Sprawozdanie - projekt nr 0

Algorytmiczne zastosowania łańcuchów Markowa

Przemysław Chojecki, Michał Rosiński

06.2022

1 Cel projektu

Głównym celem projektu jest znalezienie praktycznego algorytmu szukającego jak najwyższej wartości funkcji wiarogodności pochodzącej z rodziny dokładnie opisanej w rozdziale 2, co pozwoli na lepsze estymowanie macierzy kowariancji w modelu normalnym. W tym celu zaimplementowane zostały różne algorytmy, w tym algorytm symulowanego wyżarzania. Przeprowadzono testy dla różnych ciągów temperatur i dla różnej liczby iteracji w pojedynczym kroku wyżarzania w celu ustalenia parametrów dających najlepsze efekty.
Ponadto zbadano jak zmieniają się własności estymatora macierzy kowariancji po zastosowaniu wspomnianych algorytmów.

2 Opis rodziny funkcji celu

Z punktu widzenia pozostałej części sprawozdania, funkcja celu jest czarną skrzynką. W tym rozdziale opisano, skąd wzięła się rodzina rozważanych funkcji celu. Krótko wytłumaczono, dlaczego znalezienie argumentu z jak największą wartością tej funkcji jest istotne dla zadania uczenia maszynowego.

Skąd się wzięła funkcja celu?

Zakładamy, że wektory losowe są kolumnowe. Załóżmy, że $Z \sim N_p(0, \Sigma)$ jest wektorem losowym długości p o rozkładzie normalnym. Załóżmy, że $(Z_1, Z_2, ..., Z_n)$ jest ciągiem n zmiennych losowych iid. o takim samym rozkładzie jak Z. Jeśli $n \geqslant p$, to istnieje estymator największej wiarogodności Σ i wynosi on:

$$U = \frac{1}{n} \sum_{i=1}^{n} Z_i \cdot Z_i^T.$$

Jeśli jednak $n < p$, to nie istnieje estymator największej wiarogodności Σ. Innymi słowy, nie da się sensownie estymować macierzy kowariancji. Można jednak założyć coś więcej o macierzy Σ. Jeśli założymy na przykład, że przenumerowanie wierszy i kolumn $1 \to 2, 2 \to 3, \ldots, p-1 \to p, p \to 1$ nie zmienia macierzy, to można ją estymować, posiadając zaledwie jedną obserwację, $n = 1$. Widzimy jednak, że jest to bardzo mocne założenie.

Jeśli przenumerowanie wierszy i kolumn zgodnie z permutacją σ nie zmienia macierzy Σ, to znaczy $\Sigma = \sigma^{-1} \cdot \Sigma \cdot \sigma$, to mówimy, że „$\Sigma$ jest **niezmiennicza** względem permutacji σ". W wielu praktycznych zadaniach modelowania możemy założyć niezmienniczość macierzy kowariancji względem jakiejś permutacji. Na przykład, jeśli pomiary, których wyniki są w dwóch kolumnach, wykonywane były tym samym urządzeniem. Trudno jest jednak ekspercko zdecydować o tego typu założeniach. Warto zauważyć, że każda macierz jest niezmiennicza względem permutacji identycznościowej, co można interpretować jako brak dodatkowych założeń.

Autorzy artykułu [1] podali pod numerem (41) wzór na funkcję, która jest proporcjonalna do prawdopodobieństwa a posteriori, że dana obserwacja została wygenerowana przez rozkład niezmienniczy względem permutacji c, czyli:

$$g(\sigma) = \frac{1}{C(Z_1, ..., Z_n)} \cdot p(\Sigma = \sigma^{-1} \cdot \Sigma \cdot \sigma | Z_1, ..., Z_n).$$

Jeśli dla konkretnej obserwacji $(Z_1, ..., Z_n)$ znalezione zostanie σ, dla którego prawdopodobieństwo to będzie największe spośród wszystkich permutacji, to znalezione σ będzie estymatorem największej wiarogodności prawdziwej permutacji, z której pochodzi dany rozkład. Sensownie więc byłoby założyć, że obserwacje te pochodzą z tego rozkładu i można dzięki temu założeniu estymować macierz Σ nawet w sytuacjach, gdy $n < p$.

3 Opis porównywanych algorytmów

Zgodnie z opisem w rozdziale 2, rozważamy zadanie estymacji macierzy Σ, posiadając próbkę $(Z_1, ..., Z_n)$ iid. o rozkładzie $N_p(0, \Sigma)$. W tym rozdziale opiszemy algorytmy użyte do tego zadania.

Podstawowym i powszechnie używanym jest estymator $U = \frac{1}{n} \sum_{i=1}^{n} Z_i \cdot Z_i^T$, który w przypadku gdy $n \geqslant p$ oraz przy braku dodatkowych założeń jest estymatorem największej wiarogodności Σ.

Używając funkcji opisanej w rozdziale 2 można zdefiniować estymator największej wiarogodności dla permutacji σ względem której Σ jest niezmiennicza. Konkretnie:

$$\hat{\sigma} = \arg \max_{\sigma \in \mathfrak{S}_p} g(\sigma)$$

Niestety, definicja ta jest niepraktyczna dla $p \geqslant 10$, gdyż przestrzeń $\mathfrak{S}_p$ jest bardzo duża. Policzenie bowiem wszystkich wartości funkcji $g(\cdot)$ np. dla $p = 10$ zajmuje ponad 10 godzin, natomiast dla $p = 14$ ponad 30 lat.

W tym dokumencie porównane zostały 3 algorytmy przeszukujące przestrzeń $\mathfrak{S}_p$. Są to algorytm największego wzrostu (ang. Best Growth algorithm; BG), algorytmem Metropolisa-Hastingsa (MH) oraz algorytm symulowanego wyżarzania (Simulated Annealing; SA). Algorytmy BG oraz MH były używane na funkcji $g : \mathfrak{S}_p \mapsto \mathbb{R}^+$, gdzie $\mathfrak{S}_p$ to zbiór wszystkich permutacji zbioru $\{1, 2, ..., p\}$. Zatem dziedziną funkcji celu były permutacje o ustalonym rozmiarze. Natomiast dla algorytmu SA użyto logarytmu funkcji $g(\cdot)$, czyli $f(\cdot) := log(g(\cdot))$.

3.1 Opis algorytmu BG

Algorytm BG działa w taki sposób, że zaczyna w stanie $\sigma_0 = ()$, czyli w permutacji identycznościowej, następnie w j-tej iteracji sprawdza wartość funkcji $g(\cdot)$ dla wszystkich sąsiadów stanu σ_{j-1} w którym się znajduje i przechodzi do tego z największą. Czyli:

$$\sigma_j = \sigma_{j-1} \circ \text{argmax}_{t\text{-transpozycja}}\{g(\sigma_{j-1} \circ t)\}$$

oraz kończy, gdy wszyscy sąsiedzi są gorsi od σ_{j-1}. Zwróćmy uwagę, że w jednej iteracji funkcja $g(\cdot)$ liczona jest $\begin{pmatrix} p \\ 2 \end{pmatrix} = \frac{p(p-1)}{2}$ razy.

3.2 Opis algorytmu MH

Algorytm Metropolisa-Hastingsa w swojej podstawowej wersji ma na celu chodzenie po przestrzeni stanów tak, aby częstość odwiedzania stanu była wprost proporcjonalna do wartości funkcji celu dla tego stanu. Dzięki tej własności można stosować go jako algorytm optymalizacyjny, w którym zapisujemy wartości funkcji w odwiedzonych przez niego stanach i mamy nadzieję, że szybko trafi on do stanu jak najlepszego.

3.3 Opis algorytmu SA

Algorytm SA do szukania maksimum funkcji $f(\cdot)$ polega na cyklicznym wykonywaniu algorytmu MH, z coraz większymi wartościami β, na funkcjach

$$\Pi_\beta(i) = \frac{1}{Z_\beta} \cdot \exp(\beta \cdot f(i)),$$

czyli w naszym przypadku

$$\Pi_\beta(i) = \frac{1}{Z_\beta} \cdot g(i)^\beta,$$

gdzie

$$Z_\beta = \sum_{j \in S} \exp(\beta \cdot f(i)) = \sum_{j \in S} g(j)^\beta.$$

Funkcja akceptacji tego MH wynosi więc:

$$a_{i,j} = \min\left(1, \frac{\Pi_\beta(j)}{\Pi_\beta(i)}\right) = \min(1, \exp(\beta \cdot (f(j) - f(i))) = \min\left(1, \left(\frac{g(j)}{g(i)}\right)^\beta\right).$$

Oznacza to, że w pojedynczym kroku zastosowano algorytm Metropolisa dla ustalonej temperatury (odwrotności β). Po przejściu ustalonej liczby iteracji zapisywano większą z dwóch wartości: funkcji celu w momencie startu tego kroku oraz funkcji celu w momencie końca. Dla dodatkowych celów zapisywano też permutacje uzyskane w każdej z iteracji tego kroku oraz ułamek iteracji, w których zaakceptowano przejście do nowej permutacji (ang. acceptance rate).

Przyjęto dwa warunki stopu dla algorytmu symulowanego wyżarzania. Jeżeli określoną liczbę razy ułamek akceptacji był zbyt niski (w poszczególnych krokach) lub jeżeli określoną liczbę razy algorytm pozostawał w punkcie startowym danego kroku, prowadziło to do jego zatrzymania.

Jako startowy punkt dla algorytmu ustalono permutację identycznościową (która jest naturalnym startem dla tego problemu, gdyż zgodnie z opisem w rozdziale 2 interpretuje się ją jako brak dodatkowych założeń na macierz kowariancji, a algorytm na znaleźć założenia, które są bardziej wiarogodne).

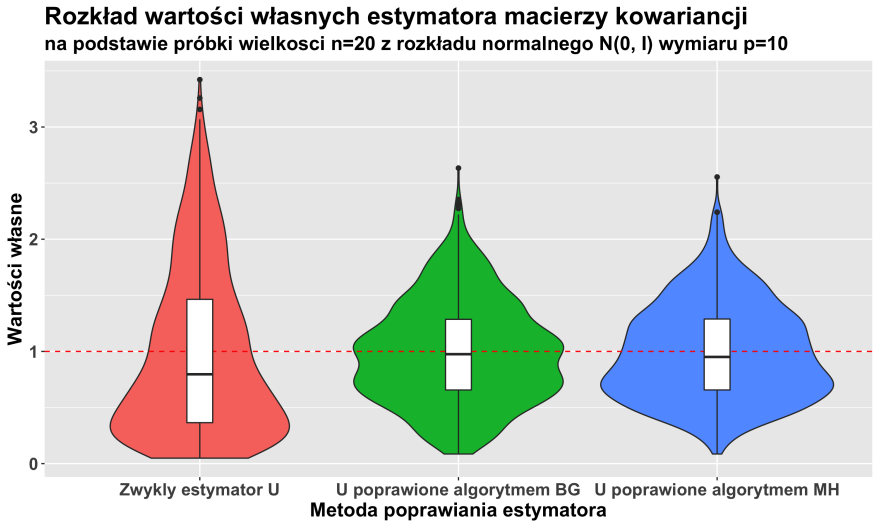
4 Jakość estymacji kowariancji

W tym rozdziale przyjrzymy się jakości estymatorów macierzy $\Sigma = I$ identycznościowej uzyskanych za pomocą podstawowego estymatora, bądź z zastosowaniem poprawek przy pomocy algorytmów BG i MH. Podstawowy estymator $U = \frac{1}{n}\sum_{i=1}^{n} Z_i \cdot Z_i^T$ porównany będzie ze swoją wersją poprawioną przy pomocy permutacji σ wskazanej bądź przez algorytm BG, bądź MH, które szukają permutacji na podstawie wylosowanych danych $(Z_1, ..., Z_n)$.

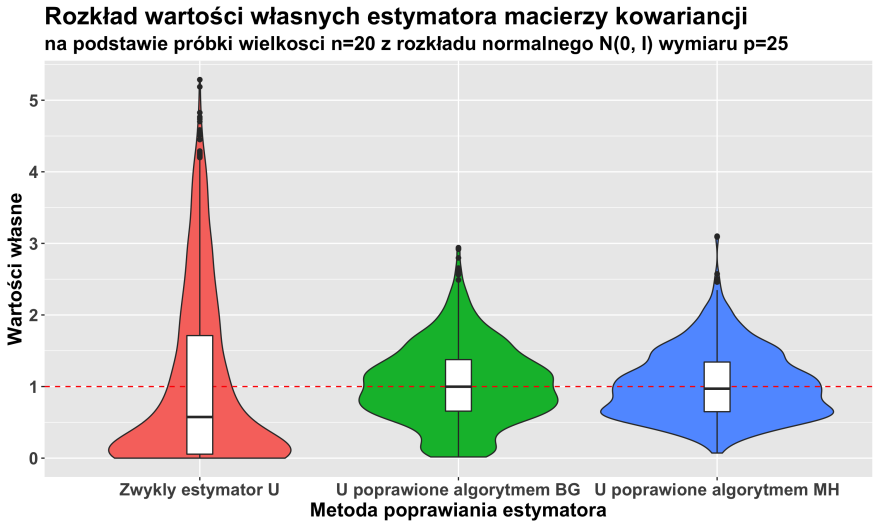
Testy wykonano dla dwóch problemów: $p = 10, n = 20$ (100 powtórzeń eksperymentu) oraz $p = 25, n = 20$ (50 powtórzeń eksperymentu).

4.1 Wartości własne estymatora

Na wykresach 1 oraz 2 zwizualizowano rozkłady wartości własnych estymatorów macierzy kowariancji. Porównano typowy estymator U oraz poprawki znalezione za pomocą algorytmów BG i MH. Widzimy na tych wykresach, że wprowadzenie poprawki zdecydowanie poprawiło rozkład wartości własnych dla obu eksperymentów.



Rysunek 1: Wykres skrzypcowy oraz boxplot wartości własnych estymatora macierzy kowariancji U, oraz jego poprawionych wersji algorytmami BG i MH. Pożądane jest jak największe zbliżenie się do wartości 1. Estymacja na podstawie próbki wielkości $n = 20$ dla wymiaru $p = 10$. Efekt skumulowanych 100 wywołań.

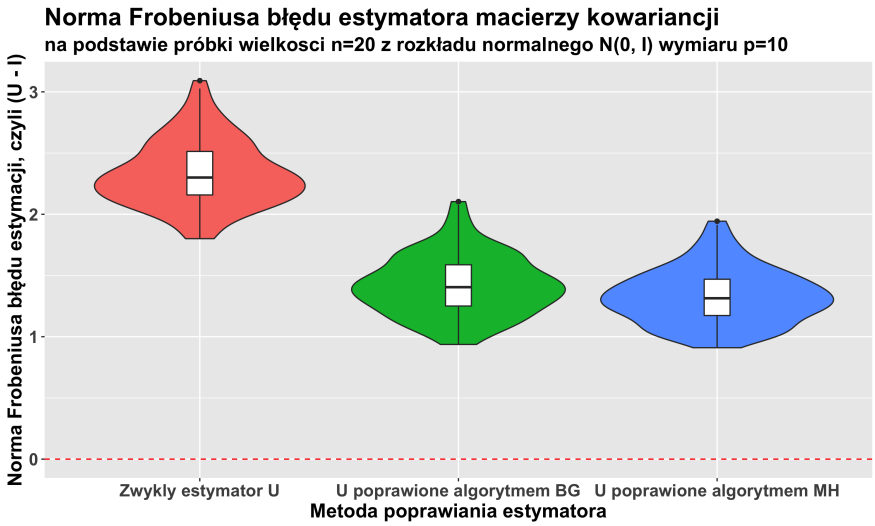


Rysunek 2: Wykres skrzypcowy oraz boxplot wartości własnych estymatora macierzy kowariancji U, oraz jego poprawionych wersji algorytmami BG i MH. Pożądane jest jak największe zbliżenie się do wartości 1. Estymacja na podstawie próbki wielkości $n = 20$ dla wymiaru $p = 25$. Efekt skumulowanych 50 wywołań.

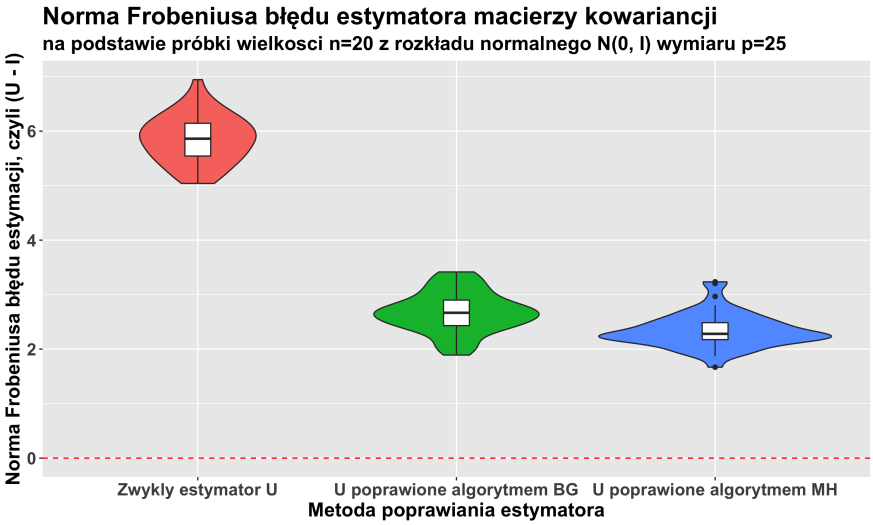
4.2 Norma Frobeniusa błędu estymacji

Na wykresach 3 oraz 4 zwizualizowano rozkłady norm Frobeniusa błędów estymatorów macierzy kowariancji. Porównano typowy estymator U oraz poprawki znalezione za pomocą algorytmów BG i MH. Widzimy na tych wykresach, że wprowadzenie poprawki zdecydowanie poprawiło jakość estymacji dla obu eksperymentów.

Wykonaliśmy test t-Studenta na równość średnich w celu porównania efektu BG i MH. W teście tym p-wartość dla eksperymentu z $p = 10$ wyniosła 0.008, a dla eksperymentu z $p = 25$ wyniosła $2 \cdot 10^{-6}$, więc w obu przypadkach odrzucamy hipotezę o równości średnich. Poprawka MH bardziej zmniejsza błąd średniokwadratowy estymatora (Mean Squared Error, MSE).



Rysunek 3: Wykres skrzypcowy oraz boxplot normy Frobeniusa błędu estymatora macierzy kowariancji U, oraz jego poprawionych wersji algorytmami BG i MH względem prawdziwej wartości. Pożądane jest jak największe zbliżenie się do wartości 0. Estymacja na podstawie próbki wielkości $n = 20$ dla wymiaru $p = 10$. Efekt skumulowanych 100 wywołań.



Rysunek 4: Wykres skrzypcowy oraz boxplot normy Frobeniusa błędu estymatora macierzy kowariancji U, oraz jego poprawionych wersji algorytmami BG i MH względem prawdziwej wartości. Pożądane jest jak największe zbliżenie się do wartości 0. Estymacja na podstawie próbki wielkości $n = 20$ dla wymiaru $p = 25$. Efekt skumulowanych 50 wywołań.

5 Dobór odpowiednich parametrów algorytmu SA

Przeprowadzono testy różnych parametrów algorytmu SA, w celu znalezienia dobrze dostosowanych dla problemu. Na potrzeby testów wyłączono warunki stopu w algorytmie symulowanego wyżarzania. Eksperymenty te zostały w tym rozdziale podsumowane.

5.1 Wykres ECDF

W celu zaprezentowania wyników eksperymentów wykorzystany został powszechnie używany w literaturze wykres zwany ECDF. W tym podrozdziale opiszemy jak został skonstruowany, jak go czytać i jak na jego podstawie porównywać algorytmy.

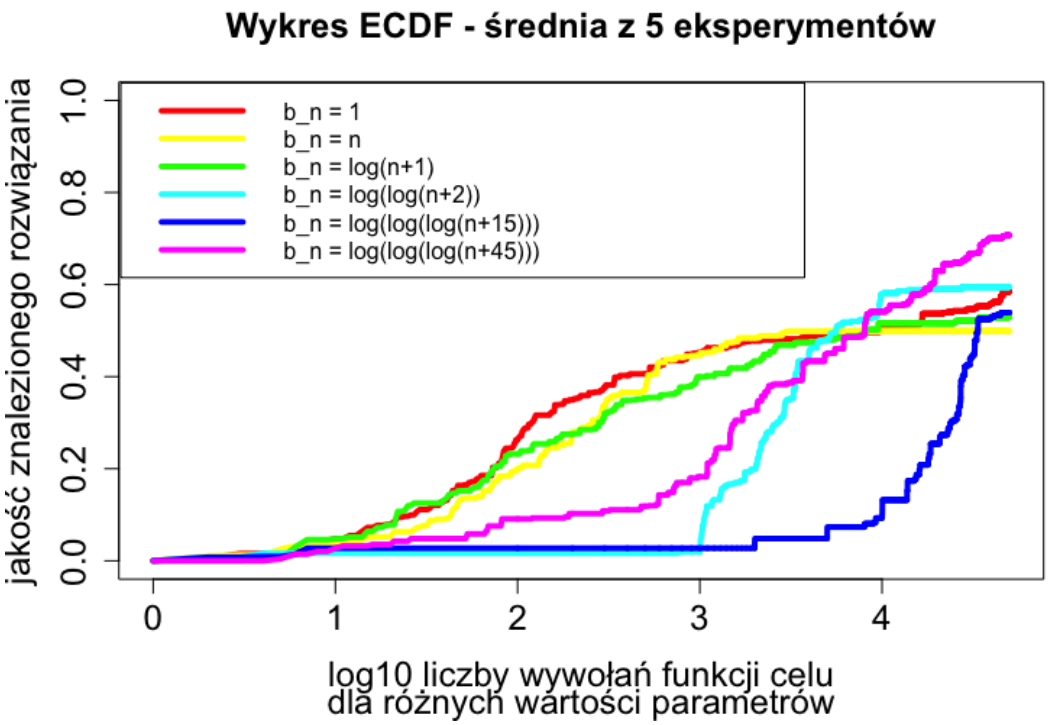
Na osi X zamieszczono logarytm liczby wywołań funkcji, który interpretowany jest jako czas działania algorytmu. Na osi Y zaznaczono część zdobytych tak zwanych "celi optymalizacyjnych". W naszym przypadku tymi celami są wartości funkcji celu przeskalowane do przedziału $[0, 1]$, gdzie 1 oznacza wartość największą (kandydata podejrzewanego o bycie maksimum, którą chcemy, żeby algorytm znalazł), a 0 wartość funkcji dla permutacji identycznościowej która jest pozycją startową dla algorytmów.

5.2 Testy dla różnych ciągów temperatur

Przeprowadzono testy dla kilku wybranych ciągów bet odpowiadających odwrotnościom temperatur:

- $b_n = 1$ (podstawowy Metropolis na funkcji $g(i)$),
- $b_n = n$,
- $b_n = \log(n+1)$,
- $b_n = \log(\log(n+2))$,
- $b_n = \log(\log(\log(n+15)))$,
- $b_n = \log(\log(\log(n+45)))$.

Testy zostały wykonane dla permutacji rozmiaru $p = 25$, liczby iteracji dla pojedynczej bety $= 1000$ oraz $n_max = 50$. Efekty są widoczne na poniższym wykresie ECDF. Dla każdego z ciągów bet uśredniono uzyskane w każdej iteracji wartości z 5 prób symulowanego wyżarzania.

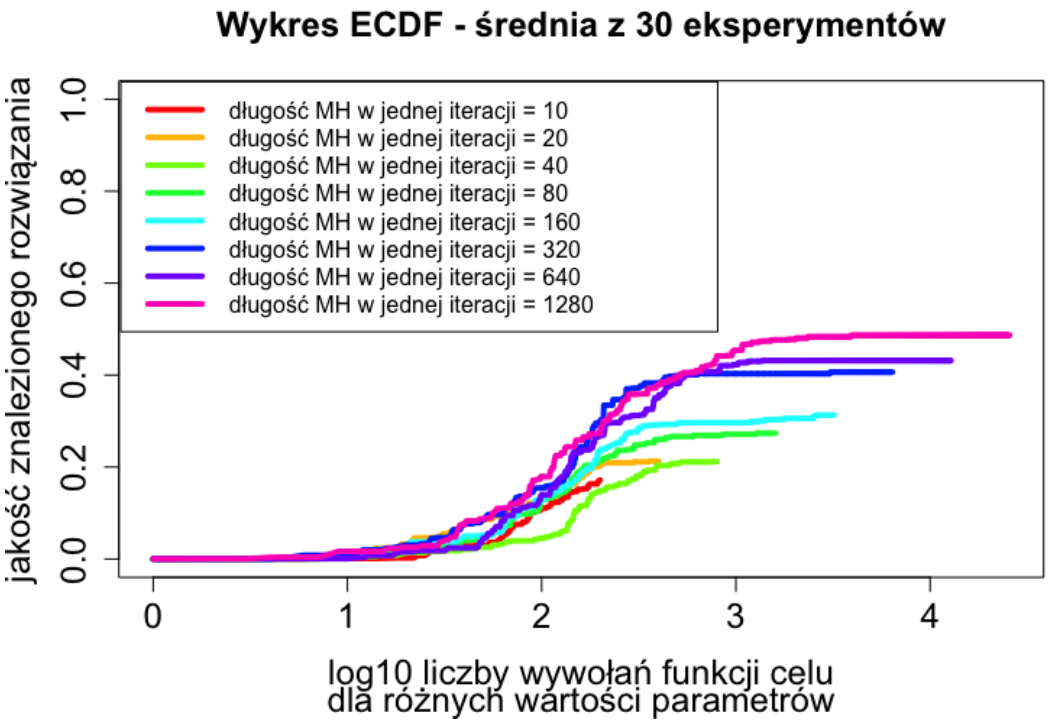


Jak widać na wykresie, przez długi czas działania symulowanego wyżarzania ciąg $b_n = \log(\log(\log(n+45))$ osiągał małe wartości funkcji celu w stosunku do trzech innych ciągów

temperatur, ale pod koniec uzyskał przewagę i to on zwrócił ostatecznie największą wartość funkcji celu. Ogólnie rzecz biorąc ten ciąg oraz ciąg $b_n = \log(\log(n+2))$ poradziły sobie najlepiej. Można zauważyć też, że ciąg $b_n = \log(\log(\log(n+15))$ nie poradził sobie zbyt dobrze, co sugeruje, że rozpoczęcie działania algorytmu dla niższej temperatury jest znacznie lepiej dopasowane do problemu.
Bardzo wolny spadek wartości temperatury i początkowa niska temperatura wydają się zatem najlepsze dla rozważanego problemu maksymalizacji.

5.3 Testy dla różnej liczby iteracji

Przeprowadzono analogiczne testy dla ustalonego ciągu bet: $b_n = \log(\log(n+2))$. Tym razem przetestowano różne liczby iteracji: $10, 20, 40, 80, 160, 320, 640, 1280$. Testy zostały wykonane dla permutacji rozmiaru $p = 10$ oraz $n_max = 20$. Wyniki przedstawia wykres 5.3.



Jak widać na wykresie, większe liczby iteracji przynoszą średnio lepsze efekty końcowe. Patrząc na różnice dla różnych liczb iteracji, można się jednak zastanawiać na jakim poziomie powinniśmy się zatrzymać. Największa różnica jest widoczna pomiędzy 160 a 320 iteracjami. Trzy największe liczby iteracji dają już mniejsze różnice w wynikach, aczkolwiek 1280 iteracji

wciąż daje istotnie najlepsze rezultaty.

6 Szczegóły techniczne rozwiązania

W trakcie projektu wykorzystano język R [5], środowisko RStudio [6] i następujące biblioteki:

- permutations [2],
- gips [3],
- ggplot2 [7],
- dplyr [8],
- tidyr [9].

Wszystkie wyprodukowane w ramach tego projektu skrypty języka R są publicznie dostępne pod adresem [4].

7 Podsumowanie

Rozważane algorytmy okazały się skuteczne zarówno w przypadku problemu poprawienia estymatora kowariancji jak i w przypadku maksymalizacji badanej funkcji celu.

Zastosowanie algorytmów BG i MH pozwoliło poprawić rozkład wartość własnych estymatorów macierzy kowariancji. Ponadto, testy wskazały, że poprawka za pomocą MH jest lepsza w sensie MSE.

W przypadku algorytmu SA zastosowanego do problemu maksymalizacji funkcji celu kluczowy okazał się dobór ciągu temperatur i liczby iteracji w pojedynczym kroku. Ciąg bet $b_n = \log(\log(\log(n + 45)))$ oraz duże liczby iteracji (320, 640, 1280) przyniosły dobre rezultaty dla $p = 25$ w zbliżeniu się do maksimum globalnego funkcji. Przy odpowiednio długim czasie działania algorytm mógłby osiągnąć dobre rezultaty również dla większych rozmiarów permutacji.

Dobranie odpowiednich wartości parametrów algorytmu Symulowanego Wyżarzania spowodowało poprawienie wyników osiąganych algorytmem Metropolisa-Hastingsa w stopniu znacznym.

Literatura

[1] Piotr Graczyk, Hideyuki Ishi, Bartosz Kołodziejek, and Hélène Massam. Model selection in the space of gaussian models invariant by symmetry. *arXiv*, 04 2020. `https://www.researchgate.net/publication/340500495_Model_selection_in_the_space_of_Gaussian_models_invariant_by_symmetry`.

[2] Robin K.S. Hankin. Introducing the permutations package. *SoftwareX*, 11, 2020. `https://CRAN.R-project.org/package=permutations`.

[3] Chojecki Przemysław. Strona GitHub z kodem pakietu 'gips'. `https://github.com/PrzeChoj/gips/`. [Online; accessed 05-06-2022].

[4] Chojecki Przemysław. Strona GitHub z kodem tego projektu. `https://github.com/PrzeChoj/SymulowaneWyzarzanie`. [Online; accessed 05-06-2022].

[5] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020.

[6] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, PBC., Boston, MA, 2020.

[7] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016.

[8] Hadley Wickham, Romain François, Lionel Henry, and Kirill Müller. *dplyr: A Grammar of Data Manipulation*, 2022. https://dplyr.tidyverse.org, https://github.com/tidyverse/dplyr.

[9] Hadley Wickham and Maximilian Girlich. *tidyr: Tidy Messy Data*, 2022. https://tidyr.tidyverse.org, https://github.com/tidyverse/tidyr.