Quantum Mechanics with Python

Numerical Methods for Physicists, Lecture 4

Matúš Medo, Yi-Cheng Zhang

Physics Department, Fribourg University, Switzerland

March 11, 2013

Schrödinger equation (here in 1D)

$$i\hbar \frac{\partial \Psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \Psi(x,t)}{\partial x^2} + V(x)\Psi(x,t)$$

- PDE with complex numbers
- `scipy.integrate.complex_ode` re-maps a given complex-valued differential equation to a set of real-valued ones and uses `scipy.integrate.ode` to integrate them
- But we have partial derivatives...

Schrödinger equation (here in 1D)

$$i\hbar \frac{\partial \Psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \Psi(x,t)}{\partial x^2} + V(x)\Psi(x,t)$$

- PDE with complex numbers
- `scipy.integrate.complex_ode` re-maps a given complex-valued differential equation to a set of real-valued ones and uses `scipy.integrate.ode` to integrate them
- But we have partial derivatives...
  1. Do re-mapping to real values and use Finite-Difference Time-Domain (FDTD) as we did for the heat equation
  2. Use the split-step Fourier method
  3. QuTiP: a quantum toolbox in Python
  4. Re-mapping & Escript (fast finite elements for PDEs)

FDTD: re-mapping

- Re-mapping to real-valued variables:

$$\Psi(x,t) = \phi(x,t) + i\xi(x,t)$$

- Schrödinger equation decouples:

$$\hbar \frac{\partial \phi(x,t)}{\partial t} = -\frac{\hbar^2}{2m} \frac{\partial^2 \xi(x,t)}{\partial x^2} + V(x)\xi(x,t)$$
$$\hbar \frac{\partial \xi(x,t)}{\partial t} = \frac{\hbar^2}{2m} \frac{\partial^2 \phi(x,t)}{\partial x^2} - V(x)\phi(x,t)$$

- Discretization in time and space: $x_l = l\,\Delta x$, $t_n = n\,\Delta t$

$$\phi(x_l, t_n) := \phi^n(l), \quad \xi(x_t, t_n) := \xi^n(l)$$

FDTD: discretization

- Real and complex part of Ψ shifted by half-interval from each other—this allows us to use the second-order approximation for time derivatives which is more precise ($O(\Delta t)$ vs $O(\Delta t^2)$)

$$\frac{\partial \phi(x_l, t_{n+1/2})}{\partial t} \approx \frac{\phi(x_l, t_{n+1}) - \phi(x_l, t_n)}{\Delta t} = \frac{\phi^{n+1}(l) - \phi^n(l)}{\Delta t}$$

$$\frac{\partial \xi(x_l, t_n)}{\partial t} \approx \frac{\xi(x_l, t_{n+1/2}) - \xi(x_l, t_{n-1/2})}{\Delta t} = \frac{\xi^{n+1/2}(l) - \xi^{n-1/2}(l)}{\Delta t}$$

- Spatial derivatives exactly as before

$$\frac{\partial^2 \phi^n(l)}{\partial x^2} \approx \frac{\phi^n(l+1) - 2\phi^n(l) + \phi^n(l-1)}{\Delta x^2}$$

FDTD: equations

- Compute the future state from the current state

$$\{\phi^n, \xi^{n-1/2}\} \rightarrow \{\phi^{n+1}, \xi^{n+1/2}\}$$

- Update equations are

$$\xi^{n+1/2}(l) = \xi^{n-1/2}(l) - c_2 V(l)\phi^n(l) + \\ + c_1\left[\phi^n(l-1) - 2\phi^n(l) + \phi^n(l+1)\right]$$

$$\phi^{n+1}(l) = \phi^n(l) + c_2 V(l)\xi^{n+1/2}(l) - \\ - c_1\left[\xi^{n+1/2}(l-1) - 2\xi^{n+1/2}(l) + \xi^{n+1/2}(l+1)\right]$$

- Where $c_1 := \hbar\Delta t/(2m\Delta x^2)$ and $c_2 := \Delta t/\hbar$

FDTD: practical issues 1

- It is easier to work in units where $\hbar = m = 1$
- The scheme is stable only if $\Delta t \leq \hbar \left(\frac{\hbar^2}{m \Delta x^2} + \frac{\max_{x,t} V(x,t)}{2} \right)^{-1}$
  - See http://www.scipy.org/Cookbook/SchrodingerFDTD (there is the rest of theory as well)
  - This is only orientational (*e.g.*, $V(x) \to \infty$ somewhere does not necessarily mean $\Delta t \to 0$)
  - Finer grid $\implies$ smaller time steps needed
  - To reach the same physical time, simulation time grows as $1/\Delta x^3$
  - Check whether we are really in the stable (convergent) regime: halve Δt and see whether the results change substantially (they should not)
- We are typically interested in the probability density $\Psi^* \Psi$, not in ϕ and ξ themselves

FDTD: practical issues 2

- It's nice to have a reversible algorithm (as in the leap-frog kind of integration schemes)
  - Evolve your system to the future and back and get the initial state
- Evolved wave-function needs to stay normalized
  - Force normalization by hand after each step (possible but non-systematic)
  - Find a "unitary" algorithm which preserves the norm
    - See Numerical recipes in C (Section 19.2) by Press *et al*
    - We approximated the time evolution operator $U(t) = \exp[-i\mathrm{H}t]$ as $U(t) \approx 1 - i\mathrm{H}t$ whereas it is better to use

$$U(t) = \exp[-i\mathrm{H}t] \approx \frac{1 - \frac{1}{2}i\mathrm{H}t}{1 + \frac{1}{2}i\mathrm{H}t}$$

    - Then naturally $U(\Delta t)U(-\Delta t) = 1$

Detour: animations with Python

- Sometimes better than static figures
- Especially valuable if you want to impress...
- First approach:
  - Write a script that saves many figures, say `frame-001.gif`, `frame-002.gif`, etc.
  - Use `avconv` (formerly `ffmpeg`) to create an animation
    `avconv -an -r 4 -i frame-%03d.png animation.avi`
  - Or use `gifsicle` to create an animated gif
    `gifsicle -delay=100 -loop -no-transparent -O2 list_of_files > output.gif`
- Second approach: animate with Matplotlib (v1.1 and higher)
  - http://jakevdp.github.com/blog/2012/08/18/matplotlib-animation-tutorial/

Split-step Fourier method: the idea

- The potential part of the Schrödinger equation is simple to solve

$$i\hbar \frac{\partial \Psi}{\partial t} = V(x)\Psi \implies \Psi(x, t + \Delta t) \overset{pot}{=} \Psi(x, t)\mathrm{e}^{-iV(x)\Delta t/\hbar}$$

- Inverse Fourier transform of Ψ

$$\Psi(x, t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{\Psi}(k, t)\mathrm{e}^{ikx}\, \mathrm{d}k$$

- By substituting this into the Schrödinger equation

$$i\hbar \frac{\partial \tilde{\Psi}}{\partial t} = \frac{\hbar^2 k^2}{2m}\tilde{\Psi} + i\tilde{V}\frac{\partial \tilde{\Psi}}{\partial k} \implies \tilde{\Psi}(k, t + \Delta t) \overset{kin}{=} \tilde{\Psi}(k, t)\mathrm{e}^{-i\hbar k^2 \Delta t/2m}$$

- Fast Fourier transform (FFT) allows us to switch between Ψ and $\tilde{\Psi}$

Split-step Fourier method: the algorithm

1. Discretize the x and k-space by choosing a, b, N, k_0:

$$\Delta x = (b-a)/N, \quad \Delta k = 2\pi/(b-a), \quad k_0 = -\pi/\Delta x,$$
$$x_n = a + n\,\Delta x, \quad k_m = k_0 + m\,\Delta k$$

   - This should be sufficient to represent states of Ψ (initial and later)
   - The choice can be tricky, experiment a bit to see if it works fine

2. Discretize the wave-functions:

$$\Psi_n(t) := \Psi(x_n, t), \quad V_n := V(x_n), \quad \tilde{\Psi}_m(t) = \tilde{\Psi}(k_m, t)$$

3. Evolve the system by one time step

4. Repeat 3 until the end time is reached

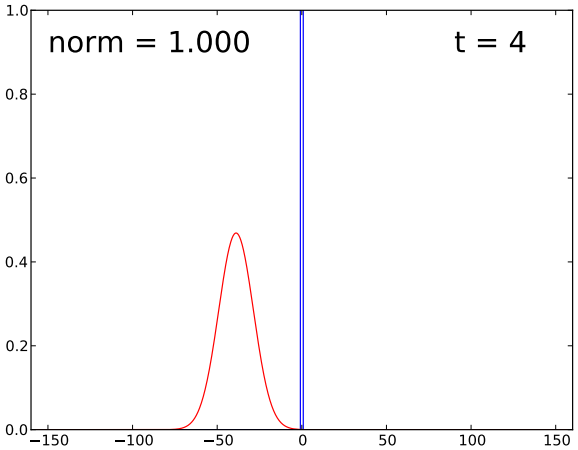
Split-step Fourier method: point 3

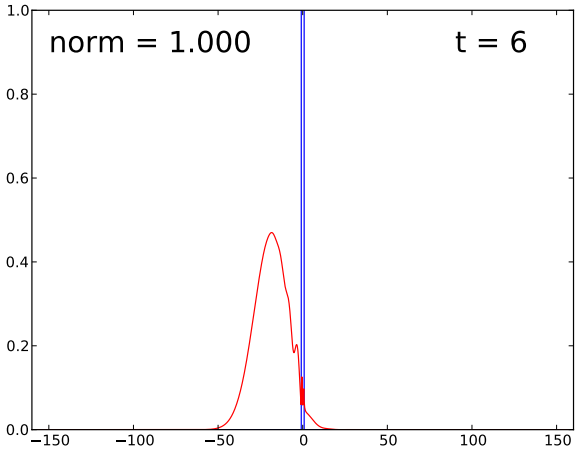
1. A half step in x: $\Psi_n \leftarrow \Psi_n \exp[-iV_n \Delta t/2\hbar]$
2. FFT: computes $\tilde{\Psi}_m$ from Ψ_n
3. A full step in k: $\tilde{\Psi}_m \leftarrow \tilde{\Psi}_m \exp[-i\hbar k^2 \Delta t/2m]$
4. Inverse FFT: computes Ψ_n from $\tilde{\Psi}_m$
5. A second half step in x: $\Psi_n \leftarrow \Psi_n \exp[-iV_n \Delta t/2\hbar]$

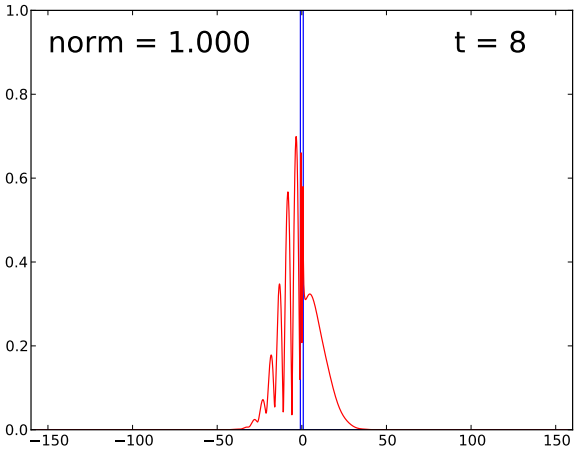
- Splitting the x-step in two parts produces a numerically more stable algorithm (as in "leapfrog" integration in mechanics which is reversible and conserves energy)
- See http://jakevdp.github.com/blog/2012/09/05/quantum-python/ for more details

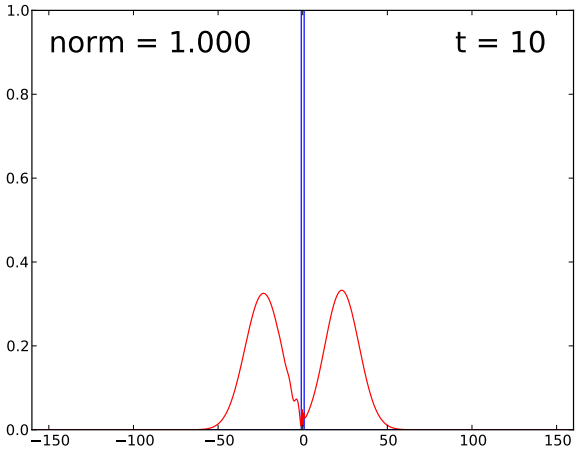
Split-step Fourier method: tunelling

Output of `ex4-1.py` (see also the animation at the url above):







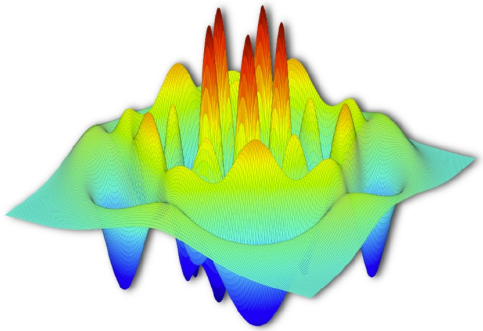


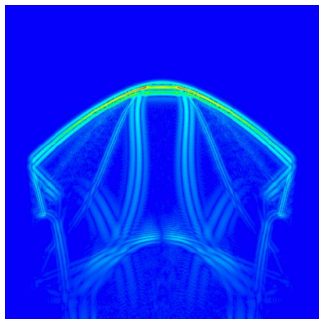
Fast Fourier transform

- Discrete Fourier transform is rather slow to compute: $O(N^2)$ where N is the number of points where the transform is computed
- The beauty of FFT: $O(N^2)$ is reduced to $O(N \log N)$ (similar as quicksort improving over bubble sort and alike)
- The idea behind: discrete Fourier transform with N points can be written as a sum of two transforms with $N/2$ points
  - Using this recursively, we move to $N/4$ points, etc.
  - That's why it's best to use FFT for N which is a power of two
- `fft` and `ifft` from `scipy.fftpack` implement it
- FFT works best when is a power of two (because of how it is constructed)
  - $N \neq 2^x$ is possible but usually leads to slower computation

QuTiP & Escript

- QuTiP:
  - “QM is not as difficult as one might think: It’s only linear algebra!”
  - States, operators, time evolution, gates, visualization
  - http://qutip.blogspot.ch and http://code.google.com/p/qutip
- Escript:
  - For non-linear, time-dependent partial differential equations
  - https://launchpad.net/escript-finley





What about the stationary states?

- Until now we addressed only time evolution in QM
- Stationary state Ψ_n satisfies

$$\mathrm{H}\Psi_n = E_n\Psi_n \iff \left(-\frac{\hbar^2}{2m}\nabla^2 + V(\boldsymbol{r})\right)\Psi_n = E_n\Psi_n$$

- Boundary conditions in a box: $\Psi(a) = \Psi(b) = 0$
  - We fix $\Psi(a) = 0$ and find E so that $\Psi(b)$ is zero too
- In open space: $\lim_{x\to-\infty}\Psi(x) = \lim_{x\to\infty}\Psi(x) = 0$
  - For a symmetric potential, we simplify by realizing that eigenstates are also symmetric
  - Even eigenstates: $\Psi'(0) = 0$; odd eigenstates: $\Psi(0) = 0$
  - We need E for which $\Psi(x)$ does go to zero as x grows

Stationary states: integration

- Again $\Psi(x_n) := \Psi_n$, $V(x_n) = V_n$
- In addition, $\hbar = m = 1$ and $h = \Delta x$
- The approximation for $f''(x)$ plus the Schrödinger equation

$$-\frac{1}{2}\Psi_n'' + V_n\Psi_n = E\Psi_n \implies \Psi_n'' = 2(V_n - E)\Psi_n$$

$$\Psi_n'' = \frac{\Psi_{n-1} - 2\Psi_n + \Psi_{n+1}}{h^2} \implies \Psi_{n+1} = 2\Psi_n - \Psi_{n-1} + 2(V_n - E)h^2$$

  - Once Ψ_0 and Ψ_1 are given, the rest can be computed

Stationary states: integration

- Again $\Psi(x_n) := \Psi_n$, $V(x_n) = V_n$
- In addition, $\hbar = m = 1$ and $h = \Delta x$
- The approximation for $f''(x)$ plus the Schrödinger equation

$$-\frac{1}{2}\Psi_n'' + V_n\Psi_n = E\Psi_n \implies \Psi_n'' = 2(V_n - E)\Psi_n$$

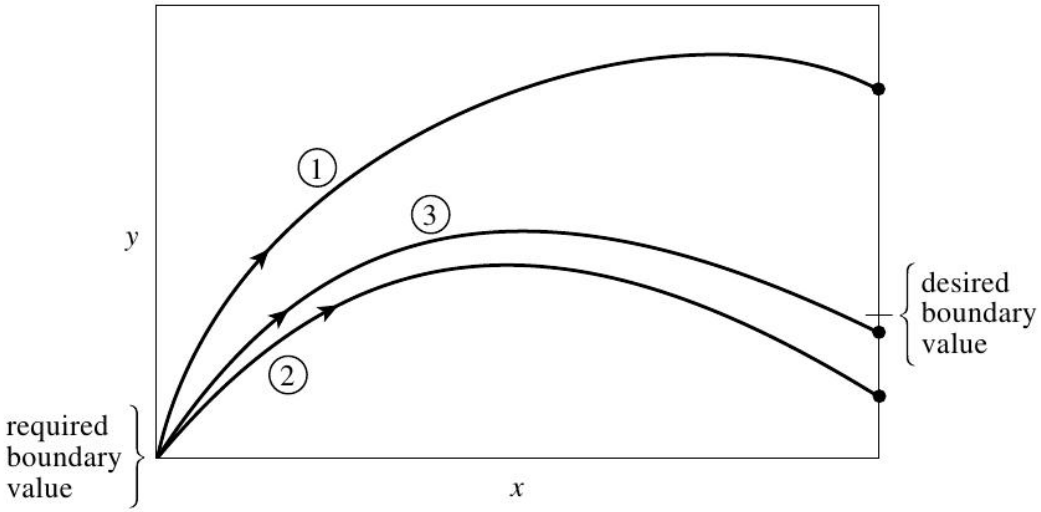
$$\Psi_n'' = \frac{\Psi_{n-1} - 2\Psi_n + \Psi_{n+1}}{h^2} \implies \Psi_{n+1} = 2\Psi_n - \Psi_{n-1} + 2(V_n - E)h^2$$

  - Once Ψ_0 and Ψ_1 are given, the rest can be computed
- Much better (order of 4 instead of 2 above) is Numerov's method

$$\left(\frac{\mathrm{d}^2}{\mathrm{d}x^2} + f(x)\right)y(x) = 0 \implies y_{n+1} = \frac{(2 - \frac{5h^2}{6}f_n)y_n - (1 + \frac{h^2}{12}f_{n-1})y_{n-1}}{1 + \frac{h^2}{12}f_{n+1}}$$

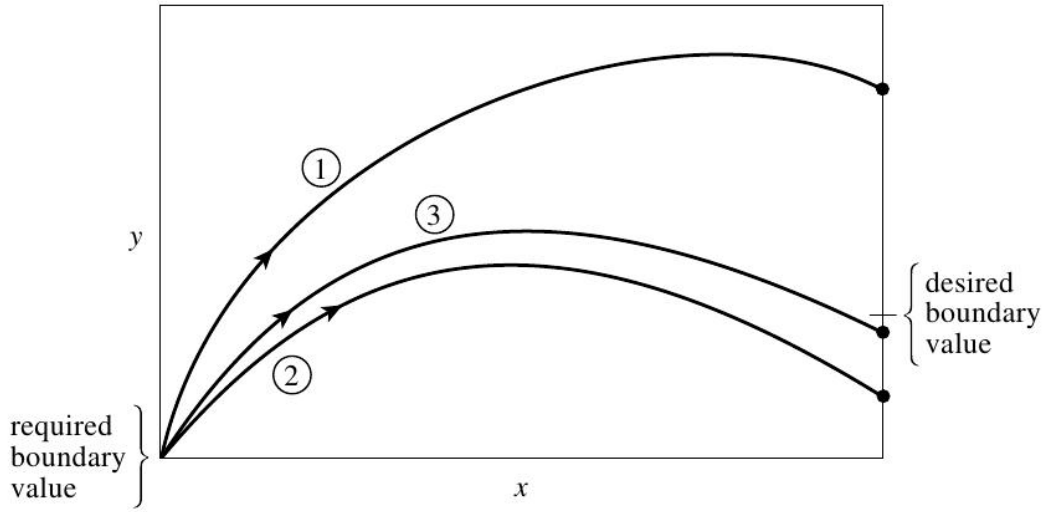
Stationary states: continuation

- How to find the right E: the shooting method



Stationary states: continuation

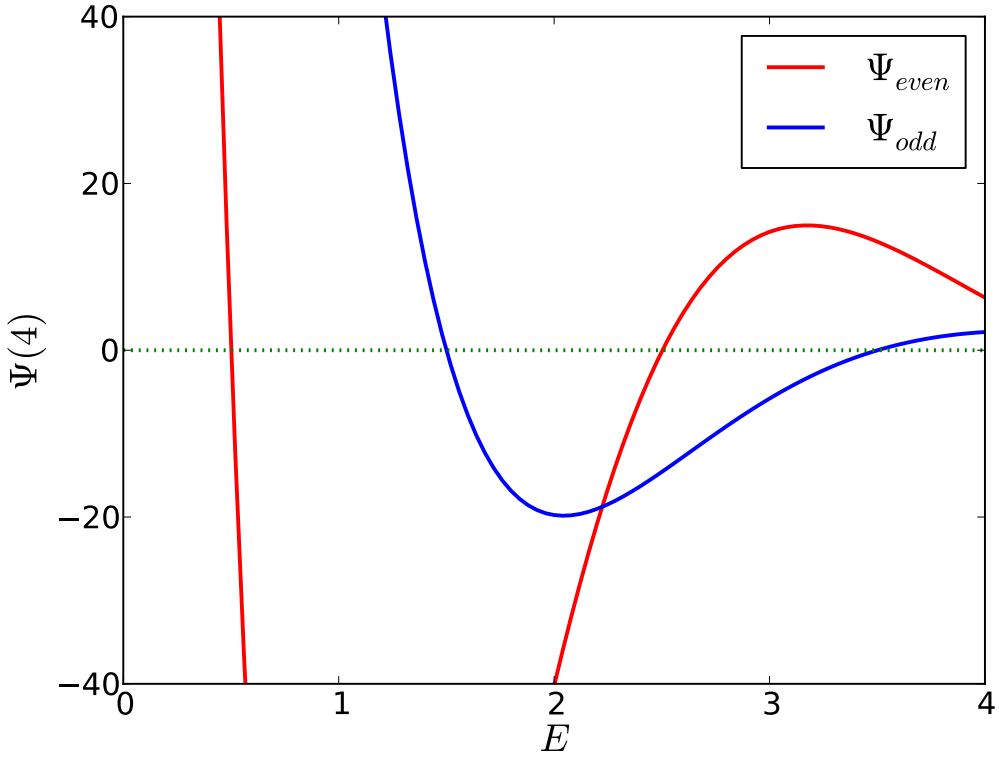
- How to find the right E: the shooting method



- It is best to write a function, say `psi_end(E)`, which takes E as an argument and returns $\Psi(x)$ at some (not too) distant point x
- Finding E efficiently: *e.g.*, `brentq(f,a,b)` from `scipy.optimize` finds a root of f between a and b

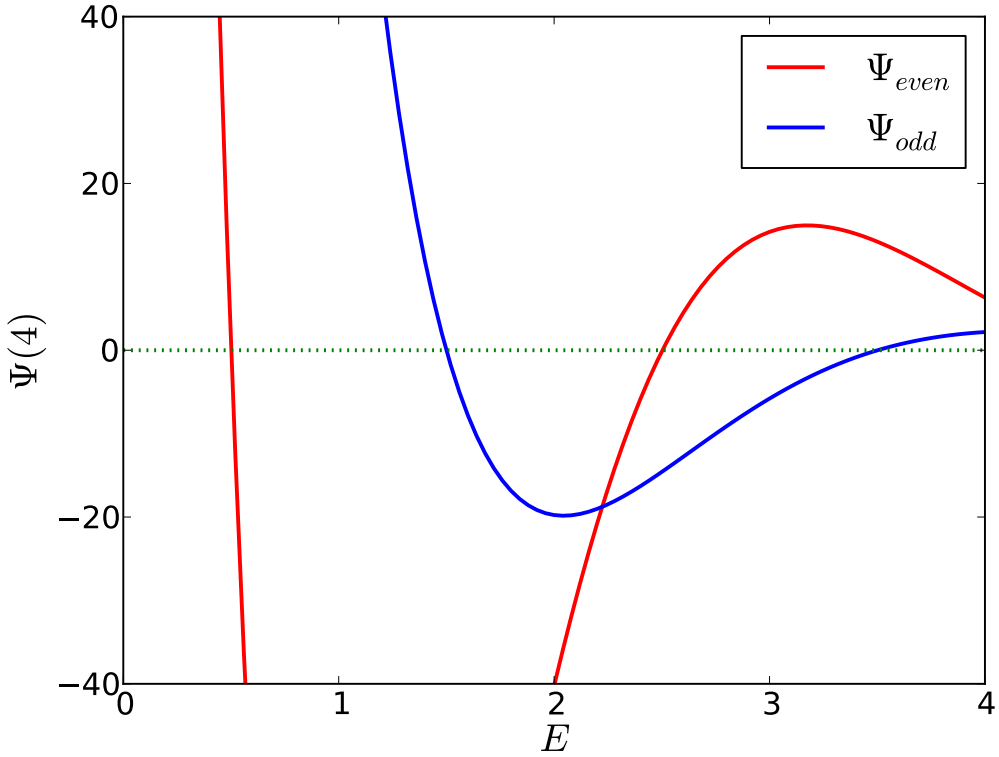
Stationary states: LHO

- We first plot $\Psi(x_m)$ for a range of energies



Stationary states: LHO

- We first plot $\Psi(x_m)$ for a range of energies



- Then individual roots can be found
- `ex4-1.py` gives 0.501 and 1.500 for the first two eigenstates

Try this at home

1: Reversibility and unitarity

Implement the Finite-Difference Time-Domain algorithm. Is it reversible? Is it unitary?

2: Ground state

Find the ground state energy of potential $V(x) = -\exp(-\sqrt{|x|})$ in the units where $\hbar = m = 1$. Can you find also the first excited state?