Summer School on Computational Materials Science

Lecture Notes: Ab Initio Molecular Dynamics Simulation Methods in Chemistry

Victor S. Batista*

Yale University, Department of Chemistry, P.O.Box 208107, New Haven, Connecticut 06520-8107, U.S.A.

I Introduction

These lectures will introduce computational methods that provide quantum mechanical descriptions of the dynamical and equilibrium properties of polyatomic systems.[1–7] According to the fifth postulate of quantum mechanics, the description of dynamics requires solving the time-dependent Schrödinger equation

$$i\frac{\partial\Psi_t(x)}{\partial t} = \hat{H}\Psi_t(x), \tag{1}$$

subject to a given initial condition, $\Psi_0(x)$. To keep the notation as simple as possible, all expressions are written in atomic units, so $\hbar = 1$. Here, $\hat{H} = \hat{p}^2/(2m) + V(\hat{x})$ is the Hamiltonian operator, $\hat{p} = -i\nabla$ is the momentum operator and $V(\hat{x})$ is the potential energy operator. A formal solution of Eq. (1) can be obtained by integration, as follows:

$$\Psi_t(x) = \int dx' \langle x|e^{-i\hat{H}t}|x'\rangle\langle x'|\Psi_0\rangle, \tag{2}$$

where the Kernel $\langle x|e^{-i\hat{H}t}|x'\rangle$ is the quantum propagator.

As an example, consider a diatomic molecule vibrating near its equilibrium position $\bar{x}$ where the potential is Harmonic,

$$V(\hat{x}) = \frac{1}{2}m\omega^2(\hat{x}-\bar{x})^2. \tag{3}$$

The description of the time-dependent bond-length $x(t)$ is given by the expectation value

$$x(t) = \langle\Psi_t|\hat{x}|\Psi_t\rangle, \tag{4}$$

where Ψ_t is defined according to Eq. (2) with the particular Kernel,

$$\langle x|e^{-i\hat{H}t}|x'\rangle = \sqrt{\frac{m\omega}{2\pi\sinh(it\omega)}}\exp\left(-\frac{m\omega}{2\sinh(\omega it)}[(x^2+x'^2)\cosh(\omega it) - 2xx']\right). \tag{5}$$

This standard formulation of quantum mechanics relies upon the tools of *calculus* (*e.g.*, derivatives, integrals, etc.) and involves equations and operations with infinitesimal quantities as well as states in Hilbert-space (the infinite dimensional space of functions L^2). These equations, however, seldom can be solved analytically as shown in the example above. Therefore, computational solutions are necessary. However, computers can not handle infinite spaces since they have only limited memory. In fact, all they *can* do is to store and manipulate discrete arrays of numbers. Therefore, the question is: how can we represent continuum states and operators in the space of memory of digital computers?

*E-mail: victor.batista@yale.edu

II Grid-Based Representations

In order to introduce the concept of a grid-representation, we consider the state,

$$\Psi_0(x) = \left(\frac{\alpha}{\pi}\right)^{1/4} e^{-\frac{\alpha}{2}(x-x_0)^2+ip_0(x-x_0)}, \tag{6}$$

which can be expanded in the infinite basis set of delta functions $\delta(x-x')$ as follows,

$$\Psi_0(x) = \int dx' c(x')\delta(x-x'), \tag{7}$$

where $c(x') \equiv \langle x'|\Psi_0\rangle = \Psi_0(x')$.

A grid-based representation of $\Psi_0(x)$ can be obtained, in the coordinate range $x = (x_{min}, x_{max})$, by discretizing Eq. (7) as follows,

$$\Psi_0(x) = \Delta \sum_{j=1}^{n} c_j \delta(x-x_j), \tag{8}$$

where the array of numbers $c_j \equiv \langle x_j|\Psi_0\rangle$ represent the state Ψ_0 on a grid of equally spaced coordinates $x_j = x_{min} + (j-1)\Delta$ with finite resolution $\Delta = (x_{max} - x_{min})/(n-1)$.

Note that the grid-based representation, introduced by Eq. (8), can be trivially generalized to a grid-based representation in the multidimensional space of parameters (*e.g.*, x_j, p_j, γ_j, ... etc.) when expanding the target state $\Psi_0(x)$ as a linear combination of basis functions $\langle x|x_j, p_j, \gamma_j\rangle$, with expansion coefficients as $c_j \equiv \langle x_j, p_j, \gamma_j|\Psi_0\rangle$.

Problem 1: Write a Fortran code to represent the wave-packet, introduced by Eq. (6) on a grid. Visualize it with Gnuplot. Choose $x_0 = 0$ and $p_0 = 0$, in the range x=(-20,20), with $\alpha = \omega m$, where $m = 1$ and $\omega = 1$. If this is your first Fortran code, copy the attached solution into a file named Problem1.f, compile it by typing f77 Problem1.f -o Problem1, run it by typing ./Problem1, and visualize the output by first typing gnuplot, and then typing plot "arch.0000". Type quit, to exit Gnuplot.

Next, we consider grid-based representations in momentum space:

$$\Psi_0(p) = \langle p|\Psi_0\rangle. \tag{9}$$

Inserting the closure relation $\hat{\mathbf{1}} = \int dx|x\rangle\langle x|$ in Eq. (9), we obtain that

$$\langle p|\Psi_0\rangle = \int dx\langle p|x\rangle\langle x|\Psi_0\rangle = (2\pi)^{-1/2}\int dx e^{-ipx}\langle x|\Psi_0\rangle. \tag{10}$$

is the Fourier transform of the initial state since

$$\langle p|x\rangle = (2\pi)^{-1/2}e^{-ip\hat{x}}. \tag{11}$$

The Fourier transform can be efficiently implemented in $O(N\log(N))$ steps, when $\langle x|\Psi_0\rangle$ is represented on a grid with $N = 2^n$ points (where n is an integer), by using the Fast Fourier Transform (FFT) algorithm.[8] In contrast, the implementation of the Fourier transform by quadrature integration would require $O(N^2)$ steps.

Problem 2: Write a Fortran code to represent the initial state, introduced by Eq. (6), in the momentum space by applying the FFT algorithm to the grid-based representation generated in Problem 1. Visualize the result with Gnuplot. Represent the wave-packet amplitudes and phases in the range p=(-4,4) and compare your output with the corresponding values obtained from the analytic Fourier transform obtained by using: $\int dx\ \exp(-a_2x^2 + a_1x + a_0) = \sqrt{\pi/a_2}\ \exp(a_0 + a_1^2/(4a_2))$.

Next, we consider the grid-based representation of operators (*e.g.*, $\hat{x}$, $\hat{p}$, $V(\hat{x})$, and $\hat{T} = \hat{p}^2/(2m)$) and learn how these operators act on states represented on grids in coordinate and momentum spaces.

Consider first applying the potential energy operator to the initial state, as follows,

$$V(\hat{x})\Psi_0(x) = V(x)\Psi_0(x) \equiv \tilde{\Psi}_0(x). \tag{12}$$

Since $\tilde{\Psi}_0(x)$ is just another function, Eq. (12) indicates that $V(\hat{x})$ can be represented on the same grid of coordinates as before (*i.e.*, equally spaced coordinates $x_j = x_{min} + (j-1)\Delta$, with finite resolution $\Delta = (x_{max} - x_{min})/(n-1)$). Since for each x_j, $\tilde{\Psi}_0(x_j) = V(x_j)\Psi(x_j)$, the operator $V(\hat{x})$ can be represented just as an array of numbers $V(x_j)$ associated with the grid-points x_j, and its operation on a state is represented on such a grid as a simple multiplication.

Problem 3: Write a Fortran code to compute the expectation values of the position $x(0) = \langle\Psi_0|\hat{x}|\Psi_0\rangle$ and the potential energy $V = \langle\Psi_0|V(\hat{x})|\Psi_0\rangle$, where $V(x)$ is defined according to Eq. (3) for the initial wave-packet, introduced by Eq. (6), with various possible values of x_0 and p_0, with $\alpha = \omega m$, where $m = 1$ and $\omega = 1$.

Now consider applying the momentum operator, $\hat{p} = -i\nabla$, to the initial state $\Psi_0(x)$ as follows,

$$G(x) = \langle x|\hat{p}|\Psi_0\rangle = -i\nabla\Psi_0(x). \tag{13}$$

One simple way of implementing this operation, when $\Psi_0(x)$ is represented on a grid of equally spaced points $x_j = x_{min} + (j-1)\Delta$, is by computing finite-increment derivatives as follows:

$$G(x_j) = -i\frac{\Psi_0(x_{j+1}) - \Psi_0(x_{j-1})}{2\Delta}. \tag{14}$$

However, for a more general operator (*e.g.*, $\hat{T} = \hat{p}^2/(2m)$) this finite increment derivative procedure becomes complicated. In order to avoid computing finite-increment derivatives, one can implement an alternative procedure: represent the initial state in momentum-space (by Fourier transform of the initial state); apply the operator by simple multiplication in momentum space, then transform the resulting product back to the coordinate representation (by inverse-Fourier transform). This method can be derived by inserting the closure relation $\hat{\mathbf{1}} = \int dp|p\rangle\langle p|$, in Eq. (13),

$$G(x) = \langle x|\hat{p}|\Psi_0\rangle = \int dp\langle x|\hat{p}|p\rangle\langle p|\Psi_0\rangle = (2\pi)^{-1/2}\int dp e^{ipx}p\langle p|\Psi_0\rangle, \tag{15}$$

since $\langle p|\Psi_0\rangle$ is defined, according to Eq. (10), as the Fourier transform of the initial state. Note that the second equality of Eq. (15) is obtained by introducing the substitution

$$\langle x|p\rangle = (2\pi)^{-1/2}e^{ix\hat{p}}. \tag{16}$$

While Eq. (15) illustrates the method for the specific operator $\hat{p}$, one immediately sees that any operator which is a function of $\hat{p}$ (*e.g.*, $\hat{T} = \hat{p}^2/(2m)$) can be analogously applied according to the Fourier transform procedure.

Problem 4: Write a Fortran code to compute the expectation values of the initial momentum $p(0) = \langle\Psi_0|\hat{p}|\Psi_0\rangle$ and the kinetic energy $T = \langle\Psi_0|\hat{p}^2/(2m)|\Psi_0\rangle$ by using the Fourier transform procedure, where Ψ_0 is the initial wave-packet introduced by Eq. (6), with $x_0 = 0$, $p_0 = 0$, and $\alpha = \omega m$, where $m = 1$ and $\omega = 1$. Compute the expectation value of the energy $E = \langle\Psi_0|\hat{H}|\Psi_0\rangle$, where $\hat{H} = \hat{p}^2/(2m) + V(\hat{x})$, with $V(x)$ defined according to Eq. (3) and compare your result with the zero-point energy $E_0 = \omega/2$.

III SOFT Method

The Split-Operator Fourier Transform (SOFT) method is a numerical approach for solving the time-dependent Schrödinger equation by using grid-based representations of the time-evolving states and operators. It relies on the Fourier transform procedure, introduced in Sec. II, to apply operators that are functions of $\hat{p}$ by simple multiplication of array elements.

The essence of the method is to discretize the propagation time on a grid $t_k = (k-1)\tau$, with $k = 1, ..., n$ and time-resolution $\tau = t/(n-1)$, and obtain the wave-packet at the intermediate times t_k by recursively applying Eq. (2) as follows,

$$\Psi_{t_{k+1}}(x) = \int dx' \langle x|e^{-i\hat{H}\tau}|x'\rangle\langle x'|\Psi_{t_k}\rangle. \tag{17}$$

If τ is a sufficiently small time-increment (*i.e.*, n is large), the time-evolution operator can be approximated according to the Trotter expansion to second order accuracy,

$$e^{-i\hat{H}\tau} = e^{-iV(\hat{x})\tau/2}e^{-i\hat{p}^2\tau/(2m)}e^{-iV(\hat{x})\tau/2} + O(\tau^3), \tag{18}$$

which separates the propagator into a product of three operators, each of them depending either on $\hat{x}$, or $\hat{p}$.

Problem 5: Expand the exponential operators in both sides of Eq. (18) and show that the Trotter expansion is accurate to second order in powers of τ.

Substituting Eq. (18) into Eq. (17) and inserting the closure relation $\hat{\mathbf{1}} = \int dp|p\rangle\langle p|$ gives,

$$\Psi_{t_{k+1}}(x) = \int dp \int dx' e^{-iV(\hat{x})\tau/2}\langle x|p\rangle e^{-ip^2\tau/(2m)}\langle p|x'\rangle e^{-iV(x')\tau/2}\Psi_{t_k}(x'). \tag{19}$$

By substituting $\langle p|x'\rangle$ and $\langle x|p\rangle$ according to Eqs. (11) and (16), respectively, we obtain:

$$\Psi_{t_{k+1}}(x) = e^{-iV(\hat{x})\tau/2}\frac{1}{\sqrt{2\pi}}\int dp e^{ixp}e^{-ip^2\tau/(2m)}\frac{1}{\sqrt{2\pi}}\int dx' e^{-ipx'}e^{-iV(x')\tau/2}\Psi_{t_k}(x'). \tag{20}$$

According to Eq. (20), then, the computational task necessary to propagate $\Psi_t(x)$ for a time-increment τ involves the following steps:

1. Represent $\Psi_{t_k}(x')$ and $e^{-iV(x')\tau/2}$ as arrays of numbers $\Psi_{t_k}(x_j)$ and $e^{-iV(x_j)\tau/2}$ associated with a grid of equally spaced coordinates $x_j = x_{min} + (j-1)\Delta$, with finite resolution $\Delta = (x_{max} - x_{min})/(n-1)$.

2. Apply the potential energy part of the Trotter expansion $e^{-iV(x')\tau/2}$ to $\Psi_{t_k}(x')$ by simple multiplication of array elements:
$$\tilde{\Psi}_{t_k}(x_j) = e^{-iV(x_j)\tau/2}\Psi_{t_k}(x_j).$$

3. Fourier transform $\tilde{\Psi}_{t_k}(x_j)$ to obtain $\tilde{\Psi}_{t_k}(p_j)$, and represent the kinetic energy part of the Trotter expansion $e^{-ip^2\tau/(2m)}$ as an array of numbers $e^{-ip_j^2\tau/(2m)}$ associated with a grid of equally spaced momenta $p_j = j/(x_{max} - x_{min})$.

4. Apply the kinetic energy part of the Trotter expansion $e^{-ip^2\tau/(2m)}$ to the Fourier transform $\tilde{\Psi}_{t_k}(p)$ by simple multiplication of array elements:
$$\widetilde{\Psi}_{t_k}(p_j) = e^{-ip_j^2\tau/(2m)}\tilde{\Psi}_{t_k}(p_j).$$

5. Inverse Fourier transform $\widetilde{\Psi}_{t_k}(p_j)$ to obtain $\widetilde{\Psi}_{t_k}(x_j)$ on the grid of equally spaced coordinates x_j.

6. Apply the potential energy part of the Trotter expansion $e^{-iV(x')\tau/2}$ to $\widetilde{\Psi}_{t_k}(x')$ by simple multiplication of array elements,
$$\Psi_{t_{k+1}}(x_j) = e^{-iV(x_j)\tau/2}\widetilde{\Psi}_{t_k}(x_j).$$

Problem 6: Write a Fortran code that propagates the initial state $\Psi_0(x)$ for a single time increment ($\tau = 0.1$ a.u.). Use $x_0 = -2.5$, $p_0 = 0$, and $\alpha = \omega m$, where $m = 1$ and $\omega = 1$. Implement the SOFT method for the Hamiltonian $\hat{H} = \hat{p}^2/(2m) + V(\hat{x})$, where $V(x)$ is defined according to Eq. (3). Compare the resulting propagated state with the analytic solution obtained by substituting Eq. (5) into Eq. (2).

Problem 7: Loop the Fortran code developed in Problem 5 with $x_0 = -2.5$ and $p_0 = 0$ for 100 steps with $\tau = 0.1$ a.u. For each step compute the expectation values of coordinates $x(t)$ and momenta $p(t)$ as done in Problems 3 and 4, respectively. Compare your calculations with the analytic solutions obtained by substituting Eq. (5) into Eq. (2). Verify that these correspond to the classical trajectories $x(t) = \bar{x} + (x_0 - \bar{x})\cos(\omega t)$ and $p(t) = p_0 - (x_0 - \bar{x})\omega m \sin(\omega t)$, which can be computed according to the Velocity-Verlet algorithm:
$$\begin{aligned} p_{j+1} &= p_j + (F(x_j) + F(x_{j+1}))\tau/2 \\ x_{j+1} &= x_j + p_j\tau/m + F(x_j)\tau^2/(2m). \end{aligned} \tag{21}$$

Problem 8: Change the potential to that of a Morse oscillator $V(\hat{x}) = De(1 - \exp(-a(\hat{x} - x_e)))^2$, with $x_e = 0$, $De = 8$, and $a = \sqrt{k/(2D_e)}$, where $k = m\omega^2$. Recompute the wave-packet propagation with $x_0 = -0.5$ and $p_0 = 0$ for 100 steps with $\tau = 0.1$ a.u., and compare the expectation values $x(t)$ and $p(t)$ with the corresponding classical trajectories obtained by recursively applying the Velocity-Verlet algorithm.

Problem 9: Simulate the propagation of a wave-packet with $x_0 = -5.5$ and initial momentum $p_0 = 2$ colliding with a barrier potential $V(x) = 3$, if abs$(x) < 0.5$, and $V(x) = 0$, otherwise. Hint: In order to avoid artificial recurrences you might need to add an absorbing imaginary potential $V_a(x) = i(\text{abs}(x) - 10)^4$, if abs$(x) > 10$, and $V_a(x) = 0$, otherwise.

IV SOFT Propagation on Multiple Surfaces

The goal of this section is to generalize the implementation of the SOFT method to the description of quantum dynamics on multiple coupled potential energy surfaces.

To keep the presentation as simple as possible, we consider a molecule with two-coupled electronic states described by the Hamiltonian,
$$\hat{H} = \hat{p}^2/(2m) + \hat{V}, \tag{22}$$
where $\hat{V} = \hat{V}_0 + \hat{V}_c$, with $\hat{V}_0 = V_1(\hat{\mathbf{x}})|1\rangle\langle 1| + V_2(\hat{\mathbf{x}})|2\rangle\langle 2|$ and $\hat{V}_c = V_c(\hat{\mathbf{x}})|1\rangle\langle 2| + V_c(\hat{\mathbf{x}})|2\rangle\langle 1|$.

The computational task ahead is to implement the SOFT method to compute the time-dependent wave-packet
$$|\Psi(\mathbf{x};t)\rangle = \varphi_1(\mathbf{x};t)|1\rangle + \varphi_2(\mathbf{x};t)|2\rangle, \tag{23}$$
given the initial conditions $\varphi_1(\mathbf{x};0)$ and $\varphi_2(\mathbf{x};0)$, where $\varphi_1(\mathbf{x};t)$ and $\varphi_2(\mathbf{x};t)$ are the time-dependent nuclear wave-packet components associated with the electronic states $|1\rangle$ and $|2\rangle$, respectively. Note that here the main challenges are that $\hat{V}_0$ and $\hat{V}_c$ do not commute, $|\Psi(\mathbf{x};t)\rangle$ involves two wave-packet components and $\hat{H}$ is a 2×2 matrix in the basis of $|1\rangle$ and $|2\rangle$.

A simple approach for propagating $\varphi_1(\mathbf{x};t)$ and $\varphi_2(\mathbf{x};t)$ involves the embedded form of the Trotter expansion,

$$e^{-i\hat{H}2\tau} \approx e^{-i\frac{\hat{p}^2}{2m}\tau} e^{-iV(\hat{\mathbf{x}})2\tau} e^{-i\frac{\hat{p}^2}{2m}\tau} \approx e^{-i\frac{\hat{p}^2}{2m}\tau} e^{-iV_0(\hat{\mathbf{x}})\tau} e^{-iV_c(\hat{\mathbf{x}})2\tau} e^{-iV_0(\hat{\mathbf{x}})\tau} e^{-i\frac{\hat{p}^2}{2m}\tau}, \tag{24}$$

which can be implemented in the basis of $|1\rangle$ and $|2\rangle$ according to the following steps:

- Step [I]. Apply the kinetic energy part of the Trotter expansion to both wave-packet components $\varphi_1(\mathbf{x};t)$ and $\varphi_2(\mathbf{x};t)$ for time τ, as follows,

$$\begin{pmatrix} \varphi_1'(\mathbf{x};t+\tau) \\ \varphi_2'(\mathbf{x};t+\tau) \end{pmatrix} = \begin{pmatrix} e^{-i\frac{\hat{\mathbf{p}}^2}{2m}\tau} & 0 \\ 0 & e^{-i\frac{\hat{\mathbf{p}}^2}{2m}\tau} \end{pmatrix} \begin{pmatrix} \varphi_1(\mathbf{x};t) \\ \varphi_2(\mathbf{x};t) \end{pmatrix}. \tag{25}$$

- Step [II]. Mix the two wave-packet components $\varphi_1'(\mathbf{x};t+\tau)$ and $\varphi_2'(\mathbf{x};t+\tau)$,

$$\begin{pmatrix} \varphi_1''(\mathbf{x};t+\tau) \\ \varphi_2''(\mathbf{x};t+\tau) \end{pmatrix} = \mathbf{M} \begin{pmatrix} \varphi_1'(\mathbf{x};t+\tau) \\ \varphi_2'(\mathbf{x};t+\tau) \end{pmatrix}, \tag{26}$$

with

$$\mathbf{M} \equiv \mathbf{L}^{-1} \begin{pmatrix} e^{-iE_1(x)\tau} & 0 \\ 0 & e^{-iE_2(x)\tau} \end{pmatrix} \mathbf{L}, \tag{27}$$

where $E_1(x)$ and $E_2(x)$ are the eigenvalues of the potential energy matrix $V = V_0 + V_c$ and $\mathbf{L}$ the matrix of column eigenvectors in the basis of diabatic states $|1\rangle$ and $|2\rangle$. Eigenvalues and eigenvectors of a symmetric matrix can be obtained by using the subroutines TRED2, TQLI and EIGSRT, as described in Numerical Recipes (Chapter 11).[9]

While this is a general procedure, the specific case of interest involves a 2×2 Hermitian matrix V, for which the matrix $\mathbf{M}$ can be found analytically,

$$\mathbf{M} \equiv \begin{pmatrix} e^{-i\hat{V}_1(\hat{\mathbf{x}})2\tau}\cos(2V_c(\hat{\mathbf{x}})\tau) & -i\sin(2V_c(\hat{\mathbf{x}})\tau)\, e^{-i(\hat{V}_1(\hat{\mathbf{x}})+\hat{V}_2(\hat{\mathbf{x}}))\tau} \\ -i\sin(2V_c(\hat{\mathbf{x}})\tau)\, e^{-i(\hat{V}_1(\hat{\mathbf{x}})+\hat{V}_2(\hat{\mathbf{x}}))\tau} & \cos(2V_c(\hat{\mathbf{x}})\tau)\, e^{-i\hat{V}_2(\hat{\mathbf{x}})2\tau} \end{pmatrix}. \tag{28}$$

- Step [III]. Propagate $\varphi_1''(\mathbf{x};t+\tau)$ and $\varphi_2''(\mathbf{x};t+\tau)$ for time τ, according to the free-particle propagator, by applying the kinetic energy part of the Trotter expansion:

$$\begin{pmatrix} \varphi_1(\mathbf{x};t+2\tau) \\ \varphi_2(\mathbf{x};t+2\tau) \end{pmatrix} = \begin{pmatrix} e^{-i\frac{\hat{\mathbf{p}}^2}{2m}\tau} & 0 \\ 0 & e^{-i\frac{\hat{\mathbf{p}}^2}{2m}\tau} \end{pmatrix} \begin{pmatrix} \varphi_1''(\mathbf{x};t+\tau) \\ \varphi_2''(\mathbf{x};t+\tau) \end{pmatrix}. \tag{29}$$

In practice, however, step [III] is combined with step [I] of the next propagation time-slice for all but the last propagation time-increment.

Problem 10: (a) Derive Eq. (28) by considering that,

$$e^{-i\mathbf{V}_c 2\tau} = \mathbf{D}^\dagger \begin{pmatrix} e^{iV_c(\mathbf{x})2\tau} & 0 \\ 0 & e^{-iV_c(\mathbf{x})2\tau} \end{pmatrix} \mathbf{D}, \tag{30}$$

with

$$\mathbf{D} = \mathbf{D}^\dagger \equiv \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}, \tag{31}$$

since

$$e^{-i\mathbf{V}_c 2\tau} = \mathbf{1} + (-i\mathbf{V}_c 2\tau) + \frac{1}{2!}(-i\mathbf{V}_c 2\tau)^2 + ..., \tag{32}$$

and

$$\mathbf{V}_c \equiv \begin{pmatrix} 0 & V_c(\mathbf{x}) \\ V_c(\mathbf{x}) & 0 \end{pmatrix} = \mathbf{D}^\dagger \begin{pmatrix} -V_c(\mathbf{x}) & 0 \\ 0 & V_c(\mathbf{x}) \end{pmatrix} \mathbf{D}, \tag{33}$$

with $\mathbf{D}\mathbf{D}^\dagger = 1$.

Problem 11: Derive Eq. (27) by writing the matrix V in the basis of adiabatic eigenstates

$$\begin{aligned} \phi_1(x) &= L_{11}(x)|1\rangle + L_{21}(x)|2\rangle, \\ \phi_2(x) &= L_{12}(x)|1\rangle + L_{22}(x)|2\rangle, \end{aligned} \tag{34}$$

with eigenvalues $E_1(x)$ and $E_2(x)$, respectively. Then, using the expansion

$$e^{-i\mathbf{V}2\tau} = \mathbf{1} + (-i\mathbf{V}2\tau) + \frac{1}{2!}(-i\mathbf{V}2\tau)^2 + ..., \tag{35}$$

show that in the adiabatic representation

$$e^{-i\mathbf{V}2\tau} = \begin{pmatrix} e^{-iE_1(x)2\tau} & 0 \\ 0 & e^{-iE_2(x)2\tau} \end{pmatrix}. \tag{36}$$

Finally, show that the diagonal matrix introduced by Eq. (36) can be rotated to the representation of diabatic states $|1\rangle$, $|2\rangle$ according to the similarity transformation

$$\mathbf{L}^{-1} \begin{pmatrix} e^{-iE_1(x)\tau} & 0 \\ 0 & e^{-iE_2(x)\tau} \end{pmatrix} \mathbf{L}. \tag{37}$$

Problem 12: (a) Write a Fortran code to implement the SOFT approach described in this section, where step [II] is numerically computed according to Eq. (27). Propagate $|\Psi(\mathbf{x};t)\rangle = \varphi_1(\mathbf{x};t)|1\rangle + \varphi_2(\mathbf{x};t)|2\rangle$, where $\varphi_1(\mathbf{x};0) = \varphi_1(\mathbf{x};0) = \Psi_0(x)$ and $\Psi_0(x)$ as defined in Eq. (6). Use $x_0 = -2.2$, $p_0 = 0$, $m = 1$, $\omega = 1$ and two coupled potential energy surfaces described by the potential energy matrix

$$V = \begin{pmatrix} V_1(x) & \delta \\ \delta & V_2(x) \end{pmatrix}, \tag{38}$$

where $\delta = 0.3$, $V_1(x) = m\omega^2(x - \bar{x})^2/2$ and $V_2(x) = -x^2/2 + x^4/22$; (b) Propagate $\Psi(\mathbf{x};t)$ according to the potential energy matrix introduced by Eq. (38), with $\delta = 0$ and compare your results with those obtained in item (a)

V SOFT Surface Hopping

The goal of this section is to introduce a numerically exact 'surface hopping' approach for simulations of nonadiabatic quantum dynamics (called Split-Operator Fourier-Transform/Surface-Hopping (SOFT/SH), throughout this section).

For simplicity, the SOFT/SH method is illustrated first as applied to the description of the time-dependent wave packet introduced by Eq. (23), evolving according to the two-level Hamiltonian introduced by Eq. (22), where the potential energy V is defined according to Eq. (38).

Considering that the coupling matrix elements are constant $V_c = \delta$, the embedded form of the Trotter expansion, introduced by Eq. (24), gives:

$$\begin{pmatrix} \varphi_1(\mathbf{x}; t+\tau) \\ \varphi_2(\mathbf{x}; t+\tau) \end{pmatrix} = \begin{pmatrix} \cos(V_c\tau) e^{-i\left(\frac{\hat{\mathbf{p}}^2}{2m}+V_1(\hat{x})\right)\tau} & -i\sin(V_c\tau) e^{-i\left(\frac{\hat{\mathbf{p}}^2}{2m}+V_A(\hat{x})\right)\tau} \\ -i\sin(V_c\tau) e^{-i\left(\frac{\hat{\mathbf{p}}^2}{2m}+V_A(\hat{x})\right)\tau} & \cos(V_c\tau) e^{-i\left(\frac{\hat{\mathbf{p}}^2}{2m}+V_2(\hat{x})\right)\tau} \end{pmatrix} \begin{pmatrix} \varphi_1(\mathbf{x}; t) \\ \varphi_2(\mathbf{x}; t) \end{pmatrix}, \quad (39)$$

when applied in conjunction with the analytic expression of the matrix M, introduced by Eq. (28), where $V_A(x) = (V_1(x) + V_2(x))/2$ is the average potential energy surface. Equation (39) indicates that the time-evolved wave-packet components $\varphi_j(\mathbf{x}; t+\tau)$ result from the interference between the corresponding components $\varphi_j(\mathbf{x}; t)$ propagated on the diabatic potential energy surfaces (*e.g.*, V_j) and the other component propagated on the *average potential*, V_A. The relative weights associated with these two contributions are given by the preexponential factors, $\cos(V_c\tau)$ and $\sin(V_c\tau)$, respectively.

The resulting time-dependent picture is particularly suitable for the development of the stochastic SOFT/SH method. The approach is based on an ensemble of realizations associated with the diabatic wave-packet components $\varphi_j(\mathbf{x}; t)$. At any given time, a realization of $\varphi_j(\mathbf{x}; t)$ is propagated on its corresponding diabatic surface V_j for time τ, with probability proportional to $\cos(V_c\tau)$, and contributes to the time-evolved $\varphi_j(\mathbf{x}; t+\tau)$. Otherwise, such a realization propagates on the average potential V_A and contributes to the other time-evolved wave-packet component. When the diabatic propagation is based on the SOFT approach, the resulting method is a numerically exact surface hopping approach with stochastic switches between the two diabatic surfaces mediated by propagation on the average potential.

Problem 12': Write a Fortran code to implement the SOFT/SH approach described in this section. Propagate $|\Psi(\mathbf{x}; t)\rangle = \varphi_1(\mathbf{x}; t)|1\rangle + \varphi_2(\mathbf{x}; t)|2\rangle$, where $\varphi_1(\mathbf{x}; 0) = \varphi_1(\mathbf{x}; 0) = \Psi_0(x)$ and $\Psi_0(x)$ as defined in Eq. (6). Use $x_0 = -2.2$, $p_0 = 0$, $m = 1$, $\omega = 1$ and two coupled potential energy surfaces described by the potential energy matrix,

$$V = \begin{pmatrix} V_1(x) & \delta \\ \delta & V_2(x) \end{pmatrix}, \quad (40)$$

where $\delta = 0.3$, $V_1(x) = m\omega^2(x - \bar{x})^2/2$ and $V_2(x) = -x^2/2 + x^4/22$. Compare your results with those obtained in Problem 12.

VI Matching Pursuit Representation

The goal of this section is to generalize the grid-based representation of states, introduced in Sec. II, to representations generated according to the matching-pursuit algorithm as implemented for overcomplete basis sets of nonorthogonal basis functions.

The main advantage of overcomplete basis sets is that they provide non-unique representations, since there are multiple ways of expanding a target state as a linear combination of nonorthogonal basis functions (*i.e*, basis functions that can be expanded as linear combinations of the other basis functions in the set). Therefore, one can define expansions that exploit the benefit of non-uniqueness in order to simultaneously achieve *sparsity* (*i.e.*, representations with the fewest possible significant terms), *superresolution* (*i.e.*, a resolution that is higher than that possible with traditional nonadaptive methods) and *speed* (i.e., representations obtainable in $O(n)$ or $O(n\log(n)$ steps, where n is the number of basis functions in the basis set.

The matching pursuit method implements a greedy algorithm for representing a target state (wavefunction) by successive orthogonal projections onto elements of an overcomplete basis set as follows: The first step requires selecting the basis element $\mid 1\rangle$ that has maximum overlap with the target state $\mid \Psi_t\rangle$ (*i.e.*, the element that is resonant with the most prominent structure in $\mid \Psi_t\rangle$). The projection of such element is defined as follows:

$$|\Psi_t\rangle = c_1|1\rangle + |\varepsilon_1\rangle, \tag{41}$$

where $c_1 \equiv \langle 1|\Psi_t\rangle$. Note that by virtue of the definition of c_1 the residual vector $|\varepsilon_1\rangle$ is orthogonal to $|1\rangle$. Therefore, $\|\Psi_t\| < \|\varepsilon_1\|$. The next step involves the sub-decomposition of the residual vector $|\varepsilon_1\rangle$ by projecting it along the direction of its best match $|2\rangle$ as follows:

$$|\varepsilon_1\rangle = c_2|2\rangle + |\varepsilon_2\rangle, \tag{42}$$

where $c_2 \equiv \langle 2|\varepsilon_1\rangle$. Note that, since $|\varepsilon_2\rangle$ is orthogonal to $|2\rangle$, the norm of $|\varepsilon_2\rangle$ is smaller than the norm of $|\varepsilon_1\rangle$. This procedure is repeated each time on the resulting residue.

After n successive orthogonal projections, the norm of the residual vector $|\varepsilon_n\rangle$ is smaller than a desired precision ϵ. Therefore, the algorithm maintains norm conservation within a desired precision,

$$\| \varepsilon_n \| = \sqrt{1 - \sum_{j=1}^{n} |c_j|^2} < \epsilon, \tag{43}$$

just as in a linear orthogonal decomposition. The resulting expansion is

$$\langle \mathbf{x}|\Psi_t\rangle \approx \sum_{j=1}^{n} c_j \langle \mathbf{x}|j\rangle, \tag{44}$$

where the coefficients c_j are recursively defined as follows:

$$c_j = \langle j|\Psi_t\rangle - \sum_{k=1}^{j-1} c_k \langle j|k\rangle. \tag{45}$$

Matching pursuit coherent-state expansions can be obtained by successively selecting the basis functions according to a gradient-based optimization technique.[10] A parallel implementation under the Message Passing Interface (MPI) environment[11] can speed up the search for a satisfactory local minimum. Starting from an initial trial coherent state $|\chi_j\rangle$, we can optimize the parameters $x_j(k)$, $p_j(k)$ and $\gamma_j(k)$ so that they locally maximize the overlap with the target state. Initial guess parameters $\gamma_j(k)$, $x_j(k)$ and $p_j(k)$ can be chosen as defined by the basis elements of the previous wave-packet representation (or initial state).

Problem 13: Write a Fortran code to represent the target state $\tilde{\Psi}_0(\mathbf{x}) \equiv e^{-i\hat{p}^2/(2m)\tau/2} e^{-iV(x)\tau} e^{-i\hat{p}^2/(2m)\tau/2} \Psi_0(x)$, where $\tau = 0.1$ a.u. and $V(x)$ is defined as in Problem 6, as a matching pursuit expansion based on coherent-states $|x_j, p_j, \gamma_j\rangle$ parametrized as follows:

$$\langle x|x_j, p_j, \gamma_j\rangle = \left(\frac{\gamma_j}{\pi}\right)^{1/4} e^{-\frac{\gamma_j}{2}(x-x_j)^2 + ip_j(x-x_0)}. \tag{46}$$

VII MP/SOFT Method for Adiabatic Propagation

The goal of this section is to introduce the implementation of the SOFT method for adiabatic quantum propagation, described in Sec. III, in terms of dynamically adaptive coherent-state representations generated according to the matching-pursuit algorithm introduced in Sec. VI.

In order to implement the Trotter expansion,

$$\Psi_{t+\tau}(\mathbf{x}) = e^{-i[\frac{\hat{\mathbf{p}}^2}{2m}+V(\hat{\mathbf{x}})]\tau}\Psi_t(\mathbf{x}) \approx e^{-iV(\hat{\mathbf{x}})\tau/2}e^{-i\hat{\mathbf{p}}^2/(2m)\tau}e^{-iV(\hat{\mathbf{x}})\tau/2}\Psi_t(\mathbf{x}), \tag{47}$$

by using representations based on matching-pursuit coherent-state expansions, one can proceed according to the following steps:

- Step 1. Decompose the target function $\tilde{\Psi}_t(\mathbf{x}) \equiv e^{-iV(\hat{\mathbf{x}})\tau/2}\Psi_t(\mathbf{x})$ into matching pursuit coherent state expansions,

$$\tilde{\Psi}_t(\mathbf{x}) \approx \sum_{j=1}^{n} c_j \langle \mathbf{x}|\chi_j\rangle. \tag{48}$$

Here, $\langle \mathbf{x}|\chi_j\rangle$ are N-dimensional coherent states,

$$\langle \mathbf{x}|\chi_j\rangle \equiv \prod_{k=1}^{N} A_j(k)\text{exp}\left(-\frac{\gamma_j(k)}{2}(x(k)-x_j(k))^2 + ip_j(k)(x(k)-x_j(k))\right), \tag{49}$$

where $A_j(k)$ are normalization factors and $\gamma_j(k)$, $x_j(k)$ and $p_j(k)$ are complex-valued parameters selected according to the matching pursuit algorithm. The expansion coefficients c_j, introduced by Eq. (48), are defined before: $c_1 \equiv \langle\chi_1|\Psi_t\rangle$, and $c_j \equiv \langle\chi_j|\Psi_t\rangle - \sum_{k=1}^{j-1} c_k\langle\chi_j|\chi_k\rangle$, for $j = 2$–N.

- Step 2. Apply the kinetic energy part of the Trotter expansion to $\tilde{\Psi}_t(\mathbf{x})$ by Fourier transforming the coherent state expansion of $\tilde{\Psi}_t(\mathbf{x})$ to the momentum representation, multiplying it by exp$[-i(\mathbf{p}^2/2m)\tau]$ and finally computing the inverse Fourier transform of the product to obtain:

$$\widetilde{\Psi}_t(\mathbf{x}) = \sum_{j=1}^{n} c_j \langle \mathbf{x}|\tilde{\chi}_j\rangle, \tag{50}$$

where

$$\langle \mathbf{x}|\tilde{\chi}_j\rangle \equiv \prod_{k=1}^{N} A_j(k)\sqrt{\frac{m}{m+i\tau\gamma_j(k)}}\exp\left(\frac{\left(\frac{p_j(k)}{\gamma_j(k)} - i[x_j(k)-x(k)]\right)^2}{\frac{2}{\gamma_j(k)}+\frac{2i\tau}{m}} - \frac{p_j(k)^2}{2\gamma_j(k)}\right). \tag{51}$$

The resulting time-evolved wave-function is thus

$$\Psi_{t+\tau}(\mathbf{x}) = \sum_{j=1}^{n} c_j e^{-iV(\mathbf{x})\tau/2}\langle \mathbf{x}|\tilde{\chi}_j\rangle, \tag{52}$$

which can be re-expanded in coherent states as in step [1].

Note that the underlying computational task necessary for MP/SOFT quantum propagation is completely reduced to generating the coherent-state expansions defined by Eq. (48), since all of the other steps can be implemented analytically.

Problem 14: Note that the Fortran code developed in Problem 13 already implements the MP/SOFT method described in this section, as applied to the Harmonic potential of Problem 5. Now, loop the code for 100 steps and make the comparison between numerical and analytical results for the whole propagation time. Use ($\tau = 0.1$ a.u.) with $x_0 = -2.5$, $p_0 = 0$, and $\alpha = \omega m$, where $m = 1$ and $\omega = 1$.

VIII MP/SOFT Simulations of Nonadiabatic Dynamics in Pyrazine

MP/SOFT simulations of nonadiabatic dynamics can be efficiently performed according to the SOFT method outlined in Sec. IV, in conjunction with the matching-pursuit representation method introduced in Sec. VI. The goal of this section is to illustrate the resulting approach as applied to the description of the S_1/S_2 interconversion of pyrazine after $S_0 \rightarrow S_2$ photoexcitation. The photophysics of pyrazine provides a standard platform for study ultrafast internal conversion responsible for a broad band photoabsorption spectrum with a rather diffuse superimposed structure. The underlying excited state nonadiabatic dynamics, following $S_0 \rightarrow S_1(\pi,\pi^*)$, $S_2(n,\pi^*)$ photoexcitation, is also ideally suited to benchmark the capabilities of new theoretical methods since it has been extensively investigated both experimentally[12,13] and theoretically.[14–20] *Ab initio* calculations have characterized the existence of a conical intersection between the S_1 and S_2 states, leading to ultrafast intramolecular energy transfer.[14] The experimental S_2 photoabsorption spectrum has been reproduced by using a 4-mode model Hamiltonian, after convoluting the resulting spectrum with an experimental resolution function,[21] or explicitly including the effect of the remaining vibrational modes as a weakly coupled harmonic bath.[15] These two models have allowed for direct comparisons between benchmark calculations and state-of-the-art semiclassical and quantum mechanical methods, including the multiconfigurational time dependent Hartree (MCTDH) approach,[15] the Herman Kluk semiclassical initial value representation (SC IVR) method,[17] the multiple spawning quantum approach,[18] the time dependent Gauss-Hermite discrete value representation (TDGH-DVR) method,[19] and the coupled coherent states (CCS) technique.[20] Here, the capabilities of the generalized MP/SOFT approach are evaluated as applied to the description of the photoabsorption spectroscopy of pyrazine.

The problem concerns the propagation of the wave-packet components $\varphi_1(\mathbf{x};t)$ and $\varphi_2(\mathbf{x};t)$, introduced by Eq. (23), as the molecule undergoes excited state interconversion dynamics at the conical intersection of the S_1 and S_2 coupled potential energy surfaces. Therefore, it can be assumed that the initial state is defined according to Eq. (23) in terms of the two ground vibrational state wave-packet components,

$$\begin{aligned}\varphi_1(\mathbf{x};0) &= 0,\\ \varphi_2(\mathbf{x};0) &= \prod_{j=1}^{N}\left(\frac{1}{\pi}\right)^{1/4} e^{-x(j)^2/2},\end{aligned} \tag{53}$$

associated with the S_1 and S_2 states, respectively. Here, N is the dimensionality of the system, as defined by the number of vibrational modes explicitly considered in the model (*i.e.*, $N = 4$ in the reduced model system, and $N = 24$ in the full-dimensional model).

The S_2 photoabsorption spectrum $I(\omega)$ can be computed as the Fourier transform of the survival amplitude $C(t)$,

$$I(\omega) \propto \omega \int_{\infty}^{\infty} dt C(t) e^{i(\omega+\epsilon_0)t}, \tag{54}$$

where ϵ_0 denotes the energy of the ground vibrational state of pyrazine, and $\omega = 2\pi c/\lambda$. The time-dependent survival amplitude,

$$C(t) = \langle\Psi(0)|e^{-i\hat{H}t}|\Psi(0)\rangle = \int d\mathbf{x}\Psi^{\star}(\mathbf{x};-t/2)\Psi(\mathbf{x};t/2), \tag{55}$$

is obtained by overlapping the time-evolved states $\Psi(\mathbf{x};t/2) = e^{-i\hat{H}t/2}\Psi(\mathbf{x};0)$ and $\Psi(\mathbf{x};-t/2) = e^{i\hat{H}t/2}\Psi(\mathbf{x};0)$, after propagating the initial state $\Psi(\mathbf{x};0)$ forward and backward in time on the nonadiabatically coupled excited electronic states. Here, $\Psi(\mathbf{x};0)$ is the initial ground state wavefunction, multiplied by the transition dipole moment which is assumed to be constant as a function of nuclear coordinates (Condon approximation).

The Hamiltonian $\hat{H} = \hat{H}_0 + \hat{V}_c$ of pyrazine is defined as follows:

$$\hat{H}_0 = \sum_j -\frac{1}{2m_j}\frac{\partial^2}{\partial Q_j^2}\left(|1\rangle\langle 1| + |2\rangle\langle 2|\right) + \sum_j \frac{1}{2} m_j \omega_j^2 Q_j^2 \left(|1\rangle\langle 1| + |2\rangle\langle 2|\right) + \Delta\left(|2\rangle\langle 2| - |1\rangle\langle 1|\right) \\ + \sum_{i\in G_1}\left(a_i|1\rangle\langle 1| + b_i|2\rangle\langle 2|\right)Q_i + \sum_{(i,j)\in G_2}\left(a_{i,j}|1\rangle\langle 1| + b_{i,j}|2\rangle\langle 2|\right)Q_iQ_j, \tag{56}$$

and

$$\hat{V}_c = \sum_{i\in G_3} c_i\left(|1\rangle\langle 2| + |2\rangle\langle 1|\right)Q_i + \sum_{(i,j)\in G_4} c_{i,j}\left(|1\rangle\langle 2| + |2\rangle\langle 1|\right)Q_iQ_j. \tag{57}$$

The parameters introduced by these equations are readily available in Ref. [15] and have been obtained at the complete-active-space self-consistent-field (CASSCF) *ab initio* level,[21] including a total of 102 coupling constants a_i, b_i, c_i, $a_{i,j}$, $b_{i,j}$ and $c_{i,j}$, explicitly describing the 24 vibrational modes of pyrazine. In addition, to facilitate the comparison with experimental data, the 24-dimensional potential energy surfaces should be shifted in energy by 4.06 eV. A more recent set of parameters has been kindly supplied by Meyer and co-workers and is available upon request.

The first and second terms, introduced by Eq. (56), define the harmonic expansion of the diabatic surfaces, where ω_j are the experimental ground-state vibrational frequencies, and $m_j = \omega_j^{-1}$. Further, $Q_j = (x_j - x_j^{eq})/(m_j\omega_j)^{-1/2}$ are the dimensionless normal-mode coordinates. The third term in Eq. (56) introduces the couplings between the S$_1$ and S$_2$ potential energy surfaces at the ground-state equilibrium configuration ($\mathbf{Q} = 0$). The fourth and fifth terms in Eq. (56) include the linear and quadratic contributions to the diabatic-state expansions, where G_1 and G_2 indicate the set of modes having A_g and B_{2g} symmetry, respectively. The nonadiabatic couplings are described to second order accuracy, as given by Eq. (57), where G_3 represents the modes with symmetry B_{1g} that linearly couple the S$_1$ and S$_2$ states, and G_4 is the set of all pairs of modes the product of which has B_{1g} symmetry, including the combinations $A_g \times B_{1g}$, $B_{2g} \times B_{3g}$, $A_u \times B_{1u}$, and $B_{2u} \times B_{3u}$.

A reduced 4-mode model can be constructed by following earlier work,[15] including only the vibronic coupling mode ν_{10a} and the three totally symmetric modes with the strongest linear coupling parameters, ν_{6a}, ν_1 and ν_{9a}. In addition, to facilitate the comparison with experimental data, the 4-dimensional potential energy surfaces should be shifted in energy by 3.94 eV.

Problem 15: (a) Assuming that the initial state of pyrazine is defined according to Eq. (53), obtain the analytic expressions of $\varphi_1'(\mathbf{x};\tau)$ and $\varphi_2'(\mathbf{x};\tau)$, as defined in Eq. (25); (b) Modify the Fortran code developed in Problem 13 in order to implement Eq. (26), defining the coupling matrix $\mathbf{M}$ in accord with Eq. (28) and the model Hamiltonian introduced by the Eqs. (56) and (57). Hint: Express the sine and cosine functions, introduced by Eq. (28), in terms of exponentials and compute the Gaussian integrals analytically by using $\int dx\, \exp(-a_2x^2 + a_1x + a_0) = \sqrt{\pi/a_2}\, \exp(a_0 + a_1^2/(4a_2))$. (c) Loop the code and propagate the wave-packet for the 4-dimensional and the full-dimensional models, storing the wave-packet at all intermediate times. (d) Compute the survival amplitude as defined in Eq. (55) and the photoabsorption spectrum $I(\omega)$, as the Fourier transform of the survival amplitude $C(t)$, introduced by Eq. (54). Check your results as compared to earlier calculations.[15,17–20]

IX MP/SOFT Computations of Thermal Correlation Functions

The goal of this section is to introduce a generalization of the MP/SOFT method for the description of thermal-equilibrium density matrices, finite-temperature time-dependent expectation values and time-correlation functions.

Consider the problem of computing thermal correlation functions,

$$C(t) = \langle A(0)B(t)\rangle = Z^{-1}\text{Tr}[e^{-\beta\hat{H}_0}\hat{A}e^{i\hat{H}_1 t}\hat{B}e^{-i\hat{H}_1 t}], \tag{58}$$

where $\langle\cdots\rangle$ indicates the Boltzmann ensemble average at temperature $T = 1/(k_B\beta)$, with k_B the Boltzmann constant; $\hat{A}$ and $\hat{B}$ are quantum-mechanical operators associated with measurements of observables at time 0 and t, respectively; $Z = \text{Tr}[e^{-\beta\hat{H}_0}]$ is the canonical partition function; and $\hat{H}_j = -\nabla^2_{\mathbf{x}}/(2m) + V_j(\hat{\mathbf{x}})$ is the Hamiltonian of the system of interest with N degrees of freedom interacting according to the potential $V_j(\hat{\mathbf{x}})$. An example is the correlation function $C(t)$ for a system evolving on the excited state potential energy surface $V_1(\hat{\mathbf{x}})$, as would result from a photoexcitation process after the initial preparation at thermal-equilibrium in the ground state potential energy surface $V_0(\hat{\mathbf{x}})$. To keep the notation as simple as possible, all expressions are written in mass-weighted coordinates and atomic units, so that all degrees of freedom have the same mass m and $\hbar = 1$.

Note that Eq. (58) provides an expression for computing not only time-dependent thermal correlation functions but also thermal-equilibrium ensemble averages $\langle A\rangle = Z^{-1}\text{Tr}[e^{-\beta\hat{H}_0}\hat{A}]$, when $\hat{B} = 1$, and finite-temperature time-dependent ensemble averages,

$$\langle B(t)\rangle = Z^{-1}\text{Tr}[e^{-\beta\hat{H}_0}e^{i\hat{H}_1 t}\hat{B}e^{-i\hat{H}_1 t}], \tag{59}$$

when $\hat{A} = 1$.

Thermal correlation functions $C(t)$ are obtained according to the following symmetrized form of Eq. (58):

$$C(t) = Z^{-1}\int d\mathbf{x}\int d\mathbf{x}'\int d\mathbf{x}''\langle\mathbf{x}|e^{-\frac{\beta}{2}\hat{H}_0}|\mathbf{x}'\rangle A(\mathbf{x}')\langle\mathbf{x}'|e^{i\hat{H}_1 t}\hat{B}e^{-i\hat{H}_1 t}|\mathbf{x}''\rangle\langle\mathbf{x}''|e^{-\frac{\beta}{2}\hat{H}_0}|\mathbf{x}\rangle. \tag{60}$$

The computational task necessary to obtain $C(t)$, according to Eq. (60), requires obtaining the matrix elements $A(\mathbf{x}')\langle\mathbf{x}'|e^{-\frac{\beta}{2}\hat{H}_0}|\mathbf{x}\rangle$ and $\langle\mathbf{x}''|e^{-\frac{\beta}{2}\hat{H}_0}|\mathbf{x}\rangle$ and the subsequent real-time propagation for time t, according to $\hat{H}_1$. The matrix elements are computed, as described below by imaginary-time integration of the Bloch equation according to $\hat{H}_0$. The extension of the MP/SOFT method, introduced in this section, involves the numerically exact treatment of both the real- and imaginary-time propagation steps as described below for the imaginary-time propagation. The real-time propagation is analogously performed by simply implementing the variable transformation $\beta \to -it$ from imaginary to real time.

The Boltzmann-operator matrix-elements are obtained by solving the Bloch equation,[22]

$$\{\frac{\partial}{\partial\beta} - \frac{1}{2m}\nabla^2_{\mathbf{x}} + V_0(\mathbf{x})\}\rho(\mathbf{x},\mathbf{x}';\beta) = 0, \tag{61}$$

for $\rho(\mathbf{x},\mathbf{x}';\beta) \equiv \langle\mathbf{x}|e^{-\beta\hat{H}_0}|\mathbf{x}'\rangle$ subject to the initial condition given by the high-temperature approximation,

$$\rho(\mathbf{x},\mathbf{x}';\epsilon) = \left(\frac{m}{2\pi\epsilon}\right)^{1/2} e^{-\frac{\epsilon}{2}[V_0(\mathbf{x})+V_0(\mathbf{x}')]}e^{-\frac{m}{2\epsilon}(\mathbf{x}-\mathbf{x}')^2}, \tag{62}$$

where ϵ defines a sufficiently high temperature $T = 1/(k_B\epsilon)$.

Equation (61) is formally integrated as follows,

$$\rho(\mathbf{x},\mathbf{x}';\beta) = \int d\mathbf{x}''\rho(\mathbf{x},\mathbf{x}'';\beta-\epsilon)\rho(\mathbf{x}'',\mathbf{x}';\epsilon), \tag{63}$$

where the propagator $\rho(\mathbf{x},\mathbf{x}'';\beta-\epsilon) \equiv \langle\mathbf{x}|e^{-(\beta-\epsilon)\hat{H}_0}|\mathbf{x}''\rangle$ is imaginary-time sliced by repeatedly inserting the resolution of identity,

$$\hat{\mathbf{1}} = \int d\mathbf{x}_j|\mathbf{x}_j\rangle\langle\mathbf{x}_j|, \tag{64}$$

yielding,

$$\langle \mathbf{x}|e^{-(\beta-\epsilon)\hat{H}_0}|\mathbf{x}''\rangle = \int d\mathbf{x}_{s-1}... \int d\mathbf{x}_1 \langle \mathbf{x}|e^{-i\hat{H}_0\tau}|\mathbf{x}_{s-1}\rangle ... \langle \mathbf{x}_1|e^{-i\hat{H}_0\tau}|\mathbf{x}''\rangle, \tag{65}$$

where $\tau \equiv -i(\beta-\epsilon)/s$ is a sufficiently thin imaginary-time slice.

Each finite-time propagator, introduced by Eq. (65), is approximated for sufficiently small imaginary-time slices τ by the Trotter expansion to second-order accuracy,

$$e^{-i\ \hat{H}_0\tau} \approx e^{-i\ V_0(\hat{\mathbf{x}})\tau/2} e^{-i\ \frac{\hat{\mathbf{p}}^2}{2m}\tau}\ e^{-i\ V_0(\hat{\mathbf{x}})\tau/2}. \tag{66}$$

The MP/SOFT propagation of the initial condition, introduced by Eq. (62), is performed according to the Trotter expansion introduced by Eq. (66) entailing the following steps:

- Step [1]: Decompose $\tilde{\rho}(\mathbf{x},\mathbf{x}';\epsilon) \equiv e^{-iV_0(\mathbf{x})\tau/2}\rho(\mathbf{x},\mathbf{x}';\epsilon)$ in a matching-pursuit coherent-state expansion:

$$\tilde{\rho}(\mathbf{x},\mathbf{x}';\epsilon) \approx \sum_{j=1}^{n} c_j \phi_j(\mathbf{x})[\phi_j'(\mathbf{x}')]^*, \tag{67}$$

where $\phi_j(\mathbf{x})$ and $\phi_j'(\mathbf{x})$ are N-dimensional coherent-states defined as follows,

$$\phi_j(\mathbf{x}) \equiv \prod_{k=1}^{N} A_{\phi_j}(k) e^{-\gamma_{\phi_j}(k)\left(x(k)-x_{\phi_j}(k)\right)^2/2} e^{i\ p_{\phi_j}(k)\left(x(k)-x_{\phi_j}(k)\right)}, \tag{68}$$

with complex-valued coordinates $x_{\phi_j}(k) \equiv r_{\phi_j}(k) + i d_{\phi_j}(k)$, momenta $p_{\phi_j}(k) \equiv g_{\phi_j}(k) + i f_{\phi_j}(k)$ and scaling parameters $\gamma_{\phi_j}(k) \equiv a_{\phi_j}(k) + i b_{\phi_j}(k)$. The normalization constants are

$$A_j(k) = \left(\frac{a_j(k)}{\pi}\right)^{1/4} \text{exp}[-\frac{1}{2}a_j(k)d_j(k)^2]\text{exp}[-d_j(k)g_j(k) - \frac{1}{2a_j(k)}\left(b_j(k)d_j(k)+f_j(k)\right)^2]. \tag{69}$$

The expansion coefficients, introduced by Eq. (67), are defined as follows:

$$c_j \equiv \begin{cases} I_j, & \text{when} \quad j=1, \\ I_j - \sum_{k=1}^{j-1} c_k \langle \phi_j|\phi_k\rangle\langle\phi_k'|\phi_j'\rangle, & \text{for} \quad j = 2-n, \end{cases} \tag{70}$$

where the overlap integral I_j is defined as follows,

$$I_j \equiv \int d\mathbf{x}' d\mathbf{x}\ \phi_j(\mathbf{x})\tilde{\rho}(\mathbf{x},\mathbf{x}';\epsilon)[\phi_j'(\mathbf{x}')]^*. \tag{71}$$

- Step [2]: Analytically Fourier transform the coherent-state expansion to the momentum representation, apply the kinetic energy part of the Trotter expansion and analytically inverse Fourier transform the resulting expression back to the coordinate representation to obtain the imaginary-time evolved Boltzmann-operator matrix elements:

$$\rho(\mathbf{x},\mathbf{x}';\epsilon+i\tau) = \sum_{j=1}^{n} c_j e^{-iV_0(\mathbf{x})\tau/2}\widetilde{\phi}_j(\mathbf{x})[\phi_j'(\mathbf{x}')]^*, \tag{72}$$

where

$$\widetilde{\phi}_j(\mathbf{x}) \equiv \prod_{k=1}^{N} A_{\widetilde{\phi}_j}(k)\sqrt{\frac{m}{m+i\tau\gamma_{\widetilde{\phi}_j}(k)}}\text{exp}\left(\frac{\left(\frac{p_{\widetilde{\phi}_j}(k)}{\gamma_{\widetilde{\phi}_j}(k)} - i(x_{\widetilde{\phi}_j}(k)-x(k))\right)^2}{\left(\frac{2}{\gamma_{\widetilde{\phi}_j}(k)}+\frac{i2\tau}{m}\right)} - \frac{p_{\widetilde{\phi}_j}(k)^2}{2\gamma_{\widetilde{\phi}_j}(k)}\right). \tag{73}$$

Note that the MP/SOFT approach reduces the computational task necessary for the imaginary- or real-time propagation of the Boltzmann operator matrix elements $\rho(\mathbf{x}, \mathbf{x}'; \beta)$ to the problem of recursively generating the coherent-state expansions introduced by Eq. (67).

Coherent-state expansions are obtained as before by combining the matching pursuit algorithm and a gradient-based optimization method as follows:

- Step [1.1]. Evolve the complex-valued parameters, that define the initial trial coherent-states $\phi_j(\mathbf{x})$ and $\phi_j'(\mathbf{x})$, to locally maximize the overlap integral I_j, introduced in Eq. (71). Parameters $x_{\phi_1}(k), p_{\phi_1}(k), \gamma_{\phi_1}(k)$ and $x_{\phi_1'}(k), p_{\phi_1'}(k), \gamma_{\phi_1'}(k)$ of the corresponding local maximum define the first pair of coherent-states ϕ_1 and ϕ_1' in the expansion introduced by Eq. (67) and the first expansion coefficient c_1, as follows: $\tilde{\rho}(\mathbf{x}, \mathbf{x}'; \epsilon) = c_1 \phi_1(\mathbf{x})[\phi_1'(\mathbf{x}')]^* + \varepsilon_1(\mathbf{x}, \mathbf{x}')$, where $c_1 \equiv I_1$, as defined according to Eq. (71). Note that due to the definition of c_1, the residue $\varepsilon_1(\mathbf{x}, \mathbf{x}')$ does not overlap with the product state $\phi_1(\mathbf{x})[\phi_1'(\mathbf{x}')]^*$. Therefore, the norm of the remaining residue $\varepsilon_1(\mathbf{x}, \mathbf{x}')$ is smaller than the norm of the initial target state $\tilde{\rho}(\mathbf{x}, \mathbf{x}'; \epsilon)$ —*i.e.*, $\| \varepsilon_1 \| < \| \tilde{\rho} \|$.

- Step [1.2]. Goto [1.1], replacing $\tilde{\rho}(\mathbf{x}, \mathbf{x}'; \epsilon)$ by $\varepsilon_1(\mathbf{x}, \mathbf{x}')$ —*i.e.*, sub-decompose the residue by its projection along the direction of its locally optimum match as follows: $\varepsilon_1(\mathbf{x}, \mathbf{x}') = c_2 \phi_2(\mathbf{x})[\phi_2'(\mathbf{x}')]^* + \varepsilon_2(\mathbf{x}, \mathbf{x}')$, where
$$c_2 \equiv \int d\mathbf{x}' d\mathbf{x}\, \phi_2(\mathbf{x}) \varepsilon_1(\mathbf{x}, \mathbf{x}')[\phi_2'(\mathbf{x}')]^*. \tag{74}$$
Note that $\| \varepsilon_2 \| < \| \varepsilon_1 \|$, since $\varepsilon_2(\mathbf{x}, \mathbf{x}')$ is orthogonal to the product state $\phi_2(\mathbf{x})[\phi_2'(\mathbf{x}')]^*$.

Step [1.2] is repeated each time on the resulting residue. After n successive projections, the norm of the residue ε_n is smaller than a desired precision ϵ —*i.e.*, $\| \varepsilon_n \| = (1 - \sum_{j=1}^{n} |c_j|^2)^{1/2} < \epsilon$, and the resulting expansion is given by Eq. (67).

It is important to mention that the computational bottleneck of the MP/SOFT method involves the calculation of overlap matrix elements $\langle \phi_j | e^{-iV_j(\hat{\mathbf{x}})\tau/2} | \widetilde{\phi}_k \rangle$ and $\langle \phi_j | e^{-iV_j(\hat{\mathbf{x}})\tau/2} | \phi_k \rangle$, where $|\phi_k\rangle$ and $|\widetilde{\phi}_k\rangle$ are localized Gaussians introduced by Eqs. (68) and (73), respectively. The underlying computational task is however trivially parallelized according to a portable Single-Program-Multiple-Data streams code that runs under the Message-Passing-Interface (MPI) environment.

The overlap integrals are most efficiently computed in applications to reaction surface Hamiltonians where a large number of harmonic modes can be *arbitrarily* coupled to a few reaction (tunneling) coordinates (see, *e.g.*, Models I and II in Ref. [3] and the reaction surface Hamiltonians in Refs. [23–25]). For such systems, the Gaussian integrals over harmonic coordinates can be analytically computed and the remaining integrals over reaction coordinates are efficiently obtained according to numerical quadrature techniques. For more general Hamiltonians, the overlap matrix elements can be approximated by analytic Gaussian integrals when the choice of width parameters $\gamma_j(k)$ allows for a local expansion of $V_j(\hat{\mathbf{x}})$ to second order accuracy. Otherwise, the quadratic approximation is useful for numerically computing the corresponding full-dimensional integrals according to variance-reduction Monte Carlo techniques.

Problem 16: Evaluate the accuracy and efficiency of the MP/SOFT methodology in terms of explicit calculations of time-dependent position ensemble averages and position-position thermal correlation functions for the asymmetric quartic oscillator described by the following Hamiltonian:
$$\hat{H}_1 = \frac{\hat{p}^2}{2m} + V_1(x), \tag{75}$$
where
$$V_1(x) = \frac{1}{2} m\omega^2 x^2 - cx^3 + cx^4, \tag{76}$$

with $m = 1$ a.u., $\omega = \sqrt{2}$ a.u., and $c = 0.1$ a.u. The system is initially prepared at thermal equilibrium, with $\beta = 0.5$ a.u. on the displaced potential energy surface,

$$V_0(x) = \frac{1}{2}m\omega^2(x-a)^2 - c(x-a)^3 + c(x-a)^4, \tag{77}$$

with $a = 1$ a.u. Check your results as compared to earlier calculations.[4,26–28]

The model system, introduced by Eqs. (75)—(77), is particularly interesting since the highly anharmonic potential leads to ultrafast dephasing within a few oscillation periods as well as later rephasing of wavepacket motion due to the effect of quantum coherences. The underlying dynamics can be described by rigorous quantum-mechanical approaches and has been investigated in terms of semiclassical approaches based on coherent-state representations.[4,26–28] Therefore, the model is ideally suited for a rigorous analysis of the accuracy and efficiency of the MP/SOFT method as compared to classical, semiclassical and benchmark quantum-mechanical calculations.

X Acknowledgments

V.S.B. acknowledges supercomputer time from the National Energy Research Scientific Computing (NERSC)Center and financial support from Research Corporation, Research Innovation Award # RI0702, a Petroleum Research Fund Award from the American Chemical Society PRF # 37789-G6, a junior faculty award from the F. Warren Hellman Family, the National Science Foundation (NSF) Career Program Award CHE # 0345984, the NSF Nanoscale Exploratory Research (NER) Award ECS # 0404191, the Alfred P. Sloan Fellowship (2005-2006) from the Sloan Foundation, a Camille Dreyfus Teacher-Scholar Award for 2005, a Yale Junior Faculty Fellowship in the Natural Sciences (2005), and start-up package funds from the Provost's office at Yale University.

References

[1] Y. Wu and V. S. Batista. *J. Chem. Phys.*, 118:6720, 2003.

[2] Y. Wu and V. S. Batista. *J. Chem. Phys.*, 119:7606, 2003.

[3] Y. Wu and V. S. Batista. *J. Chem. Phys.*, 121:1676, 2004.

[4] X. Chen, Y. Wu, and V. S. Batista. *J. Chem. Phys.*, 122:64102, 2005.

[5] Y. Wu, M. F. Herman, and V. S. Batista. *J. Chem. Phys.*, 122:114114, 2005.

[6] Y. Wu and V. S. Batista. *J. Chem. Phys.*, 124:224305, 2006.

[7] X. Chen and V. S. Batista. *J. Chem. Phys.*, 2006. submitted.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. In *Numerical Recipes*, chapter 12. Cambridge University Press, Cambridge, 1986. (http://www.library.cornell.edu/nr/bookfpdf/f12-2.pdf).

[9] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. In *Numerical Recipes*, chapter 11. Cambridge University Press, Cambridge, 1986. (http://www.library.cornell.edu/nr/bookfpdf.html).

[10] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. In *Numerical Recipes*, chapter 10. Cambridge University Press, Cambridge, 1986. (http://www.library.cornell.edu/nr/bookfpdf/f10-6.pdf).

[11] http://www-unix.mcs.anl.gov/mpi/,http://www-unix.mcs.anl.gov/mpi/tutorial/gropp/talk.html.

[12] I. Yamazaki, T. Murao, T. Yamanaka, and K. Yoshihara. *Faraday Discuss. Chem. Soc.*, 75:395, 1983.

[13] K. K. Innes, I. G. Ross, and W. R. Moonaw. *J. Mol. Spectrosc.*, 32:492, 1988.

[14] H. Köppel, W. Domcke, and L. S. Cederbaum. *Adv. Chem. Phys.*, 57:59–245, 1984.

[15] A. Raab, G. A. Worth, H. D. Meyer, and L. S. Cederbaum. *J. Chem. Phys.*, 110:936–946, 1999.

[16] G. Stock, C. Woywood, W. Domcke, T. Swinney, and B. S. Hudson. *J. Chem. Phys.*, 103:6851, 1995.

[17] M. Thoss, W. H. Miller, and G. Stock. *J. Chem. Phys.*, 112:10282–10292, 2000.

[18] M. Ben-Nun and T. J. Martinez. page 439. Wiley, New York, 2002.

[19] C. Coletti and G. D. Billing. *Chem. Phys. Lett.*, 368:289–298, 2003.

[20] D. V. Shalashilin and M. S. Child. *J. Chem. Phys.*, 121:3563–3568, 2004.

[21] C. Woywood, W. Domcke, A. L. Sobolewski, and H. J. Werner. *J. Chem. Phys.*, 100:1400, 1994.

[22] R.P. Feynman. In *Statistical Mechanics*. Benjamin, Reading, 1972.

[23] V. Guallar, V. S. Batista, and W. H. Miller. *J. Chem. Phys.*, 113:9510, 2000.

[24] V. Guallar, V. S. Batista, and W. H. Miller. *J. Chem. Phys.*, 110:9922, 1999.

[25] M. Petkovic and O. Kuhn. *J. Phys. Chem. A*, 107:8458, 2003.

[26] J.H. Shao and N. Makri. *J. Phys. Chem. A*, 103:7753, 1999.

[27] E. Jezek and N. Makri. *J. Phys. Chem. A*, 105:2851, 2001.

[28] N. Makri and W.H. Miller. *J. Chem. Phys.*, 116:9207, 2002.

Summer School on Computational Materials Science

Lecture Notes: Ab Initio Molecular Dynamics Simulation Methods in Chemistry

Victor S. Batista[*]

Yale University, Department of Chemistry, P.O.Box 208107, New Haven, Connecticut 06520-8107, U.S.A.

I Solutions to Problems

Problem 1:

In order to visualize the output of this program, cut the source code attached below save it in a file named Problem1.f, compile it by typing

```
 gfortran Problem1.f -o Problem1
```

run it by typing

```
./Problem1
```

Visualize the output as follows: type

```
gnuplot
```

then type

```
plot ``arch.0000''
```

That will show the representation of the Gaussian state, introduced in Eq. (6) in terms of an array of numbers associated with a grid in coordinate space. To exit, type

```
quit
```

[*]E-mail: victor.batista@yale.edu

Download from (http://ursula.chem.yale.edu/~batista/classes/summer/P1/Problem1.f),

```
      PROGRAM Problem_1
      call Initialize()
      CALL SAVEWF(0)
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk
      COMPLEX chi,EYE
      REAL omega,xmin,xmax,dx,pi,mass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      xmin=-20.
      xmax=20.
      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      pk=0.0
      xk=0.0
      mass=1.0
      alpha=mass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(j)
c
c     Save Wave-packet in coordinate space
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,j
      COMPLEX chi,EYE
      REAL RV,omega,xmin,xmax,dx,pi,mass,xk,pk,x,alpha,Vpot,RKE
      character*9 B
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      write(B, '(A,i4.4)') 'arch.', j
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      do kk=1,nptx
         x=xmin+kk*dx
         WRITE(1,22) x,chi(kk)
      end do
      CLOSE(1)
 22   FORMAT(6(e13.6,2x))
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 2:

In order to visualize the output of this program, cut the source code attached below save it in a file named Problem2.f, compile it by typing

```
 gfortran Problem2.f -o Problem2
```

run it by typing

```
./Problem2
```

Visualize the output as follows: type

```
gnuplot
```

then type

```
plot ``nume.0000''
```

That will show the representation of the amplitude of the Fourier transform of the Gaussian state, introduced in Eq. (6), in terms of an array of numbers associated with a grid in momentum space. In order to visualize the analytic results on top of the numerical values type

```
replot ``anal.0000''
```

In order to visualize the numerically computed phases as a function of p type

```
plot ``nume.0000 u 1:3''
```

and to visualize the analytic results on top of the numerical values type

```
replot ``anal.0000''
```

To exit, type

```
quit
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P2/Problem2.f),

```
      PROGRAM Problem2
      call Initialize()
      CALL SAVEFT()
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk
      COMPLEX chi,EYE
      REAL omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk
      xmin=-20.
      xmax=20.
      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      pk=0.0
      xk=5.0
      rmass=1.0
      alpha=rmass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SAVEFT()
c
c     Save wave-packet in momentum space
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts,j
      REAL theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri
      COMPLEX eye,chi,Psip
      character*9 B1,B2
      parameter(npts=10,nptx=2**npts)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx)
      j=0
      write(B1, '(A,i4.4)') 'anal.', j
      OPEN(1,FILE=B1)
      write(B2, '(A,i4.4)') 'nume.', j
      OPEN(2,FILE=B2)
      CALL fourn(chi,nptx,1,-1)
      pi = acos(-1.0)
      alenx=xmax-xmin
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         p=0.
         if(nx.ne.0) p = real(nx)*2.*pi/alenx
c     Numerical Solution
         chi(kx)=chi(kx)*alenx/sqrt(2.0*pi)/nptx
         re=chi(kx)
         ri=imag(chi(kx))
         IF(re.NE.0) theta=atan(ri/re)
         rm=abs(chi(kx))
         IF(abs(p).LE.(4.)) WRITE(2,22) p,rm,theta
         IF(nx.EQ.(nptx/2)) WRITE(2,22)
c     Analytic Solution
         CALL FT_analy(Psip,p)
         re=Psip
         ri=imag(Psip)
         IF(re.NE.0) theta=atan(ri/re)
         rm=abs(Psip)
         IF(abs(p).LE.(4.)) WRITE(1,22) p,rm,theta
         IF(nx.EQ.(nptx/2)) WRITE(1,22)
      end do
      CALL fourn(chi,nptx,1,1)
 22   FORMAT(6(e13.6,2x))
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine FT_analy(Psip,p)
c
c     Analytic Fourier Transform of the initial Gaussian wave-packet
c
      IMPLICIT NONE
      REAL p,pi,alpha,rmass,xk,pk,omega
      COMPLEX Psip,c0,c1,c2,eye
      common /packet/  rmass,xk,pk
      eye=(0.0,1.0)
      omega=1.
      alpha = rmass*omega
      pi=acos(-1.0)
      c2=alpha/2.
      c1=alpha*xk+eye*(pk-p)
      c0=-alpha/2.*xk**2-eye*pk*xk
      Psip=sqrt(pi/c2)/sqrt(2.0*pi)*(alpha/pi)**0.25
     1     *exp(c1**2/(4.0*c2))*exp(c0)
      return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines from Numerical Recipes
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
```

```
                    I3REV=I2REV+I3-I2
                    TEMPR=DATA(I3)
                    TEMPI=DATA(I3+1)
                    DATA(I3)=DATA(I3REV)
                    DATA(I3+1)=DATA(I3REV+1)
                    DATA(I3REV)=TEMPR
                    DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 3:

In order to visualize the output of this program, cut the source code attached below save it in a file named Problem3.f, compile it by typing

```
 gfortran Problem3.f -o Problem3
```

run it by typing

```
./Problem3
```

The printout on the screen includes the numerically expectation values $\langle\Psi_t|\hat{V}|\Psi_t\rangle$ and $\langle\Psi_t|\hat{x}|\Psi_t\rangle$.

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P3/Problem3.f),

```
      PROGRAM Problem3
      IMPLICIT NONE
      REAL x,VENERGY
      CALL Initialize()
      CALL PE(VENERGY)
      CALL Px(x)
      PRINT *, "<Psi|V|Psi>=",VENERGY
      PRINT *, "<Psi|x|Psi>=",x
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk
      COMPLEX chi,EYE
      REAL omega,xmin,xmax,dx,pi,mass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      xmin=-20.
      xmax=20.
      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      pk=0.0
      xk=0.0
      mass=1.0
      alpha=mass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,k
      COMPLEX chi
      REAL Vpot,RV,xmin,xmax,dx,x
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      dx=(xmax-xmin)/real(nptx)
      RV=0.0
      do k=1,nptx
         x=xmin+k*dx
         CALL VA(Vpot,x)
         RV=RV+chi(k)*Vpot*conjg(chi(k))*dx
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Px(RV)
c
```

```
c     Expectation Value of the position
c
      IMPLICIT NONE
      INTEGER nptx,npts,k
      COMPLEX chi
      REAL RV,xmin,xmax,dx,x
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      dx=(xmax-xmin)/real(nptx)
      RV=0.0
      do k=1,nptx
         x=xmin+k*dx
         RV=RV+chi(k)*x*conjg(chi(k))*dx
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      IMPLICIT NONE
      REAL V,x,mass,xk,pk,rk,omega
      common /packet/ mass,xk,pk
      omega=1.0
      rk=mass*omega**2
      V=0.5*rk*x*x
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 4:

In order to visualize the output of this program, cut the source code attached below save it in a file named Problem4.f, compile it by typing

```
 gfortran Problem4.f -o Problem4
```

run it by typing

```
./Problem4
```

The printout on the screen includes the numerically expectation values $\langle\Psi_t|\hat{p}|\Psi_t\rangle$, $\langle\Psi_t|\hat{T}|\Psi_t\rangle$ and $\langle\Psi_t|\hat{H}|\Psi_t\rangle$. Note that the analytic value of $\langle\Psi_t|\hat{T}|\Psi_t\rangle$ is $\hbar\omega/2 = 0.5$ in agreement with the numerical solution.

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P4/Problem4.f),

```
      PROGRAM Problem4
      CALL Initialize()
      CALL Pp(p)
      PRINT *, "<Psi|p|Psi>=",p
      CALL KE(RKE)
      PRINT *, "<Psi|T|Psi>=",RKE
      CALL PE(RV)
      PRINT *, "<Psi|H|Psi>=",RKE+RV
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk
      COMPLEX chi,EYE
      REAL omega,xmin,xmax,dx,pi,mass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      xmin=-20.
      xmax=20.
      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      pk=0.0
      xk=0.0
      mass=1.0
      alpha=mass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,k
      COMPLEX chi
      REAL Vpot,RV,xmin,xmax,dx,x
      PARAMETER(npts=10,nptx=2**npts)
      COMMON / wfunc/ chi(nptx)
      common /xy/  xmin,xmax
      dx=(xmax-xmin)/real(nptx)
      RV=0.0
      do k=1,nptx
         x=xmin+k*dx
         CALL VA(Vpot,x)
         RV=RV+chi(k)*Vpot*conjg(chi(k))*dx
      end do
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE KE(RKE)
c
```

```
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER kk,nptx,kx,nx,npts
      REAL dp,RKE,p,xmin,xmax,pi,alenx,dx,mass,xk,pk
      COMPLEX eye,chi,Psip,chic
      parameter(npts=10,nptx=2**npts)
      DIMENSION chic(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      COMMON / wfunc/ chi(nptx)
      RKE=0.0
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
      do kk=1,nptx
         chic(kk)=chi(kk)
      end do
      CALL fourn(chic,nptx,1,1)
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         p=0.
         if(nx.ne.0) p = real(nx)*dp
         chic(kx)=p**2/(2.0*mass)*chic(kx)/nptx
      end do
      CALL fourn(chic,nptx,1,-1)
      do kk=1,nptx
         RKE=RKE+conjg(chi(kk))*chic(kk)*dx
      end do
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Pp(pe)
c
c     Expectation value of the momentum
c
      IMPLICIT NONE
      INTEGER kk,nptx,kx,nx,npts
      REAL dp,pe,p,xmin,xmax,pi,alenx,dx,mass,xk,pk
      COMPLEX eye,chi,Psip,chic
      parameter(npts=10,nptx=2**npts)
      DIMENSION chic(nptx)
      common /xy/  xmin,xmax
      common /packet/mass,xk,pk
      COMMON / wfunc/ chi(nptx)
      pe=0.0
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
      do kk=1,nptx
         chic(kk)=chi(kk)
      end do
      CALL fourn(chic,nptx,1,1)
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         p=0.
         if(nx.ne.0) p = real(nx)*dp
         chic(kx)=p*chic(kx)/nptx
      end do
      CALL fourn(chic,nptx,1,-1)
      do kk=1,nptx
         pe=pe+conjg(chi(kk))*chic(kk)*dx
      end do
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      implicit none
      REAL V,x,mass,xk,pk,rk,omega
      common /packet/ mass,xk,pk
      omega=1.0
      rk=mass*omega**2
      V=0.5*rk*x*x
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines from Numerical Recipes
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
```

```
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 5:

Expanding the left-hand-side (l.h.s.) of Eq. (18) from the lecture notes gives:

$$e^{-i\hat{H}\tau} = 1 - i\hat{H}\tau - \frac{1}{2}\hat{H}^2\tau^2 + O(\tau^3), \tag{1}$$

where $\hat{H} = \hat{p}^2/(2m) + \hat{V}$. Therefore,

$$e^{-i\hat{H}\tau} = 1 - i\hat{H}\tau - \frac{1}{2}\frac{\hat{p}^4}{4m^2}\tau^2 - \frac{1}{2}\hat{V}^2\tau^2 - \frac{1}{2}\frac{\hat{p}^2}{2m}\hat{V}\tau^2 - \frac{1}{2}\hat{V}\frac{\hat{p}^2}{2m}\tau^2 + O(\tau^3), \tag{2}$$

In order to show that the Trotter expansion, introduced by Eq. (18), is accurate to second order in τ, we must expand the right-hand-side (r.h.s.) of Eq. (18) and show that suchh an expansion equals the r.h.s. of Eq. (2).

Expanding the right-hand-side (r.h.s.) of Eq. (18) gives,

$$\begin{aligned} e^{-iV(\hat{x})\tau/2}e^{-i\hat{p}^2\tau/(2m)}e^{-iV(\hat{x})\tau/2} = &\left(1 - i\hat{V}\tau/2 - \frac{1}{2}\hat{V}^2\tau^2/4 + O(\tau^3)\right)\left(1 - i\frac{\hat{p}^2}{2m}\tau - \frac{1}{2}\frac{\hat{p}^4}{4m^2}\tau^2 + O(\tau^3)\right) \\ &\times \left(1 - i\hat{V}\tau/2 - \frac{1}{2}\hat{V}^2\tau^2/4 + O(\tau^3)\right), \end{aligned} \tag{3}$$

$$\begin{aligned} e^{-iV(\hat{x})\tau/2}e^{-i\hat{p}^2\tau/(2m)}e^{-iV(\hat{x})\tau/2} = &\left(1 - i\hat{V}\tau/2 - \frac{1}{2}\hat{V}^2\tau^2/4 - i\frac{\hat{p}^2}{2m}\tau - \hat{V}\frac{\hat{p}^2}{2m}\tau^2/2 - \frac{1}{2}\frac{\hat{p}^4}{4m^2}\tau^2 + O(\tau^3)\right) \\ &\times \left(1 - i\hat{V}\tau/2 - \frac{1}{2}\hat{V}^2\tau^2/4 + O(\tau^3)\right), \end{aligned} \tag{4}$$

$$\begin{aligned} e^{-iV(\hat{x})\tau/2}e^{-i\hat{p}^2\tau/(2m)}e^{-iV(\hat{x})\tau/2} = &1 - i\hat{V}\tau/2 - \frac{1}{2}\hat{V}^2\tau^2/4 - i\frac{\hat{p}^2}{2m}\tau - \hat{V}\frac{\hat{p}^2}{2m}\tau^2/2 - \frac{1}{2}\frac{\hat{p}^4}{4m^2}\tau^2 \\ &- i\hat{V}\tau/2 - \hat{V}^2\tau^2/4 - \frac{\hat{p}^2}{2m}\hat{V}\tau^2/2 - \frac{1}{2}\hat{V}^2\tau^2/4 + O(\tau^3), \end{aligned} \tag{5}$$

$$\begin{aligned} e^{-iV(\hat{x})\tau/2}e^{-i\hat{p}^2\tau/(2m)}e^{-iV(\hat{x})\tau/2} = &1 - i\hat{V}\tau - i\frac{\hat{p}^2}{2m}\tau - \frac{1}{2}\hat{V}^2\tau^2 - \hat{V}\frac{\hat{p}^2}{2m}\tau^2/2 - \frac{1}{2}\frac{\hat{p}^4}{4m^2}\tau^2 \\ &- \frac{\hat{p}^2}{2m}\hat{V}\tau^2/2 + O(\tau^3). \end{aligned} \tag{6}$$

Note that the r.h.s. of Eq. (6) is identical to the r.h.s. of E. (2), completing the proof that the Trotter expansion, introduced by Eq. (18), is accurate to second order in τ.

Problem 6:

In order to visualize the output of this program, cut the source code attached below save it in a file named Problem6.f, compile it by typing

```
 gfortran Problem6.f -o Problem6
```

run it by typing

```
./Problem6
```

and visualize the output as follows: type

```
gnuplot
```

then type

```
set dat sty line
```

then type

```
set yrange[0:6]
```

and the type

```
plot ``arch.0002''
```

That will show the numerical propagation after one step with $\tau = 0.1$. In order to visualize the analytic result on top of the numerical propagation, type

```
replot ``arch.0002'' u 1:3
```

To exit, type

```
quit
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P6/Problem6.f),

```
      PROGRAM Problem6
c
c     1-D wave packet propagation
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump
      INTEGER istep,nstep
      REAL dt,xc,pc
      COMPLEX vprop,tprop,x_mean,p_mean
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
      DIMENSION x_mean(NN),p_mean(NN)
      COMMON /class/ xc,pc
c
      CALL ReadParam(nstep,ndump,dt)
      call Initialize()
      CALL SetKinProp(dt,tprop)
      CALL SetPotProp(dt,vprop)
      DO istep=1,nstep+1
         IF(mod(istep-1,10).EQ.0)
     1        PRINT *, "Step=", istep-1,", Final step=", nstep
         IF(istep.GE.1) CALL PROPAGATE(vprop,tprop)
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
         END IF
      END DO
 22   FORMAT(6(e13.6,2x))
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     mass (rmass), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,rmass,xk,dt
      common /packet/  rmass,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-10.0
      xmax= 10.0
      dt=0.1
      rmass=1.0
      xk=-2.5
      pk=1.0
      nstep=1
      ndump=1
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk
      COMPLEX chi0,chi,EYE,CRV
      REAL xc,pc,omega,xk2,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha,alpha2
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)
      COMMON /class/ xc,pc

      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      xc=kk
      pc=pk
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=rmass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi0(kk,1)=chi(kk,1)
      end do
c
c     Hamiltonian Matrix CRV
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         WRITE(11,22) x,real(CRV(1,1))
      END DO
 22   FORMAT(6(e13.6,2x))
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1
      COMPLEX CRV
      PARAMETER(NN=1)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CRV(1,1)=VPOT1
c
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      implicit none
      REAL V,x,rmass,xk,pk,rk,omega
      common /packet/ rmass,xk,pk
      omega=1.0
      rk=rmass*omega**2
      V=0.5*rk*x*x
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
```

```
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts
      REAL xsc,xmin,xmax,propfacx,rmass,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=9,nptx=2**npts)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
c
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./rmass*(2.*pi)**2
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         xsc=0.
         if(nx.ne.0) xsc=real(nx)/alenx
         tprop(kx)=exp(eye*(propfacx*xsc**2))
       end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,nptx,npts
      REAL xmin,xmax,dx,dt,x,VPOT
      COMPLEX vprop,eye
      parameter(npts=9,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL VA(VPOT,x)
         vprop(ii,1,1)=exp(-eye*0.5*dt*VPOT)/sqrt(nptx*1.0)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=1)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      FUNCTION Psia(x,istep,dt)
c
c     Analytic wave-packet <x|Psia(istep)> obtained by applying the
c     harmonic propagator to the initial state,
c     <x'|Psi(0)> = (alpha/pi)**.25*exp(-alpha/2*(x'-xk)**2+eye*pk*(x'-xk)),
c     where the propagator is
c     <x|exp(-beta H)|x'> = A exp(-rgamma*(x**2+x'**2)+rgammap*x*x'), with
c     A = sqrt(m*omega/(pi*(exp(beta*omega)-exp(-beta*omega)))), beta = i*t,
c     rgamma = 0.5*m*omega*cosh(beta*omega)/sinh(beta*omega) and
c     rgammap = m*omega/sinh(beta*omega).
c
      IMPLICIT NONE
      INTEGER istep
      REAL pk,rmass,xk,dt,x,t,omega,pi,alpha
      COMPLEX eye,Psia,beta,A,rgamma,rgammap,c0,c1,c2
      common /packet/  rmass,xk,pk
      eye=(0.0,1.0)
      omega=1.0
      alpha = omega*rmass
      pi=acos(-1.0)
      beta = eye*dt*istep
      IF(abs(beta).EQ.0) beta = eye*1.0E-7
      A = sqrt(rmass*omega/(pi*(exp(beta*omega)-exp(-beta*omega))))
      rgamma=0.5*rmass*omega*(exp(beta*omega)+exp(-beta*omega))
     1     /(exp(beta*omega)-exp(-beta*omega))
      rgammap=2.*rmass*omega/(exp(beta*omega)-exp(-beta*omega))
      c0=-eye*pk*xk-alpha/2.*xk**2
      c1=rgammap*x+alpha*xk+eye*pk
      c2=rgamma+alpha/2.
c
      Psia = A*(alpha/pi)**.25*sqrt(pi/c2)*
     1     exp(-rgamma*x**2)*exp(c0+c1**2/(4.0*c2))
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj
      COMPLEX chi,CRV,energy,psi,Psia
      character*9 B
      REAL V,x1,c1,c2,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN),energy(NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
c
      CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         c1=chi(kk,1)*conjg(chi(kk,1))
         c1a=Psia(x,je2,dt)*conjg(Psia(x,je2,dt))
         write(1,33) x,sqrt(c1)+real(energy(1))
     1        ,sqrt(c1a)+real(energy(1))
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         write(1,33) x
     1        ,real(chi(kk,1))+real(energy(1))
     1        ,real(Psia(x,je2,dt))+real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         write(1,33) x,CRV(1,1)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=9,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
```

```
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*rmass)*chic(kx)/nptx
         end do
         CALL fourn(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop)
c
c     Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,NN,ii,nptx,npts
      COMPLEX chi,vprop,chin1,chin2,tprop
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION chin1(nptx),chin2(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=0.0
         DO j=1,NN
            chin1(i)=chin1(i)+vprop(i,1,j)*chi(i,j)
         END DO
      END DO
c
c     Fourier Transform wave-packet to the momentum representation
c
      CALL fourn(chin1,nptx,1,-1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=tprop(i)*chin1(i)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL fourn(chin1,nptx,1,1)
```

```
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         DO j=1,NN
            chi(i,j)=vprop(i,j,1)*chin1(i)
         END DO
      END DO
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutine for FFT from Numerical Recipes
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
```

```
                    DATA(K2)=DATA(K1)-TEMPR
                    DATA(K2+1)=DATA(K1+1)-TEMPI
                    DATA(K1)=DATA(K1)+TEMPR
                    DATA(K1+1)=DATA(K1+1)+TEMPI
 15              CONTINUE
 16           CONTINUE
              WTEMP=WR
              WR=WR*WPR-WI*WPI+WR
              WI=WI*WPR+WTEMP*WPI+WI
 17        CONTINUE
           IFP1=IFP2
           GO TO 2
        ENDIF
        NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 7:

In order to visualize the output of this program, cut the source code attached below, compile it by typing

```
 gfortran Problem7.f -o Problem7
```

run it by typing

```
./Problem7
```

Visualize the output of time dependent expectation values as compared to classical trajectories as follows: type

```
gnuplot
```

then type

```
set dat sty line
```

then type

```
plot ``traj.0000''
```

That will show the numerical computation of the expectation value $< \Psi_t|\hat{x}|\Psi_t >$ as a function of time. In order to visualize the classical result on top of the quantum mechanical expectation value, type

```
replot ``traj.0000'' u 1:4
```

In order to visualize the output of $< \Psi_t|\hat{p}|\Psi_t >$ as a function of time, type

```
plot ``traj.0000'' u 1:3
```

and to visualize the classical result on top of the quantum mechanical expectation value, type

```
replot ``traj.0000'' u 1:5
```

The plot of $< \Psi_t|\hat{p}|\Psi_t >$ vs. $< \Psi_t|\hat{x}|\Psi_t >$ can be obtained by typing

```
 plot ``traj.0000'' u 3:2
```

, and the corresponding classical results $p(t)$ vs. $x(t)$

```
 plot ``traj.0000'' u 5:4
```

To exit, type

```
quit
```

The snapshots of the time-dependent wave-packet can be visualized as a movie by typing

```
 gnuplot<pp_7
```

where the file named

```
 pp_7
```

has the following lines:

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P7/pp_7)

```
set yrange[0:6]
set xrange[-10:10]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
plot "arch.0020" u 1:2 lw 3
pause .1
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
```

```
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
plot "arch.0052" u 1:2 lw 3
pause .1
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
```

```
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
plot "arch.0084" u 1:2 lw 3
pause .1
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
```

```
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P7/Problem7.f)

```
      PROGRAM Problem7
c
c     1-D wave packet propagation and Velocity-Verlet propagation
c     on a Harmonic potential energy surface
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump
      INTEGER istep,nstep,jj
      REAL dt,xc,pc
      COMPLEX vprop,tprop,x_mean,p_mean
      character*9 Bfile
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
      DIMENSION x_mean(NN),p_mean(NN)
      COMMON /class/ xc,pc
c
      jj=0
      write(Bfile, '(A,i4.4)') 'traj.', jj
      OPEN(10,FILE=Bfile)
      CALL ReadParam(nstep,ndump,dt)
      call Initialize()
      CALL SetKinProp(dt,tprop)
      CALL SetPotProp(dt,vprop)
      DO istep=1,nstep+1
         IF(mod(istep-1,10).EQ.0)
     1        PRINT *, "Step=", istep-1,", Final step=", nstep
         IF(istep.GE.1) CALL PROPAGATE(vprop,tprop)
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
            CALL XM(x_mean)
            CALL PM(p_mean)
            CALL VV(dt)
            WRITE(10,22) (istep-1.)*dt
     1           ,real(x_mean(1)),real(p_mean(1)),xc,pc
         END IF
      END DO
      CLOSE(10)
 22   FORMAT(6(e13.6,2x))
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     rmass (rmass), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,rmass,xk,dt
      common /packet/  rmass,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-10.0
      xmax= 10.0
      dt=0.1
      rmass=1.0
      xk=-2.5
      pk=0.0
      nstep=100
      ndump=1
c
      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VV(dt)
c
c     Velocity Verlet Algorithm J. Chem. Phys. 76, 637 (1982)
c
      IMPLICIT NONE
      REAL v,dx,dt,xc,pc,rmass,xk,pk,acc,xt,VPOT1,VPOT2,F
      COMMON /class/ xc,pc
      common /packet/  rmass,xk,pk
c
c     Compute Force
c
      dx=0.01
      xt=xc+dx
      CALL VA(VPOT1,xt)
      xt=xc-dx
      CALL VA(VPOT2,xt)
      F=-(VPOT1-VPOT2)/(2.0*dx)
      v=pc/rmass
c
c     Advance momenta half a step
c
      pc=pc+0.5*F*dt
c
c     Advance coordinates a step
c
      xc=xc+v*dt+0.5*dt**2*F/rmass
c
c     Compute Force
c
      dx=0.01
      xt=xc+dx
      CALL VA(VPOT1,xt)
      xt=xc-dx
      CALL VA(VPOT2,xt)
      F=-(VPOT1-VPOT2)/(2.0*dx)
c
c     Advance momenta half a step
c
      pc=pc+0.5*F*dt
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk
      COMPLEX chi0,chi,EYE,CRV
      REAL xc,pc,omega,xk2,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha,alpha2
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)
      COMMON /class/ xc,pc

      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      xc=xk
      pc=pk
```

```
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=rmass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi0(kk,1)=chi(kk,1)
      end do
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1
      COMPLEX CRV
      PARAMETER(NN=1)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CRV(1,1)=VPOT1
c
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      implicit none
      REAL V,x,rmass,xk,pk,rk,omega
      common /packet/ rmass,xk,pk
      omega=1.0
      rk=rmass*omega**2
      V=0.5*rk*x*x
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts
      REAL xsc,xmin,xmax,propfacx,rmass,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=9,nptx=2**npts)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
c
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./rmass*(2.*pi)**2
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
```

```
              nx=kx-1-nptx
           end if
           xsc=0.
           if(nx.ne.0) xsc=real(nx)/alenx
           tprop(kx)=exp(eye*(propfacx*xsc**2))
         end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,nptx,npts
      REAL xmin,xmax,dx,dt,x,VPOT
      COMPLEX vprop,eye
      parameter(npts=9,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL VA(VPOT,x)
         vprop(ii,1,1)=exp(-eye*0.5*dt*VPOT)/sqrt(nptx*1.0)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=1)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      FUNCTION Psia(x,istep,dt)
c
c     Analytic wave-packet <x|Psia(istep)> obtained by applying the
c     harmonic propagator to the initial state,
c     <x'|Psi(0)> = (alpha/pi)**.25*exp(-alpha/2*(x'-xk)**2+eye*pk*(x'-xk)),
c     where the propagator is
c     <x|exp(-beta H)|x'> = A exp(-rgamma*(x**2+x'**2)+rgammap*x*x'), with
c     A = sqrt(m*omega/(pi*(exp(beta*omega)-exp(-beta*omega)))), beta = i*t,
c     rgamma = 0.5*m*omega*cosh(beta*omega)/sinh(beta*omega) and
c     rgammap = m*omega/sinh(beta*omega).
c
      IMPLICIT NONE
      INTEGER istep
      REAL pk,rmass,xk,dt,x,t,omega,pi,alpha
      COMPLEX eye,Psia,beta,A,rgamma,rgammap,c0,c1,c2
      common /packet/  rmass,xk,pk
      eye=(0.0,1.0)
      omega=1.0
      alpha = omega*rmass
      pi=acos(-1.0)
      beta = eye*dt*istep
      IF(abs(beta).EQ.0) beta = eye*1.0E-7
      A = sqrt(rmass*omega/(pi*(exp(beta*omega)-exp(-beta*omega))))
      rgamma=0.5*rmass*omega*(exp(beta*omega)+exp(-beta*omega))
     1     /(exp(beta*omega)-exp(-beta*omega))
      rgammap=2.*rmass*omega/(exp(beta*omega)-exp(-beta*omega))
      c0=-eye*pk*xk-alpha/2.*xk**2
      c1=rgammap*x+alpha*xk+eye*pk
      c2=rgamma+alpha/2.
c
      Psia = A*(alpha/pi)**.25*sqrt(pi/c2)*
     1     exp(-rgamma*x**2)*exp(c0+c1**2/(4.0*c2))
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj
      COMPLEX chi,CRV,energy,psi,Psia
      character*9 B
      REAL V,x1,c1,c2,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN),energy(NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
c
      CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         c1=chi(kk,1)*conjg(chi(kk,1))
         c1a=Psia(x,je2,dt)*conjg(Psia(x,je2,dt))
         write(1,33) x,sqrt(c1)+real(energy(1))
     1        ,sqrt(c1a)+real(energy(1))
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         write(1,33) x
     1        ,real(chi(kk,1))+real(energy(1))
     1        ,real(Psia(x,je2,dt))+real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         write(1,33) x,CRV(1,1)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE XM(RV)
c
c     Expectation Value of the Position
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*x*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=9,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*rmass)*chic(kx)/nptx
         end do
         CALL four(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine PM(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=9,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p*chic(kx)/nptx
         end do
         CALL fourn(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop)
c
c     Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,NN,ii,nptx,npts
      COMPLEX chi,vprop,chin1,chin2,tprop
      PARAMETER(npts=9,nptx=2**npts,NN=1)
      DIMENSION chin1(nptx),chin2(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=0.0
         DO j=1,NN
            chin1(i)=chin1(i)+vprop(i,1,j)*chi(i,j)
         END DO
      END DO
c
c     Fourier Transform wave-packet to the momentum representation
c
      CALL fourn(chin1,nptx,1,-1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=tprop(i)*chin1(i)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL fourn(chin1,nptx,1,1)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         DO j=1,NN
            chi(i,j)=vprop(i,j,1)*chin1(i)
         END DO
      END DO
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutine for FFT from Numerical Recipes
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
11    CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
12                CONTINUE
13             CONTINUE
            ENDIF
            IBIT=IP2/2
1           IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
14       CONTINUE
         IFP1=IP1
2        IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
15                CONTINUE
16             CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
17          CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
18    CONTINUE
```

```
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 8:

The output of this program is analogous to Problem 6 but for a Morse potential. Cut the source code attached below, save it in a file named Problem8.f, compile it by typing

```
 gfortran Problem8.f -o Problem8
```

run it by typing

```
./Problem8
```

Visualize the output of the time dependent expectation values as compared to classical trajectories as follows: type

```
gnuplot
```

then type

```
set dat sty line
```

then type

```
plot ``traj.0000''
```

That will show the numerical computation of the expectation value $< \Psi_t|\hat{x}|\Psi_t >$ as a function of time. In order to visualize the classical result on top of the quantum mechanical expectation value, type

```
replot ``traj.0000'' u 1:4
```

In order to visualize the output of $< \Psi_t|\hat{p}|\Psi_t >$ as a function of time, type

```
plot ``traj.0000'' u 1:3
```

and to visualize the classical result on top of the quantum mechanical expectation value, type

```
replot ``traj.0000'' u 1:5
```

The plot of $< \Psi_t|\hat{p}|\Psi_t >$ vs. $< \Psi_t|\hat{x}|\Psi_t >$ can be obtained by typing

```
 plot ``traj.0000'' u 3:2
```

and the corresponding classical results $p(t)$ vs. $x(t)$

```
 plot ``traj.0000'' u 5:4
```

To exit, type

```
quit
```

The snapshots of the time-dependent wave-packet can be visualized as a movie by typing

```
 gnuplot<pp_8
```

where the file named

```
 pp_8
```

has the following lines:

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P8/pp_8)

```
set yrange[0:9]
set xrange[-5:25]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
plot "arch.0020" u 1:2 lw 3
pause .1
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
```

```
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
plot "arch.0052" u 1:2 lw 3
pause .1
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
```

```
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
plot "arch.0084" u 1:2 lw 3
pause .1
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
```

```
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P8/Problem8.f)

```
      PROGRAM Problem8
c
c     1-D wave packet propagation and Velocity-Verlet propagation
c     on a Morse potential energy surface
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump
      INTEGER istep,nstep,jj
      REAL dt,xc,pc
      COMPLEX vprop,tprop,x_mean,p_mean
      character*9 Bfile
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
      DIMENSION x_mean(NN),p_mean(NN)
      COMMON /class/ xc,pc
c     xo
      jj=0
      write(Bfile, '(A,i4.4)') 'traj.', jj
      OPEN(10,FILE=Bfile)
      CALL ReadParam(nstep,ndump,dt)
      call Initialize()
      CALL SetKinProp(dt,tprop)
      CALL SetPotProp(dt,vprop)
      DO istep=1,nstep+1
         IF(mod(istep-1,10).EQ.0)
     1        PRINT *, "Step=", istep-1,", Final step=", nstep
         IF(istep.GE.1) CALL PROPAGATE(vprop,tprop)
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
            CALL XM(x_mean)
            CALL PM(p_mean)
            CALL VV(dt)
            WRITE(10,22) (istep-1.)*dt
     1           ,real(x_mean(1)),real(p_mean(1)),xc,pc
         END IF
      END DO
      CLOSE(10)
 22   FORMAT(6(e13.6,2x))
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     rmass (rmass), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,rmass,xk,dt
      common /packet/  rmass,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-5.0
      xmax=25.0
      dt=0.2
      rmass=1.0
      xk=-.5
      pk=0.0
      nstep=100
      ndump=1
c
      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VV(dt)
c
c     Velocity Verlet Algorithm J. Chem. Phys. 76, 637 (1982)
c
      IMPLICIT NONE
      REAL v,dx,dt,xc,pc,rmass,xk,pk,acc,xt,VPOT1,VPOT2,F
      COMMON /class/ xc,pc
      common /packet/  rmass,xk,pk
c
c     Compute Force
c
      dx=0.01
      xt=xc+dx
      CALL VA(VPOT1,xt)
      xt=xc-dx
      CALL VA(VPOT2,xt)
      F=-(VPOT1-VPOT2)/(2.0*dx)
      v=pc/rmass
c
c     Advance momenta half a step
c
      pc=pc+0.5*F*dt
c
c     Advance coordinates a step
c
      xc=xc+v*dt+0.5*dt**2*F/rmass
c
c     Compute Force
c
      dx=0.01
      xt=xc+dx
      CALL VA(VPOT1,xt)
      xt=xc-dx
      CALL VA(VPOT2,xt)
      F=-(VPOT1-VPOT2)/(2.0*dx)
c
c     Advance momenta half a step
c
      pc=pc+0.5*F*dt
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk
      COMPLEX chi0,chi,EYE,CRV
      REAL xc,pc,omega,xk2,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha,alpha2
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)
      COMMON /class/ xc,pc

      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      xc=xk
      pc=pk
```

```
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=rmass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi0(kk,1)=chi(kk,1)
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1
      COMPLEX CRV
      PARAMETER(NN=1)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CRV(1,1)=VPOT1
c
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Morse Potential [Phys. Rev. (1929) 34:57]
c
      implicit none
      REAL V,x,rmass,xk,pk,rk,omega,De,xeq,a
      common /packet/ rmass,xk,pk
      xeq=0.0
      omega=1.0
      De=8.0
      rk=rmass*omega**2
      a=sqrt(rk/(2.0*De))
      V=De*(1.0-exp(-a*(x-xeq)))**2
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts
      REAL xsc,xmin,xmax,propfacx,rmass,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=10,nptx=2**npts)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
c
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./rmass*(2.*pi)**2
      do kx=1,nptx
```

```
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         xsc=0.
         if(nx.ne.0) xsc=real(nx)/alenx
         tprop(kx)=exp(eye*(propfacx*xsc**2))
       end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,nptx,npts
      REAL xmin,xmax,dx,dt,x,VPOT
      COMPLEX vprop,eye
      parameter(npts=10,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL VA(VPOT,x)
         vprop(ii,1,1)=exp(-eye*0.5*dt*VPOT)/sqrt(nptx*1.0)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=1)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj
      COMPLEX chi,CRV,energy,psi,Psia
      character*9 B
      REAL V,x1,c1,c2,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
      COMMON /ENER/ energy(NN)
```

```
c
      IF(je2.EQ.1) CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         c1=chi(kk,1)*conjg(chi(kk,1))
         write(1,33) x,sqrt(c1)+real(energy(1))
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         write(1,33) x,real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         write(1,33) x,CRV(1,1)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE XM(RV)
c
c     Expectation Value of the Position
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*x*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
```

```
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=10,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*rmass)*chic(kx)/nptx
         end do
         CALL four(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

```
      subroutine PM(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=10,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p*chic(kx)/nptx
         end do
         CALL fourn(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop)
c
c     Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,NN,ii,nptx,npts
      COMPLEX chi,vprop,chin1,chin2,tprop
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION chin1(nptx),chin2(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=0.0
         DO j=1,NN
            chin1(i)=chin1(i)+vprop(i,1,j)*chi(i,j)
         END DO
      END DO
c
```

```
c     Fourier Transform wave-packet to the momentum representation
c
      CALL fourn(chin1,nptx,1,-1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=tprop(i)*chin1(i)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL fourn(chin1,nptx,1,1)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         DO j=1,NN
            chi(i,j)=vprop(i,j,1)*chin1(i)
         END DO
      END DO
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutine for FFT from Numerical Recipes
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
```

```
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 9:

The output of this program can be generated and visualized as follows. Cut the source code attached below, save it in a file named Problem9.f, compile it by typing

```
 gfortran Problem9.f -o Problem9
```

run it by typing

```
./Problem9
```

The snapshots of the time-dependent wave-packet can be visualized as a movie by typing

```
 gnuplot<pp_9
```

where the file named

```
 pp_9
```

has the following lines:

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P9/pp_9)

```
set yrange[0:4]
set xrange[-10:10]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
plot "arch.0020" u 1:2 lw 3
```

```
pause .1
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
plot "arch.0052" u 1:2 lw 3
```

```
pause .1
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
plot "arch.0084" u 1:2 lw 3
```

```
pause .1
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P9/Problem9.f)

```
      PROGRAM Problem9
c
c     1-D wave packet propagation of tunneling through a barrier
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump
      INTEGER istep,nstep,jj
      REAL dt,xc,pc
      COMPLEX vprop,tprop,x_mean,p_mean
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
      DIMENSION x_mean(NN),p_mean(NN)
      COMMON /class/ xc,pc
c
      CALL ReadParam(nstep,ndump,dt)
      call Initialize()
      CALL SetKinProp(dt,tprop)
      CALL SetPotProp(dt,vprop)
      DO istep=1,nstep+1
         IF(mod(istep-1,10).EQ.0)
     1        PRINT *, "Step=", istep-1,", Final step=", nstep
         IF(istep.GE.1) CALL PROPAGATE(vprop,tprop)
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
         END IF
      END DO
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     rmass (rmass), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,rmass,xk,dt
      common /packet/  rmass,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-13.0
      xmax=13.0
      dt=0.1
      rmass=1.0
      xk=-4.5
      pk=1.
      nstep=100
      ndump=1
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk
      COMPLEX chi0,chi,EYE,CRV
      REAL xc,pc,omega,xk2,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha,alpha2
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)
      COMMON /class/ xc,pc

      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      xc=xk
      pc=pk
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=rmass*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi0(kk,1)=chi(kk,1)
      end do
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1
      COMPLEX CRV
      PARAMETER(NN=1)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CRV(1,1)=VPOT1
c
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Barrier
c
      implicit none
      REAL V,x,rmass,xk,pk,rk,omega
      common /packet/ rmass,xk,pk
      V=0.0
      IF(abs(x).LE.(.5)) V=3.
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts
      REAL xsc,xmin,xmax,propfacx,rmass,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=10,nptx=2**npts)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
c
```

```
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./rmass*(2.*pi)**2
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         xsc=0.
         if(nx.ne.0) xsc=real(nx)/alenx
         tprop(kx)=exp(eye*(propfacx*xsc**2))
       end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,nptx,npts
      REAL xmin,xmax,dx,dt,x,VPOT,xa
      COMPLEX vprop,eye
      parameter(npts=10,nptx=2**npts,NN=1,xa=10.)
      DIMENSION vprop(nptx,NN,NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL VA(VPOT,x)
         vprop(ii,1,1)=exp(-eye*0.5*dt*VPOT)/sqrt(nptx*1.0)
         IF(abs(x).GT.(xa))
     1        vprop(ii,1,1)=vprop(ii,1,1)*exp(-(abs(x)-xa)**4)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=1)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj
      COMPLEX chi,CRV,energy,psi,Psia
      character*9 B
      REAL V,x1,c1,c2,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION CRV(NN,NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
      COMMON /ENER/ energy(NN)
c
      IF(je2.EQ.1) CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         c1=chi(kk,1)*conjg(chi(kk,1))
         write(1,33) x,sqrt(c1)+real(energy(1))
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         write(1,33) x
     1        ,real(chi(kk,1))+real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         write(1,33) x,CRV(1,1)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,rmass,xk,pk,x,alpha
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/rmass,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,rmass,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=10,nptx=2**npts,NN=1)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  rmass,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four1(chic,nptx,1,-1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*rmass)*chic(kx)/nptx
         end do
         CALL four1(chic,nptx,1,1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop)
c
c     Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,NN,ii,nptx,npts
      COMPLEX chi,vprop,chin1,chin2,tprop
      PARAMETER(npts=10,nptx=2**npts,NN=1)
      DIMENSION chin1(nptx),chin2(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=0.0
         DO j=1,NN
```

```
            chin1(i)=chin1(i)+vprop(i,1,j)*chi(i,j)
         END DO
      END DO
c
c     Fourier Transform wave-packet to the momentum representation
c
      CALL fourn(chin1,nptx,1,-1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO i=1,nptx
         chin1(i)=tprop(i)*chin1(i)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL fourn(chin1,nptx,1,1)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO i=1,nptx
         DO j=1,NN
            chi(i,j)=vprop(i,j,1)*chin1(i)
         END DO
      END DO
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutine for FFT from Numerical Recipes
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 10:

In order to derive Eq. (28) we need to prove the following equation:

$$e^{-iV_0\tau}e^{-iV_c2\tau}e^{-iV_0\tau} = \begin{pmatrix} e^{-iV_1(\mathbf{x})2\tau}\cos(2V_c(\mathbf{x})\tau) & -i\,\sin(2V_c(\mathbf{x})\tau)\; e^{-i(\hat{V}_1(\mathbf{x})+\hat{V}_2(\mathbf{x}))\tau} \\ -i\,\sin(2V_c(\mathbf{x})\tau)\; e^{-i(V_1(\mathbf{x})+\hat{V}_2(\mathbf{x}))\tau} & \cos(2V_c(\mathbf{x})\tau)\; e^{-iV_2(\mathbf{x})2\tau} \end{pmatrix}. \tag{7}$$

where

$$e^{-iV_0\tau} = e^{-i\begin{pmatrix} V_1(\mathbf{x}) & 0 \\ 0 & V_2(\mathbf{x}) \end{pmatrix}\tau}. \tag{8}$$

Expanding the exponential on the r.h.s. of Eq. (8) gives

$$e^{-i\tau\begin{pmatrix} V_1(\mathbf{x}) & 0 \\ 0 & V_2(\mathbf{x}) \end{pmatrix}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} -i\tau V_1(\mathbf{x}) & 0 \\ 0 & -i\tau V_2(\mathbf{x}) \end{pmatrix} + \begin{pmatrix} \frac{1}{2!}V_1(\mathbf{x})^2(-i\tau)^2 & 0 \\ 0 & \frac{1}{2!}V_2(\mathbf{x})^2(-i\tau)^2 \end{pmatrix} + \ldots \tag{9}$$

Therefore,

$$e^{-i\tau\begin{pmatrix} V_1(\mathbf{x}) & 0 \\ 0 & V_2(\mathbf{x}) \end{pmatrix}} = \begin{pmatrix} e^{-iV_1(\mathbf{x})\tau} & 0 \\ 0 & e^{-iV_2(\mathbf{x})\tau} \end{pmatrix}. \tag{10}$$

In addition, according to Eq. (30),

$$e^{-i\mathbf{V}_c2\tau} = \mathbf{D}^\dagger \begin{pmatrix} e^{iV_c(\mathbf{x})2\tau} & 0 \\ 0 & e^{-iV_c(\mathbf{x})2\tau} \end{pmatrix} \mathbf{D}, \tag{11}$$

with

$$\mathbf{D} = \mathbf{D}^\dagger \equiv \begin{pmatrix} -1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}, \tag{12}$$

Therefore,

$$e^{-iV_0\tau}e^{-iV_c2\tau}e^{-iV_0\tau} = \begin{pmatrix} e^{-iV_1(\mathbf{x})\tau} & 0 \\ 0 & e^{-iV_2(\mathbf{x})\tau} \end{pmatrix} \mathbf{D}^\dagger \begin{pmatrix} e^{iV_c(\mathbf{x})2\tau} & 0 \\ 0 & e^{-iV_c(\mathbf{x})2\tau} \end{pmatrix} \mathbf{D} \begin{pmatrix} e^{-iV_1(\mathbf{x})\tau} & 0 \\ 0 & e^{-iV_2(\mathbf{x})\tau} \end{pmatrix}. \tag{13}$$

The multiplication of the five matrices on the r.h.s. of Eq. (13) gives the matrix on the r.h.s. of Eq.(7).

Problem 11:

According to the definition of the eigenstates of the potential energy matrix, given by Eq. (34),

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} = \begin{pmatrix} E_1 & 0 \\ 0 & E_1 \end{pmatrix} \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix}, \tag{14}$$

and

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} \begin{pmatrix} L_{12} \\ L_{22} \end{pmatrix} = \begin{pmatrix} E_2 & 0 \\ 0 & E_2 \end{pmatrix} \begin{pmatrix} L_{12} \\ L_{22} \end{pmatrix}. \tag{15}$$

Therefore,

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix}, \tag{16}$$

and

$$\begin{pmatrix} L_{11} & L_{21} \\ L_{12} & L_{22} \end{pmatrix} \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} = \begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix}, \tag{17}$$

or

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix}, \begin{pmatrix} L_{11} & L_{21} \\ L_{12} & L_{22} \end{pmatrix}. \tag{18}$$

Therefore, defining

$$\mathbf{L} = \begin{pmatrix} L_{11} & L_{21} \\ L_{12} & L_{22} \end{pmatrix}, \tag{19}$$

gives

$$\begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix} = \mathbf{L}^{-1} \begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix} \mathbf{L}. \tag{20}$$

Exponentiating both sides of Eq. (20), gives

$$e^{-i\tau \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}} = e^{-i\tau \mathbf{L}^{-1} \begin{pmatrix} E_1 & 0 \\ 0 & E_2 \end{pmatrix} \mathbf{L}}. \tag{21}$$

Expanding the r.h.s. of Eq. (21) gives,

$$e^{-i\tau \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \mathbf{L}^{-1} \begin{pmatrix} -i\tau E_1 & 0 \\ 0 & -i\tau E_2 \end{pmatrix} \mathbf{L} + \mathbf{L}^{-1} \begin{pmatrix} \frac{1}{2!} E_1^2 (-i\tau)^2 & 0 \\ 0 & \frac{1}{2!} E_2^2 (-i\tau)^2 \end{pmatrix} \mathbf{L} + ..., \tag{22}$$

since $\mathbf{L}^{-1}\mathbf{L} = 1$. Therefore,

$$e^{-i\tau \begin{pmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{pmatrix}} = \mathbf{L}^{-1} \begin{pmatrix} e^{-iE_1(\mathbf{x})\tau} & 0 \\ 0 & e^{-iE_2(\mathbf{x})\tau} \end{pmatrix} \mathbf{L}. \tag{23}$$

Problem 12:

The output of this program can be generated and visualized as follows. Cut the source code attached below, save it in a file named Problem12.f, compile it by typing

```
 gfortran Problem12.f -o Problem12
```

run it by typing

```
./Problem12
```

That will produce the output for item (a). In order to obtain the output for item (b), modify subroutine Hamil, so that CRV(1,2)=0.0 and CRV(2,1)=0.0, recompile and run.

The snapshots of the time-dependent wave-packet can be visualized as a movie by typing

```
 gnuplot<pp_12
```

where the file named

```
 pp_12
```

has the following lines:

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P12/P12_c/pp_12)

```
set yrange[-2:5]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
```

```
plot "arch.0020" u 1:2 lw 3
pause .1
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
```

```
plot "arch.0052" u 1:2 lw 3
pause .1
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
```

```
plot "arch.0084" u 1:2 lw 3
pause .1
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P12/P12_c/Problem12.f)

```
      PROGRAM Problem12
c
c     1-D nonadiabatic wave-packet propagation
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump
      INTEGER istep,nstep
      REAL dt
      COMPLEX vprop,tprop
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
c
      CALL ReadParam(nstep,ndump,dt)
      call Initialize()
      CALL SetKinProp(dt,tprop)
      CALL SetPotProp(dt,vprop)
      DO istep=1,nstep+1
         IF(mod(istep-1,10).EQ.0)
     1        PRINT *, "Step=", istep-1,", Final step=", nstep
         IF(istep.GE.1) CALL PROPAGATE(vprop,tprop)
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
         END IF
      END DO
 22   FORMAT(6(e13.6,2x))
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=2)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     mass (amassx), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,amassx,xk,dt
      common /packet/  amassx,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-6.0
      xmax=6.0
      dt=0.2
      amassx=1.0
      xk=-2.2
      pk=0.
      nstep=100
      ndump=1
```

```
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize()

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk
      COMPLEX chi0,chi,EYE,CRV
      REAL omega,xk2,xmin,xmax,dx,pi,amassx,xk,pk,x,alpha,alpha2
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)

      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=amassx*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)/sqrt(2.)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi(kk,2)=chi(kk,1)
c
         chi0(kk,1)=chi(kk,1)
         chi0(kk,2)=chi(kk,2)
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj
      COMPLEX chi,CRV,energy,psi,Psia
      character*9 B
      REAL V,x1,c1,c2,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION CRV(NN,NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
      COMMON /ENER/ energy(NN)
c
      IF(je2.EQ.1) CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         c1=chi(kk,1)*conjg(chi(kk,1))
         c2=chi(kk,2)*conjg(chi(kk,2))
         write(1,33) x,sqrt(c1)+real(energy(1))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
         c2=chi(kk,2)*conjg(chi(kk,2))
          write(1,33) x,sqrt(c2)+real(energy(2))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
          write(1,33) x,real(energy(2))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
          write(1,33) x,real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         CALL SCHROC1(CRV,psi,EVALUES)
         write(1,33) x,EVALUES(1)
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         CALL SCHROC1(CRV,psi,EVALUES)
         write(1,33) x,EVALUES(2)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts,NN
      REAL xsc,xmin,xmax,propfacx,amassx,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
c
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./amassx*(2.*pi)**2
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
```

```
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         xsc=0.
         if(nx.ne.0) xsc=real(nx)/alenx
         tprop(kx)=exp(eye*(propfacx*xsc**2))
       end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,kk,jj,nptx,i,j,k,npts
      REAL xmin,xmax,dx,dt,EVALUES,x
      COMPLEX vp,vprop,eye,dummy,psi,CRV
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION vprop(nptx,NN,NN),psi(NN,NN),CRV(NN,NN)
      DIMENSION vp(NN,NN),dummy(NN,NN),EVALUES(NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL HAMIL(CRV,x)
         CALL SCHROC1(CRV,psi,EVALUES)
         vp(1,1)=exp(-eye*0.5*dt*EVALUES(1))
         vp(1,2)=0.0
         vp(2,1)=0.0
         vp(2,2)=exp(-eye*0.5*dt*EVALUES(2))
         do i=1,2
            do j=1,2
               dummy(i,j)=0.
               do k=1,2
                  dummy(i,j)=dummy(i,j)+vp(i,k)*psi(j,k)
               end do
            end do
         end do
         do i=1,2
            do j=1,2
               vp(i,j)=0.
               do k=1,2
                  vp(i,j)=vp(i,j)+psi(i,k)*dummy(k,j)
               end do
            end do
         end do
         do i=1,2
            do j=1,2
               kk=ii
               vprop(kk,i,j)=vp(i,j)/sqrt(1.0*nptx)
            end do
         end do
      end do
c
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop)
c
```

```
c     Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,kk,NN,in,ii,nptx,npts
      COMPLEX chi,vprop,chin1,chin2,tprop
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION chin1(nptx),chin2(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO ii=1,nptx
         in=ii
         chin1(in)=0.0
         chin2(in)=0.0
         DO j=1,NN
            kk=ii
            chin1(in)=chin1(in)+vprop(kk,1,j)*chi(kk,j)
            chin2(in)=chin2(in)+vprop(kk,2,j)*chi(kk,j)
         END DO
      END DO
c
c     Fourier Transform wave-packet to the momentum representation
c
      CALL four(chin1,nptx,1,1)
      CALL four(chin2,nptx,1,1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO ii=1,nptx
         in=ii
         kk=ii
         chin1(in)=tprop(kk)*chin1(in)
         chin2(in)=tprop(kk)*chin2(in)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL four(chin1,nptx,1,-1)
      CALL four(chin2,nptx,1,-1)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO ii=1,nptx
         in=ii
         DO i=1,NN
            kk=ii
            chi(kk,i)=vprop(kk,i,1)*chin1(in)
     1           +vprop(kk,i,2)*chin2(in)
         END DO
      END DO
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1,VPOT2
      COMPLEX CRV
      PARAMETER(NN=2)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CALL VB(VPOT2,x)
      CRV(1,1)=VPOT1
      CRV(2,2)=VPOT2
      CRV(1,2)=0.3    ! comment this line for item (b)
      CRV(2,1)=0.3    ! comment this line for item (b)
c      CRV(1,2)=0.3   ! uncomment this line for item (b)
c      CRV(2,1)=0.3   ! uncomment this line for item (b)
c
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      implicit none
      REAL V,x,amassx,xk,pk,rk,omega
      common /packet/ amassx,xk,pk
      omega=1.0
      rk=amassx*omega**2
      V=0.5*rk*x*x
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VB(V,x1)
c
c     Potential Energy Surface: Double-Well Potential, tunneling dynamics
c
      implicit none
      REAL V,x1,x
      x=x1
      V=-0.5*x**2+1.0/(16.0*1.3544)*x**4
      RETURN
      END

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,amassx,xk,pk,x,alpha
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/amassx,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            IF(j.EQ.2) CALL VB(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,amassx,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
      COMMON / wfunc/ chi(nptx,2)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL four(chic,nptx,1,1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*amassx)*chic(kx)/nptx
         end do
         CALL four(chic,nptx,1,-1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SCHROC1(CRV,EVECT,EVALUES)
c
c     Hamiltonian Matrix Diagonalization
c
c     CRV: HERMITIAN MATRIX (INPUT)
c     EVECT: EIGENVECTORS (OUTPUT)
c     EVALUES: EIGENVALUES (OUTPUT)
c
      INTEGER N,I,J,NP
      REAL EVALUES,CRV2,EVECT2
      COMPLEX CRV,EVECT
      PARAMETER(N=2,NP=2)
      DIMENSION CRV(N,N),EVECT(N,N),EVALUES(N),E(NP)
      DIMENSION CRV2(N,N),EVECT2(N,N)
C
      DO I=1,N
         EVALUES(I)=0.0
         E(I)=0.0
         DO J=1,N
            CRV2(J,I)=CRV(J,I)
         END DO
      END DO
      CALL TRED2(CRV2,N,NP,EVALUES,E)
      CALL TQLI(EVALUES,E,N,NP,CRV2)
      CALL EIGSRT(EVALUES,CRV2,N,NP)
c
      DO I=1,N
         DO J=1,N
            EVECT(J,I)=CRV2(J,I)
         END DO
      END DO
C
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines from Numerical Recipes to compute FFT
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
```

```
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines to compute eigenvalues and eigenvectors
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE TRED2(A,N,NP,D,E)
      IMPLICIT NONE
      INTEGER I,J,K,L,N,NP
      REAL A,D,E,H,SCALE,F,G,HH
      DIMENSION A(NP,NP),D(NP),E(NP)
      IF(N.GT.1)THEN
         DO 18 I=N,2,-1
            L=I-1
            H=0.
            SCALE=0.
            IF(L.GT.1)THEN
               DO 11 K=1,L
                  SCALE=SCALE+ABS(A(I,K))
 11            CONTINUE
               IF(SCALE.EQ.0.)THEN
                  E(I)=A(I,L)
               ELSE
                  DO 12 K=1,L
                     A(I,K)=A(I,K)/SCALE
                     H=H+A(I,K)**2
 12               CONTINUE
                  F=A(I,L)
                  G=-SIGN(SQRT(H),F)
                  E(I)=SCALE*G
                  H=H-F*G
                  A(I,L)=F-G
                  F=0.
                  DO 15 J=1,L
                     A(J,I)=A(I,J)/H
                     G=0.
                     DO 13 K=1,J
                        G=G+A(J,K)*A(I,K)
 13                  CONTINUE
                     IF(L.GT.J)THEN
                        DO 14 K=J+1,L
                           G=G+A(K,J)*A(I,K)
 14                     CONTINUE
                     ENDIF
                     E(J)=G/H
                  F=F+E(J)*A(I,J)
 15             CONTINUE
                HH=F/(H+H)
                DO 17 J=1,L
                  F=A(I,J)
                  G=E(J)-HH*F
                  E(J)=G
                  DO 16 K=1,J
                     A(J,K)=A(J,K)-F*E(K)-G*A(I,K)
 16               CONTINUE
 17             CONTINUE
              ENDIF
           ELSE
              E(I)=A(I,L)
           ENDIF
           D(I)=H
 18      CONTINUE
      ENDIF
      D(1)=0.
      E(1)=0.
      DO 23 I=1,N
         L=I-1
         IF(D(I).NE.0.)THEN
            DO 21 J=1,L
               G=0.
               DO 19 K=1,L
                  G=G+A(I,K)*A(K,J)
 19            CONTINUE
               DO 20 K=1,L
                  A(K,J)=A(K,J)-G*A(K,I)
 20            CONTINUE
 21         CONTINUE
         ENDIF
         D(I)=A(I,I)
         A(I,I)=1.
         IF(L.GE.1)THEN
            DO 22 J=1,L
               A(I,J)=0.
               A(J,I)=0.
 22         CONTINUE
         ENDIF
 23   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE TQLI(D,E,N,NP,Z)
      IMPLICIT NONE
      INTEGER N,NP,I,K,L,M,ITER
      REAL D,E,Z,DD,G,R,S,C,P,F,B
      DIMENSION D(NP),E(NP),Z(NP,NP)
      IF (N.GT.1) THEN
         DO 11 I=2,N
            E(I-1)=E(I)
 11      CONTINUE
         E(N)=0.
         DO 15 L=1,N
            ITER=0
 1          DO 12 M=L,N-1
               DD=ABS(D(M))+ABS(D(M+1))
               IF (ABS(E(M))+DD.EQ.DD) GO TO 2
 12         CONTINUE
            M=N
 2          IF(M.NE.L)THEN
               IF(ITER.EQ.30) PAUSE 'too many iterations!'
               ITER=ITER+1
               G=(D(L+1)-D(L))/(2.*E(L))
               R=SQRT(G**2+1.)
               G=D(M)-D(L)+E(L)/(G+SIGN(R,G))
               S=1.
               C=1.
               P=0.
               DO 14 I=M-1,L,-1
                  F=S*E(I)
                  B=C*E(I)
                  IF(ABS(F).GE.ABS(G))THEN
                     C=G/F
                     R=SQRT(C**2+1.)
                     E(I+1)=F*R
                     S=1./R
                     C=C*S
                  ELSE
                     S=F/G
                     R=SQRT(S**2+1.)
                     E(I+1)=G*R
                     C=1./R
                     S=S*C
                  ENDIF
                  G=D(I+1)-P
                  R=(D(I)-G)*S+2.*C*B
                  P=S*R
                  D(I+1)=G+P
                  G=C*R-B
                  DO 13 K=1,N
                     F=Z(K,I+1)
                     Z(K,I+1)=S*Z(K,I)+C*F
                     Z(K,I)=C*Z(K,I)-S*F
 13              CONTINUE
 14           CONTINUE
               D(L)=D(L)-P
               E(L)=G
               E(M)=0.
               GO TO 1
            ENDIF
 15     CONTINUE
      ENDIF
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE EIGSRT(D,V,N,NP)
      IMPLICIT NONE
      INTEGER N,NP,I,J,K
      REAL D,V,P
      DIMENSION D(NP),V(NP,NP)
      DO 13 I=1,N-1
        K=I
        P=D(I)
        DO 11 J=I+1,N
          IF(D(J).GE.P)THEN
            K=J
            P=D(J)
          ENDIF
11      CONTINUE
        IF(K.NE.I)THEN
          D(K)=D(I)
          D(I)=P
          DO 12 J=1,N
            P=V(J,I)
            V(J,I)=V(J,K)
            V(J,K)=P
12        CONTINUE
        ENDIF
13    CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PIKSRT(N,ARR)
      IMPLICIT NONE
      INTEGER I,J,N
      REAL ARR,A
      DIMENSION ARR(N)
      DO 12 J=2,N
        A=ARR(J)
        DO 11 I=J-1,1,-1
          IF(ARR(I).LE.A)GO TO 10
          ARR(I+1)=ARR(I)
11      CONTINUE
        I=0
10      ARR(I+1)=A
12    CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 12p:

The output of this program can be generated and visualized as follows. Cut the source code attached below, save it in a file named Problem12.f, compile it by typing

```
 gfortran Problem12p.f -o Problem12p
```

run it by typing

```
./Problem12p
```

The snapshots of the time-dependent wave-packet can be visualized as a movie by typing

```
 gnuplot<pp_12
```

where the file named

```
 pp_12
```

has the following lines:

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P12/P12_c/pp_12)

```
set yrange[-2:5]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
plot "arch.0020" u 1:2 lw 3
pause .1
```

```
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
plot "arch.0052" u 1:2 lw 3
pause .1
```

```
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
plot "arch.0084" u 1:2 lw 3
pause .1
```

```
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

Download from (http://ursula.chem.yale.edu/∼batista/classes/summer/P12/P12_c/Problem12p.f)

```
      PROGRAM Problem12p
c
c     SOFT Surface Hopping (SOFT/SH) Method (Chen and Batista 2006)
c     1-D nonadiabatic wave-packet propagation
c
      IMPLICIT NONE
      INTEGER NN,npts,nptx,ndump,kt,ntraj
      INTEGER istep,nstep,iseed
      REAL dt,rn
      COMPLEX vprop,tprop,energy
      PARAMETER(npts=9,nptx=2**npts,NN=2,ntraj=200)
      DIMENSION vprop(nptx,NN,NN),tprop(nptx)
      COMMON /ENER/ energy(NN)
c
      iseed=912371
      call srand(iseed)
      CALL ReadParam(nstep,ndump,dt)

      DO kt=1,ntraj
            IF(mod(kt-1,10).EQ.0)
     1           PRINT *, "Traj = ", kt,", total = ", ntraj
         call Initialize(kt)
         CALL SetKinProp(dt,tprop)
         CALL SetPotProp(dt,vprop)
         CALL energies(energy)
c
         DO istep=1,nstep+1
            IF(istep.GE.1) CALL PROPAGATE(vprop,tprop,dt)
            IF(mod((istep-1),ndump).EQ.0) THEN
               CALL ACCUM(istep,ndump,dt)
            END IF
         END DO
      END DO
c
      DO istep=1,nstep+1
         IF(mod((istep-1),ndump).EQ.0) THEN
            CALL SAVEWF(istep,ndump,dt)
         END IF
      END DO
c
 22   FORMAT(6(e13.6,2x))
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE energies(energy)
      IMPLICIT NONE
      INTEGER j,NN
      COMPLEX energy,RV,RKE
      PARAMETER (NN=2)
      DIMENSION RV(NN),RKE(NN),energy(NN)
      CALL PE(RV)
      CALL KE(RKE)
      DO j=1,NN
         energy(j)=RV(j)+RKE(j)
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ReadParam(nstep,ndump,dt)
c
c     Parameters defining the grid (xmin, xmax), integration time step (dt),
c     mass (amassx), initial position (xk), initial momentum (pk),
c     number of propagation steps (nstep), and how often to save a pic (ndump)
c
      IMPLICIT NONE
      INTEGER ntype,nstep,nrpt,ireport,ndump,nlit
      REAL xmin,xmax,pk,amassx,xk,dt
      common /packet/  amassx,xk,pk
      common /xy/  xmin,xmax
c
      xmin=-6.0
      xmax=6.0
      dt=0.2
      amassx=1.0
      xk=-2.2
      pk=0.
      nstep=100
      ndump=1
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Initialize(kt)

      IMPLICIT NONE
      INTEGER NN,nptx,npts,kk,counter,j,kt,ns
      COMPLEX chi0,chi,EYE,CRV,c1
      REAL omega,xk2,xmin,xmax,dx,pi,amassx,xk,pk,x,alpha,alpha2
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION CRV(NN,NN)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / iwfunc/ chi0(nptx,NN)
      COMMON/cumul/ c1(nptx,200,2),counter(200)
      COMMON / OCCUP/ ns
c
      EYE=(0.0,1.0)
      pi= acos(-1.0)
      omega=1.
      dx=(xmax-xmin)/real(nptx)
      ns = 1
c
c     Wave Packet Initialization: Gaussian centered at xk, with momentum pk
c
      alpha=amassx*omega
      do kk=1,nptx
         x=xmin+kk*dx
         chi(kk,1)=((alpha/pi)**0.25)
     1        *exp(-alpha/2.*(x-xk)**2+EYE*pk*(x-xk))
         chi(kk,2)=chi(kk,1)*.0
c
         chi0(kk,1)=chi(kk,1)
         chi0(kk,2)=chi(kk,2)
      end do
c
      IF (kt.EQ.1) THEN
         DO kk=1,200
            DO j=1,nptx
               c1(j,kk,1)=0.0
               c1(j,kk,2)=0.0
            END DO
            counter(kk)=0
         END DO
      END IF
c
```

```
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SAVEWF(je2,ndump,dt)
c
c     Dump Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj,counter
      COMPLEX chi,CRV,energy,psi,Psia,c1,c2
      character*9 B
      REAL V,x1,c1a,x,xmin,xmax,dx,EVALUES,dt,r1,r2
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION CRV(NN,NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      COMMON/cumul/ c1(nptx,200,2),counter(200)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
      COMMON /ENER/ energy(NN)
c
c      IF(je2.EQ.1) CALL energies(energy)
      jj=je2/ndump
      write(B, '(A,i4.4)') 'arch.', jj
      OPEN(1,FILE=B)
      dx=(xmax-xmin)/real(nptx)
      ncount=(je2-1)/ndump
c
c     Save Wave-packet components
c
      do kk=1,nptx
         x=xmin+kk*dx
         r1=abs(c1(kk,jj,1))
         write(1,33) x,r1/counter(jj)+real(energy(1))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
         r2=abs(c1(kk,jj,2))
         write(1,33) x,r2/counter(jj)+real(energy(2))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
          write(1,33) x,real(energy(2))
      end do
      write(1,33)

      do kk=1,nptx
         x=xmin+kk*dx
          write(1,33) x,real(energy(1))
      end do
      write(1,33)
c
c     Save Adiabatic states
c
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         CALL SCHROC1(CRV,psi,EVALUES)
         write(1,33) x,EVALUES(1)
      end do
      write(1,33)
      do kk=1,nptx
         x=xmin+kk*dx
         CALL HAMIL(CRV,x)
         CALL SCHROC1(CRV,psi,EVALUES)
         write(1,33) x,EVALUES(2)
      end do
      CLOSE(1)
 33   format(6(e13.6,2x))
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE ACCUM(je2,ndump,dt)
c
c     Accumulate Time Evolved Wave packet
c
      IMPLICIT NONE
      INTEGER je2,nptx,npts,kk,NN,ncount,ndump,jj,counter,ns
      COMPLEX chi,CRV,energy,psi,Psia,c1,c2
      character*9 B
      REAL V,x1,c1a,x,xmin,xmax,dx,EVALUES,dt
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION CRV(NN,NN),EVALUES(NN)
      DIMENSION psi(NN,NN)
      COMMON/cumul/ c1(nptx,200,2),counter(200)
      common /xy/  xmin,xmax
      COMMON / wfunc/ chi(nptx,NN)
      COMMON /ENER/ energy(NN)
      COMMON / OCCUP/ ns
c
      jj=je2/ndump
      counter(jj)=counter(jj)+1
c
c     Accumulate Wave-packet components
c
      do kk=1,nptx
         c1(kk,jj,ns)=c1(kk,jj,ns)+chi(kk,ns)
      end do
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetKinProp(dt,tprop)
c
c     Kinetic Energy part of the Trotter Expansion: exp(-i p^2 dt/(2 m))
c
      IMPLICIT NONE
      INTEGER nptx,kx,nx,npts,NN
      REAL xsc,xmin,xmax,propfacx,amassx,xk,pi,alenx,dt,pk
      COMPLEX tprop,eye
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION tprop(nptx)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
c
      eye=(0.,1.)
      pi = acos(-1.0)
      alenx=xmax-xmin
      propfacx=-dt/2./amassx*(2.*pi)**2
      do kx=1,nptx
         if(kx.le.(nptx/2+1)) then
            nx=kx-1
         else
            nx=kx-1-nptx
         end if
         xsc=0.
```

```
            if(nx.ne.0) xsc=real(nx)/alenx
            tprop(kx)=exp(eye*(propfacx*xsc**2))
          end do
c
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine SetPotProp(dt,vprop)
c
c     Potential Energy part of the Trotter Expansion: exp(-i V dt/2)
c
      IMPLICIT NONE
      INTEGER NN,ii,kk,jj,nptx,i,j,k,npts
      REAL xmin,xmax,dx,dt,EVALUES,x,V1,V2,VA
      COMPLEX vp,vprop,eye,dummy,psi,CRV
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION vprop(nptx,NN,NN),psi(NN,NN),CRV(NN,NN)
      DIMENSION vp(NN,NN),dummy(NN,NN),EVALUES(NN)
      common /xy/  xmin,xmax
      eye=(0.,1.)
      dx=(xmax-xmin)/real(nptx)
c
      do ii=1,nptx
         x=xmin+ii*dx
         CALL HAMIL(CRV,x)
         V1=CRV(1,1)
         V2=CRV(2,2)
         VA=0.5*(V1+V2)
         vp(1,1)=exp(-eye*0.5*dt*V1)
         vp(1,2)=exp(-eye*0.5*dt*VA)
         vp(2,1)=exp(-eye*0.5*dt*VA)
         vp(2,2)=exp(-eye*0.5*dt*V2)
         do i=1,2
            do j=1,2
               vprop(ii,i,j)=vp(i,j)/sqrt(1.0*nptx)
            end do
         end do
      end do
c
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PROPAGATE(vprop,tprop,dt)
c
c     SOFT Surface Hopping (SOFT/SH) Method (Chen and Batista 2006)
c
c     SOFT = Split Operator Fourier Transform Propagation Method
c     J. Comput. Phys. 47, 412 (1982); J. Chem. Phys. 78, 301 (1983)
c
      IMPLICIT NONE
      INTEGER i,j,kk,NN,in,ii,nptx,npts,NF,ns,ns_n,ns_o
      COMPLEX chi,vprop,chin,tprop,eye,rc
      REAL cs,si,dt,rn
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION chin(nptx)
      DIMENSION tprop(nptx),vprop(nptx,NN,NN)
      COMMON / wfunc/ chi(nptx,NN)
      COMMON / OCCUP/ ns
c
      eye=(0.0,1.0)
c
c     Stochastic Jump
c
      NF=0
      cs=cos(0.3*dt)
      si=sin(0.3*dt)
      rc=cs+si
      rn=rand()*rc
      IF(rn.LE.cs)  NF=1        ! flag for adiabatic dynamics
      ns_n=ns     ! new surface index
      ns_o=ns     ! old surface index
      IF(NF.EQ.0) THEN
         rc=-eye*rc
         ns_o = ns
         IF(ns_o.EQ.1) THEN
            ns_n = 2
         ELSE
            ns_n = 1
         END IF
         ns=ns_n
      END IF
c
c     Apply potential energy part of the Trotter Expansion
c
      DO ii=1,nptx
         chin(ii)=vprop(ii,ns_n,ns_o)*chi(ii,ns_o)
      END DO
c
c     Fourier Transform wave-packet to the momentum representation
c
      CALL fourn(chin,nptx,1,1)
c
c     Apply kinetic energy part of the Trotter Expansion
c
      DO ii=1,nptx
         chin(ii)=tprop(ii)*chin(ii)
      END DO
c
c     Inverse Fourier Transform wave-packet to the coordinate representation
c
      CALL fourn(chin,nptx,1,-1)
c
c     Apply potential energy part of the Trotter Expansion
c
      DO ii=1,nptx
         chi(ii,ns_n)=rc*vprop(ii,ns_n,ns_o)*chin(ii)
      END DO
c
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE HAMIL(CRV,x)
c
c     Hamiltonian Matrix
c
      IMPLICIT NONE
      INTEGER NN
      REAL x,VPOT1,VPOT2
      COMPLEX CRV
      PARAMETER(NN=2)
      DIMENSION CRV(NN,NN)
c
      CALL VA(VPOT1,x)
      CALL VB(VPOT2,x)
      CRV(1,1)=VPOT1
      CRV(2,2)=VPOT2
      CRV(1,2)=0.3
      CRV(2,1)=0.3
c
```

```
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VA(V,x)
c
c     Potential Energy Surface: Harmonic Oscillator
c
      implicit none
      REAL V,x,amassx,xk,pk,rk,omega
      common /packet/ amassx,xk,pk
      omega=1.0
      rk=amassx*omega**2
      V=0.5*rk*x*x
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE VB(V,x1)
c
c     Potential Energy Surface: Double-Well Potential, tunneling dynamics
c
      implicit none
      REAL V,x1,x
      x=x1
      V=-0.5*x**2+1.0/(16.0*1.3544)*x**4
      RETURN
      END

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PE(RV)
c
c     Expectation Value of the Potential Enegy
c
      IMPLICIT NONE
      INTEGER nptx,npts,kk,NN,j
      COMPLEX chi,EYE,RV
      REAL Vpot,omega,xmin,xmax,dx,pi,amassx,xk,pk,x,alpha
      PARAMETER(npts=9,nptx=2**npts,NN=2)
      DIMENSION RV(NN)
      COMMON / wfunc/ chi(nptx,NN)
      common /xy/  xmin,xmax
      common /packet/amassx,xk,pk

      dx=(xmax-xmin)/real(nptx)
      DO j=1,NN
         RV(j)=0.0
         do kk=1,nptx
            x=xmin+kk*dx
            IF(j.EQ.1) CALL VA(Vpot,x)
            IF(j.EQ.2) CALL VB(Vpot,x)
            RV(j)=RV(j)+chi(kk,j)*Vpot*conjg(chi(kk,j))*dx
         end do
      END DO
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine KE(RKE)
c
c     Expectation value of the kinetic energy
c
      IMPLICIT NONE
      INTEGER NN,kk,nptx,kx,nx,npts,j
      REAL dp,theta,wm,p,xmin,xmax,amassx,xk,pi,alenx,pk,rm,re,ri,dx
      COMPLEX eye,chi,Psip,chic,RKE
      parameter(npts=9,nptx=2**npts,NN=2)
      DIMENSION chic(nptx),RKE(NN)
      common /xy/  xmin,xmax
      common /packet/  amassx,xk,pk
      COMMON / wfunc/ chi(nptx,2)
c
      pi = acos(-1.0)
      dx=(xmax-xmin)/nptx
      dp=2.*pi/(xmax-xmin)
c
      DO j=1,NN
         RKE(j)=0.0
         do kk=1,nptx
            chic(kk)=chi(kk,j)
         end do
         CALL fourn(chic,nptx,1,1)
         do kx=1,nptx
            if(kx.le.(nptx/2+1)) then
               nx=kx-1
            else
               nx=kx-1-nptx
            end if
            p=0.
            if(nx.ne.0) p = real(nx)*dp
            chic(kx)=p**2/(2.0*amassx)*chic(kx)/nptx
         end do
         CALL fourn(chic,nptx,1,-1)
         do kk=1,nptx
            RKE(j)=RKE(j)+conjg(chi(kk,j))*chic(kk)*dx
         end do
      END DO
      return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE SCHROC1(CRV,EVECT,EVALUES)
c
c     Hamiltonian Matrix Diagonalization
c
c     CRV: HERMITIAN MATRIX (INPUT)
c     EVECT: EIGENVECTORS (OUTPUT)
c     EVALUES: EIGENVALUES (OUTPUT)
c
      INTEGER N,I,J,NP
      REAL EVALUES,CRV2,EVECT2
      COMPLEX CRV,EVECT
      PARAMETER(N=2,NP=2)
      DIMENSION CRV(N,N),EVECT(N,N),EVALUES(N),E(NP)
      DIMENSION CRV2(N,N),EVECT2(N,N)
C
      DO I=1,N
         EVALUES(I)=0.0
         E(I)=0.0
         DO J=1,N
            CRV2(J,I)=CRV(J,I)
         END DO
      END DO
      CALL TRED2(CRV2,N,NP,EVALUES,E)
      CALL TQLI(EVALUES,E,N,NP,CRV2)
      CALL EIGSRT(EVALUES,CRV2,N,NP)
c
      DO I=1,N
         DO J=1,N
            EVECT(J,I)=CRV2(J,I)
         END DO
      END DO
```

```
C
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines from Numerical Recipes to compute FFT
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE FOURN(DATA,NN,NDIM,ISIGN)
      REAL*8 WR,WI,WPR,WPI,WTEMP,THETA
      DIMENSION NN(NDIM),DATA(*)
      NTOT=1
      DO 11 IDIM=1,NDIM
         NTOT=NTOT*NN(IDIM)
 11   CONTINUE
      NPREV=1
      DO 18 IDIM=1,NDIM
         N=NN(IDIM)
         NREM=NTOT/(N*NPREV)
         IP1=2*NPREV
         IP2=IP1*N
         IP3=IP2*NREM
         I2REV=1
         DO 14 I2=1,IP2,IP1
            IF(I2.LT.I2REV)THEN
               DO 13 I1=I2,I2+IP1-2,2
                  DO 12 I3=I1,IP3,IP2
                     I3REV=I2REV+I3-I2
                     TEMPR=DATA(I3)
                     TEMPI=DATA(I3+1)
                     DATA(I3)=DATA(I3REV)
                     DATA(I3+1)=DATA(I3REV+1)
                     DATA(I3REV)=TEMPR
                     DATA(I3REV+1)=TEMPI
 12               CONTINUE
 13            CONTINUE
            ENDIF
            IBIT=IP2/2
 1          IF ((IBIT.GE.IP1).AND.(I2REV.GT.IBIT)) THEN
               I2REV=I2REV-IBIT
               IBIT=IBIT/2
               GO TO 1
            ENDIF
            I2REV=I2REV+IBIT
 14      CONTINUE
         IFP1=IP1
 2       IF(IFP1.LT.IP2)THEN
            IFP2=2*IFP1
            THETA=ISIGN*6.28318530717959D0/(IFP2/IP1)
            WPR=-2.D0*DSIN(0.5D0*THETA)**2
            WPI=DSIN(THETA)
            WR=1.D0
            WI=0.D0
            DO 17 I3=1,IFP1,IP1
               DO 16 I1=I3,I3+IP1-2,2
                  DO 15 I2=I1,IP3,IFP2
                     K1=I2
                     K2=K1+IFP1
                     TEMPR=SNGL(WR)*DATA(K2)-SNGL(WI)*DATA(K2+1)
                     TEMPI=SNGL(WR)*DATA(K2+1)+SNGL(WI)*DATA(K2)
                     DATA(K2)=DATA(K1)-TEMPR
                     DATA(K2+1)=DATA(K1+1)-TEMPI
                     DATA(K1)=DATA(K1)+TEMPR
                     DATA(K1+1)=DATA(K1+1)+TEMPI
 15               CONTINUE
 16            CONTINUE
```

```
               WTEMP=WR
               WR=WR*WPR-WI*WPI+WR
               WI=WI*WPR+WTEMP*WPI+WI
 17         CONTINUE
            IFP1=IFP2
            GO TO 2
         ENDIF
         NPREV=N*NPREV
 18   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     Subroutines to compute eigenvalues and eigenvectors
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE TRED2(A,N,NP,D,E)
      IMPLICIT NONE
      INTEGER I,J,K,L,N,NP
      REAL A,D,E,H,SCALE,F,G,HH
      DIMENSION A(NP,NP),D(NP),E(NP)
      IF(N.GT.1)THEN
         DO 18 I=N,2,-1
            L=I-1
            H=0.
            SCALE=0.
            IF(L.GT.1)THEN
               DO 11 K=1,L
                  SCALE=SCALE+ABS(A(I,K))
 11            CONTINUE
               IF(SCALE.EQ.0.)THEN
                  E(I)=A(I,L)
               ELSE
                  DO 12 K=1,L
                     A(I,K)=A(I,K)/SCALE
                     H=H+A(I,K)**2
 12               CONTINUE
                  F=A(I,L)
                  G=-SIGN(SQRT(H),F)
                  E(I)=SCALE*G
                  H=H-F*G
                  A(I,L)=F-G
                  F=0.
                  DO 15 J=1,L
                     A(J,I)=A(I,J)/H
                     G=0.
                     DO 13 K=1,J
                        G=G+A(J,K)*A(I,K)
 13                  CONTINUE
                     IF(L.GT.J)THEN
                        DO 14 K=J+1,L
                           G=G+A(K,J)*A(I,K)
 14                     CONTINUE
                     ENDIF
                     E(J)=G/H
                     F=F+E(J)*A(I,J)
 15               CONTINUE
                  HH=F/(H+H)
                  DO 17 J=1,L
                     F=A(I,J)
                     G=E(J)-HH*F
                     E(J)=G
                     DO 16 K=1,J
                        A(J,K)=A(J,K)-F*E(K)-G*A(I,K)
 16                  CONTINUE
 17               CONTINUE
```

```
               ENDIF
            ELSE
               E(I)=A(I,L)
            ENDIF
            D(I)=H
 18      CONTINUE
      ENDIF
      D(1)=0.
      E(1)=0.
      DO 23 I=1,N
         L=I-1
         IF(D(I).NE.0.)THEN
            DO 21 J=1,L
               G=0.
               DO 19 K=1,L
                  G=G+A(I,K)*A(K,J)
 19            CONTINUE
               DO 20 K=1,L
                  A(K,J)=A(K,J)-G*A(K,I)
 20            CONTINUE
 21         CONTINUE
         ENDIF
         D(I)=A(I,I)
         A(I,I)=1.
         IF(L.GE.1)THEN
            DO 22 J=1,L
               A(I,J)=0.
               A(J,I)=0.
 22         CONTINUE
         ENDIF
 23   CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE TQLI(D,E,N,NP,Z)
      IMPLICIT NONE
      INTEGER N,NP,I,K,L,M,ITER
      REAL D,E,Z,DD,G,R,S,C,P,F,B
      DIMENSION D(NP),E(NP),Z(NP,NP)
      IF (N.GT.1) THEN
         DO 11 I=2,N
            E(I-1)=E(I)
 11      CONTINUE
         E(N)=0.
         DO 15 L=1,N
            ITER=0
 1          DO 12 M=L,N-1
               DD=ABS(D(M))+ABS(D(M+1))
               IF (ABS(E(M))+DD.EQ.DD) GO TO 2
 12         CONTINUE
            M=N
 2          IF(M.NE.L)THEN
               IF(ITER.EQ.30) PAUSE 'too many iterations!'
               ITER=ITER+1
               G=(D(L+1)-D(L))/(2.*E(L))
               R=SQRT(G**2+1.)
               G=D(M)-D(L)+E(L)/(G+SIGN(R,G))
               S=1.
               C=1.
               P=0.
               DO 14 I=M-1,L,-1
                  F=S*E(I)
                  B=C*E(I)
                  IF(ABS(F).GE.ABS(G))THEN
```

```
                     C=G/F
                     R=SQRT(C**2+1.)
                     E(I+1)=F*R
                     S=1./R
                     C=C*S
                  ELSE
                     S=F/G
                     R=SQRT(S**2+1.)
                     E(I+1)=G*R
                     C=1./R
                     S=S*C
                  ENDIF
                  G=D(I+1)-P
                  R=(D(I)-G)*S+2.*C*B
                  P=S*R
                  D(I+1)=G+P
                  G=C*R-B
                  DO 13 K=1,N
                     F=Z(K,I+1)
                     Z(K,I+1)=S*Z(K,I)+C*F
                     Z(K,I)=C*Z(K,I)-S*F
 13               CONTINUE
 14            CONTINUE
               D(L)=D(L)-P
               E(L)=G
               E(M)=0.
               GO TO 1
            ENDIF
 15      CONTINUE
      ENDIF
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE EIGSRT(D,V,N,NP)
      IMPLICIT NONE
      INTEGER N,NP,I,J,K
      REAL D,V,P
      DIMENSION D(NP),V(NP,NP)
      DO 13 I=1,N-1
        K=I
        P=D(I)
        DO 11 J=I+1,N
          IF(D(J).GE.P)THEN
            K=J
            P=D(J)
          ENDIF
11      CONTINUE
        IF(K.NE.I)THEN
          D(K)=D(I)
          D(I)=P
          DO 12 J=1,N
            P=V(J,I)
            V(J,I)=V(J,K)
            V(J,K)=P
12        CONTINUE
        ENDIF
13    CONTINUE
      RETURN
      END
ccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE PIKSRT(N,ARR)
      IMPLICIT NONE
      INTEGER I,J,N
      REAL ARR,A
      DIMENSION ARR(N)
      DO 12 J=2,N
        A=ARR(J)
        DO 11 I=J-1,1,-1
          IF(ARR(I).LE.A)GO TO 10
          ARR(I+1)=ARR(I)
11      CONTINUE
        I=0
10      ARR(I+1)=A
12    CONTINUE
      RETURN
      END
cccccccccccccccccccccccccccccccccccccccccccccccccccc
```

Problem 13:

The output of this program can be generated and visualized as follows. Download in the same directory the source code attached below from

http://ursula.chem.yale.edu/∼batista/classes/summer/P13/P13.tar

and the math libraries from

http://ursula.chem.yale.edu/∼batista/classes/summer/m.tar.

Untar both files by typing

```
tar -xvf P13.tar
```

and

```
tar -xvf m.tar
```

Type

```
cd P13
```

Compile the program with the script by typing

```
  comp_13
```

and run it by typing

```
  Problem13
```

.

Visualize the output as follows: type

```
gnuplot
```

then type

```
plot ``arch.0001''
```

That will show the matching representation of the amplitude of the target state, with one term in the expansion, and

```
replot ``arch.0001 u 1:3''
```

visualizes the real part of target state, also with one term in the expansion. The analytic results can be visualized on top by typing

```
replot ``arch.0001 u 1:4''
```

and

```
replot ``arch.0001 u 1:5''
```

, respectively. Note that since the potential is harmonic, the expansion with a single term is already converged. The results with two and three terms in the expansion can be visualized analogously by using arch.0002 and arch.0003, respectivley. To exit, type

```
quit
```

```
      Program Problem13
c
c     Generate a matching pursuit expansion of the target state
c     |\tilde{Psi}_0> = exp(-i p^2/(2m) tau/2) exp(-i V tau)
c                   exp(-i p^2/(2m) tau/2) |Psi_0>
c     where |Psi_0> is a Gaussian
c
      IMPLICIT NONE
      character*9 B
      INTEGER i,in,j,ISF,ID,npoints,maxbasis,NC,nta,NPT,ntraj,ndic
      REAL*8 dtv,dtt,dtp,mm,norm,normt,x,dx,xmin,xmax,x0,pi
      complex*16 xnc,pnc,FI,rnum,cg,gaussian,eye,cpc,x1,p1,g1
      complex*16 rt,it,rana,cdic,xdic,pdic,gdic
      PARAMETER(NC=1,NPT=4,nta=100,npoints=100)
      DIMENSION x(nc),normt(2),rnum(npoints),mm(NC),pdic(nta,nc)
      DIMENSION x1(nc),p1(nc),g1(nc),cdic(nta),xdic(nta,nc),gdic(nta,nc)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      eye=(0.0d0, 1.0d0)
      pi=dacos(-1.0d0)
      mm(1)=1.0
c
c     Initialize the wavepacket as a single Gaussian
c
      do i=1,NPT
         ntraj(i)=0
      enddo
      ntraj(1)=1    ! Number of terms in the expansion of the initial state
      cpc(1,1)=1.0  ! Expansion coefficients
c
      DO in=1,NC
         xnc(1,in,1) = -2.5  ! Position of initial state
         pnc(1,in,1) = 0.0   ! Momentum of initial state
         FI(1,in,1)  = 1.0    ! Width of initial state
      ENDDO
c
c     Propagation time increments for Trotter expansion
c
      dtt = 0.1
      dtv = dtt
      dtp = dtt/2.0d0
c
c     Initialize dictionary for the Matching Pursuit.
c
      isf=1
      ID=isf*2-1
      ndic=ntraj(ID)       ! number of basis functions at t(ID)
      do i=1,ndic
         do in=1,NC
            xdic(i,in) = xnc(i,in,ID)
            pdic(i,in) = pnc(i,in,ID)
            gdic(i,in) = FI(i,in,ID)
         enddo
      enddo
c
c     Output Initial State
c
      do in=1,NC
         x1(in) = xnc(1,in,ID)
         p1(in) = pnc(1,in,ID)
         g1(in) = FI(1,in,ID)
         print *, "Dimension",in
         print *, "xpg 0",x1(in),p1(in),g1(in)
      enddo
c
      cg=cpc(1,ID)               ! expansion coefficient at initial time t(ID)
      print *, "coef 0",cg
c
c     Save initial wavepacket in file step0
c
      open(unit=100,file="step0")
      xmin  =-10.0
      xmax  = 10.0
      dx=(xmax-xmin)/(npoints-1)
      do i=1,npoints
         x(1)=xmin+dx*(i-1)
         write (100,222) x(1),dreal(cg*gaussian(x,x1,p1,g1))
     $        ,imag(cg*gaussian(x,x1,p1,g1))
      enddo
      close(100)
c
c     Obtain target state |\tilde{Psi}_0> = exp(-i p^2/(2m) tau/2)
c               exp(-i V tau) exp(-i p^2/(2m) tau/2) |Psi_0>
c     as a MP coherent-state expansion
c
      ntraj(ID+1)=0        ! number of basis functions at t(ID+1)
      maxbasis=3           ! maximum # of basis functions in the dictionary
c
      call Match_Pursuit(norm,isf,ndic,gdic,xdic,pdic,
     $     cdic,maxbasis,dtv,dtp,mm)
c
c     Output MP wavepacket after finding each term by sequential
c     orthogonal decomposition
c
      x0=-2.5
      xmin=-10.0
      xmax=10.0
      dx=(xmax-xmin)/(npoints-1)
c
      do i=1,npoints
         rnum(i)=0.
      end do
c
      DO j=1,maxbasis
         write(B, '(A,i4.4)') 'arch.', j
         open(100,file=B)
         do in=1,NC
            x1(in) = xnc(j,in,ID+1)
            p1(in) = pnc(j,in,ID+1)
            g1(in) = FI(j,in,ID+1)
         enddo
         cg=cpc(j,ID+1)
c
c     Save MP wavepacket in file step1
c
         do i=1,npoints
            x(1)=xmin+dx*(i-1)
c
c     Analytic wavepacket for comparision
c
            rt=-(x(1)-x0*cos(0.1))**2/2.0
            it=sin(0.1)*(x0**2*cos(0.1)-2.0*x(1)*x0)/2.0
            rana=(1.0/pi)**0.25*(cos(-0.05)+eye*sin(-0.05))*
     $           cdexp(rt+eye*it)
            rnum(i)=rnum(i)+cg*gaussian(x,x1,p1,g1)
            write (100,222) x(1),dreal(rnum(i)),dimag(rnum(i))
     $           ,dreal(rana),dimag(rana)
```

```
         enddo
         close(1)
      END DO
 222  format(8(e13.6,2x))
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Match_Pursuit(norm,isf,ndic,gdic,xdic,pdic,
     $     cdic,maxbasis,dtv,dtp,mm)
c
      IMPLICIT NONE
      INTEGER i,in,k,j,ISF,ID,maxbasis,ntraj,ndic
      INTEGER imp,Nmax,NC,nta,NPT
      REAL*8 dtv,dtvc,dtp,mm,rcut,c1,c2,norm
      complex*16 x1,p1,x2,p2,g1,g2,xdic,pdic
      complex*16 gdic,cdic,xnc,pnc,FI,EYE,cpc,ovl,precoef,gij
      PARAMETER(rcut=1.E-8,NC=1,NPT=4,nta=100)
      DIMENSION x1(NC),p1(NC),g1(NC),x2(NC),p2(NC),g2(NC)
      DIMENSION mm(nc),xdic(nta,NC),pdic(nta,NC),gdic(nta,NC),cdic(nta)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      EYE=(0.0,1.0)
      id=isf*2                  ! index of optimized c-state
c
c     calculate the overlap cdic for each item in the dictionary.
c
      call overlap(ndic,gdic,xdic,pdic,cdic,ISF,dtv,dtp,mm)
      imp=0
      norm=0.0
 10   continue
c
C     Find the item of the dictionary that is the best match
c
      Nmax=1    ! Nmax is the index of the best match
      do i=1,ndic
         c1=abs(cdic(i))
         c2=abs(cdic(Nmax))
         if (c1.gt.c2) Nmax=i
      enddo
c
C     Use the best match as the initial guess for further optimization
c
      precoef=cdic(Nmax)
      imp=imp+1                       ! index of the term in the expansion
      do in=1,NC
         xnc(imp,in,ID)=xdic(Nmax,in)
         pnc(imp,in,ID)=pdic(Nmax,in)
         FI(imp,in,ID)=gdic(Nmax,in)
      enddo
      cpc(imp,ID)=precoef
c
      call optimize(imp,isf,dtv,dtp,mm)
      ntraj(ID)=imp
      c1=conjg(cpc(imp,ID))*cpc(imp,ID)
      norm=norm+c1
c
c     Cutoff criteria
c
c     if (c1.lt.rcut) goto 27
      if (imp.ge.maxbasis) goto 27 ! maxbasis tells the cutout for expansion
c
c     Compute expansion coefficients
c
      do in=1,NC
```

```
         x2(in)=xnc(imp,in,ID)
         p2(in)=pnc(imp,in,ID)
         g2(in)=FI(imp,in,ID)
      enddo

      do i=1,ndic
         do in=1,NC
            x1(in)=xdic(i,in)
            p1(in)=pdic(i,in)
            g1(in)=gdic(i,in)
         enddo
         call overlap_ggovlc(x1,p1,g1,x2,p2,g2,gij)        !  gij=<1|2>
         cdic(i)=cdic(i)-cpc(imp,ID)*gij         !   expansion coefficient
      enddo
      goto 10
 27   return
      end
C-----------------------------------------------------------------------
      subroutine optimize(imp,ISF,dtv,dtp,mm)
c
C     Gradient-based optimization subroutine to maximize the overlap between
c     the imp-th target function and the trial coherent state
c     which is returned in the common blocks
C     common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
c     common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      implicit none
      integer i,j,in,Nmax,imp,Ndiv,Ntrial
      integer iter,ntraj,diter,ditermax,giter,gitermax
      integer NPROC,me,ierr,rc
      integer ISF,ID
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      real*8 dx,rr,al,al0,ali,ala,alb,alc,ald,dtv,dtvc,dtp
      real*8 rtr,rtar,gain,ratio,almax,expect,norm,up,down,mm(nc)
      complex*16 xp,pp,gp
      complex*16 xa,pa,ga,qa,xb,pb,gb,qb,xc,pc,gc,qc,qd
      complex*16 dr,dp,dg
      complex*16 r1,p1,g1,r2,p2,g2
      complex*16 rm(nta,NC),pm(nta,NC),gm(nta,NC),qm(nta)
      complex*16 xnc,pnc,FI
      complex*16 qsum,c2,c1,c0,c3,c4
      complex*16 gij,ovl,qmp,cpc,qp,eye
      dimension r1(NC),p1(NC),g1(NC),r2(NC),p2(NC),g2(NC)
      dimension dr(NC),dp(NC),dg(NC),rr(6*NC)
      dimension xp(NC),pp(NC),gp(NC)
      dimension xa(NC),pa(NC),ga(NC),xb(NC),pb(NC),gb(NC)
      dimension xc(NC),pc(NC),gc(NC)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      ID=2*ISF
      ntrial=6
      eye=(0.0,1.0)
      do in=1,NC
         r1(in)=xnc(imp,in,ID)
         p1(in)=pnc(imp,in,ID)
         g1(in)=FI(imp,in,ID)
      enddo
      c0=cpc(imp,ID)
      c1=c0
      almax=0.0
      ditermax=0
      gitermax=0
      iter=0
 9    iter=iter+1
      do in=1,NC
         xa(in)=r1(in)
         pa(in)=p1(in)
         ga(in)=g1(in)
      enddo
      qa=c1
      ala=0.0
c
c     Computes the partial derivatives of the overlap with respect to
c     the adjustable CS parameters
c
      call Derivative(r1,p1,g1,c1,rr,ISF,dtv,dtp,mm)
c
      do in=1,NC
         dr(in)=rr(0*NC+in)+rr(1*NC+in)*eye
         dp(in)=rr(2*NC+in)+rr(3*NC+in)*eye
         dg(in)=rr(4*NC+in)+rr(5*NC+in)*eye
         dg(in)=0.0
      enddo
      rtr=0.0
      do in=1,6*NC
         rtr=rtr+rr(in)*rr(in)
      enddo
      rtr=sqrt(rtr)
      if (rtr.eq.0.0) goto 10
      al=abs(c1)/rtr
      if (al.gt.8.0) al=8.0
      if (al.lt.1.0e-1) al=1.0e-1
      al0=al
      diter=0
 15   diter=diter+1
      if ((diter-1)*Ntrial.gt.24) goto 10
c
c     Incrementing parameters along the direction of the gradients
c
 16   do i=1,Ntrial
         ali=al/2.0**(Ntrial-i)
         do in=1,NC
            rm(i,in)=r1(in)+dr(in)/rtr*ali
            pm(i,in)=p1(in)+dp(in)/rtr*ali
            gm(i,in)=g1(in)+dg(in)/rtr*ali
            if (dreal(gm(i,in)).lt.0.0) then
               al=al/2.0
               al0=al
               goto 16
            endif
            if (dreal(gm(i,in)).lt.dreal(g1(in))*0.3) then
               al=al/2.0
               al0=al
               goto 16
            endif
         enddo
      enddo
c
      call overlap(ntrial,gm,rm,pm,qm,ISF,dtv,dtp,mm)
c
c     Select the maximum
c
      if (al.gt.almax) almax=al
      Nmax=1
      do i=1,Ntrial
         if (abs(qm(i)).gt.abs(qm(Nmax))) Nmax=i
      enddo
      c2=qm(Nmax)
      if (abs(c2).le.abs(c1)) then
         al=al/2.0**Ntrial
         goto 15
      endif
      if (diter.gt.ditermax) ditermax=diter
      alb=al/2.0**(Ntrial-Nmax)
      qb=c2
      if (giter.gt.gitermax) gitermax=giter
      ratio=(abs(qb)-abs(c1))/abs(c1)
      if (ratio.gt.0.0) then
         do in=1,NC
            r1(in)=r1(in)+dr(in)/rtr*alb
            p1(in)=p1(in)+dp(in)/rtr*alb
            g1(in)=g1(in)+dg(in)/rtr*alb
         enddo
         c1=qb
      endif
      if ((abs(qb)-abs(c1)).gt.1.0E-5) goto 9
      if (ratio.lt.0.001) goto 10
      if (iter.gt.NC*2) goto 10
      goto 9
 10   continue
c
c     Update trial parameters with optimized parameters
c
      if (abs(c1).gt.abs(c0)) then
         do in=1,NC
            xnc(imp,in,ID)=r1(in)
            pnc(imp,in,ID)=p1(in)
            FI(imp,in,ID)=g1(in)
         enddo
         cpc(imp,ID)=c1
      endif
      qp=cpc(imp,ID)
      gain=(abs(qp)/abs(c0))**2
      return
      end
C-----------------------------------------------------------------------------
      subroutine Derivative(rin,pin,gin,c0,rr,ISF,dtv,dtp,mm)
c
c     Computes the partial derivatives of the overlap with respect to
c     the adjustable CS parameters
c
      implicit none
      integer i,k,in,Ndiv,ISF,ID
      integer NPROC,me,ierr,ntraj,nt
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      real*8 dx,rr,dtv,dtvc,dtp,mm(nc)
      complex*16 x1,p1,g1,x2,p2,g2,rin,pin,gin
      complex*16 c0,c1,eye,gvgovl,gvgovl_id,gvgovly2
      complex*16 xnc,pnc,cpc,FI,ggovl,ggovl_id,ggovlc
      complex*16 rm(nta,NC),pm(nta,NC),gm(nta,NC),qm(nta)
      COMMON /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
      dimension rin(NC),pin(NC),gin(NC)
      dimension x1(NC),p1(NC),g1(NC),x2(NC),p2(NC),g2(NC)
      dimension rr(6*NC)
      dimension gvgovl_id(nta*nta),gvgovl(nta*nta)
      dimension ggovl_id(nta*nta),ggovl(nta*nta)
c
      eye=(0.0,1.0)
      dx=0.001
      do in=1,6*NC
         rr(in)=0.0
      enddo
      do i=1,6*NC
         do in=1,NC
            rm(i,in)=rin(in)
            pm(i,in)=pin(in)
            gm(i,in)=gin(in)
         enddo
         qm(i)=0.0
      enddo
      do in=1,NC
         k=0*NC+in
         rm(k,in)=rm(k,in)+dx
         k=1*NC+in
         rm(k,in)=rm(k,in)+eye*dx
         k=2*NC+in
         pm(k,in)=pm(k,in)+dx
         k=3*NC+in
         pm(k,in)=pm(k,in)+eye*dx
         k=4*NC+in
         gm(k,in)=gm(k,in)+dx
         k=5*NC+in
         gm(k,in)=gm(k,in)+eye*dx
      enddo
      nt=6*NC
      call overlap(nt,gm,rm,pm,qm,ISF,dtv,dtp,mm)
      do i=1,6*NC
         rr(i)=(abs(qm(i))-abs(c0))/dx
      enddo
      return
      end
C----------------------------------------------------------------------------
      subroutine overlap(ndic,gdic,xdic,pdic,cdic,ISF,dtv,dtp,mm)
c
C     Find out which cs from the dictionary has maximum
c     overlap with the target function
c
      IMPLICIT NONE
      integer NPROC,me,ierr,ndiv,nta,NPT,ntraj,I,in,NC
      integer ISF,index_dic,index_ntraj,id,ndic,nmp,idx
      integer index_ntraj12,isfc,idc
      PARAMETER(NC=1,NPT=4,nta=100)
      real*8 dtv,dtvc,dtp,mm(nc)
      complex*16 g1(nc),g2(nc),x1(nc)
      complex*16 x2(nc),p1(nc),p2(nc)
      complex*16 xnc,pnc,cpc,FI,gvgovlc,ggovlc,cdic1(nta)
      complex*16 gdic(nta,nc),xdic(nta,nc),pdic(nta,nc),cdic(nta)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      id=2*isf-1
      do i=1,ndic
         cdic(i)=0.0D0
         cdic1(i)=0.0D0
      enddo
      do index_ntraj=1,ntraj(id)
         do index_dic=1,ndic
            do in=1,NC                          ! trial coherent-state
               g1(in)=gdic(index_dic,in)
               p1(in)=pdic(index_dic,in)
               x1(in)=xdic(index_dic,in)
            enddo
            do in=1,NC          ! expansion terms in the ket_{index_ntraj
               x2(in)=xnc(index_ntraj,in,ID)
               p2(in)=pnc(index_ntraj,in,ID)
               g2(in)= FI(index_ntraj,in,ID)
            enddo
c
c    <trial|Trotter exp.|ket_{index_ntraj}>
c
            call overlap_gexpvg_g1_coupling
     $           (x1,p1,g1,x2,p2,g2,gvgovlc,dtv,dtp,mm)
c
c    <trial|Trotter exp.|target>
c
            cdic1(index_dic)=cdic1(index_dic)+
     $           cpc(index_ntraj,ID)*gvgovlc
            if ((ntraj(ID+1).ge.1.).AND.(index_ntraj.EQ.1)) then
               do i=1,ntraj(ID+1)
                  do in=1,NC
                     x2(in)=xnc(i,in,ID+1)
                     p2(in)=pnc(i,in,ID+1)
                     g2(in)=FI(i,in,ID+1)
                  enddo
                  call overlap_ggovlc(x1,p1,g1,x2,p2,g2,ggovlc)
c
c     <trial|Trotter exp.|residue>
c
                  cdic1(index_dic)=cdic1(index_dic)-cpc(i,ID+1)*ggovlc
               enddo
            endif
         enddo
      enddo
      do i=1,ndic
         cdic(i)=cdic1(i)
      enddo
      return
      end
C-----------------------------------------------------------------------
      subroutine overlap_gexpvg_g1_coupling
     $     (x1,p1,g1,x2,p2,g2,gvgovl,dtv,dtp,mm)
c
c     Calculatea <CS_1| exp(-i K dt/2)*exp(-i V dt)*exp(-i K dt/2)|CS_2 >
c
      IMPLICIT NONE
      INTEGER NG,nx,ny,IND,J,NFLAG,I,in,JJ,Ngd,ISF
      integer NC,nta,NPT,ngrid,isfc
      PARAMETER(NC=1,NPT=4,nta=100)
      REAL*8 mm(nc),dtvc,dtp,pi,dtv
      real*8 x(nc),z(nc),VPOT,xi,wi,xg,wgd
      real*8 a,b,c,d,e,f,dtp1
      real*8 a1,b1,c1,d1,e1,f1
      real*8 a2,b2,c2,d2,e2,f2
      complex*16 x1,x2,p1,p2,g1,g2,gf1,gf2,gaussian_type2,expvc
      complex*16 aa,bb,cc,den,aa1,bb1,cc1,aa2,bb2,cc2,N1,N2
      COMPLEX*16 ovl,ovl1,GF,eye,gvgovl,fx,yovl,gaussian
      real*8  xpro(NC),xmax(NC),xmin(NC),dx(NC)
      dimension x1(NC),x2(NC),p1(NC),p2(NC),g1(NC),g2(NC)
      dimension a(NC),b(NC),c(NC),d(NC),e(NC),f(NC)
      dimension aa(NC),bb(NC),cc(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
      dimension a2(NC),b2(NC),c2(NC),d2(NC),e2(NC),f2(NC)

      integer jn
      real*8 conv
      complex*16 coefAs1(nc,nc),coefBs1(nc),coefCs1
      complex*16 coefAs2(nc,nc),coefBs2(nc),coefCs2
      complex*16 coefAc(nc,nc),coefBc(nc),coefCc

      complex*16 coefA1(nc,nc),coefB1(nc),coefC1
      complex*16 coefA2(nc,nc),coefB2(nc),coefC2

      complex*16 caa1(nc,nc),cbb1(nc),ccc1
      complex*16 caa2(nc,nc),cbb2(nc),ccc2

      integer dim,incx,incy,info,IPIV(nc),ifail
      character*1 trans
      complex*16 zdotu,y(nc),ia(nc,nc),F06GAF
      complex*16 overlap1,overlap2,wkspacei(nc),alpha,beta
      real*8 detr,deti,wkspace(nc)
      integer IPIVOT(nc),job
      complex work(nc),det(2),sia(nc,nc)
c
      dtp1=-dtp
      pi=dacos(-1.0d0)
      eye=(0.0,1.0)
c
      coefCs1=0.0
      coefBs1(1)=0.0
      coefAs1(1,1)=0.5
c
      coefC1=-eye*dtv*coefCs1
      do i=1,nc
         coefB1(i)=-eye*dtv*coefBs1(i)
         do j=1,nc
            coefA1(i,j)=-eye*dtv*coefAs1(i,j)
         enddo
      enddo
c
      ccc1=coefC1
      N1=1.0
      N2=1.0
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))

         a2(in)=dreal(g2(in))
         b2(in)=dimag(g2(in))
         c2(in)=dreal(x2(in))
         d2(in)=dimag(x2(in))
         e2(in)=dreal(p2(in))
         f2(in)=dimag(p2(in))
c
c     Normalization constants
c
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
     &        *sqrt(mm(in)/(mm(in)+eye*dtp1*g1(in)))
         N2=N2*(a2(in)/pi)**0.25*exp(-0.5*a2(in)*d2(in)**2
     &        -d2(in)*e2(in)-(b2(in)*d2(in)+f2(in))**2/2.0/a2(in))
     &        *sqrt(mm(in)/(mm(in)+eye*dtp*g2(in)))
c
c     Integrand=N2*exp(aa2*x^2+cc2*x+cc2)*conjg(N1*exp(aa1*x^2+cc1*x+cc1))
c                   *exp(ccc1+cbb1*x+caa1*x^2)
```

```
c
            den=2.0+2.0*eye*dtp*g2(in)/mm(in)
            aa2=-g2(in)/den
            bb2=(2.0*eye*p2(in)+2.0*g2(in)*x2(in))/den
            cc2=(p2(in)-eye*g2(in)*x2(in))**2/g2(in)/den
     &           -p2(in)**2/2.0/g2(in)

            den=2.0+2.0*eye*dtp1*g1(in)/mm(in)
            aa1=-g1(in)/den
            bb1=(2.0*eye*p1(in)+2.0*g1(in)*x1(in))/den
            cc1=(p1(in)-eye*g1(in)*x1(in))**2/g1(in)/den
     &           -p1(in)**2/2.0/g1(in)

            ccc1=ccc1+dconjg(cc1)+cc2
            cbb1(in)=dconjg(bb1)+bb2+coefB1(in)
            do jn=1,nc
               if (in.eq.jn) then
                  caa1(in,jn)=dconjg(aa1)+aa2+coefA1(in,jn)
               else
                  caa1(in,jn)=coefA1(in,jn)
               endif
            enddo
         enddo
         dim=nc
         do i=1,nc
            y(i)=0.0
            cbb1(i)=-cbb1(i)
            do j=1,nc
               caa1(i,j)=-caa1(i,j)
               ia(i,j)=caa1(i,j)
               sia(i,j)=ia(i,j)
            enddo
         enddo
c
c        NAG subroutines
c
c        call F03ADF(caa1,dim,dim,detr,deti,wkspace,ifail)
c        overlap1=dsqrt(pi**dim)/cdsqrt(detr+eye*deti)
         job=11
c
c        SGI subroutines for computations of the determinant
c
         call CGEFA(sIA,dim,dim,IPIVOT,INFO)
         CALL CGEdi(sIA, dim, dim, IPIVOT, DET, WORK, JOB)
         overlap1=dsqrt(pi**dim)/sqrt(det(1)*10.0**det(2))
c
c        call F07ARF(dim,dim,IA,dim,IPIV,info)
         call zgetrf(dim,dim,IA,dim,IPIV,info)
c        call F07AWF(dim,IA,dim,IPIV,wkspacei,dim,info)
         call zgetri(dim,IA,dim,IPIV,wkspacei,dim,info)
c
         trans='N'
         alpha=1.0d0
         beta=0.0d0
         do i=1,dim
            y(i)=0.0d0
         enddo
         incx=1     !  Matrix multiplication for exponent of the G-integral
         incy=1
c        call F06SAF(trans,dim,dim,alpha,IA,dim,cbb1,incx,beta,y,incy)
c        overlap1=overlap1*cdexp(F06GAF(dim,cbb1,incx,y,incy)/4.0d0+ccc1)
         call zgemv(trans,dim,dim,alpha,IA,dim,cbb1,incx,beta,y,incy)
         overlap1=dconjg(N1)*N2
     $      *overlap1*cdexp(zdotu(dim,cbb1,incx,y,incy)/4.0d0+ccc1)
```

```
      gvgovl=overlap1
      RETURN
      END
C-----------------------------------------------------------------------
      subroutine overlap_ggovlc(r1,p1,g1,r2,p2,g2,govl)
c
c     calculate <r1,p1,g1|r2,p2,g2> analytically, the parameters
c     are complex numbers
c
      implicit none
      integer in,NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      real*8 pi,a1,b1,c1,d1,e1,f1,a2,b2,c2,d2,e2,f2
      complex*16 r1,r2,p1,p2,g1,g2
      complex*16 N1,N2,aa1,bb1,cc1,aa2,bb2,cc2,aa,bb,cc
      complex*16 phi22,eye,govl
      dimension r1(NC),r2(NC),p1(NC),p2(NC),g1(NC),g2(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
      dimension a2(NC),b2(NC),c2(NC),d2(NC),e2(NC),f2(NC)
c
      eye=(0.0,1.0)
      pi=3.141592654
      N1=1.0
      N2=1.0
      phi22=1.0
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(r1(in))
         d1(in)=dimag(r1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
         a2(in)=dreal(g2(in))
         b2(in)=dimag(g2(in))
         c2(in)=dreal(r2(in))
         d2(in)=dimag(r2(in))
         e2(in)=dreal(p2(in))
         f2(in)=dimag(p2(in))
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
         N2=N2*(a2(in)/pi)**0.25*exp(-0.5*a2(in)*d2(in)**2
     &        -d2(in)*e2(in)-(b2(in)*d2(in)+f2(in))**2/2.0/a2(in))
         aa1=-0.5*g1(in)
         bb1=g1(in)*r1(in)+eye*p1(in)
         cc1=-0.5*g1(in)*r1(in)**2-eye*p1(in)*r1(in)
         aa2=-0.5*g2(in)
         bb2=g2(in)*r2(in)+eye*p2(in)
         cc2=-0.5*g2(in)*r2(in)**2-eye*p2(in)*r2(in)
         aa=conjg(aa1)+aa2
         bb=conjg(bb1)+bb2
         cc=conjg(cc1)+cc2
         phi22=phi22*exp(-bb**2/4.0/aa+cc)
         phi22=phi22*sqrt(-pi/aa)
         if (dreal(aa).gt.0.0) then
            print *,"r1=",r1(in)
            print *,"r1=",p1(in)
            print *,"r1=",g1(in)
            print *,"r2=",r2(in)
            print *,"p2=",p2(in)
            print *,"g2=",g2(in)
            print *,"aa=",aa
            print *,"error"
            stop
         endif
      enddo
      phi22=phi22*conjg(N1)*N2
      if (abs(phi22).gt.1.0E20) phi22=0.0
      if (abs(phi22).lt.1.0E-20) phi22=0.0
      govl=phi22
      return
      end
C---------------------------------------------------------------------------
      FUNCTION gaussian(x,x1,p1,g1)
c
c     Gaussian basis fucntion
c
      IMPLICIT NONE
      INTEGER in
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)

      REAL*8 x
      real*8 pi,a1,b1,c1,d1,e1,f1
      complex*16 x1,p1,g1
      COMPLEX*16 EYE,GAU,gaussian,N1
      DIMENSION x(NC),x1(NC),p1(NC),g1(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
c
      pi=3.141592654
      EYE=(0.0,1.0)
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
      enddo
c
      N1=1.0
      do in=1,NC
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
      enddo
c
      GAU=1.0
      DO in=1,NC
         GAU=GAU*EXP(-0.5*g1(in)*(x(in)-x1(in))**2
     &        +EYE*p1(in)*(x(in)-x1(in)))
      END DO
      GAU=GAU*N1
c
      if (abs(gau).gt.1.0E20) gau=0.0
      if (abs(gau).lt.1.0E-20) gau=0.0
      gaussian=gau
c
      RETURN
      END
C----------------------------------------------------------------------------
      FUNCTION gaussian_type2(x,x1,p1,g1,dt,m)
c
c     Gaussian basis function operated by the kinetic operator
c
      IMPLICIT NONE
      INTEGER in
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      REAL*8 x,pi,m,dt
      real*8 a1,b1,c1,d1,e1,f1
      complex*16 x1,p1,g1
      COMPLEX*16 EYE,GAU2,rnum,rden,gaussian_type2,N1
      DIMENSION x(NC),x1(NC),p1(NC),g1(NC),m(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
c
      pi=3.141592654
      EYE=(0.0,1.0)
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
      enddo
c
      N1=1.0
      do in=1,NC
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
      enddo
c
      GAU2=1.0
      DO in=1,NC
         rnum=p1(in)/g1(in)-EYE*(x1(in)-x(in))
         rden=2.0/g1(in)+EYE*2.0*dt/m(in)
         GAU2=GAU2*EXP(rnum**2/rden-p1(in)**2/(2.0*g1(in)))
     &        *sqrt(m(in)/(m(in)+eye*g1(in)*dt))
      END DO
      GAU2=GAU2*N1
      if (abs(gau2).gt.1.0E20) gau2=0.0
      if (abs(gau2).lt.1.0E-20) gau2=0.0
      gaussian_type2=gau2
      RETURN
      END
c-----------------------------------------------------------------------
```

Problem 14:

The output of this program can be generated and visualized as follows. Download in the same directory the source code attached below from

http://ursula.chem.yale.edu/∼batista/classes/summer/P14/P14.tar

and the math libraries from

http://ursula.chem.yale.edu/∼batista/classes/summer/m.tar.

Untar both files by typing

```
tar -xvf P14.tar
```

and

```
tar -xvf m.tar
```

Type

```
cd P14
```

Compile the program with the script by typing

```
  comp_14
```

and run it by typing

```
  Problem14
```

.

The snapshots of the time-dependent wave-packet can be visualized by compiling the program plot.f by executing plot_14, running the plot executable and then displaying the movie by typing

```
 gnuplot<pp_14
```

where the file named

```
 pp_14
```

has the following lines:

```
set yrange[-1:1]
set xrange[-10:10]
set dat sty l
plot "arch.0001" u 1:2 lw 3
pause .1
plot "arch.0002" u 1:2 lw 3
pause .1
plot "arch.0003" u 1:2 lw 3
pause .1
plot "arch.0004" u 1:2 lw 3
pause .1
plot "arch.0005" u 1:2 lw 3
pause .1
plot "arch.0006" u 1:2 lw 3
pause .1
plot "arch.0007" u 1:2 lw 3
pause .1
plot "arch.0008" u 1:2 lw 3
pause .1
plot "arch.0009" u 1:2 lw 3
pause .1
plot "arch.0010" u 1:2 lw 3
pause .1
plot "arch.0011" u 1:2 lw 3
```

```
pause .1
plot "arch.0012" u 1:2 lw 3
pause .1
plot "arch.0013" u 1:2 lw 3
pause .1
plot "arch.0014" u 1:2 lw 3
pause .1
plot "arch.0015" u 1:2 lw 3
pause .1
plot "arch.0016" u 1:2 lw 3
pause .1
plot "arch.0017" u 1:2 lw 3
pause .1
plot "arch.0018" u 1:2 lw 3
pause .1
plot "arch.0019" u 1:2 lw 3
pause .1
plot "arch.0020" u 1:2 lw 3
pause .1
plot "arch.0021" u 1:2 lw 3
pause .1
plot "arch.0022" u 1:2 lw 3
pause .1
plot "arch.0023" u 1:2 lw 3
pause .1
plot "arch.0024" u 1:2 lw 3
pause .1
plot "arch.0025" u 1:2 lw 3
pause .1
plot "arch.0026" u 1:2 lw 3
pause .1
plot "arch.0027" u 1:2 lw 3
pause .1
plot "arch.0028" u 1:2 lw 3
pause .1
plot "arch.0029" u 1:2 lw 3
pause .1
plot "arch.0030" u 1:2 lw 3
pause .1
plot "arch.0031" u 1:2 lw 3
pause .1
plot "arch.0032" u 1:2 lw 3
pause .1
plot "arch.0033" u 1:2 lw 3
pause .1
plot "arch.0034" u 1:2 lw 3
pause .1
plot "arch.0035" u 1:2 lw 3
pause .1
plot "arch.0036" u 1:2 lw 3
pause .1
plot "arch.0037" u 1:2 lw 3
pause .1
plot "arch.0038" u 1:2 lw 3
pause .1
plot "arch.0039" u 1:2 lw 3
pause .1
plot  "arch.0040" u 1:2 lw 3
pause .1
plot "arch.0041" u 1:2 lw 3
pause .1
plot "arch.0042" u 1:2 lw 3
pause .1
plot "arch.0043" u 1:2 lw 3
```

```
pause .1
plot "arch.0044" u 1:2 lw 3
pause .1
plot "arch.0045" u 1:2 lw 3
pause .1
plot "arch.0046" u 1:2 lw 3
pause .1
plot "arch.0047" u 1:2 lw 3
pause .1
plot "arch.0048" u 1:2 lw 3
pause .1
plot "arch.0049" u 1:2 lw 3
pause .1
plot "arch.0050" u 1:2 lw 3
pause .1
plot "arch.0051" u 1:2 lw 3
pause .1
plot "arch.0052" u 1:2 lw 3
pause .1
plot "arch.0053" u 1:2 lw 3
pause .1
plot "arch.0054" u 1:2 lw 3
pause .1
plot "arch.0055" u 1:2 lw 3
pause .1
plot "arch.0056" u 1:2 lw 3
pause .1
plot "arch.0057" u 1:2 lw 3
pause .1
plot "arch.0058" u 1:2 lw 3
pause .1
plot "arch.0059" u 1:2 lw 3
pause .1
plot "arch.0060" u 1:2 lw 3
pause .1
plot "arch.0061" u 1:2 lw 3
pause .1
plot "arch.0062" u 1:2 lw 3
pause .1
plot "arch.0063" u 1:2 lw 3
pause .1
plot "arch.0064" u 1:2 lw 3
pause .1
plot "arch.0065" u 1:2 lw 3
pause .1
plot "arch.0066" u 1:2 lw 3
pause .1
plot "arch.0067" u 1:2 lw 3
pause .1
plot "arch.0068" u 1:2 lw 3
pause .1
plot "arch.0069" u 1:2 lw 3
pause .1
plot "arch.0070" u 1:2 lw 3
pause .1
plot "arch.0071" u 1:2 lw 3
pause .1
plot "arch.0072" u 1:2 lw 3
pause .1
plot "arch.0073" u 1:2 lw 3
pause .1
plot "arch.0074" u 1:2 lw 3
pause .1
plot "arch.0075" u 1:2 lw 3
```

```
pause .1
plot "arch.0076" u 1:2 lw 3
pause .1
plot "arch.0077" u 1:2 lw 3
pause .1
plot "arch.0078" u 1:2 lw 3
pause .1
plot "arch.0079" u 1:2 lw 3
pause .1
plot  "arch.0080" u 1:2 lw 3
pause .1
plot "arch.0081" u 1:2 lw 3
pause .1
plot "arch.0082" u 1:2 lw 3
pause .1
plot "arch.0083" u 1:2 lw 3
pause .1
plot "arch.0084" u 1:2 lw 3
pause .1
plot "arch.0085" u 1:2 lw 3
pause .1
plot "arch.0086" u 1:2 lw 3
pause .1
plot "arch.0087" u 1:2 lw 3
pause .1
plot "arch.0088" u 1:2 lw 3
pause .1
plot "arch.0089" u 1:2 lw 3
pause .1
plot  "arch.0090" u 1:2 lw 3
pause .1
plot "arch.0091" u 1:2 lw 3
pause .1
plot "arch.0092" u 1:2 lw 3
pause .1
plot "arch.0093" u 1:2 lw 3
pause .1
plot "arch.0094" u 1:2 lw 3
pause .1
plot "arch.0095" u 1:2 lw 3
pause .1
plot "arch.0096" u 1:2 lw 3
pause .1
plot "arch.0097" u 1:2 lw 3
pause .1
plot "arch.0098" u 1:2 lw 3
pause .1
plot "arch.0099" u 1:2 lw 3
pause .1
```

```
      Program Problem14
c
c     MP/SOFT propagation
c
      IMPLICIT NONE
      character*9 B
      INTEGER i,in,j,ISF,ID,npoints,maxbasis,NC,nta,NPT,ntraj,ndic,nstep
      REAL*8 dtv,dtt,dtp,mm,norm,normt,x,dx,xmin,xmax,x0,pi
      complex*16 xnc,pnc,FI,rnum,cg,gaussian,eye,cpc,x1,p1,g1
      complex*16 rt,it,rana,cdic,xdic,pdic,gdic
      PARAMETER(NC=1,NPT=2,nta=100,npoints=100)
      DIMENSION x(nc),normt(2),rnum(npoints),mm(NC),pdic(nta,nc)
      DIMENSION x1(nc),p1(nc),g1(nc),cdic(nta),xdic(nta,nc),gdic(nta,nc)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      character*30 num,name
c
      eye=(0.0d0, 1.0d0)
      pi=dacos(-1.0d0)
      mm(1)=1.0
c
c     Initialize the wavepacket as a single Gaussian
c
      do i=1,NPT
         ntraj(i)=0
      enddo
      ntraj(1)=1                ! Number of terms in the initial expansion
      cpc(1,1)=1.0              ! Expansion coefficients
c
      DO in=1,NC
         xnc(1,in,1) = -2.5     ! Position of initial state
         pnc(1,in,1) = 0.0      ! Momentum of initial state
         FI(1,in,1)  = 1.0      ! Width of initial state
      ENDDO
c
c     Propagation increments for the Trotter expansion
c
      dtt = 0.1
      dtv = dtt
      dtp = dtt/2.0d0
      nstep=20                  ! propagation step.
      maxbasis=10               ! maximum # of basis functions in the dict.
c
      j=0
      call number_to_char(j,num)
      name="reinit."//num
      open(2,file=name)
      write(2,24) j,ntraj(1),dtt,norm
      do i=1,ntraj(1)
         do in=1,NC
            write(2,22) xnc(i,in,1),pnc(i,in,1),FI(i,in,1)
         enddo
         write(2,22) cpc(i,1),abs(cpc(i,1))**2
      enddo
         write(2,22)
      close(2)
c
c     Propagation loop
c
      do j=1,nstep

         isf=1
         ID=isf*2-1
         ndic=ntraj(ID)         ! number of basis functions at t(ID)

         do i=1,ndic
            do in=1,NC
               xdic(i,in) = xnc(i,in,ID)
               pdic(i,in) = pnc(i,in,ID)
               gdic(i,in) = FI(i,in,ID)
            enddo
c     print *, "xpg",xdic(i,1),pdic(i,1),gdic(i,1)
         enddo
         print *, "step ", j, "in propagation"
         ntraj(ID+1)=0          ! number of basis functions at t(ID+1)
c
         call Match_Pursuit(norm,isf,ndic,gdic,xdic,pdic,
     $        cdic,maxbasis,dtv,dtp,mm)

         do i=1,ntraj(ID+1)     ! Update Wave Function
            do in=1,NC
               xnc(i,in,ID)=xnc(i,in,ID+1)
               pnc(i,in,ID)=pnc(i,in,ID+1)
               FI(i,in,ID)=FI(i,in,ID+1)
            enddo
            cpc(i,ID)=cpc(i,ID+1)
         enddo
         ntraj(ID)=ntraj(ID+1)

         call number_to_char(j,num)
         name="reinit."//num
         open(2,file=name)
         write(2,24) j,ntraj(1),dtt,norm
         do i=1,ntraj(1)
            do in=1,NC
               write(2,22) xnc(i,in,1),pnc(i,in,1),FI(i,in,1)
            enddo
            write(2,22) cpc(i,1),abs(cpc(i,1))**2
         enddo
         close(2)
      enddo
 22   FORMAT(6(e13.6,2x))
 24   format(I6,2x,I6,2x,e13.6,2x,e13.6)
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      SUBROUTINE Match_Pursuit(norm,isf,ndic,gdic,xdic,pdic,
     $     cdic,maxbasis,dtv,dtp,mm)
c
      IMPLICIT NONE
      INTEGER i,in,k,j,ISF,ID,maxbasis,ntraj,ndic
      INTEGER imp,Nmax,NC,nta,NPT
      REAL*8 dtv,dtvc,dtp,mm,rcut,c1,c2,norm
      complex*16 x1,p1,x2,p2,g1,g2,xdic,pdic
      complex*16 gdic,cdic,xnc,pnc,FI,EYE,cpc,ovl,precoef,gij
      PARAMETER(rcut=1.E-8,NC=1,NPT=2,nta=100)
      DIMENSION x1(NC),p1(NC),g1(NC),x2(NC),p2(NC),g2(NC)
      DIMENSION mm(nc),xdic(nta,NC),pdic(nta,NC),gdic(nta,NC),cdic(nta)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      EYE=(0.0,1.0)
      id=isf*2
c
c     calculate the overlap cdic for each item in the dictionary.
c
      call overlap(ndic,gdic,xdic,pdic,cdic,ISF,dtv,dtp,mm)
      imp=0
      norm=0.0
 10   continue

c      print *, "isf",isf
c      print *, "ndic",ndic
c      print *, cdic(1)
c      print *,gdic(1,1),xdic(1,1),pdic(1,1)

c
C     Find the item of the dictionary that is the best match
c
      Nmax=1    ! Nmax is the index of the best match
      do i=1,ndic
         c1=abs(cdic(i))
         c2=abs(cdic(Nmax))
         if (c1.gt.c2) Nmax=i
      enddo
c
C     Use the best match as the initial guess for further optimization
c
      precoef=cdic(Nmax)
      imp=imp+1
      do in=1,NC
         xnc(imp,in,ID)=xdic(Nmax,in)
         pnc(imp,in,ID)=pdic(Nmax,in)
         FI(imp,in,ID)=gdic(Nmax,in)
      enddo
      cpc(imp,ID)=precoef
c
      call optimize(imp,isf,dtv,dtp,mm)
      ntraj(ID)=imp
      c1=conjg(cpc(imp,ID))*cpc(imp,ID)
      norm=norm+c1
c
c     Cutoff criteria
c
c     if (c1.lt.rcut) goto 27
      if (imp.ge.maxbasis) goto 27 ! maxbasis tells the cutout for expansion
c
c     Compute expansion coefficients
c
      do in=1,NC
         x2(in)=xnc(imp,in,ID)
         p2(in)=pnc(imp,in,ID)
         g2(in)=FI(imp,in,ID)
      enddo

      do i=1,ndic
         do in=1,NC
            x1(in)=xdic(i,in)
            p1(in)=pdic(i,in)
            g1(in)=gdic(i,in)
         enddo
         call overlap_ggovlc(x1,p1,g1,x2,p2,g2,gij)  ! gij=<1|2>
         cdic(i)=cdic(i)-cpc(imp,ID)*gij             ! expansion coefficient
      enddo
      goto 10
 27   return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine optimize(imp,ISF,dtv,dtp,mm)
c
C     Gradient-based optimization subroutine to maximize the overlap between
c     the target function and the imp-th coherent state
```

```
c     which is returned in the common blocks
C     common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
c     common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      implicit none
      integer i,j,in,Nmax,imp,Ndiv,Ntrial
      integer iter,ntraj,diter,ditermax,giter,gitermax
      integer NPROC,me,ierr,rc
      integer ISF,ID
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=2,nta=100)
      real*8 dx,rr,al,al0,ali,ala,alb,alc,ald,dtv,dtvc,dtp
      real*8 rtr,rtar,gain,ratio,almax,expect,norm,up,down,mm(nc)
      complex*16 xp,pp,gp
      complex*16 xa,pa,ga,qa,xb,pb,gb,qb,xc,pc,gc,qc,qd
      complex*16 dr,dp,dg
      complex*16 r1,p1,g1,r2,p2,g2
      complex*16 rm(nta,NC),pm(nta,NC),gm(nta,NC),qm(nta)
      complex*16 xnc,pnc,FI
      complex*16 qsum,c2,c1,c0,c3,c4
      complex*16 gij,ovl,qmp,cpc,qp,eye
      dimension r1(NC),p1(NC),g1(NC),r2(NC),p2(NC),g2(NC)
      dimension dr(NC),dp(NC),dg(NC),rr(6*NC)
      dimension xp(NC),pp(NC),gp(NC)
      dimension xa(NC),pa(NC),ga(NC),xb(NC),pb(NC),gb(NC)
      dimension xc(NC),pc(NC),gc(NC)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
      ID=2*ISF
      ntrial=6
      eye=(0.0,1.0)
      do in=1,NC
         r1(in)=xnc(imp,in,ID)
         p1(in)=pnc(imp,in,ID)
         g1(in)=FI(imp,in,ID)
      enddo
      c0=cpc(imp,ID)
      c1=c0
      almax=0.0
      ditermax=0
      gitermax=0
      iter=0
 9    iter=iter+1
      do in=1,NC
         xa(in)=r1(in)
         pa(in)=p1(in)
         ga(in)=g1(in)
      enddo
      qa=c1
      ala=0.0
      call Derivative(r1,p1,g1,c1,rr,ISF,dtv,dtp,mm)
      do in=1,NC
         dr(in)=rr(0*NC+in)+rr(1*NC+in)*eye
         dp(in)=rr(2*NC+in)+rr(3*NC+in)*eye
         dg(in)=rr(4*NC+in)+rr(5*NC+in)*eye
         dg(in)=0.0
      enddo
      rtr=0.0
      do in=1,6*NC
         rtr=rtr+rr(in)*rr(in)
      enddo
      rtr=sqrt(rtr)
      if (rtr.eq.0.0) goto 10
      al=abs(c1)/rtr
      if (al.gt.8.0) al=8.0
      if (al.lt.1.0e-1) al=1.0e-1
      al0=al
      diter=0
 15   diter=diter+1
      if ((diter-1)*Ntrial.gt.24) goto 10
 16   do i=1,Ntrial
         ali=al/2.0**(Ntrial-i)
         do in=1,NC
            rm(i,in)=r1(in)+dr(in)/rtr*ali
            pm(i,in)=p1(in)+dp(in)/rtr*ali
            gm(i,in)=g1(in)+dg(in)/rtr*ali
            if (dreal(gm(i,in)).lt.0.0) then
               al=al/2.0
               al0=al
               goto 16
            endif
            if (dreal(gm(i,in)).lt.dreal(g1(in))*0.3) then
               al=al/2.0
               al0=al
               goto 16
            endif
         enddo
      enddo
      call overlap
     $     (ntrial,gm,rm,pm,qm,ISF,dtv,dtp,mm)

      if (al.gt.almax) almax=al
      Nmax=1
      do i=1,Ntrial
         if (abs(qm(i)).gt.abs(qm(Nmax))) Nmax=i
      enddo
      c2=qm(Nmax)
      if (abs(c2).le.abs(c1)) then
         al=al/2.0**Ntrial
         goto 15
      endif
      if (diter.gt.ditermax) ditermax=diter

      alb=al/2.0**(Ntrial-Nmax)
      qb=c2
      if (giter.gt.gitermax) gitermax=giter
      ratio=(abs(qb)-abs(c1))/abs(c1)
      if (ratio.gt.0.0) then
         do in=1,NC
            r1(in)=r1(in)+dr(in)/rtr*alb
            p1(in)=p1(in)+dp(in)/rtr*alb
            g1(in)=g1(in)+dg(in)/rtr*alb
         enddo
         c1=qb
      endif
      if ((abs(qb)-abs(c1)).gt.1.0E-5) goto 9
      if (ratio.lt.0.001) goto 10
      if (iter.gt.NC*2) goto 10
      goto 9
 10   continue
      if (abs(c1).gt.abs(c0)) then
         do in=1,NC
            xnc(imp,in,ID)=r1(in)
            pnc(imp,in,ID)=p1(in)
            FI(imp,in,ID)=g1(in)
         enddo
         cpc(imp,ID)=c1
      endif
      qp=cpc(imp,ID)
      gain=(abs(qp)/abs(c0))**2
 22   format(8(e13.6,2x))
      return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine Derivative(rin,pin,gin,c0,rr,ISF,dtv,dtp,mm)
c
c     Computes the partial derivatives of the overlap with respect to
c     the adjustable CC parameters
c
      implicit none
      integer i,k,in,Ndiv,ISF,ID
      integer NPROC,me,ierr,ntraj,nt
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=2,nta=100)
      real*8 dx,rr,dtv,dtvc,dtp,mm(nc)
      complex*16 x1,p1,g1,x2,p2,g2,rin,pin,gin
      complex*16 c0,c1,eye,gvgovl,gvgovl_id,gvgovly2
      complex*16 xnc,pnc,cpc,FI,ggovl,ggovl_id,ggovlc
      complex*16 rm(nta,NC),pm(nta,NC),gm(nta,NC),qm(nta)
      COMMON /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
      dimension rin(NC),pin(NC),gin(NC)
      dimension x1(NC),p1(NC),g1(NC),x2(NC),p2(NC),g2(NC)
      dimension rr(6*NC)
      dimension gvgovl_id(nta*nta),gvgovl(nta*nta)
      dimension ggovl_id(nta*nta),ggovl(nta*nta)
c
      eye=(0.0,1.0)
      dx=0.001
      do in=1,6*NC
         rr(in)=0.0
      enddo
      do i=1,6*NC
         do in=1,NC
            rm(i,in)=rin(in)
            pm(i,in)=pin(in)
            gm(i,in)=gin(in)
         enddo
         qm(i)=0.0
      enddo
      do in=1,NC
         k=0*NC+in
         rm(k,in)=rm(k,in)+dx
         k=1*NC+in
         rm(k,in)=rm(k,in)+eye*dx
         k=2*NC+in
         pm(k,in)=pm(k,in)+dx
         k=3*NC+in
         pm(k,in)=pm(k,in)+eye*dx
         k=4*NC+in
         gm(k,in)=gm(k,in)+dx
         k=5*NC+in
         gm(k,in)=gm(k,in)+eye*dx
      enddo
      nt=6*NC
      call overlap
     $     (nt,gm,rm,pm,qm,ISF,dtv,dtp,mm)

      do i=1,6*NC
         rr(i)=(abs(qm(i))-abs(c0))/dx
      enddo
      return
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine overlap(ndic,gdic,xdic,pdic,cdic,ISF,dtv,dtp,mm)
c
C     Find out which cc from the dictionary has maximum
c     overlap with the target function
c
      IMPLICIT NONE
      integer NPROC,me,ierr,ndiv,nta,NPT,ntraj,I,in,NC
      integer ISF,index_dic,index_ntraj,id,ndic,nmp,idx
      integer index_ntraj12,isfc,idc
      PARAMETER(NC=1,NPT=2,nta=100)
      real*8 dtv,dtvc,dtp,mm(nc)
      complex*16 g1(nc),g2(nc),x1(nc)
      complex*16 x2(nc),p1(nc),p2(nc)
      complex*16 xnc,pnc,cpc,FI,gvgovlc,ggovlc,cdic1(nta)
      complex*16 gdic(nta,nc),xdic(nta,nc),pdic(nta,nc),cdic(nta)
      common /NUCLEAR/ xnc(nta,NC,NPT),pnc(nta,NC,NPT)
      common /NUC/ cpc(nta,NPT),FI(nta,NC,NPT),ntraj(NPT)
c
      id=2*isf-1
      do i=1,ndic
         cdic(i)=0.0D0
         cdic1(i)=0.0D0
      enddo
      do index_ntraj=1,ntraj(id)
         do index_dic=1,ndic
            do in=1,NC
               g1(in)=gdic(index_dic,in)
               p1(in)=pdic(index_dic,in)
               x1(in)=xdic(index_dic,in)
            enddo
            do in=1,NC
               x2(in)=xnc(index_ntraj,in,ID)
               p2(in)=pnc(index_ntraj,in,ID)
               g2(in)= FI(index_ntraj,in,ID)
            enddo
            call overlap_gexpvg_g1_coupling
     $           (x1,p1,g1,x2,p2,g2,gvgovlc,dtv,dtp,mm)

c            print *, "gv",gvgovlc
            cdic1(index_dic)=cdic1(index_dic)+
     $           cpc(index_ntraj,ID)*gvgovlc

            if ((ntraj(ID+1).ge.1.).AND.(index_ntraj.EQ.1)) then
               do i=1,ntraj(ID+1)
                  do in=1,NC
                     x2(in)=xnc(i,in,ID+1)
                     p2(in)=pnc(i,in,ID+1)
                     g2(in)=FI(i,in,ID+1)
                  enddo
                  call overlap_ggovlc(x1,p1,g1,x2,p2,g2,ggovlc)
                  cdic1(index_dic)=cdic1(index_dic)-cpc(i,ID+1)*ggovlc
               enddo
            endif
         enddo
      enddo
      do i=1,ndic
         cdic(i)=cdic1(i)
      enddo

c      print *,"cdic", cdic(1)
      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine number_to_char(filenumber,dotb)
      implicit none
      integer i,filenumber,number,jk,num(5)
      character dotb*5,c,name*10

      number=filenumber
      jk=0
      do i=1,5
         dotb(i:i)=" "
      enddo
      do while (number.gt.9)
         i=mod(number,10)
         jk=jk+1
         num(jk)=ichar('0')+i
         number=number/10
      enddo
      jk=jk+1
      num(jk)=ichar('0')+number

      do i=1,jk
         c=char(num(i))
         dotb(jk-i+1:jk-i+1)=c
      enddo
      return
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine overlap_gexpvg_g1_coupling
     $     (x1,p1,g1,x2,p2,g2,gvgovl,dtv,dtp,mm)
c
c     Calculatea <CS_1|exp(-iKdt/2)*exp(-iVdt)*exp(-iKdt/2)|CS_2>
c
      IMPLICIT NONE
      INTEGER NG,nx,ny,IND,J,NFLAG,I,in,JJ,Ngd,ISF
      integer NC,nta,NPT,ngrid,isfc
      PARAMETER(NC=1,NPT=4,nta=100)
      REAL*8 mm(nc),dtvc,dtp,pi,dtv
      real*8 x(nc),z(nc),VPOT,xi,wi,xg,wgd
      real*8 a,b,c,d,e,f,dtp1
      real*8 a1,b1,c1,d1,e1,f1
      real*8 a2,b2,c2,d2,e2,f2
      complex*16 x1,x2,p1,p2,g1,g2,gf1,gf2,gaussian_type2,expvc
      complex*16 aa,bb,cc,den,aa1,bb1,cc1,aa2,bb2,cc2,N1,N2
      COMPLEX*16 ovl,ovl1,GF,eye,gvgovl,fx,yovl,gaussian
      real*8  xpro(NC),xmax(NC),xmin(NC),dx(NC)
      dimension x1(NC),x2(NC),p1(NC),p2(NC),g1(NC),g2(NC)
      dimension a(NC),b(NC),c(NC),d(NC),e(NC),f(NC)
      dimension aa(NC),bb(NC),cc(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
      dimension a2(NC),b2(NC),c2(NC),d2(NC),e2(NC),f2(NC)

      integer jn
      real*8 conv

      complex*16 coefAs1(nc,nc),coefBs1(nc),coefCs1
      complex*16 coefAs2(nc,nc),coefBs2(nc),coefCs2
      complex*16 coefAc(nc,nc),coefBc(nc),coefCc

      complex*16 coefA1(nc,nc),coefB1(nc),coefC1
      complex*16 coefA2(nc,nc),coefB2(nc),coefC2

      complex*16 caa1(nc,nc),cbb1(nc),ccc1
      complex*16 caa2(nc,nc),cbb2(nc),ccc2
      integer dim,incx,incy,info,IPIV(nc),ifail
      character*1 trans
      complex*16 zdotu,y(nc),ia(nc,nc),F06GAF
      complex*16 overlap1,overlap2,wkspacei(nc),alpha,beta
      real*8 detr,deti,wkspace(nc)
      integer IPIVOT(nc),job
      complex work(nc),det(2),sia(nc,nc)
c
      dtp1=-dtp
      pi=dacos(-1.0d0)
      eye=(0.0,1.0)
c
      coefCs1=0.0
      coefBs1(1)=0.0
      coefAs1(1,1)=0.5
c
      coefC1=-eye*dtv*coefCs1
      do i=1,nc
         coefB1(i)=-eye*dtv*coefBs1(i)
         do j=1,nc
            coefA1(i,j)=-eye*dtv*coefAs1(i,j)
         enddo
      enddo
c
      ccc1=coefC1
      N1=1.0
      N2=1.0
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))

         a2(in)=dreal(g2(in))
         b2(in)=dimag(g2(in))
         c2(in)=dreal(x2(in))
         d2(in)=dimag(x2(in))
         e2(in)=dreal(p2(in))
         f2(in)=dimag(p2(in))
c
c     Normalization constants
c
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
     &        *sqrt(mm(in)/(mm(in)+eye*dtp1*g1(in)))
         N2=N2*(a2(in)/pi)**0.25*exp(-0.5*a2(in)*d2(in)**2
     &        -d2(in)*e2(in)-(b2(in)*d2(in)+f2(in))**2/2.0/a2(in))
     &        *sqrt(mm(in)/(mm(in)+eye*dtp*g2(in)))
c
c     Integrand=N2*exp(aa2*x^2+cc2*x+cc2)*conjg(N1*exp(aa1*x^2+cc1*x+cc1))
c             *exp(ccc1+cbb1*x+caa1*x^2)
c
         den=2.0+2.0*eye*dtp*g2(in)/mm(in)
         aa2=-g2(in)/den
         bb2=(2.0*eye*p2(in)+2.0*g2(in)*x2(in))/den
         cc2=(p2(in)-eye*g2(in)*x2(in))**2/g2(in)/den
     &        -p2(in)**2/2.0/g2(in)

         den=2.0+2.0*eye*dtp1*g1(in)/mm(in)
         aa1=-g1(in)/den
         bb1=(2.0*eye*p1(in)+2.0*g1(in)*x1(in))/den
```

```
          cc1=(p1(in)-eye*g1(in)*x1(in))**2/g1(in)/den
     &         -p1(in)**2/2.0/g1(in)

          ccc1=ccc1+dconjg(cc1)+cc2
          cbb1(in)=dconjg(bb1)+bb2+coefB1(in)
          do jn=1,nc
             if (in.eq.jn) then
                caa1(in,jn)=dconjg(aa1)+aa2+coefA1(in,jn)
             else
                caa1(in,jn)=coefA1(in,jn)
             endif
          enddo
       enddo
       dim=nc
       do i=1,nc
          y(i)=0.0
          cbb1(i)=-cbb1(i)
          do j=1,nc
             caa1(i,j)=-caa1(i,j)
             ia(i,j)=caa1(i,j)
             sia(i,j)=ia(i,j)
          enddo
       enddo
c
c      NAG subroutines
c
c      call F03ADF(caa1,dim,dim,detr,deti,wkspace,ifail)
c      overlap1=dsqrt(pi**dim)/cdsqrt(detr+eye*deti)
       job=11
c
c      SGI subroutines to compute the terminant
c
       call CGEFA(sIA,dim,dim,IPIVOT,INFO)
       CALL CGEdi(sIA, dim, dim, IPIVOT, DET, WORK, JOB)
       overlap1=dsqrt(pi**dim)/sqrt(det(1)*10.0**det(2))
c
c      call F07ARF(dim,dim,IA,dim,IPIV,info)
       call zgetrf(dim,dim,IA,dim,IPIV,info)
c      call F07AWF(dim,IA,dim,IPIV,wkspacei,dim,info)
       call zgetri(dim,IA,dim,IPIV,wkspacei,dim,info)
c
       trans='N'
       alpha=1.0d0
       beta=0.0d0
       do i=1,dim
          y(i)=0.0d0
       enddo
       incx=1     !  Matrix multiplication for exponent
       incy=1
c      call F06SAF(trans,dim,dim,alpha,IA,dim,cbb1,incx,beta,y,incy)
c      overlap1=overlap1*cdexp(F06GAF(dim,cbb1,incx,y,incy)/4.0d0+ccc1)
       call zgemv(trans,dim,dim,alpha,IA,dim,cbb1,incx,beta,y,incy)
       overlap1=dconjg(N1)*N2
     $      *overlap1*cdexp(zdotu(dim,cbb1,incx,y,incy)/4.0d0+ccc1)
       gvgovl=overlap1
       RETURN
       END
C-----------------------------------------------------------------------
       subroutine overlap_ggovlc(r1,p1,g1,r2,p2,g2,govl)
c
c      calculate <r1,p1,g1|r2,p2,g2> analytically, the parameters
c      are complex numbers
c
       implicit none
```

```
      integer in
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      real*8 pi,a1,b1,c1,d1,e1,f1,a2,b2,c2,d2,e2,f2
      complex*16 r1,r2,p1,p2,g1,g2
      complex*16 N1,N2,aa1,bb1,cc1,aa2,bb2,cc2,aa,bb,cc
      complex*16 phi22,eye,govl
      dimension r1(NC),r2(NC),p1(NC),p2(NC),g1(NC),g2(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
      dimension a2(NC),b2(NC),c2(NC),d2(NC),e2(NC),f2(NC)
c
      eye=(0.0,1.0)
      pi=3.141592654
      N1=1.0
      N2=1.0
      phi22=1.0
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(r1(in))
         d1(in)=dimag(r1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
         a2(in)=dreal(g2(in))
         b2(in)=dimag(g2(in))
         c2(in)=dreal(r2(in))
         d2(in)=dimag(r2(in))
         e2(in)=dreal(p2(in))
         f2(in)=dimag(p2(in))
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
         N2=N2*(a2(in)/pi)**0.25*exp(-0.5*a2(in)*d2(in)**2
     &        -d2(in)*e2(in)-(b2(in)*d2(in)+f2(in))**2/2.0/a2(in))
         aa1=-0.5*g1(in)
         bb1=g1(in)*r1(in)+eye*p1(in)
         cc1=-0.5*g1(in)*r1(in)**2-eye*p1(in)*r1(in)
         aa2=-0.5*g2(in)
         bb2=g2(in)*r2(in)+eye*p2(in)
         cc2=-0.5*g2(in)*r2(in)**2-eye*p2(in)*r2(in)
         aa=conjg(aa1)+aa2
         bb=conjg(bb1)+bb2
         cc=conjg(cc1)+cc2
         phi22=phi22*exp(-bb**2/4.0/aa+cc)
         phi22=phi22*sqrt(-pi/aa)
         if (dreal(aa).gt.0.0) then
            print *,"r1=",r1(in)
            print *,"r1=",p1(in)
            print *,"r1=",g1(in)
            print *,"r2=",r2(in)
            print *,"p2=",p2(in)
            print *,"g2=",g2(in)
            print *,"aa=",aa
            print *,"error"
            stop
         endif
      enddo
      phi22=phi22*conjg(N1)*N2
      if (abs(phi22).gt.1.0E20) phi22=0.0
      if (abs(phi22).lt.1.0E-20) phi22=0.0
      govl=phi22
      return
      end
C-----------------------------------------------------------------------------
      FUNCTION gaussian(x,x1,p1,g1)
```

```
c
c     Gaussian basis fucntion
c
      IMPLICIT NONE
      INTEGER in
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)

      REAL*8 x
      real*8 pi,a1,b1,c1,d1,e1,f1
      complex*16 x1,p1,g1
      COMPLEX*16 EYE,GAU,gaussian,N1
      DIMENSION x(NC),x1(NC),p1(NC),g1(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
c
      pi=3.141592654
      EYE=(0.0,1.0)
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
      enddo
c
      N1=1.0
      do in=1,NC
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
      enddo
c
      GAU=1.0
      DO in=1,NC
         GAU=GAU*EXP(-0.5*g1(in)*(x(in)-x1(in))**2
     &        +EYE*p1(in)*(x(in)-x1(in)))
      END DO
      GAU=GAU*N1
c
      if (abs(gau).gt.1.0E20) gau=0.0
      if (abs(gau).lt.1.0E-20) gau=0.0
      gaussian=gau
c
      RETURN
      END
C-----------------------------------------------------------------------
      FUNCTION gaussian_type2(x,x1,p1,g1,dt,m)
c
c     Gaussian basis function operated by the kinetic operator
c
      IMPLICIT NONE
      INTEGER in
      integer NC,nta,NPT
      PARAMETER(NC=1,NPT=4,nta=100)
      REAL*8 x,pi,m,dt
      real*8 a1,b1,c1,d1,e1,f1
      complex*16 x1,p1,g1
      COMPLEX*16 EYE,GAU2,rnum,rden,gaussian_type2,N1
      DIMENSION x(NC),x1(NC),p1(NC),g1(NC),m(NC)
      dimension a1(NC),b1(NC),c1(NC),d1(NC),e1(NC),f1(NC)
c
      pi=3.141592654
      EYE=(0.0,1.0)
```

```
c
      do in=1,NC
         a1(in)=dreal(g1(in))
         b1(in)=dimag(g1(in))
         c1(in)=dreal(x1(in))
         d1(in)=dimag(x1(in))
         e1(in)=dreal(p1(in))
         f1(in)=dimag(p1(in))
      enddo
c
      N1=1.0
      do in=1,NC
         N1=N1*(a1(in)/pi)**0.25*exp(-0.5*a1(in)*d1(in)**2
     &        -d1(in)*e1(in)-(b1(in)*d1(in)+f1(in))**2/2.0/a1(in))
      enddo
c
      GAU2=1.0
      DO in=1,NC
         rnum=p1(in)/g1(in)-EYE*(x1(in)-x(in))
         rden=2.0/g1(in)+EYE*2.0*dt/m(in)
         GAU2=GAU2*EXP(rnum**2/rden-p1(in)**2/(2.0*g1(in)))
     &        *sqrt(m(in)/(m(in)+eye*g1(in)*dt))
      END DO
      GAU2=GAU2*N1
      if (abs(gau2).gt.1.0E20) gau2=0.0
      if (abs(gau2).lt.1.0E-20) gau2=0.0
      gaussian_type2=gau2
      RETURN
      END
c-----------------------------------------------------------------------
```

Problem 15:

The solution to this problem can be obtained from
http://ursula.chem.yale.edu/∼batista/classes/summer/P15/P15.tar
and requires the math libraries from
http://ursula.chem.yale.edu/∼batista/classes/summer/m.tar.
Untar both files by typing

```
tar -xvf P15.tar
```

and

```
tar -xvf m.tar
```

Type

```
cd P15
```

By typing

```
ls
```

you will see that the problem is solved in terms of MP/SOFT and SOFT simulations in 1d and 4d, allowing for direct comparisons between grid-based calculations and MP/SOFT. In addition, the problem is solved in 24d according to the MP/SOFT method.

To start, type

```
cd P15_1dg
```

Compile the grid-based 1d-version of the program by typing

```
comp_15_1dg
```

Run the program by typing

```
Problem15_g
```

Compute the spectrum by compiling the program calcspec.f by typing

```
./comp_calcspec
```

and running the program by typing

```
./calcspecs
```

Visualize the photoabsorption spectrum by typing

```
gnuplot<peV
```

or

```
gnuplot< pw
```

Analogously, compile the MP/SOFT version of the program in the directory P15_1dmp, with the corresponding script by typing

```
  comp_15_1d
```

and run it by typing

```
  poe Problem15 -procs 6
```

The output will be the autocorrelation function as a function of time saved in file named autocorr. Results can be compared to reference calculations, stored in file named autocorr_ref.

The photoabsorption spectrum can be obtained by compiling calcspec.f by typing

```
comp_calcspecs
```

and running it by typing

```
calcspec
```

In order to visualize the spectrum, type

```
gnuplot <peV
```

or

```
gnuplot <peV
```

Simulations for the 4-dimensional and 24-d model Hamiltonians can be performed analogously. However, for the current implementation, the 24-d simulations required forward and backward propagation in order to obtain the correlation function as $C(2t) = \langle\Psi(-t)|\Psi(t)\rangle$.

Problem 16:

The solution to this problem can be obtained from
http://ursula.chem.yale.edu/∼batista/classes/summer/P16/P16.tar
Instructions for compiling and running can be obtained upon request.