Metode de diagnoză
Cursul 12

Programare şi modelare probabilistă - anul III

Facultatea de Informatică, UAIC

e-mail: adrian.zalinescu@uaic.ro

web: https://sites.google.com/view/fiicoursepmp/home

15 Ianuarie 2024

Diagnoza eşantioanelor

Ne vom concentra asupra *sampler*-urilor Metropolis şi NUTS. Deoarece în urma inferenţei se generează un eşantion finit din distribuţia a posteriori, este important să vedem dacă acesta este valid – altfel analiza va fi eronată.

- Există mai multe teste pe care le putem efectua; unele sunt vizuale, altele sunt cantitative.
- Aceste teste sunt create pentru a identifica probleme legate de eşantioane, dar nu pot dovedi că ele aproximează distribuţia corectă.
- Ele ne spun doar dacă eşantioanele noastre arată rezonabil.

Soluţii în caz că găsim probleme cu eşantionul:

- Creşterea mărimii eşantionului;
- Ştergerea unui număr de valori din eşantion de la începutul acestuia. Acest procedeu se mai numeşte *burn-in*. Faza de *tuning* a PyMC reduce nevoia de a face burn-in.
- Modificarea parametrilor sampler-ului, de exemplu creşterea lungimii fazei de tuning sau creşterea parametrului `target-accept` pentru sampler-ul NUTS. Câteodată, PyMC va oferi sugestii pentru aceste schimbări.
- Re-parametrizarea modelului, adică exprimarea modelului într-un mod diferit, dar echivalent.
- Transformarea datelor. Am văzut deja că centrarea datelor îmbunătăţeşte eşantionarea în modelele liniare.

- Vom analiza un model concret, anume un model ierarhic *minimalist*, cu doi parametri:
  - un parametru *global*, a, şi
  - un parametru *local* b (parametrul intra-grup).
- Va fi un model fără date, deci fără verosimilitate.
- Vom discuta două parametrizări alternative:

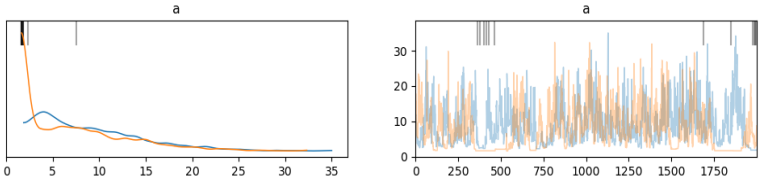
```
with pm.Model() as centered_model:
    a = pm.HalfNormal('a', 10)
    b = pm.Normal('b', 0, a, shape=10)
    idata_cm = pm.sample(2000, target_accept=0.9,
                                        return_inferencedata=True, chains=2)

with pm.Model() as non_centered_model:
    a = pm.HalfNormal('a', 10)
    b_offset = pm.Normal('b_offset', mu=0, sigma=1, shape=10)
    b = pm.Deterministic('b', 0 + b_offset * a)
    idata_ncm = pm.sample(2000, target_accept=0.9,
                                        return_inferencedata=True, chains=2)
```

Convergenţa

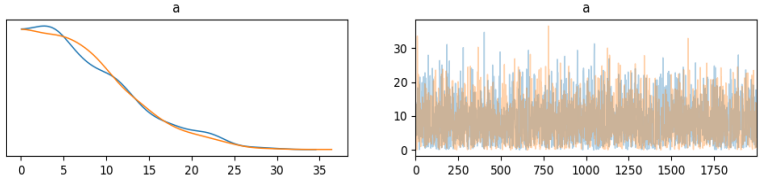
Pentru un sampler MCMC ca NUTS sau Metropolis, poate să dureze ceva timp până când converge.

- Din punct de vedere teoretic, metodele MCMC vin cu garanţii de convergenţă în condiţii foarte generale, dar pentru un eşantion infinit.
- În practic, nu putem avea decât un eşantion finit, aşa că va trebui să ne bazăm pe teste empirice care în cel mai bun caz ne spun (prin indicaţii sau/şi atenţionări) că ceva rău s-ar putea întâmpla când ele eşuează, dar nu garantează că totul este ok dacă nu eşuează.
- O primă metodă este să efectuăm o verificare vizuală cu funcţia `plot_trace` din ArviZ:

```
az.plot_trace(idata_cm, var_names=['a'], divergences='top', compact=False)
```



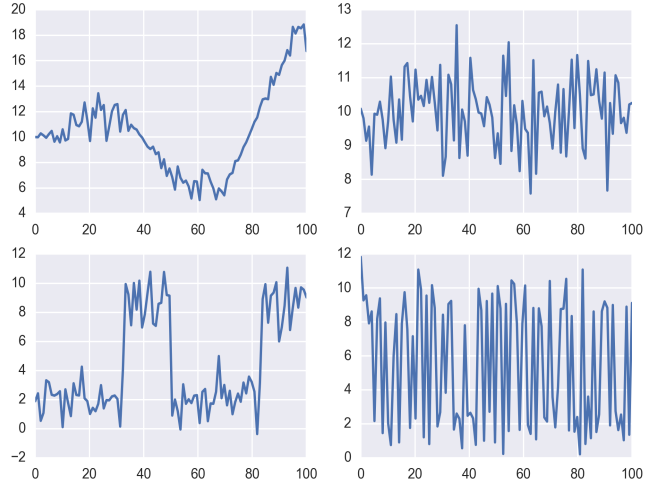

```
az.plot_trace(idata_ncm, var_names=['a'], compact=False)
```



Observăm probleme legate de primul model:

- *KDE* (figura din stânga) pentru cel de-al doilea este mai neted decât primul: un KDE neted este OK, în timp ce unul mai puţin neted poate indica o problemă, de exemplu nevoia de a mări eşantionul.
- *Urma* (*trace*) din dreapta ar trebui să arate ca un *zgomot alb* (*white noise*), adică n-ar trebui să recunoaştem nici un tipar, ca în modelul 2: spunem că avem un *mixaj bun* în cazul acesta.
- Pentru primul model, suprapunerea este mai mică; de asemenea, un lanţ (`chain`) rămâne blocat între "extragerile" 350–500, iar apoi 1750–1900, ceea ce înseamnă că sampler-ul respinge toate noile propuneri cu excepţia celor foarte apropiate.
- Astfel, eşantionarea este foarte înceată şi nu foarte eficientă; dacă eşantionul ar fi infinit, lucrul acesta n-ar fi o problemă, dar pentru unul finit, acesta introduce o *deplasare* (*bias*) a rezultatului.
- Pentru compensarea acestui lucru, am putea mări eşantionul, dar dacă deplasarea este mare, mărimea necesară a acestuia va creşte foarte repede.

Exemple de *trace*: bun (dreapta) şi rău (stânga):



- De asemenea, un bun trace MCMC are proprietatea de *auto-similaritate*: orice două intervale de "extrageri" trebuie să arate similar.
- Dacă prima parte arată diferit, aceasta este o indicaţie că sampler-ul are nevoie de un burn-in sau un eşantion mai mare.
- Dar de cele mai multe ori, dacă două părţi ale trace-ului arată diferit, probabil avem nevoie de o reparametrizare.
- Pentru modelele dificile, se poate încerca o combinaţie de aceste metode.

Statistica Rhat

- Putem de asemenea compara lanţurile (numărul lor depinde de nr. de procesoare şi poate fi specificat cu argumentul `chains`): ele ar trebui să arate similar.
- Pentru a le compara cantitativ vom folosi statistica `Rhat`: ea calculează varianţa inter-lanţuri utilizând varianţa intra-lanţuri.
- În mod ideal, aceasta ar trebui să fie egală cu 1; în practică, o valoare bună este situată sub 1.1 – o valoare mai mare semnalează lipsa convergenţei.
- Ea poate fi calculată cu funcţia `az.r_hat`; este de asemenea returnată de funcţia `az.summary` (by default) sau de `az.plot_forest` (opţional):

```
summaries = pd.concat([az.summary(idata_cm, var_names=['a']),
           az.summary(idata_ncm, var_names=['a'])])
summaries.index = ['centered', 'non_centered']
summaries
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
centered	8.351	5.740	1.610	18.899	0.515	0.365	52.0	31.0	1.03
non_centered	8.003	5.975	0.036	19.238	0.089	0.063	3082.0	1727.0	1.00

Eroarea Monte Carlo

Una din cantităţile returnate de `summary` este `mcse`.

- Aceasta este o estimare a erorii introduse de metoda de eşantionare.
- Ea ia în considerare faptul că "extragerile" nu sunt cu adevărat independente.
- `mcse_mean` este eroarea standard a mediei x a n blocuri, unde fiecare bloc este o porţiune din eşantion:

$$\texttt{mcse\_mean} = \frac{\sigma(x)}{\sqrt{n}}.$$

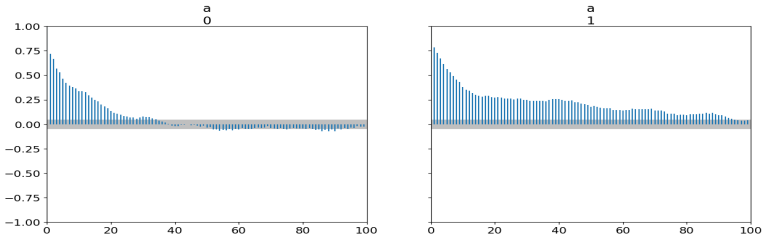
- Analog, pentru `mcse_sd` (eroarea standard a deviaţiei standard).
- Aceste erori ar trebui să fie mai mici decât precizia pe care ne-o dorim în rezultate.

Autocorelaţia

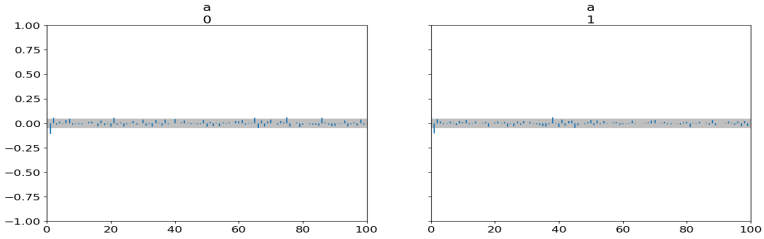
Un eşantion ideal dintr-o distribuţie ar trebui să aibă o autocorelaţie egală cu 0:

- Un eşantion este autocorelat când o valoare la o iteraţie dată nu este independentă de valorile obţinute la alte iteraţii.
- În practică, eşantioanele generate cu metoda MCMC vor fi autocorelate, în special Metropolis-Hastings, şi mai puţin NUTS sau SMC.
- ArviZ are o funcţie utilă pentru vizualizarea autocorelaţiei:

az.plot_autocorr(idata_cm, var_names=['a'])



az.plot_autocorr(idata_ncm, var_names=['a'])



Mărimea efectivă a eşantionului

- Un eşantion cu autocorelaţie posedă mai puţină informaţie decât un eşantion la fel de mare, dar fără autocorelaţie.
- De fapt, putem folosi autocorelaţia pentru a estima mărimea unui eşantion cu informaţie echivalentă, dar fără autocorelaţie.
- Aceasta se numeşte *mărimea efectivă a eşantionului* (*efective sample size*: `ess`) şi poate fi calculată cu funcţia `az.ess`; este de asemenea returnată de funcţia `az.summary` (by default) sau de `az.plot_forest` (opţional)
- În mod ideal, `ess` ar trebui să fie apropiată de mărimea eşantionului.

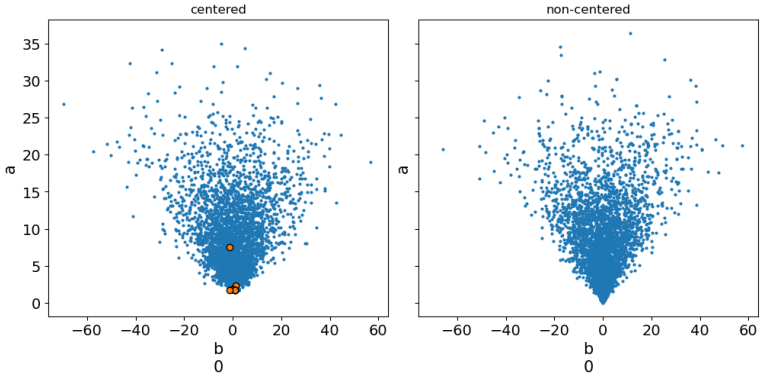
- Un avantaj al NUTS asupra Metropolis este că `ess` în NUTS este în general mult mai mare decât `ess` în Metropolis, şi de aceea, folosind NUTS vom avea nevoie de eşantioane mai mici.
- PyMC ne va avertiza dacă `ess` este mai mică de 200 pentru unul din parametri (100 este de obicei îndeajuns pentru a calcula media distribuţiei).
- Pentru majoritatea problemelor, `ess_bulk` între 1000 şi 2000 este mai mult decât de ajuns.
- Dacă dorim precizie mai mare pentru cantităţi ce depind de capetele (*tails*) distribuţiei sau evenimente ce sunt foarte rare, va trebui să considerăm acelaşi criteriu, dar pentru `ess_tail`.

Divergenţe

Vom explora teste ce sunt specifice metodei NUTS, şi care nu exprimă o proprietate a eşantioanelor generate. Acestea sunt bazate pe aşa-zisele *divergenţe* (PyMC afişează mesaje despre apariţia acestora).

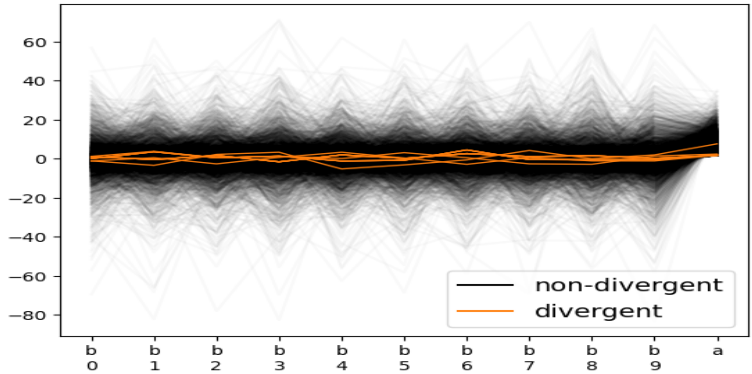
- Divergenţele pot indica faptul că NUTS a întâlnit o zonă de curbură foarte mare în distribuţia a posteriori pe care nu o poate explora eficient.
- Acest lucru ne spune că sampler-ul poate eventual să rateze o zonă din spaţiul parametrilor, şi astfel rezultatele vor fi alterate.
- Divergenţele tind să apară lângă zonele problematice din spaţiul parametrilor şi astfel putem identifica unde ar putea fi probleme.
- O posibilitate de a vizualiza divergenţele este cu funcţia `az.plot_pair` cu argumentul `divergences=True`:

```
_, ax = plt.subplots(1, 2, sharey=True, sharex=True, figsize=(10, 5),
                                                constrained_layout=True)
for idx, tr in enumerate([idata_cm, idata_ncm]):
    az.plot_pair(tr, var_names=['b', 'a'], coords={'b_dim_0':[0]}, kind='scatter',
                 divergences=True, divergences_kwargs={'color':'C1'},
                 ax=ax[idx])
    ax[idx].set_title(['centered', 'non-centered'][idx])
```

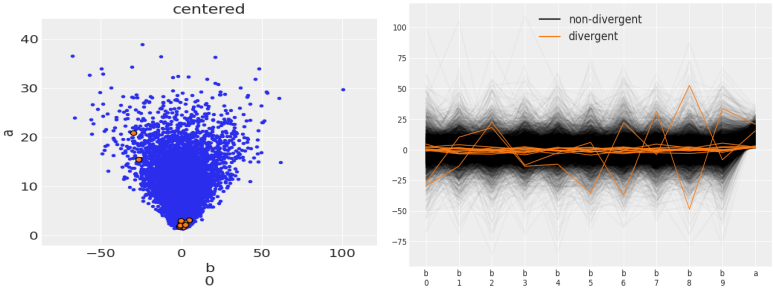


- Punctele mici şi albastre reprezintă valorile eşantionului, pe când cele mai mai mari sunt divergenţele.
- Observăm că ele apar doar în modelul 1 şi sunt în principal concentrate în vârful norului de puncte, ceea ce ne semnalează o zonă cu probleme.
- În `plot_trace`, divergenţele sunt semnalate de simbolul "|".
- O altă metodă de a vizualiza divergenţele este dată de funcţia `az.plot_parallel`:

```
az.plot_parallel(idata_cm)
```



- Aici se vede că divergenţele sunt concentrate în jurul lui 0, şi pentru b, şi pentru a.
- Vizualizarile de tipul precedent sunt importante, pentru că ne arată care părţi ale spaţiului parametrilor sunt problematice.
- PyMC foloseşte o regulă simplificată (*heuristic*) pentru a eticheta divergenţele, iar aceasta poate spune că avem o divergenţă atunci când în realitate ea nu există.
- Ca regulă de bază, atunci când divergenţele sunt împrăştiate în spaţiul parametrilor, avem probabil un fals pozitiv; dacă ele sunt concentrate, atunci probabil există o problemă acolo.



Atunci când obţinem divergenţe, există în general 3 reguli de a scăpa de ele:

1. să creştem numărul de paşi de tuning, de exemplui `pm.sample(tuning=1000);`
2. să creştem valoarea `target_accept` de la valoarea standard de 0.8 la valori mai mari (maxim 1);
3. să reparametrizăm modelul: după cum am văzut în exemplu, modelul 2 este o reparametrizare a modelului 1.