Journal of Engineering Mathematics, Vol. 7, No. 1, January 1973
Noordhoff International Publishing - Leyden
Printed in The Netherlands

On the computation of Mathieu functions

by

THE GROUP "NUMERICAL ANALYSIS" at Delft University of Technology

Dept. of Mathematics, Delft University of Technology, Delft, The Netherlands

(Received February 10, 1972)

SUMMARY

The aim of this paper is to propose a fast numerical method for the computation of Mathieu functions. These functions can be used to solve the (reduced) wave equation on an elliptic domain in two space dimensions. In the first part of this paper the theoretical background of the method is discussed, while in the second part the algol procedures are presented.

Introduction

The Mathieu functions are the solutions of the ordinary Mathieu equation and the modified Mathieu equation, which equations can be obtained after a separation of the two space variables in the (reduced) wave equation, defined on an elliptic domain. It is preferable to express the solution of the wave equation in terms of Mathieu functions, because a method based on finite differences needs a very fine discretisation of the domain in order to be able to take large frequencies (small wavelengths) into account. There are also other difficulties which arise in dealing with external problems for elliptic domains. Problems of practical interest are, for example, diffraction theory and vibrating air plane wings.

1. Theoretical background

The two differential equations of Mathieu are See [1]:

$$\frac{d^2y}{dz^2} + (a - 2q\cos 2z)y = 0 \quad \text{(ordinary equation)} \tag{1}$$

$$\frac{d^2y}{dz^2} - (a - 2q\cosh 2z)y = 0 \quad \text{(modified equation)} \tag{2}$$

in which q represents a parameter which in most applications is real and positive; a is a separation constant.

We are often only interested in solutions of (1) with period π or 2π. It can be shown, [2], that there is a countable set of characteristic values (a), for which the solutions are π or 2π periodic. We distinguish between four kinds of characteristic values (eigenvalues):

$a = a_{2r}(q)$, solution even, period π
$a = a_{2r+1}(q)$, solution even, period 2π
$a = b_{2r+2}(q)$, solution odd, period π
$a = b_{2r+1}(q)$, solution odd, period 2π
$r = 0, 1, 2, \ldots$

If $q \geqq 0$ all characteristic values are strictly separated ([2]) and:

$$a_0 < b_1 < a_1 < b_2 < \ldots < a_{2r} < b_{2r+1} < a_{2r+1} < b_{2r+2} < \ldots$$

For every characteristic value, the solution of (1) can be given as a Fourier series See [1]:

$$ce_{2r}(z, q) = \sum_{m=0}^{\infty} A_{2m}^{(2r)} \cos 2mz\,, \qquad a = a_{2r} \tag{3a}$$

$$ce_{2r+1}(z, q) = \sum_{m=0}^{\infty} A_{2m+1}^{(2r+1)} \cos(2m+1)z\,, \qquad a = a_{2r+1} \tag{3b}$$

$$se_{2r+2}(z, q) = \sum_{m=0}^{\infty} B_{2m+2}^{(2r+2)} \sin(2m+2)z\,, \qquad a = b_{2r+2} \tag{3c}$$

$$se_{2r+1}(z, q) = \sum_{m=0}^{\infty} B_{2m+1}^{(2r+1)} \sin(2m+1)z\,, \qquad a = b_{2r+1} \tag{3d}$$

Analogous formulae are known for the modified Mathieu functions. For numerical reasons we use the following series:

$$Mc_{2r}^{(j)}(z, q) = \sum_{k=0}^{\infty} (-1)^{k+r} A_{2k}^{(2r)} \{J_{k-s}(u_1) Z_{k+s}^{(j)}(u_2) + J_{k+s}(u_1) Z_{k-s}^{(j)}(u_2)\} / \varepsilon_s A_{2s}^{2r} \tag{4a}$$

with: $\varepsilon_0 = 2,\ \varepsilon_s = 1,\ s = 1, 2, 3, \ldots,\qquad a = a_{2r}$

$$Mc_{2r+1}^{(j)}(z, q) = \sum_{k=0}^{\infty} (-1)^{k+r} A_{2k+1}^{2r+1} \{J_{k-s}(u_1) Z_{k+s+1}^{(j)}(u_2) + J_{k+s+1}(u_1) Z_{k-s}^{(j)}(u_2)\} / A_{2s+1}^{(2r+1)}\,, \tag{4b}$$

$a = a_{2r+1}$

$$Ms_{2r}^{(j)}(z, q) = \sum_{k=1}^{\infty} (-1)^{k+r} B_{2k}^{(2r)} \{J_{k-s}(u_1) Z_{k+s}^{(j)}(u_2) - J_{k-s}(u_1) Z_{k+s+1}^{(j)}(u_2)\} / B_{2s}^{(2r)}\,, \tag{4c}$$

$a = b_{2r}$

$$Ms_{2r+1}^{(j)}(z, q) = \sum_{k=0}^{\infty} (-1)^{k+r} B_{2k+1}^{(2r+1)} \{J_{k-s}(u_1) Z_{k+s+1}^{(j)}(u_2) - J_{k+s+1}(u_1) Z_{k-s}^{(j)}(u_2)\} / B_{2s+1}^{(2r+1)}\,, \tag{4d}$$

$a = b_{2r+1},\ u_1 = \sqrt{q}\,e^{-z},\ u_2 = \sqrt{q}\,e^{z},$ See [2],

$Z_k^{(1)}(u) = J_k(u)\,,\quad Z_k^{(2)}(u) = Y_k(u)\,,$ (Bessel functions of the first and second kind).

These series are chosen because they converge uniformly in the whole z-plane and because they converge faster then any other known series expansion for modified Mathieu functions. An enormous numerical advantage is that for $z \geqq 0$ there will be no loss of significant figures, an advantage that other known series lack. The parameter s is arbitrary, but if one wishes to avoid loss of significant figures, the best value will be r.

Because in practical applications Mathieu functions will appear most of the time in series of Mathieu functions, we have developed procedures which simplify the handling of these series. From (4a, 4b, 4c, 4d) it will be clear that we need procedures for calculating Bessel functions and Fourier coefficients. For computing Fourier coefficients one has to know the characteristic values. In the next part a description is given of the routines for calculating Bessel functions, Fourier coefficients and characteristic values.

Computation of characteristic values

Substitution of the series (3a), (3b), (3c) and (3d) in differential equation (1), yields four sets of equations for computing the Fourier coefficients:

$$\begin{bmatrix} a_{2r} & -q & & & 0 \\ -2q & a_{2r}-4 & -q & & \\ & -q & a_{2r}-16 & -q & \ddots \\ & & \ddots & \ddots & -q \\ 0 & & & -q & a_{2r}-(2m)^2 \end{bmatrix} \begin{bmatrix} A_0^{(2r)} \\ A_2^{(2r)} \\ A_4^{(2r)} \\ \vdots \\ A_{2m}^{(2r)} \end{bmatrix} = \mathbf{0} \tag{5a}$$

$$\begin{bmatrix} a_{2r+1}-1-q & -q & & & 0 \\ -q & a_{2r+1}-9 & -q & & \\ & -q & a_{2r+1}-25 & -q & \ddots \\ & & \ddots & \ddots & -q \\ 0 & & & -q & a_{2r+1}-(2m+1)^2 \end{bmatrix} \begin{bmatrix} A_1^{(2r+1)} \\ A_3^{(2r+1)} \\ A_5^{(2r+1)} \\ \vdots \\ A_{2m+1}^{(2r+1)} \end{bmatrix} = \mathbf{0} \tag{5b}$$

$$\begin{bmatrix} b_{2r+2}-4 & -q & & & 0 \\ -q & b_{2r+2}-16 & -q & & \\ & -q & b_{2r+2}-36 & -q & \ddots \\ & & \ddots & \ddots & -q \\ 0 & & & -q & b_{2r+2}-(2m+2)^2 \end{bmatrix} \begin{bmatrix} B_2^{(2r+2)} \\ B_4^{(2r+2)} \\ B_6^{(2r+2)} \\ \vdots \\ B_{2m+2}^{(2r+2)} \end{bmatrix} = \mathbf{0} \tag{5c}$$

$$\begin{bmatrix} b_{2r+1}-1+q & -q & & & 0 \\ -q & b_{2r+1}-9 & -q & & \\ & -q & b_{2r+1}-25 & -q & \ddots \\ & & \ddots & \ddots & -q \\ 0 & & & -q & b_{2r+1}-(2m+1)^2 \end{bmatrix} \begin{bmatrix} B_1^{(2r+1)} \\ B_3^{(2r+1)} \\ B_5^{(2r+1)} \\ \vdots \\ B_{2m+1}^{(2r+1)} \end{bmatrix} = \mathbf{0} \tag{5d}$$

G. Blanche proposed in [3] a method for calculating the characteristic values, which, however, in some cases will not produce the correct result. We shall present a new method which always converges to the right solution and which, furthermore, is very fast. This method is based on work by Green and Michaelson [4]. The original method by Green and Michaelson needed very much computer time and in order to eliminate that, an essential modification was made. We shall show the method for the computation of a_{2r+1} (5b). All other characteristic values can be calculated in exactly the same way. It will be clear from (5b) that a_{2r+1} can be seen as the eigenvalue of the infinite tridiagonal matrix:

$$\begin{bmatrix} 1+q & q & & & 0 \\ q & 9 & q & & \\ & q & 25 & q & \ddots \\ & & \ddots & \ddots & q \\ 0 & & & q & (2m+1)^2 \end{bmatrix} \tag{6}$$

In calculating the eigenvalues of this infinite matrix, it is necessary to cut off the matrix after a finite number of columns and rows. The question arises: what will be the influence of the cutting on the eigenvalues?

A criterion for calculating the eigenvalue with a given accuracy is given by the following theorem proved by Wilkinson ([7]).

Theorem: *Let λ be an approximation of the eigenvalue λ_i of the $(n \times n)$ symmetric matrix A. $\boldsymbol{x}$ is the approximating eigenvector belonging to λ, with $\|\boldsymbol{x}\|_2 = 1$. Let $A\boldsymbol{x} - \lambda\boldsymbol{x} = \boldsymbol{\eta}$. So $\boldsymbol{\eta}$ is the residuvector; $\|\boldsymbol{\eta}\|_2 = \varepsilon$. Then: $|\lambda_i - \lambda| < \varepsilon$ for at least one of the i, $1 \leqq i \leqq n$.*

We apply this theorem to our particular case. Let:

$$M_n = \begin{bmatrix} 1+q & q & & & 0 \\ q & 9 & q & & \\ & \ddots & \ddots & \ddots & \\ & & q & (2n-3)^2 & q \\ 0 & & & q & (2n-1)^2 \end{bmatrix}$$

So we cut off the infinite matrix after the nth row and nth column. Let the eigenvalues of M_n be:

$$\lambda_1^{(n)} \leqq \lambda_2^{(n)} \leqq \lambda_3^{(n)} \leqq \ldots \leqq \lambda_n^{(n)}$$

$\boldsymbol{x}$ is the eigenvector corresponding to $\lambda_i^{(n)}$.

Suppose: $\lambda_i^{(n)}$ is an approximation of $\lambda_i^{(n+1)}$. We approximate the eigenvector corresponding to $\lambda_i^{(n+1)}$ by the vector $[\boldsymbol{x}, 0]^T$. Now:

$$M_{n+1}\begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix} = \begin{bmatrix} M_n & q\boldsymbol{e}_n \\ q\boldsymbol{e}_n^T & (2n+1)^2 \end{bmatrix}\begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix} = \begin{bmatrix} M_n\,\boldsymbol{x} \\ q\boldsymbol{e}_n^T\,\boldsymbol{x} \end{bmatrix} = \lambda_i^{(n)}\begin{bmatrix} \boldsymbol{x} \\ 0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{0} \\ q\alpha_n \end{bmatrix}$$

with:

$$\boldsymbol{e}_n = [0, 0, 0, \ldots, 0, 1]^T, \quad \boldsymbol{x} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$$

The vector $[\boldsymbol{0}, q\alpha_n]^T$ corresponds to the residuvector $\boldsymbol{\eta}$ in Wilkinson's theorem. So for at least one number i, $1 \leqq i \leqq n$, it is true that:

$$|\lambda_i^{(n)} - \lambda_i^{(n+1)}| < |q\alpha_n|$$

So an upper bound for $|\alpha_n|$ gives an indication for the convergence of the eigenvalues when n increases.

Green and Michaelson proposed a method of calculating the upperbound with the aid of Gerschgorin circles. They had to start with a value of n, for which the Gerschgorin circles are disjoint. It is possible to make a finer enclosure. This is a consequence of the following theorem: ([5]).

Theorem: *Let $C = A + B$ with A, B, C symmetric matrices. Let the eigenvalues of A, B, C be:*

$$A: a_1 \geqq a_2 \geqq a_3 \geqq \ldots \geqq a_n \quad B: b_1 \geqq b_2 \geqq b_3 \geqq \ldots \geqq b_n \quad C: c_1 \geqq c_2 \geqq c_3 \geqq \ldots \geqq c_n$$

Then: $c_s \leqq a_s + b_1$, $c_s \geqq a_s + b_n$. In our case:

$$\begin{bmatrix} 1+q & q & & & 0 \\ q & 9 & q & & \\ & \ddots & \ddots & \ddots & \\ & & q & (2n-3)^2 & q \\ 0 & & & q & (2n-1)^2 \end{bmatrix} = \begin{bmatrix} 1+q & & & & 0 \\ & 9 & & & \\ & & \ddots & & \\ & & & (2n-3)^2 & \\ 0 & & & & (2n-1)^2 \end{bmatrix} + \begin{bmatrix} 0 & q & & & 0 \\ q & 0 & q & & \\ & \ddots & \ddots & \ddots & \\ & & q & 0 & q \\ 0 & & & q & 0 \end{bmatrix}$$

$$C \qquad = \qquad A \qquad + \qquad B$$

The eigenvalues of A are the diagonal elements of C. According to Gerschgorin's theorem we know for the eigenvalues of B: $|b_i| \leqq 2q$. *So*:

$$a_s - 2q \leqq c_s \leqq a_s + 2q \tag{7}$$

The theorem as used here is only applicable to the sets of equations (5a), (5c) and (5d). The matrix corresponding to (5b) has the following configuration:

$$\begin{bmatrix} 0 & q & & & 0 \\ 2q & 4 & q & & \\ & q & 16 & q & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & q \\ 0 & & & q & (2n-2)^2 \end{bmatrix} \tag{8}$$

Using the familiar expression for the Sturm sequence it is evident that the eigenvalues of (8) are exactly the same as the eigenvalues of the matrix:

$$\begin{bmatrix} 0 & q\sqrt{2} & & & 0 \\ q\sqrt{2} & 4 & q & & \\ & q & 16 & q & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & q \\ & & & q & (2n-2)^2 \end{bmatrix} \tag{9}$$

Splitting matrix (9) in a diagonal matrix and a residu matrix B we get for B:

$$B = \begin{bmatrix} 0 & q\sqrt{2} & & & 0 \\ q\sqrt{2} & 0 & q & & \\ & q & 0 & q & \\ & & \ddots & \ddots & \ddots \\ & & & \ddots & q \\ 0 & & & q & 0 \end{bmatrix} \tag{10}$$

Using the same argument, however, we know that the eigenvalues of (10) are exactly the same as the eigenvalues of the matrix:

$$\begin{bmatrix} 0 & 2q & 0 & \\ q & 0 & q & \\ & \ddots & \ddots & \ddots \\ & & \ddots & q \\ 0 & & q & 0 \end{bmatrix} \tag{11}$$

So, again using Gerschgorin we know $|b_i| \leqq 2q$ and the result (7) holds for all four matrices.

We choose r such that $a_r - a_i > 4q$. The eigenvector $\boldsymbol{x}$ corresponding to λ_i will be normalized in such a way that:

$$\|\boldsymbol{x}\|_2 = 1 \quad \text{or} \quad \sum_{i=1}^{n} \alpha_i^2 = 1$$

So: $|\alpha_i| \leqq 1$, $i=1(1)n$. The set of equations for calculating the eigenvalue λ_i of M_n is, starting from the $(r-1)^{st}$ row:

$$q\alpha_{r-2}+(a_{r-1}-\lambda_i)\alpha_{r-1}+q\alpha_r = 0 \tag{12a}$$
$$q\alpha_{r-1}+(a_r-\lambda_i)\alpha_r+q\alpha_{r+1} = 0\,. \tag{12b}$$
$$q\alpha_{n-2}+(a_{n-1}-\lambda_i)\alpha_{n-1}+q\alpha_n = 0 \tag{12c}$$
$$q\alpha_{n-1}+(a_n-\lambda_i)\alpha_n = 0\,. \tag{12d}$$

According to (12d):

$$\alpha_n = -\frac{q}{a_n-\lambda_i}\,\alpha_{n-1}$$

So:

$$|\alpha_n| = \frac{q}{a_n-\lambda_i}\,|\alpha_{n-1}| \tag{13}$$

According to (7): $\lambda_i < a_i+2q$. Substitution into (13) yields:

$$|\alpha_n| < \frac{q}{a_n-a_i-2q}\,|\alpha_{n-1}| = \beta_n|\alpha_{n-1}|$$

Because $n>r$: $a_n > a_r$ and $a_r-a_i>4q$. So: $a_n-a_i>4q$. So: $0<\beta_n<\frac{1}{2}$ and sign $(\alpha_n) \neq$ sign (α_{n-1}).

Clearly $\alpha_n \neq 0$, because otherwise $\boldsymbol{x}$ would be zero. Suppose $\alpha_n<0$, so $\alpha_{n-1}>0$, $q\alpha_n > -\beta_n q\alpha_{n-1}$. Substitution in (12c) yields: $q\alpha_{n-2}+(a_{n-1}-\lambda_i-\beta_n q)\alpha_{n-1}<0$ or:

$$(\alpha_{n-1}-\lambda_i-\beta_n q)\,\alpha_{n-1} < -q\alpha_{n-2} \tag{14}$$

Substitution of (7) in (12) yields: $\{a_{n-1}-a_i-(2+\beta_n)q\}\,\alpha_{n-1} < -q\alpha_{n-2}$. Because $n-1>r$ the following holds: $a_{n-1}-a_i>4q$. So:

$$a_{n-1}-a_i-(r+\beta_n)q > q \qquad (\beta_n<\tfrac{1}{2}<1)$$

Now we have:

$$|\alpha_{n-1}| < \frac{q}{a_{n-1}-a_i-(2+\beta_n)q}\,|\alpha_{n-2}| = \beta_{n-1}|\alpha_{n-2}| \text{ with } \beta_n<\beta_{n-1}<1$$

If $\alpha_n>0$, then following an analogous reasoning exactly the same result will be obtained. Continuation of this process yields the following recursion formulae for β_k:

$$\beta_{n-j} = \frac{q}{a_{n-j}-a_i-(2+\beta_{n-j+1})q}, \qquad n-j \geqq r$$
$$\beta_{n+1} = 0\,.$$

From which the following inequalities result:

$$q|\alpha_n| < q\beta_n|\alpha_{n-1}| < q\beta_n\beta_{n-1}|\alpha_{n-2}| < \ldots < q\beta_n\beta_{n-1}\ldots\beta_r|\alpha_{r-1}|$$
$$< q\beta_n\beta_{n-1}\ldots\beta_r$$

Starting from $\beta_{n+1}=0$ we can calculate $\beta_n, \beta_{n-1}, \beta_{n-2}, \ldots, \beta_r$. Now choose n such that:

$$q|\alpha_n| < q\prod_{i=r}^{n}\beta_i < \tfrac{1}{2}\varepsilon$$

Here ε is the required accuracy of the eigenvalue λ_i. Then:

$$|\lambda_i^{(n)}-\lambda_i^{(\infty)}| < |\lambda_i^{(n)}-\lambda_i^{(n+1)}|+|\lambda_i^{(n+1)}-\lambda_i^{(n+2)}|+\ldots$$
$$< \tfrac{1}{2}\varepsilon+\tfrac{1}{4}\varepsilon+\tfrac{1}{8}\varepsilon+\ldots = \varepsilon$$

In the same way we get the same recursion formula if we cut off the matrix in the first rows. This will be applied if we need eigenvalues of high order.

Remark: For every single eigenvalue we must cut off the matrix.

In calculating eigenvalues of low order the method just described will provide an enormous gain in computer time compared with the method by Green and Michaelson. For example: computing λ_1 with $q=1000$ we need a 57×57 matrix and Green and Michaelson a 515×515 matrix.

If we modify the method by Green and Michaelson in the same way as we did here, we can use it for various Sturm-Liouville problems.

The eigenvalues of the matrix M_n are calculated with the aid of the Sturms sequence. By choosing appropriate starting values with the aid of formulae 20.2.25, 20.2.26 and 20.2.30 from [2], we can get a very fast convergence to the desired eigenvalue using the Regula Falsi method.

Computation of Fourier coefficients and Bessel functions

When computing the Fourier coefficients we choose the method described by Blanche [3]. When computing the Bessel functions we used the well-known recurrence relations for integer order. For the start of the process we needed values of $y_0(x)$, $y_1(x)$, $j_0(x)$ and $j_1(x)$. For that purpose Chebychev polynomials were used which are described in [6].

Numerical problems—loss of significant figures—occur in the calculation of $j_k(x)$ when $k>x$. To handle this problem we used a method described by G. Blanche [3]. The procedures for calculating Bessel functions are very fast, because they distinguish between the several possible cases. In most other known procedures this feature is usually missing.

Test results:

We tested the eigenvalues from $q=0$ to $q=10000$. For all different orders the results were good, although the computing time will increase when q increases. The modified Mathieu functions gave the following result:

The number of digits which are correct is given by the formula:

$$\alpha-(1+\text{entier}(\max(1,\ ^{10}\log(\sqrt{q}e^{z}))))$$

Here α is the number of digits in which the computation is performed. All tests were run on an I.C.L. 1905 computer.

2. Algol procedures

We splitted the procedure for calculating an eigenvalue for Mathieu's equation into three procedures. These are:

1 mataf: Cuts the infinite matrix, and computes the lower- and upperbound for the rownumber.

2 startvalue: Computes a startvalue for the eigenvalue, necessary for the Regula Falsi.

3 eigenvalue: Computes the eigenvalue, with the aid of the numerical results of 1 and 2.

The other procedures are:

4 Fouriercoeff.: Computes the required Fourier coefficients.

5 Bessel: Computes the required Bessel functions.

6 Mathieu: Computes all the various types of Mathieu functions, with the aid of the numerical results of the procedures 1, 2, 3, 4 and 5.

Remark:

We used in our procedures the routines even and overflow. These are standard I.C.L. routines with the following definitions:

```
integer   procedure even (i); value i: integer i;
          even:=(-1)↑i;

boolean   procedure overflow;
comment   overflow tests if overflow has occurred in the program.
          If overflow has occurred then overflow:=true
          else overflow:=false;
```

Both routines are available in most computers.

```
1. procedure mataf 1 (ev, q, order, eps, m, n, lab);
   value     ev, q, order, eps;
   integer   order, m, n;
   real      q, eps;
   boolean   ev;
   label     lab;
comment The numbers m and n are computed. m is the lower bound and n the upperbound
        of the row numbers of the finite matrix, which we need for calculating the eigenvalue
        λ_order with accuracy eps. To minimize the computertime, m and n can be taken as
        equal to m and n as computed for lower order. Otherwise m and n must be taken as
        zero at input. The integers m and n will be equal for a_2r, a_2r+1, b_2r, b_2r+1, so
        if all four must be calculated, mataf has to be called only once. The procedure checks
        the values of the parameters q and order. If one of them is wrong, the procedure
        jumps to the external label lab. Q is the parameter in Mathieu's equation. The
        Boolean ev indicated which kind of solution is required. If ev=true then an even
        solution will be calculated otherwise an odd one. The parameter order gives the
        order of the eigenvalue, so we get λ_order;
begin   real ar, h, betha, prod; integer i, r;
        if q≦0 ∨ order < 0 ∨ ¬ ev ∧ order=0 then goto lab;
comment test of input;
        if m≠0 then begin if even (order)=-1 then m:=m+m+1 else
        begin if ev ∧ m≠1 then m:=m+m-2 else m:=m+m end end;
        if n≠0 then begin if even (order)=-1 then n:=n+n+1 else
        begin if ev then n:=n+n-2 else n:=n+n end end;
comment Now the diagonal element ar corresponding to λ_order will be computed;
        ar:=order * order;
        if even (order)=-1 then begin if ev then begin
        if ar < 1+q then begin ar:=(order+2)↑2;
        if ar > 1+q then ar:=1+q end end else begin
        if order=1 then ar:=1-q end end;
            eps:=2 * eps; h:=4 * q;
        for i:=order, i+2 while i * i -ar < h do r:=i; r:=r+2;
        if n > r then n:=r else n:=n-2;
comment r is the number according to a_r-a_order > 4q;
loop 1: betha:=0; n:=n+2; prod:=1;
        for i:=n step-2 until r do begin
        betha:=q/(i * i-ar-(2+betha) * q);
        prod:=prod * betha end recursion;
comment Recursion formula for β_k;
        if q * prod > eps then goto loop 1;
```

```
        if order < 12 ∨ ar < h then begin m:=2; goto exit end;
        for i:=order, i-2 while ar-i*i < h ∧ i > 5 do r:=i; r:=r-2;
        if m=0 then m:=r-2 else m:=m+6; if m > r then m:=r;
loop 2: betha:=0; m:=m-2; prod:=1;
        if m < 4 then begin m:=2; goto exit end;
        for i:=m step 2 until r do begin
        betha:=q/(ar-i*i-(2+betha)*q);
        prod:=prod*betha; end recursion;
        if q*prod > eps then goto loop 2;
exit:   m:=(m+0.5)/2; n:=(n+0.5)/2;
        if ev ∧ even(order)=1 then
        begin n:=n+1; if m≠1 then m:=m+1 end;
        end of procedure mataf 1;
```

Remark:

The procedure as described here is the algorithmic translation of the preceding theory. There are, however, two disadvantages:

1. In proportion to the calculation of the eigenvalue, mataf takes too much computer-time.
2. The numbers m and n are certainly not the best possible.

In order to overcome these disadvantages we present the following ad hoc routine. The reader be warned, however, that the number of significant figures in the calculated eigenvalues is kept fixed at 8 and that the routine has been tested only in the region $0 < q < 1000$, $0 \leqq \text{order} \leqq 2\sqrt{q}+20$.

The routine runs as follows:

```
procedure mataf 2 (ev, h, order, m, n, lab);
value ev, h, order; integer order, m, n; real h;
Boolean ev; label lab;
comment The parameters are exactly the same as in procedure mataf 1.
        Only h=√q for convenience;
        begin integer j, n1, n2, r;
        if h≦0 ∨ order <0 ∨ ¬ ev ∧ r=0 then goto lab;
        if ev ∧ even (order)=1 then r:=order+2 else r:=order;
        r:=(r+0.5)/2; j:=entier (5.1+1.07*h);
        n1:=entier (9+0.5*h-5/(if h≧1 then h else 1));
        n2:=entier (8.5+1.5*h);
        n:= if r≧j then r+n2-j else if r+n1 < n2 then r+n1 else n2;
        m:= if order-n > 1 then order -n else 1;
        end of procedure mataf 2;
```

2.

```
procedure startvalue (ev, q, order, lambda);
value     ev, q, order;
integer   order;
real      q, lambda; Boolean ev;
comment An approximation for the eigenvalue of Mathieu's equation is computed. The
        parameters ev, q and order are the same as in the procedure mataf. Lambda is the
        calculated approximation;
        begin switch lab:=r0, r1, r2, r3, r4; real h; integer help;
        Boolean option;
        real procedure big;
```

```
comment formula 20.2.30 in [2] will be calculated;
        begin real w, w2, h2; array d[1:4];
        w2 := 2 * order + 1; w2 := w * w;
        d[1] := 9 + w2 * (34 + 5 * w2);
        d[2] := w * (405 + w2 * (410 + 33 * w2));
        d[3] := 486 + w2 * (2943 + w2 * (1260 + 63 * w2));
        d[4] := w * (41607 + w2 * (69001 + w2 * (15617 + 527 * w2)));
        h2 := 1/(h * 32);
        big := -2 * q + 2 * w * h - 0.125 * (w2 + 1) - 0.25 * h2 *
          (w * w2 + 3 * w + h2 * (d[1] + h2 * (d[2] + 0.125/h *
          (d[3] + h2 * d[4]))));
        end of big;
        real procedure small;
comment formulas 20.2.25 or 20.2.26 in [2] will be calculated;
        begin switch eigv := a0, b0, a1, b1, a2, b2, a3, b3, a4, b4,
        a5, b5, a6, b6;
        if order > 6 then begin real r2, r4, r1, r12;
        r2 := order * order; r4 := r2 * r2; r1 := r2 - 1; r12 := r1 * r1; q := q * q;
        small := r2 + q/(2 * r1) * (1 + q/(16 * r12 * (r2 - 4)) *
        (5 * r2 + 7 + q * (9 * r4 + 58 * r2 + 29)/(2 * r12 * (r2 - 9))));
        goto exit; end r > 6;
        goto eigv [2 * order + (if ev then 1 else 2)];
    a0: q := q * q;
        small := q * (-0.5 + q * (7/128 + q * (-29/2304 + q * 0.68687/188.74368)));
        goto exit;
    b0: goto exit;
    a1: q := -q;
    b1: small := 1 + q * (-1 + q * (-0.125 + q * (0.015625 + q * (-1/1536 + q * (-11/36864
          + q * (0.49/5898.24 + q * (-0.55/94371.84 - q * 0.83/353894.4)))))))
          ; goto exit;
    a2: q := q * q;
        small := 4 + q * (5/12 + q * (-763/13824 + q * (0.1002401/7.962624
        - q * 0.1669068401/45.86471424))); goto exit;
    b2: q := q * q;
        small := 4 + q * (-1/12 + q * (5/13824 + q * (-0.289/79626.24 + q * 0.21391/45
        86471.424))); goto exit;
    a3: q := -q;
    b3: small := 9 + q * q * (0.0625 + q * (-0.015625 + q * (13/20480 + q *
          (5/16384 + q * (-0.1961/2359.296 + q * 0.609/104857.6)))));
        goto exit;
    a4: q := q * q;
        small := 16 + q * (1/30 + q * (0.433/864 - q * 0.5701/272160.0));
        goto exit;
    b4: q := q * q;
        small := 16 + q * (1/30 + q * (-0.317/864 + q * 0.10049/27216.0));
        goto exit;
    a5: q := -q;
    b5: small := 25 + q * q * (1/48 + q * q * (0.11/7741.44 - q * (1/147456.0 - q * 0.37
          /8918138.88))); goto exit;
```

```
      a6: q:=q * q;
          small:=36+q * (1/70+q * (0.187/43904.0+q *0.6743617/9293598.72));
          goto exit;
      b6: q:=q * q;
          small:=36+q * (1/70+q * (0.187/43904.0-q * 0.5861633/9293598.72));
          exit: end small;
          h:=sqrt(q); option:=false;
comment   Now we must choose a procedure which will be used to calculate an approximation.
          This is the result of a lot of testwork. We claim good results for q<250;
          if order<5 then begin goto lab [order+1];
          r0: if q<1.5 then option:=true; goto choice;
          r1: if q<4 then option:=true; goto choice;
          r2: if ev then begin if q<2 then option:=true end else
              if q<7 then option:=true; goto choice;
          r3: if ev then begin if q<4 then option:=true end else
              if q<7 then option:=true; goto choice;
          r4: if ev then begin if q<7.5 then option:=true end else
              if q<9 then option:=true; goto choice;
          end order<5;
          help:=h+1;
            if ev then help:=help+q * 0.004 else begin help:=help+1;
            if q>70 then help:=help+1; if q>110 then help:=help+1; end;
            if order>help then option:=true;
          choice: if option then lambda:=small else begin
            if ¬ ev then order:=order-1; lambda:=big; end;
          end of procedure startvalue;
3. procedure eigenvalue (q, order, ev, lambda, eps, m, n);
   value    q, order, ev, eps, m, n;
   real     q, lambda, eps;
   integer  order, m, n;
   Boolean  ev;
comment   The eigenvalue λ_order of Mathieu's equation is computed. A startvalue is necessary
          which can be calculated with the procedure startvalue. All parameters are the same
          as in the procedures startvalue and mataf;
begin     real gl, gu, reps, p1, q1, y, q2, la1, la2, fun 1, fun 2;
          integer i, r, k, a1, p, scale, s, scha, deps;
          array d [m:n]; switch jump:=11, 12, 13;
          procedure diagonal;
comment   The diagonalelements of the finite matrix are computed and stored in the array
          d[m:n];
begin     if even (order)=-1 then p:=-1 else if ev
          then p:=-2 else p:=0;
          for i:=m step 1 until n do d[i]:=(i+i+p)↑2;
          if p=-1 ∧ m=1 then begin
          if ev then d[1]:=1+q else d[1]:=1-q end;
          end of diagonal;
          procedure sturms sequence;
comment   The characteristic polynomial is calculated with the known recurrence relation.
          The number of changes of sign is counted and stored in a1. The integer r indicates
```

```
        if lambda > λ_order or lambda < λ_order, depending on r > 0 or r ≦ 0. Scaling is carried
        out, in order to prevent overflow.
begin   scale := a1 := 0; p1 := 1, q1 := d[m] − lambda;
        if q1 ≧ 0 then a1 := a1 + 1;
        y := (d[m+1] − lambda) * q1 − s * q2; p1 := q1; q1 := y;
        if p1 ≧ 0 ≡ q1 ≧ 0 then a1 := a1 + 1;
        for i := m+2 step 1 until n do begin
        y := (d[i] − lambda) * q1 − p1 * q2;
        if abs(y) > 10^56 then begin
        q1 := q1 * 10^−56; y := y * 10^−56; scale := scale + 1; end overflow;
        p1 := q1; q1 := y;
        if p1 ≧ 0 ≡ q1 ≧ 0 then a1 := a1 + 1; end i;
        if q1 = 0 ∧ p1 > 0 then a1 := a1 − 1; r := k − a1;
end of sturms sequence;
procedure bisection;
comment The method of bisection is used to get an approximation of lambda with accuracy
        reps. Use is made of this procedure, if the startvalue is not good enough, or if the
        regula falsi does not converge to the right eigenvalue. We start with reps = 0.1, and
        in every new computation with bisection, reps := 0.1 * reps. gl is the lower bound
        and gu the upperbound of lambda;
begin   reps := 0.1 * reps; lambda := (gl + gu)/2;
again:  sturms sequence;
        if r ≧ 1 then gu := lambda else gl := lambda;
        lambda := (gl + gu)/2;
        if gu − gl > reps * (if abs(gu) > 1 then abs(gu) else 1)
        then goto again;
        if reps < eps then goto exit; goto restart;
        end of bisection;

        reps := 1; q2 := q * q; la 1 := lambda; diagonal;
        k := n-order/2 + 0.1; if p = 0 then k := k + 1; deps := 0.1 * eps;
        s := if m = 1 ∧ p = −2 then 2 else 1;
comment upper and lower bound of lambda will be calculated;
        lambda := order↑2;
        if p = −1 then begin if ev then begin if lambda < 1 + q
        then begin lambda := (order + 2)↑2; if lambda > 1 + q
        then lambda := 1 + q end 1 end ev else
        if order = 1 then lambda := 1 − q end p;
        gl := lambda − 2 * q; gu := lambda + 2 * q;
        if la 1 > gl ∧ la 1 < gu then lambda := la 1;
comment Now the regula falsi will be applied to the characteristic polynomial. A check is
        made for convergence, in the absence of which the procedure bisection will be used;
restart: sturms sequence;
        if r ≧ 1 then gu := lambda + abs(lambda) * deps else
          gl := lambda − abs(lambda) * deps;
        la 1 := lambda; fun 1 := q1; scha := scale;
        if r ≠ 0 ∧ r ≠ 1 then bisection;
        if r = 0 then y := 1 else y := −1;
```

```
        y:=if reps<0.5 then reps * y else 10^-3 * y;
        lambda:=lambda+y * abs(lambda);
        if y<0 then begin if lambda<gl then lambda:=gl
        end else if lambda>gu then lambda:=gu
comment Second startvalue for the regula falsi in a somewhat arbitrary manner;
iteration: sturms sequence; la 2:=lambda; fun 2:=q1;
        if r≧1 then gu:=lambda+abs(lambda) * deps else
        gl:=lambda-abs(lambda) * deps;
        if r≠0 ∧ r≠1 then bisection;
          goto jump[scale-scha+2];
13:     fun 1:=fun 1 * 10^-56; goto 12;
11:     fun 2:=fun 2 * 10^-56; scale:=scale+1;
12:     scha:=scale;
        lambda:=(la 2 * fun 1-la 1 * fun 2)/(fun 1-fun 2);
        if gl>lambda ∨ gu<lambda then bisection;
        if abs((lambda-la 2)/lambda)<10 * eps ∨ abs((lambda-la 1)
        /lambda)<10 * eps then goto exit;
        la 1:=la 2; fun 1:=fun 2; goto iteration;
exit:   end of procedure eigenvalue;
        Remark:
        It is possible that one or two figures are lost in computing the characteristic poly-
        nomial. Therefore eps must be greater or equal to 100 * d, where d represents the
        relative machine accuracy. On the other hand eps should be at least 10^-3 to prevent
        convergence to a wrong eigenvalue;
4. procedure Fouriercoeff(ev, q, n, order, lambda, F, eps);
   value    q, n, order, ev, eps;
   Boolean  ev; array F;
   real     q, lambda, eps;
   integer  n, order;
comment The Fourier coefficients belonging to the eigenvalue lambda are computed. They
        are stored in the array F[0:n], such that A2k, or A2k+1 or B2k, or B2k+1 in F[k].
        To prevent mistakes B0 is taken as 0. We normed the Fourier coefficients so that
        F[order/2-0.1]=1. The parameters ev, q, order, lambda and eps are exactly the
        same as in the procedure eigenvalue. We used the method described by G. Blanch
        in [3];
begin   integer st, k, m, m1, m2, m3;
        real p, p1, abl; array G[1:n];
        real procedure V(m); value m; integer m;
        v:=(lambda-(2 * m+k)↑2)/q;
        if even(order)=1 then begin if ev then st:=2 else st:=4;
        end else if ev then st:=1 else st:=3;
comment st is an integer distinguishing the four kinds of following solutions:
                st=1  even solution  period 2π
                st=2  even    „        „    π
                st=3  odd     „        „    2π
                st=4  odd     „        „    π;
        k:=abs(2-st)-2; order:=order/2-0.1;
forward: g[1]:=V(1)+(if even(st)=1 then 0 else sign(st-2));
```

```
        if abs(g[1])<1 then begin m1:=1; goto backward end;
        g[2]:=V(2)-(if st=2 then 2 else 1)/g[1];
        if abs(g[2])<1 then begin m1:=2; goto backward end;
        for m:=3 step 1 until n do begin g[m]:=V(m)-1/g[m-1];
        if abs(g[m])<1 then begin m1:=m; goto backward end end m;
backward: m3:=n; p:=0;
tail:   m3:=m3+5; p1:=p; p:=0;
        for m:=m3 step-1 until n do p:=1/(v(m+1)-p);
        if abs((p-p1)/p)>100*eps then goto tail; g[n]:=p;
        for m:=n step-1 until m1+1 do begin m2:=m-1;
        p:=v(m); g[m2]:=1/(p-g[m]);
        if abs(p)<2 then goto top end m;
top:    m3:=m1; if m1=m2 then goto computation F;
        abl:=0.01*lambda;
        for m:=m1 step 1 until m2 do
        if abs(lambda-(2*m+k)↑2)<abl then begin
        m3:=m; goto calculation g; end;
comment It is possible that we loose some figures because lambda could be near to (2*m+k)↑2.
        This will appear, in particular, in the case of large order. If it appears, then we put
        the chaining at that point to prevent a large error;
calculation g: for m:=m1+1 step 1 until m3-1 do g[m]:=v(m)-1/g[m-1];
        for m:=m2 step-1 until m3+1 do g[m-1]:=1/(v(m)-g[m]);
computation f: if st=2 ∧ m3=1 then g[1]:=2*g[1];
        k:=if st=4 then 1 else 0; F[0]:=0; F[order]:=1;
        for m:=order+1-k step 1 until n-k do F[m+k]:=F[m+k-1]*g[m];
        for m:=order-1-k step-1 until 0 do F[m+k]:=F[m+k+1]/g[m+1];
        end of procedure Fouriercoeff;
        Remark:
        Although loss of figures can appear in the calculation of Fouriercoefficients in the
        top, that is between m1 and m2, normally the number of digits lost will not be greater
        then one or two. It is easy to check if this number is larger, because the Fourier-
        coefficients in the top will vary very much in absolute value.
        procedure Bessel (x, nmax, j, y, c, alarm);
        value x, nmax, c;
        real x;
        integer c, nmax;
        array j, y;
        label alarm;
comment In this procedure Bessel functions are calculated with fixed argument x>0 and
        order 0, 1, 2, ..., nmax. For Bessel functions of negative order the relation exists:
        B_n(x)=(-1)^n B_n(x). (In which B_n(x) is a Bessel function of order n). First B_0(x)
        and B_1(x) are computed with Chebyshev polynomials. For Bessel functions of
        higher order the relation B_{n-1}(x)+B_{n+1}(x)=(2n/x) B_n(x) is used. If n>x, numerical
        problems will occur when calculating Bessel functions of the first kind J_n(x), and
        a method of G. Blanch [3] is used.
    x:  The argument of the Bessel functions.
n max:  The upperbound for the order of the Bessel functions.
    j:  Array for the Bessel functions of the first kind. J[n]:=j_n(x).
    y:  Array for the Bessel functions of the second kind. y[n]=y_n(x).
```

```
    c :   c=0: calculation of Bessel functions of the first kind
          c=1:      "      "   "      "      "   "  first and second kind.
alarm:    nonlocal label, to which the procedure jumps,
          if x≦0 or nmax<0;
begin:    integer m, n, k, mstar;
          real a, p, q, y0, y1;
procedure besjyn (x, c, j, y, alarm);
value:    c, x; integer c; real x, j, y; label alarm;
comment   Bessel functions of the first kind (c=0) and of the second kind (c=1) are calculated
          with chebyshev polynomials. The accuracy is at least 13 digits. See [6].
          if c=0 then j:=j0(x) else j:=j0(x) and y:=y0(x);
begin     real p, q, h, hc, hs; array a[0:30];
          procedure cheb (n, x, a, y); value n, x; integer n;
          real x, y; array a;
begin     integer i; real u, pa, pb, pc;
          u:=x+x; pb:=0; pa:=0;
          for i:=n step-1 until 1 do begin pc:=u * pb+a[i]-pa;
          pa:=pb; pb:=pc; end; y:=pb * x-pa+a[0];
          end of cheb;
          if x<0 then goto alarm;
          if x≦8 then goto l1; h:=8/x;
```

```
A[0] :=  0.99946 03493 47518 7;
A[1] :=  0.00000 00000 00000 0;
A[2] := -0.00053 65220 46813 2;
A[3] :=  0.00000 00000 00000 0;
A[4] :=  0.00000 30751 84787 5;
A[5] :=  0.00000 00000 00000 0;
A[6] := -0.00000 00517 05945 4;
A[7] :=  0.00000 00000 00000 0;
A[8] :=  0.00000 00016 30646 5;
A[9] :=  0.00000 00000 00000 0;
A[10]:= -0.00000 00000 78640 9;
A[11]:=  0.00000 00000 00000 0;
A[12]:=  0.00000 00000 05168 3;
A[13]:=  0.00000 00000 00000 0;
A[14]:= -0.00000 00000 00430 5;
A[15]:=  0.00000 00000 00000 0;
A[16]:=  0.00000 00000 00043 3;
A[17]:=  0.00000 00000 00000 0;
A[18]:= -0.00000 00000 00005 1;
A[19]:=  0.00000 00000 00000 0;
A[20]:=  0.00000 00000 00000 7;
A[21]:=  0.00000 00000 00000 0;
A[22]:= -0.00000 00000 00000 1;
```

CHEB(22, H, A, P)

```
A[0] := -0.01555 58546 05337 0;
A[1] :=  0.00000 00000 00000 0;
A[2] :=  0.00006 83851 99426 1;
A[3] :=  0.00000 00000 00000 0;
A[4] := -0.00000 07414 49841 1;
A[5] :=  0.00000 00000 00000 0;
A[6] :=  0.00000 00179 72457 2;
```

```
A[7] :=  0.00000 00000 00000 0;
A[8] := -0.00000 00007 27191 6;
A[9] :=  0.00000 00000 00000 0;
A[10] :=  0.00000 00000 42201 2;
A[11] :=  0.00000 00000 00000 0;
A[12] := -0.00000 00000 03206 7;
A[13] :=  0.00000 00000 00000 0;
A[14] :=  0.00000 00000 00300 6;
A[15] :=  0.00000 00000 00000 0;
A[16] := -0.00000 00000 00033 4;
A[17] :=  0.00000 00000 00000 0;
A[18] :=  0.00000 00000 00004 3;
A[19] :=  0.00000 00000 00000 0;
A[20] := -0.00000 00000 00000 6;
A[21] :=  0.00000 00000 00000 0;
A[22] :=  0.00000 00000 00000 1;

CHEB(22, H, A, Q);
Q := H * Q;
H := X - 0.78539 81633 97448 3;
HC := COS(H);
HS := SIN(H);
H := 0.79788 45608 02865 4/SQRT(X);
J := H * (P * HC - Q * HS);
Y := H * (O * HC + P * HS):
  GO TO EINDE; L1:      H := X/8;

A[0] :=  0.15772 79714 74890 1;
A[1] :=  0.00000 00000 00000 0;
A[2] := -0.00872 34423 52852 2;
A[3] :=  0.00000 00000 00000 0;
A[4] :=  0.26517 86132 03336 8;
A[5] :=  0.00000 00000 00000 0;
A[6] := -0.37009 49938 72649 8;
A[7] :=  0.00000 00000 00000 0;
A[8] :=  0.15806 71023 32097 3;
A[9] :=  0.00000 00000 00000 0;
A[10] := -0.03489 37694 11408 9;
A[11] :=  0.00000 00000 00000 0;
A[12] :=  0.00481 91800 69467 6;
A[13] :=  0.00000 00000 00000 0;
A[14] := -0.00046 06261 66209 3;
A[15] :=  0.00000 00000 00000 0;
A[16] :=  0.00003 24603 28821 0;
A[17] :=  0.00000 00000 00000 0;
A[18] := -0.00000 17619 46907 8;
A[19] :=  0.00000 00000 00000 0;
A[20] :=  0.00000 00760 81635 9;
A[21] :=  0.00000 00000 00000 0;
A[22] := -0.00000 00026 79253 5;
A[23] :=  0.00000 00000 00000 0;
A[24] :=  0.00000 00000 78487 0;
A[25] :=  0.00000 00000 00000 0;
A[26] := -0.00000 00000 01943 9;
A[27] :=  0.00000 00000 00000 0;
A[28] :=  0.00000 00000 00041 3;
```

```
A[29]:=  0.00000 00000 00000 0;
A[30]:= -0.00000 00000 00000 8;

CHEB(30, H, A, J);
  IF C=0 THEN GO TO EINDE;

A[0] := -0.03314 61132 03284 9;
A[1] :=  0.00000 00000 00000 0;
A[2] := -0.27447 43055 29745 3;
A[3] :=  0.00000 00000 00000 0;
A[4] :=  0.17903 43140 77182 7;
A[5] :=  0.00000 00000 00000 0;
A[6] :=  0.26156 73462 55046 6;
A[7] :=  0.00000 00000 00000 0;
A[8] := -0.17730 20127 81143 6;
A[9] :=  0.00000 00000 00000 0;
A[10]:=  0.04719 66895 95763 4;
A[11]:=  0.00000 00000 00000 0;
A[12]:= -0.00728 79624 79552 1;
A[13]:=  0.00000 00000 00000 0;
A[14]:=  0.00075 31135 93257 8;
A[15]:=  0.00000 00000 00000 0;
A[16]:= -0.00005 63207 91410 6;
A[17]:=  0.00000 00000 00000 0;
A[18]:=  0.00000 32065 32537 7;
A[19]:=  0.00000 00000 00000 0;
A[20]:= -0.00000 01440 72332 8;
A[21]:=  0.00000 00000 00000 0;
A[22]:=  0.00000 00052 48794 8;
A[23]:=  0.00000 00000 00000 0;
A[24]:= -0.00000 00001 58375 5;
A[25]:=  0.00000 00000 00000 0;
A[26]:=  0.00000 00000 04026 3;
A[27]:=  0.00000 00000 00000 0;
A[28]:= -0.00000 00000 00087 5;
A[29]:=  0.00000 00000 00000 0;
A[30]:=  0.00000 00000 00001 6;

  CHEB(30, H, A, P);
  Y :=0.63661 97723 67581 3 * LN(X) * J + P;
EINDE: END BESJYN;
procedure besjye (x, c, j, y, alarm);
value c, x; integer c, real x, j, y; label alarm;
comment Bessel functions of the first or second are calculated, with Chebyshev polynomials.
        The accuracy is at least 13 digits. See [6]; if c=0 then j :=j1(x) else j :=j1(x) and
        y :=y1(x);
begin real p, q, j, hc, hs; array a [0:30];
procedure cheb (n, x, a, y); value n, x; integer n;
        real x, y; array a;
begin comment See besjyn; end;
if x<0 then goto alarm;
if x≦8 then goto l1; h :=8/x;

A[0] :=  1.00090 30408 60013 7;
A[2] :=  0.00089 89898 33085 9;
```

```
A[4] := -0.00000 39872 84300 5;
A[6] :=  0.00000 00617 76339 6;
A[8] := -0.00000 00018 71890 7;
A[10]:=  0.00000 00000 88169 0;
A[12]:= -0.00000 00000 05704 9;
A[14]:=  0.00000 00000 00469 9;
A[16]:= -0.00000 00000 00046 8;
A[18]:=  0.00000 00000 00005 5;
A[20]:= -0.00000 00000 00000 7;
A[22]:=  0.00000 00000 00000 1;
A[1] :=  0.00000 00000 00000 0;
A[3] :=  0.00000 00000 00000 0;
A[5] :=  0.00000 00000 00000 0;
A[7] :=  0.00000 00000 00000 0;
A[9] :=  0.00000 00000 00000 0;
A[11]:=  0.00000 00000 00000 0;
A[13]:=  0.00000 00000 00000 0;
A[15]:=  0.00000 00000 00000 0;
A[17]:=  0.00000 00000 00000 0;
A[19]:=  0.00000 00000 00000 0;
A[21]:=  0.00000 00000 00000 0;

CHEB(22, H, A, P);

A[0] :=  0.04677 77870 69535 3;
A[2] := -0.00009 62772 35491 6;
A[4] :=  0.00000 09138 61525 8;
A[6] := -0.00000 00209 59781 4;
A[8] :=  0.00000 00008 22919 3;
A[10]:= -0.00000 00000 46863 6;
A[12]:=  0.00000 00000 03515 2;
A[14]:= -0.00000 00000 00326 4;
A[16]:=  0.00000 00000 00036 0;
A[18]:= -0.00000 00000 00004 6;
A[20]:=  0.00000 00000 00000 7;
A[22]:= -0.00000 00000 00000 1;
A[1] :=  0.00000 00000 00000 0;
A[3] :=  0.00000 00000 00000 0;
A[5] :=  0.00000 00000 00000 0;
A[7] :=  0.00000 00000 00000 0;
A[9] :=  0.00000 00000 00000 0;
A[11]:=  0.00000 00000 00000 0;
A[13]:=  0.00000 00000 00000 0;
A[15]:=  0.00000 00000 00000 0;
A[17]:=  0.00000 00000 00000 0;
A[19]:=  0.00000 00000 00000 0;
A[21]:=  0.00000 00000 00000 0;

CHEB(22, H, A, Q);
Q := H * Q;
H := X - 2.35619 44901 92344;
HC := COS(H);
HS := SIN(H);
H := 0.79788 45608 0.2865 4/SQRT(X);
J := H * (P * HC - O * HS);
Y := H * (Q * HC + P * HS);
```

```
    GO TO  EINDE;
L1 :    H :=X/8;

A[0] :=  0.64835 87706 05264 9;
A[1] :=  0.00000 00000 00000 0;
A[2] := -1.19180 11605 41216 9;
A[3] :=  0.00000 00000 00000 0;
A[4] :=  1.28799 40988 57678;
A[5] :=  0.00000 00000 00000 0;
A[6] := -0.66144 39341 34543 3;
A[7] :=  0.00000 00000 00000 0;
A[8] :=  0.17770 91172 39728 3;
A[9] :=  0.00000 00000 00000 0;
A[10] := -0.02917 55248 06154 2;
A[11] :=  0.00000 00000 00000 0;
A[12] :=  0.00324 02701 82683 9;
A[13] :=  0.00000 00000 00000 0;
A[14] := -0.00026 04443 89348 6;
A[15] :=  0.00000 00000 00000 0;
A[16] :=  0.00001 58870 19239 9;
A[17] :=  0.00000 00000 00000 0;
A[18] := -0.00000 07617 58780 5;
A[19] :=  0.00000 00000 00000 0;
A[20] :=  0.00000 00294 97070 1;
A[21] :=  0.00000 00000 00000 0;
A[22] := -0.00000 00009 42421 3;
A[23] :=  0.00000 00000 00000 0;
A[24] :=  0.00000 00000 25281 2;
A[25] :=  0.00000 00000 00000 0;
A[26] := -0.00000 00000 00577 7;
A[27] :=  0.00000 00000 00000 0;
A[28] :=  0.00000 00000 00011 4;
A[29] :=  0.00000 00000 00000 0;
A[30] := -0.00000 00000 00000 2;

CHEB(30, H, A, P);
J :=P * H;
  IF  C=0  THEN  GO TO  EINDE;

A[0] :=  0.02030 41058 85934 2;
A[1] :=  0.00000 00000 00000 0;
A[2] := -0.12869 73843 81350 0;
A[3] :=  0.00000 00000 00000 0;
A[4] := -0.76729 63628 86645 9;
A[5] :=  0.00000 00000 00000 0;
A[6] :=  0.67561 57807 72187 7;
A[7] :=  0.00000 00000 00000 0;
A[8] := -0.22662 49915 56754 9;
A[9] :=  0.00000 00000 00000 0;
A[10] :=  0.04231 91803 53336 9;
A[11] :=  0.00000 00000 00000 0;
A[12] := -0.00513 16411 61061 1;
A[13] :=  0.00000 00000 00000 0;
A[14] :=  0.00044 04786 29867 1;
A[15] :=  0.00000 00000 00000 0;
A[16] := -0.00002 83046 40149 5;
A[17] :=  0.00000 00000 00000 0;
```

```
A[18]:=  0.00000 14166 24364 5;
A[19]:=  0.00000 00000 00000 0;
A[20]:= -0.00000 00568 84400 4;
A[21]:=  0.00000 00000 00000 0;
A[22]:=  0.00000 00018 75470 3;
A[23]:=  0.00000 00000 00000 0;
A[24]:= -0.00000 00000 51721 2;
A[25]:=  0.00000 00000 00000 0;
A[26]:=  0.00000 00000 01211 4;
A[27]:=  0.00000 00000 00000 0;
A[28]:= -0.00000 00000 00024 4;
A[29]:=  0.00000 00000 00000 0;
A[30]:=  0.00000 00000 00000 4;

   CHEB(30, H, A, P);
   Y:=0.63661 97723 67581 3 * (LN(X) * J-1/X)+P * H;
EINDE: END BESJYE;
       if x≦0 ∨ nmax<0 then goto alarm;
       k:=entier (x); if k>nmax then k:=nmax;
       besjyn (x, c, j[0], yo, alarm);
       if nmax=0 then goto exit;
       besjye (x, c, j[1], y1, alarm);
       if nmax=1 then goto exit; a:=2/x;
       for n:=1 step 1 until k-1 do j[n+1]:=j[n] * a * n-j[n-1];
       comment recurrence relation;
       if k=nmax then goto computation y;
backward: for n:=nmax, n+1 while n * a<2.5 do m:=n;
comment  Problems can occur, when k>nmax. The method of G. Blanch will now be applied;
       n:=m+5; p:=0; mstar:=m;
again:  n:=n+5; q:=p; p:=0;
       for m:=n step-1 until mstar do p:=1/(m * a-p);
       if abs(p-q)>5 * 10^-11 * p then goto again;
       comment 5 * 10^-11 is the relative accuracy of the I.C.L. 1905 computer;
       begin array g[k+1:nmax]; g[nmax]:=p;
       for m:=nmax step-1 until k+2 do g[m-1]:=1/((m-1) * a-g[m]);
       j[k+1]:=if abs(j[k-1])>abs(j[k]) then
       j[k-1]/(a * k/g[k+1]-1) else j[k] * g[k+1];
       for m:=k+2 step 1 until nmax do j[m]:=j[m-1] * g[m];
       end of innerblock;
computation y: if c=0 then goto exit; y[0]:=yo; y[1]:=y1;
       for n:=1 step 1 until nmax-1 do begin
       y[n+1]:=y[n] * a * n-y[n-1];
       if overflow then begin for m:=n+1 step 1 until nmax-1 do y[m]:=10^76;
       comment 10^76 is the largest real number of the I.C.L. 1905
       computer; goto exit; end of overflowtest;
       end of computation y;
exit:  end of Bessel;
6. procedure Mathieu (ev, order, kmax, j, z, f, eps, h, x, no, grtm, lab, Math);
   value   ev, order, kmax, eps, h, x, no, j, z, f;
   Boolean ev; label lab;
   integer order, kmax, no;
```

```
  real      eps, h, x, grtm, Math;
  array     j, z, f;
comment Depending on the integer no a normal or a modified Mathieu function is calculated,
        according to:
  no=1: ce_order(x, q) or se_order(x, q) formulae 20.2.3, 20.2.4 in [2]
  no=2: d/dx{ce_order(x, q) or se_order(x, q)}
  no=3: Mc^(j)_order(x, q) or Ms^(j)_order(x, q) (j=1, 2) formulae 20.6.7–20.6.10 in [2]
  no=4: d/dx{M^(j)_c order(x, q) or Ms^(j)_order(x, q)}
```

The procedure is built up such that other Mathieu functions can also be calculated easily. The parameters ev, order and eps are exactly the same as in eigenvalue, $h=\sqrt{q}$, kmax is an integer which indicates how many terms in the sum must be added. If kmax is too small, the procedure jumps to the nonlocal label lab. Normally kmax=n+10 with n calculated by mataf, will be sufficient. To prevent stopping of the procedure when there is no convergence two terms are always added to the sum. The parameter grtm gives the largest term in the sum, so it is easily to check if there are digits lost in forming the sum. The calculated Mathieu function is stored in math. In the array F Fourier coefficients, in j Bessel functions of the first kind of argument $\sqrt{q}e^{-x}$, and in z Bessel functions of the jst kind must be stored. j and z are used only if modified Mathieu functions are required.

Bounds of the arrays:

F[0 :kmax]: F[k] = A_{2k}, A_{2k+1}, B_{2k} or B_{2k+1}

J, Z[−order/2+0.1 :knmax+order/2+1.9]: j[k] = $J_k(\sqrt{q}e^{-x})$

Z[k] = $J_k(\sqrt{q}e^{x})$ or $J_k(\sqrt{q}e^{x})$.

The Fourier coefficients must be normed such that F[order/2−0.1]=1;

```
begin integer lb, p, s, i, k, signum; array si, co[0 :2 * kmax+1];
real sum, asum, t, term, norm, u1, u2, ez;
procedure test;
comment test adds the absolute values of terms, and seeks the largest element in the sum;
begin t :=abs(t); asum :=asum+t;
      if t >grtm then grtm :=t; end;
procedure add;
comment add adds one term to the sum. Test is called to insure good convergence;
begin switch choice :=11, 12, 13, 14;
goto choice [no];
11: t :=F[k] * (if ev then co[k+k+p] else si[k+k+p]);
      sum :=sum+t; test; goto exit;
12: if ev then begin t :=F[k] * si[k+k+p] * (k+k+p);sum :=sum−t
    end else begin t :=F[k] * co[k+k+p] * (k+k+p);sum :=sum+t end;
    test; goto exit;
13: t :=F[k] * j[k−s] * Z[k+s+p] ;term :=t; test;
    t :=F[k] * j[k+s+p] * Z[k−s] ;term :=term+signum * t; test;
    sum :=sum+even(k+s) * term; goto exit;
14: t :=(s+s+p) * F[k] * j[k−s] * Z[k+s+p]; term :=t; test;
    t :=(s+s+p) * F[k] * j[k+s+p] * Z[k−s] * signum; term :=term−t; test;
    t :=u1 * F[k] * j[k−s+1] * Z[k+s+p]; term :=term+t; test;
    t :=u1 * F[k] * j[k+s+p+1] * Z[k−s] * signum; term :=term+t; test;
    t :=u2 * F[k] * j[k−s] * Z[k+s+p+1]; term :=term−t; test;
    t :=u2 * F[k] * j[k+s+p] * Z[k−s+1] * signum; term :=term−t; test;
    sum :=sum+even(k+s) * term;
exit: end of add;
```

```
procedure sico;
comment Computed are sin((2m+p)x) and cos((2m+p)x), p=0(1)2kmax+1 with the aid
        of the recurrence relation:
        cos(kx)=cos((k-1)x)cos(x)-sin((k-1)x)sin(x)
        sin(kx)=sin((k-1)x)cos(x)-cos((k-1)x)sin(x).
They are stored in the arrays si, co[0:2 * kmax+1] according to
si[k]:=sin(kx), co[k]:=cos(kx);
begin co[0]:=1; si[0]:=0; co[1]:=cos(x); si[1]:=sin(x);
    for k:=2 step 1 until 2 * kmax+1 do begin
    co[k]:=co[k-1] * co[1]-si[k-1] * si[1];
    si[k]:=si[k-1] * co[1]-co[k-1] * si[1]; end k;
end of sico;
procedure normalisation;
comment Computed is the norm by which the Fourier coefficients have to be divided. We
        normed the Fourier coefficients according to Ince. See formula 20.5.4 in [2];
begin norm:=0; term:=1; k:=s+1;
    for k:=k-1 while k≥1 ∧ term>eps do begin
    term:=F[k]↑2; norm:=norm+term end; term:=1;
    for k:=s+1, k+1 while term>eps do begin
    term:=F[k]↑2; norm:=norm+term end;
    norm:=sqrt(norm+(if ev ∧ p=0 then 2 else 1) * F[0]↑2);
    end of normalisation;
    if no=4 then begin ez:=exp(x); u1:=h/ez; u2=h * ez end;
    if no<3 then sico;
    lb:=if ¬ ev ∧ even(order)=1 then 1 else 0;
    grtm:=sum:=0; i:=s:=order/2-0.1;
    signum:=if ev then 1 else-1;
    p:=if even(order)=1 then 0 else 1;
again: asum:=0;
       for k:=i, i-1 do begin if k<lb then goto back; add end;
       if asum<eps * abs(sum) ∨ asum<10^-50 then goto back;
       i:=i-2; goto again;
back:  i:=s+1;
       loop: asum:=0;
       for k:=i, i+1 do begin if k>kmax then goto lab; add end;
       i:=i+2; if asum<eps * abs(sum) ∨ asum<10^-50 then math:=sum
       else goto loop;
       if no<3 then begin normalisation; grtm:=grtm/norm;
       math:=math/norm end else if order=0 then math:=math/2;
end    of procedure Mathieu;
```

Remark:

The procedure gives Mathieu functions with relative accuracy. If the required Mathieu function is small in relation to Mathieu functions in its neighbourhood then there will be loss of figures, just as happens in the calculation of sin(x). Because Mathieu functions usually appear in series of Mathieu functions it may be better to calculate the Mathieu functions with absolute accuracy.

REFERENCES

[1] N. W. McLachlan, *Theory and applications of Mathieu functions*, Clarendon Press, Oxford, England, 1947.
[2] M. Abramowitz and I. Stegun, Handbook of Mathematical Functions N.B.S., *Applied Mathematical Series* 55, U. S. Government Printing Office, Washington D.C., 1964.
[3] G. Blanch, Numerical evaluation of continued fractions, *Siam Review* 6, p. 383, 1964.
[4] D. J. Green and S. Michaelson, Series solution of certain Sturm Liouville eigenvalue problems, *The Computer Journal* 7, p. 322, 1965.
[5] J. H. Wilkinson, *The algebraic eigenvalue problem*, 1965.
[6] Mathematical tables 5, tables 12 and 14, p. 31 and 33.
[7] J. H. Wilkinson, Rigorous error bounds for computed eigensystems. *The Computer Journal* 4, p. 230.