

# Aproksimacija $\pi$ z metodo Monte Carlo v Matlabu

Mark Krajzel

Univerza v Ljubljani Fakulteta za strojništvo

23. oktober 2023



# Kazalo

- 1 Teoretične osnove
- 2 Program za približek  $\pi$
- 3 Izris
- 4 Zaključek



# Monte Carlo

Ob želji po napovedi obnašanja zapletenih matematičnih sistemov se le tega lahko lotimo z Monte Carlo metodo.

Gre za statistično metodo, ki temelji na naključnosti in jo lahko uporabimo za reševanje problemov, ki so sicer deterministični, vendar so preveč zapleteni za analitično rešitev.



## Kako lahko aproksimiramo $\pi$ ?

- Imamo enotski kvadrat ( $A_{kv} = 4r^2$ ), ki ima včrtan krog ( $A_{kr} = \pi r^2$ ).
- Verjetnost, da bo točka padla znotraj kroga je razmerje med ploščinama:  
$$P = \frac{A_{kr}}{A_{kv}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}.$$
- Približek za  $\pi$  je enak 4-krat tej razdelitvi:  $\pi = 4 \cdot P$ .

## Funkcija `mcc_pi`

- Vzame naključno generirane  $x$  in  $y$  točke in ju shrani v vektorja.
- Definira logični vektor, ki določa kater točke padejo znotraj kroga
- Določi vektor, v katerega sodijo točke iz  $x$  in  $y$  vektorjev, ki imajo v logičnem vektorju vrednost "true".
- Določi vektor, ki vsebuje vse generirane točke iz  $x$  in  $y$  vektorjev.



## Koda za mcc\_pi

```
function [tocke_v_kr, tocke_v_kv] = mcc_pi(stevilo_tock)
    %generacija naključnih točk med -1 in 1%
    x = 2 * rand(stevilo_tock, 1) - 1;
    y = 2 * rand(stevilo_tock, 1) - 1;
    %preverjamo če je koordinata znotraj ali izven kroga
    razdalja = sqrt(x.^2 + y.^2);
    % vektor, ki določa katere točke ustrezajo zgornjemu pogoju
    znotraj_kroga = razdalja <= 1;
    %elementi iz vektorja x, ki ustrezajo "true" vrednostim v zg. vektorju
    tocke_v_kr = [x(znotraj_kroga), y(znotraj_kroga)];
    %vsi elementi iz vektorjev x in y.
    tocke_v_kv = [x, y];
end
```

Slika: Funkcija mcc\_pi



## Približek $\pi$

- Preko *input* funkcije uporabnik generira željeno število točk
- Ustvarimo 2 prazna vektorja za napake in približke
- Uporabimo *for* zanko, da iterativno izračunamo približek za  $\pi$  in napako za vsako novo generirano točko. Število točk se povečuje z vsako ponovitvijo.
- Za vsako iteracijo v zanki uporabimo novo funkcijo *area\_pi*, ki izračuna približek za  $\pi$  in napako. Ta uporablja vgrajeno funkcijo *size*, da določi število vrstic v matrikah, ki vsebujejo generirane točke znotraj kroga in kvadrata.
- Približek =  $4 \frac{\text{točke\_znotraj\_kroga}}{\text{točke\_znotraj\_kvadrata}}$
- Napaka =  $|\pi - \text{Približek}|$



## Približek $\pi$

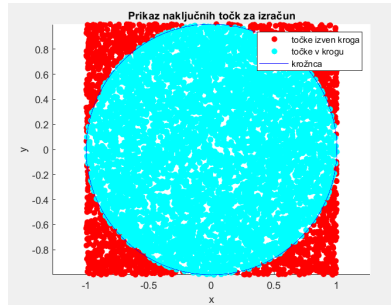
- Preko *input* funkcije uporabnik generira željeno število točk
- Ustvarimo 2 prazna vektorja za napake in približke
- Uporabimo *for* zanko, da iterativno izračunamo približek za  $\pi$  in napako za vsako novo generirano točko. Število točk se povečuje z vsako ponovitvijo.
- Za vsako iteracijo v zanki uporabimo novo funkcijo *area\_pi*, ki izračuna približek za  $\pi$  in napako. Ta uporablja vgrajeno funkcijo *size*, da določi število vrstic v matrikah, ki vsebujejo generirane točke znotraj kroga in kvadrata.
- $\text{Približek} = 4 \frac{\text{točke\_znotraj\_kroga}}{\text{točke\_znotraj\_kvadrata}}$
- $\text{Napaka} = |\pi - \text{Približek}|$





## Rezultat

- Generiramo anonimno funkcijo, ki izriše krožnico, ki loči krog od kvadrata:  
 $lok\_kroz = @(r, phi)[r * cos(phi), r * sin(phi)]$ . Uporabimo tudi vgrajene funkcije *scatter* in *plot* za izris.



Slika: 7000 točk

## Povzetek

Metoda Monte Carlo temelji na naključnosti in se uporablja za aproksimacijo matematičnih konstant, kot je  $\pi$ . Program temelji na uporabi različnih vgrajenih funkcij, kot so *rand* za generiranje naključnih točk, *zeros* za ustvarjanje matrik ničel ter funkciji *scatter* in *plot* za izris grafov. Uporaba funkcije *input* omogoča interaktivno izvajanje programa. Ključna točka je definiranje lastne funkcije in shranitev v funkcijsko datoteko, prav tako pa je uporabna tudi anonimna funkcija. Končni rezultat programa je približek za  $\pi$ , izračunana napaka in izris grafa. Glavna ugotovitev pa je, da z večjim številom generiranih točk dosežemo natančnejši približek za  $\pi$  in manjšo napako.

