Linear example: The Code for Fig 4 and 6

Consider all 1-local observables(N_O=19), fewer 1-local observables(N_O=12)

- Fig 4: 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 $\|\theta^{(0)} - \theta^*\|$=0.3658 in thetas.mat
- Fig 6: Consider all 1-local observables(N_O=19) at time $t_n = 4.1, 4.2, \cdots, 5(N_T = 10)$ and initial guess $\|\theta^{(0)} - \theta^*\|$=0.3658 in thetas.mat, numerical simulation time interval is smaller at $\delta t = 0.01$

Table of Contents

Main function

```
[theta_s, theta0, theta_after,fval,history,output,exitflag] = runfminunc;
save('exp30.3.mat','exitflag','output','history',"fval","theta_after","theta
_s")
```

- exp30.3.mat for all 1-local observable case in Fig 4
- exp30.4.mat for all 1&2-local observable case in Fig4
- exp30.6.1.mat for all 1-local observables at shifted time $t_n = 4.1, 4.2, \cdots, 5(N_T = 10)$ in Fig 6

Graph the Performance of Optimization: observableCP.m

Fig 4: from exp30.3.mat and exp30.4.mat

```
fig1 = load('exp30.3.mat');
fig2 = load('exp30.4.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','interpreter','latex','location','best','fontsize',30)
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','interpreter','latex','location','best','fontsize',30)
xlabel('Iteration');
ylabel('Relative Error')
title('Relative Error $\frac{||\theta-\theta^*||}{||\theta^*||}
$','interpreter','latex')
print('exp30.3&4cp.jpg','-djpeg')
```

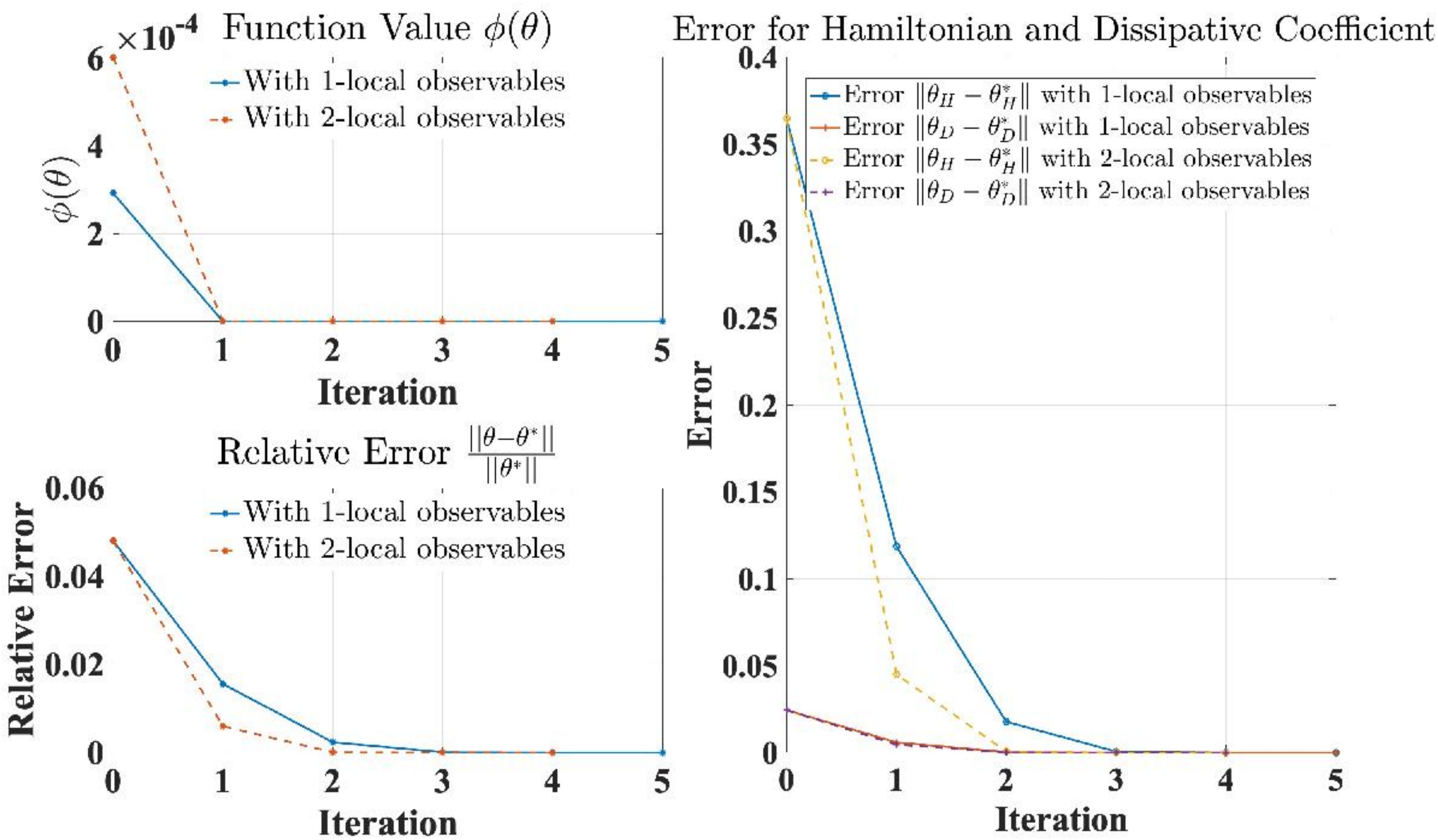
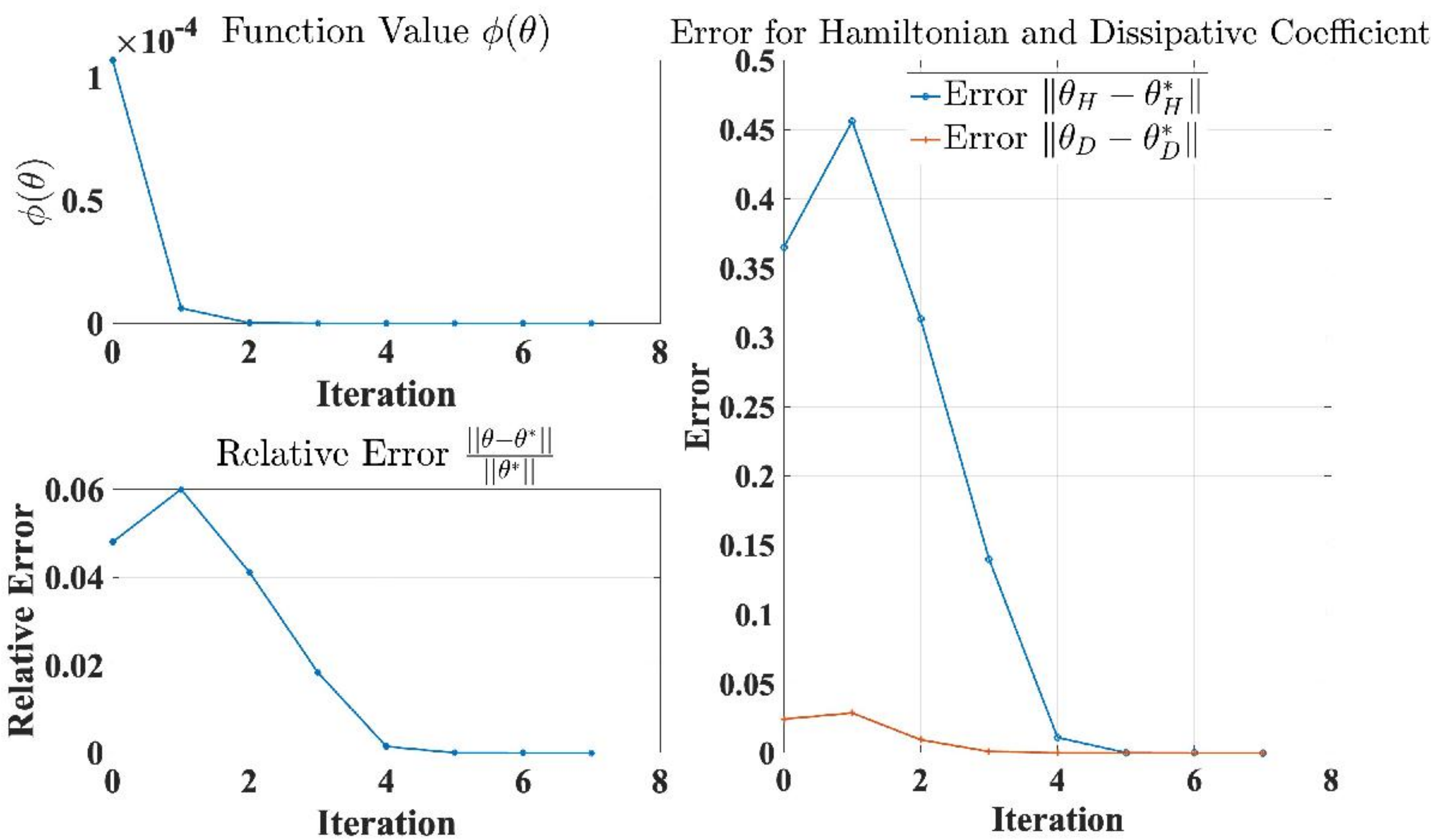


Fig6: from exp30.6.1.mat

```
load('exp30.6.1.mat')
tiledlayout(2,2)
NO = 19;
NT = 10;
NOT = NO*NT;
ax3 = nexttile;
plot(ax3,output.iteration, output.resnorm.^2./(2*NOT),'-*','LineWidth',2)
grid on
set(gca,"FontSize",34,'FontName','Times','FontWeight','bold')
xlabel('Iteration');
ylabel('$\phi(\theta)$','interpreter','latex');
title('Function Value $\phi(\theta)$','interpreter','latex')
ax1 = nexttile([2,1]);
% plot(ax1,history.iteration, history.error3H,'-o','LineWidth',2)
plot(ax1,output.iteration, output.error3H,'-o',output.iteration,
output.error3D,'-+','LineWidth',2)
grid on
set(gca,"FontSize",32,'FontName','Times','FontWeight','bold')
legend('Error $\|\theta_H-\theta_H^*\|$','Error $\|\theta_D-\theta_D^*\|
$','interpreter','latex','location','best',"FontSize",36)
xlabel('Iteration');
ylabel('Error')
```

```
title('Error for Hamiltonian and Dissipative
Coefficients','interpreter','latex')
ax4 = nexttile;
plot(ax4,output.iteration, output.error4,'-*','LineWidth',2)
grid on
set(gca,"FontSize",34,'FontName','Times','FontWeight','bold')
xlabel('Iteration');
ylabel('Relative Error')
title('Relative Error $\frac{||\theta-\theta^*||}{||\theta^*||}
$','interpreter','latex')
print('exp30.6.1.jpg','-djpeg')
```



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; % the number of spins
pauli_num = 3;
load('thetas.mat'); % nondissipation parameter only lambda1
error_in = norm(theta_s-theta0)
T0 = 0;
T1 = 0;
```

- T1 = 4 for Fig 6

```
rho0 = gen_rho;
T = T1+1;
Delta_t = 0.1;
% generate sigma
sigma = cell(N,pauli_num);
[A, Anum] = genManyA(1,N);
```

- genManyA(2,N) for Fig 4 with all possible 2-local observables

```
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

% global y_exact
```

```
fun0 = @(theta)gen_mea_data1(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1);
y_exact = fun0(theta_s);
fun1 = @(theta)gradPhilsq1(theta,y_exact,sigma,Delta_t,T,A,rho0,Anum,T0,T1);

% Call 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;
     Hnum = 63;
     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)
              xlabel('iteration');
              ylabel('Error for Hamiltonian coefficients')
              title('Error $\theta_H-\theta_H^*$','interpreter','latex')
              ax2 = nexttile;
              plot(ax2,history.iteration, history.error3D)
              xlabel('iteration');
              ylabel('Error for dissipation coefficients')
              title('Error $\theta_D-\theta_D^*$','interpreter','latex')
              ax3 = nexttile;
              plot(ax3,history.iteration, history.resnorm)
              xlabel('iteration');
              ylabel('Objective function value')
              title('Function Value $\phi(\theta)$','interpreter','latex')
              ax4 = nexttile;
              plot(ax4,history.iteration, history.error4)
              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_data1.m

```
function y= gen_mea_data1(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1)
% generate measurement data y_n for 6 spin example
[e,c,d1] = theta2tensor(theta);
N = size(e,1); % the number of spins
Sp = 6;        % the number of spin
sz = 2^Sp;     % the size of density matrix or the dimension of the Hilbert
space
pauli_num = 3;
iter_num = round((T-T0)/Delta_t);
H = zeros(sz,sz,N); % Hamiltonian
jump_num = 2*N;     % the number of jump operator
L = zeros(sz,sz,jump_num); % dissipation term

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) = (sigma{j,1}-1i*sigma{j,2})*sqrt(d1(1))/2;
    L(:,:,j+N) = sigma{j,3}*sqrt(d1(2));
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)-L(:,:,j)'*L(:,:,j)/2-L(:,:,j+N)'*L(:,:,j+N)/2;
end
%% Next: 2.0 implicit Taylor Scheme -> Kraus form
y = zeros(iter_num+1,Anum); % measurement y_i, i = 0,1,...,iter_num
I2 = eye(sz); % Identity matrix of the same size of density 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
t0 = (T0:Delta_t:T);
t1 = (T1:Delta_t:T);
start = length(t0)-length(t1)+1;
for i = start:length(t0)
    for k = 1:Anum
        y(i,k) = real(trace(A{k}*rho_s(:,:,i)));
    end
end
end
```

Calculating the Loss Function Value and its Gradients gradPhilsq1.m

```
function [y,J2] = gradPhilsq1
(theta,y_exact,sigma,Delta_t,T,A,rho_0,Anum,T0,T1)
[e,c,d1] = theta2tensor(theta);
N = size(e,1);  % the number of e_j
Sp = 6;         % 6 spins example
sz = 2^Sp;      % the size of density matrix \rho
pauli_num = 3;
iter_num = round((T-T0)/Delta_t);
t0 = (T0:Delta_t:T);
t1 = (T1:Delta_t:T);
start = length(t0)-length(t1)+1;
y_s = zeros(iter_num+1,Anum);

%%
theta_num = numel(theta);                % the number of parameter theta
rho_s = zeros(sz,sz,iter_num+1);         % save \rho_n value in order to
calculate gradient \rho_s(i):
tilde_s = cell(theta_num, iter_num,Anum);    % to save the matrix in
computing tr(A\chi_n)
```

```
tr_value = zeros(theta_num, iter_num+1,Anum);  % save the value of
tr(A\chi_n) in order 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,iter_num+1,Anum);

%% initial value of F_i
H = zeros(sz,sz,N);                  % Hamiltonian
jump_num = 2*N;
L = zeros(sz,sz,jump_num);           % dissipation term
F_num = jump_num^2+jump_num+1;
F_s = zeros(sz,sz,F_num);
% generate H_j j = 1,..., N-1
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

% generate L_j, j = 1,...,N
for j = 1:N
    L(:,:,j) = (sigma{j,1}-1i*sigma{j,2})*sqrt(d1(1))/2;
    L(:,:,j+N) = sigma{j,3}*sqrt(d1(2));
end
G = zeros(sz,sz);
for j = 1: N
    G = G-1i*H(:,:,j)-L(:,:,j)'*L(:,:,j)/2-L(:,:,j+N)'*L(:,:,j+N)/2;
end

% F0,F1,...,F6 % F(:,:,1) = F_0, F(:,:,j) = F_{j-1}
I2 = eye(sz,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=1}^lng F_s{j}\rho_s{j}F_s{j}'
```

```
% 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; %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  % rho_s{i} save the value of \rho_{i-1} = K^{i-1}\rho_0
    end
end

for i = start:length(t0)
    for k = 1:Anum
        y_s(i,k) = real(trace(A{k}*rho_s(:,:,i)));
    end
end
y = y_s(start+1:end,:)-y_exact(start+1:end,:);
y = reshape(y,[],1);
%% gradient
% calculate KKrausA_s(k) = (K^*)^{k-1}[A], k =1,2,...,iter_num
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 G
drv_G_e = cell(N,pauli_num);
drv_G_c = cell(N-1,pauli_num,pauli_num);
drv_G_d1 = cell(2,1);
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
```

```
drv_G_d1{1,1} = -(sigma{1,1}+1i*sigma{1,2})*(sigma{1,1}-1i*sigma{1,2})/8;
for j = 2:N
    sigmaminus = sigma{j,1}-1i*sigma{j,2};
    drv_G_d1{1,1} = drv_G_d1{1,1}-sigmaminus'*sigmaminus/8;
end
drv_G_d1{2,1} = -N/2*I2;
drv_G = theta2vector(drv_G_e,drv_G_c,drv_G_d1);
thetaHnum = numel(e)+numel(c);
drv_L_d1 = cell(2,jump_num);
drv_L_d1(:,:) = {zeros(sz,sz)};
for j = 1:N
    drv_L_d1{1,j} = L(:,:,j)/(2*d1(1));
    drv_L_d1{2,j+N} = L(:,:,j+N)/(2*d1(2));
end
drv_L = cell(jump_num,1);
drv_LH = cell(thetaHnum,1);
drv_LH(:,:) = {zeros(sz,sz)};
for i = 1:jump_num
    drv_L{i} = [drv_LH;drv_L_d1(:,i)];
end

% 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
% (\partial_{\theta}K_{\theta})A = \sum_{j=1}^{N+1}
\partial_{\theta}F_jAF_j^{\dag}+F_jA \partial_{\theta}F_j^{\dag}
tilde_s(:,:,:) = {zeros(sz,sz)};
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) based on equation (25)
for k =1:Anum
    for alpha = 1:theta_num
        for n = start:iter_num
            for l = 1:n
                tr_value(alpha,n,k) = tr_value(alpha,n,k)
+trace(tilde_s{alpha,l,k}*rho_s(:,:,n-l+1));
            end
            J1(alpha,n,k) = real(tr_value(alpha,n,k));
        end
    end
end
J11 = J1(:,start:iter_num,:);
size11J2 = length(t1)-1;
J2 = zeros(size11J2*Anum,theta_num);

for i = 1:theta_num
    for k = 1:Anum
        for j = 1:size11J2
            J2((k-1)*size11J2+j,i) = J11(i,j,k);
        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
        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];
    case 2
        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);
theta_s = gen_theta(N,pauli_num,alphaD);
[e,c,d1] = theta2tensor(theta_s);
theta_test = theta2vector(e,c,d1);
testerror = norm(theta_s-theta_test)
theta0 = theta_s+gen_theta(N,pauli_num,alphaD)/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)
c = randn(N-1,pauli_num,pauli_num);
e = randn(N,pauli_num);
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
end
```

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

From vector to Hamiltonian and dissipation part in matrix form: theta2tensor.m

```
function [e,c,d1] = theta2tensor(theta)
% convert theta from vector to e, c, d1
   N = 6;    pauli_num = 3;
   N1 = N*pauli_num;
   N2 = N1+(N-1)*pauli_num*pauli_num;
   e_v = theta(1:N1);
   c_v = theta(N1+1:N2);
   d1_v = theta(N2+1:end);
   e = reshape(e_v,N,pauli_num);
   c = reshape(c_v,N-1,pauli_num,pauli_num);
   d1 = reshape(d1_v, 2,1);
end
```

From parameter matrices to vector form: theta2vector.m

```
function theta = theta2vector(e,c,d1)
% convert theta from tensor to vector
   e_v = reshape(e,[],1);
   c_v = reshape(c,[],1);
   d1_v = reshape(d1,[],1);
   theta = [e_v;c_v;d1_v];
end
```