Porovnání komerčního nástroje na seskupování smluv a nástroje vlastní výroby

David Žárský
(dzarsky@koop.cz)

ACTUARIA

Program

- Seskupování smluv – co to je, proč se provádí
- Vybrané základní přístupy
- Aktuálně využívaný komerční nástroj – dvě verze
- Vývoj vlastního nástroje
- Porovnání nástrojů
- Diskuse

Model peněžních toků pro životní smlouvy

- Vstupní data – údaje o pojistných smlouvách
  - Pojistná doba, věk, sjednaná připojištění, pojistné částky, …
- Předpoklady o budoucím vývoji
  - Ekonomické – diskontní křivka, inflace, …
  - Aktuárské – dekrementy (úmrtnost, stornovost), míra invalidnění, úrazovost, nemocnost, …
- Pro každý produkt je naprogramováno fungování jeho toků
  - Přijaté pojistné, vyplacené plnění, rezervy, investice, podíly na zisku, marketingové akce, slevy/přirážky, …
- Úkolem modelu je na základě našich předpokladů pro každou smlouvu vyprodukovat projekci jejích peněžních toků až do konce jejího trvání (typicky jde až o desítky let)
- Využíváme software Prophet (FIS)
  - Alternativy: MoSes, RAFM, AXIS, …
  - Prophet: jedna smlouva = jeden „modelpoint“

Co je seskupování smluv?

- Z výpočetního softwaru získáme pro každou životní smlouvu projekci vybraných toků na desítky let dopředu
- Cílem seskupení je následně vybrat malý počet smluv a naškálovat jejich toky tak, aby dohromady co nejlépe odpovídaly hodnotám za celé portfolio (škálovací koeficient je tzv. „váha“ dané smlouvy)
- Nejlépe bychom zároveň minimalizovali počet smluv i odchylky od původních hodnot, leč tyto metriky „jdou proti sobě“
- Důvod pro seskupování: Některé projekce toků je potřeba spočítat např. tisíckrát s různými výchozími předpoklady. Díky seskupení můžeme tyto výpočty provádět jen se zlomkem původního počtu smluv a výsledky přesto vycházejí velmi podobně, jako kdybychom použili všechny smlouvy.

Vybrané základní přístupy

- **M**: matice typu řádky = toky (65 * 40 let), sloupce = smlouvy
- **v**: toky za celé portfolio = součet sloupců M, tzv. referenční vektor
- **x**: vektor hledaných vah
- a) Minimalizace odchylky při maximálním povoleném počtu smluv *k*
  - Stará verze používaného komerčního nástroje

$$\min \left\{ ||M\mathbf{x} - \mathbf{v}||_2^2 : \mathbf{x} \geq \mathbf{0},\ ||\mathbf{x}||_0 \leq k \right\},$$

$$\text{kde } ||\mathbf{x}||_0 = \text{ počet nenulových složek } \mathbf{x}$$

- b) Minimalizace počtu smluv při povolené odchylce
  - Nová verze komerčního nástroje i nástroj vlastní výroby
  - Řešíme úlohu lineárního programování pro vhodné A, b, c
  - Nízký počet nenulových vah je žádoucím „vedlejším efektem“ minimalizace účelové funkce, samotné její minimum pro nás není důležité

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \leq b \\ & \mathbf{0} \leq \mathbf{x} \end{aligned}$$

Aktuálně používaný nástroj

- Starší verze (roky používaná) = „starý nástroj“:
  - Minimalizace odchylek při maximálním dovoleném počtu smluv
  - Víme pouze, že nástroj implementuje „Lawson & Hanson’s NNLS“ (Non-negative Least Squares) algoritmus, ale samotný kód nemáme k dispozici a nemůžeme ho měnit
  - Využíváme iterativně a napříč iteracemi hledáme nejlepší výsledek
  - Cíl: plně zautomatizovat + nejlépe zrychlit a zmenšit odchylky
- Nová verze (únor 2026) = „nový nástroj“:
  - Obdrželi jsme až po začátku vývoje vlastního nástroje
  - Řeší úlohu lineárního programování s předepsanými odchylkami pomocí GLOP solveru, počty smluv se nepředepisují
  - Plně automatizováno, doběhne za 5 min, mnohem kvalitnější odchylky
  - Lehce vyšší počet smluv, ale u důležitých produktů srovnatelný
  - Kód stále nemáme k dispozici a nemůžeme měnit samotné fungování
- Lze naprogramovat nástroj aspoň stejné kvality, nad kterým ale máme plnou kontrolu (= „vlastní nástroj“) ?

Lineární programování v pythonu

- Úloha lineárního programování:

$$\begin{aligned} \min \ & \mathbf{c}^T \mathbf{x} \\ & A_{\text{ub}} \mathbf{x} \leq \mathbf{b}_{\text{ub}} \\ & \mathbf{0} \leq \mathbf{x} \end{aligned}$$

- **x** = vektor hledaných vah
- from *scipy.optimize* import *linprog*
  - *weights* = *linprog*(c=**c**, A_ub = **A**, b_ub = **b**, ...)
  - Jak nastavit **A, b, c**?

linprog – volba **A, b**

$$\begin{aligned} \min\ & \mathbf{c}^T\mathbf{x} \\ & A\mathbf{x} \le \mathbf{b} \\ & \mathbf{0} \le \mathbf{x} \end{aligned}$$

- **M**: matice typu řádky = toky, sloupce = smlouvy
- **v**: součet sloupců M (toky za celé portfolio), tzv. referenční vektor
- **x**: hledané váhy
- **t**: vektor dovolených odchylek (tolerancí) – maximum z daného procenta absolutní hodnoty toku (*rel.tol*) a předem definované hodnoty v CZK (*abs.tol*)
- Chceme, aby po složkách (tocích) platilo:

$$\mathbf{t} = \max\{\text{rel.tol} * |\mathbf{v}|, \text{abs.tol}\}$$

$$|M\mathbf{x} - \mathbf{v}| \le \mathbf{t} \iff M\mathbf{x} \le \mathbf{v} + \mathbf{t} \ \wedge \ M\mathbf{x} \ge \mathbf{v} - \mathbf{t}$$

$$|M\mathbf{x} - \mathbf{v}| \le \mathbf{t} \iff M\mathbf{x} \le \mathbf{v} + \mathbf{t} \ \wedge \ -M\mathbf{x} \le -\mathbf{v} + \mathbf{t}$$

$$\begin{pmatrix} M \\ -M \end{pmatrix} \mathbf{x} \le \begin{pmatrix} \mathbf{t}+\mathbf{v} \\ \mathbf{t}-\mathbf{v} \end{pmatrix} \implies A = \begin{pmatrix} M \\ -M \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} \mathbf{t}+\mathbf{v} \\ \mathbf{t}-\mathbf{v} \end{pmatrix}$$

linprog – volba odchylek

$$\mathbf{t} = \max\{\text{rel.tol} * |\mathbf{v}|, \text{abs.tol}\}$$

Relativní odchylky (rel.tol)

- Toky ve **v** rozděleny dle stupňů důležitosti
- 1. stupeň: prvních 10 let 0,1 %, od 11. roku postupně roste na 2 %
- 2. stupeň: 2 % každý rok projekce
- 3. stupeň: 5 % každý rok projekce

Absolutní odchylky (abs.tol)

- Toky ve **v** rozděleny na čítací (např. počet smluv) a peněžní
- Čítací: 10 (každý rok projekce)
- Peněžní: 2000 CZK (každý rok projekce)

linprog – iterace a volba **c**

$$\mathbf{c} \in \mathbb{R}^{\text{vstupní smlouvy}} \longrightarrow \text{řešení } \mathbf{x}(\mathbf{c}) \text{ úlohy } \begin{array}{l} \min \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} \leq \mathbf{b} \\ \mathbf{0} \leq \mathbf{x} \end{array} \longrightarrow \text{počet nenulových vah (složek) } N(\mathbf{x}(\mathbf{c}))$$

$$\text{Hledáme } \mathbf{c}_{\min} = \arg\min \left\{ N(\mathbf{x}(\mathbf{c})) : \mathbf{c} \in \mathbb{R}^{\text{vstupní smlouvy}} \right\} \text{ vedoucí k minimálnímu počtu smluv po seskupení.}$$

- 1. iterace: **c** = (1, 1, … , 1)
  - Získáme nějaké váhy
- Další iterace:
  - Na základě získané informace se celé portfolio snažíme reprezentovat už jen smlouvami, jimž v předchozím kroku vyšla nenulová váha, přičemž penalizujeme smlouvy s nízkou vahou
  - Vektor **b** na pravé straně zůstává stejný (referenční vektor **v** se nemění, předepsané odchylky **t** také ne)
  - V matici **M** (resp. **A**) ponecháme pouze smlouvy (sloupce), jimž vyšly v předchozí iteraci nenulové váhy
  - Nové **c** = 1/nenulové váhy z předchozího kroku (díky této penalizaci by nízké váhy z předchozího kroku mohly v následující iteraci vyjít nulové)
  - Iterování končí, když už nová iterace nezmenší počet nenulových vah

linprog – příklad iterací

$A_{1,2,3}$:= matice tvořená sloupci $1, 2, 3$ z matice A

$\mathbf{x}_{1,2,3}$:= vektor tvořený složkami $1, 2, 3$ z vektoru $\mathbf{x}$

$\mathbf{b}_{1,2,3}$:= vektor vytvořený jako $\mathbf{b}$, ale pouze ze sloupců $1, 2, 3$ matice M

1. iterace

$$\min_{x_1,x_2,x_3,x_4} x_1 + x_2 + x_3 + x_4, \quad A\mathbf{x} \leq \mathbf{b}$$

$$\text{weights}_1 = (x_1, x_2, x_3, x_4) = (2.3, 0, 1.6, 0.1)$$

2. iterace

$$\min_{x_1,x_3,x_4} \frac{1}{2.3}x_1 + \frac{1}{1.6}x_3 + \frac{1}{0.1}x_4 = \min_{x_1,x_3,x_4} 0.435x_1 + 0.625x_3 + 10x_4, \quad A_{1,3,4}\mathbf{x}_{1,3,4} \leq \mathbf{b}$$

$$x_2 = 0, \text{weights}_2 = (x_1, x_3, x_4) = (2.15, 1.85, 0)$$

3. iterace

$$\min_{x_1,x_3} \frac{1}{2.15}x_1 + \frac{1}{1.85}x_3 = \min_{x_1,x_3} 0.465x_1 + 0.541x_3, \quad A_{1,3}\mathbf{x}_{1,3} \leq \mathbf{b}$$

$$x_2 = 0, x_4 = 0, \text{weights}_3 = (x_1, x_3) = (2.2, 1.8)$$

Výsledek

$$\text{weights}_{\text{final}} = (x_1, x_2, x_3, x_4) = (2.2, 0, 1.8, 0)$$

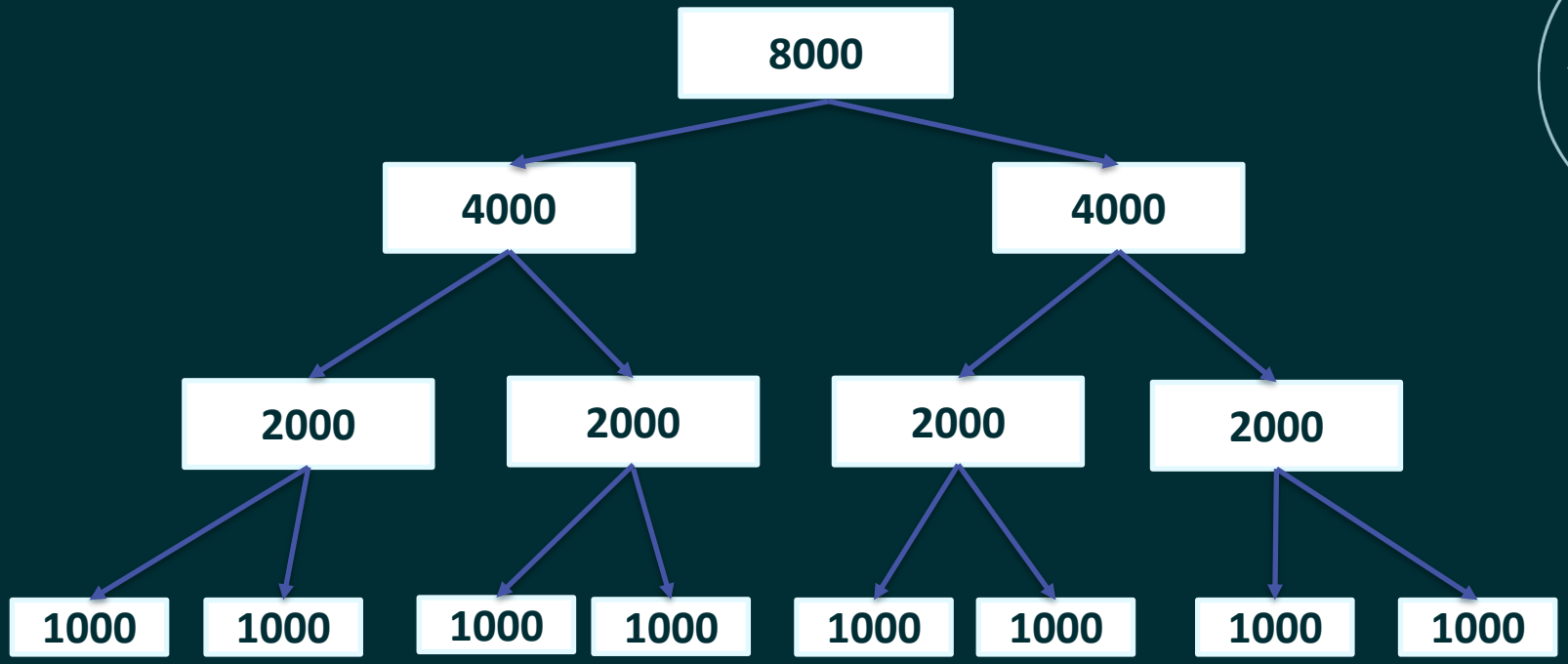
linprog – iterace v koncoroční Solvency II závěrce

Iterace	UNLI1	UNLI2	UNLI3	RSID	TERM1	TERM2	ACNT	VARI	ENDR1	ENDR2	ENDR3	ENDS1	ENDS2
0	4 685	37 956	46 743	15 688	132 009	35 458	9 480	8 677	3 491	141	12	4	213
1	290	507	458	419	365	368	318	368	284	56	9	2	85
2	202	386	365	326	269	268	213	261	236	47	8		80
3	183	359	347	299	260	255	201	247	224	44			
4	179	354	335	292	255	249	197	243	220	43			
5	177	351	329	291	253	246	196	241	218				
6	176	350	326	287	250	245	195						
7		349			249	244	194						
8		347				243							
9						242							
10						241							
11						239							

linprog – kdy je hotovo?

- Pokud jsme daný produkt schopni takto rozumně rychle seskupit, pak to může být konec algoritmu
- „Upočitatelnost“ je určena především vstupním počtem smluv, jejich rozmanitostí, předepsanými odchylkami a dostupným výpočetním výkonem
- I když tímto postupem získáme nějaký výsledek, je počáteční volba **c** = (1, 1, …, 1) pro minimalizaci počtu smluv opravdu ta nejlepší možná? Nešlo by jinou volbou získat méně smluv?
- Idea: **rozděl a panuj**
- Smlouvy rozdělíme na dostatečně malé kusy, jejichž referenční vektor bude součet toků za všechny smlouvy v daném kusu (kus „reprezentuje sám sebe“). Kusy seskupíme dle předchozího a výsledky obsahující značné množství nul následně použijeme jako výchozí pozici pro seskupení většího kusu (dostáváme pro něj startovní **c**, které už obsahuje značné množství nul).

Rekurzivní seskupování



Rekurzivní seskupování

- Na nejnižší úrovni proběhne postup zcela stejně jako předtím
- Na o jedna vyšší úrovni se slepí příslušné dva výsledky z nižší úrovně a vezmou se jako výsledek 1. iterace, potom už na této úrovni seskupení pokračuje stejně jako dříve. Díky nulovým vahám z nižší úrovně jsou další iterace rychle spočitatelné.

$$\min \mathbf{c}^T\mathbf{x} = \min c_1x_1+c_2x_2+c_3x_3+c_4x_4+c_5x_5+c_6x_6+c_7x_7+c_8x_8, \quad A\mathbf{x} \leq \mathbf{b}$$

$$\begin{array}{l|l} \min\ x_1+x_2+x_3+x_4, \quad A_{1,2,3,4}\mathbf{x}_{1,2,3,4} \leq \mathbf{b}_{1,2,3,4} & \min\ x_5+x_6+x_7+x_8, \quad A_{5,6,7,8}\mathbf{x}_{5,6,7,8} \leq \mathbf{b}_{5,6,7,8} \\ \downarrow & \downarrow \\ w_{\text{first}} = (x_1,x_2,x_3,x_4) = (0,2,2,0) & w_{\text{second}} = (x_5,x_6,x_7,x_8) = (1,3,0,0) \end{array}$$

$$\text{weights}_1 = (x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8) = (0,2,2,0,1,3,0,0)$$

$$\mathbf{c} = \left(0,\frac{1}{2},\frac{1}{2},0,1,\frac{1}{3},0,0\right)$$

$$\min \mathbf{c}^T\mathbf{x} = \min \frac{1}{2}x_2+\frac{1}{2}x_3+x_5+\frac{1}{3}x_6, \quad A_{2,3,5,6}\mathbf{x}_{2,3,5,6} \leq \mathbf{b}$$

Selhání linprogu

- Funkce *linprog* pro každý kus počítá jistou optimalizační úlohu, kterou nemusí vždy vyřešit nebo ji může řešit neúměrně dlouho
- Na každý kus má zadáno nejvyšší počet iterací, jenž může provést
- Selže-li, rozvolníme relativní odchylky u toků 2. a 3. stupně o 0,5 p.b. a zkusíme daný kus seskupit znovu
- Pokud se s nějakou povolenou relativní odchylkou dostaneme nad předepsanou maximální toleranci a stále jsme nenašli řešení, úlohu vzdáváme a do vyšší úrovně jde stejný vektor vah jako na začátku – na nižších úrovních se může stát, zvlášť při přísných odchylkách

Dělení na kusy

- Díky rekurzivní metodě lze seskupit i to, co by jinak trvalo dlouhé hodiny
- U výpočetně jednoduchých produktů však může naopak čas výpočtu prodloužit
- Zvolené rozdělení na kusy významně ovlivňuje výsledný počet nenulových vah (i délku výpočtu)
- „Sekvenční“ rozdělení (do prvního kusu jde prvních např. 1000 smluv, do druhého dalších 1000 smluv atd.) nedává stabilně dobré výsledky (na konci často vychází příliš mnoho smluv)
- Co kdybychom při rekurzi smlouvy na poloviny dělili náhodně, spočítali seskupení vícekrát a vzali nejlepší výsledek?
    - Vede již k relativně nízkému počtu nenulových vah, ale stále spoléhá až příliš na štěstí a při „smolném“ výpočtu můžeme u nějakého produktu i po hodinách iterování místo obvyklých 300 smluv stále skončit například na 1200

Jak (ne)zlepšit náhodné dělení na kusy

- Poznat a ukončit špatné iterace ještě během jejich výpočtu
  - Např. jakmile během seskupování řešíme kus s více než x nenulovými vahami, iteraci vzdáváme a raději začneme novou
  - Umožňuje vyšší počet iterací za stejný čas, ale neřeší efekt náhody
- Využít minulé výpočty
  - Např. na konci roku spustit velké množství iterací, získat kvalitní seskupení, zapamatovat si rozdělení na kusy a toto následně dál využívat. Z kvartálu na kvartál většina smluv v portfoliu zůstane, ty nové buď vyřešit odděleně, nebo náhodně rozhodit mezi kusy tvořené starými smlouvami.
  - Nefunguje, algoritmus je velmi citlivý i na drobné úpravy kusů v rámci téhož výpočtu
  - Navíc je potřeba takto „kalibrovat“ pro každý typ seskupení

Jak (ne)zlepšit náhodné dělení na kusy

- Nejprve co nejlépe seskupit každý produkt, následně využít získané váhy jako počáteční váhy pro seskupení celého portfolia najednou a doufat, že se tím příliš nezhorší kvalita již dosaženého seskupení jednotlivých produktů
  - Zhorší. Získáme sice velmi nízký počet smluv a kvalitně seskupený celek, ale u jednotlivých produktů jsou odchylky nezřídka v řádech desítek až stovek procent.
- Zcela náhodným dělením smluv na kusy tedy můžeme obdržet velmi kvalitní výsledek, avšak jeho získání často vyžaduje hodiny iterování a příliš závisí na štěstí
- Výsledky je potřeba stabilizovat a smlouvy rozdělovat systematičtěji

Clustering dle toků – základní idea

smlouvy : $\mathcal{M} = \{\mathbf{m}_1, \mathbf{m}_2, ..., \mathbf{m}_n\}$

počet složek : $d = 2600$ (65 toků · 40 let)

počet kusů : $k = 2^l$

Hledáme množiny smluv $C_1, C_2, ..., C_k$ splňující :

$$\text{argmin} \left\{ \sum_{i=1}^{k} \sum_{\mathbf{m} \in C_i} ||\mathbf{m} - \boldsymbol{\mu}_i||^2 : \{C_1, C_2, ..., C_k\} \subseteq 2^{\mathcal{M}} \right\},$$

kde $\boldsymbol{\mu}_i$ je tzv. *centroid (mean)* clusteru C_i :

$$\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{m} \in C_i} \mathbf{m}$$

- from *sklearn.cluster* import *KMeans*
- Přítomnost náhody zůstává skrz parametr *random_state* (algoritmus potřebuje někde začít, často tak, že náhodně vybere centroidy)
- Doběhne v řádu sekund, ale velikost clusterů je velmi různorodá (některé obsahují 30 000 smluv, jiné jen 2 smlouvy) – stále může být neupočitatelné v rozumném čase

Vylepšení clusteringu

- from *sklearn.preprocessing* import *StandardScaler*
  - Předpokládáme, že jednotlivé toky napříč smlouvami mají normální rozdělení, a znormalizujeme je tak, aby šlo o N(0,1)
  - Všechny toky jsou pak souměřitelné (clustering minimalizuje odchylky od centroidu, bez přeškálování dávají peněžní toky v miliardách a počty v tisících řádově jiné odchylky)
- from *k_means_constrained* import *KMeansConstrained*
  - Parametry *min_size* a *max_size* vyrovnají velikosti clusterů, lze zvolit např. jako počet smluv/počet clusterů -+ 25%
  - Při *d = 2600* nepoužitelné (běží hodiny) – potřeba redukce parametrů
  - from *sklearn.decomposition* import *PCA* (principal component analysis = analýza hlavních komponent) – technika snížení dimenze s co nejmenší ztrátou informace
    - Uživatel volí dimenzi po redukci, v algoritmu voleno d = 100
  - Malé produkty (jednotky tisíc smluv) doběhnou do jedné minuty, střední produkty (nižší desítky tisíc smluv) trvají několik minut, velké produkty desítky minut až hodiny

Vylepšení clusteringu

- Malé/střední produkty – *KMeansConstrained*
- Velké produkty – *KMeans* + „balancing“
  - Nejprve v řádu sekund získáme clustering pomocí *KMeans*
  - Následně smlouvy mezi clustery přeskupíme tak, aby žádný cluster nebyl příliš velký:
    - Vybereme nějaký „přeplněný“ (> *max_size*) cluster a v něm identifikujeme bod nejdále od centroidu
    - Najdeme takový „nedoplněný“ (< *min_size*) cluster, k jehož centroidu má tento bod nejblíž, a přesuneme ho tam (i když je vzdálenost od nového centroidu větší než od původního).
    - Končíme tehdy, když jsou velikosti všech clusterů menší než *max_size*. Může se stát, že některé clustery jsou i potom stále menší než *min_size*, ale to pro upočitatelnost není problém.
    - Ne tak kvalitní clustering jako *KMeansConstrained*, ale celý proces je do minuty hotový i na největším produktu (cca 130 000 smluv)

Celý algoritmus pro jeden produkt

- Načteme data (např. textový soubor vyprodukovaný Prophetem)
- Lze-li rychle spočítat, nejprve seskupíme bez dělení na kusy
- Volitelné: Je-li počet smluv už dostatečně malý (předepsáno), končíme
- Dále opakujeme následující kroky, dokud nevyprší předepsaný čas:
  - Smlouvy rozdělíme předepsanou metodou na předepsaný počet clusterů: náhodně, *KMeansConstrained* nebo *KMeans* (+ balancing)
  - Seskupíme pomocí rekurzivního seskupování
    - Číslo clusteru v binární soustavě udává cestu smlouvy rekurzí: 0 = vlevo, 1 = vpravo
  - Volitelné: Je-li počet smluv už dostatečně malý (předepsáno), končíme
  - Zapamatujeme si výsledek a vrátíme se na začátek smyčky, kde clustering provedeme s jiným *random_state* (jiné centroidy)
- Až vyprší čas, vezmeme nejlepší výsledek z hlediska počtu smluv
- Vytvoříme různé výstupy
  - Váhy, matice odchylek, clustering vedoucí k výsledku, seskupené modelpointy pro Prophet a další

Seskupení pro koncoroční Solvency II závěrku

- Spočtené varianty popsaného algoritmu:
  - Bez dělení na kusy
  - Náhodné dělení na kusy - 8 h
  - Nejprve bez dělení na kusy, poté dělení náhodně 8 h
  - Clustering – rychlá varianta (jakmile algoritmus najde „dostatečně dobré“ seskupení, končí)
  - Clustering – 4 h
  - Clustering – 4 h bez omezení na velikost clusterů
  - Clustering – 8 h
  - Nejprve bez dělení na kusy, poté dělení clusteringem 8 h
  - Clustering – přísné odchylky 8 h (relativní odchylka 0,1 % pro toky 1. stupně, 1 % pro ostatní toky, absolutní odchylka zachovaná)

Koncoroční Solvency II závěrka – počty smluv

- **Silná výhoda** navrhovaného algoritmu – jako jediný výrazně redukuje počet smluv UNLI, což např. ve variantě „clustering – 4h“ vede k rychlejším stochastickým běhům (-20 %) a menší náročnosti na RAM (-17 %) oproti oběma verzím současného nástroje

Produkt	Počet smluv	Nástroj starý	Náhodně 8h	Clustery přísné 8h	Bez dělení	Clustery rychle	Clustery 4h	Clustery 4h bez omezení	Clustery 8h	Bez dělení + clustery 8h	Bez dělení + náhodně 8h
UNLI1	4 685	220	156	177	176	191	180	163	173	173	156
UNLI2	37 956	500	1193	350	347	336	320	327	320	320	347
UNLI3	46 743	450	291	425	326	326	298	298	294	294	291
RSID	15 688	225	264	266	287	300	246	283	246	246	264
TERM1	132 009	220	197	233	249	252	242	247	242	242	197
TERM2	35 458	220	193	258	239	231	230	191	201	201	193
ACNT	9 480	150	177	176	194	192	181	183	181	181	177
VARI	8 677	150	219	256	241	234	225	224	225	225	219
ENDR1	3 491	200	184	237	218	212	189	194	187	187	184
ENDR2	141	50	43	43	43	43	43	43	43	43	43
ENDR3	12	12	8	8	8	8	8	8	8	8	8
ENDS1	4	4	2	2	2	2	2	2	2	2	2
ENDS2	213	27	80	80	80	80	80	80	80	80	80
TOTAL	294 557	2 428	3 007	2 511	2 410	2 407	2 244	2 243	2 202	2 202	2 161
Čas běhu			503 min	503 min	66 min	25 min	262 min	504 min	503 min	569 min	569 min

Vyhodnocení variant – bez dělení na kusy

- Vykazuje stabilní chování z hlediska počtu smluv napříč výpočty, čas běhu je však i bez náhody v kódu poměrně variabilní (z poněkud záhadných příčin) a přímo se odvíjí od největšího produktu (TERM1)
- Lze běhat bez paralelizace (multiprocessingu) za zhruba +20 min
- Suverénně nejméně náročné na délku kódu – **40 řádků** + práce s daty (načtení vstupních hodnot, vytvoření **A** a **b**, export vah)
- I tato jednoduchá varianta výrazně redukuje počet smluv UNLI

Produkt	Starý nástroj	2412	2503	2506	2509	2512	2603
UNLI1	220	193	197	194	190	176	190
UNLI2	500	353	352	340	345	347	346
UNLI3	450	336	335	337	333	326	331
RSID	225	297	293	304	286	287	298
TERM1	220	238	249	256	261	249	256
TERM2	220	227	222	247	238	239	239
ACNT	150	210	202	203	200	194	197
VARI	150	243	235	245	245	241	236
ENDR1	200	231	237	239	225	218	228
ENDR2	50	45	44	44	43	43	41
ENDR3	12	10	9	9	9	8	9
ENDS1	4	7	6	4	4	2	0
ENDS2	27	79	76	76	77	80	76
TOTAL	2 428	2 469	2 457	2 498	2 456	2 410	2 447
Čas běhu		81 min	26 min	31 min	33 min	66 min	66 min

Vyhodnocení variant – rychlý clustering

- Rovněž stabilní chování počtu smluv
- Čas běhu se odvíjí od toho, jaké štěstí máme při náhodném výběru centroidů během tvorby clusterů
- Je potřeba stanovit, co je „dostatečně dobré“ seskupení – např. jednou seběhnout velký počet iterací a podívat se na to, jaký počet smluv je rozumně dosažitelný za krátký čas (vizte dále)
- Lze běhat i bez paralelizace, čas běhu je pak většinou dvoj- až trojnásobný

Produkt	Starý nástroj	2412	2503	2506	2509	2512	2603
UNLI1	220	187	187	189	186	191	188
UNLI2	500	324	327	324	323	336	329
UNLI3	450	318	304	309	302	326	307
RSID	225	281	280	275	290	300	297
TERM1	220	237	243	236	245	252	259
TERM2	220	202	233	235	232	231	213
ACNT	150	198	197	199	197	192	193
VARI	150	235	234	233	237	234	239
ENDR1	200	224	223	221	228	212	216
ENDR2	50	45	44	44	43	43	41
ENDR3	12	10	9	9	9	8	9
ENDS1	4	7	6	4	4	2	0
ENDS2	27	79	76	76	77	80	76
TOTAL	2 428	2 347	2 363	2 354	2 373	2 407	2 367
Čas běhu		94 min	23 min	23 min	83 min	25 min	70 min

Vyhodnocení variant – clustering 4 h

- Konzistentní počty smluv
- Čas běhu fixně daný
- Je potřeba běhat paralelně (jinak *čas = počet produktů * 4 h*)
- Zdá se být nejlepší volbou, máme-li k dispozici dostatečný výkon a čas

Produkt	Starý nástroj	2412	2503	2506	2509	2512	2603
UNLI1	220	176	176	176	172	180	166
UNLI2	500	313	320	314	322	320	316
UNLI3	450	299	296	290	299	298	305
RSID	225	265	272	273	274	246	269
TERM1	220	237	213	236	237	242	215
TERM2	220	202	219	201	206	230	213
ACNT	150	187	174	183	191	181	183
VARI	150	220	221	233	231	225	219
ENDR1	200	216	216	210	209	189	189
ENDR2	50	45	44	44	43	43	41
ENDR3	12	10	9	9	9	8	9
ENDS1	4	7	6	4	4	2	0
ENDS2	27	79	76	76	77	80	76
TOTAL	2 428	2 256	2 242	2 249	2 274	2 244	2 201

Váhy varianty „clustering - 8 h“

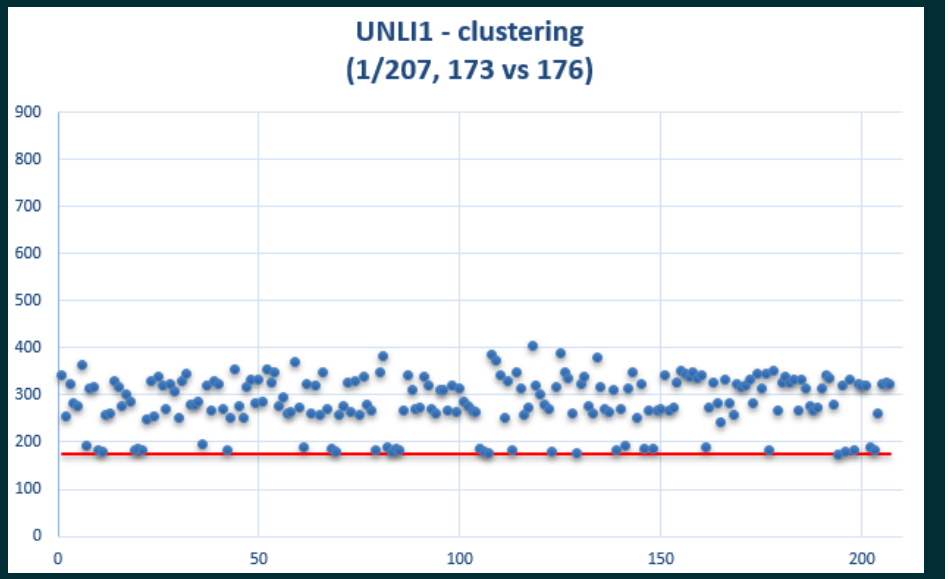
- Víme, že váhy x = (1, 1, …, 1) vždy dokonale splňují požadavek na odchylky, naštěstí však neminimalizují účelovou funkci (jejich součet, stejný jako počet smluv, je příliš velký)
- Níže jsou uvedeny statistiky o dosažených vahách z clusteringu počítaného 8 h
- Součet získaných vah je vždy menší než počet smluv na vstupu, nejmenší váha je většinou mezi 0 a 1, ale ne až tak blízká nule

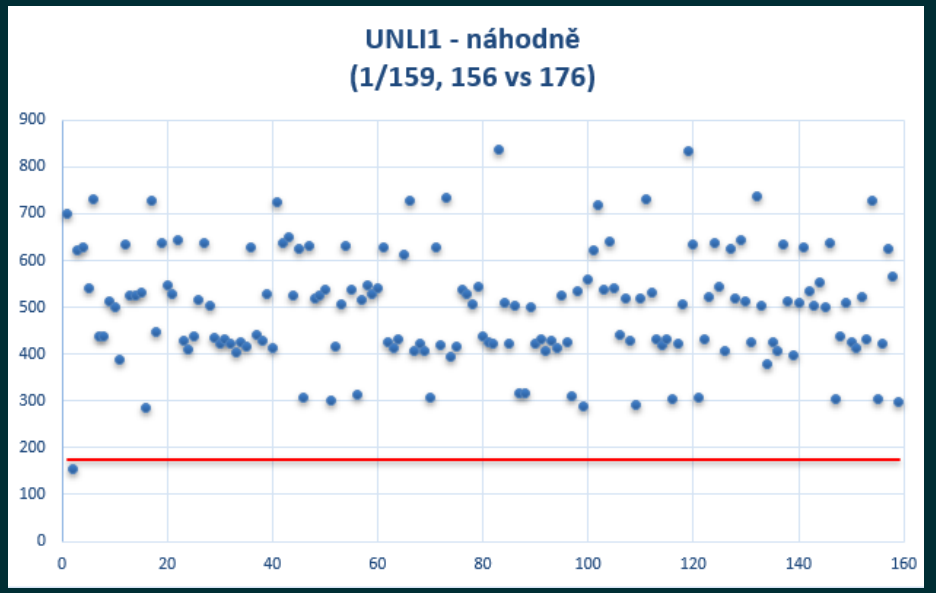
Produkt	Počet smluv	Nenulové váhy	Součet	Min	Průměr	Max
UNLI1	4 685	180	4 680.43	0.27	27.05	209.72
UNLI2	37 956	320	37 563.33	0.47	117.39	874.21
UNLI3	46 743	298	45 617.41	0.35	155.16	1 090.35
RSID	15 688	246	15 364.04	0.94	62.46	476.45
TERM1	132 009	242	131 512.70	2.38	543.44	5 561.43
TERM2	35 458	230	31 222.68	0.73	155.34	2 474.50
ACNT	9 480	181	9 448.94	0.25	52.20	338.15
VARI	8 677	225	8 582.67	0.53	38.15	187.54
ENDR1	3 491	189	3 311.33	0.16	17.71	174.64
ENDR2	141	43	127.87	0.20	2.97	10.61
ENDR3	12	8	9.68	0.64	1.21	2.11
ENDS1	4	2	3.19	1.38	1.60	1.82
ENDS2	213	80	208.86	0.06	2.61	15.99

Počty smluv v jednotlivých iteracích – clustering vs náhodně

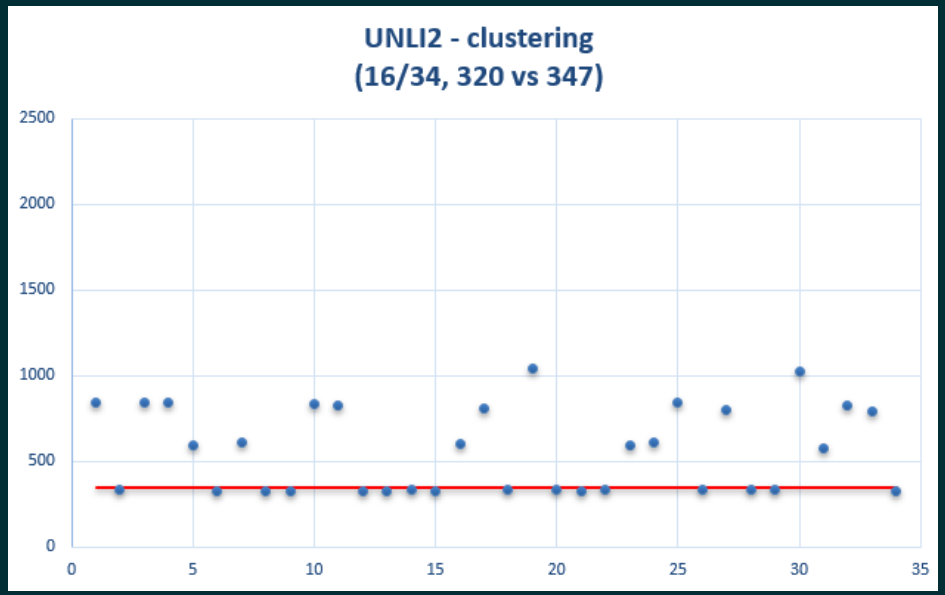
- Viděli jsme, že i když jsme schopni seskupit produkty bez dělení na kusy, můžeme skrze dělení dosáhnout lepšího výsledku
- Jak často se nám při dělení na kusy clusteringem nebo zcela náhodně podaří dosáhnout lepšího výsledku než bez dělení?
- Odpověď se různí od „skoro vždy“ po „skoro nikdy“ v závislosti na produktu
- Následující slidy ukazují počty smluv (osa Y) v jednotlivých iteracích (číslo iterace udává osa X) pro vybrané produkty. Červená čára je výsledek dosažený bez dělení na kusy.
- Je dobré mít na paměti, že uvedené chování se dle vstupních dat může mírně lišit

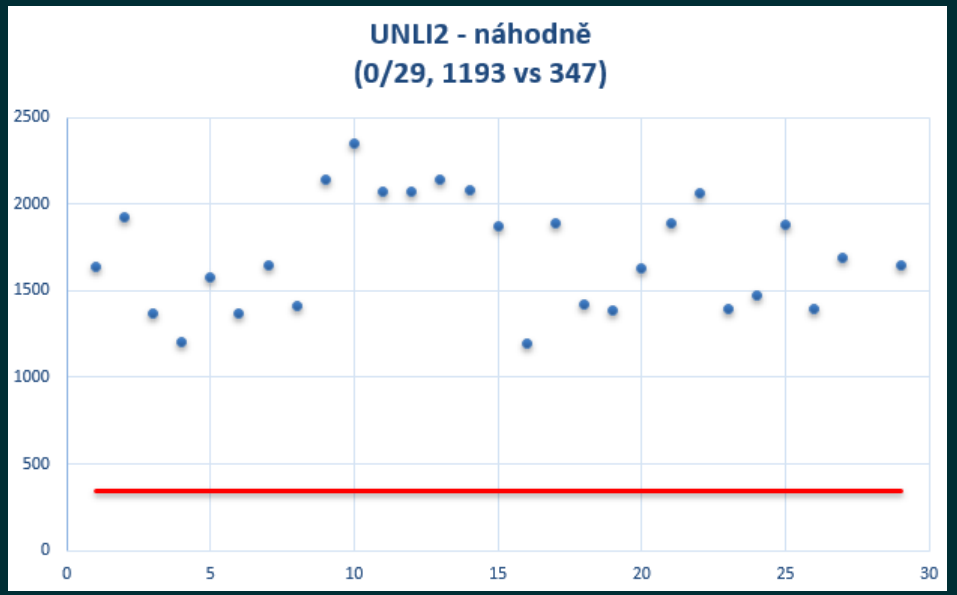
Počty smluv v jednotlivých iteracích – UNLI1



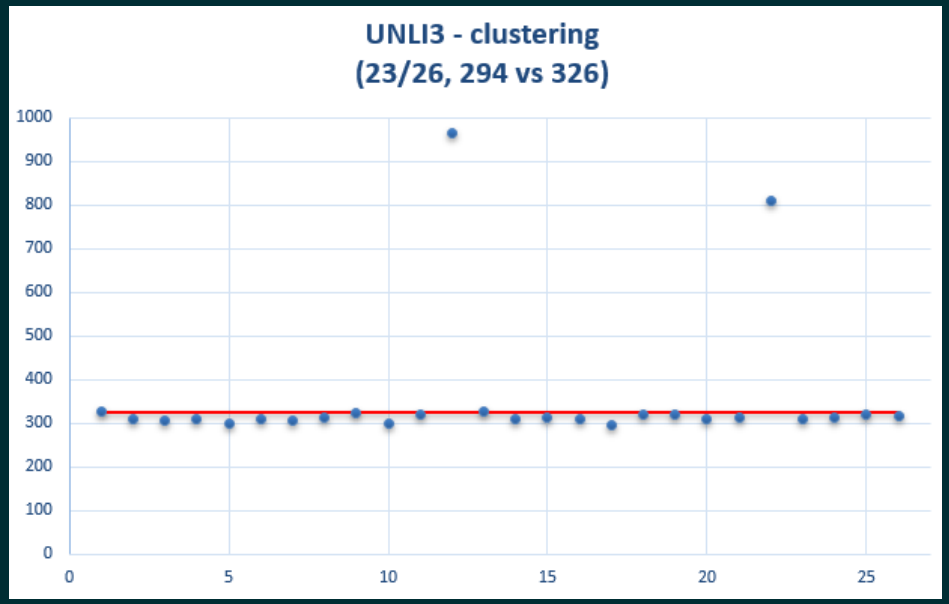


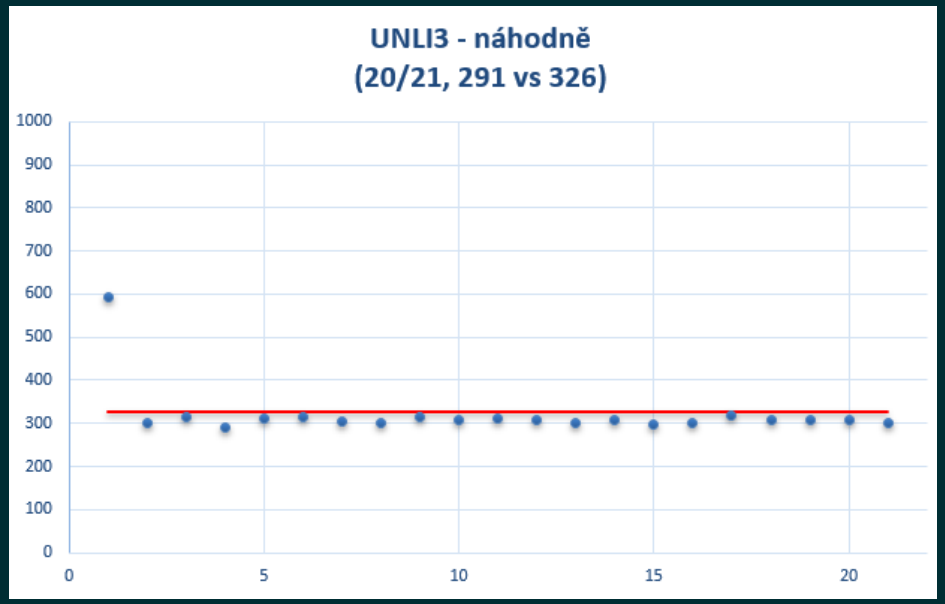
Počty smluv v jednotlivých iteracích – UNLI2



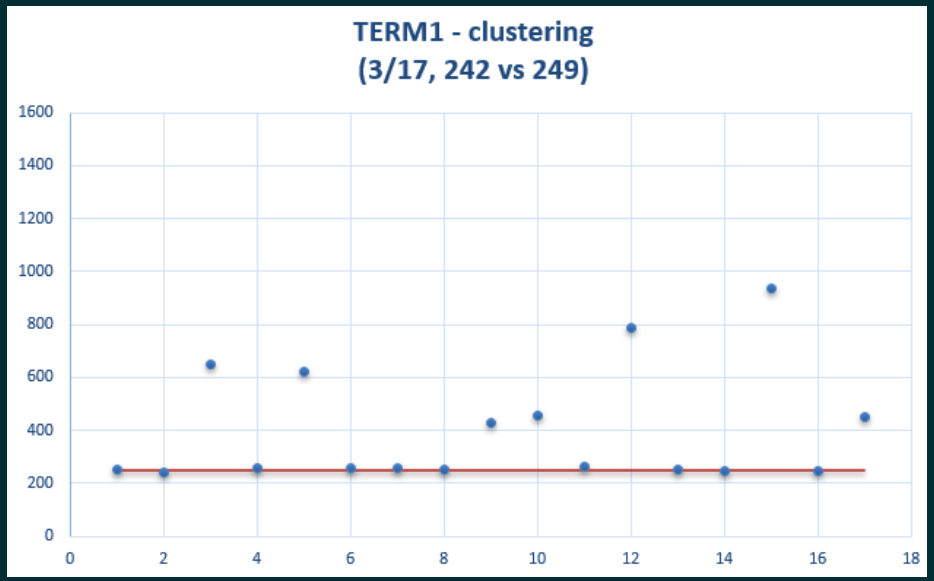


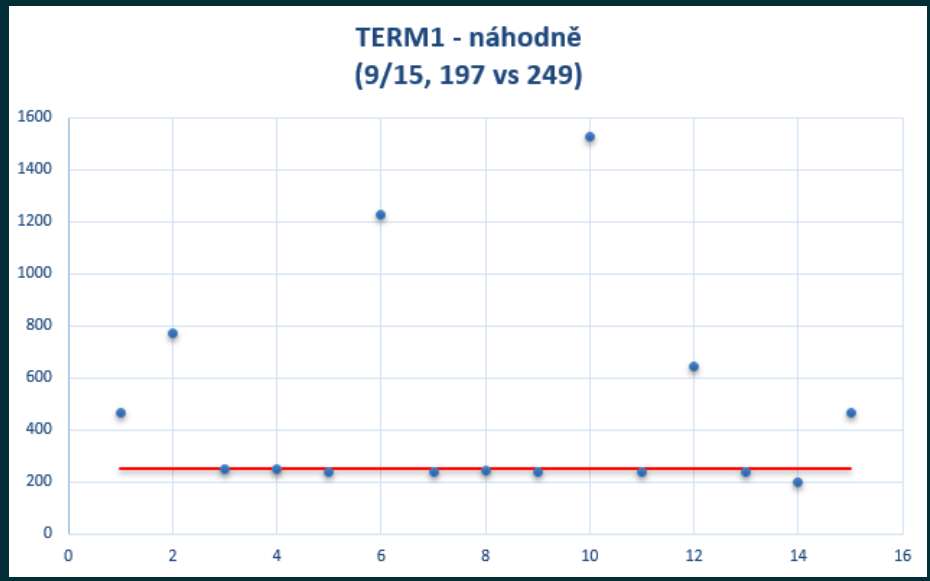
Počty smluv v jednotlivých iteracích – UNLI3



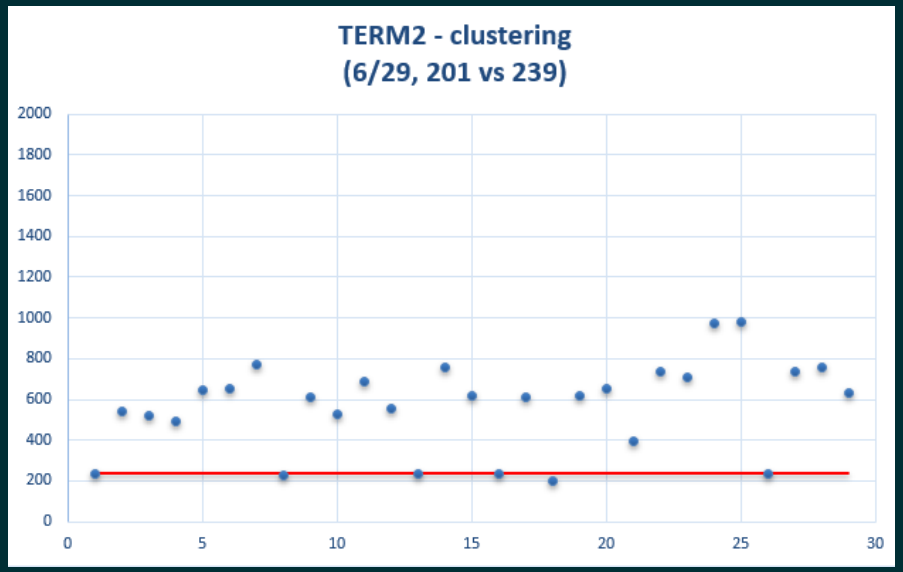


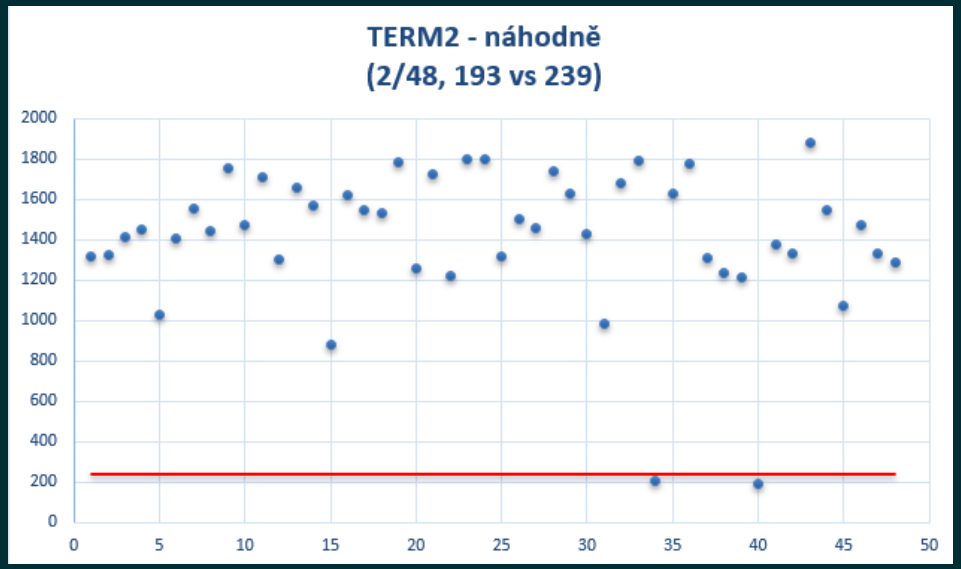
Počty smluv v jednotlivých iteracích – TERM1



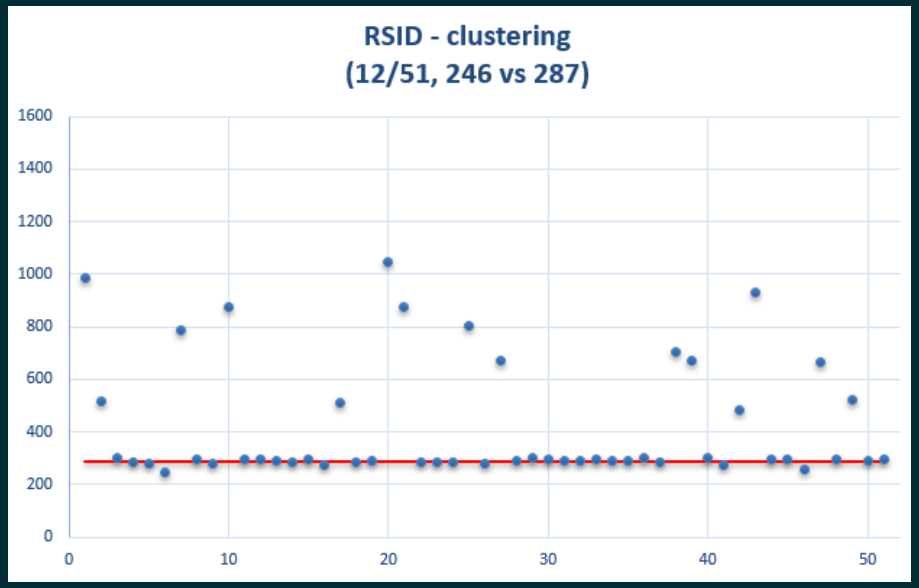


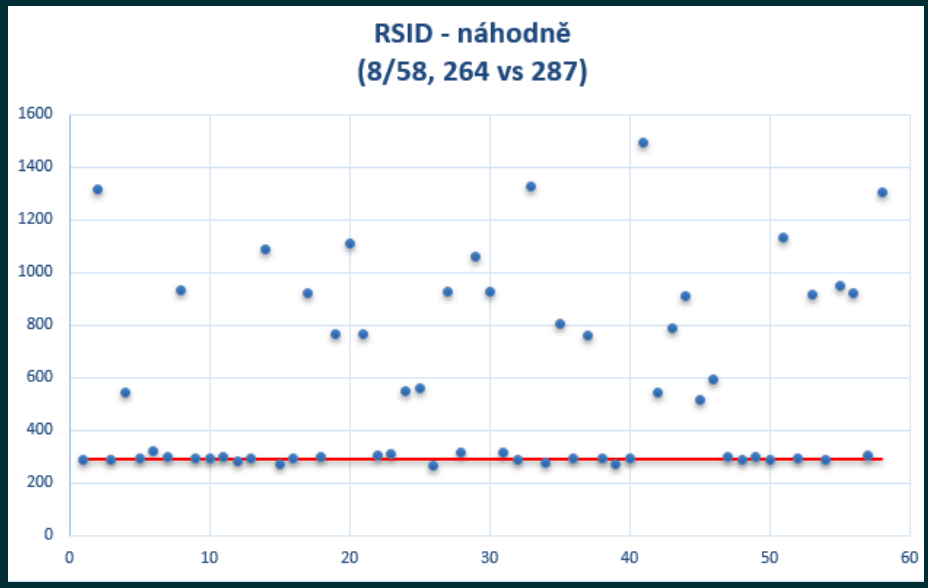
Počty smluv v jednotlivých iteracích – TERM2



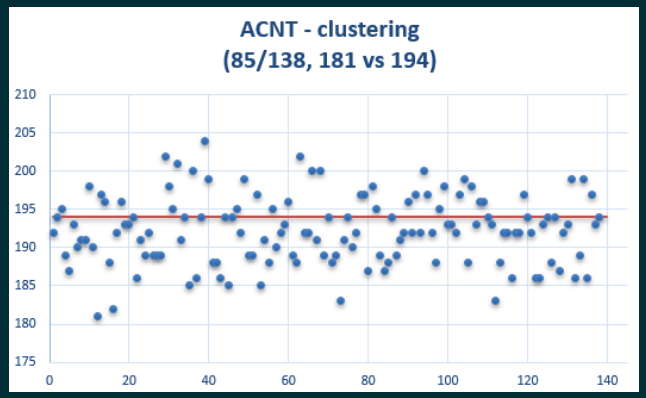


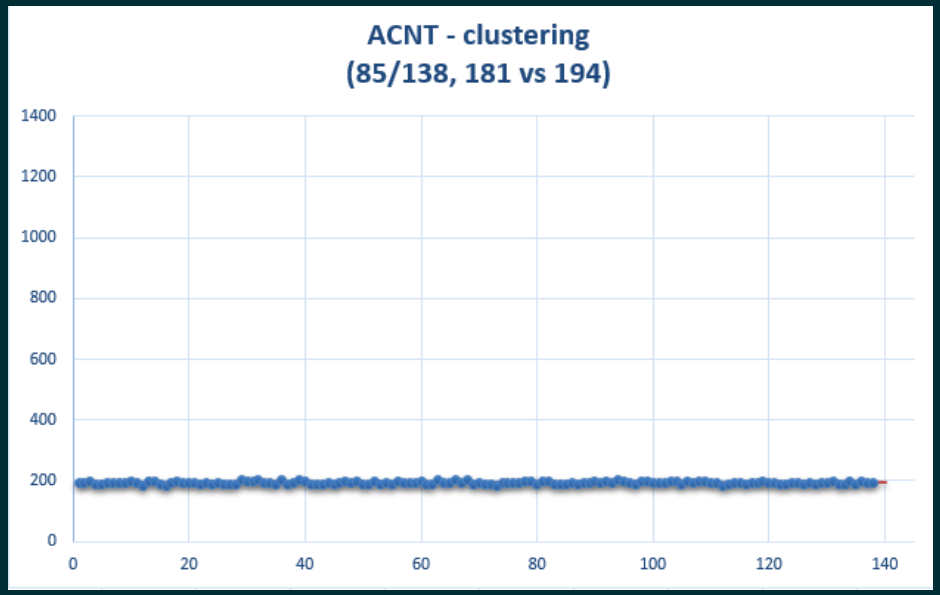
Počty smluv v jednotlivých iteracích – RSID

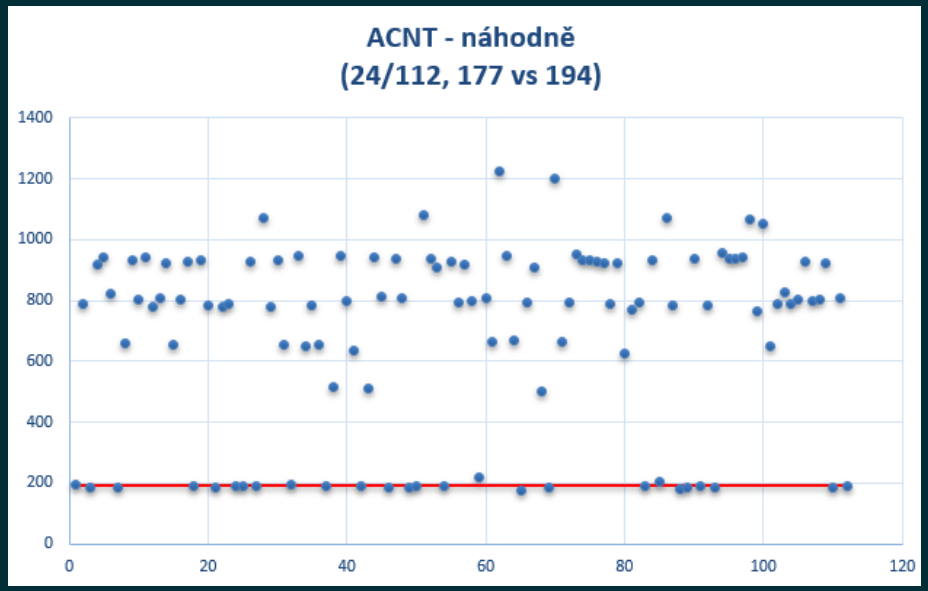




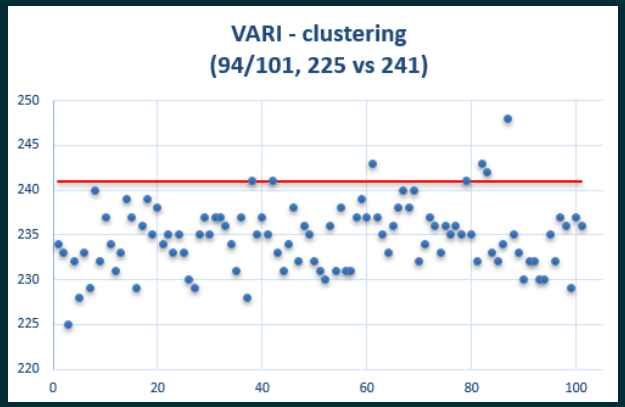
Počty smluv v jednotlivých iteracích – ACNT

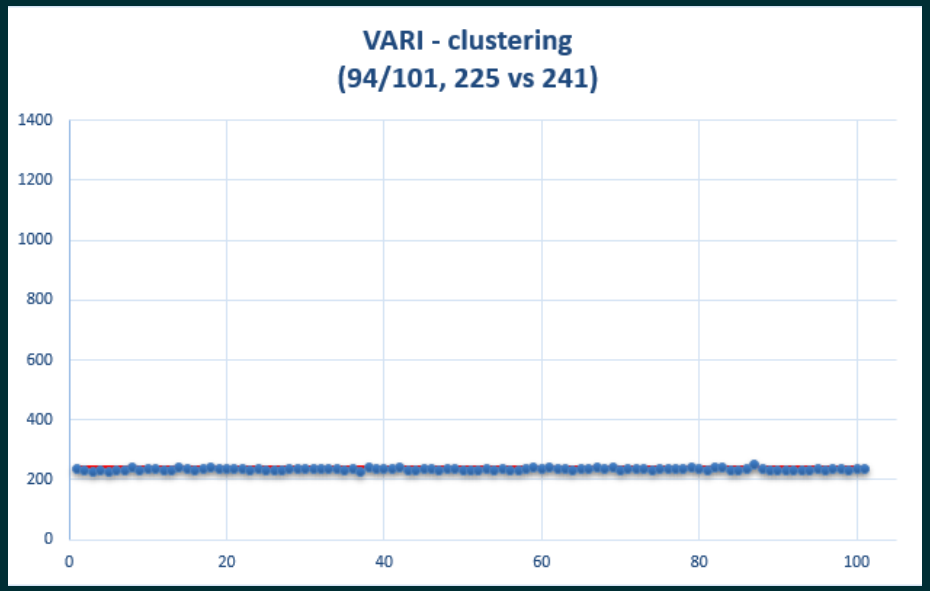


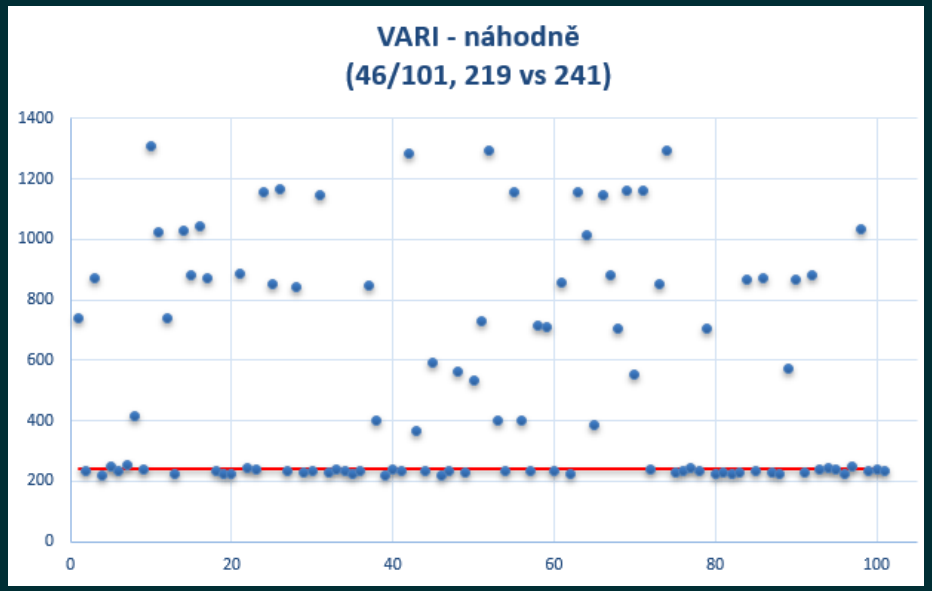




Počty smluv v jednotlivých iteracích – VARI







Počty smluv napříč nástroji a výpočty

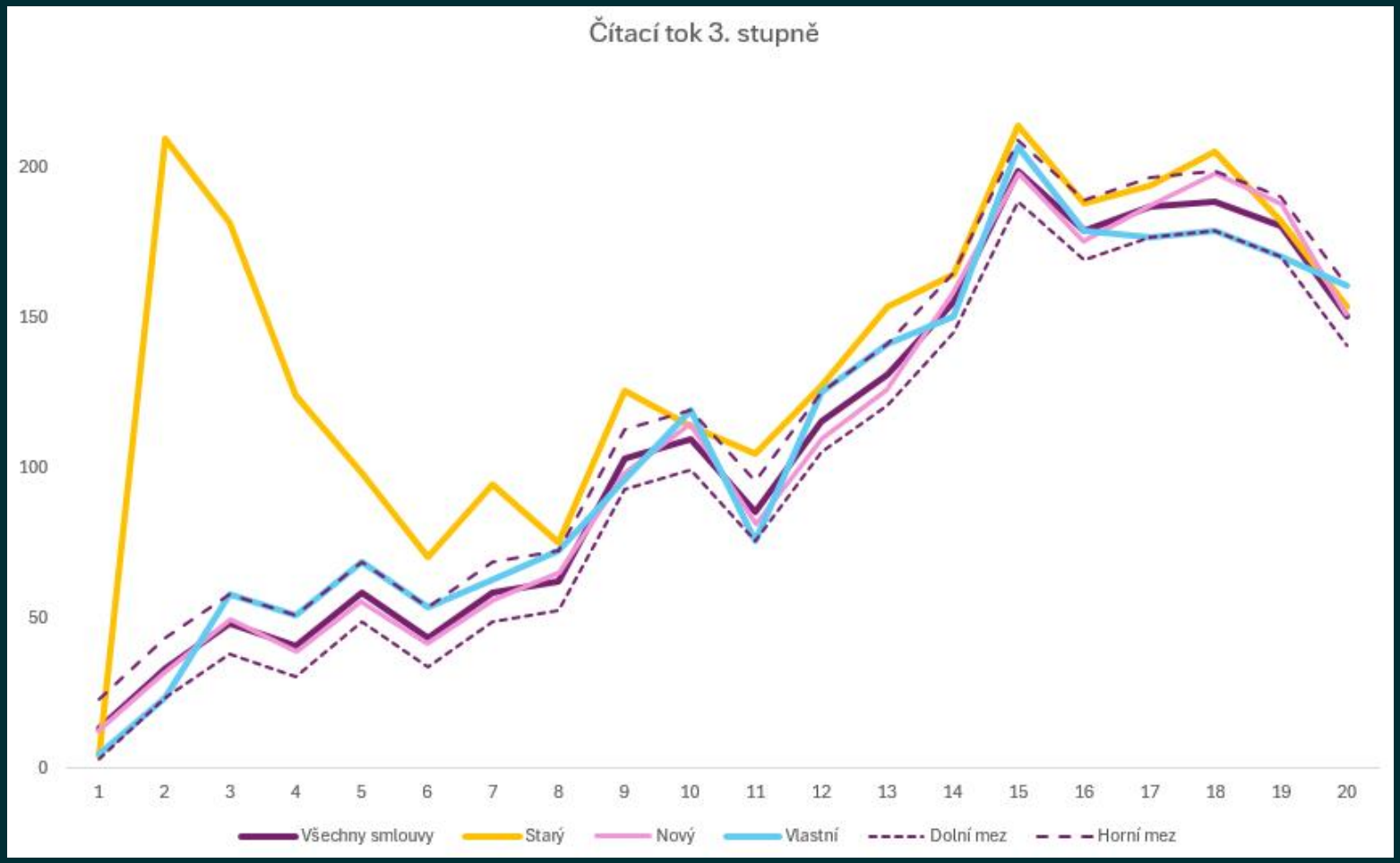
Contract boundary (CB)					Bez CB			Nový obchod - bez CB		
Produkt	Nástroj starý	Clustery 4h	Nástroj nový 60 iter (330 min)	Nástroj nový bez iter (5 min)	Nástroj starý	Clustery 4h	Nástroj nový bez iter	Nástroj starý	Clustery 4h	Nástroj nový bez iter
UNLI1	220	180	217	218	220	177	248			
UNLI2	500	320	418	454	500	332	405			
UNLI3	450	298	384	408	450	318	398			
RSID	225	246	348	361	225	286	319	225	164	226
TERM1	220	242	265	297	220	249	266	220	141	205
TERM2	220	230	269	284	220	196	232	220	35	67
ACNT	150	181	187	190	150	191	211			
VARI	150	225	227	246	150	241	273			
ENDR1	200	189	220	230	200	195	240			
ENDR2	50	43	64	64	50	45	69			
ENDR3	12	8	12	12	12	8	12			
ENDS1	4	2	4	4	4	2	4			
ENDS2	27	80	79	79	27	80	79			
TOTAL	2428	2244	2694	2847	2428	2320	2756	665	340	498

ACTUARIA

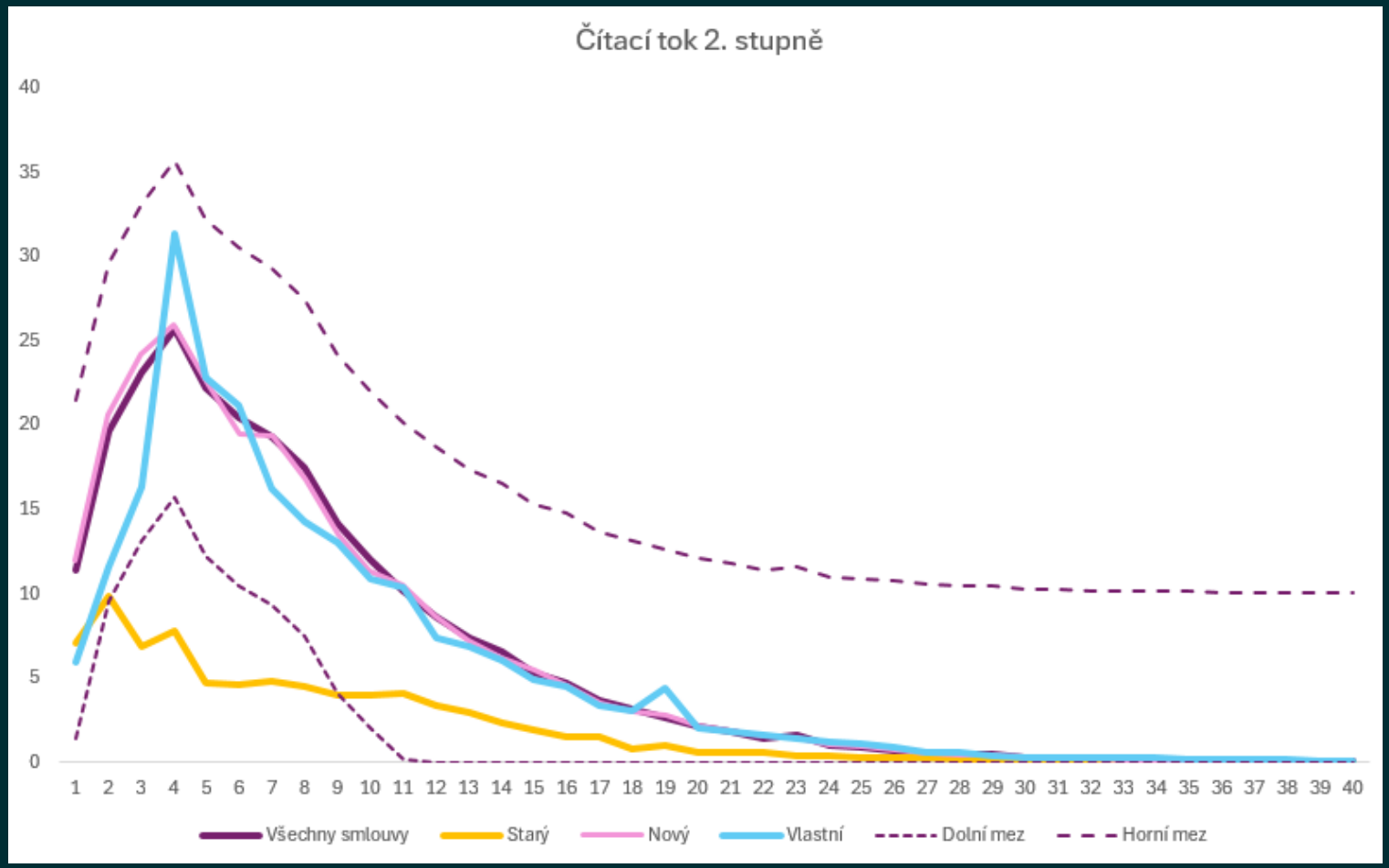
Porovnání odchylek

- Nový a vlastní nástroj při stejném nastavení dosahují velmi podobných odchylek, jež jsou mnohdy výrazně nižší než u starého nástroje
- I pro starý nástroj jsou však odchylky nejdůležitějších toků (1. stupně) téměř vždy uvnitř stanovených mezí
- Následující grafy slouží především k ilustraci různých případů, kdy starý nástroj selhává

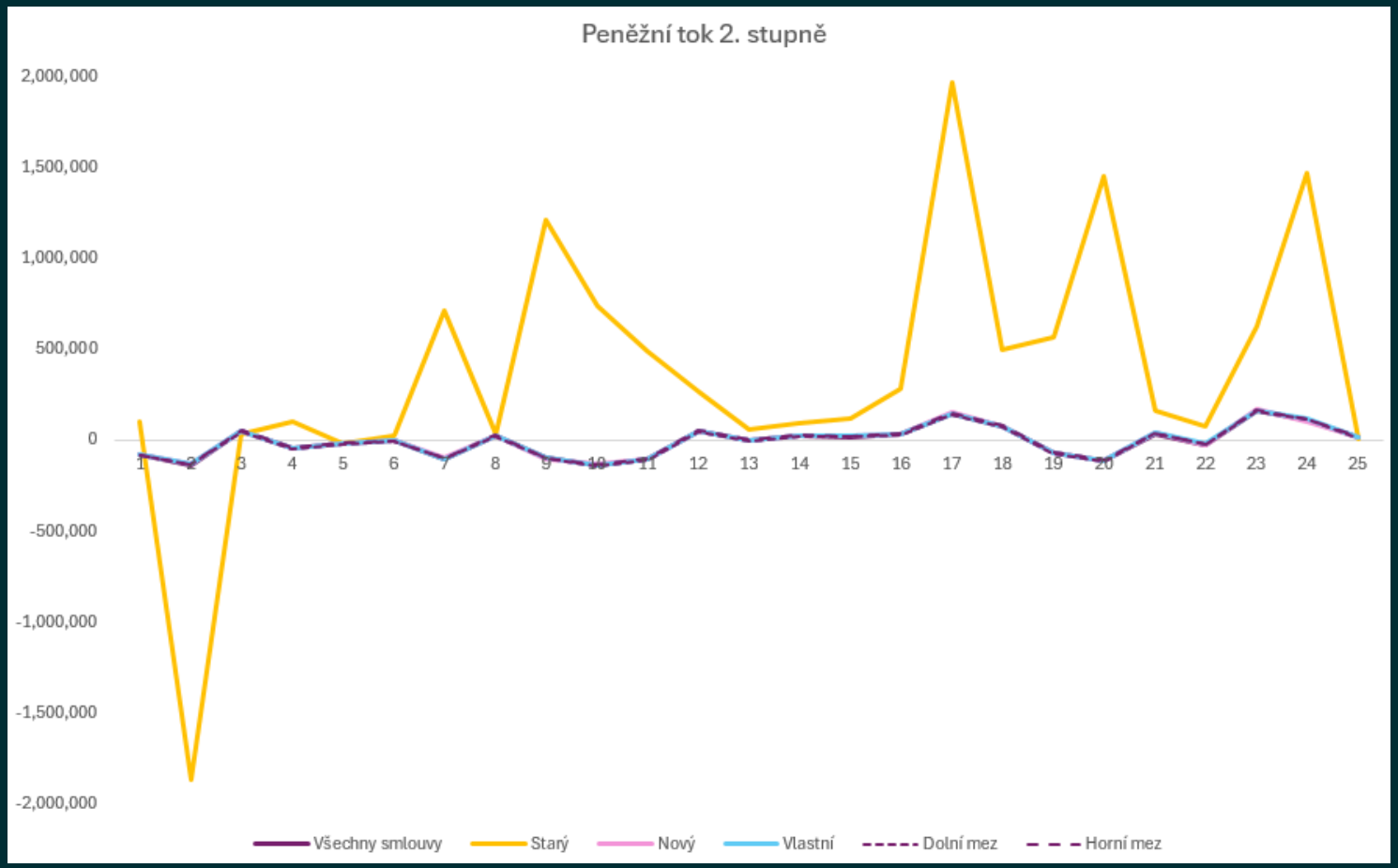
Porovnání odchylek – čítací toky



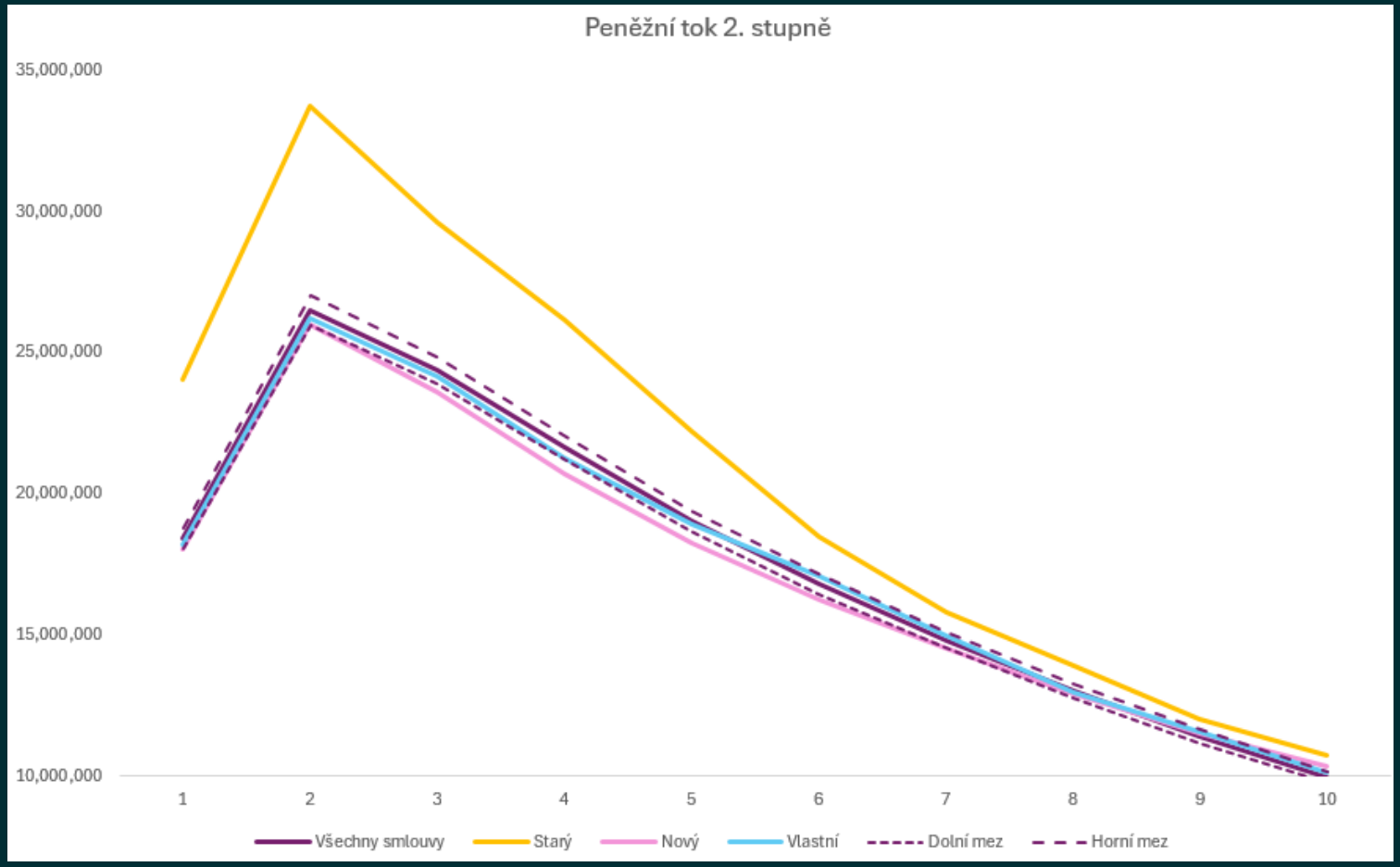
Porovnání odchylek – čítací toky



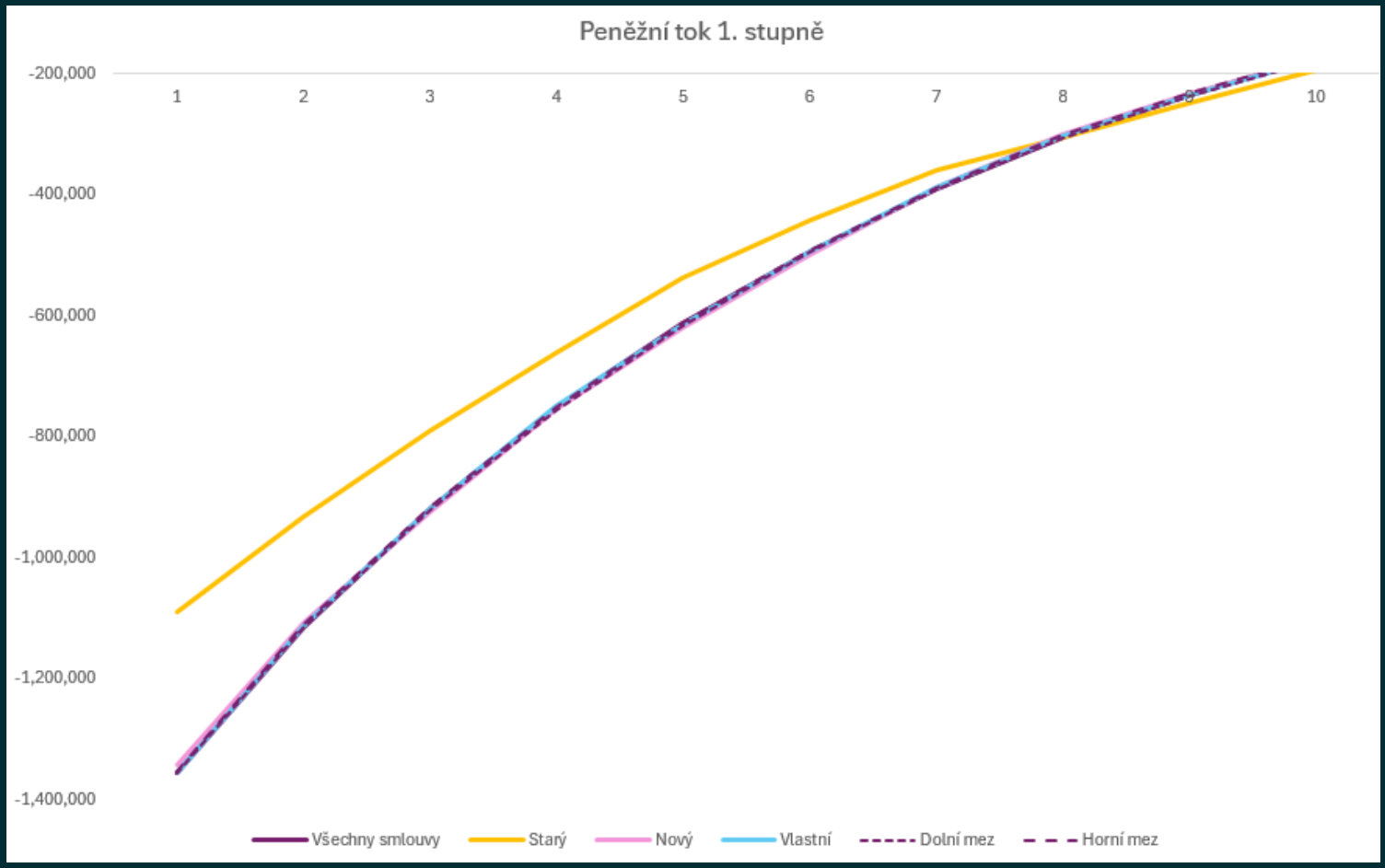
Porovnání odchylek – peněžní toky



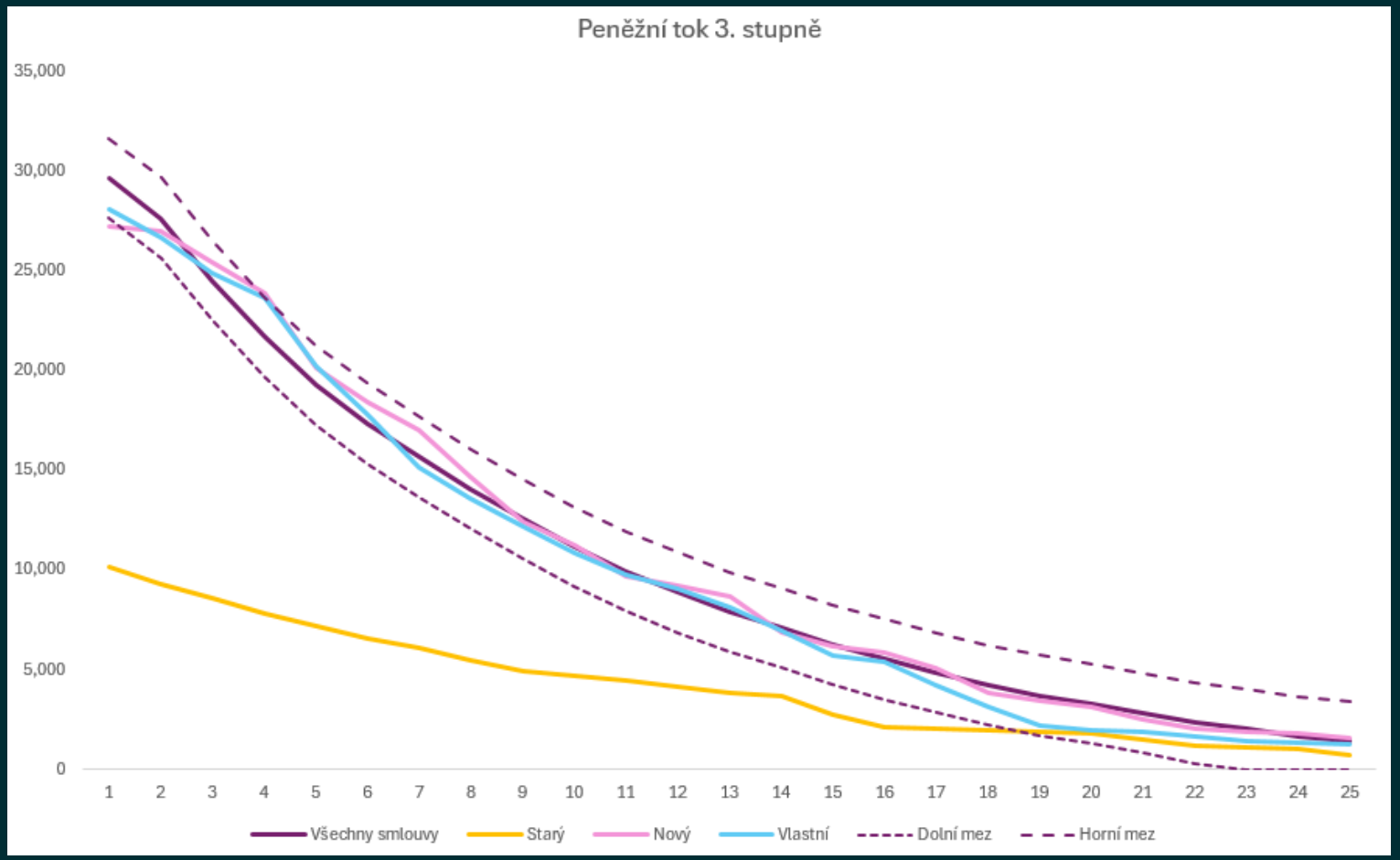
Porovnání odchylek – peněžní toky



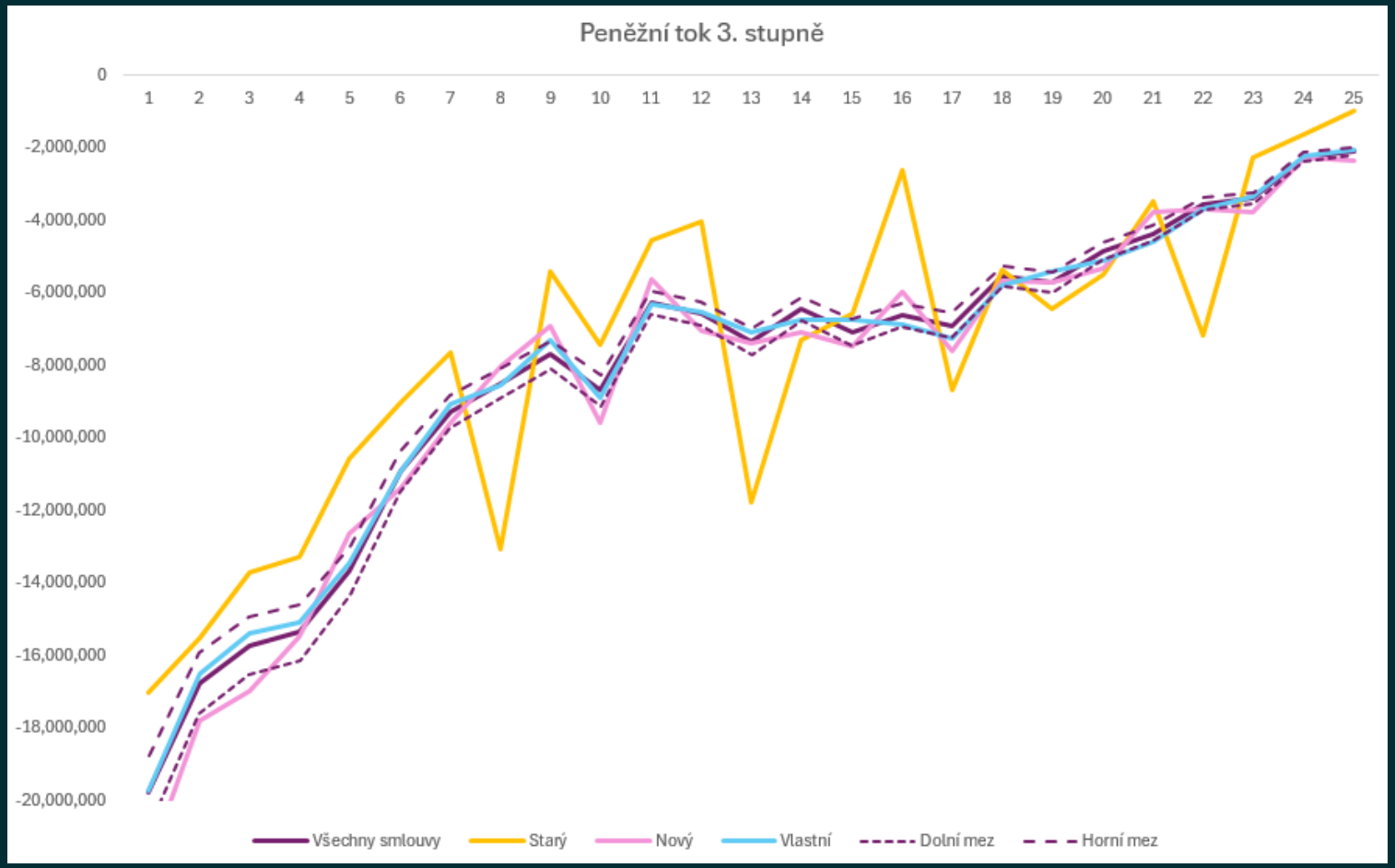
Porovnání odchylek – peněžní toky



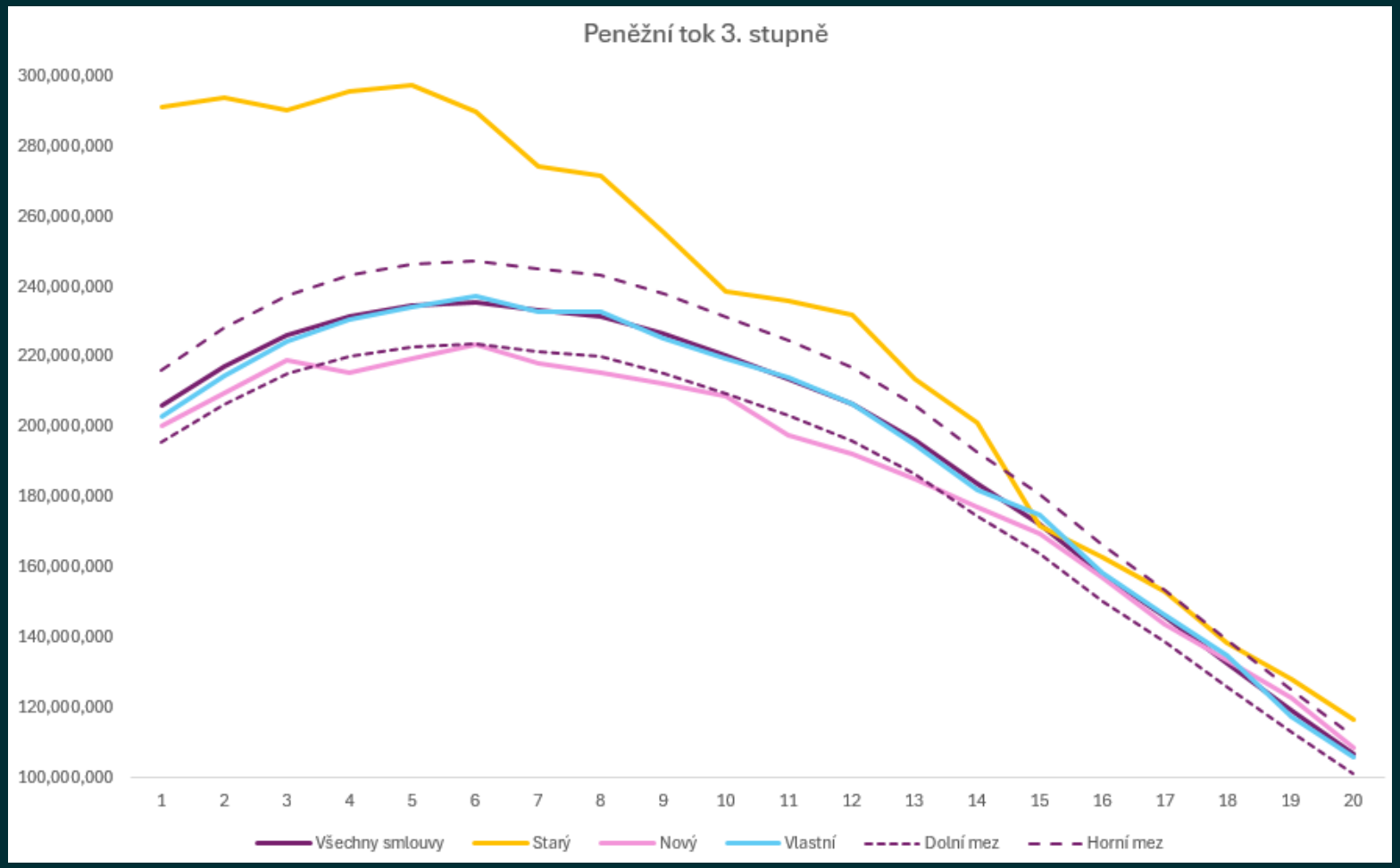
Porovnání odchylek – peněžní toky



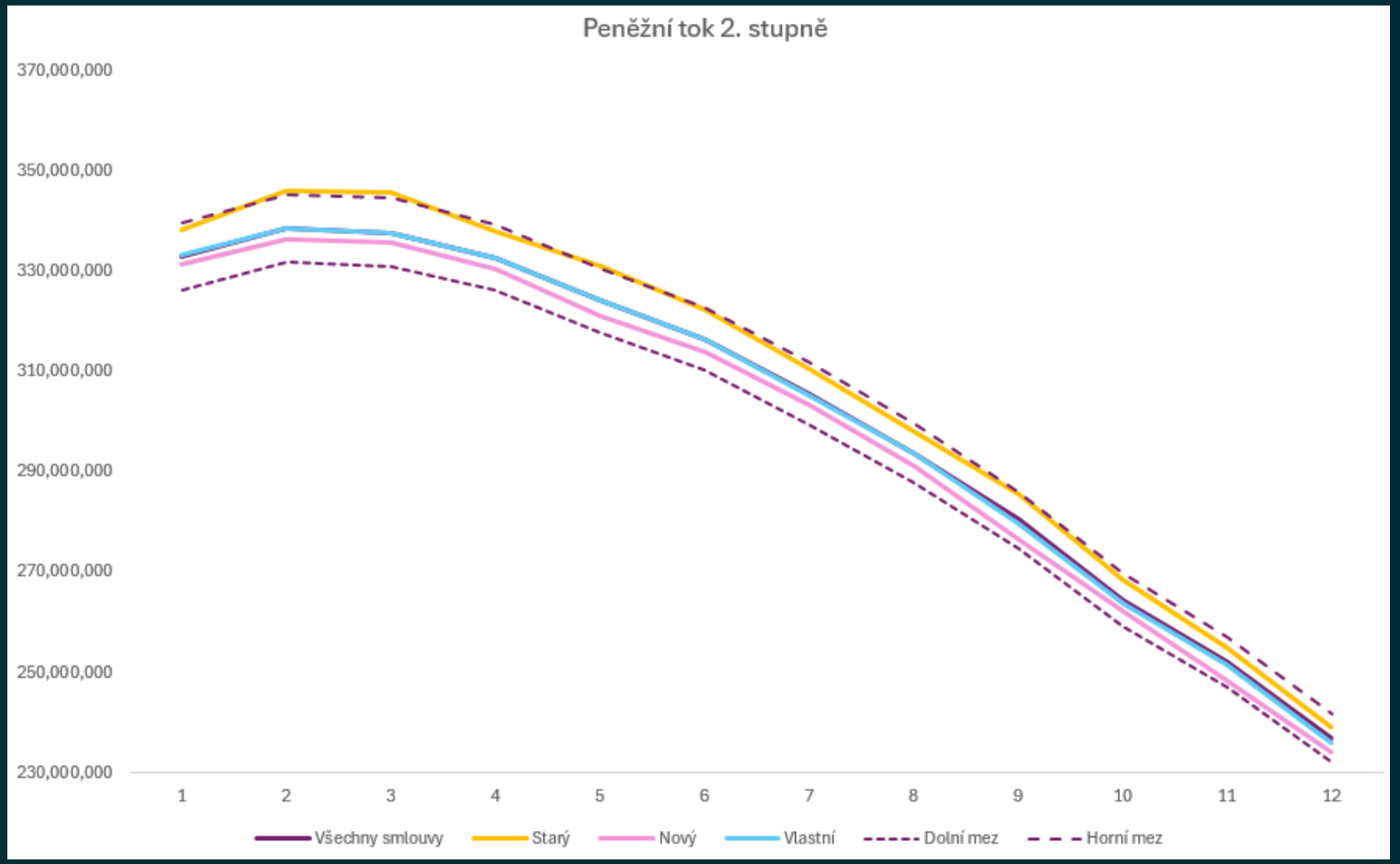
Porovnání odchylek – peněžní toky



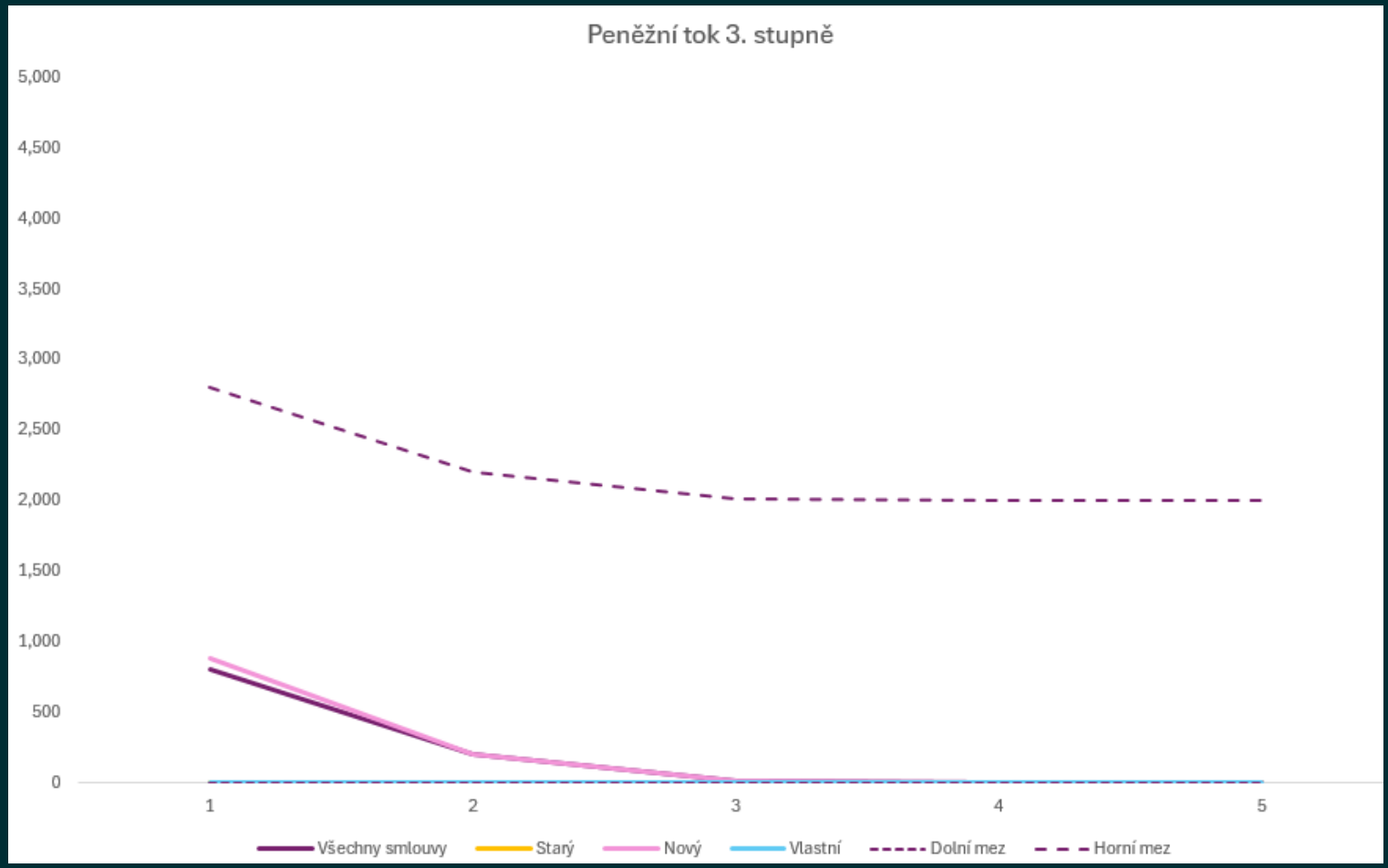
Porovnání odchylek – peněžní toky



Porovnání odchylek – peněžní toky



Porovnání odchylek – peněžní toky



Závěrečné srovnání

- Starý nástroj
  - Vždy se vejde do předepsaného počtu smluv, ale za cenu větších odchylek
  - Proces není plně automatizovaný, je třeba ruční práce
- Nový nástroj
  - Automatizovaný a velmi rychlý (5 min)
  - Výrazně kvalitnější seskupení z hlediska odchylek
  - Sice lehce vyšší celkový počet smluv, ale u důležitých produktů srovnatelný
- Vlastní nástroj
  - Vysoce flexibilní – lze nastavit variantu algoritmu dle dostupného výkonu a potřeby (rychlost seskupení, nízký počet smluv, …), možnost modifikovat kód algoritmu a hledat co nejlepší verzi pro své produkty
  - Srovnatelné odchylky s novým nástrojem
  - Nejmenší počet smluv, zejména snížení počtu smluv u produktu UNLI až o třetinu, díky čemuž se snižuje čas stochastického běhu o 20 % a náročnost na RAM o 17 %

Poznámky na konec

- Vstupní smlouvy nemusí „reprezentovat samy sebe“
  - Algoritmus nevyžaduje, aby referenční vektor byl součet toků na vstupních smlouvách
  - Díky tomu můžeme snížit počet smluv už na vstupu, což zrychlí výpočet. Referenční vektor bude stále součet toků za celé portfolio, ale do seskupení vstoupí např. pouze (nějak vybraná) třetina smluv, jejichž toky pak budeme pomocí vah škálovat, aby se trefily na celé portfolio.
    - Může výrazně pomoci u velkých produktů, kde i třetina (či ještě menší část) smluv má slušnou šanci na to, že z ní vybereme malý počet dobrých reprezentantů celého portfolia
    - Kdybychom nedělili na kusy, jde pouze o jinou volbu startovního vektoru **c**: namísto (1, 1, …, 1) bereme například (1, 0, 0, 1, 0, 0, 1, 0, 0, 1, …)
    - Pokud dělíme na kusy, je potřeba si rozmyslet, jakou část referenčního vektoru přiřadit danému kusu (už by to nebyl součet toků za smlouvy v onom kusu)
- Algoritmus lze modifikovat prakticky v každém kroku
  - Clustering nemusí být dle toků, ale může být dle jiných charakteristik smluv
  - Počet a velikost clusterů lze volit libovolně – ovlivní rychlost výpočtu a možná i počet smluv
  - Mohlo by pomoci zavést škálování dat a další…
- Pokud bychom seskupovali pouze TOTAL, celkový počet smluv vychází kolem 500