Modelování a simulace

Petr Peringer
peringer AT fit.vutbr.cz
Martin Hrubý
hrubym AT fit.vutbr.cz

Vysoké učení technické v Brně,
Fakulta informačních technologií,
Božetěchova 2/1,
612 00 Brno

(Verze: 12. září 2024)

Úvod

Text je určen pro studenty FIT. Obsahuje základní přehled problematiky modelování a simulace vhodný pro studenty bakalářského studia. Předpokládají se základní znalosti programování (C, C++, ...) a matematiky (relace, derivace, integrály, dif. rovnice).

Obsah slajdů je velmi stručný, podrobnější informace jsou součástí výkladu.

Na slajdech spolupracovali:

- Martin Hrubý – Petriho sítě, náhodné procesy

Pravidla

- Přednášky
- Minimálně 2 demo-cvičení (+doplňky)
- Samostatná práce: projekt
- Konzultace

Hodnocení celkem 100b:

- 10b půlsemestrální test
- 20b projekt
- Zápočet: alespoň 10 z výše uvedených bodů
- 70b zkouška (požadováno min. 30 bodů)

Zdroje informací

- Literatura
- WWW odkazy
- Oficiální stránka:
  `https://www.fit.vut.cz/study/course/IMS/`
- Aktuální informace pro studenty:
  `https://www.fit.vut.cz/person/peringer/public/IMS/`
- Vlastní uvažování a (simulační) experimenty
- ...

Literatura

- Fishwick P.: *Simulation Model Design and Execution: Building Digital Worlds*, Prentice-Hall, 1995
- Law A., Kelton D.: *Simulation Modelling & Analysis*, second edition, McGraw-Hill, 1991
- Rábová a kol.: *Modelování a simulace*, skriptum VUT, Brno, 1992
- Ross S.: *Simulation*, 3rd edition, Academic Press, 2002
- (Zeigler B., Muzzy A., Kofman E.: *Theory of Modelling and Simulation*, 3rd edition, Academic Press, 2019)
- ...

Poznámky:
Studijní opora — viz IS
(Poznámka: Informace k zadání Bc práce — témata.)

Modelování systémů na počítačích

Přehled

- Základní pojmy a princip
- Souvislosti a aplikace
- Výhody a nevýhody
- Alternativy
- Úvod do teorie systémů
- Typy simulace
- Velmi stručný přehled simulačních nástrojů

Základní pojmy (systém, model)

Systém =

soubor elementárních částí (prvků systému), které mají mezi sebou určité vazby.

Rozlišujeme (mimo jiné)

- reálné systémy
- nereálné systémy (fiktivní, ještě neexistující)

Model =

napodobenina systému jiným systémem.

- Model = reprezentace znalostí.
- Klasifikace: fyzikální modely, matematické modely, ...
- Přírodní zákony jsou matematické modely (Příklad: Ohmův zákon $u = Ri$).

Základní pojmy (modelování, simulace)

Modelování =

vytváření modelů systémů.

- Modelování je velmi používaná metoda
- Modelovat lze jen to, co známe a umíme popsat

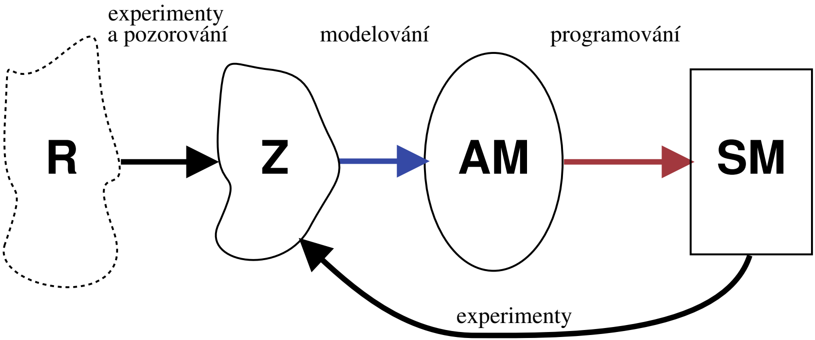
Simulace =

získávání nových znalostí o systému experimentováním s jeho modelem.

- Budeme se zabývat pouze simulací na číslicových počítačích.

Princip modelování a simulace

Realita → Znalosti → Abstraktní model → Simulační model



Cílem je získat nové znalosti o modelovaném systému.

Základní etapy modelování a simulace

1. *Vytvoření abstraktního modelu*: Formování zjednodušeného popisu zkoumaného systému.
2. *Vytvoření simulačního modelu*: Zápis abstraktního modelu formou programu.
3. *Verifikace a validace*: Ověřování správnosti modelu.
4. *Simulace*: Experimentování se *simulačním* modelem.
5. *Analýza a interpretace výsledků*: Získání nových znalostí o zkoumaném systému.

Souvislosti

- pozorování a reálné experimenty
- *Computational Science*
- učení, hry — ”co se stane když”
- programování (simulační model je program)
- algoritmy, datové struktury
- numerická matematika (integrační metody, ...)
- počítačová grafika (vizualizace výsledků)
- technické vybavení: superpočítače, ...
- teorie systémů (stabilita, citlivost, chaos, ...)
- pravděpodobnost a statistika
- + obory související s modelovaným systémem

Příklady použití simulace v praxi

- **Věda:** biologie, lékařství, ekologie, chemie, jaderná fyzika, astronomie, sociologie, ...
  (Např. předpověď počasí, zemětřesení, šíření epidemií, ...)
- **Technika:** strojírenství, stavebnictví, doprava, elektro, ...
  (Dynamika konstrukcí, simulace mikroprocesorů, optimalizace motoru, ...)
- **Ekonomika** (Vývoj cen na burze, ...)
- **Výuka** (Různé demonstrační modely)
- **Film** (Vizuální efekty všeho druhu)
- **Hry** (Simulátor letadla, ...)
- ...

Výhody simulačních metod

- Cena (např. ”crash” testy automobilů)
- Rychlost (růst rostlin, vznik krystalů, pohyb planet)
- Bezpečnost (jaderné reakce, šíření epidemií)
- ...
- Někdy jediný způsob (srážky galaxií)
- Možnost modelovat velmi složité systémy (mikroprocesory, různé biologické systémy, počasí)

Velmi často je výhodnější experimentovat s modely, než s originálními systémy.

Problémy simulačních metod

- Problém validity (platnosti) modelu.
- Někdy velmi vysoká náročnost vytváření modelů.
- Náročnost na výkon počítačů.
- Získáváme konkrétní numerické výsledky (například změna parametru vyžaduje celou simulaci opakovat).
- Nepřesnost numerického řešení.
- Problém stability numerických metod.

Alternativní přístup

Analytické řešení modelů

- Popis chování systému matematickými vztahy a jeho *matematické řešení*.
- Vhodné pro jednoduché systémy nebo zjednodušené popisy složitých systémů.
- Výsledky jsou ve formě funkčních vztahů, ve kterých se jako proměnné vyskytují parametry modelu.
- Dosazením konkrétních hodnot získáme řešení.

Shrnutí: Hlavní předností analytického řešení je přesnost a menší časová náročnost výpočtu řešení matematického modelu. Řešit ale umíme jen modely jednoduché nebo podstatně zjednodušené.

Příklad: Model volného pádu ve vakuu

Kdy použít simulační metody

Simulaci je vhodné použít když:

- neexistuje *úplná* matematická formulace problému nebo nejsou známé analytické metody řešení matematického modelu;
- analytické metody vyžadují tak zjednodušující předpoklady, že je nelze pro daný model přijmout;
- analytické metody jsou dostupné pouze teoreticky, jejich použití by bylo obtížné a simulační řešení je jednodušší;
- modelování na počítači je jedinou možností získání výsledků v důsledku obtížnosti provádění experimentů ve skutečném prostředí;
- potřebujeme měnit časové měřítko (simulace např. umožní vypočítat řešení rychleji než by proběhl příslušný děj v reálném systému).

Úvod do teorie systémů

(Systems Theory, Systems Science)

Přehled:

- Definice základních pojmů:
  - Systém
  - Prvek systému
  - Časová množina
  - Chování systému
  - Okolí systému
- Homomorfní a izomorfní systémy
- Klasifikace prvků systému a systémů

Formální definice systému

Systém S je dvojice

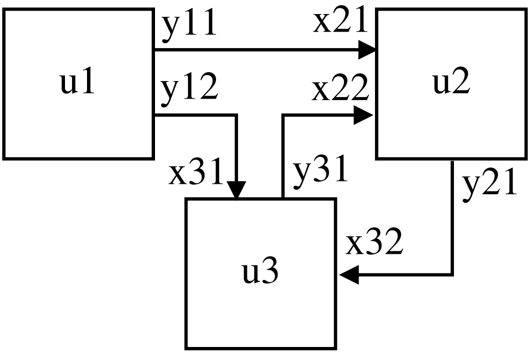
$$S = (U, R)$$

kde:

- Univerzum U je konečná množina prvků systému: $U = \{u_1, u_2, ..., u_N\}$
- Prvek systému: $u = (X, Y)$ kde
  - X je množina všech vstupních proměnných
  - Y je množina všech výstupních proměnných
- Charakteristika systému R je množina všech propojení:
  $$R = \bigcup_{i,j=1}^{N} R_{ij}$$
- Propojení prvku u_i s prvkem u_j: $R_{ij} \subseteq Y_i \times X_j$

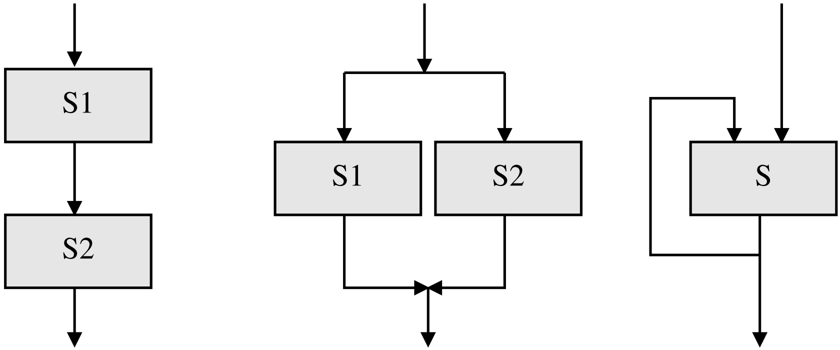
Příklad definice jednoduchého systému

$U = \{u_1, u_2, u_3\}$
$R = \{(y_{11}, x_{21}), (y_{12}, x_{31}), (y_{31}, x_{22}), (y_{21}, x_{32})\}$



Vazby mezi prvky systému

sériová, paralelní, zpětná vazba



Čas

Budeme rozlišovat tři základní pojmy:

Reálný čas ve kterém probíhá skutečný děj v reálném systému (viz fyzikální definice času).

Modelový čas je "časová osa" modelu (modeluje reálný čas ze vzorového systému — např. proměnná t v diferenciální rovnici $y'' = -g$). Při simulaci nemusí být synchronní s reálným časem.

Strojový čas je čas CPU spotřebovaný na výpočet programu (závisí na složitosti simulačního modelu, počtu procesorů a nesouvisí přímo s modelovým časem).

Poznámka: Příkaz `time` (POSIX)

Časová množina

(Time base)

T je množina všech časových okamžiků, ve kterých jsou definovány hodnoty vstupních, stavových a výstupních proměnných prvku systému.

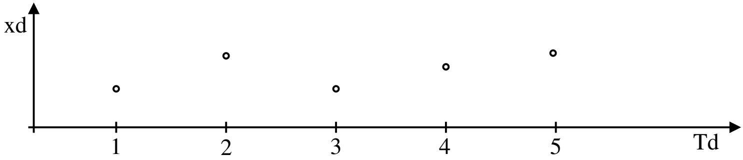
Příklady časových množin

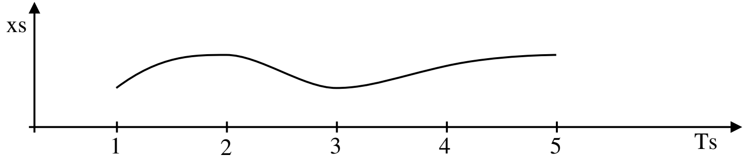
- diskrétní: $T_d = \{1, 2, 3, 4, 5\}$
- spojitá: $T_s = \langle 1.0, 5.0 \rangle \quad T_s \subset \mathbf{R}$

Poznámka:
Na číslicovém počítači se spojitá časová množina vždy diskretizuje.

Časová množina — příklady

Signály s diskrétní (T_d) a spojitou (T_s) časovou množinou:





Chování systému

- Každému časovému průběhu vstupních proměnných přiřazuje časový průběh výstupních proměnných.
- Je dáno vzájemnými interakcemi mezi prvky systému.

Chování systému S můžeme definovat jako zobrazení χ:

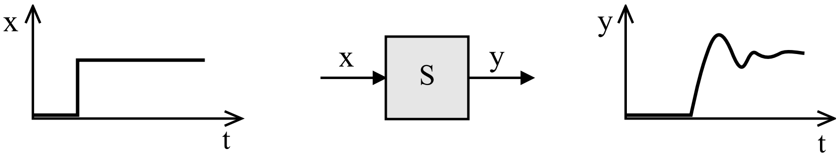
$$\chi : [\sigma_i(S)]^T \rightarrow [\sigma_o(S)]^T$$

kde:

- $[A]^T$ je množina všech zobrazení T do množiny A,
- $\sigma_i(S)$ je vstupním prostorem systému S,
- $\sigma_o(S)$ je výstupním prostorem systému S.

Chování systému — příklad

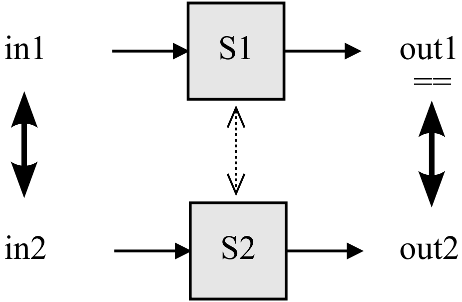
- Spojitý systém S, odezva na jednotkový skok:



Ekvivalence chování systémů

Systémy S_1 a S_2 považujeme za systémy se stejným chováním, vyvolají-li stejné podněty u obou systémů stejné reakce.

Stejnými podněty/reakcemi rozumíme ty dvojice podnětů/reakcí, které jsou spolu vzájemně přiřazeny definovaným vstupním/výstupním zobrazením.



Izomorfní systémy

Systémy $S_1 = (U_1, R_1)$ a $S_2 = (U_2, R_2)$ jsou izomorfní, když a jen když:

1. Prvky univerza U_1 lze vzájemně jednoznačně (1:1) přiřadit prvkům univerza U_2.
2. Prvky charakteristiky R_1 lze vzájemně jednoznačně přiřadit prvkům charakteristiky R_2, a to tak, že prvku charakteristiky R_1, vyjadřujícímu orientovaný vztah mezi dvěma prvky univerza U_1, je vždy přiřazen právě ten prvek charakteristiky R_2, který vyjadřuje stejně orientovaný vztah mezi odpovídající dvojicí prvků univerza U_2 a naopak.

Poznámka: Zjednodušeno (nezahrnuje chování).

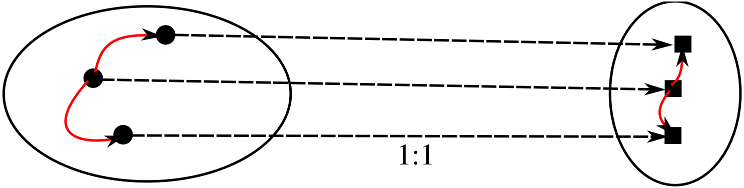
Homomorfní systémy

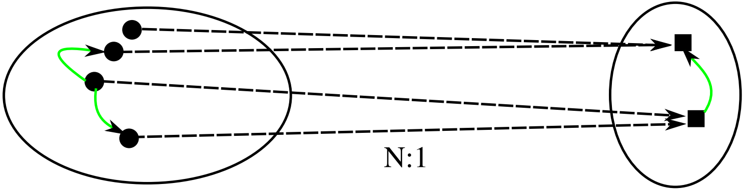
Systém $S_1 = (U_1, R_1)$ je homomorfní se systémem $S_2 = (U_2, R_2)$ právě když:

1. Prvkům univerza U_1 je možno přiřadit jednoznačně prvky univerza U_2 (opačně tomu tak být nemusí, N:1).
2. Prvkům charakteristiky R_1 je možno jednoznačně přiřadit prvky charakteristiky R_2, a to tak, že prvku charakteristiky R_1 vyjadřujícímu orientovaný vztah mezi dvěma prvky univerza U_1 je vždy přiřazen právě ten prvek charakteristiky R_2, který vyjadřuje stejně orientovaný vztah mezi odpovídající dvojicí prvků univerza U_2 ve smyslu bodu 1.

Poznámka: Vytváření homomorfních systémů je základním principem modelování.

Jednoduché příklady izomorfismu a homomorfismu





Okolí systému

Podstatné okolí systému zahrnuje vše co má vliv na chování systému a není jeho součástí.

- Uzavřený systém — nekomunikuje s okolím (často jen zanedbáváme vliv okolí)
- Otevřený systém — komunikuje s okolím (typicky má definován vstup i výstup)

Klasifikace prvků systémů

Klasifikace 1:

- Prvky se spojitým chováním
- Prvky s diskrétním chováním

Klasifikace 2:

- Prvky s deterministickým chováním
- Prvky s nedeterministickým chováním

Příklady:
Šumová dioda = spojitý prvek, stochastické chování
Rezistor = spojitý prvek, deterministické chování
FIFO Fronta = diskrétní prvek, deterministické chování

Klasifikace systémů

Typ systému závisí na typu jeho prvků.

Systémy:

- spojité: *všechny* prvky mají spojité chování
- diskrétní: *všechny* prvky mají diskrétní chování
- kombinované: obsahuje spojité i diskrétní prvky

Systémy:

- deterministické: všechny prvky deterministické
- nedeterministické: obsahuje alespoň jeden prvek s nedeterministickým chováním

Simulace

= experimentování s reprezentací simulačního modelu.

Cíl simulace:

získání nových informací o chování systému v závislosti na vstupních veličinách a na hodnotách parametrů.

Postup:

opakované řešení modelu (provádění simulačních běhů),

1. nastavení hodnot parametrů a počátečního stavu modelu,
2. zadávání vstupních podnětů z okolí při simulaci,
3. vyhodnocení výstupních dat (informací o chování systému)

Simulační běhy opakujeme tak dlouho, dokud nezískáme dostatek informací o chování systému nebo pokud nenalezneme takové hodnoty parametrů, pro které má systém žádané chování.

Typy simulace

Podle použitého popisu modelu:

- Spojitá / diskrétní / kombinovaná
- Kvalitativní / kvantitativní
- ...

Podle simulátoru:

- Na analogovém / číslicovém počítači, fyzikální
- ”Real-Time” simulace
- Paralelní a distribuovaná simulace

Další možnosti:

- Vnořená simulace (simulace v simulaci)
- *”Reality in the loop”*
- Interaktivní simulace, virtuální realita

Zpracování výsledků simulace

Postup:

- Záznam průběhu simulace
- Vizualizace výsledků, animace

Analýza získaných výsledků:

- Intuitivní vyhodnocení, heuristiky, ...
- Statistické zpracování
- Automatické vyhodnocení (např. pro optimalizaci)
- Porovnávání s naměřenými daty
- ...

Verifikace modelu

Verifikací ověřujeme korespondenci *abstraktního* a *simulačního* modelu, tj. izomorfní vztah mezi AM a SM.

- Předchází vlastní etapě simulace.
- Analogicky s programy v běžných programovacích jazycích představuje verifikace simulačního modelu jeho ladění.

Poznámka:
Abstraktní model je formální specifikací pro program (simulační model).

Validace modelu

Ověřování validity (platnosti) simulačního modelu je proces, v němž se snažíme dokázat, že skutečně pracujeme s modelem adekvátním modelovanému systému.

- Jeden z nejobtížnějších problémů modelování.
- Vyžaduje neustálou konfrontaci informací, které o modelovaném systému máme a které simulací získáváme.
- Nelze absolutně dokázat přesnost modelu.
  (Validitu modelu chápeme jako míru použitelnosti/správnosti získaných výsledků.)
- Pokud chování modelu neodpovídá předpokládanému chování originálu, musíme model modifikovat.

Simulační nástroje

Simulační systémy usnadňují vytváření modelů, provádění experimentů a analýzu výsledků.

Tyto nástroje jsou použitelné pro:

- práci s abstraktními modely (báze znalostí, ...),
- programování simulačních modelů (simulační jazyky a knihovny modelů),
- experimentování se simulačními modely (simulátory),
- vizualizaci a vyhodnocování výsledků.

Poznámka: V rámci předmětu IMS použijeme SIMLIB/C++, systémy Dymola/OpenModelica a případně SciLab/Octave/Matlab - viz odkazy na WWW IMS

Shrnutí úvodní části

- Metoda simulace
- Použití simulace v různých oborech
- Výhody a problémy
- Základní teoretické souvislosti
- Problém verifikace a validace modelů
- Stručná charakteristika simulačních nástrojů

Modely a modelování

Přehled:

- Modelování – proces vytváření modelů
  - abstraktní model
  - simulační model
- Klasifikace modelů
- Popis modelů
- Příklady jednoduchých modelů

Příklady abstraktních modelů

Způsoby matematického popisu modelů:

- Konečný automat
- Petriho síť
- Turingův stroj
- Algebraické rovnice
- Diferenciální rovnice (obyč. i parciální)
- Diferenční rovnice
- Markovské procesy
- ...

Poznámka: Klasifikace abstraktních modelů

Vytvoření abstraktního modelu I

Formulace zjednodušeného popisu systému abstrahujícího od všech nedůležitých skutečností vzhledem k *cíli a účelu* modelu.

- Nedovedeme postihnout reálný svět v celé komplikovanosti
- Zajímáme se jen o ohraničené části
- Identifikace vhodných složek systému
- Systém nemusí být definován pouze na reálném objektu — potom vycházíme ze znalostí analogických systémů.

Z hlediska teorie systémů předpokládáme mezi modelovaným systémem a abstraktním modelem *homomorfní* vztah.

Vytvoření abstraktního modelu II

Specifické cíle a účely modelů:

- *Studium chování systému* pro určitá specifická kritéria, zkoumání povahy závislostí mezi parametry a odezvou systému.
- *Predikce* — vyhodnocení chování systému za určitých podmínek.
- *Analýza citlivosti* — určení faktorů (parametrů), které jsou pro činnost systému nejvýznamnější.
- *Optimalizace* — nalezení takové kombinace parametrů, která vede k nejlepší odezvě systému.

Vymezení účelu modelu má významný dopad na celý proces budování abstraktního modelu i na vlastní experimentování se simulačním modelem.

Vytvoření simulačního modelu

simulační model = abstraktní model zapsaný formou programu

Vztahy mezi modely

homomorfní vztah: *modelovaný systém — abstraktní model*
izomorfní vztah: *abstraktní model — simulační model*

Poznámky:

- Izomorfní vztah představuje silnější vztah ekvivalence mezi abstraktními systémy — shodnost struktur a chování prvků uvažovaných systémů.
- Konkrétní implementace simulačního modelu závisí na typu modelu a na použitém simulačním nástroji.

Klasifikace modelů 1

Tradiční rozdělení:

- spojité modely
- diskrétní modely
- kombinované modely

Poznámka:
Odpovídající varianty DEVS formalismu: DEVS, DESS, DEVS&DESS

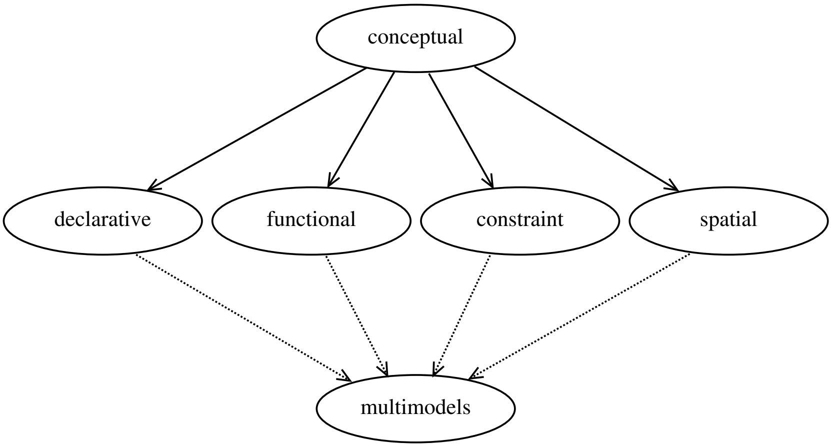
Klasifikace modelů 2

Klasifikace podle [Fishwick]:

- Konceptuální modely
- Deklarativní modely
- Funkcionální modely
- Modely popsané rovnicemi (*constraint*)
- Prostorové (*spatial*) modely
- Multimodely

Poznámka: Multimodel je složen z modelů různého typu.

Klasifikace modelů 2 — obrázek



Konceptuální modely

- Modely, jejichž komponenty (prozatím) nebyly přesně popsány ve smyslu teorie systémů.
- Obvykle se používají v počáteční fázi modelování pro ujasnění souvislostí a komunikaci v týmu.
- Mají formu textu nebo obrázků.

Deklarativní modely

- Popis přechodů mezi stavy systému.
- Model je definován *stavy* a *událostmi*, které způsobí přechod z jednoho stavu do druhého za jistých podmínek.
- Vhodné především pro diskrétní modely.
- Obvykle zapouzdřeny do objektů (hierarchická struktura).

Příklady:

- Konečné automaty (deterministické i nedeterministické, Markovovy modely)
- Petriho sítě
- Událostmi řízené systémy s kalendářem

Funkcionální modely

- Grafy zobrazující *funkce* a *proměnné*.
- Jsou možné 2 modifikace: uzel grafu je funkce nebo proměnná

Příklady:

- Systémy hromadné obsluhy se zařízeními a frontami (*"Queuing systems"*)
- Bloková schemata (spojitá simulace, ...)
- Kompartmentové systémy
- Grafy signálových toků
- Systémová dynamika

Modely popsané rovnicemi (constraint)

- Rovnice (algebraické, diferenciální, diferenční)
- Neorientované grafy (elektrická schemata, *"Bond-graphs"*)

Příklady:

- Diferenciální rovnice systému dravec-kořist:

$$\frac{dx_k}{dt} = k_1 x_k - k_2 x_k x_d$$

$$\frac{dx_d}{dt} = k_2 x_k x_d - k_3 x_d$$

- Balistika, kyvadlo, RC článek, ...
- Chaos (například *"Lorenz equation"*)
- Logistická rovnice $\quad x \leftarrow ax(1-x)$

Prostorové (spatial) modely

Rozdělují systém na prostorově menší ohraničené podsystémy.

Příklady:

- Parciální diferenciální rovnice (difůze, proudění, ...)
- Celulární automaty (hra *"Life"*)
- L-systémy
- *N-body problem*: mechanické modely těles + kolize

Multimodely

Modely složené z různých typů modelů, které jsou obvykle heterogenní (popsané různým způsobem).

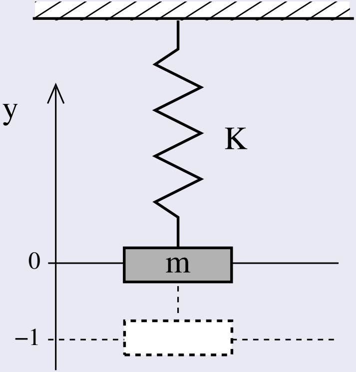
Příklady:

- Kombinované modely (např. spojité + diskrétní)
- Modely s neurčitostí (např. spojité + fuzzy)
- Modely na různé úrovni abstrakce (kvalitativní + kvantitativní)
- Spojování modelů (FMI, *"co-simulation"*, HLA, ...)
- ...

Poznámka:
Většina netriviálních modelů spadá do této kategorie.

Příklad1: Závaží na pružině (spojitý)

Obrázek = konceptuální model



Příklad1 — pokračování

Diferenciální rovnice = abstraktní model (constraint)

$$y'' = -g - \frac{K}{m} y$$

Počáteční podmínky

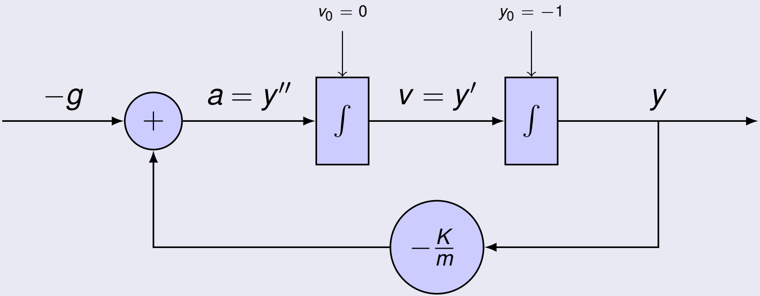
$$\begin{aligned} y(0) &= -1 \\ y'(0) &= 0 \end{aligned}$$

Poznámky:

- Počáteční podmínky jsou nutné pro jednoznačnost řešení
- Pozor na shodné jednotky (např. délka: metry / stopy ?)

Příklad1 — pokračování

Blokové schema = abstraktní model (funkcionální)



Příklad1 — pokračování

program = simulační model

```
// popis modelu v SIMLIB/C++
struct Model {
    Integrator v, y;
    Model(double m, double K, double y0) :
        v(-g - K/m * y, 0),
        y(v, y0)  {}
};

Model s(1, 1e3, -1);   // instance modelu s parametry

// vynechán popis simulačního experimentu
```

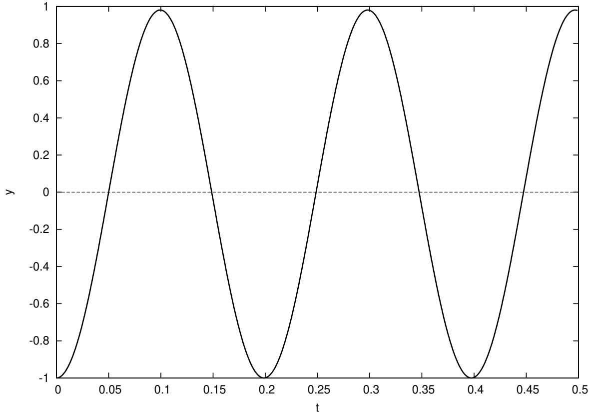
Příklad1 — pokračování

Výsledky simulace: tabulka (pozor na přesnost tisku)

```
# čas    y       v
 0.000 -1       0
 0.001 -0.9995 0.99
 0.002 -0.998  1.979
 0.003 -0.9955 2.966
 0.004 -0.9921 3.95
 0.005 -0.9876 4.93
 0.006 -0.9822 5.906
 0.007 -0.9758 6.875
 0.008 -0.9685 7.837
 0.009 -0.9602 8.792
 0.010 -0.9509 9.738
 ...
```

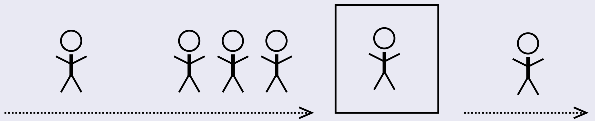
Příklad1 — pokračování

Výsledky simulace: graf (vytvořen programem Gnuplot)



Příklad2: Zákazníci v obchodě (diskrétní)

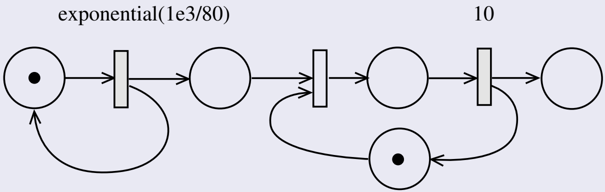
Obrázek = konceptuální model



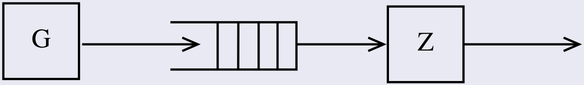
- Zákazníci přicházejí s určitým rozložením pravděpodobnosti,
- jsou obsluhováni zařízením po určitou dobu,
- vytváří se fronta zákazníků.

Příklad2 — pokračování

Petriho síť = abstraktní model (deklarativní)



Blokové schema = abstraktní model (funkcionální)



Příklad2 — pokračování

program = simulační model:

```
Facility  Linka("Linka");
class Zakaznik : public Process { // třída zákazníků
  void Behavior() {               // --- popis chování
    Seize(Linka);                 // obsazení zařízení
    Wait(10);                     // obsluha
    Release(Linka);               // uvolnění zařízení
  }
};
class Generator : public Event {  // generátor zák.
  void Behavior() {               // --- popis chování
    (new Zakaznik)->Activate();   // nový zákazník
    Activate(Time+Exponential(1e3/80)); // interval
  }
}; // ... vynechán popis experimentu a sběr statistik
```

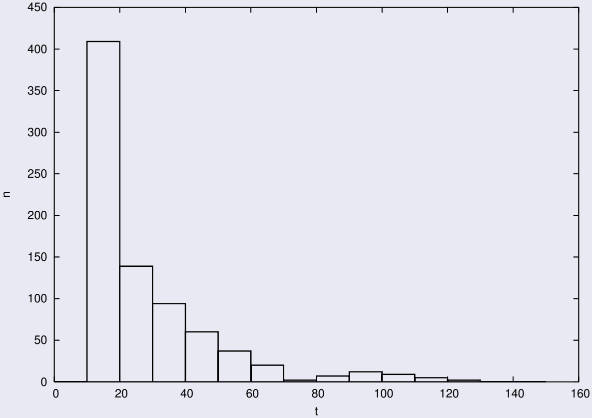
Příklad2 — pokračování

Výsledky simulace: statistiky

```
FACILITY Linka
 Time interval = 0 - 10000
 Number of requests = 797
 Average utilization = 0.796405

Input queue 'Linka.Q1'
 Maximal length = 12
 Average length = 1.37766
 Minimal time = 0.00798347
 Maximal time = 112.171
 Average time = 22.1489
 Standard deviation = 31.087
```

Příklad2 — pokračování

Výsledky simulace: histogram – čas strávený v systému



Shrnutí

- Modely různých typů a úrovní abstrakce
- Jsou možné různé popisy stejného systému
- Často nutné kombinovat = multimodely
- Je výhodné použít objekty (OOP)
- Použití hotových modelů jako komponent
- ...

Vše se odrazí v implementaci nástrojů pro simulaci.

Simulační nástroje — přehled

Možnosti:

- použití obecných jazyků (C, C++, Java, Python, ...)
- simulační knihovny pro obecné jazyky (SIMLIB/C++, SimPy, ...)
- simulační jazyky (Simula67, Modelica, ACSL, ...)
- simulační systémy (OpenModelica, Dymola, ANSYS, PowerDEVS, ...)
- propojování různých nástrojů (HLA, FMI, ...)

Poznámky:
HLA = *High Level Architecture*, standard pro distribuovanou simulaci
FMI = *Functional Mockup Interface*

Simulační jazyky

= speciální programovací jazyky

Poskytují prostředky usnadňující efektivní popis:

- struktury modelů (propojení komponent)
- chování modelů (rovnice, algoritmy)
- simulačních experimentů

Výhody:

- jednodušší popis modelu (snadnější verifikace)
- možnost automatické kontroly popisu modelu

Nevýhody:

- další jazyk = překladač, výuka, údržba
- relativně málo používáno = problémy (cena, chyby)

Typy simulačních jazyků

Klasifikace podle typu modelů:

- diskrétní
- spojité
- kombinované
- ...

Příklady:

- Simula67,
- Simscript,
- ACSL (Advanced Continuous Simulation Language),
- Modelica,
- ...

Dostupné simulační nástroje

V rámci výuky budeme používat:

- SIMLIB/C++: OO knihovna pro C++ (LGPL)
- DYMOLA/Modelica: komerční program
  OpenModelica: volně dostupné
- (Octave nebo SciLab: integrované prostředí, jazyk pro numerické výpočty, knihovny, různé specializované nadstavby – ”toolkity”)

Podpůrné nástroje:

- Vizualizace: Gnuplot
- Statistika: GNU R, ”diehard” testy
- ...

Přehled některých dalších programů

- Matlab/Simulink
- Sage
- GNU Octave (a OctaveForge)
- Simula67 (cim)
- SciPy, NumPy — Python
- SPICE — elektrické obvody
- GSL = *GNU Scientific Library* — knihovna, ISO C
- ... (další viz WWW)

Shrnutí

- Nástrojů pro podporu modelování a simulace existuje velmi mnoho.
- Některé nástroje vyžadují speciální vybavení (superpočítače).
- Většina nástrojů je zaměřena na konkrétní problém/oblast.
- Podrobnější informace o používaných nástrojích budou uvedeny později.

Reklama: Předmět *Simulační nástroje a techniky* v magisterském studiu bude zahrnovat podrobnosti o efektivní implementaci simulačních systémů a pokročilých metodách modelování a simulace.

Modelování náhodných procesů

Obsah:

- Pravděpodobnost, náhodné proměnné, ...
- Rozložení náhodných čísel
- Generování pseudonáhodných čísel
- Transformace rozložení pseudonáhodných čísel
- Testování pseudonáhodných čísel
- Metoda *Monte Carlo*

Poznámka:
Předpokládáme základní znalosti z teorie pravděpodobnosti.

Pravděpodobnost a statistika

- Co je náhodnost? (nedeterminismus, pseudonáhodnost)
- Některé části reality neumíme popsat jinak $\Rightarrow$ používáme náhodné jevy/procesy
- Každý proces má jiný charakter (a odpovídající rozložení)
- Jde o jeden ze způsobů popisu neurčitosti
- Příklady: příchody zákazníků, doba obsluhy, ...
- Postup:
  1. Změříme vzorek procesu v realitě (získáme soubor dat),
  2. ten aproximujeme analytickým vyjádřením (typicky pomocí existujícího rozložení),
  3. a nakonec náhodný proces modelujeme generátorem pseudonáhodných čísel (s odpovídajícím rozložením a parametry).

Terminologie (opakování)

- Jev
- Pravděpodobnost jevu
- Náhodná proměnná
- Rozložení pravděpodobnosti
- Distribuční funkce (*CDF*),
- Funkce hustoty pravděpodobnosti (*PDF*)
- Střední hodnota (*Mean*)
- Rozptyl (*Variance*)
- Zákon velkých čísel
- Centrální limitní věta
- ...

Náhodné proměnné

Náhodná proměnná je taková veličina, která jako výsledek pokusů může nabýt nějakou hodnotu, přičemž předem nevíme jakou konkrétně.

Náhodné proměnné rozdělujeme na:

diskrétní: nabývají jen konečně nebo spočetně mnoha různých hodnot
(Příklad: co padne při hodu kostkou)

spojité: hodnoty spojitě vyplňují určitý interval
(Příklad: čas mezi příchody zákazníků)

Poznámka:
Náhodné proměnné označujeme velkými písmeny, např. X, a jejich možné hodnoty odpovídajícími malými písmeny, např. x_1, x_2.

Náhodné proměnné – pokračování

Náhodné veličiny můžeme zadat:

- distribuční funkcí
- rozdělením pravděpodobnosti (např. funkce hustoty)

Existuje celá řada různých používaných rozložení, viz literatura. Například:

McLaughlin M.: *A Compendium of Common Probability Distributions*, 2016

`https://www.causascientia.org/math_stat/Dists/Compendium.pdf`

Diskrétní rozdělení pravděpodobnosti

určuje vztah mezi možnými hodnotami náhodné veličiny x_i a jim příslušejícími pravděpodobnostmi $p_i = P(X = x_i)$.

Obecně platí: $\sum_{i=1}^{\infty} p_i = 1$

Lze definovat například tabulkou pravděpodobností pro všechny možné hodnoty náhodné proměnné:

x_i	x_1	x_2	...	x_N
p_i	p_1	p_2	...	p_N

Musí platit: $\sum_{i=1}^{N} p_i = 1$
kde N je počet možných hodnot

Diskrétní distribuční funkce

Distribuční funkce náhodné veličiny X je funkce

$$F(x) = P(X \leq x)$$

kde $P(X \leq x)$ je pravděpodobnost toho, že náhodná veličina X nabude hodnoty menší nebo rovnu zvolenému reálnému číslu x.

Platí: $F(x) = \sum_{x_i \leq x} p_i$

Příklad: Hod kostkou

x_i	1	2	3	4	5	6
$F(x_i)$	$1/6$	$2/6$	$3/6$	$4/6$	$5/6$	1

Poznámka: Graf distribuční funkce diskrétní náhodné proměnné je po částech konstantní.

Spojité náhodné proměnné

- Distribuční funkce: $F(x) = P(X \leq x) = \int_{-\infty}^{x} f(u)du$
- Funkce hustoty pravděpodobnosti: $f(x) = \frac{dF(x)}{dx}$

Distribuční funkce $F(x)$ má tyto základní vlastnosti:

- $P(a \leq X \leq b) = F(b) - F(a)$
- $F(x_1) \leq F(x_2)$ pro $x_1 < x_2$ (je neklesající)
- $\lim_{x \to \infty} F(x) = 1$
- $\lim_{x \to -\infty} F(x) = 0$

Spojité náhodné proměnné

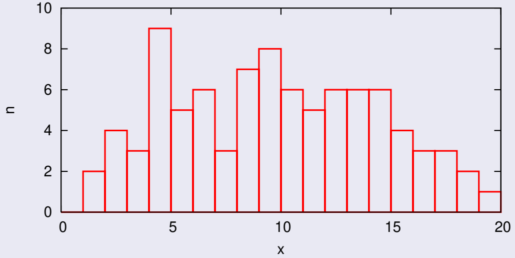
Hustota pravděpodobnosti $f(x)$ má tyto základní vlastnosti:

- $f(x) \geq 0$
- $f(x) = \frac{dF(x)}{dx}$
- $\int\limits_{-\infty}^{\infty} f(x)dx = 1$
- $P(a \leq X \leq b) = \int\limits_{a}^{b} f(x)dx$

Histogram

Mějme soubor N výsledků pokusů.
Histogram H roztřídí soubor do K tříd podle vhodně zvolených intervalů. Hodnota $H(i) =$ počet výsledků v $i-$tém intervalu.

Příklad histogramu pro $K = 20$, $N = 100$



Poznámky: Problém stanovení optimálního počtu intervalů.
(Histogram se blíží funkci hustoty pravděpodobnosti.)

Charakteristiky náhodných proměnných

Charakteristiky polohy: posunutí vzhledem k počátku (střed, modus, medián, kvantily).

Charakteristiky variability: rozsah kolísání hodnot kolem středu (rozptyl a směrodatná odchylka).

Charakteristiky šikmosti: udává nesymetričnost rozložení.

Charakteristiky špičatosti: porovnává variabilitu rozložení ve středu a na okrajích.

Charakteristiky lze stanovit podle tzv. momentů.

Obecné momenty

Je-li X náhodná veličina s frekvenční funkcí p_i resp. hustotou pravděpodobnosti $f(x_i)$, pak

obecný moment k-tého řádu je:

$$m_k(X) = \sum_i x_i^k p_i$$

$$m_k(X) = \int_{-\infty}^{\infty} x^k f(x) dx$$

Příklad: Střední hodnota je moment prvního řádu:

$$E(X) = \int_{-\infty}^{\infty} x f(x) dx$$

$$E(X) = m_1(X) = \mu$$

Centrální momenty

Je-li X náhodná veličina s p_i resp. $f(x_i)$, pak

centrální moment k-tého řádu je:

$$M_k(X) = \sum_i [x_i - E(X)]^k p_i$$

$$M_k(X) = \int_{-\infty}^{\infty} [x - E(X)]^k f(x) dx$$

Příklad: Rozptyl je centrální moment 2. řádu:

$$D(X) = \sum_i [x_i - E(X)]^2 p_i$$

$$D(X) = M_2(X)$$

Koeficient šikmosti a špičatosti

Šikmost:

$$\beta_1 = \frac{M_3(X)}{\sigma(X)^3}$$

kde $\sigma(X) = \sqrt{D(X)}$ je směrodatná odchylka

Špičatost:

$$\beta_2 = \frac{M_4(X)}{\sigma(X)^4}$$

Použití momentů

- Testování náhodných čísel
- Odhady parametrů rozložení
- ...

Poznámka:
Konkrétní parametry konkrétního rozložení se projeví v jeho momentech. Z odhadu lze zpětně vyčíslit parametry.

Některá často používaná rozložení

- Diskrétní:
  - Poissonovo
  - Binomické
  - ...
- Spojitá:
  - rovnoměrné (Uniform)
  - exponenciální (Exponential)
  - normální (Normal, Gauss)
  - Pearsonovo (χ^2)
  - ...

Poissonovo rozložení

Diskrétní rozložení

$$p_i = \frac{\lambda^i}{i!} e^{-\lambda}, \quad \lambda > 0, \quad i \in \{0, 1, 2, ...\}$$

$$E(x) = \lambda, \quad D(x) = \lambda, \quad \beta_1 = \frac{1}{\sqrt{\lambda}}, \quad \beta_2 = \frac{1}{\lambda} + 3$$

Příklady použití:

- Systémy hromadné obsluhy: počet příchodů zákazníků do obchodu za jednotku času.
  Souvisí s exponenciálním rozložením (časový interval mezi příchody — počet příchodů za jednotku času).
- Počet hovorů přes telefonní ústřednu za hodinu.
- Počet alfa částic, které vstoupí za daný časový interval do dané oblasti.
- Počet zmetků ve výrobě za 1 měsíc.

Rovnoměrné (Uniform) rozložení

Obvykle označujeme $R(a, b)$.

$F(x) = 0$ pro $x \le a$, $\frac{x-a}{b-a}$ pro $a \le x \le b$, jinak 1

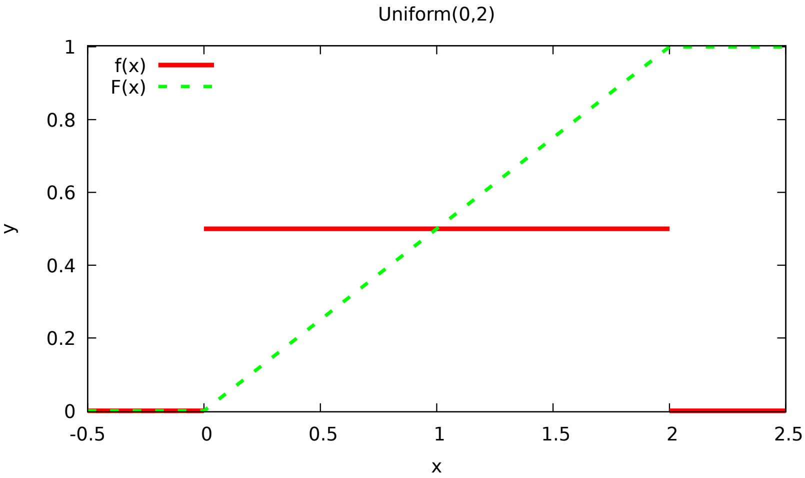
$f(x) = \frac{1}{b-a}$ pro $x \in \langle a, b \rangle$, jinak 0

V normované formě $R(0, 1)$ je základem pro generování dalších rozložení.

Charakteristiky:

- Střední hodnota: $E(X) = \frac{a+b}{2}$
- Rozptyl: $D(X) = \frac{(b-a)^2}{12}$

Příklad: Rovnoměrné rozložení



Exponenciální rozložení

$f(x) = \frac{1}{A} e^{-\frac{1}{A}(x - x_0)}$ pro $x \geq x_0$, jinak 0

$$F(x) = \begin{cases} 1 - e^{-\frac{1}{A}(x - x_0)} & x \geq x_0 \\ 0 & x < x_0 \end{cases}$$

kde A a x_0 jsou parametry.

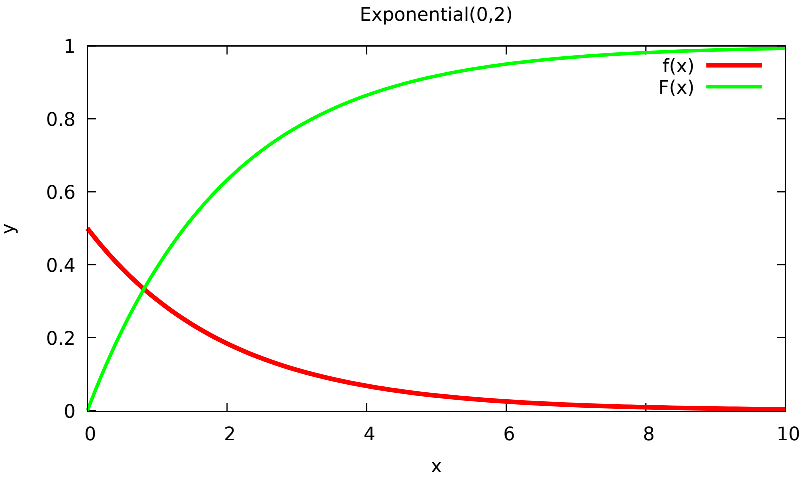
Často se používá normované rozložení s $x_0 = 0$.

- Střední hodnota: $E(X) = x_0 + A$
- Rozptyl: $D(X) = A^2$

Poznámka:
V literatuře se často používá jako parametr $\lambda = \frac{1}{A}$ ("rate").

Použití: rozložení dob obsluhy, časové intervaly mezi poruchami nebo mezi příchody požadavků.

Příklad: Exponenciální rozložení



Normální (Gaussovo) rozložení

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Parametry:

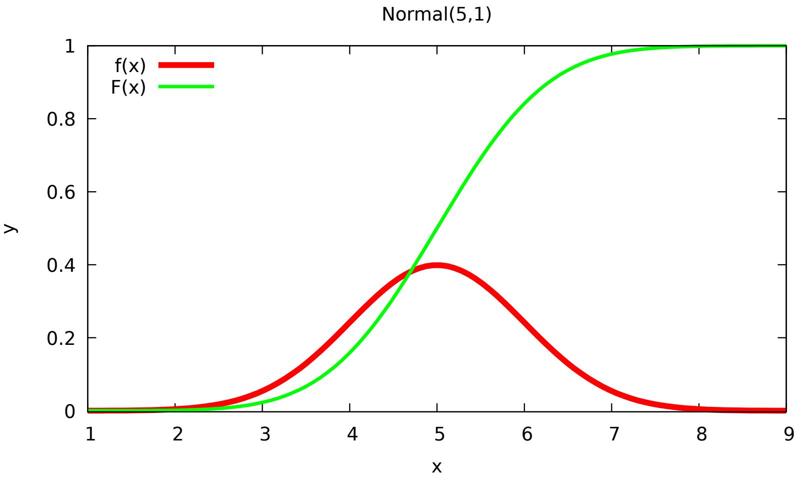
- Střední hodnota: μ
- Rozptyl: σ^2 (směrodatná odchylka: σ)

Pravidlo tří sigma:

$$P(X \in \langle \mu - 3\sigma, \mu + 3\sigma \rangle) \geq 0.99$$

Použití: Odpovídá jevům s vlivem většího počtu nezávislých faktorů.

Příklad: Normální rozložení



Pearsonovo rozložení χ_k^2

Založeno na normálním rozložení s parametry $\mu = 0, \sigma = 1$.

$$\chi_k^2 = \sum_{i=1}^{k} X_i^2, \quad X \text{ je z } \quad N(0,1)$$

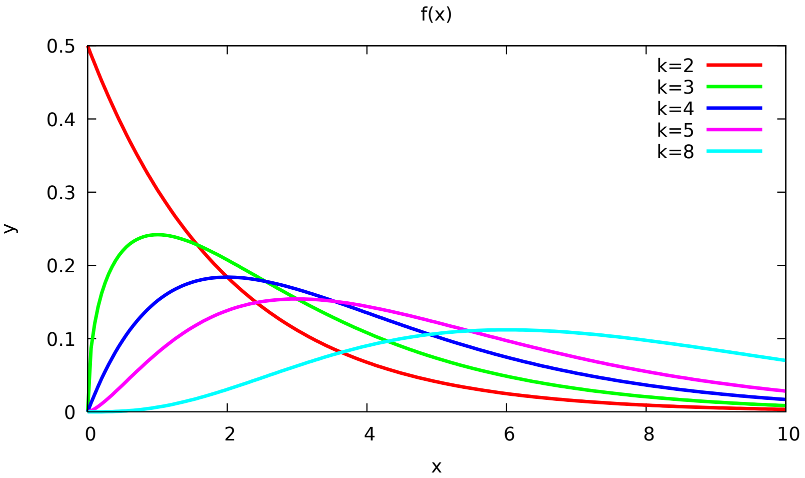
k - počet stupňů volnosti (nemělo by přesáhnout 50, protože pak výsledek konverguje k 1).

$$f(x) = (x)^{\frac{k}{2}-1} \ \exp(-\frac{x}{2}) \ 2^{-\frac{k}{2}} \ \frac{1}{\Gamma(\frac{k}{2})}$$

Charakteristiky: $E(\chi_k^2) = k, \quad D(\chi_k^2) = 2k$

Použití: Testování statistických hypotéz.

Příklady: Pearsonovo rozložení χ^2_k



Generování pseudonáhodných čísel

- základem je kvalitní generátor rovnoměrného rozložení
- transformací (rovnoměrného rozložení) získáme soubor čísel jiného rozložení

Problém:

Náhodná $\times$ Pseudonáhodná čísla

Generátory:

- Fyzikální zdroje náhodnosti: opravdu náhodné (nedeterministické), generují jen málo bitů za sekundu
- Algoritmické generátory: pseudonáhodné (deterministické), generují řádově miliardy bitů za sekundu

Kongruentní generátory

$$x_{n+1} = (ax_n + b) mod\ m$$

kde konstantní parametry a, b a m (multiplikační, aditivní, modul) musí mít vhodné hodnoty

- Generují *rovnoměrné rozložení* $\langle 0, m)$
- Rozsah $\langle 0, m)$ se převádí na normovaný $\langle 0, 1)$
- Generují periodicky se opakující posloupnost čísel. *Perioda generátoru* je maximálně m a závisí na parametrech.
- Dvě po sobě generovaná čísla nejsou statisticky nezávislá.

Příklad: Kongruentní generátor v C

Jednoduchý generátor (32 bitů), rozsah $\langle 0, 1)$

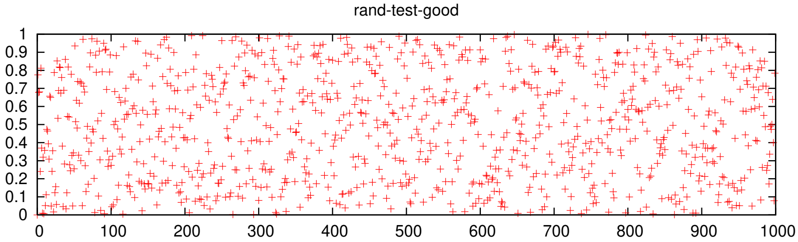
```
static uint32_t ix = SEED; // počáteční hodnota, 32b
double Random(void) {
    ix = ix * 69069u + 1u; // mod 2^32 je implicitní
    return ix / ((double)UINT32_MAX + 1.0);
}
```

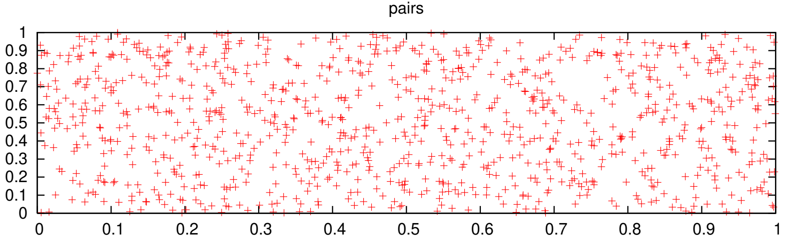
Poznámky: Nepříliš kvalitní (závislost dvojic, krátká perioda). Pro generování malých čísel nikdy nepoužívejte operaci modulo (např. `1 + (ix % 6)` pro hody kostkou je nevhodné, vždy použijte `1 + (int)(Random()*6)`).

Příklad: Kongruentní generátor v C++

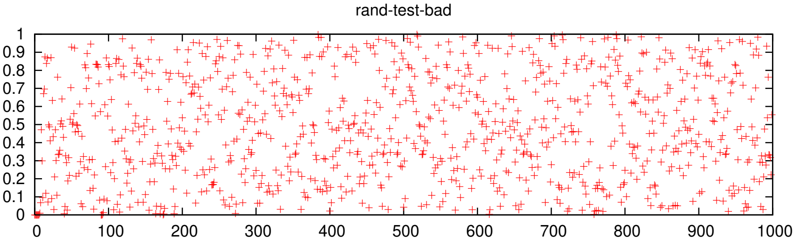
```
class Kongr {
    unsigned ix;          // stav generátoru
public:
    Kongr(unsigned seed = SEED) : ix(seed) {}
    // základní generátor <0,UINT_MAX)
    unsigned next() {
        ix = ix * A + B; // implicitně modulo
        return ix;
    }
    // mapováno na rozsah <0,1)
    double nextDouble() {
        return  next()/(UINT_MAX+1.0);
    }
} generator1;   // použití: generator1.next()
```

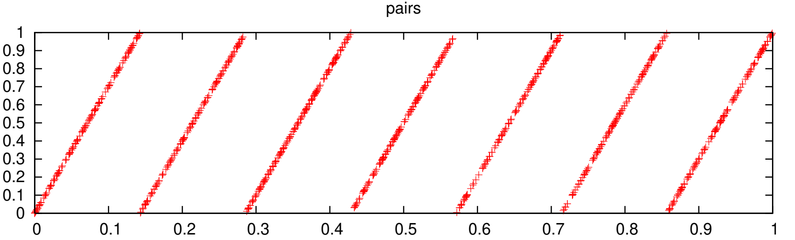
Příklad: $a = 69069$, $b = 1$, $m = 2^{32}$





Příklad: $a = 7$, $b = 1$, $m = 2^{32}$ (Nevhodné parametry!)





Další metody generování

- Mersenne twister (perioda $2^{19937} - 1$)
- Xorshift
- různé další varianty LCG (Linear Congruential Generator)
- bitové operace, carry — LFSR (Linear Feedback Shift Register)
- ...

Požadavky na generátory:

- rovnoměrnost rozložení
- statistická nezávislost generované posloupnosti
- co nejdelší perioda
- rychlost

Příklad: Xorshift generátor v C

```
// Zdroj: Marsaglia: "Xorshift RNGs"
// stav musí být inicializován (SEED!=0)
uint64_t xorshift64(uint64_t *state)
{
    uint64_t x = *state;
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    return (*state = x);
}
```

Poznámky: Dovoluje použít více různých stavů, například pro vlákna. Existují varianty pro 32, 64, 128 i více bitů ve stavu.

Transformace na jiná rozložení

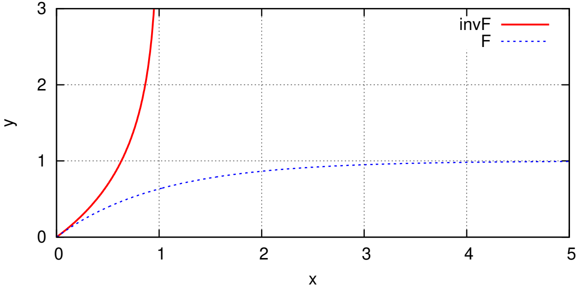
Metody:

- Inverzní transformace — používá inverzní distribuční funkci cílového rozložení. Pro některá rozložení nelze použít (např. když distribuční funkci nelze vyjádřit elementárními funkcemi).
- Vylučovací — sérií pokusů hledáme číslo, které vyhovuje funkci hustoty cílového rozložení. Nevhodná pro neomezená rozložení.
- Kompoziční — složitou funkci hustoty rozložíme na několik jednodušších (intervaly, na každý lze použít jinou metodu).
- Jiná, specificky vytvořená pro dané rozložení.

Metoda inverzní transformace

1. Inverze distribuční funkce: F^{-1}
2. Generování $x = Uniform(0, 1)$
3. Výsledek: $y = F^{-1}(x)$

Příklad: Exponenciální rozložení $F(x) = 1 - e^{-\frac{x-x_0}{A}}$
$y = x_0 - A * \ln{(1 - x)}$ viz. obrázek pro $x_0 = 0, A = 1$

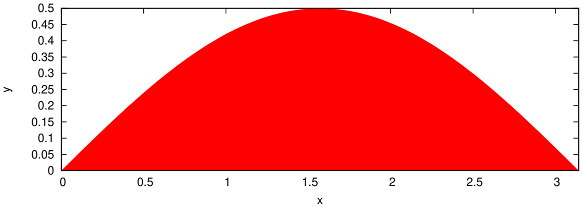


Vylučovací metoda

Náhodnou veličinu ξ s funkcí hustoty $f(x), x \in \langle x_1, x_2), f(x) \in \langle 0, M)$ (kde M je max. hodnota $f(x)$) generujeme takto:

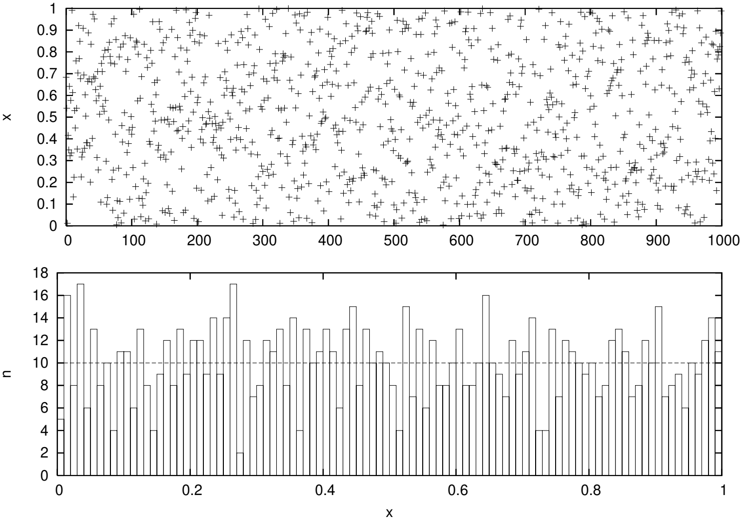
1. $x = Uniform(x_1, x_2)$
2. $y = Uniform(0, M)$
3. je-li $y < f(x)$, pak x prohlásíme za hodnotu náhodné veličiny ξ jinak goto 1

Efektivita generátoru souvisí s poměrem ploch $(x_1, x_2) \times (0, M)$ a $\int_{x_1}^{x_2} f(x)dx$

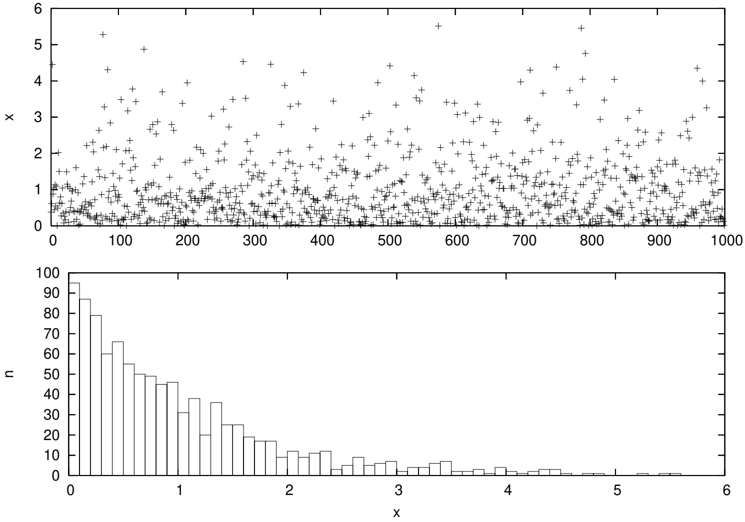


Poznámka: Nehodí se na neomezená rozložení.

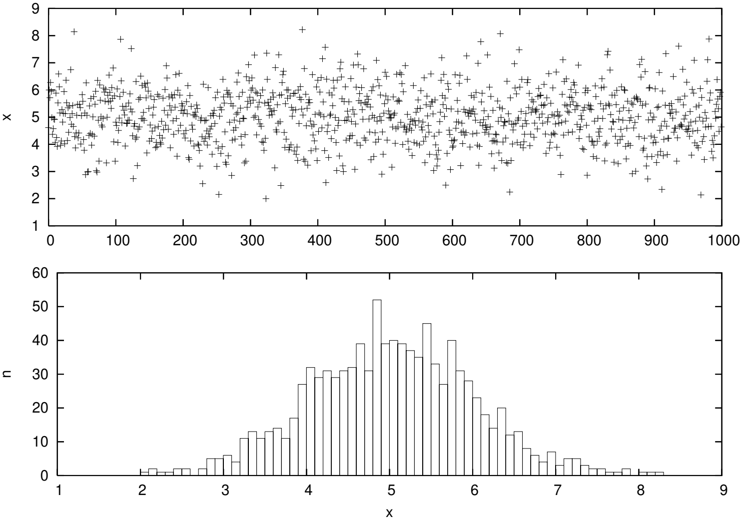
Vzorek čísel z generátoru Uniform(0,1)



Vzorek z generátoru Exponential(0,1)



Vzorek z Normal(5,1)



Testování generátorů náhodných čísel

Máme soubor (pseudo)náhodných čísel a chceme:

- Potvrdit hypotézu jeho příslušnosti k danému rozložení.
- Nalézt jeho rozložení (případně nejvíce podobné).
- Nalézt takové parametry rozložení, aby vzorek odpovídal danému rozložení s těmito parametry.

Existuje mnoho statistických testů a nástrojů pro testování náhodných čísel (např. programy *diehard*, *dieharder*)

Poznámka: náročné, problém interpretace výsledků

Test dobré shody χ^2

Příklad testování generátoru náhodné veličiny:

1. Vygenerujeme soubor n vzorků (např. $n = 10000$).
2. Vypočteme histogram souboru H (pro k-kategorií).
3. Vypočteme teoretický (analytický) histogram rozložení h.
4. Výpočet: $$\chi^2_{k-1} = \sum_{j=1}^{k} \frac{(H_j - h_j)^2}{h_j},$$
5. Výsledek testu zhodnotíme na základě tabulky χ^2 :
   - zvolená hladina významnosti p (např. 0.05), x_p je kvantil rozložení pro počet stupňů volnosti $k - 1$
   - je-li $\chi^2_{k-1} > x_p$, pak generátor nevyhovuje

Přesnější popis viz literatura

Další metody testování

- testy rovnoměrnosti rozložení (χ^2)
- rovnoměrnost dvojic/trojic
- rovnoměrnost maxima z *n* členů
- testy náhodnosti
- test na intervaly, test sběratele kuponů
- poker test (celočíselný, $0 \leq x_i < d$)
- Hammingův test
- ...

Poznámka: Testování transformovaných rozložení

Metoda Monte Carlo

Experimentální numerická (simulační) metoda (metody):

- řeší danou úlohu experimentováním se stochastickým modelem
- využívá vzájemného vztahu mezi hledanými veličinami a pravděpodobnostmi, se kterými nastanou určité jevy
- vyžaduje generování (pseudo)náhodných čísel
- není příliš přesná
- existuje řada variant (*Metropolis, Quasi-MC*)

Postup:

1. vytvoříme stochastický model
2. provádíme náhodné experimenty
3. získanou pravděpodobnost nebo průměr použijeme pro výpočet výsledku

Aplikace metody Monte Carlo

- Historie: Buffonova úloha (π), projekt *"Manhattan"*
- Výpočet určitých integrálů
  - molekulární simulace ($3N$-rozměrný integrál)
  - počítačová grafika (*Path tracing*)
  - kontrola složení (např. salámu)
- Řešení diferenciálních rovnic (náhodné procházky)
- Finance

Souvislosti: náhodné vzorkování, průzkumy, *oversampling*, některé optimalizační metody (např. *Simulované žíhání*), ...

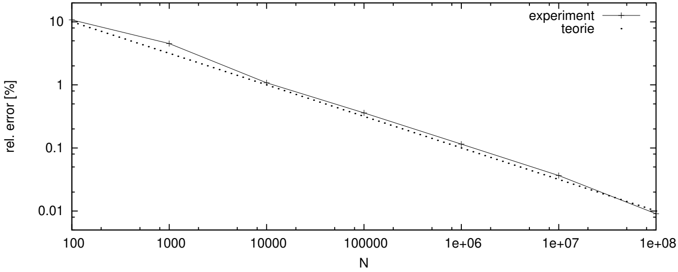
Přesnost metody Monte Carlo

Přesnost metody není velká — pro relativní chybu platí:

$$err = \frac{1}{\sqrt{N}}$$

kde N je počet provedených experimentů.

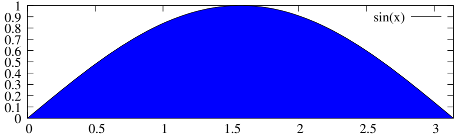
Příklad: Experiment — závislost chyby na počtu pokusů



Příklad1 — výpočet jednoduchého určitého integrálu

$$\int_0^\pi sin(x)dx$$

1. Generujeme N náhodných bodů (x_i, y_i) (rovnoměrně v rozsahu souřadnic: $rozsah_x = \langle 0, \pi \rangle$, $rozsah_y = \langle 0, 1 \rangle$)
2. Vypočteme *pravděpodobnost* P jevu $y_i < sin(x_i)$



3. Výsledek je přibližně roven $|rozsah_x| * |rozsah_y| * P$:

$$\int_0^\pi sin(x)dx \approx \pi P$$

Příklad1 — efektivnější metoda

$$\int_0^\pi sin(x)dx$$

Rychlejší a korektnější postup:

1. Generujeme N náhodných hodnot: $x_i \in \langle 0, \pi)$
2. Vypočteme *průměr*: $E = \frac{1}{N} \sum_{i=1}^{N} sin(x_i)$
3. Výsledek: $\int_0^\pi sin(x)dx \approx \pi E$

Lze dokázat, že pro $N \to \infty$ platí rovnost: $\int_0^\pi sin(x)dx = \pi E$

Poznámka: Nemusíme znát obor hodnot funkce.

Příklad2 — výpočet objemu tělesa

Výpočet objemu koule o poloměru r :

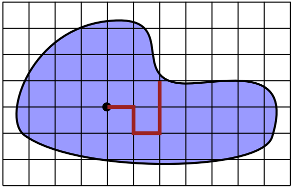
- Generujeme N náhodných bodů (x_i, y_i, z_i).
  Pro zjednodušení použijeme jako těleso kouli a pro všechny osy jen rozsah $\langle 0, r)$, který odpovídá $\frac{1}{8}$ koule.
- Vypočteme pravděpodobnost P jevu $x_i^2 + y_i^2 + z_i^2 < r^2$
- Výsledek: $objem \approx 8Pr^3$

Poznámka:
Metoda Monte Carlo je výhodná především pro vícerozměrné integrály, kdy běžné metody nejsou efektivní. Uvedené jednoduché příklady proto považujte pouze za ilustrační.

Příklad3 — Dirichletova úloha (řešení PDR) — princip

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

+ okrajové podmínky na hranici Γ: $u(Q) = f(Q), \quad Q \in \Gamma$



Postup řešení:

1. Volba čtvercové sítě, aproximace okrajů.
2. Provádíme náhodné procházky z výchozího bodu k okraji.
3. Průměrná hodnota koncových bodů náhodných procházek udává přibližnou hodnotu řešení $u(x, y)$ ve výchozím bodu.

Monte Carlo — shrnutí

- Velmi používaná metoda v případech, kdy běžné numerické metody jsou neefektivní nebo nepoužitelné (např. N-rozměrné integrály pro velké N).
- Jednoduchá implementace.
- Náročnost na kvalitu generátoru pseudonáhodných čísel.
- Relativně malá přesnost, nízká efektivita.
- Existují různé další varianty MC metod — viz literatura.

Diskrétní simulace

Agenda:

- Popis diskrétních systémů
- Systémy hromadné obsluhy (*Queuing Systems*)
  - Aktivní entity: procesy, události
  - Pasivní entity: fronty, zařízení, sklady,
- Příklady: Petriho sítě
- Implementace: Algoritmus řízení simulace, kalendář
- *"next-event"* simulace
- Nástroje pro sběr statistik
- Základní popis SIMLIB/C++
- Příklady: SIMLIB/C++

Formy popisu diskrétních systémů

- Program v (simulačním) programovacím jazyce
- Petriho sítě (C. A. Petri, 1962, existují různé varianty)
- DEVS (*Discrete Event System Specification*)
- Automaty, sítě automatů
- Procesní algebry (CCS, CSP, ...)
- π-calculus
- CHAM, PRAM
- ...

Procesy

V diskrétním modelování používáme pojem *proces*:

- Process je posloupnost událostí
- Paralelní procesy — současně prováděné procesy
- Kvaziparalelismus — provádění ”paralelních” procesů na jednoprocesorovém počítači

V modelovaných systémech často existuje mnoho paralelně probíhajících a vzájemně komunikujících procesů.

Poznámky:

- nepreemptivní implementace (např. *coroutines*)
- zapouzdření, objekty, agenti

Paralelismus

- Popis jednotlivých procesů sekvencí kroků (program).
- Popis komunikace procesů — zprávy (synchronní, asynchronní).
- Synchronizace při používání sdílených prostředků.

Petriho sítě

Definice P/T Petriho sítě:

$$\Sigma = (P, T, F, W, C, M_0)$$

kde:

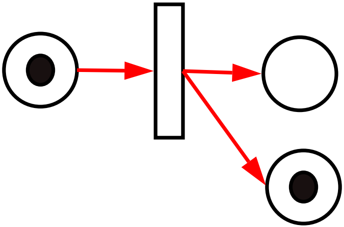
- P je množina míst (stavy)
- T je množina přechodů, $P \cap T = \emptyset$
- Incidenční relace $F \subseteq (P \times T) \cup (T \times P)$
- Váhová funkce $W: \ F \rightarrow \{1, 2, ...\}$
- Kapacity míst $C: \ P \rightarrow N$
- Počáteční značení $M_0: \ P \rightarrow N$
  (M se nazývá *značení* Petriho sítě)

Graf Petriho sítě

Obvykle zadáváme Petriho síť formou grafu:

- Místa – kružnice
- Přechody – obdélníky
- Incidenční relace – šipky (orientované hrany)
- Váhová funkce – ohodnocení hran

Příklad:



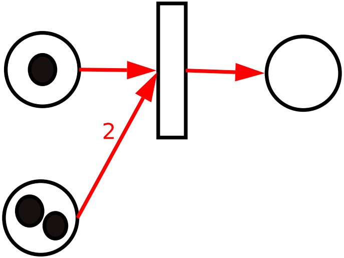
Chování Petriho sítě

Proveditelnost přechodu v Petriho síti:

Přechod je proveditelný při značení *M*, jestliže:

- ve vstupních místech čeká dostatek procesů
- a současně výstupní místa mají dostatečně volnou kapacitu.

Příklad:



Petriho sítě v modelování

Petriho sítě mohou modelovat:

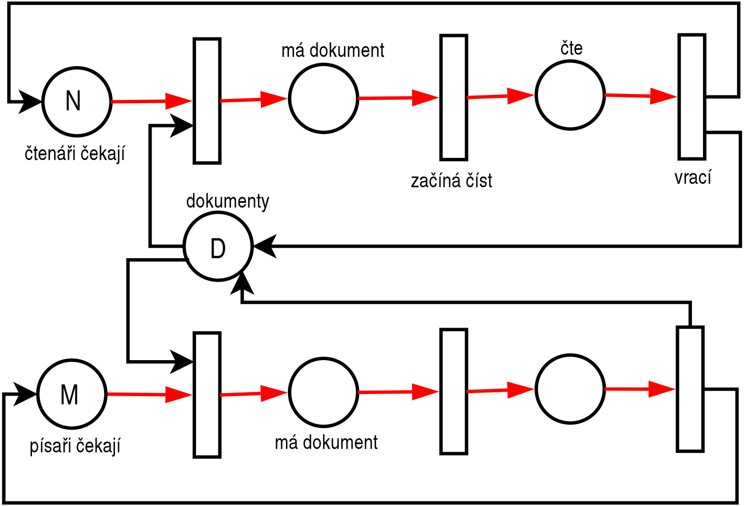
- paralelismus procesů
- komunikaci a synchronizaci procesů
- nedeterminismus

Pro modelování diskrétních systémů zavádíme do klasických P/T Petriho sítí několik rozšíření: priority, pravděpodobnosti a doby přechodů.

Další typy Petriho sítí

- Hierarchické – do sebe vnořené sítě
- Barvené – značky mají datový typ ("barvu")
- Objektově orientované – OOPN, PNtalk
- Stochastické – P/T síť s prioritami, pravděpodobnostmi a časováním přechodů.
- ...

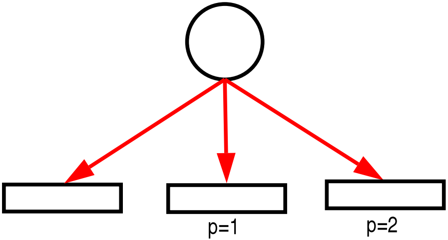
Příklad: čtenáři a písaři



Prioritní přechody

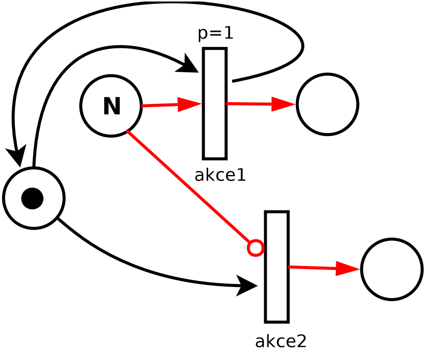
- je-li více přechodů proveditelných z jednoho značení, můžeme jim dát priority
- $p_t \in \{0, 1, 2, 3, 4, ...\}$
- vyšší číslo $\Rightarrow$ vyšší priorita
- implicitně je priorita $p_t = 0$

Příklad:

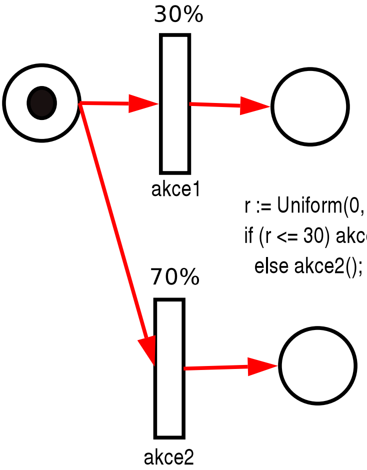


Poznámka: Inhibiční hrany

```
while ( N > 0 ) {
    akce1();
    N = N - 1;
}
akce2();
```



Pravděpodobnost provedení přechodu



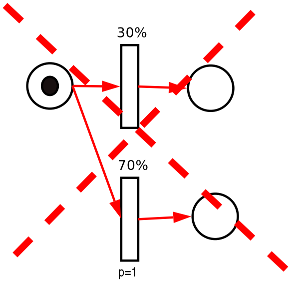

```
r := Uniform(0,100)
if (r <= 30) akce1();
  else akce2();
```

Pravidla používání rozšířených přechodů

- Přechod má pouze JEDEN parametr (priorita, pravděpodobnost, časování).
- Pozor: tento parametr NENÍ parametrem HRANY.

Příklad – CHYBNĚ

Nejednoznačnost – přechod se provede s pravděpodobností 70%, ale prioritně = NESMYSL!

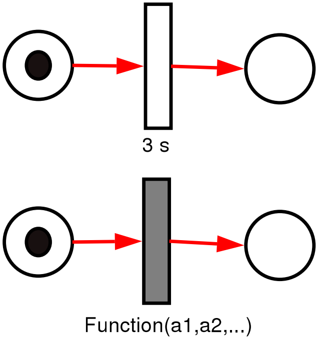


Časované Petriho sítě

Přidání modelového času:

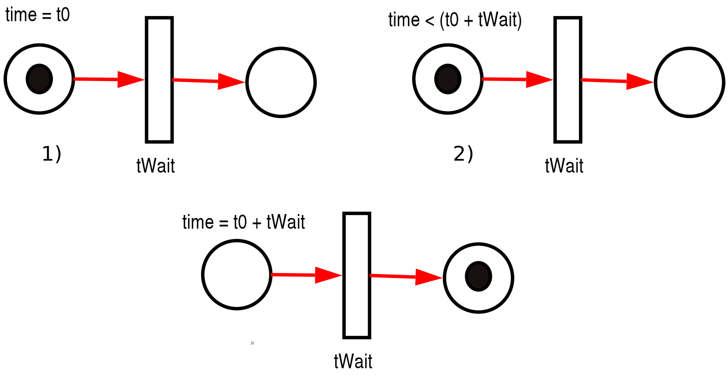
Časovaný přechod má parametr – dobu provádění:

- Konstantní čas čekání
- Náhodně generovaná doba čekání



Sémantika časovaného přechodu

- Pokud je přechod je v čase *t* proveditelný, spustí se odpočet času
- Po celou dobu odpočítávání se nemění stav značek
- Na konci doby se provede přemístění značek



Sémantika časovaného přechodu 2

Běžný časovaný přechod neomezuje počet současně čekajících.

Někdy ale zavádíme kapacitu časovaného přechodu:

Kapacita přechodu udává kolik procesů může na přechodu čekat současně:

- jeden (implicitně), vzniká fronta
- více (nutno specifikovat poznámkou u přechodu)

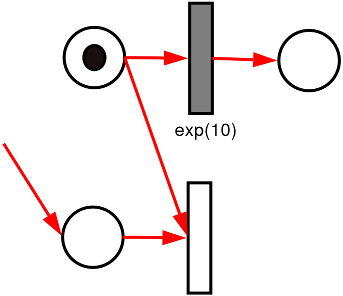
Poznámka:

Poznámka k sémantice časového přechodu: lze řešit i dočasným odstraněním značek na dobu odpočítávání času, ale to komplikuje další případy, jako je například přerušení čekání.

Přechody ve Stochastické PN (SPN)

V SPN platí:

- Máme dva druhy přechodů: časované a okamžité
- Jediným povoleným parametrem časovaného přechodu je údaj o čase (náhodný nebo konstantní)
- Parametry okamžitého přechodu: priorita, pravděpodobnost
- Okamžitý přechod má vždy vyšší prioritu než časovaný



Systémy hromadné obsluhy (SHO)

SHO (*Queueing Systems*) jsou systémy obsahující zařízení (s frontami), která poskytují obsluhu transakcím.

Typický SHO obsahuje:

- transakce (=procesy) a popis jejich příchodů
- obslužné linky (více typů) a popis obsluhy
- fronty různých typů ve kterých transakce čekají

Co sledujeme při simulaci:

- informace o čase stráveném transakcí v systému
- doby čekání ve frontách
- vytížení obslužných linek

Cíl: odhalit různá zdržení, optimalizovat výkon, ...

Vstupní tok požadavků

Obvykle jde o stochastický proces příchodů do systému

- Při modelování příchodů zadáváme:
  - Střední dobu mezi příchody (obvykle používáme exponenciální rozložení)
  - Počet příchodů za jednotku času (obvykle Poissonovo rozložení)
    Pojem: Intenzita příchodů požadavků

Fronty čekajících požadavků

Vytvoří se vždy, když požadavek chce být obsloužen již obsazeným zařízením. Pro fronty je charakteristické:

- řazení požadavků ve frontě (např. FIFO)
- způsob výběru požadavků z fronty
- největší možná délka fronty

Frontové řády : FIFO, LIFO, SIRO (Service in Random Order)

Nulová fronta : požadavek nemůže vstoupit do fronty, jde o systém se ztrátami

Fronta konečná : omezení kapacity fronty

Fronta s netrpělivými požadavky : netrpělivý požadavek opouští systém, překročí-li doba čekání určitou mez (*time-out*)

Prioritní fronty, priorita obsluhy

- Přicházející požadavky nejsou rovnocenné – požadavek na obsluhu může mít zvláštní prioritu.
- Prioritních úrovní může být více.
- U jedné obslužné linky lze vytvářet i několik front s různými prioritami.
- Vstupem požadavku s vyšší prioritou nastane jedna ze čtyř možností pro právě probíhající obsluhu požadavku s nižší prioritou – viz dále.

Prioritní obsluha

1. Započatá obsluha se normálně ukončí (slabá priorita).
2. Obsluha se přeruší a začne obsluha požadavku s vyšší prioritou obsluhy (silná priorita). Požadavek, jehož obsluha byla přerušena:
   - odchází ze systému neobsloužen
   - nebo se vrací znovu do fronty a když je později znovu obsluhován, tak:
     - obsluha pokračuje od přerušeného místa,
     - nebo začíná znovu od začátku.
3. Jsou-li všechny linky obsazeny a u každé je fronta, požadavek se sám rozhodne, do které fronty se zařadí.
4. Vytvářejí-li požadavky jednu společnou frontu, požadavek vstupuje do té obslužné linky, která se nejdříve uvolní.

Obslužná síť

Vznikne spojením několika obslužných linek.

Otevřená obslužná síť – výměna požadavků mezi sítí a okolím.

Uzavřená obslužná síť – nedochází k výměně požadavků mezi sítí a okolím.

Smíšená obslužná síť – pro některé typy požadavků je síť otevřená, pro jiné uzavřená.

Obslužná síť 2

Statické vlastnosti sítě jsou definovány:

- počtem a charakteristikou obslužných linek,
- topologií obslužné sítě.

Dynamické vlastnosti obslužné sítě jsou definovány:

- charakteristikou procesu příchodu požadavků
- charakteristikou procesu obsluhy požadavků
- charakteristikou procesu přechodu požadavků mezi obslužnými linkami
- strategií obsluhy požadavků v obslužných linkách sítě.

Kendallova klasifikace SHO

Standard stručného a přehledného vyjádření typu SHO (zavedl ji D. G. Kendall) – používá tři hlavní hlediska:

- **X** – typ stochastického procesu popisujícího příchod požadavků k obsluze
- **Y** – zákon rozložení délky obsluhy
- **c** – počet dostupných obslužných linek

Specifikace má tvar **X/Y/c**, kde:

- **X, Y** ... velká písmena M, D, G, E_k, K_n, GI – viz dále
- **c** ... přirozené číslo, včetně ∞

Příklad:
systém M/M/1

Kendallova klasifikace SHO

Symbol	X	Y
M	Poissonův proces příchodů tj. exponenciální rozložení vzájemně nezávislých intervalů mezi příchody	exponenciální rozložení doby obsluhy
E_k	Erlangovo rozložení intervalů mezi příchody s parametry λa k	Erlangovo rozložení doby obsluhy s parametry λ a k
K_n	rozložení χ^2 intervalů mezi příchody, $n$stupňů volnosti	rozložení χ^2 doby obsluhy
D	pravidelné deterministické příchody	konstantní doba obsluhy
G	žádné předpoklady o procesu příchodu	jakékoliv rozložení doby obsluhy
GI	rekurentní proces příchodů	

Modelování SHO

Při modelování SHO popisujeme:

- Procesy (transakce) v systému (příchod procesu do systému, jeho činnost, odchod)
- Stav obslužných linek a front u zařízení
- Průběh obsluhy transakcí v zařízeních

Poznámka

Aproximace trvání doby obsluhy exponenciálním rozložením pravděpodobnosti přináší podstatné zjednodušení.
Předpokládáme, že pravděpodobnost ukončení obsluhy v průběhu krátkého časového intervalu je konstantní a nezávisí na tom, jak dlouho již obsluha probíhala.

Typy obslužných linek

Podle kapacity rozlišujeme:

$kapacita = 1$	Zařízení (*Facility*)
$kapacita > 1$	Sklad (*Store*)

Modelujeme-li více zařízení stejného typu, pak:

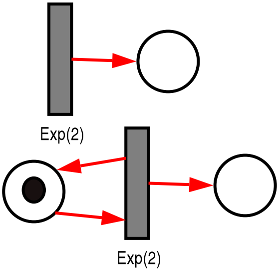
- každé zařízení má vlastní frontu $\rightarrow$ pole zařízení
- k zařízením vede jediná fronta $\rightarrow$ sklad nebo pole zařízení se sdílenou frontou

Příklad: Samoobsluha

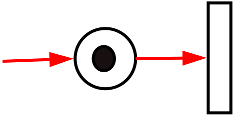
- vozíky – sklad x vozíků (jedna fronta)
- dva prodavači – např. dvě zařízení se sdílenou frontou
- pět pokladen – pět samostatných zařízení (ke každé je zvláštní fronta)

Příchod a odchod transakce

Generování příchodu transakcí (procesů) do systému:



Transakce opouští systém:

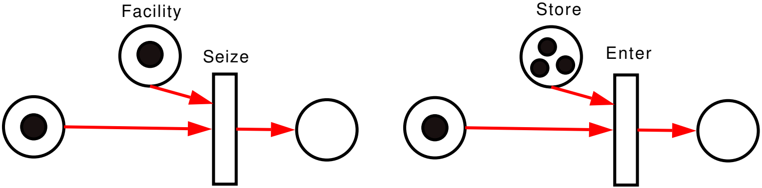


Obsazení zařízení

Obslužná linka s kapacitou 1 (Zařízení, Facility) je volná nebo obsazená.

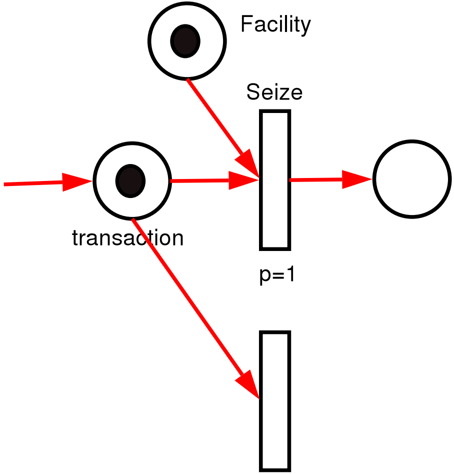
Obslužná linka s kapacitou $N > 1$ (Sklad, Store) má obsazeno 0 až N míst.

Příklad: Obsazení zařízení (Seize) a skladu (Enter)



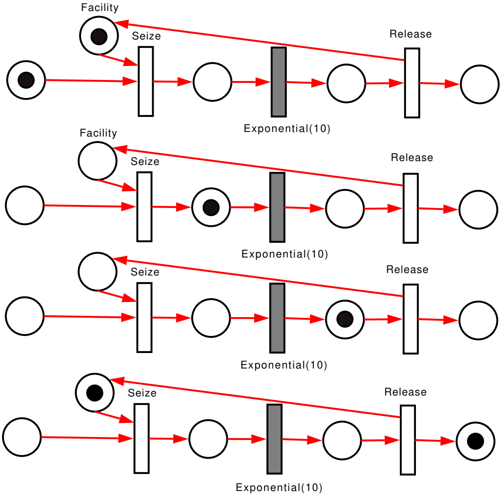
Neblokující obsazení zařízení

Transakce přistupuje k zařízení, ale nechce čekat ve frontě:



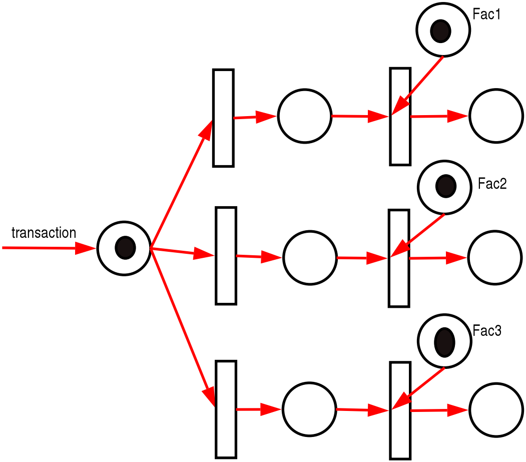
Příklad: Přehled základních operací

Obsazení linky, vykonání obsluhy a uvolnění linky



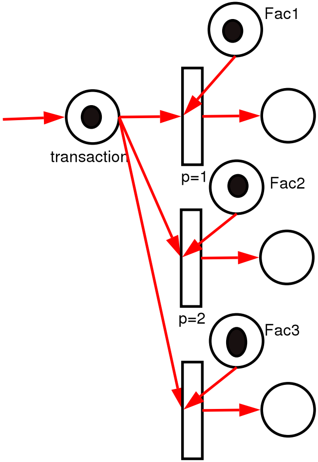
Náhodný výběr zařízení

Transakce náhodně vybírá jedno ze zařízení



Výběr s prioritou zařízení

Transakce přistupuje prioritně k jednomu ze zařízení



Příklad I – Zadání

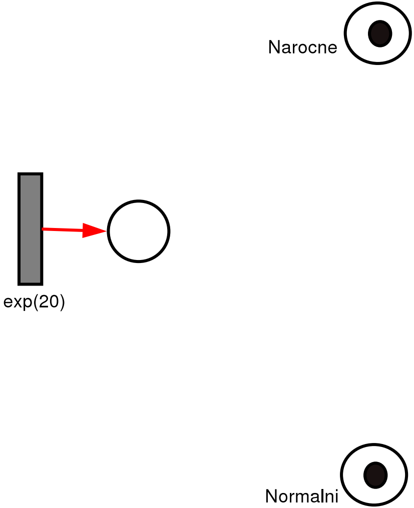
Do opravářské dílny přicházejí zákazníci v intervalech daných exponenciálním rozložením pravděpodobnosti se střední hodnotou 20 minut.

V dílně jsou dva opraváři: jeden zpracovává normální zakázky a druhý náročné zakázky. Každá třetí zakázka je náročná. Vyřízení normální zakázky trvá 15 minut s exp. rozložením pravděpodobnosti, vyřízení náročné zakázky zabere 45 minut exp. Zákazník čeká na vyřízení své zakázky a pak systém opouští.

Modelujte systém pomocí stochastické Petriho sítě.

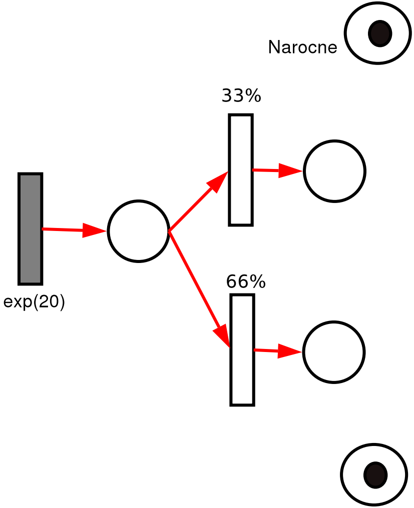
Příklad I – pokračování

Struktura systému:



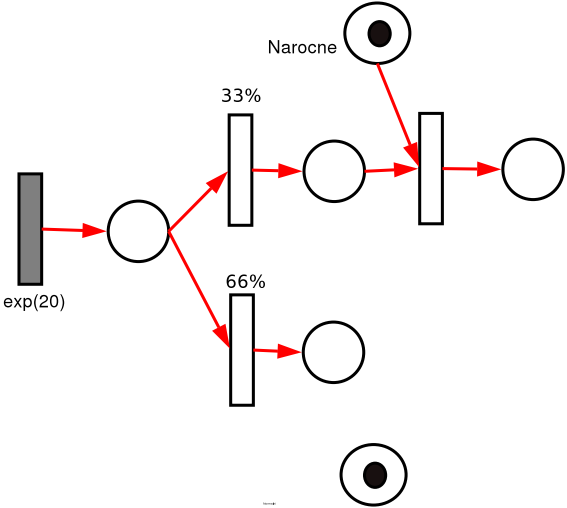
Příklad I – pokračování

Struktura systému:



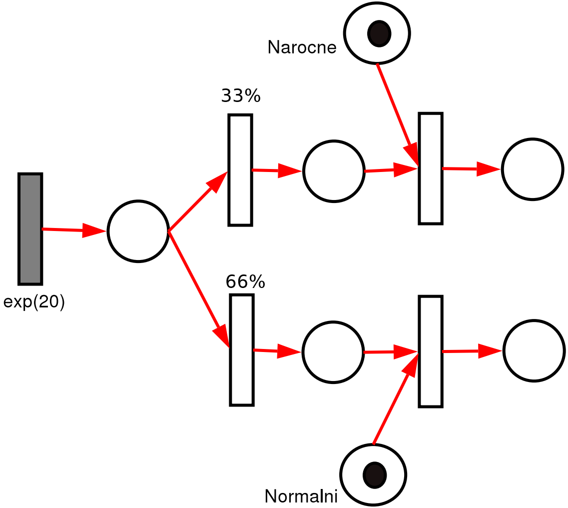
Příklad I – pokračování

Struktura systému:



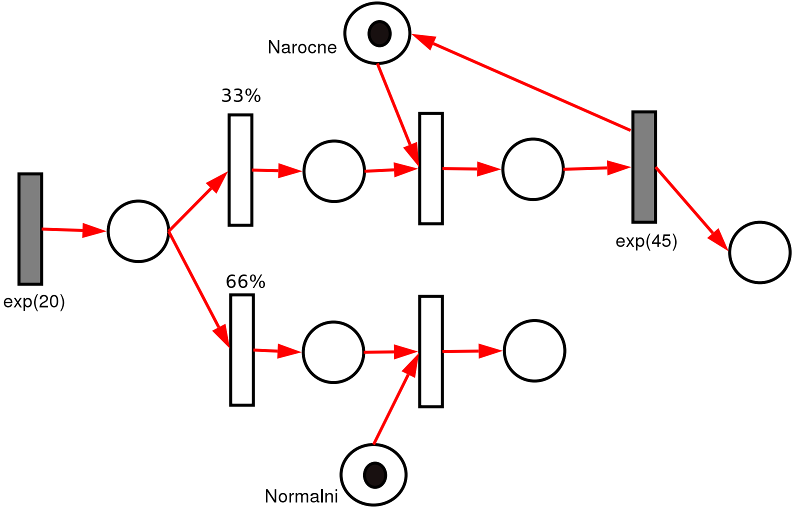
Příklad I – pokračování

Struktura systému:



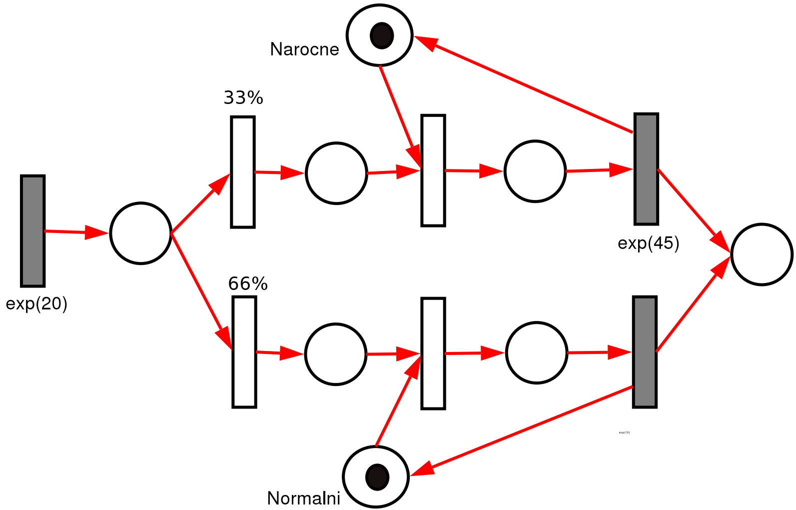
Příklad I – pokračování

Struktura systému:



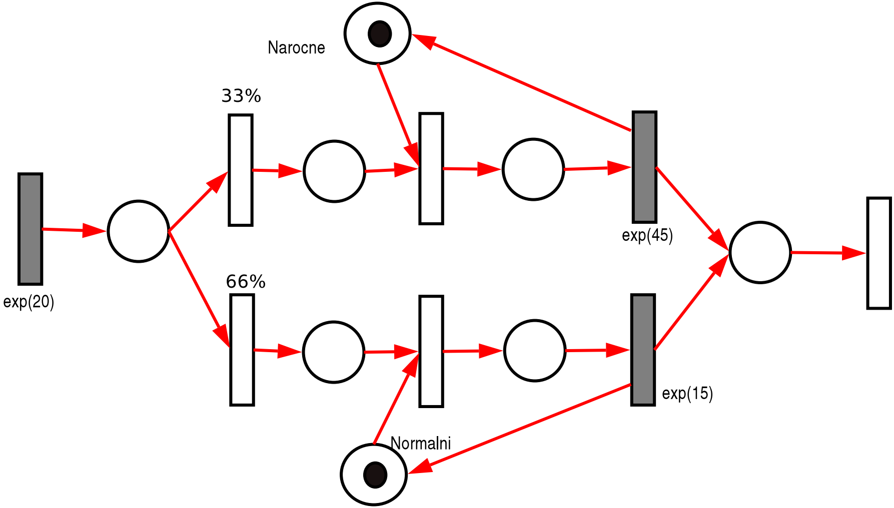
Příklad I – pokračování

Struktura systému:



Příklad I – pokračování

Model zadaného systému ve formě stochastické Petriho sítě:



Příklad I – model v SIMLIB

```
Facility narocne("Narocne");
Facility normalni("Normalni");

class Zakaznik : public Process {
   void Behavior() {
     if (Uniform(0,100) <= 33) {
        Seize(narocne);          // obsazení zařízení
        Wait(Exponential(45));
        Release(narocne);        // uvolnění
     } else {
        ...
     }
   }
};
```

Příklad I – model v SIMLIB

```
class Generator : public Event {
   void Behavior() {
      (new Zakaznik)->Activate();
      Activate(Time + Exponential(20));
   }
};

int main()          // popis experimentu
{
   Init(0, 1000);   // doba simulace
   (new Generator)->Activate();
   Run();           // start simulace
}
```

Přehled: diskrétní část SIMLIB

Prostředky pro diskrétní modelování:

Process – báze pro modelování procesů
Event – báze pro modely událostí
Facility – obslužná linka – výlučný přístup
Store – obslužná linka s kapacitou
Queue – fronta
Statistiky – sada tříd pro sběr a uchování statistik

Poznámka: Podrobnosti viz WWW dokumentace

Obecná struktura modelu

```
#include "simlib.h"

<deklarace zařízení>

<deklarace tříd - procesy, události>

<popis simulačního experimentu>
```

Simulační model v SIMLIB/C++ je program v jazyce C++. Všechny konstrukce/knihovny jazyka C++ jsou tedy použitelné.

Popis simulačního experimentu

```
int main() {
  <příkazy1>    // základní inicializace
                // například SetOutput("soubor");
  Init(<počáteční čas>, <koncový čas>);
                // inicializace simulátoru a m. času
  <příkazy2>    // inicializace modelu
                // například vytvoření objektů
  Run();        // běh simulace
  <příkazy3>    // zpracování výsledků
                // například tisk statistik
}
```

Sekvenci `Init(t0,t1); ...; Run(); ...;` lze libovolně opakovat.

Modelový čas

Modelový čas je reprezentován proměnnou:

```
double Time;
```

Do proměnné `Time` nelze zapisovat přiřazovacím příkazem. Zápis:

```
Time = 10;
```

vyvolá chybu při překladu.

Posun času řídí výhradně jádro simulátoru.

`Init(t0,t1);` nastaví počáteční čas na `t0`.

Generátory pseudonáhodných čísel

```
double Random();
-- rovnoměrné rozložení, R(0,1)

double Uniform(double L, double H);
-- rovnoměrné rozložení, R(L,H)

double Exponential(double E);
-- exponenciální rozložení se středem E

double Normal(double M, double S);
-- normální rozložení se středem M a rozptylem S

...
```

Použití objektů

OOP – třídy a instance (objekty)

- OOP vzniklo pro účely modelování a simulace (Simula 67)
- Abstrakce, hierarchie, zapouzdření, modularita; paralelnost, typování, perzistence a souvislosti
  (více v přednášce o simulačních jazycích)

Popis události

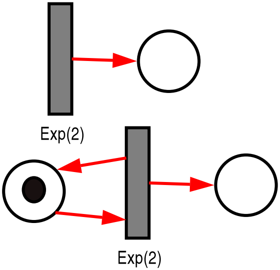
Událost je jednorázová (nepřerušitelná) akce provedená v určitém modelovém čase. V SIMLIB je vždy odvozena od abstraktní třídy `Event` (musíme definovat metodu `Behavior()`).

Často jsou nutné periodické události — událost naplánuje sama sebe:

```
class Udalost : public Event {
   void Behavior() {
      // ... příkazy události
      Activate(Time + e); // periodicky aktivovat
   }
};
// Plánování události:
(new Udalost)->Activate();  // plánuje na čas Time
(new Udalost)->Activate(t); // čas t (pozor na t<Time)
```

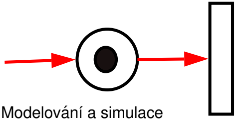
Příchod a odchod transakce

Generování transakcí (procesů):




```
class Gener : public Event {
   void Behavior() {
      (new Proc)->Activate();
      Activate(Time+Exponential(2));
   }
};
int main() {
  Init(t0, t1);
  (new Gener)->Activate();
}
```

Transakce opouští systém:



```
class Proc : public Process {
   void Behavior() {
      ...
   } // implicitně opouští systém
}
```

Popis procesu

Procesy (transakce) jsou odvozeny z abstraktní třídy `Process`.

```
class Transakce : public Process {
  public:
    Transakce( parametry ) {  // konstruktor
       // nepovinný popis inicializace procesu
    }
    void Behavior() {
       // popis chování procesu
    }
};
```

Po aktivaci procesu se volá metoda `Behavior` (chování). Provádění metody je přerušeno při čekání:

- ve frontě u zařízení (v rámci `Seize, Enter`)
- při explicitním `Wait(dt);` (abstrakce nějaké činnosti trvající *dt*)

Plánování procesu

Proces se provádí jako posloupnost událostí – např.:

```
void Behavior() {
  ...
  Wait(3);
  ... // pokračování
}
```

Proces čeká 3 časové jednotky.

Při simulaci to znamená, že se naplánuje další jeho pokračování na čas $Time + 3$ (příkazem `Activate(Time+3)`).

Aktivační záznam této události je uložen do kalendáře a proces je přerušen (a spustí se první plánovaná akce v kalendáři).

Poznámka: `Passivate()` — pozastaví na neurčito.

Kalendář událostí a algoritmus řízení simulace

Kalendář je uspořádaná datová struktura uchovávající aktivační záznamy budoucích událostí.

- Každá naplánovaná budoucí událost (”next event”) má v kalendáři záznam $[(acttime_i, priority_i, event_i), ...]$
- Kalendář umožňuje výběr prvního záznamu s nejmenším aktivačním časem a vkládání/rušení aktivačních záznamů.

Algoritmus řízení diskrétní simulace typu ”next-event”

```
Inicializace času, kalendáře, modelu, ...
while( Kalendář je neprázdný ) {
   Vyjmi první záznam z kalendáře
   if ( Aktivační čas události > T_END )
       Konec simulace
   Nastav čas na aktivační čas události
   Proveď popis chování události
}
```

Kvaziparalelismus a diskrétní simulace

Při simulaci v jednom okamžiku běží jen jeden proces (`Process::Behavior()`). Ostatní jsou pozastaveny — čekají ve frontách nebo jsou registrovány v kalendáři (*Pending Event Set, PES*).

Proto nemůže být aktivní proces nedobrovolně přerušen a v době svého běhu má teoreticky neomezený přístup ke všem zdrojům (proměnným programu).

Provádění procesu je přerušeno až na jeho vlastní žádost (viz tzv. kooperativní multitasking).

Poznámka:
Implementace přepínání procesů v SIMLIB/C++.

Vytvoření, registrování a zrušení procesu

Vytvoření instance třídy:

```
Transakce *t = new Transakce;
```

Plánování (re)aktivace procesu do kalendáře:

```
t->Activate(tm);
```

Aktivuje se v čase *tm* (implicitně je to $tm = Time$, tj. okamžitě).

Zrušení procesu i jeho registrace ve všech strukturách (fronty, kalendář):

```
t->Cancel();  // také lze použít  delete t;
```

Suspendování běžícího procesu:

```
Passivate();
```

Pro události lze použít pouze `Activate` a `Cancel`.

Příklad: Timeout – přerušení čekání ve frontě

```
class Timeout : public Event {
    Process *Id;
  public:
    Timeout(Process *p, double dt): Id(p) {
        Activate(Time+dt); // kdy bude
    }
    void Behavior() {
      Id->Cancel(); // zrušení procesu ...
      Cancel();     // a zrušení této události
    }
};
class MProc : public Process {
    void Behavior() {
      Timeout *t = new Timeout(this, 10);
      Seize(F);    // možné čekání ve frontě
      delete t;    // jen když nebyl timeout
      ...
    }
```

Čekání procesu

Explicitní: pozastavení procesu příkazem `Wait(expr)` — do kalendáře naplánuje událost reaktivace procesu na čas $\mathit{Time} + \mathit{expr}$.

Implicitní: proces může čekat ve frontě po dobu neurčitou (např. při přístupu k zařízením typu `Facility` a `Store`):

```
Facility F("F");
Store S("S",100);        // kapacita 100 míst

  Seize(F);      // před obsazením může čekat ve frontě
  Wait(5);       // F "pracuje" 5 čas. jednotek
  Release(F);    // uvolní zařízení

  Enter(S, 3);   // zabere 3 místa ve skladu nebo čeká
  Wait(50);      // ...
  Leave(S, 1);   // uvolní 1 místo
```

Priorita procesu

Proces má atribut `Priority`, který ovlivňuje jeho řazení do front.

```
class MProc : public Process {
  // ...
public:
  MProc() : Process( PRIORITA1 ) { };
  void Behavior() {
    Priority = 3; // změna priority
    Seize(F);
    Priority = 0; // = implicitní priorita
  }
};
```

Poznámka:
Neplést s prioritou obsluhy při obsazování zařízení!

Fronty — třída *Queue*

```
Queue q;
 ...
  void Behavior() { // popis chování procesu
    q.Insert(this); // vloží se do fronty
    Passivate();    // suspenduje se
    ...
  }
```

Jiný proces (nebo událost) může z fronty vybírat:

```
  ...
  if (!q.Empty()) {
    Process *tmp = q.GetFirst();
    tmp->Activate(); // aktivace
  }
```

Zařízení (*Facility*)

Zařízení je obsaditelné procesem (výlučný přístup).

Zařízení obsahuje dvě fronty požadavků:

- (vnější) fronta čekajících požadavků
- (vnitřní) fronta přerušených požadavků

```
Fac.Seize(Proces);  // priorita obsluhy = 0
Fac.Seize(Proces, PrioritaObsluhy);
```

Je třeba rozlišovat dva typy priority v SIMLIB:

priorita procesu (řazení do front, `Priority`)

priorita obsluhy v zařízení (2. parametr metody `Seize`)

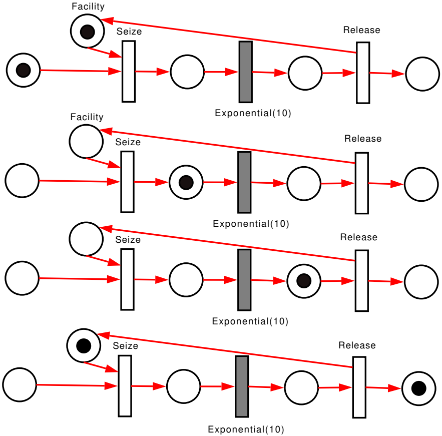
Zařízení — inicializace

```
Facility Fac2("Jmeno");
Facility Fac4("Jmeno", &moje_fronta);
Facility Fac5[10];
```

1. Jméno se tiskne ve statistikách
2. Možnost vnucení jiné fronty (např. společné s jiným zařízením)
3. Je možné kdykoli změnit (u fronty pozor na obsah):
   - `Fac5[i].SetName("Jmeno")`
   - `Fac5[i].SetQueue(moje_fronta)`

Příklad — obsazení zařízení

Obsazení linky, vykonání obsluhy a uvolnění linky




```
Facility F("Fac");
...
class P : Process {
  void Behavior() {
    ...
    Seize(F);
    Wait(Exponential(10));
    Release(F);
    ...
  }
};
```

Zařízení — priorita obsluhy

Používá se pro modelování poruch.

Jde o jiný typ priority, než je priorita procesu.

Zařízení má druhou, vnitřní frontu přerušených procesů.

```
...
Seize(Fac);
```

V obsluze je proces A se standardní prioritou obsluhy (0).

```
...
Seize(Fac, 1);
```

Jiný proces B žádá o obsluhu s vyšší prioritou obsluhy. Proces A je odstaven do vnitřní fronty a do obsluhy se dostává B.

Při uvolnění zařízení procesem B se vrátí k rozpracované obsluze proces z vnitřní fronty s nejvyšší prioritou obsluhy a dokončí se jeho obsluha.

Sklad (*Store*)

Sklad umožňuje simultánní přístup ke zdroji s určitou kapacitou (parkoviště, paměť počítače, místa v autobuse).

```
Store Voziky("Voziky", 50);
```

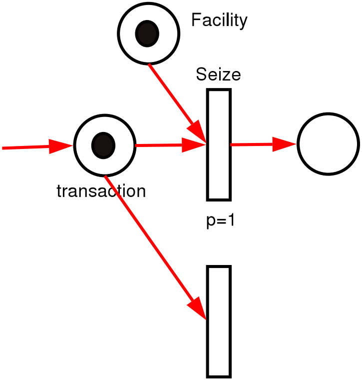
Proces přistupuje ke skladu s požadavkem na obsazení kapacity *c*. Je-li dostupná kapacita volná, přidělí se (zmenší se množství dostupné kapacity). Není-li, proces čeká ve frontě.
(Sklad nemá prioritu obsluhy.)
Proces typicky provádí operace:

```
Enter(Voziky, 1);
Leave(Voziky, 1);
```

Obdržená kapacita nesouvisí s procesem – vrátit ji může libovolný jiný proces. Při vracení se uvolní kapacita a prochází se fronta čekajících. První čekající s uspokojitelným požadavkem je obsloužen (nemusí být první ve frontě).

Zařízení — neblokující obsazení linky

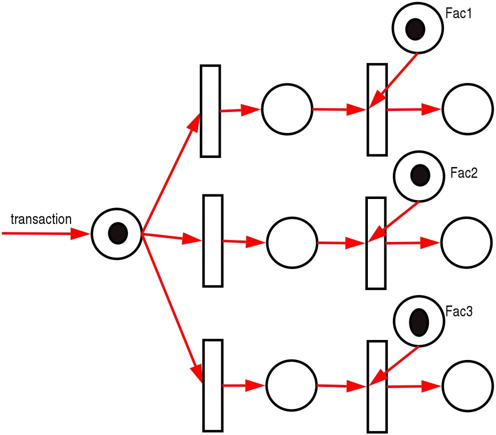
Transakce přistupuje k zařízení, ale nechce čekat ve frontě




```
Facility F("Fac");
class Proc : Process {
   void Behavior() {
      ...
      if (!F.Busy())
         Seize(F);
      else
         ...
   }
}
```

Náhodný výběr z N zařízení

Transakce přistupuje (se staví do fronty) k jednomu ze tří zařízení (náhodně vybírá)



Náhodný výběr z N zařízení

Nedeterminismus je třeba modelovat rovnoměrným rozložením.

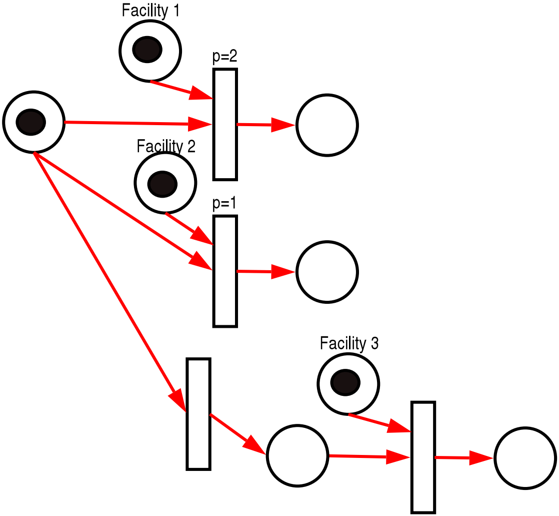
```
const int N = 3;
Facility F[N];

class Proc : Process {
    void Behavior() {
      ...
      int idx = int( N * Random() );
      Seize(F[idx]);
      ...
      Release(F[idx]);
      ...
    }
};
```

Výběr s prioritou

Transakce přistupuje k jednomu z N zařízení — vybírá první volné podle priority dané implicitně pořadím prohledávání pole (a pokud není volné žádné, vybere poslední):

```
const int N = 3;
Facility F[N];
...
    int idx;
    for(idx=0; idx < N-1; idx++)
        if(!F[idx].Busy())
            break; // první neobsazené
    Seize(F[idx]);
    ...
```

Výběr s prioritou — Petriho síť



Výběr podle délky fronty

Transakce přistupuje k zařízení s nejkratší frontou (a pokud je stejně dlouhá u více zařízení, vybere první podle pořadí prohledávání):

```
const int N = 10;
Facility F[N];

    int idx=0;
    for (int a=1; a < N; a++)
      if (F[a].QueueLen() < F[idx].QueueLen())
          idx=a;
    Seize(F[idx]);
    ...
```

Příklad: Samoobsluha

Do samoobsluhy přicházejí zákazníci v intervalech daných exponenciálním rozložením se středem 8 minut.
Každý zákazník si nejprve opatří vozík. Vozíky se koncentrují na seřadišti a je jich celkem 50. Zákazník vstoupí do prodejny a 30% jde okamžitě k pultíku s lahůdkami, kde obsluhují dvě prodavačky. Obsloužení zákazníka zde trvá 2 minuty (exponenciálně) a pak zákazník pokračuje běžným nákupem. Běžný nákup trvá 10-15 minut rovnoměrně. Nakonec přistupuje k jedné z pěti pokladen. Vybírá si pokladnu podle nejkratší fronty. Doba obsluhy u pokladny se řídí exponenciálním rozložením se středem 3 minuty. Při odchodu ze systému zákazník vrací vozík.

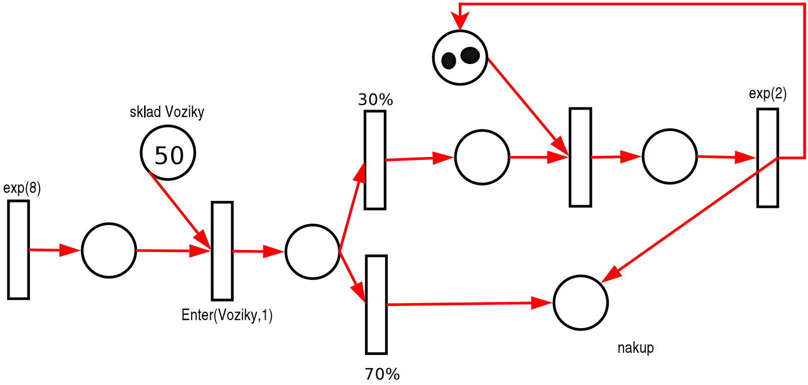
Zadání: analýza problému, model ve formě SPN, model ve formě SIMLIB

Příklad: Samoobsluha — analýza

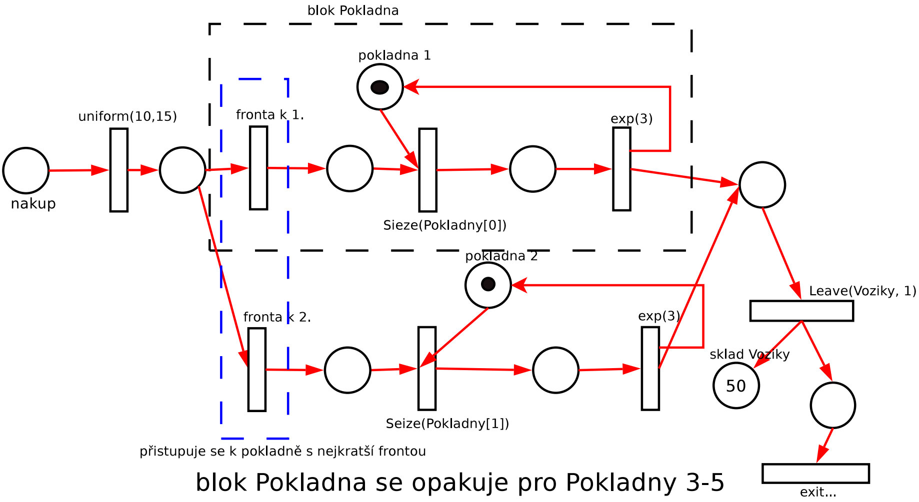
Konceptuální model:

- příchody - řídí se exponenciálním rozložením, střední hodnota je 8 minut
- proces provádí: (1) zabrat vozík, (2) 30% k lahůdkám, (3) nákup, (4) placení, (5) vrácení vozíků
- seřadiště vozíků - 50 kusů, jedna fronta, sklad
- lahůdky - jedna fronta ke dvěma prodavačkám, sklad
- pokladny - 5 pokladen, ke každé stojí zvláštní fronta

Petriho síť — první část



Petriho síť — druhá část



blok Pokladna se opakuje pro Pokladny 3-5

Modelování SHO: Statistiky

Statistiky sbíráme pro zjištění:

- vytížení zařízení (procenta doby)
- délky front, doby čekání ve frontách
- využití kapacity skladů
- celková doba, kterou transakce existuje v systému (a poměr doby užitečné činnosti/čekání ve frontě)

Statistiky:

- minimum
- maximum
- střední hodnota
- rozptyl a směrodatná odchylka

Statistiky v SIMLIB/C++

Třídy:

- `Stat`
- `TStat`
- `Histogram`

Společné operace:

- `s.Clear()` — inicializace
- `s.Output()` — tisk
- `s(x)` — záznam hodnoty `x`

Třída Stat

Objekty třídy Stat uchovávají tyto hodnoty:

- součet vstupních hodnot s_x
- součet čtverců vstupních hodnot s_x^2
- minimální vstupní hodnotu
- maximální vstupní hodnotu
- počet zaznamenaných hodnot n

Metoda `Output` tiskne některé tyto hodnoty a navíc průměrnou hodnotu a směrodatnou odchylku:

$$\sqrt{\frac{1}{n-1}(s_x^2 - n\mu^2)}$$

Třída *Stat* — příklad

```
int main() {
    Stat p;
    for (int a=0; a<1000; a++)
        p(Uniform(0, 100));
    p.Output();
}
```

```
+----------------------------------------------------------+
| STAT                                                     |
+----------------------------------------------------------+
|  Min = 0.403416                  Max = 99.9598           |
|  Number of records = 1000                                |
|  Average value = 49.8424                                 |
|  Standard deviation = 28.8042                            |
+----------------------------------------------------------+
```

Třída *TStat*

Objekty třídy `TStat` sledují časový průběh vstupní veličiny. Používají se k výpočtu průměrné hodnoty vstupu (např. délky fronty) za určitý časový interval.
Objekty třídy TStat uchovávají tyto hodnoty:

- sumu součinu vstupní hodnoty a časového intervalu
- sumu součinu čtverce vstupní hodnoty a časového intervalu
- minimální vstupní hodnotu
- maximální vstupní hodnotu
- počet vstupních hodnot
- počáteční čas

Metoda Output tiskne kromě vybraných uložených hodnot také průměrnou hodnotu vstupu za čas od inicializace statistiky (`Clear`) do okamžiku volání metody `Output`.

Příklad histogramu

```
//Histogram("Jméno pro tisk", OdHodnoty, Krok, PocetTrid);
Histogram expo("Expo", 0, 1, 15);
  ...
  for (int a=0; a<1000; a++)
     expo(Exponential(3));

+----------------------------------------------------------+
| HISTOGRAM Expo                                           |
+----------------------------------------------------------+
| STATISTIC Expo                                           |
+----------------------------------------------------------+
|  Min = 0.00037629              Max = 24.8161             |
|  Number of records = 10000                               |
|  Average value = 2.94477                                 |
|  Standard deviation = 2.91307                            |
+----------------------------------------------------------+
```

Příklad histogramu — pokračování

```
+---------+--------+------+----------+----------+
| from    | to     |  n   |   rel    |   sum    |
+---------+--------+------+----------+----------+
|   0.000 |  1.000 | 2856 | 0.285600 | 0.285600 |
|   1.000 |  2.000 | 2042 | 0.204200 | 0.489800 |
|   2.000 |  3.000 | 1480 | 0.148000 | 0.637800 |
|   3.000 |  4.000 | 1022 | 0.102200 | 0.740000 |
|   4.000 |  5.000 |  771 | 0.077100 | 0.817100 |
|   5.000 |  6.000 |  527 | 0.052700 | 0.869800 |
|   6.000 |  7.000 |  386 | 0.038600 | 0.908400 |
|   7.000 |  8.000 |  273 | 0.027300 | 0.935700 |
|   8.000 |  9.000 |  184 | 0.018400 | 0.954100 |
|   9.000 | 10.000 |  129 | 0.012900 | 0.967000 |
|  10.000 | 11.000 |  105 | 0.010500 | 0.977500 |
|  11.000 | 12.000 |   55 | 0.005500 | 0.983000 |
|  12.000 | 13.000 |   47 | 0.004700 | 0.987700 |
|  13.000 | 14.000 |   47 | 0.004700 | 0.992400 |
|  14.000 | 15.000 |   17 | 0.001700 | 0.994100 |
+---------+--------+------+----------+----------+
```

Příklad: Samoobsluha v SIMLIB

```
#include "simlib.h"

const int POC_POKLADEN = 5;

// zařízení:
Facility Pokladny[POC_POKLADEN];
Store    Lahudky("Oddělení lahůdek", 2);
Store    Voziky("Seřadiště vozíků", 50);

Histogram celk("Celková doba v systému", 0, 5, 20);
```

Příklad: Samoobsluha — pokračování

```
class Zakaznik : public Process {
  void Behavior() {
    double prichod = Time;        // záznam času příchodu
    Enter(Voziky, 1);
    if ( Random() < 0.30 ) {      // 30% pravděpodobnost
       Enter(Lahudky, 1);
       Wait(Exponential(2));      // extra obsluha
       Leave(Lahudky, 1);
    }
    Wait(Uniform(10, 15));        // běžný nákup
    // výběr pokladny podle délky fronty:
    int i = 0;
    for (int a=1; a < POC_POKLADEN; a++)
       if (Pokladny[a].QueueLen() < Pokladny[i].QueueLen())
           i=a;
    // pokračování...
```

Příklad: Samoobsluha — pokračování

```
      // ...pokračování
      Seize(Pokladny[i]);          // u pokladny
      Wait( Exponential(3) );
      Release(Pokladny[i]);
      Leave(Voziky, 1);
      celk(Time-prichod);          // záznam času
    } // Behavior
  }; // Zakaznik

  class Prichody : public Event {
    void Behavior() {
      (new Zakaznik)->Activate();
      Activate( Time + Exponential(8) );
    }
  };
```

Příklad: Samoobsluha — dokončení

```
int main()                         // popis experimentu
{
   SetOutput("samoo.dat");
   Init(0, 1000);
   (new Prichody)->Activate();     // start generátoru
   Run();                          // běh simulace

   // tisk statistik:
   celk.Output();
   Lahudky.Output();
   Voziky.Output();
   for (int a=0; a < POC_POKLADEN; a++)
       Pokladny[a].Output();
}
```

Diskrétní simulační jazyky

Základní přehled:

- Simula67 – procesy
- Simscript – popis událostmi, ...
- SIMAN/Cinema, Arena – kombinované, bloky
- GPSS – procesy, bloky
- ...

Příklady: viz WWW

Poznámky:
SIMLIB/C++, SimPack, SimPy, ...
ns-3, OMNeT++

Shrnutí

- použití diskrétní simulace
- popis modelu (události, procesy)
- generování pseudonáhodných čísel
- systémy hromadné obsluhy
- diskrétní simulační jazyky
- implementace: fronty, kalendář událostí
  algoritmus řízení simulace "next-event"

Poznámky:
Paralelní a distribuovaná simulace
Speciální typy diskrétní simulace (číslicové systémy, ...)

Celulární automaty (CA) – úvod

- Historie: J. von Neumann, J. Conway, S. Wolfram, ...
- Princip CA
- Varianty CA: diskrétní, spojité, stochastické
- Použití:
  - Simulace prostorových dynamických sytémů v oblastech: doprava, šíření epidemie/požáru, chemie, růst krystalů, koroze, šíření vln/trhlin, sypání písku/sněhu, proudění tekutin, ...
  - Modely umělého života, evoluce
  - Grafika: generování textur, fraktálů
  - Výpočty: některé CA jsou *Turing-complete*
- Souvislosti: teorie chaosu, složitost, fraktály, přírodní CA, kryptografie, ...

Definice CA

CA je typicky diskrétní systém:

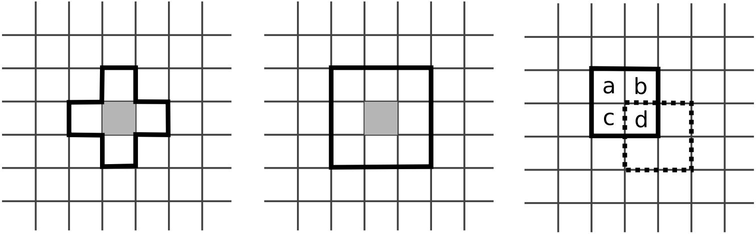
- Buňka (*Cell*): základní element, může být v jednom z konečného počtu stavů (například $\{0, 1\}$).
- Pole buněk (*Lattice*): n-rozměrné, obvykle 1D nebo 2D,
  – rovnoměrné rozdělení prostoru,
  – může být konečné nebo nekonečné.
- Okolí (*Neighbourhood*): Různé typy – liší se počtem a pozicí okolních buněk se kterými se pracuje.
- Pravidla (*Rules*): Funkce stavu buňky a jejího okolí definující nový stav buňky v čase:

$$s(t+1) = f(s(t), N_s(t))$$

Typy okolí

Závisí na rozměru prostoru a tvaru buněk.
Příklady pro 2D a čtvercové buňky:

- Von-Neumann
- Moore, Extended Moore
- Margolus

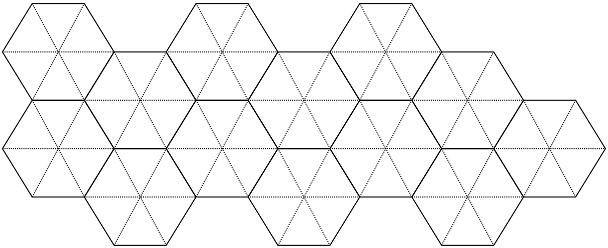


Von Neumann   Moore   Margolus

Poznámka: Existuje celá řada jiných typů okolí

Typy okolí – pokračování

- Šestiúhelníkové okolí

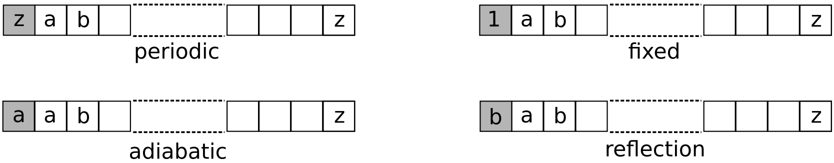


Poznámka:
Implementace: převod šestiůhelníková $\rightarrow$ čtvercová struktura

Použití: např. růst krystalů, šíření vln (FHP,...)

Okrajové podmínky

- Periodické
- Pevné (*Fixed*): konstantní hodnota
- *Adiabatic*: hodnota vedlejší buňky (= nulový gradient)
- *Reflection*: hodnota předposlední buňky



Implementace CA

Implementace uložení buněk a pravidel

- Přímá: každá buňka uložena zvlášť v poli
- Vyhledávací tabulka: jen „nenulové" buňky
- SIMD styl: více buněk v jednom int + bitové operace
- *Hash life*: cache, quad-tree, (*memoized algorithm*)
- ...

Poznámka: Snadno paralelizovatelné

Příklad1: hra Life

Hra *Life*: CA, který nastavíme na počáteční stav a spustíme.

Definice automatu pro hru Life

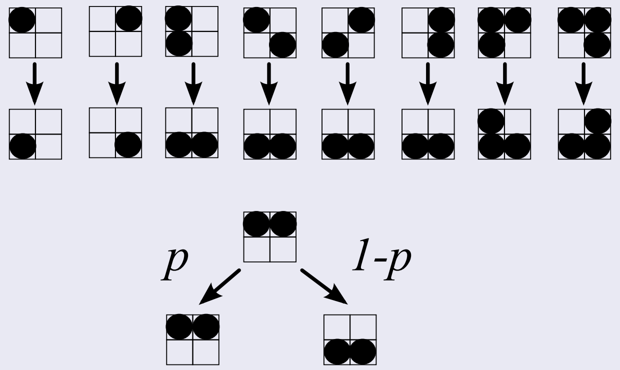
- Buňka: stavy '0' nebo '1'
- Pole buněk: dvourozměrné (2D)
- Okolí (typu *Moore*): 8 okolních buněk
- Pravidla: závislost na počtu '1' v okolí:
  - buňka '1' zůstane ve stavu '1', když má 2–3 sousedy '1'
  - buňka '0' se změní na '1', když má právě 3 sousedy '1'
  - jinak bude nový stav buňky '0'

I takto jednoduchý CA vykazuje velmi zajímavé chování – viz příklady na WWW

Příklad2: sypání písku

Sypání písku (*sand rule, sandpile* model)

- Okolí typu Margolus
- Pravidla:



Příklad3: *Ant rule*

Hypotetický mravenec (*Langton's ant*):

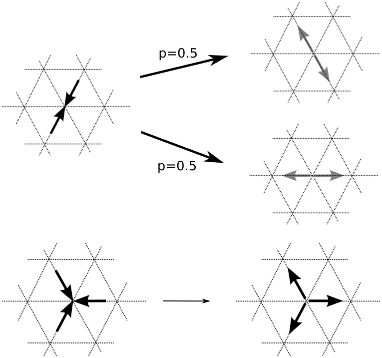
- Čtvercové pole buněk
- Buňky jsou bílé nebo šedé
- Pravidla:
  - Při příchodu na bílou buňku
    změní směr o 90 stupňů doleva a obarví ji na šedou
  - Při příchodu na šedou buňku
    změní směr o 90 stupňů doprava a obarví ji na bílou
- Vykazuje překvapivě zajímavé a složité chování

Poznámka: viz demo

Příklad4: FHP

Např. model pohybu tekutiny:

- Šestiúhelníkové okolí
- Buňky obsahují částice a jejich směr pohybu
- Pravidla viz obrázky + volný průlet v ostatních případech



Příklad5: doprava – model provozu na komunikacích

Nagel-Schreckenberg model

- Silnice je rozdělena na úseky (cca 7m)
- Úsek je buď prázdný nebo je v něm auto
- Stav auta j: rychlost ($v_j = 0, 1, ..., v_{max}$)
- Pravidla provádíme v pevném pořadí:

R1: Akcelerace – rychlost v_j zvýšíme o 1, max na v_{max}

R2: Brzdění podle vzdálenosti d_j buněk od předchozího auta $v_j = min(d_j, v_j)$

R3: Náhodná změna rychlosti na $max(v_j - 1, 0)$ s pravděpodobností p

R4: Posun auta o $v_j(t + 1)$ buněk

Poznámka: pouze minimální model, existují různé varianty.

Pravidla – obecně

Musí popisovat změnu stavu pro všechny možnosti.

$$s(t+1) = f(s(t), N_s(t))$$

- Typy pravidel:

  "legal" – z nulového vstupního stavu nesmí vzniknout nenulový stav

  "totalistic" – rozhoduje *součet* vstupních stavů

- Počet možných pravidel závisí na počtu stavů a velikosti okolí. Například pro jednorozměrné okolí, na vstupu 3 prvky se stavy 0/1 (tzv. elementární automat) existuje celkem $2^3 = 8$ možností vstupu a tedy $2^8 = 256$ všech možných funkcí/pravidel.

Reverzibilní automaty

Reverzibilní automat je systém, který neztrácí informaci při svém vývoji v čase. Proto je v každém okamžiku možno otočit běh času nazpátek a vracet se k předchozím stavům.

Například pokud definujeme nový stav buňky takto:

$$s(t+1) = f(s(t), N_s(t)) - s(t-1)$$

je možné pro libovolné f počítat předchozí stav:

$$s(t-1) = f(s(t), N_s(t)) - s(t+1)$$

Obecné vlastnosti CA

- Konfigurace CA je definována jako stav všech buněk
- Stav CA se vyvíjí v čase a prostoru podle zadaných pravidel
- Čas i prostor jsou diskretizovány
- Počet stavů buňky je konečný
- Buňky jsou identické
- Následující stav buňky závisí pouze na aktuálním stavu

Klasifikace CA

Celulární automaty můžeme rozdělit podle jejich dynamického chování do 4 kategorií:

Třídy CA

třída 1: Po konečném počtu kroků dosáhnou jednoho konkrétního ustáleného stavu

třída 2: Dosáhnou periodického opakování (s krátkou periodou) nebo zůstanou stabilní.

třída 3: Chaotické chování (výsledné posloupnosti konfigurací tvoří speciální fraktální útvary).

třída 4: Kombinace běžného a chaotického chování (například Life), nejsou reverzibilní.

Zdroj: Wolfram S.: *New Kind of Science*, Wolfram Media, 2002

Přehled implementací CA

- Možné problémy: nekonečné pole buněk, vizualizace, ...
- Existuje řada volně dostupných nástrojů.

Příklady simulátorů CA

- Golly (HashLife)
- různé Java applety – viz WWW,
- SimCell (dynamické CA),
- Xtoys (jednoduché, C, xlib),
- cage (Python),
- ...

Spojitá simulace

Obsah:

- Typické aplikace spojité simulace
- Formy popisu spojitých modelů
- Převod rovnic na blokové schéma
- Numerické metody
- Spojité simulační jazyky
- Příklady

Aplikace spojité simulace

- Elektrické a elektronické obvody
- Řízení (automatizace)
- Fyzika
- Chemie
- Astronomie (pohyb planet)
- Biologie (model srdce)
- Ekologie (rozptyl znečištění)
- ...

Poznámka: Konkrétní příklady viz WWW

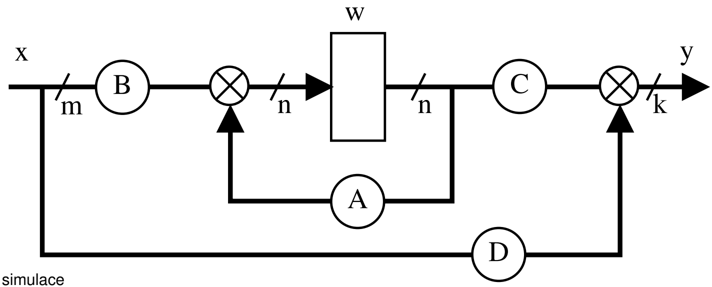
Popis spojitých systémů

- Soustavy obyč. diferenciálních rovnic (ODE)
- Soustavy algebraických rovnic
- Algebraicko-diferenciální rovnice (DAE)
- Bloková schémata
- Parciální diferenciální rovnice (PDE)
- Elektrická schémata
- ...
- Grafy signálových toků
- Kompartmentové systémy
- Systémová dynamika
- "Bond-graphs"

Soustavy dif. rovnic: maticový popis

$$
\begin{aligned}
\frac{d}{dt}\vec{w}(t) &= \mathbf{A}(t)\vec{w}(t) + \mathbf{B}(t)\vec{x}(t) \\
\vec{y}(t) &= \mathbf{C}(t)\vec{w}(t) + \mathbf{D}(t)\vec{x}(t)
\end{aligned}
$$

kde $\vec{x}$ je vektor m vstupů, $\vec{w}$ vektor n stavových proměnných, $\vec{y}$ vektor k výstupů a **A**, **B**, **C**, **D** matice koeficientů



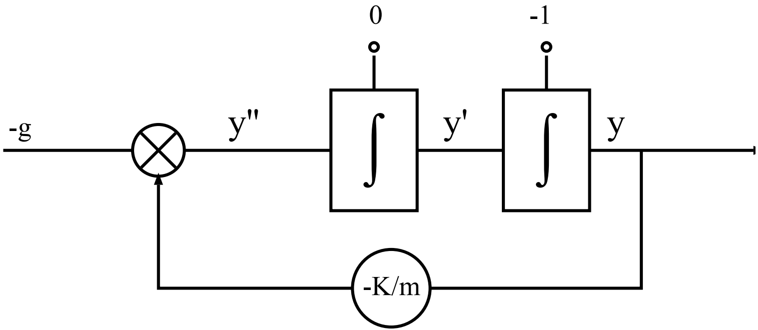
Typy dif. rovnic

Koeficienty (prvky matic **A**, **B**, **C**, **D**) mohou být:

- nezávislé na čase (stacionární systémy),
- časově proměnné,
- konstanty (lineární systémy),
- nelineární funkce (nelineární systémy).

Poznámka: Problémy při analytickém řešení

Grafový popis – bloky



Poznámka: Souvislost s analogovými počítači

Typy bloků

Funkční bloky (Bezpaměťové):

- konstanty
- T (modelový čas)
- Sin, Cos, Log, ...
- $+, \ -, \ *, \ /$
- Uživatelem definované funkce

Stavové bloky (Paměťové; mají počáteční podmínky):

- integrátory
- zpoždění
- ...

Poznámka: Hierarchie: složené bloky (i kombinované)

Převod rovnic vyššího řádu

Rovnice vyšších řádů musíme převést na soustavu rovnic prvního řádu, pro které máme vhodné numerické metody.

Metody převodu:

- Snižování řádu derivace
- Postupná integrace
- Jiné speciální metody

Poznámky:
Pozor na podmínky pro převod
Existují i num. metody pro řešení rovnic vyššího řádu

Metoda snižování řádu derivace

1. Osamostatnit nejvyšší řád derivace (viz příklad)
2. Zapojit všechny integrátory za sebe a
   ke vstupu prvního připojit pravou stranu z (1)

Podmínka: nesmí být derivace vstupů (x', x'', ...)

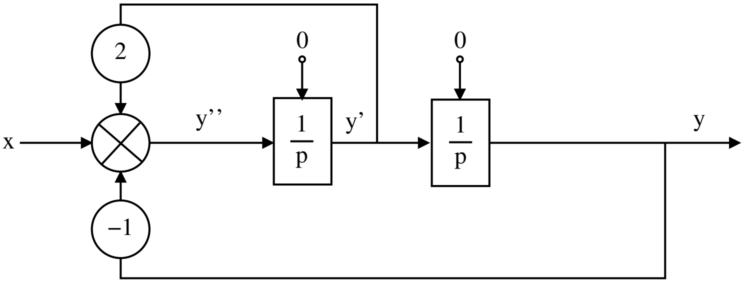
Příklad: rovnice $y'' - 2y' + y = x$

$$\begin{aligned} y'' &= 2y' - y + x \\ y' &= \int y'' \\ y &= \int y' \end{aligned}$$

Poznámky:

- Typický tvar blokového schématu
- Pozor na počáteční podmínky

Příklad: Blokové schéma



Metoda postupné integrace

Vhodná pro rovnice s derivacemi vstupů na pravé straně

1. Osamostatnit nejvyšší řád derivace
2. Postupná integrace rovnice a zavádění nových stavových proměnných
3. Výpočet nových počátečních podmínek

Podmínka: konstantní koeficienty

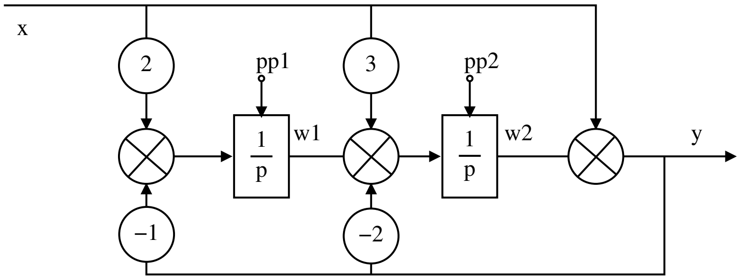
Příklad: rovnice $p^2y + 2py + y = p^2x + 3px + 2x$

$$p^2y = p^2x + p(3x - 2y) + (2x - y)$$
$$py = px + (3x - 2y) + \frac{1}{p}(2x - y), \text{ proměnná } w_1 = \frac{1}{p}(2x - y)$$
$$py = px + (3x - 2y) + w_1$$
$$y = x + \frac{1}{p}(3x - 2y + w_1), \text{ proměnná } w_2 = \frac{1}{p}(3x - 2y + w_1)$$
$$y = x + w_2$$

Metoda postupné integrace – příklad

Výsledná soustava rovnic:

$$w_1 = \frac{1}{p}(2x - y), \qquad w_1(0) = y'(0) - x'(0) - 3x(0) + 2y(0)$$
$$w_2 = \frac{1}{p}(3x - 2y + w_1), \qquad w_2(0) = y(0) - x(0)$$
$$y = x + w_2$$



Numerické metody

Při spojité simulaci potřebujeme metody pro:

- řešení ODR 1. řádu (*Initial Value Problem*)
- řešení algebraických rovnic (hledání kořenů – řešení tzv. rychlých smyček)
- (také řešení PDR atd., ale ne v tomto předmětu)

Poznámky:

- Existuje celá řada metod (viz např *Netlib*)
- Je nutné znát vlastnosti numerických metod

Metody pro řešení ODR 1.řádu

Hledáme řešení rovnice

$$y' = f(t, y)$$

které má tvar:

$$y(T) = y_0 + \int_0^T f(t, y)dt$$

Na počítači je řešení aproximováno v bodech $t_0, t_1, t_2, ...t_n$

Integrační krok: $h_i = t_{i+1} - t_i$

Poznámka: Integrační krok nemusí být konstantní

Princip, klasifikace

Obecný princip metody *N*-tého řádu:

1. Aproximace $y(T)$ polynomem N-tého stupně (Taylorův rozvoj)
2. Extrapolace – výpočet $y(t+h)$

Klasifikace metod:

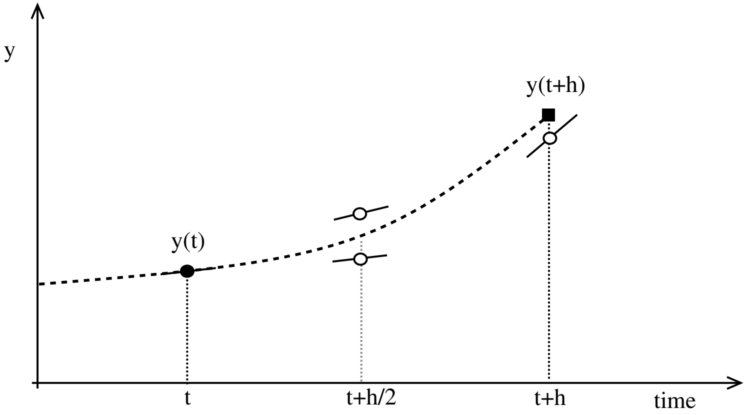
- Jednokrokové – vychází jen z aktuálního stavu
- Vícekrokové – používají historii stavů/vstupů

Další možné dělení:

- Explicitní – výsledek získáme dosazením do vzorce
- Implicitní – vyžadují řešení algebraických rovnic v každém kroku

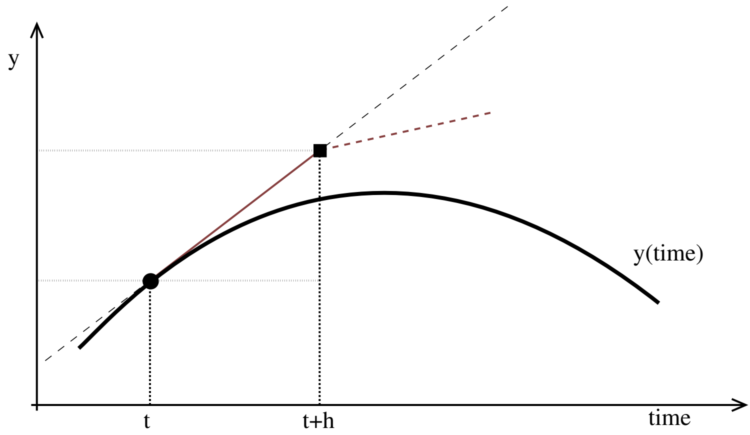
Jednokrokové metody

Princip jednokrokových metod (RK4):



Eulerova metoda — princip

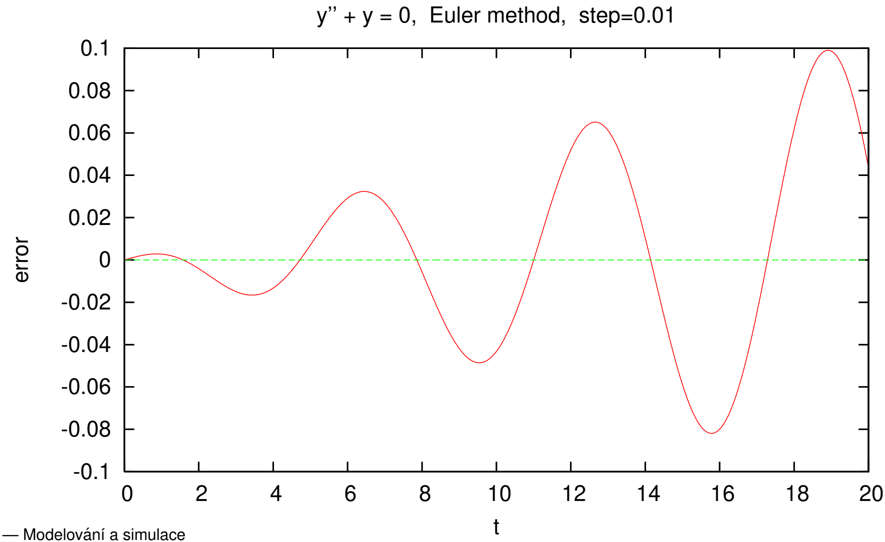
$$y(t+h) = y(t) + hf(t, y(t))$$



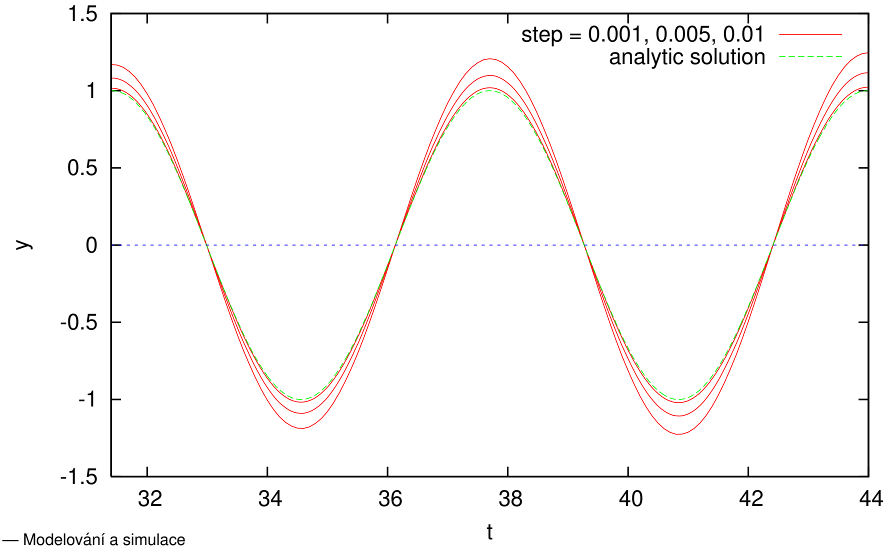
Eulerova metoda — implementace a příklad

```
double  yin[2],  y[2] = { 0.0, 1.0 },  time = 0,  h = 0.001;
void Dynamic() {          // f(t,y): výpočet vstupů integrátorů
    yin[0] = y[1];        // y'
    yin[1] = -y[0];       // y''
}
void Euler_step() {       // výpočet jednoho kroku integrace
    Dynamic();            // vyhodnocení vstupů integrátorů
    for (int i = 0; i < 2; i++)  // pro každý integrátor
        y[i] += h * yin[i];      // vypočteme nový stav
    time += h;            // posun modelového času
}
int main() { // Experiment: kruhový test, čas 0..20
    while (time < 20) {
        printf("%10f %10f\n", time, y[0]);
        Euler_step();
    }
}
```

Příklad: Absolutní chyba Eulerovy metody



Příklad: vliv velikosti kroku



Metody Runge-Kutta

Skupina metod: RK1=Euler, RK2, RK4, RK8, ...

RK2: 2. řád

$$k_1 = hf(t, y(t))$$
$$k_2 = hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2})$$
$$y(t+h) = y(t) + k_2$$

RK4: 4. řád

$$k_1 = hf(t, y(t))$$
$$k_2 = hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2})$$
$$k_3 = hf(t + \frac{h}{2}, y(t) + \frac{k_2}{2})$$
$$k_4 = hf(t + h, y(t) + k_3)$$
$$y(t+h) = y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

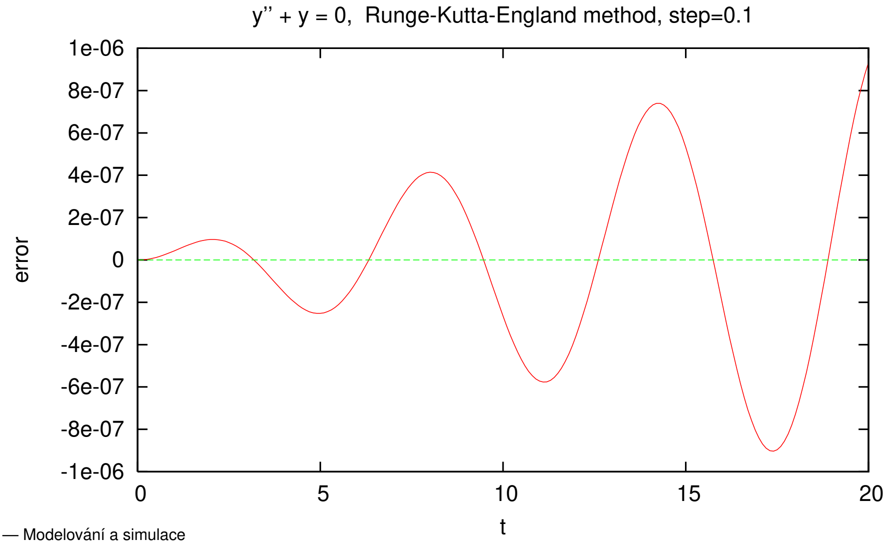
Metody Runge-Kutta — pokračování

Často používané metody — každý spojitý simulační systém obsahuje alespoň jednu RK metodu

Poznámky:

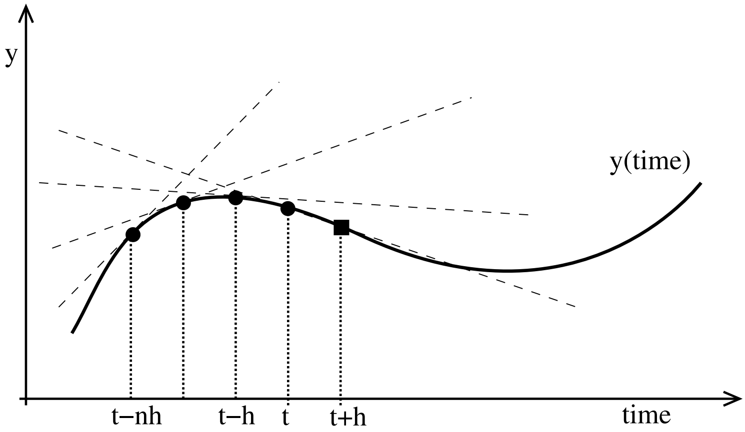
- Implementace — viz WWW
- Různé další varianty (např. Dormand-Prince 45)
- Specifikace metody tabulkou: *Butcher tableau*
- Odhad chyby
- Změna kroku na základě odhadu chyby
- Existují také implicitní metody RK — viz WWW

Přesnost metody Runge-Kutta — příklad



Vícekrokové metody – princip

Používají hodnoty zapamatované z předchozích kroků



Vícekrokové metody

Adams-Bashforth:

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$

Metody typu *prediktor/korektor* zpřesňují výsledek:

Adams-Bashforth-Moulton:

$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$$

Poznámky:

Problém startu metody (řešení: např. použití jednokrokové metody pro první kroky).

Existují i samostartující vícekrokové metody.

Vlastnosti integračních metod

Chyba metody:

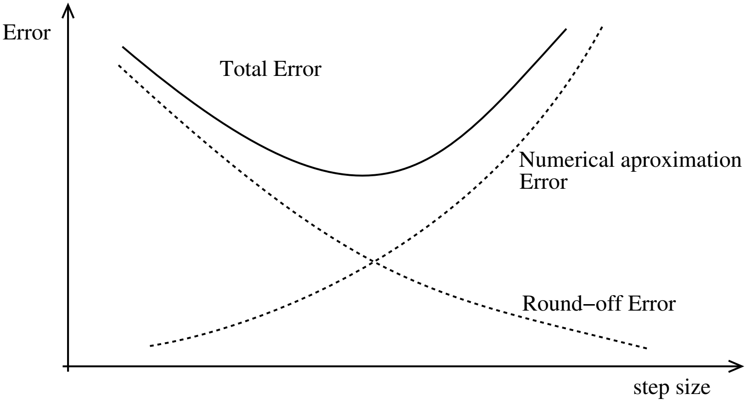
- Lokální chyba (v jednom kroku)
  - Chyba zaokrouhlovací (*round-off error*)
  - Chyba aproximace (*truncation error*)
- Akumulovaná (globální) chyba – maximum odchylky od přesného řešení.

Stabilita metody:

- Stabilita numerického řešení
- Vliv velikosti integračního kroku na stabilitu

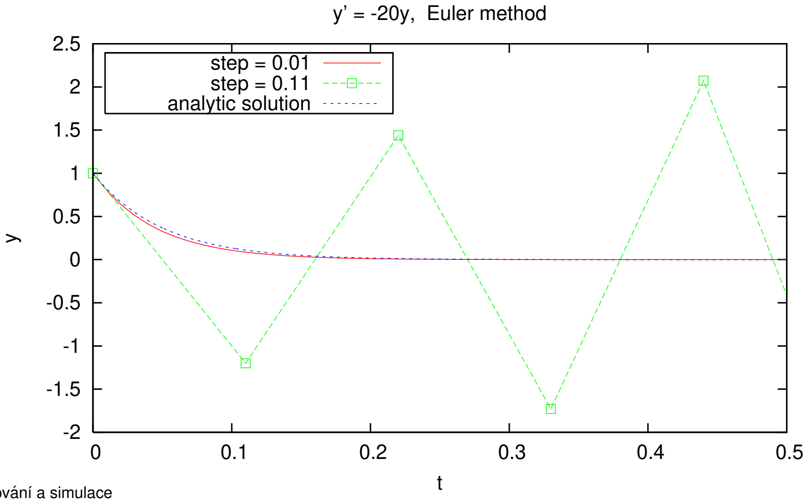
Poznámka: Příklady nestability/nepřesnosti

Lokální chyba numerické metody



Stabilita numerické metody — příklad

Rovnice: $y' = -20y$, počáteční podmínka: $y(0) = 1$



Tuhé systémy (stiff systems)

Problém: velmi rozdílné časové konstanty

Příklad tuhého systému/rovnice:

$$y'' + 101y' + 100y = 0$$

Řešení:

- Použití speciálních integračních metod (implicitních)
- Zkrácení kroku – často nelze (akumulace chyb, malá efektivita výpočtu)

Poznámky: Koeficient tuhosti, explicitní/implicitní metody, oblast stability, A-stabilita, atd. (Podrobnosti viz literatura.)

Výběr integrační metody

Neexistuje univerzální (nejlepší) metoda.

- Obvykle vyhovuje některá varianta metody Runge-Kutta 4. řádu.
- Nespojitosti ve funkci $f(t, y)$ snižují efektivitu vícekrokových metod (časté startování).
- Tuhé systémy vyžadují speciální implicitní metody.
- Pro ověření přesnosti výsledků je třeba vyzkoušet různé integrační metody nebo různé velikosti kroku.
- Existují meze velikosti kroku (viz stabilita, přesnost).
- Některé metody umí tzv. "*dense output*" (interpolaci výsledného průběhu uvnitř kroku).

Příklad: Systém dravec–kořist

Rovnice systému dravec-kořist (*Lotka-Volterra*):

$$\frac{dx}{dt} = k_1 x - k_2 xy$$

$$\frac{dy}{dt} = k_2 xy - k_3 y$$

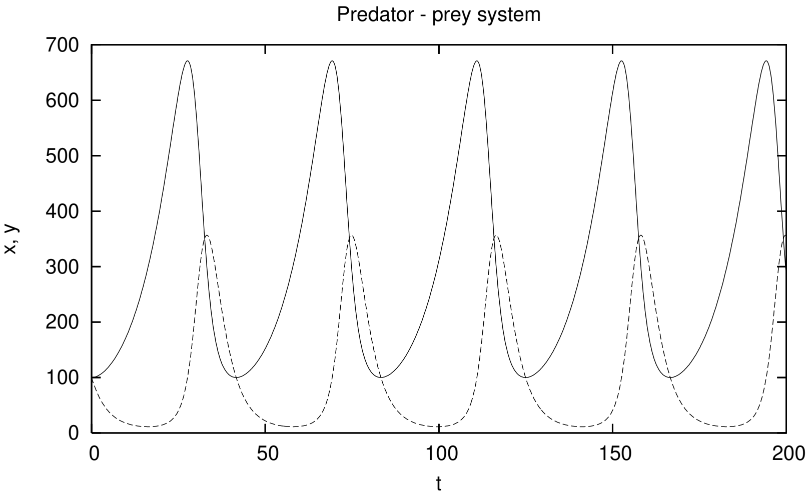
kde:

- x množství kořisti
- y množství dravců

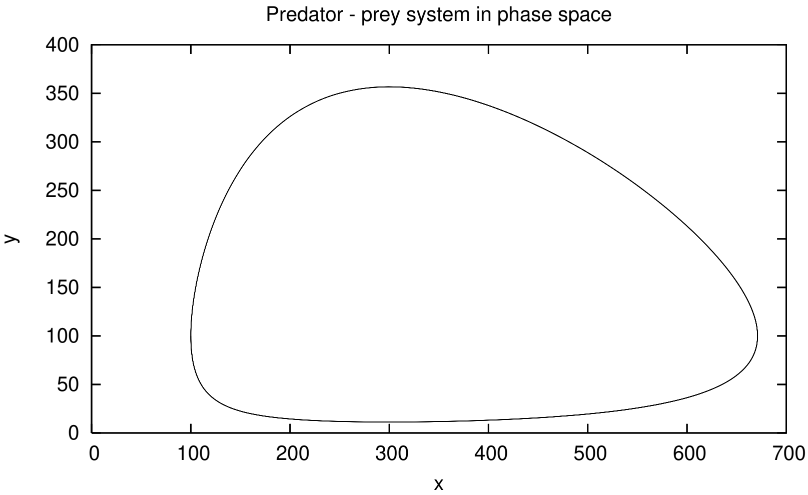
Zobrazení výsledků:

- v čase
- ve fázové rovině (*phase space*)

Příklad: dravec–kořist, zobrazení v čase



Příklad: dravec–kořist, fázová rovina



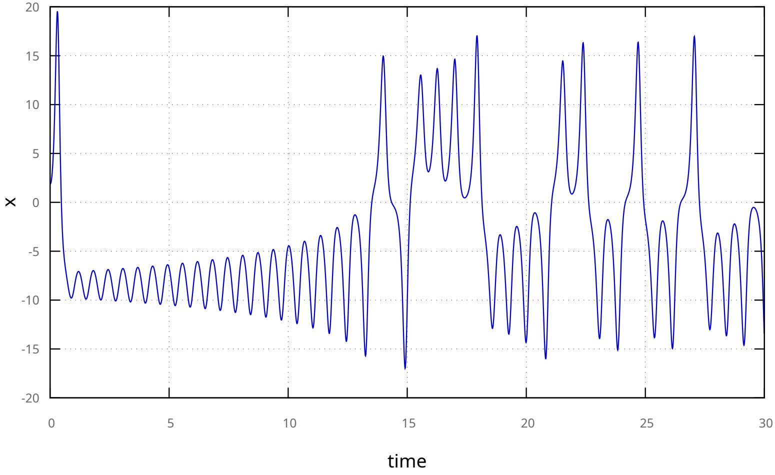
Příklad: chaos (*Lorenz equations*)

- Nelineární diferenciální rovnice

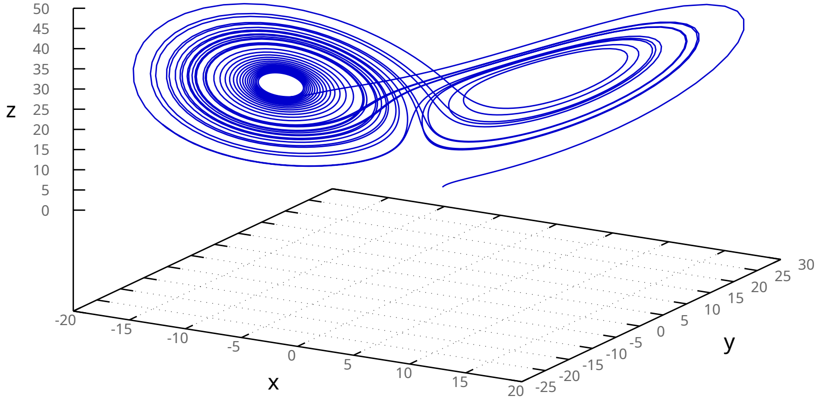
$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta * z$$

- Parametery: $\sigma = 10 \quad \rho = 28 \quad \beta = 8/3$
- Chaotické chování
- Příklad $x(0) = 2 \quad y(0) = 1 \quad z(0) = 1$

Příklad: chaos – výsledky



Příklad: chaos – výsledky 3D



Spojité simulační jazyky

Nástroje na popis modelu + popis experimentů

Algoritmus řízení spojité simulace:

1. Inicializace (nastavit počáteční stav)
2. Cyklus dokud není konec simulace:
   1. Pokud je vhodný čas provedeme výstup (tisk)
   2. Vyhodnocení derivací a výpočet nového stavu
   3. Posun modelového času
3. Ukončení, výstup

Poznámky: Pořadí vyhodnocování. Přesné dokročení na koncový čas.

Problém uspořádání funkčních bloků

Výpočet závisí na pořadí vyhodnocování *funkčních* bloků

Příklad:

```
a = fa(1,b)    # b ještě není vypočítáno = chyba
b = fb(a)
c = fc(a,b)
...
```

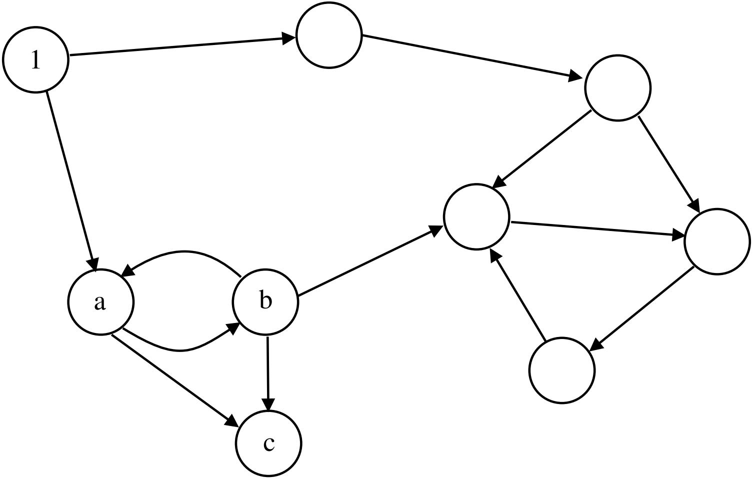
Řešení:

- Seřazení funkčních bloků (*topological sort*)
- Vyhodnocování na žádost (viz SIMLIB/C++)

Poznámka: Pamět'ové bloky (integrátory) mají oddělený vstup a výstup, proto jsou nezávislé na pořadí vyhodnocování.

Řazení funkčních bloků

Princip algoritmu (hledání silných komponent grafu):



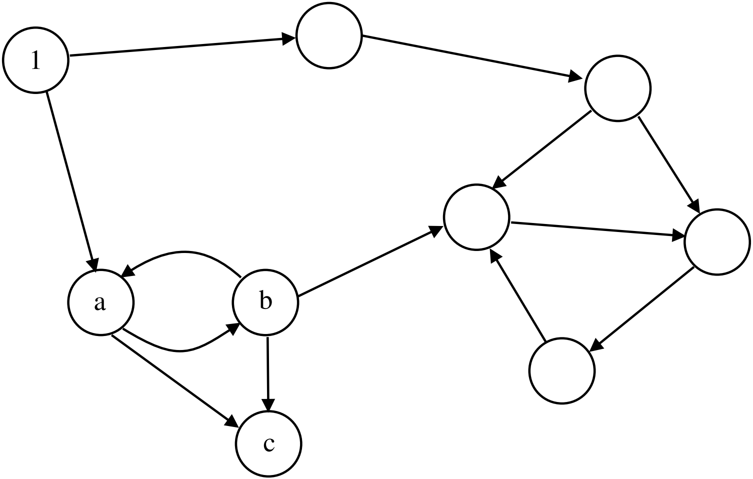
Rychlé smyčky

Problém: cyklus v grafu závislostí funkčních bloků
(způsobeno například příliš vysokou úrovní abstrakce)

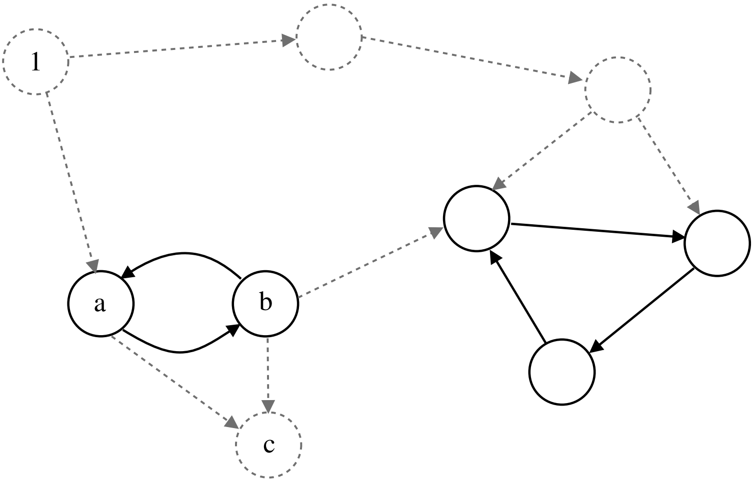
Řešení:

- Rozpojení cyklu speciálním blokem, který (například iteračně) řeší algebraické rovnice.
  Metody: půlení intervalu, Regula-falsi, Newtonova, ...
- Přepracování modelu na model bez smyček,
  například zpřesnění modelu (vložení integrátoru).

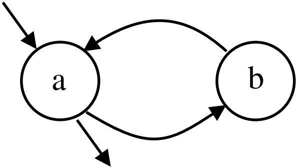
Rychlé smyčky — obrázek 1

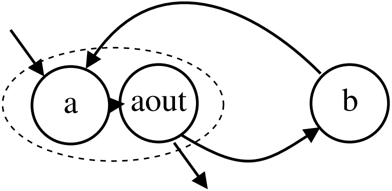


Rychlé smyčky — obrázek 2



Rychlé smyčky — možné řešení





Parciální diferenciální rovnice (PDR, *PDE*)

Obsahují derivace podle *více proměnných* (např. prostorových souřadnic)

Řešení: diskretizace v prostoru = nahrazení prostorových derivací diferencemi (metoda přímek)

Příklad: kmitající struna — řešení wiz WWW

$$\frac{\partial^2 y}{\partial t^2} = a\frac{\partial^2 y}{\partial x^2}$$

Počáteční podmínky: $y(x,0) = -\frac{4}{l^2}x^2 + \frac{4}{l}x$ a $y'(x,0) = 0$
Okrajové podmínky: $y(0,t) = y(l,t) = 0$
Diskretizace:

$$\left.\frac{\partial^2 y}{\partial x^2}\right|_{x_i} = \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2}$$

Shrnutí

- Oblasti použití spojité simulace
- Popis modelu
- Numerické metody a jejich základní vlastnosti
- Jazyky, implementace, problémy
- Nároky na výkon počítače

Poznámky: Paralelní simulace, superpočítače

Spojité simulační nástroje – přehled

- Matlab/Simulink (R), SciLab, GNU Octave
- Modelica: Dymola (R), OpenModelica
- ...
- SIMLIB/C++
- SciPy
- GNU Scientific Library
- více viz odkazy na WWW

Poznámka: Netlib, SUNDIALS, ...

SIMLIB/C++

C++ knihovna pro spojitou i diskrétní simulaci

Přehled prostředků pro spojitou simulaci:

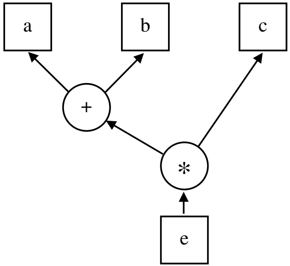
- Globální proměnné (typicky jsou pouze pro čtení): `StepSize`, `MinStep`, `MaxStep`, ...
- Bloky: `Integrator`, `Constant`, ...
- Blok reprezentující modelový čas: `T`
- Odkaz na blokový výraz: `Input` (a blok `Expression`)

Doplňky: kombinovaná simulace, 2D, 3D, fuzzy, optimalizace

Blokové výrazy v SIMLIB/C++

- Automatická konstrukce výrazových stromů (používá přetěžování operátorů v C++)
- Metoda bloku `double Value()` vyhodnotí vstupy voláním jejich metody `Value` a vrátí výsledek

Příklad: `Expression e = (a + b) * c`



Poznámka: Pozor: Blok `T` reprezentuje modelový čas, protože proměnnou `Time` nelze použít.

SIMLIB – typy bloků

Třídy definované v SIMLIB/C++:

- Konstanty, parametry, proměnné:
  `Constant`, `Parameter`, `Variable`
- Funkční bloky:
  `Function`, `Sin`, `Exp`, `Max`, `Sqrt`, `Abs`, ...
  `Lim`, `DeadZone`, `Frict`, ...
  pro blokové výrazy: `_Add`, `_Mul`, ...
- Stavové bloky:
  - `Integrator`
  - Nelinearity se stavem: `Hysteresis`, `Relay`, `Blash`

Poznámka: `Relay` přesně detekuje okamžik přepnutí

SIMLIB – popis experimentu

Sledování stavu modelu:

- třída `Sampler` – periodické volání funkce
- funkce `SetOutput(filename)` – přesměrování výstupu
- funkce `Print(fmt,...)` – tisk typu `printf`

Nastavení parametrů simulace:

- krok – `SetStep(minstep,maxstep)`
- požadovaná přesnost – `SetAccuracy(abs,rel)`
- integrační metoda – `SetMethod(name)`
  (Metody: "abm4", "euler", "rke"(default), "rkf3", "rkf5", "rkf8")

Řízení simulace:

- `Init(t0,t1)`, `Run()`
- `Stop()` – ukončení aktuálního experimentu (běhu)
- `Abort()` – ukončení programu

SIMLIB – příklad

```
// kmitání kola (verze 2 - několik experimentů)
// popis systému: y'' = ( F - D * y' - k * y ) / M
//                 nulové počáteční podmínky
#include "simlib.h"
struct Kolo {                          // popis systému
   Parameter M, D, k;
   Integrator v, y;
   Kolo(Input F, double _M, double _D, double _k):
      M(_M), D(_D), k(_k),           // parametry systému
      v( (F - D*v - k*y) / M ),      // rychlost
      y( v )  {}                     // výchylka
      void PrintParameters() {
         Print("# hmotnost = %g kg \n", M.Value());
      }
};
```

SIMLIB – příklad

```
// Parametry:
double _m=5, _d=500, _k=5e4; // implicitní hodnoty

// Objekty modelu:
Constant F(100);              // síla působící na kolo
Kolo k(F, _m, _d, _k);        // instance modelu kola

// Sledování stavu modelu:
void Sample() {
  Print("%f %g %g\n", T.Value(),k.y.Value(),k.v.Value());
}
Sampler S(Sample, 0.001);
```

SIMLIB – příklad

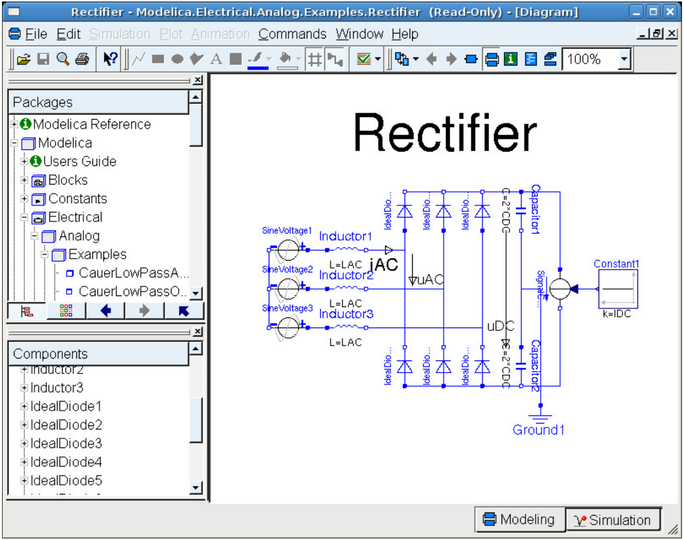
```
int main() {                    // Popis experimentů ...
 SetOutput("kolo2.dat");
 _Print("# KOLO2 - model kola (více experimentů)\n");
 for(double m=_m/2; m<=_m*5; m*=1.2) {
   k.M = m;      // parametr M
   k.D = _d;     // parametr D
   k.k = _k;     // parametr k
   k.PrintParameters();
   Print("# Time  y  v \n");
   Init(0, 0.3);                // inicializace experimentu
   SetAccuracy(1e-6, 0.001); // max. chyba integrace
   Run();                       // simulace
   Print("\n");  // oddělí výstupy experimentů (GnuPlot)
 }
}
```

Dymola, OpenModelica

- Integrované prostředí
  Modelica + GUI + knihovny
- Modelica – simulační jazyk
- Modelica library – std. knihovna
- Nástroj pro modelování fyzikálních systémů
- OpenModelica (open source)
- Dymola (komerční program, demo verze je volně k dispozici)

Poznámka: Existují i další alternativy.

Grafické rozhraní



Přehled vlastností

- Překlad Modelica $\rightarrow$ C, závislost na překladači C
- Výstupní formát kompatibilní s MatLab
- Snadné skládání modelů z komponent (knihovny)
- Snadno rozšiřitelné
- Symbolické řešení některých rovnic (algebraické smyčky, soustavy algebraických rovnic, ...) redukuje náročnost numerického řešení modelu
- Efektivní (umožňuje *real-time hardware-in-the-loop* simulaci)

Jazyk Modelica

- Simulační jazyk vyvíjený od roku 1996
- Vznikla oddělením od Dymoly
- Nezisková organizace: *Modelica Association*
- Objektově orientovaný jazyk
- Popis rovnicemi: diferenciální, algebraické, diskrétní
- Může kontrolovat fyzikální jednotky
- Multimodely pro různé fyzikální systémy
- Existuje standardní knihovna komponent
- Použití: průmysl, výzkum, ...

Modely v Modelice

- Různé knihovny komponent (modelů):
  - Mechanické: např. převodovky, motory, roboty
  - Elektrické a elektronické obvody: RLC, diody
  - Hydraulické: čerpadla, potrubí
  - Tepelné: chladiče, vedení tepla
  - ...
- Vytváření nových komponent/knihoven
- Modely řídicích systémů, ...

Příklad: elektrický obvod – RC článek

```
model rc "Model obvodu RC"
  Resistor      R(R=1000);
  Capacitor     C(C=0.001);
  SineVoltage   U(offset=5, V=0.5, freqHz=1);
  Ground        ground;
equation
  connect(U.n, C.n);    // propojovací rovnice
  connect(U.p, R.p);
  connect(R.n, C.p);
  connect(U.n, ground.p);
end rc;
```

Definice základních komponent obvodu

```
connector Pin
  Voltage       v;
  flow Current i;   // flow => součet=0
end;

partial model OnePort "abstraktní bázová třída"
  Pin p, n;
  Voltage v;    // napětí
  Current i;    // proud
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;
```

Ideální rezistor a kondenzátor

```
model Resistor "ideální rezistor"
  extends OnePort;
  parameter Real R(unit="Ohm") "odpor";
equation
  R*i = v;          // Ohmův zákon
end Resistor;
```

```
model Capacitor "ideální kondenzátor"
  extends OnePort;
  parameter Real C(unit="F") "kapacita";
equation
  C * der(v) = i;  // diferenciální rovnice
end Capacitor;
```

Modelica – shrnutí

- V praxi používané systémy (Dymola, OpenModelica)
- Otevřený jazyk Modelica + std. knihovna
- Numerické metody (DASSL, ...)
- Výhody
- Nevýhody
- Dymola (zastaralá verze, grant FRVŠ)
- Doporučení: použijte OMEdit – viz www.openmodelica.org

Kombinovaná (hybridní) simulace

= spojitá simulace + diskrétní simulace + jejich propojení

- Problém kombinace událostí a numerické integrace
- *Stavové podmínky* a detekce jejich změn
- Změna stavové podmínky způsobí *stavovou událost*
- Problém zkracování kroku ("dokročení" na stavovou událost)
- Skokové změny spojitého stavu a jejich vliv na použitou numerickou metodu
- ...

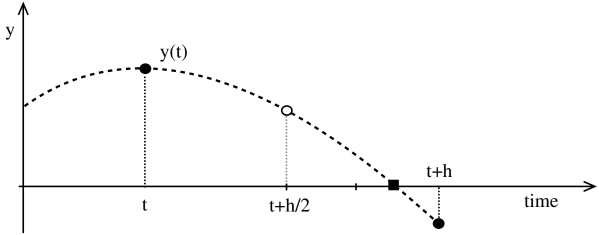
Stavové podmínky (*State Conditions*)

Problém:

Stavová událost nastane po dosažení zadané hodnoty spojité veličiny (tj. při změně stavové podmínky) – nelze ji naplánovat.

Příklad: Detekce dopadu míčku na zem

Hledáme řešení algebraické rovnice $y(t) = 0$



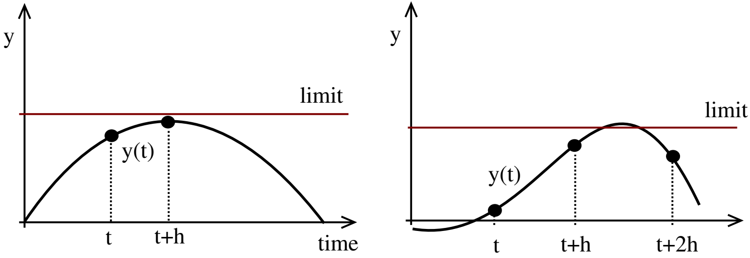
Metody: půlení intervalu, Regula-falsi, Newtonova, ...

Problémy detekce změn stavových podmínek

Problém: nedetekování stavové události způsobené

- nepřesností výpočtu
- příliš dlouhým krokem (překročení dvou změn)

Příklady:



Stavové podmínky v SIMLIB/C++

Speciální bloky – abstraktní třídy:

`Condition` (detekce jakékoli změny),
`ConditionUp` (změna false $\rightarrow$ true),
`ConditionDown` (změna true $\rightarrow$ false)

Odvozené třídy musí definovat metodu `void Action()` s popisem stavové události.

Podmínka je vždy ve tvaru `(vstup >= 0)`

Poznámka: SIMLIB používá metodu půlení intervalu při které zkracuje krok až k `MinStep`

Algoritmus řízení simulace – pseudokód

```
Inicializace stavu a podmínek
while ( čas < koncový_čas) {
    Uložení stavu a času                ***
    Krok numerické integrace a posun času
    Vyhodnocení podmínek
    if ( podmínka změněna )
        if ( krok <= minimální_krok)
            Potvrzení změn podmínek
            Stavová událost             ===
            krok = běžná_velikost_kroku
        else
            Obnova stavu a času         ***
            krok = krok/2
            if (krok < minimální_krok)
                krok = minimální_krok
}
```

Algoritmus řízení simulace – poznámky

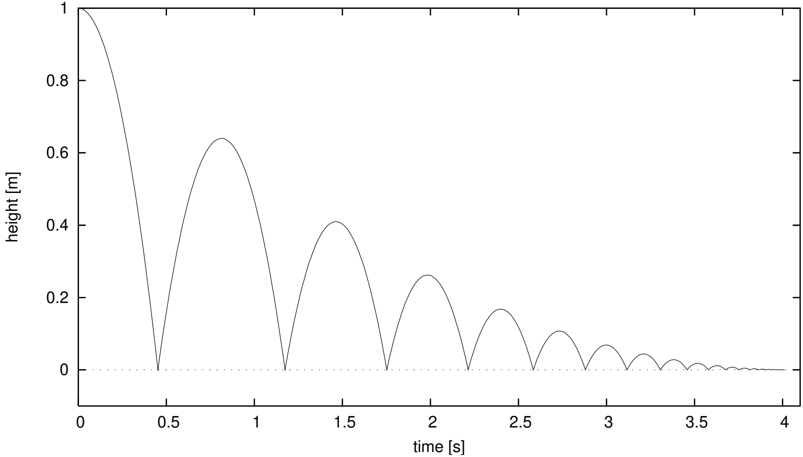
- Jde pouze o část algoritmu řízení kombinované simulace
- Pseudokód patří do algoritmu *next-event* místo:
  `Time = čas příští události`
- `koncový_čas` je čas příští události nebo čas konce simulace
- Stavová událost může plánovat událost (a tím změnit `koncový_čas`).
- `Krok numerické integrace` – délka posledního kroku před koncovým časem musí být vhodně upravena – tzv "dokročení", ale může nastat problém s minimální délkou kroku, proto je dobré použít následující kód:

```
if (Time + krok*1.01 > koncový_čas)
    krok = koncový_čas - Time;
```

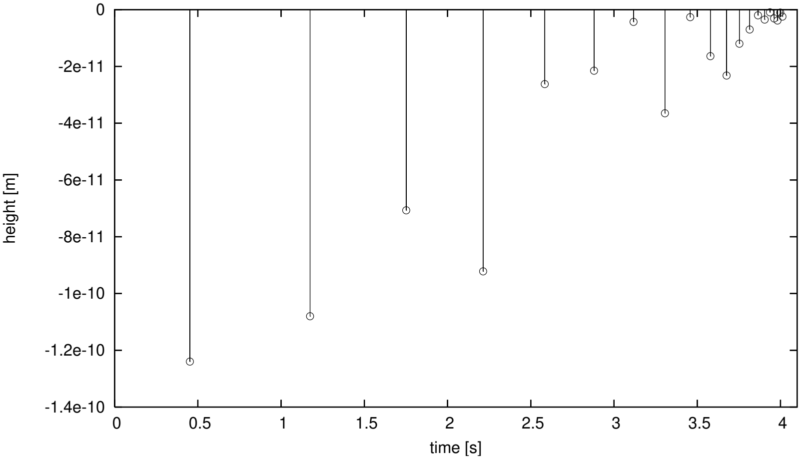
Příklad: skákající míček (SIMLIB)

```
struct Micek : ConditionDown {  // skákající míček
  Integrator v,y;               // stavové proměnné
  unsigned count;
  void Action() {
      v = -0.8 * v.Value();     // ztráta energie...
      y = 0;                    // eliminace nepřesnosti
      if(count>=20)             // max 20 dopadů
        Stop();                 // konec experimentu
  }
  Micek(double initialposition) :
    ConditionDown(y),  // (y>=0) změna TRUE --> FALSE
    v(-g),                      // y' = INTG( -g )
    y(v, initialposition),      // y  = INTG( y' )
    count(0) {}  // inicializace počtu dopadů/odrazů
};
Micek m1(1.0);                  // instance modelu
```

Příklad: skákající míček – výsledek



Příklad: míček – chyba detekce (Minstep$= 10^{-9}$)



Simulace číslicových systémů – přehled

Úrovně popisu:

- Elektrická – tranzistory, rezistory, kondenzátory (spojité modely)
- Logická – hradla, klopné obvody (diskrétní modely)
- Meziregistrové přenosy – čítače, řadiče, ALU (diskrétní modely)
- Systémová – procesory, paměti, periferie (diskrétní modely, hromadná obsluha, výkonnost)

Používají se specializované nástroje a techniky:

- SPICE: elektrická úroveň
- VHDL: logická, RTL
- ...

Modely signálů

Modely signálu

- Dvojhodnotové: jen 0 a 1 (málo používané, rychlé)
- Trojhodnotové: + neurčitá úroveň X
- 5-hodnotové: 0, 1, X, R (Rise=růst) a F (Fall=sestup)
  výhoda: přesnější popis, odhalí více hazardů
  nevýhoda: pomalé
- VHDL používá 9 hodnot (”UX01ZWLH-”)

Další možné hodnoty:

- Stav Z (vysoká impedance), různá ”síla” signálu (u CMOS)
  Statický (_/_) a dynamický (_/\/˜) hazard, ...

Modely zpoždění

Zpoždění logických členů:

- 0 — nulové (jen pro ověření log. funkce)
- 1 — jednotkové (většinou nevhodné)
- t_d — přiřaditelné (zvlášť pro 0$\rightarrow$1 a 1$\rightarrow$0)
- $\langle t_1, t_2 \rangle$ — přesné (rozsah od–do)

Poznámky:
Zpoždění na spojích
Kontrola časování (např. dodržení předstihu a přesahu u klopných obvodů)

Simulační algoritmus

Řízení událostmi $\Rightarrow$ problém velkého množství událostí v kalendáři (existují i další metody – např. s pevným krokem)

Selektivní sledování: vyhodnocování jen u prvků které jsou ovlivněny změnou na vstupu.

Implementace popisu modelu:

- řízení tabulkami (interpretace)
- kompilovaný model (provádění kódu)

Poznámky:
problém zpětných vazeb u sekvenčních obvodů (možné je např. iterační řešení),
problém inicializace (počáteční hodnoty signálů)

Algoritmus řízení simulace číslicových obvodů

Dvoufázový algoritmus (selektivní sledování):

```
inicializace, plánování události pro nový vstup
while (je plánována událost) {
   nastavit hodnotu modelového času na T
   for (u in všechny plánované události na čas T) {
      výběr záznamu události u z kalendáře
      aktualizace hodnoty signálu
      přidat všechny připojené prvky do množiny M
   }
   for (p in množina prvků M) {
      vyhodnocení prvku p
      if (změna jeho výstupu)
         plánování nové události
   }
}
```

Simulace poruch

Typy poruch:

- trvalá 0
- trvalá 1
- zkrat mezi funkčními vodiči

Činnost:

- Specifikace poruch – které poruchy budou modelovány
- Injekce poruch – převod modelu na model s poruchou
- Šíření poruch modelem
- Detekce poruch – projeví se porucha?
- Zpracování výsledků – vytvoření podkladů pro testy

Poznámka: Ověřování úplnosti diagnostického systému (vše opakovat pro každou poruchu) je časově náročné

Jazyky pro popis číslicových systémů

VHDL: vhodné pro složité systémy

Úrovně popisu (lze kombinovat):

- Popis struktury – propojení hradel
- Popis chování
  - algoritmem – proces
  - data flow – RTL (Register Transfer Level)
    např. `o <= transport i1 + i2 after 10 ns;`

Knihovny prvků

Poznámka: Příklady viz WWW — Například
`http://www.cs.ucr.edu/content/esd/labs/tutorial/`

VHDL – příklad

```
-- AND hradlo (ESD book figure 2.3)
-- převzato z
-- http://www.cs.ucr.edu/content/esd/labs/tutorial/

library ieee;
use ieee.std_logic_1164.all;

entity AND_ent is
    port(
        x: in  std_logic;
        y: in  std_logic;
        F: out std_logic
    );
end AND_ent;
```

VHDL – příklad

```
architecture behav1 of AND_ent is
begin
    process(x, y)
    begin -- popis chování
        if ((x='1') and (y='1')) then
            F <= '1';
        else
            F <= '0';
        end if;
    end process;
end behav1;
-- varianta 2
architecture behav2 of AND_ent is
begin
    F <= x and y;
end behav2;
```

Heterogenní modely

= Použití více různých forem popisu pro různé části modelu

Příklad heterogenního modelu řídicího systému:

- spojitá část (spojitý popis)
- A/D převod (vzorkování spojitého stavu)
- číslicový řídicí systém (např. Petriho síť)
- nebo použití fuzzy logiky (popis neurčitosti)
- případně použití neuronových sítí (učení)
- D/A převod (kombinace spojitý/diskrétní)
- ...

Poznámka: Jsou nutné odpovídající nástroje

SIMLIB – některá rozšíření

- Vektorové bloky a blokové výrazy
  - 2D vektorové diferenciální rovnice
  - 3D vektorové diferenciální rovnice
- *Fuzzy* popis modelů s neurčitostí
  - fuzzy množiny
  - fuzzy bloky – fuzzification, inference, defuzzification
  - editor fuzzy množin (Java)
- Optimalizační metody
- + další doplňky...

Poznámka:
Jde o prototypy = ne zcela úplné, používat opatrně

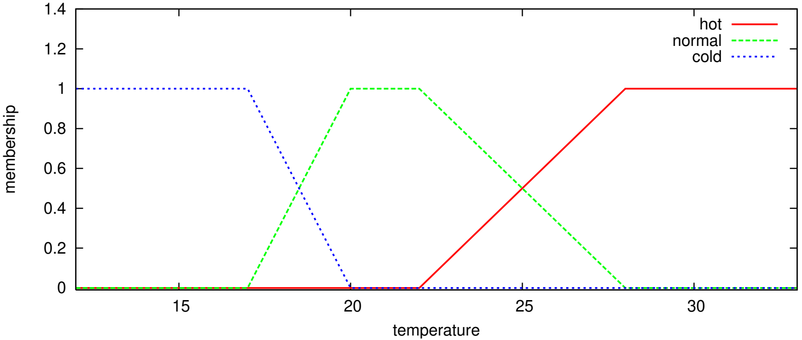
Fuzzy logika – základy

- Jde o popis jednoho typu neurčitosti – ”vágnost”
  (co znamená, že něco je ”malé” nebo ”velké” ?)
- Rozšíření Booleovské logiky (1965, Lofti Zadeh)
- Vyjádření míry určité vlastnosti – pravdivostní hodnota, stupeň příslušnosti (Pozor – vůbec nesouvisí s pravděpodobností.)
- Pojmy: fuzzy množina, funkce příslušnosti
- Použití fuzzy logiky: řízení, expertní systémy, ...

Poznámka: Podrobnosti viz např. PDF na WWW:
Navara M., Olšák P.: *Základy fuzzy množin*, ČVUT, Praha, 2002

Fuzzy množina, funkce příslušnosti

Př: teplota v místnosti, 3 fuzzy množiny a jejich funkce příslušnosti:
malá – střední – velká ("cold" – "normal" – "hot")



Fuzzifikace: převod "ostré" hodnoty na míru příslušnosti
(Příklad: 18 $^\circ\mathrm{C} \rightarrow$ cold:0.5, normal:0.5, hot:0)

Operace

Fuzzy

negace: $\neg_s \alpha = 1 - \alpha$

konjunkce: $\alpha \bigwedge_s \beta = \mathit{min}(\alpha, \beta)$

disjunkce: $\alpha \bigvee_s \beta = \mathit{max}(\alpha, \beta)$

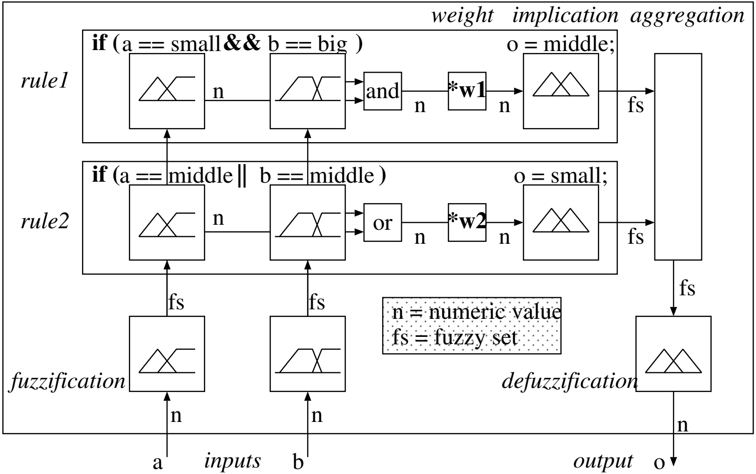
Poznámka: Existují i jiné definice operací

Fuzzy blok

Postup vyhodnocování:

- převod vstupu na míru příslušnosti (*fuzzification*)
- aplikace pravidel (*if-then rules*)
  Příklad pravidel:
  ```
  IF teplota IS malá        THEN topit=hodně
  IF teplota IS střední     THEN topit=málo
  IF teplota IS velká       THEN topit=chladit
  ```
- spojení výstupů pravidel (*aggregation*)
- převod na ”ostré” hodnoty (*defuzzification*)

Fuzzy blok – obrázek



Příklad v SIMLIB – pouze pro ilustraci možností

```
// Fuzzy množiny:
FuzzySet itype("itype", 0, 40,
  FuzzyTrapez("small", 0,0,18,20),
  FuzzyTrapez("medium", 18,20,22,28),
  FuzzyTrapez("big", 22,28,40,40)
);

class MyBlock : public FuzzyBlock {
  FuzzyInput in;
  FuzzyOutput o, o2;
  void Behavior() {     // Pravidla:
    if(in=="small")  weight(0.9), o="big";
    if(in=="big" || in=="medium") o="small", o2="z";
    if(in=="big" || in=="medium") { o="small"; o2="z"; }
  } // ...
};
```

Optimalizace parametrů modelu

Cíl: nalezení optimálních hodnot parametrů modelu

Pojmy: operační výzkum, lineární/nelineární programování

Optimalizační metody:

- gradientní
- simulované žíhání (Simulated Annealing)
- genetické
- ...

Nástroje: Scilab, MATLAB+OptimizationToolbox, ...

Poznámka: Složitost optimalizačních úloh

Optimalizace

Hledáme x pro minimum nebo maximum *cenové* funkce $F(\vec{x})$.

Minimalizace je algoritmus, který počítá:

$$\vec{x} \ : \ F(\vec{x}) = min \ \wedge \ C(\vec{x})$$

kde:

$\vec{x}$ je vektor hodnot parametrů

F je cenová (*Objective*) funkce

C reprezentuje různá omezení (*Constraints*) – například meze hodnot $\vec{x}$.

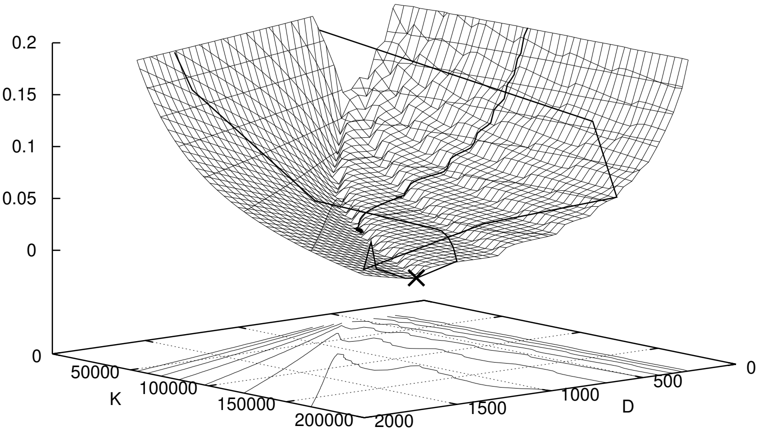
Poznámky:

Maximalizace $\Rightarrow$ použijeme $-F$.

Problém: lokální minima $\Rightarrow$ používáme *globální optimalizační metody*

Optimalizace – příklad

Ukázka hledání minima, 3 různé metody:



Vizualizace výsledků simulace

Vizualizace znamená použití interaktivních vizuálních reprezentací dat pro zlepšení našeho chápání problému.

- interaktivní diagramy
- animace
- 3D zobrazení
- video, ...
- virtuální realita

Nástroje:

- univerzální: Gnuplot, GNU plotutils, ...
- specializované: viz WWW

Poznámka: *client-server*, ...

Virtuální realita a simulace

3D interaktivní vizualizace a simulace

- prostředí, které je simulováno počítačem
- člověk je připojen na speciální rozhraní a vstupuje do simulovaného prostředí
- interakce člověk — stroj

Poznámka: Souvislost s počítačovými hrami

Analytické řešení modelů

Čistě matematické řešení modelu.

- výhody: efektivní, výsledky jsou obecné a přesné
- nevýhody: analytické řešení pro většinu modelů neznáme/neexistuje

Postup:

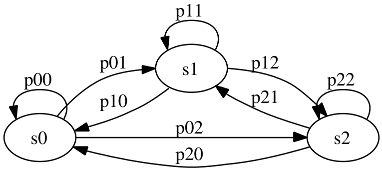
- analýza problému
- formulace matematického modelu
- zjednodušení modelu (linearizace, ...)
- matematické řešení modelu

Poznámka: Porovnat se simulací

Markovovy procesy a řetězce

- Při zkoumání dějů v SHO se často předpokládá, že v nich obsažené náhodné procesy jsou *Markovské*.
- Markovské procesy jsou náhodné procesy, které splňují *Markovovu vlastnost*: následující stav procesu závisí jen na aktuálním stavu (ne na minulosti).
- Náhodný proces $X(t)$ s diskrétním časem a diskrétními stavy, který má Markovovu vlastnost, se nazývá *Markovův řetězec*. Je ekvivalentní konečnému automatu s pravděpodobnostmi přechodů
- Pravděpodobnost stavu s_i v čase $t \in N$ označíme symbolem $\pi_i(t) := p(X(t) = s_i)$.

Markovovy řetězce



- Matice pravděpodobností přechodů: $P = \begin{bmatrix} p_{00} & p_{01} & p_{02} \\ p_{10} & p_{11} & p_{12} \\ p_{20} & p_{21} & p_{22} \end{bmatrix}$

  (součet na řádku musí být 1)
- Aplikace: SHO, náhodná procházka, hry – házení kostkou

Systémy M/M/1

M/M/1 – viz Kendallova klasifikace SHO

- máme jedno zařízení s neomezenou FIFO frontou
- příchody požadavků: exponenciální intervaly s konstantním parametrem $\lambda > 0$, nezávisí na stavu modelu a čase
- obsluha: exponenciální trvání s parametrem $\mu > 0$.

Počet požadavků v systému $X(t)$ je Markovský proces.

Popis:

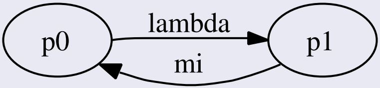
- Vektor pravděpodobností jednotlivých stavů
- Spojitý čas
- Používáme matici *intenzit* přechodů mezi stavy

Příklad 1

Jedno zařízení bez fronty – přijde-li požadavek a nemůže být obsloužen, opouští systém.
Parametry — příchody: 15 za hodinu, obsluha: 5 minut.

Jaká je pravděpodobnost, že požadavek odchází neobsloužen?

$\lambda = 15, \quad \mu = \frac{60}{5} = 12$ za hodinu, (poznámka: stabilita).



$$\begin{bmatrix} p_0 & p_1 \end{bmatrix} \begin{bmatrix} -\lambda & \lambda \\ \mu & -\mu \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} \quad \text{a současně} \quad p_0 + p_1 = 1$$

Příklad 1 – ustálený stav

V ustáleném stavu se pravděpodobnosti již nemění, proto intenzita přechodů násobená pravděpodobností stavu musí být v rovnováze.

Pro *ustálený stav* platí:

$$-\lambda p_0 + \mu p_1 = 0, \quad \text{a také} \quad p_0 = 1 - p_1$$

Dosadíme a úpravami získáme výsledek:

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

Pro naše parametry je pravděpodobnost obsazeného zařízení:

$$p_1 = \frac{5}{9}$$

Příklad 2

Systém M/M/1 – výdejna obědů.
Přichází 3 požadavky za minutu, výdej 15 sekund.

Jaká je

- pravděpodobnost p_0, že strávník nebude čekat,
- průměrná délka fronty L_w,
- průměrná doba čekání ve frontě T_w a
- průměrná doba strávená v systému T_s ?

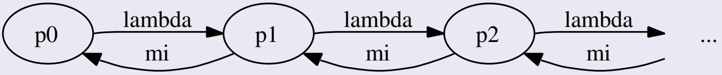
$\lambda = \frac{pocet}{cas} = \frac{3}{1} = 3$ příchody za minutu

$\mu = 4$ dokončené obsluhy za minutu (doba obsluhy $T_o = \frac{1}{\mu}$)

Systém je stabilní ($\lambda < \mu$).

Příklad 2 – rovnice pro ustálený stav

Rovnice: $\vec{\pi}(\infty)A = \vec{0}$



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

$p_1 = \frac{\lambda}{\mu} p_0 = \rho p_0$ kde jsme zavedli $\rho = \frac{\lambda}{\mu}$

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

$$p_2 = -\frac{\lambda}{\mu} p_0 + \frac{\lambda}{\mu}\frac{\lambda}{\mu} p_0 + \frac{\lambda}{\mu} p_0 = \frac{\lambda^2}{\mu^2} p_0 \quad \Rightarrow \quad p_2 = \rho^2 p_0$$

...

$$p_k = \rho^k p_0$$

Příklad 2 – výpočet pravděpodobností

Součet pravděpodobností stavů musí být 1:

$$p_0 + p_1 + p_2 + \ \dots \ = 1$$

Potom použijeme vzorec pro součet geometrické řady ($S = \frac{a}{1-q}$):

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

a po úpravě dostaneme výsledek: $p_0 = 1 - \rho$

Výsledek

Pravděpodobnost, že nebude čekat: $p_0 = 1 - \rho \ = \frac{1}{4} = 0.25$

Poznámka: To znamená, že s pravděpodobností p_0 zařízení nepracuje (průměrné využití zařízení je: $1 - p_0 = \rho = \frac{3}{4}$).

Příklad 2 – délka fronty

Průměrná délka fronty v ustáleném stavu je suma součinů (délka fronty * pravděpodobnost stavu s touto délkou fronty) pro všechny možné délky fronty:

$$L_w = \sum_{k=1}^{\infty} k.\pi_{k+1}(\infty) = \sum_{k=1}^{\infty} k.\rho^{k+1}(1-\rho) =$$
$$= (1-\rho)\rho^2 + 2(1-\rho)\rho^3 + 3(1-\rho)\rho^4 + \ \dots \ =$$
$$= \rho^2 - \rho^3 + 2\rho^3 - 2\rho^4 + 3\rho^4 + \ \dots \ =$$
$$= \rho^2 + \rho^3 + \rho^4 + \dots = \frac{\rho^2}{1-\rho} \qquad \text{// součet řady}$$

Výsledek

V našem příkladu je průměrná délka fronty:

$$L_w = \frac{\rho^2}{1-\rho} = 2.25$$

Příklad 2 – doba čekání ve frontě

Doba čekání ve frontě T_w je úměrná počtu transakcí N v systému:

$$T_w = T_o.N = T_o \sum_{k=1}^{\infty} k.\pi_k(\infty) = \frac{1}{\mu} \sum k.\rho^k (1-\rho)$$

$$= \frac{1}{\mu}\frac{1}{\rho}(\sum k.\rho^{k+1}(1-\rho)) = \qquad \text{// obsah závorky již známe}$$

$$= \frac{1}{\mu}\frac{1}{\rho}\frac{\rho^2}{1-\rho} = \frac{1}{\mu}\frac{\rho}{1-\rho} =$$

$$= \frac{1}{\mu}\frac{\rho}{1-\frac{\lambda}{\mu}} = \frac{\rho}{\mu-\lambda} = 45s$$

$$T_s = T_w + T_o = \frac{\rho}{\mu-\lambda} + \frac{1}{\mu} = \frac{\lambda+\mu-\lambda}{\mu(\mu-\lambda)} = \frac{1}{\mu-\lambda} = 60s$$

Pro náš příklad je průměrná doba čekání ve frontě 45s a průměrná doba strávená v systému je 60s.

Příklad 2 – simulační řešení

Pro srovnání provedeme simulační experimenty v SIMLIB/C++:

Výsledky pro různou dobu simulace (od 1000 do 10^7 sekund):

čas [s]	p_0 (=nečeká)	L_w	T_w [s]	T_s [s]
1000	0.207	1.378	24.38	38.51
5000	0.226	1.646	30.32	44.31
10000	0.168	2.862	54.79	70.32
100000	0.248	2.153	42.35	57.05
1e+06	0.254	2.116	42.49	57.46
1e+07	0.250	2.255	45.16	60.16
analytické	0.25	2.25	45	60

Výsledky se blíží přesným hodnotám získaným analyticky.

Modely spolehlivosti

Spolehlivost = schopnost plnit požadované funkce podle stanovených podmínek

Pojem *spolehlivost* může zahrnovat:

- bezporuchovost
- životnost
- ...

Poznámky:

- Kvalita, ISO9000
- Modely spolehlivosti: kombinatorické, markovské, ...
- *Fail-safe* systémy

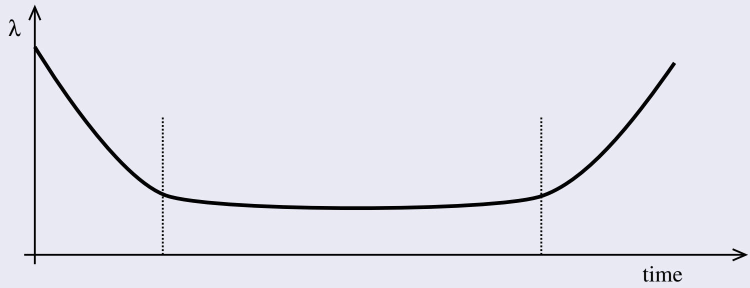
Ukazatele spolehlivosti

- Pravděpodobnost bezporuchové činnosti $R(t)$
  v intervalu $\langle 0, t\rangle$: $R(t) = e^{-\lambda t}$
- Pohotovost (*availability*) $a(t)$ = pravděpodobnost, že
  v čase t bude systém v provozuschopném stavu.
  (Vlivy: četnost výpadků, rychlost oprav)
- Střední doba bezporuchové činnosti: $T_s = \int_0^\infty R(t)dt$
  Anglicky: MTBF (Mean Time Between Failures)
- Intenzita poruch $\lambda(t) = \frac{1}{T_s}$

Poznámka:
Odolné systémy tolerují poruchy, pojem "*high-availability*".

Intenzita poruch

Typický průběh intenzity poruch $\lambda(t)$ – vanová křivka



Poznámka: Rané poruchy — provoz — stáří.

Kombinatorické modely spolehlivosti

- Sériové spolehlivostní zapojení:

$$R(t) = \prod_{i=1}^{n} R_i(t)$$

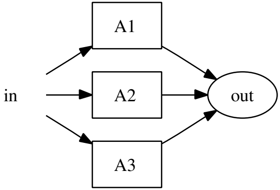
- Paralelní spolehlivostní zapojení:

$$R(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t))$$

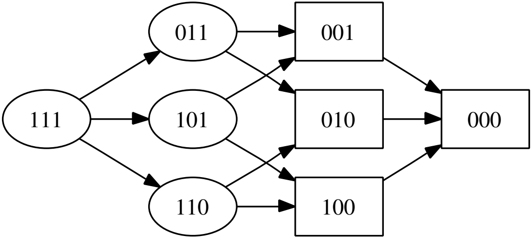
- Nevýhody: Komplikované sestavování schémat, ...

Markovské spolehlivostní modely

Příklad systému — tři prvky paralelně:



Stavový graf (1=funguje, 0=porucha, pořadí A1 A2 A3):



Shrnutí

- Markovské procesy a řetězce
- Princip analytického řešení
- Výhody/nevýhody
- Aplikace

Poznámka:
Analyticky lze řešit i jiné typy modelů (nejen výše uvedené)

Závěr

Shrnutí:

- Principy modelování a simulace
- Klasifikace systémů a modelů
- Používané metody a algoritmy
- Základy implementace simulačních nástrojů
- Aplikace simulace a souvislosti s různými obory

Poznámky:

- Co jsme vynechali
- Co se zkouší — cílové znalosti