Systém (reálný, nereálný): soubor elementárních částí (prvků systému), které mají mezi sebou určité vazby.

Model: napodobenina systému jiným systémem. (Reprezentace znalostí.)

Modelování: vytváření modelů systémů.

Simulace: získávání nových znalostí o systému experimentováním s jeho modelem.

Základní etapy modelování a simulace:

- vytvoření abstraktního modelu
- vytvoření simulačního modelu: zápis AM formou programu
- validace (AM, SM), verifikace (SM)
- simulace: experimentování se SM
- analýza a interpretace výsledků: získání nových *znalostí*

Souvislosti: pozorování, reálné experimenty, učení, hry, programování, algoritmy, datové struktury, numerická matematika, počítačová grafika, hardware, teorie systémů, pravděpodobnost, statistika.

Výhody simulačních metod: cena, rychlost, bezpečnost, někdy to jinak nejde, možnost modelovat složité systémy.

Problémy simulačních metod: validita, náročnost vytváření, výpočetní náročnost (na výkon PC), numerická řešení – konkrétní, nepřesná, problém se stabilitou.

Analytické řešení: popis chování systému matematickými vztahy a jejich matematické řešení – výsledky jsou ve formě funkčních vztahů. Přesnější a méně časově náročnější, ale ne vždy možné a vhodné.

Kdy použít simulaci:

- neexistuje úplná matematická formulace, nejsou známé analytické metody
- analytické metody by vyžadovaly příliš velké zjednodušení
- analytické metody jsou teoreticky dostupné, ale jejich použití by bylo extrémně obtížné
- modelování je jedinou možností pro získání výsledků (např. střety galaxií)
- potřebujeme měnit časové měřítko

==== STOP ====

Systém S = (U, R)
U = univerzum
R = charakteristika = množina všech propojení $R_{ij} \subseteq Y_i \times X_j$

Vazby mezi prvky: sériové, paralelní, zpětná vazba.

Čas: reálný, modelový, strojový.

Časová množina (**time base**): množina všech časových okamžiků, ve kterých jsou definovány hodnoty vstupních, stavových a výstupních proměnných prvku systému – diskrétní, spojitá.

Chování: každému *časovému průběhu* vstupních prom. přiřazuje *časový průběh* výstupních prom. $\boldsymbol{\chi_S : [\sigma_{in}(S)]^T \rightarrow [\sigma_{out}(S)]^T}$, kde $[A]^T$ je množina všech zobrazení *T do A*.

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 a reakcemi rozumíme dvojice podnětů/reakcí, které jsou spolu vzájemně přiřazeny definovaných vstupních/výstupním zobrazením.

Izomorfní systémy: systémy S_1 a S_2 jsou izomorfní, **iff**:

- prvky univerza U_1 lze vzájemně jednoznačně (bijektivně) přiřadit prvkům U_2.
- prvky charakteristiky R_1 lze bijektivně přiřadit prvkům R_2, a to tak, že prvku R_1 (vyjadřujícímu orientovanými vztah mezi dvěma prvky z U_1) je vždy přiřazen právě ten prvek R_2, který vyjadřuje stejně orientovaný vztah mezi odpovídající dvojicí prvků univerza U_2 (podle bijekce z předchozího bodu), a naopak.

Homomorfní systémy: systémy S_1 a S_2 jsou homomorfní, **iff**:

- prvkům univerza U_1 lze přiřadit jednoznačně prvky univerza U_2 (zobrazení U_1 **do** U_2). (Naopak jednoznačné být nemusí, mapování N:1.)
- prvkům charakteristiky R_1 je možné jednoznačně přiřadit prvky R_2, a to tak, že prvku R_1 vyjadřujícímu ori. vzt. mezi dvěma prvky U_1 je vždy přiřazen právě ten prvek R_2, který vyjadřuje stejně ori. vzt. mezi odpovídající dvojicí prvků U_2 ve smyslu předchozího bodu.

Modelování je typicky vytváření homomorfního systému.

(Podstatné) Okolí systému: vše, co má vliv na chování systému a není jeho součástí.

Uzavřený systém: nekomunikuje s okolím (zanedbáváme jej).

Otevřený systém: komunikuje s okolím (typicky má definované vstupy, výstupy).

Klasifikace prvků 1: se spojitým chováním (rezistor), s diskrétním chováním (FIFO fronta).

Klasifikace prvků 2: s deterministickým chováním (FIFO fronta), s nedeterministickým chováním (šumová dioda).

Typy systému: spojité (všechny prvky spojité), diskrétní (všechny prvky diskrétní), kombinované; deterministické, nedeterministické (aspoň 1 prvek s nedeterm. chováním).

Cíl simulace: získávání nových informací o chování systému v závislosti na určitých vstupních veličinách a hodnotách parametrů.

Postup simulace: nastavení hodnot parametrů, poč. stavu modelu -> zadávání vstupních podnětů z okolí v průběhu simulace -> vyhodnocení výstupních dat. Opakujeme, dokud nezískáme dostatek informací o chování nebo hodnoty parametrů, pro které má systém žádané chování.

Zpracování výsledků: záznam průběhu, vizualizace výsledků, animace; intuitivní vyhodnocení, statistické metody, automatické vyhodnocení…

Verifikace: ověřování korespondence AM a SM (izomorfního vztahu mezi nimi). AM je formální specifikací pro program (SM).

Validace: proces, v němž se snažíme dokázat, že skutečně pracujeme s modelem adekvátním modelovanému systému. („Míra použitelnosti, správnosti získaných výsledků.“)

Způsoby popisu modelů: KA, Petriho síť, Turingův stroj, algebraické rovnice, diferenciální rovnice, diferenční rovnice, markovské řetězce…

Tvorba AM – cíl: formulace zjednodušeného popisu systému *abstrahujícího* od všech *nedůležitých* skutečností vzhledem k *cíli a účelu* modelu. Zajímáme se jen o ohraničené části, musíme identifikovat vhodné složky systému. Předpokládáme homomorfní vztah mezi systémem a AM.

Cíle modelování: studium chování systému pro daná kritéria, studium závislostí; predikce; analýza citlivosti na změny parametrů; optimalizace.

Simulační model: AM zapsaný formou programu.

Klasifikace modelů podle Fishwicka:

- konceptuální
- deklarativní
- funkcionální
- constraint (popsané rovnicemi)
- spatial (prostorové)
- multimodely

Simulační nástroje: obecné jazyky, simul. knihovny pro obecné jazyky (SIMLIB, SimPy), simulační jazyky (Modelica), simulační systémy (OpenModelica, Dymola), propojování nástrojů.

==== STOP ====

Náhodné procesy: změříme vzorek procesu v realitě (soubor dat) -> aproximujeme jej analytickým vyjádřením -> modelujeme jej generátorem pseudonáhodných čísel (s odpovídajícím rozdělením).

Náhodná veličina: veličina, která jako výsledek pokusu může nabýt nějakou hodnotu, přičemž předem nevíme, jaká konkrétně bude.

Rozdělení pravděpodobnosti: určuje vztah mezi možnými hodnotami náh. veličiny x_i a jim příslušejícími pravděpodobnostmi $p_i = P(X = x_i)$.

Histogram: třídí soubor výsledků pokusů do *K* tříd podle vhodně zvolených intervalů.

Charakteristiky:

- polohy: posunutí vzhledem k počátku (modus, medián, kvantily).
- variability: rozsah kolísání hodnot kolem středu (rozptyl, směrodatná odchylka).
- šikmosti: nesymetričnost rozdělení hodnot.
- špičatosti: porovnávají variabilitu rozdělení ve středu a na okrajích.

Momenty: obecné, centrální, viz sešit.

Často používaná rozdělení: diskrétní – Poissonovo, binomické; spojitá – rovnoměrné, exponenciální, normální, Pearsonovo (χ^2), Studentovo (t)...

Poissonovo rozdělení: počet událostí za jednotku času (např. příchody zákazníků).

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

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

Exponenciální rozdělení: rozdělení dob obsluhy, časové intervaly mezi událostmi (poruchami, příchody požadavků).

Normální rozdělení – pravidlo tří sigma: $P(X \in [\mu - 3\sigma, \mu + 3\sigma]) \geq 0{,}99$

Fyzikální zdroje náhodnosti: opravdu náhodné (nedetermin.), generují pouze málo bitů za sekundu, není vždy dostupné.

Algoritmické generátory: pseudonáhodné (deterministické), generují miliardy bitů za sekundu.

Kongruentní generátory: $x_{n+1} = (ax_n + b) \bmod m$ – rovnoměrné rozložení, konečná posloupnost (perioda generátoru).

Další generátory pseudon.: Mersenne twister, Xorshift, LCG, LSFR.

Požadavky na generátory: rovnoměrnost, statistická nezávislost generované posloupnosti, co nejdelší perioda, rychlost.

Inverzní transformace rozdělení: inverzní distribuční funkce cílového rozložení (nemusí jít vyjádřit – někdy nelze použít). **$y = F^{-1}$(Uniform(0, 1))**

Vylučovací transformace rozdělení: sérií pokusů hledáme číslo, které vyhovuje funkci hustoty cílového rozdělení. Nevhodné pro neomezená rozdělení. **$x = \text{Uniform}(x_1, x_2)$**, **$y = \text{Uniform}(0, \text{Max}(f(x)))$**; **$y < f(x)$ => x je** hodnota náh. veličiny, **jinak** generujeme **znovu**.

Testování náhodných veličin: vstupem je soubor (pseudo)náhodných čísel, cílem je potvrdit hypotézu příslušnosti k danému rozdělení / najít rozdělení / najít parametry daného rozdělení, aby mu vzorek co nejlépe odpovídal.

==== STOP ====

Monte Carlo: experimentální numerická simulační metoda (třída metod). Řeší úlohu experimentováním se stochastickým modelem. Využívá vzájemného vztahu mezi hledanými veličinami a pravděpodobnostmi, se kterými nastanou určité jevy. Výpočty určitých integrálů (hlavně **vícerozměrných**), řešení diferenciálních rovnic, finance.

Přesnost Monte Carlo: relativní chyba = **1 /** $\sqrt{\mathbf{N}}$, kde N je počet provedených experimentů.

Výpočty určitého integrálů pomocí Monte Carlo: str. 114, 115.

Výpočet objemu tělesa: str. 116 (???).

Dirichletova úloha: str. 117 (???).

==== STOP ====

Proces: posloupnost událostí.

Kvaziparalelismus: simulace provádění paralelních procesů na jednoprocesorovém počítači.

P/T Petriho síť: **$\Sigma = (P, T, F, W, C, M_0)$** (P: množ. míst, T: množ. přechodů, F: incidenční relace, W: váhová funkce, C: kapacity míst, M_0: počáteční značení).

Přechod je proveditelný při značení M, iff ve vstupních místech čeká dostatek procesů a současně výstupní místa mají dostatečně volnou kapacitu.

Systémy hromadné obsluhy – SHO – Queueing Systems: systémy obsahující zařízení (obslužné linky) s frontami (různých typů), která poskytují obsluhu transakcím.

U SHO sledujeme: informace o čase stráveném transakcí v systému, doby čekání ve frontách, vytížení obslužných linek.

Cíl SHO: odhalit zdržení, optimalizovat výkon…

Fronty: charakterizovány řazením požadavků, způsobem výběru požadavku, maximální délkou.

Řády fronty: FIFO, LIFO, SIRO (service in random order).

Nulová fronta: požadavky nemohou vstupovat do fronty -> zanikají -> systém se ztrátami.

Fronta s netrpělivými požadavky: opouští systém, když doba čekání překročí danou mez (timeout).

Možnosti prioritní obsluhy (při příchodu prioritnějšího požadavku, když právě probíhá obsluha méně prio. požadavku):

- Slabá priorita: započatá obsluha se normálně dokončí.
- Silná priorita: zap. obsluha se přeruší a začne obsluha prioritnějšího pož.
    - Přerušený požadavek odchází ze systému.
    - Přerušený požadavek se vrací do fronty (obsluha pak pokračuje buď od přerušeného místa, nebo znovu začíná).

Jsou-li všechny linky obsazeny a u každé je fronta, požadavek se „sám rozhodne“, do které fronty se zařadí. Vytvářejí-li požadavek jednu společnou frontu k více linkám, požadavek vstupuje do té linky, která se uvolní nejdříve.

Obslužná síť = spojení několika obslužných linek (otevřená – vyměňuje požadavky s okolím).

Kendallova klasifikace SHO: A/S/c (**A**rrival, **S**ervice time, **c**ount of servers):

Symbol	A	S
M	**Poissonův proces příchodů** (exp. rozdělení vzájemně nezávislých intervalů mezi příchody)	**Exponenciální rozdělení doby obsluhy**
E_k	Erlangovo rozdělení intervalů mezi příchody	Erlangovo rozdělení doby obsluhy
K_n	Rozdělení χ^2 intervalů mezi příchody, *n* stupňů volnosti	Rozdělení χ^2 doby obsluhy
D	**Pravidelné deterministické příchody**	**Konstantní doba obsluhy**
G	Žádné předpoklady o procesu příchodu	Jakékoliv rozdělení doby obsluhy
GI	Rekurentní proces příchodů	

Při modelování SHO popisujeme: procesy (transakce) v systému, jejich příchody, činnost, odchody; stav obslužných linek a front u nich, průběh obsluhy transakcí v zařízeních.

Obslužné linky: zařízení (facility), kapacita 1; nebo sklad (store), kapacita > 1.

Každé zařízení má vlastní frontu -> pole zařízení.

Více zařízení stejného typu sdílí frontu -> sklad nebo pole zařízení se sdílenou frontou.

Událost (SIMLIB `Event`): jednorázová nepřerušitelná akce provedená v určitém modelovém čase. Často plánuje sama sebe.

Kalendář událostí: uspořádaná datová struktura uchovávající **aktivační záznamy** budoucích událostí. Umožňuje výběr prvního záznamu s nejmenším aktivačním časem (a nejvyšší prioritou) a vkládání/rušení záznamů. Záznam je trojice: **aktivační čas, priorita, odkaz** na obslužnou rutinu.

Next-event algoritmus:

```
Init();  // inicializace času, kalendáře, modelu, …
while (!Calendar.Empty()) {  // dokud je kalendář neprázdný
    event = Calendar.Top();  // vyjmi první záznam z kalendáře
    if (event.ActTime() > T_END) {
        EndSimulation();
        break;  // konec simulace
    }

    CurrentTime = event.ActTime();  // aktuální čas = akt. čas
    event.Behaviour();  // proveď popis chování události
}
```

Priorita procesu v SIMLIB: ovlivňuje řazení do front (v kalendáři událostí, u zařízení).

Priorita obsluhy v SIMLIB: umožňuje přerušení probíhající obsluhy událostí s nižší prioritou obsluhy v zařízení (modelování poruch; po ukončení prioritnější obsluhy je méně prioritní navrácena).

==== STOP ====

CA: typicky diskrétní systém, *n*-rozměrné pole buněk (*lattice*), buňka (*cell*) je základní element, který se nachází v jednom z konečného počtu stavů.

Lattice: obvykle 1D nebo 2D, reprezentuje rovnoměrně rozdělený prostor, konečné nebo nekonečné.

Okolí (*Neighbourhood*): množina sousedících buněk, sousednost je nějak definována (liší se počtem a pozicí).

Pravidla (*Rules*): funkce stavu buňky a jejího okolí definující nový stav buňky v čase.

Okolí ve 2D se čtvercovými buňkami: Von Neumann (4-okolí), Moore (8-okolí), Margolus (shifted square; sypání písku).

Šestiúhelníkové prostory: implementujeme převedením na čtvercové, použití např. pro růst krystalů, šíření vln.

Okrajové podmínky: periodické, pevné, adiabatic (nulový gradient), reflection (předposl. buňka).

Implementace: přímá (pole), vyhledávací tabulka (jen nenulové buňky), SIMD, **hash life** (caching, quadtrees).

Life: stavy 0/1, 8-okolí (Moore); 1 => 1 iff 2–3 sousedi 1; 0 => 1 iff právě 3 sousedi 1; jinak 0.

Sand rule: stochastický, jedno z pravidel využívá pravděpodobnost (alt. rozdělení).

Pravidla: musí popisovat změnu stavu pro všechny možnosti (rychle roste).

Typy pravidel: legal, totalistic (součet vstupních stavů).

Reverzibilní CA: neztrácí informaci při vývoji v čase (lze otočit běh času a vracet se k předchozím stavům).

Konfigurace CA: stav všech buněk.

Stav CA se vyvíjí v čase a prostoru podle pravidel, čas i prostor jsou diskretizovány, buňky jsou identické, následující stav závisí pouze na aktuálním stavu.

Klasifikace CA:

- třída 1: dosáhnou ustáleného stavu
- třída 2: dosáhnou periodického opakování
- třída 3: chaotické chování
- třída 4: kombinace běžného a chaotického chování, nejsou reverzibilní

==== STOP ====

Možnosti popisu spojitých systémů (mimo jiné):

- soustavy obyčejných diferenciálních rovnic (ODE, Ordinary differential equation)
- parciální diferenciální rovnice (PDE, Partial differential equation)
- algebraicko-diferenciální rovnice (DAE, Differential-algebraic system of equations)
- soustavy algebraických rovnic
- bloková schémata
- elektrická schémata
- grafy signálových toků
- kompartmentové systémy

Maticový popis diferenciálních rovnic:

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

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

Koeficienty mohou být:

- nezávislé na čase (stacionární systémy)
- časové proměnné

- konstanty (lineární systémy)
- nelineární funkce (nelineární)

Popis systému bloky: řád dif. rovnice odpovídá počtu integrátorů.

Bezpaměťové = funkční bloky: konstanty, modelový čas T, operátory (+, -) a funkce (sin, cos, log, uživatelsky definované).

Paměťové = stavové bloky: integrátory, zpoždění, hystereze… **mají počáteční podmínky** (stavové -> mají stav -> musí jej mít i na počátku).

Převod rovnic vyššího řádu: umíme dobře implementovat numerické metody pro řešení dif. rovnic 1. řádu (ale existují i metody pro vyšší řády), musíme tedy převést rovnice vyšších rovnic na soustavu jednodušších dif. rovnic 1. řádu.

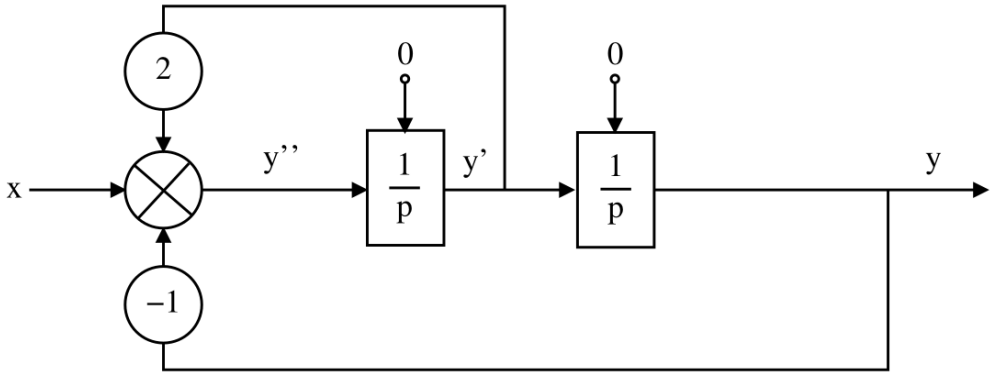
Metody převodu: snižování řádu derivace, postupná integrace, …

Metoda snižování řádu derivace: osamostatníme nejvyšší řád derivace (*ne vždy lze!*); zapojíme všechny integrátory za sebe. Nesmí tam být derivace vstupů!

Výstupy integrátorů = stavové proměnné -> potřebují počáteční podmínky.

y'' - 2y' + y = x
y'' = 2y' - y + x
y' = ∫y''
y = ∫y'

Počáteční podmínky potřebuji znát pro **y** a **y'** (nikoliv y'').



Metoda postupné integrace: vhodné pro rovnice s derivacemi vstupů. Koeficienty musí být konstantní. Nesmí se objevit vyšší řád derivace *vstupu*, než je nejvyšší řád derivace *výstupu*.

Počáteční podmínky u MPI: stavové proměnné ve výsledném schématu jsou jiné než u původního systému. Vyjádříme je z rovnic, ve kterých jsme provedli substituci.

Ty metody se dají kombinovat (někde uprostřed zjistím, že už tam nemám žádné derivace vstupu, tak budu pokračovat metodou sniž. řádu derivace…)

Numerické metody: potřebujeme řešit ODR 1. řádu (= *initial value problem*) a algebraické rovnice (hledání kořenů -> řešení rychlých smyček).

Metody řešení ODR 1. řádu: hledají řešení rovnice **y' = f(t, y(t))** – nějaké funkce času a y(t). Toto *řešení* má obecně tvar $\mathbf{y(T) = y_0 + \int_{0}^{T} f(t, y(t))\, dt}$.

Numerické řešení = diskretizace, rozdělení na malé časové *integrační kroky*: $\mathbf{h_i = t_{i+1} - t_i}$.

Typicky průběh řešení *y(T)* aproximujeme polynomem a vypočítáme $\mathbf{y(t_i + h_i)}$ – provádíme extrapolaci: ze známé pozice se posouváme dopředu na základě nějakého odhadu průběhu funkce.

Klasifikace num. metod: **jednokrokové** (vychází jen z aktuálního stavu) vs. **vícekrokové** (používají historii stavů – mají „paměť"; problém: kde vzít *historii* na začátku výpočtu -> kombinace s jednokrokovými).

Další dělení num. metod: **explicitní** (výsledek získáme dosazením do vzorce) vs. **implicitní** (vyžadují v každém kroku řešení algebraických rovnic; typicky na *tuhé systémy*!).

Eulerova metoda: $\mathbf{y(t + h) = y(t) + h * f(t, y(t))}$

(musí platit Lipschitzova podmínka – „nesmí to být moc rozlítané").

Runge-Kutta: skupina metod různých řádů (RK1=Euler, RK2, RK4, RK8…), ale stále to jsou *jednokrokové* metody. Několik vztahů, které počítají hodnoty **f** v různých mezikrocích – zejm. u vyšších řádů ale těch vyhodnocení f probíhá (příliš?) moc.

RK2: $k_1 = h * f(t, y(t))$; $k_2 = h * f(t + h/2, y(t) + k_1/2)$; $y(t + h) = y(t) + k_2$

RK stačí na spoustu věcí, ale ne např. na tuhé systémy (el. obvody s par. kapacitami).

Další varianty: Dormand-Prince 45 – kombinace RK4 a RK5 + odhady chyby pro dynamické prodlužování a zkracování kroku (*adaptivní RK metody*)

Dokročení: ideálně udělat před spočítáním kroku a spekulativně – dopředu:

```
while (time < timeMax) {
  if (time + step * 1.01 > timeMax)
    sim_step(timeMax - time);
  else
    sim_step(step);

  time += step;
```

}

Vícekrokové metody: využívají hodnoty zapamatované z předchozích kroků – zpřesňují výsledek. Obvykle se **f** vyhodnotí jen jednou (lepší než RK!). Podle předchozích hodnot se interpoluje polynonem -> extrapolace.

Řád vícekrok. metody: počet hodnot **f** využitých v jednom kroku (zapamatované + 1).

Vícekrok. metody typu prediktor/korektor: vyhodnotím y_{n+1} (v čase t+h) a zde znovu vyhodnotím f -> f_{n+1}. Dosadím do korekčního vzorce a výsledek teprve použiji jako y_{n+1}.

Problémy vícekrok. metod: start metody (ale existují i samostartující), je nutné, aby zůstal stejný krok (problém u kombinovaných simulací). U aproximačních polynomů dochází k nestabilitám (může se to zvlnit).

Vícekrok. metody jsou užitečné např. v **n-body** problémech – miliony, miliardy bodů, které na sebe působí -> počet výpočtů f je významný.

Příklad vícekrok. metody: **Adams-Bashforth**(-Moulton: s predikcí/korekcí).

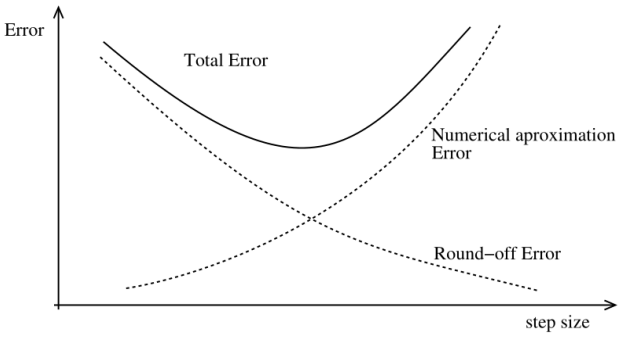
Chyby num. metod: **lokální** (zaokrouhlovací, aproximace), **akumulované**.

Lokální – zaokrouhlovací chyby: vzniká kvůli reprezentaci desetinných čísel v počítači (problém s operací sčítání) – zvyšují se se **zmenšováním kroku**.

Lokální – aproximační chyby: aproximuji průběh, který odpovídá polynomu potenciálně až nekonečného řádu, polynomem 4. řádu, logicky vznikají chyby tohoto odhadu. *Pokud řeším něco parabolického, RK2 bude stačit.* Aproximační chyby se zvyšují se **zvětšováním kroku**.

Příliš malý krok i příliš velký krok jsou špatně – jde o to, co nám stačí.

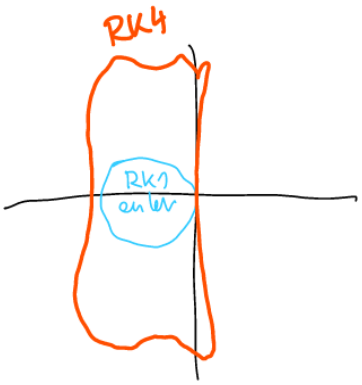
Akumulovaná chyba: z průběhu chyby v rámci výpočtu vezmu **maximum**.



Stabilita num. metod: nestabilita -> chyba. Numerické metody diskretizují řešení, vyšetřujeme, jestli „neutíká do nekonečna“ – nezačne nekontrolovaně růst nad nějaké meze. Většina fyzikálních systémů je tohoto typu – kyvadlo se kýve, kýve a časem zastaví. Když ho modelujeme s využitím num. metod, chová se to stejně?

Numericky nestabilní řešení: „rozkmitá se, když nemá“.

Oblast stability v komplexní rovině: vlastní číslo matice problému * délka kroku: **λ*h**.



Kruhový test má vlastní čísla na imaginární ose – RK1 to nedá, RK4 ano (s určitou délkou kroku).

Každá integrační metoda má nějakou svou oblast stability.

Implicitní metody (např. zpětný Euler) mají oblast stability *inverzní* – ale příliš velkou chybu.

Tuhé systémy: velmi rozdílné časové konstanty (vlastní čísla matice velmi daleko od sebe) – např. el. obvody, ve kterých řešíme parazitní kapacity (jsou tam, ale jsou řádově menší než u běžných součástek). Klasické explicitní metody se na to nehodí, je lepší použít implicitní metody (které jsou ale složité).

Příklad rovnice tuhého systému: y'' + 1001 y' + 1000 y = 0.

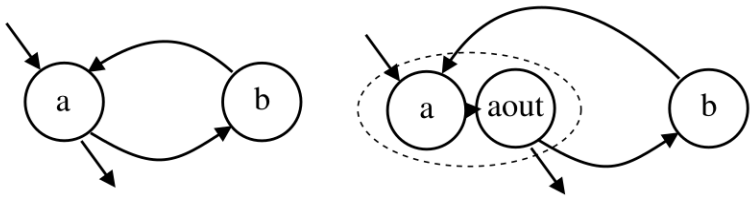
Výběr integrační metody: obvykle stačí RK4; někdy hodí vícekrokové, ale nesmí tam být mnoho *nespojitostí*; speciální metody pro tuhé systémy.

Obecný alg. řízení spojité simulace:

1. Inicializace (počáteční podmínky)
2. Dokud není konec simulace (překročený čas):
    a. Zaznamenat aktuální hodnoty, stav (občas)
    b. Vyhodnotit derivace, vypočítat nový stav – udělám krok numerické metody
    c. Posunout modelový čas
3. Ukončení, výstup

Uspořádání funkčních bloků (těch, které nemají stav): problém závislosti mezi funkčními bloky – v jakém *pořadí* provádět výpočet. Ideální stav: nemáme cykly, výpočet lze uspořádat (v okamžiku výpočtu známe všechny vstupy). Když to nejde, máme **algebraickou**/**rychlou smyčku** – silné komponenty grafu.

Řešení: buď iterační (rozpojením speciálním blokem), nebo přepracováním modelu (někam strčíme integrátor nebo to strčíme do krabice a najdeme společné analytické řešení).



a, aout řeším tak dlouho, dokud se nepřiblíží k sobě

Newtonova metoda: hledá řešení rovnice f(x) = 0 se znalostí f'(x). Začínáme od počátečního odhadu řešení $\mathbf{x_0}$, pak $\mathbf{x_{k+1} = x_k - f(x_k) / f'(x_k)}$, provádíme tak dlouho, dokud se nedostaneme dost k nule.

Topologické řazení: první v pořadí jsou uzly, které na ničem nezávisí, pak narazíme na smyčky, ty musíme vyřešit.

==== STOP ====

Kombinovaná (hybridní) simulace: propojuje spojitou a diskrétní simulaci – vkládá do spojité simulace diskrétní události; řeší *stavové podmínky* a detekci jejich změn – stavové události. Řešíme problém zkracování kroku, skokové změny spojitého stavu.

Při zpracovávání *diskrétních* událostí musíme ve *spojité* simulaci ***dokročit*** na aktuální čas (čas diskrétní události).

Celá řada zapeklitých problémů: příliš krátké kroky, jejich zanedbávání (kolik ještě můžeme zanedbat?), uspořádání stavových podmínek (jedna může měnit vstupy jiných), nepřesnosti…

Stavové události (podmínky): nastanou po dosažení zadané hodnoty spojité veličiny (= změna stavové podmínky), nelze je naplánovat.

==== STOP ====

Úrovně popisu číslicových systémů:

- elektrická (tranzistory, kondenzátory, op. zesilovače; spojité modely; SPICE)
- logická (nuly, jedničky a něco dalšího; hradla, klopáky; diskrétní modely; VHDL)
- RTL (register-transfer – mezireg. přenosy; čítače, řadiče, ALU; diskrétní modely)

- systémová (procesory, paměti; diskrétní modely; simulace propustnosti, rychlosti přenosu dat, výkonnosti…)

Modely signálu: dvojhodnotové, trojhodnotové, 5hodnotové (X, Rise, Fall), 9hodnotové.

Více hodnot = přesnější popis, ale pomalejší.

Modely zpoždění: nulové (nereálné, nedává smysl v systému se zpětnými vazbami), jednotkové (nereálné), přiřaditelné (zvlášť pro 0->1, 1->0), přesné (rozsah od–do).

Často je nutné počítat i se zpožděním na spojích.

Řízení simulace číslicových obvodů: next-event s modifikacemi (selektivní sledování; událost = změna vstupu nějakého hradla), případně nějaké kompilované modely. Vyžaduje *pořádný kalendář* (hodně záznamů).

```
while (!Calendar.Empty()) {
  nastavení modelového času
  M = {}
  for (e in Calendar.Events(T)) {
    e.Update();
    M.AddAll(e.ConnectedNodes());
  }
  for (p in M) {
    p.Update();
    if (p.OutputChanged()) {
      Delay(p);
      Calendar.AddAll(p.ChangedOutputs());
    }
  }
}
```

Simulace poruch: trvalé, přechodné, zkraty, šíření modelem, detekce.

==== STOP ====

Heterogenní modely: použití více forem popisu pro různé části modelu. (spojitý popis -> vzorkování -> číslicové řízení, fuzzy logika, neuronky -> D/A převod…)

Fuzzy logika: využívá ohodnocování výroků mírou pravdivosti, umožňuje do popisů systémů zanést neurčitost. Tam, kde v booleovské logice mohou být vstupům přiřazeny pouze hodnoty 0, nebo 1, může být ve fuzzy logice přiřazena libovolná hodnota v intervalu <0, 1> – **míra příslušnosti** určité **fuzzy množině**. (Například pro teplotu mohou být definovány množiny „zima“, „běžná teplota“, „horko“.

Fuzzifikace: převod vstupu na míru příslušnosti.

Poté aplikujeme jednoduchá pravidla určující chování, můžeme použít různé operace. Výstupy pravidel agregujeme a defuzzifikujeme (na to jsou různé metody), čímž získáme opět „ostrou“ hodnotu.

Optimalizace: cílem je najít *optimální* parametry modelu. Je nutné zadefinovat nějakou hodnoticí funkci, která říká, co vlastně hledáme. Různé metody, často inspirované přírodou: gradientní, simulované žíhání, genetické, mravenci…

Některé modely mají tisíce parametrů -> optimalizace je velmi složitý problém.

Hledání optima spadá do oblasti matematiky, které se říká *operační výzkum*.

Lineární programování: všechno je lineární.

Minimalizace je algoritmus, který počítá **x**: F(**x**) = min ∧ C(**x**), kde **x** je vektor hodnot parametrů, F je cenová funkce a C reprezentuje různá omezení (constraints). Problém: můžeme jednoduše spadnout do lokálního minima (=> různé metody, jak se tomu vyhnout).

==== STOP ====

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

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

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

==== STOP ====

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

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

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

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

pro M/M/1:

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

DOPLNIT PRO M/M/2