 1.0 - 0.5 * x.squaredNorm();

        if (F.squaredNorm() < tol) {
            std::cout << "tol reached with i = " << i << std::endl;
```

```
            return;
        }

        DF.topLeftCorner(n, n) = A - z(n) *
            Eigen::MatrixXd::Identity(n, n);
        DF.col(n) = -z;
        DF.row(n) = -z.transpose();
        DF(n, n) = 0;

        z += -DF.fullPivLu().solve(F);
    }

    std::cout << "maxItr reached" << std::endl;
}
```

3. *Gauss-Legendre quadrature rule* [14 pts.]

An n-point quadrature formula on $[a,b]$ provides an approximation of the value of an integral through a *weighted sum* of point values of the integrand:

$$\int_a^b f(x)\,\mathrm{d}t \;\approx\; Q_n(f) := \sum_{j=1}^{n} w_j^n\, f(c_j^n), \tag{7}$$

where w_j^n are called quadrature weights $\in \mathbb{R}$ and c_j^n quadrature nodes $\in [a,b]$.

The order of a quadrature rule $Q_n : C^0([a,b]) \to \mathbb{R}$ is defined as the maximal degree+1 of polynomials for which the quadrature rule is guaranteed to be exact. It can also be shown that the maximal order of an n-point quadrature rule is $2n$. So the natural question to ask is if such a family Q_n of n-point quadrature formulas exist where Q_n is of order $2n$. If yes, how do we find the nodes corresponding to it?

Let us assume that there exists a family of n-point quadrature formulas on $[-1,1]$ of order $2n$, i.e.

$$Q_n(f) := \sum_{j=1}^{n} w_j^n f(c_j^n) \approx \int_{-1}^{1} f(t)\,\mathrm{d}t\,, \quad w_j \in \mathbb{R}\,,\; n \in \mathbb{N}\,, \tag{8}$$

and the above approximation is exact for polynomials $\in \mathcal{P}_{2n-1}$.

Define the n-degree polynomial

$$\bar{P}_n(t) := (t - c_1^n)\cdot \dots \cdot (t - c_n^n)\,, \quad t \in \mathbb{R}\,.$$

If we are able to obtain $\bar{P}_n(t)$, we can compute its roots numerically to obtain the nodes for the quadrature formula.

(a) For every $q \in \mathcal{P}_{n-1}$, verify that $\bar{P}_n(t) \perp q$ in $L^2([-1,1])$ i.e.

$$\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0. \tag{9}$$

[2 pts.]

Solution:

$$\forall\, q \in \mathcal{P}_{n-1} : \quad q \cdot \bar{P}_n \in \mathcal{P}_{2n-1}$$

$$\Longrightarrow \underbrace{\int_{-1}^{1} q(t)\cdot \bar{P}_n(t)\,\mathrm{d}t}_{\langle q, \bar{P}_n\rangle_{L^2([-1,1])}} \underbrace{=}_{\text{exact QF on } \mathcal{P}_{2n-1}} \sum_{j=1}^{n} w_j^n q(c_j^n) \underbrace{\bar{P}_n(c_j^n)}_{=0,\ \forall j=(1,\dots,n)} = 0.$$

Thus, we have proved $\bar{P}_n \perp \mathcal{P}_{n-1}$ in $L^2([-1,1])$.

(b) Switching to a monomial representation of $\bar{P}_n$

$$\bar{P}_n = t^n + \alpha_{n-1}t^{n-1} + \dots + \alpha_1 t + \alpha_0\,,$$

derive

$$\sum_{j=0}^{n-1} \alpha_j \int_{-1}^{1} t^\ell t^j\,\mathrm{d}t = -\int_{-1}^{1} t^\ell t^n\,\mathrm{d}t \qquad \forall\, \ell = 0\,\dots, n-1. \tag{10}$$

[3 pts.]

Hint: Use (9) with the monomials $1, t, \ldots, t^{n-1}$ and with $\bar{P}_n$ in its monomial representation.

Solution:

We know that:

$$\int_{-1}^{1} q(t)\bar{P}_n(t)\,\mathrm{d}t = 0 \quad \forall q \in \mathcal{P}_{n-1}.$$

This yields n conditions:

$$\begin{aligned}
&\int_{-1}^{1} \bar{P}_n t^\ell \,\mathrm{d}t = 0 \qquad \forall\, \ell = 0 \ldots, n-1 \\
\Leftrightarrow &\int_{-1}^{1} t^\ell \underbrace{\left(t^n + \sum_{j=0}^{n-1} \alpha_j t^j\right)}_{\bar{P}_n} \mathrm{d}t = 0 \quad \forall \ell = 0, \ldots, n-1 \\
\implies &\sum_{j=0}^{n-1} \alpha_j \int_{-1}^{1} t^\ell t^j \,\mathrm{d}t = -\int_{-1}^{1} t^\ell t^n \,\mathrm{d}t.
\end{aligned}$$

(c) Find expressions for $\mathbf{A}$ and $\mathbf{b}$ such that the coefficients of the monomial expansion can be obtained by solving a linear system of equation $\mathbf{A}[\alpha_j]_{j=0}^{n-1} = \mathbf{b}$. [3 pts.]

Solution:

(10) can be rewritten as: $\mathbf{A}[\alpha_j]_{j=0}^{n-1} = \mathbf{b}$, where

$$\mathbf{A}_{j,\ell} = \int_{-1}^{1} t^\ell t^j \,\mathrm{d}t = \langle t^\ell, t^j \rangle_{L^2([-1,1])}.$$

and

$$\mathbf{b}_\ell = -\int_{-1}^{1} t^\ell t^n \,\mathrm{d}t = \langle t^\ell, t^n \rangle_{L^2([-1,1])}.$$

(d) Show that $[\alpha_j]_{j=0}^{n-1}$ exists and is unique. [3 pts.]
Hint: verify that $\mathbf{A}$ is symmetric positive definite.

Solution:

We can see that $\mathbf{A}$ is symmetric. Moreover,

$$\begin{aligned}
\mathbf{x}^\top \mathbf{A} \mathbf{x} &= \sum_{\ell=0}^{n-1} x_\ell \left(\sum_{j=0}^{n-1} \int_{-1}^{1} t^j t^\ell \,\mathrm{d}t\; x_j \right) \\
&= \int_{-1}^{1} \left(\sum_{\ell=0}^{n-1} x_\ell t^\ell \right) \left(\sum_{j=0}^{n-1} x_j t^j \right) \mathrm{d}t \\
&= \int_{-1}^{1} \left(\sum_{j=0}^{n-1} x_j t^j \right)^2 \mathrm{d}t > 0 \quad \text{if } x \neq 0.
\end{aligned}$$

Thus, $\mathbf{A}$ is symmetric positive definite $\implies$ $[\alpha_j]_{j=0}^{n-1}$ exists and is unique.

(e) Use a 5-point Gauss quadrature rule to compare the exact solution and the quadrature approximation of

$$\int_{-3}^{3} e^t \,\mathrm{d}t.$$

The polynomial obtained in (d) and the Legendre-polynomial P_n differ by a constant factor. Thus, the Gauss quadrature nodes $(\widehat{c_j})_{j=1}^{5}$ are also the zeros of the 5-th Legendre polynomial P_5. Here, we provide the zeros of P_5 for simplicity, but they should ideally be obtained by a numerical method for obtaining roots (e.g Newton-Raphson method). Thus,

$$(\widehat{c_j})_{j=1}^5 = [-0.9061798459, -0.5384693101, 0, 0.5384693101, 0.9061798459]$$

Recall from Theorem 6.3.1 (found in Week 9 Tablet notes - pg. 9) that the corresponding quadrature weights $\widehat{w}_j$ are given by:

$$\widehat{w}_j = \int_{-1}^{1} L_{j-1}(t)\,\mathrm{d}t, \quad j = 1, \ldots, n, \tag{11}$$

where L_j, $j = 0, \ldots, n-1$, is the j-th Lagrange polynomial associated with the ordered node set $\{\widehat{c_1}, \ldots, \widehat{c_n}\}$. [3 pts.]

Solution:

The j-th Lagrange polynomial can be obtained by:

$$L_j(t) = \prod_{k=0, k\neq j}^{n-1} \frac{t - t_k}{t_j - t_k}.$$

After obtaining the Lagrange polynomials for $j = 0, \ldots, n-1$ using the quadrature nodes $(\widehat{c_j})_{j=1}^5$, we can use (11) to obtain the quadrature weights. They are found to be:

$$(\widehat{w_j})_{j=1}^5 = [0.2369268851, 0.4786286705, 0.5688888889, 0.4786286705, 0.2369268851].$$

Note that we wish to use the quadrature formula on the interval $[-3, 3]$. However, our nodes and weights have been computed for the reference interval $[-1, 1]$. Thus, we need to perform an affine transformation

$$\Phi(\tau) = \frac{1}{2}(1-\tau)a + \frac{1}{2}(1+\tau)b\ .$$

This allows us to use the general quadrature formula with the transformed nodes and weights, i.e.

$$\int_a^b f(t)\,\mathrm{d}t \approx \sum_{j=1}^{n} w_j f(c_j)$$

with

$$c_j = \Phi(\widehat{c_j}) = \tfrac{1}{2}(1-\widehat{c_j})a + \tfrac{1}{2}(1+\widehat{c_j})b\ , \qquad w_j = \frac{|[a,b]|}{|[-1,1]|}\widehat{w}_j = \tfrac{1}{2}(b-a)\widehat{w}_j\ .$$

The solution obtained using the quadrature approximation $(\sum_{j=1}^{n} w_j e^{(c_j)}) = 20.0355777184$.

On the other hand, the exact solution is

$$\int_{-3}^{3} e^t\,\mathrm{d}t = e^3 - e^{-3} = 20.0357498548.$$