Nonlinear example: The Code for Fig 7, 8 and 10

- Fig 7: Consider all 1-local observables(N_O=19) and all 2-local observables(N_O=154) at time $t_n = 0.1, 0.2, \cdots, 1(N_T = 10)$ and initial guess $\|\theta^{(0)} - \theta^*\|$ = 0.4529 in thetas.mat
- Fig 8: Consider fewer 1-local observables(N_O=12) at time $t_n = 0.1, 0.2, \cdots, 1(N_T = 10)$ and initial guess $\|\theta^{(0)} - \theta^*\|$ = 0.4529 in thetas.mat
- Fig 10: Consider all 1-local observables(N_O=19) at time $t_n = 0.1, 0.2, \cdots, 1(N_T = 10)$ and initial guess $\|\theta^{(0)} - \theta^*\|$ = 2.0359 in thetas_further3.mat

Table of Contents

Main function

```
[theta_s, theta0, theta_after,fval,history,output,exitflag] = runfminunc;
save('exp25.2.mat','exitflag','output','history',"fval","theta_after","theta
_s")
```

- exp25.3.mat for all 1-local observable case in Fig 7
- exp25.2.mat for all 1&2-local observable case in Fig 7
- exp25.4.mat for fewer 1-local observables in Fig 8
- exp25.3.further.3.mat for larger initial guess in Fig10

Graph the Performance of Optimization: observableCP.m

Fig 7: from exp25.2.mat and exp25.3.mat

```
clear
clc
fig1 = load('exp25.3.mat');
fig2 = load('exp25.2.mat');
```

```
NT = 10;
NO1 = 19;
NO2 = 154;
tiledlayout(2,2)
ax3 = nexttile;
plot(ax3,fig1.output.iteration, fig1.output.resnorm.^2./(2*NO1*NT),'-
*',fig2.output.iteration, fig2.output.resnorm.^2./(2*NO2*NT),'--
*','LineWidth',2)
grid on
set(gca,"FontSize",34,'FontName','Times','fontweight','bold')
legend('With 1-local observables','With 2-local
observables','fontsize',30,'interpreter','latex')
xlabel('Iteration');
ylabel('$\phi(\theta)$','interpreter','latex');
title('Function Value $\phi(\theta)$','interpreter','latex')
ax1 = nexttile([2,1]);
plot(ax1,fig1.output.iteration, fig1.output.error3H,'-
o',fig1.output.iteration, fig1.output.error3D,'-+',fig2.output.iteration,
fig2.output.error3H,'--o',fig2.output.iteration, fig2.output.error3D,'--
+','LineWidth',2)
grid on
set(gca,"FontSize",32,'FontName','Times','fontweight','bold')
legend('Error $\|\theta_H-\theta_H^*\|$ with 1-local observables','Error $\|
\theta_D-\theta_D^*\|$ with 1-local observables','Error $\|\theta_H-
\theta_H^*\|$ with 2-local observables','Error $\|\theta_D-\theta_D^*\|$
with 2-local
observables','interpreter','latex','location','best','fontweight','bold','fo
ntsize',26)
xlabel('Iteration');
ylabel('Error')
title('Error for Hamiltonian and Dissipative
Coefficients','interpreter','latex')
ax4 = nexttile;
plot(ax4,fig1.output.iteration, fig1.output.error4,'-
*',fig2.output.iteration, fig2.output.error4,'--*','LineWidth',2)
grid on
set(gca,"FontSize",34,'FontName','Times','fontweight','bold')
legend('With 1-local observables','With 2-local
observables','fontsize',30,'interpreter','latex')
xlabel('Iteration');
ylabel('Relative Error')
title('Relative Error $\frac{||\theta-\theta^*||}{||\theta^*||}
$','interpreter','latex')
print('exp25.2&3cp.jpg','-djpeg')
```

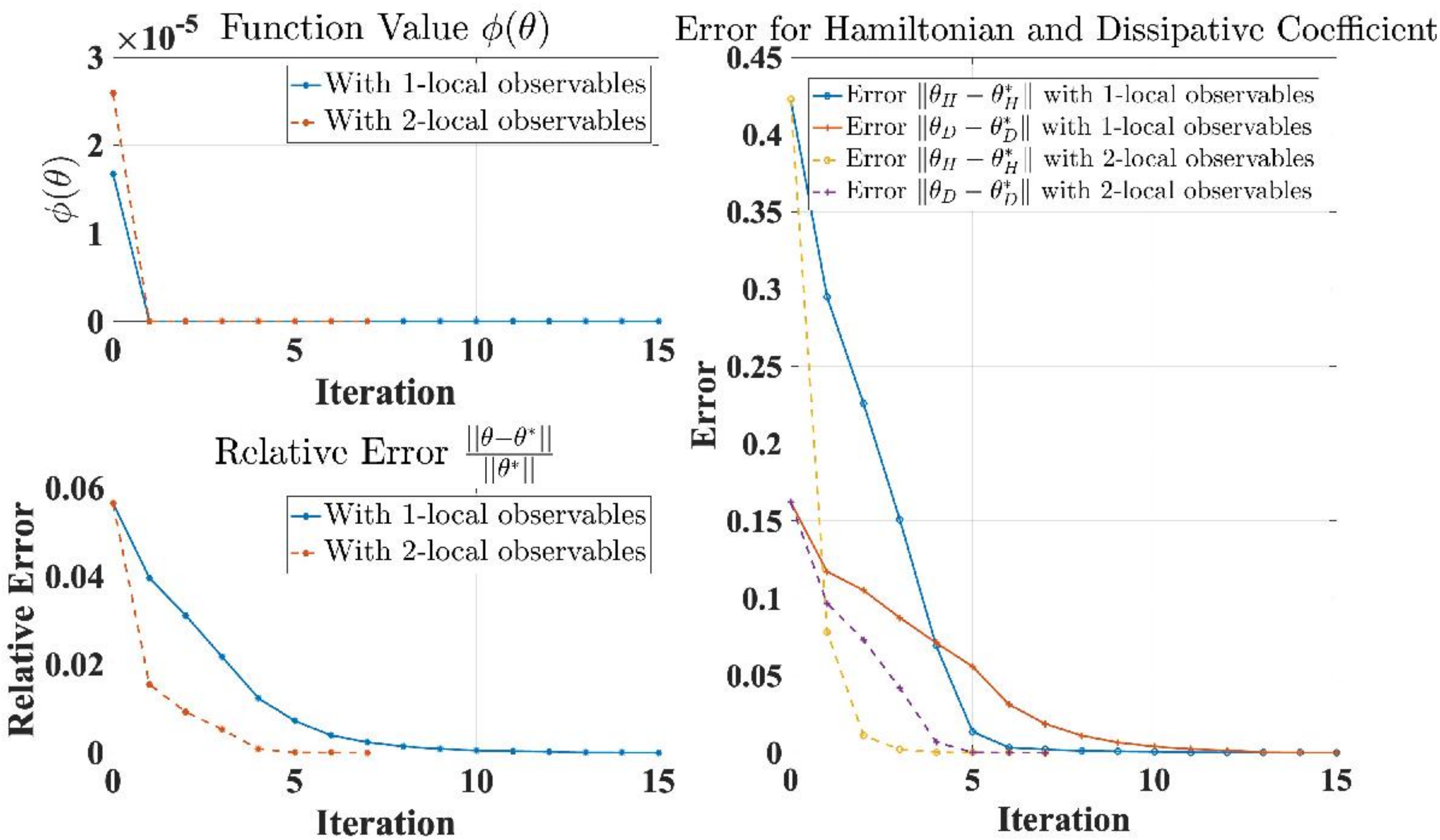


Fig8: from exp25.4.mat

```
load('exp25.4.mat')
tiledlayout(2,2)
NT = 10;
NO = 12;
ax3 = nexttile;
plot(ax3,output.iteration, output.resnorm.^2/(2*NO*NT),'-*','LineWidth',2)
grid on
set(gca,"FontSize",24,'FontName','Times')
xlabel('Iteration')
ylabel('Objective function value')
title('Function Value $\phi(\theta)$','interpreter','latex')
ax1 = nexttile([2,1]);
plot(ax1,output.iteration, output.error3H,'-o',output.iteration,
output.error3D,'-o','LineWidth',2)
grid on
set(gca,"FontSize",24,'FontName','Times')
legend('Error $\|\theta_H-\theta_H^*\|$','Error $\|\theta_D-\theta_D^*\|
$','interpreter','latex','location','best')
xlabel('Iteration');
ylabel('Error for Hamiltonian coefficients')
title('Error for Hamiltonian part and Dissipation
part','interpreter','latex')
ax4 = nexttile;
```

```
plot(ax4,output.iteration, output.error4,'-*','LineWidth',2)
grid on
set(gca,"FontSize",24,'FontName','Times')
xlabel('Iteration');
ylabel('Relative Error')
title('Relative Error $(\theta-\theta_s)/|\theta_s|$','interpreter','latex')
print('exp25.4.6spinsFormal3pics.jpg','-djpeg')
```

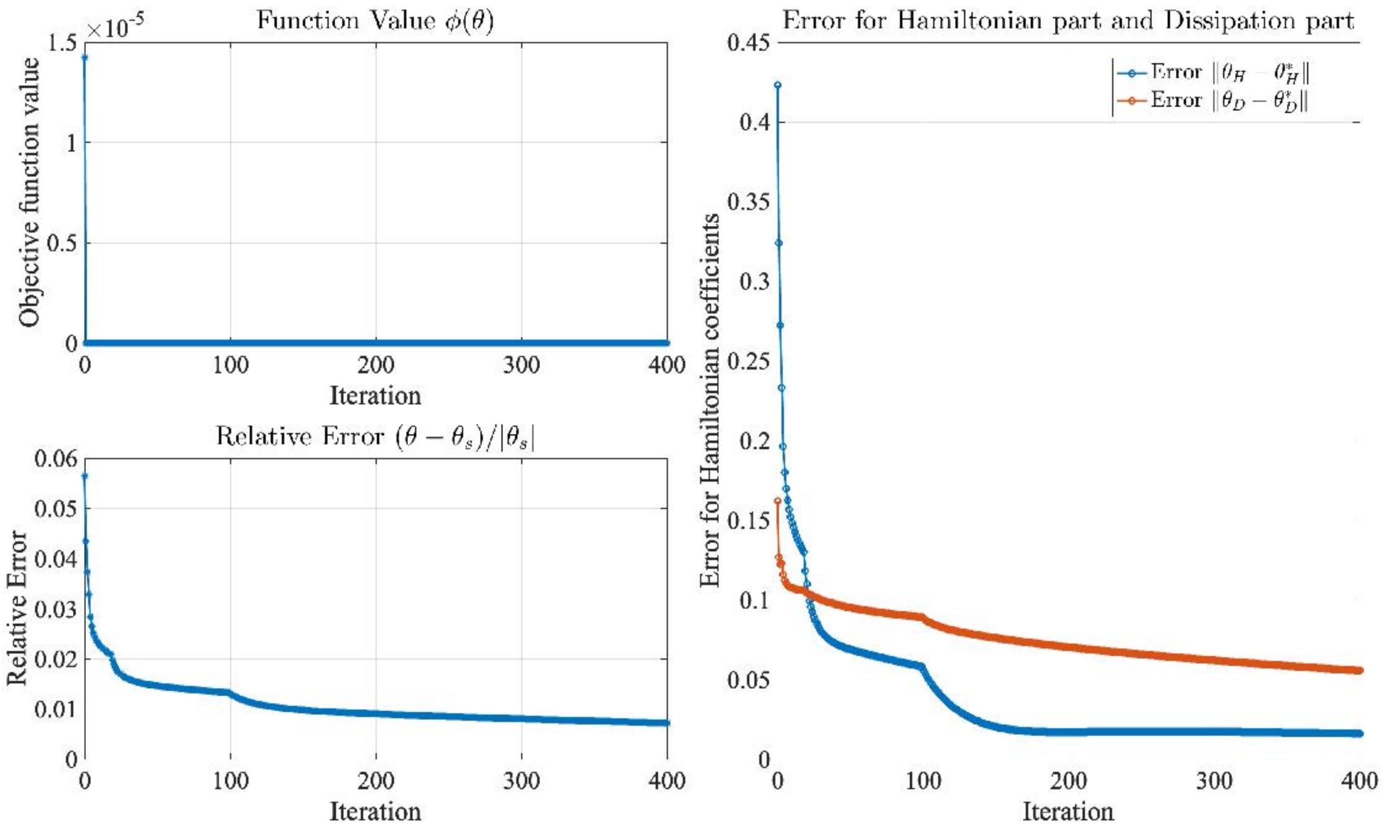
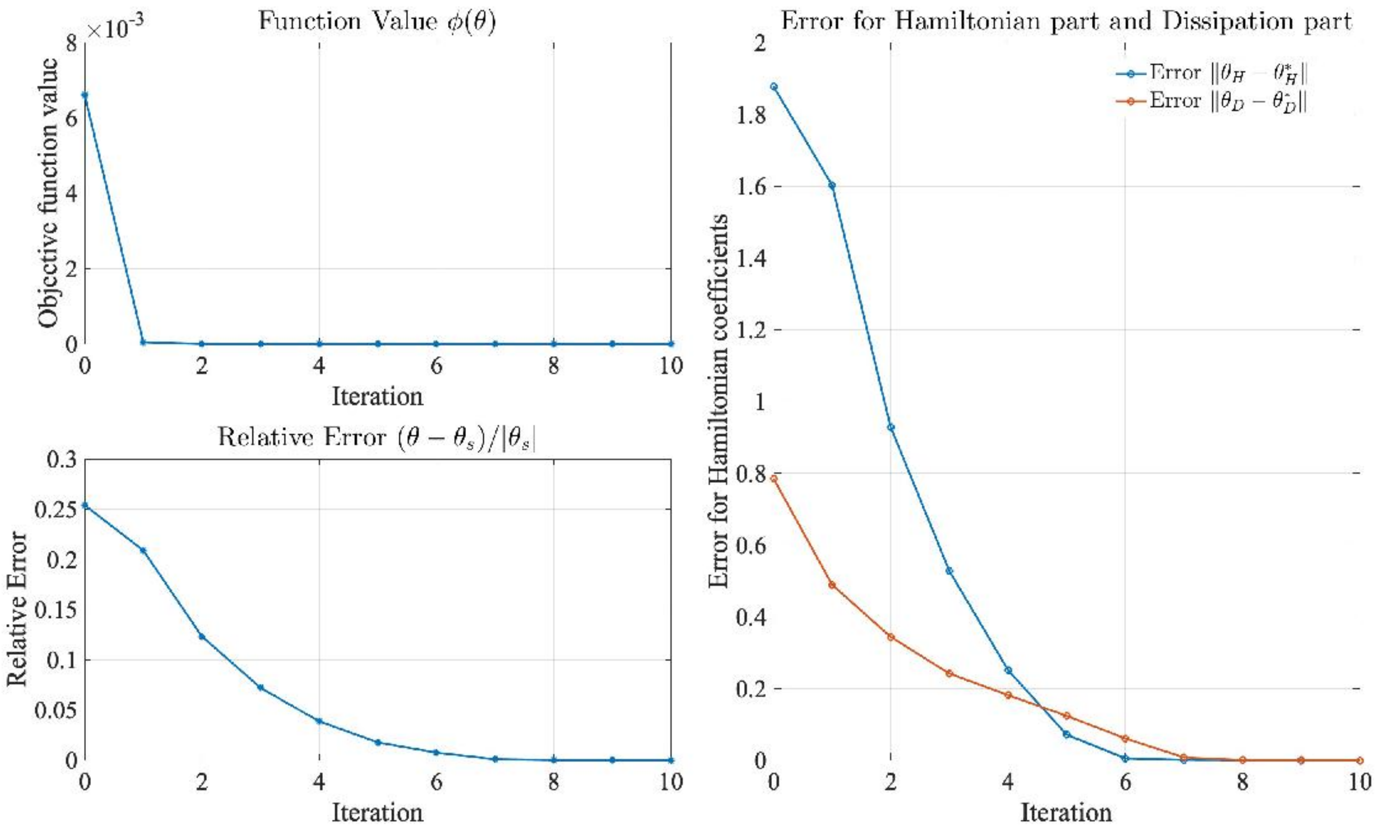


Fig10: from exp25.3.further.3.mat



Implement runfminunc.m

```
function [theta_s, theta0, theta_after,resnorm, residual,
history,output,exitflag] = runfminunc

% Set up shared variables with outfun
history.theta = [];
history.resnorm = [];
history.residual = [];
history.g = [];
history.error3H = [];
history.error3D = [];
history.error4 = [];
history.iteration = [];

N = 6; % number of spins
pauli_num = 3;
%% load theta
load('thetas.mat'); % dissipation parameter contains both d1 and d2
error_in = norm(theta_s-theta0)
```

```
T0 = 0;
T1 = 0;
rho0 = gen_rho(N);
T = T1+1; % end time
delta_t=0.01;
Lt = 10;
Delta_t = delta_t*Lt;
% generate sigma
sigma = cell(N,pauli_num);
[A, Anum] = genManyA(1,N);
sigma0 = zeros(2,2,3);
sigma0(:,:,1) = [0, 1; 1, 0];   % pauli^x
sigma0(:,:,2) = [0, -1i; 1i,0]; % pauli^y
sigma0(:,:,3) = [1, 0; 0, -1];  % pauli^z
I = eye(2);

for j = 1:N
    for a = 1:pauli_num
        sigma{j,a} = sigma0(:,:,a);
        for i = 1:N
            if i<j
                sigma{j,a} = kron(I,sigma{j,a});
            else
                if i>j
                    sigma{j,a} = kron(sigma{j,a},I);
                end
            end
        end
    end
end

% generate y_exact
fun0 =
@(theta)gen_mea_data2(N,theta,sigma,Delta_t,delta_t,Lt,T,A,rho0,Anum,T0,T1);
y_exact = fun0(theta_s);
fun1 =
@(theta)gradPhilsq2(N,theta,y_exact,sigma,Delta_t,delta_t,Lt,T,A,rho0,Anum,T
0,T1);

% Optimization
options = optimoptions('lsqnonlin','PlotFcn',@outfun,'Display','iter-
detailed','SpecifyObjectiveGradient',true,'Algorithm','levenberg-
marquardt');
[theta_after, resnorm,residual, exitflag, output] = lsqnonlin(fun1,theta0,
[],[],options);

function stop = outfun(theta,optimValues,state)
     stop = false;
     spins = 6;
     pauli_n=3;
```

```
      Hnum = spins*pauli_n+(spins-1)*pauli_n*pauli_n;
      switch state
          case 'init'
          case 'iter'
            history.resnorm = [history.resnorm; optimValues.resnorm];
            history.theta = [history.theta, theta];
            history.residual = [history.residual, optimValues.residual];
            history.g = [history.g, optimValues.gradient];
            history.iteration = [history.iteration; optimValues.iteration];
            error3 = norm(theta-theta_s)
            error3H = norm(theta(1:Hnum)-theta_s(1:Hnum))
            history.error3H = [history.error3H; error3H];
            error3D = norm(theta(Hnum+1:end)-theta_s(Hnum+1:end))
            history.error3D = [history.error3D; error3D];
            error4 = error3/norm(theta_s)
            history.error4 = [history.error4;error4];
          case 'done'
               tiledlayout(2,2)
               ax1 = nexttile;
               plot(ax1,history.iteration, history.error3H,'LineWidth',2)
               grid on
               set(gca,"FontSize",24,'FontName','Times')
               xlabel('iteration');
               ylabel('Error for Hamiltonian coefficients')
               title('Error $\theta_H-\theta_H^*$','interpreter','latex')
               ax2 = nexttile;
               plot(ax2,history.iteration, history.error3D,'LineWidth',2)
               grid on
               set(gca,"FontSize",24,'FontName','Times')
               xlabel('iteration');
               ylabel('Error for Dissipation coefficients')
               title('Error $\theta_D-\theta_D^*$','interpreter','latex')
               ax3 = nexttile;
               plot(ax3,history.iteration, history.resnorm,'LineWidth',2)
               grid on
               set(gca,"FontSize",24,'FontName','Times')
               xlabel('iteration');
               ylabel('Objective function value')
               title('Function Value $\phi(\theta)$','interpreter','latex')
               ax4 = nexttile;
               plot(ax4,history.iteration, history.error4,'LineWidth',2)
               grid on
               set(gca,"FontSize",24,'FontName','Times')
               xlabel('iteration');
               ylabel('Relative Error')
               title('Relative Error $(\theta-\theta_s)/|\theta_s|
$','interpreter','latex')
          otherwise
      end
  end
```

```
end
```

Prepare the Given measurements y^*: gen_mea_data2.m

```
function y=
gen_mea_data2(N,theta,sigma,Delta_t,delta_t,Lt,T,A,rho0,Anum,T0,T1)
% generate measurement data y_n for 6 spin example, N: the number of spins
[e,c,d1,d2] = theta2tensor2(theta,N);
sz = 2^N;                   % the size of density matrix
pauli_num = 3;
t0 = (T0:delta_t:T);        % Simulation Time Points
iter_num = length(t0)-1;
t1 = (T1:Delta_t:T);        % Measurements Time Points
mea_num = length(t1)-1;
H = zeros(sz,sz,N);         % Hamiltonian
jump_num = N;               % the number of jump operator
L = zeros(sz,sz,jump_num); % jump operators
%% Calculate Hamiltonian H, Jump operator L and G
for j = 1: N-1
    for a = 1: pauli_num
        for b = 1:pauli_num
            H(:,:,j) = H(:,:,j)+c(j,a,b)*sigma{j,a}*sigma{j+1,b};
        end
        H(:,:,j) = H(:,:,j)+e(j,a)*sigma{j,a};
    end
end
for a = 1: pauli_num
    H(:,:,N) = H(:,:,N)+e(N,a)*sigma{N,a};
end
for j = 1:N
    L(:,:,j) = L(:,:,j)+d1(j,1)*sigma{j,1};
    for a = 2:pauli_num
        L(:,:,j) = L(:,:,j)+d1(j,a)*sigma{j,a}+1i*d2(j,a-1)*sigma{j,a};
    end
end
G = zeros(sz,sz);
F_num = jump_num^2+jump_num+1;
F = zeros(sz,sz,F_num);
for j = 1: N
    G = G-1i*H(:,:,j);
end
for j = 1:jump_num
    G = G-L(:,:,j)'*L(:,:,j)/2;
end
%% 2.0 implicit Taylor Scheme of SSE -> Kraus form
y = zeros(mea_num+1,Anum); % measurement y_i, i = 0,1,...,mea_num
I2 = eye(sz);               % Identity matrix
% generate F_j
op1 = I2-G* delta_t./2;
```

```
op2 = I2+G* delta_t./2;
F(:,:,1) = op1\op2;
for j = 2: jump_num+1
    F(:,:,j) = op1\L(:,:,j-1)*op2*sqrt(delta_t);
end
for i = 1:jump_num
    for j = 1:jump_num
        F(:,:,i+jump_num*j+1) = L(:,:,i)*L(:,:,j)*delta_t/sqrt(2);
    end
end

% Kraus operator K acts on rho
rho_s = zeros(sz,sz,iter_num+1);
for i = 1:iter_num+1
    if i == 1
        rho_s(:,:,i) = rho0;
    else
        for j = 1:F_num
            rho_s(:,:,i) = rho_s(:,:,i)+F(:,:,j)*rho_s(:,:,i-1)*F(:,:,j)';
        end
    end
end
start = (T1-T0)/delta_t+1;
for i = 1:mea_num+1
    for k = 1:Anum
        id = start+(i-1)*Lt;
        y(i,k) = real(trace(A{k}*rho_s(:,:,id))); % y exact
    end
end
end
```

Calculating the Loss Function Value and its Gradients gradPhilsq2.m

```
function [y,J2] = gradPhilsq2
(N,theta,y_exact,sigma,Delta_t,delta_t,Lt,T,A,rho_0,Anum,T0,T1)
% generate measurement data y_n for 6 spin example
% N:the number of spins
[e,c,d1,d2] = theta2tensor2(theta,N);
sz = 2^N;      % the size of density matrix
pauli_num = 3;
t0 = (T0:delta_t:T); % Simulation Time Points
iter_num = length(t0)-1;
t1 = (T1:Delta_t:T); % Measurements Time Points
mea_num = length(t1)-1;
start = (T1-T0)/delta_t+1;
y_s = zeros(mea_num+1,Anum); % measurements
theta_num = numel(theta);                % the number of parameter theta
rho_s = zeros(sz,sz,iter_num+1);         % density operator at different
time points
tilde_s = cell(theta_num, iter_num,Anum);    % to compute tr(A\chi_n)
```

```
tr_value = zeros(theta_num, iter_num,Anum);  % tr(A\chi_n) to calculate
gradient, the measurement of chi_n
KKrausA_s = zeros(sz,sz,iter_num,Anum);      % save the value of (K^*)^{k-1}
[A{}], k =1,2,...,n
J1 = zeros(theta_num,mea_num,Anum);

%% Calculate Hamiltonian H, Jump operators L and G
H = zeros(sz,sz,N);         % Hamiltonian
jump_num = N;
L = zeros(sz,sz,jump_num); % Jump operators
F_num = jump_num^2+jump_num+1;
F_s = zeros(sz,sz,F_num);
for j = 1: N-1
    for a = 1: pauli_num
        for b = 1:pauli_num
            H(:,:,j) = H(:,:,j)+c(j,a,b)*sigma{j,a}*sigma{j+1,b}; % can be
simplified
        end
        H(:,:,j) = H(:,:,j)+e(j,a)*sigma{j,a};
    end
end
for a = 1: pauli_num
    H(:,:,N) = H(:,:,N)+e(N,a)*sigma{N,a};
end
for j = 1:N
    L(:,:,j) = L(:,:,j)+d1(j,1)*sigma{j,1};
    for a = 2:pauli_num
        L(:,:,j) = L(:,:,j)+d1(j,a)*sigma{j,a}+1i*d2(j,a-1)*sigma{j,a};
    end
end
G = zeros(sz,sz);
for j = 1: N
    G = G-1i*H(:,:,j);
end
for j = 1:jump_num
    G = G-L(:,:,j)'*L(:,:,j)/2;
end

% generate Kraus operators F_s
I2 = eye(sz);
op1 = I2-G* delta_t./2;
op2 = I2+G* delta_t./2;
F_s(:,:,1) = op1\op2;
for j = 2: jump_num+1
    F_s(:,:,j) = op1\L(:,:,j-1)*op2.*sqrt(delta_t);
end
for i = 1:jump_num
    for j = 1:jump_num
        F_s(:,:,i+jump_num*j+1) = L(:,:,i)*L(:,:,j)*delta_t/sqrt(2);
    end
```

```
end
%% Kraus form = \sum_{j} F_s{j}\rho_s{j}F_s{j}', \rho_{i-1} =
Kraus^{i-1}\rho_0
% loss function phi = \sum_{i=1}^mea_num (y_i-y_i^*)^2/2
for i = 1:iter_num+1
    if i == 1
        rho_s(:,:,i) = rho_0; % rho_s(:,:,1) save the value of \rho_0
    else
        for j = 1: F_num
            rho_s(:,:,i) = rho_s(:,:,i)
+F_s(:,:,j)*rho_s(:,:,i-1)*F_s(:,:,j)';
        end
    end
end

for i = 1:mea_num+1
    for k = 1:Anum
        id = start+(i-1)*Lt;
        y_s(i,k) = real(trace(A{k}*rho_s(:,:,id)));
    end
end
y = y_s(2:end,:)-y_exact(2:end,:);
y = reshape(y,[],1);
%% calculate KKrausA_s(k) = (Kraus^*)^{k-1}[A], k =1,2,...,iter_num
if nargout>1
    for k = 1:Anum
        for i = 1:iter_num
            if i == 1
                KKrausA_s(:,:,i,k) = A{k};
            else
                for j = 1: F_num
                    KKrausA_s(:,:,i,k) = KKrausA_s(:,:,i,k)+
F_s(:,:,j)'*KKrausA_s(:,:,i-1,k)*F_s(:,:,j);
                end
            end
        end
    end
    %% calculate the derivative of F_j to get \partial_{theta} K
    % the derivative of L
    thetaHnum = numel(e)+numel(c);
    drv_L_d1 = cell(N,pauli_num,jump_num);
    drv_L_d2 = cell(N,pauli_num-1,jump_num);
    drv_L_d1(:,:,:) = {zeros(sz,sz)};
    drv_L_d2(:,:,:) = {zeros(sz,sz)};
    for j = 1:jump_num
        drv_L_d1{j,1,j} = sigma{j,1};
        for a = 2:pauli_num
            drv_L_d1{j,a,j} = sigma{j,a};
            drv_L_d2{j,a-1,j} = 1i*sigma{j,a};
        end
```

```
    end
    drv_L = cell(jump_num,1);
    drv_LH = cell(thetaHnum,1);
    drv_LH(:,:) = {zeros(sz,sz)};
    for i = 1:jump_num
        Ld1 = reshape(drv_L_d1(:,:,i),[],1);
        Ld2 = reshape(drv_L_d2(:,:,i),[],1);
        drv_L{i} = [drv_LH;Ld1;Ld2];
    end
    % the derivative of G
    drv_G_e = cell(N,pauli_num);
    drv_G_c = cell(N-1,pauli_num,pauli_num);
    drv_G_d1 = cell(N,pauli_num);
    drv_G_d2 = cell(N,pauli_num-1);
    for j = 1:N
        for a = 1: pauli_num
            drv_G_e{j,a} = -1i*sigma{j,a};
        end
    end
    for j = 1:N-1
        for a = 1: pauli_num
            for b = 1:pauli_num
                drv_G_c{j,a,b} = -1i*sigma{j,a}*sigma{j+1,b};
            end
        end
    end
    for j = 1:N
        s1 = L(:,:,j)'*drv_L_d1{j,1,j};
        drv_G_d1{j,1} = -(s1'+s1)./2;
        for a = 2:pauli_num
            s1 = L(:,:,j)'*drv_L_d1{j,a,j};
            s2 = L(:,:,j)'*drv_L_d2{j,a-1,j};
            drv_G_d1{j,a} = -(s1'+s1)./2;
            drv_G_d2{j,a-1} = -(s2'+s2)./2;
        end
    end
    drv_G = theta2vector2(drv_G_e,drv_G_c,drv_G_d1,drv_G_d2);
    % the derivative of F_j
    drv_F = cell(theta_num,F_num);
    for i = 1:theta_num
        drv_F{i,1} = op1\drv_G{i}*(F_s(:,:,1)+I2).*(delta_t/2); %
derivative of F_0
    end

    for j = 2:jump_num+1
        for i = 1:theta_num
            drv_F{i,j} = op1\drv_G{i}*F_s(:,:,j).*(delta_t/2)+op1\drv_L{j-1}
{i}*op2.*sqrt(delta_t)+op1\L(:,:,j-1)*drv_G{i}.*(sqrt(delta_t)^3/2);
        end
    end
```

```
    for i = 1:jump_num
        for j = 1:jump_num
            for alpha = 1:theta_num
                drv_F{alpha,i+jump_num*j+1} = (drv_L{i}{alpha}*L(:,:,j)
+L(:,:,i)*drv_L{j}{alpha}).*(delta_t/sqrt(2));
            end
        end
    end
    %% the derivative of \partial_{theta}K based on Theorem 2
    tilde_s(:,:,:) = {zeros(sz,sz)};     % tilde_s(:,:,k) = (\partial_theta
K)^*(K_theta^*)^{k-1}[A]
    for alpha = 1: theta_num
        for k = 1:Anum
            for l = 1:iter_num
                for j = 1:F_num
                    s = F_s(:,:,j)'*KKrausA_s(:,:,l,k)*drv_F{alpha,j};
                    tilde_s{alpha,l,k} = tilde_s{alpha,l,k}+s+s';
                end
            end
        end
    end
    % calculate tr(A\chi_n)
    for k = 1:Anum
        for alpha = 1:theta_num
            for n = 1:mea_num
                id = start+n*Lt;
                for l = 1:id-1
                    tr_value(alpha,n,k) = tr_value(alpha,n,k)
+trace(tilde_s{alpha,l,k}*rho_s(:,:,id-l));
                end
            end
        end
    end
    J1 = real(tr_value);
    J2 = zeros(mea_num*Anum,theta_num);
    for i = 1:theta_num
        for k = 1:Anum
            for j = 1:mea_num
                J2((k-1)*mea_num+j,i) = J1(i,j,k);
            end
        end
    end
end
end
```

Generate observables, initial states and parameter θ

Observables: genManyA.m

```
function [A,N] = genManyA(local_num,spins)
I2 = eye(2);
switch local_num
    case 0
        N = 1;
        sigmax = [0,1;1,0];
        A = cell(1,1);
        A{1} = sigmax;
        for i = 1:spins-1
            A{1} = kron(A{1},I2);
        end
    case 1 % 1-local observables
        N = spins*3+1;
        B = cell(spins,3);
        sigmax = [0 1;1 0];
        sigmay = [0 -1i;1i 0];
        sigmaz = [1 0; 0 -1];
        Pauli = cell(4,1);
        Pauli{1} = sigmax; Pauli{2} = sigmay; Pauli{3}=sigmaz; Pauli{4} =
I2;
        for j = 1:3
            for k = 1:spins
                B{k,j} =Pauli{j};
                for l= 1:spins
                    if l < k
                        B{k,j} = kron(I2,B{k,j});
                    else
                        if l>k
                            B{k,j} = kron(B{k,j},I2);
                        end
                    end
                end
            end
        end
        A = reshape(B,[],1);
        Bend = cell(1,1); Bend{1} = eye(2^spins);
        A = [A; Bend];
```

When we consider fewer 1-local observables σ_j^x, σ_j^y

```
        N = 12; % only the first 12 observables are considered
        A = A(1:N,1);
    case 2 % 2-local observables
        pauli_num= 3;
        B = cell(spins,pauli_num);
        sigmax = [0 1;1 0];
        sigmay = [0 -1i;1i 0];
        sigmaz = [1 0; 0 -1];
        Pauli = cell(pauli_num+1,1);
```

```
        Pauli{1} = sigmax; Pauli{2} = sigmay; Pauli{3}=sigmaz; Pauli{4} =
I2;
        for j = 1:pauli_num
            for k = 1:spins
                B{k,j} =Pauli{j};
                for l= 1:spins
                    if l < k
                        B{k,j} = kron(I2,B{k,j});
                    else
                        if l>k
                            B{k,j} = kron(B{k,j},I2);
                        end
                    end
                end
            end
        end
        C_num = pauli_num^2*(spins-1)*spins/2;
        C = cell(C_num,1);
        id = 1;
        for i = 1:spins
            for j = i+1:spins
               for k = 1:pauli_num
                   for l = 1:pauli_num
                       C{id} = B{j,k}*B{i,l};
                       id = id+1;
                   end
               end
            end
        end
        B = reshape(B,[],1);
        Bend = cell(1,1); Bend{1} = eye(2^spins);
        A = [C;B;Bend];
        N = spins*pauli_num+1+C_num;
end
end
```

Initial States: gen_rho.m

```
function rho = gen_rho
   N = 6;
   spinup = [1,0]';
   psi = spinup;
   for i = 2:N
       psi = kron(psi,spinup);
   end
   rho = psi*psi';
end
```

Parameter θ: genTheta.m

```
N = 6; pauli_num = 3;
```

```
alphaD = 1/sqrt(2);
lnl = 2;
theta_s = gen_theta(N,pauli_num,alphaD,lnl);
[e,c,d1,d2] = theta2tensor2(theta_s);
theta_test = theta2vector2(e,c,d1,d2);
testerror = norm(theta_s-theta_test)
theta0 = theta_s+gen_theta(N,pauli_num,alphaD,lnl)/20;
save('thetas.mat',"theta_s","theta0")
norm(theta_s-theta0)
```

random generation subfunction: gen_theta.m

```
function theta = gen_theta(N,pauli_num,alphaD,lnl)
c = randn(N-1,pauli_num,pauli_num);
e = randn(N,pauli_num);
switch lnl
    case 1 % linear case
        d1 = randn(2,1)*alphaD; % N(0,1/2) real part of d
        d1 = abs(d1);
        theta = theta2vector(e,c,d1); % start point of parameter theta
    case 2 % nonlinear case
        d1 = randn(N,pauli_num)*alphaD;
        d2 = randn(N,pauli_num-1)*alphaD;% imaginary part of d
        theta = theta2vector2(e,c,d1,d2);
end
end
```

Tool Functions to change the parameter vector to corresponding Hamiltonian and dissipation part

From vector to Hamiltonian and dissipation part in matrix form: theta2tensor2.m

```
function [e,c,d1,d2] = theta2tensor2(theta,N)
% convert theta from vector to e, c, d1, d2 for nonlinear case
   % N = 6;
   pauli_num = 3;
   N1 = N*pauli_num;
   N2 = N1+(N-1)*pauli_num*pauli_num; % dont consider c_{N,a,b}
   N3 = N2+N*pauli_num;
   N4 = numel(theta);
   e_v = theta(1:N1);
   c_v = theta(N1+1:N2);
   d1_v = theta(N2+1:N3);
   d2_v = theta(N3+1:N4);
   e = reshape(e_v,N,pauli_num);
   c = reshape(c_v,N-1,pauli_num,pauli_num);
   d1 = reshape(d1_v, N,pauli_num);
   d2 = reshape(d2_v, N, pauli_num-1);
end
```

From parameter matrices to vector form: theta2vector2.m

```
function theta = theta2vector2(e,c,d1,d2)
% convert theta from tensor to vector for nonlinear case
e_v = reshape(e,[],1);
c_v = reshape(c,[],1);
d1_v = reshape(d1,[],1);
d2_v = reshape(d2,[],1);
theta = [e_v;c_v;d1_v;d2_v];
end
```