

Möglichkeiten zur Punkt auswertung

① rekursiv:

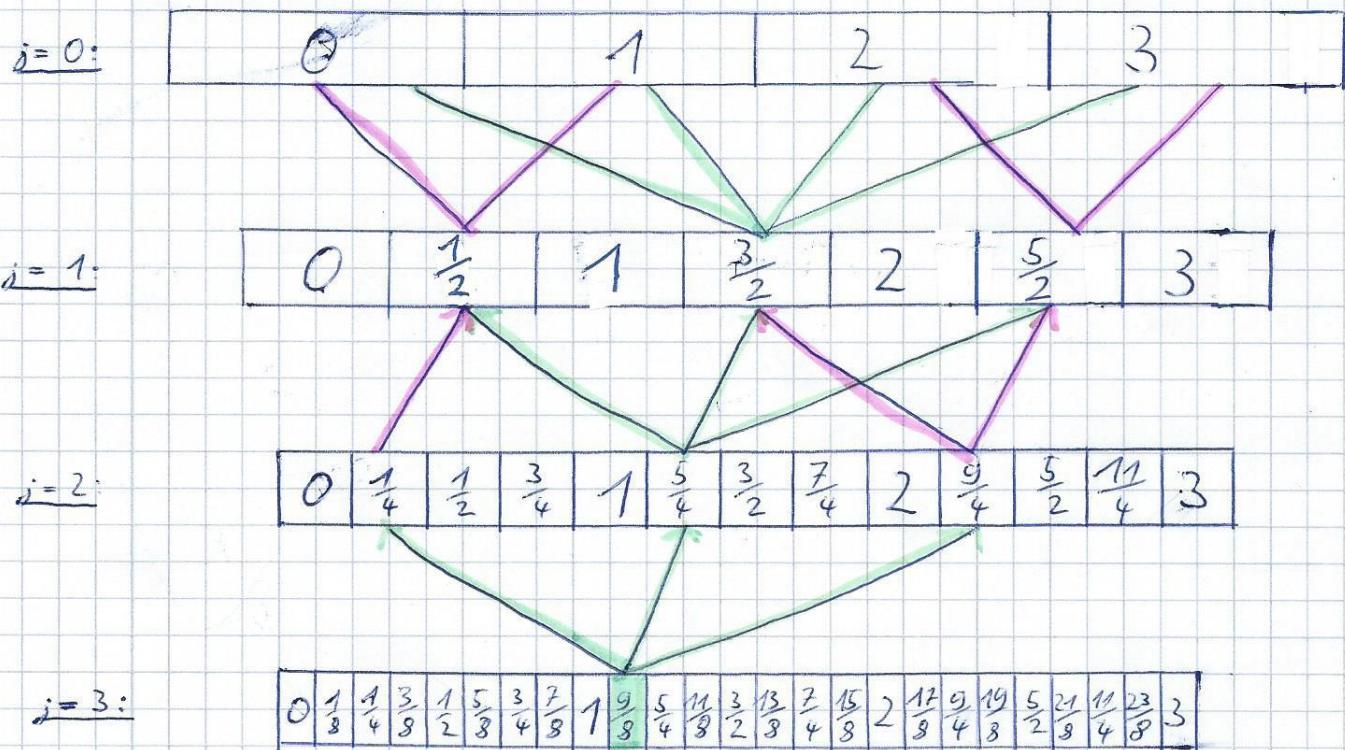
11 „naiv rekursiv“:

Vorgehen: implementiere die Formel $\varphi(2^{-j}l) = \sum_{k=0}^N a_k \varphi(2^{-j}(l-2^{j-1}k))$

mit Abbruchkriterium: wenn $2^{-j}l$ ganzzahlig ist wähle die aus dem LGS resultierende Werte.

Problem: zu hoher Rechenaufwand, denn selbst um nur einen Wert zu berechnen, berechnen wir vorherige Werte doppelt.

Beispiel:



(Die Farben dienen nur der besseren Übersichtlichkeit.)

Wir wollen in dem Beispiel $\varphi(\frac{9}{8})$ rekursiv berechnen.

Dafür benötigt der PC zunächst $\varphi(\frac{1}{4})$, $\varphi(\frac{5}{4})$ und $\varphi(\frac{9}{4})$.

Für $\varphi(\frac{1}{4})$ benötigt er: $\varphi(\frac{1}{2})$

Für $\varphi(\frac{5}{4})$ benötigt er: $\varphi(\frac{1}{2})$, $\varphi(\frac{3}{2})$, $\varphi(\frac{5}{2})$

Für $\varphi(\frac{9}{4})$ benötigt er: $\varphi(\frac{1}{2})$, $\varphi(\frac{5}{2})$

$j=0$

0	1	2	3
---	---	---	---

$j=1$

0	$\frac{1}{2}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$	3
---	---------------	---	---------------	---	---------------	---

$j=2$

0	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{3}{4}$	1	$\frac{5}{4}$	$\frac{3}{2}$	$\frac{7}{4}$	2	$\frac{9}{4}$	$\frac{5}{2}$	$\frac{11}{4}$	3
---	---------------	---------------	---------------	---	---------------	---------------	---------------	---	---------------	---------------	----------------	---

$j=3$

0	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{5}{8}$	$\frac{3}{4}$	$\frac{7}{8}$	1	$\frac{9}{8}$	$\frac{5}{4}$	$\frac{11}{8}$	$\frac{3}{2}$	$\frac{13}{8}$	$\frac{7}{4}$	$\frac{15}{8}$	2	$\frac{17}{8}$	$\frac{11}{4}$	$\frac{21}{8}$	$\frac{13}{2}$	$\frac{23}{8}$	$\frac{9}{4}$	$\frac{19}{8}$	$\frac{5}{2}$	$\frac{27}{8}$	$\frac{11}{4}$	$\frac{23}{8}$	3
---	---------------	---------------	---------------	---------------	---------------	---------------	---------------	---	---------------	---------------	----------------	---------------	----------------	---------------	----------------	---	----------------	----------------	----------------	----------------	----------------	---------------	----------------	---------------	----------------	----------------	----------------	---

$j=4$

$\frac{17}{16}$

Für $\varphi\left(\frac{17}{16}\right)$ berechnet er zunächst $\varphi\left(\frac{1}{8}\right), \varphi\left(\frac{9}{8}\right), \varphi\left(\frac{17}{8}\right)$

Für $\varphi\left(\frac{1}{8}\right)$ benötigt er: $\varphi\left(\frac{1}{4}\right)$

Für $\varphi\left(\frac{9}{8}\right)$ benötigt er: $\varphi\left(\frac{1}{4}\right), \varphi\left(\frac{5}{4}\right), \varphi\left(\frac{9}{4}\right)$

Für $\varphi\left(\frac{17}{8}\right)$ benötigt er: $\varphi\left(\frac{5}{4}\right), \varphi\left(\frac{9}{4}\right)$

2 mal je $\varphi\left(\frac{1}{4}\right), \varphi\left(\frac{5}{4}\right), \varphi\left(\frac{9}{4}\right) \rightarrow$ denn er ruft für $\varphi\left(\frac{1}{4}\right), \varphi\left(\frac{5}{4}\right), \varphi\left(\frac{9}{4}\right)$
 \Rightarrow 4 mal je $\varphi\left(\frac{1}{2}\right), \varphi\left(\frac{3}{2}\right), \varphi\left(\frac{5}{2}\right) \leftarrow$ je zweimal $\varphi\left(\frac{1}{2}, \varphi\left(\frac{3}{2}\right), \varphi\left(\frac{5}{2}\right)$ auf!
 (vgl. vorheriges Beispiel)

Anzahl der höchstens notwendigen Funktionsaufrufe für $\varphi\left(\frac{a}{2^j}\right)$:

Wir benötigen maximal 3 Funktionsaufrufe von $\varphi\left(\frac{x}{2^{j-1}}\right)$.

$$\text{denn } \varphi(2^{-j}l) = \sum_{k=0}^N a_k \varphi(2^{1-j}(l - 2^{j-1}k))$$

$$\begin{aligned} & \Rightarrow [2^{1-j}(l - 2^{j-1}k)] - [2^{1-j}(l - 2^{j-1}(k+1))] \\ & = -2^{1-j} \cdot (2^{j-1}k) + 2^{1-j} \cdot 2^{j-1}(k+1) \\ & = 1 \end{aligned}$$

Wir brauchen also z.B. $\varphi\left(\frac{1}{3}\right), \varphi\left(\frac{1+1}{3}\right), \varphi\left(\frac{1+2}{3}\right)$
der Zustand ist 1.

Ist die Trägerbreite 3 so benötigen wir also maximal 3 Aufrufe je Ebene. Außerdem es sind $\varphi(0)$ und $\varphi(3)$ mit einbezogen, \rightarrow dann 4 Aufrufe
 $\Rightarrow (3 \cdot j) + 1$ Aufrufe (eig. $3 \cdot (j-1) + 4$)

Anzahl der Funktionsaufrufe der rekursiven Variante:

Berechnungen in der Ebene j : 1 Funktions aufruf
 i.d.R. Trägerbreite.

" $j-1$: 3 Funktions aufrufe

$j-2$: 2 · 3 Funktions aufrufe

$j-3$: 2 · 2 · 3 Funktionsaufrufe

⋮
 1 : $2^{j-2} \cdot 3$ Funktionsaufrufe

0 : $2^{j-2} \cdot 3 \cdot 2 \cdot 4$ Funktionsaufrufe

so oft wurde jede Funktion bereits vorher aufgerufen!
 jetzt 4 Funktionen die aufgerufen werden: $\varphi(0), \varphi(1), \varphi(2), \varphi(3)$

$$\begin{aligned} \text{Insgesamt: } & 1 + 3 + 2 \cdot 3 + 2^2 \cdot 3 + \dots + 2^{j-2} \cdot 3 + 2^{j-1} (3+1) \\ & = 2^0 \cdot 3 + 2^1 \cdot 3 + 2^2 \cdot 3 + \dots + 2^{j-2} \cdot 3 + 2^{j-1} \cdot 3 + 2^{j-1} + 1 \\ & = 3 \cdot \sum_{k=0}^{j-1} 2^k + 2^{j-1} + 1 = 3 \cdot \left(\frac{2^j - 1}{2 - 1} \right) + 2^j + 1 \\ & = 4 \cdot 2^j - 2 \quad \text{i.d.R. Trägerbreite -1} \\ & \quad \text{i.d.R. Trägerbreite +1} \end{aligned}$$

Also rekursiv $(4 \cdot 2^j - 2)$ statt $(3 \cdot j) + 1$ Aufrufe!

Code zur „naiven rekursiven“ Variante:

Bekannt seien die Koeffizienten a_k (daraus erhalten wir die Trägerbreite, durch Anzahl der Koeffizienten - 1).

Sowie die Lösung an den ganzzahligen Stellen (Vorerst durch LGS berechnet)

Wir schreiben eine Funktion $\phi(2^j \cdot l)$ die $\varphi(2^j \cdot l)$ zurückgibt.

- Da $[0, \text{trägerB}]$ der kompakte Träger ist gibt sie für $2^j \cdot l \leq 0$ und $2^j \cdot l \geq \text{trägerB}$ 0 zurück.
- An den ganzzahligen Werten gibt sie die ganzzahl Werte zurück.
- Sonst ruft sie sich selbst rekursiv auf.

Code:

```
var ak
var width = ak.length - 1 // Trägerbreite
var ganzzahlWerte
function phi(j, l){ // gibt  $\varphi(2^j \cdot l)$  zurück
    if (l == 0) return 0; //
    else if (2^j * l >= width) return 0;
    else if (j == 0) return ganzzahlWerte[l];
    else {
        var sum = 0
        for (var k = 0; k < ak.length; k++){
            sum += ak[k] * phi(j-1, l - 2^{j-1} * k)
        }
        return sum;
    }
}
```

Kurzer Nachtrag: l muss ungerade sein, sonst muss $\frac{l}{2^j}$ gekürzt werden!

- Folgendes könnte man einfügen um den Bruch zu kürzen:
- ```
while (l % 2 == 0 && j > 0){
 l = l / 2;
 j = j - 1;
```

## 1.2. rekursiv mit Speicher:

Vorgehen: Implementiere die Rekursion wie zuvor, speichere jedoch alle berechneten Werte, bzw. prüfe ob sie bereits berechnet wurden.

Dazu folgende Festlegungen:

Fordern wir eine Genauigkeit von  $\frac{1}{2^N}$

bzw. berechnen wir  $\varphi(2^N \cdot l)$  (wobei  $l$  ungerade sein muss)

so legen wir ein Array der Länge  $(\text{Trägerbreite} \cdot 2^N) + 1$  an.

Beispiel: Daubachie-2-Wavelet (Trägerbreite=3) Genauigkeit:  $\frac{1}{4} = \frac{1}{2^2}$

$$\Rightarrow 3 \cdot 2^2 + 1 = 13 \text{ Felder}$$

|        |   |               |               |               |   |               |               |               |   |               |               |                |    |
|--------|---|---------------|---------------|---------------|---|---------------|---------------|---------------|---|---------------|---------------|----------------|----|
| x-Wert | 0 | $\frac{1}{4}$ | $\frac{1}{2}$ | $\frac{3}{4}$ | 1 | $\frac{5}{4}$ | $\frac{3}{2}$ | $\frac{7}{4}$ | 2 | $\frac{9}{4}$ | $\frac{5}{2}$ | $\frac{11}{4}$ | 3  |
| Index  | 0 | 1             | 2             | 3             | 4 | 5             | 6             | 7             | 8 | 9             | 10            | 11             | 12 |

Zwischen einem Index und dem x-Wert besteht offensichtlich der Zusammenhang

$$\boxed{\begin{aligned} 2^N \cdot x &= \text{index} \\ (\text{"step"}) & \\ \text{bzw. } x &= \text{index} \cdot 2^{-N} \end{aligned}}$$

Code: Wir modifizieren unseren vorherigen Code folgendermaßen:

- 1) Außerhalb der Funktion erstellen wir ein Array der Länge  $(\text{Trägerbreite} \cdot 2^N) + 1$  (zum Speichern der berechneten Werte)  
(Direkt mit 2 Spalten zum speichern der x und Funktionswerte)
- 2) Außerhalb der Funktion berechnen wir  $2^N$ , da wir das ständig benötigen. Und Innen  $x = 2^j \cdot l$ , sowie  $\text{index} = 2^j \cdot x$ ,
- 3) Innerhalb der Funktion fragen wir ab, ob der Wert bereits berechnet wurde und geben diesen ggf. zurück
- 4) Wir schreiben die berechneten Werte in das Array

Code:

```
var ganzzahlWerte;
var width = ak.length - 1
var step = 2^N
var values = createtray [step * width + 1, 2]
```

Nun tragen wir zunächst alle bekannten ganzzahligen Werte aus dem Array ganzzahlWerte an die entsprechenden Stellen im neuen Array values:

```
for (var i=0, i<width+1, i++) {
 values [i * step] [0] = i; // Der x-Wert ist 0, 1, ... also i
 values [i * step] [1] = ganzzahlWerte [i]; // Der Funktionswert steht bereits im Ganzzahl-Array
}

function phi [j, l] { // gibt $\phi(2^{-j} \cdot l)$ zurück
 (*)
 var x = $2^{-j} \cdot l$; // der wahre x-Wert
 var index = step * x; // dann $index = 2^N \cdot x = step \cdot x$
 if (x < 0) return 0;
 else if (x > width) return 0;
 else if (values [index] [0] != undefined) { // falls der Wert bereits im Array steht, geben wir diesen einfach zurück.
 return values [index] [1];
 }
 else {
 var sum = 0;
 for (var k=0, k<ak.length, k++) {
 sum += ak [k] * phi [j-1, l - $2^{j-1} \cdot k$];
 }
 values [index] [0] = x; // speichere den x-Wert
 values [index] [1] = sum; // und den zugehörigen Funktionswert
 }
}
```

(\*) auch hier kann es eventuell sinnvoll sein den Bruch vorher zu kürzen! (vgl. vorherige Version)

## (2) iterative Punkt auswertung:

Wie in den vorherigen Methoden gilt:

- die Koeffizienten  $a_k$  sind gegeben.
- die Trägerbreite (var width) beträgt  $a_k.length - 1$
- wir speichern  $2^N$  unter "step" ab.
- wir speichern x- und Funktionswerte in einem Array der Länge  $(step * width + 1)$
- Die Funktionswerte an den grenzähnlichen Stellen stehen zur Verfügung

Vorgehen:

- 1) Initialisiere alle oben genannten Variablen (vgl. vorherige Methode)
- 2) Übertrage die Funktionswerte von dem "ganzahl-Array" an die passende Stelle des neuen Arrays (vgl. vorherige Methode)
- 3) Nun berechnen wir die Werte an den Stellen  $\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \dots$

dann an den Stellen  $\frac{1}{4}, \frac{3}{4}, \frac{5}{4}, \dots$

dann an den Stellen  $\frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \dots$

↳ Wir haben also eine äußere Schleife für  $j$ , mit  $j = \frac{1}{2}, \dots, \frac{N}{2}, \dots, \frac{1}{2^N}$

und eine Innere für  $l = 1, 3, 5, \dots$   
solange wie  $2^{j-1} < width$  gilt ( $\rightarrow l < 2^j \cdot width$ )

Weiter steht wie zuvor fest:

Dies ist also der Index wo der neue Wert reingeschrieben wird

$$\text{index} = 2^N \cdot x = 2^N \cdot 2^{-j} \cdot l = 2^{N-j} \cdot l$$

Nun betrachten wir die Rekursionssumme etwas genauer:

$$f(2^{-j} \cdot l) = \sum_{k=0}^{\text{ak.length}} a_k f(2^{1-j} \underbrace{(l - 2^{j-1} \cdot k)}_{\in \mathbb{Z}})$$

Der erste Wert, der für die Summe benötigt wird ist

also  $f(2^{1-j} (l - 2^{j-1} \cdot 0))$  hier ist  $x_0 = 2^{1-j} (l - 2^{j-1} \cdot 0)$  Speicherindex von Oben

$$\text{also } \text{index}(x) = 2^N \cdot x = 2^N \cdot 2^{1-j} \cdot l = (2^{N-j} \cdot l) \cdot 2 = 2 \cdot \text{index}$$

Der Indexabstand zum nächsten benötigten Wert beträgt genau „step“, denn: sei  $x_k = 2^{i-j}(\ell - 2^{j-1} \cdot k)$  und  $x_{k+1} = 2^{i-j}(\ell - 2^{j-1} \cdot (k+1))$

$$\Rightarrow \text{Index}(x_k) = 2^N \cdot 2^{i-j}(\ell - 2^{j-1} \cdot k) \text{ und } \text{Index}(x_{k+1}) = 2^N \cdot 2^{i-j}(\ell - 2^{j-1} \cdot (k+1))$$

$$\Rightarrow \text{Index}(x_k) - \text{Index}(x_{k+1}) = 2^N = \text{„step“}$$

Wir können also so etwas schreiben wie:

var k=0;

var sum=0;

②

for ( k : , k < ak.length, k++ ) {

① sum += ak[k] \* value[index2][1]

index2 -= step }

value[index][1] = sum.

Es bleiben jedoch noch 2 Probleme

1) Es kann passieren, dass index2 negativ wird (wenn  $2^{j-1} \cdot k > \ell$  ist).

Dies bedeutet dass wir  $\varphi(x)$  für ein  $x < 0$  benötigen.

Diese sind aber immer 0.

Also können wir folgende Zeile einfügen

① if(index2 < 0) { break }

2) Es kann passieren, dass index2 größer als alle vorkommenden Indices ist.

Auch dies bedeutet, dass  $\varphi$  an der Stelle verschwindet.

Wir können index2 also solange verkleinern, bis wir innerhalb  $[0, \dots, \text{step} \cdot \text{width}]$  sind

Wichtig ist dass wir dabei nicht das k vergessen anzupassen. Eine Möglichkeit wäre:

② while(index2 > step · width) {

index2 -= step;  
k++;

Erste Tests haben ergeben:

| naiv-rekursiv           | rekursiv                                         | iterativ |
|-------------------------|--------------------------------------------------|----------|
| Laufzeit: zu hoch       | etwa halb so schnell wie iterativ                | schnell  |
| Vorteil: einfacher Code | schnelle Berechnung Falls weniger Werte benötigt | schnell  |