On-the-Fly Generations of the Constraint Matrix in a Maximum Entropy Problem

Xin Wang Mark Milman Feng Jiang

December 23, 1997

Abstract

The constraint matrix in the maximal entropy problem studied in [MJJ97] requires a massive amount of memory to store, so large that it prevents solving large problems even on state of the art supercomputers. Here we propose two algorithms to generate entries of the matrix from a very small set of parameter supporting points. The first algorithm is directly from the definition, and the second is an incremental version of the first, which is more efficient where entries of individual rows of the matrix are needed consecutively. These algorithms can be used for on-the-fly generation or generation on demand of the entries, and thus allow one to trade off time for space in order to avoid the massive memory requirement of storing the matrix. The algorithms are suitable for sequential as well as for parallel and distributed implementations.

1 Definition of the Constraint Matrix

In a maximum entropy problem studied in [MJJ97], a key computational issue is how to handle the constraint matrix A so that problem instances of large size can solved with reasonable computing resources (time and space). The matrix A depends on a grid formed within a parameter space. For simplicity, we only consider here the case where all parameters have an equal number of supporting points. Let q be the number of problem parameters and g be the number of supporting points for each parameter. For each $k = 0, 1, ..., q-1$, let

$$X_k = \{x_k^0, ..., x_k^{g-1}\} \subset \mathbf{R}$$

be the set of supporting points (real numbers) for the k-th parameter. Then the grid formed by these sets X_k is

$$\Omega = X_0 \times X_1 \times \cdots \times X_{q-1} \subset \mathbf{R}^q,$$

and the grid points in Ω can be enumerated as

$$x^j = (x_0^{j_0}, x_1^{j_1}, ..., x_{q-1}^{j_{q-1}}),$$

where $j = 0, 1, ..., g^q - 1$, and $(j_0, ..., j_{q-1})$ is the q-dimensional representation of j with the base g; that is,

$$j = j_0 + j_1 g + \cdots + j_{q-1} g^{q-1}.$$

It is very important to note that the enumeration above establishes a one-to-one relationship between any grid point and its parameter coordinates. With this relationship, the constraint matrix A can be defined as follows: let $n = g^q$.

- For $i = 0$ and $j = 0, 1, ..., n-1$,
$$A(i, j) = 1.0.$$
- For $i = 0, ..., q-1$ and $j = 0, 1, ..., n-1$,
$$A(i+1, j) = x_i^{j_i}.$$

- For the rest of the rows, there are three cases.

  Case 1: Means only. In this case, no more rows are required.

  Case 2: Means and auto-correlations only. For $i = 0, ..., q-1$ and $j = 0, 1, ..., n-1$,

  $$A(i+q+1, j) = (x_i^{j_i})^2.$$

  Case 3: Means, auto-correlations, and cross-correlations. This is an add-on to Case 2. For $i = 0, ..., q(q-1)/2-1$ and $j = 0, 1, ..., n-1$,

  $$A(i+2*q+1, j) = x_{i_0}^{j_{i_0}} x_{i_1}^{j_{i_1}}.$$

  Here, for each $i = 0, 1, ..., q(q-1)/2-1$, (i_0, i_1) with $i_0 > i_1$ is the 2-dimensional representation of i with base q; that is,

  $$i = i_0 + i_1 * q.$$

According to the definition above, the number of columns of A is equal to $n = g^q$, while the number of rows of A is the number m of constraints posted on the problem, which has three cases: (1) means only: $m = q+1$, (2) means and auto-correlations: $m = 2q+1$, and (3) means and cross-correlations: $m = (q^2+q)/2+1$.

For example, when $q = 2$, $g = 3$ and

$$\begin{aligned} X_0 &= \{x_0^0, x_0^1, x_0^2\} \\ X_1 &= \{x_1^0, x_1^1, x_1^2\}, \end{aligned}$$

the grid generated from X_0 and X_1 is

$$\begin{aligned} \Omega = \{&x^0 = (x_0^0, x_1^0), x^1 = (x_0^1, x_1^0), x^2 = (x_0^2, x_1^0), \\ &x^3 = (x_0^0, x_1^1), x^4 = (x_0^1, x_1^1), x^5 = (x_0^2, x_1^1), \\ &x^6 = (x_0^0, x_1^2), x^7 = (x_0^1, x_1^2), x^8 = (x_0^2, x_1^2)\}. \end{aligned}$$

The matrix is

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_0^0 & x_0^1 & x_0^2 & x_0^0 & x_0^1 & x_0^2 & x_0^0 & x_0^1 & x_0^2 \\ x_1^0 & x_1^0 & x_1^0 & x_1^1 & x_1^1 & x_1^1 & x_1^2 & x_1^2 & x_1^2 \\ x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 & x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 & x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 \\ x_1^0x_1^0 & x_1^0x_1^0 & x_1^0x_1^0 & x_1^1x_1^1 & x_1^1x_1^1 & x_1^1x_1^1 & x_1^2x_1^2 & x_1^2x_1^2 & x_1^2x_1^2 \end{bmatrix},$$

in Case 2 and is

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ x_0^0 & x_0^1 & x_0^2 & x_0^0 & x_0^1 & x_0^2 & x_0^0 & x_0^1 & x_0^2 \\ x_1^0 & x_1^0 & x_1^0 & x_1^1 & x_1^1 & x_1^1 & x_1^2 & x_1^2 & x_1^2 \\ x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 & x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 & x_0^0x_0^0 & x_0^1x_0^1 & x_0^2x_0^2 \\ x_1^0x_1^0 & x_1^0x_1^0 & x_1^0x_1^0 & x_1^1x_1^1 & x_1^1x_1^1 & x_1^1x_1^1 & x_1^2x_1^2 & x_1^2x_1^2 & x_1^2x_1^2 \\ x_0^0x_1^0 & x_0^1x_1^0 & x_0^2x_1^0 & x_0^0x_1^1 & x_0^1x_1^1 & x_0^2x_1^1 & x_0^0x_1^2 & x_0^1x_1^2 & x_0^2x_1^2 \end{bmatrix}.$$

in Case 3.

2 Memory Requirement of the Constraint Matrix

Ideally, elements of the constraint matrix are calculated once and stored in an array so that they can be used subsequently without re-calculation. However, due to the size of the matrix, storing its elements requires a substantial amount of memory. Tables 1, 2 and 3 list the memory requirement M (in MegaBytes) of the

q (# of parameters)	4	5	6	7	8	9	10
$m = q + 1$ (# of rows)	9	11	13	15	17	19	21
$n = g^q$ (# of columns)	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
M (MegaByte)	0.20	2.4	28	320	3,600	40,000	440,000

Table 1: The memory requirement of the constraint matrix in Case 1.

q (# of parameters)	4	5	6	7	8	9	10
$m = 2 * q + 1$ (# of rows)	5	6	7	8	9	10	11
$n = g^q$ (# of columns)	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
M (MegaByte)	0.36	4.4	52	600	6,800	76,000	840,000

Table 2: The memory requirement of the constraint matrix in Case 2.

matrix in Cases 1, 2 and 3 above for examples of several numbers of parameters. M is calculated according to the following formula,

$$M = s * n * m/10^6 = s * g^q * m/10^6$$

where s is the number of bytes that each floating point number takes. In the tables, s is assumed to 4 and g is 10.

Clearly, as the number of parameters q becomes greater than 6, the memory requirement M alone exceeds capabilities that a single state-of-the-art PC or workstation can offer. Moreover, according to the current serial implementation of the maximum entropy computation used in [MJJ97], 7000 to 20000 PC nodes are needed to make up the required memory space for the q = 10 case. Therefore, generating elements of the matrix on-the-fly is critical to solving problems having more than 6 parameters.

3 A Simple Algorithm

A simple algorithm for computing the entries $A(i, j)$ is a straightforward implementation of the definition above. For instance, each $A(i, j)$ in Case 3 can be computed as follows; Cases 1 and 2 can be treated as its subcases.

```
real function A(i,j)

  if (i == 0)
     return 1.0
  end
  if (1 <= i < q+1)
     call j_decomp(j, ji)
     return x(i-1, ji(i-1))
  end
  if (q+1 <= i < 2*q+1)
     call j_decomp(j, ji)
     return x(i-q-1, ji(i-1))*x(i-1, ji(i-1))
  end
  if (2*q+1 <= i < m)
     call j_decomp(j, ji)
     call i_decomp(i-2*q-1, i0, i1)
     return x(i0, ji(i0))*x(i1, ji(i1))
  end
```

q (# of parameters)	4	5	6	7	8	9	10
$m = (q^2+q)/2+q+1$ (# of rows)	15	21	28	36	45	55	66
$n = g^q$ (# of columns)	10^4	10^5	10^6	10^7	10^8	10^9	10^{10}
M (MegaByte)	0.60	8.4	112	1,440	18,000	220,000	2,640,000

Table 3: The memory requirement of the constraint matrix in Case 3.

```
end
```

Here, the subroutines j_decomp and i_decomp are used to compute j's coordinates $(j_0, ..., j_{q-1})$ and i's coordinates (i_0, i_1), which are defined as follows:

```
constant w = (1, g, g^2, ..., g^{q-1})

subroutine j_decomp(j, ji)
  jj = j
  for i = q-1 to 1 do
      ji(i) = floor(jj/w(i))
      jj = jj - ji(i)*w(i)
  end do
  ji(0) = jj
end

subroutine i_decomp(i, i0, i1)
  i0 = 0
  for (k = 0; k < q, i0+k <= i; k++)
     i0 = i0 + k
  end
  i1 = i - i0
  i0 = k
end
```

The problem with this simple algorithm is its inefficiency when it is required to compute all the entries of the matrix. This is because each entry needs to perform the rather time-consuming decompositions j_decomp and/or i_decomp.

4 A Row-Wise Incremental Algorithm

When the constraint matrix is used in the maximum entropy computation, all of its entries will be used in a row-by-row or column-by-column manner. For example, when computing the following multiplication

$$B = ADA' \tag{1}$$

where D is a diagonal matrix and $'$ denotes the matrix transposition, a trip loop is used:

```
for s = 0 to m-1 do
    for t = 0 to m-1 do
        B(s, t) = 0.0
        for j = 0 to n-1 do
            B(s, t) = A(s, j)*D(j, j)*A(t, j)
        end do
    end do
end do
```

Here the innermost loop requires generating the entries $A(s,j)$ and $A(t,j)$ in the rows s and t. Because computing j's coordinates $(j_0, ..., j_{q-1})$ is rather time-consuming, it is very desirable to compute $A(s,j)$ and $A(t,j)$ incrementally on j.

A careful examination on the definition of the constraint matrix A and an incremental relation of $(j+1)_i$ with j_i leads to the following incremental algorithm for generating the entries of A.

- For $i = 0$ and $j = 0, 1, ..., n-1$,
$$A(i,j) = 1.0.$$

- For $i = 0, ..., q-1$,

```
j = -1;
for k1 = 0:(g^i-1),
  for ji = 0:(g-1),
    for k0 = 0:(g^(q-i-1)-1),
      j = j+1;                  % invariant: 0 <= j < n
      Aij = X(i+1, ji+1);       % = A(i+1, j)
    end %g^(q-i-1)
  end %g^(q-i-1+1) = g^(q-i)
end %g^(q-i+i) = g^q = n
```

- For Case 2 and $i = 0, ..., q-1$,

```
j = -1;
for k1 = 0:(g^i-1),
  for ji = 0:(g-1),
    for k0 = 0:(g^(q-i-1)-1),
      j = j+1;                  % invariant: 0 <= j < n
      Aij = X(i+1, ji+1)*X(i+1, ji+1); % = A(i+q+1, j)
    end %g^(q-i-1)
  end %g^(q-i-1+1) = %g^(q-i)
end %g^(q-i+i) = g^q = n
```

- For Case 3, use Case 2 and

```
i0 = 0;   i1 = 0;
for i = 0:((q-1)*q/2 -1),
  i0 = i0 + 1;
  if (i0 == q),
    i1 = i1 + 1;  i0 = i1 + 1;  % invariant: q > i0 > i1 >= 0
  end

  j = -1;
  for k2 = 0:(g^i1-1),
    for ji1 = 0:(g-1),
      for k1 = 0:(g^(i0-i1-1)-1),
        for ji0 = 0:(g-1),
          for k0 = 0:(g^(q-i0-1)-1),
            j = j+1;            % invariant: 0 <= j < n
            Aij = X(i0+1, ji0+1)*X(i1+1, ji1+1); % = A(i+2*q+1, j)
          end %g^(q-i0-1)
        end %g^(q-i0-1+1) = g^(q-i0)
      end %g^(q-i0+i0-i1-1) = g^(q-i1-1)
    end %g^(q-i1-1+1) = g^(q-i1)
  end %g^(q-i1+i1) = g^q = n
end % for i = 0:((q-1)*q/2 -1),
```

5 A Column-Wise Incremental Algorithm

In (1), the diagonal matrix D has to be computed from the matrix A. Let y be a vector of real numbers of length m, and let $b = A'y$ be the product vector of A' and y. Then, the definition of the diagonal entries of the diagonal matrix D is

$$D_{kk} = e^{b_k - 1}.$$

To compute b_k's in an incremental manner, it is necessary to enumerate entries of the matrix A column-wise, as

$$b_k = \sum_{i=1}^{m} A_{ik} y_i.$$

Here is an algorithm for accomphishing this task in Case 3; again, the two other cases are its subcases.

```
ji = zeros(q, 1);
for j = 0:n-1,
  k = q;                    % figure out j's decomposition
  while ji(k) == g,
    ji(k) = 0;
    k = k-1;
    ji(k) = ji(k) + 1;
  end

  Tij = 1.0;                % i = 0;

  for i = 1:q,              % i = 1 to q
    Tij = X(i, ji(i)+1);
  end

  for i = 1:q,              % i = q+1 to 2*q+1
    Tij = X(i, ji(i)+1)*X(i, ji(i)+1);
  end

  for i1 = 1:(q-1),         % i = 2*q+1 to (q+1)*q/2 + 2*q + 1
    for i0 = (i1+1):q, % invariant 0 <= i1 < i0 <= q-1
      Tij = X(i0, ji(i0)+1)*X(i1, ji(i1)+1);
    end
  end
  ji(q) = ji(q)+1;
end % for j = 0:(n-1)
```

Based on the algorithm above that computes the constraint matrix column-wise, the matrix D can be computed as follows (again for Case 3).

```
ji = zeros(q, 1);
for j = 0:n-1,
  k = q;                    % figure out j's decomposition
  while ji(k) == g,
    ji(k) = 0;
    k = k-1;
    ji(k) = ji(k) + 1;
  end

  Tij = 1.0;                % i = 0;
  Djj = lambda(1);
```

```
  for i = 1:q,              % i = 1 to q
    Tij = X(i, ji(i)+1);
    Djj = Djj + Tij*lambda(i+1);
  end

  for i = 1:q,              % i = q+1 to 2*q+1
    Tij = X(i, ji(i)+1)*X(i, ji(i)+1);
    Djj = Djj + Tij*lambda(i+q+1);
  end

  for i1 = 1:(q-1),         % i = 2*q+1 to (q+1)*q/2 + 2*q + 1
    for i0 = (i1+1):q, % invariant 0 <= i1 < i0 <= q-1
      Tij = X(i0, ji(i0)+1)*X(i1, ji(i1)+1);
      Djj = Djj + Tij*lambda(i+2*q+1);
    end
  end
  ji(q) = ji(q)+1;

  Djj = exp(Djj - 1.0);     % the j-th entry of the matrix D
end % for j = 0:(n-1)
```

6 Conclusion

In this paper, we have given a definition of the individual entries of the constraint matrix used in solving the maximum entropy problem in [MJJ97]. The definition is based on a small set of supporting points for problem parameters, and thus makes it possible to eliminate the massive memory requirement of storing the matrix by generating its entries only when needed. To improve the efficiency of the on-the-fly generation of the entries in batch, we have devised two incremental algorithms for generating entries row-by-row and column-by-column, respectively. The algorithms are suitable for sequential as well as for parallel and distributed implementations.

References

[MJJ97] M. Milman, F. Jiang, and R. W. Jelliffe. Creating Discrete Joint Densities from Continuous Ones: The Moment Matching-Maximum Entropy Approach. LAPK Technical Report 97-2.

[Wang97] X. Wang, "Scalable Parallel Matrix Multiplication Algorithms with Application to a Maximum Entropy Problem", LAPK Technical Report 97-3, USC, September, 1997.

Appendix: Matlab Code and Test Results for the Incremental Computation of the Constraint Matrix

Matlab Code for Row-Wise Incremental Computation of the Matrix

```
function A = matrix_calc(X, q, g, opt)
%This function computes constraint matrix T incrementally

% number of columns
n = g^q;
```

```
A = [];  % for test purpose

Arow = [];  % for test purpose
for j = 0:(n-1),
  Aij = 1.0;
  Arow = [Arow, Aij];   % for test purpose
end
A = [A; Arow];  % for test purpose

for i = 0:q-1,
  Arow = [];  % for test purpose
  j = -1;
  for k1 = 0:(g^i-1),
    for ji = 0:(g-1),
      for k0 = 0:(g^(q-i-1)-1),

        j = j+1;
        Aij = X(i+1, ji+1);

        Arow = [Arow, Aij];  % for test purpose
      end
    end
  end
  A = [A; Arow];  % for test purpose
end

if (opt == 0),
  m = q+1;
  return;
end

for i = 0:q-1,
  Arow = [];  % for test purpose
  j = -1;
  for k1 = 0:(g^i-1),
    for ji = 0:(g-1),
      for k0 = 0:(g^(q-i-1)-1),

        j = j+1;
        Aij = X(i+1, ji+1)*X(i+1, ji+1);

        Arow = [Arow, Aij];  % for test purpose
      end
    end
  end
  A = [A; Arow];  % for test purpose
end

if (opt == 1),
  m = 2*q+1;
  return;
end
```

```
i = -1;
for i1 = 0:(q-2),
  for i0 = (i1+1):(q-1),  %invariant 0 <= i1 < i0 <= q-1
    i = i+1;
    Arow = [];  % for test purpose

    j = -1; % for j = 0:(n-1),
    for k2 = 0:(g^i1-1),
      for ji0 = 0:(g-1),
       for k1 = 0:(g^(i0-i1-1)-1),
          for ji1 = 0:(g-1),
           for k0 = 0:(g^(q-i0-1)-1),

              j = j+1;
              Aij = X(i0+1, ji0+1)*X(i1+1, ji1+1);

              Arow = [Arow, Aij];  % for test purpose
            end %g^(q-i0-1)
          end %g^(q-i0-1+1) = g^(q-i0)
       end %g^(q-i0+i0-i1-1) = g^(q-i1-1)
      end %g^(q-i1-1+1) = g^(q-i1)
    end %g^(q-i1+i1) = g^q = n
    A = [A; Arow];  % for test purpose
  end % for i0 = (i1+1):(q-1)
end % for i1 = 0:(q-2)

if (opt == 2),
  m = (q^2+q)/2 + q + 1;
  return
end
```

Test Results

To verify the correctness of the incremental algorithm, the following set of test data was used.

```
g = 4;
q = 3;
X = [0.3125,1.6667,3.3333,5.0000;
     0.0938,0.5000,1.0000,1.5000;
     1.0938,1.5000,2.0000,2.5000];
```

The following are the tranposes of the three constraint matrices $A1$, $A2$ and $A3$, resulting from the calculation, for Cases 1, 2 and 3.

```
    A1' =
        1.0000  0.3125  0.0938  1.0938
        1.0000  0.3125  0.0938  1.5000
        1.0000  0.3125  0.0938  2.0000
        1.0000  0.3125  0.0938  2.5000
        1.0000  0.3125  0.5000  1.0938
        1.0000  0.3125  0.5000  1.5000
        1.0000  0.3125  0.5000  2.0000
        1.0000  0.3125  0.5000  2.5000
        1.0000  0.3125  1.0000  1.0938
        1.0000  0.3125  1.0000  1.5000
```

```
    1.0000  0.3125  1.0000  2.0000
    1.0000  0.3125  1.0000  2.5000
    1.0000  0.3125  1.5000  1.0938
    1.0000  0.3125  1.5000  1.5000
    1.0000  0.3125  1.5000  2.0000
    1.0000  0.3125  1.5000  2.5000
    1.0000  1.6667  0.0938  1.0938
    1.0000  1.6667  0.0938  1.5000
    1.0000  1.6667  0.0938  2.0000
    1.0000  1.6667  0.0938  2.5000
    1.0000  1.6667  0.5000  1.0938
    1.0000  1.6667  0.5000  1.5000
    1.0000  1.6667  0.5000  2.0000
    1.0000  1.6667  0.5000  2.5000
    1.0000  1.6667  1.0000  1.0938
    1.0000  1.6667  1.0000  1.5000
    1.0000  1.6667  1.0000  2.0000
    1.0000  1.6667  1.0000  2.5000
    1.0000  1.6667  1.5000  1.0938
    1.0000  1.6667  1.5000  1.5000
    1.0000  1.6667  1.5000  2.0000
    1.0000  1.6667  1.5000  2.5000
    1.0000  3.3333  0.0938  1.0938
    1.0000  3.3333  0.0938  1.5000
    1.0000  3.3333  0.0938  2.0000
    1.0000  3.3333  0.0938  2.5000
    1.0000  3.3333  0.5000  1.0938
    1.0000  3.3333  0.5000  1.5000
    1.0000  3.3333  0.5000  2.0000
    1.0000  3.3333  0.5000  2.5000
    1.0000  3.3333  1.0000  1.0938
    1.0000  3.3333  1.0000  1.5000
    1.0000  3.3333  1.0000  2.0000
    1.0000  3.3333  1.0000  2.5000
    1.0000  3.3333  1.5000  1.0938
    1.0000  3.3333  1.5000  1.5000
    1.0000  3.3333  1.5000  2.0000
    1.0000  3.3333  1.5000  2.5000
    1.0000  5.0000  0.0938  1.0938
    1.0000  5.0000  0.0938  1.5000
    1.0000  5.0000  0.0938  2.0000
    1.0000  5.0000  0.0938  2.5000
    1.0000  5.0000  0.5000  1.0938
    1.0000  5.0000  0.5000  1.5000
    1.0000  5.0000  0.5000  2.0000
    1.0000  5.0000  0.5000  2.5000
    1.0000  5.0000  1.0000  1.0938
    1.0000  5.0000  1.0000  1.5000
    1.0000  5.0000  1.0000  2.0000
    1.0000  5.0000  1.0000  2.5000
    1.0000  5.0000  1.5000  1.0938
    1.0000  5.0000  1.5000  1.5000
    1.0000  5.0000  1.5000  2.0000
    1.0000  5.0000  1.5000  2.5000

A2' =
    1.0000  0.3125  0.0938  1.0938  0.0977  0.0088  1.1964
    1.0000  0.3125  0.0938  1.5000  0.0977  0.0088  2.2500
```

1.0000	0.3125	0.0938	2.0000	0.0977	0.0088	4.0000
1.0000	0.3125	0.0938	2.5000	0.0977	0.0088	6.2500
1.0000	0.3125	0.5000	1.0938	0.0977	0.2500	1.1964
1.0000	0.3125	0.5000	1.5000	0.0977	0.2500	2.2500
1.0000	0.3125	0.5000	2.0000	0.0977	0.2500	4.0000
1.0000	0.3125	0.5000	2.5000	0.0977	0.2500	6.2500
1.0000	0.3125	1.0000	1.0938	0.0977	1.0000	1.1964
1.0000	0.3125	1.0000	1.5000	0.0977	1.0000	2.2500
1.0000	0.3125	1.0000	2.0000	0.0977	1.0000	4.0000
1.0000	0.3125	1.0000	2.5000	0.0977	1.0000	6.2500
1.0000	0.3125	1.5000	1.0938	0.0977	2.2500	1.1964
1.0000	0.3125	1.5000	1.5000	0.0977	2.2500	2.2500
1.0000	0.3125	1.5000	2.0000	0.0977	2.2500	4.0000
1.0000	0.3125	1.5000	2.5000	0.0977	2.2500	6.2500
1.0000	1.6667	0.0938	1.0938	2.7779	0.0088	1.1964
1.0000	1.6667	0.0938	1.5000	2.7779	0.0088	2.2500
1.0000	1.6667	0.0938	2.0000	2.7779	0.0088	4.0000
1.0000	1.6667	0.0938	2.5000	2.7779	0.0088	6.2500
1.0000	1.6667	0.5000	1.0938	2.7779	0.2500	1.1964
1.0000	1.6667	0.5000	1.5000	2.7779	0.2500	2.2500
1.0000	1.6667	0.5000	2.0000	2.7779	0.2500	4.0000
1.0000	1.6667	0.5000	2.5000	2.7779	0.2500	6.2500
1.0000	1.6667	1.0000	1.0938	2.7779	1.0000	1.1964
1.0000	1.6667	1.0000	1.5000	2.7779	1.0000	2.2500
1.0000	1.6667	1.0000	2.0000	2.7779	1.0000	4.0000
1.0000	1.6667	1.0000	2.5000	2.7779	1.0000	6.2500
1.0000	1.6667	1.5000	1.0938	2.7779	2.2500	1.1964
1.0000	1.6667	1.5000	1.5000	2.7779	2.2500	2.2500
1.0000	1.6667	1.5000	2.0000	2.7779	2.2500	4.0000
1.0000	1.6667	1.5000	2.5000	2.7779	2.2500	6.2500
1.0000	3.3333	0.0938	1.0938	11.1109	0.0088	1.1964
1.0000	3.3333	0.0938	1.5000	11.1109	0.0088	2.2500
1.0000	3.3333	0.0938	2.0000	11.1109	0.0088	4.0000
1.0000	3.3333	0.0938	2.5000	11.1109	0.0088	6.2500
1.0000	3.3333	0.5000	1.0938	11.1109	0.2500	1.1964
1.0000	3.3333	0.5000	1.5000	11.1109	0.2500	2.2500
1.0000	3.3333	0.5000	2.0000	11.1109	0.2500	4.0000
1.0000	3.3333	0.5000	2.5000	11.1109	0.2500	6.2500
1.0000	3.3333	1.0000	1.0938	11.1109	1.0000	1.1964
1.0000	3.3333	1.0000	1.5000	11.1109	1.0000	2.2500
1.0000	3.3333	1.0000	2.0000	11.1109	1.0000	4.0000
1.0000	3.3333	1.0000	2.5000	11.1109	1.0000	6.2500
1.0000	3.3333	1.5000	1.0938	11.1109	2.2500	1.1964
1.0000	3.3333	1.5000	1.5000	11.1109	2.2500	2.2500
1.0000	3.3333	1.5000	2.0000	11.1109	2.2500	4.0000
1.0000	3.3333	1.5000	2.5000	11.1109	2.2500	6.2500
1.0000	5.0000	0.0938	1.0938	25.0000	0.0088	1.1964
1.0000	5.0000	0.0938	1.5000	25.0000	0.0088	2.2500
1.0000	5.0000	0.0938	2.0000	25.0000	0.0088	4.0000
1.0000	5.0000	0.0938	2.5000	25.0000	0.0088	6.2500
1.0000	5.0000	0.5000	1.0938	25.0000	0.2500	1.1964
1.0000	5.0000	0.5000	1.5000	25.0000	0.2500	2.2500
1.0000	5.0000	0.5000	2.0000	25.0000	0.2500	4.0000
1.0000	5.0000	0.5000	2.5000	25.0000	0.2500	6.2500
1.0000	5.0000	1.0000	1.0938	25.0000	1.0000	1.1964
1.0000	5.0000	1.0000	1.5000	25.0000	1.0000	2.2500
1.0000	5.0000	1.0000	2.0000	25.0000	1.0000	4.0000
1.0000	5.0000	1.0000	2.5000	25.0000	1.0000	6.2500

```
    1.0000  5.0000  1.5000  1.0938  25.0000 2.2500  1.1964
    1.0000  5.0000  1.5000  1.5000  25.0000 2.2500  2.2500
    1.0000  5.0000  1.5000  2.0000  25.0000 2.2500  4.0000
    1.0000  5.0000  1.5000  2.5000  25.0000 2.2500  6.2500

A3' =
    1.0000  0.3125  0.0938  1.0938  0.0977  0.0088  1.1964  0.0293  0.3418  0.1026
    1.0000  0.3125  0.0938  1.5000  0.0977  0.0088  2.2500  0.0293  0.4688  0.1407
    1.0000  0.3125  0.0938  2.0000  0.0977  0.0088  4.0000  0.0293  0.6250  0.1876
    1.0000  0.3125  0.0938  2.5000  0.0977  0.0088  6.2500  0.0293  0.7813  0.2345
    1.0000  0.3125  0.5000  1.0938  0.0977  0.2500  1.1964  0.1563  0.3418  0.5469
    1.0000  0.3125  0.5000  1.5000  0.0977  0.2500  2.2500  0.1563  0.4688  0.7500
    1.0000  0.3125  0.5000  2.0000  0.0977  0.2500  4.0000  0.1563  0.6250  1.0000
    1.0000  0.3125  0.5000  2.5000  0.0977  0.2500  6.2500  0.1563  0.7813  1.2500
    1.0000  0.3125  1.0000  1.0938  0.0977  1.0000  1.1964  0.3125  0.3418  1.0938
    1.0000  0.3125  1.0000  1.5000  0.0977  1.0000  2.2500  0.3125  0.4688  1.5000
    1.0000  0.3125  1.0000  2.0000  0.0977  1.0000  4.0000  0.3125  0.6250  2.0000
    1.0000  0.3125  1.0000  2.5000  0.0977  1.0000  6.2500  0.3125  0.7813  2.5000
    1.0000  0.3125  1.5000  1.0938  0.0977  2.2500  1.1964  0.4688  0.3418  1.6407
    1.0000  0.3125  1.5000  1.5000  0.0977  2.2500  2.2500  0.4688  0.4688  2.2500
    1.0000  0.3125  1.5000  2.0000  0.0977  2.2500  4.0000  0.4688  0.6250  3.0000
    1.0000  0.3125  1.5000  2.5000  0.0977  2.2500  6.2500  0.4688  0.7813  3.7500
    1.0000  1.6667  0.0938  1.0938  2.7779  0.0088  1.1964  0.1563  1.8230  0.1026
    1.0000  1.6667  0.0938  1.5000  2.7779  0.0088  2.2500  0.1563  2.5001  0.1407
    1.0000  1.6667  0.0938  2.0000  2.7779  0.0088  4.0000  0.1563  3.3334  0.1876
    1.0000  1.6667  0.0938  2.5000  2.7779  0.0088  6.2500  0.1563  4.1668  0.2345
    1.0000  1.6667  0.5000  1.0938  2.7779  0.2500  1.1964  0.8334  1.8230  0.5469
    1.0000  1.6667  0.5000  1.5000  2.7779  0.2500  2.2500  0.8334  2.5001  0.7500
    1.0000  1.6667  0.5000  2.0000  2.7779  0.2500  4.0000  0.8334  3.3334  1.0000
    1.0000  1.6667  0.5000  2.5000  2.7779  0.2500  6.2500  0.8334  4.1668  1.2500
    1.0000  1.6667  1.0000  1.0938  2.7779  1.0000  1.1964  1.6667  1.8230  1.0938
    1.0000  1.6667  1.0000  1.5000  2.7779  1.0000  2.2500  1.6667  2.5001  1.5000
    1.0000  1.6667  1.0000  2.0000  2.7779  1.0000  4.0000  1.6667  3.3334  2.0000
    1.0000  1.6667  1.0000  2.5000  2.7779  1.0000  6.2500  1.6667  4.1668  2.5000
    1.0000  1.6667  1.5000  1.0938  2.7779  2.2500  1.1964  2.5001  1.8230  1.6407
    1.0000  1.6667  1.5000  1.5000  2.7779  2.2500  2.2500  2.5001  2.5001  2.2500
    1.0000  1.6667  1.5000  2.0000  2.7779  2.2500  4.0000  2.5001  3.3334  3.0000
    1.0000  1.6667  1.5000  2.5000  2.7779  2.2500  6.2500  2.5001  4.1668  3.7500
    1.0000  3.3333  0.0938  1.0938  11.1109 0.0088  1.1964  0.3127  3.6460  0.1026
    1.0000  3.3333  0.0938  1.5000  11.1109 0.0088  2.2500  0.3127  5.0000  0.1407
    1.0000  3.3333  0.0938  2.0000  11.1109 0.0088  4.0000  0.3127  6.6666  0.1876
    1.0000  3.3333  0.0938  2.5000  11.1109 0.0088  6.2500  0.3127  8.3333  0.2345
    1.0000  3.3333  0.5000  1.0938  11.1109 0.2500  1.1964  1.6667  3.6460  0.5469
    1.0000  3.3333  0.5000  1.5000  11.1109 0.2500  2.2500  1.6667  5.0000  0.7500
    1.0000  3.3333  0.5000  2.0000  11.1109 0.2500  4.0000  1.6667  6.6666  1.0000
    1.0000  3.3333  0.5000  2.5000  11.1109 0.2500  6.2500  1.6667  8.3333  1.2500
    1.0000  3.3333  1.0000  1.0938  11.1109 1.0000  1.1964  3.3333  3.6460  1.0938
    1.0000  3.3333  1.0000  1.5000  11.1109 1.0000  2.2500  3.3333  5.0000  1.5000
    1.0000  3.3333  1.0000  2.0000  11.1109 1.0000  4.0000  3.3333  6.6666  2.0000
    1.0000  3.3333  1.0000  2.5000  11.1109 1.0000  6.2500  3.3333  8.3333  2.5000
    1.0000  3.3333  1.5000  1.0938  11.1109 2.2500  1.1964  5.0000  3.6460  1.6407
    1.0000  3.3333  1.5000  1.5000  11.1109 2.2500  2.2500  5.0000  5.0000  2.2500
    1.0000  3.3333  1.5000  2.0000  11.1109 2.2500  4.0000  5.0000  6.6666  3.0000
    1.0000  3.3333  1.5000  2.5000  11.1109 2.2500  6.2500  5.0000  8.3333  3.7500
    1.0000  5.0000  0.0938  1.0938  25.000  0.0088  1.1964  0.4690  5.4690  0.1026
    1.0000  5.0000  0.0938  1.5000  25.000  0.0088  2.2500  0.4690  7.5000  0.1407
    1.0000  5.0000  0.0938  2.0000  25.000  0.0088  4.0000  0.4690  10.000  0.1876
    1.0000  5.0000  0.0938  2.5000  25.000  0.0088  6.2500  0.4690  12.500  0.2345
```

1.0000	5.0000	0.5000	1.0938	25.000	0.2500	1.1964	2.5000	5.4690	0.5469
1.0000	5.0000	0.5000	1.5000	25.000	0.2500	2.2500	2.5000	7.5000	0.7500
1.0000	5.0000	0.5000	2.0000	25.000	0.2500	4.0000	2.5000	10.000	1.0000
1.0000	5.0000	0.5000	2.5000	25.000	0.2500	6.2500	2.5000	12.500	1.2500
1.0000	5.0000	1.0000	1.0938	25.000	1.0000	1.1964	5.0000	5.4690	1.0938
1.0000	5.0000	1.0000	1.5000	25.000	1.0000	2.2500	5.0000	7.5000	1.5000
1.0000	5.0000	1.0000	2.0000	25.000	1.0000	4.0000	5.0000	10.000	2.0000
1.0000	5.0000	1.0000	2.5000	25.000	1.0000	6.2500	5.0000	12.500	2.5000
1.0000	5.0000	1.5000	1.0938	25.000	2.2500	1.1964	7.5000	5.4690	1.6407
1.0000	5.0000	1.5000	1.5000	25.000	2.2500	2.2500	7.5000	7.5000	2.2500
1.0000	5.0000	1.5000	2.0000	25.000	2.2500	4.0000	7.5000	10.000	3.0000
1.0000	5.0000	1.5000	2.5000	25.000	2.2500	6.2500	7.5000	12.500	3.7500