International Journal of Pure and Applied Mathematics
Volume 34 No. 2 2007, 267-278

GENERATING MATRICES FOR FIBONACCI, LUCAS AND SPECIAL ORTHOGONAL POLYNOMIALS WITH ALGORITHMS

Mustafa Aşci[1] §, Bayram Çekım[2], Dursun Taşci[3]
[1,2,3]Department of Mathematics
Faculty of Sciences and Arts
University of Gazi
Teknikokullar, Ankara, 06500, TURKEY
[1]e-mail: masci@gazi.edu.tr
[2]e-mail: bayramcekim@gazi.edu.tr
[3]e-mail: dtasci@gazi.edu.tr

Abstract: In this paper we get the Fibonacci polynomials for special cases of Sturm Liouville boundary value problems. We show the orthogonallity of Fibonacci polynomials. In Section 3 by using the definition of $n \times n$ Hessenberg matrices we give the generating matrices of the Fiboniacci and Lucas polynomials and for the special orthogonal polynomials. In the last section we give two algorithms for finding the Fibonacci and Lucas polynomials.

AMS Subject Classification: 15A15, 11B39, 15A36
Key Words: Fibonacci polynomial, Sturm Liouville algorithm

1. Introduction

The Fibonacci sequence, $\{F_n\}$, is defined by the recurrence relation, for $n \geq 1$

$$F_{n+1} = F_n + F_{n-1}\,, \tag{1.1}$$

where $F_0 = F_1 = 1$. The Lucas sequence, $\{L_n\}$, is defined by the recurrence

Received: November 2, 2006

§Correspondence author

relation, for $n \geq 1$

$$L_{n+1} = L_n + L_{n-1}\,, \tag{1.2}$$

where $L_0 = 2$, $L_1 = 1$.

Large classes of polynomials can be defined by Fibonacci-like recurrence relation, and yield Fibonacci numbers. Such polynomials, called the Fibonacci polynomials, were studied in 1883 by the Belgian mathematician Eugene Charles Catalan and the German mathematician E. Jacobsthal. The polynomials $f_n(x)$ studied by Catalan are defined by the recurrence relation

$$f_n(x) = xf_{n-1}(x) + f_{n-2}(x)\,, \tag{1.3}$$

where $f_1(x) = 1$, $f_2(x) = x$, and $n \geq 3$. The Fibonacci polynomials studied by Jacobsthal were defined by

$$J_n(x) = J_{n-1}(x) + xJ_{n-2}(x)\,, \tag{1.4}$$

where $J_1(x) = 1 = J_2(x)$.

Lucas polynomials $L_n(x)$, originally studied in 1970 by Bicknell, are defined by

$$L_n(x) = xL_{n-1}(x) + L_{n-2}(x)\,, \tag{1.5}$$

where $L_0(x) = 2$, $L_1(x) = x$ and $n \geq 2$.

A differential equation defined on the interval $a \leq x \leq b$ having the form of

$$\frac{d}{dx}\left[p(x)\frac{dy}{dx}\right] + [q(x) + \lambda r(x)]\, y = 0$$

and the boundary conditions

$$\begin{aligned} a_1y(a) + a_2y'(a) &= 0\,, \\ b_1y(b) + b_2y'(b) &= 0\,, \end{aligned}$$

is called as Sturm-Liouville boundary value problem or Sturm-Liouville system where $p(x) > 0$, $q(x)$, the weighting function $r(x) > 0$ are given functions; a_1, a_2, b_1, b_2 are given constants; and the eigenvalue λ is an unspecified parameter.

For $a = -1$, $b = 1$, $p(x) = \sqrt{1-x^2}$, $q(x) = 0$, $r(x) = \frac{1}{\sqrt{1-x^2}}$ and $\lambda = n^2$ the Sturm-Liouville equation becomes the Chebyshev's differential equation

$$(1-x^2)y'' - xy' + n^2y = 0\,,$$

which is defined on $-1 < x < 1$. The solutions of the Chebyshev's differential equation with $n = 0, 1, 2, 3, ...$ is called Chebyshev polynomials $T_n(x)$ which

form a complete orthogonal set on the interval $-1 < x < 1$ with respect to $r(x) = \frac{1}{\sqrt{1-x^2}}$

For $a = -\infty,\ b = \infty,\ p(x) = e^{\frac{-x^2}{2}},\ q(x) = 0,\ r(x) = e^{\frac{-x^2}{2}}$ and $\lambda = 2n$ the Sturm-Liouville equation becomes the Hermite's differential equation

$$y'' - 2xy' + 2ny = 0\,,$$

which is defined on $-\infty < x < \infty$. The solutions of the Hermite's differential equation with $n = 0, 1, 2, 3, ...$ is called Hermite polynomials $H_n(x)$ which form a complete orthogonal set on the interval $-\infty < x < \infty$ with respect to $r(x) = e^{\frac{-x^2}{2}}$

We can get Legendre polynomials, Bessel functions, Laguerre polynomials for some special cases of the Sturm-Liouville boundary value problem.

2. Computations on Fibonacci Differential Equation

The ordinary differential equation

$$(x^2 + 4)y'' + 3xy' + \lambda y = 0 \tag{2.1}$$

is known as Fibonacci differential equation. When we solve the equation the solutions are the Fibonacci polynomials. Let us show that the eigenvalue λ is $(1 - n^2)$:

By the serial solution of ODE, suppose that $y = \sum_{n=0}^{\infty} a_n x^n$ then $y' = \sum_{n=0}^{\infty} n a_n x^{n-1}$ and $y'' = \sum_{n=0}^{\infty} n(n-1)a_n x^{n-2}$.

Substituting these values in (2.1)

$$\begin{aligned}
&(x^2 + 4)\sum_{n=0}^{\infty} n(n-1)a_n x^{n-2} + 3x\sum_{n=0}^{\infty} n a_n x^{n-1} + \lambda\sum_{n=0}^{\infty} a_n x^n = 0\,,\\
&\sum_{n=0}^{\infty} n(n-1)a_n x^n + 4\sum_{n=2}^{\infty} n(n-1)a_n x^{n-2} + 3x\sum_{n=0}^{\infty} n a_n x^{n-1}\\
&\qquad + \lambda\sum_{n=0}^{\infty} a_n x^n = 0\,,\\
&\sum_{n=0}^{\infty} n(n-1)a_n x^n + 4\sum_{n=0}^{\infty}(n+1)(n+2)a_{n+2}x^n + 3\sum_{n=0}^{\infty} n a_n x^n
\end{aligned}$$

$$+\lambda\sum_{n=0}^{\infty}a_nx^n=0\,,$$

$$\sum_{n=0}^{\infty}(n(n-1)a_n+4(n+1)(n+2)a_{n+2}+3na_n+\lambda a_n)x^n=0\,.$$

For this solution we get

$$a_{n+2}=\frac{-(n^2+2n+\lambda)}{4(n+1)(n+2)}a_n\ ,\quad n\geq 0\,,$$

or

$$a_{n+1}=\frac{-(n^2-1+\lambda)}{4n(n+1)}a_{n-1}\ ,\quad n\geq 1\,,$$

then we get $\lambda=1-n^2$

When we get $\lambda=1-n^2$ the solutions of the ODE $(x^2+4)y''+3xy'+(1-n^2)y=0$ are the Fibonacci polynomials.

Now we show that Fibonacci polynomials are orthogonal with respect to the weight function $\sqrt[2]{x^2+4}$.

Theorem 1. *Let F_n and F_m be Fibonacci polynomials as defined in* (1.3). *Then we get*

$$\int_{-2i}^{2i}\sqrt[2]{x^2+4}F_mF_ndx=\begin{cases}0, & m\neq n\,,\\ (-1)^{n+1}2i\pi, & m=n\,.\end{cases}$$

Proof. Case 1. $m\neq n$. Since F_m and F_n are the solutions of ODE (2.1)

$$(x^2+4)F_n''+3xF_n'+(1-n^2)F_n=0\,.$$

In this equation product this equation with $\sqrt[2]{x^2+4}F_m$

$$(x^2+4)F_m''+3xF_m'+(1-n^2)F_m=0$$

multiply this equation with $\sqrt[2]{x^2+4}F_n$

$$(x^2+4)^{\frac{3}{2}}F_n''F_m+3x\sqrt[2]{x^2+4}F_mF_n'+(1-n^2)\sqrt[2]{x^2+4}F_mF_n=0\,,\qquad(2.2)$$

$$(x^2+4)^{\frac{3}{2}}F_m''F_n+3x\sqrt[2]{x^2+4}F_nF_m'+(1-m^2)\sqrt[2]{x^2+4}F_mF_n=0\,.\qquad(2.3)$$

Subtract (2.2) and (2.3) we get

$$(x^2+4)^{\frac{3}{2}}\left[F_n''F_m-F_m''F_n\right]+3x\sqrt[2]{x^2+4}\left[F_mF_n'-F_nF_m'\right]$$

$$- \sqrt[2]{x^2+4} F_m F_n (n^2-m^2) = 0\,,$$

$$\frac{d}{dx}\left\{(x^2+4)^{\frac{3}{2}}\left[F_m F_n' - F_n F_m'\right]\right\} = \sqrt[2]{x^2+4} F_m F_n (n^2-m^2)\,.$$

Integrate this equations from $-2i$ to $2i$

$$\int_{-2i}^{2i} \frac{d}{dx}\left\{(x^2+4)^{\frac{3}{2}}\left[F_m F_n' - F_n F_m'\right]\right\} dx = (n^2-m^2)\int_{-2i}^{2i} \sqrt[2]{x^2+4} F_m F_n dx\,,$$

$$(x^2+4)^{\frac{3}{2}}\left[F_m F_n' - F_n F_m'\right]\Big|_{-2i}^{2i} = (n^2-m^2)\int_{-2i}^{2i} \sqrt[2]{x^2+4} F_m F_n dx\,,$$

$$(n^2-m^2)\int_{-2i}^{2i} \sqrt[2]{x^2+4} F_m F_n dx = 0\,,$$

when $n \neq m$ we get

$$\int_{-2i}^{2i} \sqrt[2]{x^2+4} F_m F_n dx = 0\,.$$

Case 2. $m = n$. This case is computed by a short *MATLAB* algorithm easily. □

3. Generating Matrices for Fibonacci, Lucas Polynomials and Special Functions

A lower Hessenberg matrix, A, is an $n \times n$ matrix, where $a_{j,k} = 0$ whenever $k > j+1$ and $a_{j,j+1} \neq 0$ for some j. That is, all entries above the superdiagonal are 0 but the matrix is not lower triangular. Throughout this paper we will refer to the following lower Hessenberg matrix

$$M_n = \begin{bmatrix} m_{1,1} & m_{1,2} & 0 & \dots & \dots & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & \ddots & \ddots & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & m_{n-1,n} \\ m_{n,1} & m_{n,2} & \dots & \dots & m_{n,n-1} & m_{n,n} \end{bmatrix}. \tag{3.1}$$

We will consider the sequence $\{\det M_n, n \geq 1\}$. Our result is stated in the following theorem.

Theorem 2. *Let M_n be as above for all $n \geq 1$ and define* $\det M_0 = 1$ *Then* $\{\det M_n, n \geq 0\}$ *satisfies:*

$$\det M_0 = 1, \quad \det M_1 = m_{1,1}$$

and

$$\det M_n = m_{n,n}.\det M_{n-1} + \sum_{r=1}^{n-1} \left((-1)^{n-r} m_{n,r} \prod_{j=r}^{n-1} m_{j,j+1}.\det M_{r-1} \right), \quad n \geq 2.$$

Proof. It is proved in [2]. □

We can obtain the family of matrices $\{F_n\}$ by letting $m_{j,j} = 1$ and $m_{j,j+1} = m_{j+1,j} = i = \sqrt{-1}, 1 \leq j \leq n-1$ in the above theorem. Then $\det M_1 = 1$, $\det M_2 = 2$ and $\det M_n = \det M_{n-1} + \det M_{n-2}$, which is exactly the Fibonacci recurrence.

By using above theorem we get the following results by determinant computations.

Corollary 1. *Let F_n be the $n \times n$ Hessenberg matrix defined as in* (3.1). *For special cases as below we get* $\det F_n = F_n(x)$

$$F_n = \begin{bmatrix} 1 & 0 & 0 & ... & ... & 0 \\ 0 & x & i & 0 & \ddots & 0 \\ 0 & i & x & i & \ddots & 0 \\ 0 & 0 & i & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & i \\ 0 & 0 & ... & 0 & i & x \end{bmatrix},$$

where $F_n(x)$ is the n-th Fibonacci polynomial.

Corollary 2. *Let L_n be the $n \times n$ Hessenberg matrix defined as in* (3.1). *For special cases as below we get* $\det L_n = L_n(x)$

$$L_n = \begin{bmatrix} 2 & 0 & 0 & ... & ... & 0 \\ 0 & \frac{x}{2} & i & 0 & \ddots & 0 \\ 0 & i & x & i & \ddots & 0 \\ 0 & 0 & i & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & i \\ 0 & 0 & ... & 0 & i & x \end{bmatrix},$$

where $L_n(x)$ is the n-th Lucas polynomial.

Corollary 3. *Let H_n be the $n \times n$ Hessenberg matrix defined as in* (3.1). *For $m_{1,1} = 1$, $m_{i,i} = 2x, i \neq 1$, $m_{i,i+1} = -\sqrt{2}i$, and $m_{i+1,i} = (n-2)\sqrt{2}i$. Then the determinants of the matrices for $n \geq 1$ gives the Hermite polynomials, that is* $\det H_n = H_{n-1}(x)$

$$H_n = \begin{bmatrix} 1 & 0 & 0 & ... & ... & 0 \\ 0 & 2x & -\sqrt{2}i & 0 & \ddots & 0 \\ 0 & \sqrt{2}i & 2x & -\sqrt{2}i & \ddots & 0 \\ 0 & 0 & 2\sqrt{2}i & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & -\sqrt{2}i \\ 0 & 0 & ... & 0 & (n-2)\sqrt{2}i & 2x \end{bmatrix},$$

where $H_n(x)$ is the n-th Hermite polynomial.

Corollary 4. *Let L_n be the $n \times n$ Hessenberg matrix defined as in* (3.1). *For special cases as below we get* $\det L_n = L_{n-1}(x)$

$$L_n = \begin{bmatrix} 1 & 0 & 0 & ... & ... & 0 \\ 0 & 1-x & -i & 0 & \ddots & 0 \\ 0 & \frac{i}{2} & \frac{3-x}{2} & -2i & \ddots & 0 \\ 0 & 0 & \frac{i}{3} & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & -(n-2)i \\ 0 & 0 & ... & 0 & \frac{i}{(n-1)} & \frac{2n-3-x}{(n-1)} \end{bmatrix},$$

where $L_n(x)$ is the n-th Laguerre polynomial.

Corollary 5. *Let P_n be the $n \times n$ Hessenberg matrix defined as in* (3.1). *For special cases as below we get* $\det P_n = P_{n-1}(x)$

$$P_n = \begin{bmatrix} 1 & 0 & 0 & ... & ... & 0 \\ 0 & x & -\frac{i}{2} & 0 & \ddots & 0 \\ 0 & i & \frac{3x}{2} & -\frac{i}{3} & \ddots & 0 \\ 0 & 0 & 2i & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & -\frac{i}{(n-1)} \\ 0 & 0 & ... & 0 & (n-2)i & \frac{(2n-3)x}{(n-1)} \end{bmatrix},$$

where $P_n(x)$ is the n-th Legendre polynomial.

4. Algorithms for Fibonacci and Lucas Polynomials

In this section we give algorithms for Fibonacci and Lucas polynomials. We give the value N and the program gives us the first N Fibonacci or Lucas polynomials as required.

First of all we give the *JAVA* codes of the Fibonacci polynomials:

```
import javax.swing.JOptionPane;
  public class Fibonacci {
  public static void main(String[] args) {
    String s1 = JOptionPane.showInputDialog("Please enter N");
    int N = Integer.parseInt(s1);
Polynomial[] F = new Polynomial[Math.max(2, N)]; // F[i] = ith Fi-
bonacci polynomial
      F[0] = new Polynomial(1, 0); // 1
      F[1] = new Polynomial(1, 1); // x
      Polynomial twox = new Polynomial(1, 1); // 2x
// compute Fibonacci polynomials
  for (int n = 2; n < N; n++) {
    Polynomial temp1 = twox.times(F[n-1]);
    F[n] = temp1.plus(F[n-2]);
                }
// print results
for (int n = 0; n < N; n++)
System.out.println(F[n]);
                    }
              }
```

Second we give the *JAVA* codes of the Lucas polynomials.

```
import javax.swing.JOptionPane;
  public class Lucas {
  public static void main(String[] args) {
    String s1 = JOptionPane.showInputDialog("Please enter N");
    int N = Integer.parseInt(s1);
Polynomial[] L = new Polynomial[Math.max(2, N)]; // L[i] = ith Lucas
polynomial
      L[0] = new Polynomial(2, 0); // 1
      L[1] = new Polynomial(1, 1); // x
      Polynomial twox = new Polynomial(1, 1); // 2x
// compute Lucas polynomials
  for (int n = 2; n < N; n++) {
    Polynomial temp1 = twox.times(L[n-1]);
    L[n] = temp1.plus(L[n-2]);
                }
```

```
// print results
for (int n = 0; n < N; n++)
System.out.println(L[n]);
                    }
          }
```

These two programmes use the same Polynomial.java class file which we give now.

```
public class Polynomial {
   private int[] coef; // coefficients
   private int deg; // degree of polynomial (0 for the zero polynomial)
   public Polynomial(int a, int b) {
      coef = new int[b+1];
      coef[b] = a;
      deg = degree();
                   }
    public int degree() {
      int d = 0;
      for (int i = 0; i < coef.length; i++)
      if (coef[i] != 0) d = i;
   return d;
            }

   public Polynomial plus(Polynomial b) {
      Polynomial a = this;
      Polynomial c = new Polynomial(0, Math.max(a.deg, b.deg));
      for (int i = 0; i < = a.deg; i++) c.coef[i] += a.coef[i];
   for (int i = 0; i < = b.deg; i++) c.coef[i] += b.coef[i];
      c.deg = c.degree();
   return c;
                       }
   public Polynomial minus(Polynomial b) {
      Polynomial a = this;
      Polynomial c = new Polynomial(0, Math.max(a.deg, b.deg));
      for (int i = 0; i < = a.deg; i++) c.coef[i] += a.coef[i];
   for (int i = 0; i < = b.deg; i++) c.coef[i] -= b.coef[i];
      c.deg = c.degree();
   return c;
                       }
   public Polynomial times(Polynomial b) {
      Polynomial a = this;
      Polynomial c = new Polynomial(0, a.deg + b.deg);
      for (int i = 0; i < = a.deg; i++)
   for (int j = 0; j < = b.deg; j++)
```

```
   c.coef[i+j] += (a.coef[i] * b.coef[j]);
   c.deg = c.degree();
return c;
                    }
public Polynomial compose(Polynomial b) {
   Polynomial a = this;
   Polynomial c = new Polynomial(0, 0);
   for (int i = a.deg; i >= 0; i–) {
Polynomial term = new Polynomial(a.coef[i], 0);
c = term.plus(b.times(c));
                  }
return c;
                        }
public int evaluate(int x) {
   int p = 0;
   for (int i = deg; i >= 0; i–)
   p = coef[i] + (x * p);
return p;
                }
public String toString() {
   if (deg == 0) return "" + coef[0];
   if (deg == 1) return coef[1] + "x + " + coef[0];
String s = coef[deg] + "xˆ" + deg;
   for (int i = deg-1; i >= 0; i–) {
   if (coef[i] == 0) continue;
   else if (coef[i] > 0) s = s + " + " + ( coef[i]);
   else if (coef[i] < 0) s = s + " - " + (-coef[i]);
   if (i == 1) s = s + "x";
   else if (i > 1) s = s + "xˆ" + i;
                  }
return s;
             }
public static void main(String[] args) {
   Polynomial zero = new Polynomial(0, 0);
   Polynomial p1 = new Polynomial(4, 3);
   Polynomial p2 = new Polynomial(3, 2);
   Polynomial p3 = new Polynomial(1, 0);
   Polynomial p4 = new Polynomial(2, 1);
   Polynomial p = p1.plus(p2).plus(p3).plus(p4);
   Polynomial q1 = new Polynomial(3, 2);
   Polynomial q2 = new Polynomial(5, 0);
   Polynomial q = q1.plus(q2);
   Polynomial r = p.plus(q);
   Polynomial s = p.times(q);
```

```
        Polynomial t = p.compose(q);
        System.out.println("zero(x) = " + zero);
        System.out.println("p(x) = " + p);
        System.out.println("q(x) = " + q);
        System.out.println("p(x) + q(x) = " + r);
        System.out.println("p(x) * q(x) = " + s);
        System.out.println("p(q(x)) = " + t);
    System.out.println("0 - p(x) = " + zero.minus(p));
System.out.println("p(3) = " + p.evaluate(3));
                    }
            }
```

References

[1] G.E. Andrews, R. Askey, R. Roy, Special Functions, *Encyclopedia of Mathematics and its Applications*, Volume **71**, Cambridge University Press, Cambridge (1999).

[2] N.D. Cahill, J.R. D'Ericco, J. Spence, Complex factorizations of the Fibonacci and Lucas numbers, *Inform. Process. Lett.*, **41**, No. 1 (2003), 13-19

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

[4] G. Freud, *Orthogonal Polynomials*, Pergamon Press, Oxford (1966).

[5] D.E. Knuth, *The Art of Computer Programming: Fundamental Algorithms*, Addison-Wesley, Reading, MA (1973).

[6] T. Koshy, *Fibonacci and Lucas Numbers with Applications*, New York, Chichester, Weinheim, Toronto, Singapore (2001).

[7] J. Margado, Note on the Chebyshev polynomials and applications to the Fibonacci numbers, *Port. Math.*, **52** (1995), 363-378.

[8] M. Püschel, J.M.F. Moura, The algebraic approach to the discrete cosine and sine transforms and their fast algorithms, *SIAM J. Comput.*, **32**, No. 5 (2003), 1280-1316.

[9] T. Rivlin, *Chebyshev Polynomials: From Approximation Theory to Algebra and Number Theory*, Second Ed., Wiley, NewYork (1990).

[10] G. Szegö, *Orthogonal Polynomials*, Amer. Math. Soc. Colloq. Pub., **23** (1939).

[11] D. Takahashi, A fast algorithm for computing large Fibonacci numbers, *Inform. Process. Lett.*, **75** (2000), 243-246.

[12] D. Tasci, E. Kilic, On the order-k generalized Lucas numbers, *Appl. Math. Comput.*, **155**, No. 3 (2004), 637-641.

[13] F.J. Urbanek, An O(logn) algorithm for computing the nth element of the solution of a difference equation, *Inform. Process. Lett.*, **11**, No. 2 (1980), 66-67.

[14] S. Vajda, *Fibonacci and Lucas Numbers and the Golden Section*, Chichester, Brisbane,Toronto, New York (1989).

[15] W.A. Webb, E.A. Parberry, Divisibility properties of Fibonacci polynomials, *Fibonacci Quart.*, **7**, No. 5 (1969), 457-463.

[16] B. Wendroff, On orthogonal polynomials, *Proc. Amer. Math. Soc.*, **12** (1961), 554-555.