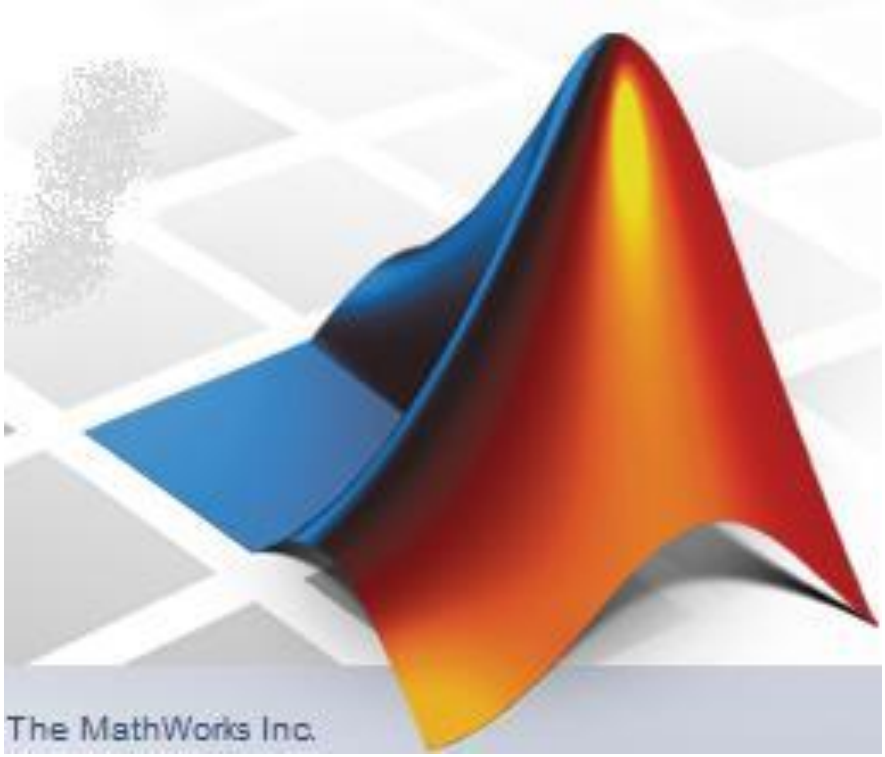
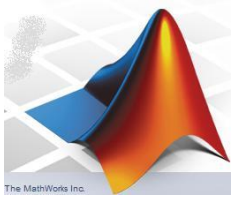
MATLAB (1) - úvod do programovania vedeckých problémov



LS 2018, 9a.predn.

Obsah (5. prednáška):

- fitovanie funkcií (Curve Fitting Toolbox)
- fitovanie funkcií (bez použitia tohot toolboxu)
- riešenie systémov lineárnych rovníc
- rýchla Fourierova transformácia (FFT) v Matlabe
- konvolúcia (v priestorovej oblasti)



dôležité príkazy – fitovanie nelineárnych funkcií:

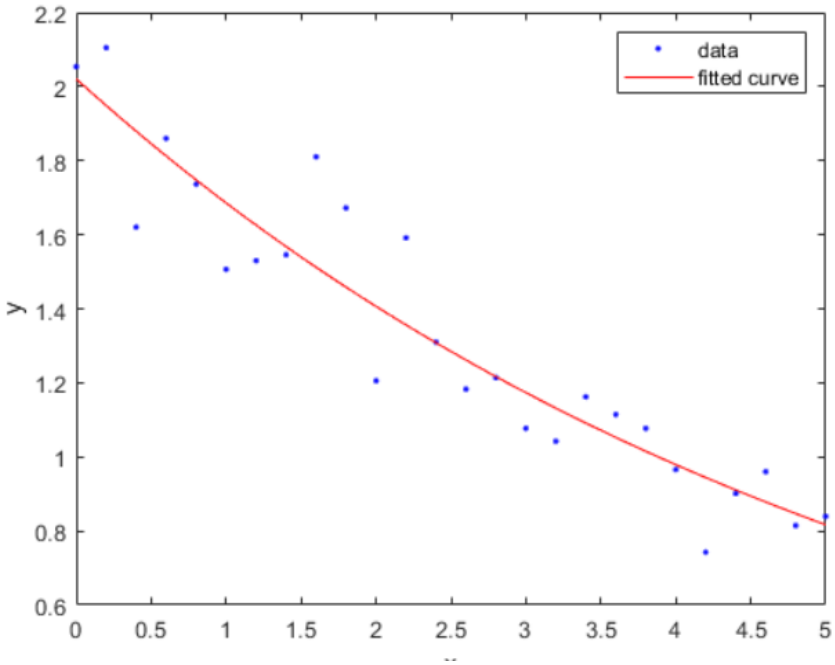
Použitie funkcie `fit()` z Curve Fitting Toolbox

Príklad: fitovanie expon. funkcie

```
x = (0:0.2:5)';
y = 2*exp(-0.2*x) + 0.1*randn(size(x));
f = fit(x,y,'exp1')
plot(f,x,y)
```

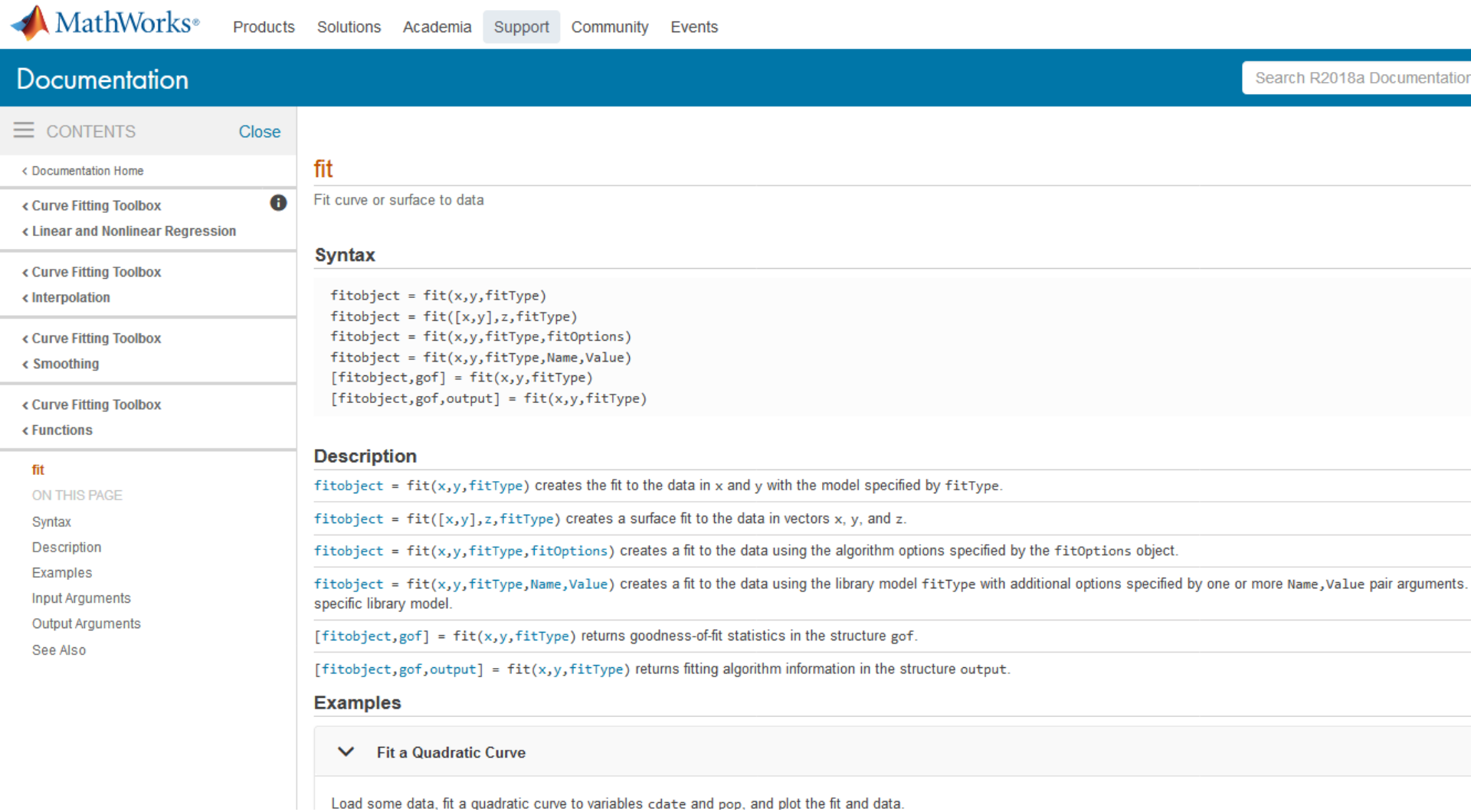
```
f =

General model Exp1:
f(x) = a*exp(b*x)
Coefficients (with 95% confidence bounds):
  a =       2.021  (1.89, 2.151)
  b =     -0.1812  (-0.2104, -0.152)
```



dôležité príkazy – fitovanie nelineárnych funkcií:

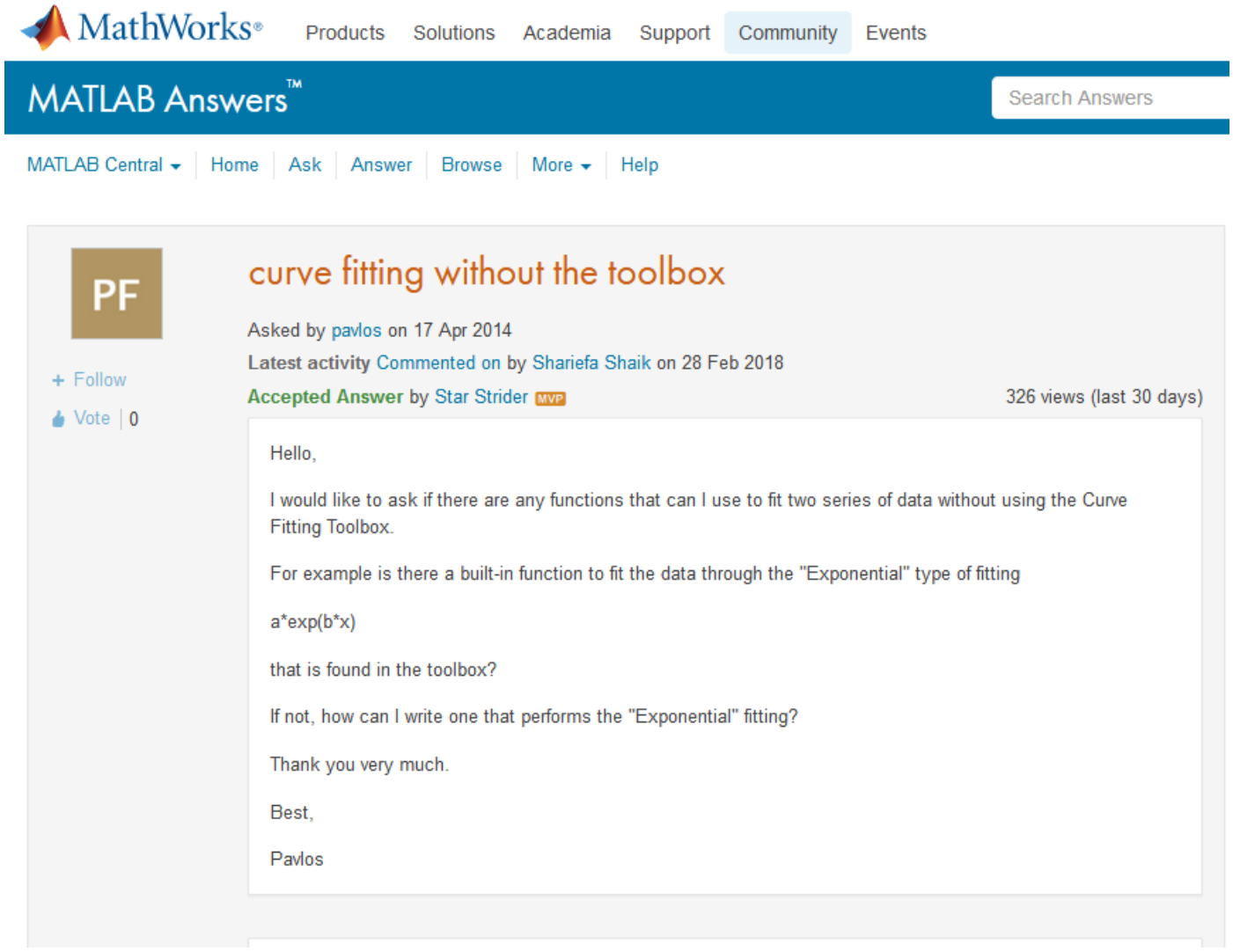
Použitie funkcie `fit()` z Curve Fitting Toolbox



dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

`https://www.mathworks.com/matlabcentral/answers/126146-curve-fitting-without-the-toolbox`



dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

https://www.mathworks.com/matlabcentral/answers/126146-curve-fitting-without-the-toolbox

There are the functions `lsqcurvefit` (Optimization Toolbox) and `nlinfit` (Statistics Toolbox) that will fit an objective function you provide. They each have their own advantages and disadvantages, depending upon what you want to do.

If you don't have those, using the MATLAB core function `fminsearch` can do the nonlinear fit with an additional line of code (the OLS cost function). (See the `fminsearch` documentation for details on what it does and how it works.)

This works:

```
y = @(b,x) b(1).*exp(-b(2).*x);             % Objective function

p = [3; 5]*1E-1;                            % Create data
x  = linspace(1, 10);
yx = y(p,x) + 0.1*(rand(size(x))-0.5);

OLS = @(b) sum((y(b,x) - yx).^2);          % Ordinary Least Squares cost function
opts = optimset('MaxFunEvals',50000, 'MaxIter',10000);
B = fminsearch(OLS, rand(2,1), opts)       % Use 'fminsearch' to minimise the 'OLS' functi
```

```
figure(1)
plot(x, yx, '*b')
hold on
plot(x, y(B,x), '-r')
hold off
grid
```

Skript `priklad_znetu.m`

dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

https://www.mathworks.com/matlabcentral/answers/126146-curve-fitting-without-the-toolbox

Z našej 6.prednášky – anonymné funkcie:

Anonymné funkcie

Anonymná funkcia je zjednodušená forma funkcie v Matlabe, ktorá nepotrebuje M-súbor. Pozostáva z jedného výrazu v Matlabe a rôzneho počtu vstupných alebo výstupných argumentov. Môže byť zadefinovaná v príkazovom riadku, ďalšej funkcii alebo skripte.

Syntax anonymnej funkcie je nasledujúci:

```
f = @ (argumenty) výraz
```

Samozrejme môžeme použiť niekoľko vstupných argumentov alebo žiadne.

Využitie anonymnej funkcie si môžeme ukázať na nasledujúcom príklade (2. mocnina):

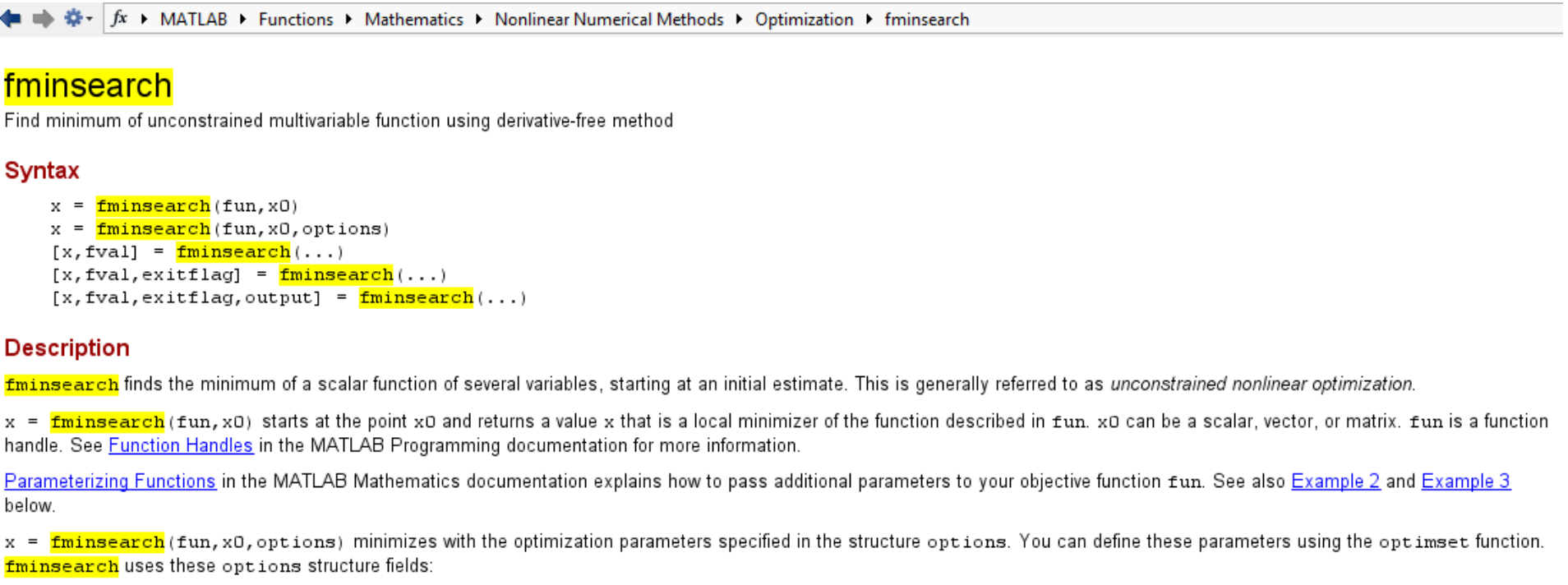
```
sqr = @(x) x.^2;
a = sqr(5)
```

dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

`https://www.mathworks.com/matlabcentral/answers/126146-curve-fitting-without-the-toolbox`

Z Helpu Matlabu – funkcia `fminsearch`:



MATLAB ▸ Functions ▸ Mathematics ▸ Nonlinear Numerical Methods ▸ Optimization ▸ fminsearch

fminsearch

Find minimum of unconstrained multivariable function using derivative-free method

Syntax

```
x = fminsearch(fun,x0)
x = fminsearch(fun,x0,options)
[x,fval] = fminsearch(...)
[x,fval,exitflag] = fminsearch(...)
[x,fval,exitflag,output] = fminsearch(...)
```

Description

`fminsearch` finds the minimum of a scalar function of several variables, starting at an initial estimate. This is generally referred to as *unconstrained nonlinear optimization*.

`x = fminsearch(fun,x0)` starts at the point `x0` and returns a value `x` that is a local minimizer of the function described in `fun`. `x0` can be a scalar, vector, or matrix. `fun` is a function handle. See Function Handles in the MATLAB Programming documentation for more information.

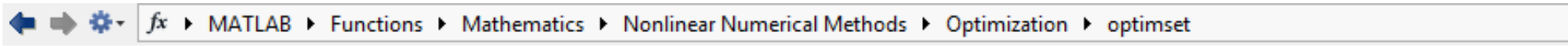
Parameterizing Functions in the MATLAB Mathematics documentation explains how to pass additional parameters to your objective function `fun`. See also Example 2 and Example 3 below.

`x = fminsearch(fun,x0,options)` minimizes with the optimization parameters specified in the structure `options`. You can define these parameters using the `optimset` function. `fminsearch` uses these `options` structure fields:

dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

`https://www.mathworks.com/matlabcentral/answers/126146-curve-fitting-without-the-toolbox`



optimset

Create or edit optimization options structure

Syntax

```
options = optimset('param1',value1,'param2',value2,...)
optimset
options = optimset
options = optimset(optimfun)
options = optimset(oldopts,'param1',value1,...)
options = optimset(oldopts,newopts)
```

Description

The function `optimset` creates an `options` structure that you can pass as an input argument to the following four MATLAB optimization functions:

- fminbnd
- fminsearch
- fzero
- lsqnonneg

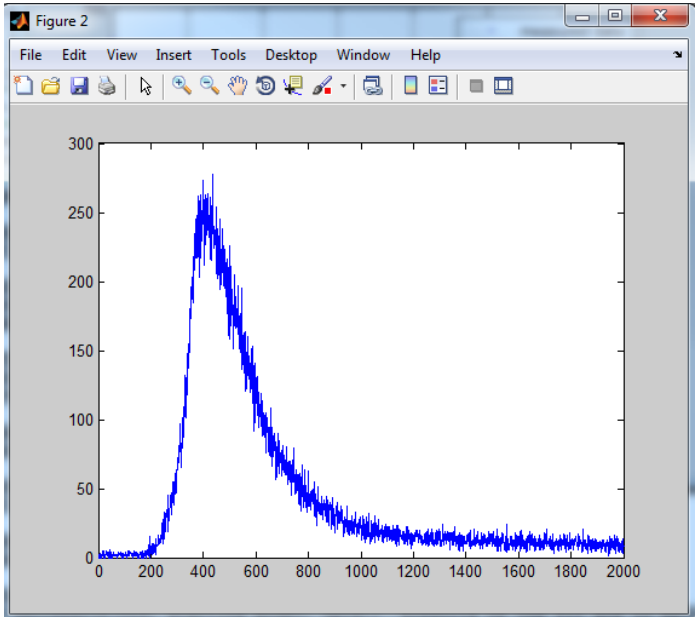
dôležité príkazy – fitovanie nelineárnych funkcií:

Bez použitia funkcie `fit()` z Curve Fitting Toolbox

Skript `fluoresc_processing.m` na fitovanie fluorescenčných dát

$$I_t = \alpha\, e^{-t/\tau}$$

It je fluorescenčná intenzita v čase t,alfa je predexponenciálny faktor a τ je fluorescenčná doba života.



ďalšie dôležité príkazy:

riešenie systému lineárnych rovníc:

$$Ax = b$$

kde *A* je tzv. matica systému *(m*-riadkov × *n*-stĺpcov*)* ,
x je vektor hľadaných neznámych – jednostĺpcová matica *(n*-riadkov × *1*-stĺpec*)*
a *b* tzv. pravá strana systému - jednostĺpcová matica *(m*-riadkov × *1*-stĺpec*)*

$$\begin{pmatrix} a_{11} & a_{12} & \ldots\ldots & a_{1n} \\ a_{21} & a_{22} & \ldots\ldots & a_{2n} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ a_{m1} & a_{m2} & \ldots\ldots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots\ldots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \ldots\ldots \\ b_m \end{pmatrix}$$

systém je riešiteľný, keď m = n (počet neznámych = počet rovníc),
prípadne keď m > n, kde ide o tzv. preurčený systém (rieši sa metódou LSQ)

ďalšie dôležité príkazy:

riešenie systému lineárnych rovníc: $Ax = b$

kde *A* je tzv. matica systému *(m*-riadkov × *n*-stĺpcov*)* ,
x je vektor hľadaných neznámych – jednostĺpcová matica *(n*-riadkov × *1*-stĺpec*)*
a *b* tzv. pravá strana systému - jednostĺpcová matica *(m*-riadkov × *1*-stĺpec*)*

- poznáme veľké množstvo metód – Matlab ich poskytuje v značnom množstve

⬇

- hlavný príkaz v Matlabe: *x = A\b;*

(\-‘backslash‘ na rozdiel od /-‘slash‘)
(*x=A/b* by riešil výpočet inverznej matice,
viď prednáška č.2, snímka č.12)

(tento je však „čiernou skrinkou = black box“ – nevieme, že pre akú metódu sa Matlab pri konečnej realizácii rozhodne)

príklad:

```
clear all; close all; clc;
A=magic(3)
b=[3; 1; 4]
x = A\b   % použitie tzv. „black-box“ metódy
```

ďalšie dôležité príkazy:

riešenie systému lineárnych rovníc: $Ax = b$

prehľad metód:

- hlavný príkaz v Matlabe: *x = A\b;*

- ďalšie možnosti:

  - LU-rozklad (LU-decomposition),

  - Choleského-rozklad (Cholesky-decomposition),

  - SVD-rozklad (SVD-decomposition),

  - QR-rozklad (QR-factorization),

  - použitie pseudo-inverznej matice (v prípade, že matica systému A je singulárna)

ďalšie dôležité príkazy:

riešenie systému lineárnych rovníc: Ax = b

príklady:

$$\begin{bmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix} \cdot \begin{bmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

```
clear all;
A=magic(3)
b=[3; 1; 4]
[L,U] = lu(A) % použitie LU-rozkladu
y = L\b   % rieši spodný trojuholníkový systém
x = U\y  % rieši horný trojuholníkový systém
```

```
clear all;
A=magic(3)
b=[3; 1; 4]
x = pinv(A)*b %použitie pseudo-inverznej matice
```

Moore–Penrose pseudoinverse

From Wikipedia, the free encyclopedia

In mathematics, and in particular linear algebra, a **pseudoinverse** A^+ of a matrix A is a generalization of the inverse matrix.[1] The Moore[2] in 1920, Arne Bjerhammar[3] in 1951 and Roger Penrose[4] in 1955. Earlier, Fredholm had introduced the concept of a pse to indicate the Moore–Penrose pseudoinverse. The term generalized inverse is sometimes used as a synonym for pseudoinverse.

```
clear all;
A=magic(3)
b=[3; 1; 4]
R = chol(A) %Choleského-rozklad; pozor matica A musí spĺňať určité
y = R'\b    %podmienky (tzv. pozitívne definitná, t.j. x^TAx > 0)
x = R\y
```

Instead of seeking arbitrary lower and upper triangular factors **L** and **U**, Cholesky decomposition constructs a lower triangular matrix **L** whose transpose $\mathbf{L}^T$ can itself serve as the upper triangular part. In other words we replace equation (2.3.1) by

$$\mathbf{L} \cdot \mathbf{L}^T = \mathbf{A} \qquad (2.9.2)$$

ďalšie dôležité príkazy:

riešenie systému lin. rovníc:

$$\begin{pmatrix} & \mathbf{A} & \end{pmatrix} = \begin{pmatrix} & \mathbf{U} & \end{pmatrix} \cdot \begin{pmatrix} w_1 & & & \\ & w_2 & & \\ & & \cdots & \\ & & \cdots & \\ & & & w_N \end{pmatrix} \cdot \begin{pmatrix} & \mathbf{V}^T & \end{pmatrix} \qquad (2.6.1)$$

príklady:

```
clear all;
A=magic(3)
b=[3; 1; 4]
x1 = A\b
[U,S,V] = svd(A)   %použitie SVD-rozkladu (Singular Value Decomposition)
for i=1: sqrt(numel(A))  %preklopenie diagonálnych prvkov matice S
SD(i,i) = 1/S(i,i);
end;
x = V*SD*(U'*b)
c = cond(A)  %tzv. condition number (hovorí o miere nestab. systému)
```

$$\mathbf{A}^{-1} = \mathbf{V} \cdot [\mathrm{diag}\,(1/w_j)] \cdot \mathbf{U}^T$$

```
clear all;
A=magic(3)
b=[3; 1; 4]
x1 = A\b
[Q,R] = qr(A) % použitie QR-rozkladu
y = Q\b
x = R\y
```

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R}$$

Here $\mathbf{R}$ is upper triangular, while $\mathbf{Q}$ is orthogonal, that is,

$$\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{1}$$

výpočet rýchlej Fourierovej transformácie (FFT)

Fourierova transformácia FT (priama FT, inverzná FT^{-1}):

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(u) e^{iux}\, du \ ,$$

$$F(u) = \mathcal{F}\{f(x)\},$$

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-iux}\, dx \ .$$

Funkcie `fft()` a `ifft()`. Výstupom z `fft()` je komplexná matica.

fx ▸ MATLAB ▸ Functions ▸ Data Analysis ▸ Fourier Transforms ▸ fft

fft

Fast Fourier transform

Syntax

```
Y = fft(x)
Y = fft(X,n)
Y = fft(X,[],dim)
Y = fft(X,n,dim)
```

Definitions

The functions `Y = fft(x)` and `y = ifft(X)` implement the transform and inverse transform pair given for vectors of length *N* by:

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)}$$

$$x(j) = (1/N)\sum_{k=1}^{N} X(k)\omega_N^{-(j-1)(k-1)}$$

where

$$\omega_N = e^{(-2\pi i)/N}$$

is an *N*th root of unity.

ďalšie dôležité príkazy:

príkaz: `conv(x,y)`

Konvolúcia: $$h_j = \sum_{i=-m/2}^{+m/2} g_{j-i}\, f_i$$

príklad: filtrácia náhodnej vygenerovanej funkcie pomocou jednoduchého 3-prvkového car-box filtra (kĺzajúci priemer)

close all; clear all; clc;
x1 = randn(100,1); % generuje maticu nahodnych cisel s rozmermi 100x1
*plot(x1); hold; %*nakresli graf funkcie x1 a pocka na dalsie kreslenie
xlabel('porad. cislo []','FontSize',10); ylabel('funkcie []','FontSize',10);

*b1 = [1; 1; 1]/3; %*definovanie 3-prvkoveho car-box filtra ako maticu 3x1

*y1 = conv(b1,x1); %*konvolucia dvoch signalov (matic)

*n1 = length(y1) %*dlzka vystupu konvolucie je n+m-1, kde m je dlzka filtra

*plot(y1,'--r') ; %*dokresli graf vyhladenej funkcie y1 cervenou prerusovanou
legend('vygenerovane','filtrovane');
% vykreslenie nie je uplne korektne… preco?