Metody Obliczeniowe w Nauce i Technice

Opracowanie pytań egzaminacyjnych

Na podstawie uaktualnionych notatek Morgula, wykładu dr Rycerz oraz źródeł różnych

Spis treści

1. Arytmetyka zmiennoprzecinkowa

1.1. Omów reprezentację zmiennoprzecinkową

- F - zbiór liczb zmiennoprzecinkowych (floating-point)

- β - podstawa
- t - dokładność
- L, U - zakres wykładnika
- e - cecha, taka że $L \leq e \leq U$
- d_i - liczby całkowite, gdzie $0 \leq d_i \leq \beta - 1, \quad i = 1, ..., t$

$$x = \pm \underbrace{\left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_t}{\beta^t} \right)}_{\text{mantysa}} \cdot \beta^e = \pm \sum_{i=1}^{t} \frac{d_i}{\beta^i} \cdot \beta^e \tag{1.1}$$

System liczb zmiennoprzecinkowych F jest unormowany, gdy:

$$\forall_{x \neq 0} \; d_1 \neq 0 \tag{1.2}$$

1.2. Wyprowadź wzór na liczbę elementów zbioru liczb zmiennoprzecinkowych

W przeciwieństwie do liczb rzeczywistych, F nie jest kontinuum - posiada skończoną liczbę elementów w ~~ilości~~ **liczbie**:

$$\#F = 2 \cdot (\beta - 1) \cdot \beta^{t-1} \cdot (U - L + 1) + 1 \tag{1.3}$$

Objaśnienie:
- $2 \rightarrow$ znak liczby $\pm$
- $(\beta - 1) \rightarrow$ liczba możliwości na pierwszym bicie mantysy (o jeden mniej, bo **unormowanie - nie ma zera**)
- $\beta^{t-1} \rightarrow$ liczba możliwości na pozostałych $t - 1$ bitach
- $(U - L + 1) \rightarrow$ zakres wykładnika
- $+1 \rightarrow$ reprezentacja zera

1.3. Wyprowadź wzór na błąd względny reprezentacji zmiennoprzecinkowej dla systemu dwójkowego

Przyjmijmy oznaczenia:
- s - znak
- c - cecha
- m - mantysa
- e_i - wartość i-tego bitu, $\quad e_i \in \{0, 1\}$
- $fl(x)$ - reprezentacja liczby zmiennoprzecinkowej x

Mantysa o nieskończonej dokładności:

$$m = \sum_{i=1}^{\infty} e_i \cdot 2^{-i} \tag{1.4}$$

Mantysa o skończonej dokładności t:

$$m_t = \underbrace{\sum_{i=1}^{t} e_i \cdot 2^{-i}}_{t\text{-bitowa mantysa}} + \underbrace{e_{t+1} \cdot 2^{-t}}_{\substack{\text{zaokrąglenie dodane do} \\ \text{ostatniego bitu reprezenracji}}} \tag{1.5}$$

Rozważmy dwa przypadki

- Liczba jest bliżej swojego zaokrąglenia w dół

  W tym przypadku pomijamy resztę bitów $(> t)$ i reprezentacja jest równa:

$$fl(x)^- = \pm \sum_{i=1}^{t} \frac{d_i}{\beta^i} \cdot \beta^e \tag{1.6}$$

  Najdalsza „końcówka” (część wychodząca poza dokładność) tej liczby ma formę:

$$0.\ e_1\ e_2\ \ldots\ e_t \underbrace{0}_{e_{t+1}} \underbrace{1}_{e_{t+2}} \underbrace{1}_{e_{t+3}} \ldots \underbrace{1}_{e_\infty} \tag{1.7}$$

  Wtedy:

$$m = \sum_{i=1}^{t} e_i \cdot 2^{-i} + 0 \cdot 2^{-(t+1)} + \sum_{i=(t+2)}^{\infty} 1 \cdot 2^{-i} = \sum_{i=1}^{t} e_i \cdot 2^{-i} + \underbrace{2^{-(t+1)}}_{\substack{\text{suma szeregu} \\ \text{potęgowego}}} \tag{1.8}$$

$$m_t = \sum_{i=1}^{t} e_i \cdot 2^{-i} + 0 \cdot 2^{-t} = \sum_{i=1}^{t} e_i \cdot 2^{-i} \tag{1.9}$$

$$m - m_t = 2^{-(t+1)} \tag{1.10}$$

- Liczba jest bliżej swojego zaokrąglenia w górę

W tym przypadku **dodajemy 1** do ostatniego bitu i reprezentacja jest równa:

$$fl(x)^+ = \pm \left(\sum_{i=1}^{t} \frac{d_i}{\beta^i} + \frac{1}{\beta^t} \right) \cdot \beta^e \tag{1.11}$$

Najdalsza „końcówka” (część wychodząca poza dokładność) tej liczby ma formę:

$$0.\ e_1\ e_2\ \ldots\ e_t \underbrace{1}_{e_{t+1}} \underbrace{0}_{e_{t+2}} \underbrace{0}_{e_{t+3}} \ldots \underbrace{0}_{e_\infty} \tag{1.12}$$

Wtedy:

$$m = \sum_{i=1}^{t} e_i \cdot 2^{-i} + \underbrace{2^{-(t+1)}}_{e_{t+1}=1} \tag{1.13}$$

$$m_t = \sum_{i=1}^{t} e_i \cdot 2^{-i} + \underbrace{2^{-t}}_{\frac{1}{\beta^t}} \tag{1.14}$$

$$m_t - m = 2^{-t} - 2^{-(t+1)} = 2^{-t} - \frac{2^{-t}}{2} = \frac{2^{-t}}{2} = 2^{-(t+1)} \tag{1.15}$$

W obydwu przypadkach, maksymalny błąd bezwzględny reprezentacji to:

$$|m - m_t| = 2^{-(t+1)} \tag{1.16}$$

W przyjętej reprezentacji najmniejsze możliwe m to $\frac{1}{2}$, więc można szacować:

$$\left| \frac{m - m_t}{m} \right| \leq \frac{2^{-(t+1)}}{\frac{1}{2}} = 2^{-t} \tag{1.17}$$

1.4. Wyjaśnij pojęcia: nadmiar, niedomiar, błędy obcięcia

- **Nadmiar** - występuje, gdy liczba bitów potrzebna do reprezentacji cechy jest za mała i wartość bezwzględna liczby jest za duża, aby ją reprezentować
- **Niedomiar** - występuje, gdy liczba bitów potrzebna do reprezentacji cechy jest za mała i wartość bezwzględna liczby jest za mała, aby ją reprezentować
- **Błędy obcięcia** - występują, gdy powinniśmy wykonać nieskończony ciąg obliczeń, który w praktyce musi być skończony 😔.
  Przykłady:
  - ograniczenie szeregu nieskończonego do skończonej liczby składników
  - aproksymacja pochodnej za pomocą ilorazu różnicowego

1.5. Podaj definicję i napisz, co określa maszynowe epsilon?

Maszynowe epsilon ε to najmniejsza liczba zmiennoprzecinkowa, dla której zachodzi jeszcze:

$$1 \oplus \varepsilon > 1 \quad (\oplus - \text{dodawanie zmiennopozycyjne}) \tag{1.18}$$

Maszynowe epsilon to wartość określająca precyzję obliczeń numerycznych wykonywanych na liczbach zmiennoprzecinkowych.

1.6. Omów własności numerycznej reprezentacji liczb rzeczywistych i arytmetyki zmiennoprzecinkowej

- **Skończona precyzja reprezentacji liczb** - każda liczba jest reprezentowana na skończonej liczbie bitów, więc liczby od samego początku są obciążone błędem reprezentacji:

  $$fl(x) = x \cdot \left(1 + \varepsilon_{\text{maszynowe}}\right) \tag{1.19}$$

  Trzeba więc projektować algorytmy tak, aby były stabilne i nie kumulowały błędów reprezentacji.
- **Nie można porównywać wartości liczb bezpośrednio** - sprawdzanie równości przez == prawie nigdy nie da prawidłowego wyniku, więc w celu sprawdzenia równości należy przyjąć niewielki błąd δ o który liczby mogą się różnić.
- **Przesuwanie przecinka podczas operacji** - podczas operacji na liczbach skrajnie różnej wielkości, wykładniki liczb muszą zostać dopasowane, co jest uzyskiwane przez przesunięcie przecinka w stronę większej z liczb. Powoduje to utratę informacji o ostatnich bitach mniejszej z liczb. Na przykład:
  - $A = 1.000 \cdot 2^{10}$
  - $C = 1.000 \cdot 2^{0}$
  - podczas operacji $A + C$ wykładniki są dopasowywane, a więc mantysa C przesuwana jest w prawo:

  $$C_{\text{przesunięte}} = 0.0000000001 \cdot 2^{10} \tag{1.20}$$

  Z tego też powodu algorytmy trzeba projektować tak, żeby wykonywać działania na liczbach o zbliżonym w miarę możliwości rzędzie wielkości.
- **Zakres reprezentowanych liczb jest skończony** - nie da się reprezentować liczb z nieskończoną dokładnością (min. przez epsilon maszynowy i określoną dokładność reprezentacji t), przez co występują przepełnienia `overlfow` i `underflow`. Trzeba odpowiednio konstruować algorytmy, by unikać przepełnień przez manipulację kolejnością obliczeń.

- **Brak łączności i rozdzielności działań** - w przeciwieństwie do zwykłej arytmetyki działania są przemienne, ale nie łączne ani rozdzielne (min. przez występowanie błędów reprezentacji, `overflow`, `underflow`, itd.), więc nie można w algorytmach opierać się na tych właściwościach.

1.7. Podaj definicje i objaśnij na przykładach pojęcia: zadanie, algorytm, realizacja zmiennoprzecinkowa algorytmu.

- **Zadanie**

Niech będą dane dane:

$$\vec{d} = (d_1, d_2, ..., d_n) \in R_d \tag{1.21}$$

oraz wynik:

$$\vec{w} = (w_1, w_2, ..., w_n) \in R_w \tag{1.22}$$

,gdzie R_d i R_w to skończenie wymiarowe, unormowane przestrzenie kartezjańskie

Celem zadania dla danych $\vec{d}$ jest znaleźć wynik postaci $\vec{w}$, który jest odwzorowaniem danych:

$$\begin{gathered} \vec{w} = \varphi\left(\vec{d}\right) \\ \varphi : D_0 \subset R_d \to R_w \\ D - \text{zbiór danych wejściowych algorytmu} \\ D_0 - \text{dziedzina funkcji } \varphi \end{gathered} \tag{1.23}$$

Przykład: Zadania posortowania danego zbioru liczb.

- **Algorytm**

Algorytmem nazywamy sposób wyznaczenia wyniku zadania w postaci $\vec{w} = \varphi\left(\vec{d}\right)$, gdzie zadanie należy do klasy zadań:

$$\{\varphi, D\}, \quad d \in D \subset D_0 \tag{1.24}$$

i rozwiązanie zadania jest dokładne (w zwykłej arytmetyce).

Przykład: Algorytm quicksort pozwalający wyznaczyć posortowany zbiór liczb na podstawie danego.

- **Realizacja zmiennoprzecinkowa algorytmu**

Realizacja zmiennoprzecinkowa algorytmu to sposób realizacji algorytmu A w postaci:

$$fl\left(A\left(\vec{d}\right)\right) \tag{1.25}$$

Podczas realizacji następuje zastąpienie arytmetyki zwykłej na zmiennoprzecinkową oraz:

$$\begin{gathered} d \to rd(d) \\ x \to rd(x) \\ \dots \end{gathered} \tag{1.26}$$

, gdzie rd oznacza arytmetykę maszynową, komputerową reprezentacje rzeczywistej liczby.

Oczekuje się, że dane i wyniki będą prezentowane z minimalnym błędem, tzn. błąd względny danych i wyniku będą co najwyżej rzędu:

$$k \cdot \beta^{1-t} \tag{1.27}$$

- β - podstawa systemu
- k - niewielka stała ≈ 10
- t - dokładność reprezentacji
- $1 - t$ - względna dokładność

Przykład: Algorytm quicksort operujący na liczbach zmiennoprzecinkowych z użyciem arytmetyki zmiennoprzecinkowej (ma wpływ na jego działanie, np. przy porównywaniu liczb).

1.8. Podaj definicje i objaśnij na przykładach pojęcia: uwarunkowanie zadania, poprawność numeryczna algorytmu, stabilność numeryczna algorytmu.

- **Uwarunkowanie zadania**

Uwarunkowaniem nazywamy **czułość zadania na zaburzenie danych wejściowych**. Charakteryzuje je wskaźnik uwarunkowania zadania, mówiący jak niewielkie zaburzenie danych wejściowych wpłynie na zaburzenie wyników. Oznaczany:

$$\mathrm{cond}(\varphi(x)) \tag{1.28}$$

Jeśli dane znamy z błędem względnym nie większym niż ε to błąd względny wyniku obliczenia nie jest większy niż $\varepsilon \cdot \mathrm{cond}(\varphi(x))$

Zadanie jest źle uwarunkowane, jeśli niewielka zmiana danych wejściowych powoduje dużą zmianę danych wyjściowych. Jest to spowodowane użyciem arytmetyki zmiennoprzecinkowej i zastąpieniem danych d_i przez $rd(d_i) = d_i \cdot (1 + \varepsilon_i)$

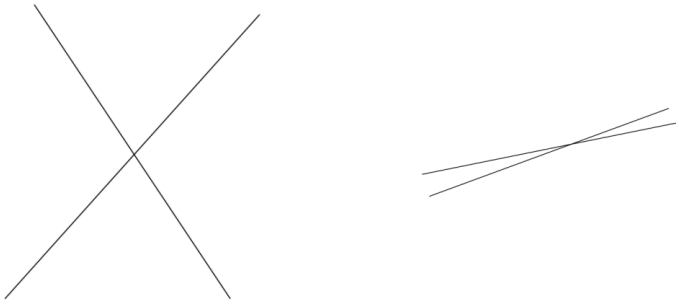
Można je opisać ilościowo, np. przez wzór uwarunkowania względnego dla funkcji $f(x)$:

$$\mathrm{cond}(x) = \frac{x \cdot f'(x)}{f(x)} \tag{1.29}$$

$$\begin{gathered} \mathrm{cond}\left(\sqrt{x}\right) = \left|\frac{x \cdot \frac{1}{2\sqrt{x}}}{\sqrt{x}}\right| = \frac{1}{2} \qquad \mathrm{cond}\left(\frac{1}{1-x}\right) = \frac{\left|x \cdot \frac{1}{(1-x)^2}\right|}{\left|\frac{1}{1-x}\right|} = \frac{x}{1-x} \ (*) \\ \text{dla } (*) \text{ i } \ \tilde{x} = 1 + 10^{-6} \Rightarrow \mathrm{cond}\left(\frac{1}{1-\tilde{x}}\right) \approx 10^6 \quad (!) \end{gathered} \tag{1.30}$$

oraz jakościowo, np. przez obrazek:

Układ dwóch równań graficznie:
niezależnie od jakości ołówka (algorytm) - lewa strona - dokładniej.



well conditioned ill conditioned

- **Poprawność numeryczna algorytmu**

Algorytm nazywamy numerycznie poprawnym, gdy dla nieco zaburzonych danych (na poziomie reprezentacji, rzędu błędu reprezentacji) dają tylko nieco zaburzone rozwiązania - są to algorytmy numeryczne najwyższej jakości.

Przyjmijmy, że δ_d, δ_w to maksymalny błąd reprezentacji odpowiednio danych i wyniku, a $\tilde{d}$ to zestaw danych zaburzonych. Algorytm jest numerycznie poprawny w klasie zadań $\{\varphi, D\}$, jeżeli istnieją takie stałe (wskaźniki kumulacji algorytmu) K_d, K_w, że:

- $\forall \vec{d} \in D$ (każdy zestaw poprawnych danych należy do D)
- dla każdej dostatecznie silnej arytmetyki β^{1-t} (oznaczenia jak powyżej), $\exists \tilde{d} \in D_0$ taki, że

$$\begin{gathered} \left\| \frac{\vec{d} - \tilde{d}}{\vec{d}} \right\| \leq \delta_d \cdot K_d \\ \left\| \frac{\varphi(\tilde{d}) - fl\left(A\left(\vec{d}\right)\right)}{\varphi(\tilde{d})} \right\| \leq \delta_w \cdot K_w \end{gathered} \tag{1.31}$$

, gdzie $\varphi(\tilde{d})$ to dokładne rozwiązanie dla danych zaburzonych, a powyższe wzory to zależności dla danych i wyniku.

Z pierwszego wzoru wynika, że błąd względny między danymi zaburzonymi a oryginalnymi jest co najwyżej równy iloczynowi maksymalnego błędu reprezentacji danych δ_d oraz wskaźnika kumulacji algorytmu dla danych K_d.

Drugi wzór - analogiczny, ale dla wyniku.

Wskaźniki kumulacji K_d, K_w:

- mają być prawdziwe dla dowolnych danych z klasy zadań $\{\varphi, D\}$,
- im mniejsze, tym lepszy algorytm (bo tym dokładniejszy jest wtedy algorytm przy danych zaburzonych)

Przykłady algorytmów poprawnych numerycznie:

- algorytm Kahana, bo jego błąd wynosi $O(1)$ i jest na poziomie reprezentacji
- rozkład LU z partial pivotingiem (np. w eliminacji Gaussa), bo minimalizuje amplifikację błędów przez wybór maksymalnego elementu w kolumnie jako pivotu, co redukuje współczynniki mnożników i kontroluje błędy zaokrągleń

- **Stabilność numeryczna algorytmu**

Algorytm nazywamy numerycznie stabilnym, gdy mały błąd na dowolnym etapie przenosi się dalej z malejącą amplitudą, czyli jakość wyniku poprawia się z każdym kolejnym etapem obliczeń.

$$\begin{gathered} \varepsilon^{n+1} = g \cdot \varepsilon^n \Rightarrow \text{stabilna: } \left|\varepsilon^{n+1}\right| \leq |\varepsilon^n| \\ g - \text{współczynnik wzmocnienia} \end{gathered} \tag{1.32}$$

Stabilność jest słabszym warunkiem od poprawności i minimalnym wymogiem dla algorytmu. W praktyce - badamy jak duży byłby błąd wyniku, gdyby dane oraz wynik zostały zaburzone na poziomie reprezentacji, ale same obliczenia byłyby wykonywane dokładnie.

Niech:

- $\vec{w} = \varphi\left(\vec{d}\right)$ - dokładne rozwiązanie dla dokładnych danych (idealne)

- $\hat{d}: \left\|\vec{d}-\hat{d}\right\| \leq \rho_d \left\|\vec{d}\right\|$ - dane zaburzone na poziomie reprezentacji
- $\hat{w} = \varphi\left(\hat{d}\right)$ - dokładne rozwiązanie dla zaburzonych danych
- $\tilde{w}: \|\hat{w}-\tilde{w}\| \leq \rho_w \|\hat{w}\|$ - zaburzone (na poziomie reprezentacji) rozwiązanie dla zaburzonych danych

Optymalny poziom błędu rozwiązania zdefiniowany jest jako:

$$P\left(\vec{d},\varphi\right) = \underbrace{\rho_w\|\vec{w}\|}_{\substack{\text{błąd reprezentacji}\\ \text{wyniku}}} + \underbrace{\max_{\hat{d}}\left\|\varphi\left(\vec{d}\right)-\varphi\left(\hat{d}\right)\right\|}_{\substack{\text{wrażliwość na zaburzenia}\\ \text{danych}}} \tag{1.33}$$

Algorytm A nazywamy **numerycznie stabilnym** w klasie $\{\varphi, D\}$, jeżeli istnieje stała K taka, że $\forall \vec{d} \in D$ i dla każdej dostatecznie silnej arytmetyki zachodzi:

$$\left\|\varphi\left(\vec{d}\right) - fl\left(A\left(\vec{d}\right)\right)\right\| \leq \underbrace{K}_{\text{małe}} \cdot P\left(\vec{d},\varphi\right) \tag{1.34}$$

Algorytm numerycznie stabilny gwarantuje uzyskanie rozwiązania z błędem co najwyżej K razy większym, niż optymalny poziom błędu rozwiązania tego zadania.

1.9. Wyznacz wskaźnik uwarunkowania podanego zadania

Cytując z Morgula

Niezależnie od zadania korzysta się z metod podanych w przykładach do definicji wskaźnika uwarunkowania.

Wskazówki:

- zadanie sprowadza się do rozdzielenia części z samymi x, y i z samymi α, β i tym samym znalezienia $\text{cond}(x,y)$
- zaczyna się zwykle od użycia któregoś ze wzorów: na błąd bezwzględny $\left(\left|\frac{f^*-f}{f}\right|\right)$, na współczynnik uwarunkowania funkcji $\left(\left|\frac{x\cdot f'}{f}\right|\right)$
- aby dostać wartość przybliżoną (z błędem), trzeba za każdy x podstawić $x\cdot(1+\alpha)$, za każdy y podstawić $y\cdot(1+\beta)$ itd.
- trzeba używać operacji $\leq$ oraz $\approx$ i dowolnych poprawnych przekształceń matematycznych

Przykład:

Zadanie:

$$\vec{x}\cdot\vec{y} = \sum_{i=1}^{n} x_i \cdot y_i \neq 0 \tag{1.35}$$

Wprowadźmy teraz błąd danych:

$$\begin{cases} x_i \to x_i\cdot(1+\alpha_i) \\ y_i \to y_i\cdot(1+\beta_i) \end{cases} \tag{1.36}$$

Posłużmy się także błędem względnym $\left|\frac{f*-f}{f}\right|$:

$$\left| \frac{\overbrace{\sum_{i=1}^{n} x_i \cdot (1+\alpha_i) \cdot y_i \cdot (1+\beta_i)}^{\text{dla danych zaburzonych}} - \overbrace{\sum_{i=1}^{n} x_i \cdot y_i}^{\text{dla danych niezaburzonych}}}{\sum_{i=1}^{n} x_i \cdot y_i} \right| \approx$$
$$\approx \left| \frac{\sum_{i=1}^{n} x_i \cdot y_i \cdot (\alpha_i + \beta_i)}{\sum_{i=1}^{n} x_i \cdot y_i} \right| \leq \max|\alpha_i + \beta_i| \cdot \frac{\sum_{i=1}^{n} |x_i \cdot y_i|}{\left|\sum_{i=1}^{n} x_i \cdot y_i\right|} \tag{1.37}$$

Stąd, naszym wskaźnikiem uwarunkowania jest:

$$\text{cond}(\vec{x} \cdot \vec{y}) = \frac{\sum_{i=1}^{n} |x_i y_i|}{\left|\sum_{i=1}^{n} x_i y_i\right|} \tag{1.38}$$

Jeżeli wszystkie iloczyny $x_i y_i$ są tego samego znaku, to

$$\frac{\sum_{i=1}^{n} |x_i y_i|}{\left|\sum_{i=1}^{n} x_i y_i\right|} = \frac{\sum_{i=1}^{n} |x_i y_i|}{\sum_{i=1}^{n} |x_i y_i|} = 1 \tag{1.39}$$

I w takiej sytuacji $\text{cond}(\vec{x} \cdot \vec{y}) = 1$.

1.10. Wyznacz wskaźnik kumulacji podanego algorytmu

Cytując z Morgula

Niezależnie od zadania korzysta się z metod podanych w przykładzie do definicji wskaźnika kumulacji.

Wskazówki:

- dla danych oryginalnych x i zaburzonych $\hat{x}$ trzeba obliczyć wyniki dokładne i fl
- δ to zwykle po prostu precyzja arytmetyki, czyli β^{1-t}
- mając błędy względne dla danych i wyników oraz δ wystarczy podstawić do wzoru i znaleźć współczynniki K_d i K_w

Przykład:

Numeryczna poprawnośc algorytmu $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i \cdot b_i$

```
A(a,b):
  s := 0;
  for i:=1 to n do:
    s := s + a[i] * b[i];
```

Realizacja algorytmu $fl\left(A\left(\vec{a}, \vec{b}\right)\right)$:

- dane - reprezentacje:

$$\begin{cases} a_i \to \hat{a}_i = rd(a_i) = a_i \cdot (1+\alpha_i) \\ b_i \to \hat{b}_i = rd(b_i) = b_i \cdot (1+\beta_i) \end{cases} \tag{1.40}$$

- działania - przybliżone, fl
  - ε_i - błąd mnożenia
  - δ_i - błąd dodawania
  - np. dla $i = 1, 2, 3$:

$$fl\big(A\big(\vec{a},\vec{b}\big)\big) = \Big\{\big[\hat{a_1}\cdot\hat{b_1}\cdot(1+\varepsilon_1)+\hat{a_2}\cdot\hat{b_2}\cdot(1+\varepsilon_2)\big]\cdot(1+\delta_2)+\hat{a_3}\cdot\hat{b_3}\cdot(1+\varepsilon_3)\Big\}\cdot(1+\delta_3) \tag{1.41}$$

$$\delta_1 = 0 \tag{1.42}$$

W ogólności:

$$\begin{aligned} fl\big(A\big(\vec{a},\vec{b}\big)\big) &= \sum_{i=1}^{n} a_i\cdot(1+\alpha_i)\cdot b_i\cdot(1+\beta_i)\cdot(1+\varepsilon_i)\cdot\prod_{j=i}^{n}\big(1+\delta_j\big) = \\ &= \sum_{i=1}^{n}\hat{a_i}\cdot\hat{b_i}\cdot(1+\varepsilon_i)\cdot\prod_{j=i}^{n}\big(1+\delta_j\big) \end{aligned} \tag{1.43}$$

- istnieją takie $\tilde{a}$, $\tilde{b}$ dla których da się wskazać K_d i K_w. Na przykład:
  - dla $\tilde{a}$ takiego, że $\underset{(a)_i}{\sim} = a_i\cdot(1+\alpha_i)$
  - dla $\tilde{b}$ takiego, że $\underset{(b)_i}{\sim} = b_i\cdot(1+\beta_i)\cdot(1+\varepsilon_i)\cdot\prod_{j=i}^{n}(1+\delta_i)$
  - mamy dokładny wynik: $K_w = 0$
  - dla zaburzonych danych $\tilde{a}$ i $\tilde{b}$ spełniających warunki:
    - $\|\vec{a}-\tilde{a}\| \le \beta^{1-t}\cdot\|\vec{a}\| \to k_{d_1} = 1$
    - $\big\|\vec{b}-\tilde{b}\big\| \le (n+1)\cdot\beta^{1-t}\cdot\big\|\vec{b}\big\| \to k_{d_2} = n+1$

  Tutaj skutki błędów zaokrągleń interpretujemy jako skutki takiego zaburzenia danych, że otrzymany wynik jest dla tych zaburzonych danych dokładny.

  Małe zaburzenia oznaczają, że algorytm jest poprawny numerycznie.

2. Interpolacja

2.1. Podaj i uzasadnij wzór na interpolację *Lagrange'a*

W interpolacji Lagrange'a szukamy wielomianu $P_n(x)$ przechodzącego przez $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$

- $x_0,x_1,...,x_n$ - punkty przez które ma przechodzić wielomian
- $f(x_k)$ - wartości funkcji w tych punktach (współczynniki w bazie Lagrange'a)
- $L_k(x_l)$ - Baza lagrange'a, wielomiany bazowe powiązane z jednym węzłem x_k

$$L_k(x_l) = \delta_{k,l} = \begin{cases} 0, & k \neq l\ (*) \\ 1, & k = l\ (**) \end{cases} \text{ dla } k\in\{0,1,...,n\} \tag{2.1}$$

$(*)$ - licznik d, iloczyn wsyzstkich $(x-x_i)$ oprócz $(x-x_k)$

$$d = (x-x_0)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n) \tag{2.2}$$

$(**)$ - mianownik m, wartośc licznika d obliczona w $x=x_k$ (normalizacja aby $L_k(x_k)=1$)

$$m = (x_k-x_0)(x_k-x_1)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n) \tag{2.3}$$

$$L_k(x) = \frac{d}{m} = \prod_{i=0,i\neq k}^{n}\frac{x-x_i}{x_k-x_i} \tag{2.4}$$

$$P_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x) \tag{2.5}$$

W skrócie, tworzymy wielomian dla każdego x_k taki, że we wszystkich węzłach $x_j \neq x_k$ będzie przyjmował wartość $L_k(x_j) = 0$, a w węźle dedykowanym $L_k(x_k) = 1$. Stąd, powyższa forma tych wielomianów, które później do siebie dodajemy.

2.2. Podaj dowód na jednoznaczność interpolacji wielomianowej

Teza: Jeżeli $P_n(x)$ to wielomian stopnia $\leq n$, przechodzący przez punkty:

$$(x_i, y_i), \quad i = 0, 1, 2, ..., n, \;\; x_i \neq x_j \tag{2.6}$$

, to P_n **jest jedynym takim wielomianem**.

Dowód:

Niech $\exists Q_n(x) \neq P_n(x)$, przechodzący przez wyżej wymienione punkty.

Ustalmy:

$$R_n(x) = P_n(x) - Q_n(x) \tag{2.7}$$

Ponieważ P_n oraz Q_n są sobie równe dla $x = x_i$, to:

$$R_n(x_i) = 0, \;\; i = 0, 1, ..., n \tag{2.8}$$

$$\#i = n + 1 \;\; (\text{jest } n+1 \text{ wartości } i) \tag{2.9}$$

Stąd, jeśli wielomian stopnia $\leq n$ ma $n + 1$ miejsc zerowych, to musi zachodzić $R_n(x) \equiv 0$.

2.3. Wyprowadź wzór na błąd interpolacji metodą Lagrange'a

Błąd interpolacji Lagrange'a

$$f(x) = P_n(x) + \frac{f^{(n+1)}(\eta)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{2.10}$$

W podanym wyżej wzorze:

1. Dla węzłów interpolacji $x = x_k$, $k = 0, 1, ..., n$:

$$f(x_k) = P_n(x_k) \Rightarrow \text{dowolny } \eta \text{ spełnia wzór} \tag{2.11}$$

2. Dla ustalonego $\tilde{x} \neq x_k$, $k = 0, 1, ..., n$ definiujemy ✨magiczną✨ funkcję $g(t), [a, b]$:

$$g(t) = f(t) - P_n(t) - [f(\tilde{x}) - P_n(\tilde{x})] \prod_{i=0}^{n} \frac{t - x_i}{\tilde{x} - x_i} \tag{2.12}$$

$$g(t) \in C^{(n+1)}[a, b] \tag{2.13}$$

Mając $g(t)$ rozważamy dwa przypadki:

1. dla $t = x_k$

$$g(x_k) = f(x_k) - P_n(x_k) - [f(\tilde{x}) - P_n(\tilde{x})] \prod_{i=0}^{n} \frac{x_k - x_i}{\tilde{x} - x_i} = 0 \tag{2.14}$$

2. dla $t = \tilde{x}$, z założenia $\tilde{x} \neq x_k$

$$g(\tilde{x}) = f(\tilde{x}) - P_n(\tilde{x}) - [f(\tilde{x}) - P_n(\tilde{x})] \prod_{i=0}^{n} \frac{\tilde{x} - x_i}{\tilde{x} - x_i} = 0 \tag{2.15}$$

Z powyższego wynika, że $g(t)$ ma $(n+2)$ miejsc zerowych: w węzłach interpolacji $x_0, x_1, ..., x_n$ oraz $\tilde{x}$ wybranego przy definicji g

Do wyprowadzenia wzory na błąd interpolacji będzie potrzebne uogólnione Tw. Rolle'a:

Założenia:

1. $f \in C[a,b]$
2. $f \in C^n(a,b)$
3. $f = 0$ w $(n+1)$ różnych punktach

Teza:

$$\exists c \in (a,b) : f^{(n)}(c) = 0 \tag{2.16}$$

Z powyższego twierdzenia:

$$0 = g^{(n+1)}(\eta) = f^{(n+1)}(\eta) - P_n^{(n+1)}(\eta) - \underbrace{[f(\tilde{x}) - P_n(\tilde{x})]}_{\text{wartość stała}} \cdot \frac{d^{n+1}}{dt^{n+1}} \left\{ \prod_{i=0}^{n} \frac{t - x_i}{\tilde{x} - x_i} \right\}_{t=n} \tag{2.17}$$

gdzie

$$\frac{d^{n+1}}{dt^{n+1}} \left\{ \prod_{i=0}^{n} \frac{t - x_i}{\tilde{x} - x_i} \right\}_{t=n} = \frac{d^{n+1}}{dt^{n+1}} \left\{ \frac{t^{n+1}}{\prod_{i=0}^{n}(\tilde{x} - x_i)} + a \cdot t^n + ... \right\}_{t=\eta} \tag{2.18}$$

$$P_n^{(n+1)}(\eta) = 0 \rightarrow (n+1) \text{ pochodna wielomianu stopnia } n \tag{2.19}$$

stąd

$$f^{(n+1)}(\eta) = [f(\tilde{x}) - P_n(\tilde{x})] \frac{(n+1)!}{\prod_{i=0}^{n}(\tilde{x} - x_i)} \tag{2.20}$$

$$\frac{f^{(n+1)}(\eta)}{(n+1)!} \prod_{i=0}^{n}(\tilde{x} - x_i) = [f(\tilde{x}) - P_n(\tilde{x})] \tag{2.21}$$

$$f(\tilde{x}) = P_n(\tilde{x}) + \frac{f^{(n+1)}(\eta)}{(n+1)!} \prod_{i=0}^{n}(\tilde{x} - x_i) \tag{2.22}$$

gdzie $\tilde{x}$ jest dowolnym $x \neq x_i$

2.4. Ilorazy różnicowe: podaj definicję i objaśnij ich związek z pochodnymi

Iloraz różnicowy to wielkość opisująca przyrost funkcji na danym przedziale $[a,b]$ postaci $\frac{f(b)-f(a)}{b-a}$.

Jest to wartość średnia funkcji f na tym przedziale (z tw. o wartości średniej). Przyjmując przedział długości $h = b - a$, otrzymuje się $\frac{f(x+h)-f(x)}{h}$, co przy $h \to 0$ daje definicję pochodnej:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.23}$$

Można także zdefiniować ilorazy różnicowe wyższych rzędów, przydatne w analizie numerycznej np. do obliczenia wielomianu interpolacyjnego Newtona metodą tablicy ilorazów różnicowych (algorytm Neville'a). Korzysta się wtedy z definicji rekurencyjnej dla danej tablicy zawierającej x_i i y_i :

$$\begin{cases} f[x_i] = y_i \\ f\left[x_i, x_{i+1}, ..., x_{i+j}\right] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+j}] - f[x_i, x_{i+1}, ..., x_{i+j-1}]}{x_{i+j} - x_i} \end{cases} \tag{2.24}$$

x_0	$f(x_0)$				
x_1	$f(x_1)$	$f[x_0, x_1]$			
x_2	$f(x_2)$	$f[x_1, x_2]$	$f[x_0, x_1, x_2]$		
...	...	...	...	...	...
x_n	$f(x_n)$	$f[x_{n-1}, x_n]$	...	...	$f[x_0, \ldots, x_n]$

Korzystając ponownie z tw. o wartości średniej można pokazać związek między powyższą definicją ilorazów różnicowych a pochodną funkcji:

$$\exists \eta \in (a, b) : f[x_0, x_1, ..., x_n] = \frac{f^{(n)}(\eta)}{n!} \tag{2.25}$$

2.5. Uzasadnij użyteczność użycia ilorazów różnicowych w interpolacji

- można łatwo skonstruować tablicę do obliczania ilorazów
- dla dobranych węzłów istnieje dokładnie jeden wielomian interpolacyjny
- łatwość dodania kolejnego węzła poprzez rozszerzenie tablicy

2.6. Przeprowadź porównanie - interpolacja Lagrange'a i Newton'a

Do **interpolacji Lagrange'a** korzysta się z wielomianu wyliczanego ze wzoru

$$P_n(x) = \sum_{k=0}^{n} f(x_k) \prod_{i=0, i \neq k}^{n} \frac{x - x_i}{x_k - x_i} \tag{2.26}$$

W przypadku **interpolacji Newtona** obliczanie wielomianu przebiega z wykorzystaniem tablicy ilorazów różnicowych. Na podstawie przekątnej tej tablicy budujemy wielomian w postaci Newtona jako:

$$\begin{gathered} P_n(x) = f[x_0] + (x - x_0) f[x_0, x_1] + ... + (x - x_0)(x - x_1)...(x - x_{n-1}) f[x_0, x_1, ..., x_n] \\ P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, ...x_k] \cdot (x - x_0)...(x - x_{k-1}) \end{gathered} \tag{2.27}$$

Jeśli chcemy dodać nowy węzeł to w przypadku interpolacji Lagrange'a zmieni nam się wzór i musimy go policzyć od nowa. Dla interpolacji Newtona wystarczy dodać nowy wiersz w tablicy ilorazów różnicowych i wziąć wartość z przekątnej jako kolejny współczynnik wielomianu w postaci Newtona.

2.7. Przeprowadź porównanie - interpolacja Hermite'a i Newton'a

W **interpolacji Hermite'a** przeprowadzonej na $k+1$ węzłach $x_0, x_1, ..., x_k$, każdy o krotności węzłów m_i, gdzie:

$$\sum_{i=0}^{k} m_i = n + 1 \tag{2.28}$$

Funkcje f w metodzie Hermite'a interpolujemy wielomianem H_n stopnia $\leq n$ takim, że w danym punkcie jego pochodne są zgodne z pochodną funkcji. Krotność m_i mówi nam, ile pochodnych ma być równych:

$$H_n^{(j)}(x_i) = f^{(j)}(x_i) \quad i = 0, 1, ..., k \quad j = 0, 1, ..., m_i \tag{2.29}$$

W przypadku **interpolacji Newtona** obliczanie wielomianu przebiega z wykorzystaniem tablicy ilorazów różnicowych. Na podstawie przekątnej tej tablicy budujemy wielomian w postaci Newtona jako:

$$\begin{gathered} P_n(x) = f[x_0] + (x - x_0)f[x_0, x_1] + ... + (x - x_0)(x - x_1)...(x - x_{n-1})f[x_0, x_1, ..., x_n] \\ P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, ...x_k] \cdot (x - x_0)...(x - x_{k-1}) \end{gathered} \tag{2.30}$$

Porównanie:

- Interpolacja Newtona korzysta z wartości funkcji $y_i = f(x_i)$ dla danej tablicy $n + 1$ węzłów.

  Interpolacja Hermite'a zakłada dodatkowo wykorzystania wartości pochodnych w punktach (przy czym, jeśli dane jest $f^{(j)}(x_i)$ to muszą być też dane f', ..., $f^{(j-1)}$). W interpolacji Hermite'a mamy dane $k + 1$ węzłów $x_0, ..., x_k$. To ile pochodnych jest danych dla węzła x_i określa tak zwana krotność węzła m_i i zachodzi:

  $$\sum_{i=0}^{k} m_i = n + 1 \tag{2.31}$$

- Dla $m = 1$ interpolacja Hermite'a działa jak interpolacja Lagrange'a / Newtona (jedyny warunek to przechodzenie przez punkty (x_i, y_i))
- Tworzenie tablicy ilorazów różnicowych w metodzie Hermite'a przebiega tak samo jak w metodzie Newtona, z tą różnicą, że tam gdzie nie można utworzyć ilorazu, to wykorzystujemy informację o pochodnej $f^{(j)}(x_i)$.

2.8. Objaśnij efekt Rungego: jak się objawia, co jest jego przyczyną, jak można zapobiegać

Efekt Rungego polega na obniżeniu jakości interpolacji pomimo zwiększania liczby węzłów, tzn. początkowo wraz ze wzrostem liczby węzłów jakość interpolacji rośnie (błąd maleje), ale potem pogarsza się (szczególnie na końcach przedziału i dla wielomianów interpolujących wysokich stopni).

Przyczyny:
- interpolacja wielomianowa z równoczesnym nałożeniem warunku równoodległości węzłów,
- szybki wzrost wartości wyższych pochodnych.

Zapobieganie:
- wykorzystywać interpolację funkcjami sklejanymi (spline'ami),
- wybierać węzły gęściej przy granicach przedziału zamiast równoodlegle, np. poprzez wykorzystanie węzłów Czebyszewa (zera wielomianów Czebyszewa),
- zaczynać od interpolacji liniowej, a potem zwiększać liczbę punktów i stopień wielomianu dla ustabilizowania kluczowych miejsc.

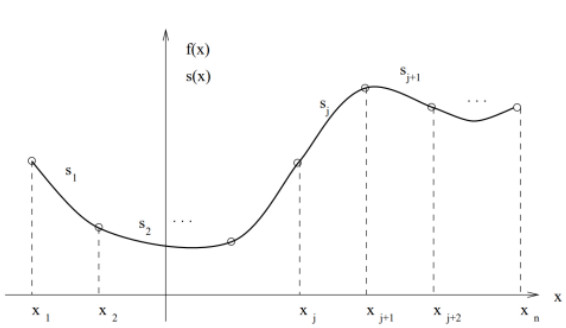
3. Spline

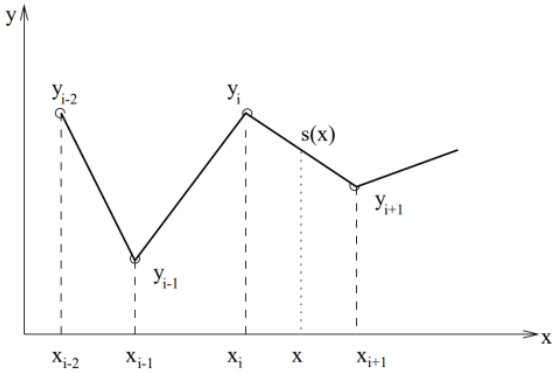
3.1. Podaj definicję funkcji sklejanej, objaśnij ją na odpowiednim rysunku, omów przydatność tych funkcji.

Funkcję $S(x) = S(x, \Delta n)$ określoną na $[a, b]$ nazywamy funkcją sklejaną stopnia m $(m \geq 1)$, jeżeli:
- $S(x)$ jest wielomianem stopnia $\leq m$ na każdym $[x_i, x_{i+1}]$
- $S(x) \in C^{m-1}[a, b]$

Δn - podział $[a, b]$ na $(n-1)$ podprzedziałów przez węzły:

$$a = x_1 < x_2 < ... < x_i < ... < x_n = b \tag{3.1}$$





Objaśnienie splajnów na podstawie rysunku:

- splajny mają różne rodzaje w zależności od tego, w jaki sposób konstruuje się wielomiany, np. splajn 0 i 1 stopnia, splajn kwadratowy (2 stopnia), splajn sześcienny (3 stopnia, cubic), naturalny splajn sześcienny
- stopień splajnu jest niezależny od liczby węzłów interpolacji
- dla każdego przedziału definiuje się inny wielomian według wzoru splajnu, ale wszystkie zachowują jego warunki
- zwykle używa się splajnów sześciennych (wymagają obliczenia współczynników dla podprzedziałów przez układ równań), w szczególności naturalnego splajnu sześciennego (natural cubic spline), gdyż jest on najgładszą funkcją interpolującą
- można stosować różne warunki brzegowe, aby kontrolować zachowanie i kształt funkcji na brzegach, np. $S''(x_1) = S''(x_n) = 0$ (natural cubic spline)

Przydatność splajnów:

- nie występuje efekt Rungego jak przy interpolacji wielomianowej, splajny są bezpieczne dla węzłów równoodległych
- łatwo konstruuje się splajny sześcienne - wystarczy rozwiązać układ równań, który można sprowadzić do symetrycznej, trójdiagonalnej macierzy
- w praktyce splajny sześcienne są wystarczające do interpolacji
- dobre do obliczania pochodnych i całek (bo mają proste wzory)
- można z ich pomocą wygładzać funkcje i powierzchnie
- B-splajny (basis splines) wykorzystuje się w grafice komputerowej

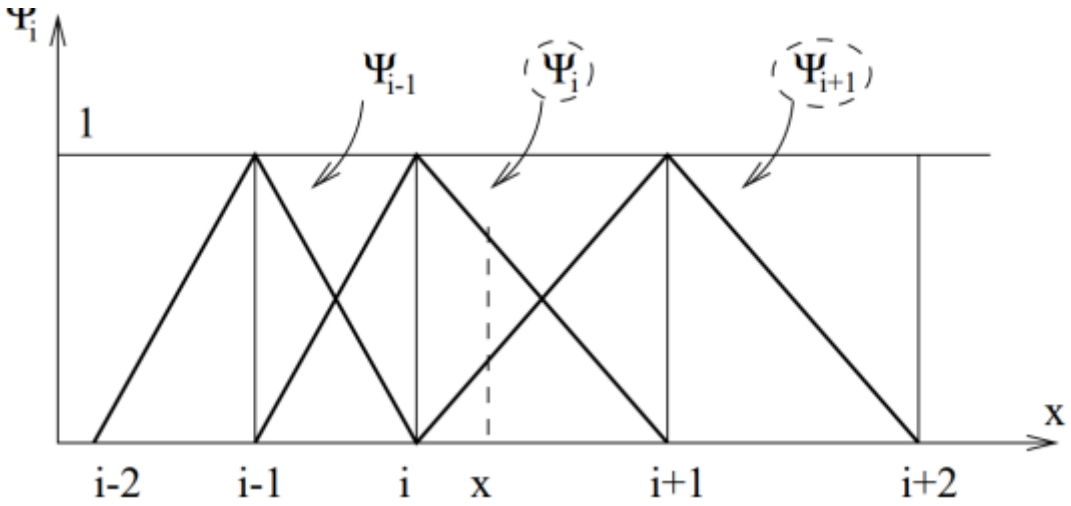
3.2. Porównaj interpolację wielomianami i funkcjami sklejanymi

Interpolującą funkcję sklejaną stopnia 1-go można zapisać:

$$S(x) = \sum_{i=0}^{n} y_i \psi_i(x) \tag{3.2}$$

gdzie $\psi_i(x) = 0, ..., n$ - baza przestrzeni liniowej funkcji sklejanych 1-go stopnia o funkcjach kształtu:

$$\Psi_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & x \in [x_i, x_{i+1}] \\ 0 & x \notin [x_{i-1}, x_{i+1}] \end{cases} \tag{3.3}$$



Różnice pomiędzy interpolacją wielomianową a splajnami:

- W interpolacji wielomianowej cały przedział przybliżamy jedną funkcją, a korzystając ze splajnów **dla każdego przedziału definiujemy inny wielomian zgodnie z zasadami dla funkcji sklejanych**
- Wielomian interpolacyjny jest jednoznaczny (jest tylko jeden wielomian o zadanych węzłach i wartościach), a **splajnów jest wiele dzięki warunkom brzegowym** (stopnie swobody), dzięki czemu poprzez ich dobór ma się wpływ na zachowanie funkcji przy końcach przedziału
- Dla wielomianów zwiększanie dokładności polega na zwiększaniu jego stopnia (ale może wystąpić efekt Rungego), a **dla splajnów wystarczające są splajny sześcienne**
- W przypadku interpolacji wielomianowej dla węzłów równoodległych występuje efekt Rungego (zmniejszenie precyzji interpolacji wraz ze wzrostem liczby węzłów), natomiast **przy interpolacji splajnami nie występuje** (**sposób rozmieszczenia węzłów nie ma znaczenia**)
- Interpolacja wielomianowa wymaga obliczenia współrzędnych wielomianu np. za pomocą tablicy ilorazów różnicowych lub wzoru, może też wymagać znajomości wartości pochodnych w węzłach interpolacji (interpolacja Hermite'a). Interpolacja splajnami wymaga **obliczenia współrzędnych wielomianów i np. dla naturalnych splajnów sześciennych wymaga rozwiązywania układów równań**, zwykle z symetrycznymi macierzami trójdiagonalnymi

3.3. Podaj i omów warunki brzegowe stosowane przy wyznaczaniu sześciennych funkcji sklejanych.

Sposoby wyznaczania warunków brzegowych:

- **Wykorzystując funkcje C**:

$$\begin{cases} C_1(x) - \text{funkcja sześcienna przez pierwsze 4 punkty} \\ C_{n(x)} - \text{ funkcja sześcienna przez ostatnie 4 punkty} \end{cases} \tag{3.4}$$

$$S'''(x_1) = C_1''' \quad S'''(x_n) = C_n''' \tag{3.5}$$

Stałe C_1''' i C_n''' mogą być określone bez znajomości $C_1(x)$ i $C_n(x)$.

- **natural cubic spline**

$$S''(x_1) = S''(x_n) = 0 \text{ (free boundary)} \tag{3.6}$$

- **clamped boundary** (pierwsze pochodne na krańcach są zanane bądź przybliżone ilorazami różnicowymi)

$$S'(x_1) = y_1', \; S'(x_n) = y_n' \tag{3.7}$$

- **drugie pochodne na krańcach** znane bądź przybliżone ilorazami różnicowymi (szczególny przypadek - natural cubic splines)

$$S''(x_1) = y_1'', \; S''(x_n) = y_n'' \tag{3.8}$$

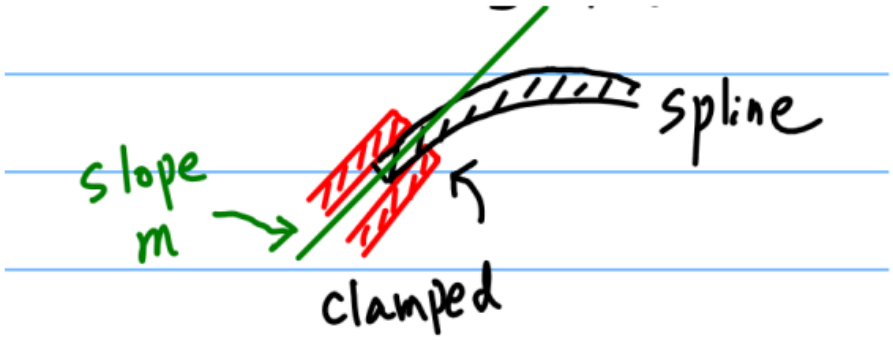
- **not-a-knot condition**

$$S_1'''(x_2) = S_2'''(x_2) \text{ oraz } S_{n-1}'''(x_{n-1}) = S_n'''(x_{n-1}) \tag{3.9}$$

$$\text{czyli } s'''(x) \text{ ciągła w } x_2 \;\; \text{i} \;\; x_{n-1} \tag{3.10}$$

- **interpolowanie spline'ami funkcji periodycznych**

$$S(x_1) = S(x_n), \; S'(x_1) = S'(x_n) \text{ oraz } S''(x_1) = S''(x_n) \tag{3.11}$$



źródło: http://runge.math.smu.edu/HiPerfSciComp/_downloads/CubicSpline.pdf

3.4. Podaj podstawowe kroki potrzebne do wyprowadzenia wzoru na kubiczne funkcje sklejane (3-stopnia)

Interpolacja sześcienna:

$$\begin{gathered} S_i \text{ na } [x_i, x_{i+1}] \to \text{cubic polynomial:} \\ S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \end{gathered} \tag{3.12}$$

- $S_i(x_{i+1}) = f(x_{i+1})$
- $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$
- $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$
- $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$

Wypisując powyższe warunki dla każdego z „wewnętrznych" punktów interpolacji i dodając warunki brzegowe, można stworzyć układ równań i go rozwiązać.

3.5. ~~Wyprowadź wzór na kubiczne funkcje sklejane (3-stopnia)~~

Zagadnienie nie obowiązuje na zerówce 2024/2025

Szukamy takiej funkcji S, która w danych węzłach t_i ma dane wartości y_i oraz w każdym przedziale $[t_i, t_{i+1}]$ jest **wielomianem stopnia co najwyżej 3** ($1 \leq i \leq n-1$).

Warunek $S_{i-1}(t_i) = y_i = S_i(t_i)$, czyli wartości w węzłach sąsiednich wielomianów **zapewniają ciągłość** funkcji S i **daje nam $2n$ warunków** (po 2 na każdy wielomian za jego brzegi).

Warunek ciągłości pierwszej pochodnej S', czyli $S'_{i-1}(t_i) = S'_i(t_i)$ daje $n-1$ warunków, tyle samo daje warunek ciągłości drugiej pochodnej.

Wielomiany mają **razem $4n$ współczynników**, powyższe **warunki dają $4n-2$ warunków**. Brakujące 2 to **stopnie swobody**, które można pozyskać na różne sposoby. Wprowadźmy oznaczenia i skorzystajmy z liniowości funkcji S''.

$$z_i = S''_i(t_i) \quad z_{i+1} = S''_i(t_{i+1}) \quad h_i = t_{i+1} - t_i \tag{3.13}$$

$$S''_i(x) = \frac{z_i}{h_i}(t_{i+1} - x) + \frac{z_{i+1}}{h_i}(x - t_i) \tag{3.14}$$

Po scałkowaniu dwukrotnie drugiej pochodnej otrzymujemy wzór na $S_i(x)$, ale z **nieznanymi współczynnikami**:

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + C_i(x - t_i) + D_i(t_{i+1} - x) \tag{3.15}$$

Stałe C_i i D_i otrzymuje się, wykorzystując warunki interpolacyjne $S_i(t_i) = y_i$, $S_i(t_{i+1}) = y_{i+1}$. Wynika z nich, że:

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x - t_i) + \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1} - x) \tag{3.16}$$

Należy jeszcze wyznaczyć $z_i = S''_i(t_i)$, co można zrobić korzystając z warunków $S'_{i-1}(t_i) = S'_i(t_i)$. Różniczkujemy najpierw powyższy wielomian, a potem podstawiamy $x = t_i$, otrzymując prawą stronę tej równości:

$$S'_i(t_i) = -\frac{h_i}{6}z_{i+1} - \frac{h_i}{3}z_i + \frac{y_{i+1}}{h_i} - \frac{y_i}{h_i} \tag{3.17}$$

Aby otrzymać lewą stronę równości, wystarczy w obliczonej pochodnej podstawić zamienić wskaźnik i na $i-1$ i podstawić $x = t_i$:

$$S'_{i-1}(t_i) = \frac{h_{i-1}}{3}z_i + \frac{h_{i-1}}{6}z_{i-1} - \frac{y_i}{h_{i-1}} + \frac{y_{i-1}}{h_{i-1}} \tag{3.18}$$

Przyrównując do siebie powyższe wyrażenia otrzymujemy:

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}) \tag{3.19}$$

Jest to **układ $n-1$ równań z $n+1$ niewiadomymi** $z_0, z_1, ..., z_n$. Wybór konkretnych z_0 i z_n daje rodzaj funkcji sklejanej, po przyjęciu tych warunków brzegowych (np. dla naturalnej funkcji sześciennej $z_0 = z_n = 0$ trzeba rozwiązać układ równań).

3.6. Podaj definicje B-splines. Omów ich przydatność.

Funkcje **B-splines** (basis splines):

- stosowane w grafice komputerowej do modelowania figur o skomplikowanych kształtach
- bazują na fakcie, że funkcje sklejane można wyrazić za pomocą kombinacji liniowej funkcji bazowych

- takie funkcje bazowe nazywamy funkcjami B-sklejanymi (B-splines)
- dla danego zestawu węzłów interpolacji - funkcje bazowe łatwo wyliczyć rekurencyjnie
- algorytmy o dobrych własnościach numerycznych

Wprowadźmy oznaczenia:

- j - indeks przedziału, na którym zdefiniowana jest funkcja B-sklejana
- k - stopień funkcji B-sklejanej

Wtedy $B_{j,k}(x)$ - B-spline rzędu k zdefiniowane jako:

- $B_{j,0} = \begin{cases} 1, & x_j \leq x \leq x_{j+1} \\ 0, & \text{poza przedziałem} \end{cases}$
- wyższe rzędy ($k > 0$) - rekurencyjnie:

$$B_{j,k}(x) = \frac{x - x_j}{x_{j+k} - x_j} B_{j,k-1}(x) + \frac{x_{j+k+1} - x}{x_{j+k+1} - x_{j+1}} B_{j+1,k-1}(x) \tag{3.20}$$

- np. $B_{j,1} = \Psi_j$ (interpolacja liniową funkcją sklejaną)

Reprezentacja funkcji sklejanej stopnia k:

$$S(x) = \sum_j p_j B_{j,k}(x) \tag{3.21}$$

gdzie p_j - współczynniki (w grafice komputerowej są to tzw. zadane punkty kontrolne)

Własności:

- $B_{j,k}(x) > 0, x \in \left[x_j, x_{j+k+1}\right]$
- $B_{j,k}(x) = 0, x \notin \left[x_j, x_{j+k+1}\right]$
- w przedziale $\left[x_j, x_{j+1}\right]$ istotne jest tylko $k+1$ funkcji: $B_{j-k,k}(x) \cdot \ldots \cdot B_{j,k}(x) \neq 0$
- normalizacja:

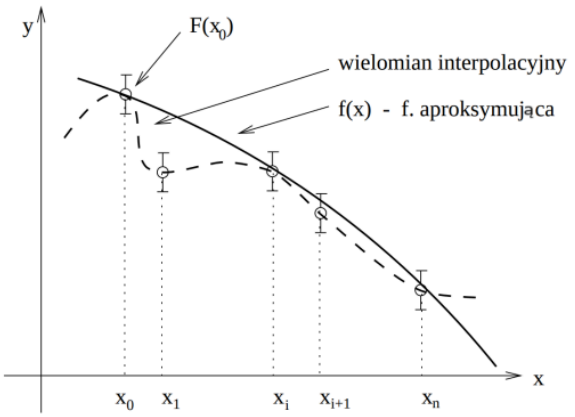
$$\sum_j B_{j,k}(x) = \sum_{j=l-k}^{l} B_{j,k}(x) = 1 \quad \text{dla } x_l \leq x \leq x_{l+1} \tag{3.22}$$

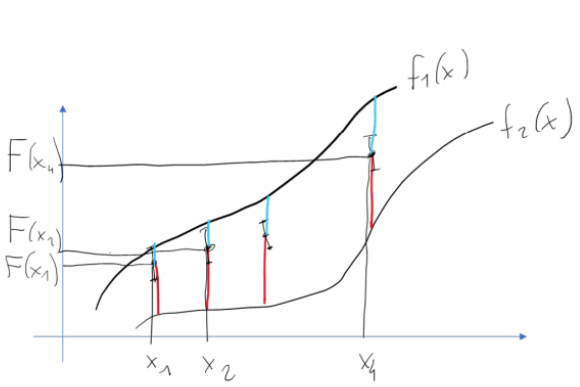
4. Aproksymacja

4.1. Podaj definicję zadania aproksymacji, objaśnij ją na odpowiednim rysunku, omów jej przydatność

Aproksymacja to **przybliżanie lub zastępowanie funkcji za pomocą innej funkcji**, jest ogólniejsza niż interpolacja.

- $F(x)$ - funkcja aproksymowana, która **może być znana** lub **podana jako tablica wartości eksperymentalnych** (z błędami, ale wtedy interpolacja nie ma sensu)
- $f(x)$ - funkcja aproksymująca - przybliżenie $F(x)$





W problemie aproksymacji dane jest:

- $(x_i, y_i = F(x_i)),\ i = 0, 1, ..., n$, czyli mamy $(n+1)$ węzłów
- układ funkcji bazowych: $\varphi_j(x),\ j = 0, 1, ..., m$

Niech $w(x) : w(x) > 0$ będzie funkcją wagową. Zwykle: $w(x_i) \sim \frac{1}{[\text{błąd } F(x)]^2}$ lub $w(x) = 1$

Szukamy wielomianu uogólnionego:

$$f(x) = \sum_{j=0}^{m} a_j \varphi_j(x) \tag{4.1}$$

czyli $\{a_j\}_{j=0}^{m}$ dla których:

$$\min \|F(x) - f(x)\| = \min \underbrace{\sum_{i=0}^{n} w(x_i) \left[F(x_i) - \sum_{j=0}^{m} a_j \varphi_j(x_i) \right]^2}_{H(a_0, a_1, ... a_m) \rightarrow \text{ funkcja kosztu}} \tag{4.2}$$

Dla przypadku ciągłego minimalizujemy wartość całki:

$$\min \int_a^b w(x)[F(x) - f(x)]^2 dx \tag{4.3}$$

Współczynniki $\{a_j\}$ znajdujemy z warunku (szukamy minimum funkcji kosztu $H(a_0, ..., a_m)$):

$$\frac{\partial H}{\partial a_k} = 0, k = 0, 1, ..., m \tag{4.4}$$

Otrzymujemy w ten sposób do rozwiązania układ $(m+1)$ równań liniowych o $(m+1)$ niewiadomych:

$$\frac{\partial H}{\partial a_k} = \frac{\partial}{\partial a_k} \sum_{i=0}^{n} w(x_i) \left[F(x_i) - \sum_{j=0}^{m} a_j \varphi_j(x_i) \right]^2 \tag{4.5}$$

$$\frac{\partial H}{\partial a_k} = -2 \cdot \sum_{i=0}^{n} w(x_i) \left[F(x_i) - \sum_{j=0}^{m} a_j \varphi_j(x_i) \right] \varphi_k(x_i) = 0 \qquad k = 0, 1, ..., m \tag{4.6}$$

Powyższy układ jest układem normalnym.

4.2. Wyprowadź wzór na aproksymację średniokwadratową jednomianami

Dane:

- $n+1$ punktów (x_i, y_i), gdzie $i = 0, 1, 2, ..., n$ oraz $y_i = F(x_i)$
- układ funkcji bazowych $\varphi_j(x) = x^j, j = 0, 1, 2, ..., m$

Szukamy funkcji postaci:

$$f(x) = \sum_{j=0}^{m} a_j x^j \tag{4.7}$$

Wyznaczamy współczynniki a_j. Wyznaczane one są tak, aby dawały najmniejsza różnice względem faktycznej wartości funkcji f w podanych punktach w normie średniokwadratowej. Określamy zbiór $x_i, i = 0, 1, 2, ..., n$, na którym określona jest $f(x_i)$ i szukamy:

$$\min \sum_{i=0}^{n} w(x_i)[F(x_i) - f(x_i)]^2 \tag{4.8}$$

, gdzie $w(x_i)$ jest funkcją wagową pozwalającą nadać różne znaczenie różnym punktom.

Bazując na powyższym tworzymy tzw. normalny układ równań, z którego otrzymamy współczynniki a_i:

$$\sum_{i=0}^{n} w(x_i) \left[F(x_i) - \sum_{j=0}^{m} a_j x_i^j \right] x_i^k = 0 \qquad k = 0, 1, 2, ..., m \tag{4.9}$$

$$\sum_{i=0}^{n} w(x_i) x_i^k \sum_{j=0}^{m} a_j x_i^j = \sum_{i=0}^{n} w(x_i) F(x_i) x_i^k \qquad k = 0, 1, 2, ..., m \tag{4.10}$$

$$\sum_{j=0}^{m} \underbrace{\left(\sum_{i=0}^{n} w(x_i) x_i^{j+k} \right)}_{G_{k,j}} a_j = \underbrace{\sum_{i=0}^{n} w(x_i) F(x_i) x_i^k}_{B_k} \tag{4.11}$$

gdzie:

$$G_{k,j} = \sum_{i=0}^{n} w(x_i) x_i^{j+k} \qquad B_k = \sum_{i=0}^{n} w(x_i) F(x_i) x_i^k \tag{4.12}$$

stąd:

$$\sum_{j=0}^{m} G_{k,j} a_j = B_k \tag{4.13}$$

Układ ten można zapisać w postaci macierzowej. Rozwiązując go można otrzymać kolejne współczynniki a_i. Ukad ten ma jedno rozwiązanie, jeśli $x_0, x_1, x_2, ..., x_n$ są różne i $m \leq n$.

$$\mathrm{Ga} = \mathrm{B} \tag{4.14}$$

$$\begin{bmatrix} \sum w_i & \sum w_i x_i & \sum w_i x_i^2 & \dots & \sum w_i x_i^m \\ \sum w_i x_i & \sum w_i x_i^2 & \sum w_i x_i^3 & \dots & \sum w_i x_i^{m+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum w_i x_i^m & \sum w_i x_i^{m+1} & \sum w_i x_i^{m+2} & \dots & \sum w_i x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum w_i F_i \\ \sum w_i F_i x_i \\ \vdots \\ \sum w_i F_i x_i^m \end{bmatrix} \tag{4.15}$$

4.3. Wyprowadź wzór na aproksymację średniokwadratową wielomianami ortogonalnymi

Dane:

- $n+1$ punktów (x_i, y_i), gdzie $i = 0, 1, 2, ...n$ oraz $y_i = F(x_i)$
- układ funkcji bazowych $\varphi_j, j = 0, 1, 2, ..., m$ będących wielomianami ortogonalnymi (np. Czebyszewa lub Legendre'a - zwłaszcza dla $w(x_i) = 1$)

Szukamy funkcji postaci:

$$f(x) = a_i \varphi_j(x) \tag{4.16}$$

Wyznaczamy współczynniki a_j. Wyznaczane one są tak, aby dawały najmniejsza różnice względem faktycznej wartości funkcji f w podanych punktach w normie średniokwadratowej. Określamy zbiór $x_i, i = 0, 1, 2, ..., n$, na którym określona jest $f(x_i)$ i szukamy:

$$\min \sum_{i=0}^{m} w(x_i)[F(x_i) - f(x_i)]^2 \tag{4.17}$$

, gdzie $w(x_i)$ jest funkcją wagową pozwalającą nadać różne znaczenie różnym punktom.

Bazując na powyższym tworzymy tzw. normalny układ równań, z których otrzymamy współczynniki a_i:

$$\sum_{i=0}^{n} w(x_i) \left[F(x_i) - \sum_{j=0}^{m} a_j \varphi_j(x_i) \right] \varphi_k(x_i) = 0 \qquad k = 0, 1, ..., m \tag{4.18}$$

$$\sum_{i=0}^{n} w(x_i) \varphi_k(x_i) \sum_{j=0}^{m} a_j \varphi_j(x_i) = \sum_{i=0}^{n} w(x_i) F(x_i) \varphi_k(x_i) \tag{4.19}$$

$$\sum_{j=0}^{m} a_j \sum_{i=0}^{n} w(x_i) \varphi_k(x_i) \varphi_j(x_i) = \sum_{i=0}^{n} w(x_i) F(x_i) \varphi_k(x_i) \tag{4.20}$$

Dzięki temu, że funkcji φ_i **są względem siebie ortogonalne** (prostopadłe), mamy właściwość, która upraszcza powyższy wzór:

$$\begin{aligned} &\sum_{i=0}^{n} w(x_i) \varphi_k(x_i) \varphi_j(x_i) = 0, && \text{jeśli } j \neq k \\ &\sum_{i=0}^{n} w(x_i) \varphi_k(x_i) \varphi_j(x_i) = \sum_{i=0}^{n} w(x_i) \varphi_k^2(x_i) \neq 0, && \text{jeśli } j = k \end{aligned} \tag{4.21}$$

, stąd:

$$a_k \sum_{i=0}^{n} w(x_i) \varphi_k^2(x_i) = \sum_{i=0}^{n} w(x_i) F(x_i) \varphi_k(x_i) \tag{4.22}$$

Powyższy układ równań z własności wielomianów ortogonalnych ma macierz diagonalną (bo dla $j \neq k$ $a_i = 0$)

4.4. Opisz podstawowe metody aproksymacji jednostajnej

Aproksymacja jednostajna to proces przybliżania funkcji za pomocą innej funkcji (np. wielomianu, funkcji trygonometrycznych), w którym **kryterium jakości przybliżenia jest minimalizacja normy jednostajnej (normy supremum, normy Czebyszewa)** różnicy między funkcją aproksymowaną a aproksymującą.

Dla funkcji aproksymowanej $F(x)$ określonej na $[a,b]$ szukamy $f(x)$ takiego, że:

$$\min\|F(x)-f(x)\| = \min \sup_{x\in[a,b]} |F(x)-f(x)| \tag{4.23}$$

Metoda Szeregów Potęgowych

Metoda Szeregów Potęgowych polega na aproksymacji jednostajnej funkcji za pomocą obliczania sum częściowych szeregu Taylora:

$$W_n(x) = \sum_{k=0}^{n} \frac{F^{(k)}(x_0)}{k!}(x-x_0)^k \tag{4.24}$$

Oszacowanie błędu aproksymacji jest równie reszcie Lagrange'a:

$$F(x)-W_n(x) = \frac{F^{n+1}(\eta)}{(n+1)!}(x-x_0)^{n+1}, \eta\in[a,b] \tag{4.25}$$

Aproksymacja Padé (rational function approximation) - Opis w Sekcja 4.5

Aproksymacja wielomianami Czebyszewa - od Sekcja 4.8

Aproksymacja wykorzystuje sumy częściowe postaci

$$F(x) \approx \sum_{j=0}^{n} c_j T_j(x) \tag{4.26}$$

, gdzie $T_i(x)$ to wielomiany Czebyszewa.

Stałe c_j oblicza się z warunku ortogonalności dla przypadku ciągłego:

$$c_0 = \frac{1}{\pi}\int_{-1}^{1} F(x)\frac{T_{i(x)}}{\sqrt{1-x^2}}dx \tag{4.27}$$

$$c_i = \frac{2}{\pi}\int_{-1}^{1} F(x)\frac{T_{i(x)}}{\sqrt{1-x^2}}dx \quad i=1,2,...,n \tag{4.28}$$

Aproksymacja wielomianami Czebyszewa wykorzystuje też często algorytm Clenshaw'a, który pozwala łatwo i szybko obliczać te kombinacje liniowe.

4.5. Objaśnij na czym polega aproksymacja Pade. Podaj i omów kroki prowadzące do wyznaczania tej aproksymacji.

Aproksymacja funkcji $F(x)$ za pomocą funkcji wymiernej $r(x)$, gdzie funkcja wymierna $r(x)$ stopnia $N=n+m$ ma postać:

$$r(x) = \frac{P_{n(x)}}{Q_{m(x)}} = \frac{p_0+p_1x+...+p_nx^n}{q_0+q_1x+...+q_mx^m} \tag{4.29}$$

Cechy $r(x)$:

- nieredukowalna (p,q są względnie pierwsze, brak wspólnych dzielników)
- określona w $x=0 \Rightarrow q_0\neq 0 \Rightarrow q_0=1$ (zwykle)
- do określenia $N+1=n+m+1$ współczynników

Celem aproksymacji jest minimalizacja $\|F(x)-r(x)\|$, przy czym:

$$F(x) - r(x) = F(x) - \frac{P_n(x)}{Q_m(x)} = \frac{F(x) \cdot Q_m(x) - P_n(x)}{Q_m(x)} \tag{4.30}$$

Funkcję $F(x)$ można przedstawić za pomocą szeregu Maclaurina, zaś wielomiany P_n oraz Q_m jako sumy. Technika aproksymacji sprowadza się do odpowiedniego doboru współczynników $\{p_i, q_j\}$, $i = 0, ..., n$ $j = 0, ..., m$ oraz rozwiązania układu równań liniowych w celu znalezienia współczynników a_i.

$$F(x) = \sum_{i=0}^{\infty} a_i x^i \quad Q_m(x) = \sum_{i=0}^{m} q_i x^i \quad P_n(x) = \sum_{i=0}^{n} p_i x^i \tag{4.31}$$

$$F(x) - r(x) = \frac{\sum_{i=0}^{\infty} a_i x^i \cdot \sum_{i=0}^{m} q_i x^i - \sum_{i=0}^{n} p_i x^i}{Q_m(x)} \tag{4.32}$$

W metodzie tej, należy tak dobrać $p_0, p_1, ..., p_n$ oraz $q_0, q_1, ...q_b$, aby:

$$F^{(k)}(0) - r^{(k)}(0) = 0, \text{ dla } k = 0, 1, ..., N \quad (*) \tag{4.33}$$

, czyli aby rozwinięcia $f(x)$ i $r(x)$ w szereg Maclaurina były jak najbardziej zgodne, więc możliwie jak najwięcej pochodnych $f(x)$ i $r(x)$ powinno być równych w $x = 0$.

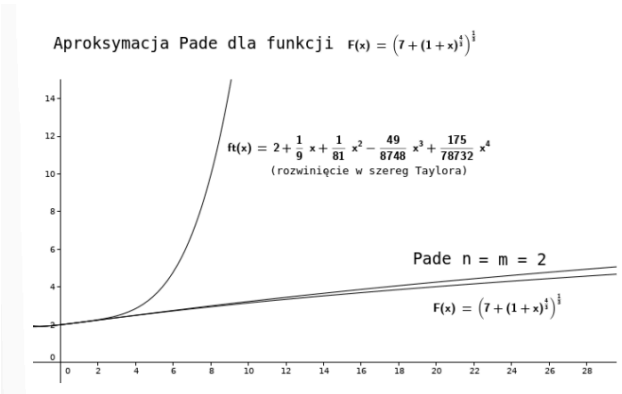
Dla uproszczenia zapisu, wyrównajmy liczbę obydwu współczynników p i q do N dopełniając je zerami :

$$p_{n+1} = p_{n+2} = ... = p_N = 0 \qquad q_{m+1} = q_{m+2} = ... = q_N = 0 \tag{4.34}$$

Przez $(*)$ można wnioskować, że licznik $F(x) - r(x)$ nie powinien mieć wyrazów stopnia $\leq N$. Jeśli ten warunek jest spełniony, to współczynniki $\sum_{i=0}^{k} a_i q_{k-i} - p_k$ są równe 0, z czego wynika, że uzyskanie funkcji aproksymującej sprowadza się do rozwiązania układu równań liniowych:

$$\begin{gathered} F(x) - r(x) = 0 \\ F(x) - \frac{P_n(x)}{Q_m(x)} = 0 \\ F(x)Q_m(x) - P_n(x) = 0 \\ \text{z obserwacji o współczynnikach:} \\ \sum_{i=0}^{k} a_i q_{k-i} - p_k = 0, \quad k = 0, 1, ..., N \end{gathered} \tag{4.35}$$

Aproksymacja Padé często daje lepszą aproksymację niż skończone sumy przybliżające szereg Taylora i może też działać tam, gdzie szereg Taylora nie jest zbieżny.



4.6. Przedstaw podstawowe własności wielomianów Czebyszewa

Wielomiany Czebyszewa to szczególna rodzina wielomianów ortogonalnych, które mają szerokie zastosowanie w analizie numerycznej, przybliżaniu funkcji, teorii aproksymacji.

Mogą być one reprezentowane w dwojaki sposób:

- **trygonometrycznie**

$$T_n(x) = \cos(n \cdot \arccos(x)), \quad x \in [-1, 1],\ n = 0, 1, 2, ... \tag{4.36}$$

- jako **relacja rekurencyjna**:

  $T_0(x) = 1$

  $T_1(x) = x$

  $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x), \quad n \geq 2$

Właściwości:

- mają czynnik wiodący w reprezentacji rekurencyjnej (czynnik przy najwyższej potędze x) równy 2^{n-1} dla $n \geq 1$
- symetria: $T_n(-x) = (-1^n) \cdot T_n(x)$, co znaczy, że jeśli x jest pierwiastkiem, to $-x$ także
- wielomiany stopnia parzystego są funkcjami parzystymi, a stopnia nieparzystego - nieparzystymi
- wszystkie ekstrema mają wartości -1 lub 1
- miejsca zerowe to **węzły Czebyszewa**, przy czym wielomian $T_n(x)$ ma w $[-1, 1]$ dokładnie n miejsc zerowych postaci:

$$x_k = \cos\left(\frac{2k+1}{n} \cdot \frac{\pi}{2}\right) \quad k = 0, 1, 2, ..., n-1 \tag{4.37}$$

- są **ortogonalne** dla przypadków ciągłego i dyskretnego (całka i suma):

$$\begin{gathered}\int_{-1}^{1} \frac{T_i(x) \cdot T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0, & i \neq j \\ \frac{\pi}{2}, & i = j \neq 0 \\ \pi, & i = j = 0 \end{cases} \\ \sum_{k=0}^{m} T_i(x_k) \cdot T_j(x_k) = \begin{cases} 0, & i \neq j \\ \frac{m+1}{2}, & i = j \neq 0 \\ m+1, & i = j = 0 \end{cases}\end{gathered} \tag{4.38}$$

- mają własność minimaksu, tzn. ze wszystkich wielomianów stopnia n ($n \geq 1$) z czynnikiem wiodącym (czynnik przy zmiennej o najwyższej potędze) równym 1, najmniejszą normę maksymalną na $[-1, 1]$:

$$\begin{gathered}\|W_n\|_\infty = \max_{x \in [a,b]} |W_n| \\ (\text{w naszym przypadku } [a, b] = [-1, 1])\end{gathered} \tag{4.39}$$

  ma wielomian:

$$\frac{1}{2^{n-1}} \cdot T_n(x) = 2^{1-n} \cdot T_n(x) \tag{4.40}$$

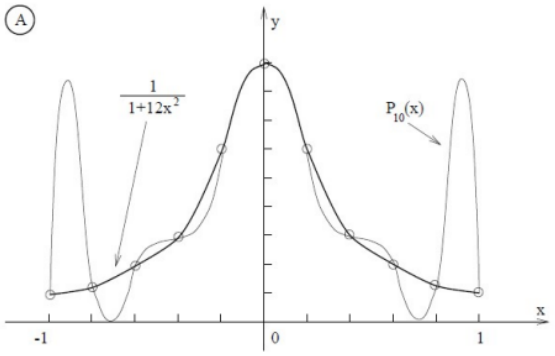
  , gdzie norma ta na $[-1, 1]$, wynosi 2^{1-n}.

Zastosowania:

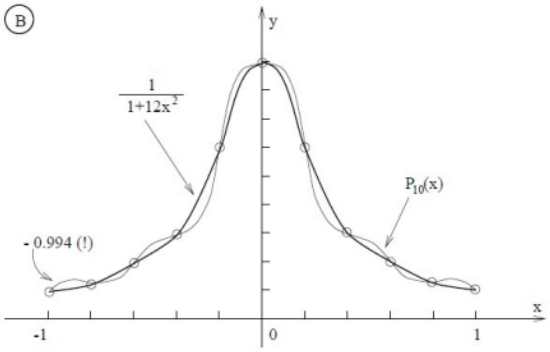
- dobór węzłów interpolacji - wybór zer (miejsc zerowych) wielomianów Czebyszewa (węzłów Czebyszewa) na węzły interpolacji zamiast węzłów równoodległych uodparnia na efekt Rungego, min. przez fakt, że zera zagęszczają się przy końcach przedziału

UWAGA
Aby móc użyć zer wielomianu Czebyszewa do interpolacji, należy przetransformować przedział interpolacji:

$$[a,b] \to [-1,1]$$
$$x = \frac{b-a}{2}t + \frac{a+b}{2} \tag{4.41}$$



Rysunek 10: Interpolacja zerami równoodległymi ($n = 11$)



Rysunek 11: Interpolacja zerami czebyszewa (T_{11})

- tworzy się z nich wielomian interpolujący Czebyszewa (szereg Czebyszewa) postaci:

$$W_n(x) = \sum_{j=0}^{n} c_j T_j(x) \tag{4.42}$$

- najlepiej ze wszystkich wielomianów przybliża się nimi zero na danym przedziale (przy czym każdy przedział da się znormalizować do wymaganego $[-1,1]$)

4.7. Udowodnij własność minimaksu wielomianów Czebyszewa

Teza: Dla wielomianu $p(x)$ stopnia n o współczynniku wiodącym 1:

- 1. $\|p\|_{[-1,1]} \geq 2^{1-n}$
- 2. znak równości osiągany jest dla $p(x) = 2^{1-n} \cdot T_n(x)$

Dowód nie wprost:

Załóżmy, że $\exists p_n(x)$ o współczynniku wiodącym = 1, takie, że

$$\forall_{x\in[-1,1]} |p_n(x)| < 2^{1-n} \tag{4.43}$$

Możemy zdefiniować sobie punkty ekstremalne, czyli takie, dla których $T_n(x)$ przyjmuje swoje ekstrema - $|T_n(x)| = 1$

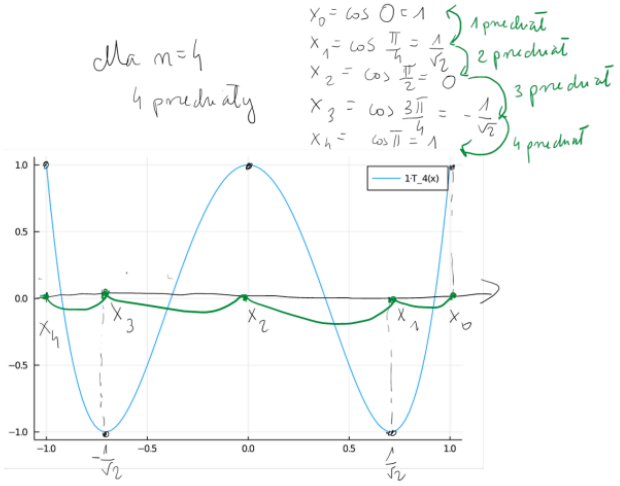
x'_k jest punktem ekstremalnym, jeśli $T_n(x'_k) = (-1)^k$.

Analogicznie, jak dla przypadku miejsc zerowych, zachodzi:

$$x'_k = \cos\frac{k\pi}{n}, \quad k = 0, 1, ..., n \tag{4.44}$$

Dla $\forall x'_k$ powinno więc zachodzić:

$$\begin{gathered} p_n(x'_0) < 2^{1-n} \cdot T_n(x'_0) \\ p_n(x'_1) > 2^{1-n} \cdot T_n(x'_1) \\ p_n(x'_2) < 2^{1-n} \cdot T_n(x'_2) \\ \dots \\ \text{aż do } x'_n \end{gathered} \tag{4.45}$$



Powyższe implikuje, że wielomian

$$[p_n(x) - 2^{1-n} \cdot T_n(x)] \tag{4.46}$$

powinien zmieniać znak w każdym z przedziałów:

$$\begin{gathered} (x'_{k+1}, x'_k), \quad k = n-1, n-2, \dots, 1, 0 \\ \text{n przedziałów} \rightarrow \text{n zer} \end{gathered} \tag{4.47}$$

czyli powinien być wielomianem stopnia n w $[-1, 1]$. Wiadomo jednak, że $p_n(x)$ i $2^{1-n} \cdot T_n(x)$ mają ten sam współczynnik wiodący. Jeżeli mają ten sam współczynnik wiodący, oznacza to, że po odjęciu dwóch wielomianów, różnica wyrazów o największej potędze x^n będzie wynosić 0. Zatem różnica wielomianów:

$$p_n(x) - 2^{1-n} \cdot T_n(x) = \text{wielomian stopnia } (n-1) \tag{4.48}$$

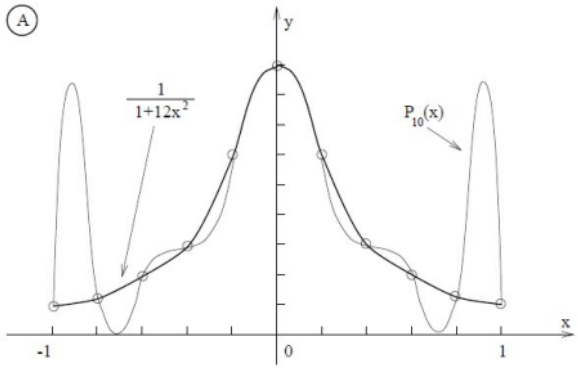
jest stopnia $n-1$ **co daje sprzeczność**.

4.8. Omów zastosowanie wielomianów Czebyszewa do interpolacji

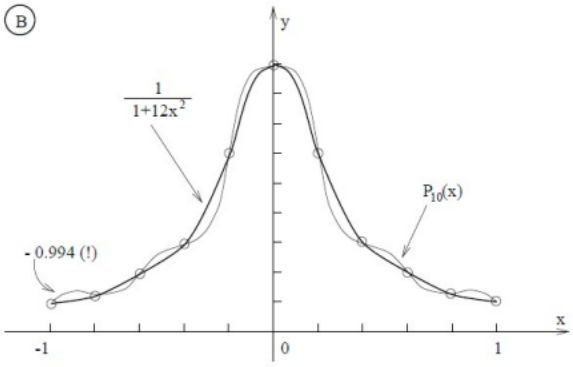
- dobór węzłów interpolacji - wybór zer (miejsc zerowych) wielomianów Czebyszewa (węzłów Czebyszewa) na węzły interpolacji zamiast węzłów równoodległych uodparnia na efekt Rungego, min. przez fakt, że zera zagęszczają się przy końcach przedziału

  UWAGA
  Aby móc użyć zer wielomianu Czebyszewa do interpolacji, należy przetransformować przedział interpolacji:

$$\begin{gathered} [a, b] \rightarrow [-1, 1] \\ x = \frac{b-a}{2} t + \frac{a+b}{2} \end{gathered} \tag{4.49}$$



Rysunek 13: Interpolacja zerami równoodległymi ($n = 11$)



Rysunek 14: Interpolacja zerami czebyszewa (T_{11})

- tworzy się z nich **wielomian interpolujący Czebyszewa** (szereg Czebyszewa) postaci:

$$W_n(x) = \sum_{j=0}^{n} c_j T_j(x) \tag{4.50}$$

Wyznaczanie wielomianu interpolującego Czebyszewa

Współczynniki c_j wyznaczamy z **własności ortogonalności** dla przypadku dyskretnego:

Jeśli

$$x_k = \cos\left(\frac{2k+1}{n+1}\frac{\pi}{2}\right) \text{ są zerami } T_{n+1}(x), \quad k = 0, 1, ..., n \tag{4.51}$$

, to zachodzi:

$$\sum_{k=0}^{n} T_i(x_k) T_j(x_k) = \begin{cases} 0, & i \neq j \\ \frac{n+1}{2}, & i = j \neq 0 \\ n+1, & i = j = 0 \end{cases} \tag{4.52}$$

Korzystamy następnie z warunku interpolacji:

$$f(x_k) = \sum_{j=0}^{n} c_j T_j(x_k) \tag{4.53}$$

Mnożymy obustronnie przez $T_i(x_k)$ dla każdego węzła i składamy w sumę $\sum_{k=0}^{n}$:

$$\sum_{k=0}^{n} f(x_k) T_{i(x_k)} = \sum_{j=0}^{n} c_j \sum_{k=0}^{n} T_i(x_k) T_j(x_k) \tag{4.54}$$

Dzięki ortogonalności, znikają nam defacto wszystkie składniki $\sum_{j=0}^{n}$ poza $j = i$, więc mamy:

$$\sum_{k=0}^{n} f(x_k) T_{i(x_k)} = c_i \sum_{k=0}^{n} T_i(x_k) T_j(x_k) \tag{4.55}$$

Dla przypomnienia:

$$\sum_{k=0}^{n} T_i(x_k)T_j(x_k) = \begin{cases} 0, & i \neq j \\ \frac{n+1}{2}, & i = j \neq 0 \\ n+1, & i = j = 0 \end{cases} \tag{4.56}$$

, z czego ostatecznie, po podzieleniu obustronnie przez sumę $\sum_{k=0}^{n} T_i(x_k)T_j(x_k)$:

$$\begin{aligned} c_0 &= \frac{1}{n+1}\sum_{k=0}^{n} f(x_k)T_i(x_k) \\ c_i &= \frac{2}{n+1}\sum_{k=0}^{n} f(x_k)T_i(x_k) \quad i = 1, ..., n \end{aligned} \tag{4.57}$$

Z praktycznych faktów - współczynniki są od siebie całkowicie niezależne. Jeżeli obliczymy mniej współczynników niż jest węzłów interpolacji, otrzymamy aproksymację funkcjami ortogonalnymi (Sekcja 4.3):

$$\sum_{i=0}^{n} w(x_i)F(x_i)\varphi_k(x_i) = a_k \sum_{i=0}^{n} w(x_i)\varphi_k^2(x_i) \tag{4.58}$$

4.9. Omów zastosowania wielomianów Czebyszewa do aproksymacji

Aproksymacja wykorzystuje sumy częściowe postaci

$$F(x) \approx \sum_{j=0}^{n} c_j T_j(x) \tag{4.59}$$

, gdzie $T_i(x)$ to wielomiany Czebyszewa.

Stałe c_j oblicza się z warunku ortogonalności dla przypadku ciągłego:

$$c_0 = \frac{1}{\pi}\int_{-1}^{1} F(x)\frac{T_{i(x)}}{\sqrt{1-x^2}}dx \tag{4.60}$$

$$c_i = \frac{2}{\pi}\int_{-1}^{1} F(x)\frac{T_{i(x)}}{\sqrt{1-x^2}}dx \quad i = 1, 2, ..., n \tag{4.61}$$

Innym sposobem aproksymacji wielomianami Czebyszewa jest metoda analogiczna do aproksymacji Padé.

Tworzymy wyrażenia wymierne postaci:

$$T_{n,k}(x) = \frac{\sum_{i=0}^{n} a_i T_i(x)}{\sum_{i=0}^{n} b_i T_i(x)} \tag{4.62}$$

o a_i i b_i dobranych tak, aby licznik wyrażenia:

$$F(x) - T_{n,k}(x) = \frac{\left[\sum_{j=0}^{\infty} c_j T_j(x)\right] \cdot \left[\sum_{i=0}^{k} b_i T_i(x)\right] - \sum_{i=0}^{n} a_i T_i(x)}{\sum_{i=0}^{k} b_i T_i(x)} \tag{4.63}$$

był równy kombinacji liniowej wielomianów Czebyszewa o wsakźnikach większych od $k+n$.

Aproksymacja wielomianami Czebyszewa wykorzystuje też często **algorytm Clenshaw'a**, który pozwala łatwo i szybko obliczać te kombinacje liniowe.

4.10. Przedstaw algorytm Clenshawa i wyjaśnij, kiedy warto go stosować.

Algorytm Clenshawa to elegancki i efektywny sposób sumowania wyrazów spełniających pewien wzór rekurencyjny:

$$f(x) = \sum_{k=0}^{N} c_k F_k(x) \tag{4.64}$$

przy czym c_k znane oraz $F(x)$ spełnia wzór rekurencyjny:

$$F_{n+1}(x) = \alpha(n,x) \cdot F_n(x) + \beta(n,x) \cdot F_{n-1}(x) \tag{4.65}$$

gdzie $\alpha(n,x), \beta(n,x)$ - pewne funkcje (no shit)

Podczas obliczania algorytmu można także przyjąć k w tzw. `downwoard order`, od największej do najmniejszej, wtedy „intuicyjnie" wzór będzie sprowadzał się do:

$$F_{n-1}(x) = \alpha(n,x) \cdot F_n(x) + \beta(n,x) \cdot F_{n+1}(x) \tag{4.66}$$

, tak jak w poniższym przykładzie. Taką sumą jest na przykład suma stosowana przy wyznaczaniu współczynników dla interpolacji Czebyszewa.

Przebieg algorytmu:

Obliczamy rekurencyjnie pomocnicze zmienne;

$$\begin{gathered} y_{N+2} = y_{N+1} = 0 \\ y_k(x) = \alpha(k,x) \cdot y_{k+1} + \beta(k+1,x) \cdot y_{k+2} + c_k \\ k = N, N-1, N-2, ..., 1 \text{ (downward order)} \end{gathered} \tag{4.67}$$

Stąd wyznaczamy zależność:

$$c_k = y_k - \alpha(k,x) \cdot y_{k+1} - \beta(k+1,x) \cdot y_{k+2} \tag{4.68}$$

Po podstawieniu do wzoru otrzymujemy:

$$f(x) = \sum_{k=N}^{0} [y_k - \alpha(k,x) y_{k+1} - \beta(k+1,x) y_{k+2}] \cdot F_k(x) \tag{4.69}$$

$$\begin{aligned}
f(x) = \sum_{k=N}^{0} &[y_k - \alpha(k,x) y_{k+1} - \beta(k+1,x) y_{k+2}] \cdot F_k(x) = \\
= [\ & y_N && - && 0 && - && 0 &&]F_N(x) \\
+[\ & y_{N-1} && - && \alpha(N-1,x) \cdot y_N && - && 0 &&]F_{N-1}(x) \\
& \dots && && \dots && && \dots && \\
+[\ & y_8 && - && \alpha(8,x) y_9 && - && \beta(9,x) y_{10} &&]F_8(x) \\
+[\ & y_7 && - && \alpha(7,x) y_8 && - && \beta(8,x) y_9 &&]F_7(x) \\
+[\ & y_6 && - && \alpha(6,x) y_7 && - && \beta(7,x) y_8 &&]F_6(x) \\
+[\ & y_5 && - && \alpha(5,x) y_6 && - && \beta(6,x) y_7 &&]F_5(x) \\
& \dots && && \dots && && \dots && \\
+[\ & y_2 && - && \alpha(2,x) y_3 && - && \beta(3,x) y_4 &&]F_2(x) \\
+[\ & y_1 && - && \alpha(1,x) y_2 && - && \beta(2,x) y_3 &&]F_1(x) \\
+[\ & y_0 && - && \alpha(0,x) y_1 && - && \beta(1,x) y_2 &&]F_0(x) =
\end{aligned}$$

Można zauważyć, że po odpowiednim zgrupowaniu czerwonych składników:

$$\begin{gathered} y_8[F_8(x) - \alpha(7,x)F_7(x) - \beta(7,x)F_6(x)] \\ \text{oraz} \\ F_8(x) = \alpha(7,x)F_7(x) + \beta(7,x)F_6(x) \end{gathered} \tag{4.70}$$

czerwone składniki się zerują. Jedynymi składnikami, które pozostaną, będą te w **dolnym lewym trójkącie „macierzy”**, czyli y_1 oraz y_0.

Wyznaczenie $f(x) = \sum_{k=0}^{N} c_k F_K(x))$ dla wzoru w `downwoard order` sprowadza się do:

- wyznaczenia y_1 oraz y_2 z formuły rekurencyjnej:

$$\begin{cases} y_{N+2} = y_{N+1} = 0 \\ y_k(x) = \alpha(k,x) \cdot y_{k+1} + \beta(k+1,x) \cdot y_{k+2} + c_k \end{cases} \tag{4.71}$$

- obliczenia sumy:

$$f(x) = \beta(1,x) \cdot y_2(x) \cdot F_0(x) + y_1(x) \cdot F_1(x) + c_0 F_0(x) \tag{4.72}$$

Note:

W zależności od różnic pomiędzy:

- $F_k(x)$ dla małych i dużych k
- c_k dla małych i dużych k

może mieć znaczenie, czy współczynniki y_k są liczone od większych do mniejszych k, czy na odwrót.

Jeżeli

$$\beta(1,x) \cdot y_2 \cdot F_0 \text{ oraz } y_1 \cdot F_1 \tag{4.73}$$

są przeciwnych znaków i prawie równe, należy zastosować `upward direction`:

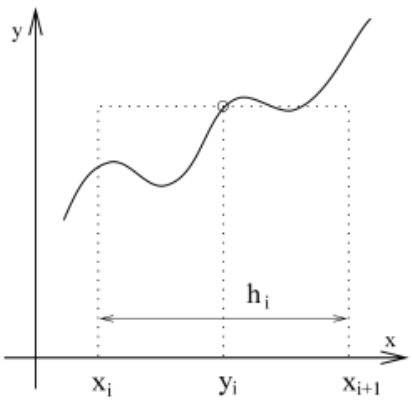
$$\begin{cases} y_{-2} = y_{-1} = 0 \\ y_k = \frac{1}{\beta(k+1,x)}[y_{k-2} - \alpha(k,x)y_{k-1} - c_k], \quad k = 0, 1, ..., N-1 \end{cases} \tag{4.74}$$

$$f(x) = c_N F_N(x) - \beta(B,x) \cdot F_{N-1}(x) \cdot y_{N-1} - F_N(x) \cdot y_{N-2} \tag{4.75}$$

5. Kwadratury

5.1. Wyprowadź wzór na kwadratury elementarne trapezów i prostokątów (z błędami) korzystając ze wzoru Taylora

Wzór prostokątów:



Szukamy całki postaci:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx f(y_i)h_i \tag{5.1}$$

Wykorzystując wzór Taylora:

$$y_i = \frac{x_i + x_{i+1}}{2} \quad h_i = x_{i+1} - x_i \quad \frac{h}{2} = x_{i+1} - y_i = y_i - x_i \tag{5.2}$$

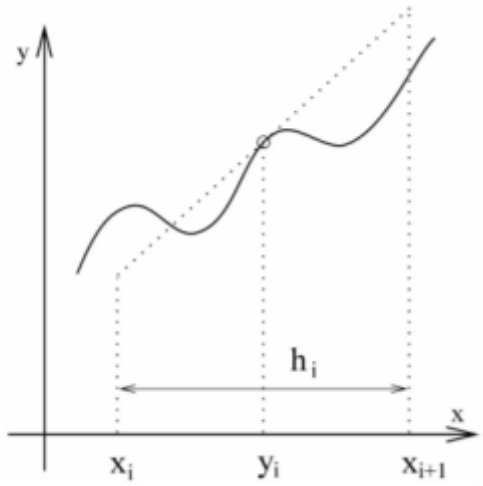
$$f(x) = f(y_i) + \sum_{p=1}^{\infty} \frac{f^{(p)}(y_i)}{p!}(x - y_i)^p \tag{5.3}$$

$$F(x) = \int f(x)dx = xf(y_i) + \sum_{p=1}^{\infty} \frac{f^{(p)}(y_i)}{(p+1)!}(x - y_i)^{p+1} \tag{5.4}$$

$$\int_{x_i}^{x_{i+1}} f(x)dx = F(x_{i+1}) - F(x_i) = \underbrace{f(y_i)h_i}_{\text{kwadratura}} + \underbrace{\frac{1}{24}h_i^3 f''(y_i) + \frac{1}{1920}h_i^5 f^{(4)}(y_i) + \ldots}_{\text{błąd}} \tag{5.5}$$

Kwadratura prostokątow ma błąd $O(h^3)$. Ma ona stopień dokładności 1 - jest dokładna dla funkcji liniowej.

Wzór trapezów:



Szukamy całki postaci:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{1}{2}[f(x_i) + f(x_{i+1})] \cdot h_i \tag{5.6}$$

Ponownie przyjmujemy oznaczenia:

$$y_i = \frac{x_i + x_{i+1}}{2} \quad h_i = x_{i+1} - x_i \quad \frac{h}{2} = x_{i+1} - y_i = y_i - x_i \tag{5.7}$$

Wykorzystując dwukrotnie wzór Taylora:

$$f(x_i) = f(y_i) + f'(y_i)\underbrace{(x_i - y_i)}_{-\frac{h_i}{2}} + \frac{f''(y_i)}{2!}\underbrace{(x_i - y_i)}_{-\frac{h_i}{2}}^2 + ... = f(y_i) - \frac{h_i}{2}f'(y_i) + \frac{\left(\frac{h_i}{2}\right)^2}{2!}f''(y_i) + ... \tag{5.8}$$

$$f(x_{i+1}) = f(y_i) + \frac{h_i}{2}f'(y_i) + \frac{\left(\frac{h_i}{2}\right)^2}{2!}f''(y_i) + ... \tag{5.9}$$

Po zsumowaniu, **wyrazy z nieparzystymi pochodnymi znoszą się ze względu na znak**.

$$\begin{gathered} f(x_i) + f(x_{i+1}) = 2\left[f(y_i) + \frac{\left(\frac{h_i}{2}\right)^2}{2!}f''(y_i) + ...\right] \\ f(y_i) = \frac{f(x_i) + f(x_{i+1})}{2} - \frac{\left(\frac{h_i}{2}\right)^2}{2!}f''(y_i) - .... \end{gathered} \tag{5.10}$$

Stąd, podstawiając powyższe do całki oznaczonej ze wzoru kwadratów:

$$\begin{gathered} \int_{x_i}^{x_{i+1}} f(x)dx = F(x_{i+1}) - F(x_i) = f(y_i)h_i + \frac{1}{24}h_i^3 f''(y_i) + \frac{1}{1920}h_i^5 f^{(4)}(y_i) + ... = \\ = \underbrace{\frac{1}{2}[f(x_i) + f(x_{i+1})] \cdot h_i}_{\text{kwadratura}} - \underbrace{\frac{1}{12}h_i^3 f''(y_i) - \frac{1}{480}h_i^5 f^{(4)}(y_i) + ...}_{\text{błąd}} \end{gathered} \tag{5.11}$$

Kwadratura trapezowa ma błąd $O(h^3)$. Ma ona stopień dokładności 1 - jest dokładna dla funkcji liniowej.

5.2. Wyprowadź wzór na kwadraturę złożoną Simpsona (wraz ze wzorem na jej błąd)

Wzór Simpsona uzyskujemy całkując parabolę przechodzącą przez x_i, $\frac{x_i+x_{i+1}}{2}$, x_{i+1} (będącą przybliżeniem funkcji):

$$\int_a^b f(x)dx \approx \frac{1}{6}(b-a)\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right] \tag{5.12}$$

Przyjmując, że kwadratura Simpsona przebiega przez trzy punkty x_0, x_1, x_2 i przyjmując $h = x_1 - x_0$ (zamiast $a - b = x_2 - x_0 = 2h$), wzór ten zapisuje się w postaci:

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \underbrace{\frac{h^5}{90}f^{(4)}(\eta)}_{\text{błąd}} \qquad h = x_1 - x_0 \tag{5.13}$$

Należy pamiętać, że powyższe kwadratury da się uogólnić (biorąc wielomiany interpolujące wyższego stopnia), co nazywa się kwadraturami Newtona-Cotesa

Metoda złożona

Aby uzyskać wzór na metodę złożoną, przyjmujemy:

$$h = \frac{b-a}{2m} \quad x_i = x_0 + i \cdot h, \ i = 0, 1, ..., 2m \tag{5.14}$$

, dzieląc dany przedział $[a, b]$ na n podprzedziałów ($n = 2m$, żeby zmieściły się „Simpsony") o równej długości.

$$\begin{aligned}\int_a^b f(x)dx = \sum_{i=1}^{m}\int_{x_{2i-2}}^{x_{2i}} f(x)dx = \sum_{i=1}^{m}\left\{\frac{h}{3}\cdot[f(x_{2i-2})+4f(x_{2i-1})+f(x_{2i})] - \frac{h^5}{90}f^{(4)}(\eta_i)\right\} = \\ = \frac{h}{3}\cdot\left[\underbrace{f(x_0)+4f(x_1)+f(x_2)}+\underbrace{f(x_2)+4f(x_3)+f(x_4)}+\ldots\right] - \sum_{i=1}^{m}\frac{h^5}{90}f^{(4)}(\eta_i) = \\ = \frac{h}{3}\cdot\left[f(x_0)+4\sum_{i=1}^{m}f(x_{2i-1})+2\sum_{i=1}^{m-1}f(x_{2i})+f(x_{2m})\right] - \sum_{i=1}^{m}\frac{h^5}{90}f^{(4)}(\eta_i)\end{aligned} \tag{5.15}$$

Błąd metody złożonej

Z tw. Weierstrassa, dla wybranych $x_1, x_2 \in [a,b]$:

$$\exists_{x_1,x_2\in[a,b]} \min_{x_1\in[a,b]} f^{(4)}(x_1) \le f^{(4)}(\eta_j) \le \max_{x_2\in[a,b]} f^{(4)}(x_2) \tag{5.16}$$

Po zsumowaniu dla każdego podprzedziału $[x_{2i-2}, x_{2i}]$:

$$m\cdot\min_{x_1\in[a,b]} f^{(4)}(x_1) \le \sum_{j=1}^{m} f^{(4)}(\eta_j) \le m\cdot\max_{x_2\in[a,b]} f^{(4)}(x_2) \tag{5.17}$$

$$\min_{x_1\in[a,b]} f^{(4)}(x_1) \le \frac{1}{m}\sum_{j=1}^{m} f^{(4)}(\eta_j) \le \max_{x_2\in[a,b]} f^{(4)}(x_2) \tag{5.18}$$

Z tw. o wartości pośredniej Darboux:

$$\exists\mu\in(a,b) : f^{(4)}(\mu) = \frac{1}{m}\sum_{i=1}^{m} f^{(4)}(\eta_i) \tag{5.19}$$

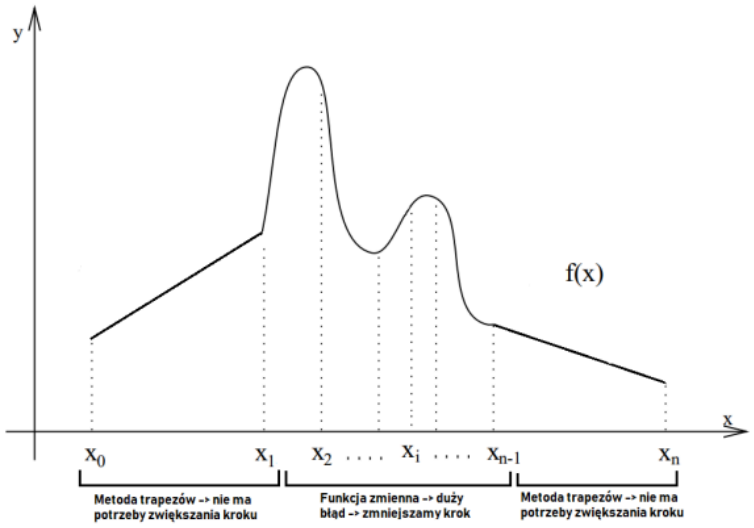
Tak więc zastępujemy dla wygody: zamiast $\frac{1}{m}\sum_{i=1}^{m} f^{(4)}(\eta_i)$ mamy pochodną w punkci μ

$$\begin{aligned}E = -\frac{h^5}{90}\cdot\boldsymbol{m}\cdot f^{(4)}(\mu) \quad m = \frac{b-a}{2h} \\ \Rightarrow E = -\frac{h^4(b-a)}{180}f^{(4)}(\mu)\end{aligned} \tag{5.20}$$

5.3. Przedstaw i objaśnij algorytm całkowania adaptacyjnego (rozpocznij od dobrego rysunku)

Wzory złożone posiadają węzły równoodległe, co ignoruje fakt, że funkcje mogą mieć w przedziale całkowania różny (przedziałami) przebieg, jak np. funkcje oscylacyjne.

Całkowanie adaptacyjne (adaptacyjny wzór Simpsona) dla $\int_a^b f(x)dx$ dzieli przedział $[a,b]$ na fragmenty, dla których klasyczne metody całkowania dają już dostatecznie dobre wyniki. Dla dużych przedziałów o małej zmienności funkcji daje od razu wynik, miejsca o dużej zmienności dzieli na krótsze przedziały i wykonuje tam gęściej obliczenia. Jest ono równie precyzyjne co klasyczne całkowanie dla „dobrych" funkcji, ale dla bardziej problematycznych funkcji daje dużo precyzyjniejsze wyniki.



Niech $S(a,b)$ oznacza wartość całki obliczoną za pomocą pojedynczej (niezłożonej) kwadratury Simpsona na przedziale $[a,b]$.

Zakładając, że $f \in C^4([a,b])$, można skorzystać ze wzoru Simpsona dla $h = \frac{b-a}{2}$ i zapisać:

$$\int_a^b f(x)dx = \underbrace{\frac{h}{3} \cdot [f(a) + 4f(a+h) + f(b)]}_{S(a,b)} - \frac{h^5}{90} f^{(4)}(\mu) \quad \mu \in (a,b) \tag{5.21}$$

Warunek, z którego korzysta całkowanie adaptacyjne dla obliczenia całki z precyzją ε, to:

$$\left| S(a,b) - \underbrace{S\left(a, \frac{a+b}{2}\right)}_{(a,a+h)} - \underbrace{S\left(\frac{a+b}{2}, b\right)}_{(a+h,b)} \right| < 15\varepsilon \quad (*) \tag{5.22}$$

Kroki algorytmu całkowania adaptacyjnego:

- Gdy spełniony jest warunek $(*)$, to $S\left(a, \frac{a+b}{2}\right) + S\left(\frac{a+b}{2}, b\right)$ przybliża $\int_a^b f(x)dx$ z dokładnością ε i można przerwać
- Gdy $(*)$ nie jest spełnione, to oceniamy błąd w $\left[a, \frac{a+b}{2}\right]$ i $\left[\frac{a+b}{2}, b\right]$, w każdym z nich $\varepsilon' = \frac{\varepsilon}{2}$
- Połowimy przedziały i wyznaczamy S
- Ten podprzedział (molżiwe, że oba), na którym odpowiednik $(*)$ nie jest spełniony, znów połowimy, nie zmieniając gotowych przedziałów.

5.4. Porównaj kwadratury Newtona-Cotesa i Gaussa; wyjaśnij różnice między nimi.

Kwadratury Newtona-Cotesa

Kwadratura Newtona-Cotesa to rodzina metod numerycznego całkowania, które polegają na aproksymacji całki funkcji za pomocą wielomianów interpolacyjnych. Sposób postępowania:

- Podział przedziału całkowania $[a,b]$ na węzły (np. równoodległe, Czebyszewa)
- Zastąpienie funkcji $f(x)$ wielomianem interpolacyjnym, często ze wzoru Lagrange'a:

$$f(x) = \sum_{i=0}^{n} L_i(x) f(x_i) + \frac{f^{(n+1)}(\eta(x))}{(n+1)!} \prod_{j=0}^{n} (x - x_j) \tag{5.23}$$

- Całkowaniu wielomianu, co daje przybliżoną wartość całki.

Metody prostokątów, trapezów i Simpsona są szczególnymi przypadkami kwadratur N-C.

Kwadratury Gaussa

Kwadratury Gaussa to rodzina metod numerycznego całkowania, które zapewniają maksymalną dokładność poprzez optymalny dobór węzłów i wag. Węzły x_i i wagi w_i są dobierane tak, aby metoda była dokładna dla wielomianów możliwie najwyższego stopnia. Przez to, że dobieramy zarówno węzły i wagi, mamy $2n$ parametrów, co pozwala uzyskiwać wielomiany $2n-1$ w przypadku n-punktowej kwadratury.

Niech będzie dana n-punktowa kwadratura Gaussa na przedziale $[a, b]$ i niech $w(x)$ będzie jej funkcją ważącą (nadającą wagi poszczególnym punktom).

Węzły (odcięte) x_i tej kwadratury są **zerami wielomianu ortognalnego** $\varphi_n(x)$ dla tego przedziału i tej samej funkcji $w(x)$.

Jeśli $\{\varphi_i\}_i^n$ - zbiór wielomianów ortogonalnych w $[a, b]$, to $\varphi_n(x)$ - ma n różnych zer $x_1, x_2, ..., x_n \in (a, b)$.

Możemy znaleźć wielomian interpolujący $f(x)$ w tych węzłach:

$$f(x) = \sum_{i=1}^{n} f(x_i) \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} + \frac{f^{(n)}(\eta(x))}{n!} \prod_{i=1}^{n} (x - x_i) \tag{5.24}$$

i wtedy policzyć całkę jako:

$$\int_a^b w(x) f(x) dx \approx \sum_{i=1}^{n} a_i f(x_i), \quad a_i = \int_a^b w(x) L_i(x) dx \tag{5.25}$$

Całka jest na pewno dokładna, gdy $f(x)$ jest wielomianem stopnia $\leq n-1$, bo wtedy $f^{(n)}(\eta) = 0$.

Porównanie kwadratur:

- **Węzły**:
  - kwadratury Newtona-Cotesa wykorzystują węzły równoodległe,
  - kwadratury Gaussa węzły będące zerami wielomianów ortogonalnych wybranego typu, np. wielomianów Czebyszewa
- **Wagi**: dla formuły $\int_a^b f(x)w(x) \approx \sum_{i=0}^{n} a_i f(x_i)$:
  - kwadratury N-C mają $w(x) = 1$, a a_i to liczby Cotesa, mamy więc n parametrów
  - dla kwadratury Gaussa $w(x)$ i a_i zależą od wybranego typu wielomianów ortogonalnych, więc dobieramy $2n$ parametrów
- **Przedział całkowania**:
  - dla N-C przedział całkowania nie wpływa na sposób obliczeń
  - dla Gaussa różne typy przedziałów całkowania pozwalają całkować na różnych obszarach, np. dla kwadratury Gaussa-Legendre'a lub Gaussa-Czebyszewa przedział całkowania to $[-1, 1]$ - jeśli nasz docelowy przedział całkowania jest inny, to trzeba go przekształcić do $[-1, 1]$
- **Typy otwarte i zamknięte**:

  - kwadratury N-C dzielą się na otwarte i zamknięte (odpowiednio niewykorzystujące i wykorzystujące punkty na brzegach przedziału całkowania)
  - dla kwadratur Gaussa nie istnieje taki podział

- **Typy elementarne i złożone**:
  - kwadratury N-C dzieli się na elementarne (jeden zwór dla całego przedziału naraz, niska dokładność dla dużych przedziałów) i złożone (podział na podprzedziały i stosowanie tam wzorów elementarnych - (Sekcja 5.2))
  - brak powyższego podziału dla Gaussa

- **Dokładność**:
  - w kwadraturach N-C dokładność zależy od parzystości n (liczby węzłów) i tego, czy kwadartura jest otwarta, czy zamknięta.
  - w kwadraturach Gaussa dla n punktów dokładność wynosi $2n-1$ (błąd wynosi zero dla wielomianów stopnia $\leq 2n-1$)

5.5. Omów zasadę tworzenia kwadratur Gaussa, podaj potrzebne twierdzenia

Podstawowe twierdzenie kwadratur Gaussa

Niech będzie dana n-punktowa kwadratura Gaussa na przedziale $[a,b]$ i niech $w(x)$ będzie jej funkcją ważącą (nadającą wagi poszczególnym punktom).

Węzły (odcięte) x_i tej kwadratury są **zerami wielomianu ortognalnego** $\varphi_n(x)$ dla tego przedziału i tej samej funkcji ważącej $w(x)$.

Jeśli $\{\varphi_i\}_i^n$ - zbiór wielomianów ortogonalnych w $[a,b]$, to $\varphi_n(x)$ - ma n różnych zer $x_1, x_2, ..., x_n \in (a,b)$.

Możemy znaleźć wielomian interpolujący $f(x)$ w tych węzłach:

$$f(x) = \sum_{i=1}^{n} f(x_i) \prod_{j=1, j\neq i}^{n} \frac{x-x_j}{x_i-x_j} + \frac{f^{(n)}(\eta(x))}{n!} \prod_{i=1}^{n}(x-x_i) \tag{5.26}$$

i wtedy policzyć całkę jako:

$$\int_a^b w(x)f(x)dx \approx \sum_{i=1}^{n} a_i f(x_i), \quad a_i = \int_a^b w(x)L_i(x)dx \tag{5.27}$$

Całka jest na pewno dokładna, gdy $f(x)$ jest wielomianem stopnia $\leq n-1$, bo wtedy $f^{(n)}(\eta) = 0$.

Wyznaczanie wag a_i

Znając odcięte (węzły) $x_1, x_2, ..., x_n$, możemy wyznaczyć wagi wyznaczamy z układu równań:

$$\begin{bmatrix} \varphi_0(x_1) & \dots & \varphi_0(x_n) \\ \vdots & \ddots & \vdots \\ \varphi_{n-2}(x_1) & \dots & \varphi_{n-2}(x_n) \\ \varphi_{n-1}(x_1) & \dots & \varphi_{n-1}(x_n) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \int_a^b w(x)\varphi_x(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.28}$$

Zera w wektorze z prawej strony wzięły się z faktu, że $\varphi_1(x), ..., \varphi_{n-1}(x)$ są ortogonalne do $\varphi_0(x)$ będącą stałą > 0, więc ich iloczyn się zeruje:

$$\int_a^b w(x)\varphi_i(x)dx = \frac{1}{\varphi_0}\varphi_0 \int_a^b w(x)\varphi_i(x)dx = \int_a^b w(x)\varphi_0(x)\varphi_i(x)dx = 0 \quad \text{dla } i = 1, ..., n-1 \tag{5.29}$$

Wagi uzyskane z powyższego układu są dokładne dla wielomianów ortogonalnych stopnia co najwyżej $n-1$ (zakładamy, że węzły x_i mamy już ustalone, dlatego tutaj nie ma $2n-1$, tylko o n mniej, bo szukamy tylko wag a_i).

Twierdzenie o stopniu dokładności kwadratur Gaussa

Kwadratura n-punktowa ma stopień dokładności $2n-1$, to znaczy że jest dokładna dla wielomianu:

$$P(x) = Q(x) \cdot \varphi_n(x) + R(x), \quad Q, R \to \text{stopnia} < n \tag{5.30}$$

Uwaga: każdy wielomian stopnia $2n-1$ można przedstawić w takiej postaci

5.6. Omów zasadę wyznaczania wag w kwadraturach Gaussa

Wyznaczanie wag a_i

Znając węzły $x_1, x_2, ..., x_n$ wielomianu interpolującego, wagi możemy wyznaczyć z układu równań:

$$\begin{bmatrix} \varphi_0(x_1) & \dots & \varphi_0(x_n) \\ \vdots & \ddots & \vdots \\ \varphi_{n-2}(x_1) & \dots & \varphi_{n-2}(x_n) \\ \varphi_{n-1}(x_1) & \dots & \varphi_{n-1}(x_n) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \int_a^b w(x)\varphi_0(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{5.31}$$

Można zauważyć, że φ_0 jest wielomianem stopnia 0 - więc wartością stałą.

Zera w wektorze z prawej strony wzięły się z faktu, że $\varphi_1(x), ..., \varphi_{n-1}(x)$ są ortogonalne do $\varphi_0(x)$ będącą stałą > 0, więc przez tą ortogonalność ich iloczyn się zeruje:

$$\int_a^b w(x)\varphi_i(x)dx = \frac{1}{\varphi_0}\varphi_0 \int_a^b w(x)\varphi_i(x)dx = \frac{1}{\varphi_0}\int_a^b w(x)\varphi_0(x)\varphi_i(x)dx = 0 \quad \text{bo } \varphi_0(x) \cdot \varphi_i(x) = 0 \text{ dla } i = 1, ..., n-1 \tag{5.32}$$

Wagi uzyskane z powyższego układu są dokładne dla wielomianów ortogonalnych stopnia co najwyżej $n-1$ (zakładamy, że węzły x_i mamy już ustalone, dlatego tutaj nie ma $2n-1$, tylko o n mniej, bo szukamy tylko wag a_i).

5.7. Podaj i scharakteryzuj poznane dotąd przykłady użyteczności wielomianów ortogonalnych w obliczeniach numerycznych

- **Węzły interpolacji** - jeśli jako węzły interpolacji wybierze się miejsca zerowe ortogonalnych wielomianów Czebyszewa, to nie występuje efekt Rungego.
- **Baza aproksymacji** - jako bazę wielomianów układu normalnego wybiera się wielomiany ortogonalne Czebyszewa lub Lagrange'a
- **Całkowanie numeryczne** - do obliczenia optymalnego rozłożenia węzłów w kwadraturze Gaussa krozysta się z wielomianu ortogonalnego n-tego stopnia. Wykorzystuje się je także w kwadraturze Clenshawa-Curtisa
- **Metody iteracyjne niestacjonarne** - metody iteracyjne niestacjonarne rozwiązywania układów równań liniowych to takie, w których macierz iteracji ulega modyfikacji po każdym kroku. Metoda

Czebyszewa jest oparta na idei wielomianów ortogonalnych, pozwala uniknąć obliczania iloczynów skalarnych macierzy. Metody takie są szybciej zbieżne od stacjonarnych.

5.8. Opisz w jaki sposób można wykorzystać metodę divide and conquer (dziel i rządź) w algorytmach całkowania numerycznego

Zrównoleglenie obliczeń - algorytmy numeryczne można techniką *divide-and-conquer* łatwo zrównoleglić, gdyż ich istotą jest niezależne wykonywanie podproblemów, co pozwala wykorzystać możliwości współczesnych maszyn wielordzeniowych. Pozwala to łatwo znacznie przyspieszyć np. obliczanie FFT (równolegle parzyste i nieparzyste wyrazy) lub obliczanie kwadratur (adaptacyjny algorytm całkowania, kolejne podprzedziały obliczane równolegle).

5.9. Przedstaw przykłady wykorzystania twierdzeń z analizy matematycznej

do tworzenia/analizy algorytmów numerycznych

- **Wykorzystanie naturalnych splajnów sześciennych** - dzięki wykorzystaniu własności wielomianów (funkcje klasy C^∞) i nałożeniu surowych warunków ciągłości (ciągłość wielomianów, pierwszej i drugiej pochodnej dla splajnu 3-go stopnia) powstają bardzo gładkie, dobrze interpolujące funkcje. Dodatkowo, nakładając warunek $s''(x_1) = s''(x_n) = 0$ otrzymujemy *natural cubic spline* - zgodnie z twierdzeniem najgładszą funkcję interpolującą.

$$\int_a^b [s''(x)]^2 dx \leq \int_a^b [f''(x)]^2 dx \tag{5.33}$$

- **Aproksymacja jednostajna wielomianami Czebyszewa** - n-ty wielomian interpolujący Czebyszewa ma postać

$$W_n(x) = \sum_{j=0}^{n} c_j T_j(x) \tag{5.34}$$

  Współczynniki c_j oblicza się korzystając z ortogonalności wielomianów Czebyszewa. Same wielomiany mają wiele ciekawych własności:
  - równomierne rozłożenie zer na $[-1, 1]$
  - $n + 2$ ekstremów na tym przedziale
  - n miejsc zerowych na tym przedziale
  - oscylację wartości na tym przedziale (wartości pomiędzy −1 i 1, które to są ekstremami)

  Dzięki temu mają także własność minimaksu - ze wszystkich wielomianów n-tego stopnia z czynnikiem wiodącym 1, to wielomian $2^{1-n} T_n(x)$ (gdzie $T_n(x)$ jest wielomianem Czebyszewa) ma najmniejszą normę maksimum na przedziale $[-1, 1]$. Te matematyczne własności sprawiają, że aproksymacja ma wszędzie na przedziale $[-1, 1]$ równomiernie rozłożony, bardzo niewielki błąd.

- **Rozwiązywanie równań nieliniowych metodą siecznych** - wykorzystując przybliżenie ze wzoru Taylora otrzymuje się styczną do funkcji, która wymaga obliczenia pochodnej, co bywa bardzo trudne i kosztowne. Z definicji pochodnej można otrzymać przybliżenie pochodnej różnicą skończoną, której obliczenie jest prostsze. Zbieżność tej metody jest prawie tak szybka jak metody Newton-Raphsona, a przy tym unika się obliczania pochodnej, nie trzeba też sprawdzać, czy funkcja ma przeciwne znaki na końcach przedziału.

- **Obliczanie wyznacznika macierzy korzystając z rozkładu LU** - wykorzystując częściowe poszukiwanie pivotu, każdą kwadratową macierz nieosobliwą da się rozłożyć do iloczynu macierzy trójkątnych L i U *in situ*. Z twierdzenia Cauchy'ego:

$$\det(A) = \det(L \cdot U) = \det(L) \cdot \det(U) \tag{5.35}$$

a z własności macierzy trójkątnej wyznacznik jest równy iloczynowi elementów na przekątnej - jako, że L lu U ma same jedynki na przekątnej to $\det(A)$ jest równy $\det(L)$ lub $\det(U)$, którakolwiek macierz nie ma jedynek na przekątnej. Pozwala to na obliczenie wyznacznika macierzy A w czasie $O(\frac{2}{3}n^3)$

- **Dowód o reszcie interpolacji Lagrange'a** - Do wyprowadzenia wzoru na błąd interpolacji wykorzystuje się uogólnione Tw. Rolle'a.

  Założenia:
  1. $f \in C[a,b]$
  2. $f \in C^{n(a,b)}$
  3. $f = 0$ w $(n+1)$ różnych punktach

  Teza:

$$\exists c \in (a,b) : f^{(n)}(c) = 0 \tag{5.36}$$

6. Równania nieliniowe

6.1. Scharakteryzuj metodę bisekcji znajdowania rozwiązań równań nieliniowych

Dane:
- a i b, gdzie $a < b$ oraz $f(a) \cdot f(b) < 0$
- N - ustalona liczba iteracji
- $d = b - a$

Przebieg algorytmu bisekcji:

```
for i := 1 to N do
  c := (a + b) / 2
  if ( f(c) * f(a) < 0 ) do
    b := c
  else
    a := c

alpha := c
E := d / 2**N
```

,

gdzie:
- α - znalezione miejsce zerowe
- E - błąd wyznaczenia miejsca zerowego

Słownie:
- wybieramy przedział początkowy $[a,b]$ w którym funkcja zmienia znak (warunek $f(a) \cdot f(b) < 0$).
- wyznaczony przedział dzielimy na połowy i obliczamy wartość funkcji w punkcie środkowym $c = \frac{a+b}{2}$
- jeśli $f(c) = 0$, to c jest szukanym pierwiastkiem
- w przeciwnym razie sprawdzamy, w której połowie przedziału zmienia się znak i powtarzamy kroki dla tego podprzedziału.

Początkowo punkt α znajduje się w przedziale początkowym $[\alpha_0, \beta_0]$ więc błąd jest proporcjonalny do $b_0 - a_0$. Po N krokach:

$$e = b_N - a_N = \frac{b_{N=1} - a_{N-1}}{2} = ... = \frac{b_0 - a_0}{2^N} \tag{6.1}$$

Właściwości metody:

- gwarancja zbieżności
- wolno zbieżna, jedyna wykorzystywana informacja to znak funkcji
- znajduje tylko jedno miejsce zerowe, nawet jeśli występuje ich więcej
- kryterium zbieżności:

$$E \approx 10^{-6} \rightarrow \text{dobre dla } |\alpha| \sim 1, \text{złe dla } |\alpha| \sim 10^{-26} \tag{6.2}$$

$$\begin{gathered}\text{zwykle: } E = \varepsilon \cdot \frac{a_0 + b_0}{2} \\ (\varepsilon - \text{epsilon maszynowy})\end{gathered} \tag{6.3}$$

6.2. Podaj i udowodnij twierdzenie o zbieżności procesu iteracyjnego

Przyjmuje się oznaczenie:

$$f(x) = 0 \quad \equiv \quad x = \Phi(x) \tag{6.4}$$

$$x_{i+1} = \Phi(x_i), \quad i = 0, 1, 2, ... \tag{6.5}$$

Twierdzenie o zbieżności procesu iteracyjnego

Niech:

- $x = \Phi(x)$ ma pierwiastek α
- na przedziale $I = [\alpha - a; \alpha + a]$ dla stałej L zachodzi:

$$|\Phi'(x)| \leq L < 1 \tag{6.6}$$

Teza: Wtedy, dla dowolnego $x_0 \in I$:

- $x_i \in I$, i = 1,2,... (gdzie x_i to kolejne przybliżenia pierwiastka)
- $\lim_{i \to \infty} x_i = \alpha$
- α jest jedynym pierwiastkiem $x = \Phi(x)$ w I

Dowód:

Niech $x_{i-1} \in I$, wtedy:

$$x_i - \alpha = \Phi(x_{i-1}) - \Phi(\alpha) \tag{6.7}$$

Z twierdzenia o wartości średniej:

$$\exists \eta_{i-1} \in I \ : \ \Phi'(\eta_{i-1}) = \frac{\Phi(x_{i-1}) - \Phi(\alpha)}{x_{i-1} - \alpha} \tag{6.8}$$

$$x_i - \alpha = \Phi(x_{i-1}) - \Phi(\alpha) = (x_{i-1} - \alpha) \cdot \Phi'(\eta_{i-1}) \tag{6.9}$$

$$|x_i - \alpha| = |x_{i-1} - \alpha| \cdot |\Phi'(\eta_{i-1})| \leq |x_{i-1} - \alpha| \cdot L \quad \Rightarrow x_i \in I \quad (*) \tag{6.10}$$

Z $(*)$ korzystamy wielokrotnie, stąd:

$$|x_i - \alpha| \leq L \cdot |x_{i-1} - \alpha| \leq L^2 \cdot |x_{i-2} - \alpha| \leq ... \leq L^i \cdot |x_0 - \alpha| \tag{6.11}$$

$$L < 1 \Rightarrow \lim_{i\to\infty} L^i = 0 \quad \Rightarrow \quad \lim_{i\to\infty} |x_i - \alpha| = 0 \tag{6.12}$$

Teraz, niech w I oprócz pierwiastka α będzie także pierwiastek β:

$$\alpha - \beta = \Phi(\alpha) - \Phi(\beta) = (\alpha - \beta) \cdot \Phi'(\eta) \tag{6.13}$$

$$|\alpha - \beta| = |\alpha - \beta| \cdot \underbrace{|\Phi'(\eta)|}_{<1} < |\alpha - \beta| \quad \rightarrow \text{sprzeczność! } \blacksquare \tag{6.14}$$

6.3. Wyjaśnij pojęcie rzędu zbieżności procedury iteracyjnej

Rząd zbieżności procedury iteracyjnej określa szybkość, z jaką kolejne przybliżenia generowane przez metodę iteracyjną zbliżają się do dokładnego rozwiązania.

Niech:

- $\{x_i\} \rightarrow$ oznacza ciąg przybliżeń generowanych przez procedurę iteracyjną (np. $x_i = \Phi(x_{i-1})$).
- $\alpha \rightarrow$ oznacza dokładne rozwiązanie, czyli punkt stały $\alpha = \Phi(\alpha)$
- $\varepsilon_i = x_i - \alpha = \Phi(x_{i-1}) - \Phi(\alpha) \rightarrow$ błąd i-tego przybliżenia

Mówimy, że ciąg $\{x_i\}$ jest zbieżny do α, jeśli $\lim_{i\to\infty} x_i = \alpha$.

Szybkość zbieżności mierzymy przez wyrażenie:

$$\left| \frac{\varepsilon_i}{\varepsilon_{i-1}} \right| = ? \tag{6.15}$$

Jeśli granica tego stosunku istnieje, określa rząd zbieżności metody.

Wyprowadzenie rzędu zbieżności:

Rozwijamy $\Phi(x)$ w Taylora wokół α:

$$\Phi(x_{i-1}) = \Phi(\alpha) + \sum_{j=1}^{p-1} \frac{\Phi^{(j)}(\alpha)}{j!} \underbrace{(x_{i-1} - \alpha)^j}_{\varepsilon_{i-1}} + \frac{\Phi^{(p)}(\eta_{i-1})}{p!}(x_{i-1} - \alpha)^p \tag{6.16}$$
$$\eta_{i-1} \in (x_{i-1}, \alpha)$$

Stąd:

$$\varepsilon_i = \Phi(x_{i-1}) - \Phi(\alpha) = \sum_{j=1}^{p-1} \frac{\Phi^{(j)}(\alpha)}{j!} \underbrace{(x_{i-1} - \alpha)^j}_{\varepsilon_{i-1}} + \frac{\Phi^{(p)}(\eta_{i-1})}{p!}(x_{i-1} - \alpha)^p \tag{6.17}$$

Aby zbieżność była wyższego rzędu niż liniowa, pochodne Φ muszą zerować się do odpowiedniego rzędu. W celu osiągnięcia rzędu p, jeżeli:

$$\Phi'(\alpha) = \Phi''(\alpha) = \ldots = \Phi^{(p-1)}(\alpha) = 0, \quad \Phi^{(p)}(\alpha) \neq 0 \tag{6.18}$$

, to:

$$\varepsilon_i = \frac{\Phi^{(p)}(\eta_{i-1})}{p!} \underbrace{(x_{i-1} - \alpha)^p}_{\Phi(x_{i-2}) - \Phi(\alpha)} = \frac{\Phi^{(p)}(\eta_{i-1})}{p!} \varepsilon_{i-1}^p \tag{6.19}$$

$$\frac{\varepsilon_i}{\varepsilon_{i-1}^p} = \frac{\Phi^{(p)}(\eta_{i-1})}{p!} \tag{6.20}$$

Stąd:

$$\lim_{i\to\infty}\left|\frac{\varepsilon_i}{\varepsilon_{i-1}^p}\right| = \frac{\Phi^{(p)}(\alpha)}{p!} \tag{6.21}$$

$$\begin{gathered}\left|\frac{\varepsilon_i}{\varepsilon_{i-1}^p}\right| \text{ - } p\text{-ty rząd zbieżności (p-th order of convergence)} \\ \frac{\left|\Phi^{(p)}(\alpha)\right|}{p!} \text{ - stała asymptotyczna błędu}\end{gathered} \tag{6.22}$$

Gdzie::

- $p = 1$ - linear
- $p = 2$ - quadratic
- $p = 3$ - cubic

Można konstruować metody iteracyjne $x_i = \Phi(x_{i-1})$ dowolnego rzędu p, lecz konieczne jest wyliczenie $0, 1, ..., p-1$ pochodnych.

6.4. Scharakteryzuj metody iteracyjne w obliczeniach numerycznych, podaj: ogólny algorytm, potrzebne twierdzenie, kiedy są przydatne

Metody iteracyjne to metody (algorytmy), których idea opiera się na ulepszaniu rozwiązania wraz z kolejnymi iteracjami.

Ogólny algorytm:

- wybierz punkt startowy x_0
- oblicz kolejne przybliżenie za pomocą funkcji iteracyjnej:

$$x_i = \Phi(x_{i-1}) \quad i = 1, 2, ... \tag{6.23}$$

- sprawdź warunek stopu (np. maksymalna liczba iteracji, błąd bezwzględny):

$$\|x_i - x_{i-1}\| < \varepsilon \;\; \text{lub} \;\; i > i_{\max} \tag{6.24}$$

- zwróć x_i jako wynik po przerwaniu pętli

Do wykorzystywania metod iteracyjnych do konkretnych problemów potrzebne są twierdzenia zapewniające warunki zbieżności. Metody można stosować tylko dla problemów, dla których te warunki są spełnione, czyli iteracje będą dawać coraz lepsze rozwiązania.

Twierdzenie te to np. twierdzenie o zbieżności procesu iteracyjnego $Ax = b$ dla układów równań liniowych lub o zbieżności procesu iteracyjnego $x = \Phi(x)$ dla równań nieliniowych.

Metody iteracyjne są szczególnie przydatne, kiedy możemy zadowolić się przybliżonym rozwiązaniem lub chcemy mieć kontrolę nad złożonością czasową problemu i związaną z tym precyzją rozwiązania. Ponadto metody te mogą mieć dodatkowe właściwości zależne od konkretnego problemu, np. w układach równań liniowych nie naruszają one struktur macierzy rzadkich.

6.5. Wyprowadź wzór na metodę Newtona-Raphsona i jej rząd zbieżności

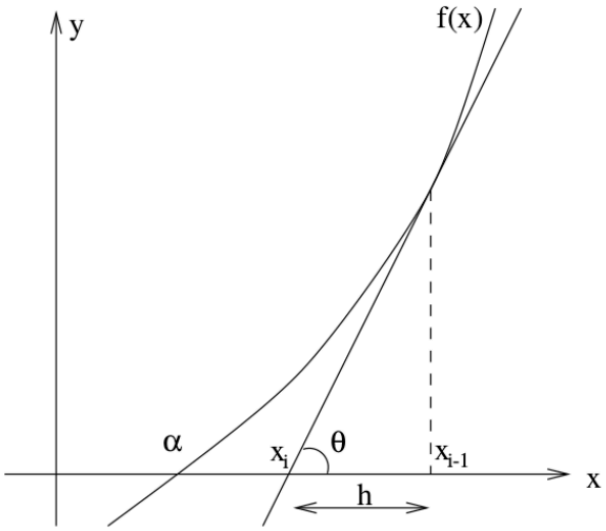
Mając daną funkcję $f(x)$ i szukając pierwiastka $\alpha : f(\alpha) = 0$ wykorzystujemy przyliżenia α postaci x_{i-1}.

Oznaczając $\alpha = x_{i-1} + h$ oraz biorąc początek szeregu Taylora rozwiniętego wokół x_{i-1} otrzymujemy wzór na styczną do funkcji:

$$f(\alpha) = 0 = f(x_{i-1} + h) = f(x_{i-1}) + f'(x_{i-1})(x_{i-1} + h - x_{i-1}) + \underbrace{...}_{\text{pomijamy}} \tag{6.25}$$

$$f(x_{i-1}) + f'(x_{i-1}) \cdot h = 0 \tag{6.26}$$

$$\begin{aligned} h &= \frac{f(x_{i-1})}{f'(x_{i-1})} \\ x_i &= x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})} \end{aligned} \tag{6.27}$$



Wzór iteracyjny ma więc postać:

$$\Phi(x) = x - \frac{f(x)}{f'(x)} \tag{6.28}$$

Warunek zbieżności:

$$\begin{aligned} \Phi'(x) &= \frac{f''(x) \cdot f(x)}{(f'(x))^2} \\ \Phi''(x) &= \frac{f'(x) \cdot f''(x)}{(f'(x))^2} + f(x) \cdot \left(\frac{f''(x)}{(f'(x))^2} \right)' \\ &\text{dla } x = \alpha : f'(\alpha) \neq 0 \end{aligned} \tag{6.29}$$

W punkcie α, wiedząc, że $f(\alpha) = 0$ mamy:

$$\begin{aligned} \Phi'(\alpha) &= \frac{f''(\alpha) \cdot \boldsymbol{f(\alpha)}}{(f'(\alpha))^2} = 0 \\ \Phi''(\alpha) &\neq 0 \end{aligned} \tag{6.30}$$

Daje nam to kwadratową zbieżność metody - tylko pierwsza pochodna jest równa 0, więc $\varepsilon_i \sim \varepsilon_{i-1}^2$ (błąd maleje kwadratowo z **liczbą** iteracji)

Dodatkowo, Powinno istnieć otoczenie α, w którym $|\Phi'(x)| < 1$ tj. przy odpowiednim doborze x_0 met. Newtona-Rhapsona jest zawsze zbieżna do α.

6.6. Przedstaw metodę Aitkena - do czego służy i kiedy się ją stosuje

Metoda Atekina to technika przyspieszania zbieżności ciągów liczbowych lub procedur iteracyjnych, szczególnie tych o zbieżności liniowej. Jest stosowana głównie w obliczeniach numerycznych do poprawiania przybliżeń rozwiązania.

Niech x_i, x_{i+1}, x_{i+2} - wartość wyznaczenia miejsca zerowego w kolejnych iteracjach

$$x_i - \alpha = \Phi(x_{i-1}) - \Phi(\alpha) = (x_{i-1} - \alpha) \cdot \Phi'(\eta_{i-1}), \quad \eta_{i-1} \in (x_{i-1}, \alpha) \tag{6.31}$$

$$x_{i+2} - \alpha = (x_{i+1} - \alpha) \cdot \Phi'(\eta_{i+1}) \tag{6.32}$$

$$x_{i+1} - \alpha = (x_i - \alpha) \cdot \underbrace{\Phi'(\eta_i)}_{(*)} \tag{6.33}$$

Założenie: $(*)$ - nieznane, ale dla dwóch kolejnych iteracji blisko rozwiązania $\Phi'(\eta_i) \approx \Phi'(\eta_{i+1})$ (ponieważ $\Phi'(\eta_i)$ dąży do stałej asymptotycznej błędu). Przekształcając więc powyższe wzory, mamy:

$$\begin{gathered} \Phi'(\eta_i) = \Phi'(\eta_{i+1}) \\ \frac{x_{i+2} - \alpha}{x_{i+1} - \alpha} = \frac{x_{i+1} - \alpha}{x_i - \alpha} \end{gathered} \tag{6.34}$$

$$\alpha = \frac{x_i \cdot x_{i+2} - x_{i+1}^2}{x_{i+2} - 2 \cdot x_{i+1} + x_i} = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2 \cdot x_{i+1} + x_i} \tag{6.35}$$

Na tej podstawie mamy wzór na nowe, lepsze przybliżenie x_{i+2}^*:

$$x_{i+2}^* = x_i - \frac{(x_{i+1} - x_i)^2}{x_{i+2} - 2 \cdot x_{i+1} + x_i} \to (*) \tag{6.36}$$

$(*)$ - zbieżność kwadratowa bez obliczania drugich pochodnych

6.7. Scharakteryzuj metodę Regula Falsi oraz jej warianty znajdowania rozwiązań równań nieliniowych

Regula falsi to sposób znajdowania pierwiastka równania nieliniowego bazujący na fałszywym twierdzeniu, że na pewnym przedziale funkcja jest liniowa. Wykorzystuje ona interpolowanie funkcji poprzez prostą, która zawsze przechodzi przez jeden punkt.

Wymagania:
- W przedziale $[a, b]$ znajduje się dokładnie 1 pojedyńczy pierwiastek
- $f(a) \cdot f(b) < 0$
- f' i f'' istnieją na tym przedziale i mają na nim stałe znaki

Reguła falsi:
- Startujemy mając punkty x_0, x_1 takie, że $f(x_0) \cdot f(x_1) < 0$ (punkty o różnych znakach)
- przez punkt x_0 i x_1 prowadzimy prostą i znajdujemy jej punkt przecięcia z osią OX:
  Wyznaczając współczynnik kierynkowy a prostej:
  - wiemy, że przechodzi przez punkty $(x_0, f(x_0))$ i $(x_1, f(x_1))$, więc:

$$a = \frac{f(x_0) - f(x_1)}{x_0 - x_1} \tag{6.37}$$

  - wiemy, że przechodzi przez punkt $(x_0, f(x_0))$ oraz przecina oś OX, więc przechodzi przez $(x_2, 0)$, stąd:

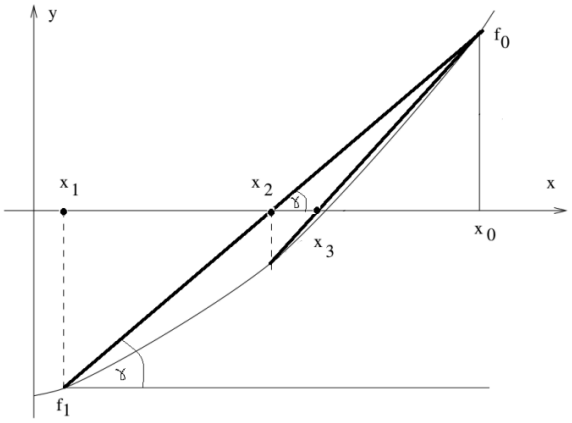
$$a = \frac{f(x_0) - 0}{x_0 - x_2} = \frac{f(x_0)}{x_0 - x_2} \tag{6.38}$$

Przyrównując do siebie współczynniki a, mamy:

$$\frac{f(x_0) - f(x_1)}{x_0 - x_1} = \frac{f(x_0)}{x_0 - x_2} \Rightarrow x_2 = \frac{f(x_1)}{f(x_1) - f(x_0)} x_0 + \frac{f(x_0)}{f(x_0) - f(x_1)} x_1 \tag{6.39}$$

- sprawdzamy po której stronie osi OX znajduje się $f(x_2)$. Punkty do kolejnej iteracji wybieramy tak, aby leżały po różnych stronach osi OX:

$$x_3 \Leftarrow \begin{cases} f(x_1) \cdot f(x_2) < 0 \rightarrow \text{zero leży w } (x_2, x_1) \\ f(x_0) \cdot f(x_2) < 0 \rightarrow \text{zero leży w } (x_0, x_2) \end{cases} \tag{6.40}$$



Ulepszeniem metody falsi są **metody Illinois i Pegasus**.
Jeśli mamy x_i, x_{i+1}, x_{i+2} takie, że:

$$\begin{cases} f(x_i) \cdot f(x_{i+1}) < 0 \leftarrow \text{"obejmują" pierwiastek} \\ f(x_{i+1}) \cdot f(x_{i+2}) > 0 \end{cases} \tag{6.41}$$

, czyli jeśli najnowsza wartość $f(x_{i+2})$ jest po tej samej stronie osi OX, co poprzednia $f(x_{i+1})$:

- stosujemy regułe falsi korzystając ze starszego punktu x_i, czyli dla punktów x_i, x_{i+2} ale ze zmodyfikowaną starszą wartością:

$$f(x_i) = f(x_i)^* = \alpha \cdot f(x_i)$$
$$\text{Illinois} \rightarrow \alpha = \frac{1}{2} \qquad \text{Pegasus} \rightarrow \alpha = \frac{f(x_{i-1})}{f(x_i) + f(x_{i-1})} \tag{6.42}$$

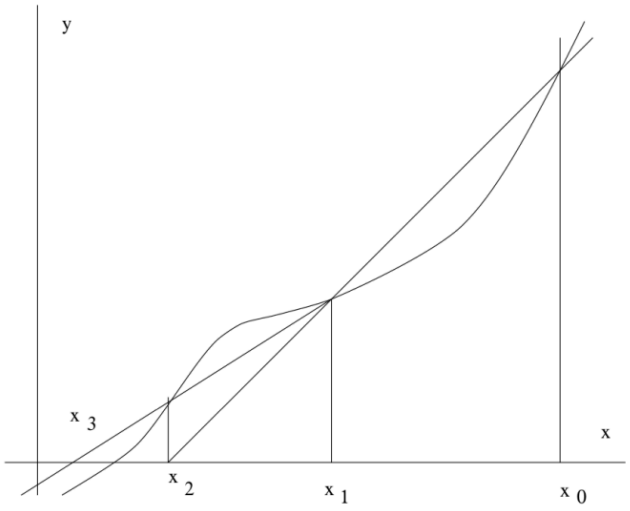
- podobnie, gdy:

$$f_{i+2} \cdot f_{i+3} \begin{cases} < 0 - \text{RF dla } x_{i+2}, x_{i+3} \\ > 0 - \text{zmodyfikowana RF dla } x_{i+1}, x_{i+3} \end{cases} \tag{6.43}$$

6.8. Scharakteryzuj metody siecznych oraz Steffensena znajdowania rozwiązań równań nieliniowych.

Metoda siecznych to modyfikacja metody Newtona-Raphsona, która dzięki przybliżeniu pochodnej poprzez różnicę skończoną pozwala uniknąć obliczania pochodnej. Nie wymaga także badania znaków $f_{x_i} \cdot f(x_{i+1})$. Wzór na kolejne przybliżenia pierwiastka ma postać:

$$\frac{1}{f'(x_i)} \approx \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$
$$x_{i+1} = x_i - \underbrace{\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}}_{\frac{1}{f'(x_i)}} \cdot f(x_i) \tag{6.44}$$



Charakterystyka:
- rząd zbieżności ~ 1.62
- lepsza od reguły falsi
- gorsza od Newtona-Rhapsona, ale nie trzeba znać f'

Metoda Steffensena jest bardzo podobna do metody Newtona, jednak nie wymaga liczenia pochodnej funkcji. Warunki początkowe są takie same jak dla metody Newtona:
- f określona na $[a, b]$
- f ciągła na $[a, b]$
- f ma różne znaki na krańcach $[a, b]$ ($f(a) \cdot f(b) < 0$)

Stosuje się ją do równań postaci:

$$x_{i+1} = x_i - \frac{f(x_i)}{g(x_i)} \tag{6.45}$$

, gdzie $g(x_i)$ to przybliżenie pochodnej poprzez iloraz różnicowy (określa średnie nachylenie funkcji $f(x)$ pomiędzy dwoma punktami):

$$g(x_i) = \frac{f(x_i + f(x_i)) - f(x_i)}{f(x_i)} \tag{6.46}$$

Kroki postępowania metody:
- budujemy sieczną używając x_i oraz $x_i + f(x_i)$
- x_{i+1} to punkt przecięcia tej siecznej z osią OX
- w kolejnym kroku budujemy sieczną używając x_{i+1} oraz $x_{i+1} + f(x_{i+1})$
- każdy krok wymaga policzenia dwóch nowych wartości funkcji (w metodzie siecznych od drugiego kroku tylko jednej)
- rząd zbieżności = 2 (lepszy od siecznych)

7. Bezpośrednie metody rozwiązywania układów równań liniowych

7.1. Przedstaw algorytm rozwiązywania układów równań liniowych metodą eliminacji Gaussa

Algorytm eliminacji Gaussa polega na zredukowaniu macierzy do postaci trójkątnej-górnej za pomocą operacji elementarnych:

- mnożenie wiersza przez $\lambda \neq 0$
- dodawanie wierszy
- zamianę wierszy

Następnie, wartości obliczane są od końca (od „dołu" układu, backward substitution).

Złożoność algorytmu - $O(n^3)$.

Przebieg algorytmu:

Sprowadzanie do macierzy trójkątnej-górnej:

- Wybieranie pivot-u w aktualnej kolumnie k:
  - znajdź wiersz $p \geq k$ z największą co do modułu wartością w kolumnie k
  - Jeśli $A[p][k] = 0$ - przerwij, układ sprzeczny lub nieoznaczony
  - Zamień wiersze k i p
- Dla każdego wiersza $i = k + 1, ..., n$:
  - Oblicz współczynnik eliminacji:

$$m = \frac{A[i][k]}{A[k][k]} \tag{7.1}$$

  - odejmij od wiersza i wiersz k pomnożony przez m:

$$A[i][j] = A[i][j] - m \cdot A[k][j] \text{ dla każdego } j = k, ..., n \tag{7.2}$$

  - Zaktualizuj wektor b:

$$b[i] = b[i] - m \cdot b[k] \tag{7.3}$$

Backward-subsitution:

- Dla $i = n, n-1, ..., 1$:
  - Oblicz $x[i]$:

$$x[i] = \frac{b[i] - \sum_{j=i+1}^{n} A[i][j] \cdot x[j]}{A[i][i]} \tag{7.4}$$

  - jeśli $A[i][i] = 0$ - układ sprzeczny lub nieoznaczony

```
n = len(A)

# Eliminacja współczynników
for k in range(n-1):
    # Wybór pivota (częściowe wybieranie)
    max_row = k
    for i in range(k+1, n):
        if abs(A[i][k]) > abs(A[max_row][k]):
            max_row = i
    A[k], A[max_row] = A[max_row], A[k]
    b[k], b[max_row] = b[max_row], b[k]
```

```
    # Eliminacja
    for i in range(k+1, n):
        m = A[i][k] / A[k][k]
        for j in range(k, n):
            A[i][j] -= m * A[k][j]
        b[i] -= m * b[k]

# backward substitution
x = [0] * n
for i in range(n-1, -1, -1):
    x[i] = b[i]
    for j in range(i+1, n):
        x[i] -= A[i][j] * x[j]
    x[i] /= A[i][i]

return x
```

Wybór jak największego p (pivotu) bierze się stąd, że później dzielenie przez p = $a[i][i]$ spowoduje możliwie mały błąd i nie spowoduje overflowu (mógłby wystąpić przy dzieleniu przez małe elementy). Formalnie wystarczyłoby jakiekolwiek $p \neq 0$, żeby metoda działała, ale taki partial pivoting zwiększa też stabilność numeryczną algorytmu.

7.2. Wyznacz złożoność obliczeniową metody Gaussa rozwiązywania układów równań liniowych

Złożoność obliczeniowa: $O(n^3)$.

Liczba działań mnożenia i dzielenia wynosi:

$$\underbrace{\sum_{i=1}^{n-1}}_{\substack{\text{suma}\\ \text{zewnętrzna}}} \left[\underbrace{(n-i)}_{\substack{\text{pętla}\\ \text{wewnętrzna}}} \left(\underbrace{1}_{\substack{\text{dzielenie}\\ \frac{A[i][k]}{A[k][k]}}} + \underbrace{n-i}_{\substack{\text{mnożenie}\\ \text{wiersza przez}\\ m}} + \underbrace{1}_{\substack{\text{mnożenie}\\ b[i]}} \right) \right] + \underbrace{\frac{n(n-1)}{2}}_{\substack{\text{backward}\\ \text{substitution}}} = \tag{7.5}$$

$$= \frac{1}{3}(n^3 + 3n^2 - n) = O(n^3)$$

Liczba działań dodawania i odejmowania wynosi:

$$\frac{1}{6}(2n^3 + 3n^2 - 5n) = O(n^3) \tag{7.6}$$

7.3. Wyjaśnij dlaczego istotnym krokiem każdej metody rozwiązywania układów równań liniowych jest szukanie elementu wiodącego (głównego), a następnie opisz gdzie i jak go się poszukuje

Element wiodący (pivot) - element macierzy wybierany do przeprowadzania głównych operacji w algorytmach algebry liniowej, np. eliminacji Gaussa czy metodzie simpleksów. Czasami jest on niezbędny, a czasami tylko zwiększa stabilność numeryczną.

W przypadku rozwiązywania układów równań liniowych (Gauss, rozkład LU) pivotem jest element, przez który dzielimy odejmowane elementy wierszy. W związku z tym, musi on być niezerowy, a im większa jest jego wartość bezwzględna, tym bardziej algorytm jest stabilny (bo przez niego dzielimy).

Istnieją trzy główne metody jego poszukiwania:

- ***Partial pivoting*** (poszukiwanie częściowe) - k-ty pivot to element z k-tej kolumny (jej części poniżej k-tego wiersza z nim włącznie) o największej wartości bezwzględnej.

  Zamienia się miejscami tylko wiersze.

- ***Full pivoting*** (wyszukiwanie pełne) - k-ty pivot wybiera się element o największej wartości bezwzględnej z podmacierzy k-tego wiersza włącznie i w dół i od k-tej kolumny włączenie w prawo.

  Wymaga zamian wierszy i kolumn (więc także np. przetrzymywania macierzy permutacji w celu pamiętania kolejności nieiwadomych).

  Daje lepszą stabilność numeryczną od częściowego poszukiwania, ale wymaga przeszukania k^2 elementów.

- ***Scaled pivoting*** (wyszukiwanie skalowane) - na k-ty pivot wybiera się element o największej wartości bezwzględnej z podmacierzy od k-tego wiersza włącznie w dół i od k-tej kolumny włącznie w prawo.

  Wymaga zamian wierszy i kolumn (ponownie macierz permutacji).

  Daje lepszą stabilność numeryczną od częściowego poszukiwania, ale wymaga przeszukania k^2 elementów.

- W przypadku **metod iteracyjnych** (głównie dla macierzy rzadkich) przestawia się wiersze macierzy tak, aby na diagonali nie było zer, a elementy niezerowe o jak największym module.

7.4. Opisz i porównaj algorytmy faktoryzacji LU Doolittle'a, Crout'a i Choleskiego

Faktoryzacja LU polega na zapisaniu macierzy A jako iloczynu macierzy trójkątnej-dolnej L i trójkątnej--górnej U. W praktyce, obie te macierze przechowuje się w jednej (*in situ*), gdyż niezależnie od przyjętej metody nigdy nie trzeba pamiętać 2 różnych elementów na diagonali.

Niech:
- l - elementy macierzy L
- u - elementy macierzy U
- a - elementy oryginalnej macierzy A

Dla $k = 1, 2, \ldots, n-1$ w k-tym kroku wykonujemy operację:

$$A = LU \tag{7.7}$$

$$a_{kk} = \sum_{s=1}^{k-1} l_{ks} u_{sk} + l_{kk} u_{kk} \quad (*) \tag{7.8}$$

$$l_{kk} = \frac{a_{kk} - \sum_{s=1}^{k-1} l_{ks} u_{sk}}{u_{kk}} \qquad u_{kk} = \frac{a_{kk} - \sum_{s=1}^{k-1} l_{ks} u_{sk}}{l_{kk}} \tag{7.9}$$

W powyższej operacji $(*)$ oblicza się l_{kk} lub u_{kk}, z której następnie można obliczyć (dla $j = k+1, k+2, \ldots, n-1$):

$$a_{kj} = \sum_{s=1}^{k-1} l_{ks} u_{sj} + l_{kk} u_{kj} \quad \text{- } k\text{-ty wiersz} \tag{7.10}$$

$$a_{ik} = \sum_{s=1}^{k-1} l_{is} u_{sk} + l_{ik} u_{kk} \quad \text{- } k\text{-ta kolumna} \tag{7.11}$$

W operacji $(*)$ trzeba znać l_{kk} lub u_{kk}, aby móc wyliczyć drugą. W zależności od wybranego algorytmu faktoryzacji otrzymuje się to w różny sposób:

- **faktoryzacja Doolitle'a**:

$$l_{ii} = 1, \quad i = 0, 1, ..., n-1 \tag{7.12}$$

  Daje ona rozkład $LDU = L(DU)$, gdzie D to macierz w wartościami na diagonali

- **faktoryzacja Crouta**:

$$u_{ii} = 1, \quad i = 0, 1, ..., n-1 \tag{7.13}$$

  Daje ona rozkład $LDU = (LD)U$

- **faktoryzacja Cholesky'ego**:

$$l_{ii} = u_{ii}, \quad i = 0, 1, ..., n-1 \tag{7.14}$$

  Daje ona rozkład $\left(LD^{\frac{1}{2}}\right)\left(D^{\frac{1}{2}}U\right) = LL^T$. Faktoryzację tę można zastosować tylko dla macierzy rzeczywistej, symetrycznej i dodatnio określonej, ale jest szybszy od pozostałych.

7.5. Objaśnij na czym polega przewaga algorytmów faktoryzacji LU nad metodą eliminacji Gaussa.

Główną przewagą faktoryzacji LU jest oddzielenie faz faktoryzacji (obliczenia macierzy L i U) oraz rozwiązania układu równań $Ax = b$, dzięki czemu:

- Można zrównoleglić obliczenia przy rozwiązywaniu układu równań wielokrotnie dla różnych wektorów b.
- Koszt obliczeń dla rozwiązania jednego układu równań jest taki sam. Jednak, gdy układ rozwiązuje się wielokrotnie dla różnych wektorów b koszt maleje:

$$\begin{gathered} \text{LU} \to O(n^3 + kn^2) \\ \text{Gauss} \to O(kn^3) \\ k - \text{liczba wektorów } b \end{gathered} \tag{7.15}$$

- Łatwo jest obliczyć wyznacznik macierzy

$$\det(A) = \det(LU) = \det(L) \cdot \det(U) = \prod_{i=1}^{n} l_{ii} \cdot \prod_{i=1}^{n} u_{ii} \tag{7.16}$$

  Podobnie, macierz odwrotną można obliczyć rozwiązując n układów $AX = I$ (dla każdej kolumny I) wykorzystując ten sam rozkład LU

7.6. Wyjaśnij na czym polega przydatność metod blokowych do rozwiązywania układów równań liniowych

Metoda blokowa to technika polegająca na podziale macierzy A i wektora b na mniejsze bloki (podmacierze i podwektory), a następnie wykorzystaniu tych bloków do efektywniejszego rozwiązania układu $Ax = b$.

Układ równań dzieli się na mniejsze bloki A_{ij}:

$$Ax = b \tag{7.17}$$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.18}$$

Następnie układ zapisać można jako układ równań blokowych:

$$\begin{cases} A_{11}x_1 + A_{12}x_2 = b_1 \\ A_{21}x_1 + A_{22}x_2 = b_2 \end{cases} \tag{7.19}$$

Po czym można zastosować:
- blokową eliminacje Gaussa
- metody iteracyjne dla bloków
- rekurencyjne rozkłady blokowe

Przydatność metod blokowych:

- Stanowią kompromis pomiędzy prostotą metod iteracyjnych a efektywnością metod dokładnych
- Operacje na blokach można równoleglić (divide and conquer), a same bloki lepiej wykorzystują cache procesora - unika się kosztownych operacji na całej macierzy
- Zapisanie macierzy rzadkiej przekątniowej jako macierzy blokowej przekątniowej nie zaburzy struktury macierzy rzadkiej, a bloki zerowe są pomijane, co przyspiesza obliczenia
- W równaniach różniczkowych cząstkowych często pojawiają się przekątniowe macierze rzadkie, które można zapisać jako macierze przekątniowe blokowe i zastosować dla nich szybkie metody bezpośrednie
- W metodach iteracyjnych (Jacobiego, S-R) korzystamy z łatwości odwracania macierzy diagonalnej D, a za tę macierz można przyjąć inną macierz łatwą do odwracania (np. trójdiagonalną), co pozwala, przy poprawnym wyborze macierzy, uzyskać szybką zbieżność i jednocześnie zachować iteracyjny charakter metody.

8. Iteracyjne metody rozwiązywania układów równań liniowych

8.1. Wyjaśnij kiedy warto używać iteracyjnych metod rozwiązywania układów równań liniowych

- dla specyficznych układów równań, które powstają w wyniku rozwiązywania równań różniczkowych (np. w przypadku Metody Elementów Skończonych)
- gdy nie jest konieczne dokładne rozwiązanie, a wystarczy jego przybliżenie
- gdy układy są nieliniowe
- dla macierzy rzadkich metody iteracyjne zachowują ich strukturę (w przeciwieństwie do metod bezpośrednich), dzięki czemu można zapamiętywać macierze rzadkie w efektywnych strukturach (jak macierze blokowe) i zmniejszać koszty pamięciowe
- Algorytmy iteracyjne (np. CG, multigrid) łatwo dzielą się na niezależne części, co pozwala na efektywne wykorzystanie GPU lub klastrów obliczeniowych.

8.2. Podaj i udowodnij twierdzenie o zbieżności procesu iteracyjnego rozwiązywania $\mathbf{Ax = b}$

Dany mamy rozkład macierzy A:

$$A = B + R \tag{8.1}$$

, gdzie:

- B - macierz dla której łatwo stworzyć B^{-1}
- R - pozostałość

$$\begin{gathered} A \cdot x = (B + R) \cdot x = b \\ B \cdot x = -R \cdot x + b \end{gathered} \tag{8.2}$$

$$B \cdot x = -(A - B) \cdot x + b \tag{8.3}$$

Metody iteracyjne dla $Ax = b$ polegają na:

- odgadnięciu wektora początkowego $x^{(0)}$
- generowaniu ciągu iteracyjnego $x^{(t)}$ według postulowanego wzoru:

$$B \cdot x^{(t+1)} = -(A - B) \cdot x^{(t)} + b \tag{8.4}$$

$$x^{(t+1)} = -B^{-1} \cdot (A - B) \cdot x^{(t)} + B^{-1} \cdot b \tag{8.5}$$

$$x^{(t+1)} = \underbrace{I - B^{-1} \cdot A}_{\substack{M \\ \text{macierz iteracji}}} \cdot x^{(t)} + \underbrace{B^{-1} \cdot b}_{W} \tag{8.6}$$

$$x^{(t+1)} = M \cdot x^{(t)} + B^{-1} \cdot b \quad (*) \tag{8.7}$$

Różne B oznaczają tworzą różne rodziny metod iteracyjnych.

Warunek zgodności formuły z szukanym rozwiązaniem:

$$\lim_{t \to \infty} x^{(t+1)} = \lim_{t \to \infty} \left(M \cdot x^{(t)} + B^{-1} \cdot b \right) \tag{8.8}$$

Twierdzenie o zbieżności procesu iteracyjnego

Teza: Ciąg $(*)$ z dowolnym wektorem startowym $x^{(0)}$ jest zbieżny do jedynego granicznego $x^{(\infty)}$ wtedy i tylko wtedy, gdy **promień spektralny** macierzy iteracji jest < 1:

$$\rho(M) < 1 \tag{8.9}$$

, gdzie promień spektralny to moduł wartości własnej o największej wartości bezwzględnej.

Dowód:

Wprowadźmy wektor błędu t-tej iteracji $\varepsilon^{(t)}$:

$$\varepsilon^{(t)} = x^{(t)} - x \tag{8.10}$$

$$x^{(t+1)} = M \cdot x^{(t)} + W \quad \text{oraz} \quad x = M \cdot x + W \quad (**) \tag{8.11}$$

Odejmując mamy:

$$x^{(t+1)} - x = \underbrace{M\left(x^{(t+1)} - x\right)}_{(**)} = M \cdot \varepsilon^{(t)} = \varepsilon^{(t+1)} \tag{8.12}$$

$$\varepsilon^{(t+1)} = M \cdot \varepsilon^{(t)} \tag{8.13}$$

Stosując wielokrotnie:

$$\varepsilon^{(t)} = \mathrm{M}^t \cdot \varepsilon^0 \tag{8.14}$$

Aby proces był zbieżny, to dla skalara $\|\varepsilon^t\|$ musi zachodzić:

$$\left\|\varepsilon^{(t+1)}\right\| < \left\|\varepsilon^{(t)}\right\| \tag{8.15}$$

Wiedząc, że macierz iteracji M ma n różnych wartości własnych λ_i i wektorów własnych s_i. Można zapisać:

$$\mathrm{M} \cdot \mathrm{s}_i = \lambda_i \mathrm{s}_i \tag{8.16}$$

Rozkładamy wektor błędu (przedstawiamy jako kombinację liniową wektorów własnych s_i):

$$\begin{gathered} \varepsilon^{(0)} = \sum_{i=1}^{n} \alpha_i \mathrm{s}_i \\ \alpha_i \ \text{ - amplituda kierunku } \mathrm{s}_i \end{gathered} \tag{8.17}$$

$$\varepsilon^{(t)} = \mathrm{M}^t \sum_{i=1}^{n} \alpha_i \mathrm{s}_i = \sum_{i=1}^{n} \alpha_i \mathrm{M}^t \mathrm{s}_i = \sum_{i=1}^{n} \alpha_i \mathrm{M}^{(t-1)} \underbrace{(\mathrm{M} \cdot \mathrm{s}_i)}_{\lambda_i \mathrm{s}_i} = \ldots = \sum_{i=1}^{n} \alpha_i \lambda_i^t \mathrm{s}_i \tag{8.18}$$

$$\varepsilon^{(t)} = \sum_{i=1}^{n} \alpha_i \lambda_i^t \mathrm{s}_i \tag{8.19}$$

Stąd:

$$\begin{gathered} \lim_{t\to\infty} \varepsilon^{(t)} = \lim_{t\to\infty} \alpha_m \lambda_m^t \mathrm{s}_m = \lim_{t\to\infty} \alpha_m \cdot (\rho(\mathrm{M}))^t \cdot \mathrm{s}_m \\ \text{ponieważ} \ \ \lambda_m = \max_i |\lambda_i| = \rho(\mathrm{M}) \end{gathered} \tag{8.20}$$

Tak więc granica istnieje, ponieważ α_m i s_m są stałe, a z tezy zadania mamy:

$$\rho(\mathrm{M}) < 1 \ \ - \ \ \textbf{warunek zbieżności} \tag{8.21}$$

8.3. Podaj wzory macierzowe dla dla metod iteracyjnych Jacobiego oraz Gaussa-Seidla (S-R)

Oznaczmy:

- D - macierz diagonalna
- L - macierz poddiagonalna (trójkątna-dolna bez diagonali)
- U - macierz naddiagonalna (trójkątna-górna bez diagonali)

Rozważmy układ równań postaci:

$$\begin{gathered} Ax = b \\ \text{gdzie } A = D + L + U \end{gathered} \tag{8.22}$$

Szumay wektora rozwiązań za pomocą kolejnych iteracyjnych przybliżeń $x^{(n+1)}$. Można więc zapisać:

$$(D + L + U)x = b \tag{8.23}$$

$$Dx^{(t+1)} = -(L + U)x^{(t)} + b \tag{8.24}$$

Metoda Jakobiego:

Korzystamy wprost z definicji dla $Dx^{(t+1)}$:

$$\begin{aligned} Dx^{(t+1)} &= -(L+U)x^{(t)} + b \\ x^{(t+1)} &= D^{-1}\left[-(L+U)x^{(t)} + b\right] \end{aligned} \tag{8.25}$$

Powyższy wzór to wzór macierzowy dla metody Jacobiego. ABy otrzymać wzór roboczy, należy zapisać:

$$x^{(t+1)} = \underbrace{\frac{1}{a_{ii}}}_{D^{-1}} \left[b_i - \underbrace{\sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(t)}}_{\substack{\text{wszystko oprócz} \\ \text{diagonali}}} \right] \quad a_{ii} \neq 0,\ i = 1, 2, ..., n \tag{8.26}$$

Na powyższym wzorze roboczym dla metody Jacobiego widać, że macierz diagonalna D nie może mieć żadnych zer na diagonali.

Metoda S-R:

Zapiszmy:

$$A = \underbrace{D + L}_{B} + U = B + U \tag{8.27}$$

$$Ax = b \tag{8.28}$$

$$(B + U)x = b \tag{8.29}$$

$$Bx + Ux = b \tag{8.30}$$

$$Bx = -Ux + b \tag{8.31}$$

$$x = -B^{-1}U \cdot x + B^{-1} \cdot b \tag{8.32}$$

$$x^{(t+1)} = \underbrace{-B^{-1}U}_{M} \cdot x^{(t)} + \underbrace{B^{-1} \cdot b}_{W} \tag{8.33}$$

Po pomnożeniu wyrażenia stronami przez $B = D + L$ otrzymujemy:

$$Bx^{(t+1)} = -Ux^{(t)} + b \tag{8.34}$$

$$(D + L)x^{(t+1)} = -Ux^{(t)} + b \tag{8.35}$$

$$Dx^{(t+1)} = -Lx^{(t+1)} - Ux^{(t)} + b \tag{8.36}$$

$$x^{(t+1)} = D^{-1}\left[-Lx^{(t+1)} - Ux^{(t)} + b\right] \tag{8.37}$$

Powyższy wzór to wzór macierzowy metody S-R (Gaussa-Seidla). Aby otrzymać wzór roboczy, wystarczy zapisać:

$$x_i^{(t+1)} = \underbrace{\frac{1}{a_{ii}}}_{D^{-1}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(t+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(t)} \right] \tag{8.38}$$

W powyższym wzorze pierwsza suma jest znana, bo otrzymuje się ją z rozwiązania poprzednich równań w bieżącej t + 1 iteracji.

8.4. Podaj wzory robocze dla metod iteracyjnych Jacobiego, Gaussa-Seidla (S-R), SOR, Czebyszewa

Niech:

- D - macierz diagonalna
- L - macierz poddiagonalna (trójkątna-dolna bez diagonali)
- U - macierz naddiagonalna (trójkątna-górna bez diagonali)
- **Metoda Jacobiego**:

$$x^{(t+1)} = D^{-1}\left[-(L+U)x^{(t)} + b\right] \tag{8.39}$$

$$x^{(t+1)} = \underbrace{\frac{1}{a_{ii}}}_{D^{-1}}\left[b_i - \underbrace{\sum_{j=1, j\neq i}^{n} a_{ij}x_j^{(t)}}_{\substack{\text{wszystko oprócz} \\ \text{diagonali}}}\right] \quad a_{ii} \neq 0,\ i = 1, 2, ..., n \tag{8.40}$$

- **Metoda Gaussa-Seidla (S-R)**:

$$x^{(t+1)} = D^{-1}\left[-Lx^{(t+1)} - Ux^{(t)} + b\right] \tag{8.41}$$

$$x_i^{(t+1)} = \underbrace{\frac{1}{a_{ii}}}_{D^{-1}}\left[b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)}}_{\text{pod diagonalą}} - \underbrace{\sum_{j=i+1}^{n} a_{ij}x_j^{(t)}}_{\text{nad diagonalą}}\right] \tag{8.42}$$

- **Metoda SOR**:

$$x_i^{(t+1)} = x_i^{(t)} + \omega \cdot \underbrace{\frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)} - \sum_{j=i}^{n} a_{ij}x_j^{(t)}\right]}_{r_i^{(t)} \text{ - poprawka do starego rozwiązania } x_i^{(t)}} \tag{8.43}$$

ω - pewna stała

Ideą metody SOR jest przyspieszenie zbieżności wykorzystując pewną stałą ω:

$$x_i^{(t+1)} = x_i^{(t)} + \omega r_i^{(t)} \tag{8.44}$$

Można także zapisać:

$$x_i^{(t+1)} = (1-\omega)x_i^{(t)} + \omega \cdot \frac{1}{a_{ii}}\left[b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij}x_j^{(t+1)}}_{\text{pod diagonalą}} - \underbrace{\sum_{\boldsymbol{j=i+1}}^{n} a_{ij}x_j^{(t)}}_{\text{nad diagonalą}}\right] \tag{8.45}$$

ω - pewna stała

- **Metoda Czebyszewa**

Niech:

- $A = L + D + U$
- ρ - promień spektralny macierzy iteracji M

Przyjmujemy poniższy wielomian macierzowy:

$$B^{-1} = W(A) \tag{8.46}$$

$$M = 1 - W(A) \cdot A \tag{8.47}$$

Wielomianem Czebyszewa jest $W(A)$ taki, że

$$\min! \max_{U} |1 - W(U) \cdot U| \tag{8.48}$$

Obliczenia wykonuje się najpierw dla węzłów parzystych, potem dla nieparzystych (odpowiednio t całkowite i t ułamkowe).

Różnice względem **SOR**, to tylko postać B^{-1} i zmienna ω, więc poza tym wzory są takie jak dla **SOR**.

$$\omega^{(0)} = 1 \tag{8.49}$$

$$\omega^{(\frac{1}{2})} = \frac{1}{1 - \frac{1}{2}\rho^2} \tag{8.50}$$

$$\omega^{(t+\frac{1}{2})} = \frac{1}{1 - \frac{1}{4}\rho^2\omega^{(t)}} \quad t = \frac{1}{2}, 1, \frac{3}{2}, \ldots \tag{8.51}$$

$$\omega^{(\infty)} = \omega_{\text{opt}} \tag{8.52}$$

Stąd:

$$x_i^{(t+1)} = \left(1 - \omega^{(t)}\right) x_i^{(t)} + \omega^{(t)} \cdot \frac{1}{a_{ii}} \left[b_i - \underbrace{\sum_{j=1}^{i-1} a_{ij} x_j^{(t+1)}}_{\text{pod diagonalą}} - \underbrace{\sum_{\boldsymbol{j=i+1}}^{n} a_{ij} x_j^{(t)}}_{\text{nad diagonalą}} \right] \tag{8.53}$$

8.5. Porównaj metody iteracyjne Jacobiego, GS, SOR, Czebyszewa

- **Metoda Jacobiego** - wynika wprost z definicji iteracyjnego przybliżenia $x^{(n+1)}$ i korzysta ze wzoru

$$A = D + L + U \qquad Ax = b \tag{8.54}$$

$$(D + L + U)x = b \tag{8.55}$$

$$Dx^{(t+1)} = -(L + U)x^{(t)} + b \tag{8.56}$$

$$x^{(t+1)} = D^{-1} \cdot \left[-(L + U)x^{(t)} + b\right] \tag{8.57}$$

  Właściwości:
  - wolno zbieżna
  - nie wykorzystywana w praktyce (dydaktyka)
  - nie wykorzystuje całej dostępnej informacji w danym kroku
  - zbieżna dla silnie diagonalnie dominujących macierzy A
- **Metoda Gaussa-Seidla (S-R)** - ulepszenie metody Jacobiego, korzysta ze wzoru:

$$A = \underbrace{D + L}_{B} + U = B + U \tag{8.58}$$

$$Ax = b \tag{8.59}$$

$$(B + U)x = b \tag{8.60}$$

$$Bx + Ux = b \tag{8.61}$$

$$Bx = -Ux + b \tag{8.62}$$

$$x = -B^{-1}U \cdot x + B^{-1} \cdot b \tag{8.63}$$

$$x^{(t+1)} = \underbrace{-B^{-1}U}_{M} \cdot x^{(t)} + \underbrace{B^{-1} \cdot b}_{W} \tag{8.64}$$

Podczas obliczania $x^{(t+1)}$ po prawej stronie równania jest znane, bo zostało obliczone wcześniej w danej iteracji.

Właściwości:
- wykorzystuje informacje z wcześniejszych obliczeń z danej iteracji, stąd wzrost efektywności
- elementy diagonali muszą być różne od 0
- zbieżna dla macierzy A silnie diagonalnie dominujących wierszowo lub kolumnowo i dla macierzy symetrycznych i dodatnio określonych

- **Metoda SOR** - ulepszenie metody S-R, wykorzystuje poprawkę rozwiązania $r_i^{(t)}$:

$$x_i^{(t+1)} = x_i^{(t)} + \omega r_i^{(t)} \tag{8.65}$$

Powyżej ω to pewna liczba (wskaźnik relaksacji), przy czym dla ważnych praktycznie klas macierzy znane są optymalne wartości ω

Właściwości:
- przyspiesza zbieżność metody S-R poprzez wykorzystywanie poprawki
- metoda S-R to szczególny przypadek metody SOR dla $\omega = 1$

- **Metoda Czebyszewa** - ulepszenie metody SOR, w którym $B^{-1} = W(A)$, gdzie $W(A)$ to wielomian macierzowy Czebyszewa.

Istotą metody jest także użycie zmiennych wartości ω zgodnie ze wzorem , wktórym obliczenia przeprowadza się najpierw dla węzłów parzystych (t całkowite), a potem dla nieparzystych (t ułamkowe)

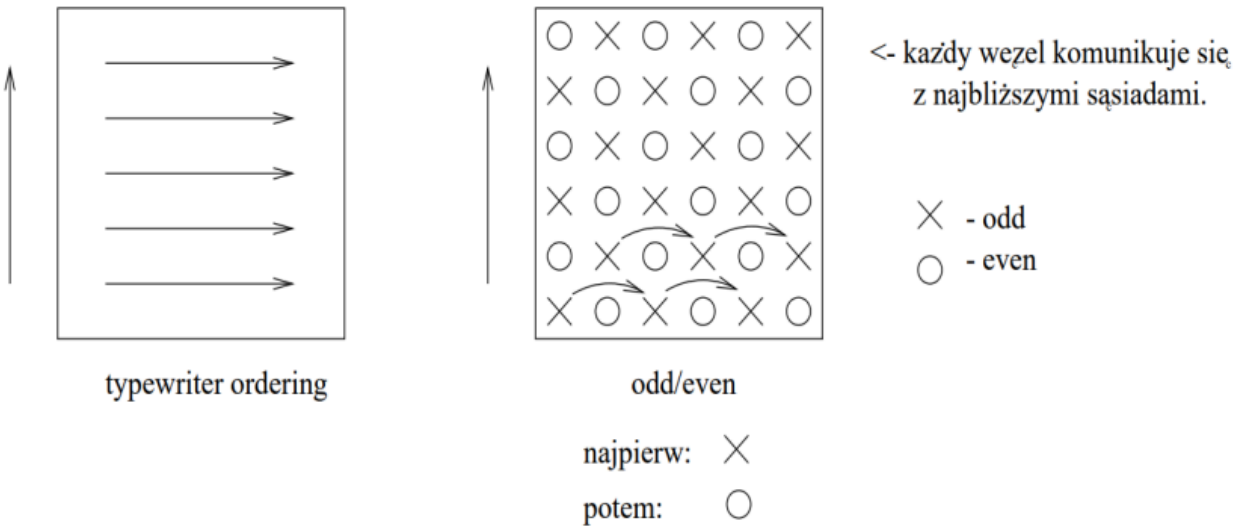
$$\omega^{(0)} = 1 \tag{8.66}$$

$$\omega^{(\frac{1}{2})} = \frac{1}{1 - \frac{1}{2}\rho^2} \tag{8.67}$$

$$\omega^{(t+\frac{1}{2})} = \frac{1}{1 - \frac{1}{4}\rho^2\omega^{(t)}} \quad t = \frac{1}{2}, 1, \frac{3}{2}, \ldots \tag{8.68}$$

$$\omega^{(\infty)} = \omega_{\text{opt}} \tag{8.69}$$

8.6. Objaśnij różnice między przeglądaniem punktów siatki typu type writer oraz odd-even



Przeglądanie typewriter:

- Punkty siatki są przeglądane w sposób liniowy, wiersz po wierszu, podobnie jak ruch głowicy w maszynie do pisania:
  - Zaczyna się w lewym górnym rogu siatki.
  - Przechodzi od lewej do prawej w pierwszym wierszu.
  - Po dotarciu do końca wiersza przechodzi do następnego wiersza i powtarza proces.

Przeglądanie Odd/Even:

- tak jak na załączonym obrazku

Odd/even pozwala na zrównoleglenie pewnych procesów iteracyjnych.

8.7. Podaj i scharakteryzuj 4 przykłady użyteczności wielomianów Czebyszewa w obliczeniach numerycznych

- **Węzły interpolacyjne Czebyszewa** - jeżeli w interpolacji wielomianowej za węzły wybierze się zera wielomianów Czebyszewa, to nie występuje efekt Rungego.
- **Baza aproksymacji** - wielomiany Czebyszewa można wybrać jako bazę aproksymacji układu normalnego z węzłami w zerach tych wielomianów, otrzymuje się wtedy aproksymację jednostajną Czebyszewa z własnością minmaksu.
- **Całkowanie numeryczne** - w kwadraturach Gaussa lub Clenshowa-Curtisa do obliczenia optymalnych węzłów wykorzystuje się wielomiany Czebyszewa.
- **Iteracyjna niestacjonarna metoda Czebyszewa** - w rozwiązaniu układów równań liniowych metody niestacjonarne są szybciej zbieżne, zmieniają one macierz iteracji. Jedną z metod jest metoda Czebyszewa wykorzystująca wielomiany macierzowe Czebyszewa, co pozwala uniknąć obliczania iloczynów skalarnych macierzy.

8.8. Porównaj zasadę działania metod iteracyjnych do rozwiązywania równań nieliniowych i do rzwiązywania układów równań liniowych: ogólny algorytm, potrzebne twierdzenia, kiedy są przydatne

Metody iteracyjne - metody (algorytmy), których idea opiera się na ulepszaniu rozwiązania wraz z kolejnymi iteracjami.

Ogólny algorytm w przypadku równań nieliniowych:

- wybierz punkt startowy x_0
- oblicz kolejne przybliżenie za pomocą funkcji iteracyjnej:

$$x_i = \Phi(x_{i-1}) \quad i = 1, 2, ... \tag{8.70}$$

- sprawdź warunek stopu (np. maksymalna liczba iteracji, błąd bezwzględny):

$$\|x_i - x_{i-1}\| < \varepsilon \;\; \text{lub} \;\; i > i_{\max} \tag{8.71}$$

- zwróć x_i jako wynik po przerwaniu pętli

Ogólny algorytm w przypadku równań liniowych:

- wybierz wektor startowy $x^{(0)}$
- rozłóż macierz $A = B + R$, wyznacz macierz iteracji M oraz resztę W
- oblicz kolejne przybliżenie za pomocą macierzy iteracji:

$$x^{(i+1)} = M \cdot x^{(i)} + W \quad i = 1, 2, ... \tag{8.72}$$

- sprawdź warunek stopu (np. maksymalna liczba iteracji, błąd bezwzględny):

$$\left\|x^{(i+1)} - x^{(i)}\right\| < \varepsilon \;\; \text{lub} \;\; i > i_{\max} \tag{8.73}$$

- zwróć $x^{(i+1)}$ jako wynik po przerwaniu pętli

Do wykorzystywania metod iteracyjnych do konkretnych problemów potrzebne są twierdzenia zapewniające warunki zbieżności. Metody można stosować tylko dla problemów, dla których te warunki są spełnione, czyli iteracje będą dawać coraz lepsze rozwiązania.

Twierdzenie te to np. twierdzenie o zbieżności procesu iteracyjnego $Ax = b$ dla układów równań liniowych (Sekcja 8.2) lub o zbieżności procesu iteracyjnego $x = \Phi(x)$ dla równań nieliniowych (Sekcja 6.2).

Przydatność metod iteracyjnych:

- Dla równań nieliniowych:
  - Gdy nie ma analitycznego rozwiązania (np. równania transcendentalne).
  - Gdy pochodne są łatwe do obliczenia (Newton) lub gdy ich obliczenie jest trudne (metoda siecznych).
  - W implementacjach numerycznych, gdzie wystarczy przybliżone rozwiązanie.
- Dla układów równań liniowych:
  - Gdy macierz A jest duża i rzadka (np. w metodach elementów skończonych).
  - Gdy bezpośrednie metody (np. eliminacja Gaussa) są zbyt kosztowne obliczeniowo.
  - Gdy macierz ma specjalną strukturę (np. diagonalna dominacja), która gwarantuje zbieżność.

8.9. Porównaj rozwiązywanie układów równań liniowych metodami bezpośrednimi i iteracyjnymi

- **Metody bezpośrednie** dostarczają dokładne rozwiązanie układu równań w skończonej liczbie kroków (przy braku błędów zaokrągleń). Są one szczególnie efektywne dla układów o niewielkiej lub średniej wielkości oraz dla macierzy gęstych.

  Do metod bezpośrednich zaliczamy eliminacje Gaussa, rozkład LU, rozkład Cholesky'ego

- **Metody iteracyjne** generują ciąg przybliżeń, który zbiega do dokładnego rozwiązania. Są one szczególnie przydatne dla dużych i rzadkich układów równań, gdzie metody bezpośrednie byłyby zbyt kosztowne obliczeniowo.

  Do metod iteracyjnych zaliczamy metodę Jacobiego, Gaussa-Seidla (S-R), SOR, Czebyszewa.

Porównanie:

- Metody bezpośrednie dają zawsze dokładne rozwiązanie (z uwzględnieniem błędów obliczeniowych), podczas gdy metody iteracyjne oferują skończone przybliżenie rozwiązania
- Dla macierzy rzadkich metody iteracyjne zachowują ich strukturę, podczas gdy bezpośrednie ją zmieniają. W przypadku rzadkich, można wykorzystać to do przechowywania macierzy w efektywnych strukturach.
- Metody iteracyjne są zwykle stabilne (błędy zaokrągleń są wygaszane w trakcie obliczeń), podczas gdy metody bezpośrednie nie mają takich właściwości.
- Dla macierzy pełnych metody bezpośrednie są szybsze od iteracyjnych.

9. Równania różniczkowe zwyczajne

9.1. Omów metodę Eulera rozwiązywania równań różniczkowych zwyczajnych

Równanie modelowe - matematyczna reprezentacja zjawiska, procesu lub systemu, która opisuje jego zachowanie za pomocą zmiennych, parametrów i zależności między nimi

Przyjmijmy oznaczenia:
- $u(t)$ - funkcja zależna od czasu t
- t^n czas w n-tym kroku obliczeniowym (dyskretny punkt na osi czasu):
  - t^0 - czas początkowy
  - $t^1 = t^0 + \Delta t$
- Δt - krok czasowy
- u^n - Wartość funkcji u w czasie t^n, czyli $u^n = u(t^n)$

Mając dany model:

$$\begin{cases} \frac{du}{dt} + f(u,t) = 0 \quad \text{, gdzie } u = u(t) \\ \text{warunki początkowe : } u(t^0) = u^0 \end{cases} \tag{9.1}$$

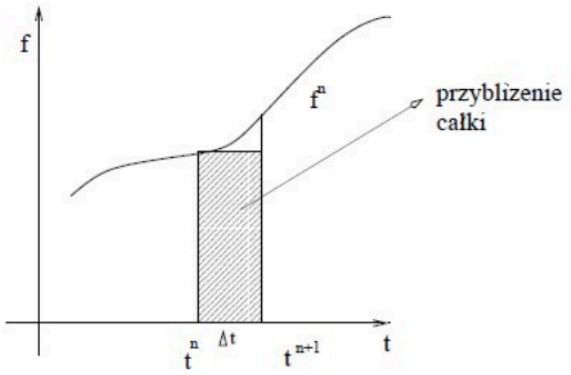
Równanie modelowe można scałkować na siatce czasowej:

$$\Delta t = t^{n+1} - t^n \tag{9.2}$$

$$u^{n+1} = u^n - \int_{t^n}^{t^{n+1}} f(u,t)dt \tag{9.3}$$

Metoda Eulera

$f(u,t) = ?$ dla $t \in [t^n, t^{n+1}] \approx f(u^n, t^n)$



W metodzie Eulera mając oryginalne równanie modelowe:

$$\frac{du}{dt} + f(u,t) = 0 \tag{9.4}$$

Przybliżamy pochodną $\frac{du}{dt}$ ilorazem różnicowym:

$$\frac{du}{dt} \approx \frac{u^{n+1} - u^n}{\Delta t} \tag{9.5}$$

$$\frac{u^{n+1} - u^n}{\Delta t} + f(u^n, t^n) = 0 \tag{9.6}$$

Stąd:

$$u^{n+1} = u^n - \Delta t \cdot f(u^n, t^n) \tag{9.7}$$

Właściwości:

- stabilna, jeżeli $\frac{\partial f}{u} > 0$
- pierwszego rzędu - błąd zmienia się liniowo ze względu na Δt: $\varepsilon = O(\Delta t)$
- jawna, prosta, efektywna

9.2. Omów sposób badania stabilności metod rozwiązywania równań różniczkowych zwyczajnych (ODE) na podstawie metody Eulera. Podaj przykłady

Stabilność - własność metod numerycznych, która określa, czy rozwiązanie obliczone przy użyciu danego schematu pozostaje ograniczone w miarę postępu obliczeń.

Metoda Eulera dla równania $\frac{du}{dt} + f(u,t) = 0$ ma postać:

$$u^{n+1} = u^n - \Delta t \cdot f(u^n, t^n) \tag{9.8}$$

Niech ε^n oznacza błąd numeryczny w kroku t^n.
W czasie t^n mamy u^n z błędem ε^n.
W czasie t^{n+1} mamy u^{n+1} z błędem ε^{n+1}

$$\varepsilon^{n+1} = g \cdot \varepsilon^n$$
$$g \text{ - współczynnik wzmocnienia błędu} \tag{9.9}$$

Gdy uwzględnimy błąd ε^n, równanie z zaburzonym rozwiązaniem staje się:

$$u^{n+1} + \varepsilon^{n+1} = u^n + \varepsilon^n - \Delta t \cdot f(u^n + \varepsilon^n, t^n) \quad (*) \tag{9.10}$$

Możemy teraz rozwinąć $f(u^n + \varepsilon^n, t^n)$ w szereg Taylora wokół u^n. Zakładamy, że ε jest mały, więc pomijamy wyrazy nieliniowe:

$$f(u^n + \varepsilon^n, t^n) \approx f(u^n, t^n) + \underbrace{\left.\frac{\partial f}{\partial u}\right|_{u^n}}_{\substack{\text{cząstkowa } f \\ \text{w punkcie } u^n}} \cdot \varepsilon^n \tag{9.11}$$

Po podstawieniu do $(*)$ mamy:

$$u^{n+1} + \varepsilon^{n+1} = u^n + \varepsilon^n - \Delta t \cdot \left(f(u^n, t^n) + \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \varepsilon^n \right) \tag{9.12}$$

$$\varepsilon^{n+1} = \varepsilon^n - \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \varepsilon^n \cdot \Delta t \tag{9.13}$$

Stąd, możemy wyprowadzić współczynnik wzmocnienia:

$$g = \frac{\varepsilon^{n+1}}{\varepsilon^n} = 1 - \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \Delta t \tag{9.14}$$

Warunkiem stabilności w rozwiązywaniu równań różniczkowych jest:

$$|g| \leq 1 \tag{9.15}$$

Jeżeli:

$$\left.\frac{\partial f}{\partial u}\right|_{u^n} < 0 \rightarrow \text{metoda jest niestabilna} \tag{9.16}$$

Dla

$$\left.\frac{\partial f}{\partial u}\right|_{u^n} > 0 \tag{9.17}$$

Warunkiem stabilności jest:

$$\left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \Delta t \leq 2 \quad \rightarrow \quad \Delta t \leq \frac{2}{\left.\frac{\partial f}{\partial u}\right|_{u^n}} \tag{9.18}$$

Pryzkłady zastosowania:

- **rozpad promieniotwórczy**

$$\frac{du}{dt} + \frac{u}{\tau} = 0, \quad u(0) = 1 \tag{9.19}$$

Rozwiązanie analityczne:

$$u = e^{-\frac{t}{\tau}} \tag{9.20}$$

Krok czasowy gwarantujący stabilność:

$$\Delta t \leq \frac{2}{\frac{\partial f}{\partial u}\Big|_{u^n}} \tag{9.21}$$

$$\frac{\partial f}{\partial u}\Big|_{u^n} = \frac{1}{\tau} \qquad \Delta t \leq 2\tau \tag{9.22}$$

- **oscylator harmoniczny**

$$\frac{d^2x}{dt^2} + \omega^2 x = 0 \tag{9.23}$$

Z powyższego otrzymujemy układ dwóch równań:

$$v = \frac{dx}{dt} \tag{9.24}$$

$$\begin{cases} \frac{dv}{dt} + \omega^2 x = 0 \\ \frac{dx}{dt} - v = 0 \;|\cdot(-i\omega) \end{cases} \Rightarrow \begin{cases} \frac{dv}{dt} + \omega^2 x = 0 \\ -i\omega\frac{dx}{dt} + i\omega v = 0 \end{cases} \tag{9.25}$$

Po dodaniu:

$$\frac{dv}{dt} - i\omega\frac{dx}{dt} + \omega^2 x + i\omega v = 0 \tag{9.26}$$

$$\frac{dv}{dt} - i\omega\frac{dx}{dt} - \boldsymbol{i^2}\omega^2 x + i\omega v = 0 \tag{9.27}$$

$$\frac{dv}{dt} - i\omega\frac{dx}{dt} + i\omega(-i\omega x + v) = 0 \tag{9.28}$$

Wprowadzamy $u = v - i\omega x$ co daje pojedyńcze równanie pierwszego rzędu:

$$\frac{du}{dt} + i\omega u = 0 \tag{9.29}$$

Współczynnik wzmocnienia:

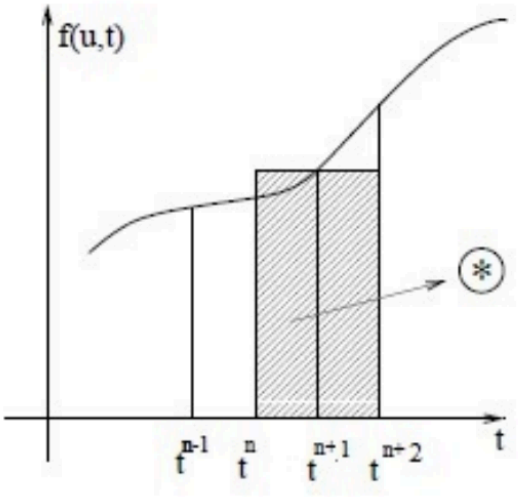
$$f(u) = i\omega u \tag{9.30}$$

$$g = 1 - \frac{\partial f}{\partial u}\Big|_{u^n} \cdot \Delta t \quad \Rightarrow \quad g = 1 - i\omega \cdot \Delta t \tag{9.31}$$

$$|g|^2 = g \cdot g^* = 1 + \omega^2 + \Delta t^2 \quad \Rightarrow |g| > 1 \quad (**) \tag{9.32}$$

Z $(**)$ można wnioskować, że metoda Eulera jest niestabilna dla równań typu oscylacyjnego. W zagadnieniach nieliniowych należy na każdym etapie wybierać Δt spełniające warunki stabilności.

9.3. Omów metodę skokową rozwiązywania równań różniczkowych zwyczajnych



W **metodzie skokowej**:

- całkujemy po kroku czasowym o podwójnej długości
- punkt czasowy w środku takiego kroku używamy dla obliczenia całki metodą prostokątów
- metoda wycentrowana w czasie $\rightarrow$ dokładność drugiego rzędu $\varepsilon = O(\Delta t^2)$

$$u^{n+1} = u^{n-1} - 2\Delta t \cdot f(u^n, t^n) \tag{9.33}$$

$$u^{n+2} = u^n - \underbrace{2\Delta t \cdot f(u^{n+1}, t^{n+1})}_{*} \tag{9.34}$$

W metodzie tej potrzebne są dwie wartości u^{n-1} i u^n do obliczenia u^{n+1}. Z początku znamy $u^0 = u(0)$ (warunek początkowy), ale potrzebne jest nam $u^1 = u(\Delta t)$, od którego w praktyce zależy całkowita dokładność. Metody wyznaczania u^1 to:

- metoda Eulera w pierwszym kroku
- rozwinięcie Taylora w celu oszacowania

Dodatkowo, jeżeli zagadnienie jest nieliniowe (a więc Δt jest zmienne) metoda przestaje być wycentrowana w czasie, co może prowadzić do niestabilności.

Stabilność metody:

$$\varepsilon^{n+1} = \varepsilon^{n-1} - \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot 2 \cdot \Delta t \cdot \varepsilon^n \tag{9.35}$$

$$g = \frac{\varepsilon^{n+1}}{\varepsilon^n} \approx \frac{\varepsilon^n}{\varepsilon^{n-1}} \qquad g^2 = \frac{\varepsilon^{n+1}}{\varepsilon^n} \cdot \frac{\varepsilon^n}{\varepsilon^{n-1}} \tag{9.36}$$

Podstawmy:

$$\alpha = \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \Delta t \tag{9.37}$$

Wtedy:

$$g^2 = 1 - \alpha \cdot 2g \quad \Rightarrow \quad g = -\alpha \pm \sqrt{\alpha^2 + 1} \tag{9.38}$$

Mamy więc dwa pierwiastki:

- jeśli α - rzeczywiste, to $|g| > 1$, więc metoda jest niestabilna
- jeśli α - urojone i równe $i\beta$, gdzie $\beta \leq 1$, to:

$$g = -i\beta \pm \sqrt{1-\beta^2} \qquad |g|^2 = g \cdot g^* = 1 \tag{9.39}$$

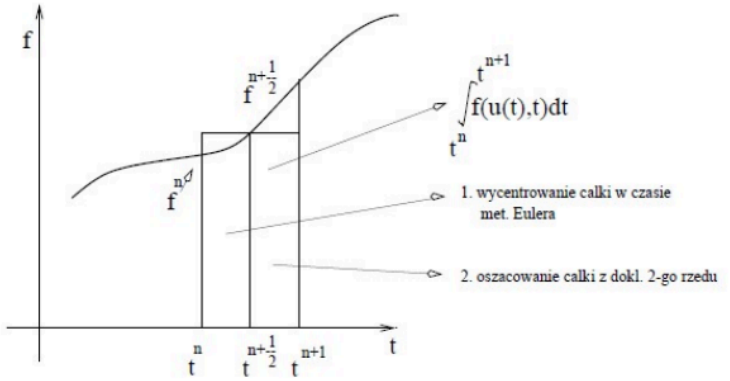
Przykład:

- **równanie oscylatora**

$$\frac{du}{dt} + i\omega u = 0 \quad \Rightarrow \quad \beta = \omega \Delta t \tag{9.40}$$

$$\beta \leq 1 \quad \Rightarrow \quad \Delta t \leq \frac{1}{\omega} \tag{9.41}$$

9.4. Omów metodę ulepszoną Eulera rozwiązywania równań różniczkowych zwyczajnych



$u^{n+\frac{1}{2}}$- pomocnicze, nie zachowywane powyżej t^{n+1}

Ulepszona metoda Eulera modyfikuje oryginalną metodę, dzieląc ją na dwa kroki:

- wyliczenie zmiennej u dla pośredniego $t^{n+\frac{1}{2}}$ metodą Eulera:

$$u^{n+\frac{1}{2}} = u^n - \frac{\Delta t}{2} \cdot f(u^n, t^n) \quad \text{(wzór pomocniczy)} \tag{9.42}$$

- wyliczenie właściwej własności u^{n+1} na podstawie $u^{n+\frac{1}{2}}$ z poprzedniego kroku

$$u^{n+1} = u^n - \Delta t \cdot f\left(u^{n+\frac{1}{2}}, t^{n+\frac{1}{2}}\right) \tag{9.43}$$

Metoda ta jest jawna - w każdym kroku znamy wszystkie wartości do wyznaczenia kolejnego kroku wprost.

Stabilność metody:

$$\varepsilon^{n+1} = \varepsilon^n = \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \Delta t \cdot \left[1 - \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \frac{\Delta t}{2}\right] \varepsilon^n \tag{9.44}$$

Podstawiając:

$$\alpha = \left.\frac{\partial f}{\partial u}\right|_{u^n} \cdot \Delta t \tag{9.45}$$

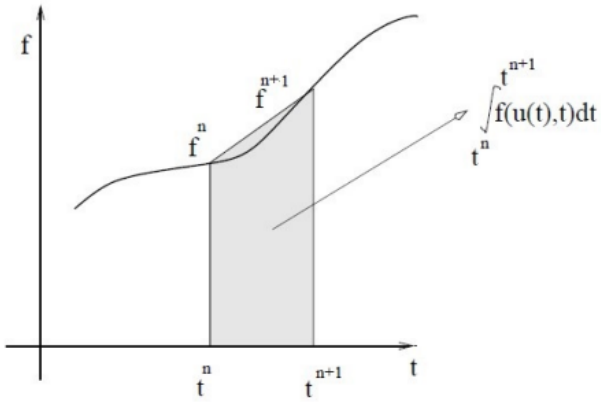
Otrzymujemy:

$$\varepsilon^{n+1} = \varepsilon^n - \alpha\left[1 - \frac{\alpha}{2}\right]\varepsilon^n \tag{9.46}$$

$$g = 1 - \alpha + \frac{\alpha^2}{2} \tag{9.47}$$

- metoda jest stabilna dla α rzeczywistego, gdy $\Delta t \leq \frac{2}{\frac{\partial f}{\partial u}\Big|_{u^n}}$
- może być niestabilna dla α urojonego

9.5. Omów niejawną metodę drugiego rzędu rozwiązywania ODE. Porównaj z metodami jawnymi: Eulera i ulepszonego Eulera



Metodę niejawną drugiego rzędu stanowi poniższy algorytm:

$$u^{n+1} = u^n - \frac{\Delta t}{2} \cdot \left[f(u^n, t^n) + f(u^{n+1}, t^{n+1})\right] \tag{9.48}$$

- u^{n+1} - uwikłane, pojawia się po obydwu stronach równania i jest niejawnym argument funkcji f, której wartości potrzebujemy, by go wyliczyć
- gdy f jest bardziej skomplikowana, to w każdym kroku czasowym może wymagać rozwiązania równania
- metoda ma dokładność drugiego rzędu

Stabilność metody:

$$g = 1 - \frac{\partial f}{\partial u}\Big|_{u^n} \cdot \frac{\Delta t}{2} - \frac{\partial f}{\partial u}\Big|_{u^{n+1}} \cdot \frac{\Delta t}{2} \cdot g \tag{9.49}$$

$$g = \frac{1 - \frac{\partial f}{\partial u}\Big|_{u^n} \cdot \frac{\Delta t}{2}}{1 + \frac{\partial f}{\partial u}\Big|_{u^{n+1}} \cdot \frac{\Delta t}{2}} \tag{9.50}$$

Przykłady:

- równanie rozpadu $\frac{\partial f}{\partial u} > 0, \quad |g| < 1$ (zawsze)
- równanie oscylacyjne $\frac{\partial f}{\partial u}$ - urojone, $|g| = 1$

W obu przypadkach metoda jest stabilna niezależnie od wyboru kroku.

- bezwzględnie stabilna, co jest ważne w zagadnieniach nieliniowych
- cena: konieczność rozwiązywania algebraicznego na u^{n+1} lub stosowania wzoru iteracyjnego

Porównanie:

- niejawna metoda jest bezwzględnie stabilna, podczas gdy metody jawne - tylko warunkowo
- w przeciwieństwie do metod jawnych, w których wystarczy obliczyć wartość funkcji, metoda niejawna wymaga rozwiązania równania nieliniowego w każdym kroku
- metoda niejawna, przez potrzebę rozwiązania równania nieliniowego, jest znacznie wolniejsza niż metoda Eulera
- metoda niejawna nadaje się do sztywnych równań i problemów wymagających stabilności nawet przy dużych krokach czasowych

9.6. Przedstaw ogólną zasadę konstruowania metod Rungego Kutty. Podaj związki z metodą Eulera oraz ulepszonego Eulera

Metoda Rungego-Kutty służy przybliżaniu rozwiązania równania różniczkowego w kolejnych punktach t oddalonych od siebie o krok czasowy h:

$$t_0, t_1 = t_0 + h, ..., t_n = t_0 + nh \tag{9.51}$$

Stąd, wartość u_{i+1} jest aproksymowana na podstawie r kombinacji ważonej wielu przybliżeń pośrednich, gdzie r to liczba etapów. W szczególności, dla $r = 1$ metoda Rungego-Kutty odpowiada metodzie Eulera.

Niech będzie dane równanie:

$$\frac{du(t)}{dt} = f(t, u), \quad t \in [a, b] \quad u_0 = u(t_0) \tag{9.52}$$

Rozwiązanie $u(t)$ wyznaczamy w punktach:

$$t_i = t_0 + i \cdot h \quad h = \frac{b-a}{n}, i = 0, 1, ..., n \tag{9.53}$$

Wzór metody

Wprowadźmy oznaczenia:

- u_i - przybliżenie rozwiązania w t_i
- $f(t, u)$ - funkcja z rozwiązywanego równania różniczkowego
- h - krok czasowy (całkowania)
- k_j - przybliżenie pośrednie (nachylenia f w różnych punktach $[t_i, t_{i+1}]$)
- $F(t, u, h)$ - funkcja przyrostu, będąca kombinacją liniową r współczynników k_j:

$$F(t, u, h) = c_1 k_1 + c_2 k_2 + ...c_r k_r \quad c_i \in \mathbb{R} \tag{9.54}$$

- F_T - dokładne rozwinięcie Taylora funkcji F wokół t, u
- c_k, a_j, b_{js} - stałe metody, dobierane tak, aby błąd lokalny metody (funkcja Φ):

$$\Phi(h) = F(t, u, h) - F_T(t, u, h) \tag{9.55}$$

  zawierała jedynie potęgi h możliwie wysokiego rzędu

Metodę Rungego-Kutty rzędu r można zdefiniować w sposób nastepujący:

$$\begin{cases} u_{i+1} = u_i + h \cdot F(t_i, u_i, h) \quad i = 0, 1, 2, ..., n-1 \\ F(t, u, h) = c_1 k_1(t, u, h) + c_2 k_2(t, u, h) + ... + c_r k_r(t, u, h) \\ k_1(t, u, h) = f(t, u) \\ k_j(t, u, h) = f\left(t + h a_j, u + h \cdot \sum_{s=1}^{j-1} b_{js} k_s(t, u, h)\right) \quad j = 2, ..., r \\ a_j = \sum_{s=1}^{j-1} b_{js} \end{cases} \tag{9.56}$$

Metoda Eulera i metoda ulepszonego Eulera są szczególnymi przypadki metody Rungego-Kutty:

- **Metoda Euelera**:
  - $r = 1$
  - $c_2 = c_3 = a_2 = a_3 = b_{32} = 0$
  - c_1 = 1

$$u_{i+1} = u_i - h \cdot f(t_i, u_i) \tag{9.57}$$

- **Metoda Ulepszonego Eulera:**
  - $r = 2$
  - $c_1 = c_3 = b_{32} = 0$
  - $c_2 = 1$
  - $a_2 = \frac{1}{2}$

$$\begin{gathered} u_{i+1} = u_i + \frac{h}{2} \cdot (k_1 + k_2) \\ k_1 = f(t_i, u_i) \quad k_2 = f(t_i + h, u_i + h \cdot k_1) \end{gathered} \tag{9.58}$$

Metoda Rungego-Kutty w kontekście układów równań ODE

Niech będą dane wektory:

$$\begin{cases} \vec{u}(t) = (u_1(t), u_2(t), ..., u_m(t)) \\ \vec{f}(t, \vec{u}) = \left(f_1(t, \vec{u}), f_2(t, \vec{u}), ..., f_{m(t,\vec{u})}\right)\vec{u_0} \end{cases} \tag{9.59}$$

oraz okład równań:

$$\begin{cases} \frac{d}{dt}\vec{u}(t) = \vec{f}(t, \vec{u}(t)) \quad t \in [a, b] \\ \text{warunek początkowy } \vec{u}(t_0) = \vec{u_0} \end{cases} \tag{9.60}$$

Metoda Rungego-Kutty ma wtedy postać:

$$\vec{u}_{i+1} = \vec{u}_i + h \cdot \vec{F}(t_i, \vec{u}_i, h) \tag{9.61}$$

gdzie:

$$\begin{cases} \vec{F}(t, \vec{u}, h) = c_1 \cdot \vec{k}_1(t, \vec{u}, h) + c_2 \cdot \vec{k}_2(t, \vec{u}, h) + ... + c_r \cdot \vec{k}_r(t, \vec{u}, h) \\ \vec{k}_1(t, \vec{u}, h) = \vec{f}(t, \vec{u}) \\ \vec{k}_j(t, \vec{u}, h) = \vec{f}\left(t + h \cdot a_{j1} \cdot \vec{u} + h \cdot \sum_{s=1}^{j-1} b_{js} \cdot \vec{k}_s(t, \vec{u}, h)\right) \quad j = 2, 3, ..., r \end{cases} \tag{9.62}$$

$$a_j = \sum_{s=1}^{j-1} b_{js}, \quad j = 2, 3, ..., r \tag{9.63}$$

a_j, b_{js}, c_r - stałe rzeczywiste, wartości takie same jak dla pojedynczego równania.

10. Fast Fourier Transfrom

10.1. Objaśnij przydatność transformat Fouriera, podaj ich główne rodzaje

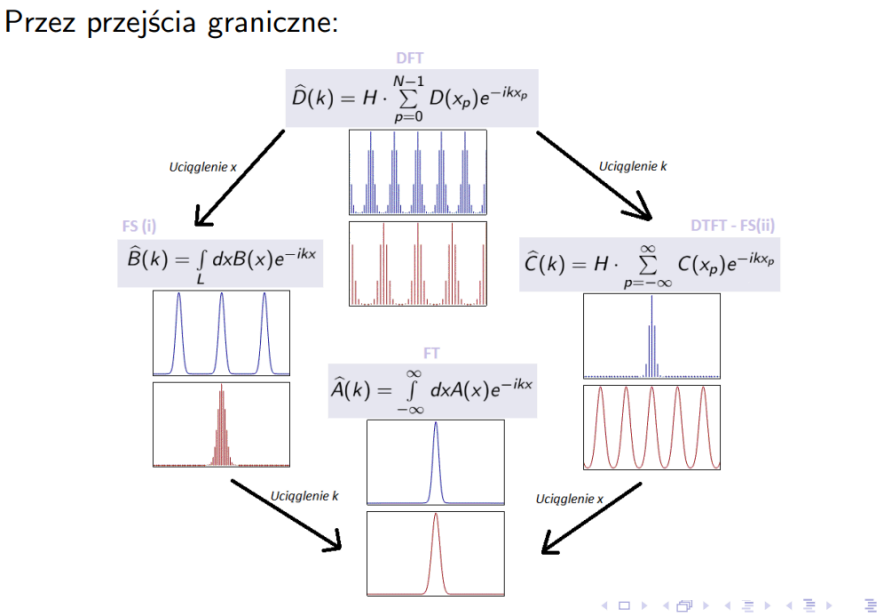
Transformata Fouriera przekształca funkcję czasu (lub przestrzeni) w funkcję częstotliwości. Innymi słowy, pokazuje, jakie częstotliwości (sinusoidy) składają się na analizowany sygnał.

Zastosowanie transformat Fouriera:

- metody spektralne:
  - fizyka, chemia: badanie właściwości atomów, cząstek
  - analiza widma promieniowania elektromagnetycznego
- algorytmy numeryczne:
  - równania różniczkowe
  - analiza: badanie jakości algorytmów (np. dla MES)
  - FFT używane do szybkiego mnożenia wielomianów
- cyfrowe przetwarzanie sygnału:
  - badanie składowych harmonicznych
  - filtracja obrazów i dźwięku
  - kompresja

Rodzaje:

- transformata Fouriera (FT)
- inverse Fourier Transform (IFT, FS (i)) - transformata odwrotna (stosowany szereg Fouriera)
- Discrete-Time Fourier Transform (DTFT)
- skończonona (dyskretna) transformata Fouriera (finite FT, fFT DFT)



Rysunek 27: Związki pomiędzy transformatami Fouriera

10.2. Objaśnij, na czym polega interpolacja trygonometryczna, kiedy ją warto stosować, jaki jest jej związek z dyskretną transformatą Fouriera

Interpolacja trygonometryczna stosowana jest w związku z faktem, że wielomiany algebraiczne nie są dobre do opisu zjawisk okresowych. Rozwiązaniem jest interpolacja wielomianami opartymi o funkcje trygonometryczne. Zadanie interpolacji trygonometrycznej ma jednoznaczne rozwiązanie.

Niech L będzie okresem funkcji okresowej g. Prawdziwa wówczas jest właściwość:

$$g(y+L) = g(y) \tag{10.1}$$

Dla funkcji trygonometrycznych okresem jest 2π. Można dokonać skalowania tego okresu, wprowadzając nową zmienną x o okresie 2π:

$$\begin{gathered} x = \frac{2\pi}{L} \cdot y \;\Rightarrow\; y = \frac{x \cdot L}{2\pi} \\ f(x) = g\left(\frac{x \cdot L}{2\pi}\right) \end{gathered} \tag{10.2}$$

W ten sposób, jeżeli L jest okresem funkcji g:

$$f(x+2\pi) = g\left(\frac{(x+2\pi)\cdot L}{2\pi}\right) = g\left(\frac{x\cdot L}{2\pi} + L\right) = g\left(\frac{x\cdot L}{2}\pi\right) = f(x) \tag{10.3}$$

Stąd 2π jest okresem funkcji f.

W interpolacji trygonometrycznej szukamy wielomianu trygonometrycznego:

$$t_{n-1}(x) = \sum_{j=0}^{n-1} c_j \cdot \left(e^{ix}\right)^j = \sum_{j=0}^{n-1} c_j \cdot e^{ijx} \tag{10.4}$$

, gdzie:

- współczynniki $c_j \in \mathbb{C}$
- e^{ix} ze wzoru Eulera wynosi:

$$e^{ix} = \cos(x) + i\sin(x) \tag{10.5}$$

$$e^{ijx} = \cos(jx) + i\sin(jx) \tag{10.6}$$

Wielomian ten w n punktach $x_k \in (0, 2\pi]$ przyjmuje te same wartości co interpolowana funkcja:

$$t_{n-1}(x_k) = f(x_k) \quad k = 0, 1, ..., n-1 \tag{10.7}$$

Ważny w metodach numerycznych przypadek szczególny:

Rozważmy interpolację trygonometryczną o n **węzłach równoodległych**:

$$x_k = \frac{2\pi}{n} \cdot k, k = 0, 1, ..., n-1 \tag{10.8}$$

Dla takich węzłów funkcje:

$$e^{ijx} \quad j = 0, 1, ..., n-1 \tag{10.9}$$

tworzą układ ortogonalny w sensie iloczynu skalarnego zdefiniowanego jako:

$$\begin{gathered} \langle f \mid g \rangle = \sum_{k=0}^{n-1} f(x_k) \cdot g^*(x_k) \\ x_k = \frac{2\pi}{n} \cdot k \quad k = 0, 1, ..., n-1 \\ g^* \text{ - sprzężenie zespolone funkcji } g \end{gathered} \tag{10.10}$$

Dokładniej:

$$\begin{gathered}\langle e^{ijx} \mid e^{ilx} \rangle = \sum_{k=0}^{n-1} e^{ijx_k} \cdot e^{-ilx_k} = n \cdot \delta_{j,l} = \begin{cases} 0 & j \neq l \\ n & j = l \end{cases} \\ x_k = \frac{2\pi}{n} \cdot k \quad k = 0, 1, ..., n-1 \end{gathered} \tag{10.11}$$

gdzie $\delta_{j,l}$ to delta Kroneckera:

$$\delta_{j,l} = \begin{cases} 0 & j \neq l \\ 1 & j = l \end{cases} \quad j, l \in \{0, 1, ..., n-1\} \tag{10.12}$$

Biorąc pod uwagę powyższy iloczyn skalarny, możemy zdefiniować odpowiednie c_j współczynniki wielomianu interpolacyjnego:

$$\textbf{I.} \qquad \langle t_{n-1}(x) \mid e^{ilx} \rangle = \sum_{j=0}^{n-1} c_j \cdot \langle e^{ijx} \mid e^{ilx} \rangle = \sum_{j=0}^{n-1} c_j \cdot n \cdot \delta_{j,l} = c_l \cdot n \tag{10.13}$$

$$c_l = \frac{1}{n} \langle t_{n-1}(x) \mid e^{ilx} \rangle \tag{10.14}$$

Z drugiej strony, iloczyn skalarny można zapisać jako (wiedząc że węzły są równoodległe oraz dla węzłów zachodzi $t_{n-1}(x_k) = f(x_k)$):

$$\textbf{II.} \qquad \langle t_{n-1}(x) \mid e^{ilx} \rangle = \sum_{k=0}^{n-1} t_{n-1}(x_k) \cdot e^{-ilx_k} = \sum_{k=0}^{n-1} f(x_k) \cdot e^{-ilx_k} \tag{10.15}$$

Porównując **I** i **II** mamy:

$$c_l \cdot n = \sum_{k=0}^{n-1} f(x_k) e^{ilx_k} \quad \Rightarrow \quad c_l = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) e^{-ilx_k} \tag{10.16}$$

Stąd ostatecznie wzór na współczynniki:

$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) e^{-ijx_k} \quad j = 0, 1, ..., n-1 \tag{10.17}$$

Tak więc interpolacja trygonometryczna sprowadza się do dwóch etapów obliczeń:

- **analiza Fouriera** - szukamy współczynników c_j, gdzie $j = 0, 1, ..., n-1$ dla danych liczb zespolonych $f(x_k) k = 0, 1, ..., n-1$.
- **synteza Fouriera** - mając współczynniki c_j szukamy:

$$\begin{gathered} f(x) = \sum_{j=0}^{n-1} c_j \cdot e^{ijx} \\ x_k = \frac{2\pi}{n} \cdot k \quad k = 0, 1, ..., n-1 \end{gathered} \tag{10.18}$$

Interpolacja trygonometryczna ma złożoność $O(n^2)$, co jest jej wadą.

10.3. Opisz własności funkcji stosowanych w interpolacji trygonometrycznej - w szczególności ortogonalność i jak z niej korzystamy

- bazą interpolacji trygonometrycznej są **wielomiany trygonometryczne**:

$$e^{ijx} \quad j = 0, 1, ..., n-1 \tag{10.19}$$

- szczególnie interesującym nas przypadkiem przy interpolacji trygonometrycznej jest przypadke węzłów równoodległych

$$x_k = \frac{2\pi}{n} \cdot k, k = 0, 1, ..., n-1 \tag{10.20}$$

- dla węzłów równoodległych funkcje e^{ijx} tworzą układ ortogonalny w sensie iloczynu skalarnego:

$$\begin{gathered} \langle f \mid g \rangle = \sum_{k=0}^{n-1} f(x_k) \cdot g^*(x_k) \\ x_k = \frac{2\pi}{n} \cdot k \quad k = 0, 1, ..., n-1 \end{gathered} \tag{10.21}$$

- dzięki iloczynowi skalarnemu zdefiniowanemu powyżej, jesteśmy w stanie dużo łatwiej wyznaczyć wagi c_j aproksmacji, gdyż odpowiednie wyrazy w iloczynie wzajemnie się zerują dzięki ortogonalności

$$\begin{gathered} \langle e^{ijx} \mid e^{ilx} \rangle = \sum_{k=0}^{n-1} e^{ijx_k} \cdot e^{-ilx_k} = n \cdot \delta_{j,l} = \begin{cases} 0 & j \neq l \\ n & j = l \end{cases} \\ x_k = \frac{2\pi}{n} \cdot k \quad k = 0, 1, ..., n-1 \end{gathered} \tag{10.22}$$

(Więcej w Sekcja 10.2)

10.4. Na czym polega FFT - szybka transformata Fouriera: przedstaw algorytm, podaj złożoność obliczeniową, porównaj z algorytmem klasycznym.

Algorytm FFT (Cooleya-Tukeya) to sposób szybkiego obliczania dyskretnej transformaty Fouriera (DFT) w złożoności $O(n \log n)$ używając metody divide-and-conquer

Dane:

$$\begin{gathered} f(x_k) \quad \text{oraz} \quad x_k = \frac{2\pi}{n} \cdot k \\ k = 0, 1, ..., n-1 \end{gathered} \tag{10.23}$$

Szukane:

$$\begin{gathered} c_j = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) \cdot e^{-ij\frac{2\pi k}{n}} \\ j = 0, 1, ..., n-1 \end{gathered} \tag{10.24}$$

Wprowadźmy oznaczenia:

$$a_k = \frac{1}{n} \cdot f(x_k) \quad \omega = e^{-i\frac{2\pi}{n}} \tag{10.25}$$

Wtedy:

$$c_j = \sum_{k=0}^{n-1} a_k \cdot \omega^{jk} \quad j = 0, 1, ..., n-1 \tag{10.26}$$

Ważnym założeniem w przypadku algorytmu Cooleya-Tukeya jest liczność punktów n:

$$n = 2^m \quad m \in \mathbb{N} \tag{10.27}$$

Liczba punktów musi być potęgą dwójki, aby można było w prosty sposób użyć metody divide-and--conquer, dzielącą główny problem o rozmiarze n na dwa podproblemy o rozmiarze $\frac{n}{2}$.

Istotą FFT jest wykorzystanie divide-and-conquer przez **rozbicie sumy na indeksy parzyste i nieparzyste**.

Wprowadźmy więc oznaczenia. Niech k będzie numerem punktu, wtedy:

$$k \text{ parzyste:} \quad k = 2 \cdot k_1 \tag{10.28}$$

$$k \text{ nieparzyste:} \quad k = 2 \cdot k_1 + 1 \tag{10.29}$$

Dla liczby punktów $n = 2^m$ mamy więc:

$$k_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.30}$$

Mając odpowiednie oznaczenie punktów parzystych i nieparzystych można rozdzielić wyznaczanie współczynników:

$$\begin{gathered} c_j = \sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1} \cdot \left(\omega^2\right)^{j \cdot k_1} + \sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1+1} \cdot \left(\omega^2\right)^{j \cdot k_1} \cdot \omega^j \\ k_1 = 0, 1, ... \frac{n}{2} - 1 \qquad j = 0, 1, ... n - 1 \end{gathered} \tag{10.31}$$

Następnie, wprowadźmy

$$j_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.32}$$

i użyjmy go aby podzielić indeksy j na dwa zbiory:

- mniejsze od $\frac{n}{2}$:

$$j = j_1 \tag{10.33}$$

- niemniejsze od $\frac{n}{2}$:

$$j = j_1 + \frac{n}{2} \tag{10.34}$$

Na wykładzie zostało to opisane w z użyciem nowej zmiennej l:

$$\begin{gathered} j = \frac{2}{n} \cdot l + j_1 \\ \begin{cases} l = 0 \text{ dla } j = 0, 1, ..., \frac{n}{2} - 1 \\ l = 1 \text{ dla } j = \frac{n}{2}, \frac{n}{2} + 1, ..., n - 1 \end{cases} \end{gathered} \tag{10.35}$$

Dla tak podzielonych indeksów mamy:

$$\begin{gathered} \left(\omega^2\right)^{jk_1} = \omega^{2 \cdot [\frac{n}{2} \cdot l + j_1] \cdot k_1} = \omega^{n \cdot l \cdot k_1 + 2 j_1 k_1} = e^{\left(-i\frac{2\pi}{n}\right)^{n \cdot l \cdot k_1 + 2 j_1 k_1}} = \left(\omega^2\right)^{j_1 k_1} \\ \text{bo } e^{-2\pi \cdot i \cdot l \cdot k_1} = 1 \end{gathered} \tag{10.36}$$

oraz

$$\omega^j = \omega^{\frac{n}{2} \cdot l + j_1} = \left(e^{-i\frac{2\pi}{n}}\right)^{\frac{n}{2} \cdot l} \cdot \omega^{j_1} = e^{-i\pi l} \cdot \omega^{j_1} \tag{10.37}$$

Stąd:

$$\begin{cases} \omega^j = \omega^{j_1} \text{ dla } j = 0, 1, ..., \frac{n}{2} - 1 \\ \omega^j = \omega^{-j_1} \text{ dla } j = \frac{n}{2}, \frac{n}{2} + 1, ..., n - 1 \end{cases} \tag{10.38}$$

Teraz, wstawiając do wyprowadzonych wcześniej wzorów:

- Dla $j = 0, 1, ...\frac{n}{2} - 1$:

$$c_j = \underbrace{\sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1} \cdot \left(\omega^2\right)^{j_1 k_1}}_{\varphi(j_1)} + \underbrace{\sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1+1} \cdot \left(\omega^2\right)^{j_1 k_1}}_{\psi(j_1)} \cdot \omega^{j_1}, \quad j_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.39}$$

- Dla $j = \frac{n}{2}, \frac{n}{2} + 1, ..., n - 1$

$$c_j = \underbrace{\sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1} \cdot \left(\omega^2\right)^{j_1 k_1}}_{\varphi(j_1)} - \underbrace{\sum_{k_1=0}^{\frac{n}{2}-1} a_{2k_1+1} \cdot \left(\omega^2\right)^{j_1 k_1}}_{\psi(j_1)} \cdot \omega^{j_1}, \quad j_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.40}$$

Można zauważyć, że każdy z dwóch wyprowadzonych wzorów jest ransformatą Fouriera. W ten sposób, zamiast pojedyńczej transformaty w n punkatch otrzymaliśmy sumę dwóch transformat w $\frac{n}{2}$ punktach wykorzystywanych dwa razy

- Dla $j = 0, 1, ..., \frac{n}{2} - 1$:

$$c_j = \varphi(j_1) + \omega^{j_1} \cdot \psi(j_1) \quad j_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.41}$$

- Dla $j = \frac{n}{2}, \frac{n}{2} + 1, ..., n - 1$:

$$c_j = \varphi(j_1) - \omega^{j_1} \cdot \psi(j_1) \quad j_1 = 0, 1, ..., \frac{n}{2} - 1 \tag{10.42}$$

Zarówno $\varphi(j_1)$ i $\psi(j_1)$ to transformaty Fouriera przeprowadzane na $\frac{n}{2}$ punktów względem oryginalnej n--punktowej transformaty. Dokonując takich podziałów rekurencyjnie otrzymujemy złożoność $O(n \log n)$

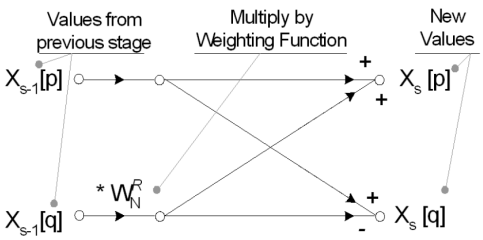
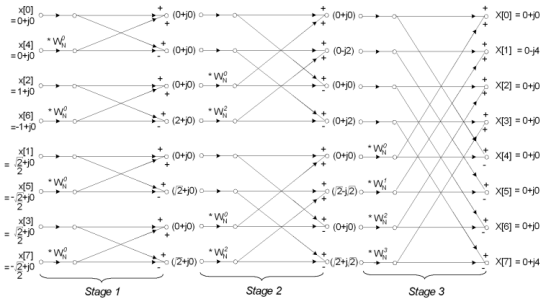


Figure: źródło: j.w.



$W_N^R = (e^{\frac{-2\pi i}{N}})^R$, W naszych oznaczeniach $N = n$, $R = j_1$, $W = \omega$

```
function FFT(a)
  n ← length[a]
  if n = 1
    then return a
  ω_n ← e^(2π·i/n)
  ω ← 1
  a_even ← (a_0, a_2, ..., a_{n-2})
  a_odd ← (a_1, a_3, ..., a_{n-1})
  y^even ← FFT(a_even)
  y^odd ← FFT(a_odd)
  for j ← 0 to n/2 − 1
    y_j ← y_j^even + ω y_j^odd
    y_{j+n/2} ← y_j^even − ω y_j^odd
    ω ← ω · ω_n
  end
  return y
end
```

Ogólnie polecam też artykuł na wikipedii: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

10.5. Pokaż jak działa algorytm FFT na przykładzie wyznaczania transformaty dla 8 punktów

Dane są wartości funkcji $f(x)$ w punktach $x = [x_0, x_1, ..., x_7]$. Szukamy współczynników postaci:

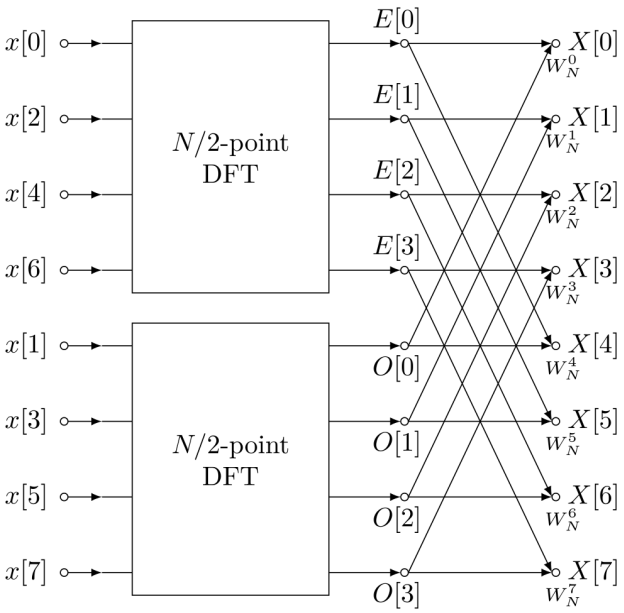
$$c_j = \frac{1}{n} \sum_{k=0}^{n-1} f(x_k) \cdot e^{-ij\frac{2\pi k}{n}} \tag{10.43}$$

Punkty trzeba ułożyć w odpowiedniej kolejności, a następnie zastosować transformaty Fouriera rekurencyjnie dla 4 i 2 elementów i połączyć wyniki.

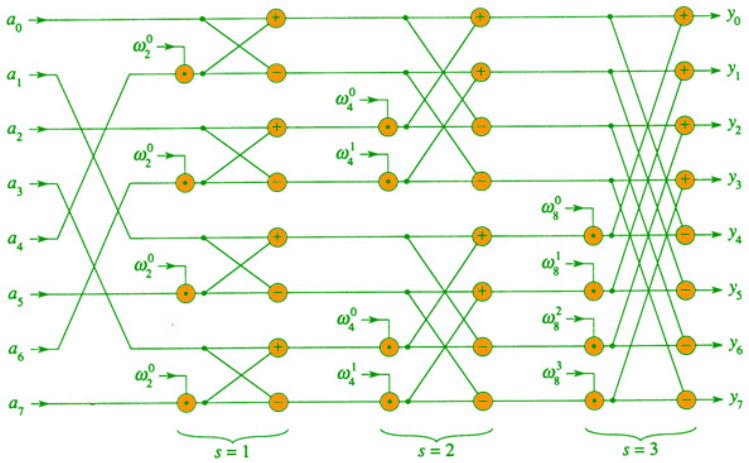
Na poniższym rysunku dla czytelności zapisano transformaty w kolejności finalnego obliczania i łączenia wyników.

Oznaczenia z pierwszego rysunku:

- x - tablica wartości funckji $f(x_k)$ w kolejnych punktach
- X - tablica wartości po dokonaniu transformaty w kolejnych punktach
- E - tablica wartości z x o indeksach parzystych
- O - tablica wartości z x o indeksach nieparzystych



Odsyłam ponownie do artykułu na wikipedii: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm



10.6. Opis zasadę „dziel i zwyciężaj” stosowaną w projektowaniu algorytmów na przykładzie algorytmu FFT

Metoda „dziel i zwyciężaj”polega na rozbiciu problemu na prostsze podproblemy w sposób rekurencyjny tak długo, aż fragmenty staną się wystarczająco proste do bezpośredniego rozwiązania. Rozwiązanie całego problemu otrzymuje się scalając rozwiązania podproblemów.

Algorytm FFT polega na podzieleniu transformaty wejściowej na dwie o połowę mniejsze transformaty rekurencyjnie. Otrzymuje się w ten sposób algorytm o złożoności $O(n \log n)$ zamiast $O(n^2)$.

Załóżmy, że aktualnie wykonujemy procedurę FFT. Mamy dane wartości $n = 2^m$ funkcji $f(x_k)$ w punktach $x_k = x_0, x_1, ..., x_{n-1}$. Przyjmijmy, że są one przechowywane w tablicy a. Mamy więc:

$$a = [f(x_0), f(x_1), ..., f(x_{n-1})] \tag{10.44}$$

i aktualnie wykonujemy :

$$\mathrm{FFT}(a) \tag{10.45}$$

Podzielmy tą tablicę na dwie części. Stwórzmy dwie tablice E i O o rozmiarze $\frac{n}{2}$, przechowujące elementy o indeksach odpowiednio parzystych i nieparzystych:

$$E = [f(x_0), f(x_2), ..., f(x_{n-2})] \quad O = [f(x_1), f(x_3), ..., f(x_{n-1})] \tag{10.46}$$

Następnie wykonajmy procedurę FFT na dwóch tych tablicach, dzieląc problem na dwa identyczne podproblemy:

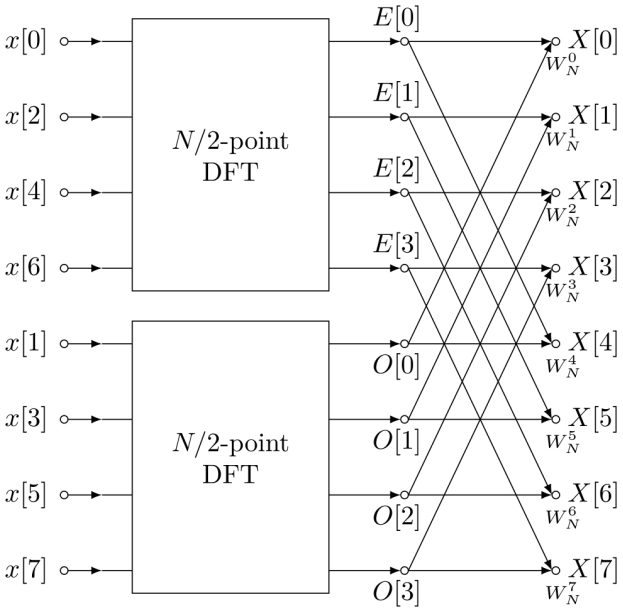
$$E_{\text{FFT}} = \text{FFT}(E) \quad O_{\text{FFT}} = \text{FFT}(O) \tag{10.47}$$

Stwórzmy nową tablicę a_{FFT} wielkości n (bądź nadpiszmy wartości a), która będzie zawierała finalne wartości obliczone zgodnie z algorytmem

$$\begin{cases} a_{\text{FFT}[j]} = E_{\text{FFT}}[j] + e^{-i\frac{2\pi j}{n}} \cdot O_{\text{FFT}}[j] \\ a_{\text{FFT}[j+\frac{n}{2}]} = E_{\text{FFT}}[j] - e^{-i\frac{2\pi j}{n}} \cdot O_{\text{FFT}}[j] \end{cases} \qquad j \in 0, 1, ... \frac{n}{2} - 1 \tag{10.48}$$

(https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm#Pseudocode)

Na koniec zwracamy a_{FFT} jako wynik.



10.7. Opisz zastosowanie FFT do algorytmu szybkiego mnożenia wielomianów

Wprowadźmy pojęcie splotu (konwolucji) dwóch funkcji f i g

$$h(x) = f * g \equiv \int_{-\infty}^{\infty} d\tau f(\tau) g(x - \tau) \tag{10.49}$$

, gdzie τ jest zmienną pomocniczą, po której całkujemy.

Własności:

- $f * g = g * f$ - przemienność
- $f * (g * h) = (f * g) * h$ - łączność
- $f * (g + h) = f * g + f * h$ - rozdzielność

Przykład operacji splotu dla dwóch wektorów kolumnowych:

$$a = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \dots \\ \alpha_{n-1} \end{bmatrix} \qquad b = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \dots \\ \beta_{n-1} \end{bmatrix} \tag{10.50}$$

$$[a * b]_k = \sum_{i=0}^{k} \alpha_i \cdot \beta_{k-i} \tag{10.51}$$

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{bmatrix} \quad \begin{bmatrix} \beta_{n-1} \\ \vdots \\ \beta_1 \\ \beta_0 \end{bmatrix} \downarrow \quad \Rightarrow \quad a * b = \begin{bmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 + \alpha_1\beta_0 \\ \alpha_0\beta_2 + \alpha_1\beta_1 + \alpha_2\beta_0 \\ \vdots \\ \alpha_{n-2}\beta_{n-1} + \alpha_{n-1}\beta_{n-2} \\ \alpha_{n-1}\beta_{n-1} \\ 0 \end{bmatrix}$$

Szybkie mnożenie wielomianów z FFT

Dane mamy:

$$P(x) = \sum_{i=0}^{n-1} a_i x^i \quad Q(x) = \sum_{i=0}^{n-1} b_i x^i \tag{10.52}$$

Jeśli potraktować $[a_i]$ oraz $[b_i]$ jako wektory, to $[c_i]$ postaci:

$$c_i = \sum_{j=0}^{i} a_j b_{i-j} \tag{10.53}$$

można zdefiniować jako ich splot. Mamy jednocześnie:

$$W(x) = P(x) \cdot Q(x) = \sum_{i=0}^{2n-2} c_i x^i \tag{10.54}$$

, gdzie współczynniki P i Q wyższe niż $n-1$ zastępujemy zerami.

FFT więc ma następującą właściwość:

$$\text{splot w dziedzinie czasu} \rightarrow \text{mnożenie w dziedzinie częstotliwości} \tag{10.55}$$

$$\text{mnozenie w dziedzinie czasu} \rightarrow \text{splot w dziedzinie częstotliwości} \tag{10.56}$$

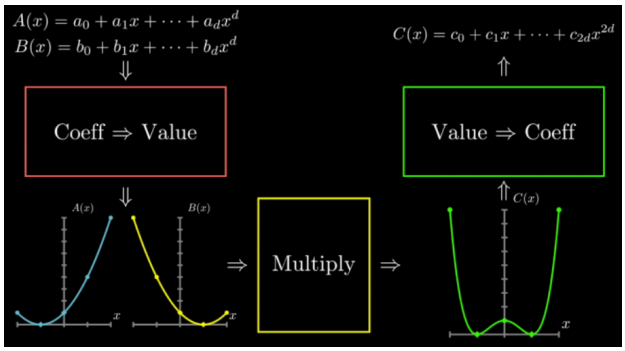
Zamiast wyliczać współczynniki wprost możemy więc użyć schematu:

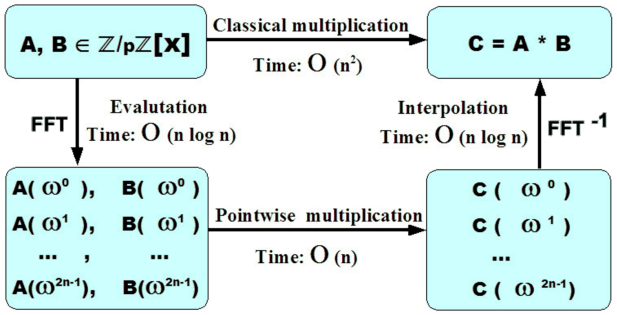
$$\text{transformata} \rightarrow \text{iloczyn} \rightarrow \text{odwrotna transformata} \tag{10.57}$$

- wyliczyć $2n-1$ wartości wielomianów $P(x_k)$ oraz $Q(x_k)$ dla

$$x_k = \omega^k = e^{i\frac{2\pi k}{2n-1}} \quad k = 0, \dots, 2n-2 \tag{10.58}$$

  używając FFT (synteza) w złożoności $O(n \log n)$
- policzyć wartości $W(x_k) = P(x_k) \cdot Q(x_k)$ w złożoności $O(n)$
- policzyć współczynniki c_i znając wartości $W(x_k)$ - inverse FFT (analiza) złożoność $O(n \log n)$



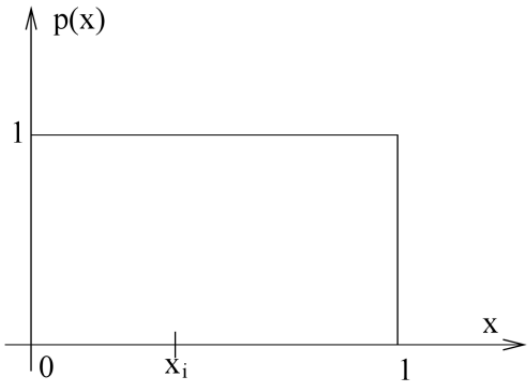


11. Liczby losowe i całkowanie Monte Carlo

11.1. Podaj przykłady i opisz działanie generatorów liczb z rozkładu równomiernego

Podstawowy typ generatora liczb to ten dla liczb o rozkładzie równomiernym.

$$P(x \in [a,b]) = \int_a^b p(x)dx \tag{11.1}$$



Generatory liczb równomiernych

Niech będzie dany generator w formie:

$$x_{n+1} = f(\underbrace{x_n, x_{n-1}, ..., x_{n-k+1}}_{k \text{ stałych początkowych}}) (\text{mod } M) \tag{11.2}$$

Gdzie:

- x_n - kolejne generowane liczby
- f funkcja generująca liczby
- k 0 liczba poprzednich wartości używanych do obliczenia następnej, określa tzw. „rzędowość” generatora
- M - moduł / zakres możliwych wartości

Dla takiego generatora zakładamy zbiór możliwych wartości Z_M dziedzinę D i przeciwdziedzinę D^{-1}. :

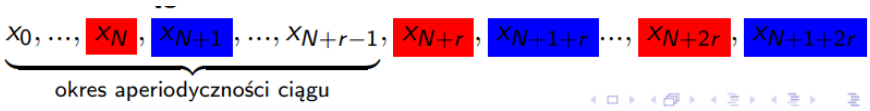
$$Z_M = \{0, 1, ..., M-1\} \tag{11.3}$$

$$D(f) = Z_M^{\otimes k} \qquad D^{-1}(f) = Z_M \tag{11.4}$$

gdzie $Z_M^{\otimes k}$ to zbiór wszystkich k-elementowych ciągów z Z_M. Funkcja f przyjmuje więc k poprzednio wygenerowanych liczb i zwraca nową z przedziału $[0, M-1]$.

Tego typu generatory są okresowymi dla okresu ciągu r:

$$\exists_{N,r} \forall_{n \geq N} x_n = x_{n+jr} \quad j = 1, 2, ... \tag{11.5}$$



Przykłady:

- **Generator Fibonacciego**

$$x_{n+1} = (x_n + x_{n-1}) \bmod m \tag{11.6}$$

  - okres $\leq M^2$
  - prosty
  - wada: korelacje w ciągach generowanych liczb

- **Generatory liniowe kongruentne**

$$l_{j+1} = (al_j + c)(\bmod M) \tag{11.7}$$

  gdzie:

$$\left.\begin{array}{l} a - \text{multiplier} \\ c - \text{incrementer} \\ m - \text{modulus} \end{array}\right\} \text{ liczby całkowite} \in [0, m] \tag{11.8}$$

  Aby uzyskać liczbę zmiennoprzecinkową $\in [0, 1)$ dzielimy przez m:

$$\frac{l_{j+1}}{m} \in [0, 1) \tag{11.9}$$

  Okres tego generatora jest $\leq m$: w sekwencji $l_1, l_2, l_3, ...$ dla $0 \leq l_i \leq m-1$, w końcua jakaś liczb musi się powtórzyć, a wtedy cały ciąg będzie się powtarzać.

  okres $\leq m$ zależy od wyboru a i c:
  - $c \neq 0 \rightarrow$ generatory mieszane,
  - $c = 0 \rightarrow$ generatory multiplikatywne

11.2. Omów wady i zalety generatorów liniowych kongruentnych

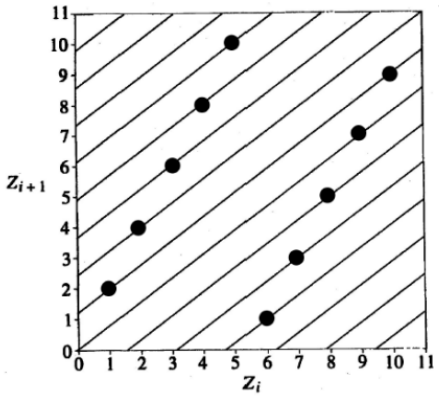
Zalety:
- mało obliczeń
- proste w konstrukcji
- szybkie

Wady:
- korelacje sekwencji:
  - k liczb losowych $\rightarrow$ punkt w przestrzeni k-wymiarowej
  - punkty nie zapełniają równomiernie przestrzeni lecz układają się na $(k-1)$-wymiarowych hiperpłaszczyznach.

    Dla generatora:

$$l_{i+1} = (2 \cdot l_i) \bmod 11, \quad l_0 = 1(\text{ziarno}) \tag{11.10}$$

  - $a = 2, m = 11, c = 0$
  - generowane liczby $\in \{0, 1, ..., 10\}$
  - możemy potraktować pary kolejnych liczb (l_i, l_{i+1}) jako punkty w przestrzeni 2D, Nie wypełniają one równomiernie kwadratu $[0; 11]^2$



- <u>niższe bity są mniej losowe niż wyższe</u>:
  - niższe bity wykazują silniejsze korelacje niż bity wyższe
  - nie należy dzielić liczby na części i używać tylko końcowych cyfr (brania *kawałków*):

✅
```
J = 1 + INT(10.0 * RANF(iseed))
```
❌
```
J = 1 + MOD(INT(100000.0 * RANF(iseed)), 10)
```

11.3. Omów wybrany sposób ulepszania jakości generatorów liczb pseudolosowych

- **dobór odpowiednich parametrów generatora**

  W literaturze pojawiają się następujące wnioski dotyczące parametrów generatorów kongruentnych (czyli na mocy magii):

  - l_0 - nie ma większego znaczenia 😌
  - a:
    - $a (\bmod 8) = 5$
    - $\frac{m}{100} < a < m - \sqrt{m}$
    - brak powtarzającego się wzorca w zapisie dwójkowym
  - c:
    - nieparzyste
    - spełniające $\frac{c}{m} \approx \frac{1}{2} - \frac{\sqrt{3}}{6}$
  - m:
    - $m = 2^t$, gdzie t to liczba bitów przeznaczonych na 1 liczbę całkowitą

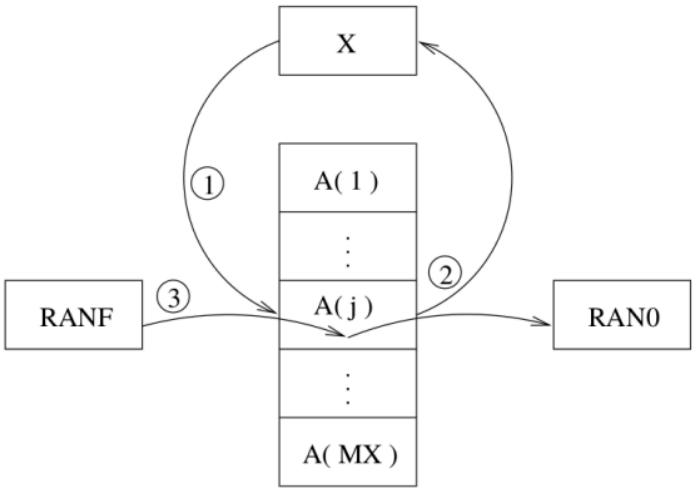
- **procedura „losowego tasowania”**

  `RANF` - generator systemowy,
  `RAND` - generator ulepszony
  `A` - tablica pomocnicza o dlugosci wyznaczonej przez liczbę pierwszą)
  Uogólnienie:
  - kilka generatorów

- jeden z nich wybiera "dostarczyciela" liczb

Kroki:

- wypełniam tablicę A i zmienną x liczbami losowymi
- x traktuje jak indeks j, który wskazuje na element tablicy A. Używam go do wybierania liczb losowych z tablicy.
- $A(j)$ wstawiam w miejsce x oraz jednocześnie zwracam jako liczbę losową ulepszonego generatora (`x=A[j]; return A[j]`)
- z generatora systemowego `RANF` losujemy brakującą liczbę w miejsce $A(j)$



Rysunek 14.1: Idea ulepszonego gen. liczb losowych

11.4. Omów metodę odwróconej dystrybuanty: do czego służy, jak ją stosować, wady, zalety

Dystrybuanta jednoznacznie definiuje rozkład prawdopodobieństwa, jest funkcją niemalejącą i określa $P(X \leq x)$:

$$F(x) = \int_{(-\infty)}^{x} p(y)dy \tag{11.11}$$

Dla rozkładu równomiernego na $(0, 1)$:

$$\begin{aligned} F(x) = x \quad &\Rightarrow \quad P(X \leq x) = x \\ p(x) = 1 \quad &x \in (0, 1) \end{aligned} \tag{11.12}$$

Metoda odwróconej dystrybuanty Jeśli zdefiniujemy U - zmienną losową o rozkładzie równomiernym na $(0, 1)$ to zmienna losowa:

$$X = F^{-1}(U) \tag{11.13}$$

ma rozkład o dystrybuancie $F(x)$.

Na przykładzie rozkładu wykładniczego:

$$p(x) = e^{-x} \quad x \in [0, \infty) \tag{11.14}$$

Dystrybuanta:

$$F(x) = \int_{-\infty}^{x} e^{-x'} dx' = 1 - e^{-x} \tag{11.15}$$

$$y = 1 - e^{-x} \tag{11.16}$$

$$x = -\ln(1 - y) \tag{11.17}$$

$$F^{-1}(y) = -\ln(1 - y) \tag{11.18}$$

Generujemy ciąg liczb losowych o rozkładzie równomiernym U:

$$u_1, u_2, u_3, ..., u_n \in (0, 1) \tag{11.19}$$

Wtedy ciąg liczb:

$$y_i = -\ln(1 - u_i) \tag{11.20}$$

ma rozkład wykładniczy.

Zalety:

- dokładne
- w niektórych przypadkach bardzo proste
- potrzeba tylko jednej liczby z U do generowania

Wady:

- w ogólności odwracanie jest trudne, wymaga kosztownych obliczeń
- w wielu przypadkach odwrócenie jest niemożliwe albo daje funkcję nieelementarną

11.5. Omów metodę Boxa-Mullera: do czego służy, jak ją stosować i dlaczego

Metoda Boxa-Mullera służy do generowania niezależnych zmiennych losowych o standardowym rozkładzie normalnym $N(0, 1)$ na podstawie zmiennych o rozkładzie jednostajnym U.

Metodę tę stosuję się, gdyż dla rozkładu normalnego:

$$p(x) = e^{-\frac{x^2}{2}} \tag{11.21}$$

dystrybuanta wynosi $\text{erf}(x)$ i jest funkcją nieelementarną (użycie metody odwrotności dystrybuanty jest kosztowne).

Kroki wyprowadzenia Boxa-Mullera:

- obliczam prawdopodobieństwo łączne dwóch niezależnych rozkładów normalnych:

$$\begin{gathered} p(x_1, x_2) = e^{-\frac{x_1^2}{2}} \cdot e^{-\frac{x_2^2}{2}} = e^{-\frac{x_1^2 + x_2^2}{2}} \\ x_1, x_2 \in (-\infty, \infty) \end{gathered} \tag{11.22}$$

- wprowadzam zmienne biegunowe:

$$r^2 = x_1^2 + x_2^2 \quad r \in [0, \infty) \tag{11.23}$$

$$\begin{cases} x_1 = r \sin(\varphi) \\ x_2 = r \cos(\varphi) \end{cases} \quad \varphi \in [0, 2\pi] \tag{11.24}$$

- przeliczam element prawdopodobieństwa (tj. prawdopodobieństwo, że x i y znajdą się w małym obszarze $dxdy$) we współrzędnych biegunowych (uwzględniając Jakobian $|J| = r$):

$$p(x, y,)dxdy = p(r, \varphi) \cdot r \; drd\varphi = e^{-\frac{r^2}{2}} \cdot r \; drd\varphi \tag{11.25}$$

- wprowadzam $z = \frac{r^2}{2}$;

$$dz = rdr \qquad e^{-z}d\varphi dz \tag{11.26}$$

- dla otrzymanego rozkładu stosuję odwrotną dystrybuantę:

$$F^{-1}(w) = -\ln(1-w) \tag{11.27}$$

  oraz odwrotną funckję do $\frac{r^2}{2} = z$ czyli:

$$r = \sqrt{2z} \tag{11.28}$$

Dodatkowo gęstość prawdopodobieństwa rozkładu $e^{\frac{r^2}{2}}$ nie zależy od $\varphi \in [0, 2\pi]$, który losujemy zgodnie z rozkładem równomiernym na $(0, 2\pi)$

Kroki algorytmu, wynikającego z wyprowadzenia:

- losuje dwie niezależne zmienne losowe U_1, U_2 z rozkładu $U(0,1)$
- obliczam:

$$x_1 = r\cos(\varphi) = \sqrt{-2\ln U_1} \cdot \cos(2\pi U_2) \tag{11.29}$$

$$x_2 = r\sin(\varphi) = \sqrt{-2\ln U_1} \cdot \sin(2\pi U_2) \tag{11.30}$$

- x_1 i x_2 są niezależnymi zmiennymi losowymi o standardowym rozkładzie normalnym $N(0,1)$

Zalety:

- generujemy od razu dwie liczby
- generowanie φ jest proste, bo bierzemy z U

Wady:

- potrzeba dwóch liczb do generowania
- nadal kosztowne obliczeniowo

11.6. Opisz dlaczego możemy wyznaczać całki metodami Monte Carlo

Niech $y = g(x)$ to pewna zmienna losowa, której wartości losujemy zgodnie z rozkładem ciągłym $p(x)$, $x \in (a, b)$.

Wartość oczekiwana y:

$$\mathbb{E}\{Y\} = \int_a^b g(x)p(x)dx \quad (*) \tag{11.31}$$

Tę właściwość możemy wykorzystać do policzenia całki:

$$I = \int_a^b f(x)dx \tag{11.32}$$

Niech:

- $p(x) > 0$ dla $x \in (a, b)$
- $\int_a^b p(x)dx = 1$
- $p(x)$ spełnia warunki, aby być gęstością rozkładu pewnej zmiennej losowej przyjmującej wartości z przedziału (a, b)

Wtedy:

$$I = \int_a^b \frac{f(x)}{p(x)} \cdot p(x)dx = \int_a^b g(x)p(x)dx \tag{11.33}$$
$$\text{całka postaci } (*)$$

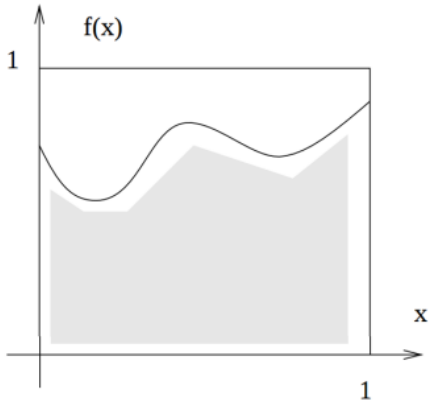
Obliczanie całki **można zawsze przedstawić jako zagadnienie obliczania wartości oczekiwanej** pewnej zmiennej losowej ciągłej.

11.7. Opisz całkowanie Monte Carlo metodami: orzeł-reszka, podstawowa, średniej ważonej

- **całkowanie metodą „orzeł-reszka”**

  Szukamy:

$$I = \int_0^1 f(x)dx \quad 0 \leq f(x) \leq 1 \tag{11.34}$$



  Niech (X, Y) będzie dwuwymiarową zmiennaą losową o rozkładzie równomiernym w kwadracie $[0, 1]^2$.

  Prawdopodobieństwo, że (X, Y) przyjmie wartość z zakreskowanej części rysunku jest równe tej powierzchni, czyli wartości całki I.

  Metoda

  Niech:
  - N - liczba wszystkich eksperymentów (obserwacji) (X, Y)
  - M - liczba eksperymentów, w których (X, Y) poniżej $f(x)$

  Jeżeli przeprowadzone obserwacje są niezależne, to M ma rozkład dwumianowy:

$$P\{M = m\} = \binom{N}{m} I^m (1 - I)^{N-m} \tag{11.35}$$

  W rozkładzie dwumianowym możemy oszacować:
  - średnią, która posłuży do obliczenia wartości całki:

$$\hat{I} = \frac{M}{N} \tag{11.36}$$

  - wariancję, która posłuży do obliczenia błędu całkowania:

$$S^2\left(\hat{I}\right) = \frac{1}{N}\frac{M}{N}\left(1 - \frac{M}{N}\right) \tag{11.37}$$

Właściwości metody:

- prosta
- łatwe uogólnienie na n-wymiarowy przypadek
- mało efektywna

- **metoda podstawowa**

Problem znalezienia całki z $f(x)$ przedstawia się jako problem znalezienia wartości oczekiwanej zmiennej losowej o gęstości $p(x)$:

$$\begin{gathered} I = \int_a^b f(x)dx = (b-a)\int_a^b f(x)\frac{1}{b-a}dx = (b-a)\int_a^b f(x)p(x)dx \\ p(x) = \frac{1}{b-a} - \text{gęstość rozkładu równomiernego na } (a,b) \end{gathered} \tag{11.38}$$

Bez utraty ogólności wystarczy rozpatrywać:

$$\begin{gathered} I = \int_0^1 f(x)dx = \mathbb{E}\{Y\} \\ Y = f(X) \\ X - \text{zmienna o rozkładzie równomiernym } (0,1) \end{gathered} \tag{11.39}$$

Oszacowanie $\mathbb{E}\{Y\}$ - średnia z N niezależnych obserwacji:

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{n} f(x_i) \quad (**) \tag{11.40}$$

Metoda:

- wylosować $x_1, x_2, ..., x_N$ według rozkładu równomiernego na $(0,1)$
- obliczyć $f(x_1), f(x_2), ..., f(x_N)$
- obliczyć średnią $(**)$

Estymator wariancji metody:

$$\sigma^2\left(\hat{I}\right) = \frac{1}{N}\sigma_f^2 \tag{11.41}$$

$$\sigma_f^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left[f(x_i) - \hat{I}\right]^2 \tag{11.42}$$

- **metoda średniej ważonej**

Jeżeli $f(x)$ jest stała to w metodzie podstawowej pojedyńczy punkt jest wynikiem dokładnym.

Stąd nasuwa się wniosek, że jeżeli $f(x)$ jest gładkie, to liczba potrzebnych losowań będzie mała.

W metodzie średniej ważonej:

$$I = \int_0^1 \frac{f(x)}{p(x)} \cdot p(x)dx = \int_0^1 \underbrace{g(x)}_{\frac{f(x)}{p(x)}} p(x)dx \tag{11.43}$$

Chcemy wybrać takie $p(x)$, że:

- $p(x)$ jest ciągła, $x \in [0,1]$
- $p(x) > 0 \quad x \in [0,1]$
- $\int_0^1 p(x)dx = 1$
- $\frac{f(x)}{p(x)} \quad x \in (0,1)$ - znacznie gładsza niż $f(x)$
- $p(x)$ - dana jest prostym wzorem analitycznym

Metoda:

- wybieramy $p_1(x) > 0$ dla $x \in (0,1)$
- dobieramy stałą:

$$p(x) = \alpha p_1(x) \quad \alpha \int_0^1 p(x)dx = 1 \tag{11.44}$$

- liczymy analitycznie dystrybuantę

$$P(x) = \int_0^x p(x')dx' \tag{11.45}$$

- losujemy z rozkładem równomiernym:

$$y_1 \in (0,1) \quad i = 1, ..., N \tag{11.46}$$

- stosujemy metodę odwrotnej dystrybuanty:

$$P^{-1}(y_i) = x_i \Rightarrow x_i \quad i = 1, ..., N \tag{11.47}$$

- przybliżamy wartość całki:

$$I \approx \frac{1}{N} \sum_{i=1}^{N} g(x_i) = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{g(x_i)} \tag{11.48}$$

11.8. Porównaj całkowanie numeryczne (Newtona-Cotesa, Gaussa) i całkowanie metodami Monte Carlo

- **Podstawowa idea** - metody Newtona-Cotesa (np. trapezów, Simpsona) polegają na aproksymacji funkcjami wielomianami i całkowaniu tych wielomianów. Metoda Gaussa dodatkowo wykorzystuje optymalnie dobrane węzły i wagi aby osiągnąć wysoką dokładność.

  Metody Monte-Carlo polegają na losowaniu punktów z dziedziny całkowania i uśrednianiu wartości funkcji w tych punktach, a wynik jest estymowany statystycznie.

- **Determinizm** - całkowanie kwadraturami jest w pełni deterministyczne, podczas gdy metody Monte Carlo opierają się na niedeterministycznych liczbach pseudolosowych.

- **Zależność od dostępnych generatorów liczb pseudolosowych** - dla kwadratur nie ma znaczenia, czy generatory są dostępne i jakiej są jakości, podczas gdy jakość metod Monte Carlo opiera się na dostępności wysokiej jakości generatorów.

- **Złożoność obliczeniowa** - - szczególnie dla całek wielowymiarowych o trudnych granicach lub bardzo dużych obszarów całkowania metody Monte Carlo są znacznie mniej wymagające obliczeniowo.
- **Precyzja obliczeń** - dla małej liczby wymiarów (najlepiej 1) i funkcji dającej się precyzyjnie przybliżyć wielomianem kwadratury są precyzyjniejsze, w szczególności kwadratura Gaussa ma stopień dokładności $2n - 1$ dla n punktów