PROJECT REPORT
On

Uncertainty Quantification using Polynomial Chaos in Spatial Population Dynamics

In partial fulfillment of the requirement for degree of
MASTER OF SCIENCE
In
ENGINEERING MATHEMATICS

Submitted by

Kshitij A. Patil
(17MAT209)

Supervisor

Dr. Amiya R. Bhowmick
Department of Mathematics

Institute of Chemical Technology
Nathalal Parekh Marg, Matunga, Mumbai-19

2017-2019

INSTITUTE OF CHEMICAL TECHNOLOGY
(university under section-3 of UGC Act 1956)
(**formerly UDCT/UICT, Mumbai**)

Elite Status & Center of Excellence - Government of Maharashtra
Matunga, Mumbai - 400019, India.

CERTIFICATE

This is to certify that the work contained in this project report entitled "**Uncertainty Quantification using Polynomial Chaos in Spatial Population Dynamics**" submitted by **Mr. Kshitij A. Patil (Roll No.: 17MAT209)** to the Institute of Chemical Technology, Matunga towards partial requirement of Master of Science in Engineering Mathematics has been carried by him under my supervision.

Dr. Amiya R. Bhowmick
Project Guide

INSTITUTE OF CHEMICAL TECHNOLOGY
(university under section-3 of UGC Act 1956)
(**formerly UDCT/UICT, Mumbai**)

Elite Status & Center of Excellence - Government of Maharashtra
Matunga, Mumbai - 400019, India.

Approval of Project

This is to certify that the work contained in this project report entitled "**Uncertainty Quantification using Polynomial Chaos in Spatial Population Dynamics**" submitted by **Mr. Kshitij A. Patil (Roll No.: 17MAT209)** to the Institute of Chemical Technology, Matunga towards partial requirement of Master of Science in Engineering Mathematics has been carried by him under my supervision.

Head : Dr. Ajit Kumar

Guide : Dr. Amiya R. Bhowmick

External Examiner:

Date:

Acknowledgements

I thank the Institute of Chemical Technology and the Department of Mathematics for conducting this versatile course, providing more than adequate facilities for us to conduct our work and a nice environment to study in. All faculty and staff have been very helpful and approachable and have helped create a home like feeling.

I am grateful to my guide Dr. Amiya R. Bhowmick for his guidance, motivation, enthusiasm and taking out the time to have discussions about any matter, even at the oddest hours. I would like to extend my gratitude to Dr. V Divya and Prof. A. Sahu, who helped me with certain numerical studies and analysis.

None of my plans would have worked out without the support and encouragement of my mother Mrs. Vaishali A. Patil, my father Dr. Anand Patil, Mr. Jainam Jain, Mr. Madhur Sethi, Ms. Simrat Otaal and Mr. Aniket Wakankar. It is needless to say that, my family and classmates have been anything short of wonderful to be around. A special thanks to Bhadur Mama and Kumar uncle for helping me with any type of task and making things easier to do. A special thanks to Dr. Nitin P. Gulhane of VJTI, without whom my master's would be incomplete and for his continued support in all my endeavours. Finally, I am indebted to Mr. Nilesh Jain, who helped me become a confident person and taught me the basic concepts of mathematics.

Mr. Kshitij A. Patil

Abstract

Different mathematical models mainly governed by Ordinary Differential Equations (for overlapping generation) or Discrete Difference Equation (for non-overlapping generation) have been proposed in the literature to describe evolution of population size over time. The spatial evolution of the population is usually described by reaction-diffusion equation. In deterministic system, all the model parameters are assumed to be fixed quantities. However, in natural populations, the parameters may vary continuously or randomly. The stochasticity in the parameters is not usually considered in the standard population dynamic studies. In this work, we consider a population growth equation governed by logistic equation in a two dimensional spatial domain. We investigate the evolution of the population in both space and time by treating one parameter to vary randomly following some known distribution. For simulation study, we consider the population growing following the logistic growth equation. We assumed the intrinsic growth rate to vary stochastically following a known distribution. Using the method of polynomial chaos expansion, we solve the reaction diffusion equation numerically. Polynomial chaos expansion is a technique where an unknown random variable is written as a function of known random variables. This method helps to quantify the uncertainty in the population size with the help of orthogonal polynomials. All the computations have been carried out in R and all functions have been hard coded by us. Options for various orthogonal polynomials have been included.

Keywords: *Spatial Population Dynamics, Polynomial Chaos Expansion, Stochastic Differential Equations*

Contents

List of Figures

Chapter 1

Introduction

Population dynamics in natural systems are usually modelled by differential equations. One of the most important mathematical models is given by :

$$\frac{du}{dt} = ru\left(1 - \left(\frac{u}{K}\right)^{\theta}\right), u(0) = u_0$$

This model describes the population growth u over time (temporally) in a deterministic environment. However, natural populations grow over spatial regions subject to stochasticity due to various reasons or sources.
Generally, the existing models make room for stochasticity due to environmental conditions or demographic variations. Typically general modelling approach gives a diffusion equation given by :

$$du = ru\left(1 - (\frac{u}{K})^{\theta}\right)dt + \sigma(u)dw_t,$$

where the red portion is the deterministic part and the blue portion is the stochastic part of the model. In essence, the red portion implies that the existing model parameters are assumed to be constants and the blue portion implies that the parameters may vary randomly due to some unknown sources.

We plan to model the uncertainty in the population size $u(x, y, t)$'s,
distribution by studying it's mean and variance in a spatial domain over time, where the uncertainty is introduced due to randomly varying model parameter.

Chapter 2

Problem Statement

We consider a population growing logistically in a spatial domain governed by :

$$\frac{\partial}{\partial t}u(x,y,t) = u(x,y,t)r\left(1-\left(\frac{u(x,y,t)}{K}\right)^{\theta}\right) + d\left(\frac{\partial^2}{\partial x^2}u(x,y,t) + \frac{\partial^2}{\partial y^2}u(x,y,t)\right), \tag{2.1}$$

where

- $u(x,y,t)$ is the population density.
- K is the maximum carrying capacity of the system.
- d is the diffusion coefficient.
- r is the intrinsic growth rate, which we have taken to be a random variable following a known distribution. This makes the population density $u(x,y,t)$ a random variable.
- We have built and solved the system for $\theta = 1$.

Goal : To solve the equation 2.1 for $u(x,y,t) := u$. Essentially to characterise the distribution of u at different times.

Chapter 3

Orthogonal Polynomials

A sequence of polynomials $\{P_n(x)\}_{n=0}^{\infty}$ where $P_i(x)$ is an i^{th} degree polynomial $\forall i$ is an orthogonal set with respect a weight function $w(x)$ on support $D = (a, b)$ if:

$$\int_D P_n(x)P_m(x)w(x)dx = h_n\delta_{nm}, n, m \in \mathbb{N}$$

, where $h_i s$ are non-zero constants and δ_{nm} is the Kronecker Delta function.
We use the weight function $w(x)$ to define an inner product of two polynomials $f(x), g(x)$ denoted by $< f(x), g(x) >_W$:

$$< f(x), g(x) >_W = \int_D f(x)g(x)W(x)dx$$

Now, if $w(X)$ is the probability density function (pdf) of some random variable X and $w(X)$ is also the weight function for an orthogonal set $P_n(x)$, then $< P_n(X), P_m(X) >_w = h_n\delta_{nm}$. The inputs of inner product are polynomial functions of random variable X who has pdf $w(X)$, which also is the weight function of a set of the orthogonal polynomials $P_n(X)$.
We tabulate below the commonly known orthogonal polynomials, which we have used for our work.[1]

Name	$P_n(x)$	Distribution	$w(x)$	D
Legendre	$\frac{(-1)^n}{2^n n!}\frac{d^n}{dx^n}(1-x^2)^n$	Uniform	$1/2$	$(-1, 1)$
Hermite	$\frac{(-1)^n}{w(x)}\frac{d^n}{dx^n}w(x)$	Normal	$e^{-x^2/2}$	$(-\infty, \infty)$
Laguerre	$\frac{1}{n!w(x)}\frac{d^n}{dx^n}(w(x)x^n)$	Gamma	$x^\alpha e^{-x}$	$(0, \infty)$

Chapter 4

Method 1: Polynomial Chaos Expansion (PCE)

Idea: We express an unknown or complicated random variable Y following and unknown distribution F_Y as a function of known random variable X. That is:

$$Y = c_0 + c_1\psi_1(X) + c_2\psi_2(X) + \cdots$$

- We now have to find the unknown deterministic constants $\{C_i\}_{i=0}^{\infty}$.
- If the functions $\{\psi_i(X)\}_{i=0}^{\infty}$ are orthogonal polynomials, we have a computational advantage.

This is the basic idea of Polynomial Chaos Expansion, to express an unknown random variable Y as a linear combination of orthogonal polynomials of a known random variable X.
We consider a known random variable X, following uniform distribution in this case and approximate the density of it's function $cos(X)$ using PCE. [9]

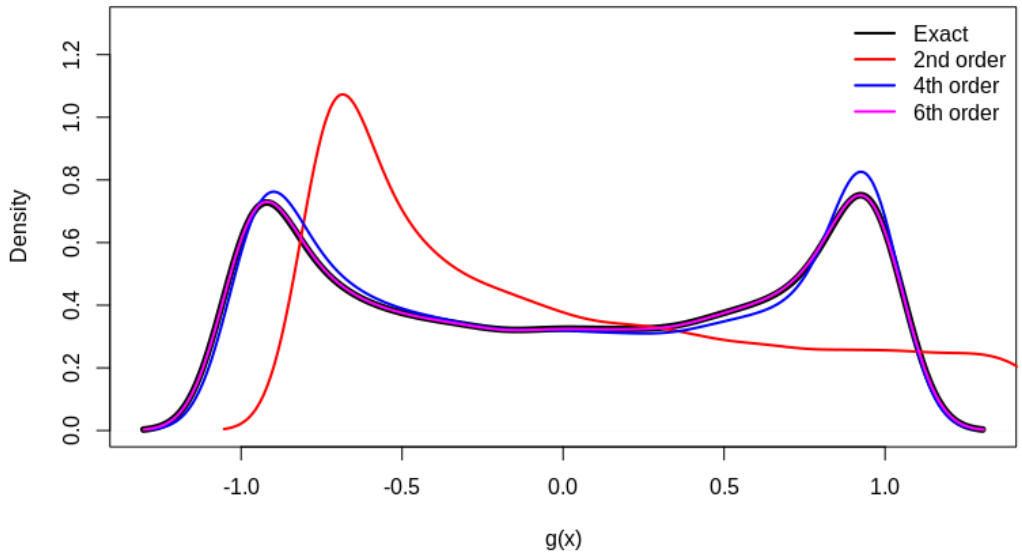

Figure 4.1: PCE Approximation of cos function of random variable X

Chapter 5

Method 2: Alternating Direction Implicit Method (ADI)

The Alternating Direction Implicit (ADI) method is an efficient method for solving parabolic differential equations having more than one variable. The method splits the differential equation into two directions with the help of finite difference schemes. It can be thought of as splitting one time step into two half time steps. We move in one direction in the first half time step, and then in the next direction in the second half time step. If the directions are names "x" and "y", then we wish to discretise the derivatives in the x direction at (i,j,n+1/2) and the derivatives in the y direction at (i,j,n+1).
This can be best summarised by the picture below:

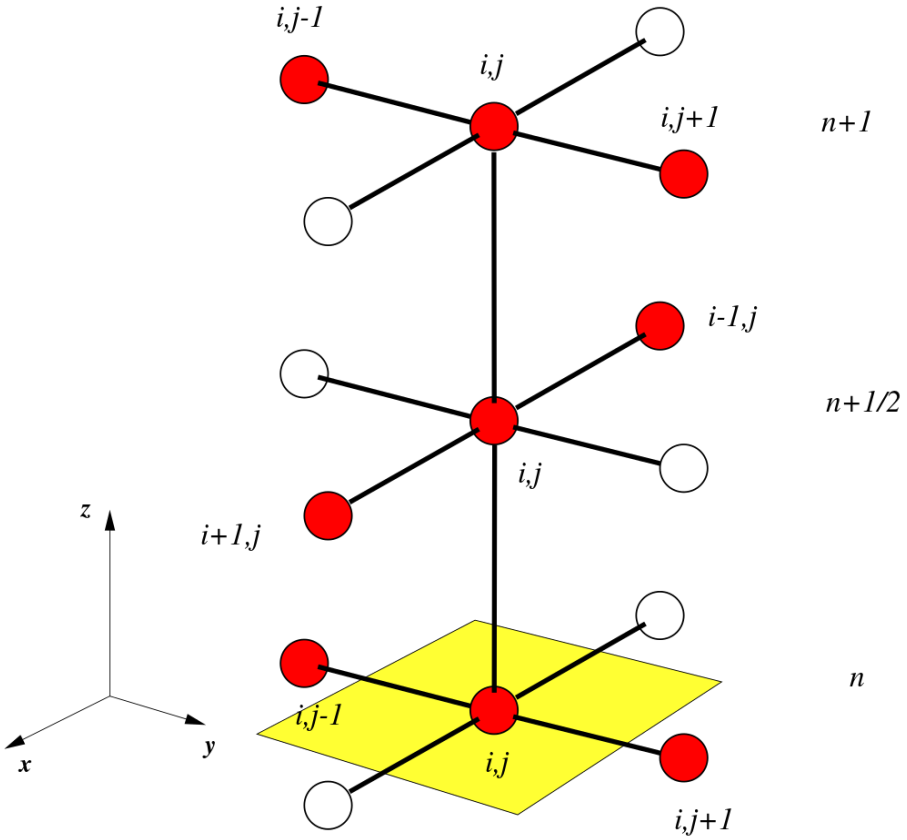

Figure 5.1: ADI Method Illustration[10]

Chapter 6

Formulation of the solution

We express the population density:

$$u(x, y, t) = \sum_{i=0}^{\infty} u_i \psi_i(\xi) \tag{6.1}$$

The equation 6.1 is the Polynomial chaos expansion(PCE) of u.

- ξ is a vector of orthonormal random variables, $\psi(\xi)$ are known orthogonal polynomials. These orthogonal polynomials have a weight function that is precisely the density function of ξ.
- The choice of ξ and ψ is free and can be chosen by the type of system.
- As it is not feasible to actually consider an infinite series, we truncate the expansion to P+1 terms with

  $$P + 1 = \frac{(N + K)!}{(N!)(K!)}$$

  where N is the number of random variables in the vector and K is the highest degree of the polynomial.[5]

The truncated PCE of u is:

$$u(x, y, t) = \sum_{i=0}^{P} u_i \psi_i(\xi) \tag{6.2}$$

Substituting equation 6.2 in equation 2.1 we have:

$$\frac{\partial}{\partial t} \sum_{i=0}^{P} u_i \psi_i(\xi) = \xi \sum_{i=0}^{P} u_i \psi_i(\xi) \left(1 - \left(\frac{\sum_{i=0}^{P} u_i \psi_i(\xi)}{K}\right)\right) + d\nabla^2 \sum_{i=0}^{P} u_i \psi_i(\xi) \tag{6.3}$$

- r is the intrinsic growth rate which is a random variable ξ following Uniform distribution in $[-1, 1]$, and so the polynomials $\psi_i(\xi)$ are the set of Legendre polynomials.
- We denote $\psi_i(\xi) := \psi_i$.
- ∇^2 is the Laplacian operator.

We take inner product with $\psi_k, \forall k = 0, 1, \cdots, P$ in equation 6.3. By exploiting orthogonality we obtain a simpler expression:

$$< \psi_k^2, 1 > (u_k)_t = \sum_{l=0}^{P} (u_i) < \psi_i(\xi), \psi_k > -\frac{1}{K} \sum_{l=0}^{P} (u_i)^2 < \psi_i^2(\xi), \psi_k > - \tag{6.4}$$
$$\frac{2}{K} \sum_{m=1}^{P} \sum_{l=0}^{m-1} (u_l u_m) < \psi_l \psi_m(\xi), \psi_k > + d\left((u_k)_{xx} + (u_k)_{yy}\right) < \psi_k^2, 1 >, k = 0, 1, \cdots, P$$

where $(u_k)_t = \frac{\partial}{\partial t} u(x, y, t, \omega), (u_k)_{xx} = \frac{\partial^2}{\partial x^2} u(x, y, t, \omega), (u_k)_{yy} = \frac{\partial^2}{\partial y^2} u(x, y, t, \omega)$

- A system has been developed for $\theta = 1$ using the ADI method.
- The partial derivatives are approximated using difference schemes.
- We have employed appropriate linearisation techniques.
- There are two systems for each time step.
- The mean and variance of the population size are [5]:

$$E[U(x, y, t)] = U_0$$

$$Var(U(x, y, t)) = \sum_{i=1}^{P} U_K^2 < \psi_K, \psi_K >$$

- We choose grid spacing $\Delta x, \Delta y$ in x and y directions and Δt time spacing. We discretise time derivative using forward differencing and the Laplacian using central differencing scheme.

$$\frac{\partial}{\partial t} u_{(i,j)}^{(n+1)} = \frac{u_{(i,j)}^{(n+1)} - u_{(i,j)}^{(n)}}{\Delta t/2}, \ \textit{Time Derivative Approximation}$$

$$\frac{\partial^2}{\partial x^2} u_{(i,j)}^{(n)} = \frac{u_{(i-1,j)}^{(n)} - 2u_{(i,j)}^{(n)} + u_{(i-1,j)}^{(n)}}{(\Delta x)^2}, \ \textit{Space Derivative Approximation}$$

- In the first direction, we move in x direction keeping y same in first half time step $(n, n + \frac{1}{2})$.
- In the next direction along y keeping x constant in time interval $(n + \frac{1}{2}, n + 1)$.
- The linearisation has been done in the following way :
  $(u_l) = ((u_l)^{(n+1/2)} + (u_l)^{(n)})/2$
  $(u_l)^2 = ((u_l)^{(n+1/2)}(u_l)^{(n)})$

- On substituting the PCE in both directions and rearranging terms, we get the following two systems. The first system gives one column of values of $U_k^{(n+0.5)}$ and the second a row of $U_k^{(n+1)}$.
- We have

  $$A1_{k,j} U_{k,j}^{(n+0.5)} = R1_{k,j}^{(n)}$$

  for the first direction and

  $$A2_{k,i} U_{k,i}^{(n+1)} = R2_{k,i}^{T\ (n+0.5)}$$

  for each $k = 0, 1, 2, ..., P; n = 0, 1, ..., N - 1$, which is a system of P+1 equations each and j indexes the columns while i indexes the rows.
- $A1_{k,j} and A2_{k,i}$ are matrices of order $(N_x + 1)x(N_x + 1)$ and $(N_y + 1)x(N_y + 1)$ respectively, where N_x and N_y are the number of intervals x and y space directions are discretised in respectively.

Chapter 7

Terms and Coefficients

The matrix of of coefficients on the left hand side for the first half time step is :

$$A1_k = \begin{pmatrix} b_{k(0,j)} & a_0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ a_0 & b_{k(1,j)} & a_0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & a_0 & b_{k(3,j)} & a_0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_0 & b_{k(N_x-2,j)} & a_0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & a_0 & b_{k(N_x-1,j)} & a_0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & a_0 & b_{k(N_x,j)} \end{pmatrix}$$

The matrix of of coefficients on the left hand side for the second half time step is :

$$A2_k = \begin{pmatrix} -f_{k(i,0)} & c_0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ c_0 & -f_{k(i,1)} & c_0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & c_0 & -f_{k(i,2)} & c_0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & c_0 & -f_{k(i,N_y-2)} & c_0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & c_0 & -f_{k(i,N_y)} & c_0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & c_0 & -f_{k(i,N_y)} \end{pmatrix}$$

The vector of unknowns in the first half time step is:

$$U_{k,j}^{(n+0.5)} = \begin{pmatrix} u_{k(0,j)}^{(n+1/2)} \\ u_{k(1,j)}^{(n+1/2)} \\ u_{k(2,j)}^{(n+1/2)} \\ \\ \vdots \\ u_{k(N_x-2,j)}^{(n+1/2)} \\ u_{k(N_x-1,j)}^{(n+1/2)} \\ u_{k(N_x,j)}^{(n+1/2)} \end{pmatrix}$$

The vector of unknowns in the second half time step is:

$$U_{k,i}^{(n+1)} = \begin{pmatrix} u_{k(i,0)}^{(n+1)} \\ u_{k(i,1)}^{(n+1)} \\ u_{k(i,2)}^{(n+1)} \\ \\ \vdots \\ u_{k(i,N-2)}^{(n+1)} \\ u_{k(i,N_y-1)}^{(n+1)} \\ u_{k(i,N_y)}^{(n+1)} \end{pmatrix}$$

The RHS vector for the first half time step is:

$$R1_{k,j} = \begin{pmatrix} rhs1_{0_{(0,j)}} \\ rhs1_{1_{(1,j)}} \\ rhs1_{2_{(2,j)}} \\ \\ \vdots \\ rhs1_{N_{x-2}(N_x,j)} \\ rhs1_{N_{x-1}(N_x,j)} \\ rhs1_{N_{x}(N_x,j)} \end{pmatrix}$$

The RHS vector for the second half time step is:

$$R2^T_{k,i} = \begin{pmatrix} rhs2_{0_{(i,0)}} \\ rhs2_{1_{(i,1)}} \\ rhs2_{2_{(i,2)}} \\ \\ \vdots \\ rhs2_{N_{y-2}(i,N_y-2)} \\ rhs2_{N_{y-1}(i,N_y-1)} \\ rhs2_{N_{y}(i,N_y)} \end{pmatrix}$$

The coefficients in the above matrices are defined below:

- $a_k = \frac{-d}{(\Delta x)^2}$
- $b_{k_{(i,j)}} = \frac{2<\psi_k^2,1>}{\Delta t} - \frac{<\psi_k(\xi),1\psi_k>}{2} + \frac{(u_k)^n_{(i,j)}}{K} < \psi_k^2(\xi),\psi_k > + \frac{1}{K}\sum_{l=0,l\neq k}^{P}(u_l)^n_{(i,j)} < \psi_l\psi_k(\xi),\psi_k > + \frac{2d}{(\Delta x)^2} < \psi_k^2, 1 >$
- $c_k = \frac{d}{(\Delta y)^2} < \psi_k^2, 1 >$
- $d_{k_{(i,j)}} = \frac{2<\psi_k^2,1>}{\Delta t} + \frac{<\psi_k(\xi),\psi_k>}{2} - \frac{1}{K}\sum_{l=0,l\neq k}^{P}(u_l)^n_{(i,j)} < \psi_l\psi_k(\xi),\psi_k > - \frac{2d<\psi_k^2,1>}{(\Delta y)^2}$
- $e_{k_{(i,j)}} = \sum_{l=0,l\neq k}^{P}(u_l)^n_{(i,j)} < \psi_l(\xi),\psi_k > - \sum_{l=0,l\neq k}^{P}(u_l^2)^n_{(i,j)} < \psi_l^2(\xi),\psi_k > - \frac{2}{K}\sum_{m=1,m\neq k}^{P}\sum_{l=0,l\neq k}^{m-1}(u_l u_m)^n_{(i,j)} < \psi_l\psi_m(\xi),\psi_k >$
- $f_{k_{(i,j)}} = \frac{2<\psi_k^2,1>}{\Delta t} - \frac{<\psi_k(\xi),\psi_k>}{2} + \frac{(u_k)^{(n+1/2)}_{(i,j)}}{K} < \psi_k^2(\xi),\psi_k > + \frac{1}{K}\sum_{l=0,l\neq k}^{P}(u_l)^{(n+1/2)}_{(i,j)} < \psi_l\psi_k(\xi),\psi_k > + \frac{2d}{(\Delta y)^2} < \psi_k^2, 1 >$
- $g_{k_{(i,j)}} = \frac{2<\psi_k^2,1>}{\Delta t} + \frac{<\psi_k(\xi),\psi_k>}{2} - \frac{1}{K}\sum_{l=0,l\neq k}^{P}(u_l)^{(n+1/2)}_{(i,j)} < \psi_l\psi_k(\xi),\psi_k > - \frac{2d<\psi_k^2,1>}{(\Delta x)^2}$
- $h_{k_{(i,j)}} = \sum_{l=0,l\neq k}^{P}(u_l)^{(n)+1/2}_{(i,j)} < \psi_l(\xi),\psi_k > - \sum_{l=0,l\neq k}^{P}(u_l^2)^n_{(i,j)} < \psi_l^2(\xi),\psi_k > - \frac{2}{K}\sum_{m=1,m\neq k}^{P}\sum_{l=0,l\neq k}^{m-1}(u_l u_m)^n_{(i,j)} < \psi_l\psi_m(\xi),\psi_k >$
- $rhs1_{k_{(i,j)}} = c_k(u_k)^{(n)}_{(i,j-1)} + d_{k_{(i,j)}}(u_k)^{(n)}_{(i,j)} + c_k(u_k)^{(n+1/2)}_{(i,j+1)} + e_{k_{(i,j)}}$
- $rhs2_{k_{(i,j)}} = a_k(u_k)^{(n+1/2)}_{(i-1,j)} - g_{k_{(i,j)}}(u_k)^{(n+1/2)}_{(i,j)} + a_k(u_k)^{(n+1/2)}_{(i+1,j)} - h_{k_{(i,j)}}$

Chapter 8

Algorithm Flowcharts

We list all the algorithm flowcharts of the codes that we have written in R.

1. Flowchart of function that multiplies two function structures in R.

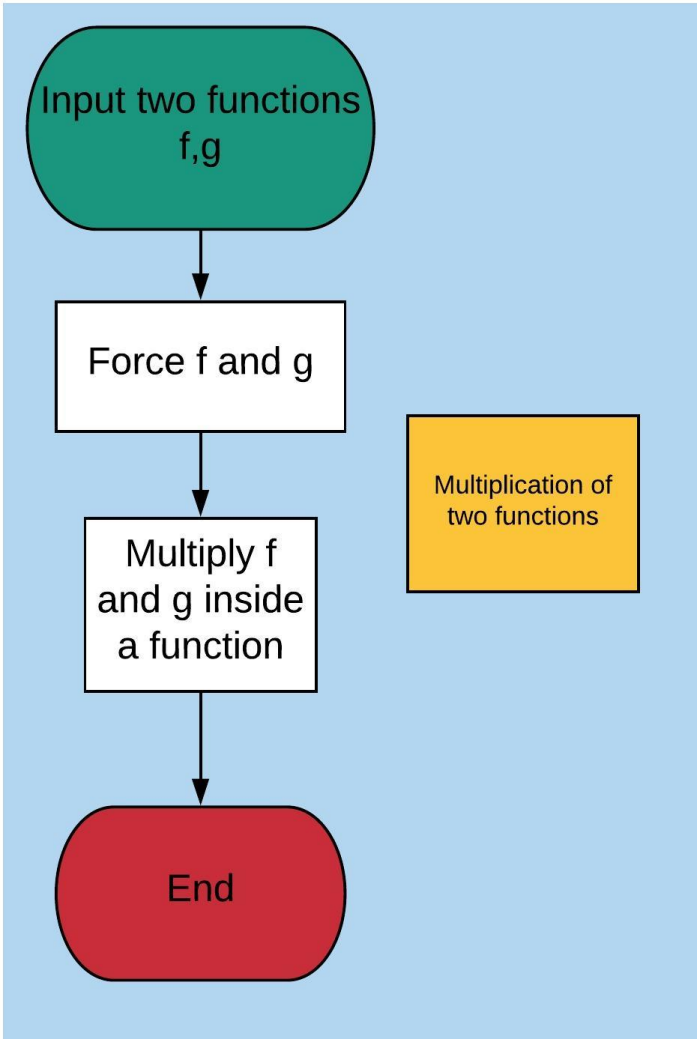

Figure 8.1: Multiplication

2. Flowchart of creation of Legendre Polynomial coefficients

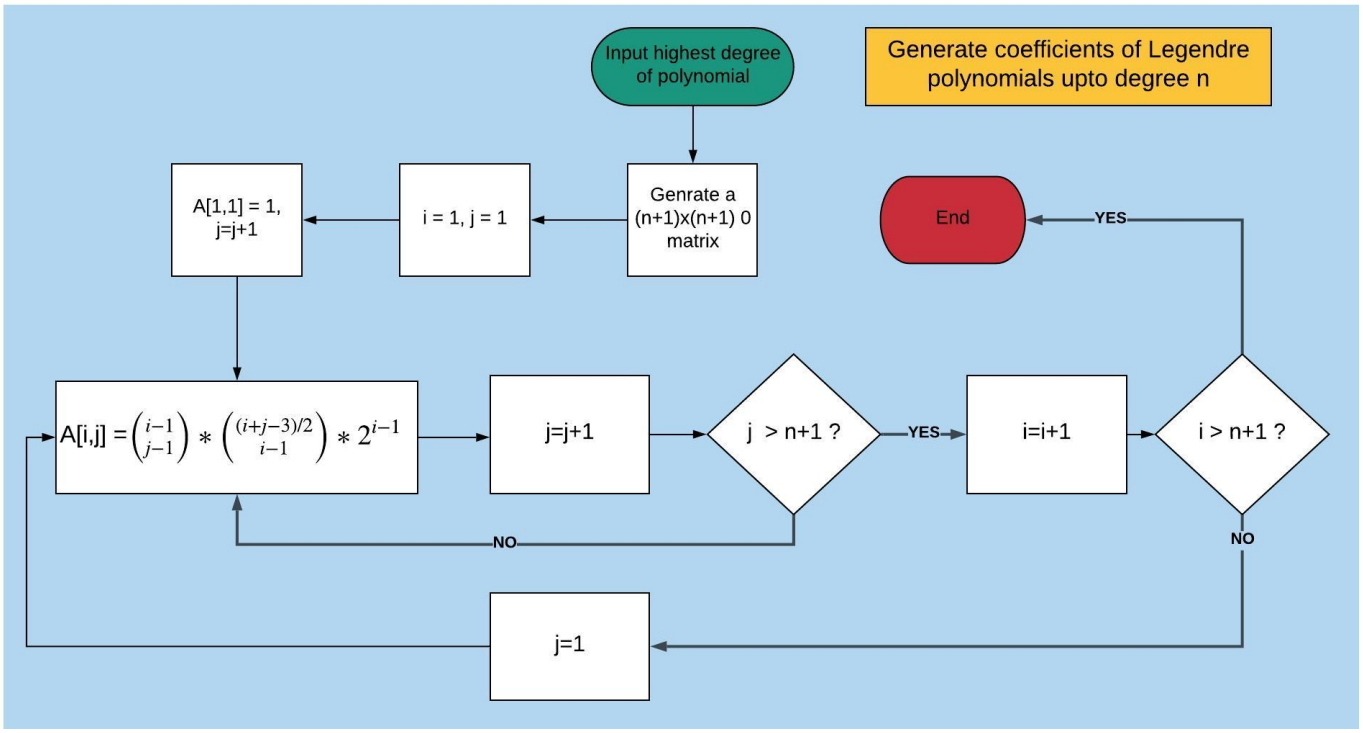

Figure 8.2: Legendre Polynomial Coefficients

3. Flowchart of creation of Hermite Polynomial coefficients

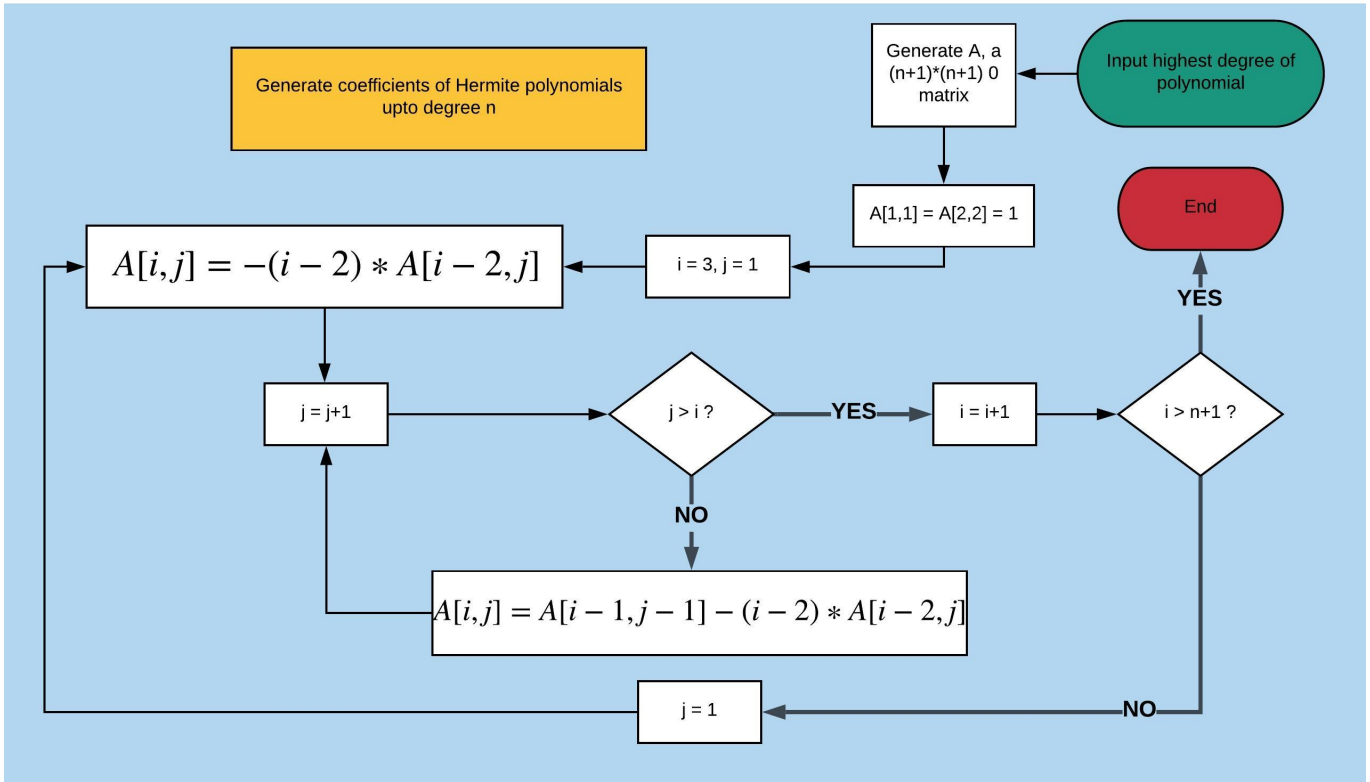

Figure 8.3: Hermite Polynomial Coefficients

4. Flowchart of creation of Laguerre Polynomial coefficients

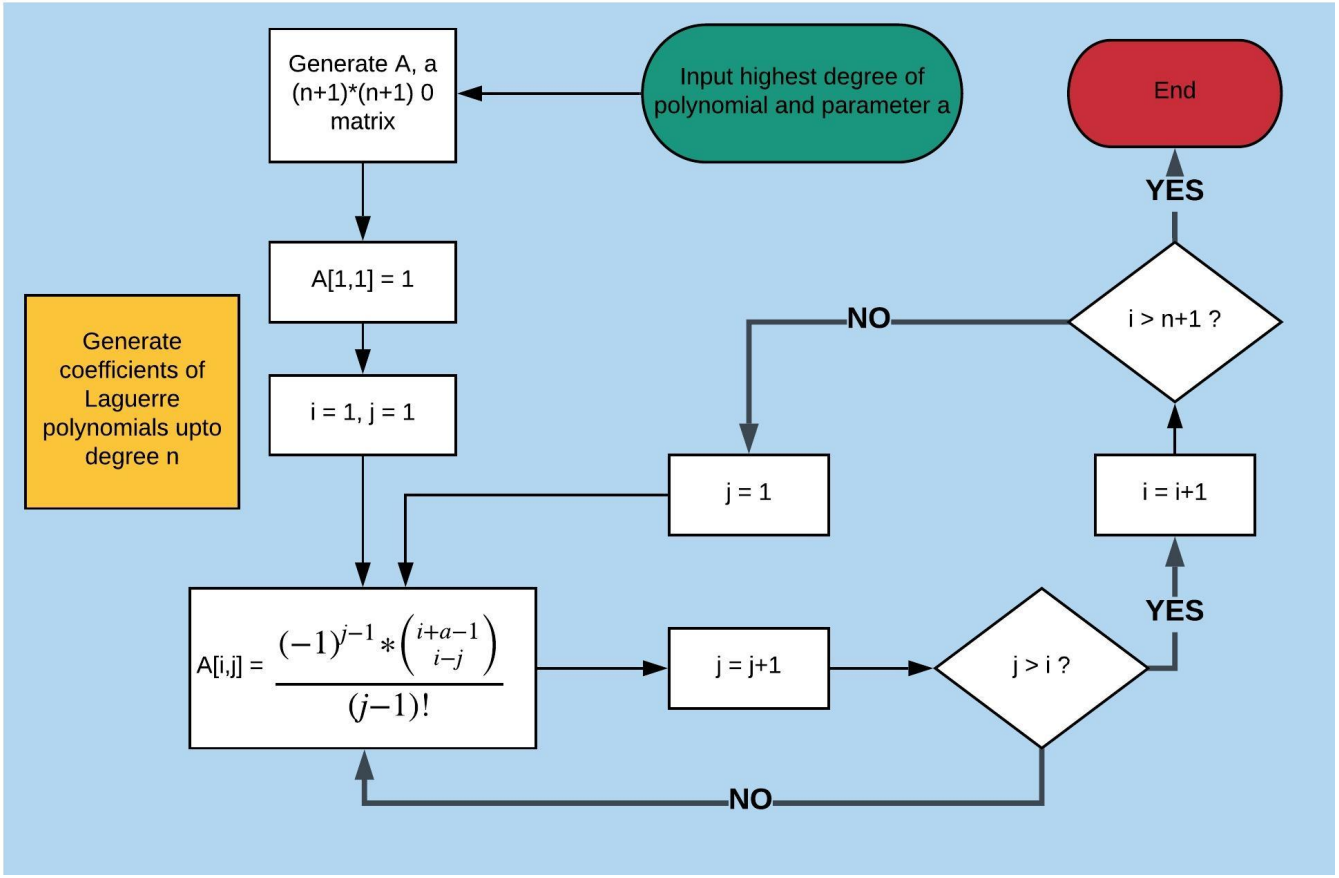

Figure 8.4: Laguerre Polynomial Coefficients

5. Flowchart of creation of General Polynomial

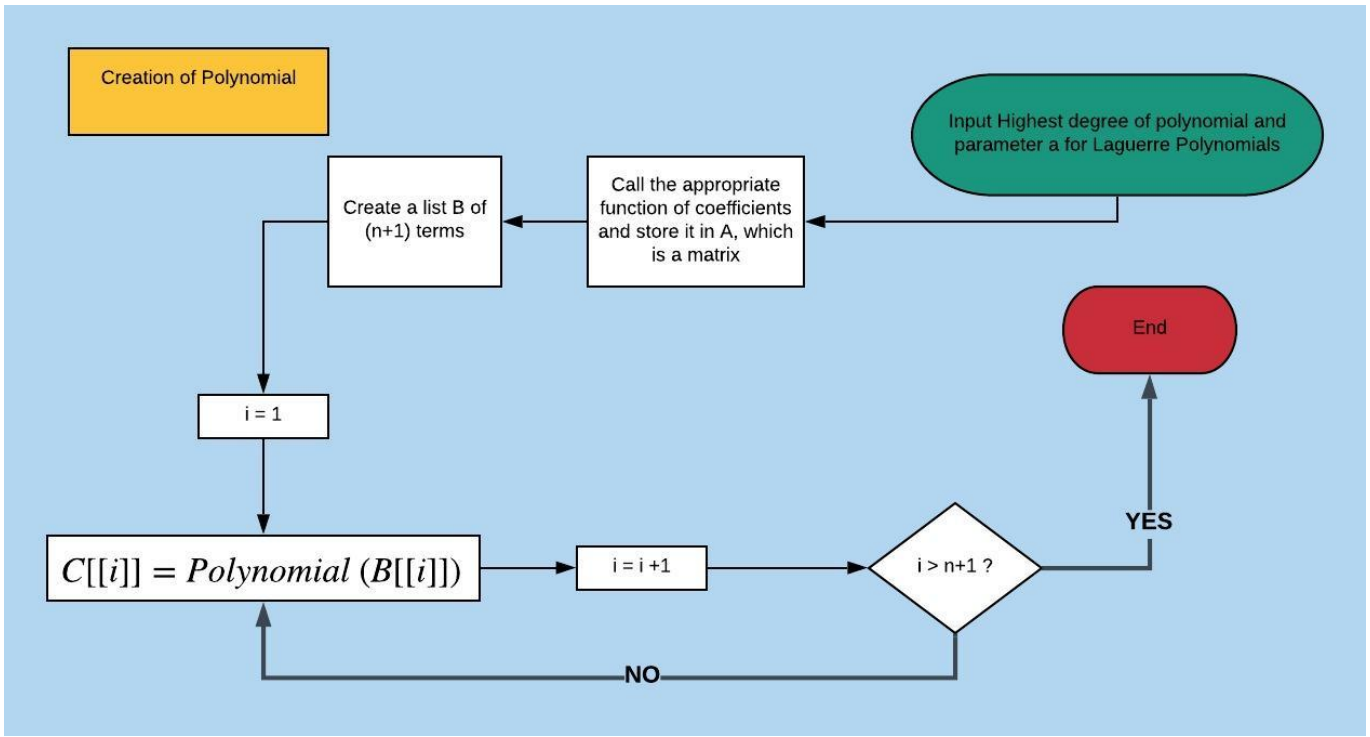

Figure 8.5: Polynomial

6. Flowchart of creation of General Polynomial Function

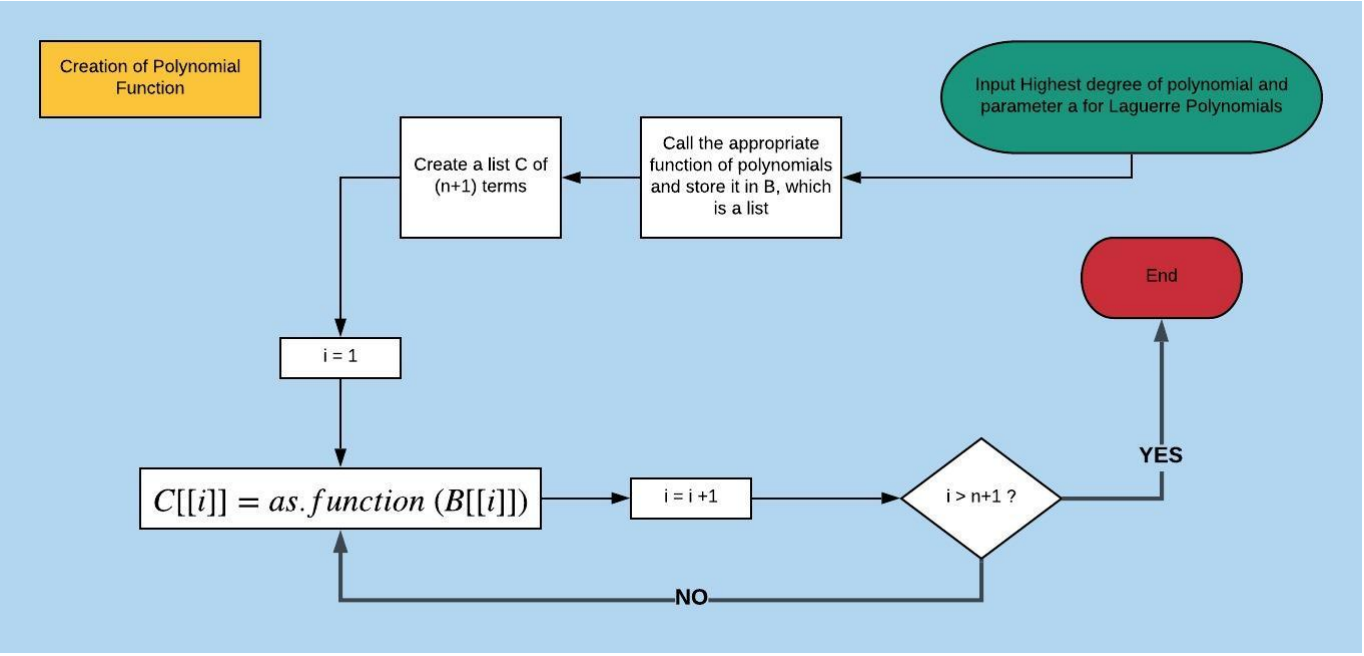

Figure 8.6: Polynomial Function

7. Flowchart of creation of Inner Product

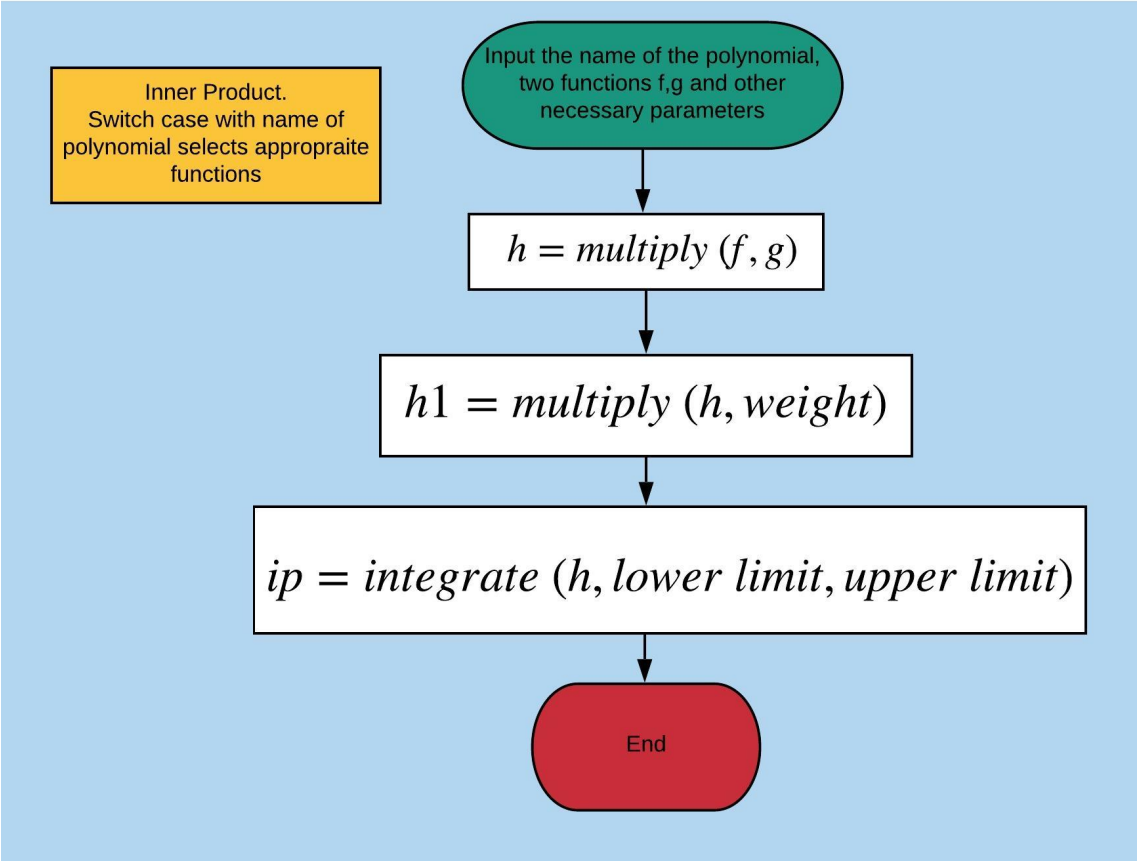

Figure 8.7: Polynomial Function

8. Flowchart of Computing the solution

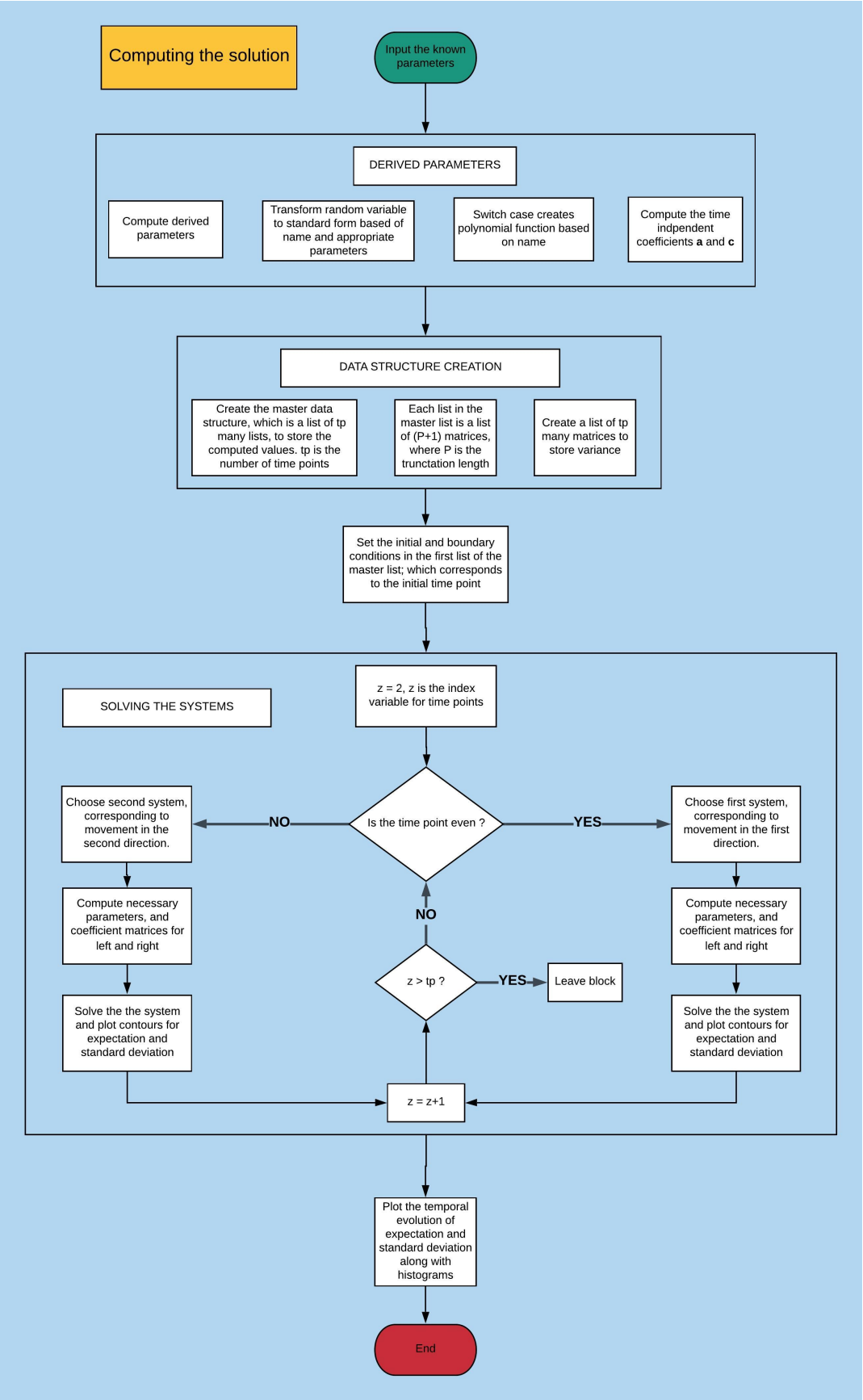

Figure 8.8: Solution

Chapter 9

Results

We set the following:

1. Equal number of cells in x and y direction = 16
2. A square, flat boundary of size 2.
3. Truncation constant, P = 3
4. Diffusion coefficient, D = 0.5
5. Maximum Carrying Capacity, K = 10
6. A value of 5 on the boundary at initial time
7. The population to be normally distributed with mean K and standard deviation 1 on the interior
8. Number of time points, tp = 2001

We show the plots of spatial and temporal evolution of the expected values of population size along with the standard deviation, and also plot the histograms at various time points.

Legendre Polynomials

We assume that the intrinsic growth rate is a uniform random variable in $(0.4, 0.6)$.

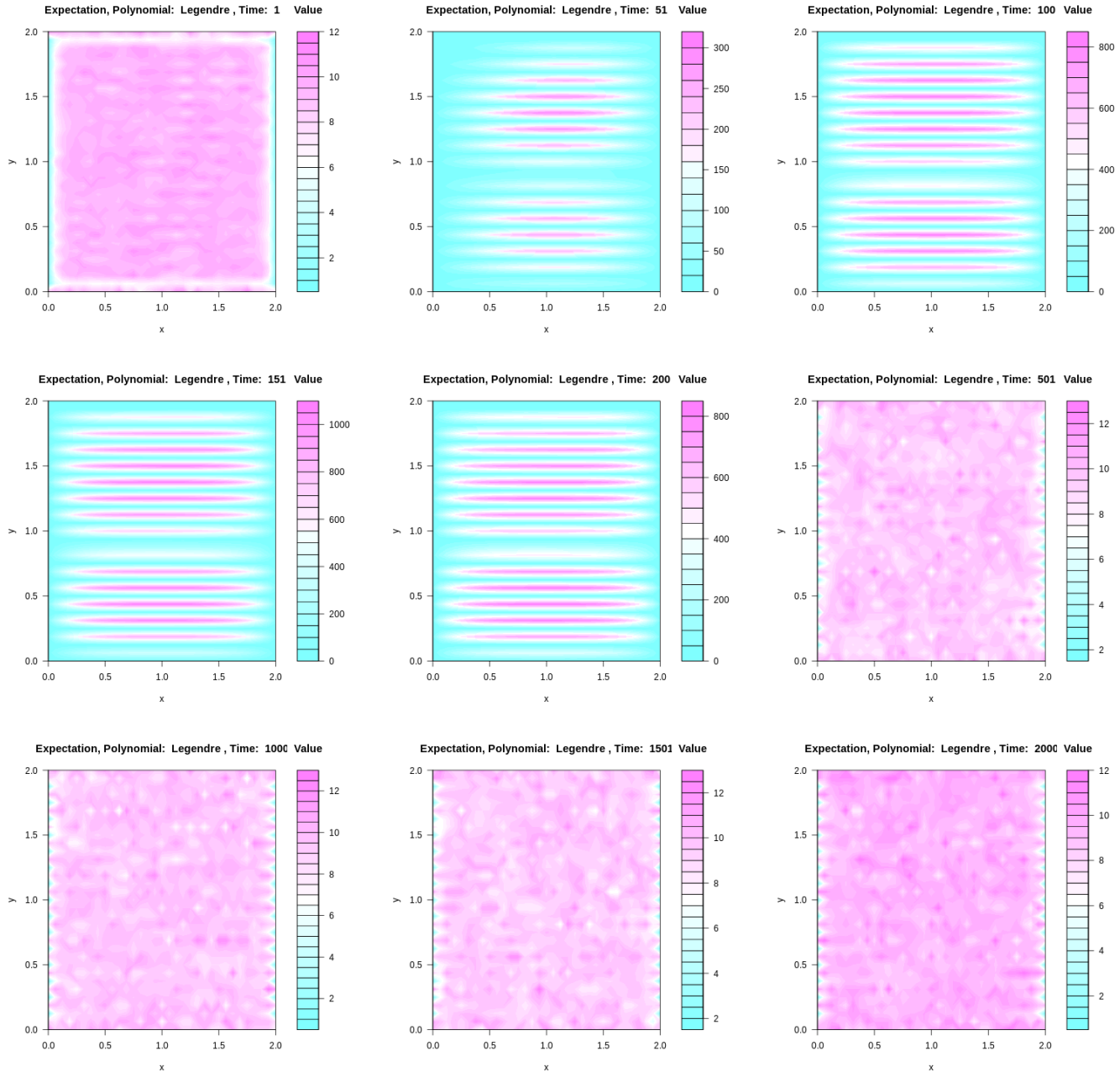

Figure 9.1: Spatial evolution of expected population size, Legendre Polynomial

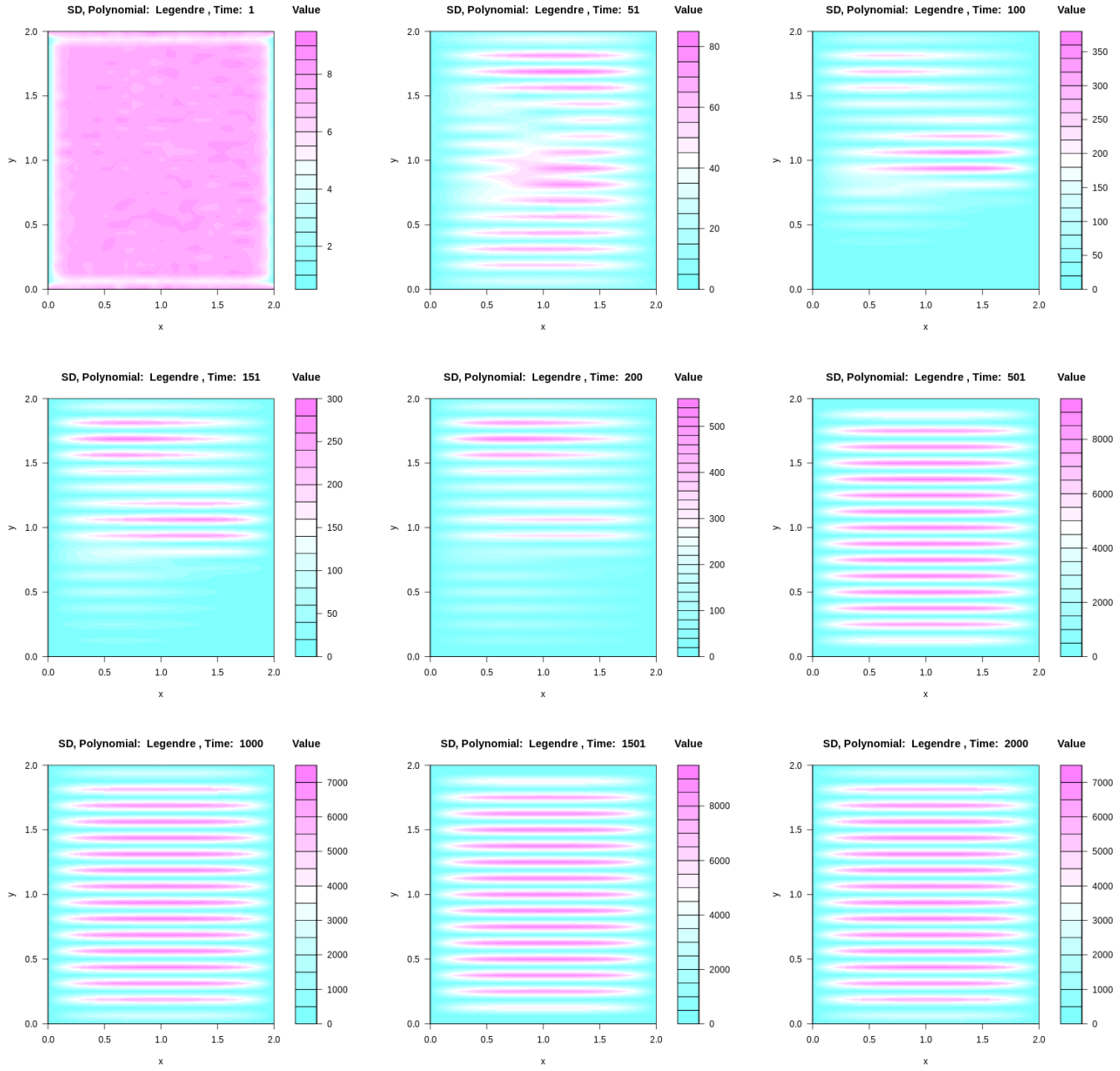

Figure 9.2: Spatial evolution of standard deviation of the population size, Legendre Polynomial

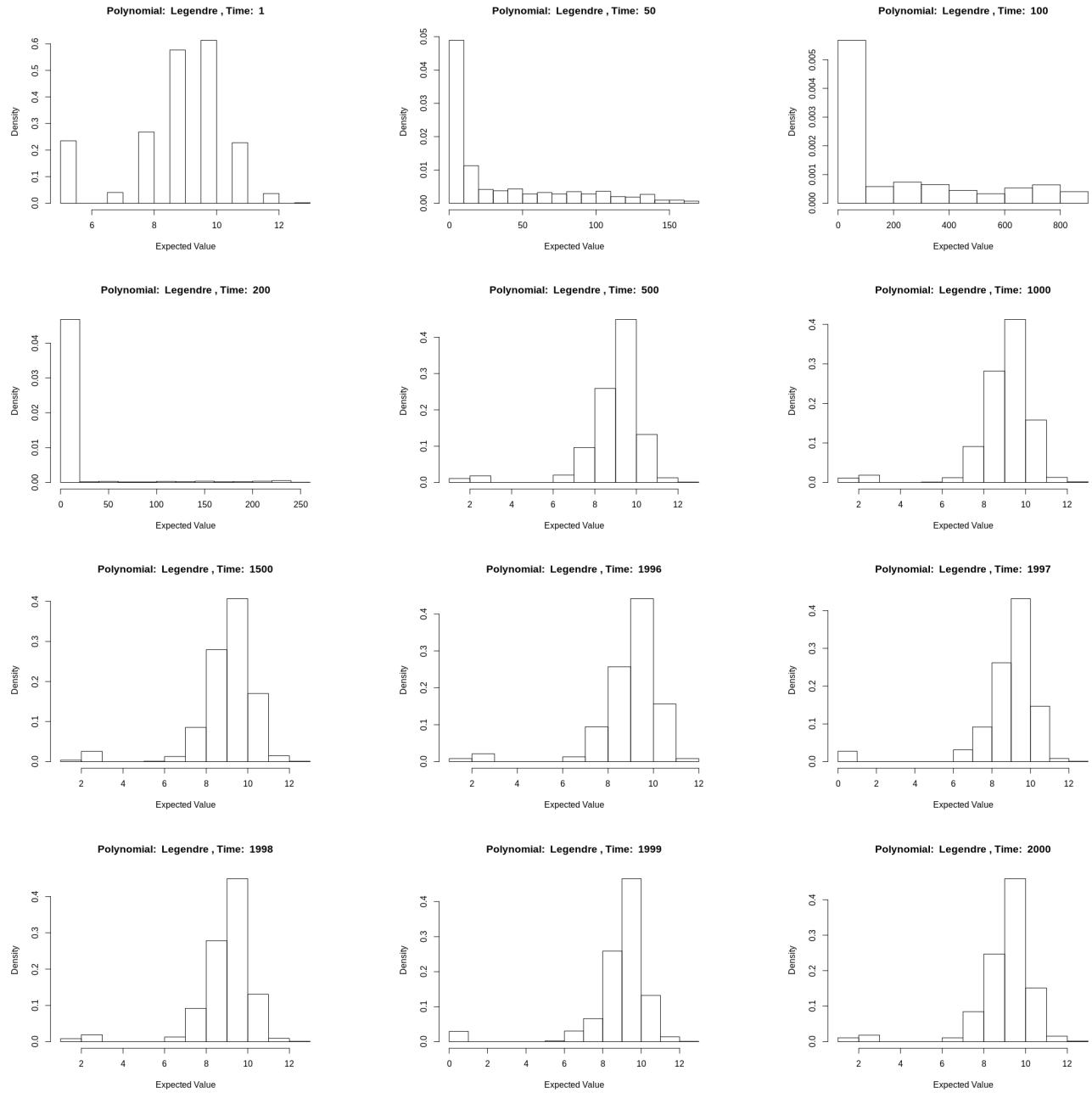

Figure 9.3: Histograms of Expected Population size, Legendre Polynomial

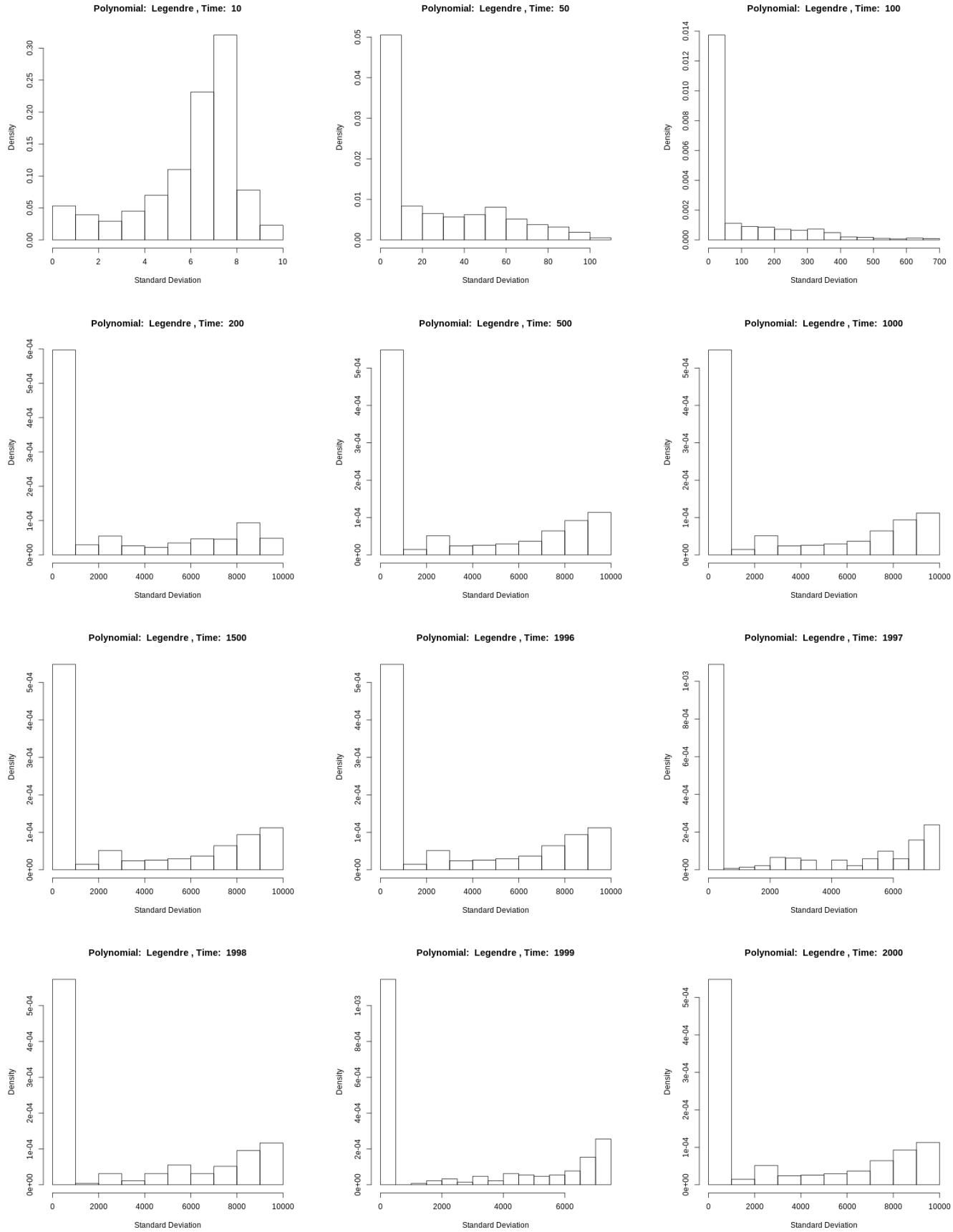

Figure 9.4: Histograms of Standard deviation of Population size, Legendre Polynomial

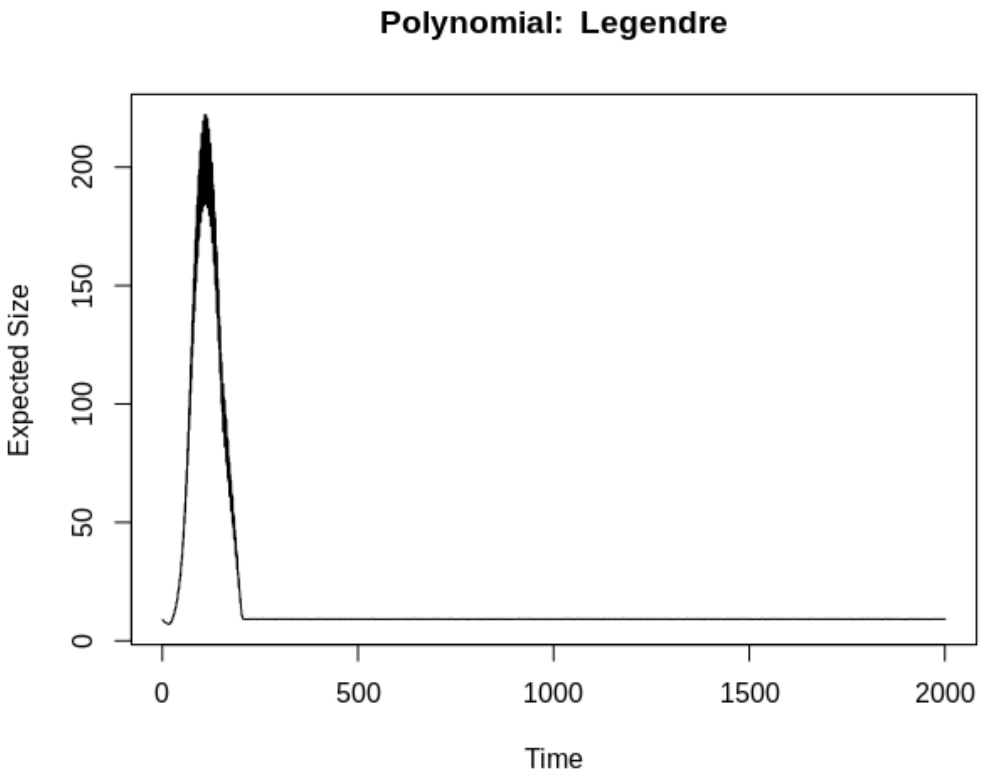

Figure 9.5: Temporal Evolution of Expected Population Size, Legendre Polynomial

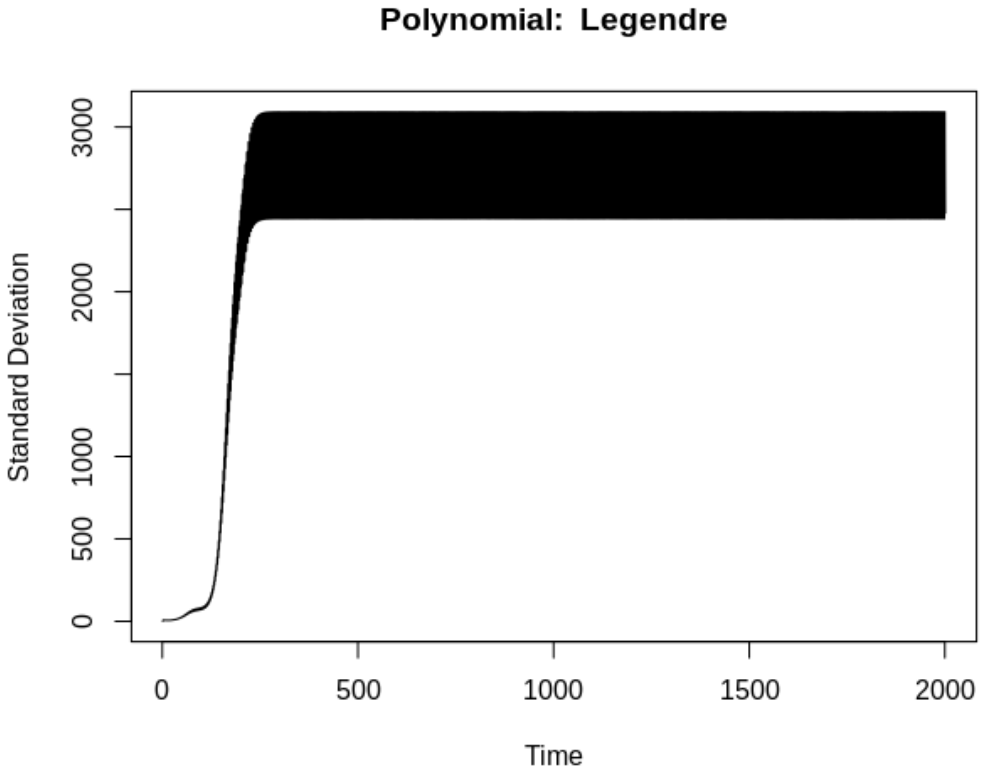

Figure 9.6: Temporal Evolution of Standard deviation of Population Size, Legendre Polynomial

Hermite Polynomials

We assume that the intrinsic growth rate is a normal random variable with mean 0.4 and standard deviation 0.6.

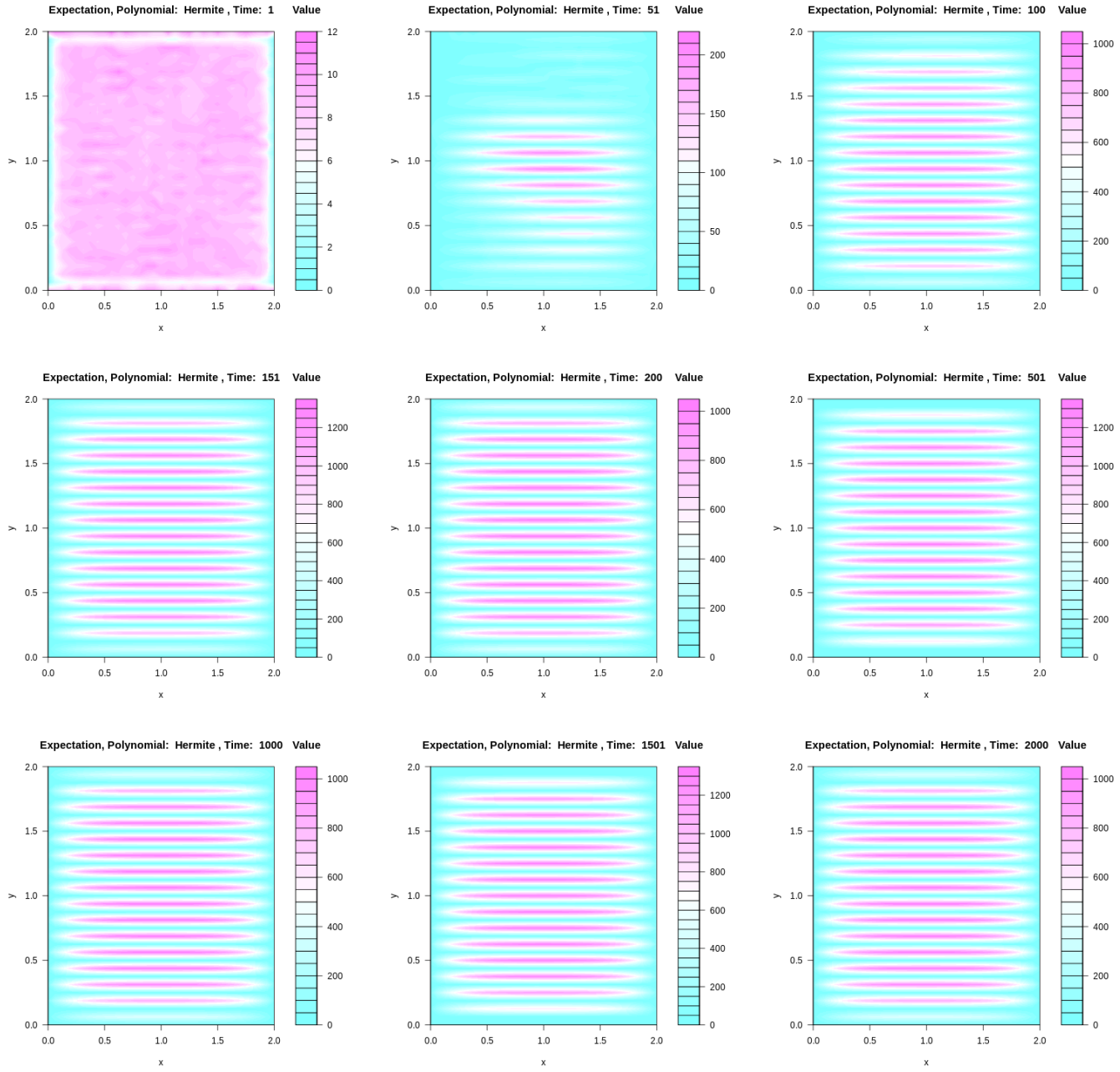

Figure 9.7: Spatial evolution of expected population size, Hermite Polynomial

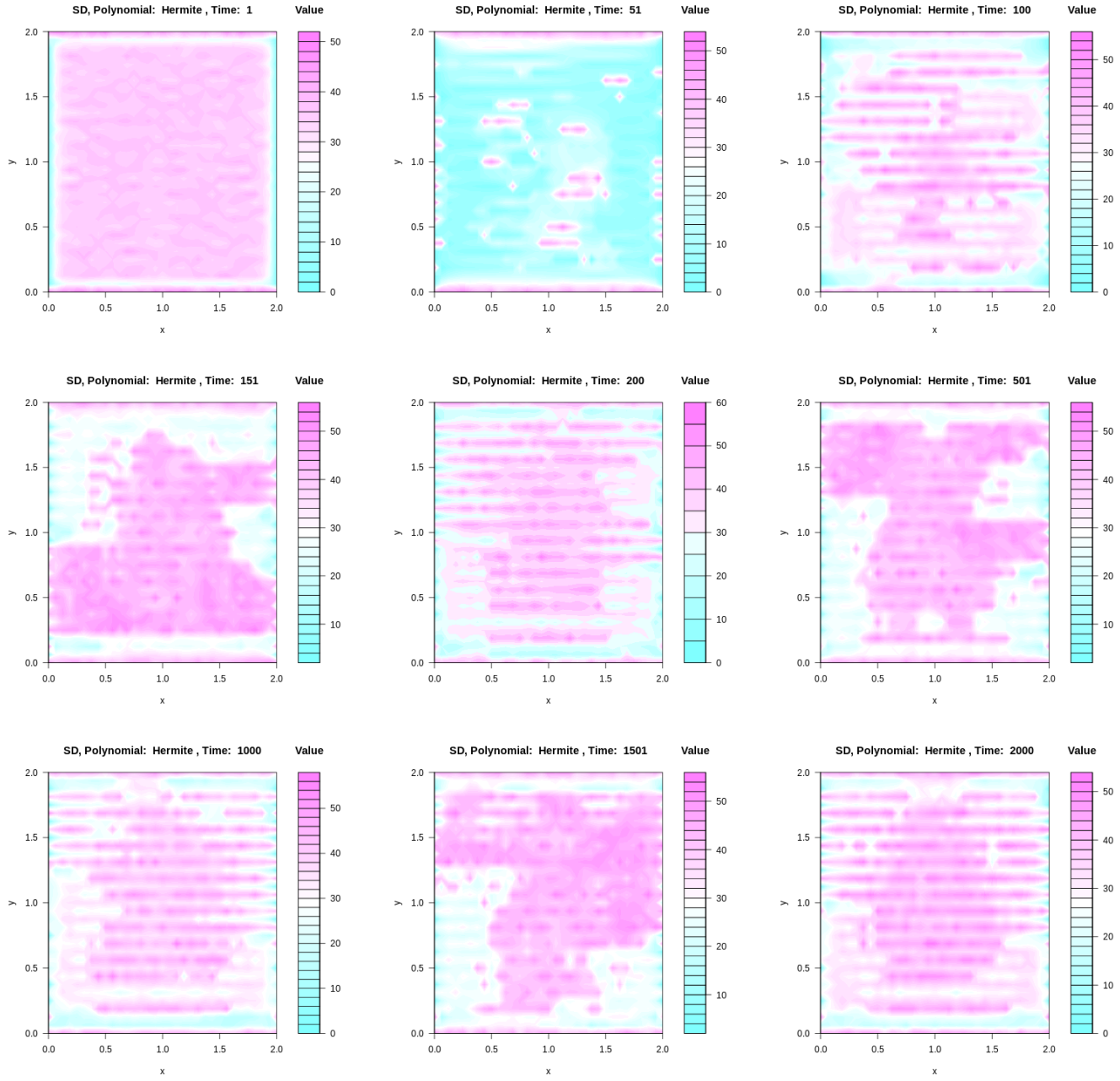

Figure 9.8: Spatial evolution of standard deviation of the population size, Hermite Polynomial

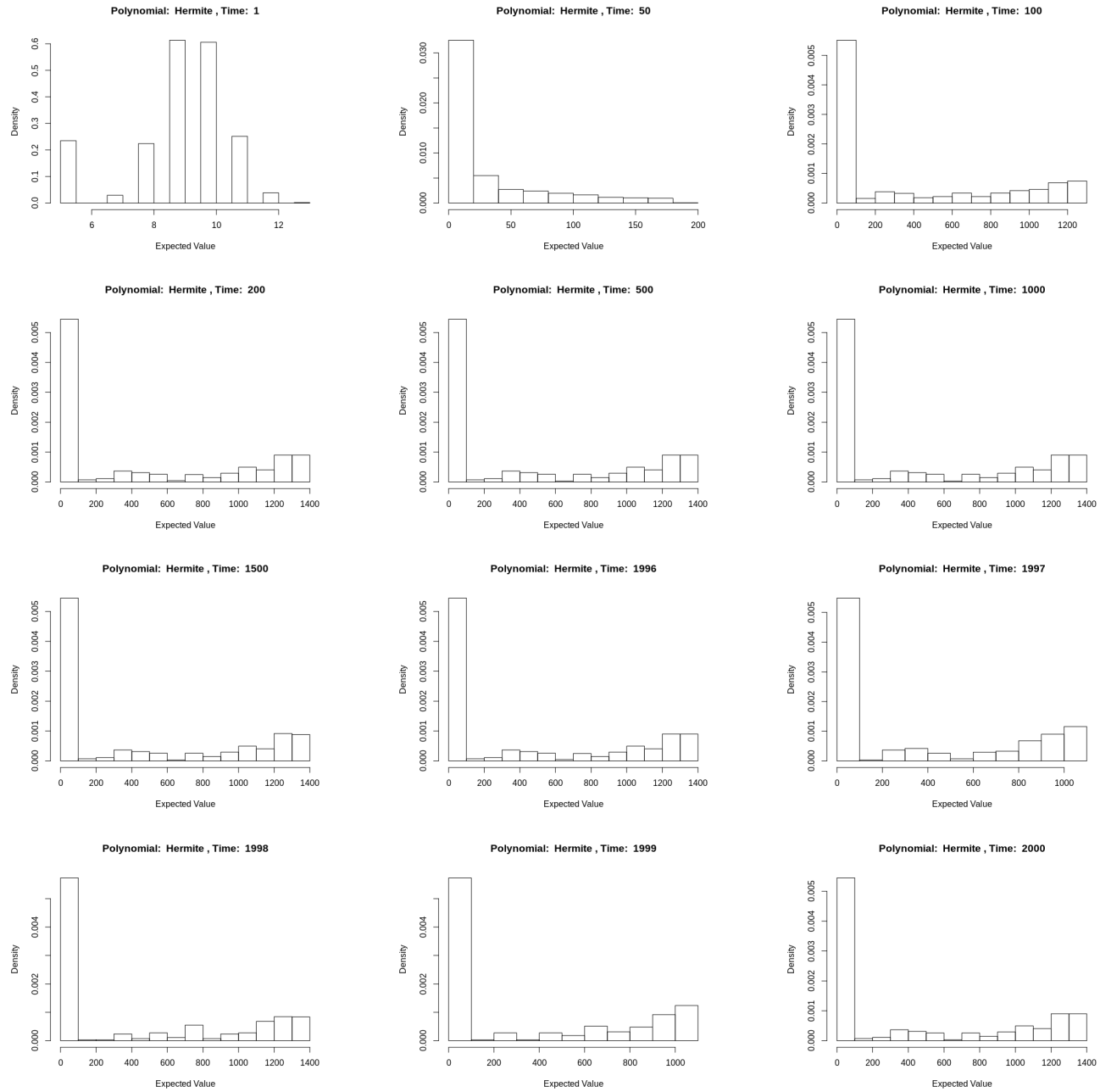

Figure 9.9: Histograms of Expected Population, Hermite Polynomial

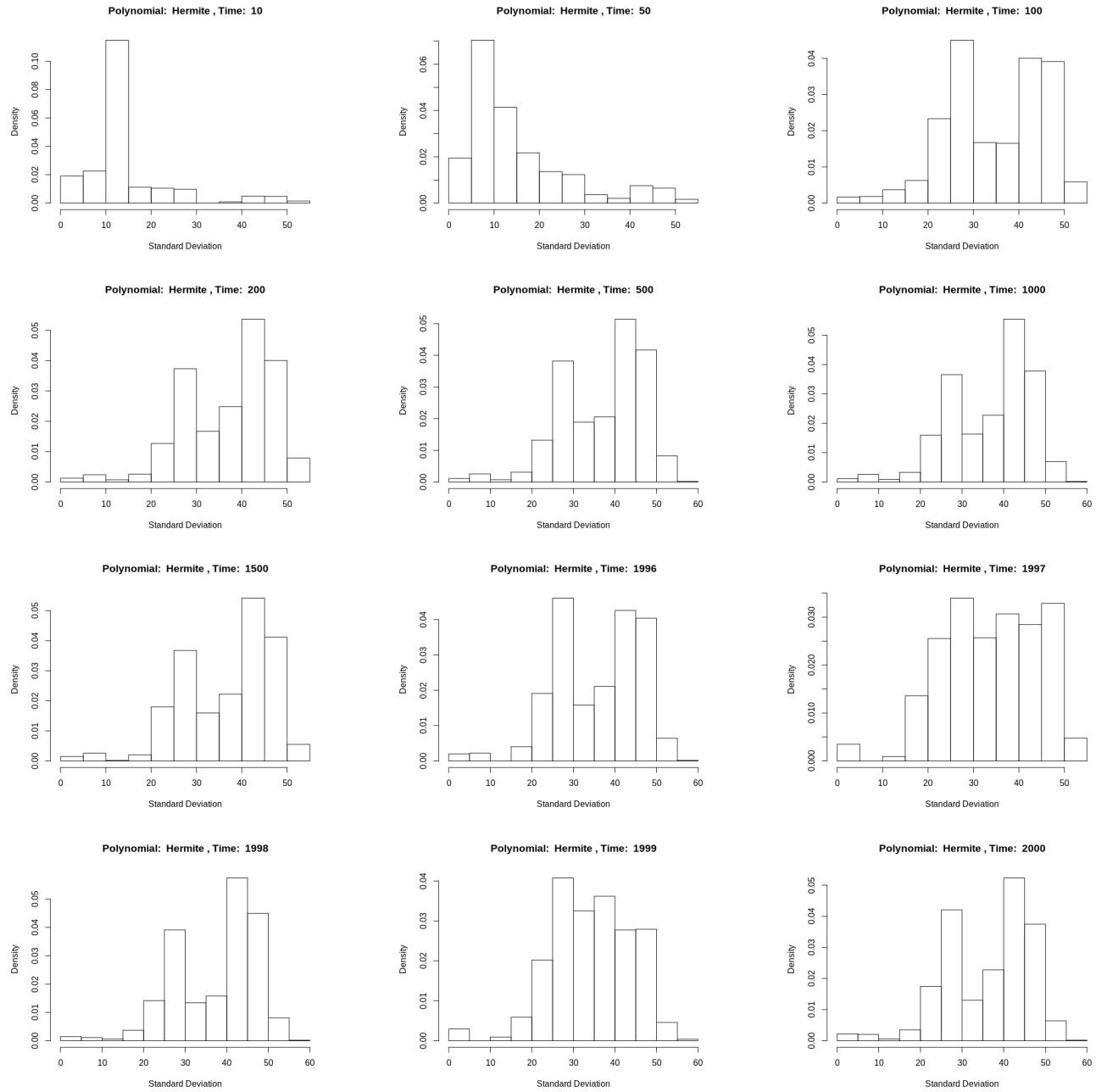

Figure 9.10: Histograms of Standard deviation of Population size, Hermite Polynomial

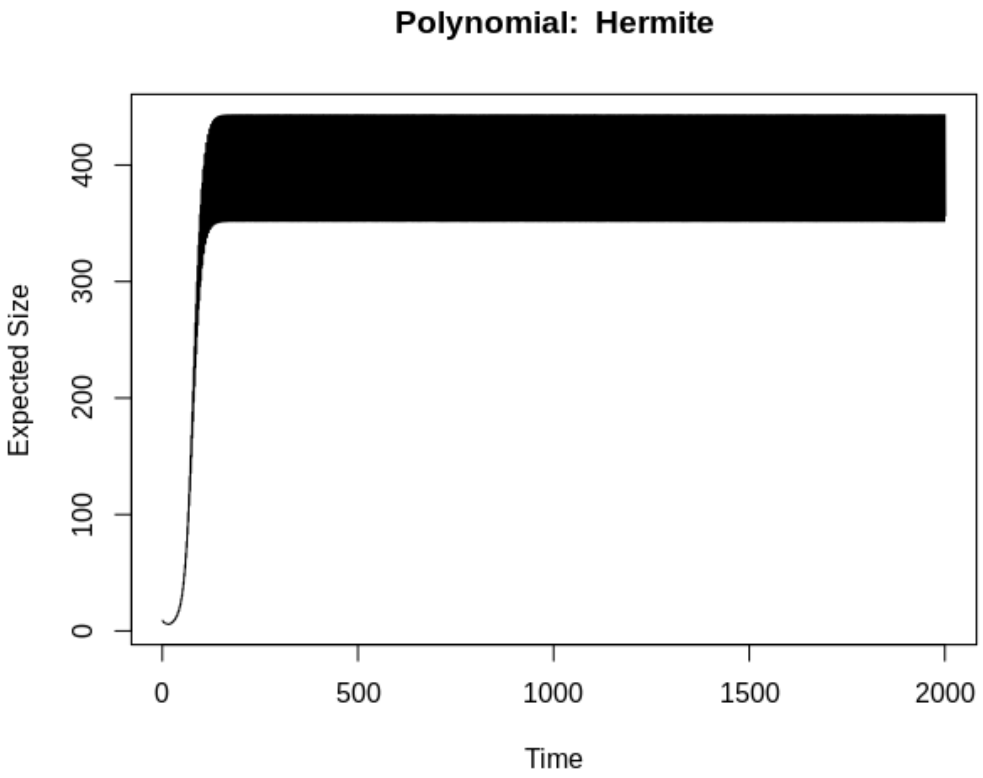

Figure 9.11: Temporal Evolution of Expected Population Size, Hermite Polynomial

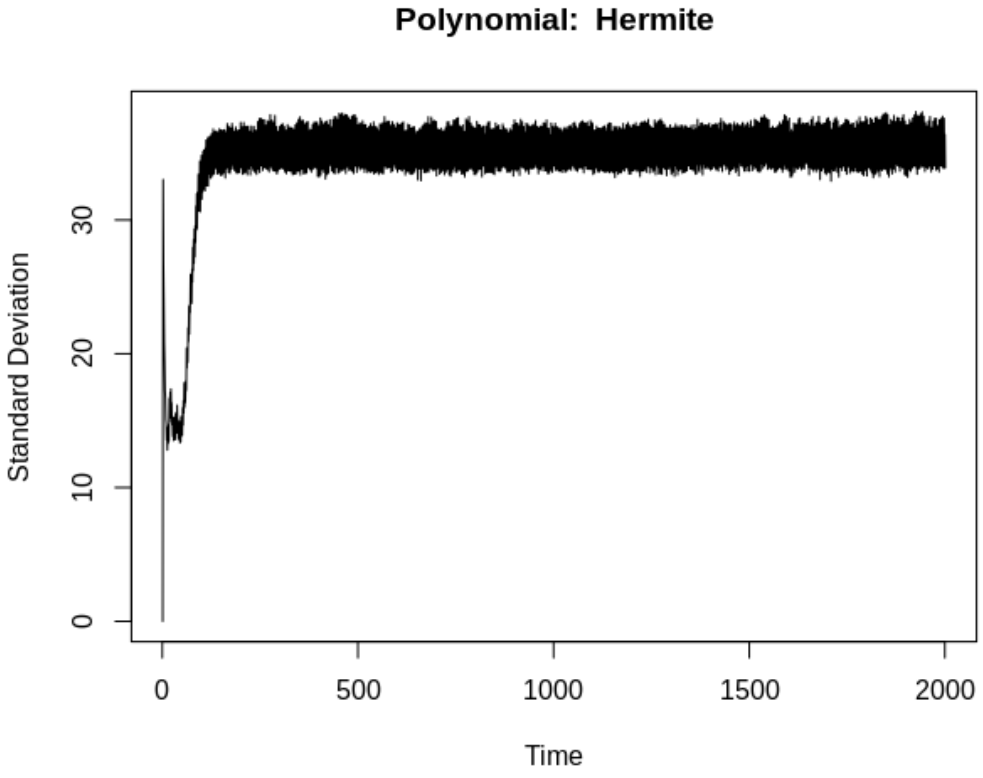

Figure 9.12: Temporal Evolution of Standard deviation of Population Size, Hermite Polynomial

Laguerre Polynomials

We assume that the intrinsic growth rate is a gamma random variable with parameter 1.

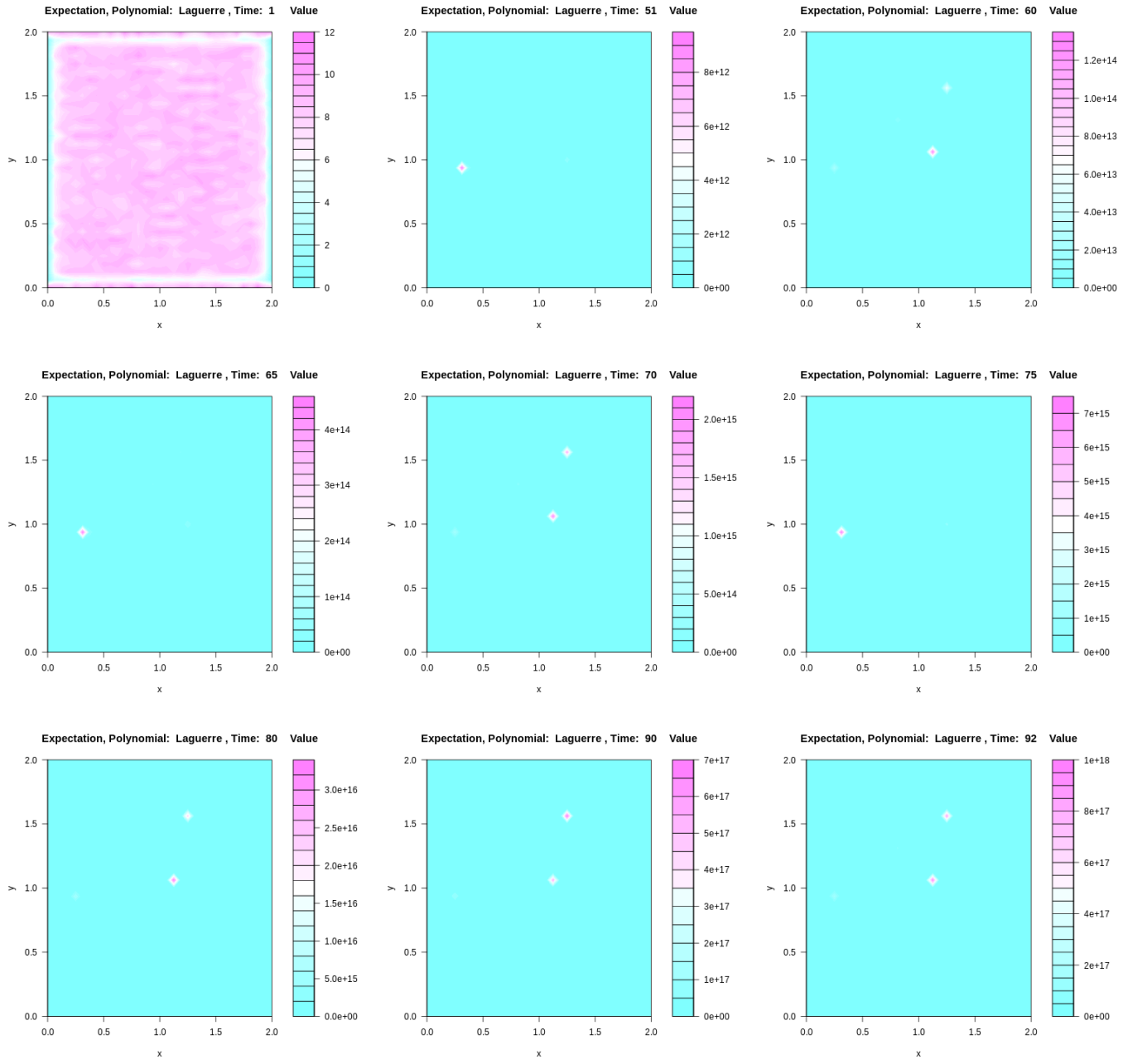

Figure 9.13: Spatial evolution of expected population size, Laguerre Polynomial

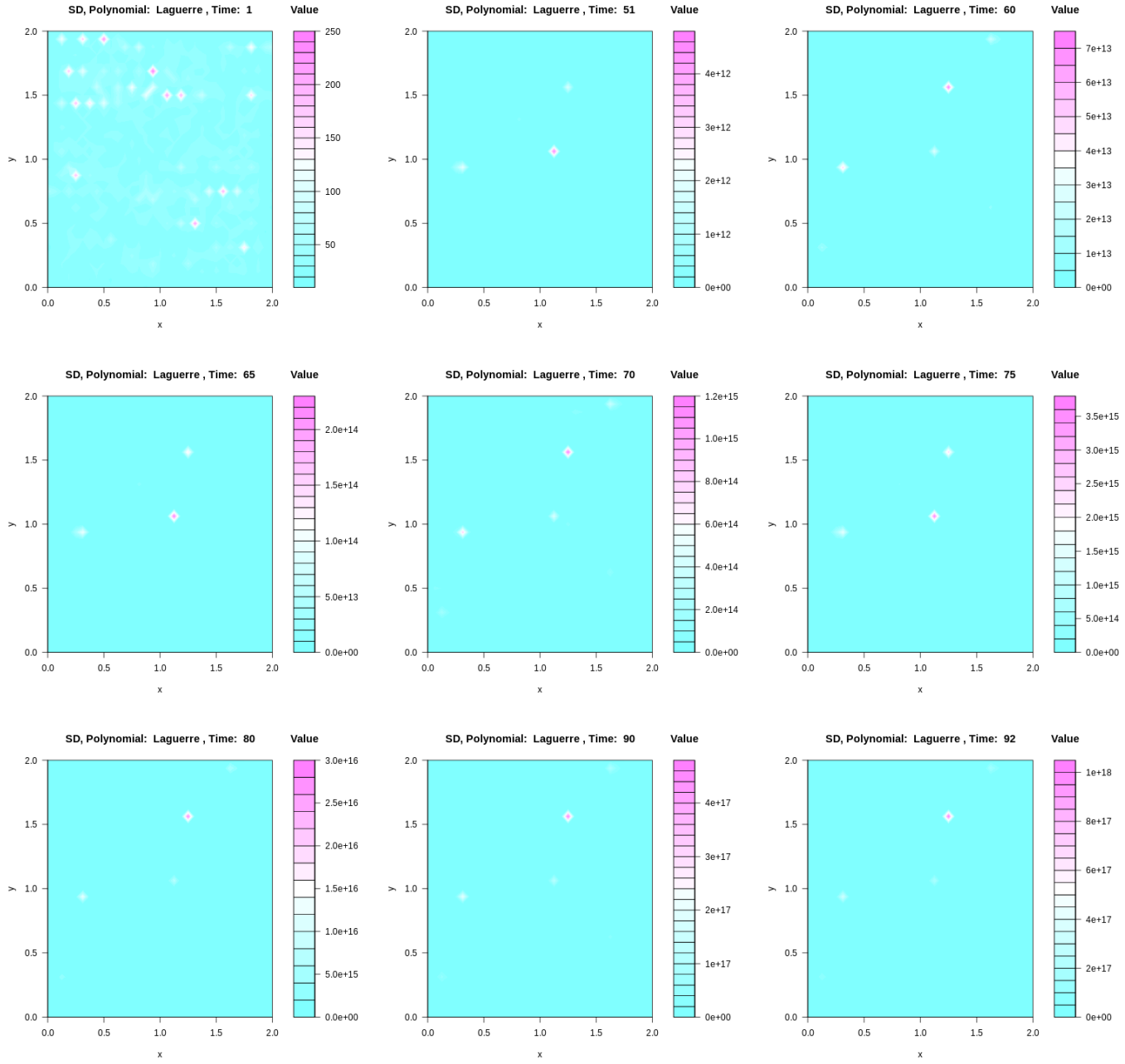

Figure 9.14: Spatial evolution of standard deviation of the population size, Laguerre Polynomial

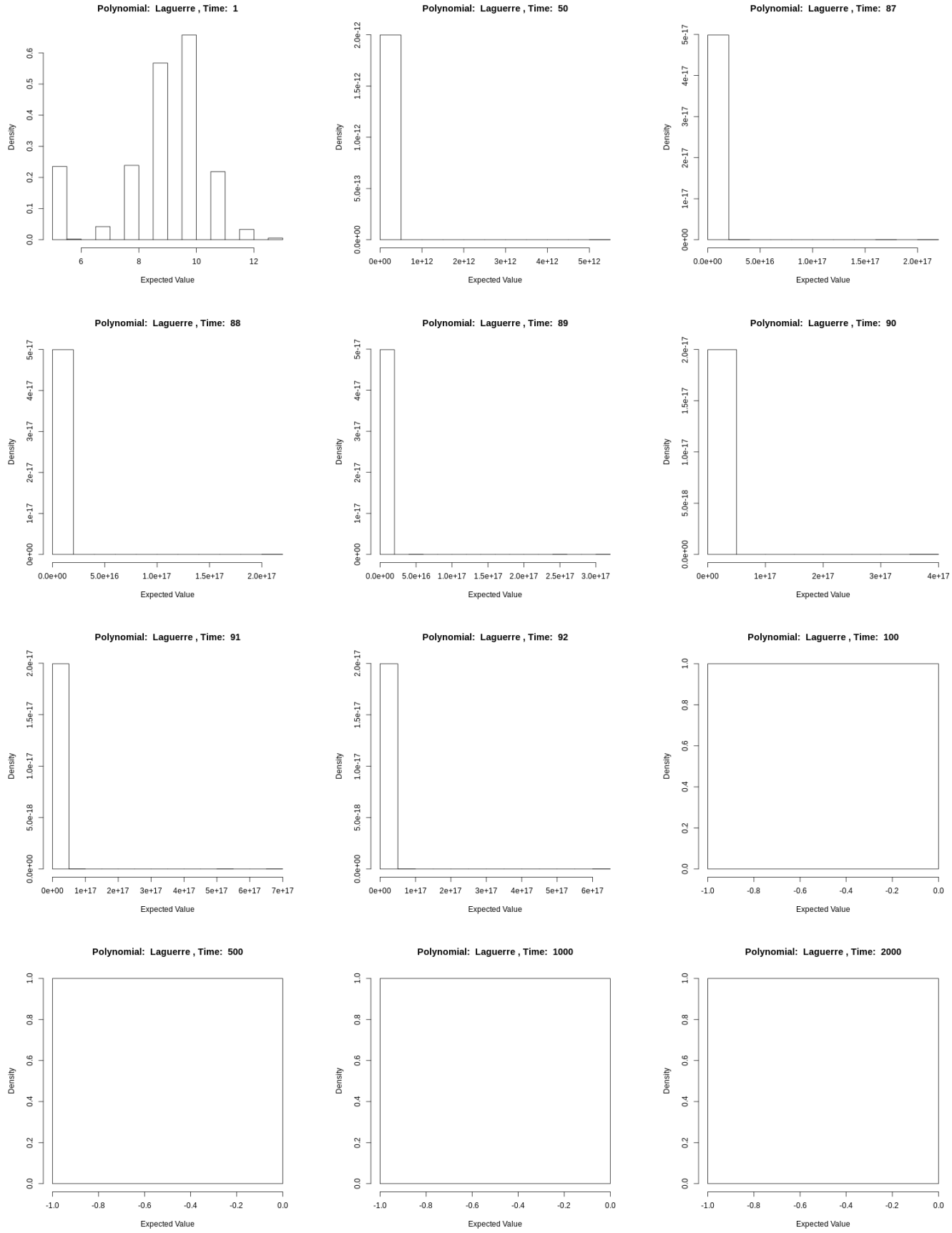

Figure 9.15: Histograms of Expected Population size, Laguerre Polynomial

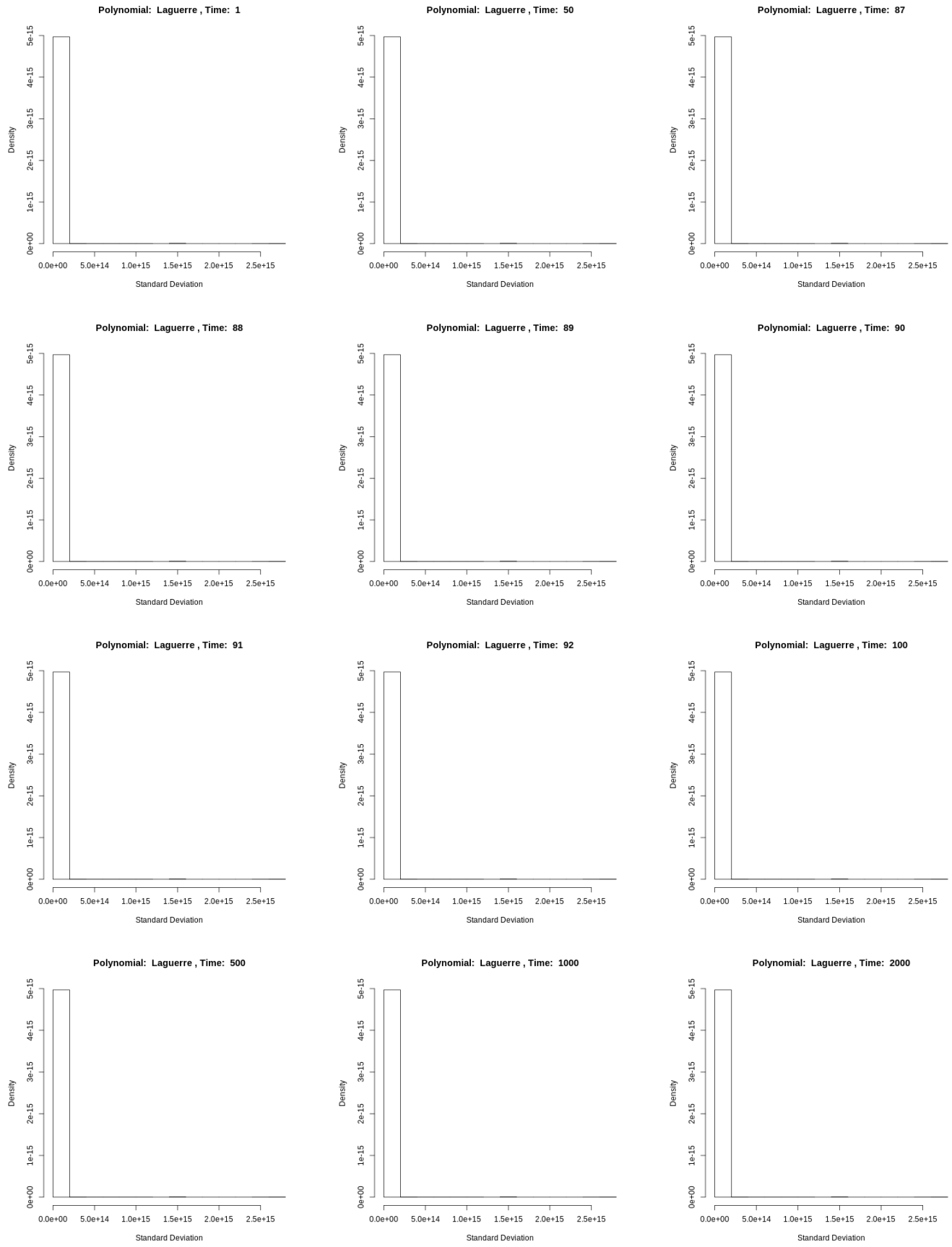

Figure 9.16: Histograms of Standard deviation of Population size, Laguerre Polynomial

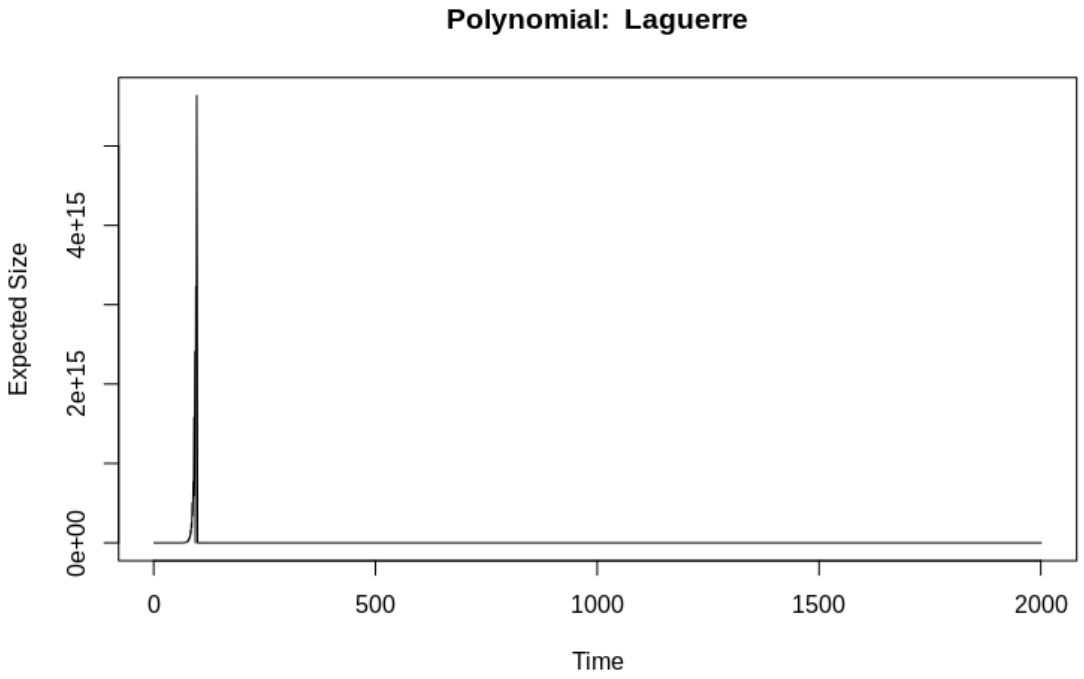

Figure 9.17: Temporal Evolution of Expected Population Size, Laguerre Polynomial

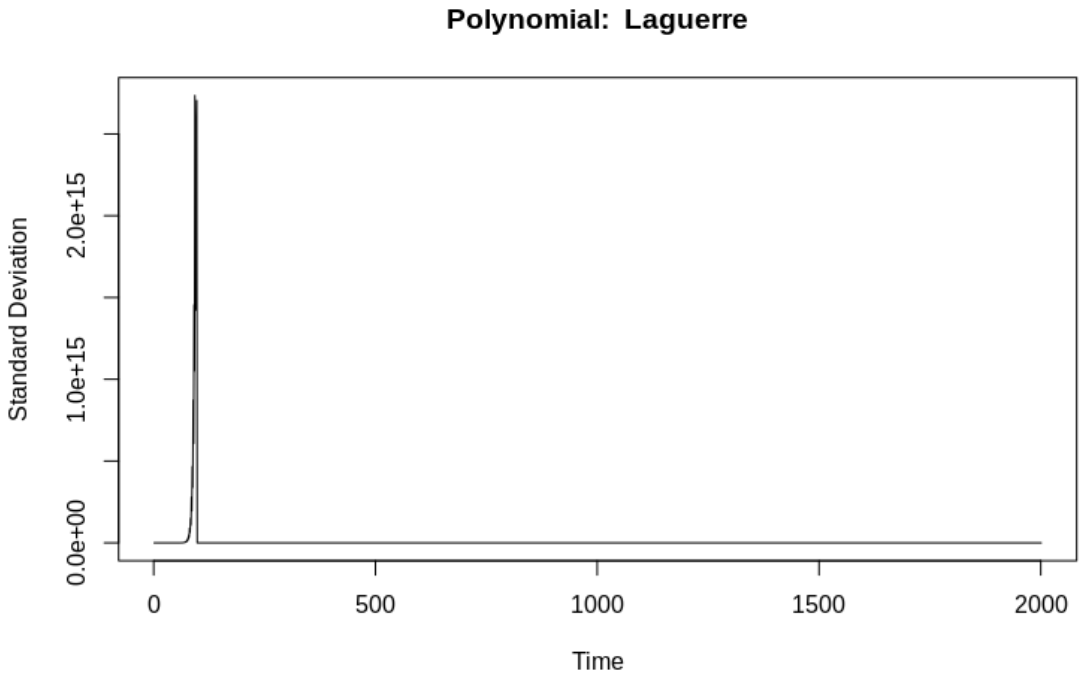

Figure 9.18: Temporal Evolution of Standard deviation of Population Size, Laguerre Polynomial

Chapter 10

Code

All codes are written in R.

1. The following code multiplies two function structures in R.

```
multiply <- function(f,g){
  force(f)
  force(g)
  function(x){f(x)*g(x)}
}
```

2. **The following code generates Legendre polynomials**

```
library(polynom)

lp_coeff <- function(n){  # Generates Legendre Polynomial Coefficients
  A <- matrix(0,nrow = (n+1), ncol = (n+1))
  A[1,1] <- 1
  for(i in 1:(n+1)){
     for(j in 1:(n+1)){
       A[i,j] <- choose((i-1),(j-1))*choose((i+j-3)/2,(i-1))
     }
     A[i,] <- A[i,]*2^(i-1)
   }
   return(A)
}

lp <- function(n){ # Generates Legendre polynomials
  A <- lp_coeff(n)
  B <- list((n+1))
  for(i in 1:(n+1)){
    B[[i]] <- polynomial(A[i,])
  }
  return(B)
}

lp_func <- function(n){ # Polynomials as functions
  B <- lp(n)
  C <- list((n+1))
  for(i in 1:(n+1)){
    C[[i]] <- as.function(B[[i]])
  }
```

```
  return(C)
}

lp_wt <- function(x){0.5} # Weight
```

3. **The following code generates Hermite Polynomials**

```
library(polynom)

hp_coeff <- function(n){  # Generates Hermite Polynomial Coefficients
  A <- matrix(0,nrow = (n+1), ncol = (n+1))
  A[1,1] <- 1
  A[2,2] <- 1
  for(i in 3:(n+1)){
    for(j in 1:i){
      if(j==1){
        A[i,j] <- -(i-2)*A[(i-2),j]
      }else{
        A[i,j] <- A[(i-1),(j-1)] - (i-2)*A[(i-2),j]
      }
    }
  }
  return(A)
}

hp <- function(n){ # Generates Hermite polynomials
  A <- hp_coeff(n)
  B <- list((n+1))
  for(i in 1:(n+1)){
    B[[i]] <- polynomial(A[i,])
  }
  return(B)
}

hp_func <- function(n){ # Polynomials as functions
  B <- hp(n)
  C <- list((n+1))
  for(i in 1:(n+1)){
    C[[i]] <- as.function(B[[i]])
  }
  return(C)
}

hp_wt <- function(x){exp(-x*x/2)} # Weight
```

4. **This code generates Laguerre Polynomials**

```
library(polynom)

lagp_coeff <- function(n,a){  # Generates Laguerre Polynomial Coefficients
  A <- matrix(0,nrow = (n+1), ncol = (n+1))
  A[1,1] <- 1
  for(i in 1:(n+1)){
    for(j in 1:(i)){
      A[i,j] <- (-1)^(j-1)*choose((i+a-1),(i-j))/factorial((j-1))
    }
  }
  return(A)
}

lagp <- function(n,a){ # Generates Laguerre polynomials
  A <- lagp_coeff(n,a)
  B <- list((n+1))
  for(i in 1:(n+1)){
    B[[i]] <- polynomial(A[i,])
  }
  return(B)
}

lagp_func <- function(n,a){ # Polynomials as functions
  B <- lagp(n,a)
  C <- list((n+1))
  for(i in 1:(n+1)){
    C[[i]] <- as.function(B[[i]])
  }
  return(C)
}

lag <- 0
f1 <- function(x){x^lag}
f2 <- function(x){exp(-x)}
lagp_wt <- multiply(f1,f2) # Weight
```

5. **This code generates inner products**

```
IP <- function(Type,f,g,a){
switch(Type,
       Legendre={
         h1 <- multiply(f,g)
         h <- multiply(h1,lp_wt)
         ip <- integrate(h,-1,1)$value
         return(ip)
       },
       Hermite={
         h1 <- multiply(f,g)
         h <- multiply(h1,hp_wt)
         ip <- integrate(h,-Inf,Inf)$value
         return(ip)
       },
       Laguerre={
         h1 <- multiply(f,g)
         h <- multiply(h1,lagp_wt)
         ip <- integrate(h,0,Inf)$value
         return(ip)
       },
       {
         return(paste('Enter polynomial Type correctly.'))
       }
)
}
```

6. **This code gives us the required solution and plots**

```
library(svMisc)

## Input Parameters
 cellsx <- 16
 cellsy <- 16
 x1 <- 0
 x2 <- 2
 y1 <- 0
 y2 <- 2
 Time <- 10
 Steps <-  100
 P <- 3
 D <- 0.5
 K <- 10
 Boundary <- 5
 name <- "Legendre"
 a0 <- 0.4 # a for legendre, mean for hermite
 b0 <- 0.6 # b for legendre, sd for hermite
 # this means that r is a uniform random variable in (a0,b0)
 # or that r is normal(a0,b0)
 lag <- 1 # alpha for gamma distribution

## Setting the polynomial
  switch(name,
        Legendre={
          PolyP <- lp_func(P)
        },
        Hermite={
          PolyP <- hp_func(P)
        },
        Laguerre={
          PolyP <- lagp_func(P,lag)   # lag = parameter of gamma
        },
        {
          PolyP <- (paste('Enter polynomial Type correctly.'))
        }
       )

## Transforming the random variable
 switch(name,
        Legendre={
          eps <- function(x){
              (b0-a0)*(x+1)/2+a0
          }
        },
        Hermite={
          eps <- function(x){
            b0*x+a0
          }
        },
        Laguerre={
          eps <- function(x){
            x
```

```
        }
      }
)


## Derived Parameters
  dx <- 1/cellsx
  dy <- 1/cellsy
  lx <- x2-x1
  ly <- y2-y1
  X <- seq(x1,x2,dx)
  Y <- seq(y1,y2,dy)
  dt <- 1/Steps
  t1 <- seq(0,Time,dt/2)
  xp <- lx/dx+1
  yp <- ly/dy+1
  tp <- length(t1)
  tp

## Data structure
  U <- list()
  for(i in 1:tp){
    U[[i]] <- list()
  }

  for(i in 1:tp){
    for(k in 1:(P+1)){
      U[[i]][[k]] <- matrix(0,nrow =xp,ncol = yp )
    }
  }

  variance <- list()
  for(i in 1:tp){
    variance[[i]] <- matrix(0,nrow=(lx/dx+1),ncol = (ly/dy+1))
  }

  ## Time indepedent coefficients
  # a
  a <- numeric(P+1)
  for(i in 1:(P+1)){
    a[i] <- -D*IP(name,PolyP[[i]],PolyP[[i]])/(dx)^2
  }
  a

  #c
  c <- numeric((P+1))
  for(i in 1:(P+1)){
    c[i] <- D*IP(name,PolyP[[i]],PolyP[[i]])/(dy)^2
  }
  c

  ## Initialising, time = 0
  for(k in 1:(P+1)){
    for(i in 2:(lx/dx)){
```

```
      for(j in 2:(ly/dy)){
        U[[1]][[k]][i,j] <- as.integer(rnorm(1,K,1))
      }
    }
  }
  for(k in 1:(P+1)){
    for(i in 1:((lx/dx)+1)){
      for(j in 1:((ly/dy)+1)){
        if(i==1|i==((lx/dx)+1)){
          U[[1]][[k]][i,j] <- Boundary
        }
        if(j==1|j==((ly/dy)+1)){
          U[[1]][[k]][i,j] <- Boundary
        }
      }
    }
  }
str(U)

  ## RUN THIS
  ## Choice of system and finding u_i's at each time point.
  for(z in 2:tp){
    o <- z-1
    if(z%%2==0){          # Choose system 1.
      ## Time dependent coefficients

      #b
      b <- list()
      for (k in 1:(P+1)){
        b[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
      }

      b_sum <- list()
      for(k in 1:(P+1)){
        b_sum[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
      }

      for (k in 1:(P+1)){
        for (i in 1:(lx/dx+1)){
          for(j in 1:(ly/dy+1)){
            for(l in 1:(P+1)){
              if(l!=k){
                b_sum[[k]][i,j] <- b_sum[[k]][i,j] + U[[(z-1)]][[l]][i,j]*
                  IP(name,multiply(PolyP[[l]],eps),multiply(PolyP[[k]],PolyP[[k]]))
              }
            }
            b_sum[[k]][i,j] <- b_sum[[k]][i,j]/K
          }
        }
      }

      for (k in 1:(P+1)){
        for (i in 1:(lx/dx+1)){
          for(j in 1:(ly/dy+1)){
```

```
      b[[k]][i,j] <- 2*IP(name,PolyP[[k]],PolyP[[k]])/dt -
        IP(name,multiply(PolyP[[k]],eps),PolyP[[k]])/2 +
        U[[(z-1)]][[k]][i,j]/(K)*
        IP(name,multiply(PolyP[[k]],multiply(PolyP[[k]],eps)),PolyP[[k]])+
        b_sum[[k]][i,j]+
        2*D*IP(name,PolyP[[k]],PolyP[[k]])/(dx)^2

    }
  }
}

#d
d <- list()
for (k in 1:(P+1)){
  d[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      d[[k]][i,j] <- 2*IP(name,PolyP[[k]],PolyP[[k]])/dt +
        IP(name,multiply(PolyP[[k]],(eps)),PolyP[[k]])/2 -
        b_sum[[k]][i,j]-
        2*D*IP(name,PolyP[[k]],PolyP[[k]])/(dy)^2
    }
  }
}

#e
e_sum1 <- list()
for (k in 1:(P+1)){
  e_sum1[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      for(l in 1:(P+1)){
        if(l!=k){
          e_sum1[[k]][i,j] <- e_sum1[[k]][i,j] + U[[(z-1)]][[l]][i,j]*
            IP(name,multiply(PolyP[[l]],(eps)),PolyP[[k]])
        }
      }
    }
  }
}

e_sum2 <- list()
for (k in 1:(P+1)){
  e_sum2[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
```

```
    for(j in 1:(ly/dy+1)){
      for(l in 1:(P+1)){
        if(l!=k){
          e_sum2[[k]][i,j] <- e_sum2[[k]][i,j] + U[[(z-1)]][[l]][i,j]^2*
            IP(name,multiply(PolyP[[l]],multiply(PolyP[[l]],eps)),PolyP[[k]])
        }
      }
      e_sum2[[k]][i,j] <- e_sum2[[k]][i,j]/K
    }
  }
}

e_sum3 <- list()
for (k in 1:(P+1)){
  e_sum3[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      for(m in 2:(P+1)){
        for(l in 1:(m-1)){
          if(l!=k & m!=k){
            e_sum3[[k]][i,j] <- e_sum3[[k]][i,j] + U[[(z-1)]][[l]][i,j]*
            U[[(z-1)]][[m]][i,j]*
              IP(name,multiply(PolyP[[l]],multiply(PolyP[[m]],eps)),PolyP[[k]])
          }
        }
      }
      e_sum3[[k]][i,j] <- 2*e_sum3[[k]][i,j]/K
    }
  }
}

e <- list()
for (k in 1:(P+1)){
  e[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      e[[k]][i,j] <- (e_sum1[[k]][i,j] - e_sum2[[k]][i,j] - e_sum3[[k]][i,j])
    }
  }
}

# rhs1
rhs1 <- list()
for (k in 1:(P+1)){
  rhs1[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
```

```
  for (i in 2:(lx/dx)){
    for(j in 2:(ly/dy)){
      rhs1[[k]][i,j] <- c[k]*U[[(z-1)]][[k]][i,j-1]+
        d[[k]][i,j]*U[[(z-1)]][[k]][i,j]+
        c[k]*U[[(z-1)]][[k]][i,j+1]+
        e[[k]][i,j]
    }
  }
}

for (k in 1:(P+1)){
  for (i in c(1,(lx/dx+1))){
    for(j in 1:(ly/dy)){
      rhs1[[k]][i,j] <- d[[k]][i,j]*U[[(z-1)]][[k]][i,j]+
        e[[k]][i,j]
    }
  }
}

for (k in 1:(P+1)){
  for(j in c(1,(ly/dy+1))){
    if(j!=1){
      for (i in 1:(lx/dx+1)){
        rhs1[[k]][i,j] <- d[[k]][i,j]*U[[(z-1)]][[k]][i,j]+
          e[[k]][i,j]
      }
    }else{
      for(i in 2:(lx/dx)){
        rhs1[[k]][i,j] <- d[[k]][i,j]*U[[(z-1)]][[k]][i,j]+
          e[[k]][i,j]
      }
    }
  }
}

## System
A <- list()
for (k in 1:(P+1)){
  A[[k]] <- list()
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    A[[k]][[l]] <- matrix(0,nrow = lx/dx+1,ncol = ly/dy+1)
  }
}

str(A)

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in 1:(lx/dx+1)){
      for(j in 1:(ly/dy+1)){
        if(i==j){
```

```
          A[[k]][[l]][i,j] <- b[[k]][i,l]
        }
      }
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in 2:(lx/dx)){
      A[[k]][[l]][i,i+1] <- a[k]
      A[[k]][[l]][i,i-1] <- a[k]
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in c(1,(lx/dx+1))){
      if(i==1){
        A[[k]][[l]][i,i+1] <- a[k]
      }else{
        A[[k]][[l]][i,i-1] <- a[k]
      }
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    U[[z]][[k]][,l] <- solve(A[[k]][[l]],rhs1[[k]][,l])
  }
}

for(k in 1:(P+1)){
  for(i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      if(U[[z]][[k]][i,j]<0){
        U[[z]][[k]][i,j] <- as.integer(rnorm(1,K,1))
      }
    }
  }
}

par(mfrow=c(1,1))
par(mar=c(4.5,4.5,4.5,4.5))
if(o==1|o==51|o==65|o==75|o==85|o==95){
png(file =
paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/New Code/Images/"
,name,"E",o,".png"))
filled.contour(seq(x1,x2,length.out = lx*cellsx+1),
seq(y1,y2,length.out = ly*cellsy+1),
U[[z]][[1]],plot.title =
title(main=paste("Expectation, Polynomial: ",name,", Time: ",o)
```

```
  ,xlab = "x",ylab = "y"),
  key.title = title(main = "Value"))
  dev.off()
  }

  EIP <- numeric(P+1)
  for(i in 1:(P+1)){
    EIP[i] <- IP(name,PolyP[[i]],PolyP[[i]])
  }



  variance[[z]] <- matrix(0,nrow = lx/dx+1,ncol = ly/dy+1)

  for(i in 2:(P+1)){
    variance[[z]] <- variance[[z]] + kronecker(U[[z]][[i]]^2,EIP[i])
  }

  sd <- sqrt(variance[[z]])

  par(mfrow=c(1,1))
  par(mar=c(4.5,4.5,4.5,4.5))
   if(o==1|o==51|o==65|o==75|o==85|o==95){
   png(file =
   paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/New Code/Images/"
   ,name,"SD",o,".png"))
   filled.contour(seq(x1,x2,length.out = lx*cellsx+1),
   seq(y1,y2,length.out = ly*cellsy+1),
   sd,plot.title =
   title(main=paste("SD, Polynomial: ",name,", Time: ",o)
   ,xlab = "x",ylab = "y"),
   key.title = title(main = "Value"))
   dev.off()
   }



}else{                # Choose system 2.
  ## Time dependent coefficients
  #f
  f <- list()
  for (k in 1:(P+1)){
    f[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
  }

  f_sum <- list()
  for(k in 1:(P+1)){
    f_sum[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
  }

  for (k in 1:(P+1)){
    for (i in 1:(lx/dx+1)){
      for(j in 1:(ly/dy+1)){
        for(l in 1:(P+1)){
```

```
        if(l!=k){
          f_sum[[k]][i,j] <- f_sum[[k]][i,j] + U[[(z-1)]][[l]][i,j]*
            IP(name,multiply(PolyP[[l]],(eps)),multiply(PolyP[[k]],PolyP[[k]]))
        }
      }
      f_sum[[k]][i,j] <- f_sum[[k]][i,j]/K
    }
  }
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      f[[k]][i,j] <- 2*IP(name,PolyP[[k]],PolyP[[k]])/dt -
        IP(name,multiply(PolyP[[k]],eps),PolyP[[k]])/2 +
        U[[(z-1)]][[k]][i,j]/(K)*
        IP(name,multiply(PolyP[[k]],multiply(PolyP[[k]],eps)),PolyP[[k]])+
        f_sum[[k]][i,j]+
        2*D*IP(name,PolyP[[k]],PolyP[[k]])/(dy)^2

    }
  }
}

# g
g <- list()
for (k in 1:(P+1)){
  g[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      g[[k]][i,j] <- 2*IP(name,PolyP[[k]],PolyP[[k]])/dt +
        IP(name,multiply(PolyP[[k]],eps),PolyP[[k]])/2 -
        f_sum[[k]][i,j]-
        2*D*IP(name,PolyP[[k]],PolyP[[k]])/(dx)^2
    }
  }
}

#h
h_sum1 <- list()
for (k in 1:(P+1)){
  h_sum1[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      for(l in 1:(P+1)){
        if(l!=k){
          h_sum1[[k]][i,j] <- h_sum1[[k]][i,j] + U[[(z-1)]][[l]][i,j]*
            IP(name,multiply(PolyP[[l]],eps),PolyP[[k]])
```

```
        }
      }
    }
  }
}

h_sum2 <- list()
for (k in 1:(P+1)){
  h_sum2[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      for(l in 1:(P+1)){
        if(l!=k){
          h_sum2[[k]][i,j] <- h_sum2[[k]][i,j] + U[[(z-1)]][[l]][i,j]^2*
            IP(name,multiply(PolyP[[l]],multiply(PolyP[[l]],eps)),PolyP[[k]])
        }
      }
      h_sum2[[k]][i,j] <- h_sum2[[k]][i,j]/K
    }
  }
}


h_sum3 <- list()
for (k in 1:(P+1)){
  h_sum3[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      for(m in 2:(P+1)){
        for(l in 1:(m-1)){
          if(l!=k & m!=k){
            h_sum3[[k]][i,j] <- h_sum3[[k]][i,j] + U[[(z-1)]][[l]][i,j]
            *U[[(z-1)]][[m]][i,j]*
            IP(name,multiply(PolyP[[l]],multiply(PolyP[[m]],eps)),PolyP[[k]])
          }
        }
      }
      h_sum3[[k]][i,j] <- 2*h_sum3[[k]][i,j]/K
    }
  }
}

h <- list()
for (k in 1:(P+1)){
  h[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
```

```
  for (i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      h[[k]][i,j] <- (h_sum1[[k]][i,j] - h_sum2[[k]][i,j] - h_sum3[[k]][i,j])
    }
  }
}

# rhs2
rhs2 <- list()
for (k in 1:(P+1)){
  rhs2[[k]] <- matrix(0,nrow = lx/dx+1,ncol=ly/dy+1)
}

for (k in 1:(P+1)){
  for (i in 2:(lx/dx)){
    for(j in 2:(ly/dy)){
      rhs2[[k]][i,j] <- a[k]*U[[(z-1)]][[k]][i-1,j]-
        g[[k]][i,j]*U[[(z-1)]][[k]][i,j]+
        a[k]*U[[(z-1)]][[k]][i+1,j]-
        h[[k]][i,j]
    }
  }
}

for (k in 1:(P+1)){
  for (i in c(1,(lx/dx+1))){
    for(j in 1:(ly/dy)){
      rhs2[[k]][i,j] <- -g[[k]][i,j]*U[[(z-1)]][[k]][i,j]
      -h[[k]][i,j]
    }
  }
}

for (k in 1:(P+1)){
  for(j in c(1,(ly/dy+1))){
    if(j!=1){
      for (i in 1:(lx/dx+1)){
        rhs2[[k]][i,j] <- -g[[k]][i,j]*U[[(z-1)]][[k]][i,j]-
          h[[k]][i,j]
      }
    }else{
      for(i in 2:(lx/dx)){
        rhs2[[k]][i,j] <- -g[[k]][i,j]*U[[(z-1)]][[k]][i,j]-
          h[[k]][i,j]
      }
    }
  }
}

## System
A1 <- list()
for (k in 1:(P+1)){
  A1[[k]] <- list()
}
```

```
for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    A1[[k]][[l]] <- matrix(0,nrow = lx/dx+1,ncol = ly/dy+1)
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in 1:(lx/dx+1)){
      for(j in 1:(ly/dy+1)){
        if(i==j){
          A1[[k]][[l]][i,j] <- -f[[k]][i,l]
        }
      }
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in 2:(lx/dx)){
      A1[[k]][[l]][i,i+1] <- c[k]
      A1[[k]][[l]][i,i-1] <- c[k]
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    for(i in c(1,(lx/dx+1))){
      if(i==1){
        A1[[k]][[l]][i,i+1] <- c[k]
      }else{
        A1[[k]][[l]][i,i-1] <- c[k]
      }
    }
  }
}

for(k in 1:(P+1)){
  for(l in 1:(ly/dy+1)){
    U[[z]][[k]][,l] <- solve(A1[[k]][[l]],rhs2[[k]][,l])
  }
}

for(k in 1:(P+1)){
  for(i in 1:(lx/dx+1)){
    for(j in 1:(ly/dy+1)){
      if(U[[z]][[k]][i,j]<0){
        U[[z]][[k]][i,j] <- as.integer(rnorm(1,K,1))
      }
    }
  }
```

```
      }

      # Mean
      par(mfrow=c(1,1))
      par(mar=c(4.5,4.5,4.5,4.5))
       if(o==60|o==70|o==80|o==90|o==92|o==94|o==96){
       png(file =
       paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/New Code/Images/"
       ,name,"E",o,".png"))
       filled.contour(seq(x1,x2,length.out = lx*cellsx+1),
       seq(y1,y2,length.out = ly*cellsy+1),U[[z]][[1]],
       plot.title =
       title(main=paste("Expectation, Polynomial: ",name,", Time: ",o)
       ,xlab = "x",ylab = "y"),
       key.title = title(main = "Value"))
       dev.off()
        }

      # SD
      EIP <- numeric(P+1)
      for(i in 1:(P+1)){
        EIP[i] <- IP(name,PolyP[[i]],PolyP[[i]])
      }
      EIP

      for(i in 2:(P+1)){
        variance[[z]] <- variance[[z]] + kronecker(U[[z]][[i]]^2,EIP[i])
      }
      sd <- sqrt(variance[[z]])
      dim(sd)
      par(mfrow=c(1,1))
      par(mar=c(4.5,4.5,4.5,4.5))
      if(o==60|o==70|o==80|o==90|o==92|o==94|o==96){
      png(file =
      paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/New Code/Images/"
      ,name,"SD",o,".png"))
      filled.contour(seq(x1,x2,length.out = lx*cellsx+1),
      seq(y1,y2,length.out = ly*cellsy+1),sd
      ,plot.title = title(main=paste("SD, Polynomial: ",name,", Time: ",o)
      ,xlab = "x",ylab = "y"),
      key.title = title(main = "Value"))
       dev.off()
         }

    } # else
    progress(z/(tp-1)*100)
    Sys.sleep(0.01)
  } # z for

par(mfrow=c(1,1))
par(mar=c(4.5,4.5,4.5,4.5))
meantemp <- numeric(length(t1))
for(i in 1:length(t1)){
    meantemp[i] <- mean(U[[i]][[1]])
```

```
}

meantemp[which(meantemp<0)] <- 0

plot(c(1:length(t1)),meantemp,type = "l",xlab = "Time",ylab = "Expected Size",main =
paste("Polynomial: ", name))
hist(meantemp)

for(i in c(1,50,87:92,100,500,1000,1500,2000)){
  png(file = paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/
  New Code/Images/",name,"H",i,".png"))
  hist(meantemp,probability=T,main = paste("Polynomial: ",name,", Time: ",i),
  xlab = "Expected Value")
  dev.off()
}

sdtemp <- numeric(length(t1))
for(i in 1:length(t1)){
    sdtemp[i] <- mean(sqrt(variance[[i]]))
  }

plot(c(1:length(t1)),sdtemp,type = "l",xlab = "Time",ylab = "Standard Deviation",
main = paste("Polynomial: ", name))
plot(sdtemp, log="y", type='h', lwd=10, lend=2)

for(i in c(1,50,87:92,100,500,1000,1500,2000)){
  png(file = paste("/home/kshitij/Desktop/College/MSc/Project/Uncertainty/
  New Code/Images/",name,"HSD",i,".png"))
  hist(sdtemp,probability=T,main = paste("Polynomial: ",name,", Time: ",i),
  xlab = "Standard Deviation")
  dev.off()
}
```

Chapter 11

Future Work

To summarize, we have solved the model for $\theta = 1$, and have tried 3 different orthogonal polynomials. In the future, we will work towards:

- Full implementation of θ logistic model for different values of θ.
- Further study concepts to be able to develop system incorporating non-constant random variable in time and space.
- Making code as general and accessible for use as possible by adding various options for boundary shape and flatness.

References

1. Orthogonal Polynomials, TU Delft
2. Probability and Measure theory, R. Ash, Catherine D.
3. Linear Algebra, K. Hoffman, R. Kunze.
4. The Wiener-Askey polynomial chaos for stochastic differential equations, D. Xiu
5. Stochastic Processes and Orthogonal Polynomials, W. Schoutens.
6. A Primer on Stochastic Galerkin methods, Paul Constantine.
7. Numerical methods for Stochastic computations, D. Xiu.
8. Individual based modeling of population growth and diffusion in discrete time, N. Tkachenko et al.
9. Polynomial Chaos Expansions for Uncertainty Quantification, Ryan G. M., 2016
10. ADI Picture: https://en.wikipedia.org/wiki/Alternating_direction_implicit_method