شبیه‌سازی رایانه‌ای در فیزیک
تمرین ششم: شبکه‌های پیچیده و مونت کارلو

سینا معمر ۹۵۱۰۲۳۱۶

۱۹ شهریور ۱۴۰۰

۱ شبکه اردوش-رنی

کد این بخش از تمرین را در فایل q1.py می‌توان مشاهده نمود. در ابتدا باید تابع analyse_erdos_renyi را با تعداد راس‌ها و میانگین درجه رئوس دل‌خواه صدا می‌کنیم. روش کار این تابع به این صورت است که ابتدا تعداد یال‌های متناسب با این داده‌ها را پیدا می‌کند و یک گراف تصادفی با تعداد رئوس و یال‌های داده شده می‌سازد. سپس درجه رئوس و خوشگی هر راس را به دست آورده و نمودار هیستوگرام آن‌ها را رسم می‌کند. نتایج به دست آمده را می‌توان در شکل‌های ۱، ۲ و ۳ مشاهده نمود.
همان‌طور که دیده می‌شود، با افزایش $\langle k\rangle$ قله‌ی منحنی توزیع درجه رئوس نیز در نمودار به سمت راست حرکت می‌کند و افزایش می‌یابد و این مقدار تقریبا برابر با همان $\langle k\rangle$ داده شده است. علاوه بر آن به دلیل افزایش تعداد k های محتمل در شبکه، داده‌های بیش‌تری خواهیم داشت و نمودار حول $\langle k\rangle$ هموارتر می‌شود.
در مورد توزیع خوشگی نکته‌ی قابل توجه، تیز بودن منحنی به دست آمده است. به این صورت که حتی برای $\langle k\rangle$ های بزرگ نیز توزیع خوشگی پهنای بسیار کمی دارد. برای $\langle k\rangle = 0.8, 1$ این مقدار تقریبا همیشه صفر است ولی با افزایش میانگین درجه رئوس مشاهده می‌کنیم که قدری پهنای آن زیاد شده و هم‌چنین قله‌ی آن به سمت راست شروع به حرکت می‌کند.
اگر بخواهیم برای ذخیره‌سازی شبکه از ماتریس مجاورت استفاده کنیم، به یک ماتریس $N \times N$ نیاز خواهیم داشت. از آن جایی هم که گراف وزن‌دار نیست، هر درایه تنها مقادیر 1 و 0 را می‌تواند داشته باشد پس تنها یک بیت نیاز است. در نتیجه برای ذخیره‌سازی شبکه به صورت ماتریس مجاورت به

$$\text{memory} = N^2 \text{ bits} = \frac{N^2}{8} \text{ bytes} \tag{۱}$$

حافظه نیاز است.
اگر بخواهیم از لیست مجاورت استفاده کنیم، باید دو برابر تعداد یال‌ها (m) حافظه داشته باشیم. برای یک ماشین $32 - bit$ ای به

$$\begin{cases} \text{memory} = 2m \times 32 \text{ bits} = 8m \text{ bytes} \\ \langle k\rangle = \frac{2m}{N} \Rightarrow m = \frac{\langle k\rangle N}{2} \end{cases} \Rightarrow \text{memory} = 4N\langle k\rangle \text{ bytes} \tag{۲}$$

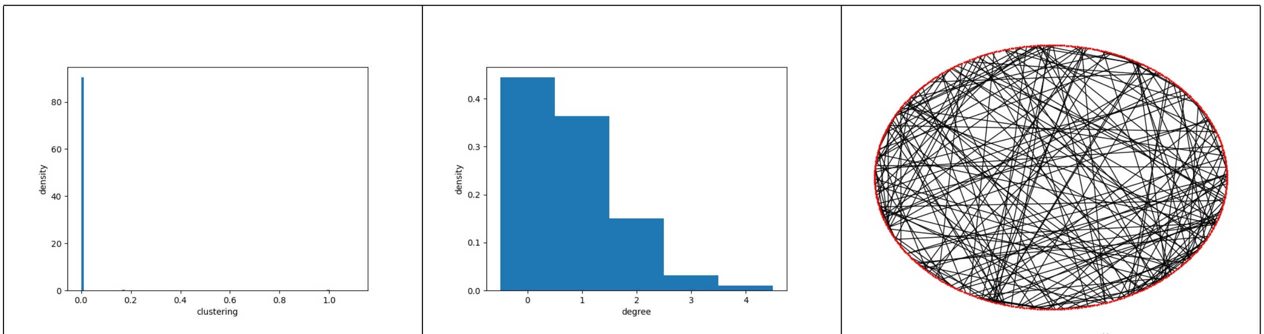
حافظه نیاز است.
برای ذخیره کردن با استفاده از لیست یال‌ها نیز به $2m$ خانه‌ی حافظه نیاز داریم. در نتیجه همانند لیست مجاورت برای یک ماشین $32 - bit$ ای به

$$\text{memory} = 4N\langle k\rangle \text{ bytes} \tag{۳}$$

حافظه نیاز است.
برای جمع‌بندی و برای این سه شبکه‌ی داده شده می توان به جدول ۱ مراجعه کرد.

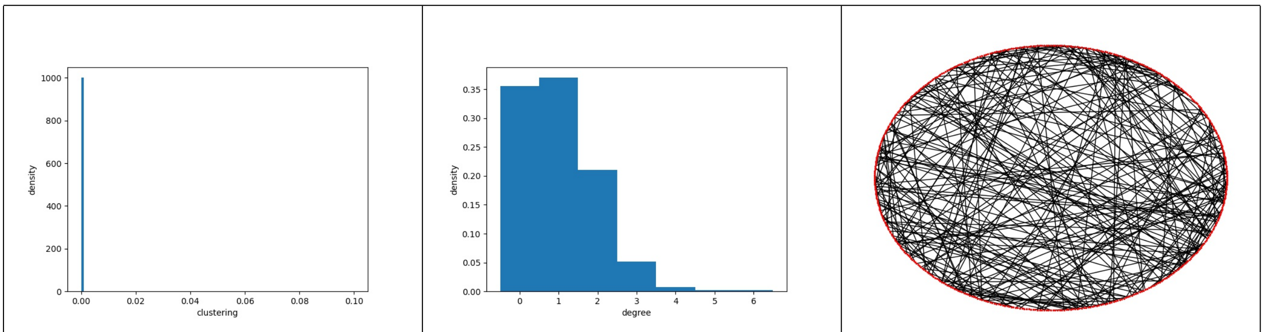
			adjacency matrix	adjacency list	edge list
N	$\langle k \rangle$	$m = \frac{N\langle k \rangle}{2}$	$\frac{N^2}{8}$ bytes	$4N\langle k \rangle$ bytes	$4N\langle k \rangle$ bytes
500	0.8	200	31.25 kilobytes	1.6 kilobytes	1.6 kilobytes
	1	250	31.25 kilobytes	2 kilobytes	2 kilobytes
	8	2000	31.25 kilobytes	16 kilobytes	16 kilobytes

جدول ۱: میزان حافظه‌ی مورد نیاز برای روش‌های ذخیره‌سازی متفاوت



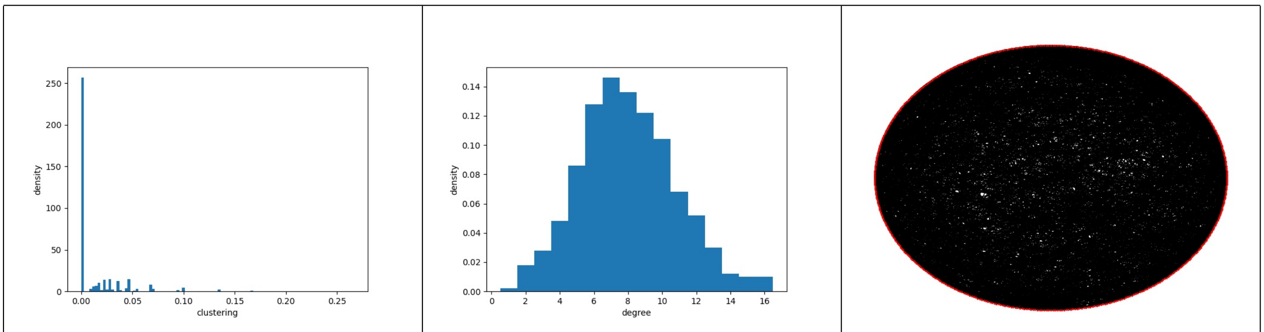
(آ) گراف شبکه (ب) منحنی توزیع درجه رئوس (ج) منحنی توزیع خوشگی رئوس

شکل ۱: شبکه‌ی اردوش-رنی با $N = 500$ و $\langle k \rangle = 0.8$



(آ) گراف شبکه (ب) منحنی توزیع درجه رئوس (ج) منحنی توزیع خوشگی رئوس

شکل ۲: شبکه‌ی اردوش-رنی با $N = 500$ و $\langle k \rangle = 1$



(آ) گراف شبکه (ب) منحنی توزیع درجه رئوس (ج) منحنی توزیع خوشگی رئوس

شکل ۳: شبکه‌ی اردوش-رنی با $N = 500$ و $\langle k \rangle = 8$

۲ انتگرال‌گیری مونت کارلو

کد این بخش از تمرین را در فایل q2.py می‌توان مشاهده نمود. در ابتدا باید تابع monte_carlo را فراخوانی کنیم. این تابع حد بالا و پایین انتگرال، تابع تولید اعداد تصادفی، تابع هوشمند و تعداد نمونه‌ها را به عنوان ورودی می‌گیرد. سپس تابع انتگرال را از تقسیم تابع داده شده به تابع هوشمند می‌سازد. در نهایت نیز به تعداد نمونه‌های داده شده، با استفاده از تابع تولید اعداد تصادفی داده شده، اعداد تصادفی می‌سازیم و مقدار تابع انتگرال را در آن نقاط حساب می‌کنیم. سپس میانگین آن‌ها را ضرب در ضریب اسکیل به عنوان مقدار انتگرال و انحراف نسبی آن را نیز به عنوان خطای آماری بر می‌گردانیم. این تابع را به صورت ساده و با دو تابع هوشمند متفاوت و با تعداد نمونه‌های مختلف برای انتگرال $\int_0^2 e^{-x^2}$ صدا می‌کنیم. نتایج به دست آمده را در جداول ۲، ۳ و ۴ می‌توان مشاهده نمود.
همان طور که دیده می‌شود، با افزایش تعداد نمونه‌ها، زمان اجرا و دقت مقدار به دست آمده هر دو افزایش می‌یابند که مطابق با انتظار ما است. علاوه بر آن مشاهده می‌شود که زمان اجرا به طور خطی افزایش پیدا می‌کند که با توجه به اینکه صرفا در تابع مورد نظر به تعداد نمونه‌ها داده شده باید مقدار تابع انتگرال را محاسبه کنیم، پس نتیجه‌ای منطقی است. همان طور که انتظار داشتیم، زمان اجرا در حالت نمونه‌برداری ساده، کم‌تر از نمونه‌برداری هوشمند است که علت آن وجود محاسبات اضافه‌تر برای به دست آوردن اعداد تصادفی با توزیع موردنظر و محاسبه‌ی مقدار تابع هوشمند است. ولی در ازای آن دقت بالاتری را در زمان کم‌تر به دست می‌آوریم. برای نمونه می‌توان به مقدار خطای واقعی 0.0006 نگاه کرد. برای نمونه‌برداری ساده، 2.0 ثانیه و برای نمونه‌برداری هوشمند، 0.41 ثانیه زمان برده است. در نتیجه اگر چه در نمونه برداری هوشمند در تعداد نمونه‌های ثابت به زمان بیش‌تری نیاز است ولی در تعداد نمونه‌های کم‌تری و در نتیجه در زمان کم‌تر می‌توان به دقت مطلوب رسید.

N	Simple Sampling				مقدار واقعی
	مقدار	خطای آماری	خطای واقعی	زمان اجرا	
10^3	0.92	0.02	-0.038	0.00015	0.882081
10^4	0.896	0.007	-0.014	0.00030	
10^5	0.884	0.002	-0.002	0.0030	
10^6	0.8809	0.0007	0.001	0.022	
10^7	0.8819	0.0002	0.0002	0.21	
10^8	0.88214	0.00007	0.00006	2.0	

جدول ۲: مقایسه مقدار انتگرال، خطای آماری، خطای واقعی و زمان اجرای مونت کارلو برای $\int_0^2 e^{-x^2}$ با نمونه‌برداری ساده

N	Important Sampling				مقدار واقعی
	$g(x) = e^{-x}$				
	مقدار	خطای آماری	خطای واقعی	زمان اجرا	
10^3	0.873	0.008	0.009	0.0001	0.882081
10^4	0.884	0.003	-0.002	0.00036	
10^5	0.8825	0.0008	-0.0004	0.0038	
10^6	0.8820	0.0003	0.00008	0.037	
10^7	0.88202	0.00008	0.00006	0.41	
10^8	0.88211	0.00003	-0.00003	3.7	

جدول ۳: مقایسه مقدار انتگرال، خطای آماری، خطای واقعی و زمان اجرای مونت کارلو برای $\int_0^2 e^{-x^2}$ و با تابع هوشمند $g(x) = e^{-x}$

N	Important Sampling				مقدار واقعی
	$g(x)=\frac{1}{1+x^2}$				
	مقدار	خطای آماری	خطای واقعی	زمان اجرا	
10^3	0.891	0.009	-0.009	0.0001	0.882081
10^4	0.886	0.003	-0.004	0.00047	
10^5	0.8821	0.0009	-0.00002	0.0051	
10^6	0.8818	0.0003	0.0003	0.050	
10^7	0.88206	0.00009	0.00002	0.55	
10^8	0.88209	0.00003	-0.00001	5.1	

جدول ۴: مقایسه مقدار انتگرال، خطای آماری، خطای واقعی و زمان اجرای مونت کارلو برای $\int_0^2 e^{-x^2}$ و با تابع هوشمند $g(x)=\frac{1}{1+x^2}$

۳ انتگرال چندگانه

کد این بخش از تمرین را در فایل q3.py می‌توان مشاهده نمود. در ابتدا باید تابع compute_com را با شعاع، تابع چگالی و تعداد نمونه‌های دل‌خواه صدا کنیم. روش کار این تابع به این صورت است که ابتدا به تعداد نمونه‌های داده شده، نقاط تصادفی در x، y و z انتخاب می‌کند. سپس اندیس آن‌هایی را که در داخل کره می‌افتند را به دست می‌آورد و آن‌ها را به عنوان نقاط تصادفی نهایی ذخیره می‌کند. در آخر نیز تابع چگالی را با این نقاط صدا کرده و چگالی‌های به دست آمده را در مختصات نقاط ضرب می‌کند. به عنوان خروجی نیز میانگین این مقادیر را به عنوان مختصات مرکز جرم و انحراف معیار نسبی آن‌ها را به عنوان خطای آماری این مقادیر بر می‌گرداند. نتایج به دست آمده برای شعاع 5 و تعداد 10^8 به صورت زیر است:

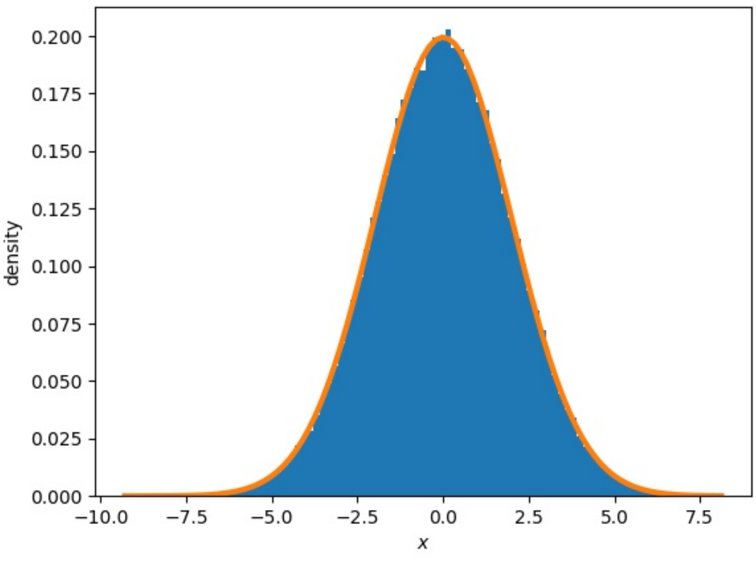
$$\begin{cases} X = 0.0000 \pm 0.0002 \\ Y = 0.0005 \pm 0.0002 \\ Z = 0.3328 \pm 0.0002 \end{cases} \tag{۴}$$

۴ متروپولیس

کد این بخش از تمرین را در فایل q4.py می‌توان مشاهده نمود. در ابتدا باید object از کلاس Metropolis با تابع توزیع دل‌خواه بسازیم. سپس تابع generate را روی آن با زمان، مکان اولیه و طول قدم دل‌خواه صدا می‌کنیم. روش کار این تابع به این صورت است که به اندازه‌ی زمان داده شده، عدد تصادفی دل‌خواه برای تعیین مکان نقطه‌ی بعدی و احتمال حرکت در آن زمان تولید می‌کنیم. سپس روی زمان داده شده پیمایش کرده و در مرحله با استفاده از انتخاب متروپولیس، می‌بینیم که آیا باید حرکت کنیم یا نه. به این ترتیب اعداد تصادفی جدید را به دست می‌آوریم. هم‌چنین در هر مرحله اگر جابجا بشویم، مقدار متغیر acceptance_count را یکی زیاد می‌کنیم. در نهایت نیز می‌توانیم نرخ قبولی را با صدا کردن تابع get_acceptance_rate به دست بیاوریم. برای به دست آوردن طول هم‌بستگی نیز روی j های مختلف پیمایش کرده و برای هر کدام مقدار هم‌بستگی را محاسبه می‌کنیم. در نهایت یک تابع نمایی را به داده‌های به دست آمده فیت می‌کنیم و از روی پارامترهای آن، طول هم‌بستگی را گزارش می‌کنیم. نتایج به دست آمده را در جدول ۵ می‌توان مشاهده نمود. همان طور که دیده می‌شود برای $0.4 < a_r < 0.5$ کم‌ترین طول هم‌بستگی را خواهیم داشت. پس برای اینکه در سریع‌ترین زمان به تابع توزیع دل‌خواه برسیم، باید a_r را در این بازه تنظیم بکنیم. در نتیجه برای نمونه می‌توانیم طول قدم را 7 در نظر بگیریم. در این صورت طول هم‌بستگی 1.74 و نرخ قبولی 0.438 می‌شود. پس اعداد تولید شده‌ای که با یکدیگر حداقل $2\xi = 3.48 \approx 3$ فاصله دارند را می‌توان بدون هم‌بستگی در نظر گرفت و به عنوان اعداد تصادفی گزارش نمود. نمودار توزیع این اعداد را در شکل ۴ می‌توان مشاهده نمود.

a_r	Δ	ξ
0.100	31.92	7.34
0.200	15.93	3.39
0.300	10.60	2.12
0.400	7.77	1.75
0.500	5.88	1.90
0.600	4.41	2.54
0.700	3.15	3.98
0.800	2.04	7.95
0.900	1.01	27.1

جدول ۵: نرخ قبولی‌های مختلف و طول قدم‌ها و طول هم‌بستگی‌های متناظر با آن‌ها برای توزیع گاوسی با $\sigma = 2$، $x_0 = 0$ و با زمان 10^6



شکل ۴: تابع توزیع اعداد تصادفی تولید شده با روش متروپولیس برای توزیع گاوسی با $\sigma = 2$، $\Delta = 7$ و با 10^6 نمونه