Algebra Symboliczna

Wykład III

Andrzej Odrzywolek

Instytut Fizyki, Zakład Teorii Względności i Astrofizyki

17.10.2007, 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/

Użyteczne pakiety

Przykłady

- **Graphics** $\longrightarrow$ liczne typy wykresów 2 i 3D
- **Miscellaneous‘RealOnly‘** $\longrightarrow$ obliczenia **bez** liczb zespolonych.
- **NumericalMath** $\longrightarrow$ wersje numeryczne licznych funkcji sybolicznych
- **Statistics‘NonlinearFit‘** $\longrightarrow$ dopasowywanie krzywych np. do wyników pomiarów
- **Calculus‘VectorAnalysis‘** $\longrightarrow$ rachunek wektorowy
- **Utilities‘Benchmark‘** $\longrightarrow$ sprawdza skuteczność komputera z punktu widzenia obliczeń symbolicznych.
- . . . oraz wiele innych przydatnych pakietów.

Ładowanie pakietów

Sposób pierwszy:

«Graphics‘ ładuje wszystkie pakiety z danej grupy. Od tego momentu możemy ich używać w naszej sesji z MATHEMATICA.

Sposób drugi:

«Miscellaneous‘RealOnly‘ ładuje pojedynczą funkcję ***RealOnly*** dostępną w pakiecie ***Miscellaneous***. Od tego momentu możemy jej używać w naszej sesji z MATHEMATICA.

Przykład

Sekwencja poleceń ładująca i uruchamiająca benchmark (test szybkości) systemu:

«Utilities‘Benchmark‘
BenchmarkReport[]

W wersji 6.0 pakiet nazywa się **Benchmarking**

Benchmark Athlon XP-M 512 kB cache 2 Ghz, FSB 150 MHz, 1GB RAM

MathematicaMark6 Detailed Timings

	Total	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Test 11	Test 12	Test 13	Test 14	Test 15
2p dual–core 3.0 GHz Intel Xeon 5160 Microsoft Windows (32–bit)	30.39	2.08	0.67	1.09	2.70	4.03	0.50	0.89	3.17	2.00	2.16	2.41	2.08	1.50	2.59	2.52
2p dual–core 2.66 GHz Intel Xeon Apple Mac OS 10.4.8 (64–bit)	40.32	1.75	0.44	1.85	3.38	11.75	0.30	0.52	3.62	2.07	2.39	2.41	1.56	1.71	3.66	2.91
2p quad–core 1.6 GHz Intel Xeon 5310 CentOS Linux 4.4 (32–bit)	45.78	2.01	1.20	0.93	4.42	8.32	0.88	1.54	5.12	1.61	3.14	3.91	2.73	3.37	3.49	3.12
2p 1.4 GHz Itanium 2 Red Hat Enterprise Linux AS 3.0 (64–bit)	48.55	3.82	0.83	1.39	1.92	2.80	0.33	0.76	7.44	3.29	4.84	7.11	4.90	2.90	2.88	3.34
2p 2.4 GHz Opteron 250 Sun Solaris 10 (64–bit)	69.34	3.92	0.60	3.68	5.79	15.35	0.41	0.90	7.62	4.32	3.31	4.27	3.52	5.68	4.53	5.45
2p dual–core 2.5 GHz G5 Apple Mac OS 10.4.8 (32–bit)	70.59	3.92	1.25	3.88	4.53	18.06	0.81	1.43	6.39	2.29	4.96	5.83	4.64	2.72	6.02	3.89
2.4 GHz Pentium 4 Microsoft Windows XP (32–bit)	86.19	5.05	1.23	3.00	8.84	7.50	0.92	1.63	9.17	8.78	5.39	6.17	5.67	4.51	9.42	8.90
>> tenor << Linux x86 (32–bit)	113.50	6.27	1.02	3.77	9.75	9.49	0.74	1.51	8.07	18.90	5.95	5.06	7.04	4.77	15.20	15.98
4p 1 GHz Power 4 IBM AIX 5.3 (64–bit)	163.75	7.34	1.56	5.90	9.20	60.13	0.94	1.64	10.15	14.40	3.30	13.18	6.99	6.04	10.46	12.50
2p 1.28 GHz UltraSPARC III Sun Solaris SPARC (64–bit)	235.18	12.01	1.97	9.66	12.69	45.34	1.13	2.43	26.31	32.34	7.40	19.71	11.92	5.91	18.90	27.47
2p 552 MHz PA–8600 HP HP–UX 11.11 (64–bit)	405.53	25.27	3.92	30.48	9.76	64.92	33.08	3.41	49.54	22.65	28.23	41.80	24.05	21.10	23.99	23.34

Timings are CPU time in seconds

Liczby zespolone

Wyrażenia zawierające I. zespolone

Sqrt[-1], Log[-1], Exp[I Pi], I^2

Przykłady

- Część rzeczywista: **Re[1+I]**, część urojona: **Im[1+I]**, moduł (wartość bezwzględna) $|1 + i|$ **Abs[1+I]**
- **Sinh[I x]** , **Sin[I x]**, **Exp[I x]**
- Część rzeczywista liczby $a + bi$:
  In:=Re[a + b I]
  `Out= -Im[b]+Re[a]`

Uwaga !!!

Wszystkie obliczenia symboliczne w MATHEMATICE są prowadzone na **liczbach zesplonych** !

Re[a+ b I] — wyjaśnienie

MATHEMATICA zakłada, że każde wyrażenie algebraiczne (**a,b,x,t,f**) jest *liczbą zespoloną* o niezdefiniowanej części rzeczywistej i urojonej.
Właśnie z tego powodu **Re[a+b I]** *nie równa się* **a**.
Nie wiemy czy **a** jest liczbą rzeczywistą !

Podany przez MATHEMATICA wynik:
Re[a+b I] == Re[a]-Im[b] jest poprawny dla dowolnych liczb zespolonych **a** i **b**.
Wyrażenie **Re[a+b I] == a** jest prawdziwe tylko dla rzeczywistych **a** i **b**.

Uwaga

Niedoswiadczeni użytkownicy często krytykują CAS za nieuproszczone „oczywiste" wyrażenia, jak w przykładzie powyżej.
Komputer *nie wie* czy **a** jest rzeczywiste, wszystkie warunki muszą być podane *explicite*!

Upraszczanie wyrażeń zespolonych zbudowanych z domyślnie rzeczywistych składników

Założenie, że wszystkie wyrażenia algebraiczne składają się z liczb *rzeczywistch* uzyskujemy używając **ComplexExpand**

Przykłady

- **Re[a+ b I] // ComplexExpand**
- **ComplexExpand[Abs[a + b I]]**
- **In:=Exp[I ϕ]**
  Out = $e^{i\phi}$
  In:=ComplexExpand[%]
  Out = Cos[ϕ] + i Sin[ϕ]
- **Sqrt[a+ b I] // ComplexExpand**

Uwaga!

ComplexExpand doprowadza do postaci **czescRe + I czescIm**

Upraszczanie „oczywistych” wyrażeń

Rozważmy przykład:

In:=Sqrt[a^2]/a // Simplify

Out = $\frac{Sqrt[a^2]}{a}$

Wyrażenie to jest „w oczywisty sposób” równe dla jednych 1, dla innych $\frac{|a|}{a}$ lub $\operatorname{sgn} a$, a może $\operatorname{sgn} \Re(a)$

Uwaga na „oczywistości” !!!

Rozważmy liczby zespolone: **In:= a={ 1,1+I,I,-1+I,-1,-1-I,-I,1-I}**

1. **Sqrt[a^2]/a** Out={1,1,1,-1,-1,-1,-1,1}
2. **Abs[a]/a** Out = $\{1, \frac{1-i}{\sqrt{2}}, -i, -\frac{1+i}{\sqrt{2}}, -1, -\frac{1-i}{\sqrt{2}}, i, \frac{1+i}{\sqrt{2}}\}$
3. **Sign[a]** Out = $\{1, \frac{1+i}{\sqrt{2}}, i, -\frac{1-i}{\sqrt{2}}, -1, -\frac{1+i}{\sqrt{2}}, -i, \frac{1-i}{\sqrt{2}}\}$
4. **Sign[Re[a]]** Out={1,1,0,-1,-1,-1,0,1}

Upraszczanie „oczywistych” wyrażeń

Rozważmy przykład:

In:=Sqrt[a^2]/a // Simplify

Out = $\frac{Sqrt[a^2]}{a}$

Wyrażenie to jest „w oczywisty sposób” równe dla jednych 1, dla innych $\frac{|a|}{a}$ lub sgn a, a może sgn $\Re(a)$

Uwaga na „oczywistości” !!!

Rozważmy liczby zespolone: **In:= a={ 1,1+I,I,-1+I,-1,-1-I,-I,1-I}**

1. **Sqrt[a^2]/a** Out={1,1,1,-1,-1,-1,-1,1}
2. **Abs[a]/a** Out = $\{1, \frac{1-i}{\sqrt{2}}, -i, -\frac{1+i}{\sqrt{2}}, -1, -\frac{1-i}{\sqrt{2}}, i, \frac{1+i}{\sqrt{2}}\}$
3. **Sign[a]** Out = $\{1, \frac{1+i}{\sqrt{2}}, i, -\frac{1-i}{\sqrt{2}}, -1, -\frac{1+i}{\sqrt{2}}, -i, \frac{1-i}{\sqrt{2}}\}$
4. **Sign[Re[a]]** Out={1,1,0,-1,-1,-1,0,1}

Upraszczanie „oczywistych” wyrażeń

Rozważmy przykład:

In:=Sqrt[a^2]/a // Simplify

Out = $\frac{Sqrt[a^2]}{a}$

Wyrażenie to jest „w oczywisty sposób” równe dla jednych 1, dla innych $\frac{|a|}{a}$ lub $\operatorname{sgn} a$, a może $\operatorname{sgn} \Re(a)$

Uwaga na „oczywistości” !!!

Rozważmy liczby zespolone: **In:= a={ 1,1+I,I,-1+I,-1,-1-I,-I,1-I}**

1. **Sqrt[a^2]/a** Out={1,1,1,-1,-1,-1,-1,1}
2. **Abs[a]/a** Out = $\{1, \frac{1-i}{\sqrt{2}}, -i, -\frac{1+i}{\sqrt{2}}, -1, -\frac{1-i}{\sqrt{2}}, i, \frac{1+i}{\sqrt{2}}\}$
3. **Sign[a]** Out = $\{1, \frac{1+i}{\sqrt{2}}, i, -\frac{1-i}{\sqrt{2}}, -1, -\frac{1+i}{\sqrt{2}}, -i, \frac{1-i}{\sqrt{2}}\}$
4. **Sign[Re[a]]** Out={1,1,0,-1,-1,-1,0,1}

Upraszczanie „oczywistych” wyrażeń

Rozważmy przykład:

In:=Sqrt[a^2]/a // Simplify

Out = $\frac{Sqrt[a^2]}{a}$

Wyrażenie to jest „w oczywisty sposób” równe dla jednych 1, dla innych $\frac{|a|}{a}$ lub $\operatorname{sgn} a$, a może $\operatorname{sgn} \Re(a)$

Uwaga na „oczywistości” !!!

Rozważmy liczby zespolone: **In:= a={ 1,1+I,I,-1+I,-1,-1-I,-I,1-I}**

1. **Sqrt[a^2]/a** Out={1,1,1,-1,-1,-1,-1,1}
2. **Abs[a]/a** Out = $\{1, \frac{1-i}{\sqrt{2}}, -i, -\frac{1+i}{\sqrt{2}}, -1, -\frac{1-i}{\sqrt{2}}, i, \frac{1+i}{\sqrt{2}}\}$
3. **Sign[a]** Out = $\{1, \frac{1+i}{\sqrt{2}}, i, -\frac{1-i}{\sqrt{2}}, -1, -\frac{1+i}{\sqrt{2}}, -i, \frac{1-i}{\sqrt{2}}\}$
4. **Sign[Re[a]]** Out={1,1,0,-1,-1,-1,0,1}

Wprowadzanie założeń

Aby nałożyć pewne warunki na wyrażenie symboliczne możemy użyć funkcji:
Assuming[warunki, obliczane wyrażenie]

Przykłady

Niech **w=Sqrt[a^2]/a**

- **Assuming[a>0,Simplify[w]]** `Out=1`
- **Assuming[a∈Reals, Simplify[w]]** `Out=` $\frac{|a|}{a}$

Uwagi

1. Zawsze należy uważać na wyrażenia typu 0/0, liczby domyślnie zespolone czy wartość bezwzględną
2. W wielu przypadkach wysiłek włożony w zmuszenie CAS do wyświetlenia wyniku w żądanej postaci *nie opłaca się*: szybciej zrobimy to ręcznie! *Miej pod ręką kartkę i długopis!*

Wybieranie interesujacego nas przypadku

Jeżeli w wyniku MATHEMATICA wyprodukuje kilka przypadków, możemy zawęzić problem instrukcją **Refine**:
Refine[wynik, założenia]

Przykłady

- **Integrate[Exp[-a x] x^n, {x, 0, Infinity}]** `Out =If[Re[a] > 0 && Re[n] > -1,` $a^{(-1-n)}$ `Gamma[1 + n],` `Integrate[`$e^{(-ax)}x^n$`, {x, 0, ∞}, Assumptions →` `Re[a] <= 0 || (Re[a] > 0 && Re[n] <= -1)]]`
  Refine[%,a>0 && n>0] `Out =` a^{-1-n} `Gamma[1+n]`
  Assuming[n∈Integers && n>0, FullSimplify[%]]
  `Out =` a^{-1-n} `n!`

Uwaga

Simplify operuje funkcjami standardowymi. **FullSimplify** także funkcjami specjalnymi, np. **Gamma**

Procedura obliczająca liczbę zer kończących silnię liczby naturalnej $n!$

Opis procedury

1. Obliczamy wartość silni (**silnia = 1000!** lub inaczej **silnia = Factorial[1000]**)
   – jest to bardzo duża liczba naturalna – na jej końcu znajduje się duża liczba zer
2. Zamieniamy liczbę naturalną na ciąg znaków (**ToString[%]**)
3. Odczytujemy uzyskany ciąg znaków „wspak” poleceniem **StringReverse[%]**
4. Z powrotem zamieniamy ciąg znaków na liczbę naturalną **ToExpression[%]** –zera są teraz na poczatku więc zostały zignorowane

Procedura obliczająca liczbę zer kończących silnię liczby naturalnej $n!$ - cd

Opis procedury

5. odwracamy liczbę z powrotem
   ToString[%] // StringReverse // ToExpression
6. otrzymaliśmy liczbę 1000! pozbawioną końcowych zer
7. podzielenie 1000! przez tę liczbę daje liczbę złożoną z jedynki i pewnej ilości zer: **1000!/%**
8. Oto wynik:
   10000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000
   00000000000000000000000000000000000000000000000000
   0000000000000000
9. logarytm przy podstawie 10 wyciągnięty z poprzedniego wyniku (**Log[10, %]**) daje szukaną liczbę zer: **249**

Chcemy utworzyć funkcję która po wywołaniu obliczy liczbę zer na końcu silni wg. ww. procedury

```
silniaZero[n_]:= nTMP = n! // ToString // StringReverse //
ToExpression // ToString // StringReverse // ToExpression;
 Log[10, n!/nTMP]
```

W naszym przykładzie:

1. nazwa funkcji $\longrightarrow$ **silniaZero**
2. zmienna $\longrightarrow$ **n** - obliczamy $n!$ z liczby naturalnej.
3. „ciało procedury” $\longrightarrow$
   nTMP = n! // ToString // StringReverse //
   ToExpression // ToString // StringReverse // ToExpression;
4. Zwracana wartość funkcji $\longrightarrow$ **Log[10, n!/nTMP]** (czyli ostatnia obliczona wartość)