Software *CCOP*
Connection Coefficients for Orthogonal Polynomials

Version 1.0

Wolfram *Mathematica*® 8

TUTORIAL

July 2012

Contents

Introduction

The software *CCOP - Connection Coefficients for Orthogonal Polynomials*, written in *Mathematica*®, concerns the connection coefficients for orthogonal polynomials and canonical sequences. It allows the recursive computation of the first connection coefficients up to a certain order and can be a useful tool to derive and demonstrate closed formulas for the connection coefficients.

The software *CCOP* is related to the article [10] and is available in the library NUMERALGO of NETLIB (`http://www.netlib.org/numeralgo/`) as `na34` package. It furnish the implementation that produced the results given in [10] and also in [8, 9]. The aim of this tutorial is to present, explain and exemplify in details that implementation.

In Chapter 1 a brief description of the content of the software *CCOP*, of its structure and of the commands is given.

In Chapter 2, we recall the basic definitions and mathematical results needed to understand the subject, namely the notions of connection coefficients, orthogonal polynomials and symmetry and, also, the recurrence relation of order two satisfied by any orthogonal sequence, the resultant general recurrence relation fulfilled by the connection coefficients and the corresponding relations in the symmetrical case. This chapter reports also definitions and results already given in the paper [10].

Afterwards, we present an extensive chapter dedicated to the symbolic computation of the connection coefficients. As this work is based on the implementation of recurrence relations, the first section is devoted to this topic taking as basic example the computation of the Fibonacci numbers. In the following section, we give explicitly the commands that implement the general recurrence relation that allows the recursive computation of the first connection coefficients up to a certain fixed order: we call them the commands *CC*. Moreover, we discuss several important details in this implementation that is conceived in order to work for any example. We note that each case is determined by the recurrence coefficients (the coefficients of the recurrence relation of order 2) and the parameters of the two orthogonal polynomials sequences corresponding to the connection coefficients that we want to compute. The names of the commands that translate these elements are arguments of *CC* and must be implemented in a certain manner. Therefore, in the Section 3.3, we propose the implementation of the recurrence coefficients and the command *MOP* that computes the corresponding monic orthogonal polynomials using the above cited recurrence relation of order two. In the Section 3.4, we show how to call the commands *CC* in order to get the first connection coefficients up to a fixed order.

Also, we established a command named *verificationRCC* based on the mathematical definition of the connection coefficients, in order to verify the results produced by *CC*. From these results, we try to infer a model to the direct closed formulas for the connection coefficients. If we succeed, it is always possible to implement those formulas in a command and compare the results produced by them with those computed recursively. This comparison is implemented in a command named *verificationDCC* presented in the Section 3.5. These last verifications are very useful in order to find possible mistakes in the model and then correct it. In the following section, we treat the main task of proving that the formulas are true through a symbolic implementation furnish by the command *demonstrationDCC*. At last, we present a list of the main commands and we give, for each one, the name, a brief description, the arguments, the *Mathematica*®'s commands and other commands employed in the implementation and the results produced.

Chapter 4 is devoted to the presentation of test examples. In the first section, we follow all the steps of the methodology, giving the implementation and the results, for the Charlier polynomials [3]. This section corresponds to the case study of [10]. Next, we express the canonical polynomials in terms of the Laguerre ones. Our goal is to explain how the commands work with an example involving the canonical sequence. We present all the steps giving the corresponding results. We finish the chapter with a descriptive list of the commands developed.

Chapter 5 concerns the symmetric case that needs a specific treatment, because, often, there are different formulas for the connection coefficients corresponding to odd or even indexes. The same is also true for the recurrence coefficients as is the case of the example of the Section 5.1 of [10]. The commands *CC*, *verificationRCC* and *verificationDCC* work perfectly in the symmetric case, but the implementation of the direct closed formulas must be more complete and the command *demonstrationDCC* must be replaced by another one named *demonstrationSymDCC*. This command is discussed in the Section 5.1 and we furnish its description in the next section. Afterwards, we present as test example the generalize Hermite polynomials [3] and we finish this chapter with the description of the corresponding commands.

We insist on the fact that all the developed commands are designed to accept any example. The characteristic elements of each case, that is, the names of the commands corresponding to the recurrence coefficients and the parameters, they are transferred as arguments of *CC*, *verificationRCC* and *MOP*. The name of the command that translates the closed formulas is transferred as argument of *verificationDCC*, *demonstrationDCC* and *demonstrationSymDCC*.

For the sake of clearness, we have preferred to exemplify the commands with simple test examples. Moreover, this methodology work perfectly in the cases of Bessel, Laguerre and generalized Hermite with the canonical sequence presented in [8]; can be used to treat partially other examples as the Gegenbauer and the Jacobi families with the canonical sequence given in [9], and allows to explore new cases as the semi-classical ones treated in [9] and [10].

Chapter 1

Description of *CCOP*

In this chapter, we present a brief description of the content of the software *CCOP* describing its structure and commands.

: Software Name: *CCOP - Connection Coefficients for Orthogonal Polynomials*

: Software Version: 1.0

: Authors:

Pascal Maroni
Laboratoire Jacques-Louis Lions - CNRS,
Université Pierre et Marie Curie
Boite courrier 187, 75252 Paris cedex 05, France
Email: `maroni@ann.jussieu.fr`

Zélia da Rocha*
Departamento de Matemática - CMUP
Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre n.687, 4169 - 007 Porto, Portugal
Email: `mrdioh@fc.up.pt`
*Corresponding author.

: Language: *Mathematica*® 8.0.4.0.

: Available at: `http://www.netlib.org/numeralgo/` as `na34` package.

: Reference: P. Maroni, Z. da Rocha, Connection coefficients for orthogonal polynomials: symbolic computations, verifications and demonstrations in the *Mathematica*® language, Numerical Algorithms, 63-3 (2013) 507-520.

: Summary:

This software is constituted by the notebook *CCOP.nb*, written in the *Mathematica*® language, and is concerned with connection coefficients for orthogonal polynomials and

the canonical sequence. It contains the results of the main reference [10] and is composed by the following five section cells:

- ***References***
- ***Main Commands***
- ***Test Examples***
- ***Symmetric semi-classical polynomials of class 1***
- ***Non-symmetric semi-classical polynomials of class 1***

The ***Main Commands*** section contains the following four subsections:

- ***Main Commands for Standard Cases***
- ***Main Commands for Symmetric Cases***
- ***Main Commands for Special Cases***
- ***Canonical Case***

The ***Main Commands for Standard Cases*** subsection contains the following commands:
- *CC* for computing recursively the connection coefficients,
- *MOP* for computing recursively the orthogonal polynomials,
- *verificationRCC* for verifying the first connection coefficients computed by the command *CC* up to a fixed order,
- *verificationDCC* for comparing the first connection coefficients computed by the command *CC* with those computed by direct closed formulas up to a fixed order,
- *demonstrationDCC* for demonstrating the direct closed formulas for the connection coefficients.

The ***Main Commands for Symmetric Cases*** subsection contains the command *demonstrationSymDCC* in order to accomplish the demonstrations when the two sequences of polynomials are symmetric.

The ***Main Commands for Special Cases*** subsection contains the command *demonstrationEvenOddDCC* in order to accomplish the demonstrations when there are four different formulas for even and odd indexes of the connection coefficients.

The ***Canonical Case*** subsection contains the commands for the canonical recurrence coefficients and the canonical polynomials.

The ***Test Examples*** section contains the following three subsections:

- ***Charlier***

  The goal is to exemplify how work the software with a simple example. The Charlier case is also presented in detail in the Section 4 of [10].

- ***Laguerre and Canonical***

  The goal is to exemplify how work the software with an example involving the canonical sequence.

- ***Generalized Hermite***

  The goal is to exemplify how work the software with a symmetric example.

  Note that for symmetric and other special cases, we use the *demonstrationSymDCC* and *demonstrationEvenOddDCC* commands, instead of *demonstrationDCC*, in order to accomplish the demonstrations and we must implement other versions of the recurrence coefficients' commands.

The remainder two last sections cells are concerned with more complex cases of orthogonal polynomials. The ***Symmetric semi-classical polynomials of class 1*** section contains the results of the Sub-section 5.1 of [10] and the ***Non-symmetric semi-classical polynomials of class 1*** section contains the results of the Sub-section 5.2 of [10]. These last formulas are new.

: How to install the software:

1. Open the notebook *CCOP.nb*
2. The user must begin by executing the cells of the **Main Commands** section. After this, it is possible to execute the examples by any order. In each example, the user must follows the order of the entries.
3. To execute a cell, just click in and press simultaneously Shift and Enter.

: Note: This software has been developed in a *MacBook Pro* with *Mac OS X Version 10.6.8*

: Acknowledgements: Research funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT Fundação para a Ciência e a Tecnologia under the project PEst-C/MAT/UI0144/2011.

Chapter 2

Theoretical framework

In this chapter, we recall the basic definitions and mathematic results needed to understand the subject, namely the notions of connection coefficients, orthogonal polynomials and symmetry and, also, the recurrence relation of order two satisfied by any orthogonal sequence, the resultant general recurrence relation fulfilled by the connection coefficients and the corresponding relations in the symmetrical case. This chapter is similar to the Section 2 of [10].

Let $\mathcal{P}$ be the vector space of polynomials with coefficients in $\mathbb{C}$ and let $\mathcal{P}'$ be its dual. We denote by $\langle u, p\rangle$ the effect of $u \in \mathcal{P}'$ on $p \in \mathcal{P}$. In particular, $\langle u, x^n\rangle := (u)_n, n \geqslant 0$ represent the moments of u.

Let $\{P_n\}_{n\geqslant 0}$ be a monic polynomial sequence with $\deg P_n = n,\ n \geq 0$, that is, $P_n(x) = x^n + ...$ A form u is said regular [6, 7] if and only if there exists a MPS $\{P_n\}_{n\geqslant 0}$, such that:

$$\begin{aligned} \langle u, P_n P_m\rangle &= 0\ ,\ n \neq m\ ,\ n, m \geq 0\ , && (2.1)\\ \langle u, P_n^2\rangle &\neq 0\ ,\ n \geq 0\ . && (2.2) \end{aligned}$$

In this case, $\{P_n\}_{n\geqslant 0}$ is said regularly orthogonal with respect to u and is called a monic orthogonal polynomial sequence. The orthogonality conditions are given by (2.1) and (2.2) corresponds to the regularity conditions.

The sequence $\{P_n\}_{n\geqslant 0}$ is regularly orthogonal with respect to u if and only if [6, 7] there exist two sequences of coefficients $\{\beta_n\}_{n\geqslant 0}$ and $\{\gamma_{n+1}\}_{n\geqslant 0}$, with $\gamma_{n+1} \neq 0,\ n \geqslant 0$, such that, $\{P_n\}_{n\geqslant 0}$ satisfies the following initial conditions and recurrence relation of order 2:

$$\begin{aligned} &P_0(x) = 1, \quad P_1(x) = x - \beta_0, && (2.3)\\ &P_{n+2}(x) = (x - \beta_{n+1})P_{n+1}(x) - \gamma_{n+1}P_n(x), \quad n \geqslant 2. && (2.4) \end{aligned}$$

Furthermore, the recurrence coefficients $\{\beta_n\}_{n\geqslant 0}$ and $\{\gamma_{n+1}\}_{n\geqslant 0}$ satisfy:

$$\begin{aligned}\beta_n &= \frac{\langle u, xP_n^2(x)\rangle}{\langle u, P_n^2(x)\rangle}, \quad n \geqslant 0,\\ \gamma_{n+1} &= \frac{\langle u, P_{n+1}^2(x)\rangle}{\langle u, P_n^2(x)\rangle}, \quad n \geqslant 0.\end{aligned} \tag{2.5}$$

We remark that, from (2.3) and (2.5), the regularity conditions (2.2) are equivalent to the conditions $\gamma_{n+1} \neq 0,\ n \geqslant 0$.

As usual, we suppose that

$$\beta_n = 0,\ \gamma_{n+1} = 0,\ P_n(x) = 0,\ n < 0. \tag{2.6}$$

We recall that the recurrence coefficients of the canonical sequence $\{X_n\}_{n\geqslant 0}$, $X_n(x) = x^n$, are

$$\beta_n = 0,\ \gamma_{n+1} = 0,\ n \geqslant 0; \tag{2.7}$$

it is a non regular sequence.

Given two monic polynomial sequences $\{P_n\}_{n\geqslant 0}$ and $\{\tilde{P}_n\}_{n\geqslant 0}$ the coefficients that satisfy the equality

$$P_n(x) = \sum_{m=0}^{n} \lambda_{n,m}\tilde{P}_m(x),\ n \geqslant 0, \tag{2.8}$$

are called the connection coefficients: $\lambda_{n,m} := \lambda_{n,m}^{P\tilde{P}} := \lambda_{n,m}(P \leftarrow \tilde{P})$.

It is obvious that these coefficients exist and are unique, because the polynomials are linearly independent.

Let us suppose that the two monic polynomial sequences $\{P_n\}_{n\geqslant 0}$ and $\{\tilde{P}_n\}_{n\geqslant 0}$ are orthogonal and are given by their recurrence coefficients $\{\beta_n\}_{n\geqslant 0}$, $\{\gamma_{n+1}\}_{n\geqslant 0}$ and $\{\tilde{\beta}_n\}_{n\geqslant 0}$, $\{\tilde{\gamma}_{n+1}\}_{n\geqslant 0}$, respectively, let us consider the problem of computing and determining closed formulas for the connection coefficients.

As demonstrate in [8, 9], the connection coefficients fulfill the following boundary and initial conditions and general recurrence relation

$$\lambda_{n,m} = 0,\ n < 0 \text{ or } m < 0 \text{ or } m > n, \tag{2.9}$$

$$\lambda_{n,n} = 1,\ n \geqslant 0, \tag{2.10}$$

$$\lambda_{1,0} = \tilde{\beta}_0 - \beta_0, \tag{2.11}$$

$$\begin{aligned}\lambda_{n,m} = &\left(\tilde{\beta}_m - \beta_{n-1}\right)\lambda_{n-1,m} - \gamma_{n-1}\lambda_{n-2,m} + \tilde{\gamma}_{m+1}\lambda_{n-1,m+1} + \lambda_{n-1,m-1}\\ &,\ 0 \leqslant m \leqslant n-1\ ,\ n \geqslant 2.\end{aligned} \tag{2.12}$$

We recall that a monic polynomial sequence, $\{P_n\}_{n\geqslant 0}$, is symmetric if and only if $P_n(-x) = (-1)^n P_n(x)$, $n \geqslant 0$; if it is orthogonal the symmetry is equivalent to $\beta_n = 0,\ n \geqslant 0$ [3].

If $\{P_n\}_{n\geqslant 0}$ and $\{\tilde{P}_n\}_{n\geqslant 0}$ are two symmetric orthogonal polynomial sequences, then the corresponding connection coefficients fulfill [8, 9], for $0 \leqslant m \leqslant n-1\ ,\ n \geqslant 1$,

$$\lambda_{2n-1,2m} = 0\ ,\ \lambda_{2n,2m+1} = 0\ , \tag{2.13}$$

$$\lambda_{2n,2m} = -\gamma_{2n-1}\lambda_{2n-2,2m} + \tilde{\gamma}_{2m+1}\lambda_{2n-1,2m+1} + \lambda_{2n-1,2m-1}\ , \tag{2.14}$$

$$\lambda_{2n+1,2m+1} = -\gamma_{2n}\lambda_{2n-1,2m+1} + \tilde{\gamma}_{2m+2}\lambda_{2n,2m+2} + \lambda_{2n,2m}\ . \tag{2.15}$$

In the case of other polynomial normalizations, that is, $B_n(x) = k_n P_n(x)$, $k_n \neq 0$ and $\tilde{B}_n(x) = \tilde{k}_n \tilde{P}_n(x)$, $\tilde{k}_n \neq 0$, $n \geq 0$, the corresponding connection coefficients to consider are

$$\lambda^{B\tilde{B}}_{n,m} := k_n \lambda^{P\tilde{P}}_{n,m} \tilde{k}_m^{-1}\ \ ,\ \ \lambda^{BX}_{n,m} := k_n \lambda^{PX}_{n,m}\ \ ,\ \ \lambda^{XB}_{n,m} := \lambda^{XP}_{n,m} k_m^{-1}\ ,\ 0 \leq m \leq n,\ n \geq 0\ .$$

Chapter 3

Symbolic computation of connection coefficients

This chapter is dedicated to the symbolic computation of the connection coefficients. As this work is based on the implementation of recurrence relations, the first section is devoted to this topic taking as basic example the computation of the Fibonacci numbers. In the following section, we give explicitly the commands that implement the general recurrence relation that allows the recursive computation of the first connection coefficients up to a certain fixed order: we call them the commands *CC*. Moreover, we discuss several important details in this implementation that is conceived in order to work for any example. We note that each case is determined by the recurrence coefficients (the coefficients of the recurrence relation of order 2) and the parameters of the two orthogonal polynomials sequences corresponding to the connection coefficients that we want to compute. The names of the commands that translate these elements are arguments of *CC* and must be implemented in a certain manner. Therefore, in the Section 3.3, we propose the implementation of the recurrence coefficients and the command *MOP* that computes the corresponding monic orthogonal polynomials using the above cited recurrence relation of order two. In the Section 3.4, we show how to call the commands *CC* in order to get the first connection coefficients up to a fixed order. Also, we established a command named *verificationRCC* based on the mathematical definition of the connection coefficients, in order to verify the results produced by *CC*. From these results, we try to infer a model to the direct closed formulas for the connection coefficients. If we succeed, it is always possible to implement those formulas in a command and compare the results produced by it with those computed recursively. This comparison is implemented in a command named *verificationDCC* presented in the Section 3.5. This kind of verifications are very useful in order to find possible mistakes in the model and then correct it. In the following section, we treat the main task of proving that the formulas are true through a symbolic implementation furnish by the command *demonstrationDCC*. At last, we present a list of the main commands and we give, for each one, the name, a brief description, the arguments, the *Mathematica*®'s commands and other commands employed in the implementation and the results produced.

3.1 How to implement recurrence relations in *Mathematica*®

In this work, the computation of the connection coefficients essentially involves the implementation of recurrence relations. In the *Mathematica*® language [11, 12], this can be done easily using two different types of function definitions, as follows.

1. $f[x\_] := rhs$, define a standard function.

   In this case, the value of the function is computed every time we ask for it.

2. $f[x\_] := f[x] = rhs$, define a function which remembers values that it finds.

   In this case, *Mathematica*® never recomputes a function value, because the first time a function value is computed it is automatically stored in memory.

Let us consider the typical example of Fibonacci numbers, given by the following initial conditions and recurrence relation

$$f(0) = f(1) = 1 \ , \ f(n) = f(n-1) + f(n-2) \ , \ n \geq 2. \tag{3.1}$$

The point is that if we calculate say $f(10)$ by just applying the recursion relation over and over again, we end up having to recalculate quantities like $f(5)$ many times. In a case like this, it is therefore better just to remember all the values of the Fibonacci numbers already computed [11, 12]. This can be implemented, using a function definition of type 2, as follows.

$$f[0] = f[1] = 1;$$
$$f[n\_/; And[n \in Integer, n \geq 2]] := f[n] = f[n-1] + f[n-2];$$

There is a trade-off involved in remembering values. It is faster to find a particular value, but it takes more memory space to store all of them. We should usually define functions to remember values only if the total number of different values that will be produced is comparatively small, or the expense of recomputing them is very great [11, 12]. Of course that we can manage other types of implementation, but the preceding ones are the most straightforward to use.

3.2 Implementation of the general recurrence relation for the connection coefficients

It is clear that, the recursive computation of the connection coefficients is more difficult than the computation of the Fibonacci numbers, because the $\lambda_{n,m}$ have two indexes, the relation (2.12) is more complicated than (3.1) and involves the computation of the recurrence coefficients β_n, γ_{n+1}, $\tilde{\beta}_n$ and $\tilde{\gamma}_{n+1}$, with the corresponding parameters. In spite of this, all the considerations cited in the preceding section remain valid for the $\lambda_{n,m}$. In this work, we have used always functions that remember values. In practice, the maximal values of n in $\lambda_{n,m}$ or in P_n are around 20 or 30, so there is not a large

amount of elements to store in memory. On the other hand, recompute the connection coefficients several times could be quite expensive, if there are several parameters to consider, so the standard function definitions are not indicated to this problem. We do not have needed to do a more careful and complicated implementation, because we have obtained all the results without any problems of time or space memory.

Let us explain exactly how the implementation is done.

The boundary and initial conditions and the general recurrence relation (2.9)-(2.12) for the connection coefficients are easily implemented as follows.

$$
\begin{aligned}
&CC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_] \\
&\quad [n\_/; And[IntegerQ[n], n < 0], m\_/; IntegerQ[m]] := \\
&CC[rc, rct][p][pt][n, m] = 0;
\end{aligned}
\tag{3.2}
$$

$$
\begin{aligned}
&CC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_] \\
&\quad [n\_/; IntegerQ[n], m\_/; And[IntegerQ[m], m < 0]] := \\
&CC[rc, rct][p][pt][n, m] = 0;
\end{aligned}
\tag{3.3}
$$

$$
\begin{aligned}
&CC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_] \\
&\quad [n\_/; And[IntegerQ[n], n \geq 0], n\_/; And[IntegerQ[n], n \geq 0]] := \\
&CC[rc, rct][p][pt][n, n] = 1;
\end{aligned}
\tag{3.4}
$$

$$
\begin{aligned}
&CC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_][1, 0] := CC[rc, rct][p][pt][1, 0] = \\
&Factor[FullSimplify[\ rct[pt][0][[1]] - rc[p][0][[1]]\ ]];
\end{aligned}
\tag{3.5}
$$

$$
\begin{aligned}
&CC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_][n\_/; IntegerQ[n], m\_/; IntegerQ[m]] := \\
&CC[rc, rct][p][pt][n, m] = \\
&\quad If[And[n \geq 0, n \leq m - 1], Return[0], \\
&\quad\quad Return[\ Factor[FullSimplify[\\
&\quad\quad\quad (rct[pt][m][[1]] - rc[p][n - 1][[1]]) * CC[rc, rct][p][pt][n - 1, m] - \\
&\quad\quad\quad rc[p][n - 1][[2]] * CC[rc, rct][p][pt][n - 2, m] + \\
&\quad\quad\quad rct[pt][m + 1][[2]] * CC[rc, rct][p][pt][n - 1, m + 1] + \\
&\quad\quad\quad CC[rc, rct][p][pt][n - 1, m - 1]\ ]] \\
&\quad\quad\quad\quad]; (* \ end\ of\ Return\ *) \\
&\quad\quad]; (* \ end\ of\ If\ *)
\end{aligned}
\tag{3.6}
$$

The arguments *rc* and *rct* correspond to the names of the recurrence coefficient's commands and must be symbols; the next arguments *p* and *pt* are the names of the parameters of the sequences P and $\tilde{P}$, in such a way that, *rc[p][n]* and *rct[pt][n]* are lists corresponding to $\{\beta_n, \gamma_n\}$ and $\{\tilde{\beta}_n, \tilde{\gamma}_n\}$, respectively. To access to the first or the

second element of a list, one joins to its name [[1]] or [[2]]. The commands corresponding to *rc* and *rct* must be given before calling the commands *CC*. Note that the triple underscore next p and pt is a pattern object that can stand for any sequence of zero or more expressions [11, 12], which means that it matches for polynomial sequences without parameters as is the case of the canonical one. The commands corresponding to *rc* and *rct* must be implemented as follows

$$rc[p\_][n\_] := rc[p][n] = Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$
$$rct[pt\_][n\_] := rct[pt][n] = Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$

In spite of the fact that the computation of recurrence coefficients is not recursive, we have used functions that remember values, because each recurrence coefficient is needed many times.

After the above explanations, we think that it is easy to accept that the definitions (3.2)-(3.3), (3.4), (3.5) and (3.6) of the commands *CC* correspond to the mathematical identities (2.9), (2.10), (2.11) and (2.12), respectively. Note that we put boundary and initial conditions ahead of the general recurrence relation. This principle has been followed by *Mathematica*® in order to avoid special rules be shadowed by more general ones [11, 12].

Remark that, the commands *CC* only accept integer values for the arguments n and m. This is done by the restrictions following the patterns arguments. The aim is to avoid acceptation of absurd values for n and m. In fact, if we could give, for example, $n = 5.5$ or n symbolic in a calling statement, the recurrence relation would never match the initial conditions and would enter in an infinite recursion process.

We have added to the body of (3.5) and (3.6) the very useful *Mathematica*®'s commands *Factor* followed by *FullSimplify* in order to get the results written in the simplest factorized form [11, 12]. In examples for which the recurrence coefficients are rational functions *Factor* should be replaced by *Together*. We note that *Factor[expr]* writes *expr* as a product of minimal factors and *Together[expr]* puts terms in a sum over a common denominator and cancels factors in the result [11, 12].

3.3 Implementation of recurrence coefficients and orthogonal polynomials

Let us consider two polynomial sequences identified, for example, by the names *Ex1P* and *Ex2P*; the first one with two parameters $p1$ and $p2$ and the second one with one parameter p, for instance. We would like to compute the connection coefficients $\lambda_{n,m} := \lambda_{n,m}(Ex1P \leftarrow Ex2P)$. The commands *CC* suppose that the corresponding recurrence coefficients are implemented in commands, named $Ex1C$ and $Ex2C$, for example, as

follows.

$$Ex1C[p1\_, p2\_][n\_] := Ex1C[p1, p2][n] = \\ Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$

$$Ex2C[p\_][n\_] := Ex2C[p][n] = Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$

In that manner, these definitions return a list of two elements $\{..., ...\}$, thus $Ex1C[p1, p2][n][[1]]$ and $Ex1C[p1, p2][n][[2]]$ correspond to β_n and γ_n; and $Ex2C[p][n][[1]]$ and $Ex2C[p][n][[2]]$ correspond to $\tilde{\beta}_n$ and $\tilde{\gamma}_n$, respectively.

Note that there is no restriction on the argument n, because nearly always, we are able to give a closed formula for the recurrence coefficients valid for all nonnegative integer n, so n can be a symbol argument or a numeric expression in the calling statement ($Ex1C[p1, p2][5]$ or $Ex1C[p1, p2][n]$, for example). Furthermore, the symbolic expressions of the recurrence coefficients are necessary in order to accomplish the demonstrations of the closed formulas for the connections coefficients.

The monic orthogonal polynomials are recursively computed using the identities (2.3)-(2.4) and are implemented in the following command.

$$MOP[rc\_Symbol][p\_\_\_][n\_/; And[IntegerQ[n], n < 0], x\_] := \\ MOP[rc][p][n, x] = 0; \tag{3.7}$$

$$MOP[rc\_Symbol][p\_\_\_][0, x\_] := MOP[rc][p][0, x] = 1; \tag{3.8}$$

$$MOP[rc\_Symbol][p\_\_\_][1, x\_] := MOP[rc][p][1, x] = \\ \quad FullSimplify[\ x - rc[0][[1]]\ ]; \tag{3.9}$$

$$MOP[rc\_Symbol][p\_\_\_][n\_/; And[IntegerQ[n], n \geq 2], x\_] := \\ MOP[rc][p][n, x] = \\ Collect[\ FullSimplify[\ (x - rc[p][n-1][[1]]) * MOP[rc][p][n-1, x] - \\ \quad rc[p][n-1][[2]] * MOP[rc][p][n-2, x]\ ], \\ \quad x,\ Factor\ ]; \tag{3.10}$$

To the sake of implementation, it is necessary to consider the shift $n \leftarrow n-2$ in (2.4) in order to have the index n isolated on the left hand side of the relation.

As set before, the arguments rc and p correspond to the name of the recurrence coefficients' command and the name of the parameters of the sequence P. As usual n is the degree and x is the variable in $P_n(x)$.

It is easy to see that the commands MOP (3.7), (3.8)-(3.9) and (3.10) correspond to the mathematical identities (2.6), (2.3) and (2.4), respectively. Remark that, in the

implementation of *MOP*, we have used functions that remember values and, in the calling statements of *MOP*, n must be a fixed integer and x should be a symbol or a numeric expression.

In order to get the polynomials simplified and written in the canonical base with factorized coefficients, we have used the *FullSimplify* and *Collect Mathematica*®'s commands. We inform that $Collect[expr, x, command]$ groups together powers of x in $expr$ and applies the $command$ to the corresponding coefficients. In rational cases, *Factor* should be replaced by *Together* [11, 12].

The commands that define the polynomials of the specified examples with which we are treating can be easily implemented in one line calling the definition *MOP* as follows.

$$Ex1P[p1\_, p2\_][n\_, x\_] := Ex1P[p1, p2][n, x] = MOP[Ex1C][p1, p2][n, x];$$

$$Ex2P[p\_][n\_, x\_] := Ex2P[p][n, x] = MOP[Ex2C][p][n, x];$$

To get the polynomials of degree 5 for example, we furnish

$$Ex1P[p1, p2][5, x] \qquad\qquad Ex2P[p][5, x]$$

If we want to consider special values to the parameters, for example, $p1 = 0$, $p2 = 1$ and $p = -1$, we call

$$Ex1P[0, 1][5, x] \qquad\qquad Ex2P[-1][5, x]$$

To evaluate the preceding in $x = 0$, for example, we give

$$Ex1P[0, 1][5, 0] \qquad\qquad Ex2P[-1][5, 0]$$

3.4 Getting and verifying results for connection coefficients computed recursively

Supposing the structure of commands previously establish, the statement to get the connection coefficients up to the order 10, for example, that is, to get $\lambda_{n,m} := \lambda_{n,m}(Ex1 \leftarrow Ex2)$, for $n = 0, ..., 10,\ m = 0, ..., n$, is

$$Table[\ CC[Ex1C, Ex2C][p1, p2][\tilde{p}][n, m],\ \{n, 0, 10\}, \{m, 0, n\}\ ]$$

where $Table[expr, \{i, imin, imax\}, \{j, jmin, jmax\}]$ generates a double list of the values of $expr$ when i runs from $imin$ to $imax$ and j runs from $jmin$ to $jmax$. The list associated with i is outermost [11, 12].

We should make some verifications in order to test our implementation. A crucial one is based on the definition of connection coefficients given by the identity (2.8), which can be reproduced with our commands and verified up to the a fixed value of $n = nmax$. This is the task of the following command.

$$
\begin{aligned}
&verificationRCC[rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_] \\
&\qquad [nmax\_/; And[IntegerQ[nmax], nmax \geq 0] := \\
&Module[\{x, answer = Table[True, \{n, 0, nmax\}]\}, \\
&\quad Table[\ FullSimplify[\ MOP[rc][p][n, x] - \\
&\qquad \sum_{m=0}^{n} CC[rc, rct][p][pt][n, m] * MOP[rct][pt][m, x]\ ] === 0 \\
&\qquad , \{n, 0, nmax\}\ ] === answer \\
&\quad]; (* \ end\ of\ Module \ *)
\end{aligned}
$$

We inform that, in *Mathematica*®, in order to inquire if two entities a and b are identical, we can give $a === b$ and have *True* or *False* as answer. Often, it is more efficient to ask if their difference is 0 entering $a - b === 0$ or $FullSimplify[a - b] === 0$. In the case of *verificationRCC*, a and b correspond to the left and right sides of the definition (2.8) expressed in terms of commands. We point out that, if *Mathematica*® is not able to decide about the veracity of the requested identity typed with $===$, it always yields *False* [11, 12]. This is a crucial point in the automatic demonstrations of the models of the connection coefficients.

In the case of our examples, the calling statement to get this verification up to 20, for instance, is

$$verificationRCC[Ex1C, Ex2C][p1, p2][\tilde{p}][20]$$

and we should have *True* as answer.

In order to get the connection coefficients corresponding to two different polynomial sequences concerning the same family *Ex1P*, but with different parameters $[p1, p2]$ and $[\tilde{p}1, \tilde{p}2]$, we call

$$Table[\ CC[Ex1C, Ex1C][p1, p2][\tilde{p}1, \tilde{p}2][n, m],\ \{n, 0, 10\}, \{m, 0, n\}\ ]$$

The corresponding verification up to 20 is

$$verificationRCC[Ex1C, Ex1C][p1, p2][\tilde{p}1, \tilde{p}2][20]$$

and we should have *True* as answer.

3.5 Getting and verifying results for connection coefficients computed by direct closed formulas

In [8, 9, 10], it is shown that, in several important cases, we are able to infer from the table of the first results produced by the definitions *CC*, what are the mathematical direct closed formulas of the connection coefficients, for all n and m. Then, we can

translate this model in a command and compare the connection coefficients given by it with those produced by the recursive computations of CC, for the first values of n up to a fixed order $nmax$. This comparison is implemented in the following command.

$$\begin{aligned}
&verificationDCC[dcc_Symbol][rc_Symbol, rct_Symbol][p___][pt___] \\
&\qquad [nmax_/; And[nmax \in Integer, nmax \geq 0]] := \\
&Module[\{answer = Table[True, \{n, 0, nmax\}, \{m, 0, n\}]\}, \\
&\qquad Table[\ FullSimplify[\ dcc[p][pt][n, m] - CC[rc, rct][p][pt][n, m]\ ] === 0 \\
&\qquad\qquad , \{n, 0, nmax\}, \{m, 0, n\}\ ] === answer\ ];
\end{aligned}$$

The argument dcc must be a symbol and is the name of the command that implements the direct closed formulas valid for integers n and m, it returns $\lambda_{n,m}$ and must be implemented as follows

$$dcc[p\_][pt\_][n\_, m\_] := dcc[p][pt][n, m] = Module[\ \{...\},\ ...;\ Return[...]\ ];$$

In the case of the examples *Ex1* and *Ex2*, we establish a command of the following type for the closed formulas.

$$\begin{aligned}
&Ex1Ex2DCC[p1_, p2_][pt_][n_, m_] := Ex1Ex2DCC[p1, p2][pt][n, m] = \\
&\qquad Module[\ \{...\},\ ...;\ Return[...]\ ];
\end{aligned}$$

Note that the parameters of the two sequences must figure separately as arguments. Remark, also, that there are no restriction on n and m, because they can be fixed integers or symbols.

The comparison up to 20 is executed as follows

$$verificationDCC[Ex1Ex2DCC][Ex1C, Ex2C][p1, p2][\tilde{p}][20]$$

and we should have *True* as answer. Of course, this does not constitute a proof of the direct formula.

3.6 Automatic demonstration of closed formulas for the connection coefficients

The mathematical demonstration corresponds to show that the direct closed formulas for the connection coefficients $\lambda_{n,m}$, that should be given for all integers n and m such that $0 \leq m \leq n-1$, $n \geq 1$, satisfy the initial condition (2.11) and are solutions of the recurrence relation (2.12). In principle, this proof can be done totally in *Mathematica*® using the next command, if the formulas for the recurrence coefficients $\{\beta_n\}$, $\{\gamma_{n+1}\}$ and $\{\tilde{\beta}_n\}$, $\{\tilde{\gamma}_{n+1}\}$ are available for all nonnegative integers and are implemented for n

symbolic, and the command *FullSimplify* is able to accomplish the necessary simplifications. For the sake of implementation, we consider separately the recurrence relation for $m = 0$ and $n = 1$, $m = 0$ and $n = 2$, and $m = 0$ and $n \geqslant 3$; and for $m = n - 1$ and $n \geqslant 2$, because for all these values the boundary conditions (2.9) and (2.10) appear in the relation (2.12) and the commands that implement the direct formulas can not satisfy them.

$$
\begin{aligned}
&demonstrationDCC[dcc\_Symbol][rc\_Symbol, rct\_Symbol][p\_\_\_][pt\_\_\_] \\
&\qquad\qquad [n\_Symbol, m\_Symbol] := \\
&And[\\
&(* \; m = 0 \;,\; n = 1 \; *) \\
&FullSimplify[\; dcc[p][pt][1,0] - (\; rct[pt][0][[1]] - rc[p][0][[1]] \;) \;] === 0 \;, \\
&(* \; m = 0 \;,\; n = 2 \; *) \\
&FullSimplify[\\
&\quad dcc[p][pt][2,0] - (rct[pt][0][[1]] - rc[p][1][[1]]) * \\
&\quad dcc[p][pt][1,0] + rc[p][1][[2]] - rct[pt][1][[2]] \;] === 0 \;, \\
&(* \; m = 0 \;,\; n \geq 3 \; *) \\
&FullSimplify[\\
&\quad dcc[p][pt][n,0] - (rct[pt][0][[1]] - rc[p][n-1][[1]]) * \\
&\quad dcc[p][pt][n-1,0] + rc[p][n-1][[2]] * dCC[p][pt][n-2,0] - \\
&\quad rct[pt][1][[2]] * dcc[p][pt][n-1,1], \\
&\quad Assumptions \to And[Element[n, Integers], n \geq 3] \;] === 0 \;, \\
&(* \; m = n - 1 \;,\; n \geq 2 \; *) \\
&FullSimplify[\\
&\quad dcc[p][pt][n,n-1] - (rct[pt][n-1][[1]] - rc[p][n-1][[1]]) - \\
&\quad dcc[p][pt][n-1,n-2], \\
&\quad Assumptions \to And[Element[n, Integers], n \geq 2] \;] === 0 \;, \\
&(* \; 0 < m < n - 1 \;,\; n \geq 3 \; *) \\
&FullSimplify[\; dcc[p][pt][n,m] - \\
&\quad \big(\; (rct[pt][m][[1]] - rc[p][n-1][[1]]) * dcc[p][pt][n-1][m] - \\
&\qquad rc[p][n-1][[2]] * dcc[p][pt][n-2,m] + \\
&\qquad rct[pt][m+1][[2]] * dcc[p][pt][n-1,m+1] + dcc[p][pt][n-1,m-1] \; \big) \;, \\
&\quad Assumptions \to \\
&\quad And[Element[n, Integers], Element[m, Integers], n \geqslant 3, m > 0, m < n - 1] \\
&\qquad\qquad] === 0; \\
&\;]; (* \; end\ of\ And \; *)
\end{aligned}
$$

The argument *dcc* is the name of the command corresponding to the direct closed

formulas valid for n and m symbolic, it returns $\lambda_{n,m}$ and must be implemented as follows

$$dcc[p\_][pt\_][n\_, m\_] := dcc[p][pt][n, m] = Module[\ \{...\},\ ...;\ Return[...]\ ];$$

The arguments *rc* and *rct* are the names of the commands corresponding to the recurrence coefficients, now, valid for n symbolic, they return $\{\beta_n, \gamma_n\}$ and $\{\tilde{\beta}_n, \tilde{\gamma}_n\}$, respectively.

It is easy to realize that *demonstrationDCC* is a translation of the identities (2.11) and (2.12) in terms of commands. Remark that the *Assumptions* option of *FullSimplify* informs that the symbols n and m represent integers corresponding to the specified values. Depending on the examples, *FullSimplify* can achieve the demonstrations without any *Assumptions*. The user should test several variants of *demonstrationDCC* and compare their results before come to a conclusion.

When *demonstrationDCC* produces *True*, this means that the direct closed formulas translated by the command *dcc* are correct and *Mathematica*® achieve the demonstration; when *demonstrationDCC* produces *False*, this can mean that those formulas are wrong, or that they are true but *Mathematica*® can not accomplish the simplifications and can not decides about the veracity of the identities presented in *demonstrationDCC*.

In the case of the examples *Ex1* and *Ex2*, supposing that the direct closed formulas are implemented in a command as follows

$$Ex1Ex2DCC[p1\_, p2\_][pt\_][n\_, m\_] := Module[\ \{...\},\ ...;\ Return[...]\ ]\ ;$$

the automatic demonstration corresponds to get *True* as answer to the following entry

$$demonstrationDCC[Ex1Ex2CC][Ex1C, Ex2C][p1, p2][\tilde{p}][n, m]$$

3.7 Main commands' description

In this section, we furnish a list with a description of the main commands implemented for the standard case, they are: *CC*, *MOP*, *verificationRCC*, *verificationDCC* and *demonstrationDCC*. For each command, we give the following information: name, brief description, arguments, *Mathematica*®'s commands and other commands employed in the implementation and the results produced.

- **Arguments** *rc*, *rct*, *p* and *pt* of the commands listed in the sequel.

  - *rc* and *rct* are the names of the commands that define the recurrence coefficients of the two polynomials sequences P and $\tilde{P}$; they must be symbols.

  - *rc* and *rct* must be implemented as follows

$$rc[p\_][n\_] := rc[p][n] = Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$
$$rct[pt\_[n\_] := rct[pt][n] = Module[\{...\},\ ...\ ; Return[\{..., ...\}]\ ];$$

and they must return $\{\beta_n, \gamma_n\}$ and $\{\tilde{\beta}_n, \tilde{\gamma}_n\}$, respectively.

- p and pt are the sequences of parameters of P and $\tilde{P}$.

- $CC[rc, rct][p][pt][n, m]$

  Description:

  $CC[rc, rct][p][pt][n, m]$ computes recursively the connection coefficient, $\lambda_{n,m} := \lambda_{n,m}(P \leftarrow \tilde{P})$, defined in (2.8), using the boundary and initial conditions (2.9)-(2.11) and the recurrence relation (2.12).

  Arguments:

  - rc, rct, p and pt described before.

  - n and m must be integers.

  ***Mathematica*®'s commands used:**

  - *Factor* or *Together*, *FullSimplify*.

  Result:

  - $\lambda_{n,m}$.

- $MOP[rc][p][n, x]$

  Description:

  $MOP[rc][p][n, x]$ computes recursively the monic orthogonal polynomial, $P_n(x)$, of degree n in the variable x, using the initial conditions (2) and the recurrence relation (3).

  Arguments:

  - rc and p described before.

  - n must be an integer.

  - x should be a symbol or a numeric expression.

  ***Mathematica*®'s commands used:**

  - *Collect*, *Factor* or *Together*, *Simplify* or *FullSimplify*.

  Result:

  - $P_n(x)$.

- **Argument** $nmax$ of the commands listed in the sequel.

  - $nmax$ must be a non negative integer.

- $verificationRCC[rc, rct][p][pt][nmax]$

  Description:

  $verificationRCC[rc, rct][p][pt][nmax]$ makes a verification of the first connection coefficients, $\lambda_{n,m} := \lambda_{n,m}(P \leftarrow \tilde{P})$, computed recursively by the command CC up

to an index $n = nmax$. *verificationRCC* is based on the definition (2.8) of the connection coefficients.

Arguments:

- rc, rct, p, pt and $nmax$ described before.

***Mathematica*®'s commands used:**

- *FullSimplify*.

Other commands used:

- *CC* and *MOP*.

Result:

- Returns *True* if the the verification is correct, returns *False* otherwise.

- **Argument** *dcc* of the commands listed in the sequel.

  - *dcc* is the name of the command that implements the direct closed formulas; it must be a symbol.

  - *dcc* must be defined as follows

$$dcc[p\_][pt\_][n\_, m\_] := dcc[p][pt][n, m] = Module[\ \{...\},\ ...; Return[...]\ ];$$

  and must returns $\lambda_{n,m}$.

- *verificationDCC*$[dcc][rc, rct][p][pt][nmax]$

  Description:

  verificationDCC$[dcc][rc, rct][p][pt][nmax]$ makes a comparison between the values of the connection coefficients, $\lambda_{n,m} := \lambda_{n,m}(P \leftarrow \tilde{P})$, computed by the command *CC* and the ones computed by the command *dcc* that implements the direct closed formulas, this, up to the index $n = nmax$.

  Arguments:

  - *dcc*, rc, rct, p, pt and $nmax$ described before.

  ***Mathematica*®'s commands used:**

  - *FullSimplify*.

  Other commands used:

  - *CC*.

  Result:

  - Returns *True* if the verification is correct, returns *False* otherwise.

- *demonstrationDCC*$[dcc][rc, rct][p][pt][n, m]$

  Description:

demonstrationDCC[*dcc*][rc, rct][p][pt][n, m] tries to demonstrate the direct closed formulas for the connection coefficients $\lambda_{n,m} := \lambda_{n,m}(P \leftarrow \tilde{P})$, for every n and m.

Arguments:

- *dcc*, rc, rct, p, pt described before.

- n and m must be symbols and represent integers such that $0 \leq m \leq n-1$, $n \geq 1$.

***Mathematica*®'s commands used:**

- *FullSimplify* with or without *Assumptions*.

Result:

- Returns *True*, if the direct closed formulas translated by the command *dcc* are true and *Mathematica*® achieves the demonstration, returns *False* otherwise.

Chapter 4

Test examples

This chapter is dedicated to the presentation of some test examples. In the first section, we follow all the steps of the methodology, giving the implementation and the results, for the Charlier polynomials [3]. This section corresponds to the Section 4 of [10]. Next, we express the canonical polynomials in terms of the Laguerre ones. Our goal is to explain how the commands work with an example involving the canonical sequence. We present all the steps giving the corresponding results. We finish the chapter with a descriptive list of the commands developed.

4.1 Charlier polynomials

Let us see how the commands we have developed work and what results they produce in the simple case of the classical discrete monic Charlier polynomials $\{P_n(\alpha,.)\}_{n\geq 0}$ with parameter α [3]. The Charlier recurrence coefficients

$$\beta_n(\alpha) = n+\alpha \ , \ n\geq 0 \ ; \ \gamma_n(\alpha) = n\alpha \ , \ n\geq 1 \ , \ \alpha\neq 0, \tag{4.1}$$

are implement in the following command.

$$\begin{aligned} &CharlierC[\alpha\_][n\_] := CharlierC[\alpha][n] = \\ &\quad If[\ And[NumericQ[n], n<0],\ Return[\{0,0\}],\ Return[\{n+\alpha, \alpha * n\}]\ ]; \end{aligned}$$

The monic Charlier polynomials are defined by the command *MOP* as follows

$$CharlierP[\alpha\_][n\_, x\_] := CharlierP[\alpha][n,x] = MOP[CharlierC][\alpha][n,x];$$

The connection coefficients $\lambda_{n,m} := \lambda_{n,m}(P(\alpha;-) \leftarrow P(\tilde{\alpha};-))$ are computed recursively up to $n=6$, for example, by the next calling statement of the command *CC*.

$$\begin{aligned} &In[\ ] := Table[\ CC[CharlierC, CharlierC][\alpha][\tilde{\alpha}][n,m], \{n,0,6\}, \{m,0,n\}\ ]// \\ &\qquad TableForm \\ &Out[\ ]//TableForm = \end{aligned}$$

$$
\begin{array}{llllll}
1 & & & & & \\
-\alpha+\tilde{\alpha} & 1 & & & & \\
(\alpha-\tilde{\alpha})^2 & -2(\alpha-\tilde{\alpha}) & 1 & & & \\
-(\alpha-\tilde{\alpha})^3 & 3(\alpha-\tilde{\alpha})^2 & -3(\alpha-\tilde{\alpha}) & 1 & & \\
(\alpha-\tilde{\alpha})^4 & -4(\alpha-\tilde{\alpha})^3 & 6(\alpha-\tilde{\alpha})^2 & -4(\alpha-\tilde{\alpha}) & 1 & \\
-(\alpha-\tilde{\alpha})^5 & 5(\alpha-\tilde{\alpha})^4 & -10(\alpha-\tilde{\alpha})^3 & 10(\alpha-\tilde{\alpha})^2 & -5(\alpha-\tilde{\alpha}) & 1 \\
(\alpha-\tilde{\alpha})^6 & -6(\alpha-\tilde{\alpha})^5 & 15(\alpha-\tilde{\alpha})^4 & -20(\alpha-\tilde{\alpha})^3 & 15(\alpha-\tilde{\alpha})^2 & -6(\alpha-\tilde{\alpha}) \quad 1
\end{array}
$$

Now, we can verify these results and the next ones up to $nmax = 20$, for example, calling

$In[\ ] := Timing[\ verificationRCC[CharlierC, CharlierC][\alpha, \tilde{\alpha}][20]\ ]$

and we get the answer

$Out[\ ] = \{75.0289, True\}$

Note that *Timing[expr]* evaluates *expr*, and returns a list of the time in seconds used together with the result obtained [11, 12].

The observation of the above results getting by the commands CC allows us to infer the following direct closed formula for the connection coefficients

$$\lambda_{n,m} = (-1)^{n-m}\binom{n}{m}(\alpha-\tilde{\alpha})^{n-m},\ 0 \le m \le n-1,\ n \ge 1, \tag{4.2}$$

which can be implement in a command as follows

$$
\begin{array}{l}
CharlierDCC[\alpha\_][\alpha t\_][n\_, m\_] := \\
(-1)\,\hat{}\,(n-m) * Binomial[n, m] * (\alpha - \alpha t)\,\hat{}\,(n-m);
\end{array}
$$

In order to compare the results given by this command with those produced by CC up to $nmax = 20$, for example, we do

$In[\ ] := Timing[\ verificationDCC[CharlierDCC][CharlierC, CharlierC][\alpha][\tilde{\alpha}][20]\ ]$
$Out[\ ] = \{0.009881, True\}$

The automatic demonstration of the formula (4.2) is achieved in *Mathematica*® doing

$In[\ ] := Timing[$
$\quad demonstrationDCC[CharlierDCC][CharlierC, CharlierC][\alpha][\tilde{\alpha}][n, m]\ ]$
$Out[\ ] = \{0.36858, True\}$

We remember that the formula (4.2) is well known and can be found in several references; see, among others, [5].

4.2 Canonical and Laguerre polynomials

Let us express the canonical sequence in terms of the classical monic Laguerre polynomials. For that purpose, we need to recall the Laguerre recurrence coefficients [3],

$$\beta_n(\alpha) = 2n + \alpha + 1\ ,\ \gamma_{n+1}(\alpha) = (n+1)(n+\alpha+1),\ \alpha \neq -n,\ n \geq 0, \tag{4.3}$$

which can be implemented in the following command.

$LaguerreC[\alpha\_][n\_] := LaguerreC[\alpha][n] =$
$If[\ And[NumericQ[n], n < 0],\ Return[\{0, 0\}],\ Return[\{2 * n + \alpha + 1, n * (n + \alpha)\}]\ ];$

The recurrence coefficients of the canonical sequence given in (2.7) and the canonical polynomials are easily implemented as follows

$CanonicalC[\ ][n\_] := CanonicalC[\ ][n] = \{0, 0\};$

$CanonicalP[\ ][n\_, x\_] := CanonicalP[\ ][n, x] = MOP[CanonicalC][\ ][n, x];$

The connection coefficients $\lambda_{n,m} := \lambda_{n,m}(X \leftarrow P(\tilde{\alpha}; -))$ are computed recursively up to $n = 4$, for example, by the next calling statement of the command *CC*.

$In[\ ] := Table[\ CC[CanonicalC, LaguerreC][\ ][\tilde{\alpha}][n, m], \{n, 0, 4\}, \{m, 0, n\}\ ]//$
$TableForm$

$Out[\ ]//TableForm =$

1				
$(1+\tilde{\alpha})$	1			
$(1+\tilde{\alpha})(2+\tilde{\alpha})$	$2(2+\tilde{\alpha})$	1		
$(1+\tilde{\alpha})(2+\tilde{\alpha})(3+\tilde{\alpha})$	$3(2+\tilde{\alpha})(3+\tilde{\alpha})$	$3(3+\tilde{\alpha})$	1	
$(1+\tilde{\alpha})(2+\tilde{\alpha})(3+\tilde{\alpha})(4+\tilde{\alpha})$	$4(2+\tilde{\alpha})(3+\tilde{\alpha})(4+\tilde{\alpha})$	$6(3+\tilde{\alpha})(4+\tilde{\alpha})$	$4(4+\tilde{\alpha})$	1

Now, we can verify these results and the next ones up to $nmax = 20$, for example, calling

$In[\ ] := Timing[\ verificationRCC[CanonicalC, LaguerreC][\ ][\tilde{\alpha}][20]\ ]$

and we get the answer

$Out[\ ] = \{106.082, True\}$

The observation of the above table allows us to infer the following direct closed formula for the connection coefficients

$$\lambda_{n,m} = \binom{n}{m} \prod_{k=0}^{n-m-1} (\tilde{\alpha} + n - k),\ 0 \leq m \leq n-1,\ n \geq 1, \tag{4.4}$$

which can be implement in a command as follows

$$CanonicalLaguerreDCC[\,][\alpha t\_][n\_, m\_] := Binomial[n, m] * \prod_{k=0}^{n-m-1} (\alpha t + m - k);$$

In order to compare the results given by this command with those produced by CC up to $nmax = 20$, for example, we do

$In[\,] := Timing[$
$verificationDCC[\ CanonicalLaguerreDCC[CanonicalC, LaguerreC][\,][\tilde{\alpha}][20]]\ ]$
$Out[\,] = \{0.017948, True\}$

The automatic demonstration of the formula (4.4) is achieved in *Mathematica*® doing

$In[\,] := Timing[$
$demonstrationDCC[\ CanonicalLaguerreDCC[CanonicalC, LaguerreC][\,][\tilde{\alpha}][n, m]]\ ]$
$Out[\,] = \{0.372348, True\}$

We remember that the formula (4.4) is well known and can be found in several references; see, among others, [1, 2, 8]. About the table of recursive results, when $\tilde{\alpha} = 0$, see [4].

4.3 Commands' description of test examples

In this section, we give a descriptive list, in the same terms as before, of the commands implemented in the section of test examples.

Commands for the Charlier polynomials:

CharlierC, ***CharlierP*** and ***CharlierDCC***

- **Argument** α of the commands listed in the sequel.
  - α is a parameter ($\alpha \neq 0$).
  - If $\alpha = 0$, then Charlier polynomials are not regular, $\gamma_n = 0, n \geq 1$.
  - α should be a symbol or a numeric expression.

- $CharlierC[\alpha][n]$

  Description:

  $CharlierC[\alpha][n]$ is the n-th recurrence coefficients, $\{\beta_n, \gamma_n\}$, of the monic Charlier polynomials (4.1).

  Arguments:

- n should be a symbol or an integer.

Result:

- $\{\beta_n, \gamma_n\}$, if $n > 0$.

- $\{\beta_n, 0\}$, if n=0.

- $\{0, 0\}$, if $n < 0$.

- *CharlierP*$[\alpha][n, x]$

  Description:

  CharlierP$[\alpha][n, x]$ is the monic Charlier polynomial of parameter α of degree n in the variable x: $P_n(x)$.

  Arguments:

  - n must be an integer.

  - x should be a symbol or a numeric expression.

  Commands used:

  - *CharlierC*, *MOP*.

  Result:

  - $P_n(x)$

- *CharlierDCC*$[\alpha][\tilde{\alpha}][n, m]$

  Description:

  CharlierDCC$[\alpha][\tilde{\alpha}][n, m]$ computes the connection coefficient $\lambda_{n,m} := \lambda_{n,m}(P(\alpha, -) \leftarrow \tilde{P}(\tilde{\alpha}, -))$, where P notes the monic Charlier polynomials, using the direct closed formula (4.2).

  Arguments:

  - α and $\tilde{\alpha}$ are the parameters of P and $\tilde{P}$.

  - n and m should be symbols or integers.

  Result:

  - $\lambda_{n,m}$.

Commands for the canonical polynomials:

CanonicalC and ***CanonicalP***

- *CanonicalC*$[\,][n]$

  Description:

  CanonicalC$[\,][n]$ is the n-th recurrence coefficients, $\{\beta_n, \gamma_n\}$, of the canonical polynomials (2.7).

Arguments:

- n should be a symbol or an integer.

Result:

- $\{0, 0\}$.

- *CanonicalP*[ ][n, x]

  Description:

  CanonicalP[ ][n, x] is the canonical polynomial, $X_n(x)$, of degree n in the variable x.

  Arguments:

  - n must be an integer.

  - x should be a symbol or a numeric expression.

  Commands used:

  - *CanonicalC*, *MOP*.

  Result:

  - x^n, if $n \geq 0$.

  - 0, if $n < 0$.

Commands for the Laguerre polynomials:

LaguerreC, ***LaguerreP*** and ***CanonicalLaguerreDCC***

- **Argument** α of the commands listed in the sequel.

  - α is a parameter ($\alpha \neq -n, n > 0$).

  - If $\alpha = -n, n > 0$, then Laguerre polynomials are not regular, $\gamma_n = 0, n \geq 1$.

  - α should be a symbol or a numeric expression.

- *LaguerreC*[α][n]

  Description:

  LaguerreC[α][n] is the n-th recurrence coefficients, $\{\beta_n, \gamma_n\}$, of the monic Laguerre polynomials (4.3).

  Arguments:

  - n should be a symbol or an integer.

  Result:

  - $\{\beta_n, \gamma_n\}$, if $n > 0$.

  - $\{\beta_n, 0\}$, if $n = 0$.

  - $\{0, 0\}$, if $n < 0$.

- *LaguerreP*$[\alpha][n, x]$

  Description:

  LaguerreP$[\alpha][n, x]$ is the monic Laguerre polynomial of parameter α of degree n in the variable x: $P_n(x)$.

  Arguments:

  - n must be an integer.

  - x should be a symbol or a numeric expression.

  Commands used:

  - *LaguerreC*, *MOP*.

  Result:

  - $P_n(x)$

- *CanonicalLaguerreDCC*$[\ ][\tilde{\alpha}][n, m]$

  Description:

  CanonicalLaguerreDCC$[\ ][\tilde{\alpha}][n, m]$ computes the connection coefficient $\lambda_{n,m} := \lambda_{n,m}(X \leftarrow P(\tilde{\alpha}, -))$, where X notes the canonical polynomials and P notes the monic Laguerre polynomials, using the direct closed formula (4.4).

  Arguments:

  - $\tilde{\alpha}$ is the parameter of $\tilde{P}$.

  - n and m should be symbols or integers.

  Result:

  - $\lambda_{n,m}$

Chapter 5

Symbolic computation of connection coefficients in the symmetric case

This chapter concerns the symmetric case that needs a specific treatment, because, often, there are different formulas for the connection coefficients corresponding to odd or even indexes. The same is also true for the recurrence coefficients as is the case of the example of the Section 5.1 of [10]. The commands *CC*, *verificationRCC* and *verificationDCC* work perfectly in the symmetric case, but the implementation of the direct closed formulas must be more complete and the command *demonstrationDCC* must be replaced by another one named *demonstrationSymDCC*. This command is discussed in the Section 5.1 and we furnish its description in the next section. Afterwards, we present as test example the generalize Hermite polynomials [3] and we finish this chapter with the description of the corresponding commands.

5.1 Automatic demonstration of closed formulas for the connection coefficients

It is quite clear that the general commands *CC*, *verificationRCC* and *verificationDCC* work perfectly in the case the two polynomials sequences are symmetric. Nevertheless, we could develop more efficient commands due to the simplifications introduced by the equalities (2.13), creating a command named *SymCC* for implementing the recurrence relations (2.14)-(2.15) and the commands *verificationSymRCC* and *verificationSymDCC* for accomplishing the corresponding verifications. We did not do that because the gain in execution time is negligible.

On the other hand, in the symmetric case, the command *demonstrationDCC* can not work, because, often, the direct closed formulas for the connection coefficients are given separately for even and odd integer indexes, that is, there are different formulas for $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$. Therefore, the commands that implement these formulas

must be defined for the arguments $[2*n\_, 2*m\_]$ and $[2*n\_+1, 2*m\_+1]$. In calling statements, the arguments corresponding to the indexes $2n+2, 2m-2$ and $2n-1, 2m-1$, for example, should be given as $[2*(n+1), 2*(m-1)]$ and $[2*(n-1)+1, 2*(m-1)+1]$ in order to match with the definitions.

We next present the command *demonstrationSymDCC* which allows to demonstrate that the direct closed formulas for $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$ satisfy the recurrence relations (2.14) and (2.15). For the sake of implementation, we consider separately the recurrence relation (2.14) for $m = 0$ and $n = 1$, and $m = 0$ and $n \geqslant 2$, and $m = n-1$ and $n \geqslant 2$; also, we consider separately the recurrence relation (2.15) for $m = 0$ and $n = 1$, and $m = n-1$ and $n \geq 2$, because for all these values the boundary conditions (2.9) and (2.10) appear in those relations and the commands that implements the direct formulas can not satisfy them.

$$
\begin{aligned}
&\mathit{demonstrationSymDCC}[\mathit{symdcc\_Symbol}][\mathit{rc\_Symbol}, \mathit{rct\_Symbol}][p\_\_\_][pt\_\_\_] \\
&\qquad\qquad [\mathit{n\_Symbol}, \mathit{m\_Symbol}] := \\
&\mathit{And}[\\
&(*\ m = 0\ ,\ n = 1\ -\ n \text{ even }, \ m \text{ even }*) \\
&\mathit{FullSimplify}[\ \mathit{symdcc}[p][pt][2,0] + \mathit{rc}[p][1][[2]] - \mathit{rct}[pt][1][[2]]\ ] === 0\ , \\
&(*\ m = 0\ ,\ n \geqslant 2\ -\ n \text{ even }, \ m \text{ even }*) \\
&\mathit{FullSimplify}[\\
&\qquad \mathit{symdcc}[p][pt][2*n, 0] + \\
&\qquad \mathit{rc}[p][1+2*(n-1)][[2]] * \mathit{symdcc}[p][pt][2*(n-1), 0] - \\
&\qquad \mathit{rct}[pt][1][[2]] * \mathit{symdcc}[p][pt][1+2*(n-1), 1]\ , \\
&\qquad \mathit{Assumptions} \to \mathit{And}[\mathit{Element}[n, \mathit{Integers}], n \geq 2]\ ] === 0\ , \\
&(*\ m = n-1\ ,\ n \geq 2\ -\ n \text{ even }, \ m \text{ even }*) \\
&\mathit{FullSimplify}[\\
&\qquad \mathit{symdcc}[p][pt][2*n, 2*(n-1)] + \mathit{rc}[p][1+2*(n-1)][[2]] - \\
&\qquad \mathit{rct}[pt][1+2*(n-1)][[2]] - \\
&\qquad \mathit{symdcc}[p][pt][1+2*(n-1), 1+2*(n-2)]\ , \\
&\qquad \mathit{Assumptions} \to \mathit{And}[\mathit{Element}[n, \mathit{Integers}], n \geq 2]\ ] === 0\ , \\
&(*\ 0 < m < n-1\ ,\ n \geq 3\ -\ n \text{ even }, \ m \text{ even }*) \\
&\mathit{FullSimplify}[\\
&\qquad \mathit{symdcc}[p][pt][2*n, 2*m] - \\
&\qquad \big(- \mathit{rc}[p][2*(n-1)+1][[2]] * \mathit{symdcc}[p][pt][2*(n-1), 2*m] + \\
&\qquad \mathit{rct}[pt][2*m+1][[2]] * \mathit{symdcc}[p][pt][2*(n-1)+1, 2*m+1] + \\
&\qquad \mathit{symdcc}[p][pt][2*(n-1)+1, 2*(m-1)+1]\ \big)\ , \\
&\qquad \mathit{Assumptions} \to \mathit{And}[\mathit{Element}[n, \mathit{Integers}], \mathit{Element}[m, \mathit{Integers}], \\
&\qquad\qquad n \geqslant 3, m > 0, m < n-1]\ ] === 0\ ,
\end{aligned}
$$

$$
\begin{aligned}
&(* \; m = 0 \;,\; n = 1 \;-\; n \text{ odd} \;,\; m \text{ odd} \; *)\\
&\mathit{FullSimplify}[\; \mathit{symdcc}[p][pt][3,1] + rc[p][2][[2]] -\\
&\qquad\qquad rct[pt][2][[2]] - \mathit{symdcc}[p][pt][2,0] \;] === 0 \;,\\
&(* \; m = n-1 \;,\; n \geqslant 2 \;-\; n \text{ odd} \;,\; m \text{ odd} \; *)\\
&\mathit{FullSimplify}[\\
&\qquad\qquad \mathit{symdcc}[p][pt][1+2*n, 1+2*(n-1)] + rc[p][2*n][[2]] -\\
&\qquad\qquad rct[pt][2*n][[2]] - \mathit{symdcc}[p][pt][2*n, 2*(n-1)],\\
&\qquad\qquad \mathit{Assumptions} \to \mathit{And}[\mathit{Element}[n, \mathit{Integers}], n \geqslant 2] \;] === 0 \;,\\
&(* \; 0 \leqslant m < n-1 \;,\; n \geqslant 2 \;-\; n \text{ odd} \;,\; m \text{ odd} \; *)\\
&\mathit{FullSimplify}[\\
&\qquad\qquad \mathit{symdcc}[p][pt][2*n+1, 2*m+1] -\\
&\qquad\qquad \big(- rc[p][2*n][[2]] * \mathit{symdcc}[p][pt][2*(n-1)+1, 2*m+1] +\\
&\qquad\qquad rct[pt][2*(m+1)][[2]] * \mathit{symdcc}[p][pt][2*n, 2*(m+1)] +\\
&\qquad\qquad \mathit{symdcc}[p][pt][2*n, 2*m] \big) \;,\\
&\qquad\qquad \mathit{Assumptions} \to \mathit{And}[\mathit{Element}[n, \mathit{Integers}], \mathit{Element}[m, \mathit{Integers}],\\
&\qquad\qquad\qquad\qquad n \geqslant 2, m \geqslant 0, m < n-1] \;] === 0 \;,\\
&\quad]; (* \; \textit{end of And} \; *)
\end{aligned}
$$

The argument *symdcc* is the name of the command corresponding to the direct closed formulas valid for n and m symbolic, it must be implemented as follows

$$
\begin{aligned}
&\mathit{symdcc}[p\_][pt\_][2*n\_, 2*m\_] := .... \;;\\
&\mathit{symdcc}[p\_][pt\_][2*n\_+1, 2*m\_+1] := .... \;;
\end{aligned}
$$

and must return $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$, respectively.

The arguments *rc* and *rct* are the names of the commands corresponding to the recurrence coefficients valid for n symbolic, they must be implement as follows

$$
\begin{aligned}
&rc[p\_][2*n\_] := .... \;;\; rc[p\_][2*n\_+1] := .... \;;\\
&rct[pt\_][2*n\_] := .... \;;\; rct[pt\_][2*n\_+1] := .... \;;
\end{aligned}
$$

and must return $\{0, \gamma_{2n}\}$ and $\{0, \gamma_{2n+1}\}$, and $\{0, \tilde{\gamma}_{2n}\}$ and $\{0, \tilde{\gamma}_{2n+1}\}$, respectively.

Is easy to see that *demonstrationSymDCC* is a translation of the recurrence relations (2.14) and (2.15) in terms of commands. Remark that the *Assumptions* option of *FullSimplify* informs that the symbols n and m represent integers corresponding to the specified values. Depending on the examples, *FullSimplify* can achieve the demonstrations without any *Assumptions*. The user should test several variants of *demonstrationSymDCC* and compare their results before come to a conclusion.

When *demonstrationSymDCC* produces *True*, it means that the direct closed formulas translated by the command *symdcc* are correct and *Mathematica*® achieves the demonstration; when *demonstrationSymDCC* furnishs *False*, it can mean that those formulas are wrong, or that they are true, but *Mathematica*® cannot accomplish the simplifications and can not decides about the veracity of the identities presented in *demonstrationSymDCC*.

Let us consider two symmetric examples identified by the names *SymEx1* and *SymEx2*. We would like to compute the $\lambda_{n,m} := \lambda_{n,m}(SymEx1P \leftarrow SymEx2P)$. Supposing that *SymEx1C* and *SymEx2C* are the names of the recurrence coefficients' commands and are implemented as follows

$$SymEx1C[p1\_, p2\_][2 * n\_] := Module[\ \{...\}, ...;\ Return[\{0, ...\}]\ ]\ ;$$
$$SymEx1C[p1\_, p2\_][2 * n\_ + 1] := Module[\ \{...\}, ...;\ Return[\{0, ...\}]\ ]\ ;$$

$$SymEx2C[pt\_][2 * n\_] := Module[\ \{...\}, ...;\ Return[\{0, ...\}]\ ]\ ;$$
$$SymEx2C[pt\_][2 * n\_ + 1] := Module[\ \{...\}, ...;\ Return[\{0, ...\}]\ ]\ ;$$

and supposing that the direct closed formulas for $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$ are implemented in the following manner

$$SymEx1SymEx2DCC[p1\_, p2\_][pt\_][2 * n\_, 2 * m\_] := Module[\ \{...\}, ...;\ Return[...]\ ];$$
$$SymEx1SymEx2DCC[p1\_, p2\_][pt\_][2 * n\_ + 1, 2 * m\_ + 1] :=$$
$$Module[\ \{...\}, ...;\ Return[...]\ ]\ ;$$

the automatic demonstration corresponds to get *True* as answer to the next entry

$$demonstrationSymDCC[SymEx1SymEx2DCC][SymEx1C, SymEx2C][p1, p2][\tilde{p}][n, m]$$

5.2 Main command's description

As before, we give a description of the main command implemented in the symmetric case: *demonstrationSymDCC*.

- $demonstrationSymDCC[symdcc][rc, rct][p][pt][n, m]$

  Description:

  $demonstrationSymDCC[symdcc][rc, rct][p][pt][n, m]$ tries to demonstrate the direct closed formulas for the connection coefficients $\lambda_{n,m} := \lambda_{n,m}(P \leftarrow \tilde{P})$, when P and $\tilde{P}$ are symmetric polynomials sequences implementing the recurrence relations (2.14) and (2.15).

Arguments:

- *symdcc* is the name of the command that implements the direct closed formulas, it must be a symbol.

- *symdcc* must be defined as follows

$$symdcc[p\_][pt\_][2*n\_, 2*m\_] := Module[\ \{...\},\ ...;\ Return[....]\ ]\ ;$$
$$symdcc[p\_][pt\_][2*n\_+1, 2*m\_+1] := ....\ ;$$
$$Module[\ \{...\},\ ...;\ Return[....]\ ]\ ;$$

and must return $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$, respectively.

- *rc* and *rct* are the names of the commands that define the recurrence coefficients of the two symmetric polynomials sequences P and $\tilde{P}$; they must be symbols.

- *rc* and *rct* must be defined as follows

$$rc[p\_][2*n\_] := Module[\ \{...\},\ ...;\ Return[\{0, ...\}\ ]\ ;$$
$$rc[p\_][2*n\_+1] := Module[\ \{...\}, ...; Return[\{0, ...\}\ ]\ ;$$

$$rct[pt\_][2*n\_] := Module[\ \{...\}, ...; Return[\{0, ...\}\ ]\ ;$$
$$rct[pt\_][2*n\_+1] := Module[\ \{...\}, ...; Return[\{0, ...\}\ ]\ ;$$

and must return $\{0, \gamma_{2n}\}$ and $\{0, \gamma_{2n+1}\}$, and $\{0, \tilde{\gamma}_{2n}\}$ and $\{0, \tilde{\gamma}_{2n+1}\}$, respectively.

- p and pt are the sequences of parameters of P and $\tilde{P}$.

- n and m are symbols and represent non negative integers such that $0 \leq m \leq n-1$, $n \geq 1$.

***Mathematica*®'s commands used:**

- *FullSimplify* with or without *Assumptions*.

Result:

- Returns *True*, if the direct closed formulas translated by the command *symdcc* are true and *Mathematica*® achieves the demonstration, returns *False* otherwise.

5.3 Test example

Let us consider the semi-classical generalized monic Hermite sequence $\{P_n(\mu; -)\}_{n\geq 0}$ with parameter μ, which is symmetric. When $\mu = 0$, we recover the classical Hermite sequence. The recurrence coefficients [3]

$$\beta_n = 0, n \geq 0\ ;\ \gamma_n := \gamma_n(\mu) = \frac{1}{2}\left(n + \mu(1 + (-1)^{n-1})\right), \mu \neq -n - \frac{1}{2},\ n \geq 1, \qquad (5.1)$$

are implemented in the following command.

$GenHermiteC[\mu\_][n\_] := GenHermiteC[\mu][n] =$
$If[\ And[NumericQ[n], n \leq 0],\ Return[0, 0],\ Return[0, (n + \mu * (1 + (-1)^{(n-1)}))/2]\ ];$

We remark that in this example, it is possible to write the recurrence coefficients only in one formula, therefore the implementation is more simple.

We can compute recursively the connection coefficients $\lambda_{n,m} := \lambda_{n,m}(P(\mu; -) \leftarrow P(\tilde{\mu}; -))$ up to $n = 5$, for example, by the next call of the command CC.

$In[\ ] := Table[CC[GenHermiteC, GenHermiteC][\mu][\tilde{\mu}][n, m], \{n, 0, 5\}, \{m, 0, n\}]//$
$TableForm$
$Out[\ ]//TableForm =$

1					
0	1				
$-\mu + \tilde{\mu}$	0	1			
0	$-\mu + \tilde{\mu}$	0	1		
$(-\mu + \tilde{\mu})(1 + \mu - \tilde{\mu})$	0	$-2(\mu - \tilde{\mu})$	0	1	
0	$(-\mu + \tilde{\mu})(1 + \mu - \tilde{\mu})$	0	$-2(\mu - \tilde{\mu})$	0	1

We can verify these results and the next ones up to $nmax = 20$, for example, calling

$In[\ ] := Timing[\ verificationRCC[GenHermiteC, GenHermiteC][\mu, \tilde{\mu}][20]\ ]$

and we get the answer

$Out[\ ] = \{27.7808, True\}$

From the above table and some more results, we can easily infer the direct closed formulas for the connection coefficients valid for $0 \leq m \leq n - 1$ and $n \geq 1$; they are

$$\lambda_{2n,2m} = (-1)^{n+m} \binom{n}{m} \prod_{k=0}^{n-m-1} (\mu - \tilde{\mu} + k)\ , \tag{5.2}$$

$$\lambda_{2n+1,2m+1} = \lambda_{2n,2m}\ ; \tag{5.3}$$

which are implemented as follows.

$$
\begin{aligned}
&GenHermiteDCC[\mu\_][\mu t\_][n\_/; And[IntegerQ[n], n \geq 0], \\
&\qquad\qquad m\_/; And[IntegerQ[m], m \geq 0]] := \\
&GenHermiteDCC[\mu][\mu t][n, m] = \\
&Module[\{n1, m1\}, \\
&If[\ Or[\ And[EvenQ[n], OddQ[m]], And[OddQ[n], EvenQ[m]]\ ], Return[0]\ ]; \\
&n1 = Quotient[n, 2];\ m1 = Quotient[m, 2]; \\
&Return[\\
&(-1)\hat{}\,(n1 - m1) * Binomial[n1, m1] * Product[\mu - \mu t + k, \{k, 0, n1 - m1 - 1\}] \\
&\qquad\ ]; (* \ end\ of\ Return\ *) \\
&\qquad\ ]; (* \ end\ of\ Module\ *)
\end{aligned}
$$

We inform that $EvenQ[exp]$ gives $True$, if exp is an even integer, and $False$ otherwise; $OddQ$ works in a similar way; $Quotient[n, m]$ returns the integer quotient of n and m [11, 12]. We remark that these commands could not furnish the desired answer if n or m are symbols. In principle, $EvenQ[exp]$ and $OddQ[exp]$ return always $False$, if exp is a symbol and $Quotient[n, m]$ gives no answer if n or m are symbols. Thus the above implementation for $GenHermiteDCC$ should work only for non-negative values of n and m as is insured by the restrictions following the corresponding pattern arguments.

In order to compare the results produced by $GenHermiteDCC$ with those obtained from CC up to $nmax = 20$, for example, we do

$$
\begin{aligned}
&In[\ ] := Timing[\ verificationDCC[GenHermiteDCC] \\
&\qquad\qquad [GenHermiteC, GenHermiteC][\mu][\tilde{\mu}][20]\ ]
\end{aligned}
$$

$$
Out[\ ] = \{0.01712, True\}
$$

In order to demonstrate the formulas (5.2) and (5.3) for every n and m, through the command $demonstrationSymDCC$, the above implementation of $GenHermiteDCC$ is not adequate and we should add the next definitions for $GenHermiteDCC$ that will be called in $demonstrationSymDCC$ for n and m symbolic. We alert that a $Symbol$ restriction following the patterns of the arguments n and m must not figure in these commands and that they do not allow computations for fixed values of n and m.

$$
\begin{aligned}
&GenHermiteDCC[\mu\_][\mu t\_][2 * n\_, 2 * m\_] := \\
&(-1)\hat{}\,(n - m) * Binomial[n, m] * Product[\mu - \mu t + i, \{i, 0, n - m - 1\}];
\end{aligned}
$$

$$
\begin{aligned}
&GenHermiteDCC[\mu\_][\mu t\_][2 * n\_ + 1, 2 * m\_ + 1] := \\
&GenHermiteDCC[\mu][\mu t][2 * n, 2 * m];
\end{aligned}
$$

Therefore the automatic demonstration is achieved doing

$In[\ ] := Timing[\ demonstrationSymDCC[GenHermiteDCC]$
$[GenHermiteC, GenHermiteC][\mu][\tilde{\mu}][n, m]\ ]$

$Out[\ ] = \{0.91238, True\}$

We remember that the formulas (5.2) and (5.3) are well known and they are available in several references; see, among others, [8].

5.4 Commands for the generalized Hermite polynomials

In this section, we give a descriptive list, in the same terms as before, of the commands implemented in order to test the symmetric case of generalized Hermite family, they are: *GenHermiteC*, *GenHermiteP* and *GenHermiteDCC*.

- **Argument** μ of the commands listed in the sequel.

  - μ is a parameter ($\mu \neq -n - \frac{1}{2}, n \geq 0$).

  - If $\mu = -n - \frac{1}{2}, n \geq 0$, then generalized Hermite polynomials are not regular, $\gamma_{2n+1} = 0, n \geq 1$.

  - μ should be a symbol or a numeric expression.

- *GenHermiteC*$[\mu][n]$

  Description:

  GenHermiteC$[\mu][n]$ is the n-th recurrence coefficients, $\{0, \gamma_n\}$, of the monic generalized Hermite polynomials (5.1).

  - n should be a symbol or an integer.

  Result:

  - $\{0, \gamma_n\}$, if $n > 0$.

  - $\{0, 0\}$, if $n \leq 0$.

- *GenHermiteP*$[\mu][n, x]$

  Description:

  $GenHermiteP[\mu][n, x]$ is the monic generalized Hermite polynomial of parameter μ of degree n in the variable x: $P_n(x)$.

  Arguments:

  - n must be an integer.

  - x should be a symbol or a numeric expression.

  Commands used:

- *GenHermiteC*, *MOP*.

Result:

- $P_n(x)$

- *GenHermiteDCC*$[\mu][\tilde{\mu}][n, m]$

  Description:

  GenHermiteDCC$[\mu][\tilde{\mu}][n, m]$ computes the connection coefficient $\lambda_{n,m} := \lambda_{n,m}(P(\mu; -) \leftarrow P(\tilde{\mu}; -))$, for non-negative integers n and m, where P notes the monic generalized Hermite polynomials, using the direct closed formulas (5.2) and (5.3).

  Arguments:

  - μ and $\tilde{\mu}$ are the parameters of P and $\tilde{P} = P(\tilde{\mu}; -)$.

  - n and m must be **non-negative integers.**

  Result:

  - $\lambda_{n,m}$.

- *GenHermiteDCC*$[\mu][\tilde{\mu}][2n, 2m]$ and *GenHermiteDCC*$[\mu][\tilde{\mu}][2n + 1, 2m + 1]$

  Description:

  GenHermiteDCC$[\mu][\tilde{\mu}][2n, 2m]$ and *GenHermiteDCC*$[\mu][\tilde{\mu}][2n+1, 2m+1]$ translate the definitions (5.2) and (5.3) of $\lambda_{2n,2m}$ and $\lambda_{2n+1,2m+1}$, for n and m symbolic, in order to be used in calling statements of the *demonstrationSymCC* command.

  Arguments:

  - μ and $\tilde{\mu}$ are the parameters of P and $\tilde{P} = P(\tilde{\mu}, -)$.

  - n and m should be **symbols**.

  Results:

  - $\lambda_{2n,2m}$, $\lambda_{2n+1,2m+1}$.

Bibliography

[1] R. Askey, Orthogonal Polynomials and Special Functions. CBMS-NSF Regional Conference Series, Appl. Math., 21, SIAM, Philadelphia, PA, 1975.

[2] G. E. Andrews, R. Askey, R. Roy, Special Functions, Cambridge University Press, 71, 1999.

[3] T.S. Chihara, An Introduction to Orthogonal Polynomials, Gordon and Breach, New York,1978.

[4] U. W. Hochstrasser, Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, In M. Abramowitz, I. A. Stegun, eds., N.Y., 1970.

[5] S. Lewanowicz, Recurrence relations for the connection coefficients of orthogonal polynomials of a discrete variable, J. Comput. Appl. Math., 76 (1996) 213-229.

[6] P. Maroni, Variations around classical orthogonal polynomials. Connected problems, J. Comput. Appl. Math., 48, 133-155, 1993.

[7] P. Maroni, Fonctions eulériennes. Polynômes orthogonaux classiques. Techniques de l'Ingénieur, traité Généralités (Sciences Fondamentales), 1994.

[8] P. Maroni, Z. da Rocha, Connection coefficients for orthogonal polynomials and the canonical sequence, Preprints CMUP, Centro de Matemática da Universidade do Porto, 29, 1-18, 2007. `http://cmup.fc.up.pt/cmup/v2/frames/publications.htm`

[9] P. Maroni, Z. da Rocha, Connection coefficients between orthogonal polynomials and the canonical sequence: an approach based on symbolic computation, Numerical Algorithms, 47-3 (2008) 291-314.

[10] P. Maroni, Z. da Rocha, Connection coefficients for orthogonal polynomials: symbolic computations, verifications and demonstrations in the *Mathematica* language, Numerical Algorithms, 63-3 (2013) 507-520.

[11] Stephen Wolfram, The Mathematica Book, 4th ed., Wolfram Media/Cambridge University Press, 1999.

[12] Virtual Book - Wolfram *Mathematica*® 8.0.4.0.