IMS
Příklady na semestrálku

Přebráno z Fitušky (fórum a wiki)

19. ledna 2016

1 Programování na papír

1.1 Monte Carlo (nick)

Napište v jazyku C funkci `double MCIntegral()` pro výpočet N-násobného určitého integrálu funkce `double f(double x[], unsigned N)` (podle všech X i , i náleží 0,...,N-1) metodou Monte Carlo (integrál musí být N-násobný, jinak 0 bodů) s přesností přibližně 1%. Meze integrálu jsou uloženy v globálním poli struktur `Range` s položkami `.min` a `.max`. Předpokládejte, že neznáte rozsah funkčních hodnot funkce `f`. Máte k dispozici pouze funkci `double Random(void)` vracející pseudonáhodná čísla v rozsahu $\langle 0, 1)$, `f` a žádnou jinou

```
1  // obecny vzorec pro vypocet presnosti: 1/sqrt(n)
2  // presnost = 1% = 1/100 => n = 10000 pokusu
3  double MCIntegral() {
4      unsigned pocetPokusu = 10000;
5      double suma = 0.0;
6
7      for (unsigned i=0; i<pocetPokusu; i++) {
8          double x[N];
9          // urceni souradnic nahodneho bodu x
10         for (unsigned j=0; j<N; j++) {
11             x[j] = Range[j].min + Random() * (Range[j].max - Range[j].min);
12         }
13         // pricteni hodnoty nahodneho bodu do celkove sumy
14         suma += f(x, N);
15     }
16
17     double refPlocha = 1.0;
18     // vypocet rozmeru prostoru ve kterem se generovaly body
19     for (unsigned k=0; k<N; k++) {
20         refPlocha *= Range[k].max - Range[k].min;
21     }
22
23     return refPlocha * (suma / pocetPokusu);
24 }
```

1.2 Kongruentní generátor pseudonáhodných čísel (nick)

V ISO C napište funkci double `Random(void)` ve které bude váš vlastní kód kongruentního generátoru s rozsahem $\langle 0, 1)$, konstanty `A`, `B`, `M` máte k dispozici. Máte zadánu funkci hustoty pravděpodobnosti $f(x)$ (nakreslete ilustrační obrázek).

$$f(x) = \frac{x}{2} \text{ pro x na intervalu } \langle 0, 2) \tag{1}$$

$$f(x) = 0 \text{ jinak} \tag{2}$$

a) Napište funkci double `RandGenInv(void)`, která generuje číslo se zadaným rozložením s využitím metody inverzní transformace (jinak -5 bodů).

b) Napište funkci double `RandGenReject(void)`, která generuje číslo se zadaným rozložením s využitím metody vylučovací (jinak -5 bodů).

c) Vypočtěte pravděpodobnost, že náhodná proměnná X s hustotou pravděpodobnosti $f(x)$ zadanou výše bude mít hodnotu 0.5.

d) Vyjádřete distribuční funkci obecného spojitého rozložení $F(x)$ pomocí její funkce hustoty pravděpodobnosti $f(x)$.

```
1 double Random() {
2     static unsigned long ix = SEED;
3     ix = (ix * a + b) % M;
4     return ix / (double) M;
5 }
```

Řešení

a) Rovnice

$$F(x) = \int \frac{x}{2} dx = \frac{x^2}{4} \tag{3}$$

$$F^{-1}(x) = \sqrt{4x} = 2\sqrt{x} \tag{4}$$

Program

```
1 double RandGenInv(){
2     return sqrt(Random()) * 2.0;
3 }
```

b) Program

```
1 double f(double x) { // funkce hustoty
2     if(0.0 <= x && x < 2.0)
3         return x / 2.0;
4     else
```

```
 5         return 0.0;
 6 }
 7 double RandGenReject() {
 8     double x,y;
 9     do {
10         x = Random()*2;
11         y = Random()*1;
12     } while (y > f(x)); // trefili jsme se nad funkci hustoty -> znovu
13     return x;
14 }
```

c)

$$\int_{0.5}^{0.5} f(x)dx = 0$$

Poznámka: Pravděpodobnost jednoho konkrétního čísla z nekonečna (na spojitém intervalu je nekonečné množství reálných čísel) je vždy nulová. Aby byla nenulová, musí to být hodnota z intervalu (pak se nebudou meze integrálu rovnat).

d)

$$F(x) = \int_{-\infty}^{x} f(t)dt$$

Poznámka: t je pomocná proměnná zavedená pro odlišení x, které je parametrem funkce $f(x)$ a podle kterého se derivuje, od x, které je vstupním parametrem funkce $F(x)$. Nesouvisí s žádnou proměnnou mimo tento výraz.

1.3 Algoritmy řízení simulace

1.3.1 Řízení diskrétní simulace – Next-event (skeWer)

```
1 Inicializace_kalendare
2 while(kalendar_neprazdny) {
3    vyjmi_prvni_zaznam_z_kalendare
4    if (aktivacni_doba_zaznamu > doba_konce_simulace) break;
5    cas = aktivacni_doba_zaznamu;  //hlavne neprehodte tyhle dva radky
6    proved_udalost  //jinak...
7 }
8 cas = doba_konce_simulace
```

1.3.2 Řízení spojité simulace (Royce)

```
1 inicializace poc. stavu a promennych a casu;
2 while (cas < koncovy_cas) {
3     Print_results();
4     Update_integrators();
5     if (cas + krok > koncovy_cas)
6         krok = koncovy_cas - cas;
7     euler();
8     cas += krok;
9 }
```

1.3.3 Řízení kombinované simulace (skeWer)

```
1 inicializace poc. stavu a promennych a casu;
2 while (cas < koncovy_cas) {
3   if (kalendar_empty) {
4     cas = koncovy_cas;
5     konec_simulace();
6   }
7   vyber_zaznam_z_kalendare();
8   while ( (cas + krok < koncovy_cas) && (cas + krok <
        aktivacni_cas_pristi_udalosti) ) { //"neprekroceni" udalosti / konce
9     krok_numericke_metody();
10    cas += krok;
11  }
12  //v tomhle miste jsme bud kousek pred udalosti (kousek <= normalni_krok),
        nebo kousek pred koncem simulace
13  //cili pred provedenim udalosti / koncem simulace musime udelat jeste
        jeden krok, ale jeho velikost urcime
14  //podle toho, jestli budeme provadet udalost nebo koncit
15  if (aktivacni_cas_pristi_udalosti > koncovy_cas)  { //prvni udalost lezi
        mimo simulaci -> kaslem na ni
16    krok = koncovy_cas - cas;
17    krok_numericke_metody();
18    cas += krok; //nebo: cas = koncovy_cas
19    konec_simulace();
20  } else {
21    krok = aktivacni_cas_pristi_udalosti - cas;
22    krok_numericke_metody();
23    cas += krok; //nebo: cas = aktivacni_cas_pristi_udalosti
24    proved_udalost();
25    krok = normalni_delka_kroku;
26  }
27 }
```

1.4 Runge-Kutta

1.4.1 RK2 (nick)

V jazyce ISO C napište funkci `Step_RK2 (double t, double st[], double in[], unsigned N, double stepSize)`, které předáte modelový čas `t`, stav všech integrátorů v poli `st`, jejich počet `N` a délku kroku `stepSize`.
Vzorec definující metodu:

$$k_1 = hf(t, y(t)) \tag{5}$$

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

$$y(t + h) = y(t) + k_2 \tag{7}$$

Chování modelu je popsáno funkcí `Dynamic (double t, double st[], double in[], unsigned N)` ve které se vypočítávají vstupy integrátorů `in[i]` pro daný stav `st[i]` a čas `t`. Počáteční podmínky se nastavují funkcí `void InitModel(double st[], unsigned N)`, tyto dvě funkce nepište.

```
1  void Run(double t1, double t2){
2      unsigned N = 2;
3      double st[N], in[N], stepSize = 0.1;
4      InitModel(st, N);
5      for(double t=t1; t < t2; t+=stepSize){
6          if(t + stepSize > t2) stepSize = t2 - t; // dokrocime
7          printf("%lf %lf %lf\n", t, st[0], st[1]); // vypis
8          Step_RK2(t, st, in, N, stepSize); // vypocet dalsiho stavu
9      }
10 }
```

Definujte všechny potřebné proměnné a napište funkci `void Run(double t1, double t2)` implementující algoritmus řízení spojité simulace s využitím `Step_RK2`. Nezapomeňte „dokročit“ na čas konce simulace $t2$. Řešení (max 30 řádků) velmi stručně komentujte.

```
1  void Step_RK2(double t, double st[], double in[], unsigned N, double
       stepSize){
2      double startSt[N], k1[N], k2[N];
3      for(unsigned i=0; i < N; i++){
4          startSt[i] = st[i];
5      }
6      Dynamic(t, st, in, N);              // in = f(t,y(t))
7      for(unsigned i=0; i < N; i++){
8          k1[i] = stepSize*in[i];         // k1 = h*f(t,y(t))
9          st[i] = startSt[i] + k1[i]/2;   // st = y(t)+k1/2
10     }
11     Dynamic(t+stepSize/2, st, in, N);   // in = f(t+h/2,y(t)+k1/2))
12     for(unsigned i=0; i < N; i++){
13         k2[i] = stepSize*in[i];         // k2 = h*f(t+h/2,y(t)+k1/2)
14         st[i] = startSt[i] + k2[i];     // y(t+h) = y(t) + k2
15     }
16 }
```

1.4.2 RK4 (nick)

Stejné zadání jako RK2, ale s rovnicemi pro RK4.

$$k_1 = hf(t, y(t)) \tag{8}$$

$$k_2 = hf(t + \frac{h}{2}, y(t) + \frac{k_1}{2}) \tag{9}$$

$$k_3 = hf(t + \frac{h}{2}, y(t) + \frac{k_2}{2}) \tag{10}$$

$$k_4 = hf(t + h, y(t) + k_3) \tag{11}$$

$$y(t + h) = y(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{12}$$

Řešení

```
1 void Step_RK4(double t,double st[],double in[],unsigned N,double h){
2     double startSt[N], k1[N], k2[N], k3[N], k4[N];
3
4     for(unsigned i=0; i<N; i++){
5         startSt[i] = st[i];
6     }
7
8     Dynamic(t, st, in, N);
9     for(unsigned i=0; i<N; i++){
10        k1[i] = h * in[i];
11        st[i] = startSt[i] + k1[i]/2;
12    }
13
14    Dynamic(t+h/2, st, in, N);
15    for(unsigned i=0; i<N; i++){
16        k2[i] = h * in[i];
17        st[i] = startSt[i] + k2[i]/2;
18    }
19
20    Dynamic(t+h/2, st, in, N);
21    for(unsigned i=0; i<N; i++){
22        k3[i] = h * in[i];
23        st[i] = startSt[i] + k3[i];
24    }
25
26    Dynamic(t+h, st, in, N);
27    for(unsigned i=0; i<N; i++){
28        k4[i] = h * in[i];
29        st[i] = startSt[i] + k1[i]/6 + k2[i]/3 + k3[i]/3 + k4[i]/6;
30    }
31 }
```

2 Markovské procesy

2.1 D/D/x, kalendáře

2.1.1 D/D/x bez poruchy (wiki)

D/D/2 popsat stav simulace, kalendář, pořadí událostí a frontu v čase $6.5s$, začíná se v $0s$, generuje se každou $1s$ a zpracování trvá $3s$.

Časový diagram:

t	0	1	2	3	4	5	6	7	8
Z0	P0	P0	P0	P2	P2	P2	P4	P4	P4
Z1		P1	P1	P1	P3	P3	P3		
Q			P2	P3	P4	P4	P5		
						P5	P6		

Seznam událostí:

0. vygenerování P0, P0 obsazuje Z0
1. vygenerování P1, P1 obsazuje Z1
2. vygenerování P2, P2 jde do fronty
3. konec obsluhy P0, P2 obsazuje Z0, vygenerování P3, P3 jde do fronty
4. konec obsluhy P1, P3 obsazuje Z1, vygenerování P4, P4 jde do fronty
5. vygenerování P5, P5 jde do fronty
6. konec obsluhy P2, P4 obsazuje Z0, vygenerování P6, P6 jde do fronty

   ---

   (dále jsou již jenom naplánované události)
7. konec obsluhy P3, vygenerování P7
8.
9. konec obsluhy P4

Fronta:

→	P5
	P6

Obsah kalendáře

čas	priorita	událost
7	0	P3→Release
7	0	Generator→Activate
9	0	P4→Release

2.1.2 D/D/x s poruchou (mesma)

D/D/3, fronta LIFO, příchody každou 1s, obsluha trvá 3s. V čase 2s přijde porucha do Z0 a trvá 2s. Vyjádřit v čase 4.5s.

Časový diagram:

t	0	1	2	3	4	5	6
Z0	P0	P0	x	x	P0		
Z1		P1	P1	P1	P3	P3	P3
Z2			P2	P2	P2		
Q				P3	P4		

Seznam událostí:

0. vygenerování P0, P0 obsazuje Z0
1. vygenerování P1, P1 obsazuje Z1
2. porucha obsazuje Z0, vygenerování P2, P2 obsazuje Z2
3. vygenerování P3, P3 jde do fronty
4. konec poruchy Z0, konec obsluhy P1, P3 obsazuje Z1, vygenerování P4, P4 jde do fronty

   ---

   (dále jsou již jenom naplánované události)
5. konec obsluhy P2, konec obsluhy P0, vygenerování P5
6.
7. konec obsluhy P3

Fronta: | → P4 |

Obsah kalendáře

čas	priorita	událost
5	0	P2→Release
5	0	P0→Release
5	0	Generator→Activate
7	0	P3→Release

3 Teorie (*wiki*)

Abstraktní model zjednodušený pospis zkoumaného systému

Buňka (cell) základní element, může být v jednom z konečného počtu stavů (0/1)

Celulární automat je souhrné označení pro určitý typ fyzikálního modelu reálné situace, slouží k časové i prostorové diskrétní ideální modelaci fyzikálních systémů, kde hodnoty veličin nabývají pouze diskrétních hodnot

Chování systému každému časovému průběhu vstupní veličiny přiřazuje časový průběh výstupních veličin, je dáno vzájemnými interakcemi mezi prvky systému a mezi systémem a jeho okolím

Diskrétní simulace simulace s dikrétními prvky a časem. Experimentování se simulačním modelem.

Markovova vlastnost následující stav procesu závisí jen na aktuálním stavu (ne na minulosti)

Markovský proces náhodný diskrétní proces se spojitým časem, který splňuje markovovu vlastnost

Markovův řetězec náhodný diskrétní proces s diskrétním časem, který splňuje markovovu vlastnost, ekvivalentem je konečný automat s pravděpodobnostmi přechodů

Model napodobenina systému jiným systémem

Modelování proces vytváření abstraktního modelu

Modelový čas časová osa modelu, může být reálný, zrychlený, zpomalený

Monte Carlo metoda používaná k výpočtu určitých integrálů, určená pro simulaci systémů, které jsou založeny na experimentování se stochastickými modely s využitím generátorů náhodných čísel

Okolí (neighbourhood) několik typů, liší se počtem okolních buněk, se kterými se pracuje

Pole buněk (lattice) n–rozměrné, obvykle 1D nebo 2D, rovnoměrné dělení prostoru, může být konečné nebo nekonečné

Pravidla (rules) funkce stavu buňky a jejího okolí definují nový stav buňky v čase $s(t+1) = f(s(t), Ns(t))$

Petriho síť graf popisující stavy a přechody mezi stavy, umožňuje znázornění paralelismu a nedeterminismu

Priorita obsluhy proces s vyšší prioritou obsluhy než současný může vyhodit současný proces a sám obsadit zařízení

Priorita procesu pro řazení ve vnější frontě u zařízení a v kalendáři (primárně podle času a sekundárně podle priority)

Proces posloupnost událostí

Reálný čas probíhá v něm skutečný děj v reálném systému

Řád metody numerické integrace stupeň polynomu N (Taylorův rozvoj) definuje tzv. řád metody, tímto polynomem aproximujeme na základě aktuální hodnoty

Rychlá smyčka tvoří cyklus v orientovaném grafu závislostí funkčních bloků.

SHO systémy obsluhující zařízení, která poskytují obsluhu transakcím, součástí SHO jsou transakce, fronty, obslužné linky

Simulace experimentování s modelem, získávání nových znalostí

Simulační model abstraktní model zapsaný kódem

Spojitý systém všechny prvky systému mají spojité chování (nemění své parametry skokově)

Statistiky vytížení zařízení, délky front, doby čekání ve frontách, využití kapacity skladů, celková doba transakce strávené v systému, maximum, minimum, střední hodnoty, směrodatná odchylka, rozptyl, ...

Stavová podmínka diskrétní podmínka ve spojité simulaci, která může nějakým způsobem ovlivnit model (jiné rovnice pro výpočet kapaliny a páry), je citlivá na hranu, takže se vykonává pouze při změně, na základě její změny vyvolám stavovou událost

Stavová událost akce, která je podmíněna změnou pravdivostní hodnoty stavové podmínky (např.: otočení hodnoty integrátoru), diskrétní skoková změna systému

Strojový čas čas CPU spotřebovaný na výpočet programu (závisí na složitosti simulačního modelu, nesouvisí přímo s modelovým časem)

Systém soubor elementárních částí, které mají mezi sebou vazby

Třídy statistik stat, tstat (bere v potaz čas), histogram, mají společné operace s.Clear() (inicializace), s.Output() (tisk), s(x) (záznam hodnoty x)

Událost změna stavu diskrétního systému, jednorázová, nepřerušitelná

Validace modelu ověřování platnosti modelu a proces, kdy se snažíme dokázat, že pracujeme s modelem adekvátním modelovanému systému

Verifikace modelu dokazování izomorfnosti (rovnosti) abstraktního modelu s modelem simulačním, jinak řečeno ověřování korespondence abstraktního modelu se simulačním modelem