Tutorial 3: Markov chains

In this tutorial you will learn how to compute Markov chains with Matlab.

Recall from class that a Markov chain defines a sequence of stochastic vectors $\mathbf{x_k}$, k=1, 2, 3 ,... via a stochastic matrix **P** such that

$$\mathbf{x}_{k+1} = \mathbf{P}\mathbf{x}_k, \quad k = 0, 1, 2, \ldots$$

Here is an example:

```
P = [0.6 0.3
     0.4 0.7];
xo = [0.1
      0.9];
```

You can run the Markov chain, for a specified number of steps, in a for loop.

```
disp('You started  with: ')
```

```
You started  with:
```

```
xo
```

```
xo = 2×1
    0.1000
    0.9000
```

```
% run the Markov chain
x = xo; % start with xo
for kk=1:10
    x = P*x;
end
disp('After 10 steps you ended up with: ')
```

```
After 10 steps you ended up with:
```

```
x
```

```
x = 2×1
    0.4286
    0.5714
```

Note that you can also compute $\mathbf{x}_k$ directly by raising the matrix **P** to the power *k*

$$\mathbf{x}_k = \mathbf{P}^k\mathbf{x}_0$$

In Matlab, you do this with this code:

```
x = (P^10)*xo
```

```
x = 2×1
    0.4286
    0.5714
```

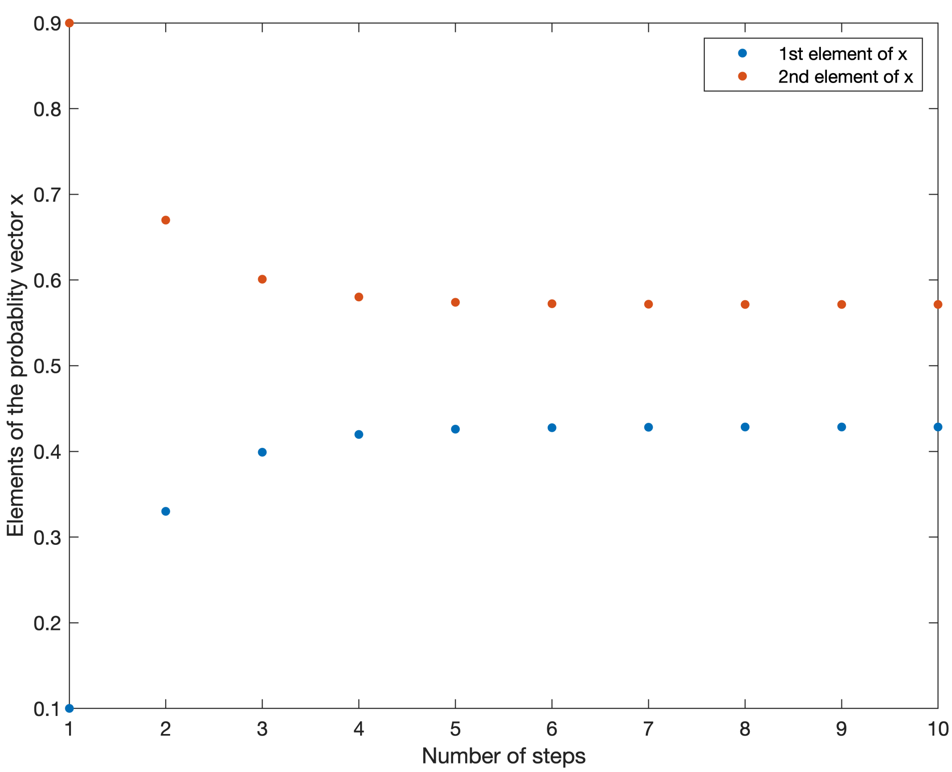
You can also save all the steps in an array with 2 rows and 100 columns.

```
X = zeros(2,10); % it is good to pre-define the array before the for loop
X(:,1) = xo; % set first column to the starting vector x
x = xo;
for kk=2:10
      x =P*x; % take a step
      X(:,kk)=x; % save the step in the array X
end
X(:,end)
```

```
ans = 2×1
    0.4286
    0.5714
```

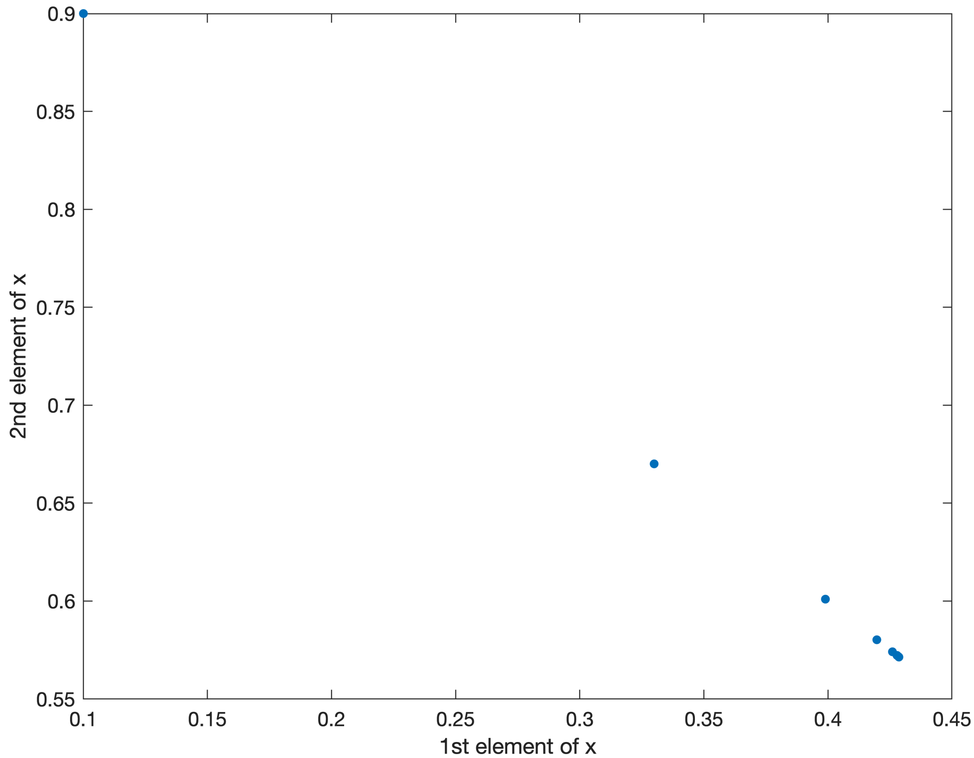
Now that you saved the steps of the Markov chain, you can also plot them:

```
figure
plot(X(1,:),'.','MarkerSize',15)
hold on, plot(X(2,:),'.','MarkerSize',15)
legend('1st element of x','2nd element of x')
xlabel('Number of steps')
ylabel('Elements of the probablity vector x')
```



You can also plot the 2nd element of **x** as a function of the 1st element

```
figure
plot(X(1,:),X(2,:),'.','MarkerSize',15)
xlabel('1st element of x')
ylabel('2nd element of x')
```



As you can see, the vector **x** does not change very much after a few steps. This means that, for sufficiently large k,

$$\mathbf{x}_{k+1} = \mathbf{P}\mathbf{x}_k \approx \mathbf{x}_k$$

Dropping the index k, this means that a steady-state vector x satisfies:

$$\mathbf{x} = \mathbf{P}\mathbf{x}$$

You can re-arrange this to the linear system

$$(\mathbf{I} - \mathbf{P})\mathbf{x} = \mathbf{0}$$

where **I** is the identity matrix.

You can thus compute the steady-state vector by solving this linear system. You know from previous tutorials that you can do that by row-reduction:

```
I = eye(2); % identity matrix
o = zeros(2,1); % a vector with zeros
rref([I-P o])
```

```
ans = 2×3
    1.0000   -0.7500         0
         0         0         0
```

Thus, the solution of $(\mathbf{I} - \mathbf{P})\mathbf{x} = \mathbf{0}$ is any scalar multiple of

```
x = [3/4
     1];
```

To make, this a probabiity vector, we need to make its elements sum to one. You can compute the sum of the elements of a vector by the command sum. Use help to find out about sum:

```
help sum
```

```
 SUM Sum of elements.
    S = SUM(X) is the sum of the elements of the vector X. If X is a matrix,
    S is a row vector with the sum over each column. For N-D arrays,
    SUM(X) operates along the first non-singleton dimension.

    S = SUM(X,'all') sums all elements of X.

    S = SUM(X,DIM) sums along the dimension DIM.

    S = SUM(X,VECDIM) operates on the dimensions specified in the vector
    VECDIM. For example, SUM(X,[1 2]) operates on the elements contained in
    the first and second dimensions of X.

    S = SUM(...,TYPE) specifies the type in which the
    sum is performed, and the type of S. Available options are:

    'double'    -  S has class double for any input X
    'native'    -  S has the same class as X
    'default'   -  If X is floating point, that is double or single,
                   S has the same class as X. If X is not floating point,
                   S has class double.

    S = SUM(...,NANFLAG) specifies how NaN (Not-A-Number) values are
    treated. The default is 'includenan':

    'includenan' - the sum of a vector containing NaN values is also NaN.
    'omitnan'    - the sum of a vector containing NaN values
                   is the sum of all its non-NaN elements. If all
                   elements are NaN, the result is 0.

    Examples:
        X = [0 1 2; 3 4 5]
        sum(X, 1)
        sum(X, 2)

        X = int8(1:20)
        sum(X)              % returns double(210), accumulates in double
        sum(X,'native')     % returns int8(127), because it accumulates in
                            % int8 but overflows and saturates.

    See also PROD, CUMSUM, DIFF, ACCUMARRAY, ISFLOAT.

    Reference page in Doc Center
       doc sum

    Other functions named sum
```

```
codistributed/sum    gpuArray/sum    tall/sum    timeseries/sum
duration/sum         sym/sum
```

You can divide all elements of **x** by its sum as follows:

```
x = x/sum(x);
x
```

```
x = 2×1
    0.4286
    0.5714
```

The result from running the Markov chain for 10 steps is quite close:

```
X(:,end)
```

```
ans = 2×1
    0.4286
    0.5714
```

Exercise

How many steps do you need to take to get the first three digits of the steady state vector? Hint: it's less than ten.

Solution:

```
nSteps = 3;
X = zeros(2,nSteps); % it is good to pre-define the array before the for loop
X(:,1) = xo; % set first column to the starting vector x
x = xo;
for kk=2:nSteps
      x =P*x; % take a step
      X(:,kk)=x; % save the step in the array X
end
X(:,end)
```

```
ans = 2×1
    0.3990
    0.6010
```

```
x
```

```
x = 2×1
    0.3990
    0.6010
```

Three steps are sufficient.