Computational Physics with Maxima or R:
Example 1
Semiclassical Quantization of Molecular Vibrations *

Edwin (Ted) Woollett

August 31, 2015

Contents

*The code examples use **R ver. 3.0.1** and **Maxima ver. 5.28** using **Windows XP**. This is a live document which will be updated when needed. Check **http://www.csulb.edu/~woollett/** for the latest version of these notes. Send comments and suggestions for improvements to **woollett@charter.net**

```
example1.pdf describes the major example which accompanies
 Chapter 1 of Computational Physics with Maxima or R,
 and is made available to encourage the use of the R and
 Maxima languages for computational physics projects of modest size.

R language free and open-source software:
  http://www.r-project.org/

Maxima language free and open-source software:
  http://maxima.sourceforge.net/

Code files available on the author's webpage are
1. example1.R :use in R: e.g., source("c:/k1/example1.R") to load
2. example1.mac  :use in Maxima: e.g., load("c:/k1/example1.mac") to load

The author uses the XMaxima interface exclusively, with the startup
  file setting: display2d:false$, which allows denser screen output.

The author normally uses the default RGui interface when coding in R.

COPYING AND DISTRIBUTION POLICY:
NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.

You may make copies of this document and distribute them
to others as long as you charge no more than the costs of printing.
```

Feedback from readers is the best way for this series of notes to become more helpful to users of **R** and **Maxima**. *All* comments and suggestions for improvements will be appreciated and carefully considered.

1 Introduction

The major example worked out in Chapter 1 of Steven Koonin's text **Computational Physics** applies root finding and quadrature methods to the task of finding the approximate vibrational energy levels of a diatomic molecule in the quasi-classical approximation.

We use the resources of the free and open source software **R** (**`http://www.r-project.org/`**) and Maxima (**`http://maxima.sourceforge.net/`**) to write code which helps to solve this type of problem.

The use of such modern powerful "command interpreters" encourages a "bottom-up" style of code development, in which small jobs are coded first, checked interactively for correct behavior, and then used as part of slightly larger coding jobs in an iterative fashion. Our discussion provides explicit examples (in both languages) of this coding style.

2 Approximate Bound State Energies in the WKB Method

See Sec. 16.2 of R. Shankar, **Principles of Quantum Mechanics**, 2nd ed., 1994, for derivations and discussions of applications of the WKB method. See also **http://en.wikipedia.org/wiki/WKB_method**.

In the usual case in which none of the boundaries are infinite, and $\mathbf{x_1}$ and $\mathbf{x_2}$ are the classical turning points, with $\mathbf{x_2} > \mathbf{x_1}$, and with $\mathbf{p(x)} = \sqrt{\mathbf{2\,m\,[E - V(x)]}}$, the possible energy eigenvalues $\mathbf{E}$ are constrained by the relation

$$\int_{\mathbf{x_1}}^{\mathbf{x_2}} \mathbf{p(x)}\,\mathbf{dx} = (\mathbf{n} + \frac{\mathbf{1}}{\mathbf{2}})\pi\,\hbar \tag{2.1}$$

in which $\mathbf{n} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$.

2.1 Example: Simple Harmonic Oscillator and the WKB Approximation

This approximate method actually gives the correct energy eigenvalues for the simple harmonic oscillator, for which the potential (energy) is

$$\mathbf{V(x)} = \frac{\mathbf{1}}{\mathbf{2}}\,\mathbf{m}\,\boldsymbol{\omega}^{\mathbf{2}}\,\mathbf{x}^{\mathbf{2}} \tag{2.2}$$

with energy eigenvalues

$$\mathbf{E_n} = \left(\mathbf{n} + \frac{\mathbf{1}}{\mathbf{2}}\right)\,\hbar\,\boldsymbol{\omega} \tag{2.3}$$

We solve for the turning points and predicted energies using Maxima (as an exercise and practice in using Maxima):

```
(%i1) V : m*w^2*x^2/2;
(%o1) m*w^2*x^2/2
(%i2) s1 : solve(E - V,x);
(%o2) [x = -sqrt(2)*sqrt(E/m)/w,x = sqrt(2)*sqrt(E/m)/w]
(%i3) x1 : rhs(s1[1]);
(%o3) -sqrt(2)*sqrt(E/m)/w
(%i4) x2 : - x1;
(%o4) sqrt(2)*sqrt(E/m)/w
(%i5) assume(m > 0, w > 0, E > 0);
(%o5) [m > 0,w > 0,E > 0]
(%i6) action : integrate(sqrt(2*m*(E - V)),x,x1,x2);
(%o6) %pi*E/w
(%i7) solve (action = %pi*hbar*(n+1/2), E);
(%o7) [E = (2*hbar*n+hbar)*w/2]
(%i8) factor(%);
(%o8) [E = hbar*(2*n+1)*w/2]
```

However, this accuracy is not the norm. Usually, the lowest eigenvalues predicted by this method are not very accurate, and the accuracy gets better for the higher eigenvalues.

3 Diatomic Molecule and the Lennard-Jones Potential

See ch. 21, Molecules, in Gordon Baym, **Lectures on Quantum Mechanics**, 1974, for an excellent discussion of diatomic molecule energy orders of magnitude and the Born-Oppenheimer approximation.

Ch.2, Sec.4 of **Computational Physics** by Steven E. Koonin introduces this example with:

> As an example combining several basic mathematical operations, we consider the problem of describing a diatomic molecule such as $\mathbf{O_2}$, which consists of two nuclei bound together by the electrons that orbit about them. Since the nuclei are much heavier than the electrons, we can assume that the latter move fast enough to readjust instantaneously to the changing position of the nuclei (Born-Oppenheimer approximation). The problem is therefore reduced to one in which the motion of the two nuclei is governed by a potential [energy], *V* , depending only upon *r* , the distance between them.

> The physical principles responsible for generating *V* will be discussed in Project VIII, but on general grounds one can say that the potential [energy] is attractive at large distances (van der Waals interaction) and repulsive at short distances (Coulomb interaction of the nuclei and Pauli repulsion of the electrons).
>
> A commonly used form for *V* embodying these features is the Lennard-Jones (or 6-12) potential [energy]:

$$\mathbf{V(r) = 4\,V_0\,\left[\left(\frac{a}{r}\right)^{12} - \left(\frac{a}{r}\right)^{6}\right]}, \tag{3.1}$$

in which $\mathbf{r}$ is the positive distance between the two nuclei, $\mathbf{V_0}$ is some adjustable positive energy, and $\mathbf{a}$ is an adjustable length parameter.

The Lennard-Jones form of the effective potential energy of interaction (responsible for the changes in the radial distance between the two nuclei) is a phenomenological model with two adjustable parameters.

The quasiclassical WKB method seeks energies $\mathbf{E_n}$ such that

$$\mathbf{\int_{r_1}^{r_2} p_n(r)\,dr = (n + \frac{1}{2})\pi\,\hbar} \tag{3.2}$$

or

$$\mathbf{\sqrt{2\,m}\int_{r_1}^{r_2} \sqrt{E_n - V(r)}\,dr = (n + \frac{1}{2})\pi\,\hbar} \tag{3.3}$$

in which the classical turning points $\mathbf{r_1}$ and $\mathbf{r_2}$ are such that $\mathbf{p_n(r) = 0}$ or, equivalently, $\mathbf{E_n = V(r)}$.

We first reduce the problem to dimensionless form, with $\mathbf{v = V/V_0}$, $\boldsymbol{\varepsilon} = \mathbf{E/V_0}$, and $\mathbf{x = r/a}$. Then

$$\mathbf{\sqrt{E_n - V(r)}\,dr = a\,\sqrt{V_0}\,\sqrt{\varepsilon - v(x)}\,dx} \tag{3.4}$$

with the dimensionless potential (energy) being

$$\mathbf{v(x) = 4\,\left[\frac{1}{x^{12}} - \frac{1}{x^{6}}\right]}. \tag{3.5}$$

The quasiclassical quantization condition then takes the form

$$\mathbf{\int_{x_1}^{x_2} \sqrt{\varepsilon_n - v(x)}\,dx = (n + \frac{1}{2})\frac{\pi}{\gamma}} \tag{3.6}$$

in which

$$\boldsymbol{\gamma} = \mathbf{\left[\frac{2\,m\,a^2\,V_0}{\hbar^2}\right]^{1/2}}. \tag{3.7}$$

Quoting Koonin again:

> The quantity γ is a dimensionless measure of the quantum nature of the problem. In the classical limit ($\hbar$ small or $\mathbf{m}$ large), γ becomes large. By knowing the moment of inertia of the molecule (from the energies of its rotational motion) and the dissociation energy (energy required to separate the molecule into its two constituent atoms), it is possible to determine from observation the parameters $\mathbf{a}$ and $\mathbf{V_0}$ and hence the quantity γ.
>
> For the H_2 molecule, $\gamma = 21.7$, while for the HD molecule, $\gamma = 24.8\ldots$ and for the much heavier O_2 molecule made of two ^{16}O nuclei, $\gamma = 150$. These rather large values indicate that a semiclassical approximation is a valid description of the vibrational motion.

The dimensionless energy ε is represented by **`e`** in our code. The dimensionless turning points $\mathbf{x_1}$ (represented by **`xin`**) and $\mathbf{x_2}$ (represented by **`xout`**) are both functions of the dimensionless energy ε, and are defined by the equation $\varepsilon = \mathbf{v(x)}$.

Using **R** to make a plot of $\mathbf{v(x)}$:

```
> fun = function(x) 4*( 1/x^12 - 1/x^6)
> curve(fun,.8,2,ylab = "v",ylim=c(-1,2),lwd=2,col="blue")
> abline(h=0)
```

we get

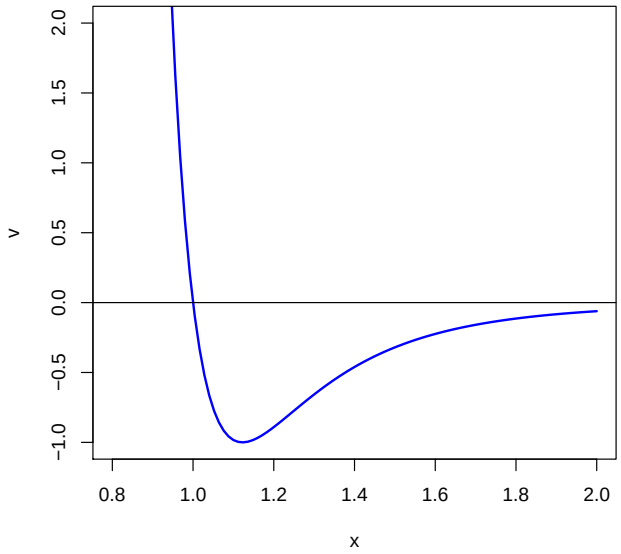

Figure 1: dimensionless Lennard-Jones potential v(x)

3.1 Analytic Positions of the Potential Minimum and Turning Points Using Maxima

We can use Maxima to help find the value of $\mathbf{x}$ at which $\mathbf{v(x)}$ takes on its minimum value (here, where the first derivative of $\mathbf{v(x)}$ is zero).

```
(%i1) v: 1/x^12 - 1/x^6;
(%o1) 1/x^12-1/x^6
(%i2) dv : diff(v,x);
(%o2) 6/x^7-12/x^13
(%i3) dv : factor(dv);
(%o3) 6*(x^6-2)/x^13
```

Solving $\mathbf{x^6 - 2 = 0}$ gives $\mathbf{x_{min} = 2^{1/6}}$. Returning to **R** we evaluate **fun** at this value to get the minimum value of $\mathbf{-1}$.

```
> xmin = 2^(1/6); xmin
[1] 1.122462
> fun(xmin)
[1] -1
```

Hence bound states must have values of the dimensionless energy ε in the range $\mathbf{-1 < \varepsilon < 0}$.

Since $\mathbf{v(x)}$ is proportional to $\left[\frac{\mathbf{1}}{\mathbf{x^{12}}} - \frac{\mathbf{1}}{\mathbf{x^6}}\right]$ or $\frac{\mathbf{1-x^6}}{\mathbf{x^{12}}}$, we see that $\mathbf{x = 1}$ is the only location where $\mathbf{v = 0}$.

Hence the classical turning points $\mathbf{x_1}$ and $\mathbf{x_2}$ are both always greater than $\mathbf{1}$.

The classical turning points $\mathbf{x_1}$ and $\mathbf{x_2}$ are positive numbers (greater than 1) determined by the equation $\mathbf{p(x) = 0}$, or $\boldsymbol{\varepsilon} = \mathbf{v(x)}$, or $\frac{\boldsymbol{\varepsilon}}{\mathbf{4}}\,\mathbf{y^2} + \mathbf{y} - \mathbf{1} = \mathbf{0}$, (in which $\mathbf{y} = \mathbf{x^6}$. Solving by hand via the usual quadratic equation formula, or using Maxima as in (recall that **e**, which represents ε, is a negative number)

```
(%i1) s1: solve((e/4)*y^2 + y -1,y);
(%o1) [y = -(2*sqrt(e+1)+2)/e,y = (2*sqrt(e+1)-2)/e]
(%i2) s1 : factor(s1);
(%o2) [y = -2*(sqrt(e+1)+1)/e,y = 2*(sqrt(e+1)-1)/e]
(%i3) s1by2 : s1/2;
(%o3) [y/2 = -(sqrt(e+1)+1)/e,y/2 = (sqrt(e+1)-1)/e]
```

Our two turning points are then proportional to the positive (1/6) root of these two expressions. We write these dimensionless turning points in terms of **`xmin = 2^(1/6)`**. We anticipate here which root is **`xin`** and which is **`xout`**.

```
(%i4) xin : xmin*(rhs(s1by2[2]))^(1/6);
(%o4) ((sqrt(e+1)-1)/e)^(1/6)*xmin
(%i5) xout : xmin*(rhs(s1by2[1]))^(1/6);
(%o5) (-1)^(1/6)*(sqrt(e+1)+1)^(1/6)*xmin/e^(1/6)
```

We then simplify the definition of **`xout`** by hand:

```
(%i6) xout : (sqrt(e+1)+1)^(1/6)*xmin/(-e)^(1/6);
(%o6) (sqrt(e+1)+1)^(1/6)*xmin/(-e)^(1/6)
```

Finally, we assign **`xmin`** to a floating point number, and evaluate **`xin`** and **`xout`** for **`e = -0.5`**, to check our suppositions:

```
(%i8) xmin : 2^(1/6),numer;
(%o8) 1.122462048309373
(%i9) xin,e = -1/2, numer,expand;
(%o9) 1.026742528828304
(%i10) xout,e = -1/2, numer,expand;
(%o10) 1.377378965676005
```

So we satisfy **`xout > xin`** .

If we were unable to find analytic expressions for the turning points as expressions depending on the dimensionless energy ε, we would need to resort to numerical root searches for each energy. We show how to set up such code at the end of this example.

3.2 Plots of Dimensionless Potential and an Energy level Using R

We now use **R** to show the potential and a hypothetical energy level, with some extra lines and text labels. We have already defined (in **R**) **`xmin`** and **`fun`**, the latter the function which describes the dimensionless form of the Lennard-Jones potential.

```
> e = -1/2; e
[1] -0.5
> xin = xmin*(sqrt(e+1)/e-1/e)^(1/6); xin
[1] 1.026743
> xout = xmin*(sqrt(e+1)+1)^(1/6)/(-e)^(1/6); xout
[1] 1.377379
```

We then use **Notepad2** to construct a plot text file called **ex2.R** with the contents:

```
## ex2.R Lennard-Jones Potential
curve(fun,.8,2,ylab = "v",ylim=c(-1,2),
         lwd=2,col="blue", las = 1)
## black lines:
abline(h=0, v=1)
##  red line for energy level:
lines(c(xin,xout),c(e,e),col="red",lwd=2)
##   black lines for xin and xout
lines(c(xin,xin),c(e,0) )
lines(c(xout,xout),c(e,0))
##  text labels for xin and xout
text(xin, 0.1, "xin", adj = 0,font=2)
text(xout, 0.1, "xout",font=2)
```

We then use the **R** function **source** to "load and execute" this code for constucting a plot, using the previously defined parameters **e**, **xmin**, **xin**, **xout**, and the function **fun**.

```
> source("c:/k1/ex2.R")
```

We can then go back and forth from **R** to our file **ex2.R**, and experiment with small changes to the commands, and easily see the overall results each time, continuing to issue **source("c:/k1/ex2.R")** to redraw the plot from scratch.

The version defined by the above code produces:

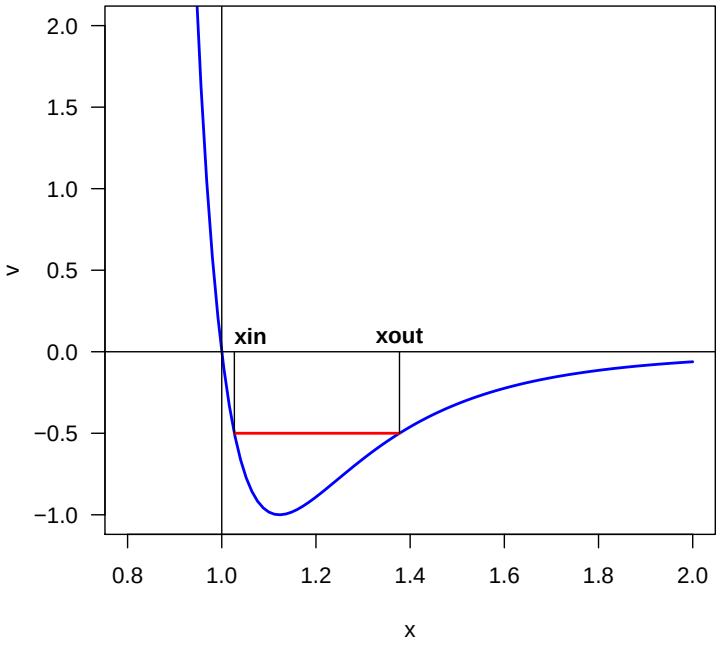

Figure 2: Lennard-Jones potential with one energy level

We can fiddle with the y-axis label by using the **R** function **par** to increase the margin on the left side of the plot, using **par(mar = c(bottom,left,top,right))**, and **mtext** to write in the margins.

We also rotate the y-axis tick mark values to horizontal, and the y-axis label to horizontal by using the option **las = 1**. We also use the option **font=2** to get a bold label. The option **side=2** specifies the left side of the plot.

Experimenting with plot parameter settings is easier if we construct a small script file **ex3.R** with the contents:

```
##  ex3.R
##  plot of lennard jones (6-12) potential
##    with energy level, xin, xout
##    add horizontal v(x) y-axis label
oldpar = par(mar=c(5.1,4.1,4.1,2.1))
##  mar = c(bottom,left,top,right)
par(mar = c(5.1,6.1,4.1,2.1))   # add extra margin area on left side
curve(fun,.8,2,ylab = "",ylim=c(-1,2),lwd=2,
         col="blue",las=1,cex.lab= 1.6)
## black lines
abline(h=0,v=1)
##  red line for energy level:
lines(c(xin,xout),c(e,e),col="red",lwd=2)
##  solid black lines for xin and xout
lines(c(xin,xin),c(e,0) )
lines(c(xout,xout),c(e,0) )
##  text labels for xin and xout
text(xin, 0.1, "xin", adj = 0,font=2)
text(xout, 0.1, "xout",font=2)
## horizontal y-axis label
mtext("v(x)",side=2,las=1,line=3,font=2,cex=1.6)
## restore default par settings
par(oldpar)
```

All of this work (which is usually not worth the effort) results in

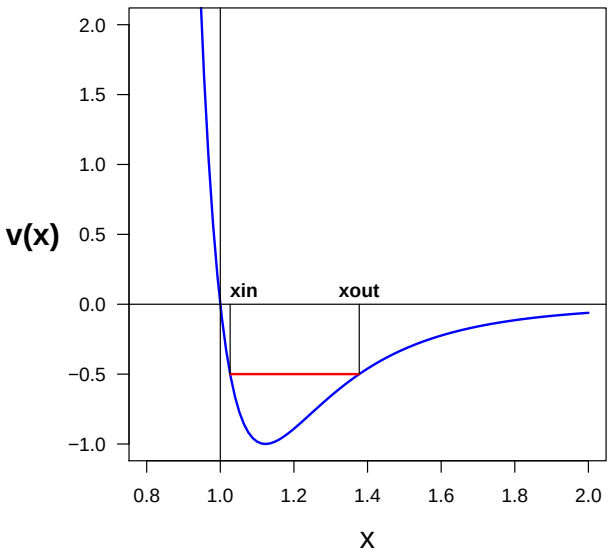

Figure 3: Lennard-Jones potential with one energy level

4 Solving for Energy Levels Using Maxima's find_root and quad_qags Functions

We will use the Maxima functions **find_root** and **quad_qags** in order to find the values of the dimensionless energy ε such that (for a given value of $\mathbf{n}$) $\mathbf{wkb}(\gamma, \varepsilon, \mathbf{n}) = \mathbf{0}$ (see Eq.(3.6)), where

$$\mathbf{wkb}(\gamma, \varepsilon, \mathbf{n}) = \int_{\mathbf{x_1}(\varepsilon)}^{\mathbf{x_2}(\varepsilon)} \sqrt{\varepsilon - \mathbf{v}(\mathbf{x})}\, \mathbf{dx} - (\mathbf{n} + \frac{\mathbf{1}}{\mathbf{2}})\frac{\pi}{\gamma} \tag{4.1}$$

is represented by **wkb(gam,e,n)** in our code. **gam** is used to represent γ in our code.

```
wkb(gam,e,n) :=
block([x,v,xmin,xin,xout,numer,qlist],numer:true,
        if e <= -1.0 then error(" e must be greater than -1 "),
        if e >= 0.0 then error(" e must be less than zero "),
        xmin : 2^(1/6),
        v : 4*(1/x^12 - 1/x^6),
        xin : xmin*(sqrt(e+1)/e-1/e)^(1/6),
        xout : xmin*(sqrt(e+1)+1)^(1/6)/(-e)^(1/6),
        qlist: quad_qags(sqrt(e - v),x,xin,xout),
        if qlist[4] # 0 then error(" quad_qags errcode = ",qlist[4]),
        qlist[1] - (n+1/2)*%pi/gam)$
```

Here we use **wkb** with **find_root** after pasting this definition into XMaxima.

```
(%i2) fpprintprec:8$
(%i3) find_root('wkb(50,e,0),e,-.99,-5e-4);
(%o3) -0.896672
(%i4) wkb(50,%,0);
(%o4) -1.38777878E-17
(%i5) find_root('wkb(50,e,1),e,%th(2),-5e-4);
(%o5) -0.710907
(%i6) wkb(50,%,1);
(%o6) 1.38777878E-16
```

In the above we assumed **gam = 50** and assumed the ground state was greater than **-0.99** and less than **-5e-4** (negative but close to zero), and found the dimensionless energy **e = -0.896672** for the **n = 0** solution (ie., the ground state).

Note also the **single quote** in **'wkb(50,e,0)**, when supplied as the first slot of **find_root**, which prevents immediate evaluation of **wkb**, letting **find_root** take charge of using and evaluating the function.

In the step **(%i4)**, we evaluated **wkb(50,e,0)** at the dimensionless energy found (the root of the function), and found that the result was numerically close to zero (remember we are using 16 digit arithmetic in Maxima, the default).

In the step **(%i5)** we sought the **n = 1** case energy value, which is assumed to lie in the range **-0.896672 < e < -5e-4** . We used the Maxima function **%th(2)** which gets the second previous output, rather than the previous output found by using **%**.

If you have (or might have!) a global value for **e** set, then you should *quote* **e**, as in **'e**, both places it appears, so the call to **find_root** would look like **find_root('wkb(50,'e,0),'e,-.99,-5e-4);**. The quote **'** prevents immediate evaluation in the global environment, and preserves the meaning of **e** as a symbol in the call to **find_root**.

It will be easier to shift the bottom starting energy used for the root search if we design a function **find_e(gam,ebott,etop,n)**.

```
find_e(gam,ebott,etop,n) := (find_root('wkb(gam,ee,n),ee,ebott,etop))$
```

and after pasting this definition into XMaxima, we get

```
(%i8) find_e(50,-0.99,-5e-4,0);
(%o8) -0.896672
(%i9) find_e(50,%,-5e-4,1);
(%o9) -0.710907
(%i10) find_e(50,%,-5e-4,2);
(%o10) -0.551659
```

We can then design a function **levels(gam,num)** which calculates the first **num** energy levels based on the chosen value of **gam**, beginning with the ground state, and prints out the corresponding values of **n** and **e**.

```
levels(gam,num) :=
block([bott:-0.99,top:-5e-4,nn,en],
    for nn:0 thru (num-1) do (
        en : find_e(gam,bott,top,nn),
        print("  ",nn,"  ",en),
        bott : en))$
```

and after pasting this definition into XMaxima, we get

```
(%i12) levels(50,6);
   0    -0.896672
   1    -0.710907
   2    -0.551659
   3    -0.41725
   4    -0.305917
   5    -0.215801
(%o12) done
(%i13) time(%);
(%o13) [3.03]
```

with the last step indicating a time of about 3 seconds to execute the command **levels(50,6)**.

If we want to print out *also* the value of **wkb(gam,e,n)** at the root found, *and* the values of the turning points, we need to do more work, as in **levels_info**:

```
levels_info(gam,num) :=
block([bott:-0.99,top:-5e-4,nn,en,xmin],local(x1,x2),numer:true,
    xmin : 2^(1/6),
    x1(e) := xmin*(sqrt(e+1)/e-1/e)^(1/6),
    x2(e) := xmin*(sqrt(e+1)+1)^(1/6)/(-e)^(1/6),
    for nn:0 thru (num-1) do (
        en : find_e(gam,bott,top,nn),
        print("  ",nn,"  ",en," ",abs(wkb(gam,en,nn))," ",x1(en)," ",x2(en)),
        bott : en))$
```

```
(%i15) levels_info(50,2);
   0    -0.896672   1.38777878E-17   1.0715111   1.197405
   1    -0.710907   1.38777878E-16   1.0447867   1.2764788
(%o15) done
(%i16) x1(-0.5);
(%o16) x1(-0.5)
(%i17) xmin;
(%o17) xmin
```

In the next to last step, we find that the function **x1(e)**, defined inside the function **levels_info**, remains unknown at the global level; this is due to the separate **local(x1,x2)** declaration used inside the definition of **levels_info**.

4.1 Energy Level Diagram Using Maxima's plot2d

To make a plot of energy levels (in addition to the **`n, energy`** printouts), we can use:

```
levels_plot(gam,num) :=
block([bott:-0.99,top:-5e-4,nn,en,level_list:[]],
    for nn:0 thru (num-1) do (
        en : find_e(gam,bott,top,nn),
        print("  ",nn,"  ",en),
        level_list : cons(en, level_list),
        bott : en),
    plot2d(reverse(level_list),[x,0,1],[y,-1,0],[style,[lines,3,1],
               [lines,3,2],[lines,3,3],[lines,3,4]],
               [xlabel,""],[ylabel,"energy"],[legend,false]))$
```

and after pasting into XMaxima,

```
(%i19) levels_plot(50,6);
   0    -0.896672
   1    -0.710907
   2    -0.551659
   3    -0.41725
   4    -0.305917
   5    -0.215801
(%o19) ""
```

which produces the energy level diagram

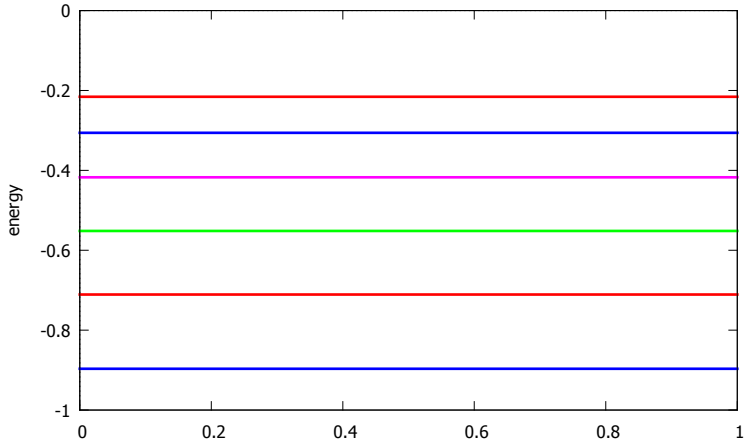

Figure 4: bottom six energy levels: gamma = 50

The eps file for inclusion in this tex file was produced by adding two extra options to the above interactive code and changing **`[lines,3,n]`** to **`[lines,5,n]`**. The two extra options added were:

```
    [gnuplot_out_file,"c:/k1/ex5.eps"],
    [gnuplot_term,'eps]
```

4.2 A Maxima Method Using a Search for the Turning Points

Koonin's code searches for the location of the turning points for a given energy (rather than use analytic expressions). His code, as a consequence, gains in versatility, since one can use the code with other potentials (with suitable scaling so that the dimensionless potential has a minimum value of -1, and the corrected value of **`xmin`** is used) for which analytic turning point expressions are not *a priori* known.

Let's first design code to find the **xin** turning point, given the dimensionless energy **e** and a starting location **xstart** which is assumed to be greater than **xin**.

```
find_xin(e,xstart) :=
block([x,fstart,xnew,fnew,dx:0.001,numer],local(fdiff),numer:true,
    fdiff(x) := e - 4*(1/x^12 - 1/x^6),
    fstart : fdiff(xstart),
    xnew : xstart - dx,
    fnew : fdiff(xnew),
    do (if fnew*fstart < 0 then return(),
        xnew : xnew - dx,
        fnew : fdiff(xnew)),
    find_root('fdiff(x),x,xnew,xstart))$
```

and after pasting this definition into Maxima,

```
(%i2) find_xin(-0.5,1.122);
(%o2) 1.026742528828304
```

A similar style of code can find **xout**.

```
find_xout(e,xstart) :=
block([x,fstart,xnew,fnew,dx:0.001,numer],local(fdiff),numer:true,
    fdiff(x) := e - 4*(1/x^12 - 1/x^6),
    fstart : fdiff(xstart),
    xnew : xstart + dx,
    fnew : fdiff(xnew),
    do (if fnew*fstart < 0 then return(),
        xnew : xnew + dx,
        fnew : fdiff(xnew)),
    find_root('fdiff(x),x,xstart,xnew))$
```

and after pasting this definition into Maxima,

```
(%i4) find_xout(-0.5,1.122);
(%o4) 1.377378965676005
```

We redesign and rename **wkb(gam,e,n)** as **wkb1(gam,e,n)** which calls **find_xin** and **find_xout**:

```
wkb1(gam,e,n) :=
block([xmin,xin,xout,x,v,qlist,numer],numer:true,
    xmin : 2^(1/6),
    xin : find_xin(e,xmin),
    xout : find_xout(e,xmin),
    v : 4*(1/x^12 - 1/x^6),
    qlist: quad_qags(sqrt(e - v),x,xin,xout),
          if qlist[4] # 0 then error(" quad_qags errcode = ",qlist[4]),
          qlist[1] - (n+1/2)*%pi/gam)$
```

and after pasting this definition into Maxima,

```
(%i6) wkb1(50,-0.5,2);
(%o6) 0.0228478
```

We redesign and rename **find_e(gam,ebott,etop,n)** as **find1_e(gam,ebott,etop,n)** which calls **wkb1**.

```
find1_e(gam,ebott,etop,n) := (find_root('wkb1(gam,ee,n),ee,ebott,etop))$
```

and after pasting in this definition, we get

```
(%i8) find1_e(50,-0.99,-5e-4,0);
(%o8) -0.896672
```

We next redesign and rename **levels(gam,num)** as **levels1(gam,num)** which calls **find1_e**.

```
levels1(gam,num) :=
block([bott:-0.99,top:-5e-4,nn,en],
    for nn:0 thru (num-1) do (
        en : find1_e(gam,bott,top,nn),
        print("  ",nn,"  ",en),
        bott : en))$
```

and after pasting in this definition, we get

```
(%i10) levels1(50,6);
   0    -0.896672
   1    -0.710907
   2    -0.551659
   3    -0.41725
   4    -0.305917
   5    -0.215801
(%o10) done
(%i11) time(%);
(%o11) [5.83]
```

with the last step indicating a time of about 6 seconds for a method incorporating turning point searches, which is about twice the time required for a method based on the analytic turning points use.

5 Solving for Energy Levels Using R's uniroot and integrate Functions

We will use the R functions **uniroot** and **integrate** in order to find the values of the dimensionless energy ε such that (for a given value of $\mathbf{n}$) $\mathbf{wkb}(\gamma, \varepsilon, \mathbf{n}) = \mathbf{0}$ (see Eq.(3.6)), where

$$\mathbf{wkb}(\gamma, \varepsilon, \mathbf{n}) = \int_{\mathbf{x_1}(\varepsilon)}^{\mathbf{x_2}(\varepsilon)} \sqrt{\varepsilon - \mathbf{v}(\mathbf{x})}\, \mathbf{dx} - (\mathbf{n} + \frac{\mathbf{1}}{\mathbf{2}})\frac{\pi}{\gamma} \tag{5.1}$$

is represented by **wkb(gam,e,n)** in our code. The symbol **gam** represents γ. The **R** function **integrate** is used to integrate between the turning points in the **R** function **wkb(gam,e,n)**.

```
wkb = function(gam,e,n) {
        if (e <= -1.0) stop(" e must be greater than -1 ")
        if (e >= 0.0)  stop(" e must be less than zero ")
        xmin = 2^(1/6)
        vfun = function(x) 4*(1/x^12 - 1/x^6)
        xin = xmin*(sqrt(e+1)/e-1/e)^(1/6)
        xout = xmin*(sqrt(e+1)+1)^(1/6)/(-e)^(1/6)
        integrand = function(x) sqrt(e - vfun(x))
        nint = integrate(integrand,xin,xout,abs.tol=1e-14,subdivisions=500L)$value
        nint - (n + 1/2)*pi/gam}
```

After pasting in the above code, we get

```
> options(digits=8)
> root = uniroot(function(e) wkb(50,e,0),c(-0.99,-5e-4),tol=1e-14)$root;root
[1] -0.89667158
> wkb(50,root,0)
[1] 6.9388939e-18
```

```
> root = uniroot(function(e) wkb(50,e,1),c(root,-5e-4),tol=1e-14)$root;root
[1] -0.71090663
> wkb(50,root,1)
[1] -9.7144515e-17
```

We assumed **gam = 50** and assumed the ground state was greater than **-0.99** and less than **-5e-4** (negative but close to zero), and found the dimensionless energy **e = -0.89667158** for the **n = 0** solution (ie., the ground state).

In the next step, we evaluated **wkb(50,e,0)** at the dimensionless energy found (the root of the function), and found that the result was numerically close to zero (remember we are using 16 digit arithmetic in **R**, the default).

In the next step we sought the **n = 1** case energy value, which is assumed to lie in the range
-0.896672 < e < -5e-4 .

It will be easier to shift the bottom starting energy used for the root search if we design a function **find.e(gam,ebott,etop,n)**.

```
find.e = function(gam,ebott,etop,n) uniroot(function(e) wkb(gam,e,n),
                 c(ebott,etop),tol=1e-14)$root
```

with the behavior

```
> find.e(50,-0.99,-5e-4,0)
[1] -0.89667158
> find.e(50,.Last.value,-5e-4,1)
[1] -0.71090663
> find.e(50,.Last.value,-5e-4,2)
[1] -0.5516587
```

We can then design a function **levels(gam,num)** which calculates the first **num** energy levels based on the chosen value of **gam**, beginning with the ground state, and prints out the corresponding values of **n** and **e**.

```
levels = function(gam,num) {
    bott = -0.99
    top = -5e-4
    for (nn in 0:(num-1)){
        en = find.e(gam,bott,top,nn)
        cat("  ",nn,"  ",en,"\n")
        bott = en}}
```

with the behavior:

```
> system.time(levels(50,6))
   0    -0.89667158
   1    -0.71090663
   2    -0.5516587
   3    -0.41725034
   4    -0.30591728
   5    -0.2158009
   user  system elapsed
   0.02    0.00    0.01
```

We have wrapped the function **levels** with the **R** function **system.time()**.

If we want to print out also the value of **wkb(gam,e,n)** at the root found, and the values of the turning points, we need to do more work, as in **levels.info**:

```
levels.info = function(gam,num) {
    bott = -0.99
    top = -5e-4
    xmin = 2^(1/6)
    x1 = function(e) xmin*(sqrt(e+1)/e-1/e)^(1/6)
    x2 = function(e) xmin*(sqrt(e+1)+1)^(1/6)/(-e)^(1/6)
    for (nn in 0:(num-1)) {
        en = find.e(gam,bott,top,nn)
        cat("  ",nn,"  ",en," ",abs(wkb(gam,en,nn))," ",x1(en)," ",x2(en),"\n")
        bott = en}}
```

with the behavior:

```
> levels.info(50,2)
   0    -0.89667158   6.9388939e-18   1.0715111   1.197405
   1    -0.71090663   9.7144515e-17   1.0447867   1.2764788
> x1(-0.5)
Error: could not find function "x1"
> xmin
Error: object 'xmin' not found
```

In the next to last step, we find that the function **x1(e)**, defined inside the function **levels.info**, remains unknown at the global level. Likewise **xmin** is not known at the global level.

5.1 Energy Level Diagram Using R

To make a plot of energy levels (in addition to the **n, energy** printouts), we can use:

```
levels.plot = function(gam,num) {
    bott = -0.99
    top = -5e-4
    plot(0:1, -1:0, type="n",xlab="",ylab="energy")
    for (nn in 0:(num-1)){
        en = find.e(gam,bott,top,nn)
        cat("  ",nn,"  ",en,"\n")
        abline(h = en,col = "blue",lwd=2)
        bott = en}}
```

```
> levels.plot(50,6)
   0    -0.89667158
   1    -0.71090663
   2    -0.5516587
   3    -0.41725034
   4    -0.30591728
   5    -0.2158009
```

which produces the plot

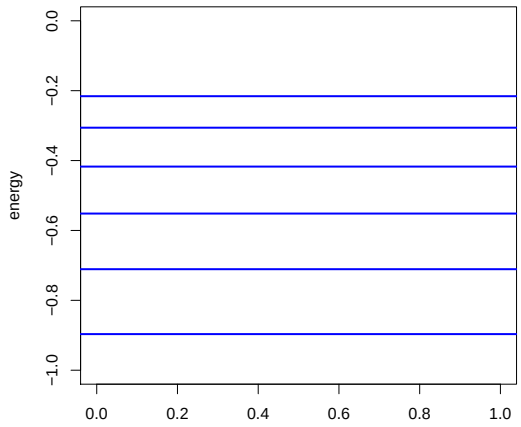

Figure 5: bottom six energy levels: gamma = 50

5.2 A R Method Using a Search for the Turning Points

Koonin's code searches for the location of the turning points for a given energy (rather than use analytic expressions). His code, as a consequence, gains in versatility, since one can use the code with other potentials (with suitable scaling so that the dimensionless potential has a minimum value of -1, and the corrected value of **xmin** is used) for which analytic turning point expressions are not *a priori* known.

Let's first design code to find the **xin** turning point, given the dimensionless energy **e** and a starting location **xstart** which is assumed to be greater than **xin**.

```
find.xin = function(e,xstart) {
    dx = 0.001
    fdiff = function(x)  e - 4*(1/x^12 - 1/x^6)
    fstart = fdiff(xstart)
    xnew = xstart - dx
    fnew = fdiff(xnew)
    repeat {
        if (fnew*fstart < 0)  break
        xnew = xnew - dx
        fnew = fdiff(xnew)}
    uniroot(fdiff,c(xnew,xstart),tol=1e-14)$root}
```

with the behavior

```
> find.xin(-0.5,1.122)
[1] 1.0267425
```

A similar style of code can find **xout**.

```
find.xout = function(e,xstart) {
    dx = 0.001
    fdiff = function(x)  e - 4*(1/x^12 - 1/x^6)
    fstart = fdiff(xstart)
    xnew = xstart + dx
```

```
    fnew = fdiff(xnew)
    repeat {
        if (fnew*fstart < 0)  break
        xnew = xnew + dx
        fnew = fdiff(xnew)}
    uniroot(fdiff,c(xstart,xnew),tol=1e-14)$root}
```

with the behavior:

```
> find.xout(-0.5,1.122)
[1] 1.377379
```

We redesign and rename **wkb(gam,e,n)** as **wkb1(gam,e,n)** which calls **find.xin** and **find.xout**:

```
wkb1 = function(gam,e,n) {
        xmin = 2^(1/6)
        xin = find.xin(e,xmin)
        xout = find.xout(e,xmin)
        vfun = function(x) 4*(1/x^12 - 1/x^6)
        integrand = function(x) sqrt(e - vfun(x))
        nint = integrate(integrand,xin,xout,abs.tol=1e-14,subdivisions=500L)$value
        nint - (n + 1/2)*pi/gam}
```

with the behavior:

```
> wkb1(50,-0.5,2)
[1] 0.022847833
```

We redesign and rename **find.e(gam,ebott,etop,n)** as **find1.e(gam,ebott,etop,n)** which calls **wkb1**.

```
find1.e = function(gam,ebott,etop,n) uniroot(function(e) wkb1(gam,e,n),
                 c(ebott,etop),tol=1e-14)$root
```

with the behavior

```
> find1.e(50,-0.99,-5e-4,0)
[1] -0.89667158
```

We next redesign and rename **levels(gamma,num)** as **levels1(gamma,num)** which calls **find1.e**.

```
levels1 = function(gam,num) {
    bott = -0.99
    top = -5e-4
    for (nn in 0:(num-1)){
        en = find1.e(gam,bott,top,nn)
        cat("  ",nn,"  ",en,"\n")
        bott = en}}
```

with the behavior:

```
> system.time(levels1(50,6))
   0    -0.89667158
   1    -0.71090663
   2    -0.5516587
   3    -0.41725034
   4    -0.30591728
   5    -0.2158009
   user  system elapsed
   0.44    0.00    0.44
```

having wrapped **levels1** in the **R** function **system.time**, to show the increase in time compared to the method which uses analytic expressions for the turning points.