Alexander Mills (adm5547), Matt Skiles (ms82657), Aidan Pimentel (amp5496)

We use the following code to find solutions to parts 1 2 and 3 of the programming assignment. The primary routine to find the inverse of a matrix is labelled `matrixInversion`, and it accepts an mxn matrix where n >= m.

```
const math = require('mathjs');

const sizeOfMatrix = parseInt(process.env.MATRIX_SIZE);

if (!Number.isInteger(sizeOfMatrix)) {
    throw new Error('please specify size of matrix using env var MATRIX_SIZE=x')
}

const generateRandomMatrix = function (m, n, density, L, U) {

    const args = [...arguments]; // copy args in case we need call func recursively

    if (m < 1) {
        throw new Error('height of matrix has to be greater than zero')
    }
    if (n < 1) {
        throw new Error('width of matrix has to be greater than zero')
    }
    if (U < L) {
        throw new Error('Upper bound must be greater than or equal to lower bound')
    }
    if (density <= 0 || density >= 1) {
        throw new Error('density must be greater than 0 and less than 1.')
```

```
    }
    const matrix = [];

    const zeroCountCol = new Map();
    for (let i = 0; i < m; i++) {
        matrix.push([]);
        let zeroCountRow = 0;
        for (let j = 0; j < n; j++) {
            const col = zeroCountCol.has(j) ? zeroCountCol.get(j) : (zeroCountCol.set(j, 0), 0);
            const num = Math.random();
            if (num > density && zeroCountRow < n && col < m) {
                zeroCountCol.set(j, col + 1);
                zeroCountRow++;  // we ensure that we don't fill the whole row with zeros
                matrix[i][j] = 0;
                continue;
            }
            const el = math.round(num * (U - L) + L, 5);
            if (el < L) {
                throw new Error('element is less than lower bound')
            }
            if (el > U) {
                throw new Error('element is greater than upper bound')
            }
            matrix[i][j] = el;
        }
    }

    if (math.det(matrix) === 0) {
        // if determinant is 0 we cannot take inverse of matrix, so try again
        return generateRandomMatrix(...args);
    }
```

```
    return matrix;
}

const getInverseFromAugmented = (m) => {
    for (let i = 0; i < m.length; i++) {
        m[i] = m[i].slice(Math.ceil(m[i].length / 2))
    }
    return [...m];
}

const generateTestAugmentedMatrix = () => {
    const squareMatrix = generateTestSquareMatrix();
    return [
        [...squareMatrix[0], 5],
        [...squareMatrix[1], 3],
        [...squareMatrix[2], 3]
    ]

}

const combineMatrixWithBVector = (m, b) => {
    for (let i = 0; i < m.length; i++) {
        m[i].push(b[i]);
    }
    return m;
}

const combineMatrixWithIdentityImmutable = (m) => {
    const identityMatrix = generateIdentityMatrix(m.length);
    for (let i = 0; i < m.length; i++) {
        m[i] = [...m[i], ...identityMatrix[i]]
    }
```

```
    return [...m];
}

const generateTestSquareMatrix = () => {
    return [
        [0, 9, 5],
        [4, 0, 4],
        [0, 0, 1]
    ]

}


const generateIdentityMatrix = (m) => {
    const ret = [];
    for (let i = 0; i < m; i++) {
        ret[i] = []
        for (let j = 0; j < m; j++) {
            ret[i][j] = i === j ? 1 : 0;
        }
    }
    return ret;
}

const findRowToSwap = (i, j, m) => {
    for (; i < m.length; i++) {
        if (m[i][j] !== 0) {
            // we swap with first row that has a non-zero entry in the desired/current column
            return i;
        }
    }
    return -1;
```

```
}

const swapRows = (m, i, j) => {
    const row = m[i];
    m[i] = m[j];
    m[j] = row;
}

const logFatal = function () {
    console.error.apply(console, arguments);
    process.exit(1);
}

const divideRow = (row, v) => {
    for (let i = 0; i < row.length; i++) {
        row[i] = math.round(row[i] / v, 5);
    }
}

const divideRowImmutable = (row, v) => {
    return row.map(z => math.round(-1 * z * v, 5));
}

const addToRow = (rowToModify, rowToAdd) => {
    for (let i = 0; i < rowToModify.length; i++) {
        rowToModify[i] = math.round(rowToAdd[i] + rowToModify[i], 5);
    }
}

const generateRandomBVector = (n, L, U, density = 0.8) => {
    const ret = [];
    for (let i = 0; i < n; i++) {
```

```
        const num = Math.random();
        const val = num > density ? 0 : math.round(num * (U - L) + L, 5);
        ret.push(val);
    }
    return ret;
}

const copyMatrix = (m) => {
    for (let i = 0; i < m.length; i++) {
        m[i] = [...m[i]];
    }
    return [...m];
}

const matrixInversion = (z) => {

    // this routine uses the Gauss-Jordan elimination method

    const m = copyMatrix(z); // for immutability
    const height = m.length;

    for (let i = 0; i < height; i++) {

        let diagElem = m[i][i];

        if (diagElem === 0) {
            const row = findRowToSwap(i + 1, i, m);
            if (row < 0) {
                logFatal('row could not be found')
            }
            swapRows(m, row, i)
        }
```

```
        diagElem = m[i][i];

        if (diagElem !== 1) {
            divideRow(m[i], diagElem);
        }

        diagElem = m[i][i];

        if (diagElem !== 1) {
            logFatal('diagonal is not 1')
        }

        for (let z = 0; z < height; z++) {
            if (z === i) {
                continue;
            }
            const row = m[z];
            if (row[i] === 0) {
                continue;
            }

            const multipliedRow = divideRowImmutable(m[i], row[i])
            addToRow(row, multipliedRow);
        }

    }

    return m;
}
```

```
const randomMatrix = generateRandomMatrix(sizeOfMatrix, sizeOfMatrix, .6, -10, 30);
const randomMatrixCopy = copyMatrix(randomMatrix);
const randomBVector = generateRandomBVector(sizeOfMatrix, 0, 50, .8);
const randomBVectorCopy = [...randomBVector];


console.log(
    `Random input matrix (of size ${sizeOfMatrix}):`,
    randomMatrix
)

console.log(
    'Random B vector:',
    randomBVector
)

const preInverted = combineMatrixWithIdentityImmutable(
    randomMatrix
);

console.log(
    'Augmented matrix prior to inversion:',
    preInverted
)

const inverted = getInverseFromAugmented(
    matrixInversion(
        preInverted
    )
);

console.log(
```

```
    'Inverted matrix:',
    inverted
)

console.log(
    'X vector solution (inverted matrix multipled by B vector:)\n',
    math.multiply(
        inverted,
        randomBVector
    )
)

const combined = combineMatrixWithBVector(
    randomMatrixCopy,
    randomBVectorCopy,
);

console.log('Matrix augmented with B vector:', combined);

const invertedMatrix = matrixInversion(
    combined
);

console.log(
    'Inverted matrix:',
    invertedMatrix
);

console.log(
    'X vector solution:',
    invertedMatrix.reduce((a,b) => (a.push(b[b.length-1]),a), [])
)
```

The following is sample output for 3x3, 5x5, and 10x10 matrices:

3x3 matrix input:

```
Random input matrix (of size 3): [ [ 0, -9.97788, 0 ], [ 0, 0, 13.93958 ], [ 5.69314, 0, 2.44658 ] ]
Random B vector: [ 13.16818, 23.3486, 0 ]
Augmented matrix prior to inversion: [
  [ 0, -9.97788, 0, 1, 0, 0 ],
  [ 0, 0, 13.93958, 0, 1, 0 ],
  [ 5.69314, 0, 2.44658, 0, 0, 1 ]
]
Inverted matrix: [ [ 0, -0.03083, 0.17565 ], [ -0.10022, 0, 0 ], [ 0, 0.07174, 0 ] ]
X vector solution (inverted matrix multipled by B vector:)
 [ -0.719837338, -1.3197149996, 1.675028564 ]
Matrix augmented with B vector: [
  [ 0, -9.97788, 0, 13.16818 ],
  [ 0, 0, 13.93958, 23.3486 ],
  [ 5.69314, 0, 2.44658, 0 ]
]
Inverted matrix: [ [ 1, 0, 0, -0.71981 ], [ 0, 1, 0, -1.31974 ], [ 0, 0, 1, 1.67499 ] ]
X vector solution: [ -0.71981, -1.31974, 1.67499 ]
```

5x5 matrix input:

```
Random input matrix (of size 5): [
  [ 9.0092, 0, -7.23917, 0, 1.27238 ],
  [ 6.62358, 8.41519, 0, 0, 4.3261 ],
  [ 8.6027, 12.6623, -6.1308, 0, 0 ],
```

```
  [ -0.58481, -2.75118, 3.87744, 0, 0 ],
  [ 0, 0, -8.64999, 3.85385, 0 ]
]
Random B vector: [ 3.50294, 9.67247, 6.17081, 12.29505, 6.90732 ]
Augmented matrix prior to inversion: [
  [ 9.0092, 0, -7.23917, 0, 1.27238, 1, 0, 0, 0, 0 ],
  [
    6.62358, 8.41519, 0,
          0,  4.3261, 0,
          1,       0, 0,
          0
  ],
  [
    8.6027, 12.6623, -6.1308,
         0,       0,       0,
         0,       1,       0,
         0
  ],
  [ -0.58481, -2.75118, 3.87744, 0, 0, 0, 0, 0, 1, 0 ],
  [ 0, 0, -8.64999, 3.85385, 0, 0, 0, 0, 0, 1 ]
]
Inverted matrix: [
  [ 0.07693, -0.02263, 0.07045, 0.25499, 0 ],
  [ -0.07106, 0.0209, 0.05524, -0.04531, 0 ],
  [ -0.03882, 0.01142, 0.04982, 0.26421, 0 ],
  [ -0.08712, 0.02562, 0.11182, 0.59301, 0.25948 ],
  [ 0.02044, 0.22514, -0.21531, -0.30226, 0 ]
]
X vector solution (inverted matrix multipled by B vector:)
 [
  3.6204415420999996,
  -0.26297746450000004,
```

```
  3.5303803913,
  9.7160515169,
  -2.7956789246999993
]
Matrix augmented with B vector: [
  [ 9.0092, 0, -7.23917, 0, 1.27238, 3.50294 ],
  [ 6.62358, 8.41519, 0, 0, 4.3261, 9.67247 ],
  [ 8.6027, 12.6623, -6.1308, 0, 0, 6.17081 ],
  [ -0.58481, -2.75118, 3.87744, 0, 0, 12.29505 ],
  [ 0, 0, -8.64999, 3.85385, 0, 6.90732 ]
]
Inverted matrix: [
  [ 1, 0, 0, 0, 0, 3.6204 ],
  [ 0, 1, 0, 0, 0, -0.263 ],
  [ 0, 0, 1, 0, 0, 3.53035 ],
  [ 0, 0, 0, 1, 0, 9.7162 ],
  [ 0, 0, 0, 0, 1, -2.79564 ]
]
X vector solution: [ 3.6204, -0.263, 3.53035, 9.7162, -2.79564 ]
```

10x10 matrix input:

```
Random input matrix (of size 10): [
  [
         0,        0,
    4.24253, -9.06306,
         0,        0,
         0,  5.23088,
    6.86058, -0.90789
  ],
  [
    -7.32223,        0,
```

```
  -6.12315, -0.68619,
         0,        0,
         0, -1.05449,
         0,  1.63538
],
[
   2.91393,        0,
  10.90349, -1.66946,
    2.3359,        0,
         0, 12.61777,
   7.27842,        0
],
[
         0,        0,
  -1.18768,        0,
         0,  8.72263,
   5.90783,        0,
         0, -2.68813
],
[
   3.16276,  10.7212,
         0, -2.52962,
   7.87232,        0,
         0,        0,
  12.78313,        0
],
[
         0, -6.82854,
         0,        0,
  -0.93116,        0,
         0,        0,
   6.76933, -7.27744
```

```
  ],
  [
    -9.13999,  4.23718,
     0.37459,  9.89809,
       8.828,        0,
    -2.84993, 11.13247,
     7.44818,        0
  ],
  [
    -5.67275,       0,
    -6.17214,       0,
    12.53056,       0,
           0, 9.01606,
           0,   0.199
  ],
  [
           0, -5.46441,
    -7.71459,        0,
           0,        0,
    -3.49711, 12.27829,
     7.70777,  9.29821
  ],
  [
         0, -3.61127,
    7.17544, -8.70604,
    1.37324, -9.34622,
          0, -7.84834,
    3.32823, -7.93525
  ]
]
Random B vector: [
  38.81684, 31.50239,
```

```
   15.3752, 38.74558,
  14.16021, 32.25886,
         0, 25.72999,
  32.63388, 28.07273
]
Augmented matrix prior to inversion: [
  [
          0,       0, 4.24253,
    -9.06306,       0,       0,
          0, 5.23088, 6.86058,
    -0.90789,       1,       0,
          0,       0,       0,
          0,       0,       0,
          0,       0
  ],
  [
    -7.32223,        0, -6.12315,
    -0.68619,        0,        0,
          0, -1.05449,        0,
     1.63538,        0,        1,
          0,        0,        0,
          0,        0,        0,
          0,        0
  ],
  [
     2.91393,        0, 10.90349,
    -1.66946,   2.3359,        0,
          0, 12.61777,  7.27842,
          0,        0,        0,
          1,        0,        0,
          0,        0,        0,
          0,        0
```

```
],
[
        0, 0, -1.18768,
        0, 0,  8.72263,
   5.90783, 0,        0,
  -2.68813, 0,        0,
        0, 1,        0,
        0, 0,        0,
        0, 0
],
[
   3.16276, 10.7212,        0,
  -2.52962, 7.87232,        0,
        0,       0, 12.78313,
        0,       0,        0,
        0,       0,        1,
        0,       0,        0,
        0,       0
],
[
        0, -6.82854,       0,
        0, -0.93116,       0,
        0,        0, 6.76933,
  -7.27744,       0,       0,
        0,        0,       0,
        1,        0,       0,
        0,        0
],
[
  -9.13999,  4.23718, 0.37459,
   9.89809,    8.828,       0,
  -2.84993, 11.13247, 7.44818,
```

```
        0,        0,       0,
        0,        0,       0,
        0,        1,       0,
        0,        0
],
[
  -5.67275,        0, -6.17214,
        0, 12.53056,        0,
        0,  9.01606,        0,
     0.199,        0,        0,
        0,        0,        0,
        0,        0,        1,
        0,        0
],
[
        0, -5.46441, -7.71459,
        0,        0,        0,
  -3.49711, 12.27829,  7.70777,
   9.29821,        0,        0,
        0,        0,        0,
        0,        0,        0,
        1,        0
],
[
        0, -3.61127,  7.17544,
  -8.70604,  1.37324, -9.34622,
        0, -7.84834,  3.32823,
  -7.93525,        0,        0,
        0,        0,        0,
        0,        0,        0,
        0,        1
]
```

```
]
Inverted matrix: [
  [
    -0.10917,  0.07026,
     0.12688, -0.09204,
     0.06251,  0.09841,
    -0.13327,  0.04768,
    -0.04688, -0.08591
  ],
  [
    -0.19429,  0.46973,
     0.36257, -0.25464,
     0.13429,  0.20473,
    -0.25893,  0.07166,
    -0.21907, -0.23766
  ],
  [
     0.23351, -0.45775,
      -0.331,  0.23137,
    -0.12069, -0.25333,
     0.28537, -0.10599,
     0.15834,  0.21599
  ],
  [
    -0.15176,  0.09367,
     0.10502, -0.02192,
     0.02573,  0.04889,
    -0.02515, -0.01215,
    -0.01657, -0.02047
  ],
  [
     0.24135, -0.56309,
```

```
  -0.43005,   0.2628,
  -0.10275, -0.26585,
   0.29893, -0.03267,
   0.20032,   0.2453
],
[
   0.50578,  -0.8087,
  -0.67621,  0.35771,
  -0.18114, -0.31998,
   0.44628, -0.12312,
   0.24057,  0.22688
],
[
  -0.62612,  0.91472,
   0.78097, -0.18927,
   0.19966,  0.27746,
  -0.48252,  0.10884,
  -0.21241, -0.17666
],
[
  -0.24785,  0.52253,
   0.45825, -0.27072,
   0.10162,  0.26495,
  -0.30973,  0.11627,
   -0.2049, -0.25269
],
[
    0.0113, -0.04606,
  -0.04986,  0.07017,
   0.01849, -0.02267,
   0.06108, -0.05419,
    0.0687,  0.06547
```

```
  ],
  [
     0.16199, -0.41155,
    -0.33163,  0.27052,
    -0.09565, -0.31656,
     0.26154, -0.11347,
     0.24384,  0.25252
  ]
]
X vector solution (inverted matrix multipled by B vector:)
 [
  -2.2946637743999996,
  -0.5068214274000002,
  -2.858301946400002,
  -1.6611480611999996,
  -2.2481215582999994,
  -4.215557364999996,
  11.8731923406,
  2.5938633139000036,
  3.1558750429,
  -0.7337916619000033
]
Matrix augmented with B vector: [
  [
           0,        0,
     4.24253, -9.06306,
           0,        0,
           0,  5.23088,
     6.86058, -0.90789,
    38.81684
  ],
  [
```

```
  -7.32223,        0,
  -6.12315, -0.68619,
         0,        0,
         0, -1.05449,
         0,  1.63538,
  31.50239
],
[
   2.91393,        0,
  10.90349, -1.66946,
    2.3359,        0,
         0, 12.61777,
   7.27842,        0,
   15.3752
],
[
         0,        0,
  -1.18768,        0,
         0,  8.72263,
   5.90783,        0,
         0, -2.68813,
  38.74558
],
[
   3.16276,  10.7212,
         0, -2.52962,
   7.87232,        0,
         0,        0,
  12.78313,        0,
  14.16021
],
[
```

```
        0, -6.82854,
        0,        0,
  -0.93116,       0,
        0,        0,
   6.76933, -7.27744,
  32.25886
],
[
  -9.13999,  4.23718,
   0.37459,  9.89809,
     8.828,        0,
  -2.84993, 11.13247,
   7.44818,        0,
        0
],
[
  -5.67275,       0,
  -6.17214,       0,
  12.53056,       0,
        0, 9.01606,
        0,   0.199,
  25.72999
],
[
        0, -5.46441,
  -7.71459,       0,
        0,        0,
  -3.49711, 12.27829,
   7.70777,  9.29821,
  32.63388
],
[
```

```
          0, -3.61127,
     7.17544, -8.70604,
     1.37324, -9.34622,
          0, -7.84834,
     3.32823, -7.93525,
    28.07273
  ]
]
Inverted matrix: [
  [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -2.29317 ],
  [ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, -0.50388 ],
  [ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, -2.86077 ],
  [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1.66094 ],
  [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, -2.2509 ],
  [ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, -4.22029 ],
  [ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 11.87878 ],
  [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2.59717 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3.15512 ],
  [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -0.73732 ]
]
X vector solution: [
  -2.29317, -0.50388,
  -2.86077, -1.66094,
   -2.2509, -4.22029,
  11.87878,  2.59717,
   3.15512, -0.73732
]
```