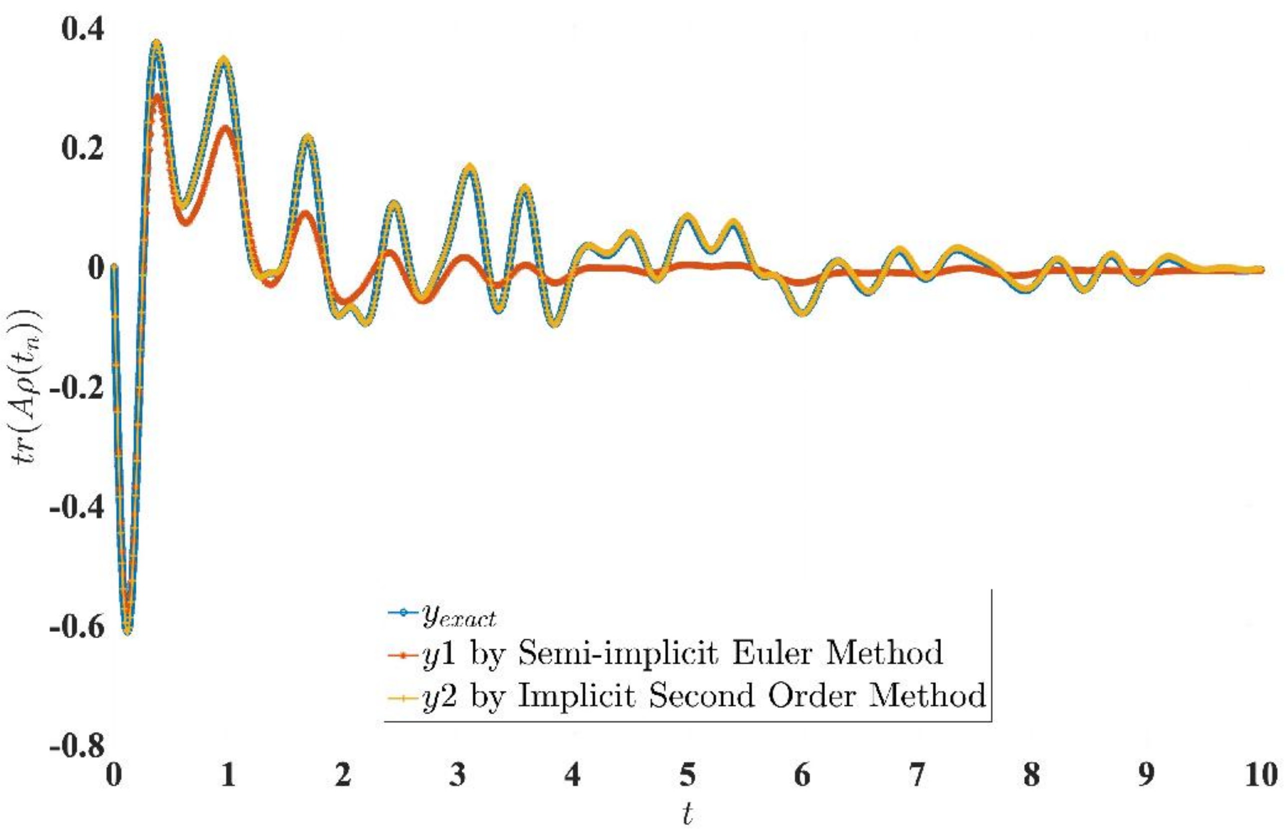
Comparison of first & second order simulation methods: The Code for Fig 3

- Fig 3: Consider all 1-local observables(N_O=19)

Table of Contents

Main function

```
N = 6; % number of spins
pauli_num = 3;
load('thetas.mat'); % dissipation parameter contains both d1 and d2
error_in = norm(theta_s-theta0)
T0 = 0;
T1 = 0;
rho0 = gen_rho;
T = T1+10;
Delta_t = 1e-4;
% generate sigma
sigma = cell(N,pauli_num);
[A, Anum] = genManyA(1);
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

tExact = T0:Delta_t:T;
fun0 = @(theta)semiEuler(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1);
y_exact = fun0(theta_s);

Delta_t = 1e-2;
tSim = T0:Delta_t:T;
fun1 = @(theta)semiEuler(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1);
y1 = fun1(theta_s);

fun2 = @(theta)gen_mea_data1(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1);
y2 = fun2(theta_s);

save('SimulationValidation.mat',"y2","y1","tSim","y_exact","tExact")
for i = 1:Anum
    plot(tExact,y_exact(:,i),'-o',tSim,y1(:,i),'-*',tSim,y2(:,i),'-
+','LineWidth',2)
    grid on
    set(gca, 'FontSize',24,'FontName', 'Times')
    xlabel('$t$','Interpreter','latex')
    ylabel('$tr(A\rho_n)$','Interpreter','latex')
    legend('$y_{exact}$','$y1$ by semi Euler','$y2$ by implicit 2.0 weak
scheme','Interpreter','latex','location','best')
end
print('SimuValidation.jpg','-djpeg')
```



Simulating the evolution by Semi-implicit Euler Method

```
function y= semiEuler(theta,sigma,Delta_t,T,A,rho0,Anum,T0,T1)
% 1.0 Scheme iteration
% generate measurement data y_n for 6 spin example, N: the number of spins
[e,c,d1] = theta2tensor(theta);
N = size(e,1);
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);
jump_num = 2*N;
L = zeros(sz,sz,jump_num); % jump operator
%% 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}; % 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) = (sigma{j,1}-1i*sigma{j,2})*sqrt(d1(1))/2;
    L(:,:,j+N) = sigma{j,3}*sqrt(d1(2));
end
% equation 38 under the example situation
G = zeros(sz,sz);
F_num = 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
%% 1.0 implicit Taylor Scheme(semi-implicit Euler) of SSE -> Kraus form
y = zeros(iter_num+1,Anum); % measurement y_i, i = 0,1,...,mea_num
I2 = eye(sz);               % Identity matrix
% generate F_j
F(:,:,1) = inv(I2-G* Delta_t); % F(:,:,1) = F_0, F(:,:,j) = F_{j-1}
for j = 2:F_num
    F(:,:,j) = F(:,:,1)*L(:,:,j-1)*sqrt(Delta_t);
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
```

Simulating the evolution by Implicit Second Order Method

```
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
```

Generate observables, initial states and parameter θ

Observables: genManyA.m

```
function [A,N] = genManyA(local_num)
spins = 6;
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];
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
```