Markov-Modulated Poisson Processes

1 Introduction

A Markov-modulated Poisson Process (MMPP) is a doubly stochastic process where the intensity of a Poisson process is defined by the state of a Markov chain, see figure 1. The Markov chain can therefore be said to modulate the Poisson process, hence the name.

This modulation introduces correlations between successive interarrival times in the process. The MMPP can be identified as a special case of the Markovian arrival process (MAP).

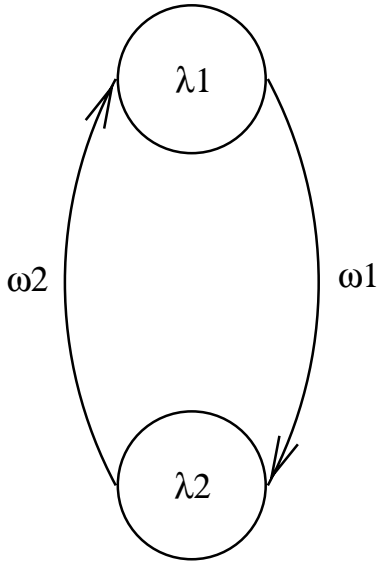

Figure 1: Two state MMPP

The main advantage that springs from using MMPPs as traffic models is that they lend them self to analysis better than some competing models. Important properties of queueing systems like MMPP/G/1 can, albeit not very easily, be derived.

An MMPP can be classified by the number of states the modulating Markov chain contains, *e.g.* a Markov chain with two states and two (different) intensities is denoted MMPP-2, or sometimes Switched Poisson process (SPP). If the two intensities are equal the model becomes an ordinary Poisson process. The special case when one intensity is zero is called Interrupted Poisson process (IPP). A useful reference for the MMPP is [7].

Much work has been done on finding methods to match the characteristics of MMPP-2s and/or IPPs to recorded data or more complex models, see for example [1, 2, 3, 4, 5]. In [6] and [8] comparisons of several of these methods were performed.

2 Some Properties

For a general MMPP we define the matrix Q to be the transition matrix of the modulating markov chain and Λ to be the matrix whose diagonal elements contains the arrival intensities that corresponds to the different states of the chain

$$\mathrm{Q} = \begin{pmatrix} -\omega_{11} & \omega_{12} & \cdots & \omega_{1m} \\ \omega_{21} & -\omega_{22} & \cdots & \omega_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{m1} & -\omega_{m2} & \cdots & \omega_{mm} \end{pmatrix},$$

$$\Lambda = \mathrm{diag}\left(\lambda_1, \lambda_2, \ldots, \lambda_m\right).$$

Several properties of the MMPP can be expressed using these definitions, a good overview is contained in [7].

An interesting feature of the MMPP is that a superposition of MMPPs is itself an MMPP. Q and Λ for the superposition can be computed from the superposed MMPPs;

$$\mathrm{Q} = \mathrm{Q}_1 \oplus \mathrm{Q}_2 \oplus \ldots \oplus \mathrm{Q}_n,$$

$$\Lambda = \Lambda_1 \oplus \Lambda_2 \oplus \ldots \oplus \Lambda_n.$$

Where $\oplus$ is the Kronecker sum, defined as

$$\mathrm{A} \oplus \mathrm{B} = (\mathrm{A} \otimes \mathrm{I_B}) + (\mathrm{I_A} \otimes \mathrm{B}),$$

where $\otimes$ is the Kronecker product

$$\mathrm{C} \otimes \mathrm{D} = \begin{pmatrix} c_{11}\mathrm{D} & c_{12}\mathrm{D} & \cdots & c_{1m}\mathrm{D} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1}\mathrm{D} & c_{n2}\mathrm{D} & \cdots & c_{nm}\mathrm{D} \end{pmatrix},$$

and $\mathrm{I_X}$ denotes an identity matrix of the same size as X.

We now turn to the special case of the two-state MMPP, as defined in figure 1, and give some specific results. Let N_t denote the number of arrivals in an interval of length t. For a stationary MMPP-2 the mean of N_t is easily found to be

$$\mathrm{E}(N_t) = \frac{\lambda_1\omega_2 + \lambda_2\omega_1}{\omega_1 + \omega_2} t.$$

According to [2] the index of dispersion for counts becomes

$$\mathrm{IDC}(t) = \frac{\mathrm{V}(N_t)}{\mathrm{E}(N_t)} = 1 + \frac{2(\lambda_1 - \lambda_2)^2\omega_1\omega_2}{(\omega_1 + \omega_2)^2(\lambda_1\omega_2 + \lambda_2\omega_1)} - \frac{2(\lambda_1 - \lambda_2)^2\omega_1\omega_2}{(\omega_1 + \omega_2)^3(\lambda_1\omega_2 + \lambda_2\omega_1)t}(1 - e^{-(\omega_1+\omega_2)t}).$$

Interesting here is to note the convergence of the IDC to a fix value as t goes to infinity

$$\mathrm{IDC}(\infty) = \lim_{t->\infty} \frac{\mathrm{V}(N_t)}{\mathrm{E}(N_t)} = 1 + \frac{2(\lambda_1 - \lambda_2)^2\omega_1\omega_2}{(\omega_1 + \omega_2)^2(\lambda_1\omega_2 + \lambda_2\omega_1)}.$$

3 Simulation of MMPP-2s

The following two programs shows how to generate “traffic” from an MMPP-2. The first generates successive interarrival times and the second generates the counting process corresponding to the number of arrivals in successive intervals.

3.1 Interarrivals

```
//
//  Generator for interarrival-times of MMPP-2
//  (C) Per Karlsson 1998
//
//  Usage: mmppigen l_1 l_2 w_1 w_2 seed arrivals
//         l_i is arrival intensity in state i
//         w_i is transition intensity from state i
//

#include <math.h>
#include <iostream.h>
#include <stdlib.h>

// Debug info
//#define DE

//Random generator state (seed)
unsigned long int z = 1;

//Parameters
double lambda[2];
double omega[2];
unsigned int arrivalsToSimulate;

//Times
double timeToTransition;
double timeToArrival;
double timeSinceLastArrival;
#ifdef DE
double T=0;
#endif

//State
int state;

//Controls
unsigned long int arrivals;

//Events
void transition();
void arrival();
```

```
//Helpers
void initialize(char* argv[]);
void reset();
void adjustTimes( double aTime );
double myrand();
double negExp( double lambda );

main(int argc, char* argv[])
{
  if( argc < 7 ) {
    cerr << "Usage: mmppigen l_1 l_2 w_1 w_2 seed arrivals" << endl;
    exit(0);
  }

  reset();
  initialize(argv);

  while( arrivals<arrivalsToSimulate ) {

    if( timeToArrival < timeToTransition )
      arrival();
    else
      transition();

  }
}


//Arrival event
void arrival()
{
  adjustTimes( timeToArrival );
  cout << timeSinceLastArrival << endl;
  timeSinceLastArrival = 0;
  timeToArrival = negExp( lambda[state] );
  arrivals++;

#ifdef DE
  cerr << T << " arrival" << endl;
#endif

}

//Transition event
void transition()
{
  adjustTimes( timeToTransition );
  state = !state;
  timeToTransition = negExp( omega[state] );
```

```
#ifdef DE
  cerr << T << " transition: " << !state << "->" << state << endl;
#endif

}


//Initialize parameters, times, and state
void initialize(char* argv[])
{
  //Get parameters
  lambda[0] = atof(argv[1]);
  lambda[1] = atof(argv[2]);
  omega[0]  = atof(argv[3]);
  omega[1]  = atof(argv[4]);
  z         = atoi(argv[5]);
  arrivalsToSimulate = atoi(argv[6]);

  //Determine start state (according to state probabilities)
  state = !(myrand() < omega[1]/(omega[0]+omega[1]));

  //Initialize times
  timeToArrival = negExp( lambda[state] );
  timeToTransition = negExp( omega[state] );

  //Initialize cout
  cout.setf( ios::fixed );
  cout.precision( 6 );
}

//Reset some variables
void reset()
{
  arrivals = 0;
  timeSinceLastArrival = 0;
}

//Update all times
void adjustTimes( double aTime )
{
  timeToArrival        -= aTime;
  timeToTransition     -= aTime;
  timeSinceLastArrival += aTime;
#ifdef DE
  T                    += aTime;
#endif
}

//Returns uniformly distributed numbers in [0..1]
double myrand()
```

```
{
  const a = 16807;
  const m = 2147483647;
  z = a*z % m;
  return (double)z/m;
}

//Returns exponentially distributed numbers with intensity lambda
double negExp( double lambda )
{
  return -1/lambda*log( myrand() );
  //(Inverse of F(t)=exp(-lambda*t))
}
```

3.2 Counts

```
//
//  Generator for counts of MMPP-2
//  (C) Per Karlsson 1998
//
//  Usage: mmppagen l_1 l_2 w_1 w_2 seed slot intervals
//         l_i is arrival intensity in state i
//         w_i is transition intensity from state i
//         slot is

#include <math.h>
#include <iostream.h>
#include <stdlib.h>

// Debug info
//#define DE

//Random generator state (seed)
unsigned long int z = 1;

//Parameters
double lambda[2];
double omega[2];
double slot;
unsigned long int intervalsToSimulate;

//Times
double timeToTransition;
double timeToArrival;
double timeToIntervalEnd;
#ifdef DE
double T=0;
#endif

//State
```

```
int state;

//Controls
unsigned long int intervals;
unsigned long int arrivals;


//Events
void transition();
void arrival();
void intervalEnd();

//Helpers
void initialize(char* argv[]);
void reset();
void adjustTimes( double aTime );
double myrand();
double negExp( double lambda );

main(int argc, char* argv[])
{
  if( argc < 8 ) {
    cerr << "Usage: mmppagen  l_1 l_2 w_1 w_2 seed slot intervals" << endl;
    exit(0);
  }

  reset();
  initialize(argv);

  while( intervals<intervalsToSimulate ) {

    if( timeToArrival < timeToTransition &&
timeToArrival < timeToIntervalEnd )
      arrival();

    if( timeToTransition < timeToArrival &&
timeToTransition < timeToIntervalEnd )
      transition();

    if( timeToIntervalEnd < timeToArrival &&
timeToIntervalEnd < timeToTransition )
      intervalEnd();

  }
}


//Arrival event
void arrival()
{
```

```
  adjustTimes( timeToArrival );
  timeToArrival = negExp( lambda[state] );
  arrivals++;

#ifdef DE
  cerr << T << " arrival" << endl;
#endif

}

//Transition event
void transition()
{
  adjustTimes( timeToTransition );
  state = !state;
  timeToTransition = negExp( omega[state] );

#ifdef DE
  cerr << T << " transition: " << !state << "->" << state << endl;
#endif

}

//Interval end event
void intervalEnd()
{
  adjustTimes( timeToIntervalEnd );
  cout << arrivals << endl;
  arrivals = 0;
  timeToIntervalEnd = slot;
  intervals++;

#ifdef DE
  cerr << T << " interval end" << endl;
#endif

}

//Initialize parameters, times, and state
void initialize(char* argv[])
{
  //Get parameters
  lambda[0] = atof(argv[1]);
  lambda[1] = atof(argv[2]);
  omega[0]  = atof(argv[3]);
  omega[1]  = atof(argv[4]);
  z         = atoi(argv[5]);
  slot      = atof(argv[6]);
  intervalsToSimulate = atoi(argv[7]);
```

```
  //Determine start state (according to state probabilities)
  state = !(myrand() < omega[1]/(omega[0]+omega[1]));

  //Initialize times
  timeToArrival = negExp( lambda[state] );
  timeToTransition = negExp( omega[state] );
  timeToIntervalEnd = slot;
}

//Reset some variables
void reset()
{
  arrivals = 0;
  intervals = 0;
}

//Update all times
void adjustTimes( double aTime )
{
  timeToArrival        -= aTime;
  timeToTransition     -= aTime;
  timeToIntervalEnd    -= aTime;
#ifdef DE
  T                    += aTime;
#endif
}

//Returns uniformly distributed numbers in [0..1]
double myrand()
{
  const a = 16807;
  const m = 2147483647;
  z = a*z % m;
  return (double)z/m;
}

//Returns exponentially distributed numbers with intensity lambda
double negExp( double lambda )
{
  return -1/lambda*log( myrand() );
  //(Inverse of F(t)=exp(-lambda*t))
}
```

4 Examples

In figure 2 and figure 3 example realizations of MMPP-2 “traffic” can be found. The traces are produced with the programs in section 3. The following parameters were used in the production of both traces: $\lambda_1 = 10$, $\lambda_2 = 1$, $\omega_1 = 0.1$, $\omega_2 = 0.1$, *seed* $= 1$.

Figure 2 show 1000 successive interarrival times. Figure 3 show the number of arrivals in 1000 intervals, each of length 1.

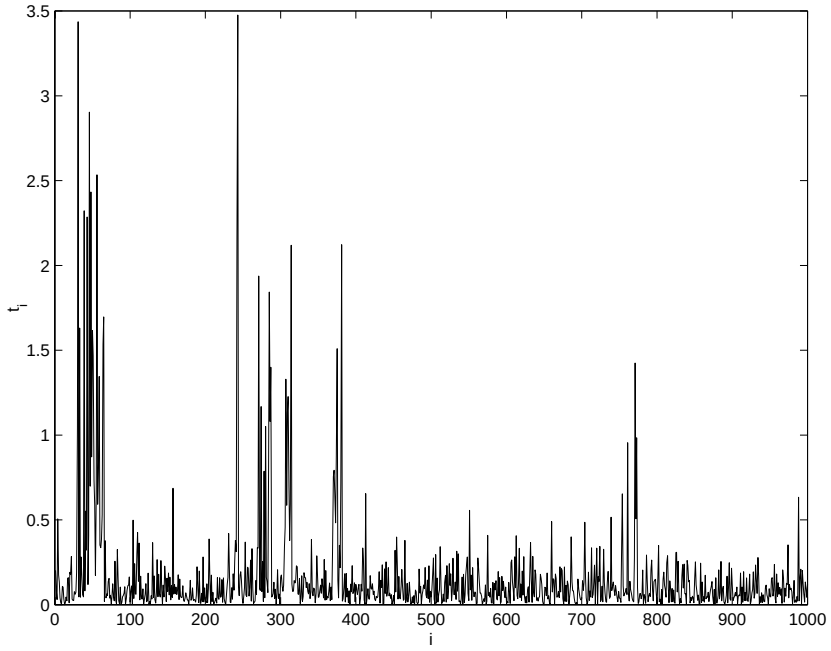

Figure 2: Example of interarrivals for an MMPP

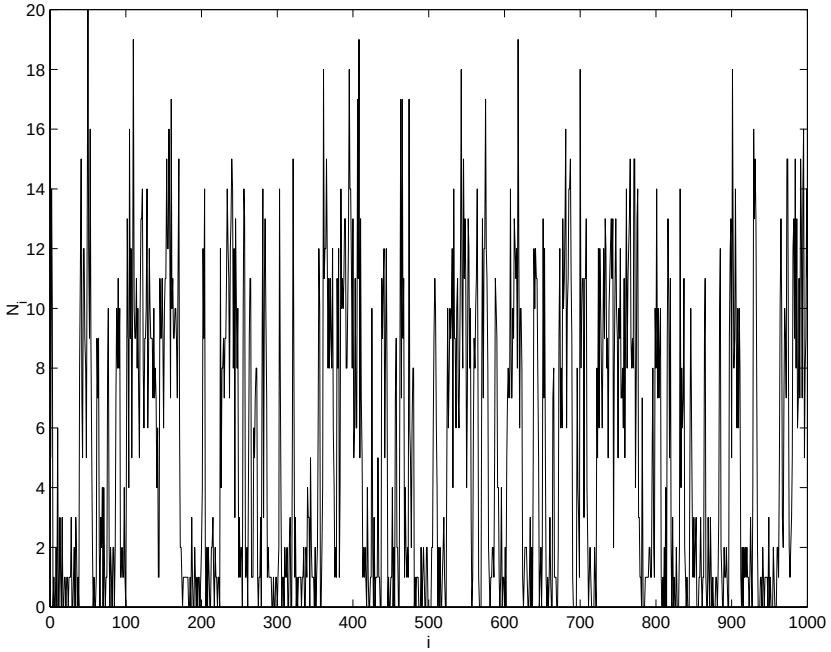

Figure 3: Example of counting process for an MMPP

References

[1] Gusella R., Characterizing the Variability of Arrival Processes with Indexes of Dispersion, IEEE J. Selected Areas in Communications, vol. 9, no. 2, 1991, pp. 203-211.

[2] Heffes H. and Lucantoni D., A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance, IEEE J. Selected Areas in Communications, vol. 4, no. 6, 1986, pp. 856-868.

[3] Rossiter M.H., A Switched Poisson Model for Data Traffic, Australian Telecommunication Research, vol. 21, no. 1, 1987, pp. 53-57.

[4] Okuda T., Akimaru H., and Sakai M., A simplified Performance Evaluation for Packetized Voice Systems, Transactions of the IEICE, vol. E73, no. 6, 1990.

[5] Meier-Hellstern K., A fitting algorithm for Markov-modulated Poisson processes having two arrival rates, European Journal of Operational Research, vol. 29, no. 3, 1987, pp. 370-377.

[6] Arvidsson Å. and Harris R., Analysis of the Accuracy of Bursty Traffic Models, Proc. First International Conference On Telecommunication System Modeling and Analysis, Nashville, Tennessee, USA 1993, pp. 206-211.

[7] Meier-Hellstern K.S. and Fischer W., The Markov-modulated Poisson process (MMPP) cookbook, Performance Evaluation 18, 1992, pp. 149-171.

[8] Arvidsson Å. and Lind C., Using Markovian Models to replicate Real ATM traffics, Performance Modelling and Evaluation of ATM Networks, vol. 2, (Kouvatsos ed.), Chapman & Hall, London, England, 1996.