

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

```
% 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 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;
```

→ define which global variables are going to be used

```
for j=1:nx
```

```
    x(j)=dx*(j-nx/2); % grid in coordinates
```

```
end
```

} position grid definition

```
np=nx;
```

```
for j=1:np
```

```
    p(j)=dp*(j-np/2); % grid in momentum
```

```
end
```

```
end
```

} 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;
```

define which global variables are going to be used

```
H=zeros(nx,nx);
```

initialize Hamiltonian **H**

```
for l=1:nx
```

```
    V(l)=0.5*x(l)^2;
```

```
    H(l,l)=V(l);
```

generate harmonic potential **V**

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

```
    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
```

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

The screenshot shows a web browser window with the URL `mathworks.com`. The page is the documentation for the `fmincon` function in R2017a. The navigation bar includes links for Products, Solutions, Academia, Support, Community, and Events. The main content area features a search bar and a navigation menu with 'CONTENTS' and 'fmincon' highlighted. The function description states it is a nonlinear programming solver that finds the minimum of a constrained nonlinear multivariable function. The optimization problem is defined by the following constraints:

$$\min_x f(x) \text{ such that } \begin{cases} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub, \end{cases}$$

Below the equation, a note explains that b and beq are vectors, A and Aeq are matrices, $c(x)$ and $ceq(x)$ are functions that return vectors, and $f(x)$ is a function that returns a scalar. It also mentions that $f(x)$, $c(x)$, and $ceq(x)$ can be nonlinear functions. Finally, it states that x , lb , and ub can be passed as vectors or matrices; see [Matrix Arguments](#).

How to find the minimum: *fmincon* function

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

$x = \text{fmincon}(\text{fun},x_0,A,b)$ starts at x_0 and attempts to find a minimizer x of the function described in fun subject to the linear inequalities $A*x \leq b$. x_0 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.

$x = \text{fmincon}(\text{fun},x_0,A,b,Aeq,beq)$ minimizes fun subject to the linear equalities $Aeq*x = beq$ and $A*x \leq b$. If no inequalities exist, set $A = []$ and $b = []$. [example](#)

$x = \text{fmincon}(\text{fun},x_0,A,b,Aeq,beq,lb,ub)$ defines a set of lower and upper bounds on the design variables in x , so that the solution is always in the range $lb \leq x \leq ub$. If no equalities exist, set $Aeq = []$ and $beq = []$. If $x(i)$ is unbounded below, set $lb(i) = -\text{Inf}$, and if $x(i)$ is unbounded above, set $ub(i) = \text{Inf}$. [example](#)

Note: If the specified input bounds for a problem are inconsistent, `fmincon` throws an error. In this case, output x is x_0 and $fval$ is $[]$.

For the default 'interior-point' algorithm, `fmincon` sets components of x_0 that violate the bounds $lb \leq x \leq ub$, 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](#).

$x = \text{fmincon}(\text{fun},x_0,A,b,Aeq,beq,lb,ub,nonlcon)$ subjects the minimization to the nonlinear inequalities $c(x)$ or equalities $ceq(x)$ defined in nonlcon . `fmincon` optimizes such that $c(x) \leq 0$ and $ceq(x) = 0$. If no bounds exist, set $lb = []$ and/or $ub = []$. [example](#)

$x = \text{fmincon}(\text{fun},x_0,A,b,Aeq,beq,lb,ub,nonlcon,options)$ minimizes with the optimization options specified in $options$. Use [optimoptions](#) to set these options. If there are no nonlinear inequality or equality constraints, set $\text{nonlcon} = []$. [example](#)

$x = \text{fmincon}(\text{problem})$ 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](#)

$[x,fval] = \text{fmincon}(_)$, 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

Documentation

Search R2017a Documentation

CONTENTS

Trial Software Product Updates Translate This Page

problem — Problem structure structure

Problem structure, specified as a structure with the following fields:

Field Name	Entry
objective	Objective function
x0	Initial point for x
Aineq	Matrix for linear inequality constraints
bineq	Vector for linear inequality constraints
Aeq	Matrix for linear equality constraints
beq	Vector for linear equality constraints
lb	Vector of lower bounds
ub	Vector of upper bounds
nonlcon	Nonlinear constraint function
solver	' fmincon '
options	Options created with optimoptions

You must supply at least the objective, x0, solver, and options fields in the problem structure.

The simplest way to obtain a problem structure is to export the problem from the Optimization app.

Data Types: struct

Define and perform optimization

```
% Initial random vector
```

```
psi = rand(nx,1);
```

→ initialize trial function as random coefficients

```
% Define optimization problem
```

```
options = optimoptions('fmincon','Algorithm','sqp');
```

```
problem.options = options;
```

```
problem.solver = 'fmincon';
```

```
problem.objective = @myobj;
```

```
problem.nonlcon = @normone;
```

```
problem.x0 = psi;
```

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

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

initial guess $|\psi\rangle$

```
% Perform optimization
```

```
psi = fmincon(problem);
```

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;
```