INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

Inwestujemy w Waszą przyszłość

INSTEPRO
Zintegrowane Sterowanie Produkcją

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ ROZWOJU REGIONALNEGO

Raport wewnętrzny projektu InStePro

Nr 4.3: Projekt modułu optymalizacji

Data

30.09.2010

Przygotował zespół:

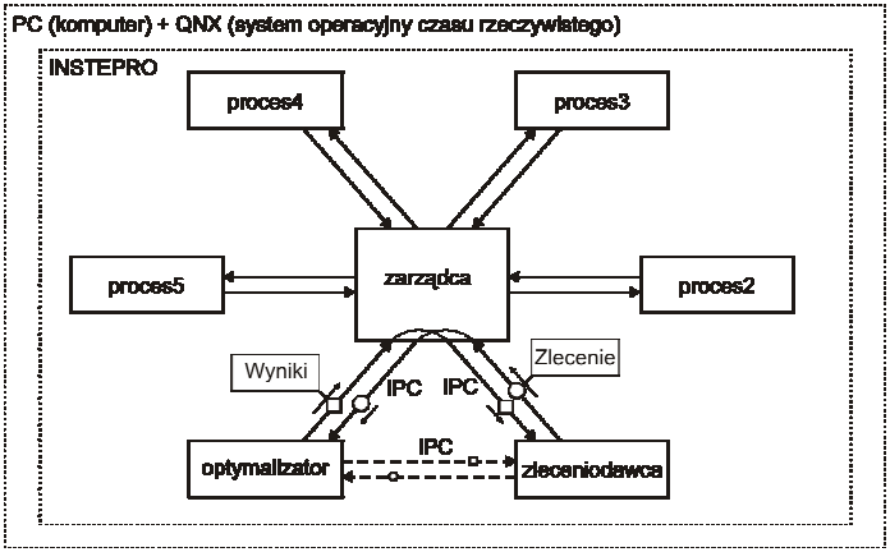
W. Grega
A. Tutaj
D. Kilian

Wstęp

Zgodnie z założeniami projektowymi, jedną z funkcjonalności systemu INSTEPRO ma stanowić algorytm optymalizacji statycznej, umożliwiający poszukiwanie minimum funkcji wielu zmiennych przy ograniczeniach równościowych i nierównościowych (zarówno liniowych jak i nieliniowych). Optymalizacja stanie się jedną z wielu różnorodnych usług dostępnych w systemie INSTEPRO, którego nadrzędnym celem jest realizacja zaawansowanych algorytmów sterowania procesami (APC – *Advanced Process Control*). W systemach automatyki algorytm minimalizacji może być przydatny do realizacji następujących zadań:

- optymalizacja parametryczna nastaw regulatorów (na podstawie liniowego lub nieliniowego modelu dynamicznego procesu).
- optymalizacja punktu pracy instalacji (na podstawie liniowego lub nieliniowego modelu statycznego).
- identyfikacja modeli statycznych i dynamicznych (liniowych i nieliniowych).
- optymalizacja przebiegu sterowania w regulacji predykcyjnej (MPC – *Model Predictive Control*)

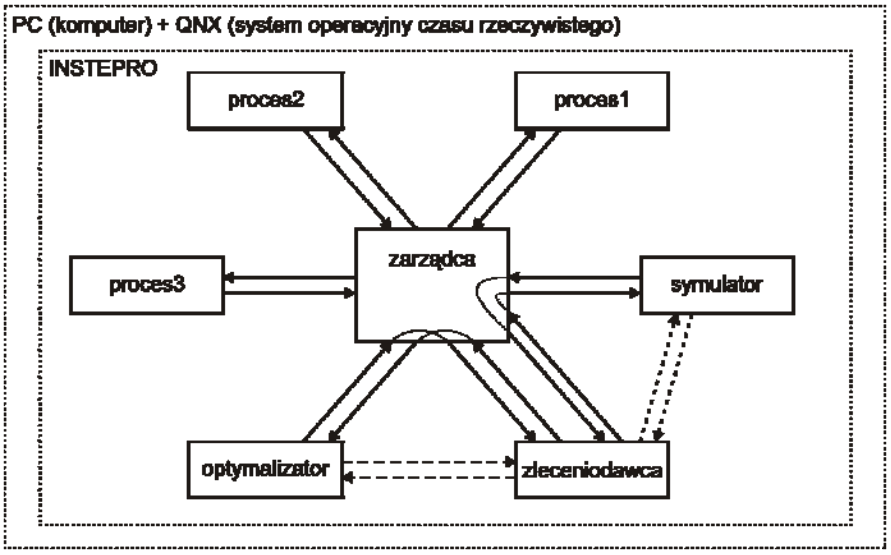
Ponieważ INSTEPRO powinien mieć budowę modułową, a będzie pracował na komputerze klasy PC pod kontrolą systemu operacyjnego czasu rzeczywistego (RTOS – *Real-Time Operating System*) QNX, optymalizacja zostanie zaimplementowana jako oddzielny proces tego systemu. Jego robocza nazwa to **optymalizator**. Będzie się on komunikował z **zarządcą** (jednym z podstawowych planowanych procesów pakietu INSTEPRO) oraz ewentualnie również z innymi procesami (patrz rysunek 1) za pośrednictwem zasadniczego mechanizmu komunikacji międzyprocesowej (IPC – *Inter-Process Communication*) dostępnego w systemie QNX, opierającego się na przesyłaniu komunikatów i odpowiadaniu na nie (Send → Receive → Raplay) oraz wysyłaniu impulsów (nie wymagających odpowiedzi).



Rys. 1. Modułowa budowa systemu INSTEPRO.

Praca systemu ma się opierać na wymianie zleceń, którą zawiaduje **zarządca**. Każdy proces wchodzący w skład systemu INSTEPRO może za pośrednictwem **zarządcy** zlecić procesowi **optymalizator** rozwiązanie zadania minimalizacji. Wyniki odsyłane będą bądź za pośrednictwem **zarządcy** (strzałka z linią ciągłą na rysunku 1), bądź bezpośrednio do moduły zlecającego obliczenia (linia przerywana). W tym drugim przypadku konieczne jest wcześniejsze nawiązanie (z pomocą **zarządcy**) bezpośredniej komunikacji między procesem zlecającym i wykonującym zlecenie. W chwili obecnej nie jest jeszcze przesądzone czy w ostatecznym systemie będzie zaimplementowana możliwość bezpośredniej komunikacji zleceniodawca-wykonawca, czy też jedynym sposobem przekazywania danych będzie korzystanie z pośrednictwa **zarządcy**.

W przypadku realizacji przed moduł optymalizatora zadania optymalizacji parametrycznej nastaw regulatora, konieczne jest wielokrotne wykonywanie symulacji w oparciu o model symulacyjny procesu. Zadanie symulacji w systemie INSTEPRO ma realizować odrębny proces, zwany **symulatorem**. Stąd rozwiązanie tego zadania wymaga współdziałania ze sobą kilku procesów: zleceniodawcy, **zarządcy**, **optymalizatora** i **symulatora**. Wszystkie niezbędne wymiany danych (wysyłanie zleceń i odsyłanie wyników) realizowane są za pośrednictwem mechanizmów IPC (rysunek 2).



Rys. 2. Optymalizacja parametryczna.

By możliwe było wykonywanie przez **optymalizator** zleceń od różnych procesów, konieczne jest ustalenie formatu, w jakim formułowane będzie zadanie optymalizacji oraz w jakim przesyłane będą dane wejściowe oraz wyniki. Innymi słowy, należy uzgodnić postać interfejsu między zleceniodawcami a procesem **optymalizator**. Głównym jego elementem będzie definicja typu strukturalnego (w języku C), którego zmienna przechowywać będzie na kolejnych swych polach dane stanowiące sformułowanie zadania minimalizacji oraz zestaw parametrów (opcji) dla algorytmu optymalizacji, a po wykonaniu obliczeń – także ich wynik. Interfejs ten powinien pozostać zasadniczo niezmieniony w trakcie budowy systemu INSTEPRO (lub zmieniać się bardzo rzadko, tylko wtedy, gdy jest to absolutnie niezbędne), natomiast sama implementacja algorytmów minimalizacji może być modyfikowana w razie potrzeby w kolejnych wersjach programu (na przykład w celu usunięcia usterki, usprawnienia obliczeń, dodania nowego algorytmu minimalizacji). Kod źródłowy programu zapisany zostanie w języku C (proceduralnie).

W chwili obecnej nie jest jeszcze oprogramowana komunikacja między **zarządcą** a procesami z użyciem mechanizmów IPC. W związku z tym na etapie testów skompilowany kod optymalizacji będzie łączony statycznie („linkowany") ze skompilowanym kodem zleceniodawców. Alternatywną metodą jest scalanie kodu źródłowego optymalizacji z kodem źródłowym zleceniodawców (na przykład z użyciem dyrektywy preprocesora *#include*). Projektanci dołożą jednak wszelkich starań, by nie korzystać z tej drugiej metody („inkludowanie" kodu C), ponieważ jest ona uciążliwa i łatwo prowadzi do błędów.

Algorytm optymalizacji jest zaimplementowany w kodzie procesu **optymalizator**. Z kolei problem (zadanie) optymalizacji jest (w ogólnym przypadku) zdefiniowane w kodzie procesu zlecającego obliczenia (jako funkcje języka C, obliczające wartość funkcji celu oraz funkcji ograniczeń dla podanych wartości zmiennych decyzyjnych). W związku z tym niemożliwe jest wyznaczenie wartości funkcji celu lub ograniczeń we wnętrzu modułu **optymalizatora** – zamiast tego polecenie wykonania tych obliczeń (wraz z niezbędnymi danymi) musi być przesłane do zleceniodawcy, a wyniki odesłane z powrotem. Ponieważ poszukiwanie minimum jest procesem iteracyjnym, zachodzi konieczność wielokrotnego przesyłania wartości zmiennych decyzyjnych z **optymalizatora** do zleceniodawcy oraz odsyłania z powrotem wyników obliczeń wartości funkcji celu i ograniczeń dla przekazanych zmiennych. W docelowej wersji systemu dane te będą przesyłane z użyciem mechanizmów IPC. Obecnie, gdy kody zleceniodawcy i **optymalizatora** są łączone (linkowane) ze sobą statycznie, wystarczy przekazanie do algorytmu optymalizacji wskaźników do odpowiednich funkcji zleceniodawcy (gdyż funkcje zleceniodawcy i optymalizatora wykonują się w obrębie wspólnej przestrzeni adresowej procesu, w ramach tego samego kontekstu).

Na wstępnym etapie tworzenia kodu procesu **optymalizator** nie przewiduje się jeszcze współpracy ze zleceniodawcami. Dlatego na samym początku działanie algorytmu będzie testowane na wybranych zadaniach testowych, zapisanych również jako kod w języku C.

Dane wejściowe i wyjściowe algorytmu

Przed rozpoczęciem obliczeń zleceniodawca definiuje zadanie optymalizacji, określając:

- rozmiary problemu (liczba zmiennych decyzyjnych n_x, liczba nieliniowych (w ogólności) ograniczeń nierównościowych n_j i równościowych n_k, liczba liniowych ograniczeń nierównościowych n_{i1} i równościowych n_{k1})
- nieliniową (w ogólnym przypadku) funkcję celu $F : R^n \to R$ lub też – gdy funkcja celu jest liniowa – wektor $C \in R^{n_x}$ współczynników kombinacji liniowej definiującej funkcję celu $F(X) = C^{\mathrm{T}} X$ (wektor kosztu), $X = [X_1, X_2, ..X_i ... X_{n_x}]^T$
- funkcje $G_j : R^n \to R$ nieliniowych (w ogólności) ograniczeń nierównościowych przyjmujących postać $G_j(X) \le 0$, gdzie $j \in \{1, 2, \ldots, n_j\}$
- funkcje $H_k : R^n \to R$ nieliniowych (w ogólności) ograniczeń równościowych przyjmujących postać $H_k(X) = 0$, gdzie $k \in \{1, 2, \ldots, n_k\}$
- wektory A_j oraz skalary b_j definiujące liniowe ograniczenia nierównościowe postaci $A_j^T X \le b_j$, $j \in \{1, 2, \ldots, n_{j1}\}$
- wektory A_k oraz skalary b_k definiujące linowe ograniczenia równościowe postaci $A_k^T X = b_k$, $k \in \{1, 2, \ldots, n_{k1}\}$
- dolne i górne ograniczenia wszystkich zmiennych decyzyjnych (nierównościowe ograniczenia proste postaci $\underline{X}_i \le X_i \le \overline{X}_i$), $i \in \{1, 2, \ldots, n_x\}$
- punkt startowy X^0

Nie wszystkie dane są wymagane. Dla prawidłowego zdefiniowani zadania wystarczy podać rozmiary problemu $(n_x, n_j, n_k, n_{j1}, n_{k1})$ i nierównościowe ograniczenia proste $(\underline{X}_i, \overline{X}_i)$. Nie jest konieczne definiowanie ograniczeń nierównościowych $(G_j(X), A_j, b_j)$ ani równościowych $(H_k(X), A_k, b_k)$. Użytkownik nie musi też podawać punktu startowego X^0. Również funkcja celu może pozostać niezdefiniowana. W takim przypadku **optymalizator** zakłada, że zadanie polega na znalezieniu dowolnego rozwiązania dopuszczalnego.

Oprócz definicji problemu, użytkownik powinien zadać także parametry dla algorytmu optymalizacji, decydujące o jego działaniu. Nie wszystkie parametry są obowiązkowe – niektóre z nich posiadają wartości domyślne, na które użytkownik może przystać. Zerowe wartości dla parametrów „epsilon" wykorzystywanych w warunkach stopu oznaczają rezygnację ze sprawdzania tych warunków. Znaczenie poszczególnych opcji (parametrów) wyjaśniono w tabeli 2.

Jako wynik obliczeń zwracany jest znaleziony punkt X^* (wstawiony w strukturze z danymi i parametrami w miejsce punktu startowego X^0), stanowiący przybliżenie rozwiązania optymalnego, oraz podawana jest wartość funkcji celu $F^* = F(X^*)$ w tym punkcie. Oprócz tego użytkownik otrzymuje komunikat o przyczynie zakończenia lub przerwania obliczeń (jak na przykład osiągnięcie maksymalnej liczby iteracji lub niepoprawność danych wejściowych). Jest on zakodowany jako liczba całkowita, ale też zwracany jako łańcuch znaków. Na podstawie zwróconego kodu (lub tekstu) zleceniodawca decyduje czy otrzymany wynik jest dla niego wiarygodny i użyteczny.

Wstępna obróbka danych wejściowych

Przed rozpoczęciem właściwej minimalizacji funkcji celu, program **optymalizator** przeprowadza następujące czynności wstępne:

1. Sprawdzenie poprawności danych wejściowych.
2. Ustalenie, które zmienne decyzyjne można potraktować jako parametry.
3. Przeskalowanie (normalizacja) zmiennych decyzyjnych.

Zleceniodawca nie musi definiować wszystkich parametrów algorytmu. Dla wielu z nich istnieje możliwość przypisania wartości domyślnych. Użytkownikowi pozostaje wówczas zmodyfikowanie tylko tych parametrów, dla których wartości domyślne nie są właściwe lub nie są zdefiniowane.

Procedura sprawdzania poprawności danych dostarczonych przez zleceniodawcę polega na:

- ustaleniu czy zbiór określony przez nierównościowe ograniczenia proste nie jest pusty (czy dla każdej zmiennej decyzyjnej ograniczenie dolne jest mniejsze lub równe górnemu)
- jeśli użytkownik podał punkt startowy – przetestowaniu czy spełnia on nierównościowe ograniczenia proste (nie są natomiast testowane pozostałe ograniczenia)
- sprawdzenie czy wartości parametrów dla algorytmu minimalizacji są poprawne

Na etapie analizy danych zadania, jeszcze przed rozpoczęciem poszukiwania minimum, program wyszukuje te zmienne decyzyjne, dla których względna różnica ograniczenia górnego i dolnego jest mniejsza od zadanej wartości E_p („p" jak „*parameter*"). Zmienne te są następnie przenoszone ze zbioru zmiennych decyzyjnych do zbioru parametrów i przypisuje się im wartość równą średniej arytmetycznej ograniczeń: górnego i dolnego. W dalszych obliczeniach są one traktowane jako stałe.

Wszystkie zmienne decyzyjne podlegają przeskalowaniu do przedziału [0, 1], na podstawie informacji o ich dolnych i górnych ograniczeniach, według wzoru:

$$x_i = \xi_i(X_i) = \frac{X_i - \underline{X}_i}{\overline{X}_i - \underline{X}_i}$$

Przyjmuje się, że zleceniodawca musi podać skończone ograniczenia dolne i górne dla wszystkich zmiennych decyzyjnych. Dla przeskalowanych zmiennych decyzyjnych nowe ograniczenia górne i dolne są równe odpowiednio 1 i 0. Skalowanie zmiennych jest jednym z pierwszych kroków programu. Wszelkie dalsze obliczenia związane z poszukiwaniem minimum prowadzone są na zmiennych przeskalowanych. Dotyczy to także sprawdzania czy spełnione są kryteria stopu. W dalszym opisie oryginalne zmienne decyzyjne (oraz odpowiadające im ograniczenia górne i dolne oraz funkcję celu i funkcje ograniczeń równościowych i nierównościowych) oznaczono wielkimi literami, zaś zmienne przeskalowane – małymi. W trakcie obliczeń zachodzi konieczność znajdowania wartości oryginalnych zmiennych decyzyjnych na podstawie wartości zmiennych przeskalowanych. Służy do tego wzór:

$$X_i = \Xi_i(x_i) = \underline{X}_i + \left(\overline{X}_i - \underline{X}_i\right)x_i$$

By wyznaczyć wartość funkcji celu w punkcie x (zmienne przeskalowane: $x = [\, x_1, x_2, ..x_i ... x_{n_x} \,]^T$), należy skorzystać ze złożenia funkcji Ξ z F:

$$F(X) = F(\Xi(x)) = (F \circ \Xi)(x)$$

Podobnie postępuje się w przypadku funkcji ograniczeń nierównościowych i równościowych:

$$G_j(X) = G_j(\Xi(x)) = (G_j \circ \Xi)(x)$$

$$H_k(X) = H_k(\Xi(x)) = (H_k \circ \Xi)(x)$$

$$A_j^T X - b_j = A_j^T \Xi(x) - b_j$$

$$A_k^T X - b_k = A_k^T \Xi(x) - b_k$$

Algorytm optymalizacji

Zaimplementowany w module **optymalizator** algorytm poszukiwania minimum ma budowę hierarchiczną i przebieg iteracyjny. Algorytm najwyższego poziomu decyduje, które nierównościowe ograniczenia proste są uwzględniane w funkcji kary. W pierwszej iteracji do funkcji kary są włączane wyłącznie te ograniczenia proste, które jawnie wskazał użytkownik. W każdej kolejnej iteracji funkcja kary jest uzupełniana o te graniczenia proste, których nie spełnia punkt znaleziony w poprzedniej iteracji. Iteracje tego algorytmu są przerywane w momencie, gdy wszystkie ograniczenia proste są spełnione lub gdy w funkcji kary uwzględniono już wszystkie ograniczenia proste. Algorytm niższego poziomu realizuje metodę

zewnętrznej funkcji kary. Jego obliczenia są przerywane, jeśli spełnione są wszystkie ograniczenia (z zadanymi dokładnościami) lub też nie ma znaczącej poprawy wartości funkcji celu bądź znaczącej zmiany wektora zmiennych decyzyjnych w stosunku do poprzedniej iteracji albo gdy rozwiązanie oddaliło się nadmiernie od zbioru określonego ograniczeniami prostymi, albo osiągnięto maksymalną liczbę iteracji. Na jeszcze niższym poziomie ulokowany jest algorytm Davidona-Fletchera-Powella (DFP) lub najszybszego spadku (do wyboru przez użytkownika), którego zadaniem jest poszukiwanie minimum bez ograniczeń dla funkcji będącej sumą funkcji celu i funkcji kary. Warunki stopu w algorytmie DFP to brak istotnej poprawy wartości minimalizowanej funkcji, brak znaczącej zmiany wartości zmiennej decyzyjnej, bliska zeru norma gradientu minimalizowanej funkcji lub osiągnięcie maksymalnej dopuszczalnej liczby iteracji. Z kolei algorytm DFP korzysta z położonej najniżej w hierarchii jednowymiarowej metody poszukiwania minimum na kierunku, w której zastosowano aproksymację wielomianową lub złoty podział. Schemat blokowy algorytmu poszukującego minimum przedstawiono na rysunku 2, zaś w tabeli 1 zebrano warunki stopu algorytmów na poszczególnych poziomach hierarchii.

Tab. 1. Warunki stopu w algorytmach poszukiwania minimum

Lp.	Algorytm	Warunki stopu
1	Wybór nierównościowych ograniczeń prostych uwzględnianych w funkcji kary	Spełnienie (z zadaną dokładnością) wszystkich nierównościowych ograniczeń prostych lub uwzględnienie w funkcji kary wszystkich ograniczeń prostych.
2	Metoda zewnętrznej funkcji kary	Spełnienie (z zadaną dokładnością) wszystkich ograniczeń (równościowych i nierównościowych; w tym prostych wskazanych przez algorytm nadrzędny), brak znaczącej poprawy wartości funkcji celu, brak znaczącej zmiany wartości zmiennej decyzyjnej, nadmierne oddalenie się rozwiązania od zbioru wyznaczonego przez ograniczenia proste, osiągnięcie maksymalnej dopuszczalnej liczby iteracji.
2a	Metoda rozszerzonej, zewnętrznej funkcji kary	j.w.
3	Metoda zmiennej metryki DFP	Brak znaczącej poprawy wartości funkcji celu, brak znaczącej zmiany wartości zmiennej decyzyjnej, bliska zeru wartość normy gradientu minimalizowanej funkcji, osiągnięcie maksymalnej dopuszczalnej liczby iteracji.
4	Minimalizacja jednowymiarowa na kierunku	Użyty algorytm poszukiwania minimum na kierunku wykonuje w pierwszej iteracji krok o maksymalnej długości, a następnie cofa się coraz mniejszymi krokami. Proces ten jest przerywany, gdy wartość minimalizowanej funkcji przestaje maleć.

Nierównościowe ograniczenia proste

Nierównościowe ograniczenia proste można uwzględnić podczas obliczeń na dwa sposoby:

1. W algorytmie poszukiwania minimum na kierunku ograniczyć maksymalną wartość kroku, w razie potrzeby zrzutowawszy wcześniej kierunek poszukiwań na hiperpłaszczyzny wyznaczone przez wybrane ograniczenia proste.
2. Potraktować wybrane ograniczenia proste identycznie jak pozostałe ograniczenia nierównościowe, uwzględniając je w funkcji kary.

Sposób pierwszy

W pierwszym przypadku stosuje się dwuetapową procedurę, składającą się z ewentualnego rzutowania kierunku poszukiwań oraz ograniczenia wielkości kroku wzdłuż nowego kierunku. Wspomniane rzutowanie polega w praktyce na wyzerowaniu wybranych elementów wektora kierunku poszukiwań. Element *i*-ty wektora kierunku jest zerowany w przypadku, gdy spełnione są jednocześnie dwa warunki:

- składowa *i*-ta wektora przeskalowanych zmiennych decyzyjnych leży zbyt blisko ograniczenia dolnego (0) lub górnego (1) lub też je przekracza

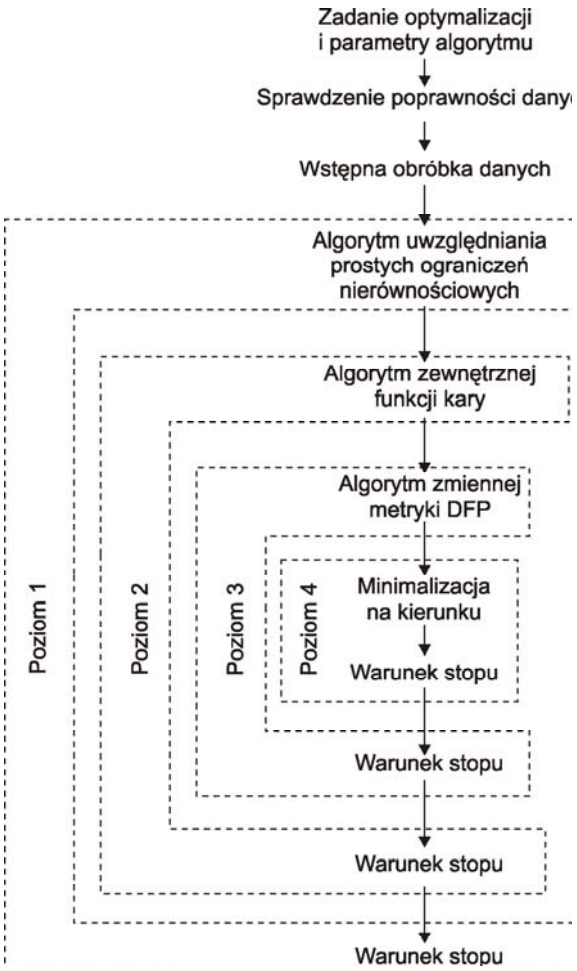
- odpowiadająca jej składowa wektora kierunku ma wartość dodatnią (w przypadku niebezpieczeństwa naruszenia górnego ograniczenia) lub ujemną (dla ograniczenia dolnego)

Następnie należy wyznaczyć maksymalny krok jaki można wykonać z punktu startowego wzdłuż kierunku poszukiwań, nie powodując przekroczenia ograniczeń. Obliczenia te można wykonać niezależnie dla każdej ze zmiennych decyzyjnych, a następnie wybrać najmniejszy z wyników. Pojedynczy wynik dla *i*-tej zmiennej decyzyjnej jest równy stosunkowi różnicy między ograniczeniem, które grozi naruszeniem, a *i*-tą składową punktu startowego, przez *i*-tą składową wektora kierunku. Oczywiście dzielenia nie wykonuje się, jeśli dzielnik ma wartość zerową, bo został wyzerowany na etapie rzutowania kierunku na ograniczenia.

Sposób drugi

Z kolei drugi z wymienionych sposobów uwzględniania ograniczeń prostych zakłada, że wybrane z nich będą włączane do funkcji kary, podobnie jak pozostałe ograniczenia nierównościowe i równościowe. W pierwszej iteracji obliczeń poszukuje się minimum wprowadzając do funkcji kary jedynie te ograniczenia proste, które jawnie wytypował i wskazał zleceniodawca. Jeśli otrzymany w wyniku minimalizacji punkt nie spełnia przynajmniej jednego z pozostałych ograniczeń prostych, obliczenia należy powtórzyć, dodawszy wcześniej wszystkie naruszone ograniczenia proste do funkcji kary. Tę procedurę należy kontynuować aż do momentu, gdy wszystkie ograniczenia proste będą spełnione lub uwzględnione w funkcji kary.

W zaimplementowanym algorytmie korzystającym z metody DFP zastosowano drugie z opisywanych rozwiązań (sukcesywne dodawanie ograniczeń prostych do funkcji kary), ponieważ testy pierwszego rozwiązania ujawniły nieprawidłowe działanie (brak zbieżności) całego algorytmu poszukiwania minimum.



Rys. 2. Schemat blokowy algorytmu minimalizacji.

Funkcja kary

Zastosowano klasyczną, kwadratową zewnętrzną funkcję kary. W funkcji kary każdemu ze składników przypisany jest indywidualny współczynnik (mnożnik).

$$min\left\{F(X)+\sum_{j=1}^{n_j} p_j \, max[0,G_j(X)]^2+\sum_{k=1}^{n_k} r_k (H_k(X))^2+\Phi_l+\Phi_p\right\}$$

$$\Phi_l=\sum_{j=1}^{n_{j1}} \overline{p}_j (max[0,(A_j{}^T X-b_j)])^2+\sum_{k=1}^{n_{k1}} \overline{r}_k (A_k{}^T X-b_k)^2$$

$$\underline{x}_i \le x_i \le \overline{x}_i),\ i \in \{1,2,\ldots,n_x\}$$

W trakcie obliczeń funkcja ta może być uzupełniana o składniki (Φ_p), odpowiadające aktywnym ograniczeniom prostym.

Przed rozpoczęciem obliczeń wszystkie współczynniki mają tę samą wartość startową (będącą parametrem, który może zadać zleceniodawca). Podczas kolejnych iteracji algorytmu funkcji kary, sukcesywnie zwiększane są tylko te współczynniki, które odpowiadają ograniczeniom naruszonym przez rozwiązanie otrzymane w poprzedzającej iteracji. Zwiększanie polega na mnożeniu przez stałą większą od jedności (będącą parametrem). Jeśli któryś ze współczynników przekroczy przyjętą maksymalną wartość graniczną (będącą parametrem), obliczenia są przerywane.

W przypadku algorytmu zewnętrznej funkcji kary może się zdarzyć, że przy oddalaniu się rozwiązania od zbioru dopuszczalnych rozwiązań tempo w jakim maleje funkcja celu przekroczy szybkość narastania funkcji kary. Wówczas funkcja kary nie spełnia swego zadania, a wobec tego algorytm optymalizacji może działać nieprawidłowo, znajdując rozwiązanie daleko poza zbiorem dopuszczalności. Dla zabezpieczenia przed takim przypadkiem w algorytmie wprowadzono warunek stopu, sprawdzający wielkość przekroczenia ograniczeń prostych i przerywający obliczenia, jeśli jest ono nadmierne:

$$x_i^m \ge 1+M_x \quad \vee \quad x_i^m \le M_x, \quad i=1,2,\ldots,n_x$$

Dwa inne, typowe dla metod iteracyjnych warunki stopu, wykorzystywane przez opisywany algorytm, są następujące:

$$\left|F\left(x^{m+1}\right)-F\left(x^m\right)\right| < \varepsilon_f \left|F\left(x^m\right)\right|$$

$$\left\|x^{m+1}-x^m\right\| < E_x$$

Użytkownik może wyłączyć sprawdzanie każdego z dwóch podanych wyżej warunków, przypisując 0 odpowiednio do parametru ε_f lub E_x. Dodatkowym warunkiem stopu jest **spełnienie** (z zadaną dokładnością) wszystkich ograniczeń równościowych i nierównościowych (w tym prostych):

$$-E_b \le x_i^m \le 1+E_b \tag{1}$$

$$G_j(x^m) \le E_g, \quad j=1,2,\ldots,n_j \tag{2}$$

$$-E_h \le H_k(x^m) \le E_h, \quad k=1,2,\ldots,n_k \tag{3}$$

$$A_j \, \Xi(x^m)-b_j \le E_g, \quad j=1,2,\ldots,n_j \tag{4}$$

$$-E_h \le A_k \, \Xi(x^m)-b_k \le E_h, \quad j=1,2,\ldots,n_k \tag{5}$$

Jeszcze jeden warunek stopu jest związany z przekroczeniem dopuszczalnej liczby iteracji N_{PF} („PF" jak „*Penalty Function*").

Rozszerzona funkcja kary

Po wprowadzeniu mnożników Lagrange'a ($\lambda_j, \mu_k, \overline{\lambda}_j, \overline{\mu}_k$), odpowiadających ograniczeniom nierównościowym (λ_j) $j \in \{1,2,\ldots,n_j\}$ i równościowym (μ_k), $k \in \{1,2,\ldots,n_k\}$, po wprowadzeniu zmiennych osłabiających s_j otrzymujemy zmodyfikowaną funkcję celu:

$$min\left\{F(X)+\sum_{j=1}^{n_j} p_j(G_j(X)+s_j^2))^2+\sum_{k=1}^{n_k} r_k(H_k(X))^2+\sum_{j=1}^{n_j}\lambda_j(G_j(X)+s_j^2)+\sum_{k=1}^{n_k}\mu_k H_k(X)+\Phi_l+\Phi_p\right\}$$

W trakcie obliczeń funkcja ta może być uzupełniana o składniki (Φ_p)odpowiadające aktywnym ograniczeniom prostym.

Po eliminacji zmiennych osłabiających: $s_i^2 = -[G_j(X)+\frac{\lambda_j}{2p_j})]$, otrzymamy:

$$min\left\{F(X)+\sum_{j=1}^{n_j} p_j(max[0,(G_j(X)+\frac{\lambda_j}{2p_j})])^2-\sum_{j=1}^{n_j}\frac{\lambda_j^2}{4p_j}+\sum_{k=1}^{n_k} r_k(H_k(X))^2+\sum_{k=1}^{n_k}\mu_k H_k(X)+\Phi_l+\Phi_p\right\}$$

gdzie

$$\Phi_l=\sum_{j=1}^{n_{j1}}\overline{p}_j(max[0,(A_j{}^T X-b_j+\frac{\overline{\lambda}_j}{2\overline{p}_j})])^2-\sum_{j=1}^{n_j}\frac{\overline{\lambda}_j^2}{4\overline{p}_j}+\sum_{k=1}^{n_{k1}}\overline{r}_k(A_k{}^T X-b_k)^2+\sum_{k=1}^{n_{kl}}\overline{\mu}_k(A_k{}^T X-b_k)$$

$\underline{x}_i \le x_i \le \overline{x}_i)$, $i \in \{1,2,\ldots,n_x\}$

Algorytm działa jak następuje (A. Bhati, „Practical Optimization methods", Springer, 2000, Kręglewski i inn. „Metody Optymalizacji w jezyku Fortran", PWN 1984):
Dla każdego ograniczenia sprawdzamy:

$$\left|V(x^m)-V(x^l)\right| \le 0{,}25 \quad \text{(ogólnie } E_p \text{)} \qquad (6)$$

gdzie $V(.)$ - wskaźnik zmiany naruszenia ograniczenia w wyniku poprzedniej iteracji DFP, liczony według (1) –(5), (*n* – numer kolejnej modyfikacji mnożników Lagrange'a, *m* – numer kolejnego poszukiwania DFP)

- gdy ograniczenie nie jest istotnie naruszone - (6) jest spełnione - mnożniki Lagrange'a ($\lambda_j, \mu_k, \overline{\lambda}_j, \overline{\mu}_k$) podlegają (odpowiednio) następującej modyfikacji:

$$\lambda_j^{n+1} = \lambda_j^n + 2p_j^m(max[0,(G_j(X^m)+\frac{\lambda_j^n}{2p_j^m})])^2$$

$$\mu_k^{n+1} = \mu_k^n + 2r_k^m H_k(X^m)$$

$$\overline{\lambda}_j^{n+1} = \overline{\lambda}_j^n + 2\overline{p}_j^m(max[0,(A_j{}^T X^m - b_j+\frac{\overline{\lambda}_j^n}{2\overline{p}_j^m})])^2$$

$$\overline{\mu}_k^{n+1} = \overline{\mu}_k^n + 2\overline{r}_k^m(A_k{}^T X^m - b_k)$$

$$l = l+1$$

- w przeciwnym przypadku: **(6) nie spełnione -** współczynniki kary ($p_j, r_k, \overline{p}_j, \overline{r}_k,$) modyfikujemy według ogólnej zasady,

- zawsze: $m = m + 1$ i powrót do DFP

Idea metody rozszerzonej funkcji kary polega na poszukiwaniu punktu siodłowego tzw. rozszerzonej funkcji Lagrange'a. Modyfikacje współczynników kary można interpretować jako maksymalizację funkcji dualnej poprzez wykonanie kroku w kierunku gradientu tej funkcji.

Algorytm DFP

Kod źródłowy zastosowanej metody DFP (*Davidon–Fletcher–Powell*) pochodzi z płyty CD dołączonej do książki *Numerical Recipes. The Art of Scientific Computing* (William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery; Cambridge University Press; 2007). Jednym z warunków stopu w tej metodzie jest przekroczenie maksymalnej dopuszczalnej liczby iteracji N_{DFP}.

Minimalizacja na prostej

Kod źródłowy zastosowanej metody jednowymiarowego poszukiwania minimum na kierunku pochodzi z płyty CD dołączonej do książki *Numerical Recipes. The Art of Scientific Computing*. Wykorzystuje ona aproksymację wielomianową, a jej iteracje przerywane są po spełnianiu przynajmniej jednego z dwóch warunków stopu. Pierwszy z nich związany jest z przekroczeniem maksymalnej dopuszczalnej liczby iteracji N_{LS} („LS" jak „*Line Search*"). Drugi warunek stopu to brak poprawy wartości funkcji celu.

Punkt startowy

Zleceniodawca może podać punkt startowy dla zadania minimalizacji lub też może zażądać, by algorytm minimalizacji samodzielnie znalazł taki punkt. W pierwszym przypadku, przed rozpoczęciem minimalizacji program sprawdza czy podany przez zleceniodawcę punkt startowy spełnia wszystkie ograniczenia proste. Jeśli choć jedno z nich jest naruszone, zgłaszany jest błąd i obliczenia są przerywane. W drugim przypadku program sam poszukuje punktu startowego. Stosowana jest przy tym następująca procedura. Początkowo wszystkim przeskalowanym zmiennym decyzyjnym x_i przypisywana jest wartość 0.5. Następnie uruchamiany jest algorytm poszukiwania minimum metodą funkcji kary, przy założeniu że funkcja celu jest identycznościowo równa zeru w całej przestrzeni R^n. Dowolne optymalne rozwiązanie tego problemu jest zarazem rozwiązaniem dopuszczalnym zadania oryginalnego i tym samym dobrym kandydatem na punk startowy.

Inna metoda rozwiązania problemu punktu startowego, którą brano pod uwagę, lecz ostatecznie nie zaimplementowano, polega na relaksacji (osłabieniu) ograniczeń skutkującej rozszerzeniem zbioru dopuszczalnych rozwiązań. Początkowo zbiór ten należy rozszerzać aż do momentu, gdy przyjęty punkt startowy stanie się dopuszczalny. Następnie w każdej kolejnej iteracji konsekwentnie zacieśnia się ograniczenia, aż do osiągnięcia pierwotnego zbioru rozwiązań dopuszczalnych.

Kod w języku C

Na moduł **optymalizatora** składa się szereg plików źródłowych języka C, implementujących różne algorytmy minimalizacji. W kilku innych plikach zapisany jest kod wykorzystywany do testowania algorytmów optymalizacji (definicje zadań, algorytm symulacji Rungego-Kutty, kod umożliwiający wywołanie obliczeń z linii poleceń). Poniżej wymieniono i scharakteryzowano poszczególne pliki:

- **zfk.h** – algorytm zewnętrznej funkcji kary
- **mns.h** – metoda najszybszego spadku
- **quasinewton.h** – metoda zmiennej metryki (DFP – Davidow-Fletcher-Powell)
- **zmns.h** – zmodyfikowana metoda najszybszego spadku
- **min.c / min.h** - funkcje i struktury interfejsowe dla użytkownika (MinState – struktura przechowująca definicje zadania i parametry algorytmu, a także wyniki obliczeń; minDefaults – funkcja przypisująca domyślnych wartości polom struktury typu MinState; minMain – główna funkcja interfejsowa, odpowiedzialna za przeprowadzenie obliczeń z użyciem wybranego algorytmu minimalizacji; realizuje również procedury sprawdzające poprawność danych dostarczonych przez użytkownika, przeprowadza kwalifikacja niektórych zmiennych decyzyjnych jako parametry,

odpowiada za skalowanie zmiennych; wartość zwracana przez funkcję minMain to kod informujący o przyczynie zakończenia obliczeń (błąd lub powodzenie)

- **rk4.c / rk4.h** – algorytm Rungego-Kutty wykorzystywany do symulacji systemów dynamicznych (w zadaniach optymalizacji parametrycznej nastaw regulatora przy wskaźniku jakości wyznaczanym na podstawie przebiegów czasowych sygnałów w układzie regulacji)
- **main.c** - plik używany do testowania implementowanych algorytmów (nie będzie wchodził w skład docelowego pakietu, podobnie jak kilka innych plików)
- **_*pid.c, _*pi.c** - zadanie testowe na dobór nastaw regulatora PI lub PID w zamkniętym układzie regulacji automatycznej
- **example.h** – pomocniczy kod dla preprocesora C, włączany do plików źródłowych zawierających definicje niektórych zadań testowych
- **diag.h** – definicje makr wykorzystywanych do wyprowadzania komunikatów diagnostycznych

Sposób wywołania algorytmu optymalizacji

W obecnej, rozwojowej wersji **optymalizatora**, skorzystanie z niego wymaga statycznego połączenia (linkowania) jego plików obiektowych z plikami obiektowymi kodu zleceniodawcy. Zleceniodawca powinien również włączyć do jednego ze swoich plików źródłowych plik nagłówkowy **min.h** optymalizatora. Plik ten zawiera między innymi definicje typu strukturalnego **MinState** oraz prototypy funkcji **minDefaults** i **minMain**, stanowiących interfejs umożliwiający zleceniodawcy skorzystanie z funkcjonalności **optymalizatora**. Poniżej podano wskazówki, jak z tego interfejsu skorzystać.

1. Zleceniodawca tworzy zmienną strukturalną typu **MinState**, w której przechowywana będzie definicja zadania oraz parametry dla algorytmu optymalizacji, a po zakończeniu obliczeń – ich wyniki:
   ```
   MinState zadanie;
   ```
2. Zleceniodawca deklaruje zmienną całkowitą, w której po zakończeniu obliczeń zapisany zostanie kod błędu / status wyniku:
   ```
   int status;
   ```
3. W celu wskazania z której wersji interfejsu korzysta, zleceniodawca przypisuje polu **version** utworzonej struktury typu **MinState** stosowny numer wersji:
   ```
   zadanie.version=1;
   ```
4. By przypisać parametrom algorytmu optymalizacji wartości domyślne, należy wywołać funkcję **minDefaults**, przekazując jej jako argument wskaźnik do struktury typu **MinState**:
   ```
   minDefaults(&zadanie);
   ```
5. Zleceniodawca definiuje problem optymalizacji oraz ustala parametry (opcje), przypisując odpowiednie wartości polom struktury typu **MinState**:
   ```
   zadanie.xxx=yyy;
   ```
   ...
   Wykaz wszystkich pól struktury znajduje się w tabeli 2.
6. Użytkownik zarządza wykonanie obliczeń, wywołując funkcję **minState**:
   ```
   status=minMain(&zadanie);
   ```
7. Użytkownik sprawdza status zakończenia obliczeń / kod błędu zwrócony przez funkcję **minState** i korzysta z wyników zapisanych w strukturze typu **MinState**.

Tab. 2. Pola struktury typu MinState.

Numer kolejny	Oznaczenie	Wymagane przypisanie	Domyślnie po zainicjowaniu	Dana wejściowa / wyjściowa	Opis
Pola opisujące problem optymalizacji					
1	n	TAK		wejściowy	Liczba zmiennych decyzyjnych
2	useStart	TAK		wejściowy	Określa, czy algorytm ma użyć punktu startowego zapisanego w zmiennej *x* (wartość inna niż 0). W przeciwnym przypadku zostaje podjęta próba znalezienia punktu startowego. Wartość zmiennej *x* jest wówczas ignorowana i może mieć początkowo dowolne wartości.
3	x	TAK		wejściowy wyjściowy	Na wejściu zawiera wektor startowy (jeżeli *useStart* != 0) lub wektor z dowolnymi danymi o rozmiarze *n*. Na wyjściu zawiera znaleziony punkt optymalny. Nawet jeśli użytkownik nie zamierza podawać punktu startowego dla procedury poszukiwania minimum, i tak musi zaalokować tablicę zmiennych typu *double* o rozmiarze *n* i przypisać jej wskaźnik do pola *x*. W zmiennej tej zwracany jest bowiem wynik obliczeń, a algorytm minimalizacji oczekuje, że to użytkownik zaalokuje pamięć na ten wynik.
4	f	NIE		wyjściowy	Po zakończeniu algorytmu zawiera wartość funkcji celu w punkcie *x*.
5	varMin	TAK		wejściowy	Wektor o długości *n* zawierający dolne ograniczenia wszystkich zmiennych decyzyjnych.
6	varMax	TAK		wyjściowy	Wektor o długości *n* zawierający górne ograniczenia wszystkich zmiennych decyzyjnych. Jeżeli varMax[i] == varMin[i] (z dokładnością *epsslb*) to zmienna decyzyjna x[i] zostanie potraktowana jako parametr i nie będzie zmieniana w trakcie optymalizacji.
7	funcPre	NIE	NULL	wejściowy	Funkcja wywoływana każdorazowo przed funkcją celu lub funkcją ograniczeń. Jej zadaniem jest wstępne przygotowanie danych dla dwóch pozostałych funkcji i umieszczenie ich w pomocniczym buforze. Dane te obliczane są na podstawie podanego wektora zmiennych decyzyjnych. Przykład zastosowania: w przypadku zadania optymalizacji parametrycznej opisywana funkcja może przeprowadzać symulację pracy układu regulacji i umieszczać w buforze uzyskane przebiegi czasowe sygnałów. Dopiero na ich podstawie funkcja celu wyznacza wartość przyjętego całkowego (lub innego) wskaźnika jakości.
8	userBuffer	NIE	NULL	wejściowy	Wskaźnik bufora alokowanego przez użytkownika, przeznaczonego na wyniki obliczeń prowadzonych przez funkcję „funcPre", a wykorzystywane w funkcji celu i ograniczeń.
9	funcMin	NIE	NULL	wejściowy	Wskaźnik do funkcji celu.
10	ng	NIE	0	wejściowy	Liczba ograniczeń nierównościowych zapisanych w funkcji *funcOgr*.

11	nh	NIE	0	wejściowy	Liczba ograniczeń równościowych zapisanych w funkcji *funcOgr*.
12	funcOgr	NIE	NULL	wejściowy	Wskaźnik do pojedynczej funkcji ograniczeń, która oblicza wartości *ng* ograniczeń nierównościowych i *nh* ograniczeń równościowych.
13	A	NIE	NULL	wejściowy	Macierz ograniczeń liniowych. Jest ona zapisana jako wektor utworzony z kolejnych wierszy macierzy.
14	b	NIE	NULL	wejściowy	Wektor wyrazów wolnych ograniczeń liniowych. Macierz A i wektor b definiują ograniczenia liniowe zarówno nierównościowe jak i równościowe. Rozmiar wektora b wynosi $lng+lnh$.
15	lng	NIE	0	wejściowy	Liczba liniowych ograniczeń nierównościowych, opisanych macierzą A i wektorem b.
16	lnh	NIE	0	wejściowy	Liczba liniowych ograniczeń równościowych, opisanych macierzą A i wektorem b.
17	ud	NIE	NULL	wejściowy	Wskaźnik do dowolnych danych dostarczonych przez użytkownika, które będą przekazywane przy każdym wywołaniu do funkcji wstępnej, celu i ograniczeń. Organizacja i rozmiar danych mogą być dowolne. Ich podanie nie jest wymagane.
18	method	NIE	METHOD _ZFK_ DFP	wejściowy	Określa numer zastosowanego algorytmu optymalizacji. Numery algorytmów określone są przez definicje postaci *MIN_METHOD_**.
19	dfpitmax	NIE	400	wejściowy	Maksymalna liczna iteracji w metodzie DFP.
20	eps	NIE	$\approx 2{,}2\cdot 10^{-16}$	wejściowy	Precyzja wartości zmiennoprzecinkowej (używana w DFP).
21	fdeps	NIE	$\sqrt{eps}$	wejściowy	Krok używany podczas numerycznej aproksymacji pochodnej funkcji (mianownik ilorazu różnicowego).
22	tolx	NIE	$eps\cdot 10^4$	wejściowy	Parametr używany w warunku stopu metody DFP: $\max\left(\frac{\lvert x-x_{-1}\rvert}{\max(\lvert x\rvert,1)}\right)<tolx$
23	gtol	NIE	10^{-6}	wejściowy	Dokładność zerowania się składowych gradientu w warunku stopu metody DFP.
24	sbtol	NIE	10^{-6}	wejściowy	Graniczna odległość między punktem startowym w algorytmie poszukiwania na kierunku a brzegiem zbioru „kostkowego" określonego ograniczeniami prostymi, przy której wdraża się procedurę rzutowania gradientu na ograniczenia.
25	stpmx	NIE	0.4	wejściowy	Maksymalna długość kroku, jaką może wykonać procedura szukania na kierunku wywoływana przez metodę DFP.
26	k0	NIE	0.1	wejściowy	Początkowa wartość wszystkich współczynników kary w metodzie zewnętrznej funkcji kary.
27	ni1	NIE	10^{-6}	wejściowy	Dokładność spełnienia ograniczeń nierównościowych, wykorzystywana w warunkach stopu.
28	ni2	NIE	10^{-6}	wejściowy	Dokładność spełnienia ograniczeń równościowych, wykorzystywana w warunkach stopu.
29	km	NIE	10^6	wejściowy	Maksymalna wartość współczynnika kary.
30	itmax	NIE	60	wejściowy	Maksymalna liczba iteracji metody zewnętrznej funkcji kary.

31	eps1	NIE	10^{-6}	wejściowy	Współczynnik używany w warunku stopu metody zewnętrznej funkcji kary: $\left\|f(x)-f(x_{-1})\right\| < eps_1 \cdot \left\|f(x)\right\|$. Jeżeli *eps1* = 0, to warunek nie jest aktywny i nie będzie sprawdzany – nigdy nie zadziała.
32	eps2	NIE	10^{-6}	wejściowy	Współczynnik używany w warunku stopu metody zewnętrznej funkcji kary: $\left\|x-x_{-1}\right\| < eps_2$ Jeżeli *eps2* = 0, to warunek nie jest aktywny i nie będzie sprawdzany – nigdy nie zadziała.
33	epssb	NIE	10^{-5}	wejściowy	Maksymalna długość przedziału zdefiniowanego ograniczeniami prostymi dla zmiennej x_i, przy którym zmienna ta jest traktowana jako parametr.
Dodatkowe parametry					
34	version	TAK		wejściowy	Określa wersję struktury przechowującej parametry algorytmu i sformułowanie problemu. Obecna wersja ma numer 1. Temu polu należy przypisać odpowiednią wartość (obecnie 1) przed wywołaniem funkcji *minDefaults*. Numer wersji będzie zwiększany o 1 za każdym razem, gdy w wyniku modyfikacji kodu w strukturze przybędą nowe pola.
35	diagLevel	NIE	0	wejściowy	Określa jak wiele informacji diagnostycznych ma być kierowanych do standardowego wyjścia *stdout*. 0 oznacza brak jakichkolwiek informacji, 6 – wszystkie możliwe informacje.
36	info	NIE		wyjściowy	Po zakończeni działania algorytmu zawiera podsumowujący komunikat diagnostyczny, informujący o przyczynie zakończenia obliczeń, w formie tekstowej.

Uwagi końcowe

Programując algorytm minimalizacji zdecydowano się na użycie języka C (wersja proceduralna), ponieważ większość istniejących modułów systemu INSTEPRO jest zakodowana w tym właśnie języku, a w systemie operacyjnym QNX dostępny jest kompilator i inne narzędzia dla tego języka.

Część użytego kodu źródłowego pochodzi z płyty CD dołączonej do książki *Numerical Recipes. The Art of Scientific Computing*. Kod ten przepisano z oryginalnej wersji w języku C++ do postaci wykorzystującej jedynie konstrukcje języka C.

W testowej wersji programu w wielu miejscach algorytmu umieszczono instrukcje wyprowadzające na standardowe wyjście (*stdout*) informacje diagnostyczne, przydatne na etapie uruchamiania („odpluskwiania") programu. O ilości wyprowadzanych informacji (inaczej – o „gadatliwości" programu) decyduje parametr całkowitoliczbowy, który można podać jako argument wywołania programu w linii komend. Wartość 0 wstrzymuje wyprowadzanie komunikatów diagnostycznych, zaś 7 zapewnia wypisywanie najobszerniejszych informacji.

W pierwszych wersjach testowych programu przyjęto, że problem optymalizacji zapisany będzie w pliku tekstowym (zawierającym funkcję celu, funkcje nieliniowych (w ogólności) ograniczeń równościowych i nierównościowych, nierównościowe ograniczenia proste oraz punkt startowy). Stwarzało to konieczność napisania funkcji analizujących wzory zapisane w takim pliku i tłumaczących je na postać dogodną do dalszych obliczeń numerycznych (zakodowany ciąg operacji, które należy kolejno wykonać na argumentach). Później jednak zrezygnowano z tego rozwiązania. Obecnie testowe zadania optymalizacji należy zakodować w języku C, skompilować i dołączyć („dolinkować") statycznie do skompilowanego kodu programu optymalizacji. Dla porządku przyjęto, że każdemu zadaniu odpowiada oddzielny plik źródłowy. Po uruchomieniu programu użytkownik otrzymuje listę dostępnych zadań i wybiera jedno z nich. W przyszłości użytkownik zadecyduje również o metodzie która ma być wykorzystana do rozwiązania zadania. Wyniki obliczeń wyprowadzane są na ekran (a dokładniej – na standardowe wyjście *stdout*).

Podczas minimalizacji numerycznej wielokrotnie obliczane są wartości funkcji celu oraz funkcji ograniczeń. Odwzorowania te definiuje zleceniodawca, kodując je jako funkcje języka C. W docelowym systemie moduł **optymalizator** oraz moduł zleceniodawcy stanowić będą niezależne zadania (procesy) systemu operacyjnego QNX. W związku z tym niemożliwe będzie wywołanie wprost w kodzie **optymalizatora** funkcji zdefiniowanej w kodzie zleceniodawcy. Zamiast tego zajdzie konieczność skorzystania z mechanizmów IPC oferowanych przez system QNX. **Optymalizator** przygotuje wektorowy argument X i prześle go do zleceniodawcy. Ten wyznaczy odpowiadające mu wartości funkcji celu i ograniczeń i odeśle je **optymalizatorowi**. Ponieważ procedura ta powtarzane będzie wielokrotnie, wydaje się pożądane, by procesy wymieniające między sobą dane (argumenty i wartości funkcji) porozumiewały się ze sobą bezpośrednio, bez pomocy **zarządcy**. Ten ostatni zainicjuje jedynie komunikację między zainteresowanymi procesami.

Obecnie dostępny jest tylko jeden algorytm optymalizacji. Końca dobiegają prace nad drugim algorytmem (metoda najszybszego spadku dla minimalizacji wielowymiarowej, metoda złotego podziału dla poszukiwania na kierunku, rzutowanie kierunku poszukiwań na hiperpłaszczyzny wyznaczone przez ograniczenia proste oraz ograniczenia kroku poszukiwań na kierunku). W przyszłości planowane jest oprogramowanie kilu kolejnych algorytmów i pozostawienie zleceniodawcy możliwości wyboru, który z nich zastosować. Przewiduje się, że najszersze zastosowanie znajdzie algorytm przesuwanej funkcji kary (rozszerzonej funkcji Lagrange'a). Nie jest on jeszcze oprogramowany, ale prowadzone obecnie prace związane z zewnętrzną funkcją kary stanowią etap przygotowawczy przed implementacją metody przesuwanej funkcji kary.

Wykorzystane zadania testowe

Przykłady „akademickie”

Optymalizacja konstrukcji mechanicznych

Belka spawana

Belka żelbetowa - problem częściowo dyskretny

Funkcja nieliniowa z ograniczeniami

Reduktor wielostopniowy Golinskiego

Sprężyna ściskana

Zbiornik ciśnieniowy

Optymalizacja nastaw regulatorów

Regulator PI dla obiektu inercyjnego trzeciego rzędu

Regulator PID dla obiektu inercyjnego trzeciego rzędu

Regulator PID dla helikoptera na uwięzi (o jednym stopniu swobody)

Regulator PI dla kaskady zbiorników

Regulator PID dla kaskady zbiorników

$$\Delta H_3(t) = H_3^* - H_3(t)$$

$$\Delta Q_0(t) = Q_0(t) - Q_0^*$$

$$J = \int_0^T \left(\Delta H_3(t)^2 + p\,\Delta \dot{H}_3(t)^2 + q\,\Delta Q_0(t)^2 \right) dt$$

$$Q_0(s) = K_p\,\Delta H_3(s) + \frac{K_i}{s}\Delta H_3(s) + \frac{K_d\,s}{1+T_n\,s}\Delta H_3(s)$$

Problemy do dyskusji

Pytania A

1. Czy algorytm działa poprawnie w przypadku, gdy zbiór rozwiązań dopuszczalnych jest miary 0 w R^n?
2. Czy algorytm działa poprawnie dla funkcji jednej zmiennej?
3. Użytkownik powinien mieć możliwość zdecydowania, które ograniczenia proste powinny być uwzględniane w funkcji kary już począwszy od pierwszej iteracji.
4. Docelowo planuje się zaimplementowanie kilku różnych algorytmów optymalizacji, do wyboru przez użytkownika.
5. Dostępne będą dwie różne wersje algorytmu poszukiwania minimum na kierunku.
6. Kod programu należy przepisać do postaci proceduralnej (język C zamiast C++).
7. Czy algorytm zadziała prawidłowo, jeśli użytkownik zada 1 jako wartość parametru „maksymalna liczba iteracji"?
8. Na płycie CD do książki *Numerical Recipes* można znaleźć kody metod optymalizacji zarówno w wersji obiektowej (w języku C++), jak i w wersji proceduralnej (w C).
9. Czy algorytm zadziała prawidłowo przy braku ograniczeń równościowych i nierównościowych (nie uwzględniając prostych).
10. Czy nie należy umożliwić użytkownikowi zdefiniowanie ograniczeń liniowych (równościowych i nierównościowych) przez podanie stosownych macierzy i wektorów (A*x=b)?
11. Czy między „epsilonowymi" warunkami stopu dla zmiany wartości funkcji i dla zmiany wartości zmiennej decyzyjnej jest spójnik „i" czy „lub"?
12. Czy iloraz ciągu współczynników kary jest obecnie parametrem programu?

Pytania B

1. Linia o numerze 69 w pliku quasinewton.h ma postać:
   ```
   temp = 0.880622146671798 / (temp - 0.337770382695507) -0.329783693843594;
   ```
   Skąd takie wartości stałych?
2. Czy w metodzie poszukiwania minimum na kierunku za pomocą aproksymacji kwadratowej jest stosowany warunek stopu sprawdzający przekroczenie maksymalnej liczby iteracji?

Pytania C

1. Czy użytkownik może zdefiniować zamiast funkcji celu wektor kosztu (w przypadku, gdy funkcja celu jest liniowa)?
2. Czy użytkownik może zdefiniować mnożnik wykorzystywany przy zwiększaniu współczynników kary (iloraz postępu geometrycznego)?
3. Czy użytkownik może zadać ograniczenia proste, które będą uwzględniane w funkcji celu już od pierwszej iteraci? Ten mechanizm należy uodpornić przed sytuacją, gdy użytkownik wskaże ograniczenia na zmienną, która stanie się parametrem (ze względu na równość górnego i dolnego ograniczenia prostego).
4. Jak działa zmodyfikowana metoda najszybszego spadku?

Postulaty

1. Należy ujednolicić nazewnictwo pól struktury MinState, decydując się bądź wyłącznie na język polski, bądź wyłączni angielski.
2. Oprócz funkcji celu i ograniczeń należy jeszcze wprowadzić funkcję obliczeń wstępnych. Jej zadaniem będzie obliczanie na podstawie zmiennych decyzyjnych pewnych pomocniczych danych, z których będą mogły korzystać funkcje celu i ograniczeń. Te pomocnicze dane byłyby umieszczane w specjalnym buforze, wskazywanym przez jedno z pól struktury. Zadanie alokacji pamięć na bufor spoczywa na użytkowniku. Definiowanie funkcji obliczeń wstępnych nie jest obowiązkowe – wskaźnik NULL oznacza, że użytkownik nie chce jej wykorzystywać. Wprowadzenie do definicji problemu optymalizacji tej dodatkowej funkcji nie ogranicza uniwersalności narzędzia. Zawsze

bowiem można z tej funkcji zrezygnować i zdefiniować tylko funkcje celu i ograniczeń, który w dalszym ciągu otrzymują jako jeden ze swoich argumentów wektor zmiennych decyzyjnych.

3. Może pogrupować opcje odnoszące się do poszczególnych algorytmów w oddzielne struktury (podstruktury)? Przykład: zadanie.dfp.nmax, zadanie.