131890-S Metody Optymalizacji **Letni 2023/2024**

Zajęcia 3: Aproksymacje numeryczne

Daniel Kaszyński

3.1 Różnice skończone

Różnice skończone to wyrażenia matematyczne w postaci $f(x+b) - f(x+a)$. Jeśli podzielimy różnicę skończoną przez $b-a$, otrzymamy iloraz różnicowy. Te przybliżenia pochodnych są często stosowane w metodach różnic skończonych do numerycznego rozwiązywania równań różniczkowych. Ilorazy różnicowe omawiane były na wykładzie na temat pochodnych. Ta sekcja posłuży jako podsumowanie tych skończonych różnic.

3.1.1 Różnice skończone "wstecz" i "w przód"

Najpopularniejszymi metodami aproksymacji pochodnych są różnice skończone "wstecz" i "w przód". To właśnie z nich najczęściej korzystaliśmy na poprzednich wykładach.

Definicja 1: Pochodna funkcji

Pochodną funkcji $f : \mathbb{R} \to \mathbb{R}$ opisaną przez **różnicę skończoną "w przód"** nazywamy:

$$f'(x) = \frac{df}{dx}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.1}$$

Pochodną funkcji $f : \mathbb{R} \to \mathbb{R}$ opisaną przez **różnicę skończoną "wstecz"** nazywamy:

$$f'(x) = \frac{df}{dx}(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h} \tag{3.2}$$

Algorytmy różniczkowania numerycznego szacujące pochodną funkcji matematycznej przy użyciu różnicy skończonej "w przód" lub różnicy skończonej "wstecz" obarczone są błędem $O(h)$ (można to udowodnić przy pomocy twierdzenia Taylora).

Z twierdzenia Taylora wiemy, że:

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + ... \tag{3.3}$$

Wyrażenie to możemy przekształcić na:

$$f'(x)h = f(x+h) - f(x) - \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 - ... \tag{3.4}$$

i dzieląc to wyrażenie przez h otrzymujemy:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{2}f''(x)h - \frac{1}{6}f'''(x)h^2 - ... \tag{3.5}$$

Możemy więc wywnioskować, że błąd różnicy "w przód" jest rzędu $O(h)$.

Stosując tę samą metodologię, możemy przekształcić poniższy wzór:

$$f(x-h) = f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 + ... \tag{3.6}$$

we wzór na różnicę "wstecz":

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \frac{1}{2}f''(x)h - \frac{1}{6}f'''(x)h^2 - ... \tag{3.7}$$

Różnica wsteczna ma również błąd rzędu $O(h)$.

3.1.2 Centralna różnica skończona

Oprócz tych dwóch sposobów obliczania pochodnej funkcji istnieje jeszcze trzeci - wykorzystujący **różnicę centralną**. Ta metoda jest czasami nazywana pochodną symetryczną.

Definicja 2: Centralna różnica skończona funkcji

Pochodną funkcji $f : \mathbb{R} \to \mathbb{R}$ opisaną przez **centralną różnicę skończoną** nazywamy:

$$f'(x) = \frac{df}{dx}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h} \tag{3.8}$$

Algorytmy różniczkowania numerycznego szacujące pochodną funkcji matematycznej za pomocą różnicy centralnej obarczone są błędem $O(h^2)$. Jest to preferowane, gdyż błąd jest wtedy mniejszy niż w przypadku poprzednich metod.

Podobnie jak w przypadku opisanych wcześniej przybliżeń, z twierdzenia Taylora możemy wyprowadzić wzór na różnicę centralną. Tym razem jednak będziemy potrzebować dwóch równań wyjściowych:

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + ... \tag{3.9}$$

$$f(x-h) = f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 + ... \tag{3.10}$$

Odejmując od siebie oba powyższe wzory, otrzymujemy:

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{1}{3}f'''(x)h^3 + ... \tag{3.11}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{3}f'''(x)h^3 - ... \tag{3.12}$$

3.2 Błąd anulowania subtraktywnego

Obliczając pochodne funkcji, możemy napotkać problemy nie tylko natury matematycznej, ale także techniczne/sprzętowe. Jednym z popularnych problemów tego rodzaju jest **błąd anulowania subtraktywnego**.

Podczas stosowania arytmetyki zmiennoprzecinkowej możemy napotkać błąd anulowania subtraktywnego. Kiedy komputery pracują z liczbami rzeczywistymi, muszą w jakiś sposób przechowywać w tych liczbach

potencjalnie nieskończoną liczbę miejsc po przecinku. W tym celu wykorzystują m.in. zmienne typu float, które stanowią skończone przybliżenie liczb rzeczywistych. Większość języków programowania wykorzystuje standard techniczny arytmetyki zmiennoprzecinkowej zwany **IEEE 754**. Niestety, w ten sposób częściowo traci się precyzję, ale jest to spowodowane ograniczoną pamięcią komputera.

Błąd anulowania subtraktywnego to zjawisko, które może wystąpić podczas odejmowania dwóch prawie równych liczb. Liczby zmiennoprzecinkowe w komputerach mają ograniczoną precyzję i gdy odejmie się dwie liczby o bardzo zbliżonej wartości, w wyniku może wystąpić utrata cyfr znaczących.

Przykład 1. Niech $a = 0.3 + 0.3 + 0.4 - 1$ oraz $b = -1 + 0.3 + 0.3 + 0.4$.

Kierując się prostą matematyką, możemy stwierdzić, że a jest równe b. Niestety obliczenia wykonane na zmiennych zmiennoprzecinkowych mogą nie dać tego samego wyniku.

```r
a <- 0.3+0.3+0.4-1
b <- -1+0.3+0.3+0.4

print(a == b) # FALSE
print(a) # 0
print(b) # 5.551115e-17
```

Listing 1: Przykład błędu anulowania subtraktywnego w języku R

W tym przykładzie obliczenia wykonane w celu uzyskania a i b mogą dać bardzo zbliżone, ale jednak różne wartości. Wyniki mogą nie być na tyle dokładne, na ile można by się spodziewać, ze względu na błąd anulowania subtraktywnego. Dokładność wyniku zależy od liczby cyfr znaczących, które można przedstawić w formacie zmiennoprzecinkowym. Trzeba zatem pamiętać, że przy odejmowaniu zmiennych typu float może wystąpić błąd, który choć niewielki, może zepsuć niektóre proste porównania i obliczenia.

Aby złagodzić błędy anulowania subtraktywnego, można zastosować różne techniki i algorytmy, takie jak zmiana układu wyrażenia, aby uniknąć odejmowania prawie równych liczb lub w razie potrzeby użyć arytmetyki o większej precyzji. Ponadto zrozumienie ograniczeń arytmetyki zmiennoprzecinkowej i świadomość potencjalnych źródeł błędów jest kluczowa podczas pracy z obliczeniami numerycznymi w programach komputerowych.

3.3 Złożona pochodna schodkowa

Aby uniknąć problemu błędów subtraktywnego odejmowania omówionego w poprzedniej sekcji, można zastosować różne typy metodologii i transformacji formuł. Jednym z możliwych rozwiązań jest użycie **Złożonej pochodnej schodkowej** (ang. *Complex Step Derivative*). Główną ideą tej metody jest wykorzystanie równania Taylora i liczb zespolonych w celu wyeliminowania konieczności odejmowania dwóch zmiennych zmiennoprzecinkowych.

Zacznijmy od równania Taylora używając liczb zespolonych:

$$f(x + ih) = f(x) + f'(x)ih - \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)ih^3 + ... \tag{3.13}$$

Zakładając, że chcemy obliczyć pierwszą pochodną funkcji, musimy przekształcić wzór:

$$f'(x)ih = f(x + ih) - f(x) + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)ih^3 + ... \tag{3.14}$$

Następnie musimy wyizolować pochodną. Możemy zacząć od podzielenia obu stron przez h:

$$f'(x)i = \frac{f(x+ih) - f(x)}{h} + \frac{1}{2}f''(x)h + \frac{1}{6}f'''(x)ih^2 + ... \tag{3.15}$$

Aby otrzymać tylko pochodną, musimy zadbać także o część urojoną:

$$f'(x) = Im\left(\frac{f(x+ih) - f(x)}{h}\right) + \frac{1}{6}f'''(x)h^2 + ... \tag{3.16}$$

Na szczęście jesteśmy w stanie usunąć $\frac{1}{2}f''(x)h$, ponieważ nie zawiera on części urojonej. Uprości to naszą formułę, ale nadal nie jest tak dobra, jak byśmy sobie tego życzyli. Nie pozbyliśmy się jeszcze odejmowania dwóch instancji funkcji. W tym celu powinniśmy rozdzielić pierwszą część naszego wzoru:

$$f'(x) = Im\left(\frac{f(x+ih)}{h}\right) - Im\left(\frac{f(x)}{h}\right) + \frac{1}{6}f'''(x)h^2 + ... \tag{3.17}$$

Możemy zauważyć, że $Im\left(\frac{f(x)}{h}\right)$ nie ma części urojonej. Możemy więc usunąć go z naszego równania. W ten sposób otrzymujemy wzór, który pomaga uniknąć problemu błędu anulowania subtraktywnego:

$$f'(x) = Im\left(\frac{f(x+ih)}{h}\right) + \frac{1}{6}f'''(x)h^2 + ... \tag{3.18}$$

Ten wzór jest esencją złożonej pochodnej schodkowej. Jest to czwarty już wspomniany sposób obliczania pochodnej funkcji.

3.4 Porównanie różnic skończonych

Aby lepiej zrozumieć różnice pomiędzy opisanymi różnicami skończonymi, warto przeprowadzić szereg testów i porównań.

Niech $f = sin(x^2)$. Korzystając z zasad różniczkowania symbolicznego, możemy wnioskować, że $f'(x) = 2xcos(x^2)$.

Niech $u = x^2$. Wtedy, $\frac{du}{dx} = 2x$ oraz $\frac{df}{du} = cos(u) = cos(x^2)$. Jeśli połączymy te informacje, otrzymamy następujące równanie:

$$f'(x) = \frac{df}{dx} = \frac{du}{dx}\frac{df}{du} = 2xcos(x^2) \tag{3.19}$$

Dla pewności możemy użyć języka programowania **R** do obliczenia pochodnej funkcji f. W tym celu musimy najpierw zaimportować bibliotekę *Deriv*, która służy do obliczania pochodnych:

```
1 if(!require(Deriv)) install.packages('Deriv');
```

Listing 2: Importowanie biblioteki *Deriv*

Korzystając z bibliotek zewnętrznych, warto jest sprawdzać ich dokumentację. W języku programowania **R** można to zrobić dodając znak '?' przed nazwą biblioteki:

```
1 # Dostep do dokumentacji biblioteki
2 ?Deriv
```

Listing 3: Dostep do dokumentacji biblioteki *Deriv*

Następnie korzystając z tej biblioteki możemy obliczyć pochodną funkcji f:

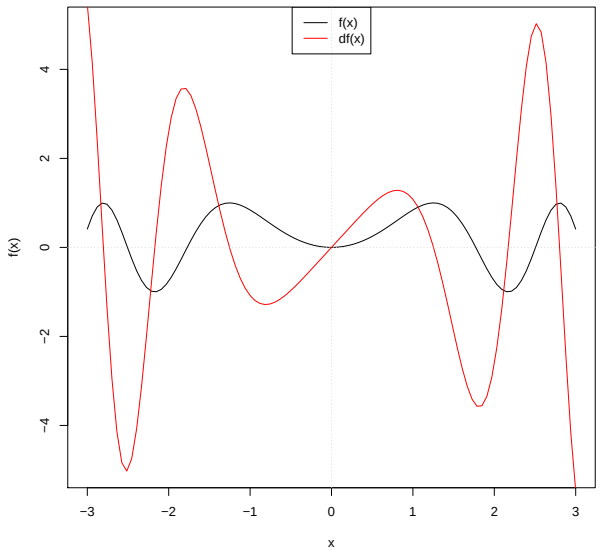
```
f <- function(x) sin(x^2);
df <- Deriv(f)

cat('f = ', deparse(f)[2], '\n')
cat('df = ', deparse(df)[2], '\n')
```

Listing 4: Pochodna funkcji f obliczona przy pomocy biblioteki *Deriv*

W wyniku tych obliczeń otrzymujemy: $f = sin(x^2)$ and $df = 2xcos(x^2)$.

Wykresy funkcji f i jej pochodnej df wyglądają następująco:



Rysunek 3.1: Wykresy funkcji f i jej pochodnej df

Oczywiście, aby porównać różne rodzaje aproksymacji pochodnych funkcji w języku programowania **R**, najpierw musimy posiadać implementacje tych metod. Znając definicje i wzory metod, stworzenie ich implementacji nie jest trudnym zadaniem. Przykładowe definicje mogą w kodzie wyglądać następująco:

```
diff_forward <- function (f, x, h = 10^-6) (f(x + h) - f(x)) / (h);
diff_backward <- function (f, x, h = 10^-6) (f(x) - f(x - h)) / (h);
diff_central <- function (f, x, h = 10^-6) (f(x + h) - f(x - h)) / (2*h);
diff_complex <- function (f, x, h = 10^-6 ) Im(f(x + h*1i )) / (h);
```

Listing 5: Implementacje różnic skończonych

Następnie możemy sprawdzić różnice w wartościach otrzymanych w wyniku tych przybliżeń w podanym punkcie $x_0 = 1$. Poniższy fragment kodu zapewnia nam wartości dla każdej skończonej różnicy:

```
x0 <- 1

cat('df = ' , format( df(x0), nsmall = 20 ), " \n ")
cat('----------------------------------------' , " \n ")
cat('diff_forward = ' , format( diff_forward(f, x0), nsmall = 20), " \n ")
cat('diff_backward = ' , format( diff_backward(f, x0), nsmall = 20), " \n ")
```

```
cat('diff_central = ' , format( diff_central(f, x0), nsmall = 20), " \n ")
cat('diff_complex = ' , format( diff_complex(f, x0), nsmall = 20), " \n ")
```

Listing 6: Wartości uzyskane przez różnice skończone dla punktu $x_0 = 1$

df = 1.08060461173627953002

---

diff_forward = 1.08060346903915416306

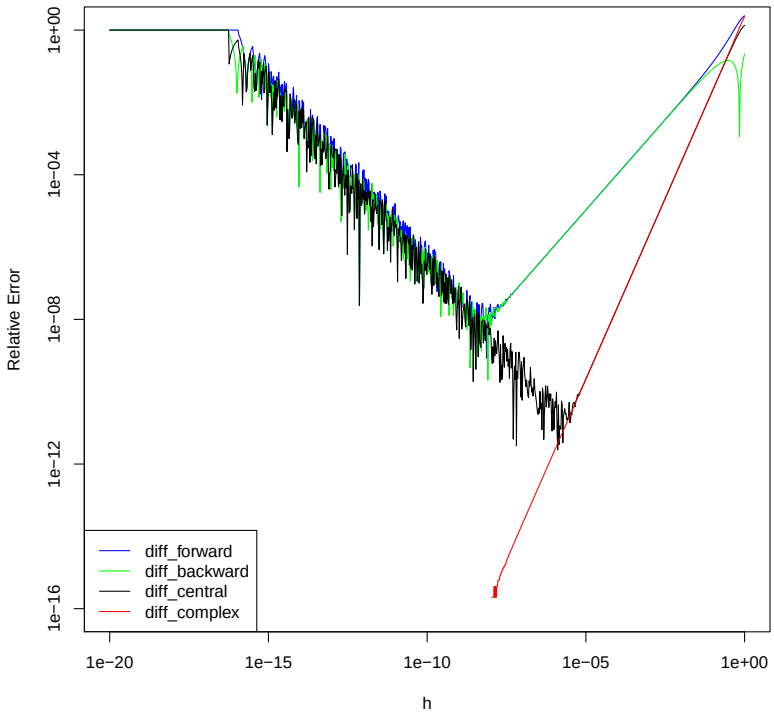
diff_backward = 1.08060575443325035394

diff_central = 1.08060461179171340973

diff_complex = 1.08060461173868294082

Jak widać wartości te są do siebie zbliżone i nie odbiegają zbytnio od rzeczywistej wartości pochodnej funkcji. Metoda diff_complex osiągnęła najlepszy wynik, ponieważ jest najbliższa wartości rzeczywistej.

Oprócz sprawdzenia poszczególnych wartości funkcji dla każdej ze skończonych różnic, możemy również wykonać wiele innych porównań. Przykładowo możemy wykreślić jak szybko błąd względny zbiega się do 0 w odniesieniu do wszystkich różnic numerycznych. Wykres stworzony w tym celu, powinien mieć skalę wykładniczą, ponieważ wszystko co nas interesuje będzie miało miejsce w bardzo wąskich przedziałach. Aby poprawić widoczność, zaleca się stosowanie skali logarytmicznej.



Rysunek 3.2: Błąd względny dla każdej ze skończonych różnic na f

Jak widać na powyższym wykresie, charakterystyka zmian błędu względnego w zależności od h jest różna dla każdej metody aproksymacji. Śledząc wartości względem osi h od prawej do lewej, możemy zobaczyć, jak błąd względny zbiega się do zera (lub w większości przypadków przynajmniej próbuje się zbiegać). Dla każdej skończonej różnicy możemy zdefiniować następujące zachowanie:

- **diff_forward** oraz **diff_backward** - Różnica skończona "w przód" i różnica skończona "wstecz" zachowywały się bardzo podobnie przy próbie zbiegania do zera. Obie metody osiągnęły minimalny błąd przy h wynoszącym około $1e-08$. Od tego momentu błąd zaczął jednak rosnąć. Było to spowodowane opisanym wcześniej błędem anulowania subtraktywnego (patrz sekcja 3.2). Błąd spowodowany odjęciem dwóch zmiennych zmiennoprzecinkowych dla mniejszych wartości h znacząco wpływał na jakość wyników w tych przypadkach.
- **diff_central** - Centralna różnica zachowywała się podobnie do dwóch wcześniej opisanych metod, z kilkoma zauważalnymi różnicami. Po pierwsze, na początku metoda ta osiągała minimum znacznie szybciej w okolicach $1e-05$. Nachylenie zbiegania błędu względnego dla tego przybliżenia było znacznie bardziej strome. Było to spowodowane faktem, że różnica środkowa obarczona jest błędem $O(h^2)$. Co więcej, wartości błędów względnych pozostawały średnio nieco poniżej wartości uzyskiwanych w przypadku różnic skończonych "w przód" i "wstecz"ł.
- **diff_complex** - Ze wszystkich opisanych metodologii najlepiej wypadła aproksymacja złożonej pochodnej schodkowej. Nie tylko zbiega do zera w tempie porównywalnym z różnicą centralną, ale także nie był dotknięty problemem błędu anulowania subtraktywnego. Pozwoliło to na osiągnięcie w tej metodzie wartości błędów względnych bliskich zeru w takim stopniu, że język **R** nie rozróżniał ich od zera (dlatego zniknęły z wykresu).

3.5 Automatyczne różniczkowanien

Automatyczne różniczkowanie (ang. *automatic differentiation*), znany również jako różniczkowanie algorytmiczne lub autoróżnicowanie, to technika stosowana do wydajnego i dokładnego oceniania pochodnych funkcji matematycznych. Podstawowym celem tej techniki jest automatyczne i systematyczne obliczanie pochodnych danej funkcji, co czyni ją szczególnie przydatną w optymalizacji, uczeniu maszynowym i obliczeniach naukowych.

Automatic differentiation różni się od różniczkowania symbolicznego i numerycznego. Różniczkowanie symboliczne napotyka wyzwania podczas przekształcania programu komputerowego w ujednolicone wyrażenie matematyczne, co często skutkuje nieefektywnym kodem. Z drugiej strony różniczkowanie numeryczne, wykorzystujące metodę różnic skończonych, która może wprowadzać błędy w procesie dyskretyzacji i powodować problemy związane z anulowaniem. Te tradycyjne metody sprawiają także problemy przy obliczaniu pochodnych wyższego stopnia. Natomiast automatyczne różnicowanie skutecznie rozwiązuje wszystkie te problemy.

Aby w pełni zrozumieć, jak działa automatyczne różniczkowanie, musimy najpierw zapoznać się z kilkoma podstawowymi koncepcjami, które za nim stoją. Po pierwsze, rozkład różniczek zapewniony przez **regułę łańcuchową** pochodnych cząstkowych ma fundamentalne znaczenie dla automatycznego różniczkowania.

Reguła łańcuchowa to podstawowe pojęcie w rachunku różniczkowym, które opisuje jak znaleźć pochodną funkcji złożonej. Jeśli mamy złożenie dwóch funkcji $f(x)$ i $g(x)$ takie, że $f(g(x))$, to reguła łańcuchowa stwierdza, iż pochodna tego złożenia po x jest iloczynem pochodnej f względem jej argumentu $g(x)$ i pochodnej g względem x:

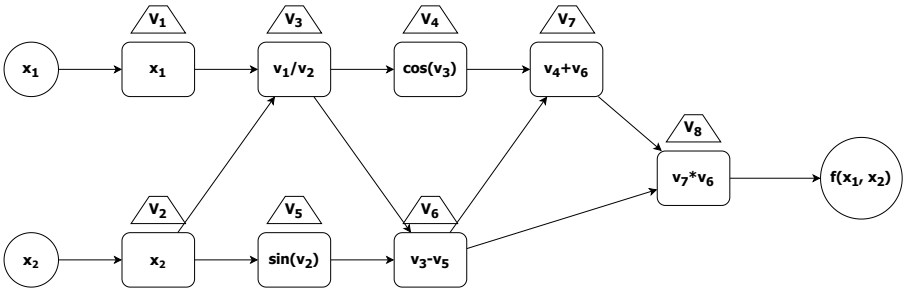
$$\frac{df}{dx}f(g(x)) = \frac{d}{dx}(f \circ g)(x) = \frac{df}{dg}\frac{dg}{dx} = f'(g(x))g'(x) \tag{3.20}$$

Kolejną kwestią, na którą warto zwrócić uwagę jest fakt, że automatyczne różnicowanie wykorzystuje grafy

obliczeniowe (jawnie lub nie). Graf obliczeniowy jest reprezentacją wyrażenia matematycznego lub procesu obliczeniowego. Powszechnie używana się go do wizualizacji i zrozumienia przepływu obliczeń związanych z oceną funkcji lub wykonaniem serii operacji.

Większość wzorów matematycznych można podzielić na serię podstawowych operacji arytmetycznych (np. dodawanie, mnożenie, potęgowanie). Aby utworzyć graf obliczeniowy, najpierw tworzymy wierzchołki. Wierzchołki grafu obliczeniowego reprezentują operacje lub funkcje matematyczne. Każdy wierzchołek posiada przypisane do siebie działanie, takie jak dodawanie, mnożenie lub bardziej złożona operacja. Krawędzie w grafie przedstawiają przepływ danych lub zależności pomiędzy operacjami. Krawędź między wierzchołkami wskazuje, że wynik pierwszej operacji jest używany jako wejście dla drugiej operacji. Dane wejściowe na grafie obliczeniowym są zwykle przedstawiane jako wierzchołki bez krawędzi przychodzących, podczas gdy wyjścia to wierzchołki bez krawędzi wychodzących.

Przykład 2. Niech $f(x_1, x_2) = (cos(\frac{x_1}{x_2}) + \frac{x_1}{x_2} - sin(x_2)) \cdot (\frac{x_1}{x_2} - sin(x_2))$. Graf obliczeniowy takiej funkcji wyglądałby następująco:



Rysunek 3.3: Przykładowy graf obliczeniowy funkcji $f(x_1, x_2)$

Kolejną istotną częścią algorytmu jest użycie *liczb podwójnych* (ang. *dual numbers*). Dzięki nim W każdym wierzchołku grafu obliczeniowego nie tylko obliczymy wartości liczbowe funkcji, ale jednocześnie obliczymy ich pochodne. Ten prosty trik pomoże nam ponownie wykorzystać fragmenty obliczone w poprzednich krokach w przyszłych operacjach. Jednocześnie ograniczamy się do obliczania pochodnych tylko prostych operacji arytmetycznych.

W języku programowania **R** możemy zaimplementować takie *liczby podwójne* za pomocą programowania obiektowego:

```
DualNumber <- function(val, eps=0) {
    obj <- list(val = val, eps = eps)
    class(obj) <- "DualNumber"
    return(obj)
}
```

Listing 7: Implementacja liczb podwójnych

Na koniec pozostaje jeszcze kwestia obliczenia wyników poszczególnych operacji na grafie obliczeniowym. Eleganckim sposobem podejścia do tego problemu jest użycie **przeciążanie operatorów** (ang. *operator overloading*). Każdy operator (taki jak „+" lub „-") możemy go wykorzystać do zdefiniowania innego zachowania, specjalnie dostosowanego do naszych *liczb podwójnych*.

Stwórzmy przeciążenia dla każdej z podstawowych operacji. Na początek możemy zacząć od operatora dodawania - '+':

```
"+" <- function (x, y) {
  if (class(x) == "DualNumber") {
    val <- x$val + y$val
    eps <- x$eps + y$eps
    return(DualNumber(val, eps))
  } else {
    .Primitive("+")(x, y)
  }
}
```

Listing 8: Przeciążenie operatora dodawania

Każdy operator jest funkcją, która przyjmuje dwa parametry jako dane wejściowe i podaje wynik. Najpierw powinno się rozróżnić wpływ operatora na *liczby podwójne* i inne wartości (dla których działanie operatora pozostanie niezmienione). Jeśli chcemy dodać dwie *liczby podwójne*, musimy dodać zarówno ich wartości, jak i wartości ich pochodnych. Następnie możemy zwrócić nowo utworzony wynik jako *dual number*.

Postępując w podobny sposób możemy zaimplementować przeciążanie operatora dla operatora odejmowania - '-':

```
"-" <- function (x, y) {
  if (class(x) == "DualNumber") {
    val <- x$val - y$val
    eps <- x$eps - y$eps
    return(DualNumber(val, eps))
  } else {
    .Primitive("-")(x, y)
  }
}
```

Listing 9: Przeciążenie operatora odejmowania

Przy przeciążaniu operatora mnożenia '*' warto przypomnieć sobie *wzór Leibniza* (wzór służący do wyznaczania pochodnych iloczynu dwóch funkcji):

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \tag{3.21}$$

```
"*" <- function (x, y) {
  if (class(x) == "DualNumber") {
    val <- x$val*y$val
    eps<-y$val*x$eps + x$val*y$eps
    return(DualNumber(val, eps))
  } else {
    .Primitive("*")(x, y)
  }
}
```

Listing 10: Przeciążenie operatora mnożenia

Jako ostatni operator, który opracujemy na potrzeby tej przykładowej implementacji, możemy przeciążyć operatora potęgowania. Pomocny może okazać się wzór na pochodną funkcji potęgowej:

$$f'(x) = \frac{d}{dx}(x^k) = kx^{k-1} \tag{3.22}$$

Należy tylko pamiętać o pomnożeniu wartości uzyskanej z tego wzoru przez obliczoną wcześniej wartość pochodnej:

```
"^" <- function (x, k) {
  if (class(x) == "DualNumber") {
```

```
    val <- x$val^k
    eps <- k*x$val^(k-1)*x$eps
    return(DualNumber(val, eps))
  } else {
    .Primitive("^")(x, k)
  }
}
```

Listing 11: Przeciążenie operatora potęgowania

Do działania algorytmu automatycznego różniczkowania wystarczą *dual numbers* i przeciążone operatory. Teraz możemy przejść do testowania algorytmu.

Przykład 3. Niech $A = 5$ z uwzględnieniem pierwszej zmiennej, $B = 4$ z uwzględnieniem drugiej zmiennej oraz $C = 7$ z uwzględnieniem trzeciej zmiennej. Ponadto, niech $f(x_1, x_2, x_3) = x_1^2 + x_1x_2 + x_2 + x_3$:

```
# Deklaracja wartosci DualNumer
A <- DualNumber(5, c(1,0,0))
B <- DualNumber(4, c(0,1,0))
C <- DualNumber(7, c(0,0,1))

# Obliczanie wartosci funkcji f(A, B, C).
A^2 + A*B + A + C
```

Listing 12: Przykład automatycznego różniczkowania

Gdy wykonamy te obliczenia, otrzymamy następujące wyniki:

$val

[1] 57

$eps

[1] 15 5 1

attr(,"class")

[1] "DualNumber"

Jak widać, wykonując prostą operację na liczbach typu *dual number*, możemy otrzymać zarówno wartość funkcji f, jak i wartość pochodnej po każdej ze zmiennych wejściowych. Co więcej, otrzymane wartości dla *$eps* są dokładnie gradientem funkcji f w danym punkcie.