

گزارش کار تمرین شماره ۱

ارمیا اعتمادی بروجنی

مقدمه

در این سری تمرین، فرآکتال های متفاوتی را به کمک روش های تعیینی رسم می کنیم. کد های تمرین به زبان زیبا و کاربردی C++ نوشته شده اند. برای رسم از برنامه gnuplot استفاده شده که به کمک هدر gnuplot-iostream.h می توانیم از درون کد فرآمین رسم را به ارسال کنیم. در صورتی که قصد دارید کد را کامپایل کنید به عنوان نمونه از دستور زیر در gcc استفاده کنید:

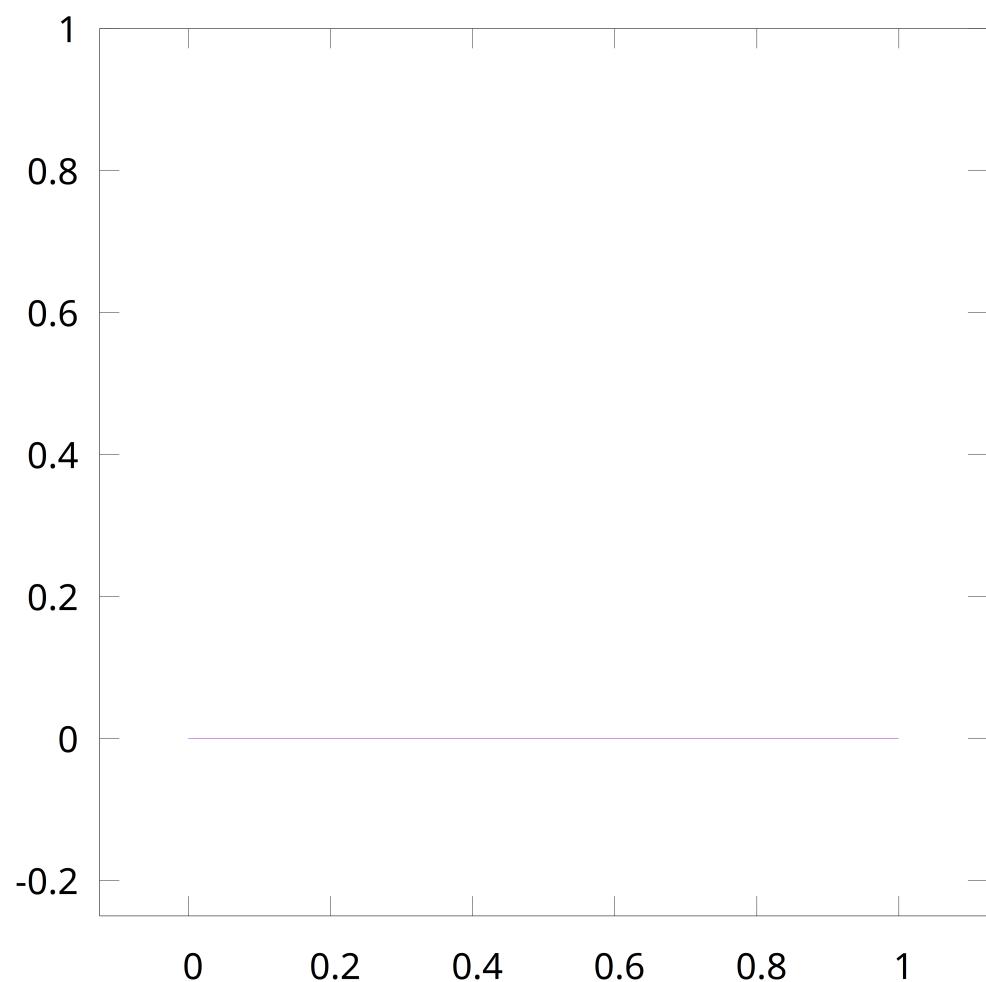
```
1 g++ HW1-98100594-P4.cpp -o hw1-P4-run.exe -lboost_iostreams -lboost_system -lboost_filesystem
```

مسئله اول: دانه برف کخ

فرم کلی برنامه ساده است و در مسائل بعدی نیز تکرار می شود. چهار تابع تبدیل کننده را تعریف کرده و روی نقطه اولیه اثر می دهیم. سپس نقاط به دست آمده را به با یک دیگر اجتماع می گیریم تا به نقاط نهایی در مرحله اول برسیم. با تکرار این کار می توانیم مجموعه های مرتبه بالاتر را بدست آوریم.

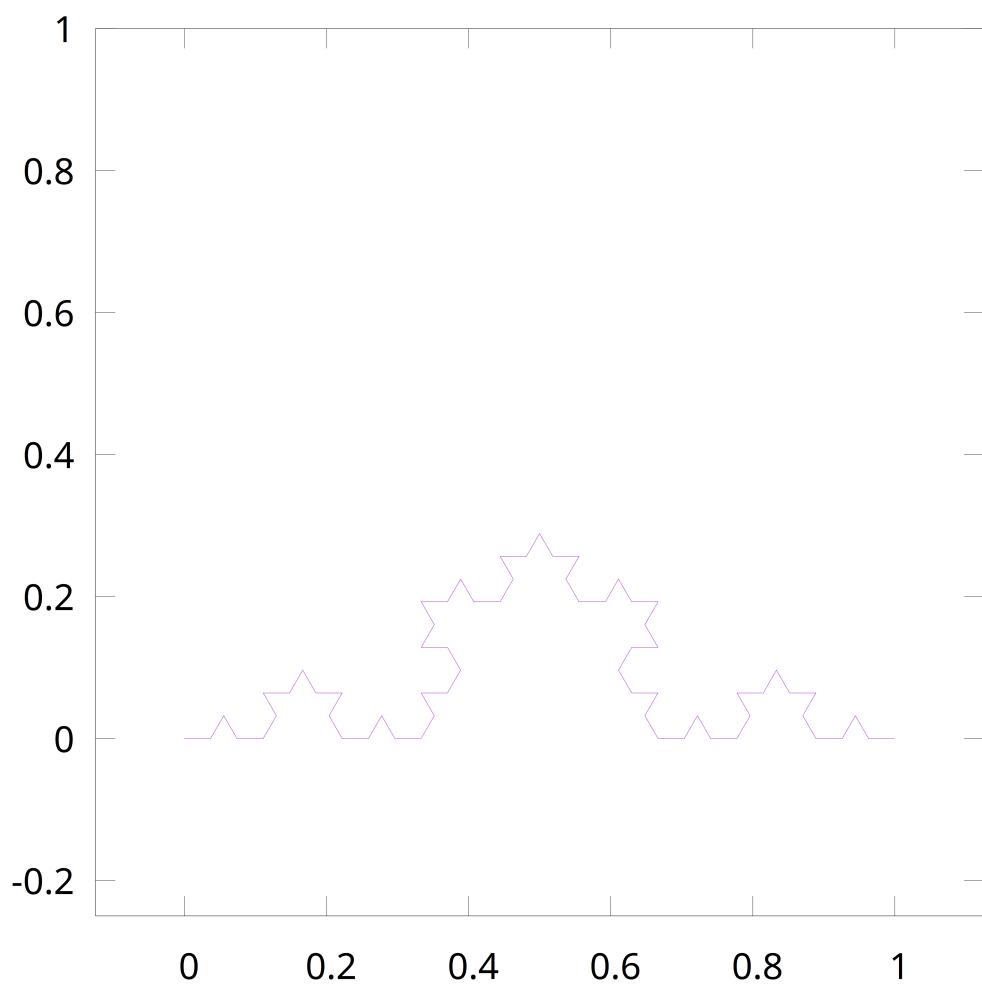
برای ذخیره سازی نقاط از آرایه های دینامیک vector در کتابخانه std استفاده شده که هر نقطه یک tuple از کتابخانه boost است. نقاط به کمک ۴ تابع func1 تا func4 تبدیل می شوند و سپس به کمک دستور insert چهار آرایه خروجی با یک دیگر اجتماع گرفته می شوند. نقاط سپس به gnuplot فرستاده شده و رسم می شوند. نتیجه برای چند مرتبه مختلف در ادامه آورده شده اند:

Koch Fractal N = 0



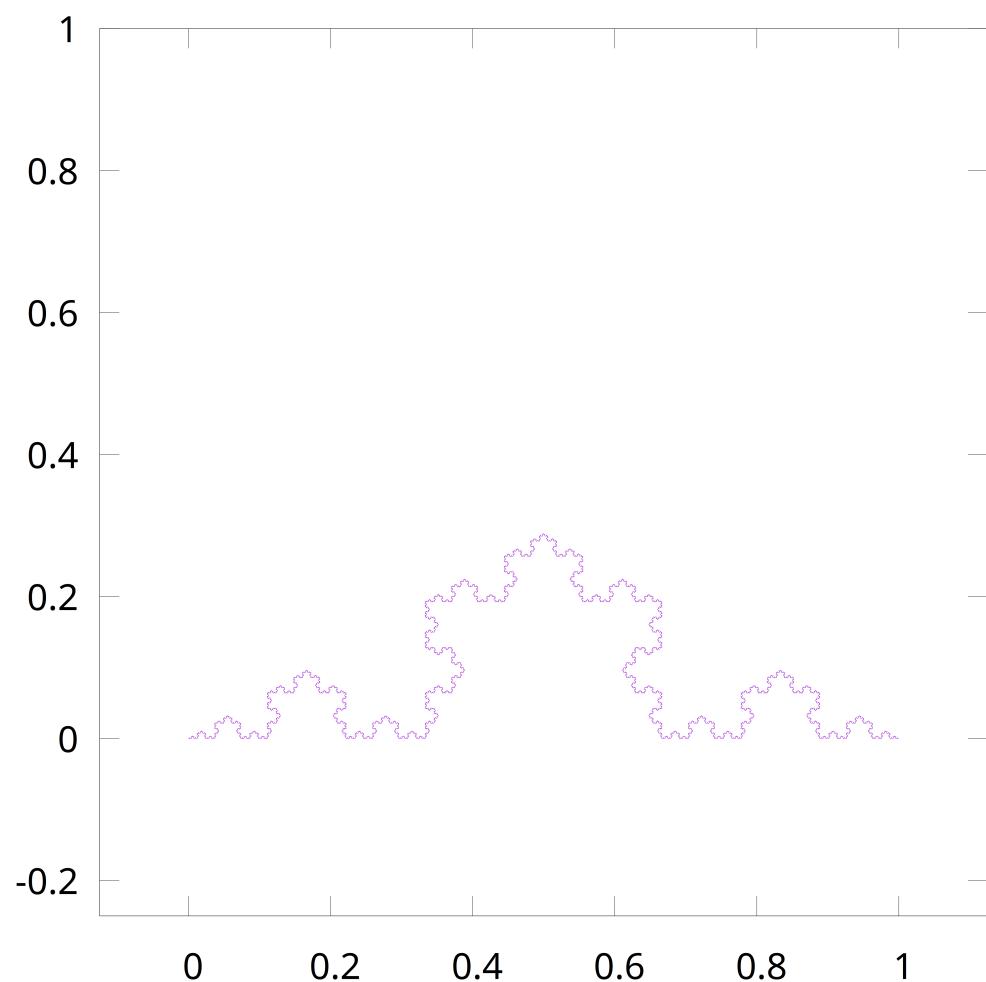
y

Koch Fractal N = 3



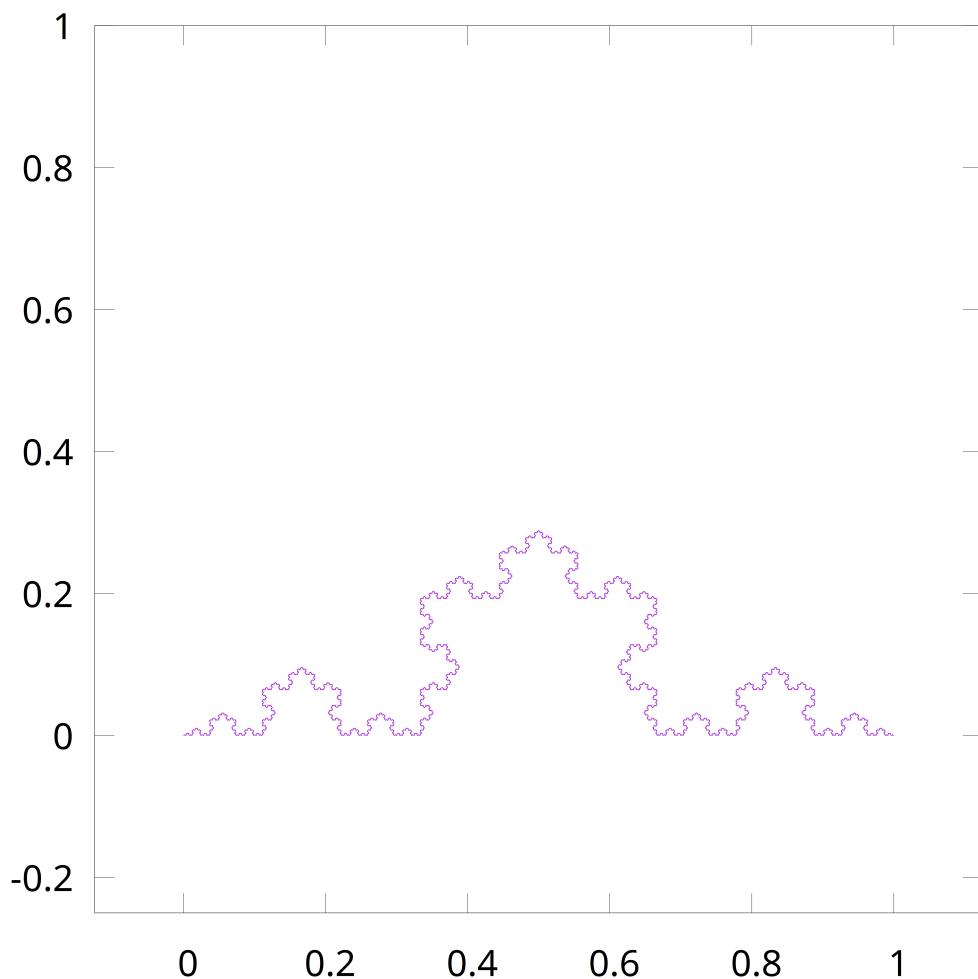
r

Koch Fractal N = 6



r

Koch Fractal N = 8

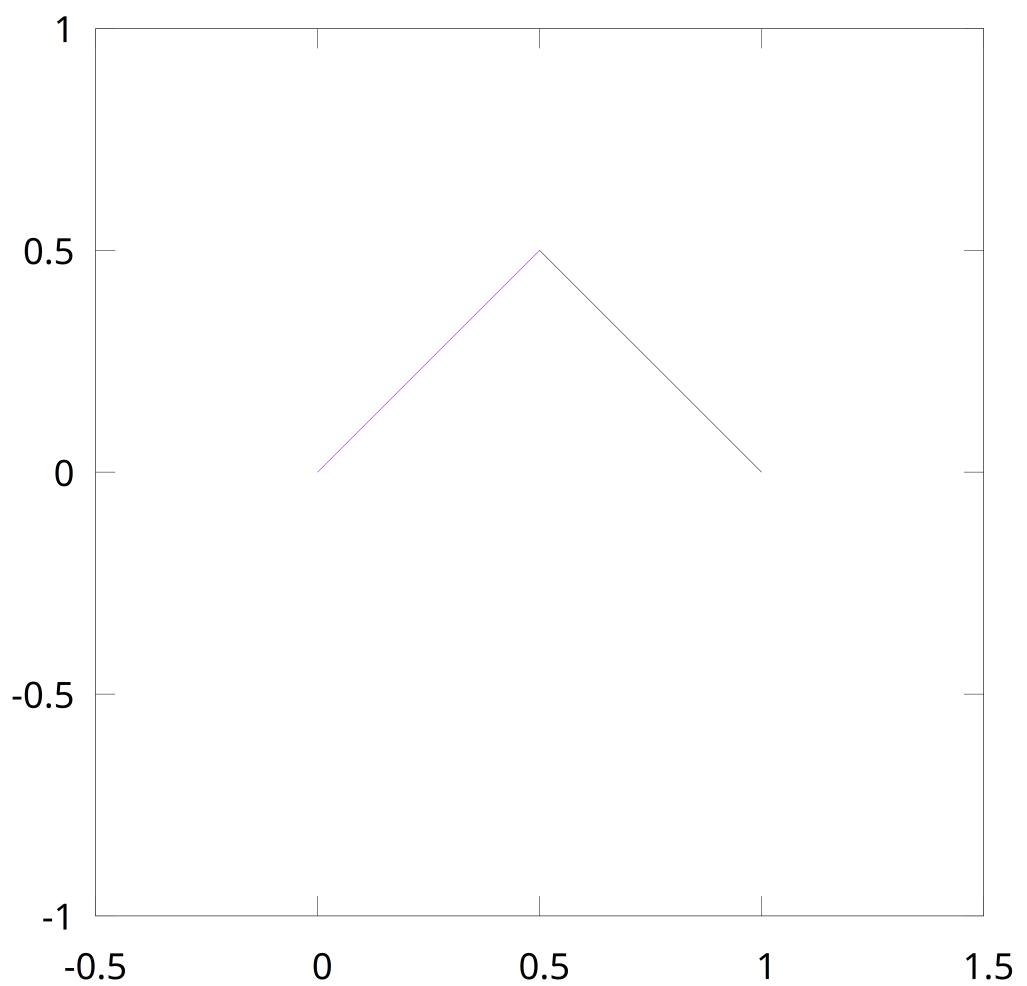


خطوط نازک کشیده شده اند تا در مرتبه بالا نیز توان تفکیک خوب باشد.
کد نوشته شده تا مرتبه زیر 10° به خوبی عمل می کند و در مقادیر بالاتر کمی زمانبر می شود که با بررسی بیشتر مشخص شد مربوط بخش رسم است
و الگوریتم اصلی سرعت مناسبی دارد

مسئله دوم: اژدهای هی وی

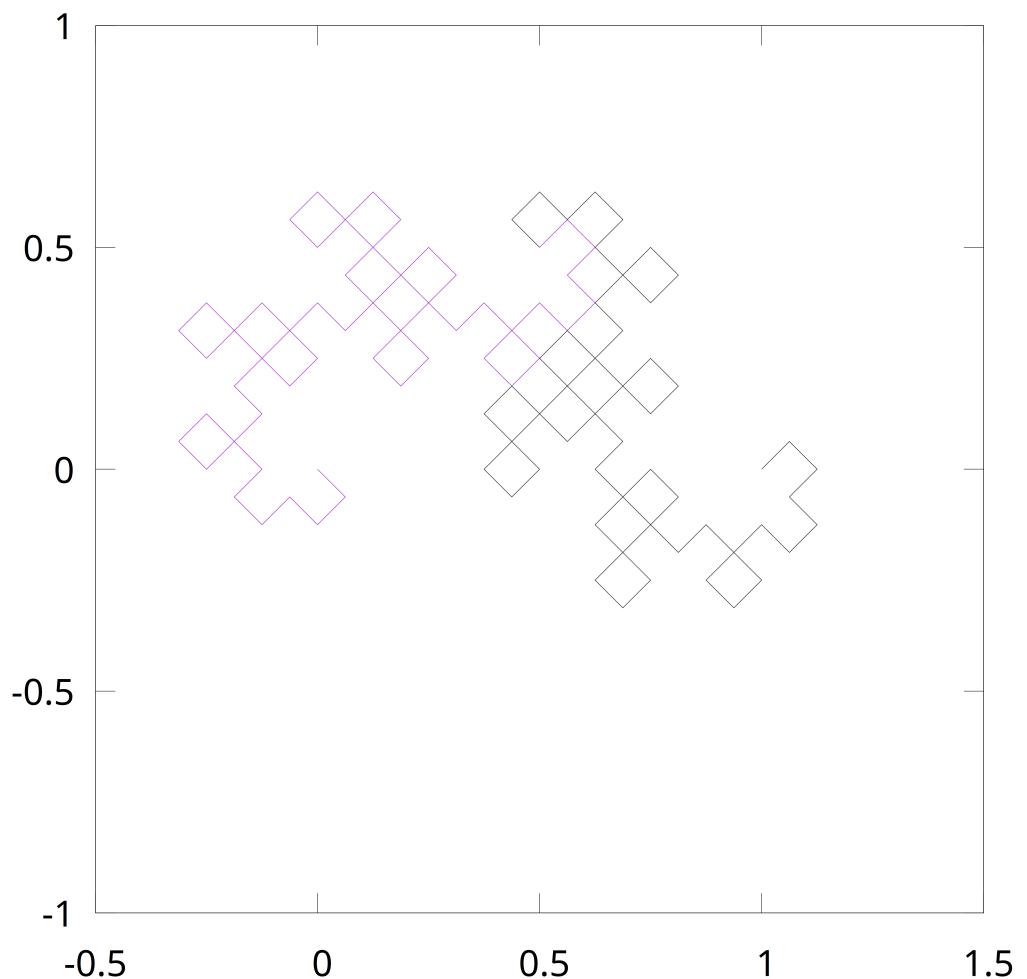
این فراکتال مشابه مسئله قل ساخته می شود با این تفاوت که تنها دوتابع تبدیل کننده داریم و نقاط اولیه متفاوت هستند. همچنین برای رسم، این فراکتال را به دو بخش رنگی مجزا تقسیم می کیم که برخورد نکردن دو مولفه آن بسیار مشخصتر باشد. در ادامه چند مرتبه مختلف از این فراکتال آورده شده است (برای تمام فراکتال ها، مراتب مختلف دیگر نیز در فایل result وجود دارد).

Dragon Fractal N = 0



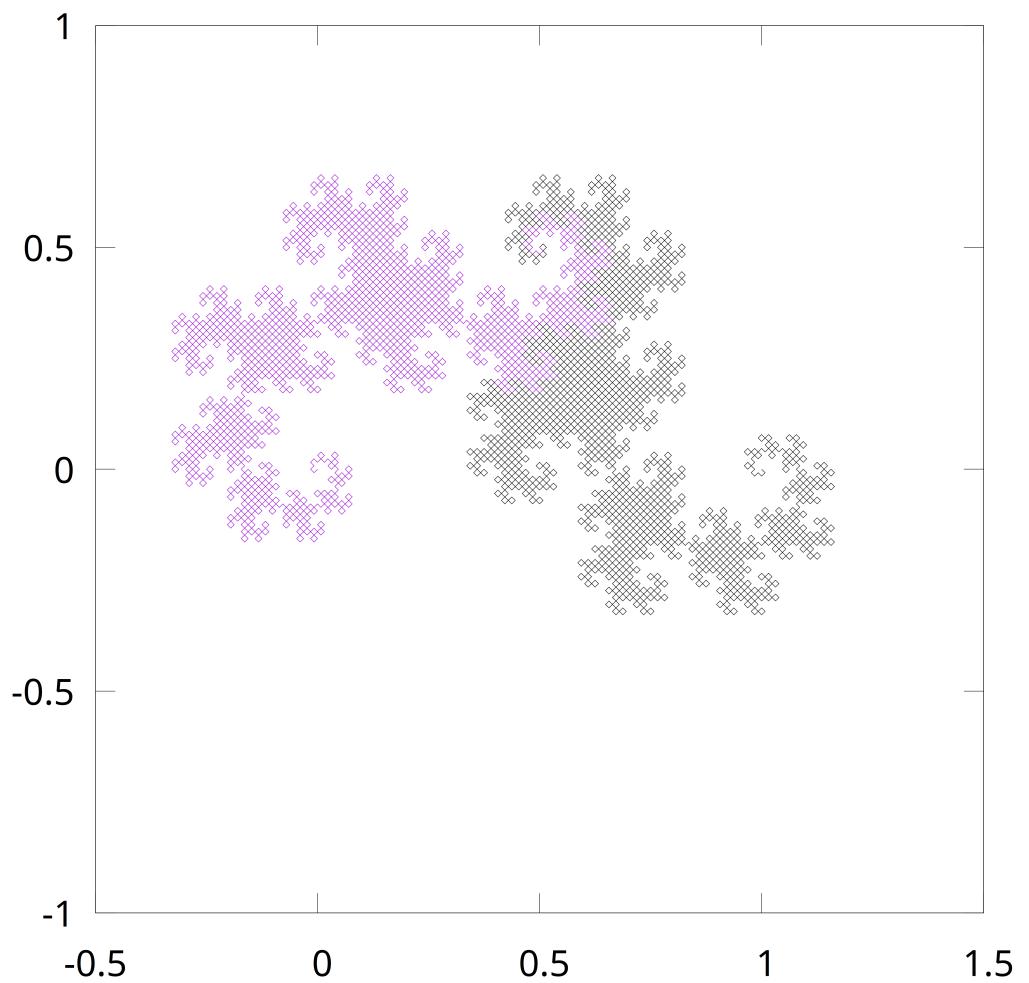
f

Dragon Fractal N = 6



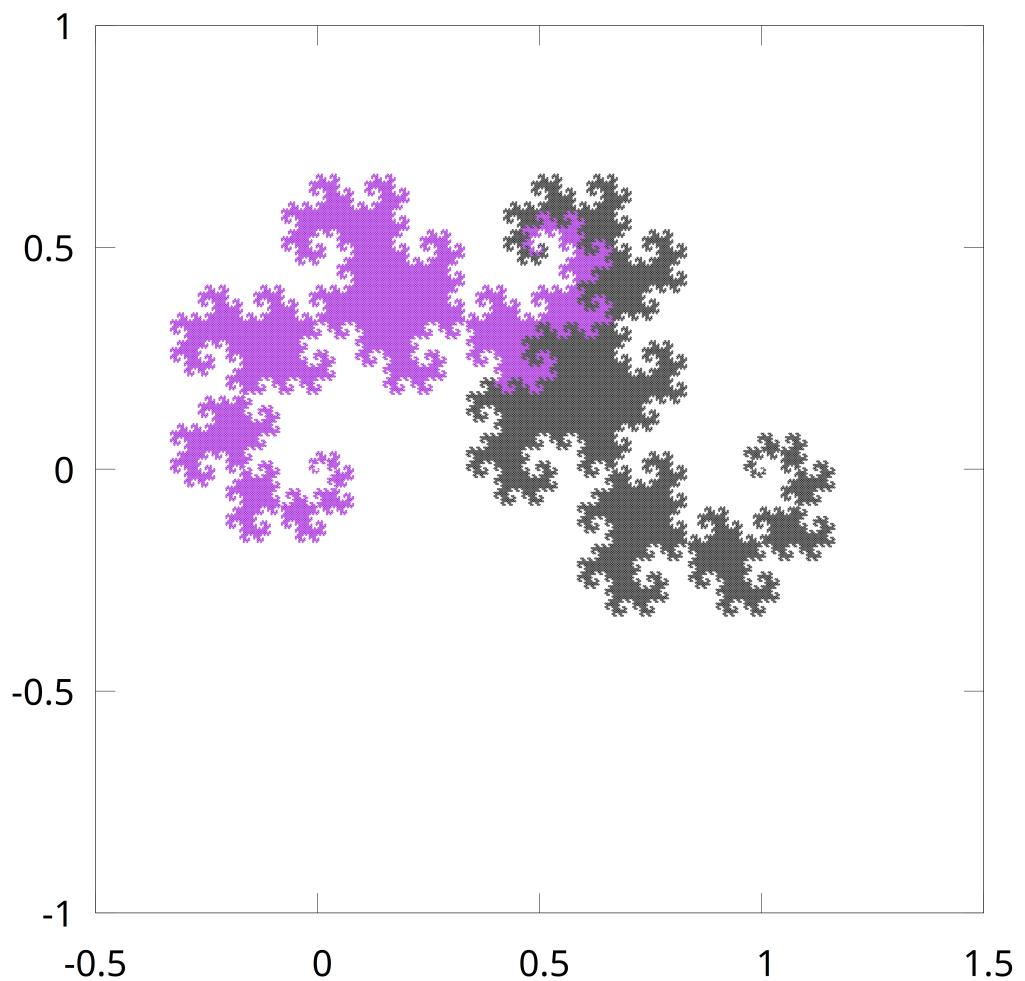
v

Dragon Fractal N = 12

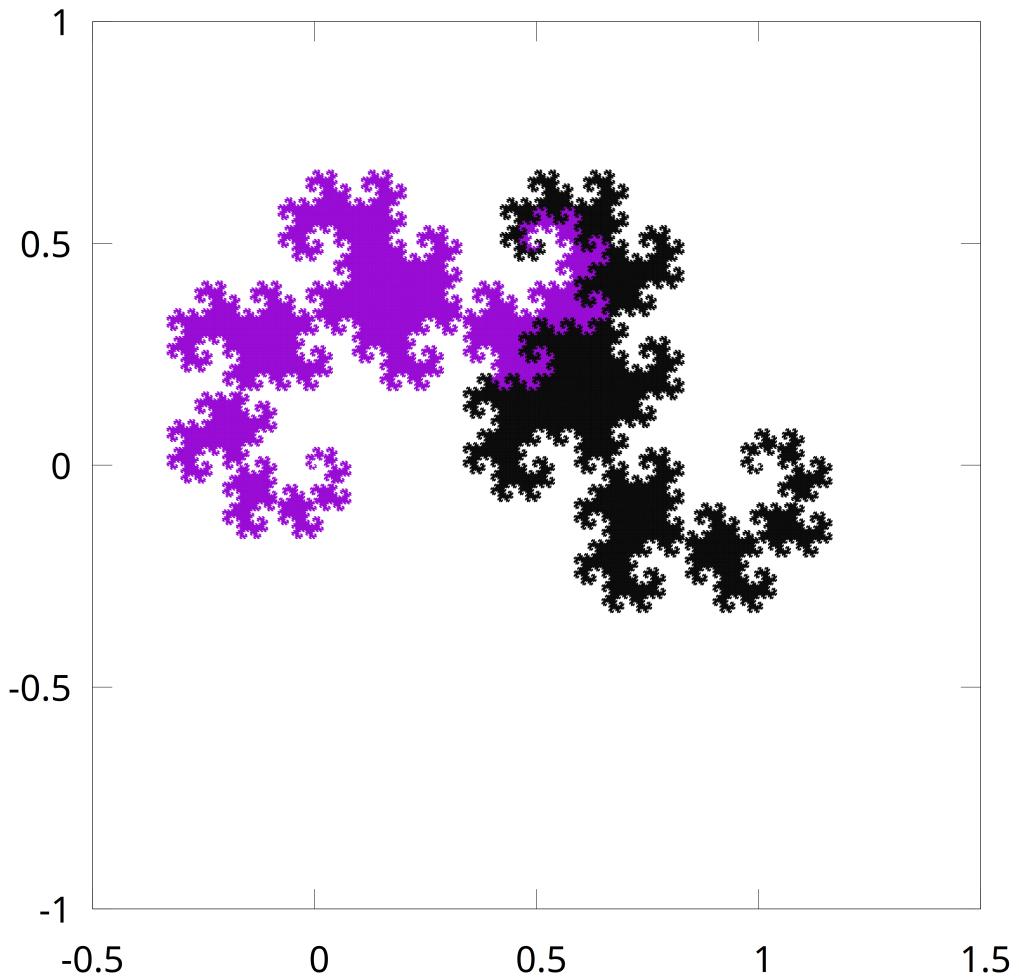


λ

Dragon Fractal N = 16



Dragon Fractal N = 18

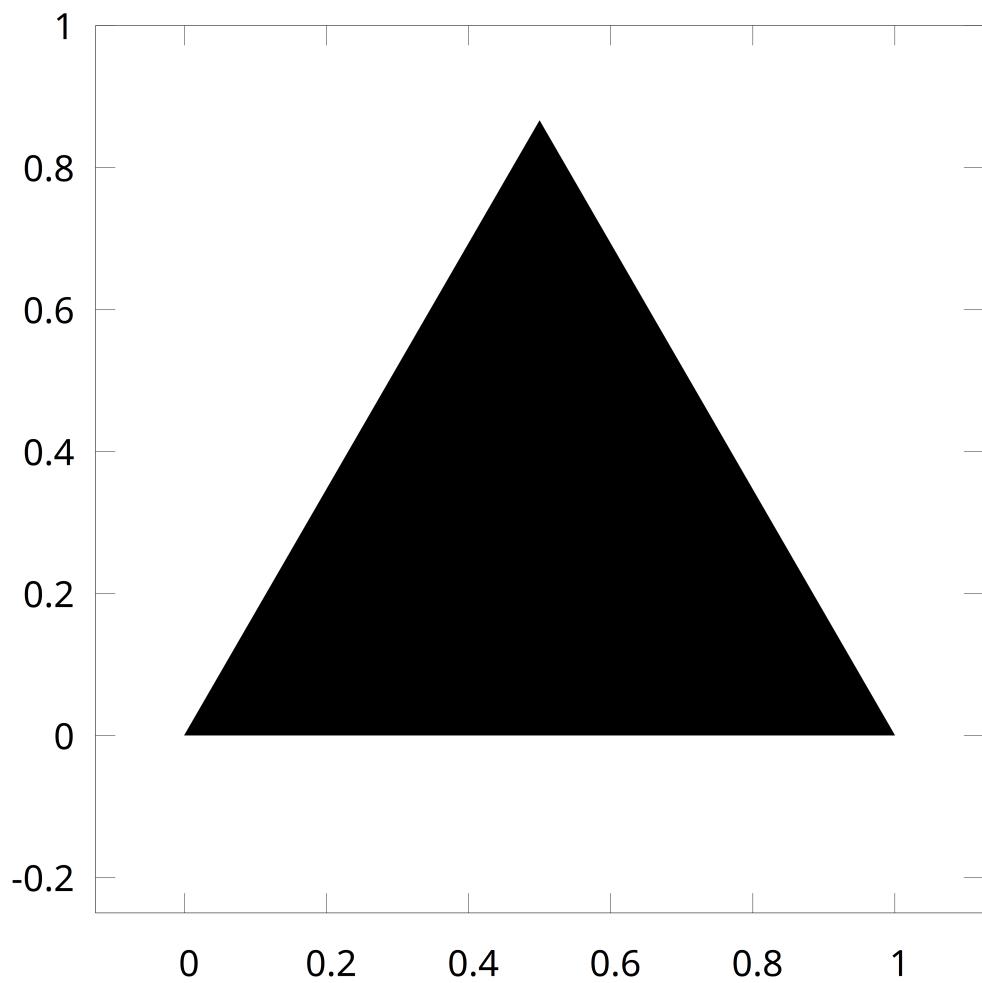


در این تفکیک رنگی، مرز دو بخش آبی و قرمز بسیار زیبا شده است. همچنین در مرحله آخر رنگ قرمز به سیاهی رفته که گمان میکنم به دلیل تراکم بالای نقاط است!

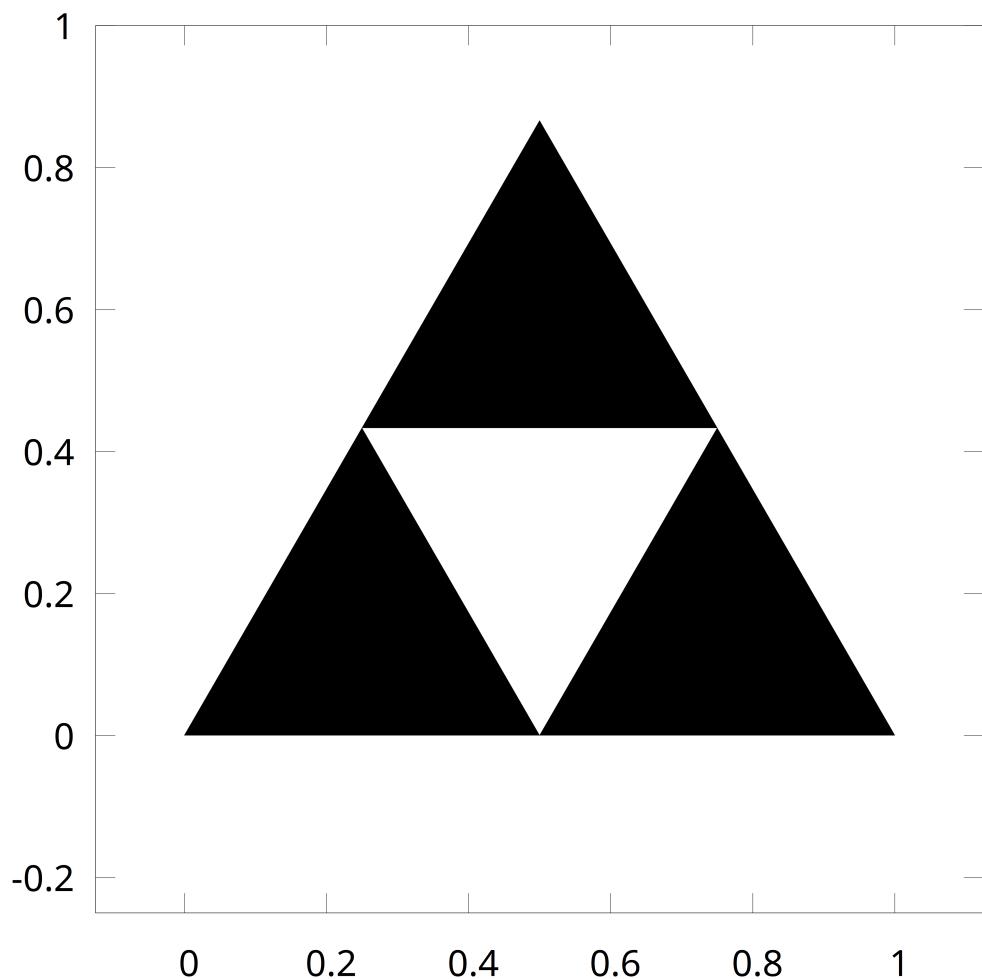
مسئله سوم: مثلث سرپرینسکی

این مسئله نیز مشابه مسائل قبلی است با این تفاوت که سه تابع تبدیل داریم و نقاط اولیه نیز رئوس یک مثلث بسته هستند. برای رسم مثلث توپر در gnuplot کافیست از آرگومان filledcurves استفاده کنیم. اما برای رسم صحیح لازم است که هر مثلث به صورت جدا به برنامه داده شود که کمی از لحظه نوشتنی نازیبا است. با این حال نتیجه خروجی بسیار مطلوب است که در ادامه آورده شده است:

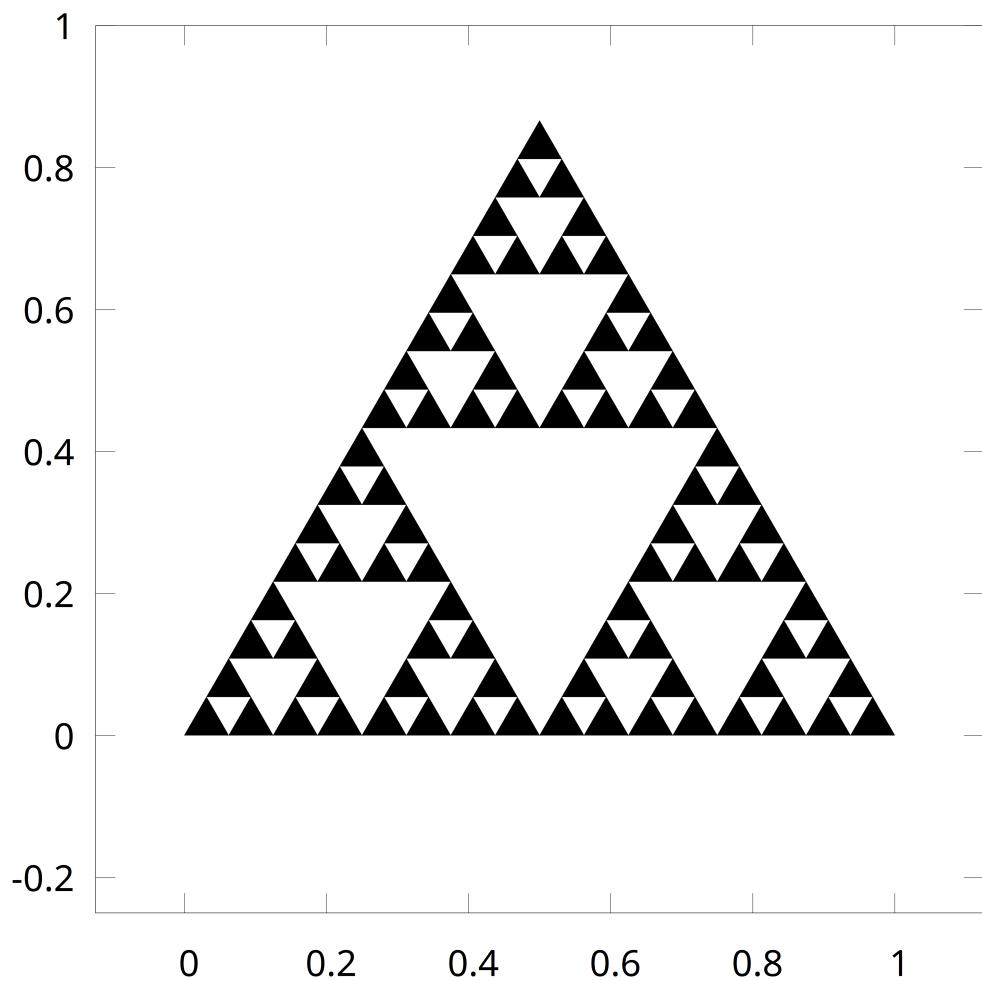
Sierpiński Fractal N = 0



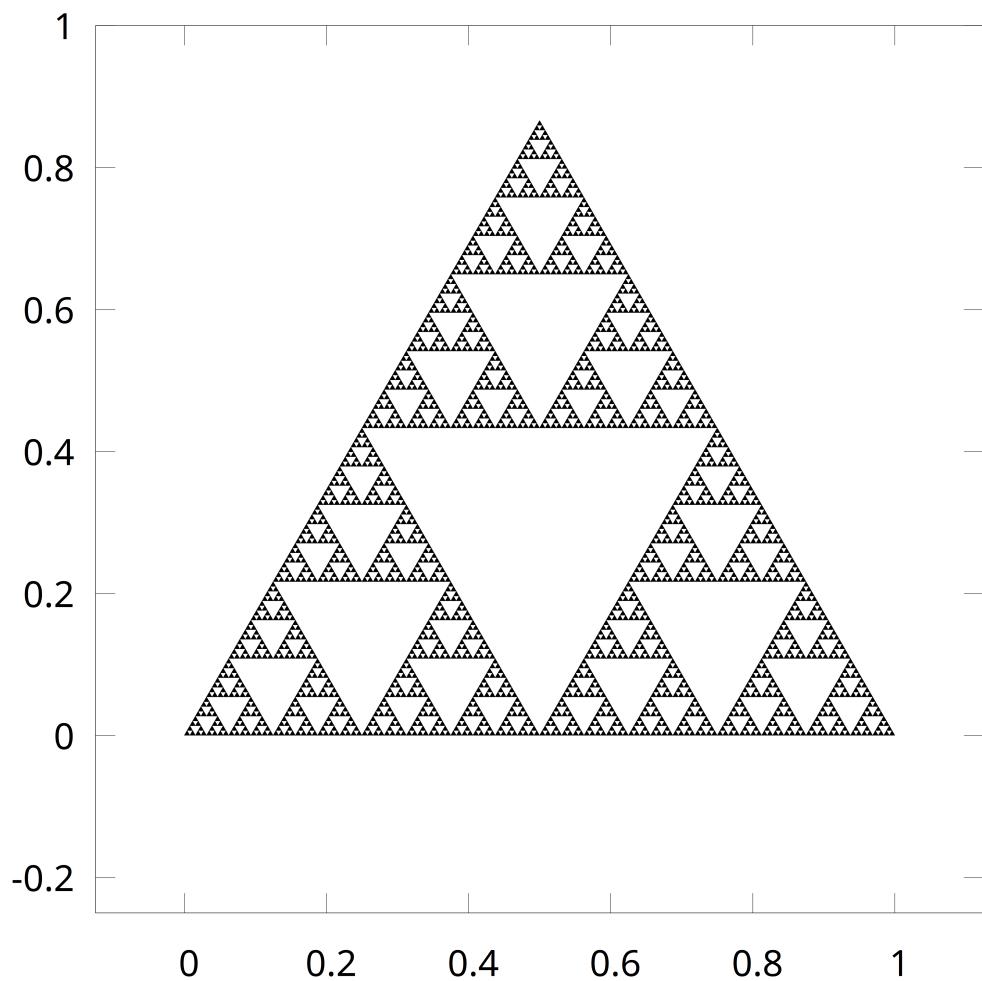
Sierpiński Fractal N = 1



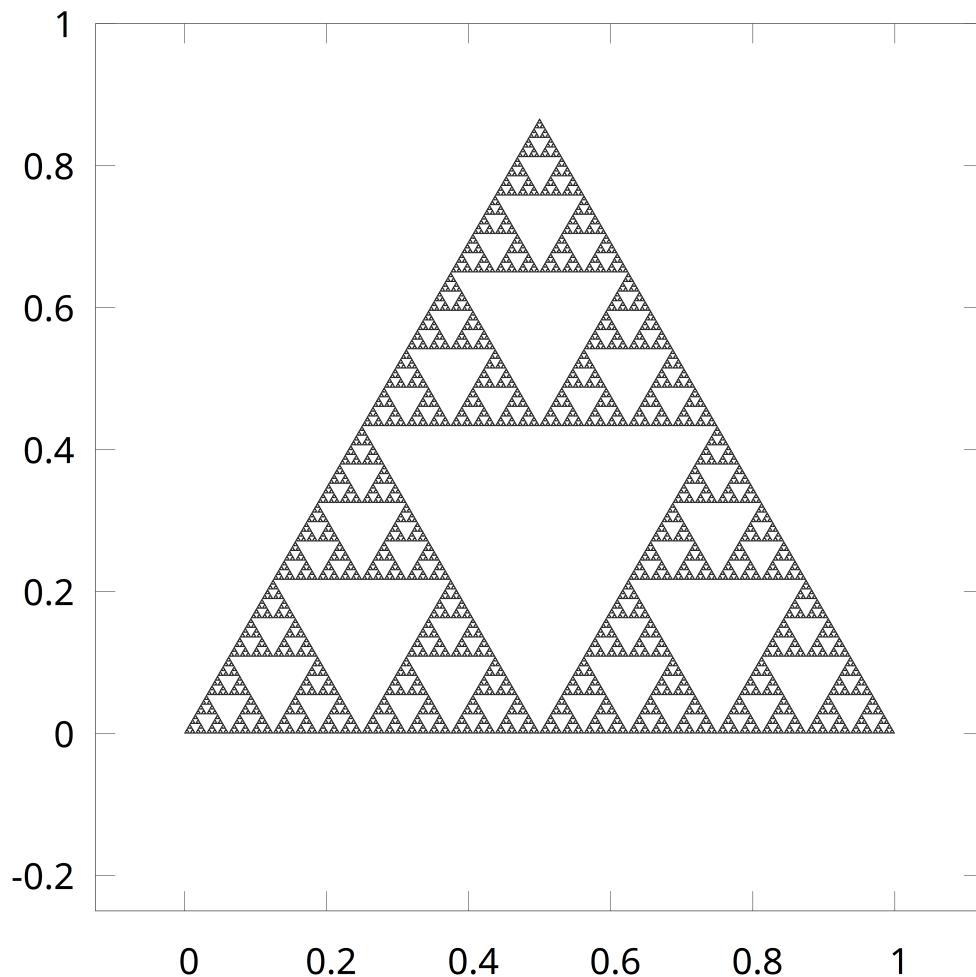
Sierpiński Fractal N = 4



Sierpiński Fractal N = 7



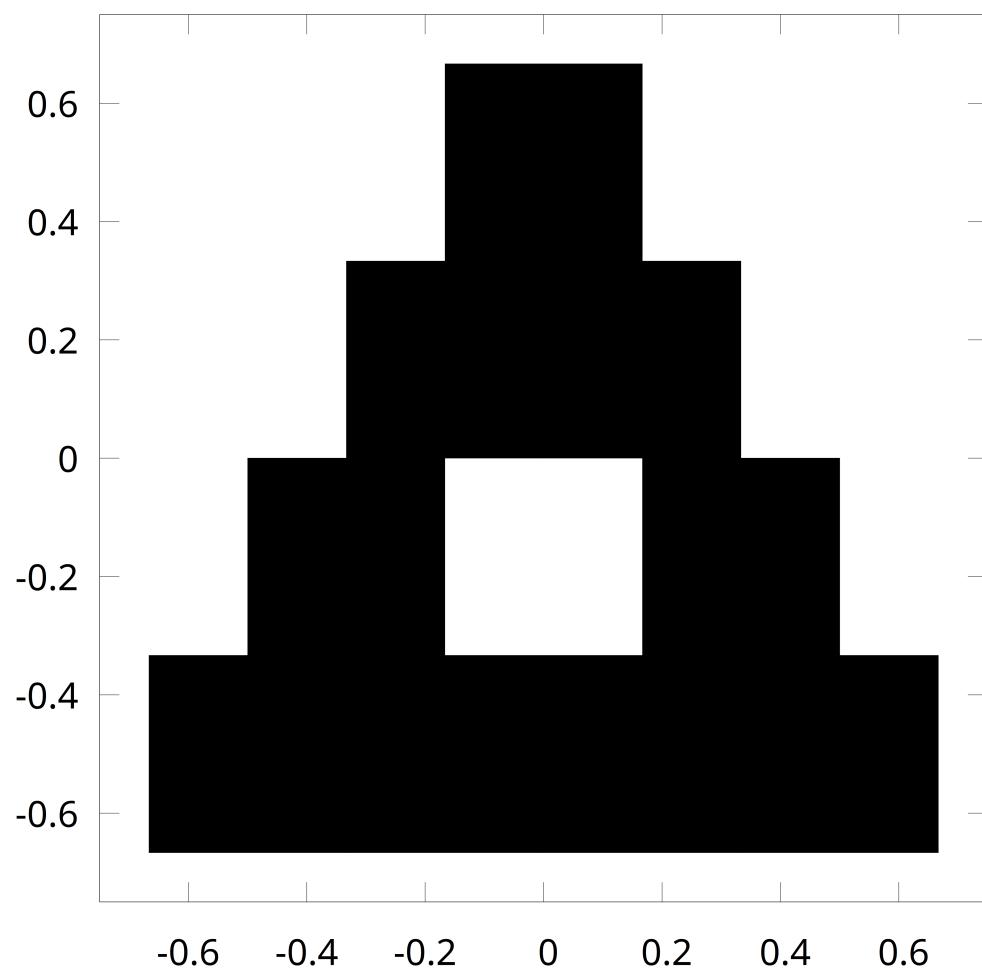
Sierpiński Fractal N = 10



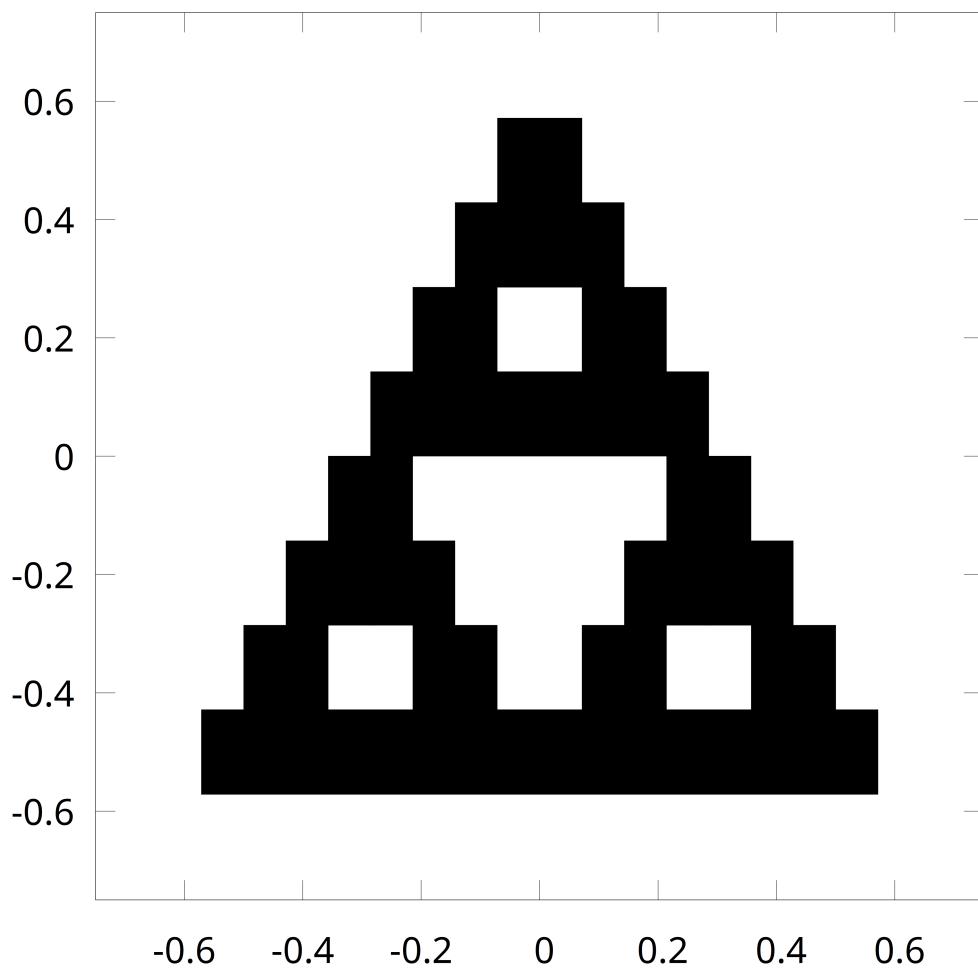
مسئله چهارم: مثلث خیام پاسکال

در این مسئله ابتدا باید بتوانیم ضرایب بسط دو جمله‌ای را محاسبه کنیم. این کار به کمک رابطه بازگشتی قابل محاسبه است که در تابع `combination calc` انجام شده. همچنین نیاز داریم مختصات مربوط به نقاط مثلث را داشته باشیم که این کار به کمک تابع `rn to cord` انجام می‌شود. پیکسل‌ها در این کد به صورت مربع‌هایی در نظر گرفته شده‌اند که به کمک تابع `make square` در اندازه و مکان دلخواه ساخته می‌شوند. سرانجام تابع `make KPT` با گرفتن N یک مثلث خیام پاسکال به این اندازه می‌سازد و در هرجا درایه فرد داشت، پیکسل مربعی مشکی می‌کشد. نتیجه نهایی در ادامه آورده شده:

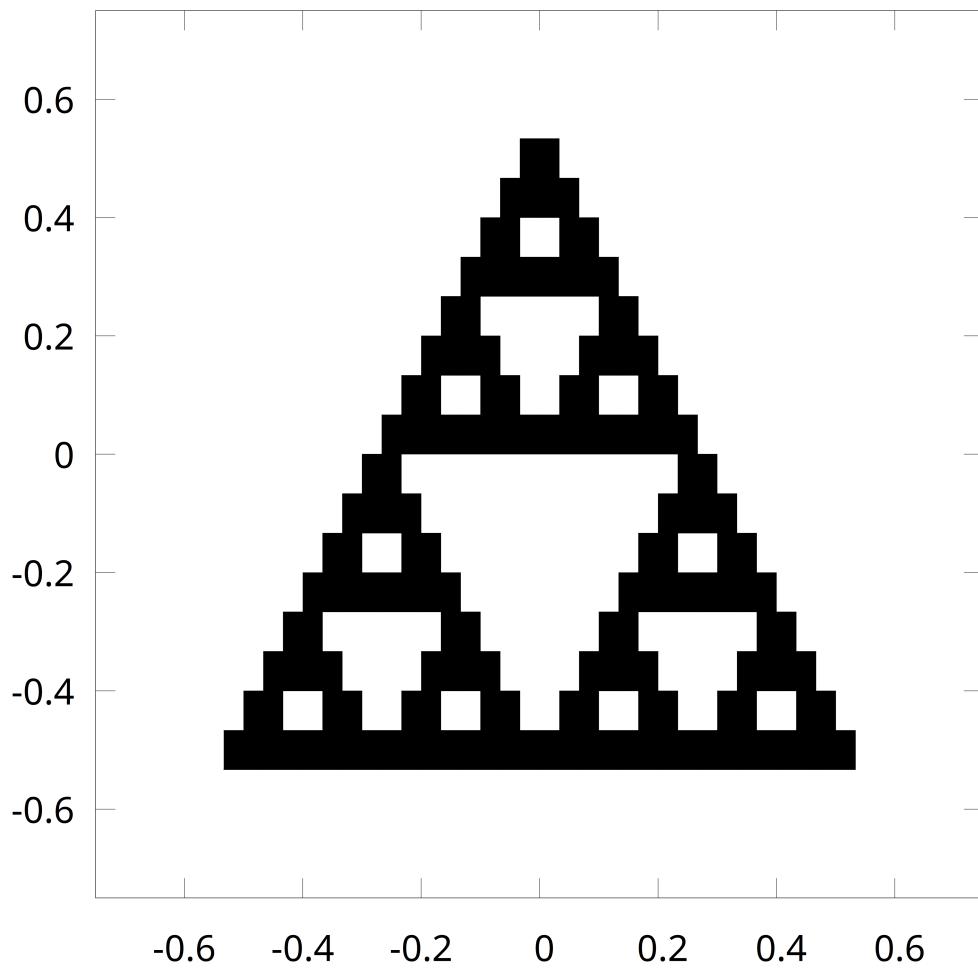
Khayyam-Pascal Triangle N = 3



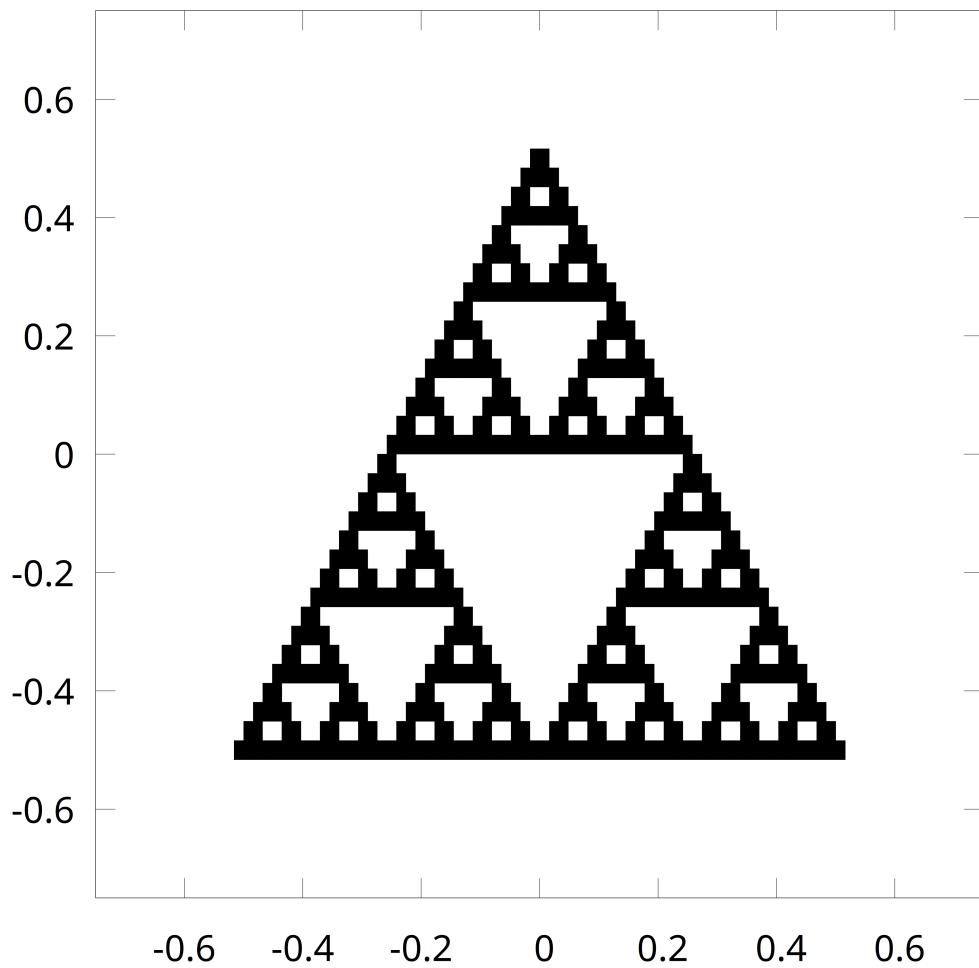
Khayyam-Pascal Triangle N = 7



Khayyam-Pascal Triangle N = 15



Khayyam-Pascal Triangle N = 31



نتیجه نهایی رضایت بخش است اما در مقادیر بالا سرعت کد بسیار کند می شود که این قضیه احتمالاً مربوط به محاسبه ضرایب دو جمله‌ای است. برای بهینه تر کردن این بخش باید از روش‌های دیگری مانند برنامه نویسی پوریا استفاده کیم.