

Minimalizace je algoritmus, který počítá \mathbf{x} : $F(\mathbf{x}) = \min \wedge C(\mathbf{x})$, kde \mathbf{x} je vektor hodnot parametrů, F je cenová funkce a C reprezentuje různá omezení (constraints). Problém: můžeme jednoduše spadnout do lokálního minima (\Rightarrow různé metody, jak se tomu vyhnout).

===== STOP =====

Analytická řešení: modelem je soustava rovnic, řešení je čistě matematické. V jistém směru je *dokonalé* – výsledky jsou obecné, přesné, je to efektivní. Problém: pro většinu modelů toto řešení neexistuje, neznáme jej nebo je extrémně obtížné jej najít.

Je špatné numericky řešit něco, co lze jednoduše popsat analyticky.

Postup analytického řešení: analýza problému, formulace matematického modelu, zjednodušení modelu (linearizace...), matematické řešení.

===== STOP =====

Markovské řetězce: popisuje náhodné procesy, které splňují vlastnost **memorylessness** (Markovova vlastnost): následující stav procesu (pravděpodobnost přechodu do jistého stavu) závisí pouze na jeho aktuálním stavu (ne na předchozích). *Velmi stochastická záležitost*.

Markovský řetězec = náhodný proces $X(t)$, který má Markovovu vlastnost. Je ekvivalentní konečnému automatu s pravděpodobnostmi přechodů.

SHO M/M/1 (příchody, doba obsluhy s exp. rozložením, 1 neomezená FIFO fronta) vyjádřené pomocí MC

- příchody: konstantní parametr $\lambda > 0$ (nezávisí na stavu modelu, čase)
- doba obsluhy: konstantní parametr $\mu > 0$

$\left\{ \begin{array}{l} \text{INTENZITA} \\ \text{PŘECHODŮ} \end{array} \right.$

pro M/M/1:

- pravděpodobnost stavu: $p_k = p^k p_0 = (\lambda/\mu)^k p_0$
- pravděpodobnost, že nebude čekat (že zařízení nepracuje): $p_0 = 1 - p$
- průměrná délka fronty: $L_w = p^2 / (1 - p)$
- průměrná doba čekání: $T_w = p / (\mu - \lambda)$
- průměrná doba strávená v systému: $T_s = T_w + T_o = T_w + 1/\mu$

geom. řady.

$$S = \frac{a}{1-q}$$

DOPLNIT PRO M/M/2

první délka fronty: sumu součinu (délka \cdot P) pro všechny možné délky:

$$L_w = \sum_{k=1}^{\infty} k \cdot \pi_{k+1} = \sum_{k=1}^{\infty} k \cdot p^{k+1} \cdot p_0 = \sum_{k=1}^{\infty} k \cdot p^{k+1} \cdot (1-p) = \boxed{\frac{p^2}{1-p}}$$

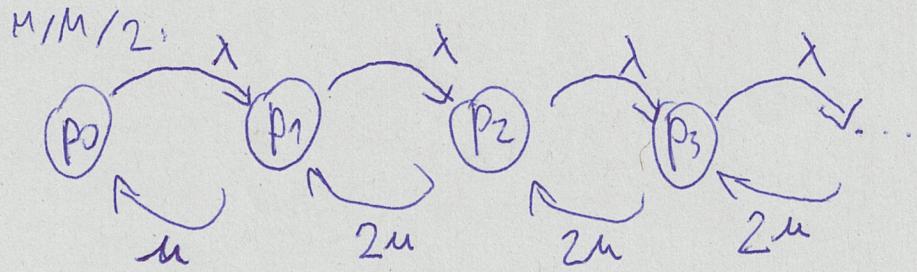
$$T_w = T_o \cdot N = T_o \cdot \sum_{k=1}^{\infty} k \cdot \pi_k = \frac{1}{\mu} \cdot \sum_{k=1}^{\infty} k \cdot p^k \cdot (1-p) = \boxed{\frac{p}{\mu-\lambda}}$$

\downarrow
Počet transakcí

$$T_s = T_w + T_o$$

$$\frac{1^x e^{-x}}{x!}$$

$$R e^{-\lambda(x-x_0)}$$



$$-\lambda p_0 + \mu p_1 = 0$$

$$-\lambda p_1 - \mu p_1 + \lambda p_0 + 2\mu p_2 = 0$$

$$-\lambda p_2 - 2\mu p_2 + \lambda p_1 + 2\mu p_3 = 0$$

$$P_k = \frac{1}{k!} \cdot \cancel{Q^k} \cdot p_0$$

M/M/2

Snelheid. Fronten

Křížené kombinované simulace

```
T-END = 20, STEP-MIN = 0,01, STEP-BASE = 0,1 // Konstanty  
step = STEP-BASE, t=0 // Inicializace model. času, kroků  
InitCalendar(), InitModel() // Inicializace kalendáře, modelu, skvělých (převěnných)  
while (t < T-END):  
    next-time = Min(T-END, Calendar.Peek()?.time ?? T-END) // Zjištění koncového času (když dálí ud. z kalendáře, nebo T-END)  
    while (t < next-time): // Provádění kroků spoj. simulace, dokud nedojde ke ud. z kalendáře (nebo konci sim.)  
        Save State() // Uloží aktuální čas t a stavové proměnné  
        if ((t + step * 1.01) > next-time): // Dokročení  
            RK4-Step(next-time - t) // Krok spoj. simulace (vyhodnocení stav. proměnných)  
            t = next-time // Posun času  
        else:  
            RK4-Step(step) // Vyhodnocení stav. podmínek  
            t += step // Polohu se změnila hodnota stav. podmínek, hledáme kdyz  
        new-state = StateCondition() // Vyhodnocení stav. podmínek  
        if (current-state != new-state): // Polohu se změnila hodnota stav. podmínek, hledáme kdyz  
            if (step <= STEP-MIN): // Potvrdí změnu podmíny  
                current-state = new-state // provede stav. událost  
                StateEvent() // Obnáší délku kroků  
                step = STEP-BASE  
            else:  
                RestoreState() // Obnoví založený čas a stav  
                step = step / 2 // hledání polohy v intervalu  
                if (step < STEP-MIN):  
                    step = STEP-MIN  
        next-event = Calendar.Pop() // provede naplánované diskr. události z kalendáře  
        if (next-event == NULL):  
            break  
        t = next-event.time  
        next-event.Behavior()  
    }
```

```
double step = 0.01, t = 0;  
double st[4] = {1, 2, 3, 4};  
while (t < T-MAX) {  
    if ((t + step * 1.01) > T-MAX) // Dokročení  
        step = T-MAX - t;  
    RK3(step, st); // Posun model. času  
    t += step;  
    printf("time: %f, y: %f\n", t, st[1]);  
}
```

Základní next-event algoritmus

```

T-START = 0, T-MAX = 20
Init()
while (!Calendar.Empty()):
    e = Calendar.Pop()           // Inicializace kalendáře modelu...
    if (e.time > T-MAX):
        break                     // Konec simulace
    Time = e.time                // Nastav modelový čas na čas záznamu
    e.Behaviour()               // Provedu popis chování záznamu
Time = T-MAX                      // Konec simulace - např. výpis výsledků, deaktivace...
End()

```

Rízení spojité simulace, Euler, RK (řízení na druhé straně)

```

void Dynamic (const double st[], double in[]) {
    in[0] = st[1] + 1;
    in[1] = st[0];
}

```

```

void Euler (double step, double st[]) {
    double in[2] = {0, 0};
    Dynamic(st, in);
    for (int i=0; i<2; i++)
        st[i] += step * in[i];
}

```

```

void RK3 (double step, double st[]) {
    double st_orig[2];
    for (int i=0; i<2; i++) st_orig[i] = st[i];
    double in[2], k1[2], k2[2];

```

```

Dynamic(st, in)                                //  $k_1 = f(t, y(t))$ 
for (int i=0; i<2; i++) {
    k1[i] = in[i];
    st[i] += step / 2 * k1[i];                  //  $st \leftarrow y(t) + \frac{h}{2} \cdot k_1$ 
}
Dynamic(st, in)                                //  $k_2 = f(t + \frac{h}{2}, y(t) + \frac{h}{2} \cdot k_1)$ 
for (int i=0; i<2; i++) {
    k2[i] = in[i];
    st[i] = st_orig[i] - step * k1[i] + 2 * step * k2[i]; //  $st \leftarrow y(t) - h \cdot k_1 + 2 \cdot h \cdot k_2$ 
}

```

```

Dynamic(st, in);                                //  $k_3$ 
for (int i=0; i<2; i++) {

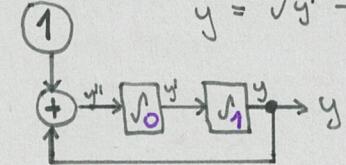
```

```

    st[i] = st_orig[i] + step * (k1[i] + 4 * k2[i] + in[i]) / 6; //  $y(t+h) = y(t) + h(k_1 + 4k_2 + k_3) / 6$ 
}

```

$$\begin{aligned}
 y'' - y &= 1 \\
 y'' &= 1 + y = 1 + f(y) \\
 y' &= \sqrt{y} \\
 y &= \sqrt{y}
 \end{aligned}$$



RK3:

$$\begin{aligned}
 y(t+h) &= y(t) + \frac{h}{6} (k_1 + 4k_2 + k_3) \\
 k_1 &= f(t, y(t)) \\
 k_2 &= f(t + \frac{h}{2}, y(t) + \frac{h}{2} \cdot k_1) \\
 k_3 &= f(t + h, y(t) - h \cdot k_1 + 2h \cdot k_2)
 \end{aligned}$$

Facility, store

Facility:

Priority Queue queue

Process current = NULL

int max Queue Len = INT_MAX

constructor (int max Queue Len = INT_MAX):

this.max Queue Len = max Queue Len

def Seize (Process p):

if (current == NULL):

 current = p

else if (queue.Len() < maxQueueLen):

 p.Passivate()

 queue.Enqueue(p)

else:

 throw Error()

def Release (Process p):

if (current != p):

 throw Error()

if (queue.Len() > 0):

 current = queue.Dequeue()

 current.Activate()

else:

 current = NULL

Kongruenti gen.

static uint32_t x_i = SEED;

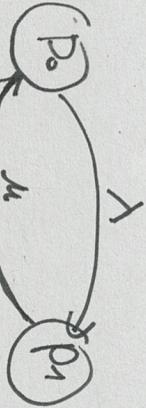
double Random (void) {

 x_i = x_i * 69069 + 1u; // implicit mod 2^{32}

 return x_i / ((double)(INT32_MAX)) ;

 + 1.0

}



$$\lambda = 15 \text{ (přích. za hod.)}$$

$\mu = 12$ (obsluh za hodinu) ~ jehož za 5 min

Zálež: M2 použit M/M/2

generátor - pseudorandom

Monte Carlo - 150 C v

projektovat & přidat

$$-\lambda p_0 + \mu p_1 = 0$$

$$p_0 = 1 - p_1$$

$$\frac{\cancel{\lambda p_0} + \cancel{\mu p_1}}{\cancel{\lambda p_0 + \mu p_1}} = 1$$

$$\frac{\cancel{\lambda p_0} - \cancel{\lambda p_1} + \cancel{\mu p_1}}{\cancel{\lambda p_0 + \mu p_1}} = 1$$

$$-\lambda(1-p_1) + \mu p_1 = 0$$

$$-\lambda + \lambda p_1 + \mu p_1 = 0$$

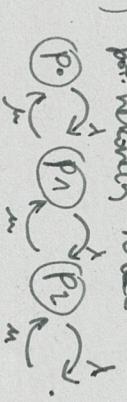
$$p_1 = \frac{\lambda}{\lambda + \mu}$$

$$\left. \begin{array}{l} \text{průměr } 3 / \text{min.} \\ \text{výd: } 15 \text{ s} \end{array} \right\} \frac{\lambda}{\lambda + \mu} / \text{min}$$

$$p_1 = \frac{15}{27} = \frac{5}{9} \Rightarrow \text{predp. obzervace záv. zóny}$$

- 1) p_1 je nějaký číslo?
- 2) avg. délka čekání
- 3) avg. doba čekání
- 4) avg. doba v systému

$$\boxed{p_0 = \frac{4}{9}}$$



$$\lambda = 3 \text{ příchod/min}$$

$$\mu = \frac{60}{15} = 4 \text{ obsluha/min.}$$

$\lambda < \mu \Rightarrow$ stabilní.

$$p_0 p_0: -\lambda p_0 + \mu p_1 = 0 \Rightarrow p_1 = \frac{\lambda}{\mu} \quad p_0 = \varphi p_0$$

$$p_1: \lambda p_0 - \lambda p_1 + \mu p_2 - \mu p_1 = 0 \Rightarrow p_2 = \varphi^2 p_0$$

$$p_2 = \varphi p \dots \quad p_k = \varphi^k \cdot p_0$$

$$p_0 + \varphi p_0 + \varphi^2 p_0 + \dots = \frac{p_0}{1-\varphi}$$

$$\frac{p_0}{1-\varphi} = 1 \Rightarrow$$

$$\boxed{p_0 = 1-\varphi}$$

P_1 je nejdůležitější

