FAT-112

الفبایِ مونت‌کارلو با متمتیکا

امیرحسینِ فتح‌اللهی

چکیده: مفاهیم و عناصرِ اولیه‌ی مونت‌کارلو معرفی میشن. مثالهایِ محاسباتی بر اساسِ نرم‌افزارِ محاسباتیِ متمتیکا ارائه میشن.

0 مقدمه

شاید یه حس مشترک بین تمومِ اونائی که روشِ مونت‌کارلو رو یادمیگیرن وجود داشته باشه. به نظر شگفت‌انگیزه که میشه به یک مسئله‌ی واحد، مثلاً حساب‌کردنِ سطحِ زیرِ یک منحنی، دو نگاهِ متفاوت داشت که اگرچه هر دو دقتِ مورد نظرِ ریاضی‌دونها را دارن، اما یکی‌شون مثلِ یه بازیه و حتا تا حدّی مفرّح بطوری که میشه اون رو با بچه‌هایِ بازیگوشِ مدرسه هم اجرا کرد! جدای این تفاوت، یکی از این نوع نگاههاس که میتونه محاسباتی رو با کامپیوترهایِ امروزی ممکن کنه که با نگاهِ دیگه، حتا اگر قدرتِ کامپیوترها صد برابر هم بشه، ممکن نیس. نمونه‌اش این که شما با روش مونت‌کارلو میتونید گذارِ فازِ مدلِ آیزینگ دو بُعدی رو در یک شبکه‌ی 10×10 و در محاسباتی به طولِ دو-سه دقیقه ببینید، ولی اگر قرار باشه به روشِ معمول و جمعِ دقیق ببینید یک شبکه‌ی $10^{10} \times 10^{10}$ هم، به فرضِ این که کامپیوتری داشته باشین که انجامش بده، کافی نیس!

در اینجا قضایایِ ریاضی و اثباتِ اونها نمیان. ادعاها با مثالهایِ محاسباتی، اثبات که نَه، موجّه میشن! مطالبی که میاد جسته-گریخته خوندم که موثرترینشون رو آخرِ نوشته میارم. محاسباتی که در ادامه میان با نرم‌افزارِ محاسباتیِ متمتیکا 7 انجام شده که صریحاً اینجا آوردمشون.

1 شروعِ بازی!

فرض کنید میخوایم انتگرالِ زیر رو حساب کنیم:

$$\int_0^1 \exp(x^2)\,\mathrm{d}x \tag{1}$$

که در متمتیکا اینجوری میشه:

```
f1[x_]:=Exp[x^2];
exact1 = NIntegrate[f1[x], {x, 0, 1}]
```

1.46265

میدونید که این انتگرال جوابِ تحلیلی نداره، و میخوایم خودمون به جایِ متمتیکا اون رو با تبدیلِ انتگرال به جمع (که بهش تقریبِ ریمن میگن) حساب کنیم. در این تقریب انتگرال اینجوری جایگزین میشه:

$$\int_a^b f(x)\,\mathrm{d}x \;\to\; d\,\sum_{n=1}^{M} f(x_n) \tag{2}$$

که در اون:

$$x_n = a + n\,d, \qquad d = \frac{b-a}{M} \tag{3}$$

که بطورِ معادل میشه:

$$\int_a^b f(x)\,\mathrm{d}x \;\to\; \frac{b-a}{M}\,\sum_{n=1}^{M} f(x_n) \tag{4}$$

برایِ انتگرالی که داشتیم چند نمونه از اینطور تقریب‌زدن در زیر اومده با مقادیرِ

$$d = 0.5, 0.1, 0.01, 0.001, 0.0001$$

که در متمتیکا اینطور میشه:

```
Table[{approx1 = Sum[f1[n]d, {n, 0, 1, d}]//N;
Print[d, " ", approx1, " error=", (approx1 - exact1)/exact1 * 100]},
{d, {0.5, 0.1, 0.01, 0.001, 0.0001}}];
0.5  2.50115  error=71.0013
0.1  1.65309  error=13.02
0.01  1.48129  error=1.27417
0.001  1.46451  error=0.127139
0.0001  1.46284  error=0.0127111
```

در بالا مقدارِ d، نتیجه‌ی محاسبه و درصدِ خطا اومده؛ آخری واقعاً نزدیکه که در اون فاصله‌ی 0 تا 1 ده‌هزار قسمت شده!

حالا بیاید دوباره از رابطه‌ی (4) استفاده کنیم ولی اینبار x_n ها رو الأبختکی از فاصله‌ی 0 تا 1 بَرداریم؛ چند تا؟ برایِ شروع 100 تا. اجرایِ پنج دفعه‌اش اینطوری میشه:

```
xr:=RandomReal[{0, 1}];
M = 100;
Table[MCInt1 = Mean[Table[f1[xr], {m, 1, M}]]; Print[MCInt1], {5}];
1.44552
1.37184
1.41913
1.47244
1.49409
```

با نتیجه‌ی دقیقش مقایسه کنید؛ بَدَک نیستن! اگر اجرایِ دَه دفعه با 1000 بار رو میانگین بگیرین (یا در واقع 10000 تاش) میشه:

```
xr:=RandomReal[{0,1}];
M = 1000;
Mean[Table[MCInt1 = Mean[Table[f1[xr],{m,1,M}]],{10}]]
```
1.46421

میشه اثبات کرد که نتیجه در تعدادِ به اندازه‌ی کافی بزرگش به دلخواه میتونه دقیق بشه.

میتونید بپرسید که «ما که جواب رو با دقتِ خوبی داریم؛ این کارا برایِ چیه؟». جواب اینه که «چون جواب رو همیشه به این خوبی یا راحتی نداریم و این روشِ جدید ممکنه کمکِ بزرگی باشه». اما قبلش بیاید سراغِ یه مثالِ دیگه بریم. همون انتگرالِ اولی ولی شش‌گانه‌ش که در متمتیکا میشه:

```
f6[x1_,x2_,x3_,x4_,x5_,x6_]:=Exp[x1^2+x2^2+x3^2+x4^2+x5^2+x6^2];
exact6 = NIntegrate[f6[x1,x2,x3,x4,x5,x6],{x1,0,1},{x2,0,1},{x3,0,1},{x4,0,1},
{x5,0,1},{x6,0,1}];//Timing
Print[exact6]
```
{20.078, Null}

9.79142

که در واقع باید توانِ شش نتیجه‌ی قبلی باشه: $1.46265^6 = 9.79135$. در بالا دستور Timing بکار رفته و نشون میده که حدود بیست ثانیه طول کشیده تا جواب بیرون بیاد. اگر انتگرال هفت یا هشت‌گانه بود اینقدر طول میکشید که حوصله‌تون جلوی مونیتور سر میرفت! حالا بیاید از روش گسسته‌سازی که اول گفتیم انتگرال بالا رو حساب کنیم. برای سلولهای شش-بُعدی به ابعادِ 0.1 اینطوری میشه:

```
d = 0.1;
approx6 = d^6 Sum[f6[n1,n2,n3,n4,n5,n6],{n1,0,1,d},{n2,0,1,d},{n3,0,1,d},
{n4,0,1,d},{n5,0,1,d},{n6,0,1,d}];//Timing
Print[approx6," error=",(approx6-exact6)/exact6*100]
```
{12.652, Null}

20.4069 error=108.416

جواب میشه حدودِ بیست با بیش از صددرصد خطا! برای این نتیجه جمع رویِ $(1/0.1)^6$ سلول یعنی یک‌میلیون سلول انجام شده که همونطور که در بالا اومده بیش از دوازده ثانیه طول کشیده! به همین دلیل هم اندازه‌ی سلولها را کوچکتر نگرفتیم. مثلاً اگر d را 0.01 میگرفتیم باید رویِ 10^{12} سلول جمع میزدیم که اگرچه جواب بهتر از بیست میشد ولی زمانش حدود یک‌میلیون برابرِ دوازده ثانیه، یعنی حدود صد روز طول میکشید! پس تا همینجا اشکالهایِ اساسیِ روش معمول تبدیلِ انتگرال به جمع رو میبینیم. اولیش اینه که در ابعادِ بیشتر خطا به سرعت زیاد میشه. بطورِ دقیقتر، در بُعدِ D خطا، اونهم برای توابعِ با مشتقِ کراندار، برایِ تعدادِ M سلول

اینطوریه:

$$\text{error} \propto \frac{\sqrt{D}}{M^{1/D}} \tag{5}$$

که نشون میده اولاً، خطا با جذرِ بُعد زیاد میشه، و از اون بدتر، در بُعدِ بالاتر زیادکردنِ تعدادِ سلول خطا رو خیلی کم کاهش میده. برایِ مثال، در بُعدِ شش، اگر تعدادِ سلولها رو دو برابر کنید، زمانِ محاسبه دو برابر میشه، ولی خطا به نسبتِ $2^{1/6} = 1.12$ یعنی حدودِ $\frac{1.12-1}{1} \times 100 = 12\%$ کم میشه. در موردِ مثالِ خودمون بیست در دوازده ثانیه رو میکنه حدود نوزده در بیست‌وچهار ثانیه (در مقایسه با جوابِ 9.79). حالا بیاید از بازیی که تازه یاد گرفتیم استفاده کنیم؛ اجرای پنج دفعه‌ش هرکدوم صد انتخابِ شش-تائی میشه:

```
xr:=RandomReal[{0,1}];
M = 100;
Table[MCInt6 = Mean[Table[f6[xr,xr,xr,xr,xr,xr],{m,1,M}]];Print[MCInt6],{5}]//
Timing
9.46752
8.74826
10.9643
8.67465
10.6157
{0.031,{Null,Null,Null,Null,Null}}
```

همه‌شون حول‌وخوشِ جوابن؛ در چه زمانی؟ مجموعاً 0.03 ثانیه! اجرایِ پنج دفعه‌ی 10000 تائی مجموعاً در زمانِ نیم ثانیه میشه:

```
xr:=RandomReal[{0,1}];
M = 10000;
Table[MCInt6 = Mean[Table[f6[xr,xr,xr,xr,xr,xr],{m,1,M}]];Print[MCInt6],{5}]//
Timing
9.72741
9.82157
9.80803
9.93387
9.74117
{0.515,{Null,Null,Null,Null,Null}}
```

میانگینِ ده اجرا در تقریباً یک ثانیه میده:

```
xr:=RandomReal[{0,1}];
M = 10000;
```

```
Mean[Table[MCInt6 = Mean[Table[f6[xr, xr, xr, xr, xr, xr], {m, 1, M}]], {10}]]
9.79694
```

که واقعاً خوبه. در همینجا یک مزیتِ روشِ جدید معلوم میشه: خطاش، برعکسِ روشِ جمعِ معمولی، مستقل از بُعده. بطورِ دقیقتر، مستقل از بُعدِ انتگرال و مثلِ اکثرِ پدیده‌هایِ آماری با $\frac{1}{\sqrt{M}}$ متناسبه. یادآوری بشه در روشِ قبلی بود $\frac{\sqrt{D}}{M^{1/D}}$.

به عنوانِ یه مثالِ دیگه از بازیِ جدیدمون میخوایم از یه روشِ بامزه عددِ π رو تعیین کنیم. اینطور به قضییه نگاه میکنیم که یک دایره میگیریم که در یک مربع محاط شده. حالا میگیم اگر تعدادی نقطه به طورِ تصادفی و با وزنِ یکسان داخلِ این مربع انتخاب بشن، کسری از اونها که داخلِ دایره میافته به کلشون متناسبه نسبتِ مساحتِ دایره به مساحتِ مربعه، یعنی:

$$\frac{N_{\text{in}}}{N_{\text{tot.}}} = \frac{\pi\, a^2}{(2\, a)^2} = \frac{\pi}{4} \tag{6}$$

حالا اگر به تعدادِ زیاد و بطورِ تصادفی این نقاط انتخاب و شمرده بشن میشه نسبتِ بالا رو حساب کرد، و عددِ π رو به دست آورد. در واقع باز داریم یه انتگرال رو حساب میکنیم، که هست انتگرالِ عددِ 1 رویِ داخلِ دایره:

$$\iint\limits_{x^2+y^2\leq a^2} 1 \cdot \mathrm{d}x\, \mathrm{d}y = \pi\, a^2 \tag{7}$$

بدست آوردنِ π به این روش رو میشه با چند تا بچه‌مدرسه‌ای به عنوانِ یه بازی انجام داد. مثلاً رویِ زمین یه مربع با یه دایره توش رسم کرد و مقداری مثلاً عدس رو بطورِ یه‌نواخت رویِ سطحِ مربع ریخت؛ البته یه‌نواختی مهمه. بعد تعدادِ کلِ عدسهایِ داخلِ مربع و دایره رو شمرد و به هم تقسیم کرد. اگه چند بار انجام بشه و میانگین بگیرین که چه بهتر.

به عنوانِ یه تمرینِ کوچک محاسبه‌ی بالا در متمتیکا رو به عهده‌ی خواننده میذارم ولی یه نمونه از اجراش رو برایِ انتخابِ 3000 نقطه در اینجا میارم (شکل ۱). عددِ اول تخمینِ π و دومی اختلافشه:

```
3.12933
-0.0122593
```

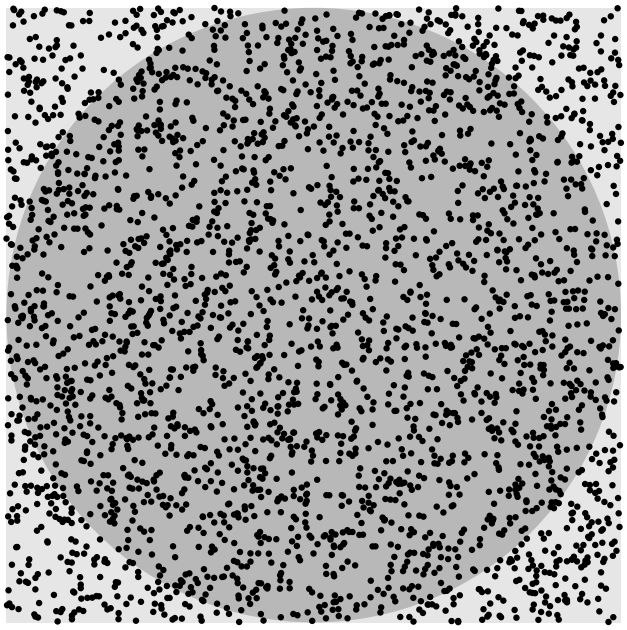
ادامه‌ی کارمون بهینه‌سازیِ همین بازیه برایِ مواقعی که توابعی که باهاشون سروکار داریم رفتارهای عجیب دارن. اسم بازیمون میشه «روش مونت‌کارلو» و تعریفش میشه: محاسبه‌ی جمع یا انتگرال با کمکِ اعداد تصادفی. در حالتِ بهینه‌شده غالباً این اعدادِ تصادفی خودشون از یک فرآیند تصادفی که مناسبِ مسئله‌ی مورد نظر ماس میان. به طورِ ساده‌ای که تا الان گفتیم، که ممکنه مونت‌کارلویِ خام [۱] نامیده بشه، انتگرال رویِ ناحیه‌ی R به بُعدِ n تبدیل به جمع رویِ اعدادِ تصادفی با وزنِ یکسان میشه:

$$\int_R f(x)\, \mathrm{d}^n x \;\rightarrow\; \frac{\mathrm{vol}(R)}{M} \sum_{n=1}^{M} f(x_n) \tag{8}$$

که در اون $\mathrm{vol}(R)$ حجمِ ناحیه‌ی R است.

---

[۱] Crude Monte Carlo



شکل ۱: نمونه‌ی اجراشده برایِ محاسبه‌ی عددِ π به روش دایره‌ی محیط در مربع.

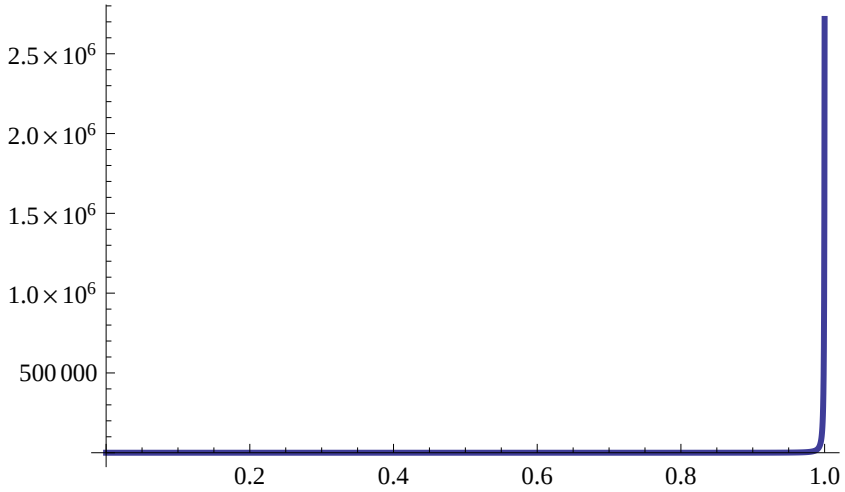
2 بهینه‌سازی و تولیدِ اعدادِ تصادفی

همونطور که گفته‌شد برایِ مسائل خاص روشمون باید بهینه بشه. عمده‌ی این بهینه‌سازی به این مربوطه که اعدادِ تصادفی با وزنِ یکسان در جمع نیان و طوری انتخاب بشن که به مسئله بخورن. اینطوری جوابهایِ بهتر در تعداد مراحل کمتر و در نتیجه زمانِ کمتر به دست میاد. برایِ مثال فرض کنید که میخوایم انتگرالِ زیر رو حساب کنیم:

$$\int_0^1 \frac{\exp(x^2)}{(1.001-x)^2}\,\mathrm{d}x \tag{9}$$

جوابِ این انتگرال به همراهِ رسمِ انتگرالده‌ش (شکل ۲) در متمتیکا این میشه:

```
g1[x_]:=Exp[x^2]/(1.001 - x)^2;
Plot[g1[x], {x, 0, 1}, PlotRange → All]
exact1 = NIntegrate[g1[x], {x, 0, 1}]
2688.74
```



شکل ۲: رسمِ انتگرالدهِ (9).

همونطور که می‌بینید این تابع در فاصله‌ی 0 تا 1 تغییراتِ شدیدی داره؛ از حدودِ 1 تا حدودِ 2,500,000. ببینیم کاربرد همینطوری روشمون چی میده. با ده دفعه 100 نقطه‌ی انتخابی با وزنِ یکسان اینطوری میشه:

```
xr:=RandomReal[{0,1}];
M = 100;
Table[MCInt1 = Mean[Table[g1[xr], {m,1,M}]]; MCInt1, {10}]
Mean[%]
Variance[%%]^0.5
{3920.65, 7245.54, 365.33, 1748.46, 583.776, 563.923, 358.704, 178.019, 4205.49, 2249.13}
2141.9
2327.29
```

اعدادِ فهرست‌شده نتیجه‌ی ده بار اجراس که میبینین چقدر متفاوتن. دو خطِ آخر میانگینِ ده دفعه و انحراف‌معیارِ اوناس. این بار به خوبی مثالِ اولمون نشد. دلیلش هم اینه که این تابع تغییراتِ شدیدی داره و انتخاب با وزنِ مساویِ اعدادِ تصادفی، اگرچه با انتخابِ تعدادِ بیشترِ نقاط جوابِ دقیق میده، ولی بهینه نیس. حالا میخوایم طوری اعدادِ تصادفی رو انتخاب کنیم تا جواب بهتر بشه. ایده‌ی اصلی از اتحادِ ساده‌ی زیر میاد:

$$\int_a^b f(x)\,\mathrm{d}x = \int_a^b \frac{f(x)}{w(x)}\,w(x)\,\mathrm{d}x = \int_{y(a)}^{y(b)} \frac{f\big(x(y)\big)}{w\big(x(y)\big)}\,\mathrm{d}y \tag{10}$$

که در اون $\mathrm{d}y = w(x)\,\mathrm{d}x$. در اینجا سعی بر اینه که تابعِ $w(x)$ طوری انتخاب بشه تا تغییراتِ $\frac{f(x)}{w(x)}$ به شدتِ

خودِ $f(x)$ نباشه. در عوض به جایِ جمع رویِ x ها با وزنِ یکسان، رویِ y ها با وزنِ یکسان جمع بزنیم، یعنی:

$$\int_a^b f(x)\,\mathrm{d}x = \int_{y(a)}^{y(b)} \frac{f\big(x(y)\big)}{w\big(x(y)\big)}\,\mathrm{d}y \;\to\; \frac{y(b)-y(a)}{M}\;\sum_{n=1}^{M}\frac{f\big(x(y_n)\big)}{w\big(x(y_n)\big)} \tag{11}$$

توجه کنید که y طوری انتخاب شده که انتخابش با وزنِ یکسان xهائی با توزیعِ $w(x)$ تولید میکنه. اسم عمومیِ این نسخه از بهینه‌سازی میشه «نمونه‌سازیِ ترجیحی» [۲]. همونطور که واضحه در نمونه‌سازیِ ترجیحی باید بتونیم اعدادِ تصادفی که از توزیعِ خاصی پیروی میکنن، و نه وزنِ یکسان دارند، تولید کنیم. در اینجا سه روشِ تولید این نوع اعدادِ تصادفی رو معرفی میکنیم.

2.1 روشِ تبدیلِ وارون [۳]

در همین مثالِ آخری فرض کنید بشه اعدادِ تصادفی با توزیعِ

$$w(x) = \frac{A}{(1.001-x)^2} \tag{12}$$

درست کرد. در بالا A چنان انتخاب میشه که $\int_0^1 w(x)\,\mathrm{d}x = 1$. اون وقت در جمع به جای استفاده از اعدادِ تصادفی با وزنِ مساوی از اونا استفاده میکنیم ولی فقط جمع رو رویِ مضربِ ثابتی از $\exp(x^2)$ میزنیم که تغییراتش خیلی کمتر از انتگرالدهِ اولیه‌ای هست که داشتیم. مطابقِ اونچه که قبلاً گفتیم باید اعدادِ y با شرطِ

$$\mathrm{d}y = w(x)\,\mathrm{d}x \tag{13}$$

داشته باشیم. در روشِ تبدیلِ وارون فرض بر اینه که رابطه‌ی بالا را میشه حل کرد و با گرفتنِ تابعِ وارون، x را بر حسبِ y به دست آورد: $x = \big(x(y)\big)$. در عمل تابعِ $w(x)$ طوری انتخاب میشه که تغییراتِ $\frac{f(x)}{w(x)}$ کم باشه. به علاوه باید طوری باشه که بشه x را برحسبِ y بدست آورد. اونوقت با انتخاب تصادفیِ y در فاصله‌ی $y(a)$ تا $y(b)$ و با وزنِ مساوی، به خاطرِ رابطه‌ی (13) در واقع x رو به طورِ تصادفی ولی با توزیعِ $w(x)$ انتخاب کردیم. در مثالی که داشتیم ثابتِ A و متغیرِ y به شکلِ زیر میشن:

```
A = 1/Integrate[1/(1.001 - x)^2, {x, 0, 1}]
Integrate[1/(1.001 - x)^2, x]
0.001001
1.
1.001-1.x
```

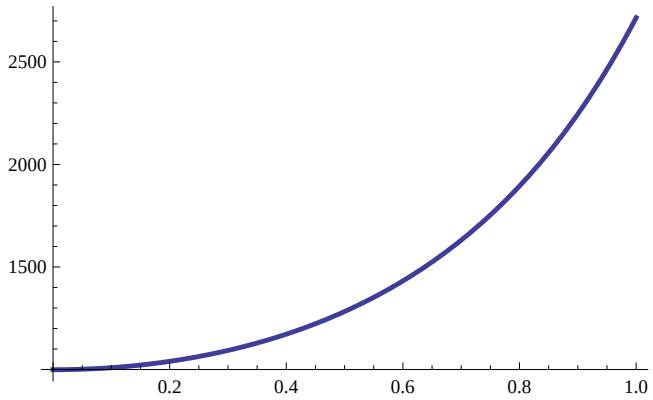
که آخری y رو برحسبِ x میده و میشه وارونش کرد. در این صورت رسمِ نسبتِ انتگرالدهِ اولمون بر $w(x)$ اینطوری میشه (شکل ۳):

```
w[x_]:=A/(1.001 - x)^2
Plot[g1[x]/w[x], {x, 0, 1}, PlotRange → All]
```

---

[۲] Importance Sampling

[۳] Inverse Transform Method



شکل ۳: رسمِ نسبتِ انتگرالده به تابعِ $w(x)$.

میبینید که تغییراتِ این یکی خیلی کمتر از اولیه‌س. حالا اگه روشمون رو به کار ببریم باید y رو با وزن مساوی بینِ $y(0)$ و $y(1)$ انتخاب کنیم، و از روش به کمکِ تابعِ وارون x رو پیدا کرد، ولی جمع رو رویِ $\frac{f(x)}{w(x)} = \frac{\exp(x^2)}{A}$ زد. نتیجه با ده دفعه‌ی 100 تائی اینطوری میشه:

```
yr:=RandomReal[{A/1.001, A/0.001}]
xr:=1.001 - A/yr
M = 100;
Table[Mean[Table[xr1 = xr; g1[xr1]/(A/(1.001 - xr1)^2), {m, 1, M}]], {10}]
Mean[%]
Variance[%%]^0.5
{2680.37, 2682.52, 2691.68, 2704.27, 2684.72, 2680.45, 2697.66, 2697.6, 2698.06, 2694.57}
2691.19
8.57928
```

که باز هم آخریها میانگین و انحراف‌معیارن. اگه با دفعه ی قبل که همین تعدادا بودن مقایسه کنید واقعاً بهتره.

از معایبِ این روش اینه که در بسیاری موارد $y(x)$ رو نمیشه تحلیلی به دست آورد؛ یعنی $w(x)$ ئی که مناسبه انتگرالش رو تحلیلی نداریم. دیگه این که ممکنه تابعِ وارون که باهاش $x(y)$ بدست میاد رو نداشته باشیم. مثلاً اگه اینطوری باشه:

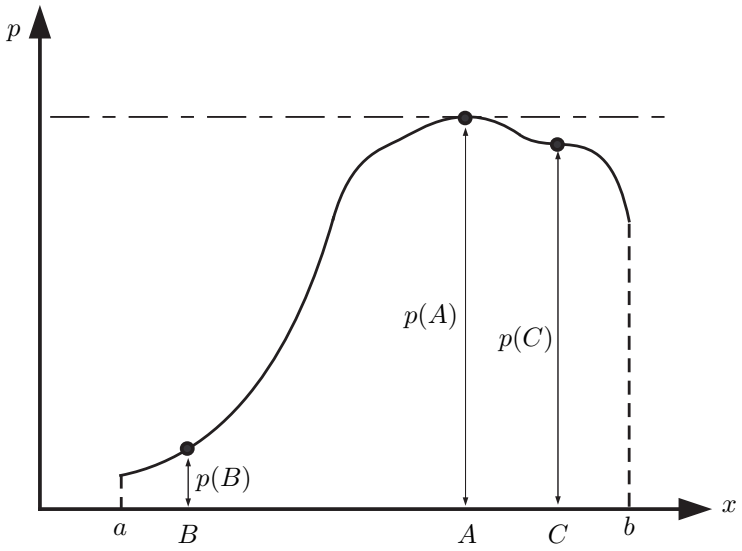
$$y(x) = \cosh^3 x + 1 + x^2$$

.

برایِ همین باید از روشهایِ دیگه‌ای که متکی به کارِ عددیَن استفاده کرد.

2.2 روشِ ردّی یا قبول-رد [۴]

فرض کنید به هر دلیلی میخواین اعدادِ تصادفی‌ئی با تابع‌توزیعِ $p(x)$ و در فاصله‌ی $[a,b]$ تولید کنید. یه مثالش در شکلِ ۴ اومده. روشِ ردّی میگه اول یه کرانِ بالا برای توزیع‌تون پیدا کنید؛ یعنی یه عددی که از همه‌ی مقادیرِ $p(x)$ در فاصله‌ی گفته‌شده بیشتر باشه. در موردِ مثالِ در شکل این نقطه رو میشه بیشینه‌ی تابع گرفت، یعنی $p(A)$. حالا بیاید یه عددِ تصادفی با وزنِ یکسان در فاصله‌ی $[a,b]$ انتخاب کنید، مثلِ x_0 و نسبتِ $p(x_0)/p(A)$ رو به دست بیارین. بَعد بیاید یه عددِ تصادفیِ دیگه با وزنِ یکسان ولی اینبار در فاصله‌ی $[0,1]$ بردارین، مثلاً x_r. حالا بیاید نسبتی که داشتید با x_r مقایسه کنید؛ اگر x_r کوچکتر بود x_0 رو بردارید و اگر نبود بندازینش دور و برید سراغِ انتخابِ یه x_0 دیگه و همین کارا رو تکرار کنین. فهمیدنِ این که چرا این نسخه اعدادِ با توزیعِ خواسته‌شده رو میده سخت نیست. فرض کنید x_0ِ انتخابی نقطه‌ی B توی شکل باشه که چون $p(B)$ کوچیکه انتظار داریم با شانسِ کمی قبول بشه. در واقع نسخه‌ی ما هم همین کار رو میکنه، چون احتمالِ اینکه x_r از $p(B)/p(A)$ کوچکتر باشه پائینه. در عوض اگه x_0 نقطه‌ی C باشه با شانس قابلِ توجهی x_r کمتر از $p(C)/p(A)$ هست و C انتخاب میشه. به همین ترتیب وقتی تعدادِ انتخابها زیاد بشه مجموعه‌ای از اعدادِ تصادفی داریم که توزیع‌شون مطابقِ شکله. البته همه‌ی این حرفا اثباتِ ریاضی داره.



شکل ۴: نمونه‌ای از نقشِ نقاط در روشِ قبول-رد.

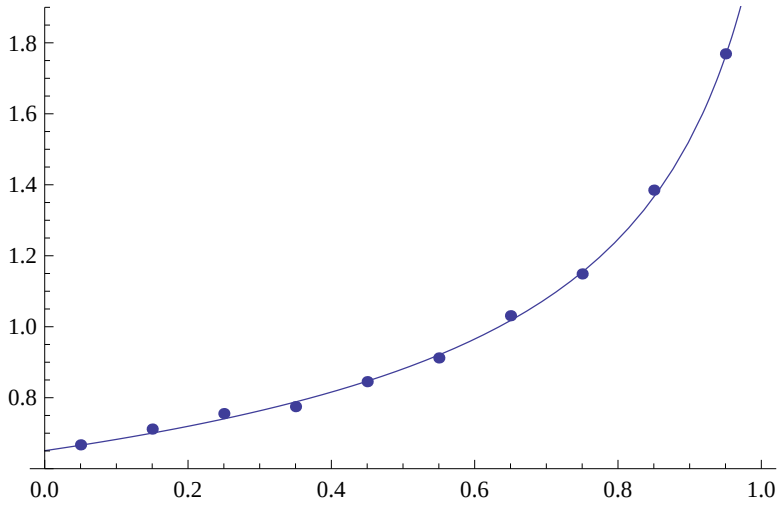
برایِ مثال تابع‌توزیعِ به شکلِ زیر رو انتخاب میکنیم:

$$p(x) \propto \frac{1}{\sqrt{1.1-x}}, \quad 0 \le x \le 1 \tag{14}$$

[۴] Rejection Method یا Accptance-Rejection Method

ضریبِ تناسب طوریه که $\int_0^1 p(x)\,\mathrm{d}x = 1$. در این مثال نقش نقطه‌ی A را $x = 1$ بازی میکنه. نسخه‌ای که گفتیم اجراش برای صدهزار انتخابِ اولیه اومده. در اجرایِ زیر از این صدهزارتا نهایتاً 46343 تاشون قبول شدن. در ادامه‌ی اجرا این اعداد رو در فاصله‌های 0.1 از 0 تا 1 شمردیم و به همراهِ خودِ توزیع در شکل ۵ رسم کردیم که کاملاً همخونی داره.

```
renp = NIntegrate[1/Sqrt[(1.1 - x)], {x, 0, 1}];
p[x_]:=(1/renp)1/Sqrt[(1.1 - x)];
xr:=RandomReal[{0, 1}];
M = 10^5;
sample = {}; Do[{xo = xr; If[p[xo]/p[1] > xr, sample = Append[sample, xo]]}, {M}];
dis = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; Do[dis[[Floor[sample[[i]] * 10] + 1]]++, {i, Length[sample]}];
Total[dis]
Length[sample] - Total[dis]
dis/Total[dis]/0.1
Show[
ListPlot[Partition[Riffle[{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95},
dis/Total[dis]/0.1], 2], PlotRange → {0.6, 1.9}, PlotStyle → PointSize[0.015]],
Plot[p[x], {x, 0, 1}, PlotRange → All]]
46343
0
{0.667199, 0.71165, 0.755238, 0.774874, 0.845219, 0.912112, 1.03101, 1.14883, 1.38489, 1.76898}
```



شکل ۵: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ (14) به روشِ قبول-رد به همراهِ خودِ تابع.

حالا فرض کنید به وسیله‌ی این اعدادِ تصادفی بخوایم انتگرالِ زیر رو حساب کنیم:

$$\int_0^1 \frac{\exp(x^2)}{\sqrt{1.1-x}}\,\mathrm{d}x \tag{15}$$

اگر اعدادِ تصادفی رو با وزنِ یکسان برداریم باید همه‌ی انتگرالده در بالا را برداریم، و البته تعدادِ اعدادِ تصادفیِ زیادی رو انتخاب کنیم تا نتیجه خوب بشه. اما حالا که اعداد با توزیعِ مناسب رو داریم می‌تونیم فقط $\exp(x^2)$ رو روش جمع بزنیم و با تعدادِ انتخابِ کمتر. نتیجه در پائین برایِ پنج دفعه انتخابِ فقط 10 تا عدد اومده:

```
renp = NIntegrate[1/Sqrt[(1.1 - x)], {x, 0, 1}];
p[x_]:=(1/renp)1/Sqrt[(1.1 - x)];
f1[x_]:=Exp[x^2];
NIntegrate[f1[x]p[x], {x, 0, 1}]
1.62352
xr:=RandomReal[{0, 1}];
M = 10;
Table[sample = {}; Do[{xo = xr; If[p[xo]/p[1] > xr, sample = Append[sample, xo]]}, {M}];
Mean[Table[f1[x], {x, sample}]], {5}]
Mean[%]
Variance[%%]^0.5
{1.77585, 1.34204, 1.54029, 1.91586, 1.64282}
1.64337
0.219789
```

دو خطِ آخر میانگین و انحراف‌معیارِ جوابن که با جوابِ 1.62352 نزدیکه اونهم فقط برایِ 50 تا عدد.

به عنوانِ یک مثالِ دیگه تابع‌توزیعِ نمائی $\exp(-x)$ رو میگیریم. برایِ صدهزار انتخاب، که فقط 63045 تاشون میمونن اینطوری میشه (شکل ۶):

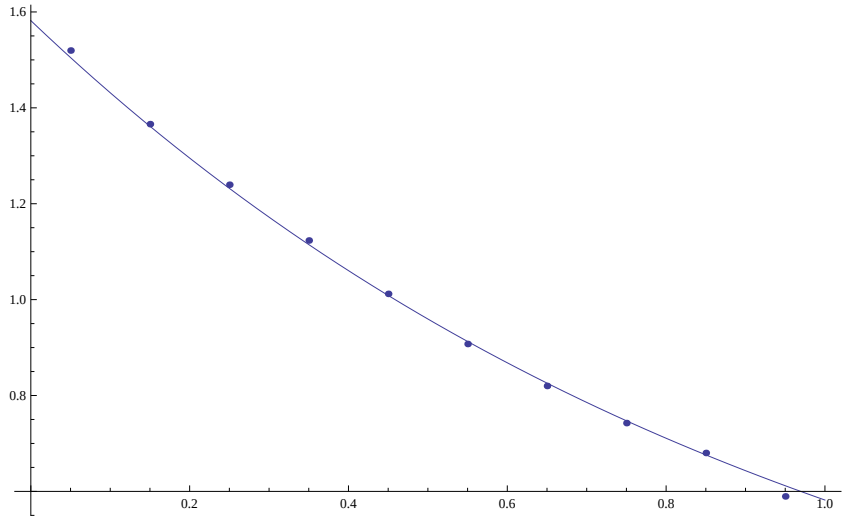
```
p[x_]:=E/(E - 1)Exp[-x];
xr:=RandomReal[{0, 1}];
M = 10^5;
sample = {}; Do[{xo = xr; If[p[xo]/p[0] > xr, sample = Append[sample, xo]]}, {M}];
dis = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; Do[dis[[Floor[sample[[i]] * 10] + 1]]++, {i, Length[sample]}];
Total[dis]
Length[sample] - Total[dis]
dis/Total[dis]/0.1
Show[
ListPlot[Partition[Riffle[{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95},
dis/Total[dis]/0.1], 2], PlotRange → All, PlotStyle → PointSize[0.008]],
```

Plot[$p[x]$, $\{x, 0, 1\}$, PlotRange $\to$ All]]

63045

0

$\{1.51955, 1.36601, 1.23959, 1.12332, 1.01198, 0.907447, 0.819891, 0.742486, 0.680149, 0.589579\}$



شکل ۶: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ $\exp(-x)$ به روشِ قبول-رد به همراهِ خودِ تابع.

برایِ تابعِ $\exp(-x/0.2)$ با 20000 انتخاب میشه (شکل ۷):

$t = 0.2$;

rc $= (1/t)/(1 - 1/E\text{^}(1/t))$;

p[x_]:=rcExp$[-x/t]$;

xr:=RandomReal$[\{0, 1\}]$;

$M = 2 \times 10\text{^}4$;

sample $= \{\}$; Do$[\{$xo $=$ xr; If$[p[$xo$]/p[0] >$ xr, sample $=$ Append[sample, xo]$]\}, \{M\}]$;

dis $= \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$; Do[dis[[Floor[sample$[[i]] * 10] + 1]]$++, $\{i,$ Length[sample]$\}]$;

Length[sample] $-$ Total[dis]

dis/Total[dis]$/0.1$

Show[

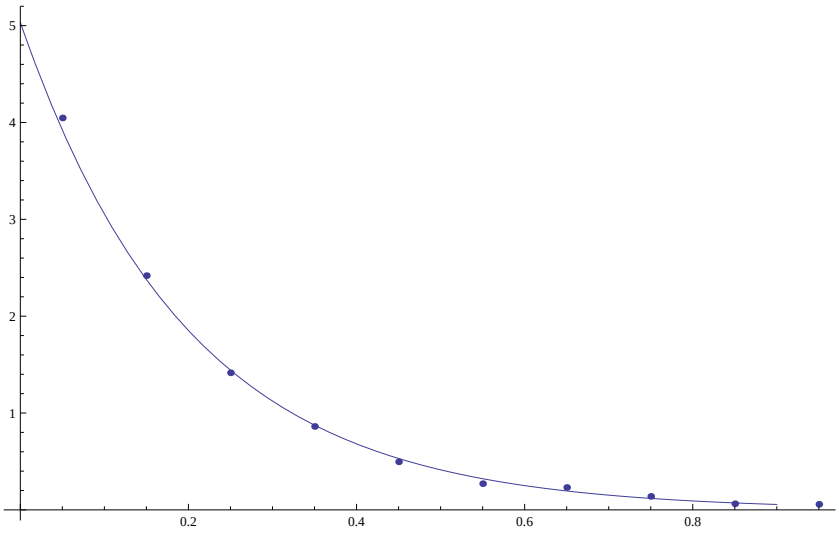
ListPlot[Partition[Riffle[$\{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95\}$,

dis/Total[dis]$/0.1], 2]$, PlotRange $\to$ All, PlotStyle $\to$ PointSize$[0.008]]$,

Plot$[p[x], \{x, 0, 0.9\}$, PlotRange $\to$ All$]]$

0

$\{4.04674, 2.41859, 1.41544, 0.861345, 0.496324, 0.270483, 0.231092, 0.139181, 0.0630252, 0.0577731\}$



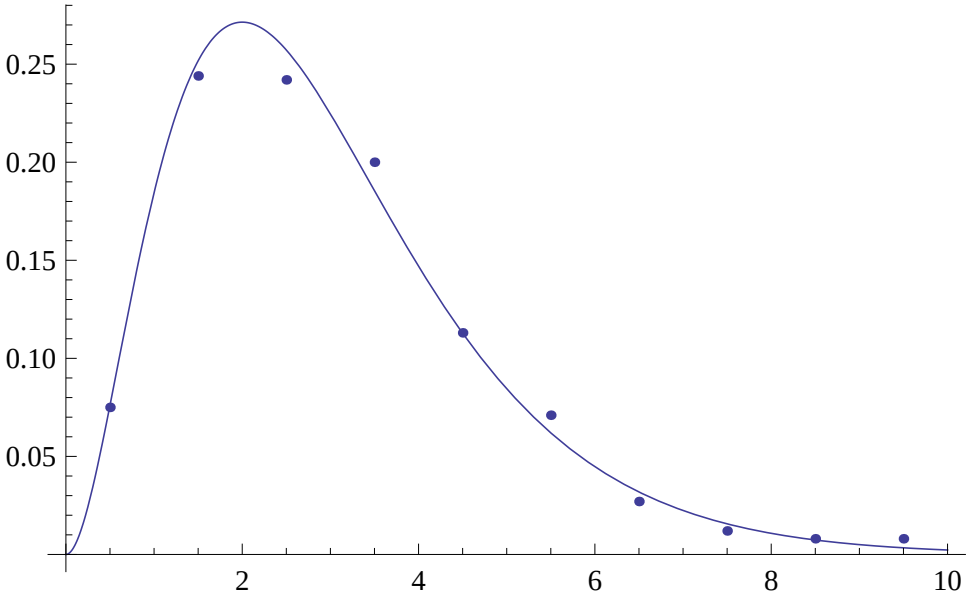
شکل ۷: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ $\exp(-x/0.2)$ به روشِ قبول-رد به همراهِ خودِ تابع.

همونطور که دیده شد در این روش تضمینی برایِ این که چند عدد به دست میاد وجود نداره. البته با کمی تغییر میشه این عیب رو برطرف کرد. در مثالِ آخر برای تابع‌توزیعِ $x^2\ \exp(-x)$ با استفاده از دستورِ `While` از برنامه میخوایم به تولیدِ عدد اونقدر ادامه بده تا به تعدادِ دلخواهِ هزارتا برسه (شکل ۸):

```
p[x_]:=x^2Exp[-x];
max = FindRoot[p'[u] == 0, {u, 2}][[1]];
xmax = u/. max;
xr:=RandomReal[{0, 10}];
xt:=RandomReal[];
M = 10^3;
sample = {};
While[Length[sample] < M, {xo = xr; If[p[xo]/p[xmax] > xt, sample = Append[sample, xo]]}];
dis = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; Do[dis[[Ceiling[sample[[i]]]]]++, {i, Length[sample]}];
Total[dis]
dis/Length[sample]/1//N
Show[ListPlot[Partition[Riffle[{0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5}, dis/Length[sample]/1], 2]],
Plot[p[x]/NIntegrate[p[t], {t, 0, 10}], {x, 0, 10}], PlotRange → All]
```

1000

$\{0.075, 0.244, 0.242, 0.2, 0.113, 0.071, 0.027, 0.012, 0.008, 0.008\}$

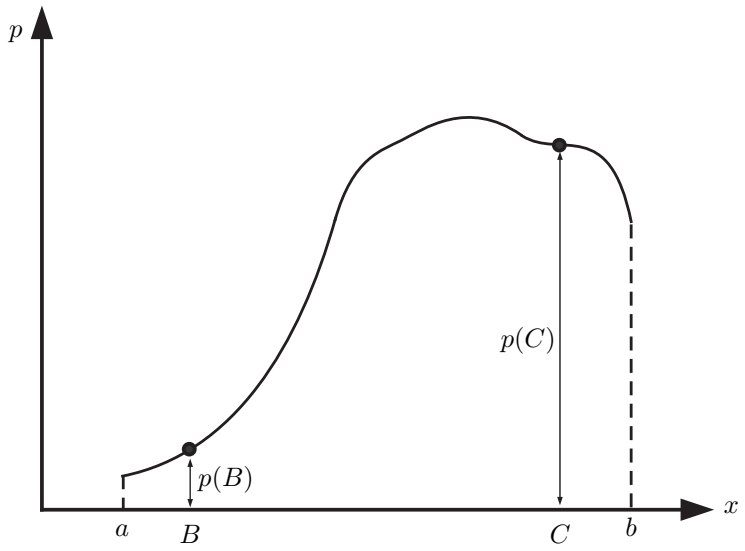


شکل ۸: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ $x^2 \exp(-x)$ به روشِ قبول-رد به همراهِ خودِ تابع.

2.3 روشِ متروپولیس [۵]

در روشِ ردّی مهم اینه که یک حدِ بالایِ مناسب برای تابع‌توزیع داشته باشیم تا به وسیله‌ی اون نسخه رو اجرا کنیم. به علاوه کنترل کمتری رویِ تعدادِ اعدادِ به دست اومده یا تعداد دفعاتی که باید برنامه اجرا بشه داریم. در روشی که اینجا معرفی میشه این عوارض نیست و به علاوه همچنان ساده‌س. در این روش ابتدا یک عددِ تصادفیِ با وزن یکسان در فاصله‌ی مورد نظر انتخاب مبشه، مثلاً x_1. بعد یه عدد دیگه انتخاب میشه مثلاً x_2. مطابقِ نسخه اگر $p(x_2) \geq p(x_1)$ بود، x_2 انتخاب میشه. در غیرِ اینصورت نسبتِ $\frac{p(x_2)}{p(x_1)}$، که حالا حتماً بینِ صفر و یکه، با یه عددِ تصادفی تو فاصله‌ی صفرویک مقایسه میشه؛ اگر بزرگتر بود باز هم x_2 بیرون میاد وگرنه x_1. فهمیدن این که چرا این نسخه اعدادِ مطابقِ توزیعِ $p(x)$ باز هم سخت نیست. مثلاً در شکل ۹ فرض کنید اگر $x_1 = B$ و $x_2 = C$ که C میاد بیرون. ولی اگر $x_1 = C$ و $x_2 = B$ نسبتِ $\frac{p(B)}{p(C)}$، با یه عدد از صفر تا یک مقایسه میشه. مثلاً در موردی که در شکل اومده چون با توجه به اندازهها این نسبت خیلی کوچکه شانسِ کمی برای بیرون اومدنِ B هست اگرچه این شانس صفر نیست. به همین دلیل اعدادی که بیرون میان در نهایت با توزیع میخونن. در عمل بهتره که با یک x_1 ورودی، در تعداد مراحلِ بیشتر از یک بار x_2 انتخاب و در معرضِ مقایسه با x_1 قرارش بدیم. اینطوری حتا برایِ تعداد کمی اعدادِ خروجی، چون رویِ هر کدوم «محک» چندین بار اعمال شده، همخونی با توزیع خوب میشه.

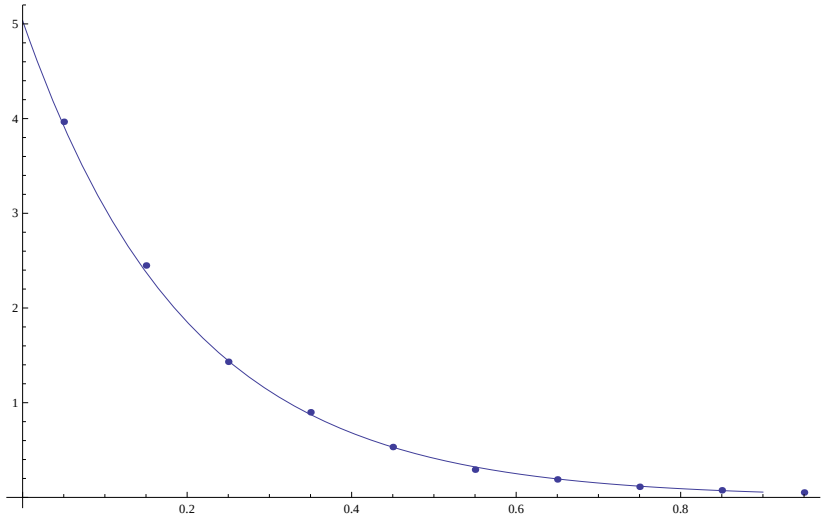
[۵] Metropolis Method



شکل ۹: نمونه‌ای از نقشِ نقاط در روشِ متروپولیس.

به عنوانِ مثال باز هم تابعِ نمائیِ $\exp(-x/0.2)$ رو داریم که برایِ ده‌هزارتا عددِ خروجی، که هر کدوم با کمکِ دستورِ Do از یک فرآیند با بیست بار مقایسه بیرون اومدن، میده (شکل ۱۰):

```
t = 0.2;
rc = (1/t)/(1 - 1/E^(1/t));
p[x_] := rcExp[-x/t];
xr := RandomReal[{0, 1}];
M = 10^4;
sample = Table[xi = xr; Do[xj = xr; If[xj < xi, xi = xj, If[p[xj]/p[xi] > xr, xi = xj, xi]], {20}];
xi, {M}];
dis = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; Do[dis[[Ceiling[sample[[i]] * 10]]]++, {i, M}];
Total[dis] - M
dis/M/0.1
Show[
ListPlot[Partition[Riffle[{0.05, 0.15, 0.25, 0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95}, dis/M/0.1],
2], PlotStyle → PointSize[0.008]],
Plot[p[x], {x, 0, 0.9}, PlotRange → {0, p[0]}], PlotRange → All]
0
{3.966, 2.449, 1.432, 0.899, 0.532, 0.293, 0.19, 0.112, 0.075, 0.052}
```



شکل ۱۰: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ $\exp(-x/0.2)$ به روشِ قبول-رد به همراهِ خودِ تابع.

به عنوانِ مثالِ آخر برای $x^2\ \exp(-x)$، با دَه بار مقایسه برایِ هر خروجی، اعدادِ تصادفی تولید میکنیم (شکل ۱۱):

p[x_]:=x^2Exp[$-x$];

xr:=RandomReal[{0, 10}];

xt:=RandomReal[];

$M = 10$^3;

sample = Table[xi = xr; Do[xj = xr; If[p[xj] > p[xi], xi = xj, If[p[xj]/p[xi] > xt, xi = xj, xi]], {10}]; xi, {M}];

dis = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; Do[dis[[Ceiling[sample[[i]]]]]++, {i, M}];

Total[dis] − M

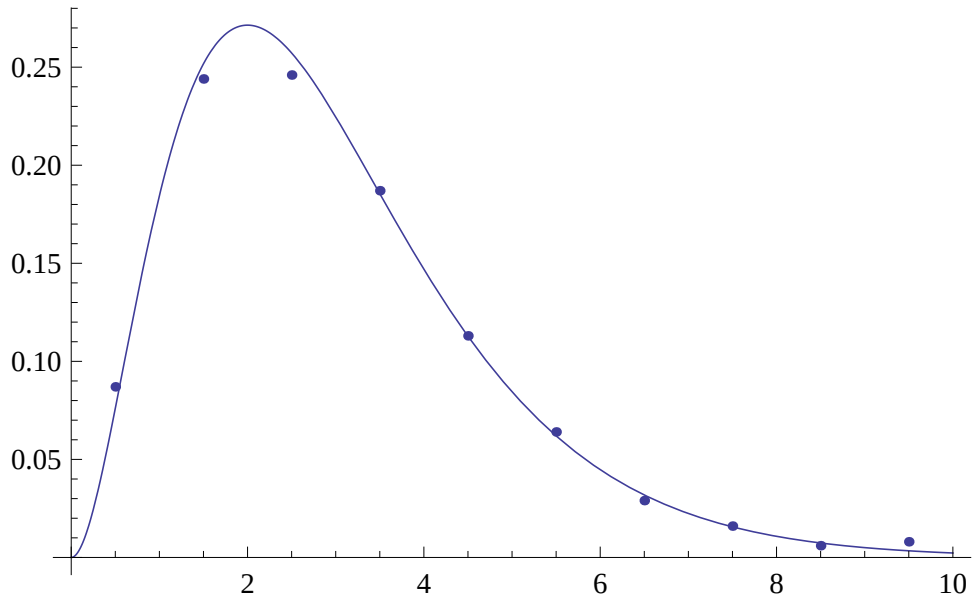
dis/M/1//N

Show[ListPlot[Partition[Riffle[{0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5}, dis/M/1], 2]],

Plot[$p[x]$/NIntegrate[$p[t]$, {t, 0, 10}], {x, 0, 10}, PlotRange → {0, p[0]}], PlotRange → All]

0

{0.087, 0.244, 0.246, 0.187, 0.113, 0.064, 0.029, 0.016, 0.006, 0.008}



شکل ۱۱: توزیعِ اعدادِ تصادفیِ تولیدشده برایِ تابعِ $x^2 \exp(-x)$ به روشِ قبول-رد به همراهِ خودِ تابع.

روشِ متروپولیس در مکانیکِ آماری کاربردِ زیادی داره. چیزی که اینجا ما در دو پله‌ی انتخابِ x_1 یا x_2 گفتیم، در عمل برای به تعادلِ ترمودینامیک رسوندنِ سیستمها در تعدادِ پله‌های زیاد انجام میشه. در اونجا مبنای مقایسه انرژیِ حالتهای مختلفِ سیستم در دمایِ T ئه، جوری که با تابع‌توزیعِ بولتزمان بخونه: $\exp(-E/T)$.

منابع

- Kristopher R. Beevers, *CSCI-6971 Lecture Notes: Monte Carlo integration*, 2006.
- Riku Linna, *LECTURE 9: Monte Carlo Methods I*, http://www.lce.hut.fi/teaching/S-114.1100/
- D.J.C. Mackay, *Introduction to Monte Carlo Methods*.
- Mats Wallin, *Monte Carlo simulation in statistical physics*, 2005.
- Jonathan Pengelly, *Monte Carlo Methods*, 2002.