Chemistry 126: Computers in Chemistry.

Daniel Neuhauser

Introduction

There are many uses of computers in chemistry. Broadly, they are divided to the following categories and sub-categories.

Programming:

- **Time-dependent differential equations**, in which the initial conditions are known, and one wants to know how they evolve forward in time: these include:
  - ❖ Chemical Reactions (evolution of concentrations with time)
  - ❖ Classical dynamics (evolution of atomic positions with time)
  - ❖ Time-dependent Quantum dynamics (atoms under lasers)

- **Time-independent differential equation:** primarily used for the (time-independent) Schroedinger equation, which is solved in two main fields of chemical interest:
  - ❖ Structure of molecules and their energies (electronic structure, or ab-initio).
  - ❖ Vibrational frequencies

- **Random variables and stochastic processes** (important for diffusion, noise, and similar processes which affects chemical reactions, among other places
- **Data-analysis**, including especially Fourier (and today Wavelet) transforms, 3-D data plotting algorithms, etc. A related area is:
- **Bioinformatic tools** are emerging for describing DNA genomics, etc.

Most if not all of these fields have extensive program suites that are quite user friendly. This is especially true for electronic structure, where suites like “Gaussian”, “Spartan”, and several others can give for many compounds chemical accuracy for stationary structures, predicting what is the structure that a specific organic (and increasingly inorganic) species would adopt.

A second class that you may encounter is of programs for biochemical description of enzymes, proteins, and their evolution in time.

In this course we’ll try to give you the tools needed to program much of these problems. We will concentrate on **building our own programs**, to give you the experience necessary when you go to grad. School or industry; to help you I have built much of the

programs so that you will, with careful choice, need **just to do limited modifications** to the existing programs.

Useful References:

1) Numerical Recipes. The Art of Scientific Computing. W. H. Press, S. A. Teukolsky, B. P. Flannery, W. T. Vettering, 1986 Cambridge University Press, Cambridge.) (This is the **bible of computing** – it has a Fortran version and an additional Fortran-90 version. You should use it mostly for the explanations, less for the actual programs). You can also download it as:

http://lib-www.lanl.gov/numerical/index.html

2) Computational Physics : Fortran Version
by Steven E. Koonin, Dawn C. Meredith

Science oriented; More advanced than our course, but similar spirit. You should read it too if youre serious about computing.

3) Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,
by Milton Abramowitz (Editor), Irene A. Stegun (Editor).

Old fashioned but has some useful information.

4) For programs, try finding IMSL libraries.

5) And of course, try www.google.com

Useful Information:

(1) Conversion units: see also:

http://physics.nist.gov/cuu/Constants/

For masses: http://www.webelements.com/ (click on an element and then search for "isotopes")

1a.u.(distance)=1bohr radius=0.529177 x Angstrom

1a.u.(energy) =1Hartree= 27.2116 eV (=2 Rydbergs).

1eV=8065.5 wavenumbers*hc

1 a.m.u. (atomic mass unit) = amu = 1822.89 me (me=mass of electron=1 in a.u.),so in brief 1amu=1822.89 in atomic units

Also, for reference, 1amu=1.660 538 73 x 10^{-27} kg

mass(^{12}C) = 12 amu (exactly)

mass(^{1}H)= 1.007825 amu

mass(^{4}He)= 4.00260 amu

mass(^{40}Ar)= 39.9624amu

mass(^{16}O)= 15.9949amu

mass(^{14}N)= 14.00307amu

(2) Lennard-Jones parameters (for vdW interacting molecules). Taken from:

http://www.scf.fundp.ac.be/~jpvigner/thermodynamique/page22.html.

$$V_{LJ}(r) = \varepsilon\left[\left(\sigma/r\right)^{12} - 2\left(\sigma/r\right)^{6}\right]$$

Atome	Paramètre σ(angström)	Paramètre ε(meV)
He	2.87	0.875
Ne	3.075	3.125
Ar	3.82	10.4
Kr	4.10	14.1
Xe	4.47	20.0

Remember to convert this to a.u. in your program! THESE NUMBERS cannot be used directly before conversion!

Derivatives and Differential Equations

(remember to also read the notes by Prof. Jonsson)

Derivatives:

The first set you'll see would introduce you to f90 programming, and to see how one can calculate numerical derivatives. You'll study the simplest expression for a numerical derivative:

df/dx ~ [f(x+dx)-f(x)]/dx

and see that numerically dx needs to be small, but not too small (to avoid round-off errors). Further, you'll compare it to more accurate expressions:

df/dx ~ [f(x+dx)-f(x-dx)]/2/dx.

Remember to read the first assignment of Prof. Jonsson.

Assignment: Numerical Derivatives

First, copy all the FORTRAN files we'll use to your directory.

cp -r ~/../dxn/126/126_fortran ~/126_fortran

Also, copy my .cshrc file, which contains aliases, into your program:

cp ~/../dxn/.cshrc ~
source ~/.cshrc

Next go to the *numerical_derivatives* directory.

cd
cd 126_fortran
cd numerical_derivatives

There, first *open and read* n*1.f90

emacs num_drv_prt1.f90 (or simply: **emacs n*1.f90**)

Then *compile:*

cf n*1.f90

(cf is an abbreviation for

lf95 -c --chkglobal -M ~/126_fortran/library

the Lahey FORTRAN compiler with flags to produce .o files , check in detail the program when running it, producing .o files, and linking to .mod files in the library directory)

Then link the files:

fl n*1.f90

where: fl is an alias for

lf95 --chkglobal -M ~/126_fortran/library ~/12*n/library/*.o

Finally **run**

./a.out

And interpret the results.

Repeat the same for n*2.f90, n*3.f90 and n*4.f90.

Time-Dependent Propagation (Ordinary Differential Equations).

The prevalent problem in much that we'll do is to propagate a differential equation, specifically:

$$d\mathbf{Q}/dt = \mathbf{F}(\mathbf{Q})$$

where bold-face is used (here) for a vector.

Chemical Kinetics:

As an example, which will be used in the set on chemical reactions, we'll consider 4 species, A,B,C,D , where A is the reactant and D the final product, where the reactions are:

A ←→B (forward rate const. kab, backward kba)
B ←→2C (forward kbc, backward kcb)
C ←→D (forward only kcd).

The relevant chemical reactions are (since these are **elementary** reactions)

$$dA/dt = -kab\ A + kba\ B$$ (I should've used [A], etc., but am too lazy)
$$dB/dt = +kab\ A - kba\ B - kbc\ B + kcb\ C^2$$
$$dC/dt = +2kbc\ B - 2kcb\ C^2 - kcd\ C$$
$$dD/dt = +kcd\ C$$

We can convert it to the vector form above, by identifying **Q** as a 4-component vectors with components

$Q_1 = A$,
$Q_2 = B$
$Q_3 = C$
$Q_4 = D$

While the components of **F** are

$$F_1 = -kab\ A + kba\ B,$$

etc.

OK, So this shows how we can convert a general chemical kinetics (and later on also many other problems) into a problem of the generic form:

$$d\mathbf{Q}/dt = \mathbf{F}(\mathbf{Q})$$

The next stage is to propagate this set of equation. By propagating we mean to get a rule, such that knowing **Q** at t, we can find it at t+dt. The simplest is Euler's propagation –

$$\mathbf{Q}(t+dt) = \mathbf{Q}(t)+\mathbf{F}(\mathbf{Q}(t))*dt$$

There are much better ways which, in many problems, are much more efficient and many times converge well with a large dt. Chief among these are Runge-Kutta algorithms, which is discussed below. At any rate, you generally wont have to code these algorithms – all you have to do is to give them your initial vector, your initial time and final desired time, and the name of a subroutine that calculates **F** given a **Q.** In the **chemical reactions set** you'll learn to use the 4'th-order-fixed-time-step Runge-Kutta algorithm, and compare it to Euler's algorithm.

Once you have a good numerical propagation algorithm, you can start to ask Physical and chemical questions, such as: can we employ the local-equilibrium approximation (in which the second of the 3 reactions above is in equilibrium always).

You will be asked to apply the local eq. approximation, i.e., reduce the 4 species problem into a 3-species problem (A,B, and D, where C is known from the local eq. Approximation, kbc B = kcb C^2), and compare the result.

You'll also be asked to check the steady state approximation, in which one approximates that B and C (the intermediates) have concentration that don't change with time, get dA/dt and dD/dt from that approximation, and solve for A and D. (the reactants and products).

Details of the assignment are in the actual FORTRAN programs for this set.

Appendix: Runge-Kutta Algorithm.

Runge-Kutta Algorithms are natural extensions of Euler's algorithm. They're discussed in numerical recipes, but their spirit is very simple.

The Euler algorithm has

$$\mathbf{Q}(t+dt) = \mathbf{Q}(t)+\mathbf{F}(t,\mathbf{Q}(t))*dt$$

(where I allow for explicit t dependence). The problem with this algorithm is that it uses **Q** at time t to predict the **F** term. This is not symmetric (t and t+dt are not treated symmetrically). A better algorithm is therefore to:

(1) **First** use **Q(**t) to find a reasoanble approximation to the value of the derivative at t+dt/2
Qmid= Q(t)+**F(** t,**Q**(t))*dt/2

(2) Now use **Qmid** for the propagation:
Q(t+dt)=**Q**(t)+**F(** t,**Qmid**)*dt

The combined approach is called 2'nd order Runge-Kutta Algorithm. (2'nd order since we dvide dt into 2.)

Exercise: code this approach youself and compre its accuracy to the Euler for our system. Also read about it in numerical recipes, chapter 16.1,16.2

Classical dynamics:

Classical Dynamics is simple, but useful for biochemical simulations. The equations are simply Newton's equation. For simplicity, consider first just a single particle in a position-dependent potential. Then we need to propagate forward its position **r=**(x,y,z) and velocity.

In the general scheme we learned, we will group these two things (the position and velocity, or the velocity and momentum, **p=**m**v**) into a single vector. We will review this year; but note that most simulations for classical dynamics use a different approach, called the Verlet algorithm, which is reviewed in the Jonsson's lectures and will be briefly reviewed later on.

Returning to the general (e.g., Runge-Kutta) propagator, we need to deal with a length-six vector,
defined as:

$Q_1=x$
$Q_2=y$
$Q_3=z$
$Q_4=p_x$
$Q_5=p_y$
$Q_6=p_z$

Sometimes this is abbreviated as **Q**=(**r,p**).

Newton's equations,

$d\mathbf{r}/dt = \mathbf{p}/m$
$d\mathbf{p}/dt = \mathbf{f(r)}$

where

f = -grad(V)

become

$d\mathbf{Q}/dt = \mathbf{F}$

where

$F_1= Q_4/m$ (i.e., p_x/m)
$F_2= Q_7/m$
$F_3= Q_6/m$
$F_4= f_x$=-partial V(x,y,z)/ partial x
$F_5=f_y$
$F_6=f_z$

Thus, we can use the same routine you used for the Chemical Kinetic equation, and just use it to simulate this system.

As an example, **consider the dynamics of a particle attached to the origin through a central force (**a force associated with a potential energy which depends only on r, the distance from the origin). This can be useful for describing the motion of the earth

around the sun, or of a particle attached to a much heavier particle weakly through a Lennard-Jones force, or a classical electron around a nucleus.

The only question is then how to calculate the force, e.g.,

$$f_x = -\frac{\partial V}{\partial x}$$

Here we can use the fact that V depends only on r, defined as

$$r = \sqrt{x^2 + y^2 + z^2}$$

and also that we can use the chain rule, to get:

$$f_x = -\frac{dV}{dr}\frac{\partial r}{\partial x}$$

where the partial derivative is evaluated for a fixed y-z. The partial derivative is:

$$\frac{\partial r}{\partial x} = \frac{1}{2\sqrt{x^2 + y^2 + z^2}} * 2x = \frac{x}{r}$$

Therefore,

$$f_x = -\frac{dV}{dr}\bullet\frac{x}{r}$$

and similarly,

$$f_y = -\frac{dV}{dr}\bullet\frac{y}{r}$$

$$f_z = -\frac{dV}{dr}\bullet\frac{z}{r}$$

We're ready then to the first project in classical dynamics.

Project 1 in Classical Dynamics

Task:

Consider a particle in a central force field (V=V(r)). In the first project we'll consider the Lennard-Jones force field:

$$V(r) = \varepsilon_0\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right)$$

(different sources use different parameterizations), where we introduce the parameters:

$$\varepsilon_0,$$

$$\sigma$$

relating to the well-depth, the typical rise of the potential, and the equilibrium point.

Parameters and values: To avoid unit problems, we'll use **atomic units**. **Note however that many potential routines are given in terms of Angstrom, so for those routines you'll need to rescale the distances (r(Angstrom) = r(a.u.)*0.529, see class).**

Reasonable parameters for covalently-bound molecules are

epsilon_0 ~ 0.2a.u. (~5.5eV) (0.005 a.u. for vdW molecules)
sigma ~ 1a.u. (=0.529 Angstrom).
mass : mass(hydrogen) **= 1836 a.u.**~10^{-27}kg)
time unit (a.u.)= **1a.u. =** 2.5*10^{-17}sec

We will use different values of epsilon_0 and sigma, but for covalently bound these are reasonable values.

(Assignment: get from physical chemistry books more accurate values for the conversions between other units – such as eV, femtosecond, AMU -- and atomic units!)

Your task is to follow the dynamics as a function of time for different initial conditions.

(1) Derive analytically the equilibrium distance (we'll call it "a") in terms of sigma.
(2) Verify it numerically (i.e., write a Force routine, with an input of x,y,z, and verify that when r=a, the force is zero, and it isn't zero otherwise.
(3) Write a classical dynamics routine
Your classical dynamic routine should be a simple **modification of the chemical kinetics routine –** with different inputs, and a different number of the vector of variables (6 rather then 4), and a different dvector_dt routine – but similar in structure.
As a check, ensure that the energy is conserved. **For more details on the** program planning, see below.

(4) Now start with a particle slightly away from equilibrium and along the x axis (at r(start)=r_0+a), and zero initial velocity, and plot x(t). Make sure your plotting time is long enough that the particle undergoes several vibrations.

(5) Now let the particle have an initial velocity in the **y** direction, which is large enough that the particle does a complete revolution in the time it takes it to do a single vibration (you'll need to play with the initial velocity). Plot x(t), y(t) and the trajectory (i.e., y vs. x) for that case. (Set z at zero initially,

Program planning:
You'll need to write the program based on the chemical kinetics program you wrote earlier. The changes are small. The program needs have the following component (most of which you already have):

- Parameter fixing – read from a separate file the particle mass (M), as well as D,a, and r_0. For this project, we'll consider Ar2 and CH.

  The relevant mass for Ar2 is the reduced mass which is HALF the mass of Argon. Use atomic units (How much is the mass of Argon in atomic units – remember that the mass of hydrogen is 1836 in a.u.)

  You'll need to read from a separate text-book the values of these parameters.

  Read also dt and Nt.

- Initialization: read in (again from the data file) the initial position and velocity

- Propagation: your "vector" would have six components (3 for position, 3 for momentum). Propagate it by the Runge-Kutta. Of course, you'll need a new routine that given the vector **Q** of length 6 **(**i.e., **r** and **p)** calculates **F.** The equations for **F** are given above but are summarized here:

  x=Q1, y=Q2, z=Q3, r = sqrt(x**2+y**2+z**2)
  px=Q4,py=Q5,pz=Q6

  F_1= px/m
  F_2= py/m
  F_3= pz/m
  F_4= -x/r* dV_dr
  F5=-y/r* dV_dr
  F6=-z/r* dV_dr

  Where you'll supply a separate routine that given r calculates dV_dr.

- While propagating, do the plotting of x(t), y(t), and the trajectory y(x). (If z and pz are zero initially, they remain so at all times – but **check** this).

  Also: check by plotting the total energy, i.e., the **sum of** the kinetic energy (px**2+py**2+pz**2)/(2m), and the potential energy (V(r)), and ensuring that it is constant.

  Also: plot the distance as a function of time, r(t).

Program analysis:

You should run your program until r(t) oscillates several times. You'll notice that the motion is periodic. Read the period. Compare it (look in a physical chem. book) to the period of Ar2 and CH that the book quotes (the book may quote the transition frequency, hbar*w, so extract w by dividing by hbar).

Also, see what happens if you increase the initial displacement significantly from “a” (i.e., increase the amplitude of vibration). You should notice that the oscillations in r(t) get SLOWER with higher AMPLITUDE. I’ll explain this in class—but try to think why that should be!

Second classical-dynamics project – a collection of Ar

A collection of particles (e.g., Ar solid)

Here we'll extend what we learned to a collection of "N" particles that interact by 2-body potentials. The notes by Prof. Jonsson explain this, but I'll summarize and add details below.

Instead of having just one distance-vector like before, r, we now have to consider N particles. For each of these particles, denoted by "i" (where i=1,2,…,N), we have its x-position (x_i, or in the program x(i)), y(i),z(i),px(i), py(i), and pz(i). So altogether we have 6*N variables.

[You may wonder what happens if N=2. Here we say that we need N*6=12 variables; but before we had just 6 variables – the reason is that before the variable we considered was the **distance vector (the difference in position vectors** between the two atom), and now we consider the actual **positions** of the **two** atoms]

Newton's equation would have the symbolic (FORTRAN-ready!) form

i=1, Natoms
dx_dt(i) = px(i) / m (where now "m" would be the true mass of the Ar atoms, not half)
dy_dt(i) = py(i) / m
dz_dt(i) = pz(i) / m
dpx_dt(i) = force_x(i)
dpy_dt(i) = force_y(i)
dpz_dt(i) = force_z(i).

Now we just need force_x(i), i.e., the force in direction x on atom i, defined as:
force_x(i) = -partial V/ partial x(i).

The force we consider is pair-additive. Many of the forces you'll consider are pair-additive, although sometimes you'll need 3-body forces, etc.

The force is

$$V = \frac{1}{2}\Sigma_{jk} v(r_{jk})$$

where $v(r_{jk})$ is a two-body force (e.g., the Lennard-Jones), j and k are not equal, r_{jk} is the distance between atoms j and k, and each runs over 1 to N. (The ½ accounts for double counting, e.g., j=2 and k=3, and j=3 and k=2).

To calculate the force, let's take a specific example, say N=4, and we want the force on the second atom labeled as "i=2".
Then,
$V = v(r_{12})+v(r_{13})+\ldots+v(r_{34})$.

and

force_x(2) (force in direction x on atom 2) = $-\frac{\partial V}{\partial x_2}$

To calculate that force (on i=2) we need to consider how the potential changes when atom "2" moves. Clearly the only terms that would change would be $v(r_{21})$, $v(r_{23})$ and $v(r_{24})$, i.e., the sum over all "k" (except for k=i) of v(r_ik).

so in general:

force_x(atom i)= [partial v(r_ik) / partial x_i]

(where k is summed from 1 to Natoms, but is not equal to i).
(Also: I use _ interchangeably with subscripts).

As far as the derivative of $v(r_{ik})$ w.r.t. a change in $x_{i,}$ we'll use again the chain rule to get

$$-\frac{\partial v(r_{ik})}{\partial x_i} = -\frac{\partial v(r_{ik})}{\partial r_{ik}} * \frac{\partial r_{ik}}{\partial x_i}$$

The first term is evaluated by the same force subroutine we have already; the second term is obtained as

$$\frac{\partial r_{ik}}{\partial x_i} = \frac{\partial \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2 + (z_i - z_k)^2}}{\partial x_i} = \frac{1}{2r_{ik}} 2(x_i - x_k) = \frac{x_i - x_k}{r_{ik}}$$

Similarly to the expression we had in the previous project.

OK, so to summarize : the crucial subroutine in the project is summarized below:

```
subroutine sub_dQ_dt(Q, dQ_dt, time)
use parameters_module                ! in this module you'll store information such
                                     ! as the number of atoms, and the mass.  At that
                                     ! module define
                                     ! N(=Natoms) as a parameter so you can
                                     ! very simply define things like arrays, x(N), etc.
implicit none
real*8 Q(6*N), dQ_dt(6*N)
real*8 x(N), y(N), z(N), px(N), py(N),pz(N)
real*8 fx(N), fy(N), fz(N)
real*8 dV_dr

integer i,k, icounter

! we need to get the x's, as follows:
```

```
! x(1) =Q(1), y(1)=Q(2),z(1)=Q(3), px(1)=Q(4),...,pz(1)=Q(6),
! x(2)=Q(7), etc.

do i=1,N
        icounter = 6*(i-1)      ! when i=1, icounter=0; i=2-->icounter=6,
        x(i) = Q(icounter+1)   ! etc.; icounter+1 is the first point in memory associated
        y(i) = Q(icounter+2)   ! with atom i.
        z(i) = Q(icounter+3)
        px(i) = Q(icounter+4)
        py(i) = Q(icounter+5)
        pz(i) = Q(icounter+6)
enddo

do i=1, N
        fx(i) = 0
        fy(i) = 0
        fz(i) = 0
        do k=1,N
                if(k /= i)  then
                        r = sqrt( (x(i)-x(k))**2 + (y(i)-y(k))**2 + (z(i)-z(k))**2 )
                        call sub_dV_dr( r, dV_dr)

                        fx(i)  = fx(i) + (-dV_dr)*(x(i)-x(k))/r
                        fy(i)  = fy(i) + (-dV_dr)*(y(i)-y(k))/r
                        fz(i)  = fz(i) +  (-dV_dr)*(z(i)-z(k))/r
                endif
        enddo
enddo

do i=1, N
        icounter = 6*(i-1)
        dQ_dt(icounter+1)   = px(i)/mass
        similarly for y and z...
        dQ_dt(icounter+4) = fx(i)
        similarly for fy and fz..
enddo

!---------------------------------------------------------------
```

The subroutine above will be the main ingredient in the program you will write. You'll need to pass it to the Runge-Kutta. The overall program structure would be:

A module containing N, mass, Lennard-Jones parameters, dt, Nt,etc.
Program:
        Read parameters from the data file.

Initialize the vectors x,y,z,px,py,pz (by reading from the data file).
Propagate, and plot the x's.

Tasks:
(1)
use your program to test the propagation of a wave. Specifically, start with a set of Ar atoms, all one a line. . Put N=10 particles, such that particle 1 is at the origin, 2 is at the equilibrium distance, 3 is spaced from 2 by the same distance, etc.

1-----2-----3-----4---etc.

But stretch 10 (away from 9) a little bit outward (by 20%). Then plot on a file the 10 x's (x(1),...,x(10) at each time step.

Assume that the potential for Ar atoms on the line is

$$V(\mathbf{r}_1,\mathbf{r}_2,...,\mathbf{r}_N)=\sum_{j=1}^{N-1} v(|\mathbf{r}_{i+1}-\mathbf{r}_i|)$$

where

$v(r)=$ Lennard-Jones,

(see table at the begining).

I'll give you a separate 2-D contour plotting routine that would allow you to view propagation of the wave as a function of time.

(2)
As an added exercise, try increasing the initial displacement of atom 10 away from equilibrium, and see what happens to the wave – is it propagating faster or slower? We'll talk about it in class.

(3) The third task would be to use the FFT routine (you'll learn about next) to find the periodicity of the motion (if it has any). For this, start with two particles only (a diatom). Initially, place the particles, at t=0, slightly away from the equilibrium distance (and with zero momentum). Plot the velocity of the first (in the x direction of course) as a function of time, and FFT it – you'll get a peak at the associated frequency of the motion (we'll derive that later). Now start increasing the initial amplitude and show that the frequency **decreases** and that **you get harmonics**. We'll discuss this later.

!----------

Fast Fourier Transforms

(i) Fourier-Transforms and integration limits:

Fourier transforms (**FT** henceforth) are a very important tool in dynamics. In principle they are quite simple, but in practice this could be a problem. So I'll spend a lot of space on this.

Say we are given a signal: $C(t)$. The definition of the Fourier transform is then:

$$F(w) = \text{integral } C(t) \exp(-iwt)\, dt$$

where i is the complex "i", and some definitions have "iwt", instead of "-iwt" in the exponent. The (quite) tricky points are what are the integration limits, and what values of w should we consider.

There are two options. Analytically, one usually puts *–infinity to infinity*. But on the computer we don't have *–infinity or infinity*, so we put a fixed range. This is OK if the function is vanishingly small initially and finally, but many times it is not. (See figures in class).

Another option analytically is to consider functions *C(t)* that are periodic, so that for all times

$$C(t) = C(t+T)$$

In that case we can expand the wavefunction as:

$$C(t) = \sum_{w} a_j \cos(w_j t) + b_j \sin(w_j t)$$

where w_j denotes all frequencies such that $cos(w_j t)$ and $sin(w_j t)$ is periodic – i.e.,

$$cos(w_j(t+T)) = cos(w_j t)$$

i.e.,

$$w_j = \frac{2\pi}{T} j$$

And j would be positive (for the $sin(w_j\, t)$ term), and positive and zero (for the $cos(w_j\, t)$) term. This is the form you probably saw in your algebra/calculus courses.

An alternate form for periodic functions is

$$C(t) = \sum_{j=-\infty}^{\infty} F_j e^{iw_j t}$$

where now, as noted, j can be <u>both negative and positive</u>. The relation between F_j and a_j,b_j, is obtained by equating the relevant term. For example, say we consider the "j=3" term in the sin, cos term of the sum:

$$a_3\cos(w_3 t)+b_3\sin(w_3 t)$$

In the "exp" form, it needs to be equal to

$$F_3 \exp(iw_3 t)+F_{-3}\exp(iw_{-3} t)$$

But since $w_{-3} = -w_3$, we get (replacing here w3 by w):

$$F_3 \exp(iwt) + F_{-3}\exp(-iwt) =$$
$$\mathbf{F_3(\cos(wt)+i*\sin(wt)) + F_{-3}(\cos(wt)-i*\sin(wt))}$$

so:

$$a_3 = F_3+F_{-3}$$
$$b_3=i(F_3-F_{-3}).$$

So to summarize, the expression we have is:

$$C(t) = \sum_{j=-\infty}^{\infty} F_j e^{iw_j t}$$

$$w_j = \frac{2\pi}{T} j$$

Again, this expression is true for any periodic function. To get F_j we need to integrate:

$$F_j = \frac{1}{T}\int_0^T C(t)e^{-iw_j t}dt$$

Discrete Data.

The formulas we had so far were for a general continuous (but periodic) function. But the computer would have a finite set of "N" data points, $C(t_k)$ at t=0,dt,2*dt,...,(N-1)*dt (i.e., t_k=(k-1)*dt). We can then represent $C(t_k)$ still by the same expression

$$C(t_k) = \sum_{j=-\infty}^{\infty} F_j e^{iw_j t_k}$$

$$w_j = \frac{2\pi}{T} j$$

but now six questions emerge.

- First, what is T?
- Next, what values of j are allowed?
- How do we get F_j exactly now that we have a discrete data?

- What happens if the data is not given from 0 to T but for other time ranges?
- How do we handle "half-transforms", i.e., Fourier-transforms of data that's defined from 0 to infinity.

- And finally, how expensive it is to get the F's from the C's and vice-versa.

We'll answer them one by one.

First, what is *T*? For example, say we have 20 data points, $C_{1,}...,C_{20}$, corresponding to the values of the function at t=0,dt,...19*dt. Naively, we may think that T would be the last time (19*dt). But that would be incorrect. The wavefunction needs to be PERIODIC with a periodicity of T. Therefore, T would be the "next" data point, not on the grid we have, such that, at that point, the function would have the same value as C_0 (refer to the figure in class). That point would be at T=20*dt.

So: in conclusion: **T=N*dt**, where N is the number of data points

Second, **what values of "j" are needed**? Well it is clear that there are N independent (possibly complex) values of C_j. So there should be N independent values of F_j. Thus, "j" should cover "N" values. Naively we may think that we need j to go from –N/2 to N/2 --- the problem is that then j would have N+1 values (e.g., if N=20, then –N/2 to N/2 would be –10 to 10 – and there would be 21 such values). But we are saved by noting that <u>on the grid points, the j=-N/2 and j=N/2 exponential terms would give identical contributions</u> since:

For j=N/2, w_j=j*2*pi/T = N/2 *****2*pi/(N*dt) = pi/dt

and since t_k=(k-1)*dt

$\exp(iw_jt_k) = \exp(i*pi/dt * (k-1)*dt) = \exp(i*pi*(k-1)) =$

exp(i*0),exp(i*pi),exp(i*2*pi),exp(i*3*pi),... =

1, -1, 1, -1,...

It is easy to show that the exponent with j=-N/2 give the same contribution. Therefore, we can just as well take one or the other (but not both) – typically one starts with j=-N/2 and does not take the j=N/2 term, thus concluding with:

j=-N/2, -N/2+1, -N/2+2,..., -1, 0, 1,...,N/2-1
w_j=j*2*pi/N

In practice we will relabel so that j start from 1 and goes to N, and
w_j=wmin+(j-1)*dw, j=1,...,N
wmin = -N/2*dw

dw = 2*pi/T

The **third question is how to handle data sets where the initial time is not 0 but another (t_{min}) so that the time-frame is not [0,T] but [t_{min}, t_{max}=T+t_{min}]** (and, as mentioned, the data would be then given at [t_{min},t_{min}+dt,t_{min}+2*dt,...,t_{max}-dt].) This is very important if we have a function that decays as t goes to +-infinity, so that we typically represent it on a range [-T/2,T/2].

To get the expressions with a general t_{min} we only need to remember that then

t_k=tmin+(k-1)*dt, k=1,...,N

With this, the expression for C(t) becomes:

$$C(t_k) = \sum_{j=1}^{N} F_j e^{iw_j t_k}$$

$$= \sum_{j=1}^{N} F_j \exp\left(i\left(t_{\min} + (k-1)*dt\right)\left(w_{\min} + (j-1)dw\right)\right)$$

$$= \exp(it_{\min} w_{\min}) \exp(i(k-1)dt\, w_{\min}) \sum_{j=j\min}^{j\max} \bar{F}_j \exp\left(i(k-1)(j-1)dtdw\right)$$

where

$$\bar{F}_j = F_j \exp(i(j-1)dw\, t_{\min})$$

and :

$$T = Ndt$$

$$w_j = j\, dw$$

$$dw = 2\pi / T$$

$$w_{\min} = -\pi / dt = -dw * N / 2$$

The fourth question is how to get expression for the F_j. Here, again, in principle we could have given two different answers – but luckily they'll turn out to be the same!

One approach would be to view the relation between the F's and the C's as a matrix-vector multiplication:

$C_k = \Sigma_k M_{kj} F_j$

or in matrix form:

F= M C

(where **F** and **C** are vectors, **M** is an NxN matrix). Multiplying from the left by $\mathbf{M^{-1}}$ we get

C= $\mathbf{M^{-1}}$ F

i.e.,

$$\boxed{F_j = \Sigma_k (M^{-1})_{jk} C_k.}$$

The other alternative is to take the expression for F_j that we had before (in the analytic case) and try discritize that, i.e., write

$$F_j(=F(w_j)) = \frac{1}{T}\sum_{k=1,\ldots,N} C(t_k)\exp(-iw_j t_k)dt, \qquad t_k = (k-1)*dt$$

which becomes:

$$F_j = \frac{1}{T}\int_{t\min}^{t\max} C(t)e^{-iw_j t}dt \rightarrow$$

$$F_j = \frac{dt}{T}\sum_k C(t_k)\exp(-iw_j t_k)$$

i.e.,

$$F_j = \frac{1}{N}\sum_k C(t_k)\exp(-iw_j t_k)$$

where we used dt/T= 1/N.

Ok, so now we have two possible expressions for F_j (in frames). One that views the transformation as a discrete transformation of one vector to another (i.e., multiplying by a matrix), and the other that starts from the analytic expression.

We could have expected that the two expressions would only agree in the limit dt-->0, but LUCKILY, they always agree, no matter how small N is or how big dt is!. Thus, we can use the analytic expression always. (This is a particular property of Fourier transforms; it is not true if we were to use a representation of C(t) as, say, a sum of Bessel functions or a sum of Polynoms).

(Finally: Note how (except for the tmin factor) symmetric these form looks – to go from C to F or F to C we need almost the same transformation, except that the sign of the "i" needs to be changed, and we need to divide by 1/N.

The fifth question is how to handle data that's defined from 0 to infinity. This is very common in calculation of correlation functions, as we'll see later. There are various subpoints here:

- We usually want to find B(w)=integral_0^infinity C(t) exp(-iwt) dt, where C(t) vanishes at long times.

To find this, you'll think we simply need to calculate the FFT for a fixed range 0 to T, and multiply by T. But a small correction is needed – since the function is not periodic, the end point of the integral needs to be multiplied by ½; I.e.,

$$\int_0^T \exp(iwt)C(t)dt$$
$$\simeq \frac{C(t=0)}{2}dt + C(dt)*dt + C(2*dt)*dt + ... + C((N-1)*dt)*dt + \frac{C(N*dt)}{2}dt$$

and the last term can be thrown away if C((N*dt)=T) is small. The resulting sum is, up to a constant factor of T, exactly the same as the FFT – except that **you need to multiply the first element of C (C(t=0)) by ½ before you FFT it.**

The **sixth, final question is how fast the FT transformation can be done?** Naively, if we have N points, we need about N^2 operations. (sum over N values of t_k; a separate sum for each j). However, **it turns out that this can be done in about $N\log_2 N$ operations – almost linear in N!** (Compare N^2 to $N\log_2 N$ for N=1,000,000!). The algorithm has been discovered several times, starting with Gauss around 1800's, and finally rediscovered in 1965. It is the basis of modern data-processing. We wont discuss it but refer to "Numerical Recipes" for further discussion. (In one word – it relies on the fact that our matrix of transformation, M_{jk}, has elements that are related one to other in a very simple ways, since they are exponents).

Caution: The fast algorithm was originally derived for N that are powers of 2 ($N=2^m$), i.e., N=1,2,4,8,16,32,128,256,512,etc. There are extensions for N that are products of powers of 2 and powers of 3, 5 etc. – but the algorithm is more efficient for N's that are powers of 2 alone. **In practice,** if you have to use N which is not a power of 2, then, if your data set vanishes for high t, simply "pad" your array with zeroes.

Exercises:

(1) Our exercises would use the FFT in
library_fft_3.f90
(which in turns call a canned FFT routine).

Study that file. To access its routines, use the example subroutine in
library_fft_example*.f90
As stands, the example plots the FFT of sin(8t)+sin(7.2t), using N=2048 points.

(2) First, plot the results in the output file (fort.21, fort.22).
Don't change dt. How many point (N), and therefore, how long a time (T=dt*N) are necessary to clearly separate the peaks? Compare to the analytic results.
(3) Now fix T, and change dt (and therefore N). How small does dt have to be? explain.

it to the uncertainty principle
(4) Modify the example to get the Fourier transform of the Gaussian function f(t) = exp(-t^2/2 s^2).

Take tmin=-5*s, tmax=5*s (so that the function vanishes at the ends of the array). Make sure you multiply your numerical FFT transforms by T.

Compare to the analytical expression,
f(w)= sqrt(pi/s) exp(-w^2 s^2/2)

(note: use pi=dacos(-1.d0))

(5) Do numerically the FFT of the function
$\exp(iw_A t) + \exp(iw_B t)$

Show that if w_A and w_B are well-separated, then the FFT has two peaks; for a fixed w_A and w_B, as T increases, the peaks in the FFT become better separated. Note that the FFT wont be "clean" for a general w_A and w_B, but that's OK. Do several graphs to prove these specifically. Also: include on your graphs the real part of C(t), the imaginary part, and the absolute value.

(5) Do an HALF-FOURIER transform of the function
$C(t) = AA*\exp(-a*t)\cos(w_A t)+BB*\exp(-bt)\cos(w_B t)$, t>=0.
again, choose some reasonable values for the parameters; show how the FFT can separate the peaks if a and b are not too large (i.e., if the function does not decay rapidly). In addition, plot of course C(t).

Show the real and imaginary part. Note how they fall off differently. One would fall off rapidly with w, the other less so. The rapidly falling part is called the absorption part, and the other the dispersion part.

Remember to include the correction (multiply by half) that we mentioned before.

Notes: First, a use of Fourier-transforms we have not gone over is convolution, i.e., in many places you need to evaluate things of the form:
C(t)=integral B(t-z) A(z) dz

called **convolution**. It is easy to show that

C(w)=const.*B(w)A(w)

i.e., the Fourier transform of the convolution function equals the product of the Fourier transforms of A and B. See **Numerical Recipes** for applications.

We'll use Fourier transforms in later projects.

Matrices

I assume you dealt with matrix, as least basically, before, so I'll just summarize. Also, I'll just review the properties that are simple &/or we'll need, and leave out many important properties.

We'll generally deal with square (N*N) matrices, denoted **A,B,C,etc. (**capital roman bold-face), or with vectors (of length N), denoted as **v,u, etc.** Fortran-90 is very suitable for matrices, as we'll see later, although you have to be **very** careful with one or two points.

We generally will do 5 things with matrices (more things to do with complex matrices, but that's later).

- Multiply them, i.e.,

  A=B*C, i.e.,

  $A_{ij}=\text{Sum\_k } B_{ik}\, C_{kj}$

Example: say we deal with 2*2 matrices, and define:

A =

3.0000 -0.5000
-0.5000 3.0000

B =

0.7000 2.0000
2.0 0.7000

Then: **C=A*B =**

1.1000 5.6500
5.6500 1.1000

- Transpose them , i.e., given **A** define $\mathbf{A^T}$ such that $(A^T)_{ij}=A_{ji..}$

For example, in our case $\mathbf{A=A^T}$**;** also, if we define

D =

2.5000 0.3000
0 1.0000

Then

$\mathbf{D^T}$**=**

2.5000 0
0.3 1.0000

- Invert them, i.e., given **A** find a new matrix $\mathbf{A^{-1}}$ such that $\mathbf{A*A^{-1}=I}$, where **I** is the unity matrix, i.e., $I_{ij}=1$ if i=j, 0 otherwise. It is easy to verify that for the **A** defined above,

$\mathbf{A^{-1}}$ =

0.3429	0.0571
0.0571	0.3429

- Find their determinant det(**A**), sometimes denoted by |**A**|, and their trace (Trace(**A**)=**sum**_j Ajj ; in our case trace(**A**)=6. In the course we won't deal much, if at all with the determinant, but it is important to note that if it is zero, then **A** cannot be inverted.

- Finally, we can **diagonalize** them. Generally, we would want to diagonalize real matrices (for us real*8). [Later we may get to diagonalization of complex matrices, complex*16.] Further, we'll restrict ourselves to real and symmetric matrices ($\mathbf{A^T=A}$). An eigenvector **(v)** and eigenvalue (w) of the matrix **A** are defined as a vector and number which together fulfill:

$$\mathbf{A*v}=w*\mathbf{v}$$

Note that the left hand side is a vector (an N*N matrix multiplied by a vector, i.e., an N*1 matrix, is again a vector), and so obviously is the right hand. It turns out that for real and symmetric matrices we're guaranteed that they will have N different eigenvectors. (There are further interesting properties of the eigenvectors, but we wont deal with them now). We'll label them by an index, j, i.e., define the set of eigenvectors of a matrix **A** as vectors, $\mathbf{v}_j$.

Note an important point. Each $\mathbf{v}_j$ is a **vector**, not an element of a vector. The eigenvalue problem then takes the form

$$\mathbf{A*v_j}=w_j*\mathbf{v_j}$$

We also define a matrix, **V, the eigenvector matrix**. The eigenvectors are columns of this matrix.

We also define a diagonal matrix **D** such that the eigenvalues are the diagonal elements of this matrix, i.e., $D_{jj}=w_j$. Then, the equation above is equivalent to

$$\mathbf{A*V}=V*D$$

Proof: say we want to find the first column of the matrix on the left and the right. (The first column of a matrix **F** is simply the vector of numbers F_{j1}, where j=1,...,N) . So we need to find

$(\mathbf{A*V})_{j1}$=sum_k A_{jk} V_{k1} = $(\mathbf{A*v_1})_j$

where we used the fact that the elements of the first column of the matrix, V_{k1} are simply the components of the vector $\mathbf{v_1}$.

Similarly, we want to find the first column of the matrix on the left. It is

$\mathbf{(V*D)}_{j1}=V_{j1}*D_{11}$ (where we use the fact that **D** is diagonal)

$=(\mathbf{v_1})_j * w_1$

So the first column of the two sides are equal (since $\mathbf{A*v_1=v_1}*w_1$). The second, third etc. columns will also be equal, which finishes the proof.

Examples:

For the **A** we had above, the associated eigenvector matrix is

V=

-0.7071 0.7071

-0.7071 -0.7071

(we could have taken also the matrix
-1 1
-1 -1
or
1 –1
-1 –1
or other combinations; the reason for the 0.7071 (actually 1/sqrt(2)) is that we often want to make V orthogonal, i.e., make $\mathbf{V*V^T=I}$).

The associated eigenvalue matrix is
D=
2.5000 0
0 3.5000,

i.e., the eigenvalues are w_1=2.5, w_2=3.5; the first eigenvector is
$\mathbf{v_1}$=

-0.7071
-0.7071

and the second eigenvector (the second column of **V**) is
$\mathbf{v_2}$=

0.7071
-0.7071

I leave it up to you to verify that theses two vectors are indeed eigenvectors of **A (**you can verify it explicitly).

MATRICES IN Fortran-90

Fortran 90 is very suitable for matrices. Matrices are arrays; I presume you read about arrays in the relevant section of the fortran-90 notes.

The important point is one:
Matrix multiplication in fortran-90 is C=matmul(A,B); A*B means regular term-by-term multiplication, which is rarely used.

(I.e., C=A*B is equivalent to

```
do i=1,N
        do j=1,N
          C(i,j)=A(i,j)*B(i,j)
        enddo
enddo
```

While C=matmul(A,B) is equivalent to the regular matrix multiplication, i.e.,

```
do i=1,N
        do j=1,N
                C(i,j) = 0.d0
                do k=1,N
                        C(i,j)=C(i,j)+A(i,k)*B(k,j)
                enddo
        enddo
enddo
```

Matmul works with every variable-type (real*8, integer, complex*16, etc.)

Fortran-90 also has automatically the transpose statement. (**B**=transpose(**A**))

I have wrote simple-to-use interfaces to matrix inversion and matrix diagonalization which we would go over—see **library_matrices_example.f90**.

Normal Modes and Matrices

These are just brief introduction – you should also read the relevant chapters by Prof. Jonsson.

Say we have a molecule, where the particles interact so that the total potential is

V(**r**)

where **r** is a vector of length 3*N, where N is the number of molecules. (I remind you that we can view **r** as either a single vector, or as a matrix of size [3,N] – whatever more convenient, i.e., we associate

r(1) = x_1
r(2) = y_1
r(3) = z_1
r(4) = x_2, etc.

Let's **first** also assume for simplicity that all the masses of the atoms are equal (if not, then we can scale as we'll do **later**).

The primary equation is simply Newton's law (we use classical Mechanics language, but it turns out that this discussion is also relevant to Q.M.):

$$m\frac{d^2 r_j}{dt^2} = F_j = -\frac{\partial V}{\partial r_j}$$

Now since the molecule is bound, let's assume that it undergoes only small vibrations, so that the potential energy can be Taylor expanded in the small-deviations from the equilibrium bond-distances (where the potential is the lowest). Let's recall what's a Taylor expansion for a function of one variable:

$$g(x) = g(x_0) + (x - x_0)\frac{dg}{dx}\bigg|_{x=x_0} + \frac{1}{2}(x - x_0)^2 \frac{d^2 g}{dx^2}\bigg|_{x=x_0} + higher\_order\_terms$$

Similarly, the force can be expanded:

$$F_j(\mathbf{r}) = F_j(\mathbf{r}_0) + \sum_{k=1}^{3N}(r_k - r_{k0})\frac{\partial F_j}{\partial r_k}\bigg|_{\mathbf{r}=\mathbf{r}_0} \qquad i.e.,$$

$$F_j(\mathbf{r}) = 0 + \sum_{k=1}^{3N}(r_k - r_{k0})\left(-\frac{\partial^2 V}{\partial r_k \partial r_j}\bigg|_{\mathbf{r}=\mathbf{r}_0}\right)$$

The first term is zero, since at equilibrium the force, i.e., the gradient of the potential, is zero. If we define the 2-nd derivative matrix **U** as:

$$U_{ji} = \frac{\partial^2 V}{\partial r_j \partial r_i}$$

and the 3*N deviation vector **s** as:

$$\mathbf{s} = \mathbf{r} - \mathbf{r}_0$$

we find that Newton's law takes the form:

$$\frac{d^2 s_j}{dt^2} = \left(-\frac{1}{m} U \right)_{jk} s_k \quad i.e.,$$

$$\frac{d^2 \mathbf{s}}{dt^2} = -\mathbf{B}\mathbf{s}$$

where

$$\mathbf{B} = \frac{1}{m}\mathbf{U}$$

Now let's consider a **normal mode**. It is **defined as a deviation which is periodic in time**

$$\mathbf{s}_L(t) = \bar{\mathbf{s}}_L \cos(w_L t + \phi_L)$$

where $\bar{\mathbf{s}}_L$ is here a specific time-independent vector (not a component of the vector) associated with the L'th possible "normal-mode". If we plug the two equations above one to the other we get the normal mode equation:

$$\frac{d^2 \mathbf{s}_L}{dt^2} = \frac{d^2 \left(\bar{\mathbf{s}}_L \cos(w_L t + \phi_L) \right)}{dt^2} = -{w_L}^2 \cos(w_L t + \phi_L) \bar{\mathbf{s}}_L$$

and:

$$-\mathbf{B}\mathbf{s}_L = -\mathbf{B}\bar{\mathbf{s}}_L \cos(w_L t + \phi_L)$$

so:

$$-{w_L}^2 \cos(w_L t + \phi_L) \bar{\mathbf{s}}_L = -\mathbf{B}\bar{\mathbf{s}}_L \cos(w_L t + \phi_L)$$

i.e.,

$$\mathbf{B}\bar{\mathbf{s}}_L = {w_L}^2 \bar{\mathbf{s}}_L$$

So the normal modes frequencies squared are simply the squares eigenvalues of the force-constant matrix (scaled by 1/M).

Exercise: prove that for a 1-D particle in a Harmonic potential (1/2*kappa*x^2) you get back exactly the usual w=sqrt(kappa/M), expression.

I wont prove it, but you can similarly show that if the masses of the particles are not equal, you still need to solve the same equation, expect that now

$$B_{ij} = \frac{1}{\sqrt{m_i}} U_{ij} \frac{1}{\sqrt{m_j}}$$

Note however that j extends here from 1 to 3N, since we deal with 3 dimensions (x,y,z), so this is what the m_j mean here:

m_1=mass(atom 1)
m_2=mass(atom 1)
m_3=mass(atom 1)
m_4=mass(atom 2)
m_5=mass(atom 2)
m_6=mass(atom 2)
m_7=mass(atom 3)
....
m_{3k-2}=mass(atom k)
m_{3k-1}=mass(atom k)
m_{3k} = mass(atom k)
....
m_{3N} =mass(atom N)

where k=1,...,N.

The final point is how to numerically calculate the **U** matrix. Here we need an important rule: **DONT TRY TO OPTIMIZE** your program unless you have to! Specifically, if we deal with many particles (like in biological simulation) we will try tricks to calculate very efficiently the force constant matrix, relying on the fact that when we calculate the second derivative w.r.t. changes in r_i and r_j, then we only need to calculate the part in the total potential which change when r_i and r_j change – and typically this is just a small number of the total potential. But in your simulations, where you deal with a small number of particles, just do it simply.

To see how to do it simply, let's note how to do a derivative (force) very simply. If we want to calculate the force at **r,** we just calculate:

$$F_j(\mathbf{r}) = -\frac{\partial V}{\partial r_j} = -\frac{V(\mathbf{r}+\eta\mathbf{e}_j) - V(\mathbf{r}-\eta\mathbf{e}_j)}{2\eta}$$

$$\mathbf{e}_j = (0,0,0,...,0,1,0,...,0)$$

$$\eta : small$$

where the 1 is exactly at the j'th position, and eta is a small number (1d-5 or so, as we saw in the beginning of the course). Similarly we can write:

$$U_{ij}(\mathbf{r}) = -\frac{\partial F_j}{\partial r_i} = -\frac{F_j(\mathbf{r}+\eta\mathbf{e}_i) - F_j(\mathbf{r}-\eta\mathbf{e}_i)}{2\eta}$$

$$\mathbf{e}_i = (0,0,0,...,0,1,0,...,0)$$

We can use this approach by,

- defining a subroutine that given **r** calculates the potential;
- defining another subroutine that given **r** calculates the force in a direction j by calling the potential subroutine above at two different values of **r**
- defining a third subroutine that given **r** calculates U_ij by calling the second subroutine above at two different values of **r.**

Note that it is not completely automatic from what we said that the **U** that we get would be exactly symmetric (i.e., even for finite eta) but it turns out that it is, as you **should verify.**

I've written a simple routine, **normal_modes.f90**, that does this calculation. Read it and use it in the exercises below.

Optimization

The only things missing from our previous discussion (and which you **should include in the normal_modes.f90** program), is to prepare a general molecule in its equilibrium position. For the Ar_3 molecule it was easy due to the symmetry, and the fact that the bond distance was the same as for diatomics. But for a general molecule, we need to prepare it at a position **r0** (a 3N vector) such that V(**r**) is minimum. This is a general example of multi-dimensional optimization. To solve it, the simplest, and quite effective thing we can do is to define a trajectory. The idea is simple, as follows:

We would do a loop where we try to get closer and closer to the minimum. Say that we are at a 3N position **r**, where the potential is V(**r).** The gradient of the potential is minus the force, **-F(r).** The potential **increase**s **the most** along the direction of the gradient (see picture in class), and it therefore **decreases the most if "we"** (i.e., **r)** move **a little ("**a**")** in the negative direction of the gradient, i.e.,

r --> r+a***F(r)**

And this step needs to be done repetitively many times (typically 100 or more).

Here "a" needs to be optimized – if it is too big we would move too much and overshoot the minimum. If it is too small, we would barely move. A good idea is to plot V while we optimize, and once in a while (say every 100 iteration) optimize (typically slightly reduce) "a".

While you optimize, check the size of the force matrix (abs(force) in our language) – it should decrease over time, except for occasional overshootings.

Note: More Sophisticated Optimization algorithms are divided to three:

(a) For 1-d problems, there are very fast algorithms (logarithmic search, and Newton-Ralphsons' method – both converge with very few iterations (5-50) to machine accuracy.
(b) For problems similar to what we have, where we can start reasonably close to the minimum, there are algorithms like "conjugate gradients", where one uses not only the current force but also the force from the previous iteration – see Numerical Recipes.
(c) For the most difficult and interesting problems, where we have no clue about the minimum and we have many molecules, there are wilder algorithms such as "genetic" algorithms – if we have time we'll go over them at the end of the course.

Exercises:

(1) Run normal_modes.f90. It is set now to an harmonic potential in the distance between two carbon atrom.

Then write a simple routine to refine the equilibrium position, as explained earlier.

How different are the eigenvalues you get? Now, 5 of them should be very close to zero, and only one is non-zero. **Why?**

(2) Now run the program with a different potential, dpes_h2o, for water (see attached d*f file and directions in that file and in pot_h2o)

Optimize the distances for this molecule. Report them, and check with what a regular text book in general or organic chemistry would have. Then find the normal modes using both force-constant-matrix diagonalization. Compare to what's written in textbooks.

(3) Now determine the normal mode frequencies from a Fourier-transform of the correlation function. Start the atoms at their equilibrium position, give the atoms a small initial speed at an arbitrary direction $\mathbf{v_0}$**, (it is a 9 dimensional vector – associated with the three velocities at each dimension)** (you'll need to see yourself what small means – by looking at the eventual positions as a function of time), and then determine the atomic position and speed as a function of time.

Plot the velocity-velocity correlation function ($C(t) = \mathbf{v_0}*\mathbf{v}(t)$ – it is actually defined usually more general than that, but never mind). Do a Fourier-Transform of this function, and extract the normal mode frequencies from that. If $\mathbf{v_0}$ is small, you should get the same frequencies as from the diagonalization of the force-constant matrix, but as the magnitude of $\mathbf{v_0}$ changes, new modes should appear, and the frequencies of the old modes should continuously change.

Also: how many modes have zero frequency? These modes are associated with translation and rotation – prove it by actually plotting the modes (we'll talk about it in class).

How do these frequencies (from normal modes and from Fourier-transforms compare with the quantum mechanical transition frequencies?

Print out the eigenvector, so you can associate them with the eigenvalue (bending, symmetric and anti-symmetric stretch).

Quantum Mechanics:

QM of Electrons--Huckel Hamiltonian

The matrix methods and program we have is useful for simple problems in quantum mechanics. The simplest is the use of approximate methods for molecular orbitals, especially in organic chemistry. Among the simplest methods, the simplest of all is the Huckel approach for planar organic molecules, in which we concentrate on the pz orbitals (perpendicular to the plane), and assume that from the individual pz orbitals on each atom "j", $\eta_{PZ,j}$, we make molecular orbitals, psi_k, as follows:

$$\psi_k(\mathbf{q}) = \sum_{j=1}^{N} A_{kj}\eta_j(\mathbf{q})$$

(where for brevity we omit the pz subscript). N is the number of atomic pz orbitals (in practice the number of carbon atoms if we deal with purely hydrocarbons planar systems) and **q** is the 3-D position of the electron. The Schroedinger equation for the molecular orbitals is then

$$H\psi_k = \varepsilon_k\psi_k$$

i.e.,

$$H\sum_{j=1}^{N} A_{jk}\eta_j(\mathbf{q}) = \sum_{j=1}^{N} A_{jk}\eta_j(\mathbf{q}) * \varepsilon_k$$

where H is the approximate Hamiltonian governing the motion of the orbitals, and epsilon_k is the orbital energy. Both sides on the equation above depend on **q,** the position of the electron. We can get an equation governing the "a_kj" coefficients by a standard trick – for each "*i*" ranging in value from 1 to N multiply both sides of the equation above by $\eta_i(\mathbf{q})$ and integrate over all volume (all values of **q)**. The result is then

$$\int \eta_i(\mathbf{q}) H\sum_{j=1}^{N} A_{jk}\eta_j(\mathbf{q})d^3\mathbf{q} = \int \eta_i(\mathbf{q})\sum_{j=1}^{N} A_{jk}\eta_j(\mathbf{q})d^3\mathbf{q} * \varepsilon_k$$

or

$$\sum_{j=1}^{N}\left(\int \eta_i(\mathbf{q}) H\eta_j(\mathbf{q})d^3\mathbf{q}\right)A_{jk} = \sum_{j=1}^{N}\left(\int \eta_i(\mathbf{q})\eta_j(\mathbf{q})d^3\mathbf{q}\right)A_{jk} * \varepsilon_k$$

or, if we call the terms in parentheses in the equation above $\boldsymbol{H_{ij}}$ and $\boldsymbol{S_{ij}}$, then we get:

$$\sum_{j=1}^{N} H_{ij}A_{jk} = \sum_{j=1}^{N} S_{ij}A_{jk}\varepsilon_k$$

or in matrix notation

$$HA = SAD$$

where *D* is the diagonal matrix such that on the diagonal D_{kk}=*epsilon_k.*

This equation $HA = SAD$ has a name in matrix math: Generalized eigenvalue problem (generalized since it has "S" on the right – the regular eigenvalue problem does not have "S"). There are several good ways to solve it, some of them obvious, some less (such as the QL decomposition) – read Numerical Recipes to find out about it. (My library routines have a routine for that – see library*gene*).

At any rate, the generalized eigenvalue equation above as it stands corresponds to something called "extended Huckel", which is quite useful and somewhat quantitative

The simpler, original, method by Huckel corresponds to a further approximation, i.e., assuming that the "S" is the unit matrix, i.e., that it is zero for different orbitals. At the same time, "H" is not assumed to be a unit matrix. This is quite a drastic and not wholly consistent approximation, but it turns out that it is OK for qualitative analysis. The equation that results is then a regular eigenvalue equation:

$$HA = AD$$

The next assumption is that "H" has diagonal and off-diagonal terms, which are defined as follows. The diagonal term in H depends on the atom that carries the pz orbital. Since we usually only deal with Carbon (the hydrogens are unimportant since they don't carry pz orbitals), then the diagonal term would be a fixed number, typically taken as

H_{ii}=-6.6 for "i" a Carbon atom.

in eV units (remember that one eV~ 96kJ/mol ~ 1/27.12 atomic units).

The off-diagonal coupling is assumed to be non-zero **only if the atoms are bonded**

H_{ij}= 0 if atoms i and j are not bonded.

For regular double-bonds between carbon atoms, a reasonable finding is that:

H_{ij}= -2.7 eV if "i" and "j" are different Carbon atoms that are bonded.

Note the limitation of the approach: it can only be applied to a planar system, where the C are pi-bonded. (Don't apply it to ethane!)

First Exercise:

Program the Huckel Hamiltonian for Benzene (C6H6). Remember the bonding: H_{ij}=0 unless i=j, or i=j+1 or j-1, or unless i=1, j=6 or i=6,j=1 (plot the molecule to see what this means). Find the eigenvalues, and plot them . For each one do a simple plot of the eigenvectors– the result should exactly correspond to what you learned in Organic chemistry. Also, for neutral C6H6, how many of these orbitals are filled? (Answer—3!)

Second Exercise: Pierles transition, Cyclical and Wire of polyacetylene.

(i)$C_{100}H_{100}$
Now extend what you did to cyclical $C_{100}H_{100.}$ Plot just the eigenvalue spectrum. The simplest way to plot the spectrum is simply to put in a file "j, epsilon(j)".

(ii)
The properties of the cyclical C100H100 are similar to those of an elongated system (a wire). To see this, plot the eigenvalues for a system which is not cyclical –wire.(What modification do you have to do?)

(iii) In practice, there can be a lowering of the energy if the bonds alternately contract and expand (so they look more like double and single bond). In practice, you can see this by changing the coupling to:

-2.55 in the weaker bond;
-2.85 in the stronger bond

Calculate for the wire and the cyclic system what happens to the eigenvalue spectrum. You'll notice a clear gap! This is a band-gap, since it occurs exactly at the middle of the spectrum, associated with half-filling. We'll talk about the physical consequence of this gap.

QM of vibrational motion.

In this part we'll do some simulations of simple 1-D vibrational spectra, i.e., spectra of vibrating molecules – some of which would exhibit a very interesting vibrations. Note that we dealt with this overall issue before when we talked about classical modes. Here, we restrict the description to 1D, since Q.M. is much more expensive than classical dynamics.

Consider the 1-D Schroedinger Equation (SE):

$$\left(\frac{p^2}{2m}+V(x)\right)\phi_n(x)=\left(-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}+V(x)\right)\phi_n(x)=\varepsilon_n\phi_n(x)$$

It is for a continuous variable, but it does look very much like the problem of diagonalization of a matrix that we've seen already. So all we need now is to discritize the grid. So consider that x is between $x_{\min}$ and $x_{\max}$. divide x into N values:

$$[x_1=x_{min},\ x_2=x_{min}+dx,\ x_3=x_{min}+2*dx,...,x_N=x_{min}+(N-1)*dx=x_{max}]$$

where we need to make sure that we converge the results (i.e., make xmin small enough, xmax large enough, and N large enough, i.e., dx small enough).

The equation above is then considered to be valid at each grid point j, i.e., (replacing phi_n by psi)

$$[H\psi](x_j)=\left(-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2}(x=x_j)+V(x_j)\psi(x_j)\right)=\varepsilon_n\psi(x_j)$$

The next and quite important question is which approximation for the second-derivative would we use. The simplest is the 3-point formulae we've seen above:

$$[H\psi](x=x_j)=-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2}(x=x_j)+V(x_j)\psi(x_j)\simeq$$
$$-\frac{\hbar^2}{2m}\left[\psi(x=x_j+dx)-2*\psi(x=x_j)+\psi(x=x_j-dx)\right]+V(x_j)\psi(x_j)$$

which, if we now change psi($x=x_j$) to simply psi_j, we get

$$[H\psi]_j=-\frac{\hbar^2}{2mdx^2}\left[\psi_{j+1}-2\psi_j+\psi_{j-1}\right]+V(x_j)\psi_j$$

(Don't confuse j, the grid index, with n, the eigenvalue index).

Before proceeding we need to decide what to do with the wavefunction at the end points.

The simplest option is to imagine that the grid actually extends over a larger region then [xmin, xmax], but that the wavefunction is (and is later forced to be) zero at that region; thus, values of j beyond the end points do not contribute to the 3-point derivative at those points

$$\left[\frac{d^2\psi}{dx^2}\right]_j \simeq \frac{\psi_{j+1} - 2\psi_j + \psi_j}{dx^2} \qquad 1 < j < N$$

$$\left[\frac{d^2\psi}{dx^2}\right]_j \simeq \frac{\psi_{j+1} - 2\psi_j + 0}{dx^2} \qquad j = 1$$

$$\left[\frac{d^2\psi}{dx^2}\right]_j \simeq \frac{0 - 2\psi_j + \psi_{j-1}}{dx^2} \qquad j = N$$

From the equations above, it is clear that we can redefine a MATRIX **H,** such that the action of the matrix on the vector **psi** (=(psi_1, psi_2,...,psi_N)^T) equals [Hpsi]_j; the elements of this matrix are:

$$H_{ij} = \begin{cases} \dfrac{2\hbar^2}{2mdx^2} + V\left(x_j\right) & i = j \\ -\dfrac{\hbar^2}{2mdx^2} & i = j \pm 1 \\ 0 & otherwise \end{cases}$$

i.e., a tridiagonal matrix.

Thus, the eigenvalue equation is converted to the following equation

$$\sum_k H_{jk}\phi_n(x_k) = \varepsilon_n \phi_n(x_j)$$

or, if we define

$$\Phi_{kn} = \phi_n(x_k)$$

(note the order – the eigenvalue index is on the right so each Column is an eigenvector) we are left with the regular eigenvalue equation:

$$H\Phi = \Phi D$$

where D, as before is the diagonal eigenvalue matrix.

So to summarize: to find Phi (and therefore the eigenfunctions, which, **up to a constant** are simply the eigenvectors; e.g., **the n'th eigenvector is Phi(:,n) (again, up to a constant of normalization),** we need to:

- Define the NxN matrix H
- Diagonalize it

- Normalize the eigenvectors so that for each eigenvector "i",
  Sum_j phi_i(x_j)2 * dx=1**
  In fortran-90 it is simple:
  Do i=1, Ntot
  phi(:,i)=phi(:,i)/sqrt(sum(phi(:,i)**2*dx))
  enddo

- And check the that [H phi_n](xj) = espilon_n phi_n(xj) is fulfilled.
  This last stage, for example, is very simple in FORTRAN 90:

  do i=1, Ntotal

  diff = sum(abs(matmul(H, Phi(:,i)) – epsilon(i)* Phi(:,i)))

  write(6,*)' diff for i'th eigenvalue ',i,diff
  if(diff>1e-7) stop
  enddo

NOTES:

SPARSENESS
Note that this **H** matrix is **sparse;** only a small fraction (3N-2 out of N^2) of its elements are non-zero. Thus, it is very cheap to **act** with this matrix on a general vector (the prescription for calculating the vector **Hpsi ,** given **psi**, is given above – it involves about 3N operation). Compare it to the usual cost of multiplying a matrix by a vector, which is N^2 , or to diagonalize a matrix, N^3. Much of the developments in describing large quantum systems has been associated with this observation and with development of methods whereby one inverts or diagonalize a matrix by multiplying it many times by a vector.

Accuracy of the 3-point formula
The 3-point formula we have is not terribly accurate unless dx is especially small. In practice, there are several remedies.

The simplest, and about as good as any other approach, has been to use 5 point or 7 point formulae (there are ways to derive them, we wont worry about them right now).

Another approach formally better (but not much better in practice then the 5 and 7 points formulae) is to use **spectral approaches. (The main reference is: R. Kosloff**, Time-dependent quantum-mechanical methods for molecular dynamics. Journal of Physical Chemistry, 21 April 1988, vol.92, (no.8):2087-100)**-you can download it from www.cdlib.org, as I'll show you.)**.

Briefly, the idea is to represent the wavefunction as a sum of Fourier components:

$$\psi(x) = \sum_j A_j \exp(ik_j x)$$

where the A_j are found by FFT from the values of psi(x) on the grid.

$$\psi(x_m) = \sum_j A_j \exp(ik_j x_m),$$

i.e.,

$$FFT(\psi(x_m)) \rightarrow A_j$$

The next stage is to treat the equation above as if it is valid at all values of x between xmax and xmin, even those that are not on the grid. In other words, we view the equation above an **interpolation formula.** Then, to get the 2'nd derivative we **analytically differentiate** the equation above:

$$\frac{d^2\psi}{dx^2} = \sum_j -A_j k_j^{\,2} \exp(ik_j x)$$

and specifically, at our grid points:

$$\frac{d^2\psi}{dx^2}(x_m) = \sum_j -A_j k_j^{\,2} \exp(ik_j x_m)$$

Thus the algorithm in brief is:

$$A_j \leftarrow FFT(\psi(x_m))$$

$$\frac{d^2\psi}{dx^2} \leftarrow Inverse\_FFT(-k_j^{\,2} A_j)$$

The library has examples for how this is done and used, but it would be **a good exercise for you to write your own program** to do this. (And then to see how much it is more accurate then the 3-points formula when it works on something like, e.g., a Gaussian).

Chemistry with the 3-point 1-D Schroedinger Hamiltonian.

Now that I talked about how much one can improve on the 3-point formula, let's forget about improvements (that's our motto) and simply try the H defined above.

Note that this Hamiltonian is essentially the same as the one you used for the Huckel model (except for the x-dependence of the diagonal terms; the different constant and diagonal terms; and the fact that there is no "wrapping around", i.e., j=1 is not coupled to j=N).

So:

Exercise 1: Calculate the eigenvalues of the Hamiltonian for a Harmonic Oscillator, where $V(x) = \frac{1}{2} m w^2 (x-x_0)^2$. Try to converge the first 4 eigenvalues to 5 digits. Of course, you know what the values should be: (0.5, 1.5, 2.5, 3.5)*hbar*w.

For your first calculation, it is a good idea to do first a dimensionless calculation – where you set w=m=hbar=1, and then do a calculation where you put in values for w,m,hbar – use the values for **H2 diatom, where:**

w~0.5eV/hbar~0.02a.u., and

$hbar^2$/2m(hydrogen) = 1/2/1836a.u.

(but the reduced mass for H2 is HALF the mass of hydrogen – so MULTIPLY the coefficient above by 2).

x0 ~ 1.6 a.u. (you can just put in these values and forget about units – i.e., use eV for energies and Angstrom for distances).

Then the only questions are : what should N be and what should xmin and xmax be. These you will decide by convergence, but a good starting point as far as xmin and xmax is to make them in the classically forbidden region – i.e., such that xmin is much lower than the equilibrium point x0, xmax is much larger than x0, and V(xmin) and V(xmax) are significantly larger then the energy of the fourth eigenstate that we look for . Show your results for several values of xmin, xmax and N.

One important note on how to check convergence: what we should really **converge** is not so much N, but **dx**. (Since dx determines directly the accuracy of the second derivative expression). So when you **decrease xmin and increase xmax, keep dx fixed (by increasing N); conversely, when you check convergence in dx, keep xmax and xmin fixed.**

After you converge the eigenvalues, plot the eigenvectors (phi_n(x)). Also plot the overlap integrals:

$$< \phi_n \mid \phi_k >= \int \phi_n(x)\phi_k(x)dx \approx \sum_{j=1}^{N} \phi_n(x_j)\phi_k(x_j)dx$$

You'll verify that the integrals are zero if n=k; you need to normalize the functions so that the overlap would be 1 for n=k.

A note on overlap integrals: you may think that we should have used a more fancy overlap integral expression (with j-dependent weights, like in Simpson integration). However, it turns out that we should use **exactly** this expression. The reasons have to do with properties of hermitic (or symmetric here) operators and matrices – symmetric operators have eigenvectors that are orthogonal if we use summation without weights, like shown above. (For proof see class or QM or linear algebra books).

Exercise 2: a better description of the H2 potential is with a Morse Oscillator,

$$V(x) = D\left(\exp\left(-\frac{2(x-x_0)}{\sigma} \right) - 2\exp\left(-\frac{(x-x_0)}{\sigma} \right) \right)$$

where D=4.74eV, and sigma is adjusted so that, if we expand V to 2'nd order in x-x0, the coefficients of the $(x\text{-}x0)^2$ term is the same as in the Harmonic expansion above.

Calculate numerically the eigenvalue spectrum for the H2 with this potential, and compare it to the spectrum you found above (you can actually also find the eigenvalue spectrum analytically for the Morse Oscillator, but that's not interesting for us now). What do you see as far as the level spacing? Plot this and the associated Harmonic potential, and explain the different behavior in both cases. Plot also the eigenvectors – do you see anything interesting?

Exercise 3:

The He2 diatom is very interesting since it is very large and very weakly bound. It was found out only in the last decade. Use the Lennard-Jones parameters to calculate its vibrational lowest eigenvalue (ground state). Remember to use the correct mass/energy/distance units (convert to a.u.)

You'll find a very interesting phenomena – even though the vdW radius is reasonably small (a few a.u.), **the ground vibrational state is very extended (use a very large grid, until you reach convergence.)**

Calculate <x> for the ground state of He2.

Plot the vibrational ground-state.

Do any of the vibrational eigenstate you find numerically beyond the ground state correspond to true physical eigenstates? (A quick way to see that is whether their energy is **both below E=0**, and **is convergent as a function of grid parameters**.)

Exercise 4:

This exercise is also very interesting – double well. An example, if a potential of the form:

$$V(x) = ax^4 - bx^2 + gx + C_0$$

where a,b are positive constants, and g and C0 can have any value. To see this potential, plot it for ammonia, where

b = 22000 cm^{-1} $Å^{-2}$ (**convert to a.u.!)**

a = 61000 cm^{-1} $Å^{-4}$

mass = 2.47 amu.

Such a potential describes, for example, the Umbrella motion of the Ammonia molecule, or many reacting system (where the system goes from one minimum to another through a barrier). You can also set without loss of generality C0 as zero – it just shifts the eigenvalues by a constant

For checking, the resulting energy difference values for Ammonia (in **wavenumbers):**

e2 – e1 = 0.5378
e3 – e2 = 932.5978

These are the values for Ammonia. Find "a" and "b" (from symmetry, "g" is zero for regular Ammonia).

So your job is to: first find a,b; then find the first 6 eigenvalues; plot the eigenvectors for the two cases – concentrate especially on the first two, then on the next two.

You'll see that the eigenvectors are <u>delocalized.</u>
Now put the ammonia aligned in an electric field. The contribution of the electric field is:

V(x) ---> V(x)-uFx

where u is the derivative of the dipole moment of the Ammonia w.r.t. x, F is the electric field. Thus, -u*F is exactly a linear term of the sort that we introduced before.

In our atomic units (a.u.), u has the value which you can determine, by noting that the expectation value of the dipole moment (ux) equals:
<psi_0|ux|psi_1> = 0.6 (in a.u.) (=1.5Debeye).

Determine what F needs to be to make the first two eigenvectors appreciably **more localized** (you'll see that by inspection). You can convert it to regular Field, in Volt/cm, by converting from our units

1a.u.(electric field)
=energy(a.u.)/charge_electron/distance(a.u.)
=27.12eV/e/[0.529Angstrom] =
=[27.12/0.529]*10^8 Volt/cm
~ 5bilion V/cm

After conversion, judge if this is a realistic value – the most that we can get reasonably in air is ~$3*10^6$ V/m (there are ways to get more microscopically, or in solution, etc.).

<u>Exercise 5: (really a project)-Predissociation.</u>

Predissociation corresponds to a case where a particle is quasi-bound, i.e., trapped in a well out of which it can tunnel out. One example for a potential which can produce predisocciation is a diatom that undergoes a rotation:

$$V(r) = V_{vibrational}(r) + \frac{\hbar^2 l(l+1)}{2mr^2}$$

where r is the distance between the atom, and l is an integer. If you plot this potential, you'll see that for l sufficiently large (but not too large), it will have a well, but that well will have its lowest value higher then the minimum at infinity.

The question is how to estimate the energy in the well, and the rate it takes the particle to get out. The answer is

You add to the potential a component, $-iV_I(r)$, which is placed at the end of the grid (for large r, not small). The potential should be NEGATIVE IMAGINARY – for example, a triangularly-shaped potential of maximum height 0.2a.u. and width 2 a.u. should do. (For details – see :

D. Neuhauser and M. Baer,
The Time Dependent Schrödinger Equation: Application of Absorbing Boundary Conditions,
J.Chem.Phys. **90**, 4351 (1989), and

Leforestier, C.; Wyatt, R.E.,
Optical potential for laser induced dissociation.
J. Chem. Phys, **78**, 2334 (1983)

Then diagonalize the resulting COMPLEX (non-Hermitian, if you know what Hermitian means) Hamiltonian. You'll see that one or more eigenvalues would have a reasonably SMALL IMAGINARY part of their eigenvalue – these are associated with the resonance. The imaginary part is associated with the rate of tunneling out of the barrier.

How much is it? How does it depend on l?

TIME DEPENDENT Q.M.

So far we dealt with the time-independent Schroedinger equation,

(Hphi_n)(x) = epsilon_n phi_n(x)

However, here we deal with the time-dependent Schroedinger equation (TDSE):

(H psi)(x,t) = i d/dt psi

where here i is of course the complex "i", sqrt(-1). To avoid confusion I would call it ci in the program. **NOTE: we don't include here an hbar factor – this is equivalent to using a scaled time unit, t/hbar, instead of the true t. How much is an atomic unit of time in seconds?– calculate exactly**

There are two types of reasons to deal with the TDSE.

First, we can do it for the same time of Hamiltonians we dealt before, which are time-independent, and then it can be done for: **clarity, numerical reasons, and for interpreting actual process of scattering and particle motion.**

The **Second** reason is if the Hamiltonian is intrinsically time-dependent.
For example, take a particle under the influence of a laser field; the Field adds a term -x*u*F*cos(wt), where w is the frequency of the field. That term oscillates the particle (both classically and quantum mechanically) until it can in some cases leave.

In both cases, we deal with **Hamiltonians that are matrices, i.e., the relevant equation is (using f90 notation)**

i (d/dt psi) = matmul(H, psi)

where psi = psi(1:ntot)

We'll start with the

First Class: Time-independent (T.I.) Hamiltonians. There are four classes of ways to do those.

The first and simplest way notes that the solution to the time-independent equation can be **formally written as**:

psi = exp(-i*H*t) psi0

where psi0 is the initial wavefunction. (Verify that this psi fulfils the TDSE when H is a constant number). But how do we **exponentiate matrices? (or more precisely: how do we evaluate the action of the exponential of a matrix on an initial vector?).**

There are two equivalent ways to answer it. Physically, we can decompose the initial wavefunction (psi0) in terms of eigenstates, like we've done earlier

$$\psi_{t=0}(x) = \sum_{k=1}^{N} a_k \phi_k(x)$$

Or in matrix language:

$$\psi_{j,t=0} = \sum_{k=1}^{N} a_k \phi_{j,k}$$

We further assume here that we normalize the phi_k without the dx, i.e.,

$$\sum_{j=1}^{N} \phi_{j,k} \phi_{j,m} = \begin{cases} 0 & k \neq m \\ 1 & k = m \end{cases}$$

To find the a_k, we multiply by phi_j,m (for a specific m) and sum over j:

$$\sum_{j=1}^{N} \phi_{j,m} \psi_{j,t=0} = \sum_{j=1}^{N} \phi_{j,m} \sum_{k=1}^{N} a_k \phi_{j,k} = \sum_{k=1}^{N} a_k \sum_{j=1}^{N} \phi_{j,m} \phi_{j,k} = \sum_{k=1}^{N} a_k (1 \quad if \quad k = m) = a_m$$

Now since

$$H\phi_k = \varepsilon_k \phi_k$$

it follows that any function of H when it works on phi_k, is equivalent to the same function of H but with H replaced by (the number) epsilon_k. To see that explicitly, let's consider some simple functions, e.g., H^2

$$H^2 \phi_k = HH\phi_k = H\varepsilon_k \phi_k = \varepsilon_k H\phi_k = (\varepsilon_k)^2 \phi_k$$

The same goes for H^3 and other functions of H. OK, so therefore

$$\exp(-iHt)\phi_k = \exp(-i\varepsilon_k t)\phi_k$$

and thus,

$$\exp(-iHt)\psi_{t=0} = \sum_{k=1}^{N} a_k \exp(-iHt)\phi_k = \sum_{k=1}^{N} a_k \exp(-i\varepsilon_k t)\phi_k$$

(A different derivation is as follows. Any matrix H that be diagonalized can be denoted as

$$H\Phi = \Phi D \, .$$

Multiply from the right by Phi_inverse:

$$H = \Phi D \Phi^{-1}$$

Similarly, H^2 can be written as

$$H^2 = \Phi D \Phi^{-1} \Phi D \Phi^{-1} = \Phi D D \Phi^{-1} = \Phi D^2 \Phi^{-1}$$

where D^2 is simple, because D is diagonal – each diagonal element is squared. Similarly,

$$\exp(-iHt) = \Phi \exp(-iDt) \Phi^{-1}$$

so:

$$\psi = \exp(-iHt)\psi_0 = \Phi \exp(-iDt) \Phi^{-1} \psi_0$$

If you expand the terms you'll see that this is the same as the equations we derived with the a_n. This approach is very convenient if we use matmul in f90**)**

The good things about the algorithms are:

- It is exact;
- and simple, and very quick for small systems (N up to 100-1000)

The bad aspects, however, are that:

- It scales as N^3 (due to the matrix diagonalization), which could be quite expensive for large systems.

Propagation by Runge-Kutta-type algorithms

The next approach is simple, and general, but not terribly efficient (even though it beats the previous approach, exponentiation by diagonalization, for really large system). We simply view the propagation as a general ordinary-differential equation for a **complex** vector of N components

dpsi_dt = F(psi)

where in our case

F(psi) = -i*matmul(H,psi).

You can use the same algorithms we used for classical mechanics, except that now we replace the quantum mechanics of 1-D system with N grid points by the classical mechanics of N (1-D) particles. You see why Quantum mechanics is so expensive – even for 1D it can require the equivalent of the work needed to describe 100 classical particles or more!.

The difficulty with this algorithm is that is requires quite small time-steps, but it is general and useful.

Note, in applying this algorithm, remember to use the subroutine **complex_rk4** (since psi and F(psi) would be complex functions now!)

Propagation methods Tailored for the Schroedinger equation.

This is outside our scope, but you should know that there is an algorithm that's exact, and works very well with large time-steps requiring little overall cost. The algorithm is based

on a polynomial expansion of H. To illustrate what is a polynomial expansion, consider the following expansion:

$$\exp(-iHt)\psi_0 = \sum_{k=0}^{\infty} \frac{(-it)^k}{k!} H^k \psi_0$$

It is straightforward to evaluate the terms in the expansion by iteration, i.e., we write something like:

$$\exp(-iHt)\psi_0 = \sum_{k=0}^{\infty} \frac{(-it)^k}{k!} \eta_k$$

$$\eta_0 = \psi_0$$

$$\eta_k = H\eta_{k-1} \quad for \quad k > 0$$

(Note how simple this would be in FORTRAN 90:

```
psi_t(      1:Ntot) =psi_0     ! complex*16 array defined earlier
eta_term(1:Ntot) =psi_0  ! complex*16 array defined earlier
c_factor = 1

do k=1, kmax
        eta_term = matmul(H, eta_term)
        c_factor = c_factor *(-ci*t)/dble(k)
        psi_t = psi_t + c_factor*eta_term
enddo
```

In practice, if H is sparse, we may want to write a more efficient subroutine to calculate matmul(H, eta_term) which takes sparseness into account.)

The **problem** with this Taylor-expansion algorithm is that **it fails badly (unless t is very small).** The reason is that the expansion of exp(-iHt), while convergent, will usually consist of very large positive and negative contributions that will cancel each other (to see that, consider the Taylor expansion of exp(-x) where x is large; the result is very small, but the individual terms, $(-1)^n x^n/n!$ could be very large (positive and negative) if x is big).

However, it turns out that there is an expansion which does converge, called the **Chebyshev expansion**, which converges well. To read about the expansion, see the feature article by Kossloff in J. Phys. Chem. 1988, **92,**2087(1988), mentioned earlier).

The implementation of that algorithm is not much more difficult, and I wrote a black box which does it, called lib*prop*cheby*f90

Finally, there is another algorithm, called the **split-operator algorithm**, which, if we have time, we'll discuss.

Second Class: Time-dependent (T.D.) Hamiltonians.

The second class is time-dependent Hamiltonians, where H depends on time explicitly, so one needs to solve the Schroedinger equation

i d/dt psi = -iH psi

using time-dependent methods. The split-operator is a good choice, but if we don't get to studying it, you can use the Runge-Kutta algorithm we talked about earlier.

Exercises:

(1) The first exercise would be to study the behavior of the double-well system we studied before, under the double-well Hamiltonian and an added electric field

$$H = \frac{P^2}{2m} + V_{double\_well}(x) + V_{laser}(x,t)$$
$$V_{double\_well}(x) = ax^4 - bx^2 + gx + C_0$$
$$V_{laser}(x,t) = -Dx\cos(wt)$$

where w is **LARGER** then the spacing between the lowest two eigenstates (e2-e1), but significantly **SMALLER** then the spacing in each well (i.e., then the e3-e2 values).

Plot the probability for the atom to be on the right as a function of time, for several values of *D*. You'll notice that there are some magical values of *D* (which are about 2-10 times larger then e2-e1) where the particle does not tunnel to the right!

This phenomena is called "coherent destruction of tunneling", and is discussed in a paper: by:
Grossmann, F.; Dittrich, T.; Jung, P.; Hanggi, P.
Coherent destruction of tunneling.
Physical Review Letters, 22 July 1991, vol.67, (no.4):516-19.

Read the paper and use their parameters before you start programming it! Then modify the parameters to those of ammonia and see if you still get this behavior

(2) Use the same time-dependent program to study the evolution of a wave-function in a Harmonic potential (V=kappa/2 * x^2), under the influence of an electric field,

$$-D*x*\cos(w_e t)$$

The initial wavefunction should be a Gaussian.

Show that if you wait a reasonably long time (several oscillations) then if the frequency of the field (w_e) matches the frequency of the Harmonic potential (w=sqrt(kappa/m)), the wavefunction would start moving (<x> would oscillate and increase in amplitude), but not otherwise.

More pointedly, plot the total energy (defined as integral(psi^*(x,t)(Hpsi)(x,t) dx)) as a function of time for w_e=w and for several other values.

Also plot for several values of w_e, and t |psi(x,t)| as a function of x. Show that it **keeps** the **same** (Gaussian) shape! This can be proven analytically, as we'll discuss in class. **Remember of course to make sure your program converges as far as step size (dx and dt) and as far as xmin,xmax.**

Final Topic:

Random Numbers and Stochastic Processes

To be written!