

NUMERICAL METHODS

ECSE 543 - ASSIGNMENT 1

MIDO ASSRAN - 260505216

QUESTION 1

Part a. The Choleski implementation is provided in **Listing 2**.

Code structure. To maintain portability and modularity of the code, object oriented principles were used for the software architecture. The choleski implementation is included in the *CholeskiDecomposition()* class. The *solve(A, b)* method solves the linear system of equations shown in equation (1) by performing choleski elimination.

$$(1) \quad Ax = b$$

The method accepts the matrix A , and the vector b (both of which will eventually be overwritten by the algorithm in order to conserve memory resources), and returns the vector x corresponding to the solution of equation (1). The algorithm works in two stages. The first stage performs a choleski factorization of A into LL^T (overwriting the lower triangular part of A by L), while simultaneously solving lower triangular system $Ly = b$ using forward substitution (overwriting b with the solution y). At the end of this stage, the program state now contains L in the lower triangular half of the matrix A , and the solution to $Ly = b$ in the vector b . In the second stage the program solves the system $L^T x = y$ using backwards substitution (overwriting y again with the solution x), where y is the solution to the system solved in the first stage. The program subsequently returns the vector x , which is the solution to equation (1).

Part b. For testing purposes, it was necessary to create a symmetric positive definite matrix. Such a matrix was created using the *generate_positive_semidef($order, seed$)* method contained in the *utils* file in **Listing 1**. Given an order (the dimension of the desired matrix), and an integer valued seed (used to seed the random number generator with a standard normal distribution), the function creates a random matrix, multiplies it by its transpose, and returns the result. The mathematical proof for why such a matrix is symmetric positive definite is well established. Whether or not the matrix is singular in this semidefinite method is important, and this is being checked by comparing the rank of the matrix to its order. If the rank of the matrix is not equal to the order of the matrix, then the matrix is singular and a warning is printed to the console. Note that this check still does not prevent the matrix from having a poor condition number.

Date: October 16, 2016.

Part c. The testing of the choleski implementation was conducted using the code provided under the *main()* method in **Listing 2** lines 90 – 111. The vector x^* , corresponding to the variable x in equation (1), is randomly generated with a standard normal distribution, and subsequently multiplied by the matrix A in order to generate a third vector b (i.e $b = A \cdot x^*$). The matrix A and the vector b are subsequently supplied to the solver, and the result is compared with the vector x^* that was originally used to create b . A sample of the console output is provided in Figure 1 - the matrix A is of order 10 in this example. The error in the produced result is quantified using the 2-norm:

$$\text{error} = \| \text{solve}(A, b) - x^* \|_2$$

As is seen in the console output, the error is only $2 \cdot 10^{-13}$, indeed the algorithm is producing the correct result. A possible reason for such a value of the error could be the roundoff error related to the condition of the randomly generated matrices.

FIGURE 1. Choleski Elimination Testing

```

Assignment_1 -- bash -- 136x41
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python choleski.py

# ----- TEST ----- #
# ----- Choleski Decomposition ----- #
# ----- #

A:
[[ 10.11336322 -5.41531694 -0.72864663 -0.54291123  0.89407964 -1.85960191 -2.64327259  0.15300401  5.07992412  2.45200326]
 [-5.41531694 12.69424122 -1.43458604  2.88235932  1.96113485  1.64030737  5.01771023  5.30699955 -5.3261667  -3.04390599]
 [-0.72864663 -1.43458604  4.28163672 -0.70195516  1.55763716 -0.73451178  0.73154923 -2.89174514 -1.44378971 -0.26001702]
 [-0.54291123  2.88235932 -0.70195516  5.36973562  1.0878062  4.50989735  0.71655986 -3.14179264  0.09320816  1.52721415]
 [ 0.89407964  1.96113485  1.55763716  1.0878062  3.86593736  1.17975588  1.72206133 -1.12802922  0.60412141 -0.46750886]
 [-1.85960191  1.64030737 -0.73451178  4.50989735  1.17975588 11.46766375 -1.90328549 -6.31715469  1.151696  2.67943195]
 [-2.64327259  5.01771023  0.73154923  0.71655986  1.72206133 -1.90328549 12.41522537  4.11954942 -0.015122 -2.23271915]
 [ 0.15300401  5.30699955 -2.89174514 -3.14179264 -1.12802922 -6.31715469  4.11954942 13.92141859 -4.97851072 -3.88234439]
 [ 5.07992412 -5.3261667 -1.44378971  0.09320816  0.60412141  1.151696 -0.015122 -4.97851072  9.44116694  1.53376537]
 [ 2.45200326 -3.04390599 -0.26001702  1.52721415 -0.46750886  2.67943195 -2.23271915 -3.88234439  1.53376537  3.93945221]]

x:
[ 0.79242262  0.17076445 -1.75374086  0.63029648  0.49832921  1.01813761 -0.84646862  2.52080763 -1.23238611  0.72695326]

b (=Ax):
[ 4.7223966  18.33819524 -15.06379199  2.28028854 -3.26059057  1.41602519 -4.87765149  32.04405234 -15.87690987 -1.58622103]

Execution time:
0.0002442909753881395

result = solve(A, b):
[ 0.79242262  0.17076445 -1.75374086  0.63029648  0.49832921  1.01813761 -0.84646862  2.52080763 -1.23238611  0.72695326]

2-norm error:
2.97842536819e-13

# ----- #

midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ _

```

Part d. A program used to solved tor the node voltages in a linear resistive network is provided in **Listing 3**. The *LinearResistiveNetworkSolver()* class is initialized with a filename from which to read the circuit description. The program, in the intializer, reads

a list of network branches (J_k, R_k, E_k) and a reduced incidence matrix from a CSV file. The format of the file is as follows: a set of rows (corresponding to each branch in the network), containing the comma separated branch current, resistance, and voltage in that respective order. Then a period is printed on a new line, to signify the end of the network data. The subsequent comma separated rows denote the incidence matrix, where each row corresponds to a node, and each column to a branch. An entry of -1 is used to indicate current entering a branch, 1 is used to indicate current leaving a branch, and 0 is used to indicate that the branch does not interact directly with the given node. The program reads the data in the file sequentially (i.e first the rows of the branch data are read, and then the rows of the incidence matrix are read). Once the data is read, the program subsequently generates a linear system of equations using the aforementioned data, and solves the system via choleski elimination.

Test Circuits. Test circuit CSV descriptions (used to test the program), and their equivalent circuit diagrams and corresponding console outputs are shown below. In each case, the console output was consistent with the analytical results obtained by hand.

Test Circuit 1

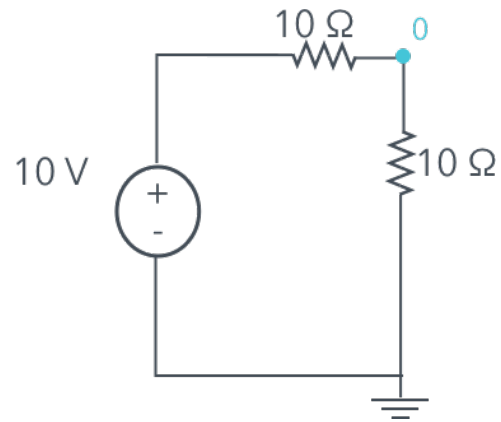
test.c1.csv

0, 10, 10

0, 10, 0

.

-1, +1



```
Assignment_1 — -bash — 96x20
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python lrn_solver.py
# ----- TEST ----- #
# ----- Linear Resistive Network Solver ----- #
# ----- Manual CSV Data ----- #
# ----- #
Execution time:
2.152204979211092e-05
Voltages:
Node 0: 5.0 Volts
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ _
```

Test Circuit 2

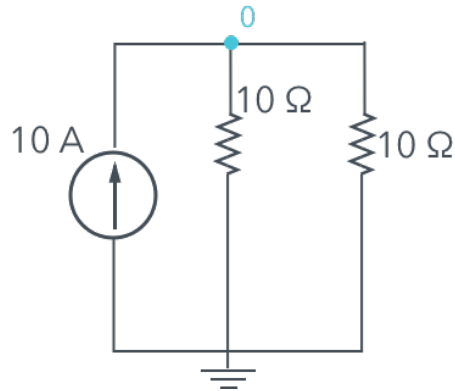
test_c2.csv

-10, 10, 0

0, 10, 0

.

-1, 1



```
Assignment_1 — -bash — 96x20
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python lrn_solver.py

# ----- TEST ----- #
# ----- Linear Resistive Network Solver ----- #
# ----- Manual CSV Data ----- #
# ----- #

Execution time:
4.49499930255115e-05

Voltages:
Node 0: 50.0 Volts

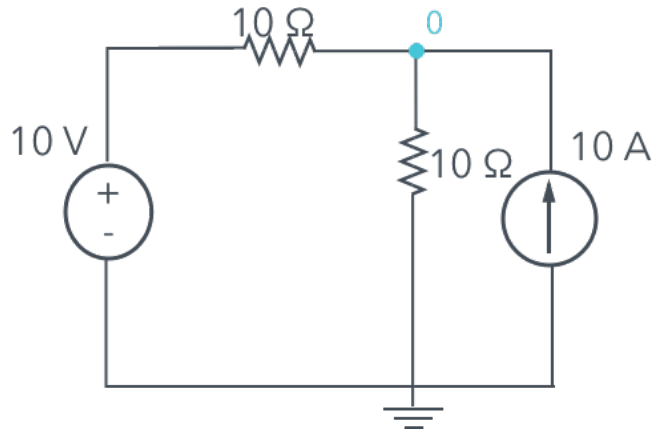
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ _
```

*Test Circuit 3***test_c3.csv**

```

0, 10, 10
-10, 10, 0
.
-1, -1

```



```

Assignment_1 — -bash — 96x20
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python lrn_solver.py

# ----- TEST ----- #
# ----- Linear Resistive Network Solver ----- #
# ----- Manual CSV Data ----- #
# ----- #

Execution time:
2.130295615643263e-05

Voltages:
Node 0: 55.0 Volts

midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ _

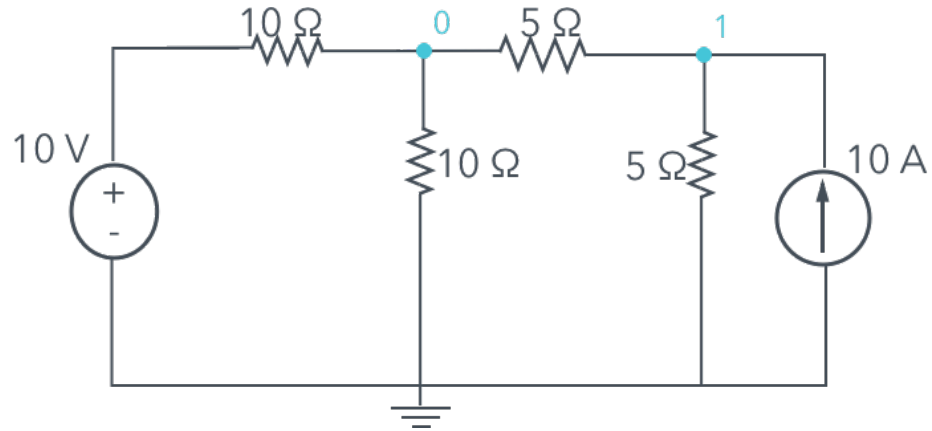
```

*Test Circuit 4***test_c4.csv**

```

0, 20, 10
0, 10, 0
0, 10, 0
0, 30, 0
0, 30, 0
0, 30, 0
.
-1, +1, +1, 0, 0, 0
0, -1, 0, +1, +1, 0
0, 0, -1, -1, 0, +1

```



```

Assignment_1 — -bash — 96x20
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python lrn_solver.py

# ----- TEST ----- #
# ----- Linear Resistive Network Solver ----- #
# ----- Manual CSV Data ----- #
# ----- #

Execution time:
2.855702769011259e-05

Voltages:
Node 0: 20.0 Volts
Node 1: 35.0 Volts

midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ 

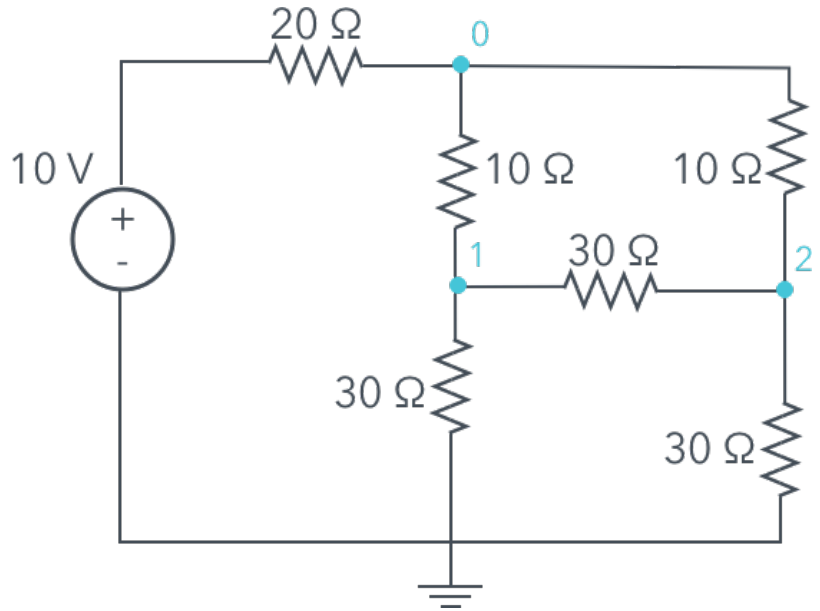
```

*Test Circuit 5***test_c5.csv**

```

0, 20, 10
0, 10, 0
0, 10, 0
0, 30, 0
0, 30, 0
0, 30, 0
.
-1, +1, +1, 0, 0, 0
0, -1, 0, +1, +1, 0
0, 0, -1, -1, 0, +1

```



```

Assignment_1 — -bash — 96x20
midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ python lrn_solver.py

# ----- TEST ----- #
# ----- Linear Resistive Network Solver ----- #
# ----- Manual CSV Data ----- #
# ----- #

Execution time:
3.575999289751053e-05

Voltages:
Node 0: 5.0 Volts
Node 1: 3.75 Volts
Node 2: 3.75 Volts

midoassran@Midos-MacBook-Pro:~/documents/McGill/U(4)/ECSE 543/Assignment_1$ _

```


QUESTION 2

Part a. To find the resistance across two diagonally opposing corners of a linear resistive N by N finite difference mesh, the linear resistive network solver, provided in **Listing 3**, was used. This is the same program that was used in Question 1. The static method `create_lrn_mesh_data(N , $fname$)` accepts an integer, N , denoting the size of the mesh, and a filename, to which a CSV description of the created mesh should be saved. It should be noted that this method also includes in the circuit description a test source placed across the diagonal of the mesh. This test source has a voltage of $1V$, and an output resistance of 1Ω . The `main()` method in **Listing 3** - lines 166-177 - calls the appropriate methods to create the resistive finite difference mesh, and subsequently solve for all the node voltages. Once all the node voltages are known, the voltage difference between the two corners of the mesh is used to construct a simple voltage division equation that is used to solve for the equivalent resistance of the mesh.

```

1 # ----- #
2 # Utils
3 # ----- #
4 # Author: Mido Assran
5 # Date: 5, October, 2016
6 # Description: Utils provides a cornucopia of useful matrix
7 # and vector helper functions.
8
9 import random
10 import numpy as np
11
12 def matrix_transpose(A):
13     """
14     :type A: np.array([float])
15     :rtype: np.array([floats])
16     """
17
18     # Initialize A_T(ranspose)
19     A_T = np.empty([A.shape[1], A.shape[0]])
20
21     # Set the rows of A to be the columns of A_T
22     for i, row in enumerate(A):
23         A_T[:, i] = row
24
25     return A_T
26
27 def matrix_dot_matrix(A, B):
28     """
29     :type A: np.array([float])
30     :type B: np.array([float])
31     :rtype: np.array([float])
32     """
33
34     # If matrix shapes are not compatible return None
35     if (A.shape[1] != B.shape[0]):
36         return None
37

```

```
39 A_dot_B = np.empty([A.shape[0], B.shape[1]])
   A_dot_B[:] = 0 # Initialize entries of the new matrix to zero
41
43 B_T = matrix_transpose(B)
45
47 for i, row_A in enumerate(A):
   for j, column_B in enumerate(B_T):
       for k, v in enumerate(row_A):
           A_dot_B[i, j] += v * column_B[k]
49
   return A_dot_B
51
def matrix_dot_vector(A, b):
53     """
54     :type A: np.array([float])
55     :type b: np.array([float])
56     :rtype: np.array([float])
57     """
59     # If matrix shapes are not compatible return None
60     if (A.shape[1] != b.shape[0]):
61         return None
63
64     A_dot_b = np.empty([A.shape[0]])
65     A_dot_b[:] = 0 # Initialize entries of the new vector to zero
67
68     for i, row_A in enumerate(A):
69         for j, val_b in enumerate(b):
70             A_dot_b[i] += row_A[j] * val_b
71
72     return A_dot_b
73
def vector_to_diag(b):
74     """
75     :type b: np.array([float])
76     :rtype: np.array([float])
77     """
```

```
79     diag_b = np.empty([b.shape[0], b.shape[0]])
80     diag_b[:] = 0      # Initialize the entries to zero
81
82     for i, val in enumerate(b):
83         diag_b[i, i] = val
84
85     return diag_b
86
87 def generate_positive_semidef(order, seed=0):
88     """
89     :type order: int
90     :type seed: int
91     :rtype: np.array([float])
92     """
93
94     np.random.seed(seed)
95     A = np.random.randn(order, order)
96     A = matrix_dot_matrix(A, matrix_transpose(A))
97
98     # TODO: Replace matrix_rank with a custom function
99     from numpy.linalg import matrix_rank
100     if matrix_rank(A) != order:
101         print("WARNING: Matrix is singular!", end="\n\n")
102
103     return A
```

Listing 1 . utils.py

```
1 # ----- #
2 # Choleski Decomposition
3 # ----- #
4 # Author: Mido Assran
5 # Date: 30, September, 2016
6 # Description: CholeskiDecomposition solves the linear system of equations:
7 #  $Ax = b$  by decomposing matrix A using Choleski factorization and using
8 # forward and backward substitution to determine x. Matrix A must
9 # be symmetric, real, and positive definite.
10
11 import random
12 import timeit
13 import numpy as np
14 from utils import matrix_transpose
15
16 DEBUG = True
17
18 class CholeskiDecomposition(object):
19
20     def __init__(self):
21         if DEBUG:
22             np.core.arrayprint._line_width = 200
23
24     def solve(self, A, b):
25         """
26         :type A: np.array([float])
27         :type b: np.array([float])
28         :rtype: np.array([float])
29         """
30
31         start_time = timeit.default_timer()
32
33         # If the matrix, A, is not square, exit
34         if A.shape[0] != A.shape[1]:
35             return None
36
37         n = A.shape[1]
```

```

39 # ----- #
41 # Simultaneous Choleski factorization of A and chol-elimination
42 # ----- #
43 # Choleski factorization & forward substitution
44 for j in range(n):
45
46     # If the matrix A is not positive definite , exit
47     if A[j,j] <= 0:
48         return None
49
50     A[j,j] = A[j,j] ** 0.5      # Compute the j,j entry of chol(A)
51     b[j] /= A[j,j]            # Compute the j entry of forward-sub
52
53     for i in range(j+1, n):
54
55         A[i,j] /= A[j,j]      # Compute the i,j entry of chol(A)
56         b[i] -= A[i,j] * b[j] # Look ahead modification of b
57
58         if A[i,j] == 0:      # Optimization for matrix sparsity
59             continue
60
61         # Look ahead modification of A
62         for k in range(j+1, i+1):
63             A[i,k] -= A[i,j] * A[k,j]
64
65 # ----- #
66
67 # ----- #
68 # Now solve the upper traingular system
69 # ----- #
70 # Transpose(A) is the upper-tiangular matrix of chol(A)
71 A[:] = matrix_transpose(A)
72
73 # Backward substitution
74 for j in range(n - 1, -1, -1):
75     b[j] /= A[j,j]
76
77

```

```

79         for i in range(j):
80             b[i] -= A[i,j] * b[j]
81         # ----- #
82
83         elapsed_time = timeit.default_timer() - start_time
84
85         if DEBUG:
86             print("Execution time:\n", elapsed_time, end="\n\n")
87
88         # The solution was overwritten in the vector b
89         return b
90
91 if __name__ == "__main__":
92     from utils import generate_positive_semidef, matrix_dot_vector
93
94     order = 10
95     seed = 5
96
97     print("\n", end="\n")
98     print("# ----- TEST ----- #", end="\n")
99     print("# ----- Choleski Decomposition ----- #", end="\n")
100    print("# ----- #", end="\n\n")
101    chol_d = CholeskiDecomposition()
102    # Create a symmetric, real, positive definite matrix.
103    A = generate_positive_semidef(order=order, seed=seed)
104    x = np.random.randn(order)
105    b = matrix_dot_vector(A=A, b=x)
106    print("A:\n", A, end="\n\n")
107    print("x:\n", x, end="\n\n")
108    print("b (=Ax):\n", b, end="\n\n")
109    v = chol_d.solve(A=A, b=b)
110    print("result = solve(A, b):\n", v, end="\n\n")
111    print("2-norm error:\n", np.linalg.norm(v - x), end="\n\n")
112    print("# ----- #", end="\n\n")

```

Listing 2 . choleski.py

```

1 # ----- #
  # Linear Resistive Network Solver
3 # ----- #
  # Author: Mido Assran
5 # Date: 30, September, 2016
  # Description: LinearResistiveNetworkSolver reads a CSV description of
7 # a linear resistive network, and determines all the node voltages
  # of the circuit by constructing a linear system of equations,
9 # and solving the system using Choleski Decomposition.

11 import random
   import csv
13 import numpy as np
   from choleski import CholeskiDecomposition
15 from utils import matrix_transpose, matrix_dot_matrix, matrix_dot_vector, vector_to_diag

17 DEBUG = False

19 class LinearResistiveNetworkSolver(object):

21     #-----Instance Variables-----#
     # _A -> The matrix 'A' in the system of equations Ax = b
23     # _b -> The vector 'b' in the system of equations Ax = _b

25     def __init__(self, fname):
        """
27         :type fname: String
         :rtype: void
29         """
        if DEBUG:
31             np.core.arrayprint._line_width = 200

33         #-----Load data from file-----#
         # Program first reads branch data, then switches to reading the
35         # incidence matrix. Flag goes high when the the program
         # switches to reading the incidence matrix.
37         flag = False
         network_branches = []

```



```

39 incidence_matrix = []
reader = csv.reader(open(fname, 'r'))
41 for row in reader:
    if len(row) == 1 and row[0] == ".":
43         flag = True
        continue
    elif len(row) == 0:
45         continue
    if not flag:
47         network_branches += [list(row)]
    else:
49         incidence_matrix += [list(row)]
51 network_branches = np.array(network_branches, dtype=np.float64)
incidence_matrix = np.array(incidence_matrix, dtype=np.float64)
53 J = network_branches[:, 0]
Y = vector_to_diag(1 / network_branches[:, 1])
55 E = network_branches[:, 2]
A = matrix_dot_matrix(A=matrix_dot_matrix(A=incidence_matrix, B=Y),
57                     B=matrix_transpose(incidence_matrix))
b = matrix_dot_vector(A=incidence_matrix,
59                     b=(J - matrix_dot_vector(A=Y, b=E)))

self._A = A
61 self._b = b

63 def solve(self):
    """
65     :rtype: numpy.array([float64])
    """
    chol_decomp = CholeskiDecomposition()
    # Choleski decomposition will overwrite A, and b
67     return chol_decomp.solve(A=self._A, b=self._b)

71
@staticmethod
73 def create_lrn_mesh_data(N, fname):
    """
75     :type N: int
    :type fname: String
77     :rtype: void

```

```

79     """
80     num_nodes = (N + 1) ** 2
81     num_branches = 2 * (N ** 2) + 2 * N + 1
82     incidence_matrix = np.empty([num_nodes, num_branches])
83     network_branches = np.empty([num_branches, 3])
84     incidence_matrix[:] = 0
85     network_branches[:] = 0
86
87     for i, row in enumerate(network_branches):
88         if i == (num_branches - 1):
89             network_branches[i, :] = np.array([0, 1, 1])
90         else:
91             network_branches[i, :] = np.array([0, 1e3, 0])
92
93     node_num = 0
94
95     # Iterate through node rows of mesh
96     for level in range(N + 1):
97
98         # Iterate through node columns of mesh
99         for column in range(N + 1):
100
101             # If the node has a left branch
102             if (node_num % (N + 1) != 0):
103                 left_branch = node_num + (level * N) - 1
104                 incidence_matrix[node_num, left_branch] = -1
105                 if DEBUG:
106                     print("L:", node_num, left_branch, end="\t")
107
108             # If the node has a right branch
109             if ((node_num + 1) % (N + 1) != 0):
110                 right_branch = node_num + (level * N)
111                 incidence_matrix[node_num, right_branch] = 1
112                 if DEBUG:
113                     print("R:", node_num, right_branch, end="\t")
114
115             # If the node has a top branch
116             if (node_num < (num_nodes - (N + 1))):
117                 top_branch = node_num + ((level + 1) * N)

```

```

117         incidence_matrix[node_num, top_branch] = 1
118         if DEBUG:
119             print("T:", node_num, top_branch, end="\t")
120
121         # If the node has a botom branch
122         if (node_num > N):
123             bottom_branch = (node_num - 1) + ((level - 1) * N)
124             incidence_matrix[node_num, bottom_branch] = -1
125             if DEBUG:
126                 print("B:", node_num, bottom_branch, end="\t")
127
128         if DEBUG:
129             print("\n")
130
131         node_num += 1
132
133         # Add the branch of the test source
134         incidence_matrix[0, -1] = -1
135         incidence_matrix[-1, -1] = 1
136
137         # Write data to file fname.csv
138         fwriter = csv.writer(open(fname, 'w'))
139         for i, row in enumerate(network_branches):
140             fwriter.writerow(row)
141
142         # Write a period to separate network_branches from
143         # the incidence_matrix
144         fwriter.writerow(".")
145
146         for i, row in enumerate(incidence_matrix):
147             fwriter.writerow(row)
148
149
150
151 if __name__ == "__main__":
152     # print("\n", end="\n")
153     # print("# _____ TEST _____ #", end="\n")
154     # print("# _____ Linear Resistive Network Solver _____ #", end="\n")

```

```

157 # print("# ----- Manual CSV Data ----- #", end="\n")
# print("# ----- #", end="\n\n")
# lrn = LinearResistiveNetworkSolver("data/test_c1.csv")
159 # voltages = lrn.solve()
# print(" Voltages:", end="\n")
161 # for i, v in enumerate(voltages):
#     print(" Node", i, end=": ")
163 #     print(v, " Volts", end="\n")
# print("\n", end="\n")

165
# print("# ----- TEST ----- #", end="\n")
167 # print("# ----- Linear Resistive Network Solver ----- #", end="\n")
# print("# ----- Finite Difference Mesh ----- #", end="\n")
169 # print("# ----- #", end="\n\n")
new_fname = "data/test_save.csv"
171 N = 15
# print("Mesh size:\n", N, "x", N, end="\n\n")
173 LinearResistiveNetworkSolver.create_lrn_mesh_data(N=N, fname=new_fname)
lrn = LinearResistiveNetworkSolver(new_fname)
175 voltages = lrn.solve()
r_eq = (voltages[0] - voltages[-1]) / (1 - (voltages[0] - voltages[-1]))
177 # print("Resistance:\n", r_eq, "Ohms", end="\n\n")

```

Listing 3 . lrn_solver.py