

Równania i wykresy przy pomocy modułów Perl

# Matematyka jest prosta

Miliony uczniów i studentów drżą na samą myśl o zajęciach z algebry – tak było i chyba będzie zawsze. My jednak, zamiast zagłębiać się w tabele z wzorami matematycznymi, pozwólmy pracować skryptom języka Perl.

W bieżącym numerze omówimy dwa moduły – pierwszy z nich umożliwia rozwiązywanie równań, drugi tworzy eleganckie wykresy dla tych równań.

MICHAEL SCHILLI



www.photocase.de

Nawet w życiu codziennym walczymy z problemami, w których pojawia się jakaś zmienna  $x$ . Bez względu na to, czy będziemy próbować obliczyć zużycie paliwa, czy wielkość raty kredytu, możemy zastosować do nich zasady algebraiczne. Rozwiązując równania przy użyciu kartki papieru i długopisu możemy jednak popełnić wiele błędów. Na szczęście CPAN daje nam do dyspozycji moduł, dzięki któremu zapanujemy nad tajemnicami algebry.

Zacznijmy od prostego wzoru – w Wielkiej Brytanii i Stanach Zjednoczonych producenci samochodów podają zużycie paliwa w milach na galon. Dla osób mieszkających w Europie kontynentalnej te wielkości niewiele znaczą – do nas przemawia zużycie paliwa wyrażone w litrach na 100 kilometrów. Tak więc dla osób z kontynentu europejskiego wyższa wartość oznacza wyższe zużycie paliwa.

## Zrozumieć Anglosasów

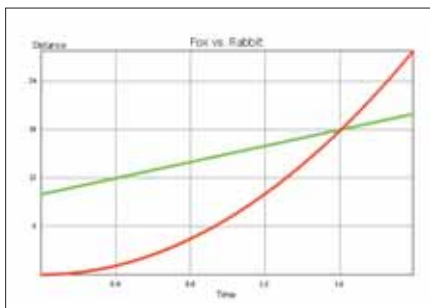
Nasze zadanie będzie polegać na zamianie zużycia paliwa wyrażonego w milach na ga-

lon na europejskie litry na 100 kilometrów. Skrypt `mpgal`, pokazany na Listingu 1, wykorzystuje do tej operacji moduł `Math::Algebra::Symbols` pobrany z serwisu CPAN. Skrypt definiuje dwa symbole: `$gallons`

i `$miles`. Wzór stworzono krok po kroku, zaczynając od linii 16, gdzie definiujemy 1 galon jako 3,7854118 litra. Należy tu zwrócić uwagę na dwie sprawy: moduł `Math::Algebra::Symbols` w obecnej wersji

### Listing 1: Skrypt `mpgal`

```
01 #!/usr/bin/perl
02 #####
03 # mpgal - miles/gallon =>
04 # liters/100km
05 # Mike Schilli, 2004
06 # (m@perlmeister.com)
07 #####
08 use warnings;
09 use strict;
10
11 use Math::Algebra::Symbols;
12
13 my ($gallons, $miles) =
14 symbols(qw(gallons miles));
15
16 my $liters = $gallons *
17 37854118/10000000;
18 my $kilometers = $miles *
19 1609344/1000000;
20 my $usage = $liters /
21 $kilometers * 100;
22
23 print „Formuła: $usage\n”;
24
25 for $miles (qw(20 30 40)) {
26
27 $gallons = 1;
28
29 printf „$miles m/gal: '.
30 '%4.1f l/100km\n',
31 eval $usage;
32 }
```



Rysunek 1: Przy stałym przyspieszeniu lis dogoni zająca biegnącego ze stałą prędkością po około 1,6 sekundy, mimo że zając miał przewagę 10 metrów. Dzięki dwóm modułom Perl zaprogramowanie tej funkcji i wyświetlenie wyników w postaci wykresu jest bardzo proste.

1.16 nie obsługuje rozbudowanych wartości zmiennoprzecinkowych, druga sprawa – symbol \$liters określa liczbę zużytych litrów paliwa. Musimy więc jeszcze pomnożyć liczbę galonów przez 3,7854118. Dwa galony to zatem 7,5708236 litra (\$liters). Ta sama zasada dotyczy kilometrów i mil (1 mila to 1,609344 kilometrów).

Wzór z linii 20 dokonuje obliczenia zużycia paliwa na 100 kilometrów. Jako że zdefiniowaliśmy już wzory dla \$liters i \$kilometers w naszym skrypcie, moduł Math::Algebra::Symbols odejmuje od siebie dane symbole i tworzy równanie, do którego potrzebne są jeszcze wartości galonów (\$gallons) i mil (\$miles). Linia 23 powoduje wyświetlenie wyniku na ekranie:

```
Formuła: 94635295/402336*
$gallons/$miles
```

Moduł Math::Algebra::Symbols przy pomocy matematyki ułamkowej dokonuje skrócenia wartości stałych. Następnie mpgal przypisuje wartości do zmiennych \$gallons i \$miles w celu wprowadzenia rzeczywistych wartości do wzoru i wywołuje w linii 25 eval \$usage. Skrypt dokonuje następnie porównania otrzymanych liczb zużycia paliwa:

```
20 m/gal: 11.8 l/100km
30 m/gal: 7.8 l/100km
40 m/gal: 5.9 l/100km
```

Z takim zadaniem równie dobrze poradziłyby sobie prosta funkcja Perla. Moduł Math::Algebra::Symbols umożliwia jednak przeprowadzanie analiz zmiennych na bardziej skomplikowanych wzorach niż to, co pokazano na Listingu 1.

## Lisy i zające

Następne zadanie to typowe szkolne zadanie z treścią: lis zauważa w odległości 10 metrów od siebie zająca, który biegnie ze stałą prędkością 5 metrów na sekundę. Lis podejmuje za nim pościg. Przyspieszenie lisa wynosi 7 metrów na sekundę do kwadratu. Ile czasu zajmie lisowi dogonienie zająca?

Aby rozwiązać ten problem, skorzystamy z programu Race, pokazanego na Listingu 2, w którym definiujemy symbol \$t jako czas wyrażony w sekundach i z jego użyciem tworzymy układ równań biorący pod uwagę odległość, którą przebiegną zając i lis (linie 13 i 14):

```
my $hare = 10 + 5 * $t;
my $fox = 7 * $t * $t;
```

Zając ma na początku przewagę 10 metrów. Aby obliczyć pokonany dystans, będziemy musieli użyć wzoru na pokonaną drogę ze stałą prędkością ( $S = V * t$ ). W określonym czasie  $t$  zając pokona odległość  $10 + 5 * t$ . Dzięki zastosowaniu wzoru na stałe przyspieszenie ( $S = a * t^2$ ) możemy stwierdzić, że lis dogoni zająca dokładnie po  $7 * t^2$  metrach. Lis dogoni zająca, gdy obie wartości będą sobie równe, czyli w miejscu, gdzie równanie z linii 17 osiągnie wartość zero.

Aby rozstrzygnąć ten problem na papierze, musielibyśmy rozwiązać równanie kwadratowe, co oznaczałoby rozpoczęcie poszukiwań potrzebnych wzorów i macierzy. Dzięki modułowi Math::Algebra::Symbols możemy rozwiązać to w prosty sposób

(\$gotcha) przy zastosowaniu metody \$gotcha->solve('t') wobec niewiadomej \$t. Podczas rozwiązywania równania kwadratowego dojdziemy do punktu, w którym uzyskamy w rezultacie tablicę (macierz) dwóch równań (linia 20):

```
Solution: 1/14*sqrt(305)+5/14
Solution: -1/14*sqrt(305)+5/14
```

Jako że ujemne wartości dla czasu nie są oczywiście dopuszczalne, możemy z czystym sumieniem odrzucić drugi wynik z linii 25. Aby przekształcić uzyskane rozwiązanie na wartość zmiennoprzecinkową, korzystamy w linii 23 z modułu Perl o nazwie eval.

Na nieszczęście dla zająca, pościg będzie trwał około 1,60 sekundy. Po podstawieniu pod zmienną \$t tej wartości w linii 30 otrzymamy także dystans, jaki przebiegnie lis: używamy eval \$fox i otrzymujemy odległość około 18,02 metrów.

## Magia wykresów

Listing 3 to graficzne przedstawienie pościgu (Rysunek 1). Moduł Imager::Plot wraz z kilkoma liniami kodu w Perlu pozwala generować zaawansowane wykresy w różnych formatach plików graficznych. W linii 14 utworzono nowy obiekt Imager::Plot przy użyciu czcionki wektorowej tahoma.ttf, wykorzystanej w legendzie wykresu. W zależności od posiadanej dystrybucji, będziemy musieli zmienić ścieżkę dostępu do czcionki.

Pętla for, rozpoczynająca się w linii 24, generuje dane do wykresu z krokiem co 0,01 od zera do wartości 2,0, tworząc w ten

## Listing 2: Skrypt race

```
01 #!/usr/bin/perl
02 #####
03 # race – wyścig lisa z zającem
04 # Mike Schilli, 2004
05 # (m@perlmeister.com)
06 #####
07 use warnings;
08 use strict;
09
10 use Math::Algebra::Symbols;
11
12 my ($t) = symbols(qw(t));
13
14 my $hare = 10 + 5 * $t;
15 my $fox = 7 * $t * $t;
16
17 my $gotcha = ($hare - $fox);
18
19 for my $solution
20 (@{$gotcha->solve('t')}) {
21 print 'Solution: ',
22 „$solution\n”;
23 my $val = eval $solution;
24 if($val < 0) {
25 print „Discarded\n”;
26 next;
27 } else {
28 printf „%.2f seconds\n”,
29 $val;
30 $t = $val;
31 printf „%.2f meters\n”,
32 eval $fox;
33 }
34 }
```

## Listing 3: Graf do wyścigu lisa z zajęcem

```

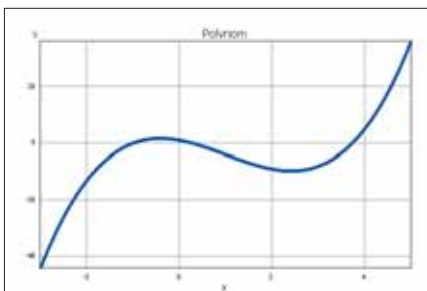
01 #!/usr/bin/perl
02 #####
03 # graph -- graf do zadania wyścig
04 # lisa z zajęcem
05 # Mike Schilli, 2004
06 # (m@perlmeister.com)
07 #####
08 use strict;
09 use warnings;
10
11 use Imager;
12 use Imager::Plot;
13
14 my $plot = Imager::Plot->new(
15 Width => 550,
16 Height => 350,
17 GlobalFont =>
18 '/usr/share/fonts'.
19 '/truetype/tahoma.ttf');
20
21 my (@t, @hare, @fox);
22
23 # Generate function data
24 for(my $i = 0.0; $i < 2.0;
25 $i += 0.01) {
26 push @t, $i;
27 push @hare, 10 + 5 * $i;
28 push @fox, 7 * $i * $i;
29 }
30
31 # Add hare plot
32 $plot->AddDataSet(
33 X => \@t,
34 Y => \@hare,
35 style => {
36 marker => {
37 size => 2,
38 symbol => 'circle',
39 color =>
40 Imager::Color->new(
41 'green')}}});
42
43 # Add fox plot
44 $plot->AddDataSet(
45 X => \@t,
46 Y => \@fox,
47 style => {
48 marker => {
49 size => 2,
50 symbol => 'circle',
51 color =>
52 Imager::Color->new(
53 'red')}}});
54
55 my $img = Imager->new(
56 xsize => 600,
57 ysize => 400);
58
59 $img->box(filled => 1,
60 color => 'white');
61
62 # Add text
63 $plot->{'Ylabel'} =
64 'Distance';
65 $plot->{'Xlabel'} = 'Time';
66 $plot->{'Title'} =
67 'Fox vs. hare';
68
69 $plot->Render(Image => $img,
70 Xoff => 40, Yoff => 370);
71
72 $img->write(
73 file => „graph.png”);

```

sposób trzy układy: @t dla wartości na osi X oraz @hare lub @fox dla wartości przebytej odległości zajęcia i lisa. Wartości te przedstawiono w stosunku do czasu przy pomocy odpowiednich formuł.

W linii 32 dodano na układzie współrzędnych zieloną linię wykresu dla zajęcia, w wierszu 44 i kolejnych dodano czerwoną linię dla lisa. Funkcja Render, znajdująca się w linii 69, obsługuje rysowanie, a funkcja write, zawarta w wierszu 72, powoduje zapisanie całego wykresu w pliku graficznym w formacie PNG.

Zadanie z pościgiem lisa za zajęciem bez wątplenia nie wykorzystuje w pełni możli-



Rysunek 2: Wykres wielomianu  $t^3 - 3t^2 - 3t + 1$ . Moduł Perl `Math::Algebra::Symbols` umożliwia leniwym programistom łatwe określenie ekstremów funkcji.

wości modułu `Math::Algebra::Symbols`. Rysunek 2 przedstawia wielomian  $y = t^3 - 3t^2 - 3t + 1$ . Pokazane dwie amplitudy przedstawiają wartości maksymalne i minimalne. Z czasów szkolnych pamiętamy, że nachylenie krzywej w tych punktach wynosi zero. Aby odczytać odpowiadające tym punktom wartości X, musimy wyznaczyć pochodną, przyrównać ją do zera i rozwiązać stworzone w ten sposób równanie. Na szczęście moduł `Math::Algebra::Symbols` potrafi wyznaczać pochodne funkcji prostych:

```

my ($x) = symbols('x');
my $y = $x**3 - 3*$x**2 - 3*$x + 1;
my $diff = $y->d('x');

my $extrema = $diff->solve('x');
print eval($_), "\n" >
for @$extrema;

```

Metoda `$y->d('x')` oblicza pochodną funkcji zdefiniowanej w `$y` do `$x`, przetwarzając wielomian trzeciego stopnia na wielomian kwadratowy. Funkcja `solve()` rozwiązuje ten wielomian, zwracając wynik w postaci macierzy dwuelementowej, którą funkcja

eval przetwarza na wartości zmiennoprzecinkowe:

```

-0.414213562373095
2.41421356237309

```

W ten sposób otrzymujemy wartości X dla ekstremów. Moduł `Math::Algebra::Symbols` potrafi obsługiwać także dowolną liczbę funkcji trygonometrycznych, mimo że napotkaliśmy na problemy przy bardziej złożonych funkcjach tego typu.

## Do pracy!

Moduły `Math::Algebra::Symbols` i `Imager::Plot` można pobrać ze strony CPAN. Są one wprost stworzone do rozwiązywania problemów matematycznych. `Math::Algebra::Symbols` jest w zasadzie modułem we wcześniejszej wersji alpha, ale jego autor Philip Brenan ciągle prowadzi prace nad rozwinięciem jego funkcjonalności. Być może pewnego dnia będzie to moduł równie potężny co słynna Mathematica. ■

## INFO

[1] Listingi programów z artykułu:  
<http://www.linux-magazine.com/Magazine/Downloads/46/Perl>