CHAPTER 1

Markov Computations

[lraise=0.1]Many of the results and derivations in this chapter are taken directly from [?], though the derivations presented here are likely much more detailed. The former reference focuses on Markov chains more generally, and is the appropriate source to consult for general questions.

1.1. Representing models

Consider the simple model of fig. 1.1.1. We can express this model in a computer-readable manner in many different ways, but one relatively human-friendly manner is TML. TML is a text-based language explicitly developed for representing Markov chain usage models [?]. The given model is equivalent to the following TML.

```
model example
[Enter]  ($  1  $)  "a"  [A]
[A]      ($ 1/2 $)  "b"  [B]
         ($ 1/2 $)  "c"  [C]
[B]      ($ 1/2 $)  "b"  [B]
         ($ 1/4 $)  "c"  [C]
         ($ 1/4 $)  "e"  [Exit]
[C]      ($ 1/4 $)  "a"  [A]
         ($ 1/2 $)  "e"  [Exit]
         ($ 1/4 $)  "f"  [Exit]
end
```

A short primer on TML is included as Appendix **??**, but we note that whitespace is not significant to TML, and the model could have been written in the following form, where the probabilities are inferred.

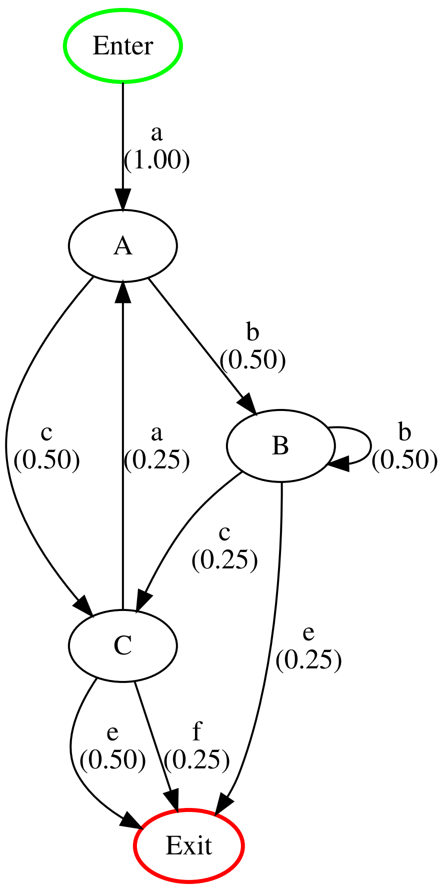

FIGURE 1.1.1. Example model

```
($assume(1) normalize(1)$)
model example
[Enter]   "a" [A]
[A]       "b" [B]
          "c" [C]
[B] ($2$) "b" [B]
          "c" [C]
          "e" [Exit]
[C]       "a" [A]
    ($2$) "e" [Exit]
          "f" [Exit]
end
```

There are two special states in this model: the *source* or start state `[Enter]`, and the *sink* or stop state `[Exit]`. The sink state represents the end of a single use and thus no usage events are possible from this state.

This model can be described by two matrices. For these matrices let the states be indexed in the order `[Enter]`, `[A]`, `[B]`, `[C]`, and `[Exit]`, and let the stimuli be indexed in alphabetic order. The first matrix is the *(state) transition matrix*, $P = [p_{i,j}]$, for which $p_{i,j}$ is the probability that the next state is j given that the current state is i. For the example model the transition matrix is the following.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 & \frac{3}{4} \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Note that the example matrix has been made *recurrent* by connecting `[Exit]` back to `[Enter]` with probability one. This will be important for several computations, and is assumed where necessary. Note that there are two transitions from state `[C]` to state `[Exit]`; these two transitions are mutually exclusive, so the probability that the next state is `[Exit]` given that the current state is `[C]` is the sum of the two: $\frac{1}{4} + \frac{1}{2} = \frac{3}{4}$. We can represent the transition matrix in R as follows, remembering that we need to use column-major form.

```
P <- matrix(c(
  0,   0,   0,   0,   1,
  1,   0,   0, 1/4,   0,
  0, 1/2, 1/2,   0,   0,
  0, 1/2, 1/4,   0,   0,
  0,   0, 1/4, 3/4,   0), nrow=5)
```

For many computations it will be useful to have a reduced matrix $Q = [q_{i,j}]$ in which the row and column for the model sink have been removed. For the example model the reduced matrix is the following.

$$Q = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{4} & 0 & 0 \end{bmatrix}$$

Both P and Q are always square matrices. We can extract the Q matrix from the P matrix in R as follows.

```
n <- nrow(P)
Q <- P[-n,-n]
```

The second matrix is the *stimulus (occurrence) matrix* $S = [s_{i,j}]$, for which $s_{i,j}$ is the probability that the next usage event (or *stimulus*) will be j, given that the current state is i. For the example model the stimulus matrix is the following.

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{4} & 0 & 0 & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

The usage models used here are deterministic; a stimulus can only label one outgoing transition from a state, so there is no need to sum transition probabilities in the stimulus matrix. We can represent the stimulus matrix in R as follows.

```
S <- matrix(c(
  1,   0,   0, 1/4,
0, 1/2, 1/2,   0,
0, 1/2, 1/4,   0,
0,   0, 1/4, 1/2,
0,   0,   0, 1/4), nrow=4)
```

It will occasionally be convenient to discuss the probability of a transition from state i to state j labeled with stimulus k. This probability will be denoted $p_{i,j,k}$ when needed. The *restriction* of a matrix A to just those elements corresponding to stimulus k will be denoted $A|_k$. Thus $Q|_k = [q_{i,j,k}]$ for fixed k, and we also have the following.

$$Q = \sum_k Q|_k$$

Throughout the rest of this document, the source will be assumed to have index one, while the sink will be assumed to have the highest index n. The number of stimuli for a model will be denoted s. Thus P is always an $n \times n$ matrix, while S is always an $(n-1) \times s$ matrix, since there can be no outgoing arcs (other than the unlabeled recurrence loop) from the sink.

For every state i in a connected model, there is a non-zero probability path from the source to i, and a non-zero probability path from i to the sink. Because of the recurrence loop the graph of the model is strongly connected, meaning that from any state i there is a path to any other state j. It follows that P is an irreducible matrix (see Theorem 6.2.24 in [?]). All models will be assumed to be connected in this manner.

1.2. Computing the number of occurrences of a state in a test case

The number of occurrences of a state in a random walk from source to sink (a test case) can be computed using the reduced matrix Q. If the sink is made absorbing (letting $p_{n,n} = 1$ and $p_{n,1} = 0$), then one can compute the number of occurrences of a state prior to absorption at the sink. Let T denote the set of transient (non-absorbing) states, which will be every state other than the sink. Let $n_{i,j}$ be a random variable counting the number of occurrences of state j prior to absorption, given that one starts in state i (if $i = j$ then we count the initial occurence, too, and use $\delta_{i,j}$ to do that). From i we may move to absorbing state k (in our models, the sink n) with probability $p_{i,k}$ and gain contribution $\delta_{i,j}$. Otherwise we move to a transient state k and gain contribution $\delta_{i,j}$ from the initial state plus $n_{k,j}$ from the later steps of the realization. Since these different possibilities are disjoint, we can simply sum them up.[1]

$$\begin{aligned}
n_{i,j} &= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}(\delta_{i,j} + n_{k,j}) \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}(\delta_{i,j} + n_{k,j}) \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,k} + \sum_{k\in T} p_{i,k}(\delta_{i,j} + n_{k,j}) \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}\delta_{i,j} + n_{k,j} \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}\left(\delta_{i,j} + n_{k,j}\right) \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}n_{k,j} \\
&= \sum_{k} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}n_{k,j} \\
&= \delta_{i,j}\sum_{k} p_{i,k} + \sum_{k\in T} p_{i,k}n_{k,j} \\
&= \delta_{i,j} + \sum_{k\in T} p_{i,k}n_{k,j} \qquad (1.2.1)
\end{aligned}$$

Let $N = [n_{i,j}]$ be the matrix of these expectations. Then eq. 1.2.1 can be re-written in matrix form as follows.

$$\begin{aligned}
N &= I + QN \\
N - QN &= I \\
(I - Q)N &= I \\
N &= (I - Q)^{-1} \qquad (1.2.2)
\end{aligned}$$

The matrix N defined by eq. 1.2.2 is called the *fundamental matrix* for absorbing chains. Many Markov chain results can be obtained from this fundamental matrix. The expected number of occurences of each state exists and is finite (because absorption is certain); we can conclude that the inverse in eq. 1.2.2 exists.

What about the variance associated with this expectation? We need to find the second moment $n^2_{i,j}$, and we can do this using the same method as we did for $n_{i,j}$, above. If we start in state i then we may move to an absorbing state k (in the specific case of our models, only the sink n) with probability $p_{i,k}$ and gain contribution $\delta^2_{i,j}$. Otherwise we move to a transient state k and gain the square of the sum of $\delta_{i,j}$ (from the initial state) and $n_{k,j}$ (from the later steps). This gives

[1]This derivation may seem excessive, but we include this detail to provide a pattern for future derivations which may be less obvious. We will skip the initial steps, which are "obvious," in future derivations.

the following equation where, we note, $\delta_{i,j}^2 = \delta_{i,j}$.

$$\begin{aligned}
n_{i,j}^2 &= \sum_{k\notin T} p_{i,k}\delta_{i,j}^2 + \sum_{k\in T} p_{i,k}(\delta_{i,j} + n_{k,j})^2 \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j}^2 + \sum_{k\in T} p_{i,k}\delta_{i,j}^2 + 2\delta_{i,j}n_{k,j} + n_{k,j}^2 \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j}^2 + \sum_{k\in T} p_{i,k}\delta_{i,j}^2 + 2\delta_{i,j}n_{k,j} + n_{k,j}^2 \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}\left(\delta_{i,j} + 2\delta_{i,j}n_{k,j} + n_{k,j}^2\right) \\
&= \sum_{k\notin T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}\delta_{i,j} + \sum_{k\in T} p_{i,k}2\delta_{i,j}n_{k,j} + \sum_{k\in T} p_{i,k}n_{k,j}^2 \\
&= \delta_{i,j}\sum_{k} p_{i,k} + 2\delta_{i,j}\sum_{k\in T} p_{i,k}n_{k,j} + \sum_{k\in T} p_{i,k}n_{k,j}^2 \\
&= 1 + 2\sum_{k\in T} p_{j,k}n_{k,j} + \sum_{k\in T} p_{i,k}n_{k,j}^2 \qquad (1.2.3)
\end{aligned}$$

Let $N_2 = \left[n_{i,j}^2\right]$ be the matrix of these expectations. Then eq. 1.2.3 can be re-written in matrix form as follows.

$$\begin{aligned}
N_2 &= I + 2(QN)_d + QN_2 \\
N_2 - QN_2 &= I + 2(QN)_d \\
N_2(I - Q) &= I + 2(QN)_d \\
N_2 &= (I - Q)^{-1}(I + 2(QN)_d) \\
N_2 &= N(I + 2(QN)_d) \\
N_2 &= N + 2N(QN)_d \qquad (1.2.4)
\end{aligned}$$

Since we already know $N = I + QN$, we can conclude that $QN = N - I$. We substitute in eq. 1.2.4 and obtain the following.

$$\begin{aligned}
N_2 &= N + 2N(N - I)_d \\
&= N + 2NN_d - 2N \\
&= 2NN_d - N \\
&= N(2N_d - I) \qquad (1.2.5)
\end{aligned}$$

Now, with eq. 1.2.5 in hand, we can compute the matrix of variances as $[n_{i,j}] = N_2 - N \circ N$.

These results allow the computation of the expected number of occurrences for each state and the associated variance by the R function in algorithm 1. Note that for a usage model one is primarily concerned with only the first row of the returned matrices, since one is concerned with the behavior when the model is started from the source. The results obtained for the example model are presented in table 1.

Let l denote the number of state transitions from the source to the sink in a test case. The expected length l of a test case generated from a Markov chain usage model can be quickly computed once N is known by just summing the expected occurrences of each state.

$$l = \sum_{k} n_{1,k} = \sum_{k} n_{1,k}$$

Algorithm 1 Compute the expected number of occurrences of each state

```
# Compute the non-terminal expectation and
# variance matrices for the stochastic matrix
# P.
#
# P: a row-stochastic matrix
# return: list(N,V) where
#   N: the expected occurrence of each state
#   V: the associated variances
get_nte <- function(P) {
   n <- nrow(P)
   Q <- P[-n, -n]
   N <- solve(diag(n - 1) - Q)
   V <- N %*% (2 * diag(diag(N)) - diag(n - 1)) - N * N
   list(N, V)
}
```

TABLE 1. Expected occurrence and associated variance for each state

State	Occurrence (visits per test case)	Variance
`[Enter]`	1.000	0.000
`[A]`	1.231	0.284 0
`[B]`	1.231	2.556
`[C]`	0.923 1	0.497 0
`[Exit]`	1.000	0.000

That is, the average length of a test case is the number of state transitions from the source to the sink. Note that l does not include the sink; the total average number of *state visits* (including the sink) is one more, since the sink is always visited exactly once. In a later section we will revisit this and also discuss how to obtain an associated variance.

1.3. Computing the long-run state probabilities

Assume that many test cases (random walks from source to sink) are generated from a usage model. For each state one can sum the number of occurrences of that state, and then divide by this by the total number of occurrences of all states. As the number of test cases becomes very large, this ratio will approach a fixed value called the *long-run occupancy* or *long-run probability* of the state. Since the usage model is always in one of its states, the sum of all long-run occupancies is one.

Let $\Pi = [\pi_1, \pi_2, \ldots, \pi_n]$ be the vector of long-run probabilities for the n states. This vector can be found for a given transition matrix P as the unique stochastic vector solution to the left-eigenvector equation for eigenvalue 1.

$$\Pi = \Pi P \tag{1.3.1}$$

Algorithm 2 Compute the Perron eigenvector (long-run probabilities)

```
# Compute the Perron eigenvector for the
# stochastic matrix P and return it. The
# computation is performed by computation
# of the fundamental matrix.
#
# P: a square row-stochastic matrix
# N: the fundamental matrix, if available
# return: the Perron eigenvector
get_pi <- function(P, N) {
   n <- nrow(P)
   if (nargs() < 2) {
      N <- get_nte(P)[[1]]
   }
   len <- 1 + sum(N[1, ])
   pi <- N[1, ]/len
   append(pi, c(1 - sum(pi)))
}
```

The left-eigenvector Π is sometimes called the *Perron* eigenvector. Eq. 1.3.1 is equivalent to the following system of equations.

$$\begin{aligned}
\pi_1 &= \pi_1 p_{1,1} + \pi_2 p_{2,1} + \cdots + \pi_n p_{n,1} \\
\pi_2 &= \pi_1 p_{1,2} + \pi_2 p_{2,2} + \cdots + \pi_n p_{n,2} \\
&\vdots \\
\pi_n &= \pi_1 p_{1,n} + \pi_2 p_{2,n} + \cdots + \pi_n p_{n,n} \\
1 &= \pi_1 + \pi_2 + \cdots + \pi_n
\end{aligned}$$

Note that there are n unknowns and $n+1$ equations. Incidentally, each row of $P^\infty = \lim_{n\to\infty} P^n$ is equal to the vector Π, unless P is periodic [?]. There are many ways to solve this system. For a simple solution by hand, back-substitution will work just fine using the above equations.

The fundamental matrix N can also be used to obtain Π. The average number of state visits in a test case is $l+1$. The fraction of time one spends in state i in the long run is π_i, so the average number of times state i will be visited in a single test case is $\pi_i(l+1)$. We conclude the following.

$$\begin{aligned}
n_{1,i} &= \pi_i\,(l+1) \\
\pi_i &= \frac{n_{1,i}}{l+1} \qquad (1.3.2)
\end{aligned}$$

Note that $n_{1,n}$ is not computed by equation 1.2.2, but we can conclude that $n_{1,n} = 1$ since the sink is always visited exactly once per test case. Algorithm 2 uses this relationship to compute Π. The results for the example model are presented in table 2. The algorithm does not give a value for `[Exit]`, but that can be obtained by subtracting the sum of the other values from one.

An alternate method for obtaining the long run distribution is to use an iterative method called the *power method*. Given the ith guess at the long-run distribution Π_i, one computes a slightly better guess by $\Pi_{i+1} = \Pi_i P$. This method is guaranteed

Table 2. Long-run occupancies

State	Long-run Occupancy
[Enter]	0.185 7
[A]	0.228 6
[B]	0.228 6
[C]	0.171 4
[Exit]	0.195 7

to converge to the Perron eigenvector if the matrix is *primitive* [**?**, p. 516]. Without going into too much detail, if there is a non-zero entry on the diagonal then the matrix is primitive (though the converse is not true). Thus if $p_{j,j} \neq 0$ for any j, then the power method will converge to the Perron eigenvector.

One way to guarantee primitivity is to introduce a "dummy" state d with a self-loop to ensure primitivity (that is, $p_{d,d} > 0$), apply the power method, then "remove" the dummy state and correct for its presence. This can be done by adding the dummy state on the recurrence loop from the sink to the source, so $p_{n,d} = 1$, $p_{d,d} = 1/2$, and $p_{d,1} = 1/2$. Flow always passes straight through the dummy state, so any of the probability mass absorbed by the dummy state can be evenly redistributed among the other states. This change can be made to the example model's transition matrix P as follows.

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} \end{bmatrix}$$

(Note that the matrix was primitive *before* this change, since $p_{3,3} \neq 0$.) This change guarantees primitivity by making $p_{d,d} \neq 0$. The power method is then applied to yield the long-run probabilities Π. The entry for the dummy state is dropped from Π and the vector is re-normalized to obtain the correct long-run probabilities. This is actually easier than it sounds; algorithm 3 implements this approach. For the example model convergence to within 1×10^{-6} takes 46 iterations starting from the default guess.

1.4. Performing sensitivity analysis

A change in a transition probability impacts the occupancies of all states since the model is ergodic. Consider the example model and the exit arcs from state `[C]`. If one changes the probabilities of `"e"` or `"f"` to 0.9, this has a significant effect on the model. Alternately, changing the probability of `"a"` to 0.9 has a very different effect. Suppose we are attempting to validate a model against our expectations for what a "use" should look like. We discover that the model is spending far too much time in one state versus another. One way to determine how to adjust model probabilities to correct this is to use sensitivity analysis.

For each pair of state i and j in the model such that $0 < p_{i,j} < 1$, we can set $p_{i,j}$ to some value $0 < q < 1$ (adjusting the other probabilities, of course) and compute the long-run occupancies. We will use the following notation for this new

Algorithm 3 Compute the approximation to the Perron eigenvector via the power method

```
# Compute the Perron eigenvector for the row
# stochastic matrix P and return it.  The
# computation is performed using the power
# method.
#
# P: a row-stochastic matrix
# g: (optional) an initial guess
# return: list(y,step) where
#   y: an approximation of the Perron eigenvector
#   step: the number of iterations required
get_pi_approx <- function(P, g) {
   n <- nrow(P)
   d <- n + 1
   P <- rbind(cbind(P, 0), 0)
   P[n, d] = 1
   P[d, d] = 1/2
   P[d, 1] = 1/2
   P[n, 1] = 0
   if (missing(g)) {
      yold <- rep(1/d, d)
   } else {
      g <- c(g, g[n] * 2)
      yold <- g/sum(g)
   }
   y <- yold %*% P
   step <- 1
   limit <- 1e-06
   while (sum(abs(yold - y)) > limit) {
      step = step + 1
      yold = y
      y = y %*% P
   }
   y = y[1:5]
   y = y/sum(y)
   list(y, step)
}
```

vector.

$$\Pi^{(p_{i,j}=q)} = \left[\pi_k^{(p_{i,j}=q)}\right]$$

The *sensitivity* of state k with respect to the transition from state i to state j is defined as follows.

$$z_{i,j,k} = \frac{\pi_k^{(p_{i,j}=0.95)} - \pi_k^{(p_{i,j}=0.05)}}{0.90}$$

The matrix of sensitivities is computed by algorithm 4. The sensitivities for the example model are given in table 3. We interpret the sensitivities as the effect that

TABLE 3. Transition sensitivities

From	To	[Enter]	[A]	[B]	[C]	[Exit]
[A]	[B]	-0.047 25	-0.094 49	0.378 0	-0.189 0	-0.047 24
[A]	[C]	0.047 25	0.094 49	-0.378 0	0.189 0	0.047 24
[B]	[B]	-0.163 9	-0.201 7	0.680 8	-0.151 3	-0.163 9
[B]	[C]	-0.003 152	-0.037 82	-0.182 8	0.138 7	-0.003 151
[B]	[Exit]	0.080 67	0.064 54	-0.161 3	-0.064 54	0.080 67
[C]	[A]	-0.132 7	0.096 50	0.096 50	0.072 37	-0.132 7
[C]	[Exit]	0.132 7	-0.096 50	-0.096 50	-0.072 37	0.132 7

increasing the transition probability has on the long-run occupancy of each state. That is, we would expect that increasing the transition probability from state [A] to state [B] will decrease the long-run occupancy of state [C] and increase the long-run occupancy of state [B].

Suppose that one wishes to direct more of the model's flow to state [C]. Table 3 indicates that the best way to do so is to increase the probability associated with the arc from state [A] to state [C], as this is where we find the largest positive value in the column for state [C]. The best way to increase the length of a test case is to decrease the long-run occupancy of state [Exit], which can be done by increasing the probability associated with the transition from state [B] to state [B].

1.5. Computing other long run statistics

The long run occupancy of the transition from state i to state j is obtained as $\pi_i p_{i,j}$. The long run occupancy for a particular stimulus σ_k can likewise be obtained by summing the long run occupancies of the arcs labeled with the stimulus.

$$\sigma_k = \frac{1}{1-\pi_n} \sum_{i=1}^{n-1} \pi_i s_{i,k}$$

Algorithm 4 Compute the matrix of sensitivities

```
# Compute the matrix of arc sensitivities.
# The first column of the returned matrix
# is the source state, the second column is
# the target state. The remaining columns
# are the changes in the occupancies.
#
# P: the state transition matrix
# pi: the (optional) long run occupancies
# return: the sensitivities matrix
get_sensitivities <- function(P, pi) {
   n <- nrow(P)
   if (missing(pi)) {
      pi <- get_pi_approx(P)
   }
   Z <- matrix(rep(c(0), (n + 2)), nrow = 1)
   m <- 1
   for (i in 1:n) {
      for (j in 1:n) {
         if (0 < P[i, j] && P[i, j] < 1) {
            t <- P[i, ]
            x <- 1 - P[i, j]
            P[i, ] = t/x * 0.05
            P[i, j] = 0.95
            ph <- get_pi_approx(P, pi)[[1]]
            P[i, ] = t/x * 0.95
            P[i, j] = 0.05
            pl <- get_pi_approx(P, pi)[[1]]
            Z = rbind(Z, 0)
            Z[m, 1:2] = c(i, j)
            Z[m, 3:(n + 2)] = (ph - pl)/0.9
            m = m + 1
            P[i, ] = t
         }
      }
   }
   Z[1:m - 1, ]
}
```