# Použití MATLABu pro matematickou optimalizaci

Ing. Arnoštka Netrvalová
(e-mail: netrvalo@kiv.zcu.cz, ZČU v Plzni, FAV, KIV - místnost: UK 412, tel.: 377632425)

Obsah:


Pro využití metod matematické optimalizace je v MATLABu vytvořen optimalizační toolbox OPTIM.

## 1. Co je optimalizační toolbox?

Optimalizační toolbox je soubor funkcí, které rozšiřují možnosti prostředí numerických výpočtů MATLABu. Toolbox obsahuje podprogramy pro mnoho typů optimalizačních úloh:

- nepodmíněná nelineární minimalizace (unconstrained nonlinear minimization)
- podmíněná nelineární optimalizace, úlohy s penalizacemi, úlohy minimaxu, semi-infinitní minimalizační úlohy (constrained nonlinear minimization, goal attainment problems, minimax problems, and semi-infinite minimization problems)
- kvadratické a lineární programování (quadratic and linear programming)
- nelineární metody nejmenších čtverců a aproximace křivek (nonlinear least squares and curve-fitting)
- řešení soustav nelineárních rovnic (nonlinear system of equation solving)
- lineární metody nejmenších čtverců s vazbami(constrained linear least squares)
- úlohy s řídkými maticemi a strukturované rozsáhlé úlohy (sparse and structured large-scale problems)

Všechny funkce v toolboxu jsou tzv. MATLAB M-soubory, vytvořené příkazy MATLABu, které implementují specializované optimalizační algoritmy. Příkazem *type function_name* je možno si prohlédnout kód těchto funkcí. Dále je možno rozšiřovat rozsah optimalizačního toolboxu zápisem vlastních (uživatelských) M-souborů nebo využitím kombinací s ostatními toolboxy případně použitím *Simulink*®.

# 2. Spolupráce s ostatními toolboxy

V MATLABu existuje možnost využívat při práci i ostatní toolboxy, použitelné pro různé typy úloh, které lze kombinovat s optimalizačním toolboxem. Více informace lze získat:

- v online dokumentaci (je-li produkt nainstalován) či z dokumentace na CD
- na webové stránce *http://www.mathworks.com* v sekci produktů

| Toolbox | Popis |
|---|---|
| Curve Fitting Toolbox | aproximace a analýza |
| Data Acquisition Toolbox | získání a odeslání dat z výměnných datových oblastí |
| Database Toolbox | výměna dat s relačními databázemi |
| Financial Time Series Toolbox | analýza finančních časových řad |
| Financial Toolbox | modely finančních dat a vytváření algoritmů finanční analýzy |
| GARCH Toolbox | analýza finanční volality užitím univariate GARCH modelů |
| LMI Control Toolbox | konstrukce robustního řízení užitím konvexních optimalizačních technik |
| Neural Network Toolbox | vytváření a simulace neuronových sítí |
| Nonlinear Control Design Blockset | optimalizace parametrů v nelineárních systémech řízení |
| Signal Processing Toolbox | analýza signálů a vývoj algoritmů |
| Simulink | návrh a simulace v čase spojitých a diskrétních systémů |
| Spline Toolbox | vytváření a manipulace se spline aproximačními modely dat |
| Statistics Toolbox | použití statistických algoritmů a pravděpodobnostních modelů |
| Symbolic/Extended Symbolic Math Toolbox | provádění výpočtů užitím symbolické matematiky |
| System Identification Toolbox | vytváření lineárních dynamických modelů z naměřených vstupních/výstupních dat |

## Informace ke konfiguraci

Instalovanou verzi OPTIMu je možno zjistit příkazem *ver*. Vypíše se rovněž seznam všech instalovaných toolboxů včetně čísel jejich verzí.
Pokud není optimalizační toolbox naistalován, je třeba zkontrolovat instalační dokumentaci pro danou platformu a návod na instalaci.
Aktualizaci je možno provádět podle informací na webových stránkách Math Works: *http://www.mathworks.com*.

# 3. Jednoduchý příklad použití optimalizace

Jako pomoc na počátku práce s optimalizačním toolboxem je zde uveden následující jednoduchý příklad.

- Příklad – úvod k příkladu
- Popis zadání příkladu – ukazuje přípravu zadání problému před aplikací optimalizačních funkcí
- Hledání řešení – ukazuje, jak řešit problém užitím lineární optimalizace - metody nejmenších čtverců – funkce *lsqlin*.

### Příklad

Předpokládejme, že chceme najít bod (nejblíže počátku) na ploše dané rovnicí:

$$x_1 + 2x_2 + 4x_3 = 7$$

Nejjednodušší cesta, jak vyřešit tento problém, je minimalizace kvadrátu vzdálenosti bodu $x = (x_1, x_2, x_3)$ na ploše od počátku. Tím získáme tentýž optimální bod jako minimalizací skutečné vzdálenosti. Můžeme popsat problém minimalizace kvadrátu vzdálenosti následovně:

$$\min_x f(x) = x_1^2 + x_2^2 + x_3^2$$

s omezující podmínkou:

$$x_1 + 2x_2 + 4x_3 = 7$$

Funkce $f(x)$ je nazývána účelovou funkcí a $x_1 + 2x_2 + 4x_3 = 7$ je vazební rovnice.

Složitější úloha by mohla obsahovat jiné omezující podmínky, podmínky s nerovnostmi, s dolní a horní mezí. Tento příklad využívá k řešení úlohy funkci *lsqlin* pro lineární případ metody nejmenších čtverců.

### Formulace příkladu

Na tomto místě ukážeme formulaci příkladu před použitím funkce *lsqlin*, jež řeší úlohu lineární metodou nejmenších čtverců ve tvaru:

$$\min_x f(x) = \| Cx - d \|^2$$

kde $\| \|$ je norma s podmínkami omezení:

$$Ax \le b$$
$$Aeq \cdot x = beq.$$

Nejprve je třeba vytvořit proměnné pro parametry C, d, A, b Aeq, beq. Funkce *lsqlin* přijme tyto proměnné jako vstupní argumenty s následující syntaxí:

```
x = lsqlin(C, d, A, b, Aeq, beq)
```

Následující kroky vedou k vytvoření proměnných:

**a) Vytvoření proměnných pro účelovou funkci**
Jelikož chceme minimalizovat $x_1^2 + x_2^2 + x_3^2 = \| x \|^2$

*C* zvolíme jako jednotkovou matici 3. řádu a *d* bude nulový vektor o třech prvcích, takže $Cx - d = x$.

```
C = eye (3);
d  = zeros(3,1);
```

**b) Vytvoření proměnných pro omezení**
Jelikož tento příklad nemá omezující nerovnost, můžeme ve vstupních argumentech zadat *A* a *b* jako prázdné matice.

Omezující podmínku $x_1 + 2x_2 + 4x_3 = 7$ lze zapsat v maticové podobě jako:

$$Aeq \cdot x = beq$$

kde: $Aeq = [1\ 2\ 4\ ]$ a $beq = [7]$.

K vytvoření proměnných pro *Aeq* a *beq* zapíšeme:

```
Aeq = [1 2 4];
beq = [7];
```

## Hledání řešení

Pro řešení optimalizačního problému zadáme:

```
[ x,  fval ]  = lsqlin (C, d, [ ], [ ], Aeq, beq)
```

a funkce *lsqlin* vrátí:

```
x =
     0.3333
     0.6667
     1.3333

fval =
     2.3333
```

Minimum nastává v bodě *x* a *fval* je kvadrát vzdálenosti mezi *x* a počátkem.

Poznámka k tomuto příkladu:
Funkce *lsqlin* oznámí varování o přepnutí standardního nastavení algoritmů large-scale na "medium-scale". Tuto zprávu je možno ignorovat, nemá vliv na výsledek výpočtu.

# 4. Optimalizační funkce a jejich použití

Optimalizační algoritmy lze rozdělit na:

- standardní algoritmy
- algoritmy "velkého rozsahu" (large scale)

Optimalizace se týká minimalizace nebo maximalizace funkcí. Optimalizační toolbox obsahuje funkce, které provádějí minimalizaci (nebo maximalizaci) na obecných nelineárních funkcích. Optimalizační toolbox poskytuje funkce pro řešení nelineárních rovnic i metodu nejmenších čtverců (aproximace křivek).
Dále se budeme zabývat úlohami používajícími optimalizační funkce. Následující tabulky shrnují funkce použitelné pro minimalizaci, řešení rovnic, řešení úloh metodou nejmenších čtverců a aproximaci křivek.

**Table 2-1: Minimization**

| Type | Notation | Function |
|---|---|---|
| Scalar Minimization | $\min_a f(a)$ such that $a_1 \le a \le a_2$ | `fminbnd` |
| Unconstrained Minimization | $\min_x f(x)$ | `fminunc`, `fminsearch` |
| Linear Programming | $\min_x f^T x$ such that $A \cdot x \le b, \quad Aeq \cdot x = beq, \quad l \le x \le u$ | `linprog` |
| Quadratic Programming | $\min_x \frac{1}{2} x^T H x + f^T x$ such that $A \cdot x \le b, \quad Aeq \cdot x = beq, \quad l \le x \le u$ | `quadprog` |

**Table 2-1: Minimization (Continued)**

| Type | Notation | Function |
|---|---|---|
| Constrained Minimization | $\min_x f(x)$ such that<br>$c(x) \le 0,\ ceq(x) = 0$<br>$A \cdot x \le b,\ Aeq \cdot x = beq,\ l \le x \le u$ | `fmincon` |
| Goal Attainment | $\min_{x,\gamma} \gamma$ such that<br>$F(x) - w\gamma \le$ goal<br>$c(x) \le 0,\ ceq(x) = 0$<br>$A \cdot x \le b,\ Aeq \cdot x = beq,\ l \le x \le u$ | `fgoalattain` |
| Minimax | $\min_x \max_{\{F_i\}} \{F_i(x)\}$ such that<br>$c(x) \le 0,\ ceq(x) = 0$<br>$A \cdot x \le b,\ Aeq \cdot x = beq,\ l \le x \le u$ | `fminimax` |
| Semi-Infinite Minimization | $\min_x f(x)$ such that<br>$K(x,w) \le 0$ for all $w$<br>$c(x) \le 0,\ ceq(x) = 0$<br>$A \cdot x \le b,\ Aeq \cdot x = beq,\ l \le x \le u$ | `fseminf` |

**Table 2-2: Equation Solving**

| Type | Notation | Function |
|---|---|---|
| Linear Equations | $C \cdot x = d$, $n$ equations, $n$ variables | `\ (slash)` |
| Nonlinear Equation of One Variable | $f(a) = 0$ | `fzero` |
| Nonlinear Equations | $F(x) = 0$, $n$ equations, $n$ variables | `fsolve` |

**Table 2-3: Least-Squares (Curve Fitting)**

| Type | Notation | Function |
|---|---|---|
| Linear Least-Squares | $\min_x \|C \cdot x - d\|_2^2$, $m$ equations, $n$ variables | `\ (slash)` |
| Nonnegative Linear-Least-Squares | $\min_x \|C \cdot x - d\|_2^2$ such that $x \ge 0$ | `lsqnonneg` |
| Constrained Linear-Least-Squares | $\min_x \|C \cdot x - d\|_2^2$ such that<br>$A \cdot x \le b,\ Aeq \cdot x = beq,\ l \le x \le u$ | `lsqlin` |
| Nonlinear Least-Squares | $\min_x \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_i F_i(x)^2$ such that $l \le x \le u$ | `lsqnonlin` |
| Nonlinear Curve Fitting | $\min_x \frac{1}{2}\|F(x, xdata) - ydata\|_2^2$ such that $l \le x \le u$ | `lsqcurvefit` |

Většina těchto minimalizačních podprogramů vyžaduje definici M-souboru obsahujícího funkci, jež má být minimalizována, tj. účelovou funkci. Alternativně lze využít přímo objekt vytvořený prostředky MATLABu. Maximalizace dosáhneme zápisem –f, kde f je funkce, která má být optimalizována.
Optimalizační volby předávané podprogramům mění optimalizační parametry. Využívá se standardní nastavení optimalizačních parametrů, ale parametry mohou být též měněny strukturou voleb. Gradienty se počítají užitím adaptivní metody konečných prvků, nejsou-li přímo poskytnuty funkcí. Parametry mohou být dosazeny přímo do funkcí.

Prostředky optimalizačního toolboxu nabízejí výběr algoritmů metod spádových směrů. Základní algoritmy pro minimalizaci bez omezení jsou Nelder-Meadova simplexová metoda a BFGS (Broyden, Fletcher, Goldfarb, a Shanno) kvazinewtonovská metoda. Pro minimalizaci s omezením jsou pak použity metoda minimaxu, "goal attainment" metoda, semi - infinite minimalizace a sekvenční kvadratické programování (SPQ). Nelineární problémy jako nelineární případ metody nejmenších čtverců používá Gaussovo-Newtonovu a Levenbergovo-Marquardtovu metodu. K řešení nelineárních rovnic se také užívá "trust-region" algoritmus. Pro minimalizaci bez omezení a nelineární úlohu nejmenších čtverců je možno použít metody typu "line-search". Tyto metody používají kubické a kvadratické interpolační a extrapolační metody.
Víceúčelové algoritmy (s výjimkou lineárního programování) jsou metody typu "trust-region". Úlohy s omezením jsou řešeny uvažovanými Newtonovými metodami. Úlohy s podmínkami ve tvaru rovností jsou řešeny "projective preconditioned conjugate gradient" iterací. Metoda lineárního programování je variantou algoritmu Mehrotrova prediktoru-korektoru, "primal-dual interior-point" metody.

## a) Standardní algoritmy

Přehled optimalizací
- úvod do optimalizace jako cesty pro nalezení množiny parametrů, jež mohou být nějakým způsobem definovány jako optimální. Tyto parametry získáme minimalizací či maximalizací účelové funkce, s omezujícími podmínkami (rovností nebo nerovností) a okrajovými body.

Optimalizace bez vazeb
- popisuje užití kvazinewtonovské metody a metody spádových směrů pro optimalizaci bez vazeb. Je popisován způsob aktualizace (a jeho implementace) Hessovy matice v metodě spádových směrů v kvazinewtonově algoritmu, který je použit ve funkci *fminunc*.

Optimalizace pomocí metody nejmenších čtverců
- pojednává o užití Gauss - Newtonovy a Levenberg - Marquardtovy metody pro optimalizační podprogramy nelineárních úloh metodou nejmenších čtverců *lsqnonlin a lsqcurvefit*.

Soustavy nelineárních rovnic
- zabývá se užitím Gauss - Newtonovy metody, Newtonových a "trust-region dogleg" metod užitých funkcí *fsolve*

Optimalizace s vazbami

- pojednává o Kun - Tuckerových rovnicích (podmínkách) jako základu pro sekvenční kvadratické programovací metody (SQP). Je popisován způsob aktualizace Hessovy matice, řešení úloh kvadratického programování užitím funkcí *fmincon*, *fminimax*, *fgoalattain* a *fseminf*. Vysvětluje simplexovou metodu, jež je optimálním algoritmem pro lineární programování.

Víceúčelová optimalizace

- je úvodem k víceúčelové optimalizaci a pojednává o strategiích pro práci s protikladnými kritérii. Detailně pojednává o užití "goal attainment" metody a navrhovaném zlepšení SQP s využitím těchto metod.

### b) Algoritmy "velkého rozsahu" (large scale)

Algoritmy large scale popisují metody použité v optimalizačním toolboxu k řešení large scale optimalizačních úloh. Obsahuje následující sekce:

Trust-Region metody pro nelineární minimalizaci

- popisuje užití metod pro nelineární minimalizaci bez vazeb.

Preconditioned Conjugate Gradients

- předkládá algoritmus, jež užívá PCG pro řešení velkých symetrických pozitivně definitních soustav lineárních rovnic.

Úlohy s lineárními podmínkami

- pojednává o řešení lineárních rovnic s omezujícími podmínkami a typu rovností a nerovností.

Nelineární případ metody nejmenších čtverců

- popisuje řešení úloh metodou nejmenších čtverců pro nelineární případ.

Kvadratické programování

- popisuje řešení úloh minimalizace pomocí kvadratických účelových funkcí.

Lineární případ metody nejmenších čtverců

- popisuje řešení úloh metodou nejmenších čtverců.

Lineární programování

- popisuje užití LIPSOL (Linear Interior Point Solver) pro řešení úloh large scale pomocí lineárního programování.

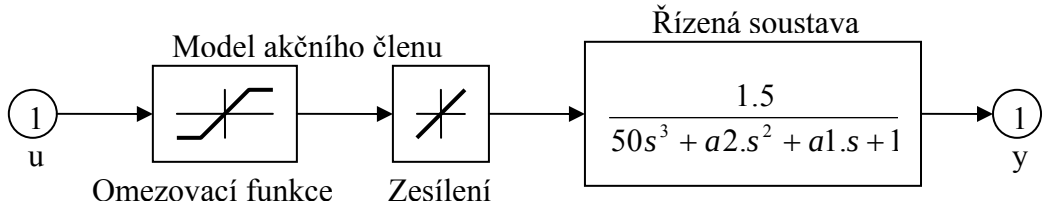
## 5. Přehled optimalizačních funkcí

- viz příloha

# 6. Příklad optimalizace parametrů PID regulátoru

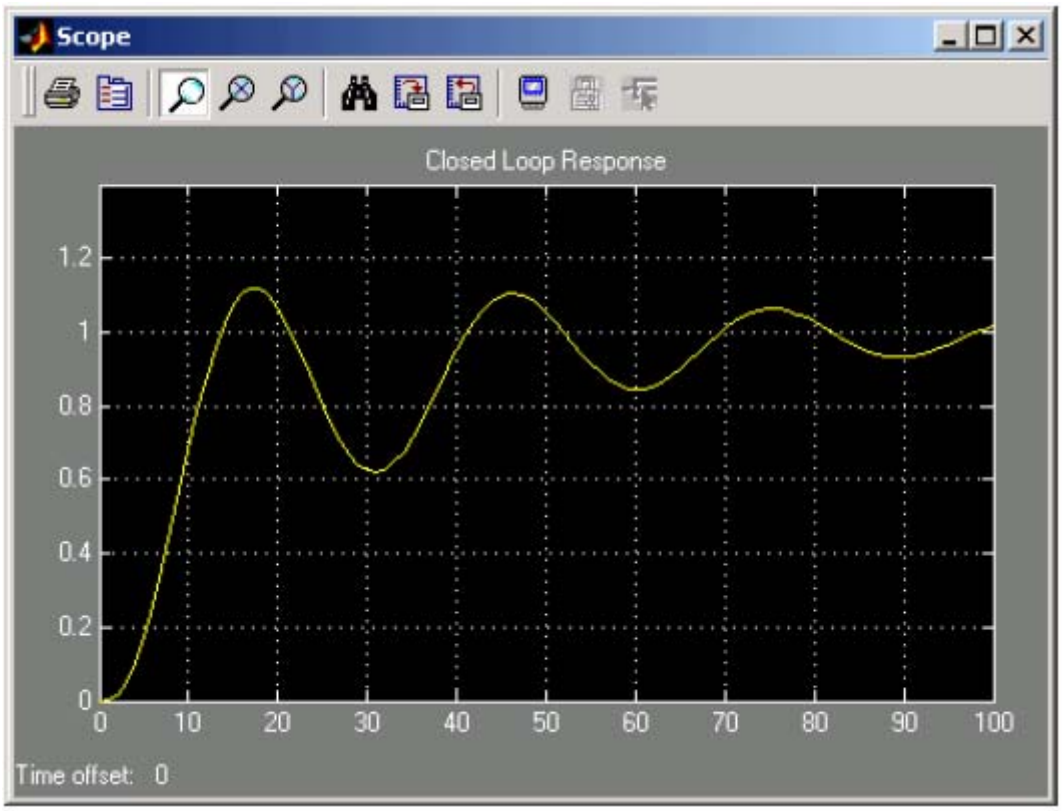
Tento příklad ukazuje řešení víceparametrických úloh použitím funkcí *lsqnonlin*, *fminimax*, a *fgoalattain*. Ukazuje, jak optimalizovat parametry PID regulátoru pomocí modelu Simulink.

Můžeme říci, že chceme optimalizovat parametry řízení modelem Simulink optsim.mdl (tento model se nachází v optimalizačním toolboxu *optim directory*). Nelineární proces (zadaný blokovým diagramem) je zobrazen na obr.1.



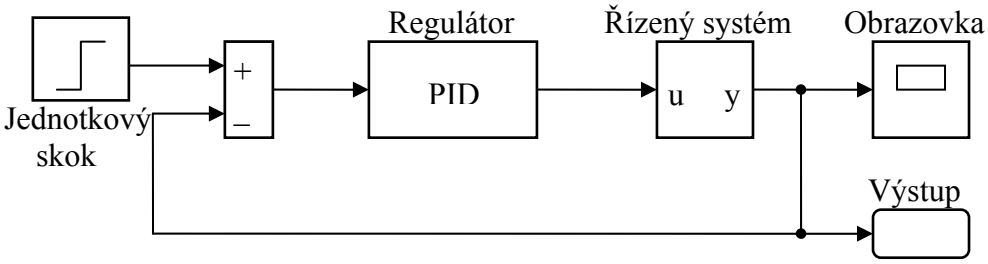

**Obr. 1: Model řízené soustavy**

Řízená soustava je dána přenosovou funkcí 3.řádu s omezeným akčním členem. Omezení jsou saturační limit a nastavitelné zesílení. Saturační limit ořezává vstupní hodnoty větší než 2 jednotky nebo menší než -2 jednotky.Zesílení je 0.8 jednotek/sec. Odezva uzavřené smyčky systému je zobrazena na obr.2. Tuto odezvu je možno získat zápisem funkce *optsim* v příkazovém řádku nebo klikem na jméno modelu a pak výběrem volby *Start* v menu *Simulation*.




**Obr.2: Odezva uzavřené smyčky systému**

Úkolem je zobrazení odezvy na jednotkový skok v uzavřené smyčce. Uzavřená smyčka je tvořena blokovým schématem s řízenou soustavou (obr.1). Obrazovka vykresluje výstupní trajektorie v průběhu zobrazovacího procesu. Model s uzavřenou smyčkou je ukázán na obr.3.




**Obr. 3: Model s uzavřenou smyčkou**

Jednou z možností řešení této úlohy je minimalizace chyby mezi vstupním a výstupním signálem. Proměnné jsou parametry PID regulátoru. Pokud potřebujeme pouze minimalizovat chybu v jednom časovém okamžiku, pak to bude jednoduchá účelová funkce. Ale cílem je minimalizovat chybu pro všechny kroky v čase od 0 do 100, jak to provádí víceparametrová funkce (funkce pro všechny časové kroky). Podprogram *lsqnonlin* je užit k předvedení úpravy nejmenších čtverců na sledovaném výstupu. To je popsáno pomocí MATLAB funkcí v souboru *tracklsq.m*, který definuje chybový signál. Chybový signál y_out, výstup se spočte voláním *sim*, minus jednotkový vstupní signál. Funkce *tracklsq* rozbíhá simulaci. Pro spuštění simulace v *optsim* musí být definovány všechny proměnné *Kp, Ki, Kd*, *a1* a *a2* (*a1* a *a2* jsou proměnné v řízeném systému). *Kp, Ki a Kd* jsou proměnné, jež mají být optimalizovány. Je možno inicializovat *a1* a *a2* před voláním *lsqnonlin* a pak výpočet provést s těmito dvěma proměnnými jako dodatečnými argumenty. Funkce *lsqnonlin* využije *a1* a *a2* pro *tracklsq* pokaždé, když jsou volány, takže nemusíme použít globální proměnné. Po zvolení řešení pomocí *simset* funkce, běží simulace pomocí sim. Simulace pracuje s pevným krokem metody 5.stupně do 100 sekund. Když je simulace hotova, proměnné *t_out*, *x_out* a *y_out* jsou v aktuálním pracovním prostoru (jež je , pracovním prostorem funkce *tracklsq*). Výstupní port je užit v blokovém diagramu modelu k umístění *y_out* do průběžného pracovního prostoru na konci simulace.

**Krok 1: Zápis M-file tracklsq.m.**

```
function F = tracklsq(pid,a1,a2)
Kp = pid(1); % Move variables into model parameter names
Ki = pid(2);
Kd = pid(3);
% Choose solver and set model workspace to this function
opt = simset('solver','ode5','SrcWorkspace','Current');
[tout,xout,yout] = sim('optsim',[0 100],opt);
F = yout-1; % Compute error signal
```

**Krok 2: Volání optimalizačních podprogramů.**

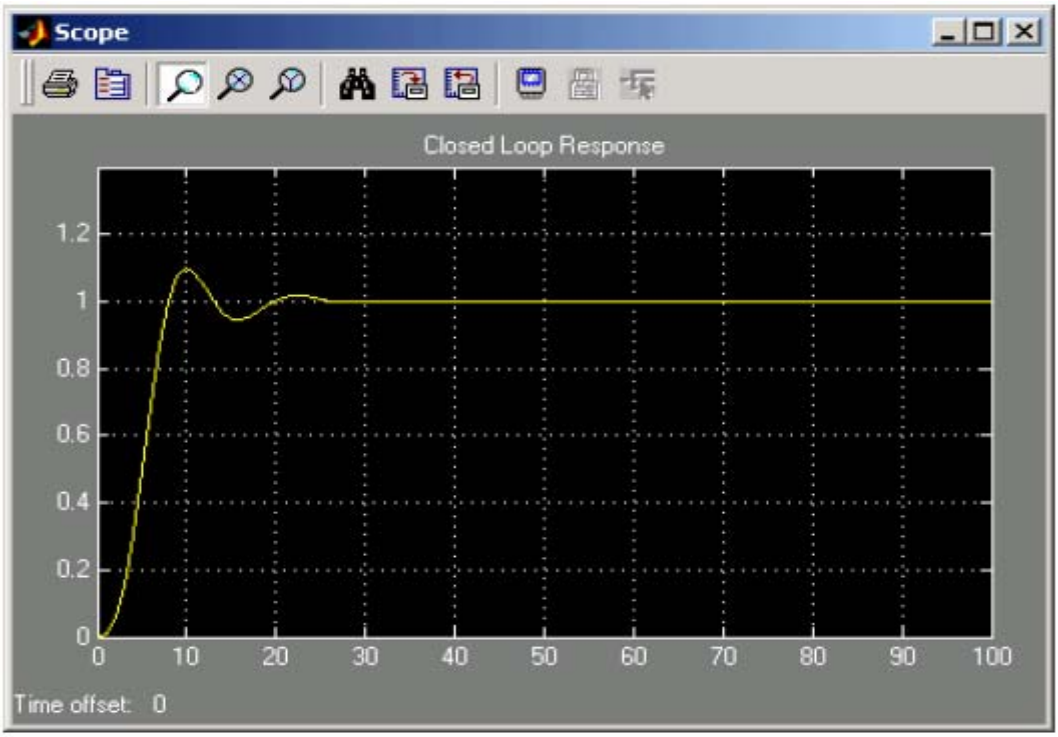
```
optsim % Load the model
pid0 = [0.63 0.0504 1.9688] % Set initial values
a1 = 3; a2 = 43; % Initialize plant variables in model
options = optimset('LargeScale','off','Display','iter',...
'TolX',0.001,'TolFun',0.001);
pid = lsqnonlin(@tracklsq, pid0, [ ], [ ], options, a1, a2)
% Put variables back in the base workspace
Kp = pid(1); Ki = pid(2); Kd = pid(3);
```

Volby proměnných provedené v *lsqnonlin* definují kriteria a zobrazení charakteristik. V tomto případě jste dotázáni na výstup, použije se "medium-scale" algoritmus a zadání tolerancí kroku a účelové funkce (od 0.001). Optimalizace dává řešení pro proporcionální, integrační, a derivační (*Kp, Ki, Kd*) konstanty regulátoru po 64 funkčních vyčísleních:

| Iteration | Func-count | Residual | Step-size | Directional derivative | Lambda |
|---|---|---|---|---|---|
| 1 | 3 | 8.66531 | 1 | -3.48 | |
| 2 | 17 | 5.21602 | 85.4 | -0.00813 | 0.0403059 |
| 3 | 24 | 4.54036 | 1 | -0.0331 | 0.393189 |
| 4 | 31 | 4.47786 | 0.918 | -0.00467 | 0.201985 |
| 5 | 39 | 4.47552 | 2.12 | 0.00121 | 0.100992 |
| 6 | 46 | 4.47524 | 0.203 | -0.00193 | 0.0718569 |
| 7 | 64 | 4.47524 | -4.11e-007 | -0.00157 | 2595.3 |

Optimization terminated successfully:
Search direction less than tolX

pid = 2.9186 0.1398 12.6221




**Obr. 4: Odezva uzavřené smyčky dosažená použitím funkce *Isqnonlin***

Nelineární systém řízení je doporučen pro řešení víceparametrových optimalizačních úloh v součinnosti s řešením s proměnným krokem v Simulinku. To poskytuje speciální numerický výpočet gradientu, který pracuje se Simulinkem a vyvaruje se nedostatku hladkosti v úvodní fázi. Dalším přiblížením řešení je užití *fminimax* funkce. V tomto případě, více než minimalizace chyby mezi výstupním a vstupním signálem, minimalizujeme maximální hodnotu výstupu v libovolném čase t mezi 0 a 100. Potom ve funkci *trackmmobj* je výstup jednoduché účelové funkce out vrácen příkazem *sim*. Ale minimalizací výstupního maxima ve všech časových krocích může dojít k zesílení pro některé časové kroky hodně

pod jednotku. Abychom získali výstup okolo 0.95 po prvních 20 vteřinách, musíme k omezující funkci *trackkmmcon* přičíst omezení *y_out* >= 0.95 od *t* =20 do *t* =100. Protože podmínka musí být dána nerovností *g* <= 0, je omezení *g* = -*y_out*(20:100)+.95. Obě funkce *trackmmobj* a *trackmmcon* užívají výsledný *y_out* ze *sim*, spočtený z aktuálních *pid* hodnot. Nelineární podmíněná funkce je vždy volána bezprostředně po účelové funkci v *fmincon, fminimax*, *fgoalattain* a *fseminf* s týmiž hodnotami. Tak se můžeme vyvarovat dvojího volání simulace použitím *assignin* a stanovením aktuální hodnoty *F* pro proměnnou *F_TRACKMMOBJ* v základním pracovním prostoru. Pak prvním krokem v *trackmmcon* je použití *evalin* k vyčíslení proměnné *F_TRACKMMOBJ* a zadání výsledku do *F* lokálně v *trackmmcon*.

**Krok 1: Zápis M-file trackmmobj.m pro výpočet účelové funkce.**

```
function F = trackmmobj(pid,a1,a2)
Kp = pid(1);
Ki = pid(2);
Kd = pid(3);
% Compute function value
opt = simset('solver','ode5','SrcWorkspace','Current');
[tout,xout,yout] = sim('optsim',[0 100],opt);
F = yout;
assignin('base','F_TRACKMMOBJ',F);
```

**Krok 2: Zápis M-file trackmmcon.m pro výpočet nelineárního omezení.**

```
function [c,ceq] = trackmmcon(pid,a1,a2)
F = evalin('base','F_TRACKMMOBJ');
% Compute constraints
c = -F(20:100)+.95;
ceq = [ ];
```

**Krok 3: Spuštění optimalizačního podprogramu s omezením.**

```
optsim
pid0 = [0.63 0.0504 1.9688]
a1 = 3; a2 = 43;
options = optimset('Display','iter',...
'TolX',0.001,'TolFun',0.001);
pid = fminimax(@trackmmobj,pid0,[ ],[ ],[ ],[ ],[ ],[ ],...
'trackmmcon',options,a1,a2)
% Put variables back in the base workspace
Kp = pid(1); Ki = pid(2); Kd = pid(3);
```

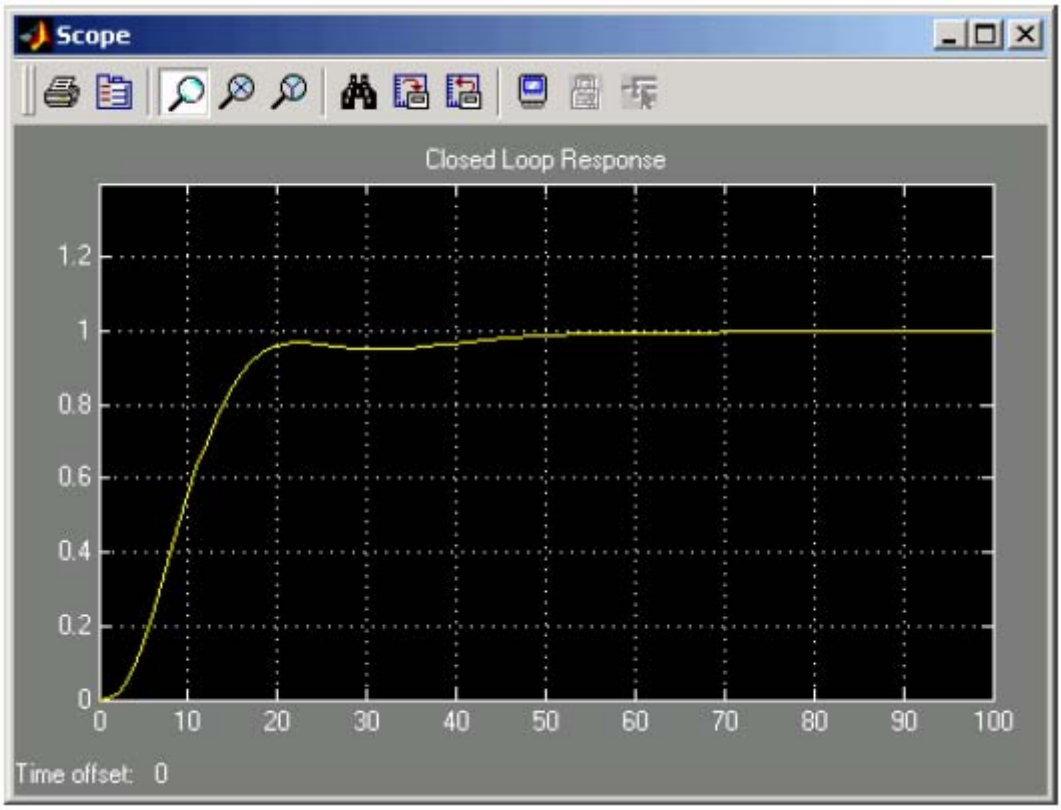
| Iter | F-count | {F,constraints} | Step-size | Max derivative | Directional | Procedure |
|---|---|---|---|---|---|---|
| 1 | 11 | 1.264 | 1 | 1.18 | | |
| 2 | 17 | 1.055 | 1 | -0.172 | | |
| 3 | 23 | 1.004 | 1 | -0.0128 | | Hessian modified twice |
| 4 | 29 | 0.9997 | 1 | 3.48e-005 | | Hessian modified |
| 5 | 35 | 0.9996 | 1 | -1.36e-006 | | Hessian modified twice |

Optimization terminated successfully:
Search direction less than 2*options.TolX and
maximum constraint violation is less than options.TolCon

Active Constraints:
1
14
182

pid = 0.5894  0.0605  5.5295

Poslední hodnota zapsaná v *MAX*{*F*,constraints} ve sloupci výstupu ukazuje, že maximální hodnota pro všechny časové kroky je 0.9996. Odezva uzavřené smyčky s tímto výsledkem je zobrazena na obr.5. Toto řešení se liší od řešení pomocí *lsqnonlin*, protože jsme řešili odlišně formulovanou úlohu.




**Obr. 5: Odezva uzavřené smyčky dosažená použitím funkce *fminimax***

# 7. Zdroje – url


[1] Optimization Toolbox User's Guide, Version 2, 1990 - 2003 by The MathWorks, Inc.

[2] Coleman T., Branch M.A., Grace A.: Optimization Toolbox User's Guide, Version 2, 1990 -1999, by The MathWorks, Inc.: *http://www.mathworks.com/access/helpdesk/help/toolbox/optim/optim.shtml*

[3] MATLAB - www stránky: http://www.mathworks.com/

[4] Introduction to MATLAB, The University of Queensland: *http://www.maths.uq.edu.au/~gac/mlb/mlb.html*

[5] Kermit Sigmon: MATLAB Primer CZ, Department of Mathematics, University of Florida (překlad Petr Klášterecký): *http://adela.karlin.mff.cuni.cz/~hlubinka/matlabprimer/matlab-primer.html*

[6] Optimization Toolbox Reference: *http://www-ccs.ucsd.edu/matlab/toolbox/optim/optbookt.html*

# Příloha:

# Přehled optimalizačních funkcí

# Functions – By Category

The Optimization Toolbox provides these categories of functions.

| | |
|---|---|
| Minimization | Minimization functions |
| Equation Solving | Solution of linear and nonlinear equations |
| Least Squares (Curve Fitting) | Linear and nonlinear curve fitting |
| Utility | Setting and retrieving optimizations parameters |
| Demos of Large-Scale Methods | Demonstration programs of large-scale methods |
| Demos of Medium-Scale Methods | Demonstration programs of medium-scale methods |

## Minimization

| | |
|---|---|
| `fgoalattain` | Multiobjective goal attainment |
| `fminbnd` | Scalar nonlinear minimization with bounds |
| `fmincon` | Constrained nonlinear minimization |
| `fminimax` | Minimax optimization |
| `fminsearch`, `fminunc` | Unconstrained nonlinear minimization |
| `fseminf` | Semi-infinite minimization |
| `linprog` | Linear programming |
| `quadprog` | Quadratic programming |

## Equation Solving

| | |
|---|---|
| `\` | Use \ (left division) to solve linear equations. See the Arithmetic Operators reference page in the online MATLAB documentation. |
| `fsolve` | Nonlinear equation solving |

| | |
|---|---|
| `fzero` | Scalar nonlinear equation solving |

## Least Squares (Curve Fitting)

| | |
|---|---|
| `\` | Use \ (left division) for linear least squares with no constraints. See the Arithmetic Operators reference page. |
| `lsqlin` | Constrained linear least squares |
| `lsqcurvefit` | Nonlinear curve fitting |
| `lsqnonlin` | Nonlinear least squares |
| `lsqnonneg` | Nonnegative linear least squares |

## Utility

| | |
|---|---|
| `fzmult` | Multiplication with fundamental nullspace basis |
| `gangstr` | Zero out “small” entries subject to structural rank |
| `optimget` | Get optimization options parameter values |
| `optimset` | Create or edit optimization options parameter structure |

## Demos of Large-Scale Methods

From the MATLAB Help browser, click the demo name to run the demo. Look for information and additional instructions in the MATLAB Command Window.

| | |
|---|---|
| `circustent` | Quadratic programming to find shape of a circus tent |
| `molecule` | Molecule conformation solution using unconstrained nonlinear minimization |
| `optdeblur` | Image deblurring using bounded linear least squares |

## Demos of Medium-Scale Methods

From the MATLAB Help browser, click the demo name to run the demo. Look for information and additional instructions in the MATLAB Command Window.

| | |
|---|---|
| `bandemo` | Minimization of the `banana` function |
| `dfildemo` | Finite-precision filter design (requires the Signal Processing Toolbox) |
| `goaldemo` | Goal attainment example |
| `optdemo` | Menu of demo routines |
| `tutdemo` | Script for the medium-scale algorithms. The script follows the "Tutorial" chapter of the Optimization Toolbox User's Guide. |

# Function Arguments

The Optimization Toolbox functions use these arguments.

| | |
|---|---|
| Input Arguments | General descriptions of input arguments used by toolbox functions. |
| Output Arguments | General descriptions of output arguments used by toolbox functions. |

Individual function reference pages provide function-specific information, as necessary.

## Input Arguments

| Argument | Description | Used by Functions |
|---|---|---|
| `A, b` | The matrix `A` and vector `b` are, respectively, the coefficients of the linear inequality constraints and the corresponding right-side vector: `A*x <= b`. | `fgoalattain`, `fmincon`, `fminimax`, `fseminf`, `linprog`, `lsqlin`, `quadprog` |
| `Aeq, beq` | The matrix `Aeq` and vector `beq` are, respectively, the coefficients of the linear equality constraints and the corresponding right-side vector: `Aeq*x = beq`. | `fgoalattain`, `fmincon`, `fminimax`, `fseminf`, `linprog`, `lsqlin`, `quadprog` |
| `C, d` | The matrix `C` and vector `d` are, respectively, the coefficients of the over or underdetermined linear system and the right-side vector to be solved. | `lsqlin`, `lsqnonneg` |
| `f` | The vector of coefficients for the linear term in the linear equation `f'*x` or the quadratic equation `x'*H*x+f'*x`. | `linprog`, `quadprog` |

| Argument | Description | Used by Functions |
|---|---|---|
| fun | The function to be optimized. fun is a function or an inline object. See the individual function reference pages for more information on fun. | fgoalattain, fminbnd, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, fzero, lsqcurvefit, lsqnonlin |
| goal | Vector of values that the objectives attempt to attain. The vector is the same length as the number of objectives. | fgoalattain |
| H | The matrix of coefficients for the quadratic terms in the quadratic equation x'*H*x+f'*x. H must be symmetric. | quadprog |
| lb, ub | Lower and upper bound vectors (or matrices). The arguments are normally the same size as x. However, if lb has fewer elements than x, say only m, then only the first m elements in x are bounded below; upper bounds in ub can be defined in the same manner. You can also specify unbounded variables using -Inf (for lower bounds) or Inf (for upper bounds). For example, if lb(i) = -Inf, the variable x(i) is unbounded below. | fgoalattain, fmincon, fminimax, fseminf, linprog, lsqcurvefit, lsqlin, lsqnonlin, quadprog |
| nonlcon | The function that computes the nonlinear inequality and equality constraints. See the individual reference pages for more information on nonlcon. | fgoalattain, fmincon, fminimax |
| ntheta | The number of semi-infinite constraints. | fseminf |
| options | An optimization options parameter structure that defines parameters used by the optimization functions. For information about the parameters, see Table , Optimization Parameters, or the individual function reference pages. | All functions |

| Argument | Description | Used by Functions |
|---|---|---|
| P1, P2, ... | Additional arguments to be passed to fun, nonlcon (if it exists), and seminfcon (if it exists), when the optimization function calls the functions fun, nonlcon, or seminfcon using these calls:<br>`f = feval(fun,x,P1,P2,...)`<br>`[c, ceq] = feval(nonlcon,x,P1,P2,...)`<br>`[c,ceq,K1,K2,...,Kn,s]= ...`<br>`feval(seminfcon,x,s,P1,P2,...)` | fgoalattain, fminbnd, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, fzero, lsqcurvefit, lsqnonlin |
| | Using this feature, the same fun (or nonlcon or seminfcon) can solve a number of similar problems with different parameters, avoiding the need to use global variables. | |
| seminfcon | The function that computes the nonlinear inequality and equality constraints *and* the semi-infinite constraints. seminfcon is the name of an M-file or MEX-file. See the function reference pages for fseminf for more information on seminfcon. | fseminf |
| weight | A weighting vector to control the relative underattainment or overattainment of the objectives. | fgoalattain |
| xdata, ydata | The input data xdata and the observed output data ydata that are to be fitted to an equation. | lsqcurvefit |
| x0 | Starting point (a scalar, vector or matrix).<br>(For fzero, x0 can also be a two-element vector representing an interval that is known to contain a zero.) | All functions except fminbnd |
| x1, x2 | The interval over which the function is minimized. | fminbnd |

## Output Arguments

| Argument | Description | Used by Functions |
|---|---|---|
| `attainfactor` | The attainment factor at the solution `x`. | `fgoalattain` |
| `exitflag` | The exit condition. For the meaning of a particular value, see the function reference pages. | All functions |
| `fval` | The value of the objective function `fun` at the solution `x`. | `fgoalattain`, `fminbnd`, `fmincon`, `fminimax`, `fminsearch`, `fminunc`, `fseminf`, `fsolve`, `fzero`, `linprog`, `quadprog` |
| `grad` | The value of the gradient of `fun` at the solution `x`. If `fun` does not compute the gradient, `grad` is a finite-differencing approximation of the gradient. | `fmincon`, `fminunc` |
| `hessian` | The value of the Hessian of `fun` at the solution `x`. For large-scale methods, if `fun` does not compute the Hessian, `hessian` is a finite-differencing approximation of the Hessian. For medium-scale methods, `hessian` is the value of the Quasi-Newton approximation to the Hessian at the solution `x`. | `fmincon`, `fminunc` |
| `jacobian` | The value of the Jacobian of `fun` at the solution `x`. If `fun` does not compute the Jacobian, `jacobian` is a finite-differencing approximation of the Jacobian. | `lsqcurvefit`, `lsqnonlin`, `fsolve` |

| Argument | Description | Used by Functions |
|---|---|---|
| `lambda` | The Lagrange multipliers at the solution `x`. `lambda` is a structure where each field is for a different constraint type. For structure field names, see individual function descriptions. (For `lsqnonneg`, `lambda` is simply a vector, as `lsqnonneg` only handles one kind of constraint.) | `fgoalattain`, `fmincon`, `fminimax`, `fseminf`, `linprog`, `lsqcurvefit`, `lsqlin`, `lsqnonlin`, `lsqnonneg`, `quadprog` |
| `maxfval` | `max{fun(x)}` at the solution `x`. | `fminimax` |
| `output` | An output structure that contains information about the results of the optimization. For structure field names, see individual function descriptions. | All functions |
| `residual` | The value of the residual at the solution `x`. | `lsqcurvefit`, `lsqlin`, `lsqnonlin`, `lsqnonneg` |
| `resnorm` | The value of the squared 2-norm of the residual at the solution `x`. | `lsqcurvefit`, `lsqlin`, `lsqnonlin`, `lsqnonneg` |
| `x` | The solution found by the optimization function. If `exitflag > 0`, then `x` is a solution; otherwise, `x` is the value of the optimization routine when it terminated prematurely. | All functions |

# Optimization Parameters

This table describes fields in the optimization parameters structure `options`. The column labeled L, M, B indicates whether the parameter applies to large-scale methods, medium scale methods, or both:

- L – Large-scale methods only
- M – Medium-scale methods only
- B – Both large- and medium-scale methods

See the Optimization Toolbox `optimset` reference page, the MATLAB `optimset` reference page, and the individual function reference pages for information about parameter values and defaults.

---

**Note** Links in this table are to Optimization Toolbox functions. These links are to the corresponding MATLAB optimization functions: `fminbnd`, `fminsearch`, `fzero`, `lsqnonneg`, `optimget`, `optimset`.

---

| Parameter Name | Description | L, M, B | Used by Functions |
|---|---|---|---|
| `DerivativeCheck` | Compare user-supplied analytic derivatives (gradients or Jacobian) to finite differencing derivatives. | B | `fgoalattain`, `fmincon`, `fminimax`, `fminunc`, `fseminf`, `fsolve`, `lsqcurvefit`, `lsqnonlin` |
| `Diagnostics` | Display diagnostic information about the function to be minimized or solved. | B | All but `fminbnd`, `fminsearch`, `fzero`, and `lsqnonneg` |
| `DiffMaxChange` | Maximum change in variables for finite-difference derivatives. | M | `fgoalattain`, `fmincon`, `fminimax`, `fminunc`, `fseminf`, `fsolve`, `lsqcurvefit`, `lsqnonlin` |

| Parameter Name | Description | L, M, B | Used by Functions |
|---|---|---|---|
| `DiffMinChange` | Minimum change in variables for finite-difference derivatives. | M | `fgoalattain`, `fmincon`, `fminimax`, `fminunc`, `fseminf`, `fsolve`, `lsqcurvefit`, `lsqnonlin` |
| `Display` | Level of display. `'off'` displays no output; `'iter'` displays output at each iteration; `'final'` displays just the final output; `'notify'` displays output only if function does not converge. | B | All. See the individual function reference pages for the values that apply. |
| `GoalsExactAchieve` | Number of goals to achieve exactly (do not over- or underachieve). | M | `fgoalattain` |
| `GradConstr` | Gradients for the nonlinear constraints defined by the user. | M | `fgoalattain`, `fmincon`, `fminimax` |
| `GradObj` | Gradients for the objective functions defined by the user. | B | `fgoalattain`, `fmincon`, `fminimax`, `fminunc`, `fseminf` |
| `Hessian` | If `'on'`, function uses user-defined Hessian or Hessian information (when using `HessMult`), for the objective function. If `'off'`, function approximates the Hessian using finite differences. | L | `fmincon`, `fminunc` |
| `HessMult` | Hessian multiply function defined by the user. | L | `fmincon`, `fminunc`, `quadprog` |

| Parameter Name | Description | L, M, B | Used by Functions |
|---|---|---|---|
| HessPattern | Sparsity pattern of the Hessian for finite differencing. The size of the matrix is n-by-n, where n is the number of elements in x0, the starting point. | L | fmincon, fminunc |
| HessUpdate | Quasi-Newton updating scheme. | M | fminunc |
| Jacobian | If 'on', function uses user-defined Jacobian or Jacobian information (when using JacobMult), for the objective function. If 'off', function approximates the Jacobian using finite differences. | B | fsolve, lsqcurvefit, lsqnonlin |
| JacobMult | Jacobian multiply function defined by the user. | L | fsolve, lsqcurvefit, lsqlin, lsqnonlin |
| JacobPattern | Sparsity pattern of the Jacobian for finite differencing. The size of the matrix is m-by-n, where m is the number of values in the first argument returned by the user-specified function fun, and n is the number of elements in x0, the starting point. | L | fsolve, lsqcurvefit, lsqnonlin |
| LargeScale | Use large-scale algorithm if possible. | B | fmincon, fminunc, fsolve, linprog, lsqcurvefit, lsqlin, lsqnonlin, quadprog |
| LevenbergMarquardt | Choose Levenberg-Marquardt over Gauss-Newton algorithm. | M | lsqcurvefit, lsqnonlin |

| Parameter Name | Description | L, M, B | Used by Functions |
|---|---|---|---|
| LineSearchType | Line search algorithm choice. | M | fminunc, fsolve, lsqcurvefit, lsqnonlin |
| MaxFunEvals | Maximum number of function evaluations allowed. | B | fgoalattain, fminbnd, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, lsqcurvefit, lsqnonlin |
| MaxIter | Maximum number of iterations allowed. | B | All but fzero and lsqnonneg |
| MaxSQPIter | Maximum number of SQP iterations allowed | M | fmincon |
| MaxPCGIter | Maximum number of PCG iterations allowed. | L | fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog |
| MeritFunction | Use goal attainment/minimax merit function (multiobjective) vs. fmincon (single objective). | M | fgoalattain, fminimax |
| MinAbsMax | Number of *F(x)* to minimize the worst case absolute values | M | fminimax |
| NonlEqnAlgorithm | Choose Levenberg-Marquardt or Gauss-Newton over the trust-region dogleg algorithm. | M | fsolve |
| OutputFcn | Specify a user-defined function that the optimization function calls at each iteration. See "Output Function" on page 5-15. | B | fgoalattain, fmincon, fminimax, fminunc, fseminf, lsqcurvefit, lsqnonlin |
| PrecondBandWidth | Upper bandwidth of preconditioner for PCG. | L | fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog |

| Parameter Name | Description | L, M, B | Used by Functions |
|---|---|---|---|
| Simplex | If 'on', function uses simplex algorithm. | M | linprog |
| TolCon | Termination tolerance on the constraint violation. | B | fgoalattain, fmincon, fminimax, fseminf |
| TolFun | Termination tolerance on the function value. | B | fgoalattain, fmincon, fminimax, fminsearch, fminunc, fseminf, fsolve, linprog (large-scale only), lsqcurvefit, lsqlin (large-scale only), lsqnonlin, quadprog (large-scale only) |
| TolPCG | Termination tolerance on the PCG iteration. | L | fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog |
| TolX | Termination tolerance on $x$. | B | All functions except the medium-scale algorithms for linprog, lsqlin, and quadprog |
| TypicalX | Typical x values. The length of the vector is equal to the number of elements in x0, the starting point. | L | fmincon, fminunc, fsolve, lsqcurvefit, lsqlin, lsqnonlin, quadprog |

# Output Function

The `Outputfcn` field of the `options` structure specifies an M-file function that an optimization function calls at each iteration. Typically, you might use an output function to plot points at each iteration or to display data from the algorithm. To set up an output function, do the following:

1. Write the output function as an M-file function or subfunction.
2. Use `optimset` to set the value of `Outputfcn` to be a function handle, that is, the name of the function preceded by the @ sign. For example, if the output function is `outfun.m`, the command

   ```
   options = optimset('OutputFcn', @outfun);
   ```

   sets the value of `OutputFcn` to be the handle to `outfun`.
3. Call the optimization function with `options` as an input argument.

See "Calling an Output Function Iteratively" on page 2-75 for an example of an output function.

## Structure of the Output Function

The function definition line of the output function has the following form:

```
stop = outfun(x, optimValues, state, varargin)
```

where

- `x` is the point computed by the algorithm at the current iteration.
- `optimValues` is a structure containing data from the current iteration. "Fields in optimValues" on page 5-16 describes the structure in detail.
- `state` is the current state of the algorithm. "States of the Algorithm" on page 5-21 lists the possible values.
- `varargin` contains other problem-dependent input arguments that the optimization function might pass to `outfun`. "Other Input Arguments" on page 5-22 describes these arguments.
- `stop` is a flag that is `true` or `false` depending on whether the optimization routine should quit or continue. See "Stop Flag" on page 5-22 for more information.

The optimization function passes the values of the input arguments to `outfun` at each iteration.

### Fields in optimValues

The following table lists the fields of the `optimValues` structure. A particular optimization function returns values for only some of these fields. For each field, the Returned by Functions column of the table lists the functions that return the field.

**Corresponding Output Arguments.** Some of the fields of `optimValues` correspond to output arguments of the optimization function. After the final iteration of the optimization algorithm, the value of such a field equals the corresponding output argument. For example, `optimValues.fval` corresponds to the output argument `fval`. So, if you call `fmincon` with an output function and return `fval`, the final value of `optimValues.fval` equals `fval`. The Description column of the following table indicates the fields that have a corresponding output argument.

**Command-Line Display.** The values of some fields of `optimValues` are displayed at the command line when you call the optimization function with the `Display` parameter of `options` set to `'iter'`, as described in "Displaying Iterative Output" on page 2-69. For example, `optimValues.fval` is displayed in the `f(x)` column. The Command-Line Display column of the following table indicates the fields that you can display at the command line.

In the following table, the letters L, M, and B mean the following:

- L — Function returns a value to the field when using large-scale algorithm.
- M — Function returns a value to the field when using medium-scale algorithm.
- B — Function returns a value to the field when using both large and medium-scale algorithms.

| OptimValues Field (optimValues.field) | Description | Returned by Functions | Command-Line Display |
|---|---|---|---|
| `cgiterations` | Number of conjugate gradient iterations at current iteration. Final value equals optimization function output `output.cgiterations`. | `fmincon` (L), `lsqcurvefit` (L), `lsqnonlin` (L) | `CG-iterations`<br>See "Displaying Iterative Output" on page 2-69. |
| `constrviolation` | Maximum constraint violation | `fgoalattain` (M), `fmincon` (M), `fminimax` (M), `fseminf` (M) | `max constraint`<br>See "Displaying Iterative Output" on page 2-69. |
| `degenerate` | Measure of degeneracy. A point is *degenerate* if<br>• The partial derivative with respect to one of the variables is 0 at the point.<br>• A bound constraint is active for that variable at the point.<br>See "Degeneracy" on page 5-21. | `fmincon` (L), `lsqcurvefit` (L), `lsqnonlin` (L) | None |

| OptimValues Field (optimValues.field) | Description | Returned by Functions | Command-Line Display |
|---|---|---|---|
| `directionalderivative` | Directional derivative in the search direction | `fgoalattain` (M), `fmincon` (M), `fminimax` (M), `fminunc` (M), `fseminf` (M), `lsqcurvefit` (M), `lsqnonlin` (M) | `Directional derivative`<br>See "Displaying Iterative Output" on page 2-69. |
| `firstorderopt` | First-order optimality (depends on algorithm). Final value equals optimization function output `output.firstorderopt`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (M), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | `First-order optimality`<br>See "Displaying Iterative Output" on page 2-69. |
| `funcount` | Cumulative number of function evaluations. Final value equals optimization function output `output.funcCount`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (B), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | `F-count`<br>See "Displaying Iterative Output" on page 2-69. |
| `fval` | Function value at current point. Final value equals optimization function output `fval`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (B), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | `f(x)`<br>See "Displaying Iterative Output" on page 2-69. |

| OptimValues Field (optimValues.field) | Description | Returned by Functions | Command-Line Display |
|---|---|---|---|
| `gradient` | Current gradient of objective function — either analytic gradient if you provide it or finite-differencing approximation. Final value equals optimization function output `grad`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (M), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | None |
| `iteration` | Iteration number — starts at `0`. Final value equals optimization function output `output.iterations`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (B), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | `Iteration`<br>See "Displaying Iterative Output" on page 2-69. |
| `lambda` | The Lagrange multipliers at the solution `x`. `lambda` is a structure where each field is for a different constraint type. For structure field names, see individual function descriptions. Final value equals optimization function output `lambda`. | `fgoalattain` (M), `fmincon` (M), `fminimax` (M), `fseminf` (M), `lsqcurvefit` (M), `lsqnonlin` (M) | None |
| `positivedefinite` | • `0` if algorithm detects negative curvature while computing Newton step<br>• `1` otherwise | `fmincon` (L), `lsqcurvefit` (L), `lsqnonlin` (L) | None |

| OptimValues Field (optimValues.field) | Description | Returned by Functions | Command-Line Display |
|---|---|---|---|
| `procedure` | Procedure messages | `fgoalattain` (M), `fmincon` (M), `fminimax` (M), `fseminf` (M) | `Procedure`<br>See "Displaying Iterative Output" on page 2-69. |
| `ratio` | Ratio of change in the objective function to change in the quadratic approximation | `fmincon` (L), `lsqcurvefit` (L), `lsqnonlin` (L) | None |
| `residual` | 2-norm of the residual squared. Final value equals optimization function output `residual`. | `lsqcurvefit` (B), `lsqnonlin` (B) | `Residual`<br>See "Displaying Iterative Output" on page 2-69. |
| `searchdirection` | Search direction | `fgoalattain` (M), `fmincon` (M), `fminimax` (M), `fminunc` (M), `fseminf` (M), `lsqcurvefit` (M), `lsqnonlin` (M) | None |
| `stepsize` | Current step size. Final value equals optimization function output `options.stepsize`. | `fgoalattain` (M), `fmincon` (B), `fminimax` (M), `fminunc` (B), `fseminf` (M), `lsqcurvefit` (B), `lsqnonlin` (B) | `Step-size`<br>See "Displaying Iterative Output" on page 2-69. |
| `trustregionradius` | Radius of trust region | `fmincon` (L), `lsqcurvefit`, `lsqnonlin` (L) | `Trust-region radius`<br>See "Displaying Iterative Output" on page 2-69. |

**Degeneracy.** The value of the field `degenerate`, which measures the degeneracy of the current optimization point `x`, is defined as follows. First, define a vector `r`, of the same size as `x`, for which `r(i)` is the minimum distance from `x(i)` to the *i*th entries of the lower and upper bounds, `lb` and `ub`. That is,

```
r = min(abs(ub-x, x-lb))
```

Then the value of `degenerate` is the minimum entry of the vector `r + abs(grad)`, where `grad` is the gradient of the objective function. The value of `degenerate` is 0 if there is an index `i` for which both of the following are true:

- `grad(i) = 0`
- `x(i)` equals the *i*th entry of either the lower or upper bound.

## States of the Algorithm

The following table lists the possible values for `state`:

| State | Description |
|---|---|
| `'init'` | The algorithm is in the initial state before the first iteration. |
| `'interrupt'` | The algorithm is in some computationally expensive part of the iteration. In this state, the output function can interrupt the current iteration of the optimization. At this time, the values of `x` and `optimValues` are the same as at the last call to the output function in which `state=='iter'`. |
| `'iter'` | The algorithm is at the end of an iteration. |
| `'done'` | The algorithm is in the final state after the last iteration. |

The following code illustrates how the output function might use the value of `state` to decide which tasks to perform at the current iteration.

```
switch state
    case 'iter'
          % Make updates to plot or guis as needed
    case 'interrupt'
          % Probably no action here. Check conditions to see
          % whether optimization should quit.
```

```
    case 'init'
          % Setup for plots or guis
    case 'done'
          % Cleanup of plots, guis, or final plot
otherwise
end
```

### Other Input Arguments

The argument `varargin` contains additional, problem-dependent arguments that you provide to the optimization function, which the function also passes to the objective function, `fun`, as well as the constraint functions `nonlcon` and `seminfcon`, if they exist. For example, if you call `fmincon` with the syntax

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2,...)
```

`fmincon` passes the arguments `P1, P2,` and so on to `outfun` as `varargin(1)`, `varargin(2)`, and so on. You can also specify these arguments explicitly in the first line of the output function, using the syntax

```
stop = outfun(x, optimValues, state, P1, P2, ...)
```

### Stop Flag

The output argument `stop` is a flag that is `true` or `false`. The flag tells the optimization function whether the optimization should quit or continue. The following examples show typical ways to use the `stop` flag.

**Stopping an Optimization Based on Data in optimValues.** The output function can stop an optimization at any iteration based on the current data in `optimValues`. For example, the following code sets `stop` to `true` if the directional derivative is less than `.01`:

```
function stop = outfun(x, optimValues)
stop = false;
% Check if directional derivative is less than .01.
if optimValues.directionalderivative < .01
    stop = true;
end
```

**Stopping an Optimization Based on GUI Input.** If you design a GUI to perform optimizations, you can make the output function stop an optimization when a user presses a **Stop** button on the GUI. The following code shows how to do

this, assuming that the **Stop** button callback stores the value `true` in the `optimstop` field of a `handles` structure called `hObject`.

```
function stop = outfun(x)
stop = false;
% Check if user has requested to stop the optimization.
stop = getappdata(hObject,'optimstop');
```