Вагомим мінусом фреймворку JavaFX є те, що побудований на ньому інтерфейс можна побачити лише після компіляції програми, що в деяких випадках може дуже сповільнювати процес розробки. Проте з часом і це перестало бути проблемою, оскільки появився JavaFX Scene Builder, який дозволяє за допомогою миші розміщувати та змінювати компоненти на інтерфейсі. JavaFX Scene Builder використовує свою власну мову розмітки FXML, CSS для написання стилів та скрипти для написання робочого коду.

3. Інтерфейс програми

У своїй програмі я намагався створити інтерфейс, який буде справді зручним для користувача, який буде хоча б трохи інтуїтивним та не вимагатиме від користувача якихось особливих знань.

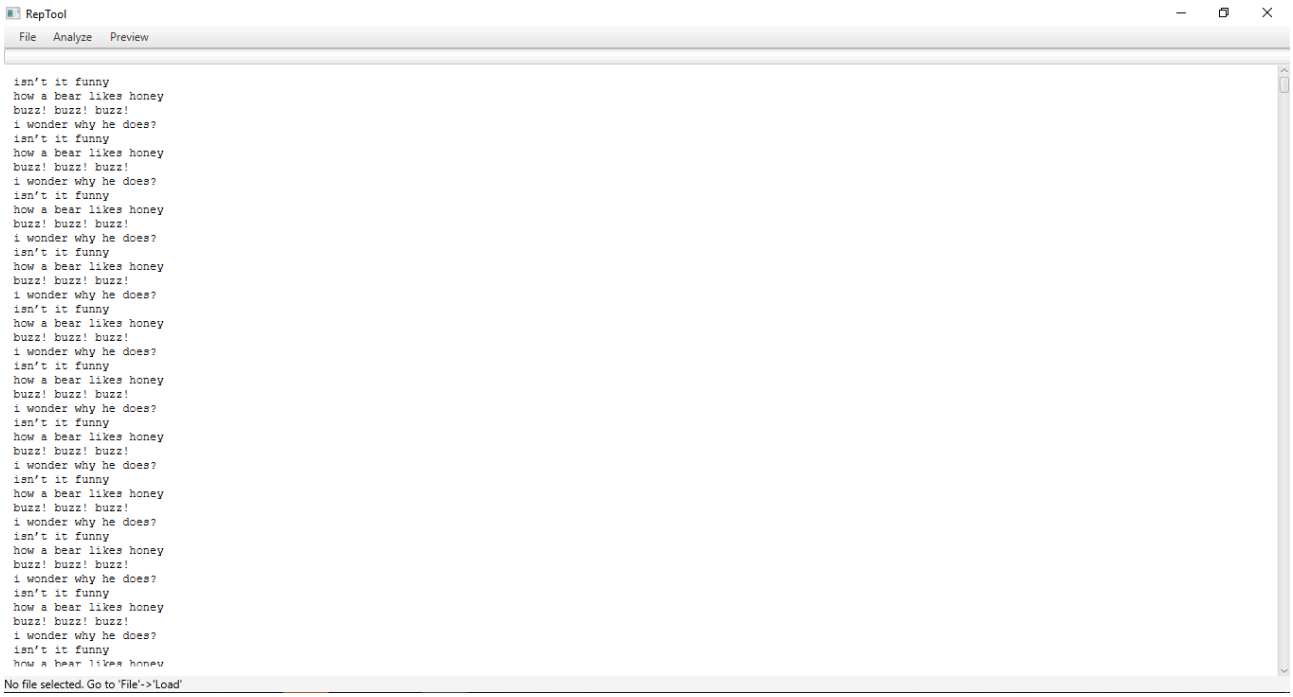

Рис. 1. Інтерфейс програми

Інтерфейс програми можна поділити на умовні секції. Згори розміщено меню, яке розділене на категорії File, Analyze та Preview

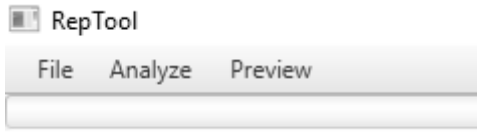

Рис. 2. Вигляд меню

Категорія File містить у собі опції відкриття файлів з текстом та збереження результатів аналізу тексту.

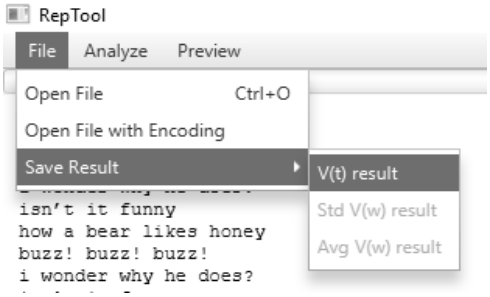

Рис. 3. Опції категорії File

Опція Open File відкриває діалогове вікно з можливістю вибрати потрібний користувачеві файл на комп'ютері. Після того, як користувач обере файл програма постарається сама в автоматичному режимі розпізнати кодування вмісту вибраного файлу, проте якщо якщо це не вдасться зробити, то буде вжито стандартне кодування UTF-8. Оскільки є велика ймовірність хибного розпізнавання кодування вмісту файлів я додав можливість користувачеві самому обрати кодування, з яким він хоче відкрити цей файл. Для того, щоб полегшити цю задачу для користувача я реалізував простенький autocomplete алгоритм для поля вводу формату кодування. Процес зчитування вмісту файлу супроводжується ProgressBar-ом, який дозволяє користувачеві візуально зрозуміти скільки часу приблизно залишилось до кінця зчитування. При успішному зчитуванні файлу його вміст появиться у компоненті CodeArea, яка і буде цей вміст відображати в інтерфейсі.

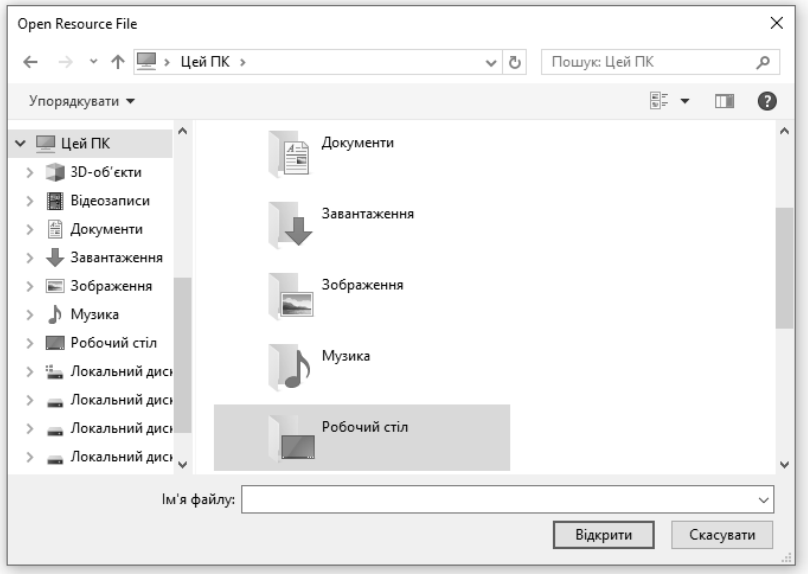

Рис. 4.

Діалогове вікно відкриття файлу

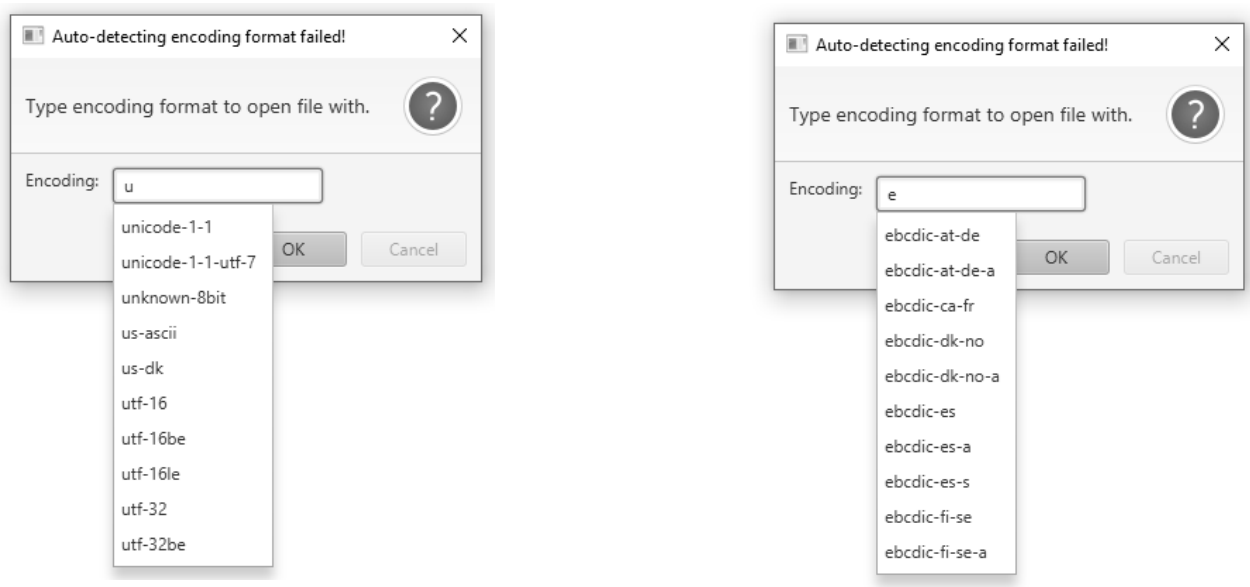

Рис. 5. Приклад роботи алгоритму autocomplete

За допомогою опції Save Result можна зберегти результати аналізу тексту у файл з розширенням *.CSV . Оскільки у програмі реалізовано режим біжучого вікна, результатами роботи якого є залежність $averageV(w)$ та $stdV(w)$, де V – це параметр повторюваності, а w – це ширина вікна, то додані додаткові опції для можливості збереження цих залежностей окремо одна від одної.

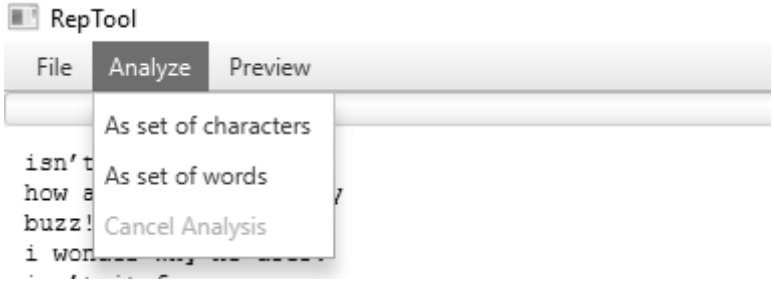

Рис. 6. Вигляд категорії Analyze

Категорія Analyze містить у собі опції для аналізу тексту, який зберігає у собі компонента CodeArea, попередньо перетворивши цей текст або в масив символів, або ж у масив слів (залежно від опції). Якщо користувач захоче скасувати процес аналізу, то він може це зробити вибравши опцію Cancel Analysis. Після того, як користувач обере якусь з перших двох опцій перед ним появиться вікно додаткових налаштувань, які будуть застосовані при анілізі тексту.

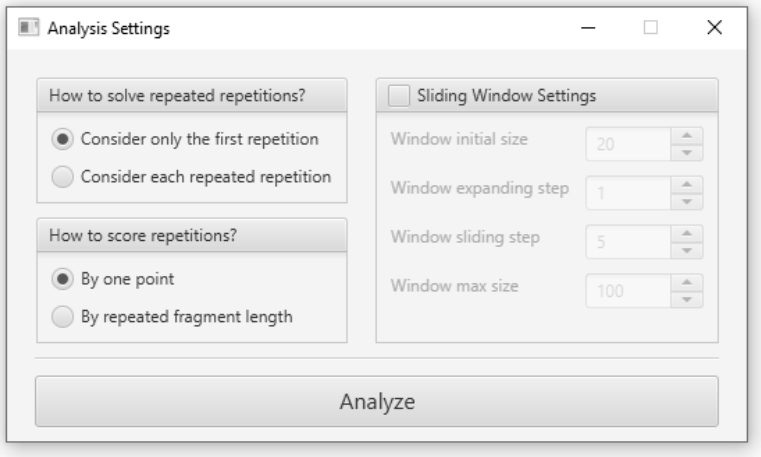

Рис. 7. Вигляд вікна налаштувань аналізу

Для аналізу текстів у програмі використовується алгоритм суфіксного дерева (Suffix Tree). Він використовує структуру даних граф для свого функціонування. Граф забезпечує високу швидкодію і що важливіше — займає відносно мало пам'яті. Якщо не вдаватись у деталі, то алгоритм суфіксного дерева зберігає у нодах графа повторювані фрагменти тексту, причому без дублювання. Кількість повторень певного фрагменту тексту виражається кількістю дочірніх листків(ноди, які не мають нащадків) нода, у якому цей фрагмент тексту

зберігається. Для того щоб знайти залежність параметра повторюваності від поточної позиції(індексу) в тексті алгоритм зараховує один бал за кожен повторюваний фрагмент тексту шляхом інкрементації певного лічильника. Сам же параметр повторюваності рівний поточному значенню лічильника розділеного на поточну позицію в тексті $V(t) = \frac{counter}{t}$. За замовчуванням алгоритм зараховує один бал за кожен фрагмент, який повторюється вперше, усі наступні повторення цього фрагменту в тексті не враховуються. Натомість я додав можливість користувачеві за допомогою вікна налаштувань вибрати такі сценарії, при яких до лічильника додається така кількість балів, яка і довжина повторюваного фрагмента, а також можливість врахування повторних повторів алгоритмом. Це вже дозволяє генерувати чотири комбінації вибраних параметрів, проте саме у цьому вікні налаштувань є можливість вибрати також режим Sliding Window, який повністю сумісний з усіма попередніми параметрами аналізу.

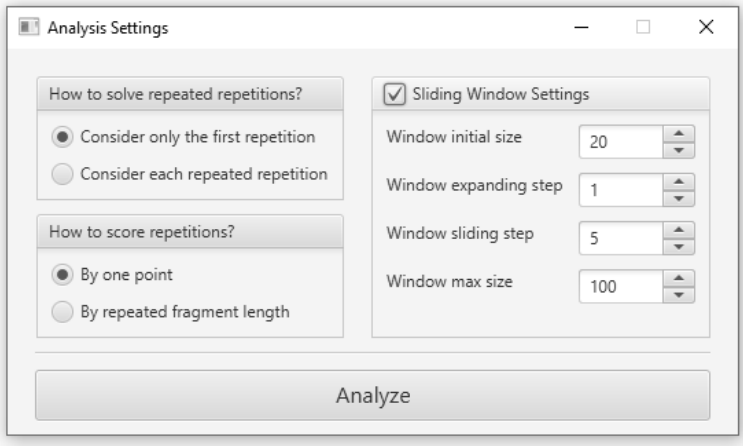

Рис. 8. Вигляд вікна налаштувань з ввімкненим режимом Sliding Window

Режим біжучого вікна(Sliding або Running window) характеризується чотирма основними параметрами: початковий розмір вікна, крок розширення вікна, крок зсуву(переміщення) вікна та максимальний розмір вікна аби якось обмежити роботу алгоритму, оскільки вона може бути надзвичайно тривалою.

Після натиснення кнопки Analyze внизу цього вікна у паралельному потоці почнеться аналіз тексту алгоритмом суфіксного дерева із вибраними налаштуваннями. Прогрес цього аналізу буде супроводжуватись ProgressBar-ом

вгорі, який дозволить оцінити користувачеві скільки вже тексту було проаналізовано та скільки ще залишилось. Також внизу програми, по завершенні аналізу з'являється StatusBar, який сповіщає користувача про те, скільки було проаналізовано фрагментів, який тип цих фрагментів (слова/символи), скільки часу у секундах тривав аналіз та вказівка про те, що в опції Save Result можна зберегти результат аналізу.

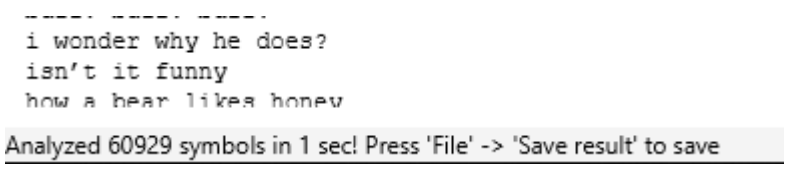

Рис. 9. Вигляд StatusBar-а по завершені аналізу

Оскільки збережені у файл дані результату аналізу представляються числами, записаними в один стовпець, що безумовно є не надто презентабельним, відносно складним для розуміння та банально займає велику кількість часу для сприйняття цих даних я вирішив додати вікно попереднього огляду даних результату аналізу. У цьому вікні дані результату представлені у вигляді графіка, який вже куди простіше може бути сприйнятий користувачем. Цей графік будується не по усіх вихідних даних, обсяг яких може бути надзвичайно великим і залежить від довжини тексту, а тому і процес побудови графіка може затягнутись, чого я допускати не хотів, тому кількість точок у цьому preview графіку це стала величина рівна тисячі точок, а у випадку, коли кількість проаналізованих фрагментів менша за цю сталу величину, то кількість точок для відображення графіка буде відповідною.

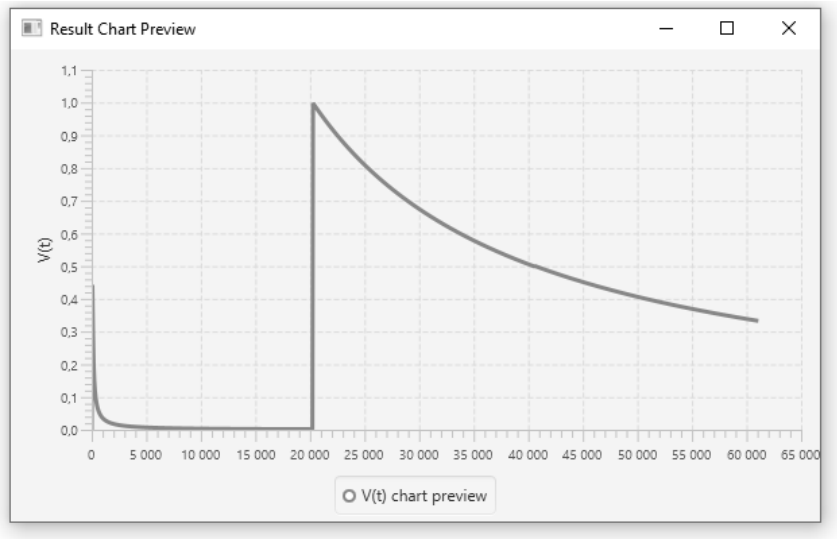

Рис. 10. Вигляд вікна Preview Chart

І остання категорія меню — це Preview. Необхідність цієї категорії і опцій у ній зумовлена тим, що вікно preview графіка може бути закрите користувачем і для того, аби не змушувати користувача заново проводити аналіз, який може бути тривалим я додав цю категорію, натиснувши на якусь з її опцій появиться preview вікно з відповідним побудованим графіком. Останні дві опції - це побудова графіків по результатах роботи режиму біжучого вікна.

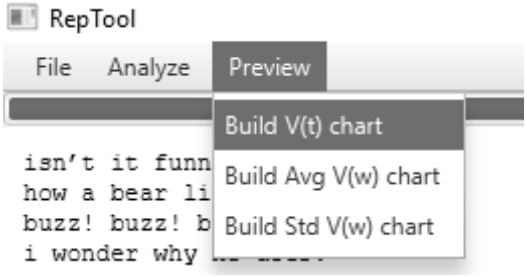

Рис. 11. Вигляд категорії Preview

5. Sliding Window

Як вже було описано попередньо, режим біжучого або ковзного вікна характеризується чотирма основними парамстрами: початковий розмір вікна, крок розширення вікна, крок зсуву(переміщення) вікна та максимальний розмір вікна.

Алгоритм працює наступним чином: існує два цикли, один вкладсний в інший, зовнішній цикл на кожній своїй ітерації збільшує розмір вікна на величину кроку розширення вікна та зсуває вікно на величину кроку зсуву. Внутрішній цикл маючи поточну позицію вікна та його ширину витягує або ж копіює відповідний фрагмент у тій же позиції і з тією ж довжиною/шириною з вихідного тексту, якщо вибрано режим слів, то цей фрагмент різіб'ється на слова, які будуть проаналізовані алгоритмом суфіксного дерева, якщо ж ні, то проаналізується множина символів. Таким чином внутрішній цикл проаналізує увесь текст паралельно записуючи результати аналізу, коли ж внутрішній цикл досягне кінця тексту програма знаходить середнє арифметичне та середнє квадратичне відхилення по цих результатах та також їх запам'ятовує. На наступній ітерації алгоритм біжучого вікна знову проаналізує увесь текст проте вже з вікнами іншої ширини та іншим кроком зсуву. Так триватиме доти, поки ширина вікна не досягне величини максимальної ширини вікна, яка задається в налаштуваннях після чого і можуть бути побудованими графіки залежностей $avgV(t)$ та $stdV(t)$.

Застосування алгоритму біжучого вікна дозволяє згладити графіки залежностей завдяки тому, що береться середнє арифметичне значення. Проте цей алгоритм також має один дуже вагомий мінус — аналіз надзвичайно тривалий. Також процес аналізу тексту у цьому режимі має тенденцію дещо сповільнюватись із плином часу, хоч це і відбувається лише тому, що з кожною новою ітерацією зовнішнього циклу алгоритму доводиться обробляти щоразу більше даних. Саме через тривалість роботи цього алгоритму і є надзвичайно затратним у часі дослідження щодо того, наскільки кращими чи гіршими є

результати залежностей у порівнянні зі результатами без використання біжучого вікна. Нижче приведені результати:

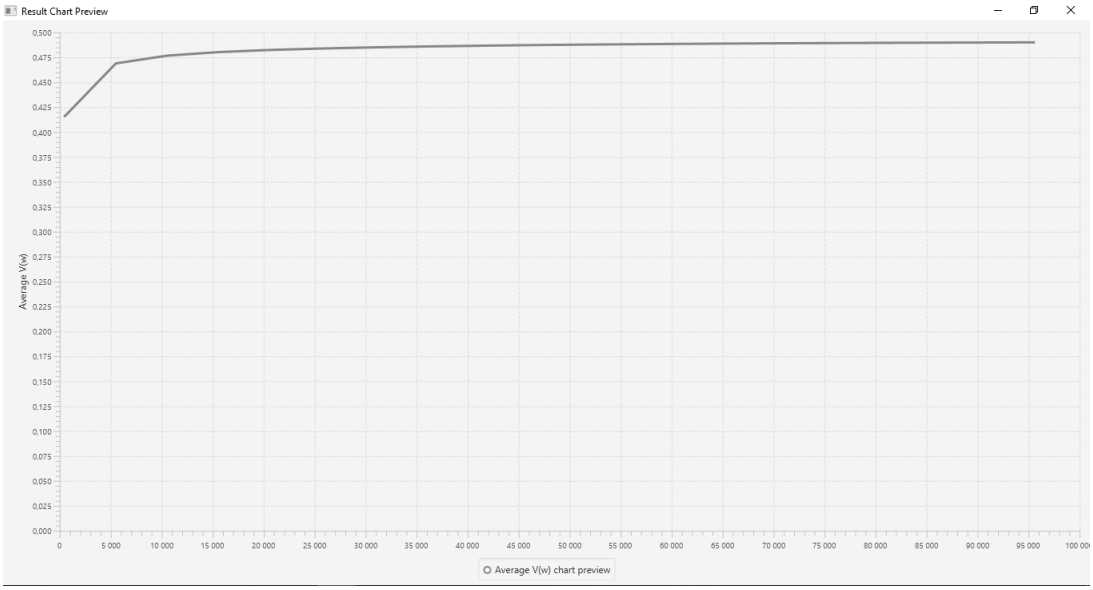

Рис.25. Залежність avg V(w) в режимі біжучого вікна

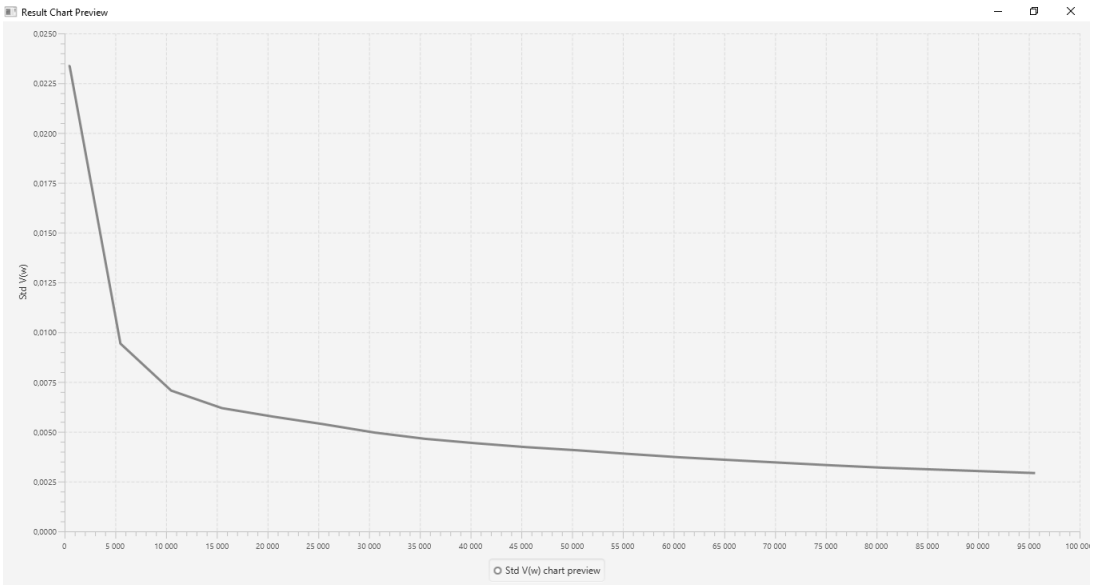

Рис. 26. Залежність std V(w) в режимі біжучого вікна

Графіки на рис. 25 та 26 — результати аналізу в режимі біжучого вікна по символах, проведені при налаштуваннях: 500, 5000, 5000, 100000 (згідно того порядку, у якому я перелічив їх на початку параграфу 5). На рис. 27 та 28 представлені графіки тих же залежностей при тих самих налаштуваннях, але вже з вибраним режимом слів, а не символів.

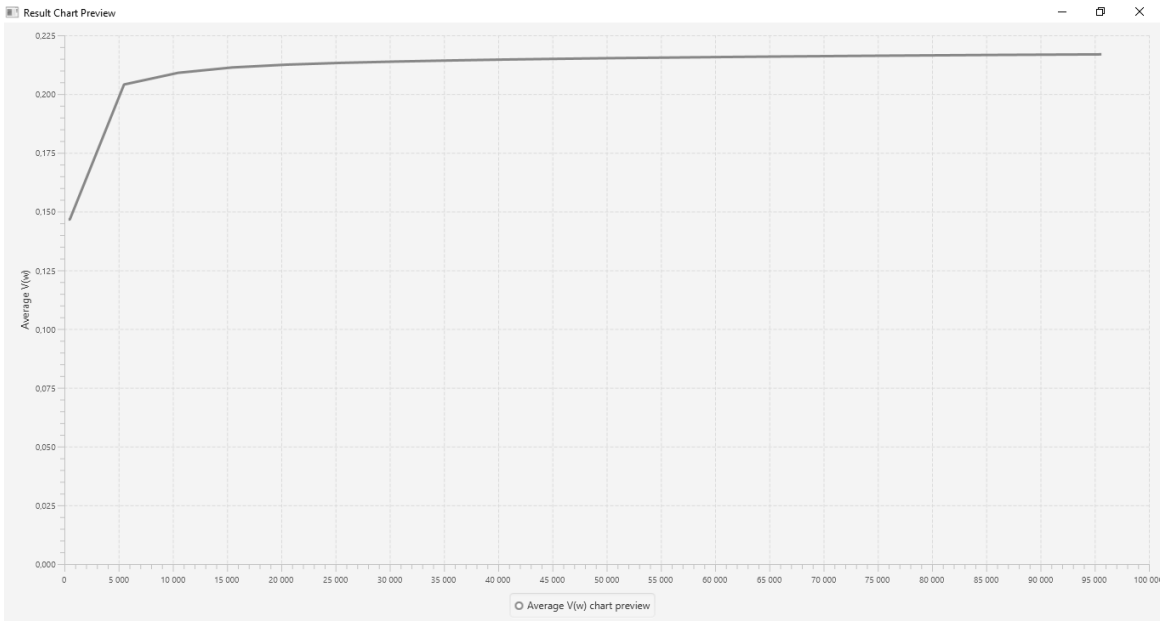

Рис. 27. Залежність avg V(w) в режимі біжучого вікна, режим слів

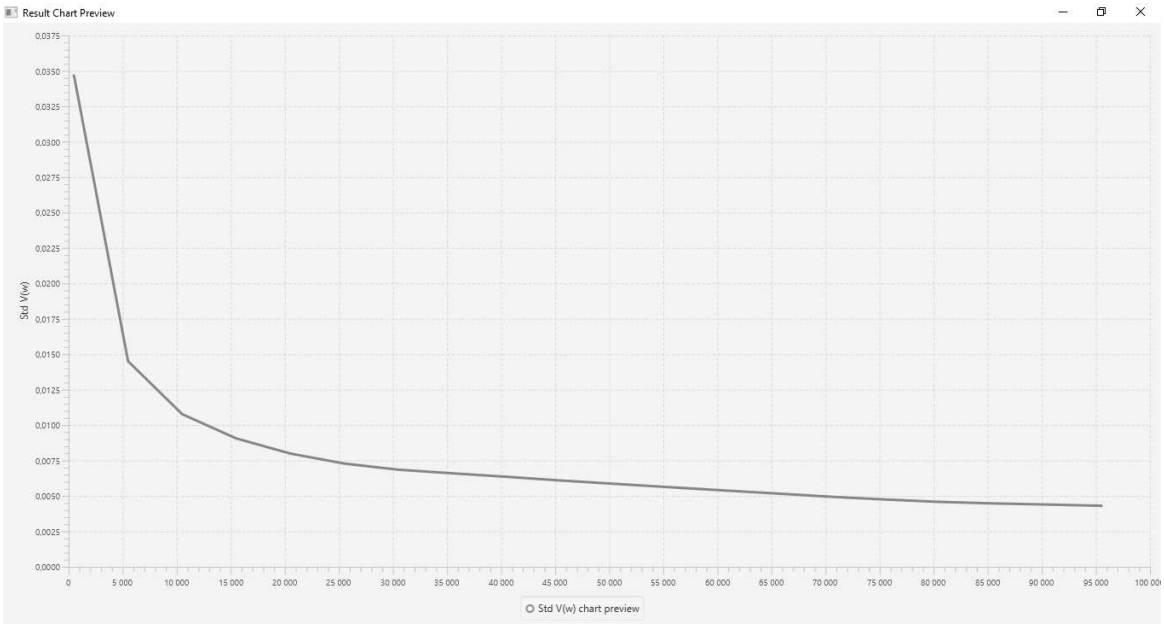

Рис. 28. Залежність std V(w) в режимі біжучого вікна, режим слів

Через використання біжучого вікна, а точніше усереднення, яке в ньому проводиться максимуми як в режимі слів, так і в режимі символів трохи просідають в порівнянні з результатами, коли біжуче вікно не використовується, проте це просідання — це величини менші 10% від максимуму результату без використання біжучого вікна, а тому не є надто критичними, зрештою при усередненні завжди втрачається певна кількість даних.