# Modellierung und Optimierung mit OPL

## 3 Techniken der binären Modellierung

Andreas Popp

# Inhalt








A. Popp



3.1 Abbildung logischer Ausdrücke

3.2 Entscheidungsabhängige Nebenbedingungen

Die Big-M-Methode

OPL: Modellieren von Zeitperioden

Disjunktive Nebenbedingungen

3.3 OPL: Kompakte Schreibweise

3.4 Stückweise definierte Funktionen

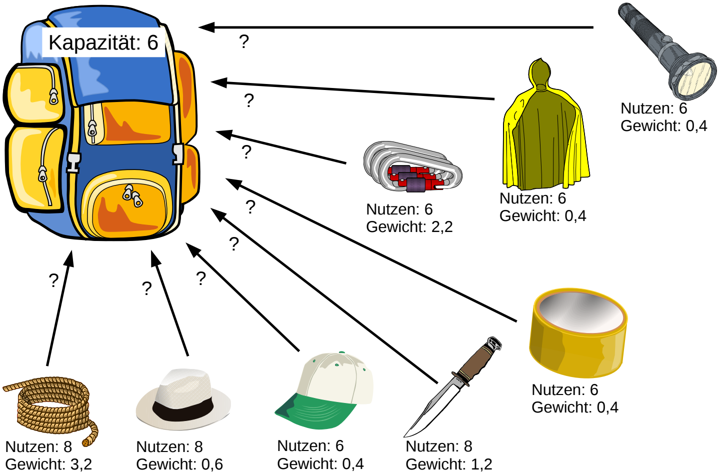
Treppenfunktionen

Stückweise lineare Funktionen

OPL: Der piecewise-Befehl

# 3.1 Abbildung logischer Ausdrücke

# Beispiel: Adventure Inc.



# Modell: Rucksackproblem

**Indexmengen**:

$I$ Menge der Gegenstände

**Parameter**:

$w_i$ Gewicht von Gegenstand $i \in I$

$u_i$ Nutzen von Gegenstand $i \in I$

$c$ Kapazität des Rucksacks

**Entscheidungsvariablen**:

$x_i$ Binäre Entscheidungsvariable; zeigt an ob Gegenstand $i \in I$ eingepackt wird

**Modellbeschreibung**:

$$\max \sum_{i \in I} u_i \cdot x_i$$

$$s.t. \sum_{i \in I} w_i \cdot x_i \leq c \qquad \text{(I)}$$

$$x_i \in \{0,1\} \qquad \forall i \in I$$

# Logische Verknüpfungen

- $\neg$ Die logische **Negation**
- $\wedge$ Das logische **Und**
- $\vee$ Das logische **Oder**
- $\veebar$ Das logische **exklusive Oder**(„Entweder-Oder“)
- $\Rightarrow$ Die logische **Implikation**
- $\Leftrightarrow$ Die logische **Äquivalenz**

## Wahrheitstafel in numerischer Beschreibung

| $A$ | $B$ | $\neg A$ | $\neg B$ | $A \wedge B$ | $A \vee B$ | $A \veebar B$ | $A \Rightarrow B$ | $A \Leftrightarrow B$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |

# Logische Verknüpfungen in binären Optimierungsmodellen

Beispiel: Seien $x_1$ und $x_2$ binäre Entscheidungsvariablen eines Rucksackproblems, die angeben, ob die Gegenstände $I_1$ und $I_2$ eingepackt werden.

$\neg I_1$: Man möchte wissen ob $I_1$ nicht eingepackt ist.

- $1 - x_1$

$I_1 \wedge I_2$: Sowohl $I_1$ als auch $I_2$ müssen eingepackt werden.

- $x_1 + x_2 = 2$

$I_1 \vee I_2$: Mindestens einer der Gegenstände muss eingepackt werden.

- $x_1 + x_2 \geq 1$

$\neg(I_1 \wedge I_2)$: Höchstens einer der Gegenstände darf eingepackt werden.

- $x_1 + x_2 \leq 1$

# Logische Verknüpfungen in binären Optimierungsmodellen

Beispiel: Seien $x_1$ und $x_2$ binäre Entscheidungsvariablen eines Rucksackproblems, die angeben, ob die Gegenstände $I_1$ und $I_2$ eingepackt werden.

$\neg(I_1 \vee I_2)$: Keiner der Gegenstände darf eingepackt werden.

- $x_1 + x_2 = 0$

$I_1 \veebar I_2$: Genau einer der Gegenstände muss eingepackt werden.

- $x_1 + x_2 = 1$

$I_1 \Rightarrow I_2$: Wenn $I_1$ eingepackt ist, muss auch $I_2$ eingepackt werden.

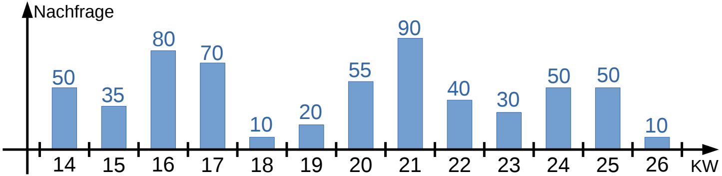
- $x_1 \leq x_2$

$I_1 \Leftrightarrow I_2$: Die Entscheidung für beide Gegenstände ist identisch.

- $x_1 = x_2$

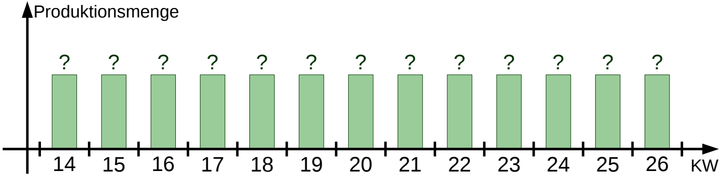
# 3.2 Entscheidungsabhängige Nebenbedingungen

# Beispiel: Lewig Wakuxi




Rüstkosten pro Periode: 25.000€

Lagerkosten von einer Periode auf die nächste pro Einheit: 100€



# Modell: Wagner-Whitin-Problem

**Indexmengen**:

| | |
|---|---|
| $T$ | Menge der Planungsperioden $\{t_{min}, \dots, t_{max}\}$ |

**Parameter**:

| | |
|---|---|
| $d_t$ | Nachfrage in Periode $t \in T$ |
| $s_t$ | Rüstkostensatz in Periode $t \in T$ |
| $h_t$ | Lagerkostensatz in Periode $t \in T$ |
| $i_{t_{min}-1}$ | Anfangsbestand |
| $M$ | Eine große Zahl |

**Entscheidungsvariablen**:

| | |
|---|---|
| $x_t$ | Produktionsmenge in Periode $t \in T$ |
| $i_t$ | Bestand am Ende von Periode $t \in T$ |
| $y_t$ | Produktionsentscheidung in Periode $t \in T$ |

**Modellbeschreibung**:

$$\min \sum_{t \in T} s_t \cdot y_t + h_t \cdot i_t$$

$$\begin{aligned} s.t. \quad & i_t = i_{t-1} + x_t - d_t && \forall t \in T && \text{(I)} \\ & x_t \leq M \cdot y_t && \forall t \in T && \text{(II)} \\ & x_t, i_t \geq 0;\ y_t \in \{0,1\} && \forall t \in T && \end{aligned}$$

# Die Big-M-Methode







Sei $\bar{\mathbf{x}}$ der Vektor der Entscheidungsvariablen und $f$ eine lineare Funktion. Die Nebenbedingung

$$f(\bar{\mathbf{x}}) \leq b \qquad \text{bzw.} \qquad f(\bar{\mathbf{x}}) \geq b$$

soll nur zwingend sein, wenn eine Entscheidung getroffen wurde, die dadurch repräsentiert wird, dass die Binärvariable $y$ den Wert 0 annimmt.

## Entscheidungsabhängige Nebenbedingungen

Sei $M$ eine ausreichend große Zahl.

$$f(\bar{\mathbf{x}}) \leq b \quad \rightarrow \quad f(\bar{\mathbf{x}}) \leq b + M \cdot y$$
$$f(\bar{\mathbf{x}}) \geq b \quad \rightarrow \quad f(\bar{\mathbf{x}}) \geq b - M \cdot y$$

# Problem bei der Implementierung von Zeitperioden

## Nebenbedingung aus Beispiel „Lewig Wakuxi“

$$i_t = i_{t-1} + x_t - d_t \qquad \forall t \in T \qquad \text{(I)}$$

## Implementierungsversuch 1

```
{string} T = {"KW14", "KW15", "KW16", "KW17"};
dvar float+ i[T];
  forall(t in T) i[t] == i[t-1] + x[t] - d[t];
```

Operator für string - int nicht verfügbar.

# Problem bei der Implementierung von Zeitperioden

## Nebenbedingung aus Beispiel „Lewig Wakuxi“

$$i_t = i_{t-1} + x_t - d_t \qquad \forall t \in T \qquad \text{(I)}$$

## Implementierungsversuch 2

```
{int} T = {14, 15, 16, 17};
dvar float+ i[T];
  forall(t in T) i[t] == i[t-1] + x[t] - d[t];
```

Der Index für den Array "i" liegt außerhalb des gültigen Bereichs: 13.






A. Popp

# Problem bei der Implementierung von Zeitperioden

## Nebenbedingung aus Beispiel „Lewig Wakuxi“

$$i_t = i_{t-1} + x_t - d_t \qquad \forall t \in T \qquad \text{(I)}$$

## Implementierungsversuch 3

```
{int} T = {14, 15, 16, 17};
{int} T0 = {13, 14, 15, 16, 17};
dvar float+ i[T0];
  forall(t in T) i[t] == i[t-1] + x[t] - d[t];
```

# Problem bei der Implementierung von Zeitperioden

## Nebenbedingung aus Beispiel „Lewig Wakuxi“

$$i_t = i_{t-1} + x_t - d_t \qquad \forall t \in T \qquad \text{(I)}$$

## Implementierungsversuch 4

```
int Tmin = 14;
int Tmax = 17;
range T = Tmin..Tmax;
dvar float+ i[Tmin-1..Tmax];
  forall(t in T) i[t] == i[t-1] + x[t] - d[t];
```






A. Popp

# Disjunktive Nebenbedingungen I

Ein Modell habe die beiden folgenden Nebenbedingungen:

$$f(\bar{\mathbf{x}}) \leq b$$
$$g(\bar{\mathbf{x}}) \leq d$$

Es reicht eine der beiden Bedingungen zu erfüllen.

## Disjunktive Nebenbedingungen

Sei $M$ eine ausreichend große Zahl und $y$ eine binäre Hilfsvariable.

$$f(\bar{\mathbf{x}}) \leq b + M \cdot y$$
$$g(\bar{\mathbf{x}}) \leq d + M \cdot (1 - y)$$

$\geq$-Nebenbedingungen analog

# Disjunktive Nebenbedingungen II

Ein Modell habe die folgende Nebenbedingungen:

$$g(\bar{\mathbf{x}}) \leq d$$

Diese muss nur erfüllt werden, wenn gilt:

$$f(\bar{\mathbf{x}}) > b$$

## Disjunktive Nebenbedingungen

Sei $M$ eine ausreichend große Zahl und $y$ eine binäre Hilfsvariable.

$$f(\bar{\mathbf{x}}) \leq b + M \cdot y$$
$$g(\bar{\mathbf{x}}) \leq d + M \cdot (1 - y)$$

$\geq$-Nebenbedingungen analog







A. Popp

# 3.3 OPL: Kompakte Schreibweise

# Entscheidungsausdrücke

Zielfunktion des Wagner-Whitin-Problems:

```
// Zielfunktion
minimize sum(t in T)(s[t]*y[t] + h[t]* i[t]);
```

Zerlegen in Entscheidungsausdrücke:

```
// Entscheidungsausdrücke
dexpr float setupCost = sum(t in T)(s[t]*y[t]);
dexpr float inventoryCost = sum(t in T)(h[t]*i[t]);

// Zielfunktion
minimize setupCost + inventoryCost;
```

# Arrays von Entscheidungsausdrücken

Zielfunktion des Wagner-Whitin-Problems:

```
// Zielfunktion
minimize sum(t in T)(s[t]*y[t] + h[t]* i[t]);
```

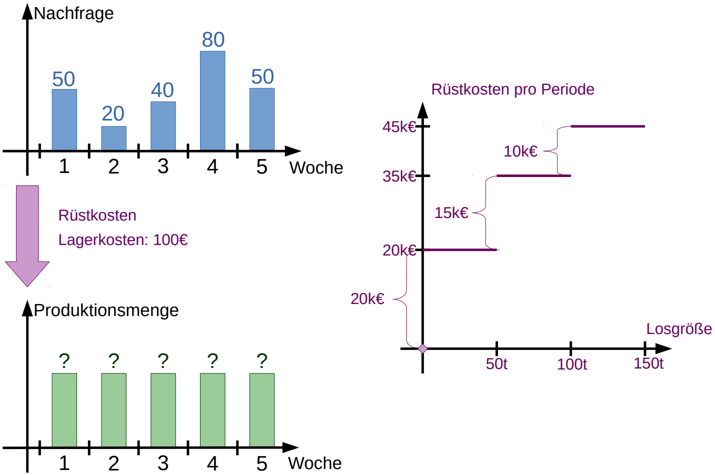
Zerlegen in Entscheidungsausdrücke:

```
//Entscheidungsausdrücke
dexpr float periodCost[t in T]
 = s[t]*y[t] + h[t]*i[t];

 // Zielfunktion
minimize sum (t in T)(periodCost[t]);
```

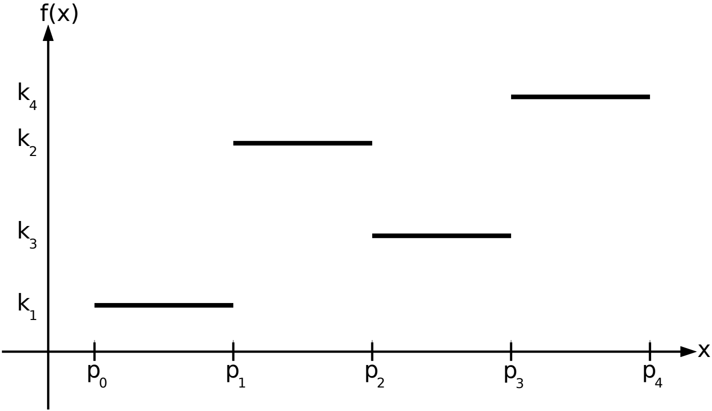
# 3.4 Stückweise definierte Funktionen

# Beispiel: Lewig Xanxi



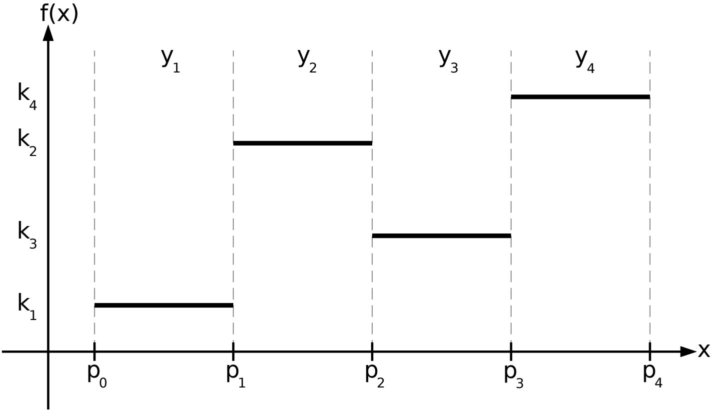
# Treppenfunktionen

Sei $x$ eine kontinuierliche Entscheidungsvariable:



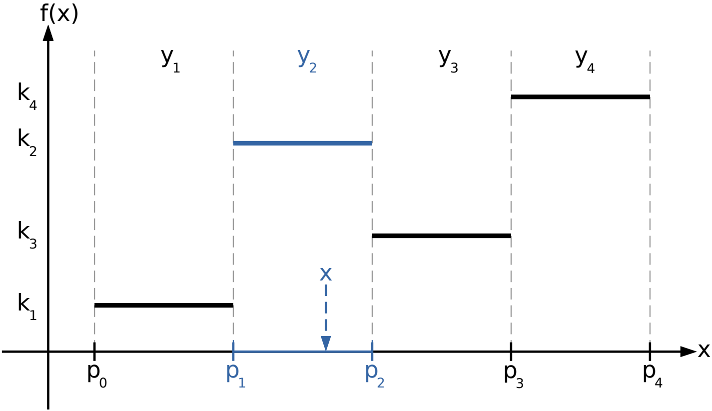
# Treppenfunktionen

Sei $x$ eine kontinuierliche Entscheidungsvariable:



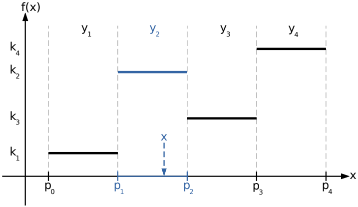
# Treppenfunktionen

Sei $x$ eine kontinuierliche Entscheidungsvariable:




- z.B.: $x = \frac{1}{3} \cdot p_1 + \frac{2}{3} \cdot p_2$

# Entscheidungsvariablen als Konvexkombination der Stützstellen

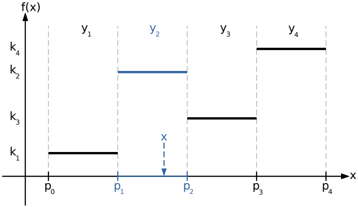



$$x = \sum_{n=0}^{N} z_n \cdot p_n$$

$$\sum_{n=0}^{N} z_n = 1$$

$$0 \leq z_n \leq 1 \quad \forall n \in \{1, \ldots, N\}$$

# Auswahl des korrekten Intervalls



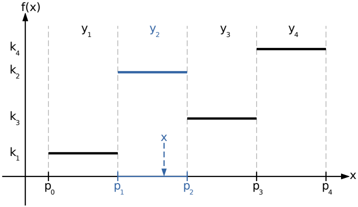

$$\sum_{n=1}^{N} y_n = 1$$

$$z_0 \leq y_1$$

$$z_n \leq y_n + y_{n+1} \quad \forall n \in \{1, \ldots, N-1\}$$

$$z_N \leq y_N$$

# Vollständige Modellierung




$$f(x) = \sum_{n=1}^{N} y_n \cdot k_n$$

$$x = \sum_{n=0}^{N} z_n \cdot p_n$$

$$\sum_{n=0}^{N} z_n = 1$$

$$0 \leq z_n \leq 1 \quad \forall n \in \{1, \dots, N\}$$

$$\sum_{n=1}^{N} y_n = 1$$

$$z_0 \leq y_1$$

$$z_n \leq y_n + y_{n+1} \quad \forall n \in \{1, \dots, N-1\}$$

$$z_N \leq y_N$$

# Beispiel: Lewig Tadelbach

Ausgangssituation ist ein Produktionsproblem.

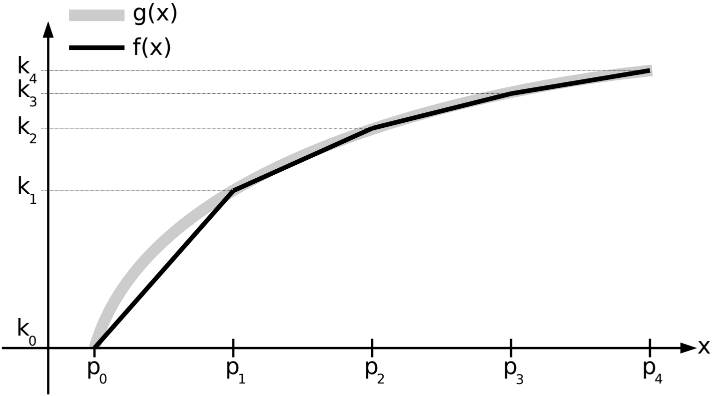
## Kapazitäten

| | Maschine $A$ | Maschine $B$ | Maschine $C$ |
|---|---|---|---|
| $l_1$ | 2,2 | 1,6 | 2,8 |
| $l_2$ | 1,2 | 1,9 | 2,3 |
| | 72 | 48 | 60 |

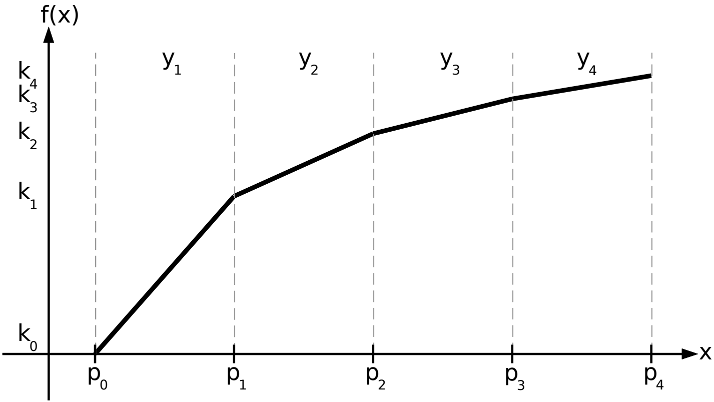
## Verkaufspreise

$$p_1(x_1) = 2000 \cdot \sqrt{x_1}$$
$$p_2(x_2) = 1800 \cdot \sqrt{x_2}$$

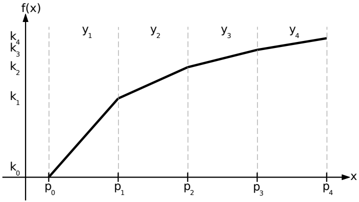
# Stückweise lineare Funktionen



# Stückweise lineare Funktionen



# Funktionswerte als Konvexkombination



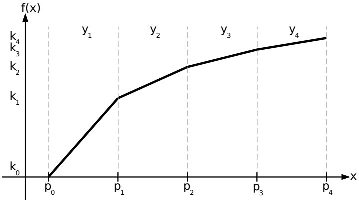

$$x = \sum_{n=0}^{N} z_n \cdot p_n$$

$$f(x) = \sum_{n=0}^{N} z_n \cdot f(p_n)$$

$$\sum_{n=0}^{N} z_n = 1$$

$$0 \leq z_n \leq 1 \quad \forall n \in \{1, \ldots, N\}$$

# Vollständige Modellierung




$$f(x) = \sum_{n=0}^{N} z_n \cdot k_n$$

$$x = \sum_{n=0}^{N} z_n \cdot p_n$$

$$\sum_{n=0}^{N} z_n = 1$$

$$0 \leq z_n \leq 1 \quad \forall n \in \{1, \ldots, N\}$$

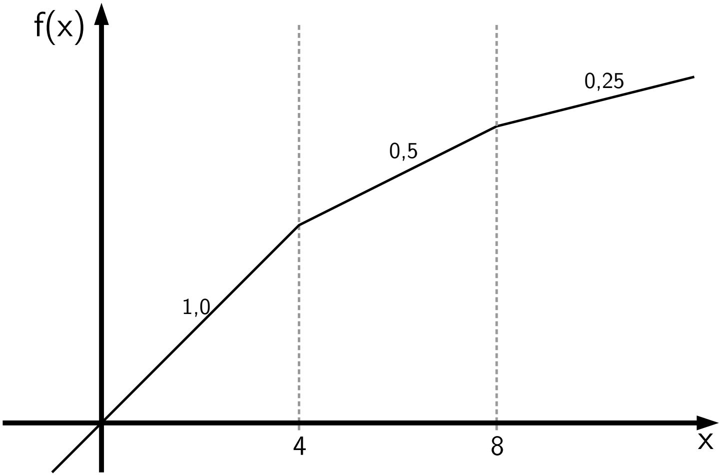
$$\sum_{n=1}^{N} y_n = 1$$

$$z_0 \leq y_1$$

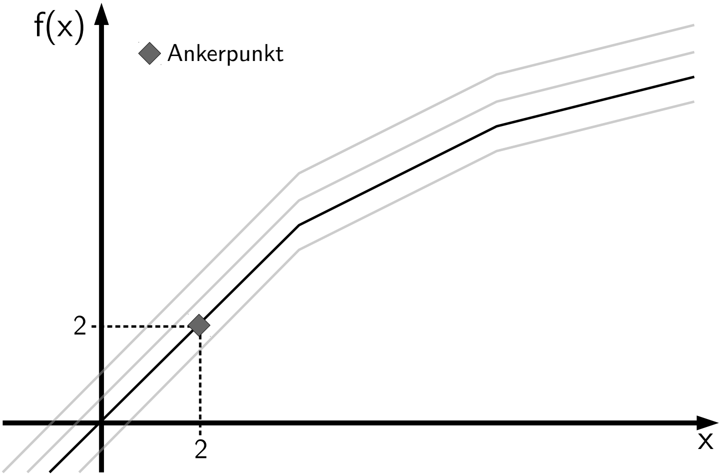
$$z_n \leq y_n + y_{n+1} \quad \forall n \in \{1, \ldots, N-1\}$$

$$z_N \leq y_N$$

# Stückweise lineare Funktionen nach Steigung



# Verankerung von stückweise linearen Funktionen



# Syntax des `piecewise`-Befehls

Array `p` der Stützstellen und Array `s` der Steigungen:

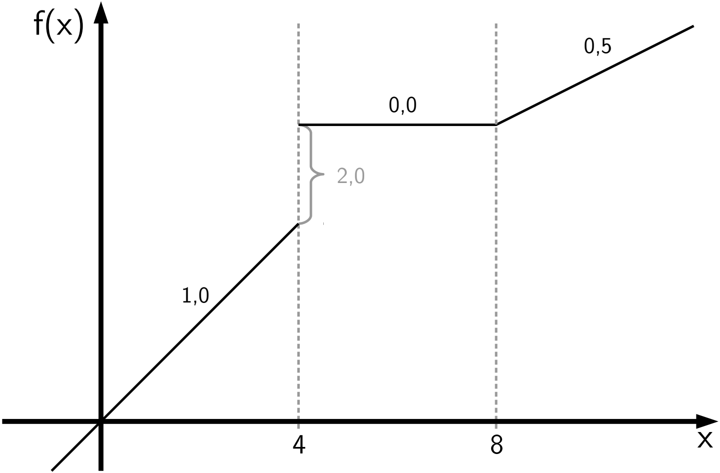
```
piecewise(i in 1..N){
  s[i] -> p[i];
  s[N+1]
} (Ankerpunkt) x;
```

## Beispiel aus obiger Abbildung

```
int N = 2;
float p[1..N] = [4, 8];
float s[1..N+1] = [1.0, 0.5, 0.25];
dvar float+ x;

piecewise(i in 1..N){
  s[i] -> p[i];
  s[N+1]
} (2, 2) x;
```

# Treppenfunktionen und allgemeine Unstetigkeiten



# Treppenfunktionen und allgemeine Unstetigkeiten







Zweiter Steigungswert an einem Punkt im `piecewise`-Befehl wird zu Sprungwert.

## Beispiel aus obiger Abbildung

```
int N = 3;
float p[1..N] = [4, 4, 8];
float s[1..N+1] = [1.0, 2.0, 0.0, 0.5];
dvar float+ x;

piecewise(i in 1..N){
  s[i] -> p[i];
  s[N+1]
} x;
```