Politechnika Wrocławska

Optymalizacja

Wojciech Myszka

19 stycznia 2024

HR EXCELLENCE IN RESEARCH

1. Pakiety

2. Optymalizacja

Pakiety

Politechnika Wrocławska

Dostępne pakiety

1. Optimization Toolbox

   zadania:

   - liniowe
   - kwadratowe
   - całkowitoliczbowe
   - nieliniowe

   z ograniczeniami i bez

2. Global Optimization Toolbox

   Udostępnia funkcje wyszukujące **globalne** rozwiązania problemów zawierających wiele maksimów i minimów

Drugi. . .

wymaga pierwszego

Optymalizacja

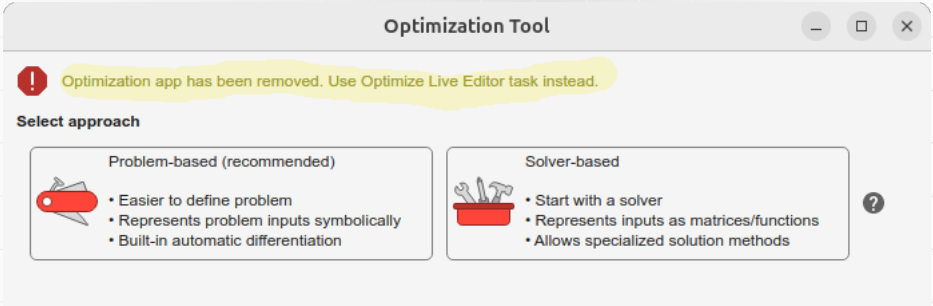
Krótki opis I

1. Możliwe dwa podejścia:
   - 1.1 *Problem based*
     - łatwiejszy
     - funkcja celu i ograniczenia definiowane w sposób „symboliczny”
     - nie zezwala na zmienne zespolone
   - 1.2 *Solver based*
     - trudniejszy
     - funkcja celu i ograniczenia definiowane jako funkcje (ew macierze)
     - nieco szybszy
     - wyposażony w interfejs graficzny
   - 1.3 W każdym przypadku w zależności od kategorii problemu MATLAB potrafi wybrać/zaproponować najlepszy solwer,

Uruchomienie I

1. Z Menu głównego MATLABa wybieramy APPS, a następnie Optimization

   Otwiera się okienko



Uruchomienie II

Wybranie którejś z opcji uruchamia LiveEditor z odpowiednim przykładem

2. W oknie *Live editora* wybieramy z Menu Insert i wśród zadań (Task) znajdujemy Optimize w dziale Mathematics and Optimization

   Jak wyżej musimy wybrać *Problem-based* albo *Solver-based*.

Optimize

Optimize

Autorun

`problem` = Solve an optimization problem or system of equations

Create optimization variables

Name	Dimensions	Type	Lower bound	Upper bound	Initial point
Set variable name	1x1	Continuous	-Inf	Inf	0

Define problem

Goal: Minimize | Maximize | Feasibility | Solve equations

Objective: Define on one line — 5*x^2 + 7*cos(y)

Constraints: Add

Specify problem-dependent solver options

Display results

Problem | Solution | Reason solver stopped | Objective value

Select task mode

Define problem | Solve problem

Show code

Politechnika Wrocławska

Klasy problemów optymalizacyjnych I

1. Funkcja celu:
   - liniowa
   - nielinowa
     - kwadratowa
2. Ograniczenia
   - równości
   - nierówności
   - zmienne całkowite
     - zwłaszcza w przypadku problemów liniowych
3. Rozwiązywanie równań
4. Minimalizacja średniokwadratowa

Optimization Toolbox I

Postępowanie jest nieco skomplikowane, zwłaszcza jeżeli problem jest rozbudowany.

1. Definicja problemu to:
   - jego rozmiar (czy rozmiar zmiennych niezależnych)

     ```
     x = optimvar('x', 1, 2);
     ```

     „ludzka' nazwa zmiennej to `x`; jest typu wektor 1 wiersz i dwie kolumny; odwołujemy się do niej jako `x(1)` i `x(2)`.
   - funkcja celu:

     ```
     obj = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
     ```

Optimization Toolbox II

to działa trochę jaj w pakiecie symbolicznym — skoro `x` jest zmienną optymalizacyjną, to `obj` staje się wyrażeniem optymalizacyjnym (*OptimizationExpression*)

- i sam problem optymalizacyjny:

```
prob = optimproblem('Objective',obj);
```

Zaznaczamy tu tylko, że funkcją celu (`'Objective'`) jest `obj`.

- teraz trzeba zdefiniować ograniczenia

```
nlcons = x(1)^2 + x(2)^2 <= 1;
```

ze względu na formę wyrażenia MATLAB traktuje to jako *Optimizationequality*; i tę nierówność trzeba teraz powiązać z problemem jako **ograniczenia**

```
prob.Constraints.circlecons = nlcons;
```

Optimization Toolbox III

trzeci człon tej struktury to nadana przez nas nazwa ograniczenia (tak dobrana, żeby dobrze się kojarzyła); pierwszy człon (`prob`) to zdefiniowany przez nas problem optymalizacyjny.

2. Krótkie podsumowanie zadania uzyskamy wydając polecenie `show(prob)`

```
>> show(prob)
  OptimizationProblem :
 Solve for:
       x
 minimize :
       ((100 .* (x(2) - x(1).^2).^2) + (1 - x(1)).^2)
 subject to circlecons:
       (x(1).^2 + x(2).^2) <= 1
```

Optimization Toolbox IV

3. Teraz trzeba zdefiniować wartość startową (początkową)

   ```
   x0.x = [0 0];
   ```

4. I zaczynamy:

   ```
   [sol,fval,exitflag,output] = solve(prob,x0)
   ```

   Funkcja `solve()` rozwiązuje problem `prob` rozpoczynając poszukiwania od punktu `x0`.

5. Zwracane przez procedurę wartości to:

   - `sol` — współrzędne „optymalnego" punktu (czyli rozwiązanie);
   - `fval` — wartość funkcji celu w tym punkcie;

Optimization Toolbox V

- `exitflag` —zmienna mówiąca o tym jak zakończyła się procedura (1 oznacza, że znaleziono optimum; 0 — nie udało się znaleźć; odsyłam do dokumentacji w kwestii innych wartości)
- `output` — różne informację na temat procesu optymalizacji; w naszym przypadku efekty wyglądają tak:

```
Solving problem using fmincon.

Local minimum found that satisfies the constraints.

Optimization completed because the objective function is
non-decreasing in feasible directions, to within the value
of the optimality tolerance, and constraints are satisfied
to within the value of the constraint tolerance.
```

Optimization Toolbox VI

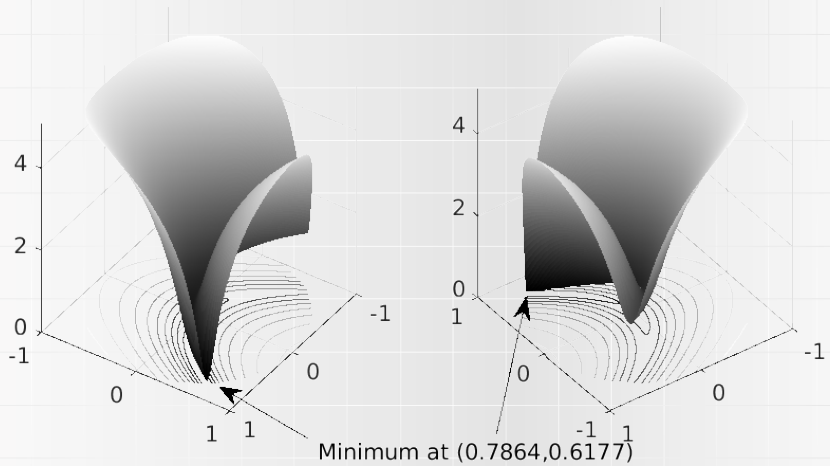
```
<stopping criteria details>
sol =
  struct with fields:
    x: [0.7864 0.6177]
fval =
    0.0457
exitflag =
    OptimalSolution
output =
  struct with fields:
              iterations: 24
               funcCount: 34
          constrviolation: 0
                stepsize: 6.9161e-06
               algorithm: 'interior-point'
```

Optimization Toolbox VII

```
          firstorderopt: 2.1625e-08
          cgiterations: 4
               message: 'Local minimum found that satisfies
          bestfeasible: [1×1 struct]
    objectivederivative: "reverse-AD"
  constraintderivative: "closed-form"
                solver: 'fmincon'
```

Funkcja celu wygląda jakoś tak:

Optimization Toolbox VIII



Funkcja solve() I

- ma bardzo wiele opcji i bardzo wiele zastosowań
- jej użycie może być dosyć proste:

```
x = optimvar('x');
cel = (x+3)*(x+7);
%ogr1 = x >= 10;
%ogr2 = x <= 100;
prob = optimproblem("Objective", cel);
%prob.Constraints.c1 = ogr1;
%prob.Constraints.c2 = ogr2;
x0.x = 50;
sol = solve(prob, x0)
```

- można użyć do rozwiązywania układów równań:

Funkcja solve() II

```
x = optimvar('x');
y = optimvar('y');
eq1 = x^2 + y^2 == 20;
eq2 = x^2 - y^2 == -2;
prob = eqnproblem;
prob.Equations.eq1 = eq1;
prob.Equations.eq2 = eq2;
show(prob);
  EquationProblem :
  Solve for:
       x, y
  eq1:
       (x.^2 + y.^2) == 20
  eq2:
```

Funkcja solve() III

```
        (x.^2 - y.^2) == -2
x0.x = -4;
x0.y = -4;
[sol,fval,exitflag] = solve(prob,x0)
Solving problem using fsolve.

Equation solved.
fsolve completed because the vector of function values is
near zero as measured by the value of the function tolerance,
and the problem appears regular as measured by the gradient.
<stopping criteria details>
sol =
  struct with fields:
    x: -3.0000
```

Funkcja solve() IV

```
        y: -3.3166
    fval =
      struct with fields:
        eq1: 2.4443e-11
        eq2: 2.4196e-11
    exitflag =
        EquationSolved
```