1 Przypadek jednej niewiadomej rzeczywistej, bez parametrów

1.1 Zalecana metodologia rozwiązywania

W tym rozdziale zajmiemy się zagadnieniem polegającym na rozwiązaniu równania o jednej niewiadomej *bez parametrów* w domenie liczb rzeczywistych. Oznacza to, że w równaniu występuje tylko jedna niewiadoma wielkość, zwykle oznaczona jako x, a wszystkie pozostałe składniki równania są bądź podane, bądź umiemy je obliczyć.

Oto przykłady takich równań:

$$x^2 + 5 = 2 \tag{1a}$$

$$x^3 + x - 1 = 0 \tag{1b}$$

$$e^2 = x^2 \tag{1c}$$

$$\Gamma\left(\frac{1}{9}x + |x| + \pi\right) = \frac{1}{1 + x^2} \tag{1d}$$

$$\sin^2 3x = 1 - \cos x \tag{1e}$$

Metodologia rozwiązywania tego typu równań w Mathematice i innych CAS jest następująca:

1. Rysujemy lewą i prawą stronę równania na jednym wykresie w celu wizualnego znalezienia punktów przecięcia, które *de facto* same z siebie stanowią już przybliżone rozwiązanie danego równania w przedziale zadanym jako (na ogół obowiązkowy!) parametr intrukcji rysującej, typowo **Plot**. Podstawowa informacja którą otrzymujemy to, *istnienie* i *ilość* rozwiązań. W razie potrzeby wykonujemy więcej wykresów w różnych przedziałach.

   UWAGA: Równanie może nie mieć *żadnego* rozwiązania!

2. Zawężając przedział, w którym rysujemy obie strony równania lub/i posługując się apletem pozwalającym na zaznaczenie punktu na wykresie i skopiowanie jego współrzędnych klawiszami Ctrl+C, Ctrl+V odczytujemy *przybliżone* rozwiązania. W wielu przypadkach o to właśnie chodzi, i zadanie zostaje rozwiązane.

3. Jeżeli potrzebujemy znacznie dokładniejszego rozwiązania, używamy odczytanych wartości jako punktów startowych (znów obowiązkowych!) w instrukcji **FindRoot**.

4. Jeżeli interesuje nas istnienie rozwiązań symbolicznych wyrażonych przez funkcje elementarne (w rozumieniu programu Mathematica oznacza to znacznie więcej funkcji niż większość użytkowników kiedykolwiek napotka) to próbujemy instrukcji, w podanej kolejności:

   - **Solve**
   - **Reduce**
   - **FindInstance**

   Nie zapomnijmy o dodaniu opcjonalnego argumentu **Reals**, co zawęzi obszar poszukiwań do liczb rzeczywistych.

5. Nanosimy rozwiązania uzyskane w punktach 3 lub/i 4 na wykresy uzyskane w pkt. 1. Upewni nas to, że ani my, ani *Mathematica* nie popełniła żadnego błędu. Jeżeli wyniki uzyskane metodami numerycznymi lub symbolicznymi pokrywają się z przecięciami krzywych i wynikiem numerycznym (z **FindRoot**) istnieje wysokie prawdopodobieństwo, że równanie zostało rozwiązane poprawnie.

Ponieważ większość studentów z oporami przymuje wyżej podany schemat działania, z uporem usiłując wykonać go w odwrotnej kolejności, należy się kilka słów wyjaśnienia. Metody graficzne często uważa się za „gorsze" od innych. Nic bardziej mylnego. Po pierwsze, ich stosowanie w *Mathematice* praktycznie „nic nie kosztuje" dzięki ogromnej mocy obliczeniowej współczesnych procesorów i zaawansowanej grafice komputerowej. Po drugie, narysowanie lewej i prawej strony równania upewnia nas, że nie popełniliśmy błędów składniowych, oraz, że po ich obliczeniu otrzymujemy liczby rzeczywiste. Pusty wykres oznacza na ogół:

- błędną składnię np. **xSin[x]** zamiast **x Sin[x]** lub **x*Sin[x]**; *Mathematica* potraktuje symbol **xSin[x]** jako pewną nieokreśloną funkcję argumentu x o nazwie **xSin**
- brak rzeczywistych numerycznych wartości funkcji; należy pamiętać, że współrzędne piksela na ekranie są liczbami całkowitymi (!) i komputer nie może wyświetlić piksela o współrzędnych np. { 12 a, 14 a }, gdzie a to pewien nieokreślony symbol. Podobnie w przypadku liczb zespolonych, nawet jeżeli część urojona jest zaniedbywalnie małym czynnikiem powstałym na skutek błędów zaokrąglania.
- źle dobrana skala na osiach lub/i zbyt mała dokładność wykresu

Dobrą praktyką jest obliczenie współrzędnych przynajmniej jednego punktu wykresu poleceniem **N** *przed* wywołaniem instrukcji **Plot**. Jeżeli nie otrzymamy liczby rzeczywistej[1] wykres nie będzie możliwy do narysowania. Unikamy dzięki temu większości problemów z poleceniem **Plot** i pokrewnymi.

Kolejnym argumentem wysuwanym przez studentów przeciwko rysowaniu równań, jest brak informacji o obszarze w którym szukamy rozwiązań. Rzeczywiście, o ile nie jest to podane w treści wypadałoby zbadać całą oś rzeczywistą. Typowo używany przedział $x = \{-10, 10\}$ nadaje się głównie do spreparowanych zadań szkolnych oraz wąskiej klasy problemów. Nic jednak nie stoi na przeszkodzie, aby wykonać wiele wykresów w różnych skalach. Jeżeli nic nie wiemy o x, lub spodziewamy się, że zmienia się o wiele rzędów wielkości pomocne są instrukcje **LogPlot**, **LogLogPlot** oraz **LogLinearPlot**. Dokładność wykresu możemy zwiększać przy pomocy opcji np. `PlotPoints → 100, MaxRecursion → 15`. W bardzo wątpliwych przypadkach można dokonać zamiany zmiennej x w taki sposób aby nowa zmienna przyjmowała wartości w skończonym przedziale, np. $t = \tanh x$ lub $t = x/\sqrt{1+x^2}$. Takiej automatycznej zamiany zmiennych dokonuje Maple przy wywołaniu polecenia np. **plot(exp(x),x=-infinity..infinity);**. W przypadku problemów fizycznych oraz technicznych pomaga użycie adekwatnych jednostek miar i przedrostków lub wprowadzenie zmiennych bezwymiarowych.

Warto dodać, że zaawansowane metody symboliczne których używa **FindInstance** dosłownie wręcz przeszukują całą oś rzeczywistą - nie jest to więc nic dziwnego czy niespotykanego.

Należy pamiętać też, że rekursywnie rysując wykres i zawężając przedział x jesteśmy w stanie *dowolnie dokładnie* rozwiązać każde równanie. Dlatego formalnie już sam wykres stanowi przybliżone rozwiązanie równania. Instrukcje typu **FindRoot** nie robią nic bardziej egzotycznego[2] - nie wyświetlają tylko wyników pośrednich na wykresie, a zawężania przedziału dokonują automatycznie, w pamięci podręcznej procesora. Dlatego metody ręcznej biskecji nie należy stosować na codzień - do tego służy **FindRoot**.

[1] Precyzyjnie mówiąc, chodzi o liczbę zmiennoprzecinkową. Np: 3.14, 0.012, 1.23e-11 itp.

[2] Szczególnie algorytm bisekcji.

Potrzeba wyrażenia znalezionego rozwiązania numerycznego przez symbole matematyczne jak $\pi, \frac{3}{7}, \sqrt{3}, \sin$ itp. zależy od kontekstu. Wynik symboliczny jest lepszy od każdego numerycznego, gdyż posiada *nieskończoną* precyzję. Podanie nawet miliona cyfr rozwinięcia liczby π nie zastępuje wiedzy, że wynik faktycznie wynosi π! Odpowiedź na pytanie czy $x < \pi$ nie jest rozstrzygalne numerycznie, a może decydować np. o stabilności układu. Błędy w wynikach uzyskanych poprzez manipulacje symboliczne nie akumulują się.[3] Dlatego warto poświęcić przynajmniej trochę czasu na próbach znalezienia rozwiązania symbolicznego. Może to okazać się znacznie trudniejsze niż kroki 1-3, na dodatek bez gwarancji powodzenia. Ze względu na charakter algorytmów używanych w manipulacjach symbolicznych czynnikiem limitujacym może okazać się czas procesora czy pamięć komputera.

Kolejny argument aby **nie zaczynać** od ostatniego punktu, ma charakter „obserwacyjny". Często widzę studentów, którzy po przeczytaniu treści zadania, wklepują ją z klawiatury dodając **Solve** czy **Reduce**, wciskają `Shift+Enter`, zakładają ręce za głowę i wpatrują się w monitor przez pół godziny, po czym uznają zadanie za niewykonalne. W innej typowej sytuacji student opada zdumiony na krzesło obserwując jak *Mathematica* wyświetla 10 ekranów wypełnionych kombinacjami instrukcji logicznych przemieszanych z licznymi wyrażeniami typu **Root** zamiast spodziewanego np. `x==3/2`. Wiedza matematyczna potrzebna do zinterpretowania takich wyników często przekracza spodziewany rezultat o całe lata edukacji. W skrajnych wypadkach student może nie odróżnić wyniku od błędu programu.

Jeżeli natomiast wcześniej rozwiązaliśmy równanie graficznie, i znamy jego przybliżenie numeryczne, jesteśmy gotowi do zinterpretowania wyniku. Oczekujemy *pewnej liczby rzeczywistej*, powiedzmy $x \simeq 1.5$. Jeżeli **Reduce** zablokuje komputer na 30 minut, zrobi to w momencie w którym już posiadamy rozwiązanie graficzne i numeryczne. Jeżeli mamy cierpliwość i nowoczesny komputer, możemy poczekać. Obliczenia z **Reduce** trwające kilka minut są typowe, a **FindInstace** bez problemu zajmuje komputer na godzinę. Wczesniej zwykle wyczerpana zostaje wolna pamięc.

Poszukiwanie rozwiązań symbolicznych ciągle jest sztuką. Oto garść porad:

- Zacznij od **Solve**. Znajduje część rozwiązań używając głównie funkcji odwrotnych, np. **Solve[Sin[x]==4/3,x]** daje $\arcsin 4/3$. Bardzo często, np. w zadaniach z fizyki czy obliczeniach inżynierskich właśnie tego rozwiązania szukamy.

  UWAGA: **Solve** operuje w dziedzinie **liczb zespolonych**; w przykładzie wyżej $\arcsin 4/3 = \frac{\pi}{2} - i \ln \frac{4+\sqrt{7}}{3}$!

- **Reduce**, o ile to możliwe, znajduje komplet rozwiązań, czyli daje poprawną w sensie matematycznym odpowiedź. Opcjonalny argument `Reals` używamy, jeżeli interesują nas tylko wyniki rzeczywiste. Standardowo **Reduce** szuka wszystkich rozwiązań *zespolonych*. Jest ich zwykle znacznie więcej niż spodziewają się osoby nawet dobrze obeznane z uniwersytecką i szkolną matematyką. Zrozumienie i ogarnięcie wyniku wyświetlonego przez **Reduce** zajmuje często sporo czasu, ze względu na postać w postaci wyrażeń logicznych. Proponuję przeznaczyć ten czas na rozwiązanie graficzne i przemyślenie treści zadania.

- **FindInstance** jest poleceniem o potężnych możliwościach, aktywnie rozwijanym. Wywołane bez opcji znajduje jedno rozwiązanie równania, lub udowadnia, że rozwiązanie nie istnieje. Rozwiązuje prawie wszystko, co da się zapisać wzorem matematycznym składającym się z funkcji elementarnych. Potrafi znajdować wiele rozwiązań a nawet definiować nowe stałe przestępne za pomocą specjalnej postaci polecenia **Root**. Jeżeli podamy przedział w którym znajdują się rozwiązania oraz ich ilość to można oczekiwać, że **FindInstance** znajdzie i poda

[3] Nie rozważamy tu błędów mających charakter software-owych bug-ów!

wszystkie! Niestety, czas obliczeniowy pożerany przez to polecenie jest wyjątkowo długi. Dla przykładu: rozwiązanie równania $x^x + \ln x^{x^x} == 0$ przez **FindInstance** zajmuje 1000 razy więcej czasu[4] niż przez **Solve**.

1.2 Rozwiąż równanie $2x + 3 = 4$.

Oto najprostszy przykład, może nawet trywialny, ale dzięki temu możemy się skupić na obsłudze oprogramowania, wyjaśnienieniu użytej notacji oraz poleceń.

Rozwiążemy w dziedzinie liczb rzeczywistych równanie, w notacji „tablicowej" podane jako:

$$2x + 3 = 4 \tag{2}$$

Rozwiązanie tradycyjne: obustronnie odejmujemy 3 (równoważna czynność, przenosimy 3 z lewej na prawą stronę zmieniając znak):

$$2x = 4 - 3,$$

co daje:

$$2x = 1,$$

Następnie, obustronnie dzielimy przez 2:

$$x = \frac{1}{2}.$$

Dysponując komputerem **zawsze** należy rozpocząć od metod graficznych, czyli rysowania wykresów wyrażeń pojawiających się jako składowe równania. Dotyczy to przede wszystkim równań, które poza niewiadomą (w tym wypadku x) nie zawierają żadnych innych symboli, za wyjątkiem liczb (np : 2, 3, 1/2). Dotyczy to także powszechnie znanych symboli np: liczby π (**Pi**). Typowo, równanie składa się z dwóch stron przedzielonych znakiem równości (logicznej). Określamy je zwykle jako lewa i prawa strona, odpowiednio do tego, z której strony znaku równości się znajdują. Z języka angielskiego pochodzą skróty:

- LHS (Left Hand Side, , , po stronie lewej dłoni") - lewa strona równania
- RHS (Right Hand Side, , , po prawej stronie dłoni") - prawa strona równania

Najważniejszym stwierdzeniem tego podrozdziału jest:

rozwiązaniem graficznym równania, jest punkt, lub zbiór punktów, w których wykresy lewej i prawej strony równania przecinają się.

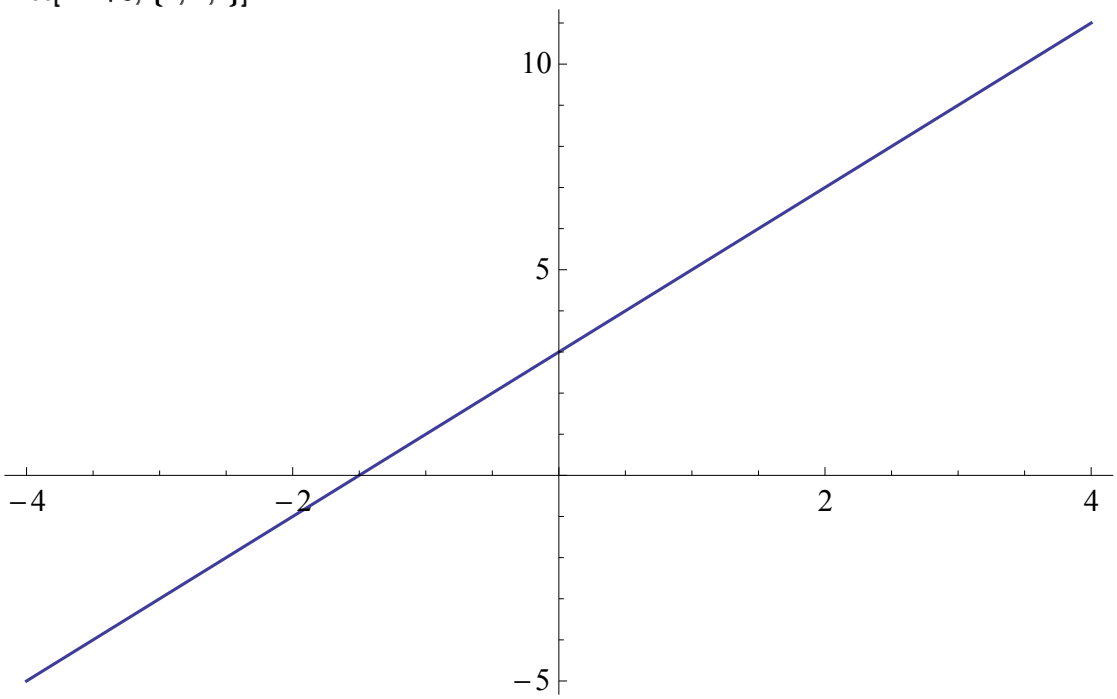
UWAGA: bardzo często prawa strona równania to ZERO. W takim wypadku rozwiązaniem równania jest przecięcie z osią Ox.Programy komputerowe nie zawsze rysują tę oś! Dlatego warto dla pewności narysować wykres funkcji równej zero; jest to taka sama funkcja jak wszystkie inne. Jej wykresem jest linia pozioma. W naszym, przykładzie lewa strona to 2 x + 3, natomiast prawa strona to 4.

UWAGA: symbole 2 x + 3 oznaczają „2 razy x dodać trzy". W notacji „tablicowej" symbol mnożenia na ogół pomija się. Należy przy tym przestrzegać niepisanej reguły, która mówi, że w wyrażeniu matematycznym liczby całkowite powinny być na początku wyrażenia.

[4]Na moim komputerze PC, **Solve** potrzebowało 0.02 sekundy, natomiast **FindRoot** 25 sekund. Prawo Moore'a stwierdza, że ilość tranzystorów podwaja się co 18 miesięcy. Za 20 lat **FindInstance** będzie więc, być może, działać równie szybko (relatywnie do czasu ludzkiej reakcji) jak **Solve** dzisiaj.

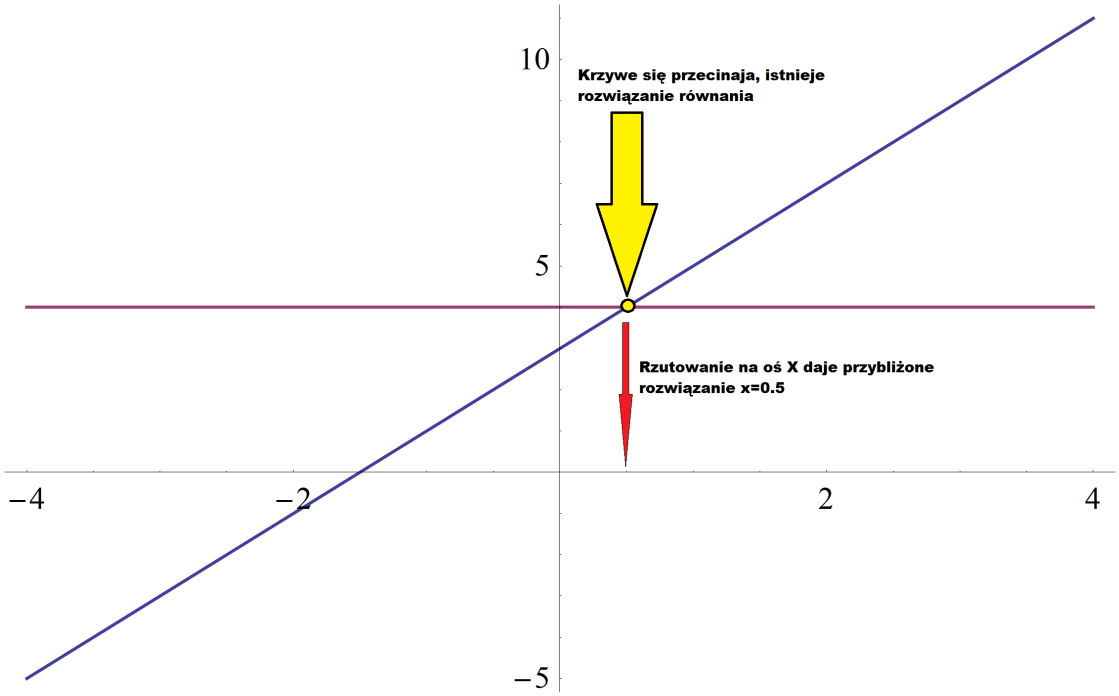
Podstawowa instrukcja graficzna to **Plot**, czyli polecenie rysujące wykresy wyrażeń matematycznych. Jego struktura jest prototypowa dla większości przydatnych studentom funkcji. Narysujemy na początek wykres wyrażenia 2x+3:

Plot[2*x+3, {x,-4,4}]



Spróbujemy teraz rozwiązać graficznie równanie „2 x + 3 = 4". Rysujemy w tym celu dwa wyrażenia: $2x + 3$ oraz 4, na jednym wykresie:

Plot[{2x+3, 4},{x,-4,4}]

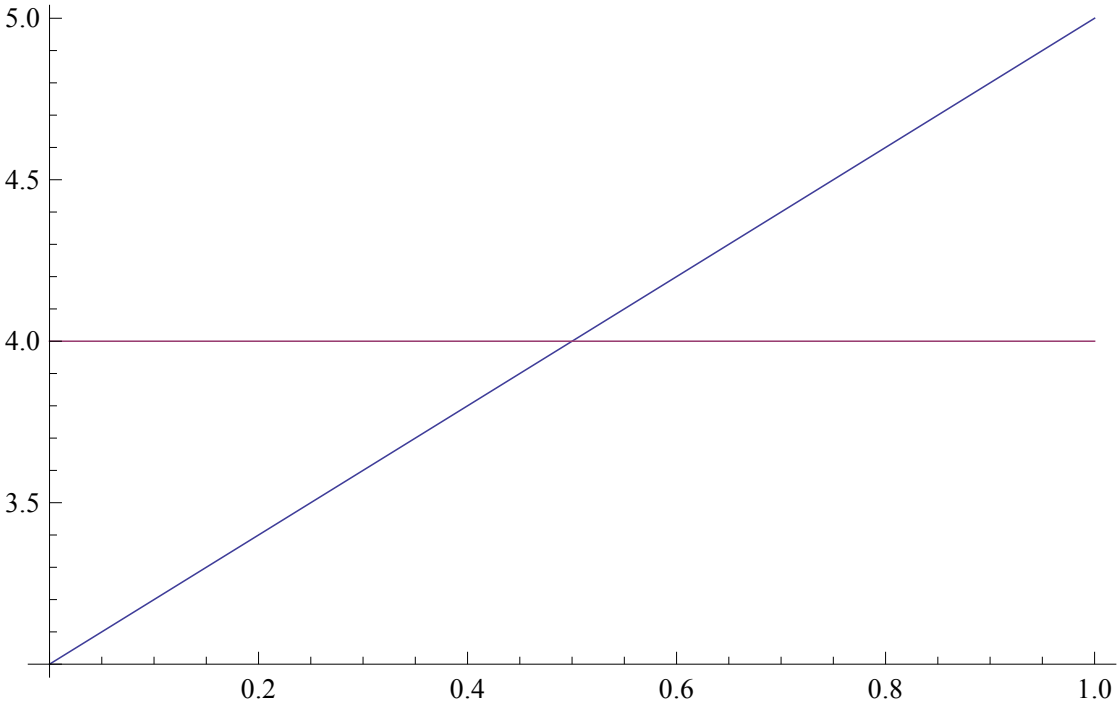


Z wykresu odczytujemy łatwo, że rozwiązaniem jest $x \simeq 0.5$. Metody graficzne nie pozwalają na ustalenie czy jest to dokładnie 1/2. Zyskujemy jednak kilka cennych informacji:

1. widzimy wykresy, co oznacza, że rozumiemy treść zadania i potrafimy poprawnie przepisać ją w *Mathematice*
2. rozwiązanie jest prawdopodobnie tylko jedno
3. wykresami są linie proste, lub niemal proste (nie są to skomplikowane krzywe)
4. wykres pojawił się natychmiast, co oznacza, że wyrażenia są łatwe do obliczenia
5. rozwiązanie jest w pobliżu x=0.5

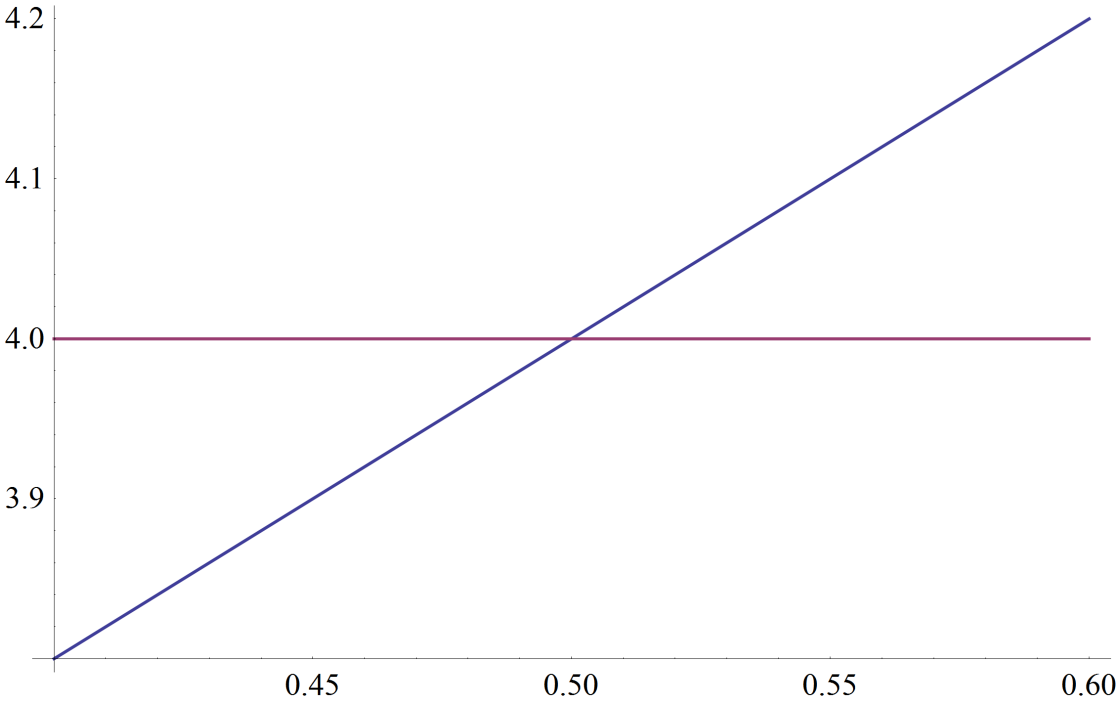
Rozwiązanie łatwo uściślić, rysując wykres „zoomowany" na $x = 0.5$, np: w przedziale $0 < x < 1$:

Plot[{2x+3,4},{x,0,1}]

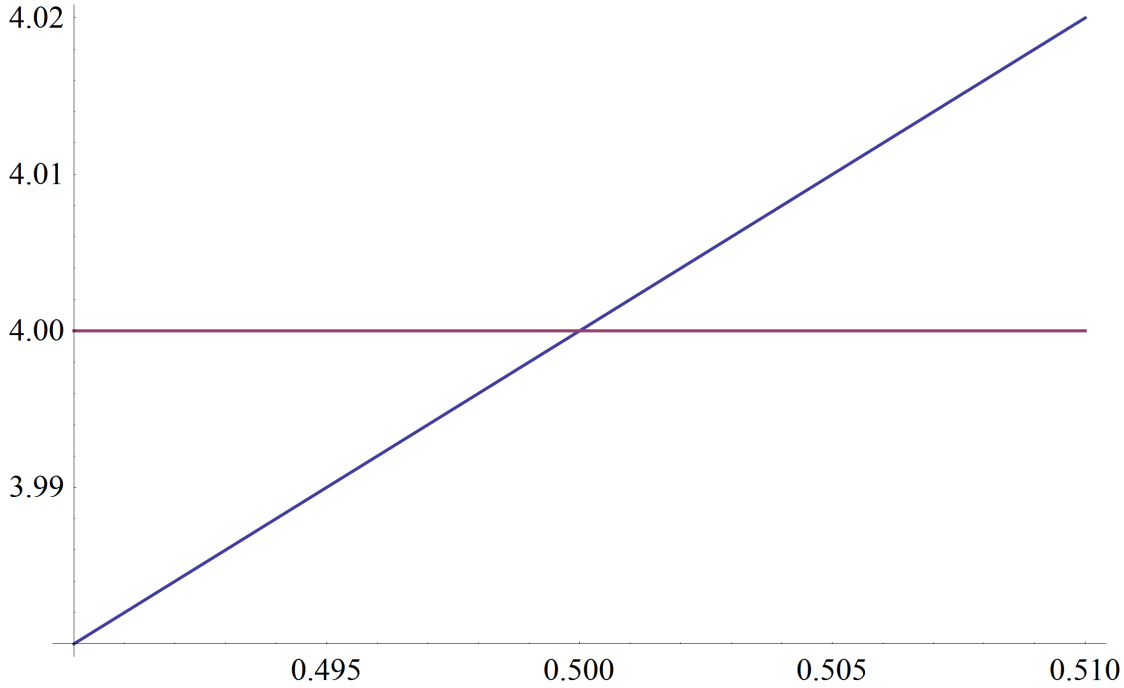


Powtarzając tę czynność, możemy w zasadzie dowolnie dokładnie wyznaczyć rozwiązanie:

Plot[{2x+3,4}, {x,0.4,0.6}]



Plot[{2x+3,4},{x,0.49,0.51}]



Oprócz metod graficznych mamy dostępne w *Mathematice* **metody numeryczne** oraz **metody symboliczne**. Ich zastosowanie wymaga umiejętności zapisania równania. W większości istniejących języków programowania, i *Mathematica* nie jest tu wyjątkiem, istnieją dwa osobne symbole na oznaczenie równości matematycznej (Equal, podwojony symbol ==) oraz instrukcji przypisania (Set , =).

UWAGA: omyłkowe użycie znaku pojedynczej równości zamiast podwojonego, jest najczęstszym błędem[5] popełnianym przez studentów!

Próba wpisania równości $2x + 3 = 4$ z użyciem **pojedynczego** znaku równości kończy się komuniaktem o błędzie:

2 x +3 = 4

```
2x+3=4 Set::write: Tag Plus in 3+2 x is Protected. >>
```

Poprawny sposób wymaga użycia znaku podwójnej równości:

2x+3==4

Znak pojedynczej równości, inaczej instrukcja przypisania, służy do zapisywania pod wygodnymi dla nas nazwami wyrażeń, wyników itp. Różnicę pomiędzy instrukcją przypisania a równością logiczną ilustruje ponizsza linijka :

rownanie = (2x+3==4)

Przykładowe równanie $2x+3 = 4$ zostało „zapamiętane" pod nazwą **rownanie**. Rolą nawiasów okrągłych ( ) jest grupowanie wyrażeń[6] i ustalenie kolejności wykonywania działań. Od tego momentu wypisanie polecenia:

rownanie

[5]Bardzo przydatna jest umiejętność usunięcia skutków tego błedu, czyli przypisanej wartości np: x = 1. Można to zrobić przynajmniej na dwa sposoby (1) **Clear[x]** lub (2) **x=.** Początkujący często popełniają omyłkowe przypisania wielokrotnie. W takiej sytuacji konieczne może był wyłączenie i ponowne włączenie Kernela. W tym celu za pomocą kursora myszki wybieramy z Menu polecenie Evaluation → Quit Kernel → Local.

[6]Nawiasy okrągłe nie są w tym wypadku konieczne. Dałem je dla przykładu.

da po naciśnięciu `Shift+Enter`:

```
3+2 x==4
```

Aby podstawić do równania pewną wartość x, np: $x = 2$, należy użyć operatora „zamień wszystko", zapisywanego jako **/.** (ukośnik + kropka), oraz reguły podstawiania (**Rule**) x→2 (minus + znak większości). Na przykład, wstawmy do równania $x = 1$:

rownanie /. x → 1

Jako wynik dostajemy `False`. Po podstawieniu:

rownanie /. x → 1/2

lub:

rownanie /.x → 0.5

otrzymamy `True` (ang. prawda), co oznacza, że $x = 1/2$ spełnia równanie.

Warto zwrócić uwagę na nastepującą ciekawostkę. Wstawiając różne wartości x możemy przypadkowo natrafić na poprawne rozwiązanie. Gdy napiszemy odpowiedni program, komputer może te podstawienia wykonywać automatycznie. Można w ten sposób „rozwiązać" niektóre problemy. Szczególnie dotyczy to zadań szkolnych, w których niepisane reguły zmuszają autorów pytań do konstruowania treści w taki sposób, aby wynik nie był zbyt skomplikowany.

Umiejąc zapisać równanie[7], możemy użyć metod numerycznych lub symbolicznych.

Oto lista najważniejszych instrukcji:

- metody **numeryczne**:
  1. **FindRoot** (dowolne równania)
  2. **NSolve** (problemy algebraiczne, np. pierwiastki wielomianów, układy równań wielomianowych)
- metody **symboliczne**:
  1. **Solve** („naiwne" algorytmy rozwiązywania równań, szczególnie przydatne w fizyce i technice)
  2. **Reduce** (ścisłe matematycznie rozwiązania)
  3. **FindInstance** (współczesne, stale rozwijane algorytmy dla liczb rzeczywistych, możliwość wyszukiwania wielokrotnych rozwiązań i definiowania nowych symboli liczb przestępnych, tylko dla równań bez parametrów)

UWAGA: metody symboliczne pozwalają także na rozwiązywanie nierówności.

KOLEJNOŚĆ w jakiej wymienione zostały powyższe polecenia nie jest przypadkowa. Odzwierciedla w przybliżeniu spodziewany czas w jakim należy spodziewać się uzyskania wyniku. Z drugiej strony „ścisłość" uzyskanego rezultatu jest tym większa im dalsza pozycja na powyższej liście. Te zagadnienia omawiam dokładniej na kolejnych przykładach.

Spodziewane efekty uruchomienia polecenia z powyższej listy są następujące:

- METODY NUMERYCZNE
  - **FindRoot** - jedno, przybliżone rozwiązanie równania; wynik na ogół pewny w przedziale zadanej dokładności, uzyskany zwykle w czasie niezauważalnie małym
  - **NSolve** - wszystkie rozwiązania równań wielomianowych, ale każde z nich przybliżone; metoda pewna w podanej kategorii równań, może wymagać długiego czasu obliczeń

[7] Zastosowanie metody graficznej *nie wymaga* takiej wiedzy!

- METODY SYMBOLICZNE
  - **Solve** - rozwiązuje równania „naiwym" algorytmem, np. stosując f. odwrotne; odpowiedź jest często niekompletna (brakuje rozwiązań), zdarza się, że błędna; dosyć szybka
  - **Reduce** - rozwiązuje równania ściśle matematycznie, uwzględniając wszystkie przypadki; odpowiedź jest na ogół bardzo złożona, a jej uzyskanie wymaga sporo czasu; wynik na ogół jest zupełnie niezrozumiały dla początkujących użytkowników; rozwiązania, o ile zostaną znalezione są na ogół poprawne
  - **FindInstance** - najsilniejsze, ale także najwolniejsze z poleceń (bywa, że wymaga godzinnych obliczeń); potrafi znajdować wiele rozwiązań, a także udowodnić, że nie ma innych; stosuje bardzo zaawansowane metody matematyczne; działa tylko dla równań bez parametrów; definiuje nowe liczby przestępne

Zobaczmy jak to wygląda w praktyce. **FindRoot** powinien znaleźć rozwiązanie w pobliżu punktu startowego, np: odczytanego z wykresu:

FindRoot[rownanie,{x,0.5}]

```
{x → 0.5}
```

W tym przypadku możemy użyć także **NSolve**, gdyż równanie zawiera tylko wielomiany. Nie potrzeba punktów statowych:

NSolve[rownanie, x]

```
{{x → 0.5}}
```

Solve znajduje ścisłe rozwiązanie:

```
In:= Solve[rownanie, x]

Out:= {{x → 1/2}}
```

Reduce podaje wynik w innej, równoważnej postaci:

Reduce[rownanie, x]

```
x==1/2
```

FindInstance szuka zadanej liczby rozwiązań. Jeżeli zażądaliśmy znalezienia co najmniej dwóch rozwiązań, a w wyniku otrzymaliśmy pojedyncze rozwiązanie, możemy uznać, że *Mathematica* udowodniła, że istnieje tylko jedno rozwiązanie:

FindInstance[rownanie, x, 2]

```
{{x → 1/2}}
```

Proszę zwrócić uwagę na formę uzyskanej odpowiedzi, jak wiemy równoważnej $x = 1/2$ w notacji tradycyjnej.
Wynik jest podawany w postaci reguły transformacyjnej, za wyjątkiem **Reduce**. Wynikiem jest *lista* rozwiązań, w tym wypadku jednoelementowa, gdyż rozwiązanie jest jedno. **FindRoot** zawsze zwraca jeden wynik, i postać listy to uwzględnia. Pozostałe omawiane funkcje mogą zwracać wiele wyników, i dlatego odpowiedź jest listą dwukrotnie „zagnieżdżoną".
Aby wydobyć z takiej postaci tylko wartość rozwiązania (0.5), postępujemy następująco:

```
In:=    x /. {x → 0.5 }

Out:=    0.5
```

czyli podstawiamy w miejsce symbolu **x** wartość zgodnie z regułą transformacyjną **{x → 0.5 }** .

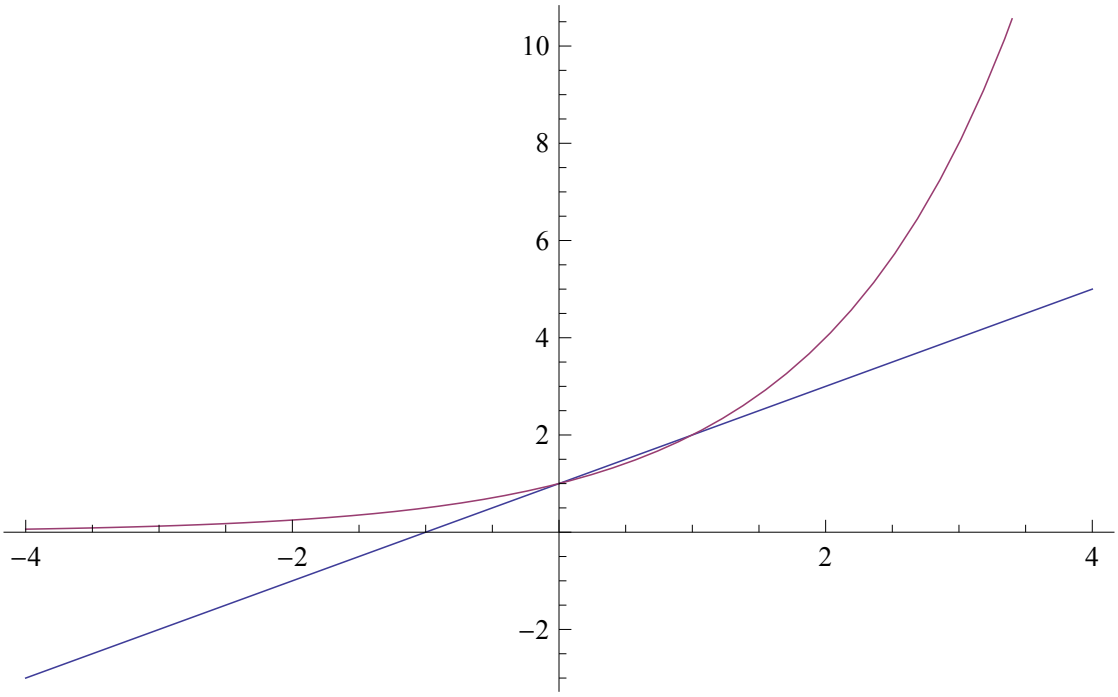
1.3 Rozwiąż równanie $x + 1 = 2^x$.

Rozwiąż w dziedzinie liczb rzeczywistych równanie, w notacji „tablicowej” podane jako:

$$x + 1 = 2^x \tag{3}$$

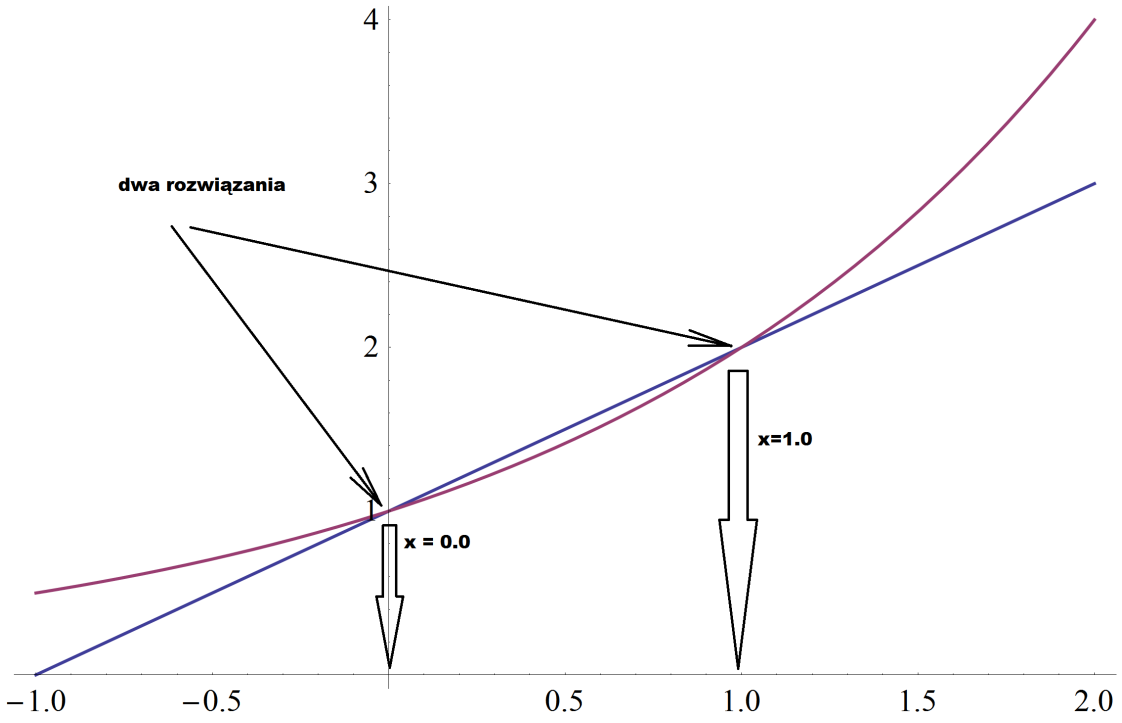
Ten przykład, wyglądający w pierwszym momencie na równie prosty jak poprzedni, zilustruje konieczność stosowania zalecanej kolejności rozwiązywania równań. Jest on nierozwiązywalny algebraicznymi metodami szkolnymi, pomimo istnienia „oczywistej” odpowiedzi. My, nauczeni doświadczeniem, zaczynamy od wykresu:

Plot[x+1,2^x,{x,-4,4}]



Wykres pokazuje , że prawdopodobnie są dwa rozwiązania. Aby to lepiej zobaczyć, zmieniam zakres x na $-1 < x < -2$:

Plot[{x+1,2^x},{x,-2,2}]



Z wykresu powyżej odczytujemy przybliżone rozwiązania:

$$x \simeq 0.0, \qquad x \simeq 1.0$$

Zgodnie z zalecanym schematem, zastosujemy teraz (dwukrotnie) **FindRoot**:

FindRoot[x+1==2^x, {x,0}]

```
{ x → 0. }
```

FindRoot[x+1==2^x, {x,1}]

```
{ x → 1. }
```

FindRoot potwierdza nasze przypuszczenie, że rozwiązaniem są liczby 0 i 1. Dopiero teraz możemy pokusić się o użycie **Solve**.

Solve[x+1==2^x, x]

```
Solve::ifun: Inverse functions are being used by Solve, so some solutions may not
be found; use Reduce for complete solution information. >>

{{x → 0 }, {x → (-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2] }}
```

Powyższy rezultat stanowi odpowiedź na pytanie po co rysować wykresy i szukać rozwiązania numerycznego zanim wywołamy **Solve**. Postawmy się na miejscu studenta, który od tego właśnie zaczął. Komunikat o błędzie, oraz niezrozumiała dla większości początkujących w tym temacieodpowiedź w drugim rozwiązaniu, zamiast spodziewanego x=1 to zniechęcający wynik. Na szczęście my już wiemy, że drugim rozwiązaniem powinno być 1, lub przynajmniej coś bardzo bliskiego 1. Sprawdzamy numeryczną wartość wyrażenia za pomocą ważnej funkcji **N** (argument zaznaczamy i kopiujemy myszką):

N[(-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2]]

1.

Wygląda na to, że powyższe wyrażenie jest równe 1. Przy standardowym wywołaniu polecenia **N** *Mathematica* używa koprocesora numerycznego w celu wyliczenia odpowiedzi, co jest na ogół najszybsze, i wystarczającej dokładności (16 cyfr po przecinku). Mamy jednak możliwość wykonania obliczeń o *dowolnie dużej dokładności* (chyba, że braknie pamięci). Np: obliczmy powyższe wyrażenie z dokładnością 128 cyfr po przecinku:

N[(-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2], 128]

```
1.0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Wyrażenie ewidentnie jest równe 1, z bardzo dużą dokładnością. Dlaczego więc *Mathematica* podaje tak dziwną odpowiedź? Otóż **Solve**, aby zaoszczędzić czas, nie dokonuje uproszczenia wyników. Możemy to zrobić sami za pomocą polecenia **Simplify** (uprość):

Simplify[(-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2]] -((Log[2]+ProductLog[-1,-(Log[2]/2)])/Log[2])

Użycie **Simplify** nic nie dało. Dla doświadczonego użytkownika jest to oczywiste. **Simplify** potrafi upraszczać wyłącznie wyrażenia zawierające funkcje elementarne (znane ze szkoły), jak dodawanie, mnożenie, niektóre pierwiastki (**Sqrt**) , logarytmy (**Log**), funkcje trygonometryczne (**Sin**, **Cos**, **Tan**) itd. Wyrażenie zawiera niestety funkcję specjalną **ProductLog**. Aby uprościć takie wyrażenie musimy zastosować „mocniejszą" (ale też wymagającą większej mocy obliczeniowej) wersję **Simplify**, czyli **FullSimplify**:

FullSimplify[(-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2]]

1

Wreszcie dostajemy 1. Jaki wynik da **Reduce**?

Reduce[x+1==2^x,x]

```
(C[1] ∈ Integers &&x == -1-ProductLog[C[1],-(Log[2]/2)]/Log[2])||x==0
```

Odpowiedź wygląda jeszcze inaczej. Operator || to LUB logiczne, natomiast && to I logiczne. **Integers** oznacza zbiór liczb całkowitych (**Reals** - rzeczywistych, **Rationals** - wymiernych, brak pojęcia liczby naturalnej). Symbol C[1] oznacza dowolną liczbę całkowitą. W notacji tablicowej zwykle używamy literki k lub n.

Zapis powyższy oznacza istnienie nieskończonej liczby rozwiązań numerowanej liczbą całkowitą C[1]. Jest to zaskakujące, bo dotąd spodziewaliśmy się dwóch rozwiązań: x=0 i x=1. Niestety (dla studentów pierwszego roku), *Mathematica* używa zwykle liczb zespolonych, o ile nie zażądamy inaczej. W dziedzinie liczb zespolonych funkcja 2^x jest okresowa, podobnie jak np. $\sin(x)$ dla liczb rzeczywistych. Aby znaleźć wyłącznie rozwiązania rzeczywiste, musimy to jawnie podać:

Reduce[x+1==2^x,x,Reals]

```
x==0||x==(-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2]
```

Powyższa uwaga dotyczy także **FindInstance**:

FindInstance[x+1==2^x, x, Reals, 10]

```
{x → 0},{x → (-Log[2]-ProductLog[-1,-(Log[2]/2)])/Log[2]}}
```

Ponieważ zażądaliśmy znalezienia 10 rozwiązań, a otrzymaliśmy 2, mamy pewność, że nie ma ich więcej.

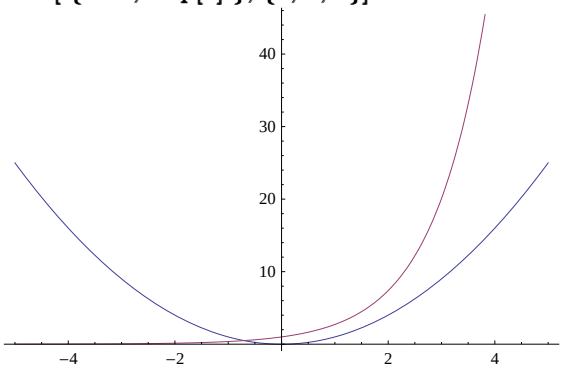
1.4 Rozwiąż $x^2 = e^x$ w domenie liczb rzeczywistych.

Rozwiąż równanie:

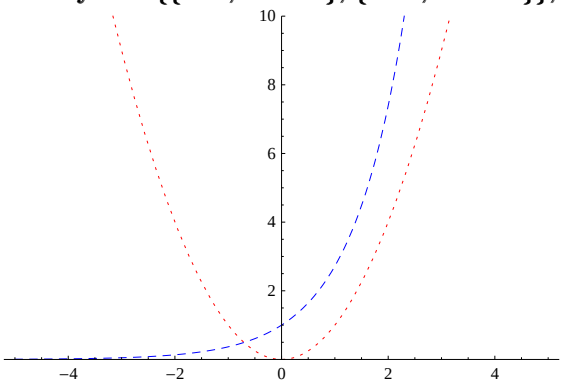
$$x^2 = e^x \tag{4}$$

w domenie liczb rzeczywistych.

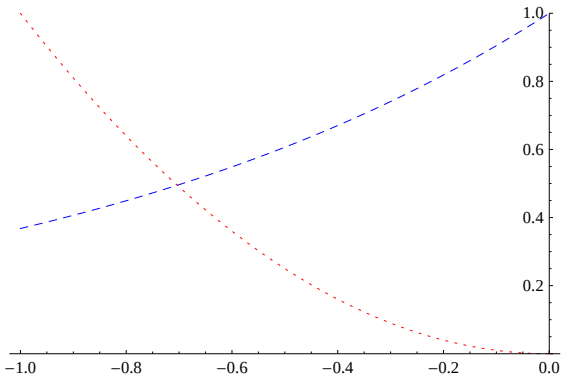
1. Wykonujemy wykres poleceniem:
   Plot[{ x^2, Exp[x] }, {x,-5,5 }]



2. Nie jest jasne który wykres jest który, program dobrał też (w mojej opini) źle skalę. Dlatego dodaję opcje:
   Plot[{ x^2, Exp[x] }, {x,-5,5 },
   PlotStyle → {{Red, Dotted}, {Blue, Dashed}}, PlotRange → {0, 10}]



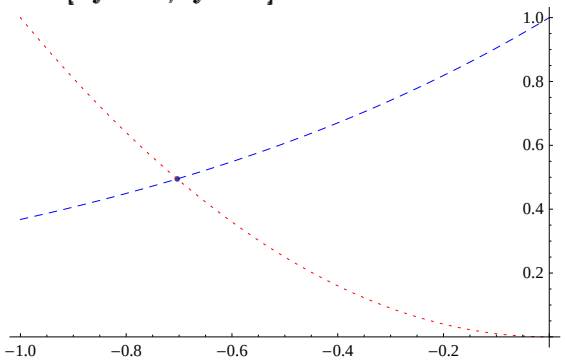
3. Widzę, że istnieje jedno rozwiązanie w przedziale {-1,0}. Powiększam wykres:
   Plot[{ x^2, Exp[x] }, {x,-1,0 },
   PlotStyle → {{Red, Dotted}, {Blue, Dashed}}, PlotRange → {0, 1}]



4. Widzę, że rozwiązanie jest dla $x \simeq -0.7$. Aby odczytać dokładniej powiększam myszką wykres, a następnie uruchamiam paletę z Menu→Graphics→Drawing Tools. Wybieram przycisk „Get Coordinates" (pierwszy od góry po prawej stronie, w kształcie „celownika") i naciskam na punkt przecięcia krzywych. Klawiszami Ctrl+C, Ctrl+V kopiuję współrzędne. Otrzymuję $x \simeq 0.705$ (lub bardzo podobny wynik).

5. Rozwiązuję równanie numerycznie, używając znalezionego graficznie przybliżenia:
   FindRoot[Exp[x]==x^2, {x,-0.705}]
   Otrzymuję:
   `{ x → -0.703467 }`
   Mathematica wyświetla standardowo 6 miejsc po przecinku. Wynik jest jednak podany dokładniej, co można sprawdzić kopiując do myszą, lub używając **FullForm[%]**. Wynik numeryczny to:
   $$x \simeq -0.7034674224983917$$
   Oczekujemy, że prawie wszystkie podane cyfry są poprawne.

6. Sprawdzam graficznie wynik:
   wykres1 = ListPlot[{{rozw, Exp[rozw]}}]
   wykres2 = Plot[{x^2, Exp[x]}, {x, -1, 0},
   PlotStyle → {{Red, Dotted}, {Blue, Dashed}}, PlotRange → {0, 1}]
   Show[wykres1,wykres2]



   Wszystko idealnie się zgadza!

7. Zrelaksowani, spróbujemy odpowiedzieć na pytanie, czy liczba $x \simeq -0.7034674224983917$ może być wyrażona przez jakieś znane nam lub *Mathematice* symbole i funkcje. Próbujemy **Solve**: **Solve[Exp[x] == x^2, x]**

i otrzymujemy:
{{x → -2 ProductLog[-(1/2)]}, {x → -2 ProductLog[1/2]}} Dostajemy 2 rozwiązania, wyrażone przez funkcję specjalną **ProductLog** Wiemy jednak, że istnieje jedno rozwiązanie rzeczywiste. Ale **Solve** zawsze szuka rozwiązań zespolonych, poza tym otrzymaliśmy kilka komunikatów z ostrzeżeniami. Najprostszym sposobem na przekonanie się, czy rozwiązania są zespolone, jest obliczenie ich wartości numerycznej:
N[%]
co daje:
{{x → 1.58805 - 1.54022 I }, {x → -0.703467}}
Jedno z rozwiązań okazuje się być rzeczywiste, o wartości numerycznej zgodnej z poprzednio znalezioną. Czyli:

$$x = -2W(1/2)$$

gdzie, W to tradycyjne oznaczenie (**TraditionalForm**) funkcji **ProductLog**, znanej także jako LambertW (stąd W).

8. Sprawdzamy rozwiązanie symboliczne:
Exp[x] == x^2 /. x → -2*ProductLog[1/2]
FullSimplify[%]
i otrzymujemy upragnione `True`, oznaczające, że równanie *jest spełnione*.

9. Jeżeli mamy na to czas, warto spróbować też innych poleceń:
Reduce[Exp[x] == x^2 , x]
`C[1] ∈ Integers && (x == -2 ProductLog[C[1], -(1/2)] || x == -2 ProductLog[C[1], 1/2])`
Okazuje się, że jest nieskończenie wiele rozwiązań zespolonych tego równania, dla dowolnej całkowitej liczby C[1]. Dla liczb rzeczywistych mamy:
Reduce[Exp[x] == x^2 , x, Reals]
`x == -2 ProductLog[1/2]`
Otrzymujemy jedno rozwiązanie rzeczywiste, zgodne z wynikiem **Solve**.

10. A co z **FindInstance**?
FindInstance[Exp[x] == x^2 , x, Reals]
{{x → -2 ProductLog[1/2]}}
Otrzymujemy znane już rozwiązanie. Jeżeli wywołamy polecenie w ten sposób: **FindInstance[Exp[x] == x^2 , x,** to żądamy znalezienia 20 rozwiązań. Otrzymujemy jedno. Oznacza to, że **FindInstance** potrafi *udowodnić*, że innych rozwiązań **nie ma**! W przeciwnym wypadku zostałoby wyświetlone odpowiednie ostrzeżenie: FindInstance::incs: Warning: FindInstance was unable to prove that the solution set found is complete. »

1.5 Podsumowanie : przypadek bez parametrów

Wykonane wyżej czynności są zdecydowanie nadmiarowe w porownaniu ze skalą trudności zadania. Nikt przy zdrowych zmysłach nie zrobiłby tego, np. na kolokwium z ręcznych rachunków. Pozwalamy sobie na to, ponieważ obliczenia wykonuje za nas komputer. W przypadku prostych, specjalnie spreparowanych zadań nie widać potrzeby dokonywania tak szczegółowej analizy. Jeżeli jednak mamy zmierzyć się z problemem nowym, o nieznanym rozwiązaniu, podejście takie będzie procentować. Ponownie podkreślam, że wykonanie tych czynności nic nas praktycznie nie kosztuje, zarówne z punktu widzenia trudności poszczególnych kroków, jak i poświęconego na nie czasu.

Równania tworzymy przy pomocy znaku podwójnej równości, == (polecenie Equal). Zalecana kolejność czynności przy rozwiązywaniu równań bez parametrów: Plot - FindRoot - [NSolve] - Solve - Reduce - FindInstance Podana niewiadoma, np: x, jest traktowana jako liczba zespolona ! Aby szukać tylko rozwiązań rzeczywistych metodami symbolicznymi (Reduce, FindInstance) należy dodać opcję Reals, oznaczającą zbór liczb rzeczywistych. Wyniki symboliczne należy obliczyć za pomocą polecenia N, sprawdzając czy nie są liczbami zespolonymi. Skomplikowane wyrażenia spróbować uprościć poleceniami Simplify (jeżeli składają się z funkcji elementarnych) oraz FullSimplify (jeżeli zawierają funkcje specjalne lub Simplify nie zadziałało).