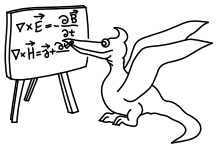


Seriál: Numerické metody a počítačové simulace

V tomto díle seriálu pokročíme k numerickému tématu, které je ve fyzice všudypřítomné: řešení obyčejných diferenciálních rovnic (ODR). Ačkoli nejsou tak složité jako parciální diferenciální rovnice, vyžaduje jejich analytické řešení v mnoha případech pokročilé matematické metody. Numerika nám umožní se matematickým komplikacím vyhnout – pomocí diskretizace derivací získáme integrační schémata, pomocí nichž dokážeme vyřešit každou ODR se zvolenou přesností.

Simulační část seriálu nepokročí, ale naopak zůstane u tématu z minulého dílu. Ukážeme si, že jiné modely růstu povrchů vedou na kvalitativně odlišný vývoj hrubosti povrchu a také si předvedeme, jak použít růstové model v mikrobiologii. Poté se vrátíme k náhodným procházkám a využijeme je k simulaci růstu fraktálního krystalu.

Obyčejné diferenciální rovnice

to-do-to-do, to-do, to-do-to-do-to-do-to-do-to-dooo, to-do-do-do

Růst povrchů II – balistická depozice a Edenův model

Než začneme vysvětlovat nové pojmy, zopakujme si nejprve ve stručnosti poznatky z minulého dílu seriálu. Říkali jsme si, že růst povrchu lze modelovat pomocí stochastického vícestavového buněčného automatu. Automat je stochastický, protože buňku, která v daném kroce bude růst, vybíráme náhodně. Buňkou je zde myšlen jeden sloupeček molekul na mřížce. Že je automat vícestavový znamená, že výška h_i sloupečku, kterou ztotožňujeme se stavem buňky, může nabývat během vývoje více než dvou různých hodnot ($h_i \in \mathbb{N}$). Hrubost povrchu v čase t (po t krocích) na 1D mřížce velikosti L definujeme jako

$$W(t,L) = \sqrt{\frac{1}{L}\sum_i h_i^2 - \left(\frac{1}{L}\sum_i h_i\right)^2}\,. \tag{1}$$

Model náhodné depozice náhodně vybere buňku i a zvětší hodnotu jejího stavu o jedna. Balistická depozice má mírně složitější pravidlo,

$$h_i(t+1) = \max\left(h_{i-1}(t),\, h_i(t)+1,\, h_{i+1}(t)\right)\,. \tag{2}$$

Růst povrchu modelovaný metodou balistické depozice si můžeme představit tak, že částice sestupují z nekonečna na povrch a zastaví se tehdy, když je nějaká částice ihned pod nimi, ale také tehdy, nachází-li se nějaká částice v sousední buňce nalevo nebo napravo. V materiálu tak vznikají dutiny, což se děje i ve skutečných experimentech, například při růstu povrchu pomocí naprašování (epitaxe). Porózní materiál má pak samozřejmě jiné vlastnosti. Ukázka materiálu vzniklého balistickou depozicí je na obrázku 1.

Balistická depozice[1] se však od náhodné depozice neliší pouze porozitou. Podívejme se, jak se vyvíjí hrubost povrchu. Na obrázku 2 je vykreslen vývoj hrubosti vygenerovaný pomocí kódu níže. Vidíme, že na počátku hrubost poměrně rychle narůstá, ale růst zpomaluje a po cca 10 000 krocích se zastaví – došlo k saturaci (nasycení). Tento efekt jsme nepozorovali u náhodné depozice (viz řešení úlohy ke 4. dílu seriálu), kdy hrubost neustále rostla jako mocnina $t^{1/2}$. Obecně můžeme rozdělit vývoj hrubosti na tři časové úseky na základě času přechodu t_x:

1. $t \ll t_\mathrm{x} \to W(t,L) \sim t^\beta$;
2. $t \approx t_\mathrm{x} \to$ přechodová oblast, růst se zpomaluje;
3. $t \gg t_\mathrm{x} \to W(L) \sim L^\alpha$.

Parametry α a β se nazývají *kritické škálovací parametry*. Dále se zavádí *dynamický škálovací parametr* $z = \alpha/\beta$, který popisuje chování času přechodu jako $t_\mathrm{x} \sim L^z$. Nalezení parametrů α a β pro model balistické depozice bude vaším úkolem v seriálové úloze.

```
# BALISTICKA DEPOZICE
# nacteme grafickou knihovnu a numericku knihovnu
import matplotlib as mpl
from matplotlib import pyplot as plt
import numpy as np

# definujeme rozmer mrizky a pocet kroku a interval pro vypocet hrubosti
pocet = 100
roz = 100
krok = 30000
inter = 30
# vytvorime pole naplnene nulami pro stavy a pro hrubost
stav = np.zeros(roz,dtype=float)
hrub = np.zeros(krok/inter,dtype=float)
sumhrub = hrub

for j in range(pocet):
    # v kazdem kroku nahodne vybereme bunku a zvysime jeji hodnotu o 1
    hrub = np.zeros(krok/inter,dtype=float)
    stav = np.zeros(roz,dtype=float)
    for i in range(krok):
        nahod = np.random.randint(roz)
        stav[nahod] = max(stav[nahod]+1.,stav[(nahod-1) % roz],stav[(nahod+1) % roz])
        if i % inter == 0:
            hrub[i/inter] = np.sqrt(1./roz*np.sum(stav**2) - (1./roz*np.sum(stav))**2)
    sumhrub += hrub

sumhrub *= 1./pocet

# nakreslime graf vyvoje hrubosti
# funkce linspace rovnomerne rozdeli pocet kroku na intervaly
body = np.linspace(0,krok-inter,krok/inter)
hrubplot = plt.plot(body,sumhrub,'r',linewidth=2,label='hrubost')
plt.xlabel('krok')
plt.ylabel('hrubost')
plt.show()
```

Druhým růstovým modelem, o kterém se zde blíže zmíníme, je Edenův model. Jeho princip je jednoduchý – částice se může přilepit na libovolné místo na povrchu (tj. seshora i zboku).

[1] Pokud budete hledat o balistické depozici něco na internetu, možná narazíte na jinou definici. Zde jsme definovali NN (nearest neighbour) depozici.

Popis pomocí 1D mřížky se zde ale nehodí. Model si proto představíme jako binární automat (pouze stavy 0 a 1) na 2D mřížce a povolíme růst ze všech směrů. Během simulace je tedy potřeba pamatovat si celý povrch, a to včetně dutin uvnitř. Správným přístupem je ukládat si souřadnice buněk na povrchu, ale programátorsky jednodušší je vytvořit pole $N \times N$, kde N je zvolený rozměr mřížky, a ukládat hodnotu každé buňky. Výpočet je pak samozřejmě pomalý a musíme se omezit na malé mřížky. Příklad povrchu vytvořeného pomocí Edenova modelu vidíte na obrázku 3. K vygenerování byl použit kód níže.

Edenův model přiřazuje růstu uvnitř dutin v materiálu stejnou pravděpodobnost jako růstu na volném povrchu. Porozita tak jednak zvyšuje celkovou délku povrchu, kterou musíme udržovat v paměti, a také nemusí odpovídat skutečnému chování fyzikálního systému, který studujeme. Proto se často zavádějí úpravy modelu, které zvyšují pravděpodobnost zaplňování dutiny úměrně její vzdálenosti od povrchu. Tato úprava samozřejmě ovlivní hodnoty škálovacích koeficientů ve vztazích pro hrubost, ale ne příliš výrazně.

```
# EDENUV MODEL
# nacteme grafickou knihovnu a numericku knihovnu
import matplotlib as mpl
from matplotlib import pyplot as plt
import numpy as np

# definujeme rozmer mrizky a pocet kroku
roz = 200
krok = 20000
# vytvorime pole nul, do ktereho budeme ukladat stavy bunek
# zavorka (kroky, rozmer) udava velikost 2D pole
pole = np.zeros((roz, roz),dtype=np.int)
# inicializujeme
pole[100,100] = 1

for i in range(krok):
    # funkce where vrati indexy pruku, ktere maji hodnotu jedna,
    # ve forme seznamu obsahujiciho dve pole, kde kazde z techto poli
    # obsahuje prislusne x,y souradnice
    indexy = np.where(pole == 1)
    # nahodne vybereme index
    rand = np.random.randint(len(indexy[0]))
    ind = [(indexy[0])[rand],(indexy[1])[rand]]
    # prozkoumame okoli odpovidajiciho pruku
    soused1 = pole[(ind[0]+1) % roz,ind[1]]
    soused2 = pole[ind[0],(ind[1]+1) % roz]
    soused3 = pole[(ind[0]-1) % roz,ind[1]]
    soused4 = pole[ind[0],(ind[1]-1) % roz]
    sousedi = np.array([soused1,soused2,soused3,soused4])
    sous_indexy = np.where(sousedi == 0)
    # pokud jsou sousedi zaplneni, nastavime pole[ind] na 2,
    # abychom ho jiz priste neprohledavali
    if len(sous_indexy[0]) == 0:
        pole[ind[0],ind[1]] = 2
    # nahodne vybereme prazdnou sousedni bunku a zaplnime ji
    else:
        sous_ind = (sous_indexy[0])[np.random.randint(len(sous_indexy[0]))]
        if sous_ind == 0:
            pole[(ind[0]+1) % roz,ind[1]] = 1
        elif sous_ind == 1:
            pole[ind[0],(ind[1]+1) % roz] = 1
        elif sous_ind == 2:
```

```
            pole[(ind[0]-1) % roz,ind[1]] = 1
        elif sous_ind == 3:
            pole[ind[0],(ind[1]-1) % roz] = 1

# definujeme barevne schema pomoci vyctu, v nasem pripade cernobile
barvy = mpl.colors.ListedColormap(['white','black','grey'])
# definujeme prechod mezi barvami kdekoli mezi 0 a 1
hranice=[0,0.5,1.5,2]
# pouzijeme barvy a hranice k normovani barevne mapy
norma = mpl.colors.BoundaryNorm(hranice, barvy.N)
# vykreslime barevnou mapu a zobrazime ji
mapa = plt.imshow(pole,cmap=barvy,interpolation='None',norm=norma,origin='lower')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Kvůli radiální geometrii nelze přímo aplikovat vzorec pro hrubost (1). Proto se nebudeme hrubostí zabývat, ale zavedeme do modelu malou změnu: s pravděpodobností p_1 dojde změníme stav buňky na rozhraní z 0 (prázdná) na 1 (obsazená) a s pravděpodobností p_2 naopak změníme stav buňky na rozhraní z 1 na 0. Tento model, někdy nazývaný Williams-Bjerknesův, je vhodný pro simulaci růstu nádorových buněk nebo bakterií v Petriho misce. V seriálové úloze bude vaším úkolem zkoumat vývoj nádorových buněk pomocí právě popsaného modelu v závislosti na p_1, p_2.

Pro úplnost uveďme, že modelovat růst povrchů lze i jinými metodami než pomocí buněčných automatů. Ke každému zmíněnému růstovému modelu existuje ekvivalentní diferenciální rovnice, kterou lze v některých případech řešit analyticky a získat tak stejné škálovací parametry jako numerickým modelováním. Jedná se však o stochastické diferenciální rovnice, tj. rovnice obsahující člen, který reprezentuje šum. K řešení takových rovnic je potřeba zavést speciální integrální počet, tzv. Itův stochastický kalkulus.

DLA – difúzí limitovaná agregace

Zatím zmíněné růstové modely dokázaly poměrně dobře simulovat laboratorní tvorbu materiálů nebo růst bakteriálních kolonií. Pokud bychom ale chtěli modelovat růst krystalu vznikajícího v roztocích, např. pomocí elektrolýzy nebo sedimentací, bude lepší volbou model DLA (Diffusion limited aggregation).

Fyzikální představa procesu je následující: částice z nekonečna se brownovsky pohybují prostorem, dokud nedojde k jejich kontaktu s krystalizačním jádrem, ke kterému se přilepí. Hustota částic je malá, vzájemně spolu nekolidují (brownovský pohyb zajišťují molekuly v pozadí, které na jádru neulpívají). V počítačové praxi nemůžeme poslat částici z nekonečna, vytvoříme ji proto v určité vzdálenosti od krystalu a necháme ji pohybovat se tak dlouho, dokud nedojde ke kolizi s krystalem, nebo se nedostane za určitou radiální hranici – potom částici přestaneme sledovat a vytvoříme novou.

Níže je příklad kódu (výsledek simulace je na obrázku 4), kde máme jednu částici v počátku hexagonální mřížky a nové částice generujeme ve vzdálenosti, která je rovna radiální vzdálenosti od počátku nejvzdálenější částice krystalu (plus jedna). Částici přestaneme sledovat, pokud se vzdálí na dvojnásobek počáteční vzdálenosti. Snížíme-li maximální povolenou vzdálenost, běh kódu se výrazně zrychlí, ale krystal potom začne vykazovat preferovaný směr růstu. Kód je

poměrně komplikovaný kvůli implementaci šestiúhelníkové mřížky. V seriálové úloze budete simulovat růst na čtvercové mřížce, tím se kód podstatně zjednoduší.

```
# DIFFUSION LIMITED AGGREGATION
# nacteme grafickou knihovnu a numericku knihovnu
import matplotlib as mpl
from matplotlib import pyplot as plt
import numpy as np

# definujeme rozmer mrizky a maximalni pocet castic
roz = 20
mass_max = 100
# pole pro spiralni 1D index, bezpecne vetsi nez je potreba
roz_1d = 6*roz**2
# pole, do kterzch budeme ukladat prepoctene kartezske souradnice
# definujeme vektory pomahajici pri prepoctu z hexa na ctvercovou mrizku
xs = np.zeros(mass_max,dtype=float)
ys = np.zeros(mass_max,dtype=float)
xhelp = [np.sqrt(3.)/2.,0.,-np.sqrt(3.)/2.,-np.sqrt(3.)/2.,0.,np.sqrt(3.)/2.]
yhelp = [-0.5,-1.,-0.5,0.5,1.,0.5]
# pole nul, do ktereho budeme ukladat stavy bunek
pole = np.zeros(roz_1d,dtype=np.int)
# inicializujeme bunky, hmotnost a nejvetsi vzdalenost od stredu
pole[0] = 1
mass = 1
rmax = 0

while mass < mass_max:
    # pocatecni vzdalenost castice
    r = rmax+1
    # pricist k hranici v radialnim smeru, za kterou castice nesmi utect
    rkill = rmax
    # vyber nahodny prvek z sestiuhelniku o rozmeru r
    i = int(6*r*np.random.random() + 1)
    nn = 0
    # pohyb difundujici castice
    while nn == 0:
        i_full = i+3*r*(r-1) # pozice v poli 'pole'
        # nejsme-li ve vrcholu sestiuhelniku
        if (i % r) != 0:
            # nalezneme sousedy
            if i == 1:
                ileft = 6*r
                idownleft = 3*r*(r-1)
            else:
                ileft = i-1
                idownleft = (i*(r-1))//r + 3*(r-1)*(r-2)
            # dvojite lomitko vraci floor
            pole_nn = [i+1+3*r*(r-1),ileft+3*r*(r-1),idownleft,
                       (i*(r-1))//r+3*(r-1)*(r-2)+1,
                       (i*(r+1))//r+3*r*(r+1),
                       (i*(r+1))//r+3*r*(r+1)+1]
            obsazeno = np.where(pole[pole_nn] == 1)
            nn = len(obsazeno[0])
            # je-li nejaky soused obsazen, prilep se
            if nn > 0:
                pole[i_full] = 1
                rmax = max([rmax,r])
                mass += 1
            # jinak difunduj dal
            else:
                step = int(np.random.random()*6)
                # radialni sestup
```

```
                if (step == 2 or step == 3):
                    r -= 1
                # radialni vzestup
                elif (step == 4 or step == 5):
                    r += 1
                i = pole_nn[step] - 3*r*(r-1)
        # pokud jsme ve vrcholu sestiuhelniku
        else:
            if i == 6*r:
                iright = 1
                iupright = 1
            else:
                iright = i+1
                iupright = i*(r+1)/r+1
            if i == 1:
                ileft = 6
            else:
                ileft = i-1
            pole_nn = [iright+3*r*(r-1),ileft+3*r*(r-1),
                       i*(r-1)//r+3*(r-1)*(r-2),i*(r+1)//r+3*r*(r+1),
                       iupright+3*r*(r+1l),i*(r+1)//r-1+3*r*(r+1)]
            obsazeno = np.where(pole[pole_nn] == 1)
            nn = len(obsazeno[0])
            if nn > 0:
                pole[i_full] = 1
                rmax = max([rmax,r])
                mass += 1
            else:
                step = int(np.random.random()*6)
                if step == 2:
                    r -= 1
                elif ((step == 3 or step == 4) or step == 5):
                    r += 1
                i = pole_nn[step] - 3*r*(r-1)
        # pokud castice utekla, zacneme znovu
        if r > (rmax+rkill):
            r = rmax+1
            i = int(6*r*np.random.random()+1)
            continue
    x0 = r*np.sin(2.*np.pi/(6.*r)*(i - (i % r)))
    y0 = r*np.cos(2.*np.pi/(6.*r)*(i - (i % r)))
    x = x0 + (i % r)*xhelp[min(int(i//r),5)]
    y = y0 + (i % r)*yhelp[min(int(i//r),5)]
    xs[mass-1] = x
    ys[mass-1] = y

# vykreslime castice pomoci sestiuhelnikovych symbolu
dlaplot = plt.scatter(xs,ys,c='k',s=30000/(roz**2),marker=(6,0,30))
plt.xlim(-roz, roz)
plt.ylim(-roz, roz)
plt.axes().set_aspect('equal')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

Na vzniklý krystal (shluk částic) můžeme pohlížet jako na fraktál. Nebudeme zde zabíhat do hluboké teorie, fraktál si prostě představíme jako objekt, který při bližším pohledu (myšleno ve větším měřítku) vykazuje stále stejnou strukturu – je soběpodobný. Příkladem z přírody je například list kapradiny, který se skládá z mnoha menších listů, a ty zase z menších lístků. V matematické abstrakci lze na objekt „zoomovat“ donekonečna, vždy budeme nacházet stále stejnou geometrickou strukturu. Podobně na obrázku 4 najdeme na každé větvi rostoucí z

počátku další větev, ze které rostou další větve atd.

Pro fraktály je typické, že jejich dimenze je větší než u „obyčejných“ geometrických objektů. Například kružnice má dimenzi 1, ale Kochova vločka (obr. 5), což je vlastně jenom jinak zakřivená čára, má dimenzi přibližně 1,26. Na Kochově vločce si vysvětlíme definici dimenze[2]. Nechť má strana počátečního trojúhelníku délku $\varepsilon = 1$. Počet úseček v jedné straně je samozřejmě $N = 1$. V dalším kroku již jedna strana obsahuje $N = 4$ úsečky délky $\varepsilon = 1/3$, v dalším kroku $N = 16$ úseček délky $\varepsilon = 1/9$ atd. Fraktální dimenze je pak definována jako

$$D = \lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log \frac{1}{\varepsilon}}\,, (\mathbf{??})$$

přičemž pro vločku můžeme psát $\varepsilon(n) = 1/3^n$, $N(n) = 4^n$, a tedy

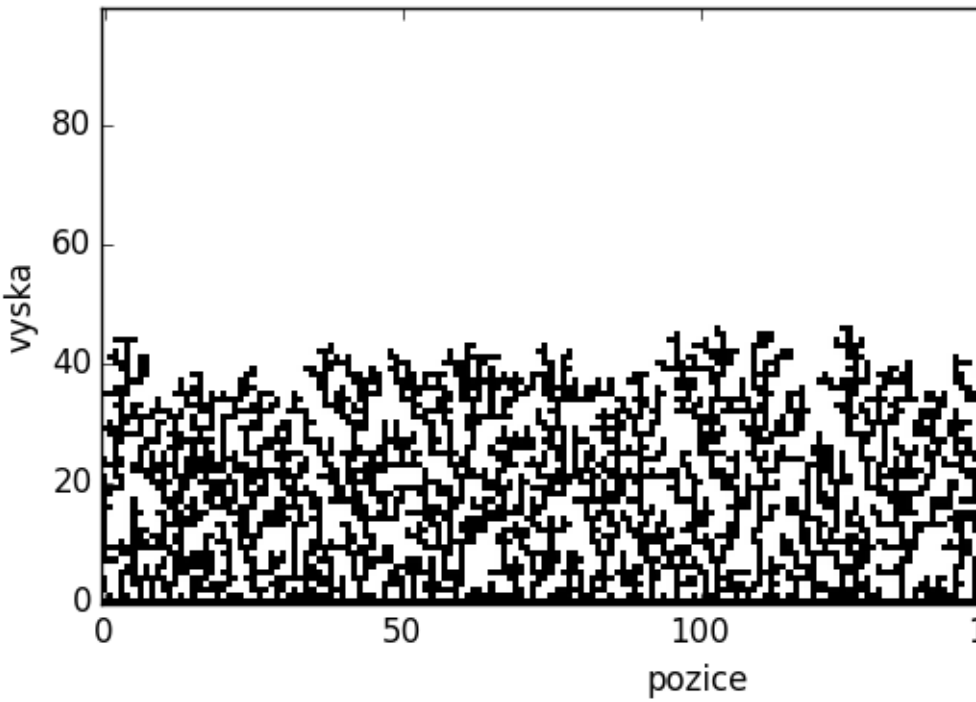
$$D_{\mathrm{Koch}} = \lim_{n \to \infty} \frac{\log 4^n}{\log 3^n} = \frac{\log 4}{\log 3} \doteq 1{,}26\,.$$

Limitní proces nemá smysl v případě Kochovy vločky, ale má smysl v případě fraktálu vytvořeného DLA. Místo strany trojúhelníku zde budeme měřit lineární rozměr krystalu (radiální vzdálenost nejvzdálenější částice od počátku), místo počtu úseček máme počet částic. Přítomnost limity nám říká, že pro co nejpřesnější výsledek chceme nechat krystal růst co nejdéle. Pro krystal na obrázku 4 vychází dimenze $D \doteq 1{,}75$.

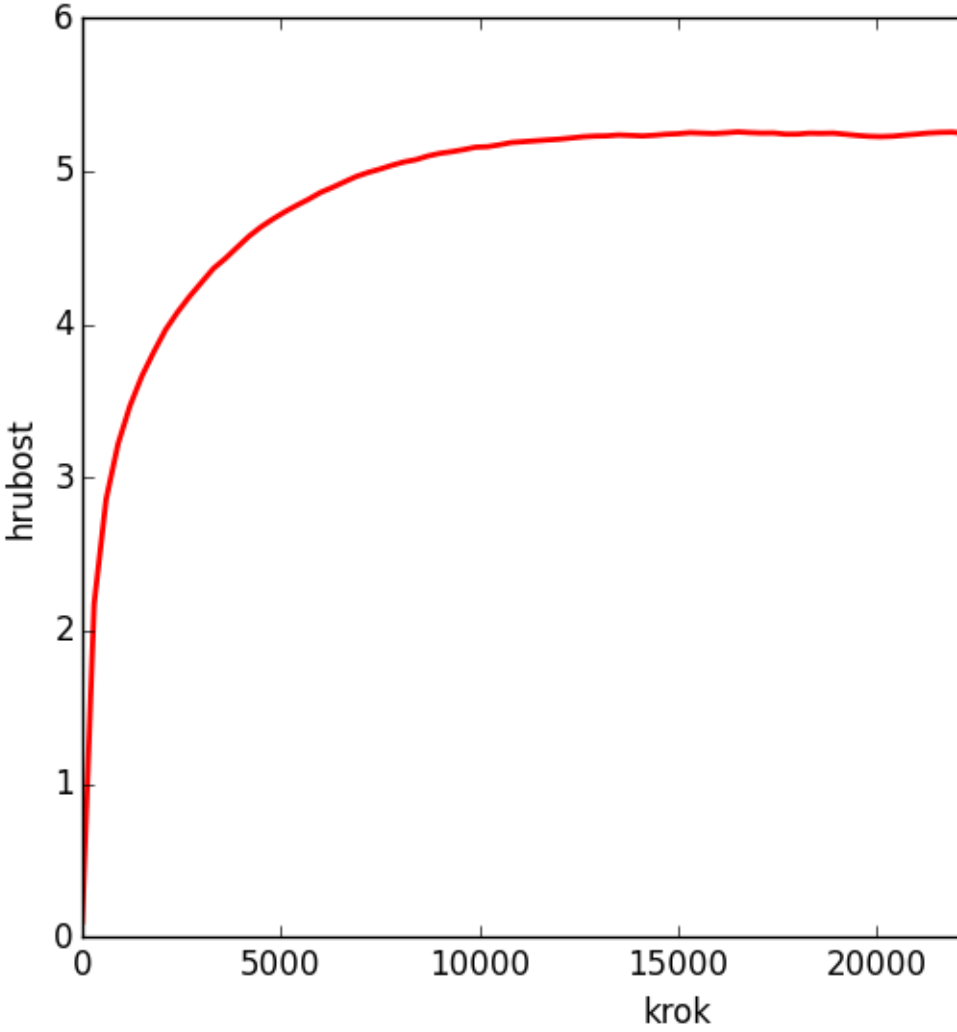
Nakonec ještě dodejme, že v případě materiálu získaného na základě Edenova modelu můžeme pohlížet na jeho povrch jako na fraktál, tělo takového materiálu není fraktální.

Fyzikální korespondenční seminář je organizován studenty MFF UK. Je zastřešen Oddělením pro vnější vztahy a propagaci MFF UK a podporován Ústavem teoretické fyziky MFF UK, jeho zaměstnanci a Jednotou českých matematiků a fyziků.

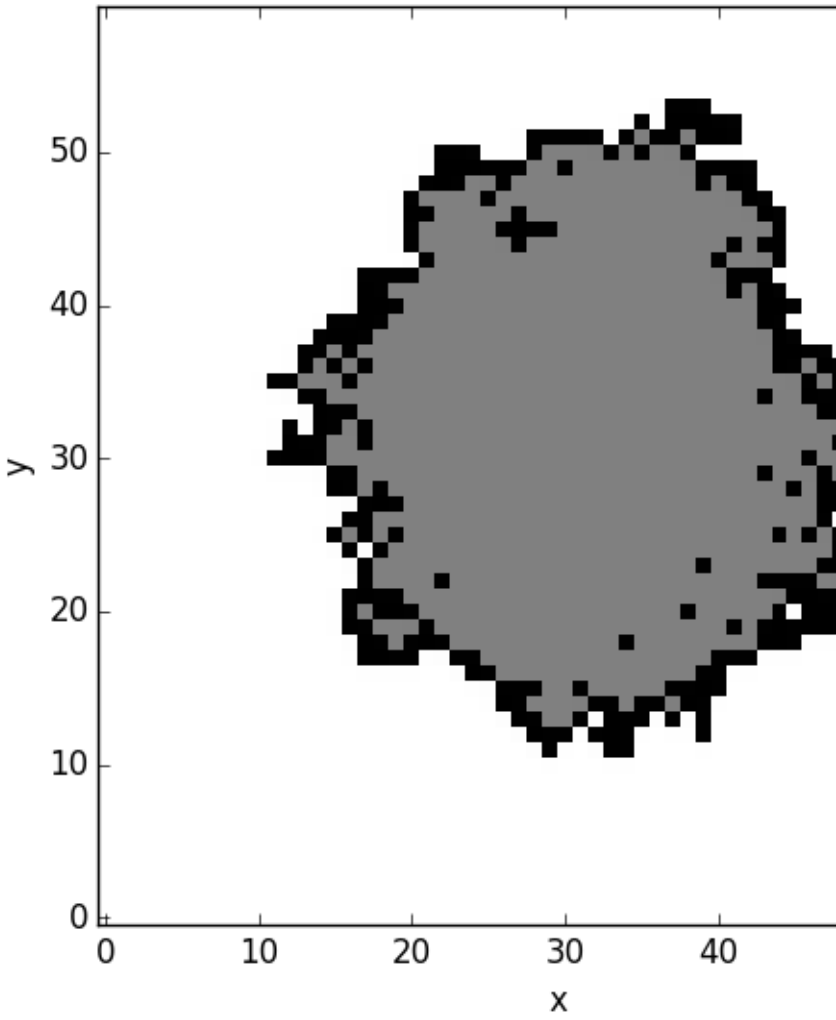
[2] Existuje mnoho způsobů, jak definovat fraktální dimenzi, zde jsme vybrali jeden z nich.



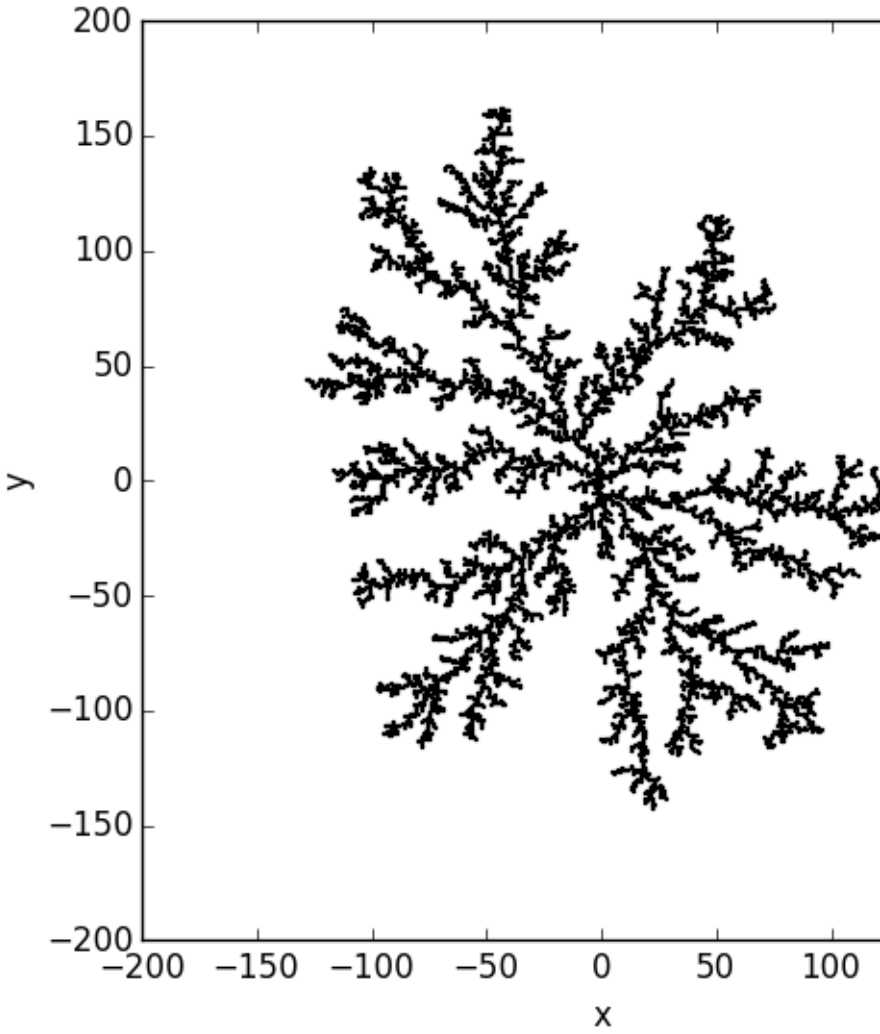
Obr. 1: Struktura povrchu získaného na základě modelu balistické depozice. Všimněte si vysoké porozity.



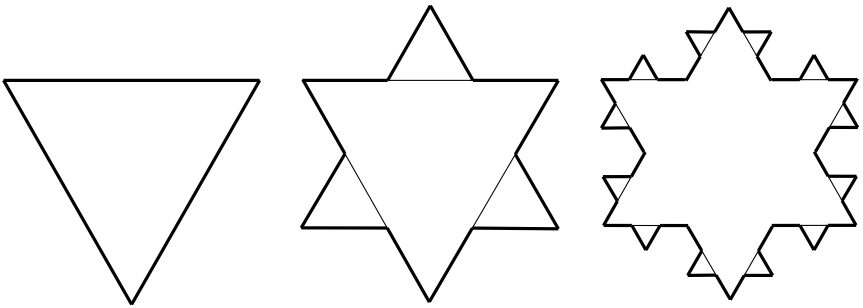
Obr. 2: Příklad vývoje hrubosti povrchu pro simulaci růstu metodou balistické depozice. Průměrováno přes 1000 běhů.



Obr. 3: Výsledek simulace růstu podle Edenova modelu po 2000 krocích. Šedou barvou jsou označeny ty buňky, které byly náhodně vybrány, ale už neměly neobsazené sousedy (bylo by možné vykreslit je také černou barvou, jde jen o ukázku toho, jak funguje použitý algoritmus). Porozita je v porovnání s balistickou depozicí minimální.



Obr. 4: Výsledek simulace růstu podle DLA modelu na hexagonální mřížce. Počet částic v krystalu je 8000. Všimněte si, že i přes poměrně benevolentní limit na maximální vzdálenost difundující částice je na krystalu vidět preferovaný směr růstu.



Obr. 5: První tři iterace Kochovy vločky. Tenké čáry jsou vždy z předchozího kroku.