Excel w nauce i technice. Receptury

Autor: David Bourg
Tłumaczenie: Zbigniew Waśko
ISBN: 83-246-0477-4
Tytuł oryginału: Excel Scientific and Engineering Cookbook
Format: B5, stron: 432

Excel to nie tylko arkusz kalkulacyjny – to potężne narzędzie obliczeniowe

Większość użytkowników komputerów kojarzy Excel z programem wykorzystywanym w biurach i urzędach do tworzenia zestawień, tabelek, wykresów i przeprowadzania obliczeń matematycznych. Tymczasem jego możliwości są o wiele większe. Excel, dzięki wbudowanym funkcjom, dodatkowym pakietom i językowi VBA może służyć jako narzędzie do wykonywania złożonych operacji obliczeniowych, przydatnych naukowcom i inżynierom. Dziś do przeprowadzenia zaawansowanych obliczeń i symulacji nie trzeba już wyspecjalizowanych stacji roboczych i trudnych w obsłudze aplikacji – wystarczy komputer i Excel.

Książka „Excel w nauce i technice. Receptury" to zbiór sposobów, które ułatwią odkrycie wszystkich możliwości obliczeniowych tego programu. Opisano w niej metody rozwiązywania różnych, nawet bardzo skomplikowanych zadań matematycznych, związanych z porządkowaniem danych i tworzeniem wykresów. Przedstawiono sposoby przeprowadzania analiz statystycznych i analiz ciągów czasowych, aproksymowania wykresów funkcji, obliczeń macierzowych, rozwiązywania równań liniowych, nieliniowych i różniczkowych oraz numerycznego różniczkowania i całkowania.

- Podstawy obsługi Excela
- Programowanie w języku VBA
- Importowanie danych do arkuszy
- Sortowanie i filtrowanie
- Obliczanie przedziałów ufności
- Analiza i predykcja ciągów czasowych
- Regresja liniowa
- Aproksymacja funkcji
- Rozwiązywanie równań i układów równań
- Całkowanie i różniczkowanie
- Optymalizacja
- Obliczenia finansowe

Dzięki tej książce przekonasz się, że Excel jest narzędziem przydatnym w każdym biurze projektowym i pracowni naukowej.

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 9.

Rozwiązywanie równań

9.0. Wprowadzenie

W tym rozdziale skoncentrujemy się na rozwiązywaniu równań za pomocą Excela. Rozwiązywanie równań w sensie ogólnym może polegać na znajdowaniu pierwiastków pojedynczych równań, wyznaczaniu wartości zmiennych niezależnych, przy których zmienna zależna przyjmuje zadaną wartość, lub rozwiązywaniu układów równań nieliniowych. Istnieje wiele tradycyjnych, ręcznych i komputerowych, metod rozwiązywania równań. Metoda Newtona iteracyjnego znajdowania pierwiastków równań nieliniowych czy metoda eliminacji Gaussa z podstawianiem wstecznym dla rozwiązywania układów równań liniowych to przykłady takich klasycznych metod. W swojej, zaliczanej już do klasyki, książce *Introduction to Numerical Analysis* Hildebrand przedstawił kilka klasycznych metod rozwiązywania równań. W książkach z serii *Numerical Recipes* możemy znaleźć algorytmy rozwiązywania równań zapisane w różnych językach programowania[1]. Algorytmy prezentowane w tych i innych książkach poświęconych metodom numerycznym są skuteczne i dobrze służą naukowcom oraz inżynierom. W tym rozdziale chciałbym jednak pokazać, jak łatwo można, wykorzystując możliwości Excela, rozwiązywać równania przy małej ilości programowania, a w niektórych przypadkach bez żadnego programowania poza tworzeniem formuł arkuszowych. Po tym, jak zobaczymy, co Excel ma nam do zaoferowania, jeśli Czytelnik nadal będzie chciał napisać program dla jakiejś klasycznej metody, pokażę implementację kilku takich metod przy użyciu języka VBA i Excela.

Niewiele można zyskać przez zastosowanie Excela lub innego tego typu narzędzia do rozwiązywania równań, które można z łatwością rozwiązać ręcznie, stosując proste przekształcenia algebraiczne. Dlatego w prezentowanych tu przykładach skoncentrujemy się niemal wyłącznie na równaniach nieliniowych.

Należy również dodać, że opisywane tutaj techniki mogą być stosowane nie tylko do równań mających postać wyrażeń matematycznych, mimo iż większość przykładowych równań jest tutaj prezentowana w takiej właśnie postaci. Możemy na przykład mieć do czynienia z „równaniem" złożonym z kilku formuł realizujących wyszukiwanie w tabelach lub podobnych, które można łatwo utworzyć za pomocą arkusza kalkulacyjnego. Taki arkusz może potem

[1] Patrz: F.B. Hildebrand, *Introduction to Numerical Analysis*, Dover Publications, 1974 oraz książki *Numerical Recipes* wydane przez Cambridge University Press, których autorami są: Press, Teukolsky, Vetterling i Flannery.

reprezentować równanie nieliniowe, które będziemy chcieli rozwiązać. Możemy mieć również do czynienia z obliczeniami rozłożonymi na wiele arkuszy. Ostatecznie, niezależnie od tego, czy mamy do czynienia z czysto matematycznym wyrażeniem, czy też ze skomplikowanym arkuszem, zawsze możemy wskazać zmienne niezależne i zmienne zależne. Rozwiązywanie tego typu równań polega zwykle na znajdowaniu takich wartości zmiennych niezależnych, przy których zmienne zależne przyjmują wartości zadane. W pewnych przypadkach zmienne zależne mogą być tylko pośrednio związane z rozwiązywanym równaniem, a wtedy możemy próbować znaleźć pierwiastek równania dla pewnej miary efektu (z taką sytuacją mamy do czynienia na przykład przy dopasowywaniu krzywej metodą najmniejszych kwadratów).

Ten typ obliczeń występuje również w zagadnieniach (lub częściowo pokrywa się z nimi) związanych z analizą „co-jeśli" oraz optymalizacją (patrz rozdział 13.) W obu tych przypadkach często zachodzi potrzeba stosowania tych samych technik iteracyjnych, jakich używa się do rozwiązywania równań nieliniowych.

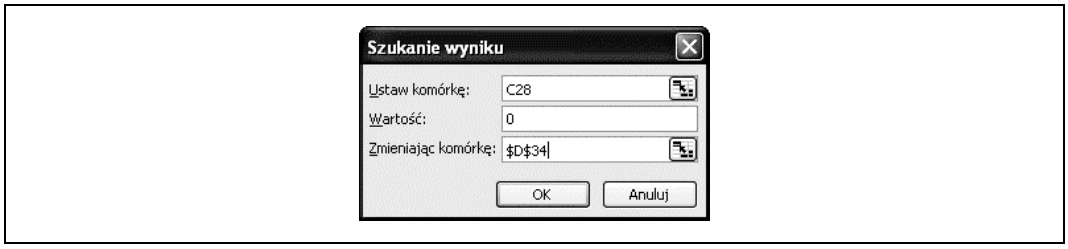
Ponieważ w niniejszym rozdziale bardzo często będziemy korzystać z takich narzędzi Excela, jak polecenie *Szukaj wyniku* i dodatek Solver, chciałbym je pokrótce przedstawić, pokazując ich możliwości oraz występujące między nimi różnice.

Zarówno polecenie *Szukaj wyniku*, jak i dodatek Solver są użytecznymi narzędziami umożliwiającymi przeprowadzanie obliczeń iteracyjnych bez konieczności pisania odpowiednich programów. Ja wykorzystuję je bardzo często do rozwiązywania różnych problemów, począwszy od rozwiązywania układów równań nieliniowych, przez analizę optymalizacyjną, po prognozowanie. Dodatek Solver wykorzystałem nawet do rozwiązania równań ruchu zbudowanego i testowanego przeze mnie wodolotu.

Z pozoru może wydawać się, że obydwa narzędzia robią to samo — pozwalają znaleźć w sposób iteracyjny wartość w komórce docelowej przez zmianę wartości w innej, powiązanej z nią komórce. Istnieją jednak między tymi narzędziami pewne różnice, o których należy pamiętać, decydując się na wybór jednego z nich w celu rozwiązania określonego problemu. Dalsza część niniejszego wprowadzenia zawiera opis tych różnic.

Polecenie Szukaj wyniku

Polecenie *Szukaj wyniku* jest bardzo łatwe w użyciu. Wystarczy po prostu podać docelową wartość dla komórki docelowej i określić niezależną komórkę, której zawartość będzie zmieniana, aby osiągnięta została zadana wartość docelowa. Na rysunku 9.1 pokazany jest interfejs tego narzędzia. Aby uzyskać do niego dostęp, należy z głównego menu wybrać polecenie *Narzędzia/Szukaj wyniku*.



Rysunek 9.1. Okno dialogowe polecenia Szukaj wyniku

Pole *Ustaw komórkę* zawiera odwołanie do komórki docelowej, której zawartość ma być ostatecznie równa wartości podanej w polu *Wartość*. Odwołanie do komórki, której zawartość ma być zmieniana, podajemy w polu *Zmieniając komórkę*. Komórka docelowa musi zawierać formułę, która z kolei musi bezpośrednio lub pośrednio odwoływać się do komórki podanej w polu *Zmieniając komórkę*.

Polecenie *Szukaj wyniku* może być wywołane również z podprogramu napisanego w języku VBA. Nie musimy więc robić tego ręcznie, co ma szczególne znaczenie w sytuacji, gdy polecenie to musi być wywoływane wielokrotnie. Przykład takiej sytuacji zawiera receptura 9.3.

Posługując się poleceniem *Szukaj wyniku*, możemy zmieniać zawartość tylko jednej komórki, co oznacza, że możemy go stosować tylko w przypadkach z jedną zmienną niezależną. Ponadto musimy podać wartość docelową w sposób jawny, tzn. w postaci konkretnej liczby. Polecenie *Szukaj wyniku* nie pozwala nam również nakładać żadnych ograniczeń na zawartość komórki niezależnej (tzn. tej, którą podajemy w polu *Zmieniając komórkę*).

Zgodnie z tym, co można przeczytać w krótkim artykule na stronie pomocy technicznej Microsoftu (artykuł nr 100782), polecenie *Szukaj wyniku* wykorzystuje algorytm przeszukiwania liniowego, przyjmując wartości początkowe zbliżone do wartości podanej w komórce niezależnej. Oznacza to, że wartość znajdująca się w komórce niezależnej w chwili uruchomienia polecenia *Szukaj wyniku* służy jako wartość początkowa mająca charakter prognozy wyniku. Należy o tym pamiętać szczególnie podczas rozwiązywania problemów, które mają więcej niż jedno rozwiązanie. Na przykład, jeśli próbujemy znaleźć pierwiastki równania trzeciego stopnia, podana przez nas wartość początkowa może prowadzić do jednego rozwiązania, podczas gdy inna wartość początkowa może dać inne rozwiązanie. Wynika stąd również, że jeśli polecenie *Szukaj wyniku* nie znajduje żadnego rozwiązania, wówczas należy spróbować zastosować inną wartość początkową.

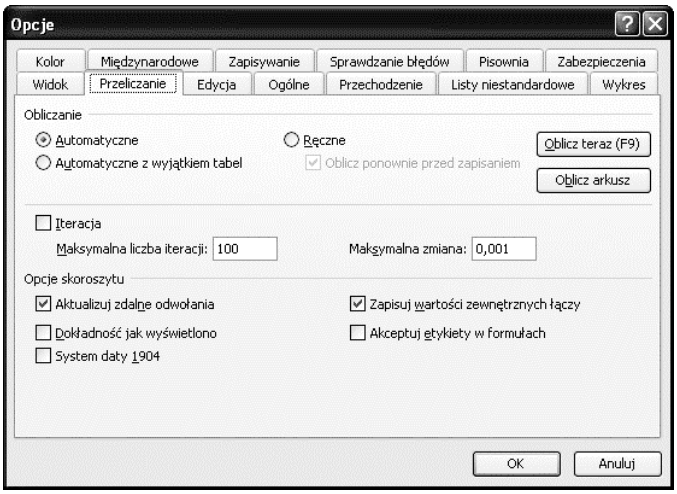
Polecenie *Szukaj wyniku* kończy obliczenia, gdy maksymalna wielkość zmian między iteracjami spada poniżej określonego progu. Przy ustawieniach domyślnych próg ten wynosi 0,001. Obliczenia zostaną przerwane również wtedy, gdy liczba iteracji przekroczy ustaloną wartość (domyślnie 100). Kryteria te możemy zmienić, wybierając z głównego menu polecenie *Narzędzia/Opcje*, które otwiera okno dialogowe *Opcje* pokazane na rysunku 9.2.

W oknie tym należy otworzyć zakładkę *Przeliczanie*. W jej środkowej części zobaczymy pole opcji *Iteracja* oraz dwa pola edycyjne *Maksymalna liczba iteracji* i *Maksymalna zmiana*. W polach tych należy wpisać odpowiednie wartości i zaznaczyć opcję *Iteracja*, aby nowe kryteria zaczęły obowiązywać.

W przypadkach prostych problemów iteracyjnych polecenie *Szukaj wyniku* sprawdza się bardzo dobrze i jest łatwe w użyciu. Jednak gdy mamy do czynienia z bardziej złożonym problemem lub potrzebujemy większej kontroli nad procesem iteracyjnym, należy sięgnąć po bardziej wyrafinowane narzędzie, jakim jest dodatek Solver.

Dodatek Solver

Dodatek Solver jest podobny w swoim działaniu do polecenia *Szukaj wyniku* — również pozwala znaleźć wartość docelową w komórce docelowej przez iteracyjną zmianę wartości w komórce niezależnej. Jednak jego możliwości są o wiele większe, co znajduje odzwierciedlenie w bardziej rozbudowanym interfejsie (rysunek 9.3). Dodatek Solver jest dostępny w menu



Rysunek 9.2. Okno dialogowe Opcje

Narzędzia. Jeśli nie ma go wśród opcji tego menu, należy wybrać polecenie *Narzędzia/Dodatki* i na liście dostępnych dodatków zaznaczyć pozycję *Dodatek Solver*. Po wykonaniu tych czynności można go uruchomić, wybierając polecenie *Narzędzia/Solver*.

Rysunek 9.3. Okno dialogowe Solver - Parametry

Dodatek Solver możemy wykorzystać do rozwiązywania problemów o wielu zmiennych. W takich przypadkach wartość docelowa w komórce docelowej będzie uzyskiwana przez zmianę wartości w kilku komórkach niezależnych. Komórki, których zawartość będzie zmieniana, podajemy w polu *Komórki zmieniane*. Możemy przy tym zastosować dowolny styl odwołań oraz odwołania do wielu komórek oddzielone średnikami. Wersja Solvera dołączana do Excela pozwala na podanie maksymalnie 200 komórek niezależnych.

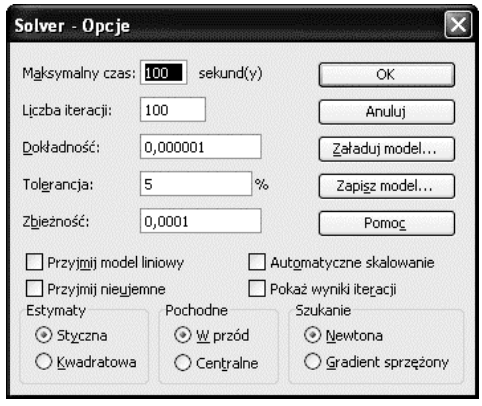
Dodatek Solver nie wymaga nawet podawania wartości docelowej. Zamiast tego możemy wybrać opcję minimalizowania lub maksymalizowania tej wartości. Odwołanie do komórki docelowej umieszczamy w polu *Komórka celu* (musi to być pojedyncza komórka zawierająca formułę). Przyciski wyboru opcji *Równa* pozwalają zdecydować, czy zawartość komórki docelowej ma być minimalizowana, maksymalizowana lub ustalona przez przypisanie jej konkretnej wartości liczbowej.

Klikając przycisk *Odgadnij*, możemy pozwolić Solverowi „odgadnąć", które komórki powinien uznać za niezależne dla danego problemu. Ja jednak unikam korzystania z tej możliwości. Sprowadza się ona do wybrania wszystkich komórek, do których odwołuje się formuła w komórce docelowej i które same nie zawierają formuł. W przypadku prostej formuły z niewielką liczbą zmiennych niezależnych, z których wszystkie chcemy zmieniać, takie rozwiązanie może być przydatne. Stwierdziłem jednak, że w większości przypadków, z jakimi miałem do czynienia, wybranych zostało zbyt dużo komórek — nawet te, których nie zamierzałem w ogóle zmieniać (bo zawierały np. stałe lub inne wartości, które z pewnych względów powinny pozostać niezmienne). Z drugiej strony, jeżeli mamy do czynienia z dużą liczbą zmiennych, ułatwieniem może być zlecenie Solverowi wybrania ich wszystkich, a następnie ręczne usunięcie z pola *Komórki zmieniane* tych, których nie chcemy zmieniać.

To jeszcze nie wszystkie możliwości Solvera. Na komórki będące częścią rozwiązywanego problemu możemy nakładać określone ograniczenia. Mogą one mieć charakter równości lub granic górnych i dolnych. Jest to przydatne, gdy interesuje nas rozwiązanie z określonego obszaru lub gdy przeprowadzamy proces optymalizacji z ograniczeniami. Receptura 9.4 zawiera przykład wykorzystania tych ograniczeń przy rozwiązywaniu układu równań za pomocą Solvera. Z kolei rozdział 13. zawiera receptury z przykładami zastosowania Solvera do przeprowadzania optymalizacji z ograniczeniami. Dodatek Solver (w wersji dołączanej do Excela) pozwala utworzyć maksymalnie 100 ograniczeń dla zagadnień nieliniowych i 200 dla zagadnień liniowych.

Solver korzysta również z większej liczby algorytmów znajdowania rozwiązań niż polecenie *Szukaj wyniku*. Do rozwiązywania problemów liniowych wykorzystuje **metodę sympleks**. Solver może także rozwiązywać problemy nieliniowe, wykorzystując **metodę uogólnionego gradientu zredukowanego**. W przypadku, gdy rozwiązywany problem wymaga zmiennych o wartościach całkowitych, Solver korzysta także z algorytmu **podziałów i ograniczeń** (ang. *branch and bound*).

Niewielką kontrolę nad tymi algorytmami umożliwia okno dialogowe *Solver - Opcje*. Aby otworzyć to okno (pokazane na rysunku 9.4), należy kliknąć przycisk *Opcje* w oknie dialogowym *Solver - Parametry*.



Rysunek 9.4. Okno dialogowe Solver - Opcje

Jak widać, jest tu znacznie więcej opcji w porównaniu z tylko dwiema dostępnymi dla polecenia *Szukaj wyniku*. Opcje Solvera to:

Maksymalny czas
: Opcja ta reprezentuje maksymalny czas, jaki Solver może przeznaczyć na poszukiwanie wyniku. Po upływie tego czasu proces zostanie przerwany, a Solver wyświetli okno z informacją, że nie mógł znaleźć rozwiązania w wyznaczonym czasie. Wartość tę możemy ustalić maksymalnie na 32 767 sekund (nieco ponad 9 godzin), ale chyba nigdy nie zachodzi potrzeba ustawiania aż tak długiego czasu.

Liczba iteracji
: Pozwala określić maksymalną liczbę iteracji, jakie Solver może wykonać w poszukiwaniu wyniku. Jeżeli ta liczba zostanie osiągnięta, zanim Solver znajdzie rozwiązanie, proces zostanie przerwany, a my zostaniemy poinformowani, że rozwiązanie nie zostało znalezione przy zadanej liczbie iteracji. Wartość tę możemy ustalić maksymalnie na 32 767, chociaż w mojej praktyce nigdy nie musiałem stosować wartości większych niż 1000, a najczęściej pozostawiam wartość domyślną, czyli 100.

Dokładność
: Jest to liczba dziesiętna z przedziału od 0 do 1, a jej wartość domyślna wynosi $1{,}0\times10^{-6}$. Solver wykorzystuje ją do oceny, czy nałożone ograniczenia są spełnione. Im mniejsza jest ta liczba, tym większa precyzja. W praktyce bardzo rzadko zmieniam tę wartość.

Tolerancja
: Tolerancja jest używana do określenia, czy ograniczenie wykorzystujące wartości całkowite jest spełnione. Wyrażana jest w procentach, a jej wartość domyślna wynosi 5%. Jeśli nie stosujemy ograniczeń z wartościami całkowitymi, tolerancja nie jest używana.

Zbieżność
: Ten parametr jest używany do określenia, czy Solver znalazł rozwiązanie. Jeśli dla pięciu ostatnich iteracji zmiana wartości w komórce docelowej nie przekracza wartości podanej w polu *Zbieżność*, Solver uznaje, że rozwiązanie zostało znalezione, i wyświetla okno ze stosowną informacją. Obliczenia będą kontynuowane do momentu osiągnięcia maksymalnej liczby iteracji lub maksymalnego czasu. Jeżeli zostanie napotkane którekolwiek z zadanych ograniczeń, Solver wyświetli komunikat z informacją, że nie może odnaleźć rozwiązania.

Przyjmij model liniowy
: Jeżeli wiemy, że model rozwiązywanego problemu ma charakter liniowy, możemy zalecić Solverowi zastosowanie metody sympleks zamiast algorytmu uogólnionego gradientu zredukowanego. Gdy zaznaczymy tę opcję, Solver przeprowadzi kilka testów sprawdzających, czy według jego kryteriów nasz model jest rzeczywiście liniowy. W przypadku negatywnego wyniku tych testów zostanie wyświetlony komunikat z ostrzeżeniem, że model nie spełnia kryteriów liniowości.

Przyjmij nieujemne
: Zaznaczenie tej opcji oznacza automatyczne nałożenie ograniczeń w postaci dolnej granicy wartości dla wszystkich komórek zmienianych. Jeśli wiemy, że wszystkie zmienne powinny być zawsze nieujemne, i chcemy, aby rzeczywiście nie przyjmowały wartości ujemnych, wówczas powinniśmy zaznaczyć tę opcję. W ten sposób możemy uniknąć ręcznego nakładania ograniczeń na każdą zmienną.

Automatyczne skalowanie

W niektórych zagadnieniach wartości zmiennych niezależnych mogą znacznie różnić się pod względem wielkości od wartości zmiennych zależnych. W takich przypadkach dobrze jest tak przeskalować model, aby wartości wejściowe i wyjściowe były tego samego rzędu. Jeżeli wcześniej nie wykonaliśmy takiego skalowania, możemy zaznaczyć tę opcję, aby Solver zrobił to za nas. Wartości wejściowe i wyjściowe zostaną wówczas przeskalowane w wyniku podzielenia ich przez wartości początkowe podane w komórkach zmienianych i docelowych. W takich sytuacjach szczególnej wagi nabiera dobór odpowiednich wartości początkowych. Zawsze najlepiej jest ustalać te wartości zgodnie ze zdrowym rozsądkiem i realiami rozważanego problemu.

Pokaż wyniki iteracji

Jeśli chcemy na bieżąco śledzić pracę Solvera podczas rozwiązywania danego problemu, możemy tę opcję zaznaczyć. Pojawi się wówczas okno dialogowe informujące nas o tym, że Solver przerwał pracę i że aktualne wyniki iteracji są wyświetlane w aktywnym arkuszu. Opcja ta przydaje się, gdy chcemy prześledzić wyniki poszczególnych etapów iteracji, na przykład wtedy, gdy chcemy ustalić, dlaczego Solver nie może znaleźć rozwiązania. W zwykłych warunkach nie zaznaczam tej opcji, ponieważ taki tryb pracy wymaga za każdym razem kliknięcia odpowiedniego przycisku nakazującego Solverowi przejście do następnego etapu. W sytuacji, gdy rozwiązanie danego problemu wymaga dużej liczby iteracji, może to być bardzo żmudne i czasochłonne.

Estymaty

Tutaj możemy określić sposób, w jaki Solver będzie szacować wartości początkowe zmiennych niezależnych. Opcja *Styczna* oznacza wykorzystanie do tego celu interpolacji liniowej, a opcja *Kwadratowa* — interpolacji kwadratowej, dającej lepsze rezultaty (szybsze znajdowanie rozwiązania) w przypadku problemów nieliniowych. Szczerze mówiąc, przy szybkości współczesnych procesorów trudno jest zauważyć istotną różnicę między tymi opcjami, jeśli chodzi o czas znajdowania rozwiązania typowych problemów.

Pochodne

Do obliczania gradientów Solver wykorzystuje metodę różnic skończonych, dając nam możliwość wyboru jednego z dwóch schematów tej metody. Opcja *W przód* pozwala wybrać schemat różnicy przedniej (progresywnej), a opcja *Centralne* — schemat różnicy centralnej. Różniczkowanie w oparciu o schemat różnicy centralnej wymaga większej ilości obliczeń, ale jest bardziej dokładne. I znów, jeśli chodzi o czas trwania obliczeń, różnica między tymi opcjami jest praktycznie niezauważalna na współczesnych komputerach. Ja najczęściej wybieram różniczkowanie centralne. (Analizę różnic między różniczką przednią a różniczką centralną zawiera receptura 10.6).

Szukanie

Tutaj możemy nakazać Solverowi rozwiązywanie problemu metodą Newtona lub metodą sprzężonego gradientu. Solver używa tych metod do określania kierunku poszukiwań podczas każdej iteracji. Metoda Newtona wymaga mniejszej liczby obliczeń niż metoda sprzężonego gradientu, ale za to angażuje więcej zasobów pamięciowych. Jeśli pamięć jest kwestią istotną — na przykład, gdy rozwiązujemy rozbudowany problem z dużą liczbą zmiennych i ograniczeń lub gdy zasoby pamięciowe naszego komputera są ograniczone — wówczas możemy wybrać metodę sprzężonego gradientu.

Istnieją jeszcze dwie inne opcje Solvera, które, mimo że nie wpływają bezpośrednio na proces obliczeniowy, są bardzo użyteczne. Opcjami tymi są: *Załaduj model* i *Zapisz model*. Ujmując rzecz krótko: Solver pozwala nam zapisywać modele i ponownie je wykorzystywać. To może być przydatne podczas sprawdzania różnych kombinacji zmiennych, ograniczeń i opcji Solvera pod kątem najlepszego dopasowania modelu do danego zagadnienia. Daje to również możliwość odtworzenia modelu w dowolnym momencie (bez konieczności pamiętania wszystkich ustawień), jeśli zdecydujemy się na ponowne wykonanie tych samych obliczeń.

Aby zapisać model, należy kliknąć przycisk *Zapisz model*. Pojawi się wówczas małe okno dialogowe wzywające do określenia zakresu komórek, w których dane modelu mają być zapisane. Zakres ten powinien być położony z dala od innych danych i formuł znajdujących się w arkuszu, aby nie zostały one zastąpione danymi zapisywanego modelu. Do określenia tego zakresu wystarczy podanie jednej komórki — Solver umieści dane modelu w kolumnie, rozpoczynając od podanej komórki i zajmując trzy komórki plus jeszcze tyle, ile jest ograniczeń w zapisywanym modelu.

Aby później odtworzyć zapisany model, wystarczy otworzyć okno *Solver - Opcje* i kliknąć przycisk *Załaduj model*. Zobaczymy wówczas małe okno dialogowe wzywające nas do zaznaczenia zakresu komórek, w których model został zapisany. Tym razem musimy zaznaczyć ten zakres w całości — zaznaczenie tylko pierwszej komórki nie wystarczy.

Jeszcze jednym przydatnym elementem Solvera — jakiego nie ma w przypadku polecenia *Szukaj wyniku* — jest okno dialogowe *Solver - Wyniki*. Po znalezieniu rozwiązania Solver informuje nas o tym, otwierając okno dialogowe, takie jak to pokazane na rysunku 9.5.

Rysunek 9.5. Okno dialogowe Solver - Wyniki

Mamy tutaj do wyboru zachowanie wyniku uzyskanego przez Solver lub przywrócenie oryginalnych wartości początkowych. Ponadto możemy wydać Solverowi polecenie wygenerowania jednego lub kilku spośród trzech raportów: *Wyników*, *Wrażliwości* i *Granic*. W tym celu należy kliknąć odpowiednią nazwę raportu. Możemy wybrać jeden, wszystkie lub dowolną kombinację dwóch raportów. Po kliknięciu przycisku *OK* Solver utworzy nowy arkusz dla każdego zaznaczonego raportu. Raporty te przydają się podczas interpretowania uzyskanych wyników, szczególnie w przypadku optymalizacji z ograniczeniami. Zagadnienia związane z tymi raportami i optymalizacją są szerzej opisane w rozdziale 13.

Zawsze możemy skorzystać z pomocy, jaką dodatek Solver oferuje nam po kliknięciu przycisku *Pomoc* dostępnego we wszystkich jego oknach dialogowych. Czytelnikom pragnącym poszerzyć swoją wiedzę na temat historii i wewnętrznych mechanizmów Solvera polecam artykuł pt. *Design and Use of the Microsoft Excel Solver*, autorstwa Daniela Fylstry, Leona Lasdona, Johna Watsona i Allana Warena, opublikowany na łamach pisma „Interfaces", nr 5 (1998).

9.1. Rozwiązywanie równań metodą graficzną

Problem

Chcemy graficznie wyznaczyć pierwiastki równania.

Rozwiązanie

Należy zapisać równanie w postaci $y = g(x)$, obliczyć wartości zmiennej y dla zadanego zakresu wartości zmiennej x i, wykorzystując możliwości Excela, utworzyć wykres tych wyników. Informacje na temat tworzenia wykresów w Excelu można znaleźć w rozdziale 4.

Analiza

Jeśli musimy znaleźć pierwiastki równania, którego nie możemy rozwiązać ręcznie, powinniśmy rozpocząć od wykonania wykresu dla tego równania. Uzyskamy w ten sposób cenną informację na temat natury samego równania i położenia jego pierwiastków. Jeśli tylko możemy przedstawić równanie w postaci $y = g(x)$, gdzie x oznacza zmienną niezależną, a y — zmienną zależną, wówczas z łatwością możemy zestawić kolumny z obliczeniami zmiennej y dla różnych wartości zmiennej x.

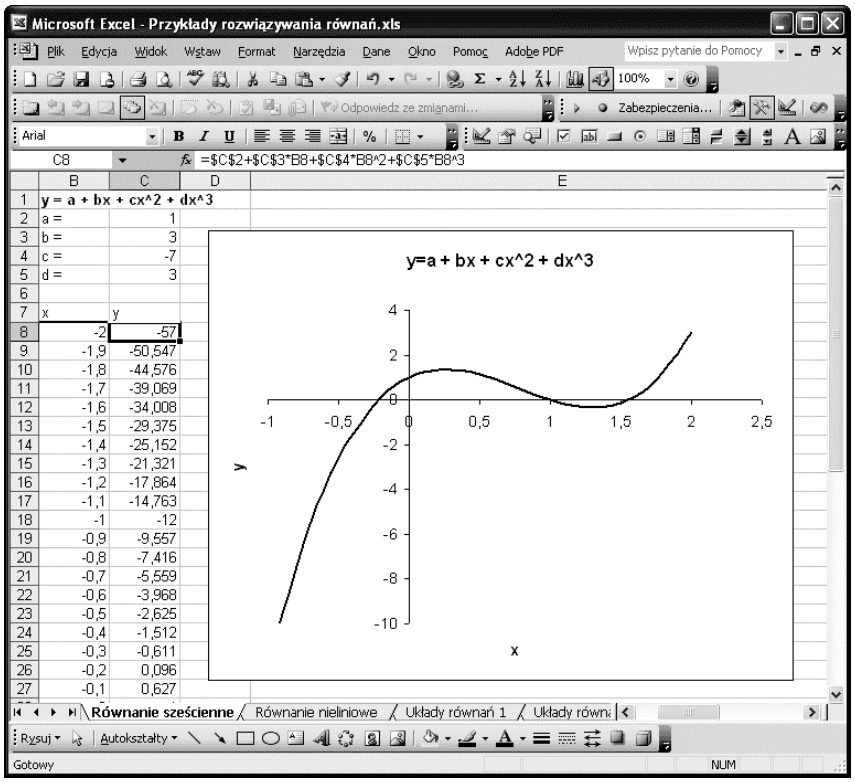
To wszystko wydaje się bardzo proste i prawdę mówiąc, w kategoriach operacji arkuszowych, nie jest to trudne do wykonania. Jednak gdy mamy do czynienia z jakimś szczególnie zawikłanym równaniem, które na dodatek może mieć kilka pierwiastków, lepiej byłoby dysponować wiedzą o położeniu tych pierwiastków. Wynika to stąd, że większość iteracyjnych metod rozwiązywania równań wymaga od nas podania przypuszczalnego położenia tych pierwiastków. Niektóre metody wymagają podania dwóch wartości początkowych, a inne — określenia przedziałów, w których pierwiastki mogą się znajdować. Sukces w wyznaczeniu pierwiastków za pomocą każdej z tych metod zależy więc od jakości naszych przewidywań i ustalanych na tej podstawie wartości początkowych. (Konkretne przykłady można znaleźć w pozostałych recepturach z tego rozdziału).

Rozważmy następujące równanie wielomianowe trzeciego stopnia:

$$y = a + bx + cx^2 + dx^3$$

To równanie ma oczywiście trzy pierwiastki, co oznacza, że istnieją trzy wartości zmiennej x, dla których zmienna y przyjmuje wartość równą zero. Bez trudu można to wykazać, wykreślając krzywą zależności y od x. Na rysunku 9.6 został przedstawiony wykres powyższego równania wykonany w Excelu.

Jak widać, rzeczywiście krzywa przecina oś x w trzech punktach, a to dowodzi, że dla trzech wartości zmiennej x zmienna y przyjmuje wartość równą zero. Aby wykreślić tę krzywą, zestawiłem dwie kolumny: jedna zawiera wartości zmiennej x, a druga — obliczone wartości zmiennej y. Na rysunku 9.6 są to kolumny, odpowiednio: *B* i *C*. W tym przykładzie przyjąłem dowolne wartości współczynników wielomianu i umieściłem je w kolumnach od *C2* do *C5*. Na pasku formuły można zobaczyć formułę, jakiej użyłem do obliczania wartości y



Rysunek 9.6. Wykres wielomianu trzeciego stopnia

odpowiadających poszczególnym wartościom zmiennej *x*. (Na zrzucie ekranu zaznaczona jest komórka *C8* i na pasku formuły widoczna jest formuła zawarta w tej komórce). Formuła ta ma następującą postać: `=$C$2+$C$3*B8+$C$4*B8^2+$C$5*B8^3`.

Teraz widzimy wyraźnie, jak zachowuje się nasze równanie i gdzie leżą jego pierwiastki. Wyznaczenie dokładnych wartości zmiennej *x*, dla których $y = 0$ możemy zrealizować na kilka sposobów. Jeden z nich polega na aproksymowaniu wartości zmiennej *x* przez interpolację wartości w kolumnach *B* i *C*. W tym celu należy odszukać takie wartości zmiennej *x*, dla których zmienna *y* zmienia swój znak, a następnie należy dokonać interpolacji między tymi wartościami. Inny sposób wyznaczenia miejsc zerowych może polegać na wykorzystaniu technik iteracyjnych. Przykłady takich technik zawarte są w recepturach 9.2 i 9.3. Z kolei receptura 9.6 pokazuje, jak wyznaczyć pierwiastki opisywanego tu równania trzeciego stopnia, stosując metodę Newtona i metodę siecznych.

Jednak co robić w sytuacji, gdy mamy do czynienia z równaniem nie dającym się zapisać w postaci $y = g(x)$? Podczas próby wykonania wykresu takiego równania w opisany wyżej sposób napotkamy oczywiście problemy. Na szczęście Excel oferuje narzędzia, które możemy wykorzystać, aby ominąć te trudności. Wyjaśnienie tego, co mam na myśli, znajduje się w recepturze 9.2.

9.2. Iteracyjne rozwiązywanie równań nieliniowych

Problem

Użytkownik chciałby rozwiązać w Excelu równanie nieliniowe metodą iteracyjną, ale nie wie, jak się do tego zabrać.

Rozwiązanie

Należy wykorzystać polecenie *Szukaj wyniku* lub dodatek Solver. Informacje na temat różnic w korzystaniu z tych narzędzi oraz zalet i wad każdego z nich zawarte zostały we wprowadzeniu do niniejszego rozdziału.

Analiza

W tej recepturze chcę pokazać na konkretnym przykładzie, jak rozwiązać równanie nieliniowe za pomocą takich narzędzi jak polecenie *Szukaj wyniku* i dodatek Solver. Nasze rozważania skoncentrujemy na następującym równaniu:

$$\frac{0{,}242}{\sqrt{C_f}} = \log(R_N C_f)$$

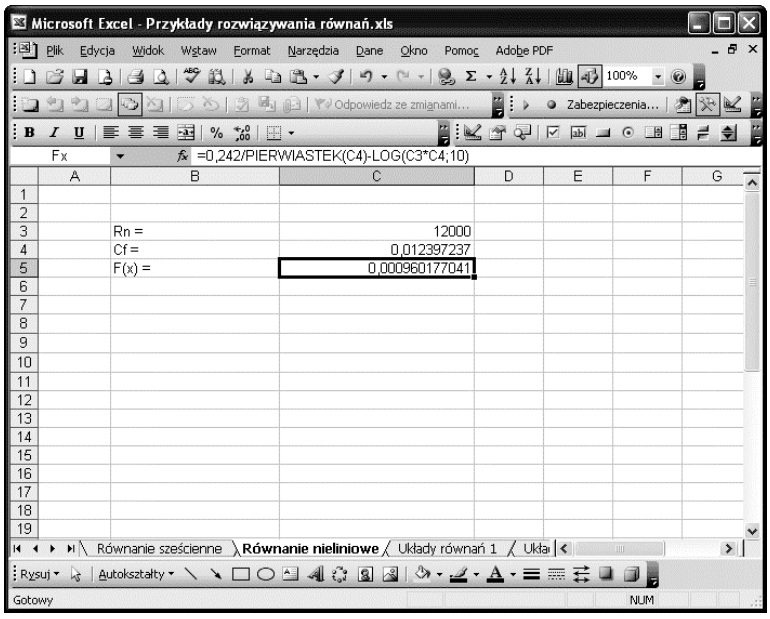
Równanie to jest wykorzystywane w obliczeniach oporów ruchu statków do szacowania wartości współczynnika oporu tarcia C_f jako funkcji **liczby Reynoldsa** R_N. Istnieją jeszcze inne równania służące do tego samego celu, ale wybrałem ten klasyczny przypadek, ponieważ nie można go zapisać w postaci $y = g(x)$. Musimy więc sięgnąć po metody iteracyjne, aby obliczyć wartość C_f odpowiadającą danej wartości R_N.

Pokażę teraz, jak wykorzystać polecenie *Szukaj wyniku* i dodatek Solver do rozwiązania tego problemu. W obu przypadkach musimy najpierw przekształcić to równanie do następującej postaci:

$$0 = \frac{0{,}242}{\sqrt{C_f}} - \log(R_N C_f)$$

Teraz możemy zastosować jedno z wymienionych wyżej narzędzi do iteracyjnego poszukiwania takiej wartości C_f, która przy zadanej wartości R_N sprawi, że prawa strona tego równania będzie równa 0. Wcześniej jednak musimy prawą stronę tego równania umieścić w arkuszu w postaci formuły, tak jak zostało to pokazane na rysunku 9.7.

Formuła reprezentująca prawą stronę naszego równania znajduje się w komórce *C5*. Postać tej formuły jest następująca: `=0,242/PIERWIASTEK(C4)-LOG(C3*C4;10)`, co zresztą widać na pasku formuły na rysunku 9.7. Komórka *C3* zawiera zadaną liczbę Reynoldsa, a komórka *C4* — współczynnik tarcia wyznaczony za pomocą Solvera lub polecenia *Szukaj wyniku*. Aby skorzystać z tych narzędzi, musimy w komórce *C4* umieścić wartość początkową, którą powinna być przewidywana przez nas wartość współczynnika tarcia. Jak już wskazywałem we wstępie do niniejszego rozdziału, taka wartość początkowa powinna być dobierana bardzo starannie.



Rysunek 9.7. Przykład równania nieliniowego

Wyznaczanie C_f za pomocą polecenia Szukaj wyniku

Aby znaleźć rozwiązanie za pomocą tego narzędzia, należy z głównego menu wybrać polecenie *Narzędzia/Szukaj wyniku*. Zostanie okno dialogowe *Szukanie wyniku* pokazane na rysunku 9.8.

Rysunek 9.8. Okno dialogowe Szukanie wyniku dla przykładowego równania nieliniowego

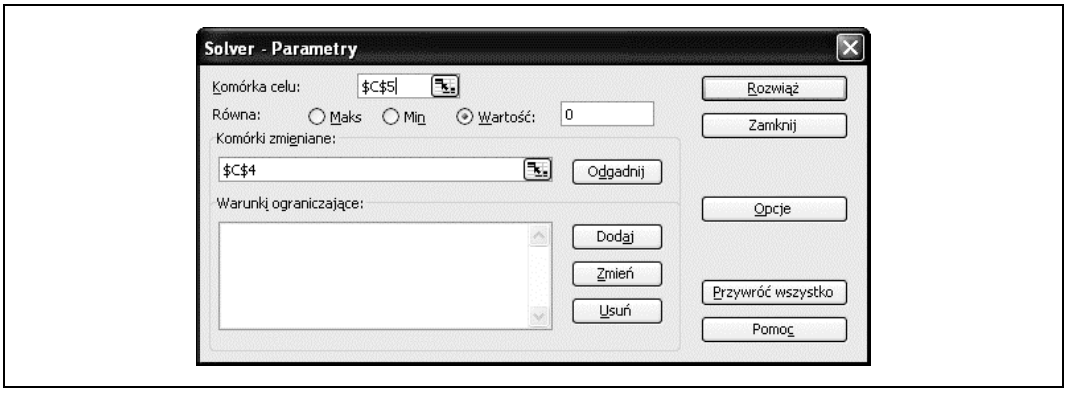
W polu *Ustaw komórkę* należy wpisać `C5` lub kliknąć małą ikonę na prawo od pola edycyjnego, aby tymczasowo przejść do arkusza i zaznaczyć tam komórkę *C5* (po jej zaznaczeniu należy wcisnąć klawisz *Enter*, aby powrócić do okna *Szukanie wyniku*). Przypominam, że komórka *C5* zawiera formułę reprezentującą prawą stronę naszego równania i chcemy, aby wynik tej formuły był równy zero. Dlatego w polu *Wartość* należy wpisać `0`. Jest to nasza wartość docelowa. Teraz w polu *Zmieniając komórkę* należy umieścić odwołanie do komórki *C4*, która zawiera wartość współczynnika tarcia. Po kliknięciu przycisku *OK* zawartość komórki *C4* będzie dotąd zmieniana, aż formuła w komórce *C5* osiągnie wartość dostatecznie bliską 0. Wynik tych obliczeń jest pokazany na rysunku 9.7. Widać tu, że reszta w komórce *C5* wynosi $9{,}6\times10^{-4}$.

Powinienem dodać, że przyjęta przeze mnie wartość początkowa dla komórki *C4* wynosiła 1×10^{-7}. Mogłem przyjąć wartość 0, ponieważ wiem, że współczynnik tarcia powinien być bardzo małą liczbą, ale tego nie zrobiłem, bo to wywołałoby błąd dzielenia przez zero. Wystąpienie takiego błędu podczas pracy polecenia *Szukaj wyniku* powoduje awaryjne zakończenie jego działania i wyświetlenie komunikatu o następującej treści: *Formuła w komórce musi dawać w wyniku liczbę*.

Reszta wynosząca $9,6\times10^{-4}$ nie oznacza złego wyniku, ale możemy uczynić go jeszcze lepszym. Jak wspominałem we wprowadzeniu, możemy zmienić ustawienia zbieżności dla polecenia *Szukaj wyniku* w oknie dialogowym *Opcje* (rysunek 9.2). Aby go otworzyć, należy z głównego menu wybrać polecenie *Narzędzia/Opcje*. Na zakładce *Przeliczanie* zaznaczyłem opcję *Iteracja* i ustawiłem parametr *Maksymalna liczba iteracji* na 5 000, a parametr *Maksymalna zmiana* na 1×10^{-8}. Przy takich ustawieniach polecenie *Szukaj wyniku* znalazło rozwiązanie ze znacznie mniejszą resztą wynoszącą $3,4\times10^{-9}$. Współczynnik tarcia dla zadanej liczby Reynoldsa wyniósł przy tym 0,0124.

Wyznaczanie C_f za pomocą Solvera

Polecenie *Szukaj wyniku* sprawdza się dobrze w tym prostym przykładzie i w praktyce nie musielibyśmy tutaj stosować bardziej zaawansowanego narzędzia, jakim jest dodatek Solver. Jednak w celach ilustracyjnych chcę pokazać, jak można go wykorzystać do rozwiązania tego samego problemu. Arkusz należy przygotować tak samo jak poprzednio, natomiast z głównego menu należy wybrać polecenie *Solver* zamiast *Szukaj wyniku*. Okno dialogowe *Solver - Parametry* dla tego konkretnego przykładu zostało pokazane na rysunku 9.9.



Rysunek 9.9. Okno dialogowe Solver - Parametry dla przykładu z równaniem nieliniowym

W polu *Komórka celu* umieściłem odwołanie do komórki *C5*. Spośród opcji *Równa* wybrałem *Wartość* i w polu obok wpisałem `0`. Tak jak poprzednio, komórka *C5* zawiera prawą stronę naszego równania, która ma przyjąć wartość równą 0. Następnie w polu *Komórki zmieniane* umieściłem odwołanie do komórki *C4*, która zawiera wartość początkową współczynnika tarcia. Kliknięcie przycisku *Rozwiąż* rozpoczyna proces rozwiązywania równania.

Solver rzeczywiście znajduje rozwiązanie, dając wartość współczynnika tarcia równą 0,0124 przy reszcie wynoszącej $-5,45\times10^{-7}$. Są to wartości porównywalne z uzyskanymi za pomocą polecenia *Szukaj wyniku*.

Spróbujmy teraz, trochę dla zabawy, rozwiązać ten sam problem, ale z większą wartością początkową współczynnika tarcia, np. 1 lub 5 albo jeszcze więcej. W takim przypadku Solver nie potrafi znaleźć rozwiązania! Przy takich wartościach początkowych Solver przeskakuje rzeczywiste rozwiązanie i w efekcie próbuje przyjąć dla dalszych iteracji wartość współczynnika tarcia równą 0, co oczywiście oznacza w tym przypadku błąd dzielenia przez zero i przedwczesne zakończenie działania Solvera. Ten prosty przykład świadczy dobitnie o tym, jak wielkie znaczenie ma właściwy dobór wartości początkowych. Ponadto pokazuje, że jeżeli pierwsza próba jest nieudana, należy zmienić wartość początkową i spróbować ponownie.

9.3. Automatyzowanie żmudnych zadań za pomocą VBA

Problem

Wiemy już, jak stosować polecenie *Szukaj wyniku* i dodatek Solver do rozwiązywania równań nieliniowych, ale kłopot polega na tym, że potrzebujemy rozwiązać takie równanie dla określonego przedziału wartości, a ręczne wykonanie tego zadania staje się niezwykle żmudne, bo wymaga wielokrotnego stosowania tych narzędzi.

Rozwiązanie

Można zautomatyzować ten proces, wykorzystując język VBA, który umożliwia programowe wywoływanie Solvera lub polecenia *Szukaj wyniku*.

Analiza

Dodatek Solver i polecenie *Szukaj wyniku* są wygodnymi narzędziami do rozwiązywania rozmaitych problemów w Excelu. Ale jeszcze lepsze jest to, że możemy używać tych narzędzi bezpośrednio w kodzie VBA, nawet bez ręcznego otwierania ich okien dialogowych, zaznaczania komórek itp. Oznacza to, że możemy zautomatyzować wiele zadań wymagających użycia Solvera lub polecenia *Szukaj wyniku*, oszczędzając w ten sposób wiele cennego czasu.

Rozważmy ponownie przykładowe równanie z receptury 9.2. Dla wygody przytaczam je tutaj jeszcze raz:

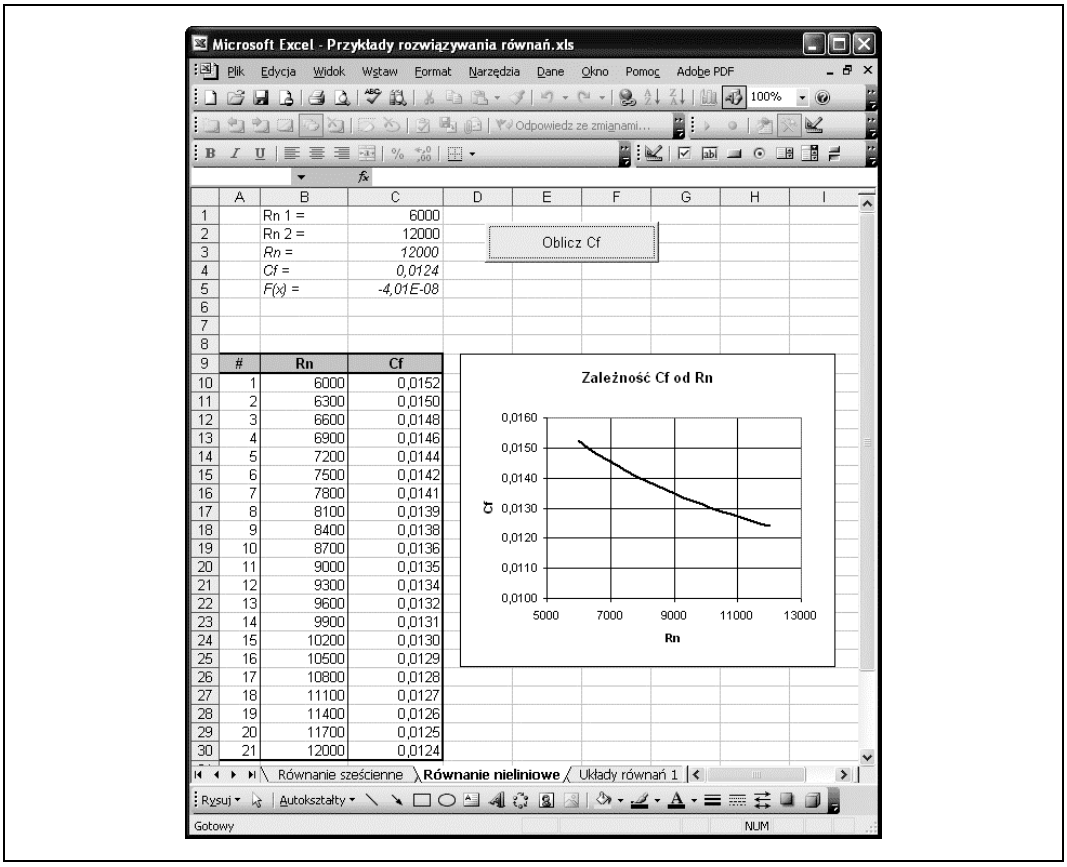
$$\frac{0{,}242}{\sqrt{C_f}} = \log(R_N C_f)$$

Powstaje pytanie: jak utworzyć wykres tego równania? Gdybyśmy chcieli wykreślić krzywą zależności C_f od R_N, musielibyśmy dla każdego punktu wyznaczającego tę krzywą użyć Solvera lub polecenia *Szukaj wyniku*. Nawet gdybyśmy nie zamierzali tworzyć wykresu, lecz tylko samą tabelę z wartościami C_f dla różnych wartości R_N, wówczas również czekałby nas trud ręcznego, wielokrotnego stosowania tych narzędzi dla każdej wartości R_N. A można sobie wyobrazić jeszcze bardziej złożony przypadek, gdy równanie zawiera jeszcze jeden parametr, który również chcemy systematycznie zmieniać. To musiałoby być naprawdę nudne zajęcie!

Na szczęście twórcy Solvera i polecenia *Szukaj wyniku* umożliwili wykorzystanie możliwości tych narzędzi z poziomu języka VBA, dzięki czemu możemy tego typu zadania zautomatyzować.

Automatyzowanie polecenia Szukaj wyniku

Aby pokazać zautomatyzowany sposób wykorzystania polecenia *Szukaj wyniku*, rozpoczniemy od wprowadzenia pewnych modyfikacji do arkusza pokazanego na rysunku 9.7. Naszym celem będzie utworzenie podprogramu VBA wywołującego polecenie *Szukaj wyniku*, które z kolei będzie obliczać wartości C_f dla określonego zakresu wartości R_N. Aby móc uruchomić podprogram VBA, dodałem do arkusza element sterujący w postaci przycisku. Kliknięcie tego przycisku będzie rozpoczynało proces rozwiązywania. Wstawiłem również tabelę, która będzie automatycznie wypełniana obliczonymi wartościami C_f i R_N. Na koniec dodałem jeszcze wykres krzywej reprezentującej uzyskane wyniki. Zmodyfikowany w ten sposób arkusz został przedstawiony na rysunku 9.10.



Rysunek 9.10. Przykład automatyzacji polecenia Szukaj wyniku

Jak widać, jest tutaj kilka nowych elementów. Każdy z nich postaram się objaśnić. Elementy pochodzące z oryginalnego arkusza zostały wyróżnione czcionką pochyłą (patrz rysunek 9.10), natomiast cała reszta jest tutaj nowa.

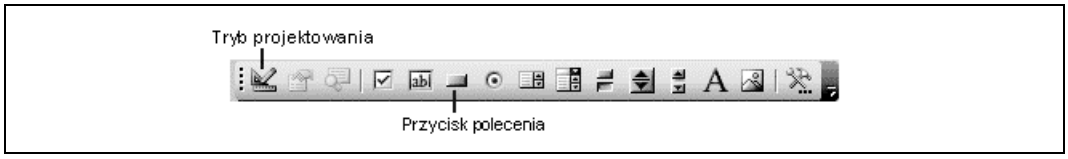
Pierwszą rzeczą, jaką dodałem do tego arkusza, było wpisanie w komórkach *C1* i *C2* wartości określających początek i koniec rozważanego przedziału wartości R_N. Na lewo od tych komórek umieściłem odpowiednie etykiety objaśniające.

Następnie w komórkach od *A9* do *C30* skonstruowałem tabelę przeznaczona do przechowywania wartości R_N i odpowiadających im wartości C_f . Początkowo tabela ta zawierała tylko nagłówki kolumn w wierszu *9* i numery wierszy w kolumnie *A*. Kolumny C_f i R_N pozostały na razie puste — zostaną wypełnione po uruchomieniu podprogramu VBA, który wkrótce zaprezentuję.

Po zestawieniu tabeli utworzyłem wykres, który będzie prezentował zależność C_f od R_N. Jako że tabela danych na razie jest pusta, na wykresie nie zobaczymy żadnej krzywej. Jednak przygotowanie wykresu już na tym etapie oznacza, że po zakończeniu obliczeń wykonywanych przez podprogram VBA zostanie on automatycznie zaktualizowany i oczekiwana krzywa pojawi się. (Jeśli Czytelnik potrzebuje dodatkowych informacji na temat tworzenia wykresów, proponuję zajrzeć do rozdziału 4.).

Ostatnim elementem, jaki musimy jeszcze dodać do arkusza, jest przycisk, który na rysunku 9.10 nosi nazwę *Oblicz Cf*. Excel pozwala nam dodawać do arkuszy różne elementy sterujące, umożliwiając w ten sposób konstruowanie nawet bardzo rozbudowanych interfejsów graficznych (ang. GUI — *Graphical User Interface*) podobnych do tych, które znamy ze standardowych programów działających w systemie Windows. Dla celów niniejszego przykładu pokażę, jak wstawić do arkusza prosty przycisk i powiązać go z kodem VBA, tak aby kliknięcie tego przycisku wywoływało jakąś akcję. W tym przypadku kod VBA obliczy wartości C_f dla całego zakresu wartości R_N i wypełni tabelę danych.

Zanim przystąpimy do umieszczenia przycisku w arkuszu, musimy upewnić się, że pasek narzędzi *Przybornik formantów* jest widoczny w Excelu. Pasek ten został pokazany na rysunku 9.11.



Rysunek 9.11. Pasek narzędzi Przybornik formantów

Jeśli jest niewidoczny, należy z głównego menu Excela wybrać polecenie *Widok/Paski narzędzi/Przybornik formantów*.

Aby dodać przycisk do arkusza, wystarczy kliknąć ikonę *Przycisk polecenia* (rysunek 9.11), a następnie kliknąć w obrębie arkusza, tam, gdzie chcemy przycisk umieścić. Po wykonaniu tych czynności na naszym arkuszu powinien pojawić się nowy element w postaci przycisku. Początkowo będzie on otoczony uchwytami (kółeczkami), które możemy przeciągać, zmieniając w ten sposób rozmiary przycisku. Możemy również przeciągać sam przycisk, aby zmienić jego położenie.

W momencie dodawania nowego elementu sterującego do arkusza Excel przechodzi w **tryb projektowania**, co jest sygnalizowane wciśnięciem przycisku *Tryb projektowania* na pasku *Przybornik formantów*. W tym trybie możemy zmieniać rozmiary elementów kontrolnych i przemieszczać je. Po wyjściu z trybu projektowania kliknięcie elementu kontrolnego spowoduje

jego uaktywnienie. Tryb projektowania można na przemian włączać i wyłączać, klikając wspomniany już przycisk *Tryb projektowania* na pasku narzędzi.

Elementy sterujące dostępne na pasku *Przybornik formantów* są kontrolkami ActiveX. Jeśli w trybie projektowania klikniemy taki element prawym przyciskiem myszy, wówczas z menu podręcznego możemy wybrać polecenie *Właściwości*, które otwiera okno udostępniające wszystkie właściwości tego elementu. W tym oknie możemy dokonać edycji wielu charakterystyk wybranego elementu kontrolnego. Na przykład zmiana właściwości *Caption* oznacza zmianę tekstu wyświetlanego na samym elemencie kontrolnym. W naszym przykładzie ten tekst został zmieniony na *Oblicz Cf*, co widać na rysunku 9.10.

Po dodaniu przycisku musimy powiązać go jakimś kodem VBA, jeśli kliknięcie tego przycisku ma powodować podjęcie jakiejkolwiek akcji. A zatem następny etap będzie polegał na napisaniu odpowiedniego podprogramu i powiązaniu go z nowym przyciskiem.

Przykład 9.1 przedstawia procedurę, jaką przygotowałem na potrzeby naszego przykładu. (Czytelników, którzy poszukują informacji na temat dołączania takich jak ta procedur VBA do arkuszy Excela, odsyłam do rozdziału 2.).

Przykład 9.1. Procedura ComputeCf

```
Public Sub ComputeCf()
    Dim inc As Double

    With Worksheets("Równanie nieliniowe")
        inc = (.Range("Rn_2") - .Range("Rn_1")) / 20

        For i = 0 To 20
            .Range("Rn") = .Range("Rn_1") + (inc * i)
            .Range("Fx").GoalSeek goal:=0, ChangingCell:=.Range("Cf")
            .Cells(10 + i, 2) = .Range("Rn")
            .Cells(10 + i, 3) = .Range("Cf")
        Next i
    End With
End Sub
```

Procedura, której nadałem nazwę `ComputeCf`, pobiera granice przedziału zmienności R_N zapisane w komórkach *C1* i *C2*, a następnie, wykorzystując polecenie *Szukaj wyniku*, oblicza 20 wartości C_f . Po obliczeniu wszystkich wartości C_f wyniki są zapisywane w tabeli obejmującej komórki arkusza od *A9* do *C30*.

Pierwsza linia procedury zawiera deklarację zmiennej lokalnej o nazwie `inc`, która będzie przechowywać przyrost zmiennej R_N (przyrost ten jest obliczany w dalszej części procedury), czyli różnicę między kolejnymi wartościami tej zmiennej.

W następnej linii występuje instrukcja `With`. W językach zorientowanych obiektowo, takich jak VBA, gdzie dostęp do składników obiektu realizowany jest poprzez składnię „z kropką" (widoczną w powyższym przykładzie), instrukcje `With` pełnią bardzo użyteczną rolę. Pozwalają one określić, który obiekt ma być przyjęty jako domyślny przy odwoływaniu się do poszczególnych składników. Dzięki temu możemy zaoszczędzić pisania, opuszczając w tych odwołaniach nazwę obiektu-rodzica. Na przykład w tym podprogramie użyłem instrukcji `With Worksheets("Równanie nieliniowe")`, informując w ten sposób VBA, że chcę uzyskać dostęp do obiektu typu `Worksheet` o nazwie `Równanie nieliniowe`. Jest to nazwa arkusza

wykorzystywanego w naszym przykładzie i można ją zobaczyć na zakładce arkusza w dolnej części rysunku 9.10. Teraz za każdym razem, gdy używam instrukcji takiej jak `.Range("Rn_2")`, odwołując się w tym przypadku do zakresu komórek, VBA przyjmuje domyślnie, że jest to odwołanie do składnika typu `Range` obiektu `Worksheet` o nazwie `Równanie nieliniowe`. Bez użycia instrukcji `With` musielibyśmy napisać: `Worksheets("Równanie nieliniowe").Range ("Rn_2")`.

Instrukcji `With` musi towarzyszyć instrukcja zamykająca `End With`, tak jak w przykładzie 9.1. W ten sposób działanie instrukcji `With` zostaje ograniczone do tych instrukcji, które są zawarte między `With` a `End With`.

Następna instrukcja po `With` oblicza wartość zmiennej `inc`, odwołując się do komórek zawierających granice przedziału zmiennej R_N za pomocą obiektów typu `Range`, tak jak to zostało wcześniej opisane. Jako odwołania do tych komórek zastosowałem ich nazwy, `Rn_2` i `Rn_1`. Uważam, że nazwy są bardziej czytelne niż standardowe odwołania. O nadawaniu nazw komórkom można przeczytać w recepturze 1.14.

Po obliczeniu wartości `inc` podprogram wchodzi w pętlę `For` przebiegającą przez wszystkie wartości R_N. Dla każdej wartości R_N obliczana jest iteracyjnie wartość C_f za pomocą polecenia *Szukaj wyniku*. Oto objaśnienie każdej linii tej pętli:

`.Range("Rn") = .Range("Rn_1") + (inc * i)`
: W tej linii obliczana jest kolejna wartość R_N, która zapisywana jest w komórce o nazwie `Rn` (komórka *C3*).

`.Range("Fx").GoalSeak goal:=0, ChangingCell:=.Range("Cf")`
: Tutaj następuje wywołanie polecenia *Szukaj wyniku* (ang. *Goal Seak*) dla komórki o nazwie `Fx` (komórka *C4*) jako komórki docelowej, której zawartość ma osiągnąć wartość 0. Wartość docelowa jest tutaj ustalana przez `goal:=0`. Parametr *Zmieniając komórkę* jest określany przez `ChangingCell:=.Range("Cf")`. Wszystko to stanowi informację dla polecenia *Szukaj wyniku*, że zawartość komórki `Cf` ma być tak zmieniana, aby w komórce `Fx` pojawiła się wartość równa zero.

`.Cells(10 + i, 2) = .Range("Rn")`
: W tej linii zawartość komórki `Rn` jest zapisywana w tabeli danych, począwszy od komórki *B10*. Numer wiersza jest zwiększany o 1 przy każdym przejściu pętli. Dla wygody zastosowałem tutaj styl odwołań *W1K1*. (Więcej informacji na temat tego stylu zawiera receptura 1.6).

`.Cells(10 + i, 3) = .Range("Cf")`
: Wartość C_f wyznaczona przez polecenie *Szukaj wyniku* jest zapisywana do tabeli danych. Tutaj również zastosowałem styl *W1K1*.

Teraz możemy powiązać napisaną właśnie procedurę `ComputeCf` z przyciskiem dodanym wcześniej do arkusza. W tym celu należy najpierw włączyć tryb projektowania (w sposób opisany wcześniej), a następnie kliknąć ten przycisk prawym przyciskiem myszy. Z otwartego w ten sposób menu podręcznego należy wybrać polecenie *Wyświetl kod*, które przeniesie nas do edytora VBA, gdzie powinniśmy zobaczyć nową procedurę przygotowaną dla naszego przycisku. Procedura ta została pokazana w przykładzie 9.2.

Przykład 9.2. Procedura przycisku

```
Private Sub CommandButton1_Click()

End Sub
```

Jest to procedura obsługująca zdarzenie polegające na kliknięciu przycisku. Ponieważ nasz przycisk nosi domyślną nazwę `CommandButton1`, stąd nazwą tej procedury jest `CommandButton1_Click`.

Nazwę elementu sterującego typu ActiveX można zmienić przez zmianę właściwości *Name* w oknie właściwości tego elementu. Aby dokonać takiej zmiany, należy włączyć tryb projektowania, kliknąć wybrany element prawym przyciskiem myszy i z menu podręcznego wybrać polecenie *Właściwości*. W otwartym w ten sposób oknie *Properties* można zmodyfikować parametry wybranego elementu.

Opisywana procedura jest na razie pusta. Umieścimy w niej wywołanie napisanej wcześniej procedury `ComputeCf`, tak jak to zostało pokazane w przykładzie 9.3.

Przykład 9.3. Wywołanie procedury ComputeCf

```
Private Sub CommandButton1_Click()
    ComputeCf
End Sub
```

Jeśli teraz klikniemy przycisk w Excelu (po wyłączeniu trybu projektowania), spowoduje to wykonanie procedury obliczającej wartości C_f dla zadanego zakresu wartości R_N. Wyniki tych obliczeń zobaczymy w tabeli oraz na wykresie. Jeżeli zechcemy wyznaczyć wartości C_f dla innego zakresu wartości R_N, jedyne, co musimy zrobić, to wpisać granice nowego zakresu (w komórkach *C1* i *C2*) i ponownie kliknąć przycisk.

Excel zawiera jeszcze jeden zestaw elementów sterujących dostępnych na pasku *Formularze* (widocznym po wybraniu polecenia *Widok/Paski narzędzi/Formularze*). Elementy te wyglądają tak jak stosowane w tym przykładzie elementy ActiveX, ale dają nam większą kontrolę i różnią się od nich sposobem modyfikowania właściwości oraz przypisywania funkcji. Ogólnie, elementy typu ActiveX są nowsze i lepiej oprogramowane, a elementy formularzowe są starsze i utrzymywane dla zachowania zgodności ze starszymi wersjami programów.

Automatyzowanie Solvera

Wykorzystanie Solvera możemy zautomatyzować w sposób analogiczny do tego, jaki zastosowaliśmy w celu zautomatyzowania użycia polecenia *Szukaj wyniku*. W rzeczywistości, aby w rozważanym przykładzie zamiast polecenia *Szukaj wyniku* zastosować dodatek Solver, wystarczą niewielkie modyfikacje kodu VBA. Wszystkie elementy arkusza mogą pozostać takie same jak poprzednio, a zmodyfikować musimy tylko procedurę `ComputeCf`. Nową wersję tej procedury przedstawia przykład 9.4.

Przykład 9.4. Procedura ComputeCf z użyciem Solvera

```
Public Sub ComputeCf()
    Dim inc As Double

    With Worksheets("Równanie nieliniowe")
        inc = (.Range("Rn_2") - .Range("Rn_1")) / 20
```

```
        For i = 0 To 20
            .Range("Rn") = .Range("Rn_1") + (inc * i)

             ' Tutaj rozpoczyna się nowy kod:
            .Range("Cf") = 0.001

            SolverOK SetCell:=Range("Fx"), MaxMinVal:=3, ValueOf:=0,
                     ByChange:=Range("Cf")

            SolverSolve UserFinish:=True

            SolverFinish KeepFinal:=1

            .Cells(10 + i, 2) = .Range("Rn")
            .Cells(10 + i, 3) = .Range("Cf")
        Next i
    End With
End Sub
```

Zmiana w stosunku do poprzedniej wersji obejmuje cztery linie kodu następujące po komentarzu `'Tutaj rozpoczyna się nowy kod:`. Te nowe linie to:

`.Range("Cf") = 0.001`
: W tej linii ustawiana jest wartość początkowa zmiennej C_f. W tym przypadku będzie ona wynosić 0,001 i będzie służyć jako wartość wstępna, ustawiana przed wywołaniem Solvera.

`SolverOK SetCell:=Range("Fx"), MaxMinVal:=3, ValueOf:=0, ByChange:=Range("Cf")`
: W tej linii następuje wywołanie wewnętrznej procedury Solvera o nazwie `SolverOK` w celu inicjalizacji modelu Solvera. Komórka docelowa (jest nią komórka o nazwie `Fx`) jest określana przez `SetCell:=Range("Fx")`. Ponieważ chcemy jej przypisać konkretną wartość, ustalamy `MaxMinVal:=3`. Gdyby chodziło nam o maksymalizowanie wartości docelowej, należałoby napisać `MaxMinVal:=1`, a w przypadku minimalizowania — `MaxMinVal:=2`. Kolejny zapis `ValueOf:=0` jest informacją dla Solvera, że wartość docelowa wynosi 0. Z kolei zapis `ByChange:=Range("Cf")` informuje Solver, że ma poszukiwać rozwiązania przez zmianę zawartości komórki o nazwie `Cf`.

`SolverSolve UserFinish:=True`
: Wywołanie procedury `SolverSolve` uruchamia Solvera. Przypisanie `UserFinish:=True` ma na celu poinformowanie Solvera, aby po odnalezieniu rozwiązania nie wyświetlał okna dialogowego *Solver - Wyniki*. Wyświetlanie tego okna może być przydatne np. podczas debugowania tego typu procedur, a wówczas należy ustawić ten parametr na `False`.

`SolverFinish KeepFinal:=1`
: Ta linia nakazuje Solverowi zachować znalezione wyniki.

Po dokonaniu tych zmian w kodzie procedury i uruchomieniu jej przez kliknięcie przycisku na przykładowym arkuszu, powinniśmy ujrzeć wyniki bardzo podobne do tych, jakie uzyskaliśmy, stosując polecenie *Szukaj wyniku*.

Jeśli podczas uruchamiania nowej procedury wystąpi błąd sygnalizowany komunikatem o treści *Sub or Function not Defined*, należy sprawdzić, czy Solver jest zarejestrowany w środowisku VBA. W tym celu należy z głównego menu edytora VBA wybrać polecenie *Tools/References* i w otwartym w ten sposób oknie dialogowym *References* upewnić się, że zaznaczona jest pozycja *SOLVER*. Na koniec należy kliknąć przycisk *OK*.

Informacje dodatkowe

Za pomocą kodu VBA można kontrolować znacznie więcej aspektów Solvera, niż tutaj pokazałem. Więcej informacji można znaleźć w pomocy Excela oraz na stronie internetowej pomocy technicznej Microsoftu *http://support.microsoft.com*, gdzie należy odszukać artykuły na temat Solvera (wśród wielu tematów dotyczących innych produktów tej firmy). Dla pragnących zgłębić tajniki stosowania Solvera najbardziej pomocnym może okazać się artykuł pt. „How to create Visual Basic macros by using Excel Solver in Excel 97" (artykuł nr 843304)[2].

9.4. Rozwiązywanie układów równań liniowych

Problem

Chcielibyśmy rozwiązać układ równań liniowych za pomocą Excela.

Rozwiązanie

Układy równań liniowych można rozwiązywać w Excelu, stosując różne podejścia. Na przykład możemy napisać program w języku VBA będący implementacją jednego ze standardowych algorytmów rozwiązywania układów równań. Takie podejście jest niewątpliwie skuteczne, ale czasochłonne — wymaga napisania programu i jego przetestowania w celu usunięcia ewentualnych błędów. Jako że Excel udostępnia nam narzędzia do wykonywania działań na macierzach, możemy układ równań przedstawić w postaci macierzowej i znaleźć rozwiązanie przez odwrócenie odpowiedniej macierzy. Jeszcze łatwiejszy sposób polega na wykorzystaniu dodatku Solver. W poniższej analizie zaprezentowane zostaną dwa ostatnie rozwiązania.

Analiza

Jeśli potrafimy zapisać układ równań w postaci równania macierzowego, to możemy znaleźć rozwiązanie, stosując proste odwracanie macierzy. Jeśli nie możemy lub nie chcemy tego zrobić, albo mamy do czynienia z układem równań nieliniowych, możemy wybrać jedno z podejść opartych na wykorzystaniu Solvera.

Odwracanie macierzy

W recepturze 7.10 pokazałem, jak wykorzystać funkcje wbudowane Excela operujące na macierzach do mnożenia i odwracania macierzy. Te same funkcje możemy zastosować do rozwiązania układu równań liniowych zapisanego w następującej postaci:

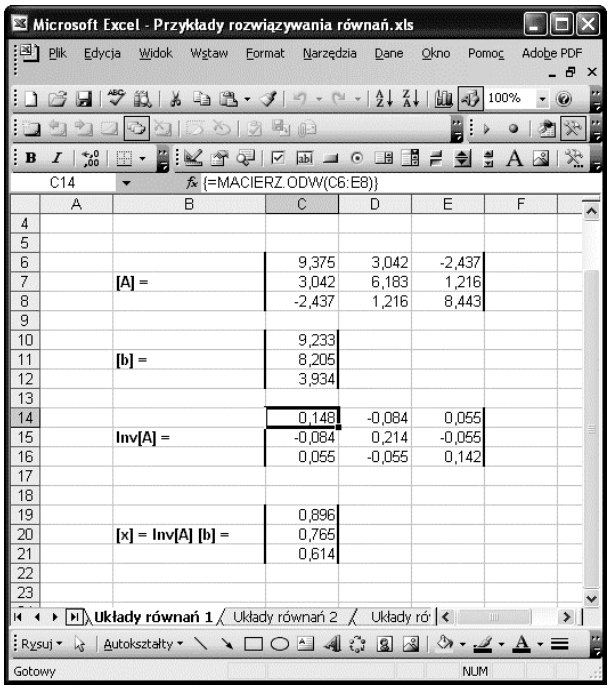
$$[A][x]=[b]$$

Nasze podejście do rozwiązania tego zadania będzie polegało na bezpośrednim zastosowaniu wspomnianych funkcji. Wartości niewiadomych x spełniające powyższe równanie znajdziemy przez rozwiązanie następującego równania macierzowego:

[2] Artykuł ten dostępny jest tylko w języku angielskim — *przyp. tłum.*

$$[x] = [A]^{-1}[b]$$

Załóżmy na przykład, że dane są macierze [*A*] i [*b*], takie jak na rysunku 9.12.



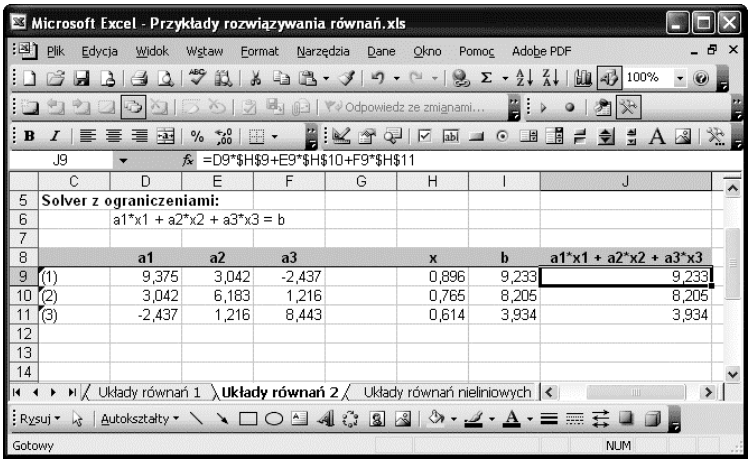
Rysunek 9.12. Rozwiązywanie równania macierzowego

Pierwszy etap rozwiązywania tego równania to odwrócenie macierzy [*A*]. W tym celu należy użyć formuły `=MACIERZ.ODW(C6:E8)`, pamiętając przy tym, że jest to formuła tablicowa, którą należy wprowadzić do określonego zakresu komórek i zatwierdzić wciśnięciem klawiszy *Ctrl+Shift+Enter*. Rezultat takiego odwrócenia macierzy jest widoczny na rysunku 9.12 w komórkach od *C14* do *E16*.

Drugi i zarazem ostatni etap polega na pomnożeniu odwróconej macierzy [*A*] przez macierz [*b*] za pomocą formuły `=MACIERZ.ILOCZYN(C14:E16;C10:C12)`, która również jest formułą tablicową i należy ją wprowadzić do zakresu komórek. Ostateczny wynik został pokazany na rysunku 9.12 w komórkach od *C19* do *C21*.

Wykorzystanie Solvera i ograniczeń

Zamiast funkcji operujących na macierzach, które mogą się okazać się mało poręczne w przypadku dużych macierzy, do rozwiązania układu równań liniowych (a także nieliniowych, jak się okaże w recepturze 9.5) możemy wykorzystać dodatek Solver. Pokażę teraz, jak za pomocą tego dodatku i odpowiednich ograniczeń można rozwiązać ten sam układ równań co poprzednio. Odpowiednie ustawienie arkusza pokazane jest na rysunku 9.13.



Rysunek 9.13. Rozwiązywanie układu równań liniowych za pomocą Solvera z ograniczeniami

Problem polega w zasadzie na rozwiązaniu układu trzech równań z trzema niewiadomymi x_1, x_2 i x_3. Równania mają postać: $a_1x_1+a_2x_2+a_3x_3 = b$. Tym razem interesować nas będzie układ równań w takiej właśnie postaci, a nie, jak poprzednio, w postaci macierzowej. Jest to ten sam układ, tylko inaczej zapisany.

Rozwiązywanie rozpoczniemy od przygotowania tabeli z wartościami współczynników *a* i *b* oraz wartościami początkowymi (przewidywanymi) niewiadomych *x*. Jak widać na rysunku 9.13, współczynniki *a* umieściłem w komórkach od *D9* do *F11*, współczynniki *b* — w komórkach od *I9* do *I11*, a niewiadome *x* — w komórkach od *H9* do *H11*.

Następnie w komórkach od *J9* do *J11* umieściłem formuły następującej postaci: `=D9*$H$9+E9*$H$10+F9*$H$11`. Każda z tych formuł oblicza współczynnik *b* dla odpowiedniego równania w oparciu o wartości współczynników *a* i początkowe wartości niewiadomych *x*. W przypadku idealnym wyniki w tej kolumnie powinny być równe wartościom współczynników *b* umieszczonym w sąsiedniej kolumnie. Ponieważ przyjęte przez nas wartości niewiadomych *x* są tylko wartościami przewidywanymi, dlatego w tym momencie zawartości tych kolumn mogą się różnić. Możemy jednak użyć Solvera do wyznaczenia takich wartości niewiadomych *x*, dla których współczynniki *b* w obu kolumnach będą takie same. W ten sposób uzyskamy rozwiązanie naszego układu równań.

Otwórzmy okno dialogowe Solvera, wybierając z głównego menu polecenie *Narzędzia/Solver*. Tym razem jednak nie będziemy minimalizować, maksymalizować ani ustawiać konkretnej wartości dla komórki docelowej. Pole *Komórka celu* pozostawimy puste, tak jak na rysunku 9.14.

W polu *Komórki zmieniane* umieścimy odwołanie do zakresu komórek zawierających wartości niewiadomych *x*, W tym przypadku są to komórki od *H9* do *H11*. Teraz nadszedł czas na dodanie kilku ograniczeń.

Naszym celem jest pozwolenie Solverowi na zmienianie wartości niewiadomych *x* przy nałożonych ograniczeniach polegających na tym, że obliczane wartości współczynników *b* w komórkach od *J9* do *J11* mają być równe zadanym wartościom tych współczynników w komórkach od *I9* do *I11*. Musimy więc utworzyć takie ograniczenie dla każdego współczynnika *b*.

Rysunek 9.14. Solver z ograniczeniami

Utworzone przeze mnie ograniczenia widoczne są w oknie dialogowym Solvera pokazanym na rysunku 9.14. Aby utworzyć takie ograniczenie, należy otworzyć okno dialogowe *Dodaj warunek ograniczający* (rysunek 9.15) kliknięciem przycisku *Dodaj*.

Rysunek 9.15. Okno dialogowe Dodaj warunek ograniczający

W polu *Adres komórki* należy umieścić odwołanie do komórki zawierającej obliczaną wartość współczynnika *b*, a w polu *Warunek ograniczający* — odwołanie do komórki zawierającej odpowiednią wartość zadaną tego współczynnika. Z listy rozwijanej w środkowej części okna należy wybrać znak równości (=). Kliknięcie przycisku *Dodaj* spowoduje zatwierdzenie utworzonego ograniczenia bez zamykania okna *Dodaj warunek ograniczający*, tak więc możemy od razu przystąpić do tworzenia kolejnych dwóch ograniczeń. Po dodaniu ostatniego ograniczenia i zamknięciu okna kliknięciem przycisku *Anuluj* powinniśmy zobaczyć te ograniczenia w postaci takiej jak na rysunku 9.14.

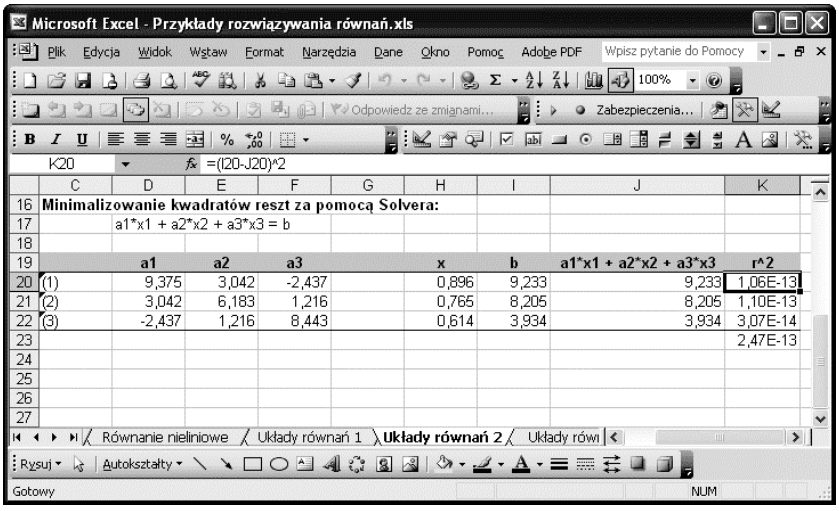
Być może Czytelnik zastanawia się, dlaczego nie określiliśmy komórki docelowej. Otóż okazuje się, że w takiej sytuacji Solver tworzy tzw. pozorną komórkę docelową. Ponieważ dla samego rozwiązania naszego zadania zawartość tej komórki jest nieistotna, możemy poprzestać na zdefiniowaniu samych ograniczeń.

Po skompletowaniu wszystkiego możemy kliknąć przycisk *Rozwiąż*. Solver powinien szybko znaleźć rozwiązanie, takie jak na rysunku 9.13, gdzie kolumna wartości *x* (komórki od *H9* do *H11*) reprezentuje rozwiązanie naszego układu równań. Jak widać, pokrywa się ono z rozwiązaniem uzyskanym poprzednio z wykorzystaniem macierzy.

Minimalizowanie reszt za pomocą Solvera

Inne podejście do wykorzystywania ograniczeń w Solverze polega na takim sformułowaniu zadania, aby sprowadzić go do problemu minimalizacji reszt. Możemy zrobić to bardzo ła-

two, dodając do tabeli z rysunku 9.14 jeszcze jedną kolumnę. Poszerzona w ten sposób tabela została pokazana na rysunku 9.16.



Rysunek 9.16. Rozwiązywanie układu równań liniowych za pomocą Solvera metodą minimalizacji reszt

Kolumna *K* zawiera kwadraty różnic (reszt) między zadanymi (kolumna *I*) a obliczonymi (kolumna *J*) wartościami współczynników *b*. Formuły w tej kolumnie mają postać: `=(I20-J20)^2`. Komórka *K23* zawiera sumę kwadratów tych reszt obliczaną za pomocą formuły `=SUMA(K20:K22)`.

W zasadzie mamy tu do czynienia z problemem najmniejszych kwadratów, polegającym na minimalizowaniu sumy kwadratów reszt. Korzystając z Solvera, możemy bez trudu zminimalizować tę sumę przez zmianę wartości niewiadomych *x* w kolumnie *H*.

Sprowadza się to do bezpośredniego zastosowania opisywanych już w tym rozdziale technik opartych na wykorzystaniu Solvera. Na rysunku 9.17 został pokazany model właściwy dla naszego przykładu.

Rysunek 9.17. Solver bez ograniczeń

W polu *Komórka celu* należy umieścić odwołanie do komórki zawierającej sumę kwadratów reszt, czyli *K23*. Tym razem będziemy chcieli zminimalizować zawartość tej komórki. W polu *Komórki zmieniane* należy umieścić odwołanie do komórek zawierających wartości x, które będą zmieniane. Warto podkreślić, że nie ma tutaj żadnych ograniczeń.

Po kliknięciu przycisku *Rozwiąż* powinniśmy zobaczyć wyniki takie jak na rysunku 9.16. Jak należało się spodziewać, pokrywają się one z wynikami uzyskanymi za pomocą metod opisywanych wcześniej.

To, które podejście jest lepsze, zależy od naszych upodobań i rozmiarów konkretnego układu równań. W przypadku układu o dużych rozmiarach prawdopodobnie zastosowałbym metodę trzecią — Solver bez ograniczeń — ponieważ pozwala ona najszybciej dokonać odpowiednich ustawień. Jak wynika z analizy powyższego przykładu, ten sam problem można rozwiązać w Excelu, stosując z równym powodzeniem różne metody.

9.5. Rozwiązywanie układów równań nieliniowych

Problem

Musimy rozwiązać układ równań nieliniowych.

Rozwiązanie

Należy sformułować problem w kategoriach reszt i zastosować Solver w celu ich zminimalizowania, uzyskując w ten sposób rozwiązanie układu.

Analiza

Do rozwiązywania układów równań nieliniowych można zastosować te same techniki wykorzystujące dodatek Solver, które zostały opisane w recepturze 9.4. Jako przykład rozważmy następujące dwa równania:

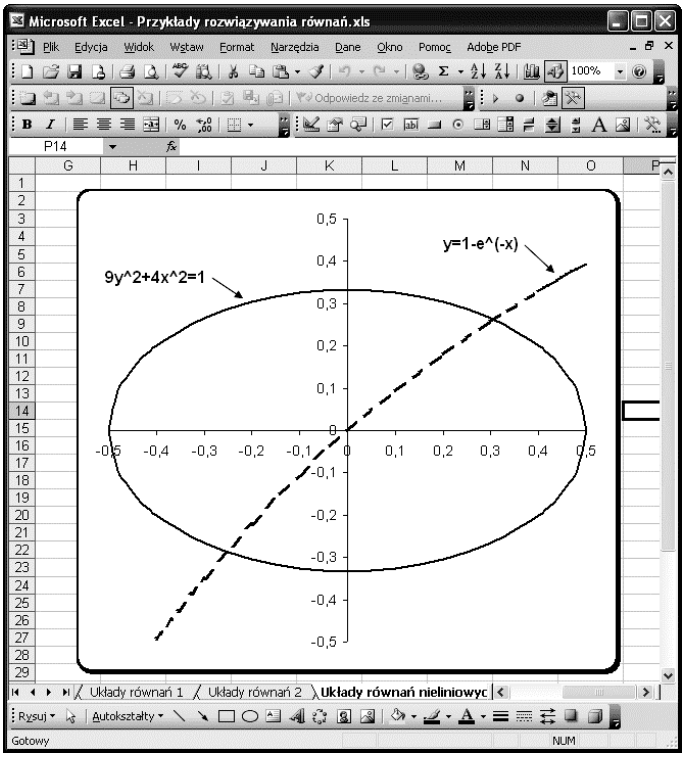
$$y = 1 - e^{-x}$$

$$9y^2 + 4x^2 = 1$$

Załóżmy, że chcemy znaleźć wartość lub wartości zmiennej x spełniające oba równania jednocześnie. Jak wynika z wykresu tych dwóch równań (rysunek 9.18), istnieją dokładnie dwa rozwiązania.

Jeśli chodzi o rozwiązywanie tego typu układów równań, mamy do wyboru kilka różnych podejść. W przypadku, gdy możemy z łatwością rozwiązać te równania względem zmiennej y (w powyższym przykładzie jest to możliwe), wówczas możemy je wzajemnie przyrównać i zapisać w postaci $0 = f(x)–g(x)$, a następnie zastosować jedną z technik opisanych w recepturze 9.2.

Jeśli układ składa się z więcej niż dwóch równań lub nie możemy tych równań rozwiązać względem wspólnej zmiennej, wówczas możemy próbować znaleźć rozwiązanie, stosując metodę minimalizacji reszt opisaną w recepturze 9.4, a konkretnie w punkcie „Minimalizowanie



Rysunek 9.18. Wykres równań nieliniowych

reszt za pomocą Solvera". Stosując to ostatnie podejście, znalazłem dwie wartości zmiennej x spełniające powyższy układ równań, a mianowicie: $x = 0{,}306$ oraz $x = -0{,}253$. Aby móc zastosować Solver do wyznaczenia obu rozwiązań, musiałem przyjąć dwie różne wartości początkowe dla zmiennej x. W pierwszym przypadku wybrałem wartość początkową większą od 0,5, a w drugim — mniejszą od –0,5.

9.6. Rozwiązywanie równań metodami klasycznymi

Problem

Wiemy już, jak wykorzystywać wbudowane funkcje i narzędzia Excela do rozwiązywania równań liniowych i nieliniowych oraz układów takich równań, ale chcielibyśmy również zobaczyć, jak można zaimplementować w Excelu klasyczne algorytmy rozwiązywania równań.

Rozwiązanie

Posługując się językiem VBA, możemy napisać program realizujący dowolny algorytm, tak jak w każdym innym języku programowania. W poniższej analizie pokażę, jak za pomocą VBA można zaimplementować dwie klasyczne metody: Newtona oraz siecznych.

Analiza

Na przykładach prezentowanych w niniejszej recepturze pokażę, jak można zaimplementować metodę Newtona i metodę siecznych. Są to metody dobrze znane i dokładnie opisane w wielu publikacjach z dziedziny matematyki wyższej i metod numerycznych. Ponadto w internecie można znaleźć implementacje tych metod napisane w różnych językach, np. w Fortranie czy C. Nie powinno więc stanowić problemu zaimplementowanie tych metod w VBA jako funkcji własnych, a następnie wywołanie ich z poziomu arkusza w Excelu.

W poniższych przykładach zastosujemy równanie wielomianowe trzeciego stopnia, które analizowaliśmy już wcześniej w recepturze 9.1. Dla przypomnienia, przytaczam go tutaj ponownie:

$$y = a + bx + cx^2 + dx^3$$

Wykres pokazany na rysunku 9.6 pozwala zorientować się, gdzie leżą pierwiastki tego równania. Za pomocą dodatku Solver moglibyśmy z łatwością wyznaczyć dokładną wartość każdego z tych pierwiastków, ale możemy również wykorzystać do tego celu metodę Newtona lub metodę siecznych (lub jedną z wielu innych metod, dla której potrafimy napisać program), co właśnie teraz uczynimy.

Metoda Newtona

Metoda Newtona polega na przybliżaniu wartości x będącej pierwiastkiem równania przez wyznaczanie punktu przecięcia z osią odciętych stycznej do rozważanej krzywej w punkcie o określonej współrzędnej x. W ten sam sposób wyznacza się kolejne przybliżenia, aż do uzyskania zakładanej zbieżności szacowanej wartości x z wyznaczanym pierwiastkiem. Podstawowy wzór służący do wyznaczania każdej nowej wartości x jako przybliżenia pierwiastka ma następującą postać:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Jak widać, metoda Newtona wymaga obliczenia, lub przynajmniej oszacowania, wartości pierwszej pochodnej rozważanej krzywej. W naszym przykładzie mamy do czynienia z równaniem, które pozwala dokładnie obliczyć tę pochodną. W sytuacjach, w których dokładne obliczenie pochodnej jest trudne bądź niemożliwe (na przykład wtedy, gdy funkcja określona jest przez dane tabelaryczne), należy skorzystać z numerycznych metod różniczkowania. We wstępie do tego rozdziału była już mowa o tym, że Solver w swych obliczeniach stosuje różniczkowanie numeryczne i nawet pozwala nam wybrać schemat różnicy przedniej lub centralnej (zagadnienia te są przedmiotem analizy w recepturze 10.6).

Struktura powyższego równania powinna być dla nas wskazówką, jak należy implementować metodę Newtona. Po pierwsze, potrzebny nam będzie podprogram obliczający wartości rozważanej funkcji. Po drugie, musimy mieć również podprogram obliczający wartości pochodnej tej funkcji. I po trzecie, potrzebny będzie podprogram realizujący powyższe równanie, począwszy od pewnego zadanego punktu aż do osiągnięcia zakładanej zbieżności rozwiązania. Zaprezentuję teraz te podprogramy w wersji przygotowanej przeze mnie dla potrzeb opisywanego tu przykładu. Pierwszy z nich jest pokazany w przykładzie 9.5 i służy do obliczania wartości rozważanej funkcji dla zadanej wartości zmiennej x.

Przykład 9.5. Funkcja Fx

```
Public Function Fx(x As Double) As Double
   Fx = 1 + x * (3 + x * (3 * x – 7))
End Function
```

Podprogram ten jest właściwie funkcją VBA i dlatego zwraca wartość do wywołującego go podprogramu. W tym przypadku zwraca wartość równania sześciennego, które jest przedmiotem naszych rozważań. (Gdy będziemy mieć do czynienia z innym równaniem, wówczas należy odpowiednio zmodyfikować funkcję `Fx`).

Następny podprogram, pokazany w przykładzie 9.6, jest funkcją VBA zwracającą wartość pochodnej naszego równania sześciennego dla zadanej wartości x.

Przykład 9.6. Funkcja dFx

```
Public Function dFx(x As Double) As Double
   dFx = 3 + x * (9 * x – 14)
End Function
```

Ostatni z potrzebnych nam podprogramów jest pokazany w przykładzie 9.7 i stanowi właściwą implementację metody Newtona, wywołując cyklicznie funkcje `Fx` oraz `dFx`.

Przykład 9.7. Metoda Newtona

```
Public Function NewtonsMethod(xO As Double, e As Double, n As Integer) As Double
   Dim i As Integer
   Dim err As Double
   Dim xn As Double
   Dim xn1 As Double

   i = O
   err = 9999
   xn = xO

   While (err > e) And (i < n)
      xn1 = xn - Fx(xn) / dFx(xn)
      err = Abs(xn1 - xn)
      xn = xn1
   Wend
   NewtonsMethod = xn
End Function
```

Zauważmy, że ten podprogram, `NewtonsMethod`, także jest funkcją VBA. To oznacza, że będziemy mogli ją wywołać bezpośrednio z komórki arkusza, a obliczona przez tę funkcję wartość zostanie zwrócona do wywołującej ją komórki. Wkrótce pokażę, jak należy to zrobić, ale wcześniej chciałbym objaśnić kolejne linie kodu funkcji `NewtonsMethod`.

Funkcja ta przyjmuje trzy argumenty. Pierwszy, `xO`, służy jako wartość początkowa dla poszukiwań pierwiastka. Drugi argument, `e`, określa tolerancję zbieżności, która powinna być liczbą rzeczywistą o małej wartości (np. 0,001). Argument ten jest używany do sprawdzania, czy kolejne wyniki obliczeń są dostatecznie zbieżne. Trzeci i ostatni argument, `n`, określa maksymalną liczbę iteracji. Służy on do zatrzymania iteracji, gdyby miała trwać nieskończenie długo z powodu braku zbieżności wyników.

Bezpośrednio pod nagłówkiem tej funkcji deklarowane są pewne zmienne. Kolejno są to: `i` — zmienna licznikowa, `err` — zmienna przechowująca różnicę między dwoma kolejnymi wynikami szacowań wartości x, `xn` — zmienna przechowująca oszacowaną wartość x_n, `xn1` — zmienna przechowująca oszacowaną wartość x_{n+1}.

W kolejnych liniach następuje inicjalizacja tych zmiennych. Zmiennej `i` przypisywana jest wartość `0`, zmiennej `err` — dowolna duża liczba, a zmiennej `xn` — wartość początkowa (argument) `x0`.

Miejscem, gdzie tak naprawdę realizowana jest metoda Newtona, jest pętla `While`. Obliczenia w tej pętli wykonywane są tak długo, aż zmienna `err` osiągnie wartość mniejszą od zadanej tolerancji zbieżności `e` lub osiągnięta zostanie maksymalna liczba iteracji `n`.

Pętla `While` obejmuje trzy instrukcje. W pierwszej obliczana jest kolejna wartość x na podstawie aktualnego oszacowania. Wykorzystany został tutaj wzór Newtona oraz wywoływane są funkcje `Fx` i `dFx`. W drugiej linii obliczany jest błąd zbieżności. W trzeciej i ostatniej linii tej pętli następuje uaktualnienie szacowanej wartości x przed kolejnym etapem iteracji.

Po wyjściu z pętli `While` — z powodu osiągnięcia czy to odpowiedniej zbieżności, czy też maksymalnej liczby iteracji — funkcja zwraca aktualne oszacowanie wartości x.

Aby wywołać tę funkcję z poziomu arkusza, możemy w dowolnej komórce umieścić na przykład taką formułę: `=NewtonsMethod(0; 0,0001; 100)`. Excel będzie wywoływał funkcję `NewtonsMethod` za każdym razem, gdy zajdzie potrzeba aktualizacji arkusza (tym razem nie musimy używać żadnego przycisku, ponieważ `NewtonsMethod` jest funkcją). W tym przypadku zawartość takiej komórki będzie wynosić –0,215, co rzeczywiście jest pierwiastkiem rozważanego równania. Należy zauważyć, że została tu przyjęta wartość początkowa równa 0.

Pozostałe pierwiastki można wyznaczyć, zmieniając odpowiednio wartość początkową. Ustalenie tej wartości na 0,5 da pierwiastek dla x = 1,0. Po zmianie wartości początkowej na 1,75 otrzymamy pierwiastek dla x =1,549.

Metoda siecznych

Zamiast metody Newtona można zastosować metodę siecznych. Ma ona tę zaletę, że nie wymaga obliczania pochodnej rozważanego równania. Musimy za to podać dwie wartości początkowe, aby skonstruować sieczną zamiast stycznej w celu wyznaczenia przybliżonej wartości x. Wzór na obliczanie kolejnych przybliżeń x_{n+1} w metodzie siecznych przedstawia się następująco:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Do zaimplementowania tej metody potrzebna nam będzie nadal funkcja `Fx` z przykładu 9.5, ale nie będzie już potrzebna funkcja `dFx`.

Funkcję VBA stanowiącą właściwą implementację metody siecznych dla naszego równania prezentuje przykład 9.8.

Przykład 9.8. Metoda siecznych

```
Public Function SecantMethod(x0 As Double, x1 As Double, e As Double, n As Integer) As
Double
   Dim i As Integer
   Dim err As Double
   Dim xn As Double
   Dim xn1 As Double
   Dim xnm1 As Double

   i = 0
```

```
    err = 9999
    xn = x1
    xnm1 = x0

    While (err > e) And (i < n)
       xn1 = xn - Fx(xn) * (xn - xnm1) / (Fx(xn) - Fx(xnm1))
       err = Abs(xn1 - xn)
       xnm1 = xn
       xn = xn1
    Wend
    SecantMethod = xn
End Function
```

Funkcja ta jest istotnie podobna do funkcji `NewtonsMethod`. Jest tu jednak kilka różnic, które postaram się objaśnić. Pierwszą z tych różnic jest obecność dodatkowego argumentu `x1`, który reprezentuje drugą wartość początkową dla x. Przypominam, że metoda siecznych wymaga podania dwóch wartości początkowych.

Pozostałe dwie różnice występują w pętli `While`. Jak widać, wyrażenie `xn1` zostało zastąpione wzorem obowiązującym w metodzie siecznych. Ponadto przy każdym przejściu pętli uaktualniane są obie wartości: x_n oraz x_{n-1} przechowywane w zmiennych, odpowiednio: `xn` i `xnm1`.

Funkcję `SecantMethod` możemy wykorzystać w formule arkuszowej, podobnie jak robiliśmy to z funkcją `NewtonsMethod`. Ja na przykład wprowadziłem do trzech różnych komórek następujące formuły:

```
=secantmethod(0; -0,1; 0,0001; 100)
=secantmethod(0,5; -0,6; 0,0001; 100)
=secantmethod(1,75; 1,7; 0,0001; 100)
```

Wartości, jakie otrzymałem, wyniosły, odpowiednio: –0,215, 1,000 i 1,549. Jak widać, są one identyczne z tymi, które otrzymaliśmy, stosując metodę Newtona, i w obu przypadkach są zgodne z wnioskami wynikającymi z analizy wykresu przedstawionego na rysunku 9.6.