Variational Approach

$$\langle \psi | \hat{H} | \psi \rangle \geq E_0$$

where
- $|\psi\rangle$ → trial wave function
- E_0 → ground state energy

IDEA

$$|\psi\rangle = \sum_{j=1}^{n_x} c_j |x_j\rangle$$

Use delta basis function and obtain the best coefficients $\mathbf{c}_j$ that minimize the energy

STEPS for solving the problem

- Define parameters
- Define grids (with user-defined functions)
- Define Hamiltonian (with user-defined function)
- Minimize (with built–in function *fmincon*)
- Plot results

Define parameters (my_prob.m)

```
% Demonstration of the Variational theorem
% Variational calculation of the ground using the
% Hamiltonian defined by a Fourier Grid DVR
% September 30, 2016 - CHEM 470a/570a - Victor S.
Batista

clear;
global xmin xmax nx m dx dp x p H V;

% Define grids
m=1;
xmin=-5;   % minimum value of x
xmax=5;    % maximum value of x
nx=32;     % number of grid points in x
dx= (xmax-xmin)/nx;  % x grid spacing
dp= 2*pi/(xmax-xmin); % p grid spacing
```

define ***global*** variables that are going to be accessible to functions

define some parameters as mass, range of x axis number of points in x and grid spacing.

Define grids using a function

grids.m

```
function [x,p] = grids()
global dp dx nx;

for j=1:nx
    x(j)=dx*(j-nx/2); % grid in coordinates
end

np=nx;
for j=1:np
    p(j)=dp*(j-np/2); % grid in momentum
end
end
```

define which global variables are going to be used

position grid definition

momenta grid definition

my_prob.m

```
[x,p] = grids();
```

use user-defined function ***grids*** to define the grids **x** and **p**

Define Hamiltonian using a function

Hamiltonian.m

```
function [H,V] = Hamiltonian()
global dp dx nx m x p;
np=nx;

H=zeros(nx,nx);

for l=1:nx

    V(l)=0.5*x(l)^2;
    H(l,l)=V(l);

    for j=1:nx
        for k=1:np
            H(l,j)=H(l,j)+exp(i*(x(j)-x(l))*p(k))*p(k)^2/(2.0*m)*dx*dp/(2*pi);
        end
    end
end
```

define which global variables are going to be used

initialize Hamiltonian **H**

generate harminoc potential **V**

assign potential to diagonal element of Hamiltonian **H**

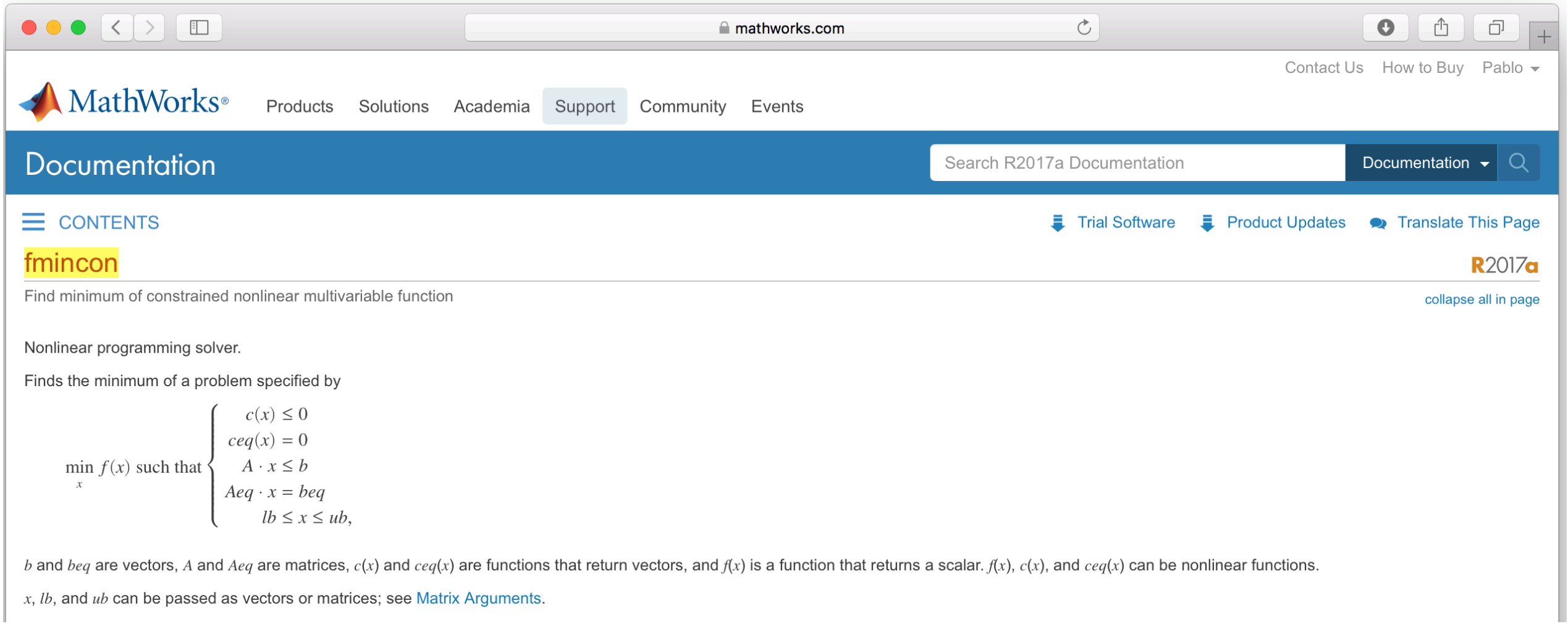
kinetic energy

my_prob.m

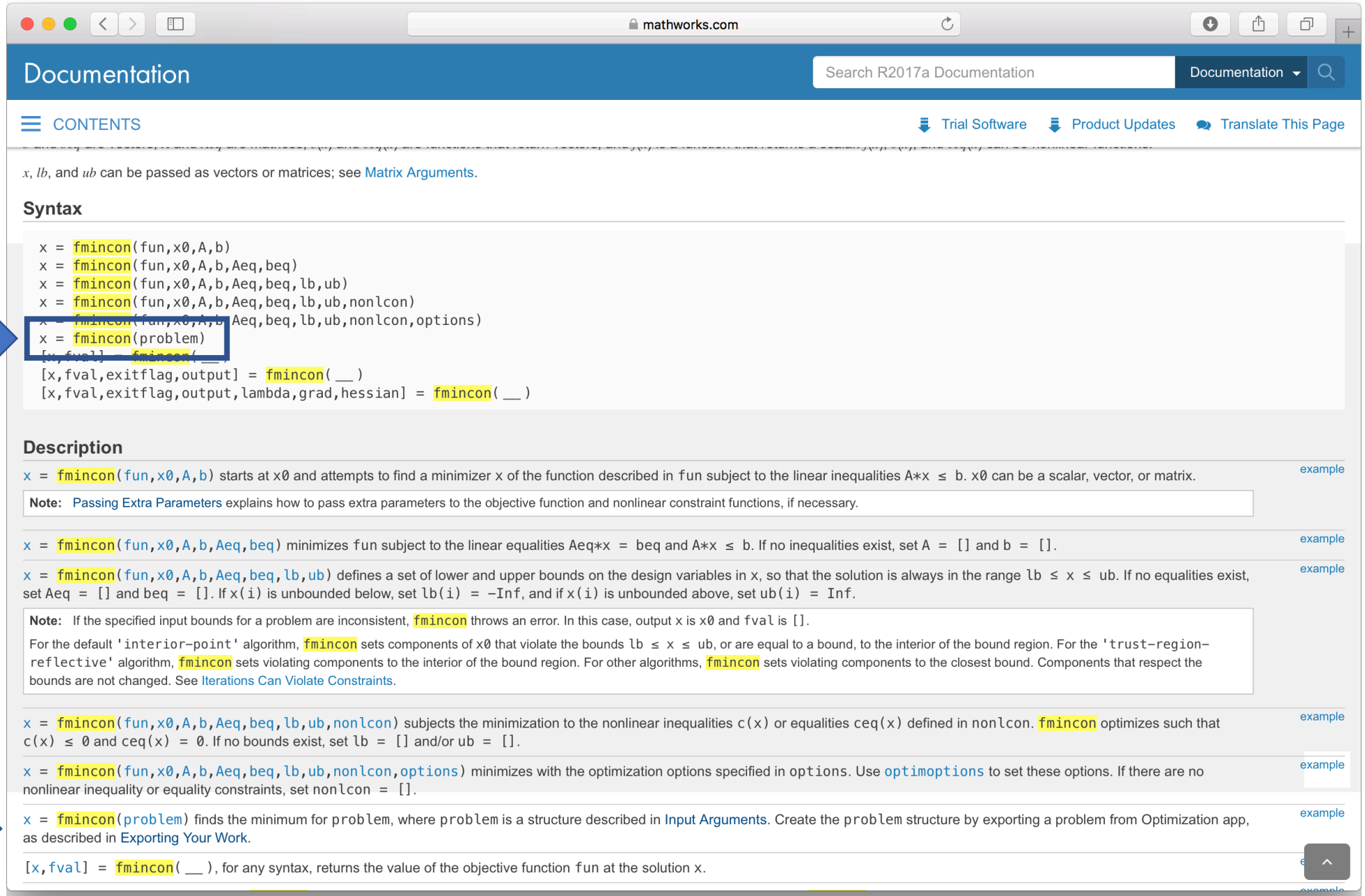
```
[H,V] = Hamiltonian();
```

use user-defined function ***Hamiltonian*** to define the Hamiltonian **H** and potential **V**

How to find the minimum: ***fmincon*** function



How to find the minimum: *fmincon* function



mathworks.com

Documentation

Search R2017a Documentation

Documentation

CONTENTS

Trial Software | Product Updates | Translate This Page

x, lb, and ub can be passed as vectors or matrices; see Matrix Arguments.

Syntax

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
[x,fval] = fmincon(__)
[x,fval,exitflag,output] = fmincon(__)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(__)
```

Description

`x = fmincon(fun,x0,A,b)` starts at `x0` and attempts to find a minimizer `x` of the function described in `fun` subject to the linear inequalities `A*x ≤ b`. `x0` can be a scalar, vector, or matrix. example

Note: Passing Extra Parameters explains how to pass extra parameters to the objective function and nonlinear constraint functions, if necessary.

`x = fmincon(fun,x0,A,b,Aeq,beq)` minimizes `fun` subject to the linear equalities `Aeq*x = beq` and `A*x ≤ b`. If no inequalities exist, set `A = []` and `b = []`. example

`x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)` defines a set of lower and upper bounds on the design variables in `x`, so that the solution is always in the range `lb ≤ x ≤ ub`. If no equalities exist, set `Aeq = []` and `beq = []`. If `x(i)` is unbounded below, set `lb(i) = -Inf`, and if `x(i)` is unbounded above, set `ub(i) = Inf`. example

Note: If the specified input bounds for a problem are inconsistent, `fmincon` throws an error. In this case, output `x` is `x0` and `fval` is `[]`.

For the default `'interior-point'` algorithm, `fmincon` sets components of `x0` that violate the bounds `lb ≤ x ≤ ub`, or are equal to a bound, to the interior of the bound region. For the `'trust-region-reflective'` algorithm, `fmincon` sets violating components to the interior of the bound region. For other algorithms, `fmincon` sets violating components to the closest bound. Components that respect the bounds are not changed. See Iterations Can Violate Constraints.

`x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)` subjects the minimization to the nonlinear inequalities `c(x)` or equalities `ceq(x)` defined in `nonlcon`. `fmincon` optimizes such that `c(x) ≤ 0` and `ceq(x) = 0`. If no bounds exist, set `lb = []` and/or `ub = []`. example

`x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)` minimizes with the optimization options specified in `options`. Use `optimoptions` to set these options. If there are no nonlinear inequality or equality constraints, set `nonlcon = []`. example

`x = fmincon(problem)` finds the minimum for `problem`, where `problem` is a structure described in Input Arguments. Create the `problem` structure by exporting a problem from Optimization app, as described in Exporting Your Work. example

`[x,fval] = fmincon(__)`, for any syntax, returns the value of the objective function `fun` at the solution `x`.

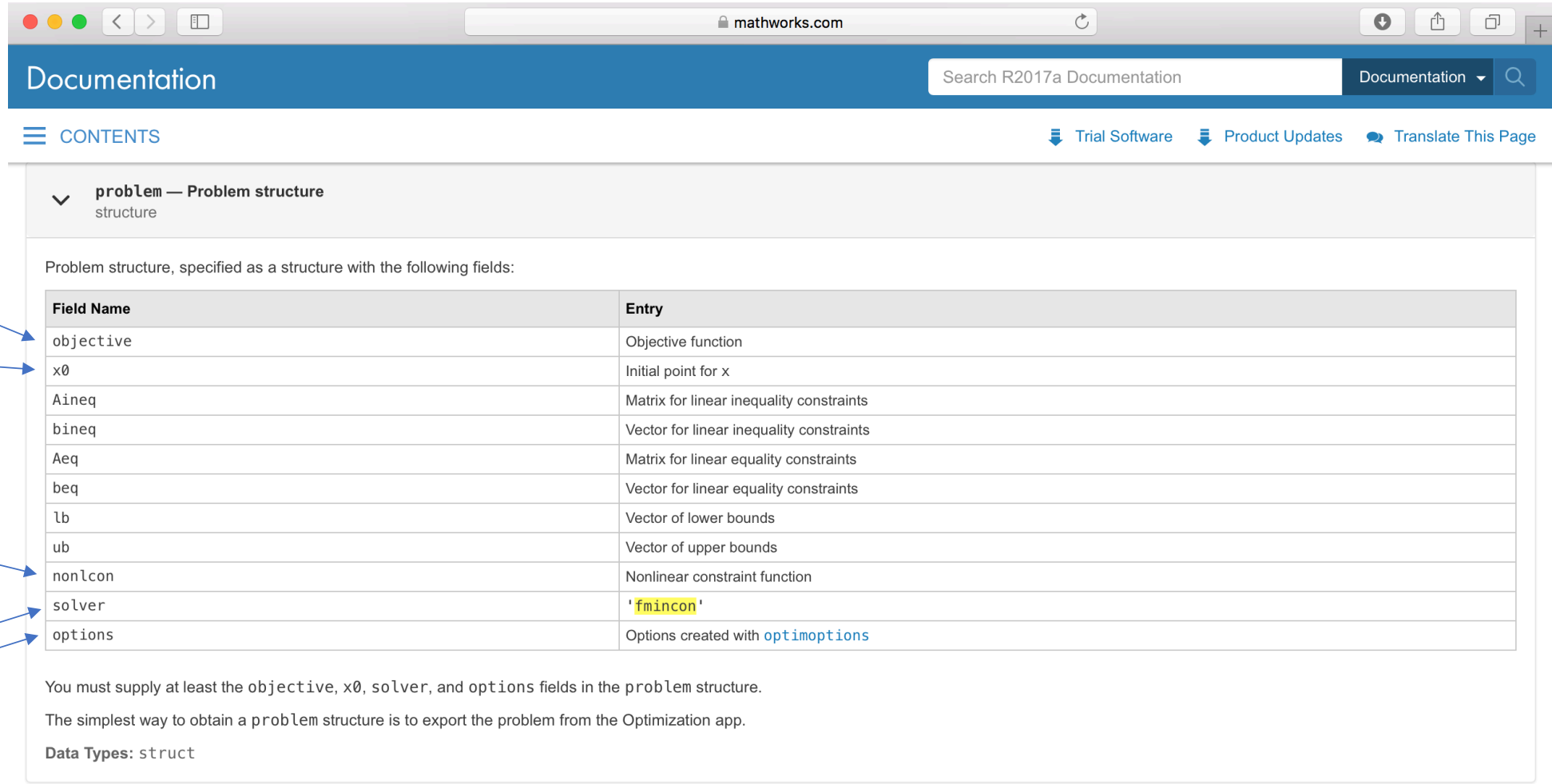
How to find the minimum: *fmincon* function

objective function
$\langle\psi|\hat{H}|\psi\rangle$

initial guess
$|\psi\rangle$

constraint
$\langle\psi|\psi\rangle = 1$

optimizer
options



Define and perform optimization

```
% Initial random vector
psi = rand(nx,1);

% Define optimization problem
options = optimoptions('fmincon','Algorithm','sqp');
problem.options = options;
problem.solver = 'fmincon';
problem.objective = @myobj;
problem.nonlcon = @normone;
problem.x0 = psi;

% Perform optimization
psi = fmincon(problem);
```

initialize trial function as random coefficients

objective function $\langle\psi|\hat{H}|\psi\rangle$

constraint $\langle\psi|\psi\rangle - 1 = 0$

initial guess $|\psi\rangle$

Functions definition

myobj.m

```
function [my] = myobj(psi)
% myobj is the expectation value of the
Energy
global nx H;
my=0;
 for j=1:nx
    for k=1:nx
        my = my+conj(psi(k))*H(k,j)*psi(j);
    end
 end
rn=0;
for j=1:nx
    rn = rn+conj(psi(j))*psi(j);
end
my=real(my)/real(rn);
end
```

normone.m

```
function [my,ceq] = normone(psi)
global dx;
my=norm(psi)^2*dx-1;
ceq = [];
end
```

Plot results

```
% Visualize results

[Energy] = myobj(psi);
[Normone,ceq] = normone(psi);
X = sprintf('Energy= %d, Norm= %d',Energy,Normone+1);
disp(X);
plot(x,Energy+psi);
hold on;
plot(x,Energy+psi*0);
plot(x,V);
hold off;
```