

Ein- und mehrdimensionale Fibonacci-Suche - ganz einfach!

0. Inhalt

1. Einleitung / Vorbemerkungen.....	Seite 1
2. Kurzbeschreibung der Fibonacci-Zahlen.....	Seite 1
3. Beschreibung eines einfachen Algorithmus'.....	Seite 1
4. Fibonacci-Suche bei diskreten Funktionen.....	Seite 4
5. Programm für die n-dimensionale Fibonacci-Suche.....	Seite 5

1. Einleitung / Vorbemerkungen

Über Fibonacci-Zahlen gibt es in der Literatur viele Abhandlungen, deshalb wird auf die Besonderheiten in diesem Artikel nur kurz eingegangen.

Diese Zahlen haben auch für Suchfunktionen eine hervorragende Bedeutung. Auch hierzu findet man in der Literatur und im Internet viele Abhandlungen.

Viele Beschreibungen von Suchfunktionen mittels Fibonacci-Zahlen gehen davon aus, dass ausgehend von einem Intervall $[a,b]$ innerhalb dessen das Minimum bzw. Maximum einer Funktion bestimmt werden soll, nach bestimmten Regeln mit m vorgegebenen Suchschritten immer weitere kleinere Intervalle bestimmt werden, in denen das gesuchte Minimum bzw. Maximum liegt.

Dieses Verfahren ist natürlich korrekt, aber der daraus abgeleitete Algorithmus ist recht unübersichtlich und für eine mehrdimensionale Suche äußerst unpraktikabel.

Sozusagen als Nebenprodukt meiner Diplomarbeit (vor ca. 35 Jahren), die sich mit zeitoptimalen Steuerungen befasste, hatte ich einen Algorithmus entworfen, der die Fibonacci-Suche extrem vereinfachte und auch problemlos auf mehrdimensionale Probleme anwendbar machte. Im damaligen Fall waren es bis zu 4-dimensionale Berechnungen.

Neu hinzugenommen wurde eine Abhandlung zur Fibonacci-Suche bei diskreten Funktionen (Seite 4).

Ab Seite 5 finden Sie ein Python-Programm für die n-dimensionale Fibonacci-Suche.

2. Kurzbeschreibung der Fibonacci-Zahlen

Für Fibonacci-Zahlen gilt folgende Bildungsregel:

$$\begin{aligned}F_0 &= 0 \\F_1 &= 1 \\F_m &= F_{m-2} + F_{m-1} \quad \text{für } m > 1\end{aligned}$$

Damit ergibt sich die unendliche Zahlenfolge: 0, 1, 1, 2, 3, 5, 8, 13, 21,

Die einzelnen Werte können aber nicht nur rekursiv, sondern auch direkt berechnet werden mit

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

3. Beschreibung eines einfachen Algorithmus`

Bei der Fibonacci-Suche wird zu Beginn festgelegt mit wie viel Schritten das Minimum/Maximum einer Funktion $y=f(x)$ in einem Intervall $x \in [a,b]$ gefunden werden soll.

Ist m die Anzahl der Suchschritte, so ergibt sich am Ende der Berechnung ein Unsicherheitsintervall von

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

Damit muss m so gewählt werden, dass der Restfehler entsprechend klein wird.

Wichtig ist, dass bei der eindimensionalen Suche die Funktion in dem untersuchten Intervall unimodal ist. Bei der mehrdimensionalen Suche muss das untersuchte Gebiet konvex sein. In jedem Fall darf also nur ein lokales Minimum/Maximum in dem untersuchten Intervall auftreten. Anderenfalls führt die Suche nicht zum Erfolg.

Wie bereits unter Pkt. 1 bemerkt, funktioniert der übliche Algorithmus derart, dass immer kleiner werdende Intervalle gebildet werden. Nach dem m-ten Schritt wird dann die Lösung gefunden.

Mit dem hier beschriebenen Algorithmus wird das Verfahren extrem einfach.

Grundlage für den vereinfachten Algorithmus war der Gedanke, die Bestimmung der verschiedenen Intervalle zu umgehen. Dazu wurde ermittelt, wie sich die Abstände der untersuchten Punkte verhalten. Dabei ergab sich dann ein ganz einfacher Zusammenhang.

Gegeben sei das zu untersuchende Intervall $[a,b]$. Die Anzahl der Suchschritte sei m . Weiterhin gehen wir der einfacheren Erklärung wegen davon aus, dass ein Minimum gesucht werden soll.

In diesem Intervall sind die ersten 2 zu untersuchenden Punkte

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

Die Abstände von x_1 zur linken Intervallgrenze und x_2 zur rechten Intervallgrenze sind gleich.

Bei unseren Betrachtungen gehen wir jetzt nur von der linken Intervallgrenze a aus.

Der Abstand des 1. Suchpunktes von a ist

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

Für den Abstand des 2. Suchpunktes vom 1. folgt

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

Je nachdem an welchem der Punkte der kleinere Funktionswert auftritt, wird dieser als Ausgangspunkt für den folgenden Suchpunkt genommen.

Weitere Betrachtungen, auf die hier nicht näher eingegangen wird, ergeben den Abstand des 3. Suchpunktes vom vorhergehenden zu

$$v_3 = \frac{b-a}{F_{m+2}} \cdot F_{m-2} \quad \text{usw. bis zum m-ten Suchschritt} \quad v_m = \frac{b-a}{F_{m+2}} \cdot F_1 = \frac{b-a}{F_{m+2}}$$

Allgemein ergibt sich für die Schrittweite beim k -ten Suchschritt ($k=1, \dots, m$)

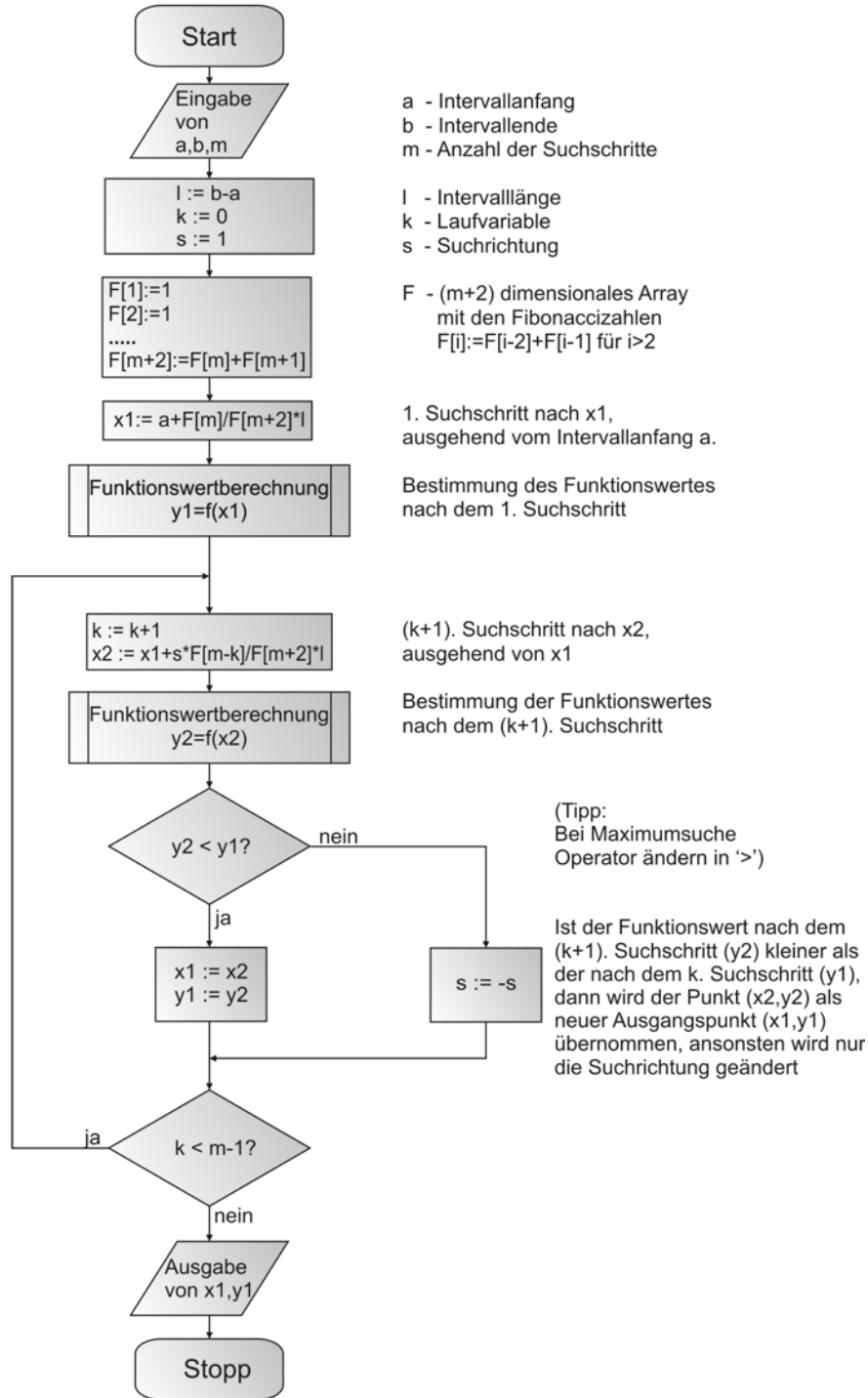
$$v_k = \frac{b-a}{F_{m+2}} \cdot F_{m-k+1} \quad \text{Gl. (3)}$$

Damit ist die Grundlage des auf der nächsten Seite als Flussdiagramm dargestellten Algorithmus für die eindimensionale Fibonacci-Suche gegeben.

Er ist so einfach, so dass die Erweiterung auf höhere Dimensionen überhaupt kein Problem darstellt. Z.B. bei einer 2-dimensionalen Suche $y=f(x_1, x_2)$ werden die Schleifen ineinander verschachtelt. Die äußere Schleife geht zum 1. Suchpunkt von x_1 , danach vollzieht die innere Schleife alle Suchschritte für x_2 . Dann geht die äußere Schleife zum Suchpunkt 2 für x_1 , die innere Schleife geht wiederum alle Suchpunkte für x_2 durch, usw. Natürlich kann für jede Dimension eine unterschiedliche Anzahl von Suchschritten gewählt werden.

Einfache Minimumsuche mit Fibonaccizahlen

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



4. Fibonacci-Suche bei diskreten Funktionen

Liegt keine Funktion $y=f(x)$ vor, sondern gibt es y nur in Form einer endlichen Zahlenfolge wie sie beispielsweise in einem Array abgespeichert werden kann, dann muss der beschriebene Algorithmus etwas modifiziert werden.

Nehmen wir an in einem Array **Ay** der Länge l ist die Zahlenfolge, bestehend aus l diskreten Werten, abgespeichert, von der das Minimum oder Maximum bestimmt werden soll.

Es ist sofort einzusehen, dass in diesem Fall nur ganzzahlige Schrittweiten auftreten dürfen.

Aus Gl.(3) ergibt sich, dass das zutrifft, wenn die Intervalllänge $(b-a) = F_{m+2}$ ist.

Dabei sind aber folgende 3 Besonderheiten zu berücksichtigen:

1. Die Länge l des Arrays **Ay** ist nicht die Differenz zwischen dem letzten und ersten Index, sondern sie ist um 1 größer.
2. Aus Gl.(2) ergibt sich für $(b-a) = F_{m+2}$ ein Unsicherheitsintervall von 1.
3. Bei der Fibonacci-Suche werden die Funktionswerte an den Intervallenden nicht berücksichtigt. Bei der diskreten Suche ist das aber notwendig.

Damit wird

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

Für die optimale Berechnung des Minimums/Maximums einer solchen Folge ergeben sich jetzt folgende Regeln:

1. Die Länge des Arrays bzw. die Anzahl diskreter Werte plus 1 muss eine Fibonacci-Zahl sein.
2. Nach der Bestimmung des Indexes m der Fibonacci-Zahl kann dann mit F_m Schritten das Minimum/Maximum bestimmt werden.
3. Damit alle Wert berücksichtigt werden, wird am Anfang ein Dummy-Wert hinzugefügt. Der Wert ist unwichtig, da er nur als Startpunkt benötigt wird und nicht in die Rechnung eingeht. Wird kein Dummy hinzugefügt, muss die Länge des 1. Suchschrittes um 1 reduziert werden.

Beispiel

Es liegt folgende Menge einer Zahlenfolge vor: $\{2,3,5,6,8,9,11,13,15,17,19,18\}$

Es sind 12 Werte. Mit 1 addiert ergibt 13 und das ist die Fibonacci-Zahl F_7 . Daraus folgt die Anzahl der Suchschritte zu $m=7-2=5$.

Mit dem hinzugefügten Dummy entsteht die Zahlenfolge $\{0,2,3,5,6,8,9,11,13,15,17,19,18\}$

Die Reihenfolge der Suchlängen ist: F_5, F_4, F_3, F_2, F_1 (5,3,2,1,1).

Bei der Minimumsuche würden der Reihe nach folgende Zahlen der obigen Folge abgefragt:

8->13->5->3->2 Daraus folgt der minimale Wert zu 2.

Bei der Maximumsuche wäre es folgende Reihenfolge:

8->13->17->19->18. Daraus folgt der maximale Wert zu 19.

5. Programm für die n-dimensionale Fibonacci-Suche

Ein Python-Script, welches das Minimum einer Funktion $y=f(x_1, \dots, x_n) = f(x)$ sucht, ist auf den folgenden Seiten zu finden. Die Anzahl der unabhängigen Variablen ist beliebig ($n \geq 1$). Das wurde erreicht durch rekursives Aufrufen der Suchfunktion. Die unabhängigen Variablen sind die Komponenten des Vektors x , also $x[0], \dots, x[n-1]$.

Grundlage des Programms ist der leicht modifizierte Algorithmus für die eindimensionale Suche (s. Seite 3).

Für die korrekte Funktion des Programms sind folgende Eingaben erforderlich:

1. In das Array ‚m‘ werden der Reihe nach für die n Variablen $x[0], \dots, x[n-1]$ die Anzahl der geforderten Suchschritte eingetragen.
2. In das Array ‚intervall‘ werden der Reihe nach für die n Variablen $x[0], \dots, x[n-1]$ die Suchintervallgrenzen eingetragen.
3. Bei der Funktionsdefinition `def FUNKTION(x)`: erfolgt die Berechnung des Funktionswertes $y=f(x)$ und mit `return` dessen Rückgabe.

Um das Maximum einer Funktion $y = f(x)$ mit diesem Programm zu bestimmen, gibt es 2 Möglichkeiten:

1. In allen Fällen, in denen y_2 und y_1 verglichen werden, den Vergleichsoperator ‚<‘ durch ‚>‘ zu ersetzen.

Oder noch einfacher

2. als Funktion $y = -f(x)$ einsetzen.

Das Programm (`fibonacci_suche.py`) kann problemlos in jede andere Programmiersprache portiert werden.

Es kann einfach über

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

geöffnet, kopiert und abgespeichert werden.

Ein- und mehrdimensionale Fibonacci Suche

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

# Programm fuer eine n-dimensionale Suche des Minimums einer Funktion
# mit Hilfe der Fibonacci-Zahlen.

# Das Programm ist ein python-script(Version 2.5.4), kann aber voellig problemlos
# in jede andere Programmiersprache portiert werden.

#####
# Version 1.1
# Datum: 08.12.2009
# Autor: Klaus-Eckart Schulz / Berlin
# Das Programm kann beliebig genutzt, modifiziert und weitergegeben werden,
# jedoch muss der Hinweis auf den Autor erhalten bleiben.
# Fuer jegliche Anwendung uebernimmt der Autor keinerlei Haftung.
#####

# Beispiel fuer n=5-dimensional

#----- Dateneingabe -----
m=[12,11,20,15,22] # Anzahl der Suchschritte fuer jede der in diesem Fall n=5 Variablen
intervall=[
    [0.5,1.0], # Suchintervalle fuer die n=5 Variablen
    [-2,-3],
    [16,40.0],
    [20.0,22.0],
    [-50.0,-55.0]
]

# Definition der Funktion  $y=f(x_1,\dots,x_n)=f(\underline{x})$ 
def FUNKTION(x):
    return (x[0]-0.5)**4+(x[1]+3)**2+(x[2]-16.0)**2+(x[3]-22.0)**2+(x[4]+50.8)**2
#-----

#===== eigentliches Programm fuer die n-dimensionale Fibonacci-Suche ====
def fibo_suche(dim,k,x0,x1,y1,x2,y2,s,m,l_fib,F):
    for k[dim] in range(0,m[dim]):
        if k[dim]==0:
            x1[dim]=x0[dim]+F[m[dim]-k[dim]]*l_fib[dim]
            x2[dim]=x1[dim]
            y1[dim]=FUNKTION(x1)
            if dim<len(m)-1:
                s[dim+1]=1.0
                fibo_suche(dim+1,k,x0,x1,y1,x2,y2,s,m,l_fib,F)
            y1[dim]=y2[dim]
        else:
            if k[dim]==m[dim]-1:
                x2[dim]=x1[dim]+s[dim]*F[m[dim]-k[dim]]*l_fib[dim]
                y2[dim]=FUNKTION(x2)
                if dim<len(m)-1:
                    s[dim+1]=1.0
                    fibo_suche(dim+1,k,x0,x1,y1,x2,y2,s,m,l_fib,F)
                if y2[dim]<y1[dim]:
                    x1[dim]=x2[dim]
                    y1[dim]=y2[dim]
                if dim>0:
                    y2[dim-1]=y1[dim]
            else:
                x2[dim]=x1[dim]+s[dim]*F[m[dim]-k[dim]]*l_fib[dim]
                y2[dim]=FUNKTION(x2)
                if dim<len(m)-1:
                    s[dim+1]=1.0
                    fibo_suche(dim+1,k,x0,x1,y1,x2,y2,s,m,l_fib,F)
                if y2[dim]<y1[dim]:
                    x1[dim]=x2[dim]
                    y1[dim]=y2[dim]
                else:
                    s[dim]=s[dim]*(-1)
```

```
#+++++ Initialisierung der Variablen ++++++

# Bestimmung des Fibonaccizahlen-Arrays '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]

# Bestimmung von 'Intervalllaenge/Fib[m+2]' fuer alle Dimensionen
l_fib=(1.0*len(m))
for i in range(0,len(m)):
    l_fib[i]=(intervall[i][1]-intervall[i][0])/F[m[i]+2]

# Arrays 's' fuer die Suchrichtung und 'k' fuer die Laufvariablen
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]

# Programmaufruf
fibosuche(0,k,x0,x1,y1,x2,y2,s,m,l_fib,F)

# Ergebnis-Ausgabe
print x1          # n-dimensionales Array mit den n Loesungen
print y1[0]      # dazu gehoerender Funktionswert
```