One- and multidimensional Fibonacci search – very easy!

0. Content

1. Introduction / Preliminary remarks

About the Fibonacci numbers, there are many papers in the literature, therefore, discusses the features in this article only briefly. In the literature and on the Internet. many treatises can be found. Many descriptions of search using Fibonacci numbers assume that starting from an interval [a, b] within which the minimum or maximum of a function to be determined, will be determined according to certain rules with m given search steps more and more smaller intervals in where the minimum or maximum is sought. This process is of course correct, but the derived algorithm is rather complex and for a multi-dimensional search highly impractical. As a by-product of my diploma thesis (about 35 years ago) that dealt with time-optimal control, I had designed an algorithm, which extremely simplified the Fibonacci search and easily applicable to multidimensional problems made. From page 5 you will find a Python program for the n-dimensional Fibonacci search.
Recently added was a treatise on the Fibonacci search for discrete functions (page 4).

2. Short description of the Fibonacci numbers

For Fibonacci numbers is the following formation rule:

$F_0 = 0$
$F_1 = 1$
$F_m = F_{m-2} + F_{m-1}$ for m>1

This results in the infinite sequence of numbers: 0,1,1,2,3,5,8,13,21,…..

The individual values can not only recursive, but also directly calculate with the formula

$$F_m = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^m - \left(\frac{1-\sqrt{5}}{2}\right)^m\right] \qquad \text{Eq. (1)}$$

3. Description of a simple algorithm

At the start of the Fibonacci search it has to be set with how many steps the Minimum / Maximum of a function y = f (x) in an interval x ϵ [a, b] shall be found.

Is m the number of search steps, at the end of the calculations, it remains an uncertainty

$$\varepsilon_m = \frac{b-a}{F_{m+2}} \qquad \text{Eq. (2)}$$

Thus m must be chosen so that the residual error is correspondingly small.

Dipl.-Ing (TU) Klaus-Eckart Schulz / December 2009
Birnbaumring 64 Addition Aug. 2012
Correction Nov. 2012
D-13159 Berlin Germany

It is important that in the one-dimensional search that the function in the interval must be unimodal. In the multi-dimensional search the examined area must be convex. That means, in the interval may exist only one minimum / maximum. Otherwise the search will not be successful.
As noted in item 1, the usual algorithm functions such that ever-smaller intervals are formed. After the m-th step is then the solution found.

With the algorithm described here, the method is extremely simple.

Basis for the simplified algorithm was the idea to avoid the determination of the various intervals. There were the distances between the various examined points determined. As a result was found a very simple relationship.
Consider the interval to be examined [a, b]. The number of search steps is m.
Furthermore, we assume that a minimum should be searched.

In this interval, the first two points to be examined are

$$x_1 = a + \frac{b-a}{F_{m+2}} \cdot F_m \qquad \text{and} \qquad x_2 = a + \frac{b-a}{F_{m+2}} \cdot F_{m+1}$$

The distances of x_1 to the left interval boundary and x_2 to the right interval boundary are equal.
In our considerations, we start the calculation from the left interval boundary a.

The distance between the first Search point and a is

$$v_1 = \frac{b-a}{F_{m+2}} \cdot F_m$$

For the distance of the second Search point from the first follows

$$v_2 = \frac{b-a}{F_{m+2}} \cdot F_{m+1} - \frac{b-a}{F_{m+2}} \cdot F_m = \frac{b-a}{F_{m+2}} \cdot \left(F_{m+1} - F_m\right) = \frac{b-a}{F_{m+2}} \cdot F_{m-1}$$

The point with the smaller function value is taken as a starting point for the next search step. Other considerations, which will not be discussed in detail here showed the distance of the third Point from the previous search to

$$v_3 = \frac{b-a}{F_{m+2}} \cdot F_{m-2}$$ and so on up to the m-th search step $$v_m = \frac{b-a}{F_{m+2}} \cdot F_1 = \frac{b-a}{F_{m+2}}$$

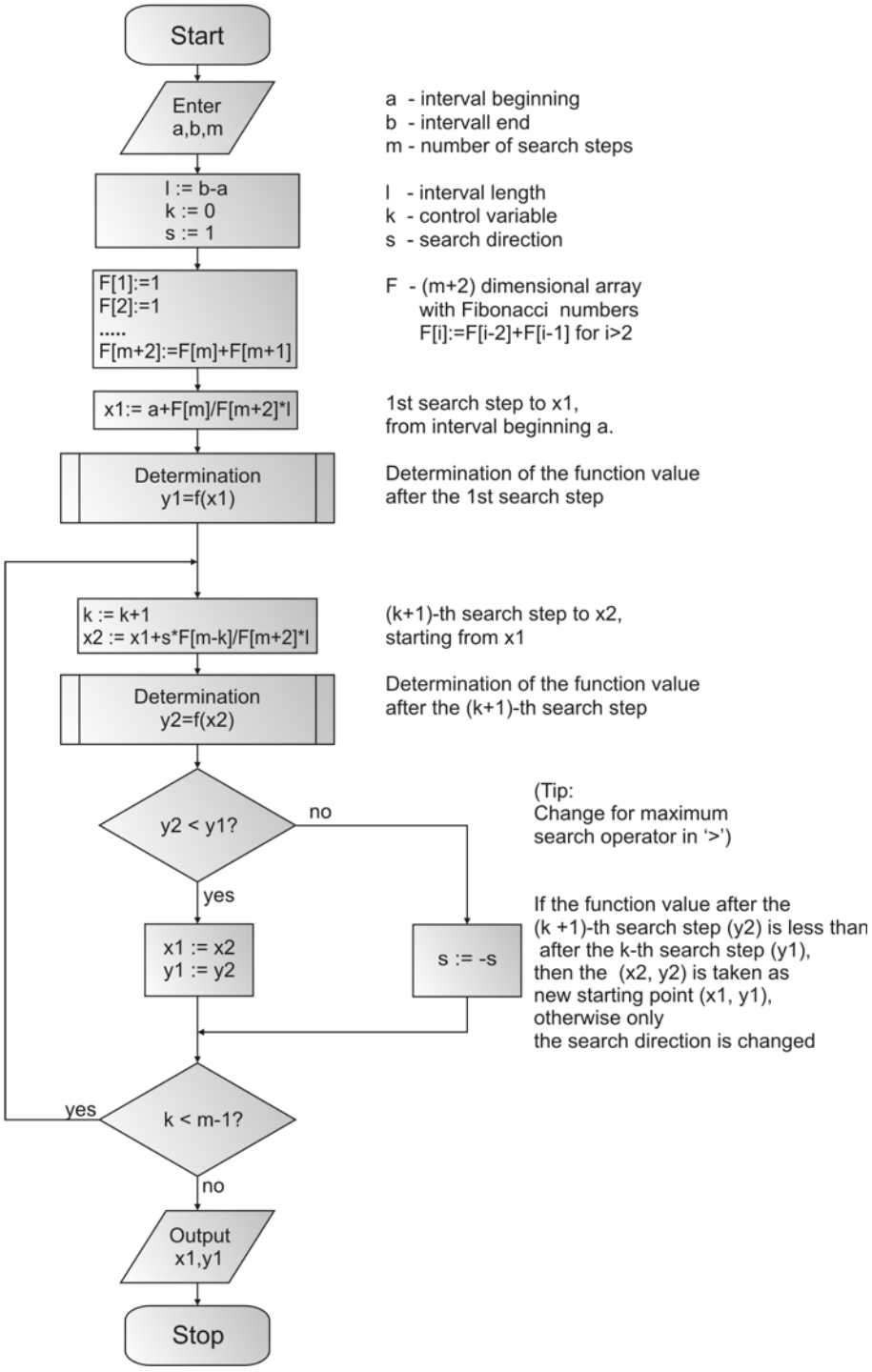
Generally follows the step width of the k-th search step to

$$v_k = \frac{b-a}{F_{m+2}} \cdot F_{m-k+1} \qquad \text{Eq. (3)}$$

This is the basis of a flow chart for the one-dimensional Fibonacci search algorithm like shown on the next page. It is so simple, so that the extension to higher dimensions is not a problem.

Simple minimum search with Fibonacci numbers

Dipl.-Ing. (TU) Klaus-E. Schulz
klaus-e.schulz@t-online.de
D-13159 Berlin
Birnbaumring 64
Germany



4. Fibonacci search for discrete functions

Is not given a function y = f (x) , but it is only y in the form of a finite sequence of numbers as they can for example be stored in an array, then the algorithm like described before must be modified somewhat.

Suppose in an array **Ay** of the length l is the sequence consisting of **l** discrete values stored. From this values the minimum or maximum value shall be determined.

It is understood immediately that in this case, only integer increments may be used.

From Eq. (3) shows that this is true, if the interval length (b-a) is equal F_{m+2}.

The following three features are to be considerd:

1. The length l of the array Ay is not the difference between the first and last index, but it is larger by one.
2. From Eq. (2) results from b-a = F_{m+2} an uncertainty interval of 1.
3 In the Fibonacci search the function values at the interval ends are not considered. When the search is discrete but this is necessary.

This follows

$$\frac{b-a}{F_{m+2}} = \frac{l-1}{F_{m+2}} = 1 \qquad \Rightarrow \qquad l-1 = F_{m+2}$$

For the optimal calculation of the minimum / maximum of such a sequence arise now following rules:

1. The length of the array or the number of discrete values plus 1 must be a Fibonacci number.
2. After determination of the index m of the Fibonacci number can then be determined by F_m steps the minimum / maximum.
3. To consider all of the values, at the beginning of the sequence a dummy is to be added. The value is unimportant, since it is only needed as a starting point and does not enter into the calculation. If no dummy added, the length of the first search step must be reduced by 1

Sample

Given is the following sequence: {2,3,5,6,8,9,11,13,15,17,19,18}

There are 12 values. Added by 1 is 13 and this is the Fibonacci number F_7. Consequently, the number of search steps is m = 7-2 = 5.

With the added dummy follows the sequence {0,2,3,5,6,8,9,11,13,15,17,19,18}

The order of search lengths is: F_5, F_4, F_3, F_2, F_1 (5,3,2,1,1).

At the minimum search sequentially following the above sequence numbers would be queried:

8->13->5->3->2 This follows the minimum value of 2.

At maximum, it would be the following search order:

8->13->17->19->18. This follows the maximum value to 19.

5. Program for the n-dimensional Fibonacci search

A Python script that the minimum of a function y = f (x_1,…, x_n) = f (x) calculates is to find on the following pages. The number of independent variables is arbitrary (n ≥ 1). This was made possible by recursively calling the search function. The independent variables are the n components of the vector x, x[0], ..., x [n-1].
Basis of the program, the slightly modified algorithm for the one-dimensional search (see page 3).

For the correct operation of the program, the following entries are required:

1. In the array m are for the n variables x[0], ..., x [n-1] entered the required number of search steps.
2. In the array intervall are for the n variables x[0], ..., x [n-1] entered the search interval boundaries.
3. In the function definition def FUNCTION (x): the function y = f(x) is calculated and returned.

To the maximum of a function y = f (x) to determine with this program, there are two possibilities:

1. In all cases, where y_1 and y_2 be compared, the comparison operator '<' will replaced by '>' .
   Or even simpler
2. Set function y = -f(x).

The program can be easily ported to any other programming languages.

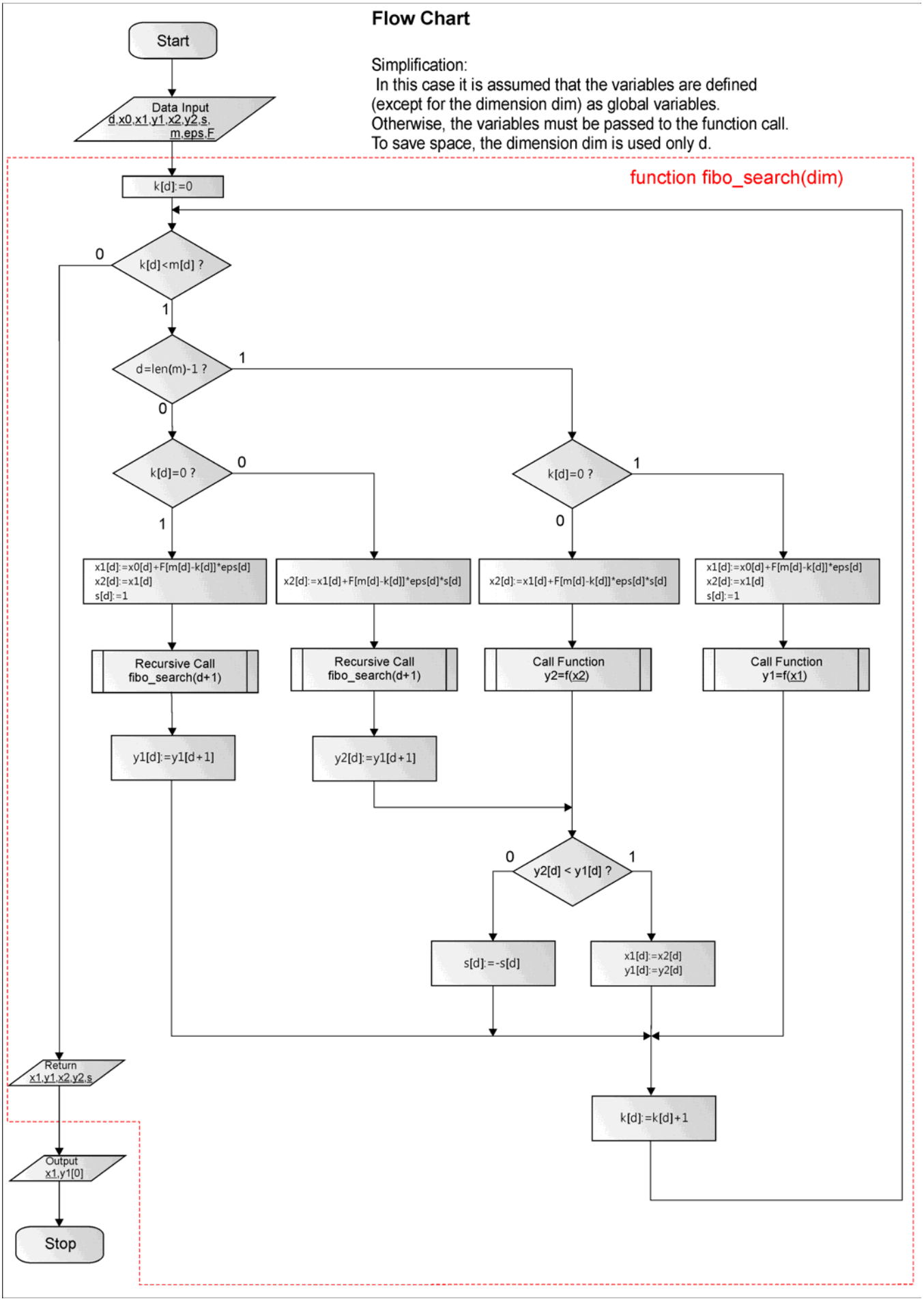
It can be downloaded from

http://www.klaus-e-schulz.de/dokumente/fibonacci_search.py

```
#!/usr/bin/env python

# Program for an n-dimensional search of the minimum of a function
# by means of the Fibonacci numbers.

# The program is written as an python file(Version 2.5.4).
# It can be easily ported to any other programming language.

#*****************************************************************************************
# Version 2.1 / revised version
# Date: 06.12.2012
# Author: Klaus-Eckart Schulz / Berlin
# The program can be used as required, modified and passed on,
# however, the reference to the author maintained.
# For any application the author assumes no liability.
#*****************************************************************************************

# Very simple example for n=5-dimensional

#------------------ Data entry ------------------------------------------------------
m=[4,6,8,10,12]         # Number of search steps for each of the n=5 intervals.
intervall=[[0,8],       # n=5 search intervals
           [-8,0],
           [0,8],
           [0,8],
           [0,8]
           ]

# Definition of the function y=f(x1,...,xn) =f(x)
def  FUNKTION(x):
        return  (x[0]-4)**2+(x[1]+4)**2+(x[2]-4)**2+(x[3]-4)**2+(x[4]-4)**2
#-------------------------------------------------------------------------------------

#===== Recursive procedure for the n-dimensional Fibonacci search =====#
def  fibo_search(dim):
    global k,x0,x1,y1,x2,y2,s,m,eps,F
    k[dim]=0
    while k[dim] < m[dim]:
        if dim==len(m)-1:
            if k[dim]==0:
                x1[dim]=x0[dim]+F[m[dim]-k[dim]]*eps[dim]
                x2[dim]=x1[dim]
                y1[dim]=FUNKTION(x2)
                s[dim]=1
            else:
                x2[dim]=x1[dim]+F[m[dim]-k[dim]]*eps[dim]*s[dim]
                y2[dim]=FUNKTION(x2)
                if y2[dim]<y1[dim]:
                    x1[dim]=x2[dim]
                    y1[dim]=y2[dim]
                else:
                    s[dim]=s[dim]*(-1)
        else:
            if k[dim]==0:
                x1[dim]=x0[dim]+F[m[dim]-k[dim]]*eps[dim]
                x2[dim]=x1[dim]
                s[dim]=1
                fibo_search(dim+1)
                y1[dim]=y1[dim+1]
            else:
                x2[dim]=x1[dim]+F[m[dim]-k[dim]]*eps[dim]*s[dim]
```

```
                fibo_search(dim+1)
                y2[dim]=y1[dim+1]
                if y2[dim]<y1[dim]:
                    x1[dim]=x2[dim]
                    y1[dim]=y2[dim]
                else:
                    s[dim]=s[dim]*(-1)
        k[dim]=k[dim]+1
#############################################################################

#+++++ Initialization of the variables ++++++++++++++++++++++++++++++++++++

# Determination of the array for the Fibonacci numbers  'F'
m_max=max(m)
F=[0]*(m_max+3)
F[0]=0.0
F[1]=1.0
for i in range(2,m_max+3):
    F[i]=F[i-1]+F[i-2]

# Determination of 'eps[]=interval_length/Fib[m+2]' for all dimensions
# eps[] is the max. possible absolute error for each dimension
eps=([0]*len(m))
for i in range(0,len(m)):
    eps[i]=(intervall[i][1]-intervall[i][0])/F[m[i]+2]

# Arrays 's' for the search direction, and 'k' for the control variables
s=[1]*len(m)
k=[0]*len(m)

x0=[0.0]*len(m)
x1=[0.0]*len(m)
x2=[0.0]*len(m)
y1=[0.0]*len(m)
y2=[0.0]*len(m)

for i in range(0,len(m)):
    x0[i]=intervall[i][0]

#   Start of the Program
fibo_search(0)

#  Result Output
print x1              # n-dimensional array with the n solutions
print y1[0]           # Result y1=f(x1)
```

Very simple n-dimensional search with Fibonacci numbers (n≥1, this means dim≥0)



Appendix

6. Derivation of the Fibonacci formula

Introductory Remarks

For the Derivation of formula a tool is used, as is often the case in mathematics. The original function from the original area (time area) is first transformed into the image area, where it is processed with simple tools and then transformed back into the original area. For example, the Laplace transform is a very suitable tool for solving differential equations. With it a time function f(t) will assigned with the so-called one-sided Laplace transformation by means of the Laplace integral

$$L\{f(t)\} = F(p) = \int_0^\infty f(t) \cdot e^{-pt} dt \qquad mit \qquad p = \delta + j\varpi$$

a complex function of frequency.

The Laplace transform of derivatives can be determined as follows:

$$L\{f'(t)\} = L\{f^{(1)}(t)\} = p \cdot L\{f(t)\} - f(0)$$

Or generally

$$L\{f^{(n)}(t)\} = p^n \cdot L\{f(t)\} - \sum_{k=0}^{n-1} p^{n-1-k} \cdot f^{(k)}(0)$$

Important is the time shifting function to right.

$$L\{f(t-t_1)\} = e^{-pt_1} \cdot L\{f(t)\} \qquad (t_1 > 0)$$

Using Laplace transform a differentiation in the original area shall be a multiplication in the image area.

For the reverse transformation back to the original field can be used the so-called Residue formula.

$$f(t) = L^{-1}\{F(p)\} = \sum_\nu \operatorname{Res}_{p=p_\nu}\{F(p) \cdot e^{pt}\} = \sum_\nu \lim_{p \to p_\nu} \frac{1}{(n-1)!} \cdot \frac{d^{n-1}}{dp^{n-1}}\{F(p) \cdot e^{pt} \cdot (p-p_\nu)^n\}$$

Here, n is the order of the pole at $p=p_v$.

If f(t) not a continous function but a sequence $[x_n]$, then can be used the discrete Laplace-Transformation (L*) to transform with the relation

$$L^*\{x_n\} = X_L^*(p) = \sum_{n=0}^\infty x_n \cdot e^{-npT}$$

If in this formula will set $e^{pT}=z$, then it comes to so-called z-transform, which we will use for our case. This produces the following transformation and inverse transformation rules:

$$Z\{x_n\} = X^*(z) = \sum_{n=0}^\infty x_n \cdot z^{-n}$$

$$x_n = \sum_k \operatorname{Res}_{z=z_k}\left[X^*(z) \cdot z^{n-1}\right] = \sum_k \lim_{z \to z_k} \frac{1}{(p-1)!} \cdot \frac{d^p}{dz^p}\{X^*(z) \cdot z^{n-1} \cdot (z-z_k)^p\}$$

In this case p is the order of the pole at $z = z_k$.

Again, the time shifting is important, especially for the solution of our problem.

It is

$$Z\{x_{n-k}\} = z^{-k} \cdot Z\{x_n\}$$

and

$$Z\{x_{n+k}\} = z^k \cdot \left[Z\{x_n\} - x_0 - x_1 z^{-1} - \ldots - x_{k-1} z^{-(k-1)}\right] = z^k \cdot \left[Z\{x_n\} - \sum_{i=0}^{k-1} x_i z^{-i}\right]$$

Such as the Laplace transform is suitable for solving differential equations, the z-transform can be used to solve difference equations.

Calculation

For the Fibonacci numbers is (look page1)

$F_m = F_{m-2} + F_{m-1}$ for m>1 and the initial conditions $F_0=0$ und $F_1=1$

To can use the initial conditions, the 2nd time shift must be applied.

It follows (using the variable x)

$$x_{m+2} = x_m + x_{m+1} \quad \text{with} \quad x_0=0 \text{ and } x_1=1$$

The application of the z-transformation yields

$$z^2 \cdot \left[X^*(z) - x_0 - x_1 \cdot z^{-1} \right] = X^*(z) + z\left[X^*(z) - x_0 \right]$$
$$z^2 \cdot \left[X^*(z) - 0 - 1 \cdot z^{-1} \right] = X^*(z) + z\left[X^*(z) - 0 \right]$$

This follows

$$X^*(z) = \frac{z}{z^2 - z - 1}$$
$$X^*(z) = \frac{z}{(z-z_1)(z-z_2)} \quad \text{with} \quad z_{1,2} = \frac{1}{2} \pm \frac{\sqrt{5}}{2} = \frac{1}{2}\left(1 \pm \sqrt{5}\right)$$

Now the reverse transformation takes place by means of Residue formula.

Since z_1 and z_2 are simple poles, is for both cases p = 1.

$$x_m = \sum_k \operatorname*{Res}_{z=z_k} \left[X^*(z) \cdot z^{m-1} \right] = \sum_k \lim_{z \to z_k} \frac{1}{(p-1)!} \cdot \frac{d^{p-1}}{dz^{p-1}} \left\{ X^*(z) \cdot z^{m-1} \cdot (z-z_k)^p \right\}$$
$$x_m = \sum_{k=1}^{k=2} \lim_{z \to z_k} \left\{ X^*(z) \cdot z^{m-1} \cdot (z-z_k)^p \right\}$$
$$x_m = \lim_{z \to z_1} \frac{z \cdot z^{m-1}}{(z-z_1)(z-z_2)}(z-z_1) + \lim_{z \to z_2} \frac{z \cdot z^{m-1}}{(z-z_1)(z-z_2)}(z-z_2)$$
$$x_m = \frac{z_1^m}{z_1 - z_2} + \frac{z_2^m}{z_2 - z_1} = \frac{z_1^m - z_2^m}{z_1 - z_2} = \frac{\left[\frac{1}{2}\left(1+\sqrt{5}\right)\right]^m - \left[\frac{1}{2}\left(1-\sqrt{5}\right)\right]^m}{\frac{1}{2}\left(1+\sqrt{5}\right) - \frac{1}{2}\left(1-\sqrt{5}\right)}$$

$$\underline{\underline{x_m = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^m - \left(\frac{1-\sqrt{5}}{2}\right)^m\right]}}$$

Literature

Gerhard Wunsch: Systemanalyse Bd. 1, Lineare Systeme, VEB Verlag Technik Berlin,1969

Dipl.-Ing (TU) Klaus-Eckart Schulz / December 2009
Birnbaumring 64 Addition Aug. 2012
Correction Nov. 2012
D-13159 Berlin Germany