Algebra Symboliczna

Wykład XIII

Andrzej Odrzywolek

Instytut Fizyki, Zakład Teorii Względności i Astrofizyki

01.23.2008, środa, 13:15

Dane kontaktowe

dr Andrzej Odrzywołek

pokój 447, IV piętro

E-mail: `odrzywolek@th.if.uj.edu.pl`

Wykład: środy 13.15-15.00 s. 128

Ćwiczenia: piątki 10.30-12.00

Konsultacje: środy ∼11-13, czwartki 10-12

WWW: http://ribes.if.uj.edu.pl/alsymb/

Najczęstsze błędy składniowe

- Użycie pojedynczego znaku równości = zamiast == jest szczególnie uciążliwym błędem. Należy pamiętać, że:
  1. Pojedyncze **x=2+a** dokonuje *przypisania* wartości **2+a** zmiennej o nazwie **x**
  2. Podwójne **x==2+a** definiuje równanie w sensie matematycznym (logicznym)

  Bardzo często omyłkowo dokonujemy przypisania typu **y[0]=1**, co powoduje stworzenie zmiennej *o nazwie* **y[0]** i nadania jej wartości **1** (!) a nie utworzenie równości mówiącej, że funkcja **y** przyjmuje wartość **1** w zerze, jak oczekujemy.

Uwaga

Błąd usuwamy instrukcją **Clear[]** która usuwa przypisanie, np: aby usunąć wcześniejsze przypisanie **x=3** wywołujemy **Clear[x]**.
W ostateczności należy zrestartować kernel.

Najczęstsze błędy składniowe

- Brak spacji oznaczającej mnożenie
  1. mnożenie liczb $a \times b$ wpisujemy jako **a*b** lub **a b**
  2. brak spacji **ab** spowoduje utworzenie zmiennej o nazwie **ab** !

  Jest to bardzo niebezpieczny błąd, gdyż często prowadzi do jakiegoś wyniku, na ogół błędnego, bez żadnych ostrzeżeń o błędzie składniowym !

Uwaga

1. Błędu tego typu można uniknąć *zawsze* wpisując znak mnożenia zamiast spacji.
2. Jest on łatwy do wykrycia np. poprzez podstawienie

   In:=ab /. { a → 2, b → 3 }

   `Out = ab`

   In:=a b /. { a → 2, b → 3 }

   `Out = 6`

Najczęstsze błędy składniowe: błędne nazwy funkcji

1. każda funkcja matematyczna posiada w MATHEMATICE swoją nazwę, zwykle (ale nie zawsze) podobną do tradycyjnej – użycie innej nazwy zostanie potraktowane jako wprowadzenie *nowej i niezdefiniowanej funkcji* np: tangens to **Tan[x]**; wprowadzenie **Tg[x]**, lub co gorsza **tan(x)** czy **tg x** w najlepszym przypadku doprowadzi do komunikatów o błędach, w najgorszym do nonsensownego wyniku!

Uwaga

1. w dowolnym momencie możemy sprawdzić definicję funkcji w Help-ie, np: **?Tan**
2. ponieważ tego typu błędy są bardzo trudne do wykrycia (np: **tg x** zostanie potraktowane jako mnożenie 2 liczb **tg*x**) jedynie dobra znajomość składni i praktyka dają gwarancję ich uniknięcia

Uwaga

1. otrzymanie nonsensownego wyniku analitycznego bez komunikatów o błędach często wskazuje na tego typu błąd (intiucja matematyczna)
2. próba obliczenia numerycznej wartości wyrażenia zawierającego tego typu błędy może posłużyć za test np:

In:= tg x /. { x → *Pi*/4 }

Out = $\frac{\pi\, tg}{4}$

In:= Tg[x] /. { x → *Pi*/4 }

Out = $\mathrm{Tg}\left[\frac{\pi}{4}\right]$

In:=Tan[x] /. { x → *Pi*/4 }

Out = 1

. . . o ile wiemy, że $\mathrm{tg}\,\pi/4 = 1$ . . .

Zarezerwowane nazwy zmiennych

- Należy uważać na nazwy które są zarezerwowane przez MATHEMATICE:
  1. jednostka urojona: **I**
  2. instrukcja obliczająca wartość numeryczną: **N**
  3. liczba π: $\boldsymbol{\pi}$

Uwaga

1. Najlepszym sposobem na uniknięcie konfliktu oznaczeń jest stosowanie wyłącznie nazw rozpoczynających się od *małej litery* – MATHEMATICA zawiera definicje rozpoczynające się zawsze od *dużej litery*.

Ważne i często używane konstrukcje składniowe

Podstawienie **ReplaceAll** (zamień wszystkie)

- Najczęściej używana jest skrócona wersja **/.** (ukośnik-kropka)
- Pozwala dokonać niemal dowolnych podstawień w wyrażeniu
- Lista możliwych do zamienienia składników wyrażenia **w** może być sprawdzona poleceniem **FullForm[w]**, które wyświetla wszystkie składniki wyrażenia

Przykłady

Niech **w= Sin[x] Cos[x] + Sin[y] + 1** . Następujące postawienia są możliwe:

- Zamieniamy **Sin[x]** na **Sinh[x]**:
  w /. Sin[x] → Sinh[x]
- Zamieniamy *wszystkie* sinusy (**Sin**) na sinusy hiperboliczne (**Sinh**):
  w /. Sin → Sinh

Przykłady

- pełna lista składników wyrażenia:
  In:=FullForm[w]
  `Out=Plus[1,Times[Cos[x],Sin[x]],Sin[y]]`
- Zamieniamy mnożenie (**Times**) na dodawanie (**Plus**):
  w /. Times → Plus
- i dodawanie na mnożenie: **w /. Plus → Times**

Uwagi

1. należy zapamiętać następującą konstrukcję składniową:
   In:= x /. {x → wyrazenie }
   `Out = wyrazenie`
   która pozwala na *wydobycie* interesującej nas wartości liczbowej (lub wyrażenia analitycznego) z wyrażeń typu **{ x → 2 }** . W takiej postaci otrzymujemy m. in. rozwiązania równań algebraicznych i różniczkowych generowanych przez **Solve, FindRoot, DSolve, NDSolve**.

Poprawna metodologia rozwiązywania zadań

Właściwa kolejność rozwiązywania zadań

1. o ile jest to możliwe i sensowne staramy znaleźć graficzne rozwiązanie zadania, czyli odczytać przybliżone wartości w wykresu lub/i animacji. Dotyczy to m. in:
   - rozwiązywania równań i układów równań
   - szukania wartości maksymalnych
   - dopasowania funkcji
   - szukania wzorów przybliżonych
   - obliczania granic
2. szukamy rozwiązania numerycznego/ wartości numerycznej (**FindRoot, NIntegrate, NDSolve**, dla macierzy f. numeryczne wywoływane są jeżeli występują w nich liczby zmiennoprzecinkowe, np. **Det[N[A]]**) – jeżeli w problemie występują nieokreślone z góry parametry nadajemy im jakiekolwiek wartości
3. szukamy symbolicznego rozwiązania zadania

Poprawna metodologia rozwiązywania zadań: 1

Rozwiązanie graficzne

Narysowanie badanej funkcji w celu sprawdzenia czy np. posiada maksimum lub przecina oś X daje nam przynajmniej 3 cenne informacje:

1. ogólne pojęcie o stopniu trudności problemu, niejednokrotnie wykres funkcji zapisanej przy pomocy niezwykle skomplikowanej formuły okazuje się być niemal linią prostą, lub wręcz przeciwnie
2. istnienie oraz ilość potencjalnych rozwiązań
3. przybliżone rozwiązanie zadania, często o wystarczającej dokładności – można je użyć jako punkt startowy dla metod numerycznych np. **FindRoot**

Poprawna metodologia rozwiązywania zadań: 2

Rozwiazanie numeryczne

- Większość typowych zadań można rozwiązać numerycznie, na ogół łatwiej i szybciej niż symbolicznie.
- W przypadku pojedynczych problemów bez symbolicznych parametrów rzecz sprowadza się do wywołania odpowiedniej funkcji numerycznej: **FindRoot, NIntegrate, NDSolve**, **Det[N[A]]**.
- Aby manipulowanie rozwiązaniami numerycznymi z paramterami było skuteczne, należy sprawnie posługiwać się funkcjami **Table, Flatten, Interpolation**

Bardzo ważne!

Nie jest możliwe (nawet w MATHEMATICE) rozwiązanie numeryczne problemu symbolicznego! Wszystkie parametry, stałe itp. muszą mieć nadane wartości *numeryczne* lub posiadające wartość numeryczną (np. **Pi**)

Poprawna metodologia rozwiązywania zadań: 3

Rozwiązanie symboliczne

1. skuteczność systemu MATHEMATICA (i innych) w znajdowaniu rozwiązań analitycznych łatwo prowadzi do złudzenia, że wszytko można rozwiązać w taki sposób
2. w rzeczywistości to problemy egzaminacyjne, zbiory zadań jak również uświęcone przez czas metody i przykłady są specjalnym podzbiorem wszystkich zagadnień, dobranym tak aby można było znaleźć wynik symboliczny!
3. jeżeli atakujemy istotnie nowy problem, szanse na wynik symboliczny są niewielkie, ale istnieją, dlatego zalecam szukanie rozwiązania tego typu dopiero gdy mamy pewność że rozwiązanie istnieje (metoda graficzna) oraz wiemy jak wygląda w przybliżeniu (metoda numeryczna)

Uwagi

1. nawet jeżeli rozwiązanie symboliczne istnieje, znalezienie go może wymagać niezwykle długiego czasu
2. reguły podane powyżej należy stosować ze zdrowym rozsądkiem: nie należy graficznie i numerycznie rozwiązywać równań kwadratowych !
3. mówiąc inaczej, należy stosować metody symboliczne – jeżeli spodziewamy się lub podejrzewamy że taki wynik może istnieć – gdyż dają one największą dokładność
4. moc obliczeniowa współczesnych komputerów pozwalają na wypróbowanie wielu metod – robi to również sama MATHEMATICA dobierając algorytmy automatycznie

Typowe zadania: rozwiązanie równania/układu równań

- **Solve[rownania, niewiadome]** gdzie **rownania** – lista równań, **niewiadome** – lista niewiadomych rozwiązuje *symbolicznie* równania i układy równań. W wielu przypadkach znajduje tylko część rozwiązań.
- **Reduce[rownania, niewiadome]** rozwiązuje rygorstycznie równania lub układy równań znajdując (o ile to możliwe) wszystkie rozwiązania.
- **FindRoot[rownanie, { x, x0 }]** rozwiązuje numerycznie **rownanie** ze względu na zmienną **x** zaczynając od **x=x0**
- **NSolve[wielomiany, zmienna]** rozwiązuje numerycznie równania i układy równań zawierające wyłącznie operacje arytmetyczne (mnożenie, dodawanie)

Rozwiązanie równania/układu równań: przykłady

$x + e^x = \ln 2$

- graficzne rozwiązanie:
  Plot[{x+Exp[x], Log[2]}, {x,-1,1 }]
  odpowiedź: $x \simeq -0.2$
- rozwiązanie numeryczne:
  FindRoot[x+Exp[x] == Log[2], { x, -0.2 }
  odpowiedź: $x \simeq -0.159458$
- rozwiązanie analityczne (rzeczywiste):
  Solve[x+Exp[x] == Log[2], x] lub
  Reduce[x+Exp[x] == Log[2], x, Reals] odp:
  $x = Log[2] - ProductLog[2]$
- wszystkie rozwiązania zespolone:
  Reduce[x+Exp[x] == Log[2], x]
  odpowiedź: Log[2] - ProductLog[C[1], 2] dla dowolnej całkowitej liczby C[1]

Typowe zadania: całkowanie/różniczkowanie

- obliczenie całki oznaczonej $\int_a^b f\,dx$: **Integrate[f, {x,a,b}]**
- obliczenie całki oznaczonej numerycznie: **NIntegrate[f, {x,a,b}]**
- obliczenie pochodnej wyrażenia F $\frac{\partial F}{\partial x} \equiv \frac{dF}{dx} \equiv F'$: **D[F,x]**

Typowe zadanie: rozwiązanie równania (układu r.) różniczkowego

- **DSolve[rownania i warunki poczatkowe, szukane funkcje, zmienna po ktorej rozniczkujemy]**
- rozwiązanie ogólne r. różniczkowego $y' = F(x, y)$: **DSolve[y'[x]==F[x,y[x]],y[x],x]**
- jak wyżej, z warunkiem początkowym: **DSolve[{ y'[x]==F[x,y[x]], y[0]=-1 },y[x],x]**

Uwagi

1. MATHEMATICA wymaga aby szukana przez **DSolve** funkcja zawsze posiadała argument np: **y[x]** a nie samo **y**
2. zmienna po której różniczkujemy musi być podana jawnie jako trzeci argument

Typowe zadanie: własności macierzy i operacje na macierzach (wektorach)

- macierze i wektory w MATHEMATICE są tożsame z listami np. wektor $\mathbf{v} = a, b, c$ to trójelementowa lista **v = {a,b,c}** a macierz 2×2 to **{ {a11, a12}, {a21, a22 } }**
- aby odwołać się do jednego z elementów macierzy (listy) używamy *podwójnych* nawiasów kwadratowych np: **a[[1,2]]** lub **a[[1]][[2]]**
- macierz o zadanych elementach najłatwiej utworzyć instrukcją **Table** np: **a=Table[i-j, {i,1,10} , {j,1,10 }]** wygeneruje macierz 10×10 której składowe są równe **a[[i,j]]==i-j**

Uwagi

1. najprościej wymusić numeryczne metody dla macierzy sprowadzając wcześniej jej elementy do postaci zmiennoprzecinkowej np: **Det[N[a]], Eigenvalues[N[a]]** itp

Rozwiązanie krok po kroku *vs* rozwiązanie „szybkie”

Zadanie 1

Rozwiąż w przedziale $(0, 1)$ równanie różniczkowe:

$$y' + y = \int_0^t f(p)\, dp$$

z warunkami początkowymi $y(0) = 1$, gdzie $f(p)$ jest rzeczywistym rozwiązaniem równania:

$$x + 17p = 4$$

Próba rozwiązania w jednej linijce jest *skuteczna*

```
DSolve[{y[0] == 1, y'[t] + y[t] == Integrate[x /.
Solve[x + 17*p == 4, x][[1]], {p, 0, t}]}, y[t], t]
```

Rozwiązanie krok po kroku *vs* rozwiązanie „szybkie”

Zadanie 1

Rozwiąż w przedziale $(0, 1)$ równanie różniczkowe:

$$y'' + y^2 = \int_0^t f(p)\, dp$$

z warunkami początkowymi $y'(0) = 1, y(0) = 1$, gdzie $f(p)$ jest rzeczywistym rozwiązaniem równania:

$$0 = -47 - 4p + 48x + px - 12x^2 + x^3$$

Próba rozwiązania w jednej linijce jest *nieskuteczna*

```
DSolve[{y'[0] == 1, y[0] == 1, y''[t] + y[t]^2 ==
 Integrate[ x /. Solve[(x - 4)^3 + p*(x - 4) + 17 == 0, x][[1]],
 {p, 0, t}]}, y[t], t]
```

Rozwiązanie krok po kroku

Problem dzielimy na etapy

1. Rozwiązanie równania :

$$0 = -47 - 4p + 48x + px - 12x^2 + x^3$$

2. obliczenie całki:

$$h(t) = \int_0^t f(p)\,dp$$

3. rozwiązanie r. różniczkowego:

$$y'' + y^2 = h(t)$$

   z zadanymi warunkami początkowymi

Ponieważ każdy z podpunktów potrafimy wykonać, rozwiążemy też całe zadanie.

Rozwiązanie krok po kroku (1)

Rozwiązanie równania

1. zapisujemy wielomian do zmiennej **w**:
   w=-47 - 4 p + 48 x + p x - 12 x^2 + x^3
2. pierwsze z rozwiązań jest rzeczywiste:
   f1=Solve[w==0,x][[1]]
3. alternatywnie, rozwiązujemy równanie czysto numerycznie, zmieniając p co 0.1:
   f2 = Table[{p, x /. FindRoot[w == 0, {x, 0.5}] },
   {p, 0, 1, 0.1}] // Interpolation
4. sprawdzamy czy funkcja **f2[p]** i wyrażenie **f1[p]** dają ten sam wynik:
   Plot[{ f2[p],f1 } ,{ p,0,1 }]

Rozwiązanie krok po kroku (2)

Obliczenie całki

1. próba obliczenia całki analitycznie nie prowadzi do wyniku:
   Integrate[f1, {p, 0, t}]
   wyjaśnia to czemu nie udało się znaleźć wyniku wcześniej
2. obliczamy więc całkę numerycznie:
   h1=Table[{t, NIntegrate[f1, {p, 0, t}, WorkingPrecision → 50]}, {t, 0.0,1, 0.02}] //Interpolation
   okazuje się, że otrzymany wzór analityczny prowadzi do gigantycznych błędów dla $t < 0.2$ i musimy zwiększyć precycję numeryczną do 50 miejsc po przecinku aby dostać wynik. O wiele łatwiej całkujemy wynik czysto numeryczny:
   h2 = Table[{t, NIntegrate[f2[p], {p, 0, t}]}, {t, 0.0, 1, 0.02}] // Interpolation

Rozwiązanie krok po kroku (3)

Rozwiązanie równania różniczkowego

1. rozwiązujemy numerycznie r. różniczkowe:
   rozw=NDSolve[{y'[0] == 1, y[0] == 1, y''[t] + y[t]^2 == h1[t]}, y[t], {t, 0, 1}]
2. i rysujemy ostateczny wynik:
   Y = y[t] /. rozw[[1]]
   Plot[Y, {t,0,1}]

Kluczowe elementy zaawansowanej analizy numeryczno-analitycznej

1. generowanie tablic wartości funkcji w przedziale $\{x1, x2\}$ co dx: **tbl=Table[{x , f[x] }, {x,x0,x1,dx }]** , gdzie **f** zawiera np. **FindRoot** lub **NIntegrate**
2. interpolacja i utworzenie nowej funkcji: **g = Interpolation[tbl]**
3. użycie nowo zdefinowanej funkcji **g** w kolejnym kroku
4. powrót do punktu 1

Lista komend *koniecznych* do zaliczenia kolokwium

Lista instrukcji

- grafika
  Plot, ParametricPlot, ImplicitPlot, Evaluate, Show, Plot3D z postatwowymi opcjami
  PlotRange, PlotStyle, PlotLabel, PlotPoints
- sterowanie **Table, If**
- analiza matematyczna **Integrate , D, Sum, Limit, Series**
- rozwiązywanie równań **Solve, Reduce**
- upraszczanie wyrażeń
  Simplify, Expand, Factor, ComplexExpand, Re, Im, Abs, Sign
- dodatkowe warunki **Assuming, Refine**
- równania różniczkowe **DSolve**
- funkcje elementarne **Sin, Cos, Tan, Log, Exp, Sqrt, !**

Lista komend *koniecznych* do zaliczenia kolokwium (c.d.)

Lista instrukcji

- obliczanie wartości numerycznej
  N, NIntegrate, FindRoot, NDSolve
- rachunek macierzowy **Det, Tr, Eigenvalues, Eigenvectors**, **CharactericticPolynomial**, **Dot, Cross**, **Transpose**, **MatrixPower, Inverse, IdentityMatrix, MatrixExp**
- funkcje interpolujące **Interpolation**
- podstawowe stałe **I, Pi, Infinity, E**
- operacje na listach **Dimensions, [[]], Flatten, Select, Union**

Minimum na zaliczenie:
http://ribes.if.uj.edu.pl/alsymb/Wyklad/Zagadnienia.pdf