פרוייקט גמר בקורס אנאליזה נומרית:
אינטרפולציית Neville
Neville's Method

אדם אהרוני מאור ולדמן

מכון טכנולוגי חולון HIT

יום שני, 01. מרץ, 2021.

פרולוג:

במסגרת פרוייקט זה, נסביר, נפרט, ונראה בפעולה על שיטת האינטרפולצייה לפי Neville.
נוכיח גם איך היא עובדת, נראה דוגמאות, וכן נשווה מול שיטות אינטרפולצייה אחרות שלמדנו בקורס.

במסגרת הפרוייקט הרחבנו את אופקינו בתחום האנאליזה הנומרית והאינטרפולצייה.
הפרוייקט גרם לנו לחקור את שיטת Neville עד תום, ובכך נתן לנו מקור להשוואה עם שיטות אינטרפולצייה אחרות.

פרולוג:

בהינתן $1 + n$ נקודות דאטה, קיים פולינום ייחודי מסדר $n \leq$ העובר דרך הנקודות. באמצעות שיטת Neville ניתן להגיע אליו, או במקרה שלנו, לערך שלו בנקודה כלשהי.

- השיטה מבוססת על שיטת האינטרפולצייה של ניוטון (Newton) ועל מציאת הפרשים מחולקים.

פולינום האינטרפולצייה:

בהינתן $n+1$ נקודות דאטה $(x_k; y_k)$, $k = 0; 1; \ldots; n$, כאשר שיעורי ה-x שונים, פולינום האינטרפולצייה הוא פולינום מסדר של לכל היותר n המקיים את:

$$\forall k = 0; 1; \ldots; n: \quad p(x_k) = y_k$$

הפולינום הזה קיים והוא ייחודי. שיטת Neville מוצאת את ערך פולינום האינטרפולצייה הנ.ל. בנקודה x כלשהי.

נגדיר פולינום $p_{i;j}(x)$ (כאשר מתקיים $i \leq j$) בתור פולינום מסדר של $j - i$ העובר דרך נקודות הדאטה הנתונות $(x_k; y_k)$ עבור $k = i; i + 1; \ldots; j$.

תהליך מציאת הפולינום:

על הפולינומים לקיים את היחס הרקורסיבי:

$$\begin{cases} p_{i;i}(x) & = y_i \\ p_{i;j}(x) & = \frac{(x-x_j)p_{i;j-1}(x)-(x-x_i)p_{i+1;j}(x)}{x_i-x_j} \end{cases} ; \quad \begin{matrix} 0 \le i \le n \\ 0 \le i < j \le n \end{matrix}$$

נוסחת הרקורסייה הזו יכולה לחשב את $p_{0;n}(x)$, שהוא הפולינום אותו אנו מחפשים.

הוכחת הנוסחה הרקורסיבית:

הביטוי $y_i = p_{i;i}(x)$ הוא מיידי, ואת הנוסחה הרקורסיבית נוכיח באמצעות אינדוקצייה:

בהינתן $i; j$ עם פולינום $p_{i;j-1}(x)$ העובר דרך הנקודות $(x_k; y_k)$ עם $k = i; i+1; \ldots; j-1$,
וכן הפולינום $p_{i+1;j}(x)$ העובר דרך הנקודות $(x_k; y_k)$ עם $k = i+1; i+2; \ldots; j$.
לפי ההנחה הנ.ל. מתקיים:

$$\begin{cases} p_{i;j-1}(x_k) = y_k & i \le k \le j-1 \\ p_{i+1;j}(x_k) = y_k & i+1 \le k \le j \end{cases}$$

ולכן, עבור $i+1 \le k \le j-1$, מתקבל:

$$p_{i;j}(x_k) = \frac{(x_k - x_j)\, p_{i;j-1}(x_k) - (x_k - x_i)\, p_{i+1;j}(x_k)}{x_i - x_j} =$$

$$= \frac{(x_k - x_j)\, y_k - (x_k - x_i)\, y_k}{x_i - x_j} = y_k$$

וכן:

$$\begin{cases} p_{i;j}(x_i) & = \frac{(x_i - x_j)\, p_{i;j-1}(x_i)}{x_i - x_j} = y_i \\ p_{i;j}(x_j) & = \frac{-(x_j - x_i)\, p_{i+1;j}(x_j)}{x_i - x_j} = y_j \end{cases}$$

ולכן $p_{i;j}(x)$ עובר דרך כל הנקודות $(x_k; y_k)$ כאשר $k = i; i+1; \ldots; j$.

הוכח.

דוגמה:

ניקח את $n = 4$. נוכל להשתמש בנוסחת הרקורסייה על מנת למלא את התרשים משמאל לימין באופן הבא:

$$
\begin{array}{ccccccccc}
p_{0;0}(x) = y_0 & & & & & & & & \\
& p_{0;1}(x) & & & & & & & \\
p_{1;1}(x) = y_1 & & p_{0;2}(x) & & & & & & \\
& p_{1;2}(x) & & p_{0;3}(x) & & & & & \\
p_{2;2}(x) = y_2 & & p_{1;3}(x) & & \boxed{p_{0;4}(x)} & & & & \\
& p_{2;3}(x) & & p_{1;4}(x) & & & & & \\
p_{3;3}(x) = y_3 & & p_{2;4}(x) & & & & & & \\
& p_{3;4}(x) & & & & & & & \\
p_{4;4}(x) = y_4 & & & & & & & &
\end{array}
$$

דוגמה

התהליך מביא לנו את $p_{0;\,4}\left(x\right)$, שהוא בעצם ערך הפולינום העובר דרך כל $n+1$ נקודות הדאטה (מתנאי ההתחלה) בנקודה x כלשהי.

(ניתן גם לומר שאלגוריתם זה רץ בסיבוכיות של $O\left(n^2\right)$, הוכחה בהמשך)

זמן הריצה של האלגוריתם:

בהתחשב בכך שכדי למצוא את פולינום $p_{0;\,n}(x)$ נצטרך להשתמש ברקורסייה שבה מחשבים לראשונה n פולינומים, לאחר מכן $n-1$ פולינומים וכן הלאה...
מכאן, גודל הפולינומים שמחשבים מובא על ידי:

$$(n+1)+n+(n-1)+(n-2)+\cdots+1=$$

$$=\sum_{i=1}^{n+1} i$$

מדובר בסכום סדרה חשבונית סטנדרטית. לא קשה לראות שהסכום מקיים:

$$S = \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} = \frac{n^2+3n+2}{2} = \\ = \boxed{O\left(n^2\right)}$$

ומכאן, הגענו לזמן ריצה של $O\left(n^2\right)$.

הוכח.

זמן ריצת התוכנית כולה:

קיימות שתי דרכים ליישום שיטת Neville בקוד Python:

- שיטת פתירת הרקורסייה לכל ערך של x בנפרד.
- שיטת פתירת הרקורסייה פעם אחת באופן סימבולי, ואז הצבה עבור כל ערכי ה-x.

שיטת פתירת הרקורסייה לכל ערך x בנפרד:

בשיטה זו, נציב עבור k ערכים בתחום כלשהו ובמרווח שווה (`numpy.linspace`) ונחשב לכל ערך x את ערך הפולינום באותה נקודה.
למשל: עבור הערך $x = 1$ נחשב את ערך הפולינום בדוגמה הקודמת כך:

$$
\begin{array}{ccccc}
p_{0;\,0}(1) = y_0 & & & & \\
& p_{0;\,1}(1) & & & \\
p_{1;\,1}(1) = y_1 & & p_{0;\,2}(1) & & \\
& p_{1;\,2}(1) & & p_{0;\,3}(1) & \\
p_{2;\,2}(1) = y_2 & & p_{1;\,3}(1) & & \boxed{p_{0;\,4}(1)} \\
& p_{2;\,3}(1) & & p_{1;\,4}(1) & \\
p_{3;\,3}(1) = y_3 & & p_{2;\,4}(1) & & \\
& p_{3;\,4}(1) & & & \\
p_{4;\,4}(1) = y_4 & & & &
\end{array}
$$

זמן ריצת הקוד שלנו

בגלל שנצטרך להפעיל את האלגוריתם על כל k נקודות הדגימה זמן הריצה של האלגוריתם יהיה בסדר גודל של $O\left(kn^2\right)$, בהתחשב בכך ש-k גדול באופן משמעותי מ-n.

את אלגוריתם זה ניתן לממש בספריית NumPy.

שיטת פתירת הרקורסייה באופן סימבולי:

בשיטה זו, ראשית נפתור את הנוסחה הרקורסיבית באופן סימבולי, ואז נציב עבור k ערכים בתחום כלשהו ובמרווח שווה (`numpy.linspace`) ונחשב לכל ערך x את ערך הפולינום באותה נקודה.

זמן ריצת הקוד שלנו

בגלל שנצטרך להפעיל את האלגוריתם רק פעם אחת, ואז להציב k ערכים בפולינום המתקבל, זמן הריצה של האלגוריתם יהיה בסדר גודל של $O\left(n^2+k\right)$, או ברוב המקרים $O\left(k\right)$, בהתחשב בכך ש-k גדול באופן משמעותי מ-n.

את אלגוריתם זה ניתן לממש בספריית SymPy.

בחירה באלגוריתם הראשון:

למרות שבמבט ראשון ניתן לחשוב שהאלגוריתם השני עדיף על הראשון, יש לזכור כי ספריית SymPy איטית משמעותית מ-NumPy, ולכן בחרנו באלגוריתם המהיר יותר.

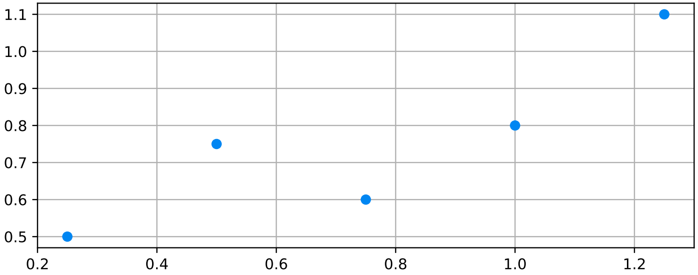
(ההפרש בין סדרי הגודל של זמני הריצה אמנם קריטי על הנייר, אך בפועל לא)

דוגמאות

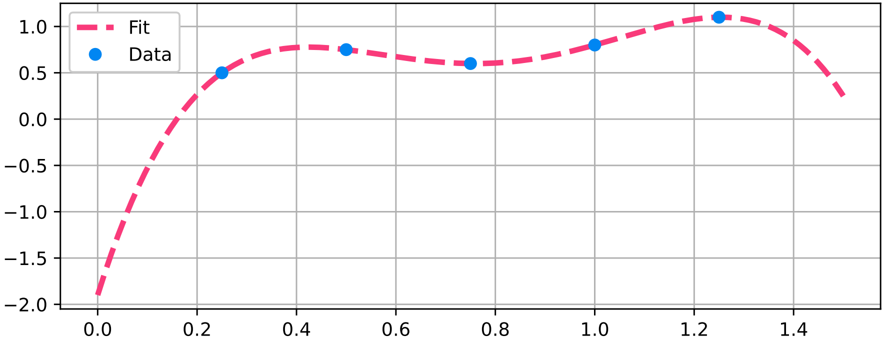
- רוב הפונקציות באו מהרצאות, תרגולים, ומטלות בית.

דוגמה N°1:

הפונקצייה לפני אינטרפולצייה:

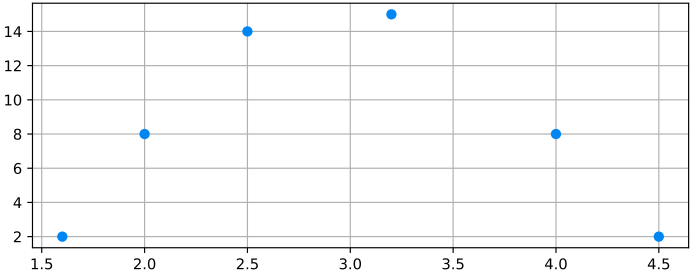


הפונקצייה אחרי אינטרפולצייה:

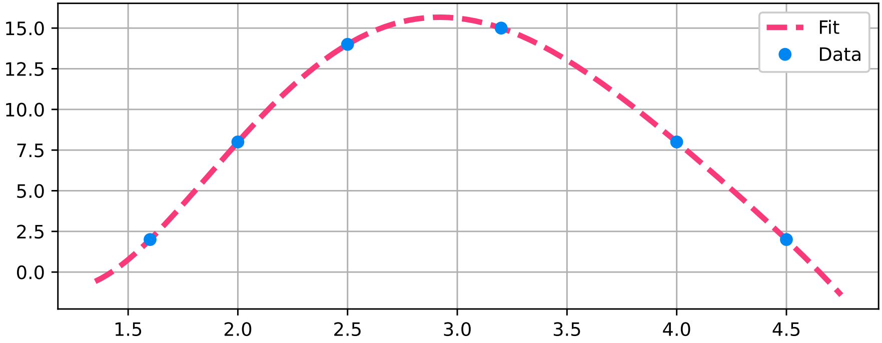


דוגמה N°2:

הפונקצייה לפני אינטרפולצייה:

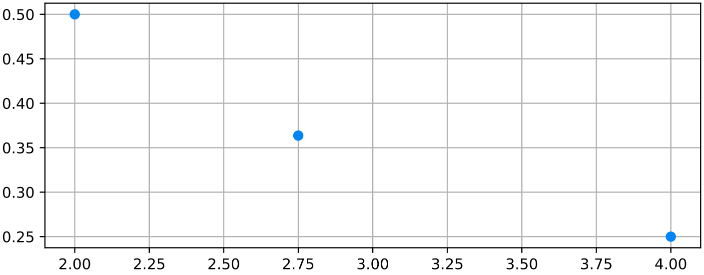


הפונקצייה אחרי אינטרפולצייה:

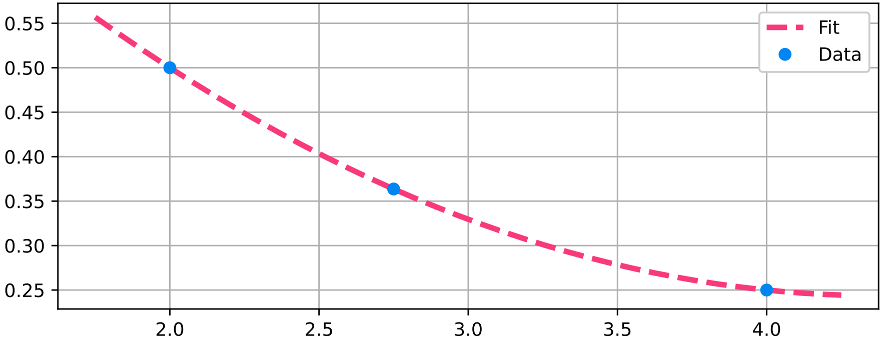


דוגמה N°3:

הפונקצייה לפני אינטרפולצייה:

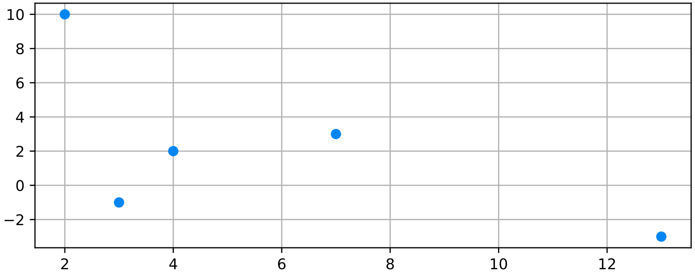


הפונקצייה אחרי אינטרפולצייה:

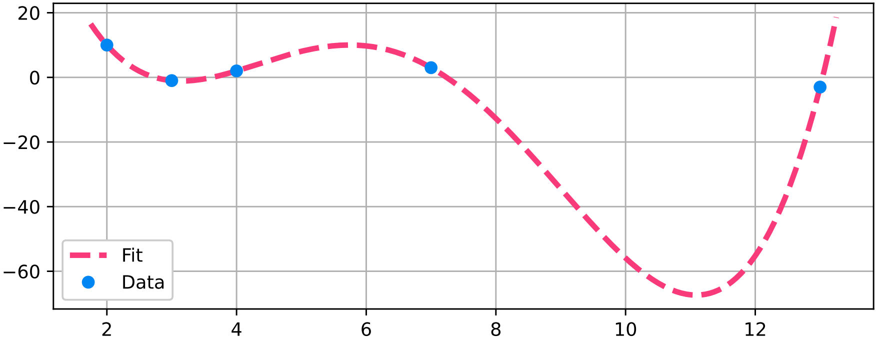


דוגמה N°4:

הפונקצייה לפני אינטרפולצייה:

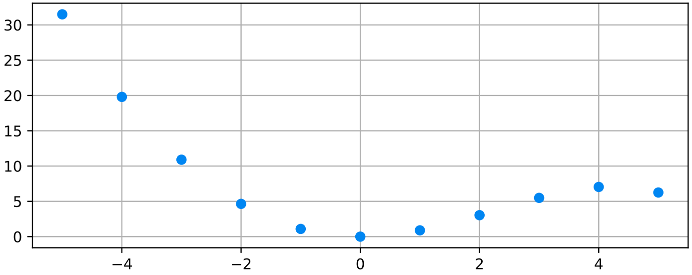


הפונקצייה אחרי אינטרפולצייה:

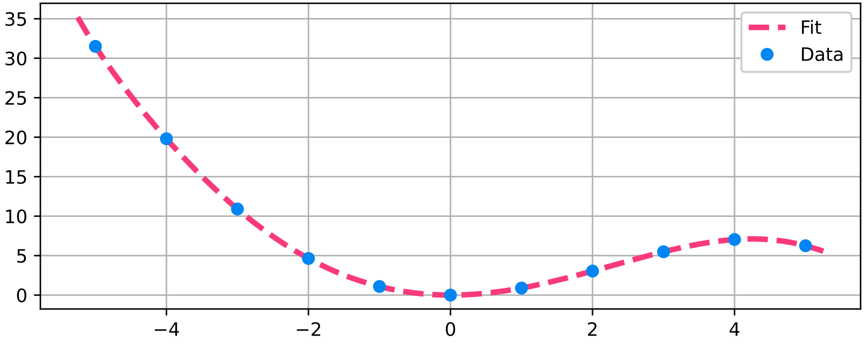


דוגמה N°5:

הפונקצייה לפני אינטרפולצייה:

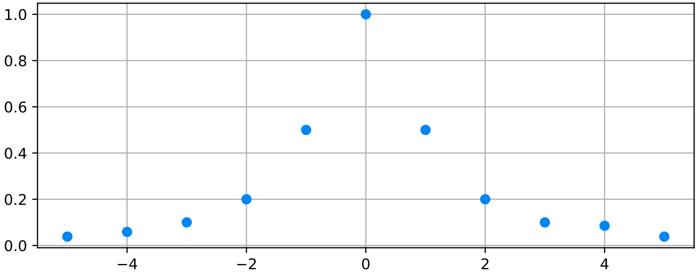


הפונקצייה אחרי אינטרפולצייה:

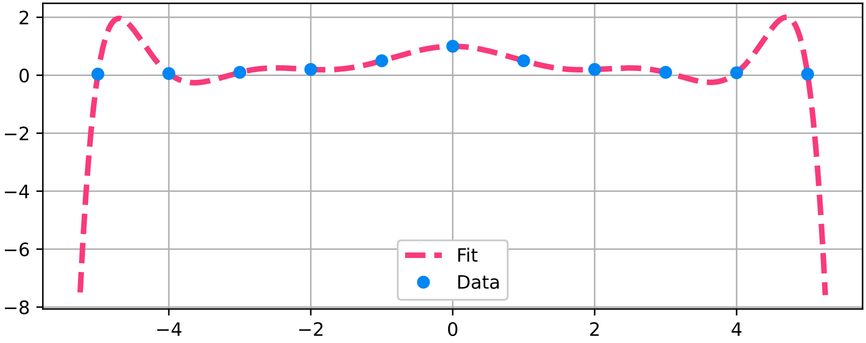


דוגמה N°6:

הפונקצייה לפני אינטרפולצייה:



הפונקצייה אחרי אינטרפולצייה:



השוואה עם טכניקות אחרות:

כעת, נערוך השוואה עם טכניקות אינטרפולצייה אחרות כמו:

- אינטרפולצייה לפי לאגראנז׳ (Lagrange)
- אינטרפולצייה לפי `numpy.polyfit`