Implementing Density Functional Theory using Finite Difference Method

Fadjar Fathurrahman

1 Introduction

Kohn-Sham equations:

$$\hat{H}_{\mathrm{KS}}\,\psi_i(\mathbf{r}) = \epsilon_i\,\psi_i(\mathbf{r}) \tag{1}$$

2 Schroedinger equation in 1d

We are interested in finding bound states solution to 1d time-independent Schroedinger equation:

$$\left[-\frac{1}{2}\frac{\mathrm{d}^2}{\mathrm{d}x^2} + V(x)\right]\psi(x) = E\,\psi(x) \tag{2}$$

with the boundary conditions:

$$\lim_{x\rightarrow\pm\infty}\psi(x) = 0 \tag{3}$$

First we need to define a spatial domain $[x_{\mathrm{min}}, x_{\mathrm{max}}]$ where $x_{\mathrm{min}}, x_{\mathrm{max}}$ chosen such that the boundary condition 3 is approximately satisfied. The next step is to divide the spatial domain x using equally-spaced grid points which we will denote as $\{x_1, x_2, \ldots, x_N\}$ where N is number of grid points. Various spatial quantities such as wave function and potential will be discretized on these grid points. The grid points x_i, $i = 1, 2, \ldots$ are chosen as:

$$x_i = x_{\mathrm{min}} + (i-1)h \tag{4}$$

where h is the spacing between the grid points:

$$h = \frac{x_{\mathrm{max}} - x_{\mathrm{min}}}{N-1} \tag{5}$$

The following code can be used to initialize the grid points:

```julia
function init_FD1d_grid( x_min::Float64, x_max::Float64, N::Int64 )
    L = x_max - x_min
    h = L/(N-1) # spacing
    x = zeros(Float64,N) # the grid points
    for i = 1:N
        x[i] = x_min + (i-1)*h
    end
    return x, h
end
```

Approximating second derivative

Our next task is to find an approximation to the second derivative operator present in the Equation (2). One simple approximation that we can use is the 3-point (central) finite difference:

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\psi_i = \frac{\psi_{i+1} - 2\psi_i + \psi_{i-1}}{h^2} \tag{6}$$

where we have the following notation have been used: $\psi_i = \psi(x_i)$. By taking $\{\psi_i\}$ as a column vector, the second derivative operation can be expressed as matrix multiplication:

$$\vec{\psi}'' = \mathbb{D}^{(2)}\vec{\psi} \tag{7}$$

where $\mathbb{D}^{(2)}$ is the second derivative matrix operator:

$$\mathbb{D}^{(2)} = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -2 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -2 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -2 \end{bmatrix} \tag{8}$$

An example implementation can be found in the following function.

```
function build_D2_matrix_3pt( N::Int64, h::Float64 )
    mat = zeros(Float64,N,N)
    for i = 1:N-1
        mat[i,i] = -2.0
        mat[i,i+1] = 1.0
        mat[i+1,i] = mat[i,i+1]
    end
    mat[N,N] = -2.0
    return mat/h^2
end
```

Before use this function to solve Schroedinger equation we will to test the operation in Equation (8) for a simple function which second derivative can be calculated analytically.

$$\psi(x) = \mathrm{e}^{-\alpha x^2} \tag{9}$$

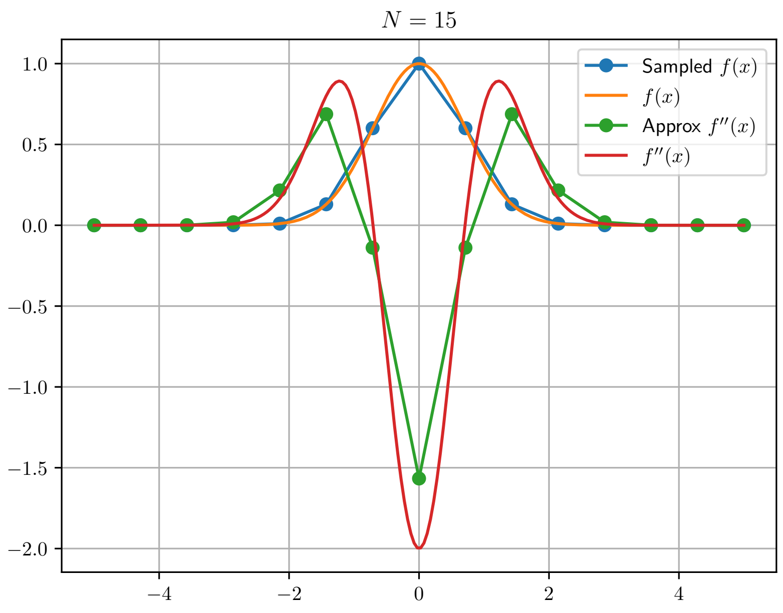
which second derivative can be calculated as

$$\psi''(x) = \left(-2\alpha + 4\alpha^2 x^2\right) \mathrm{e}^{-\alpha x^2} \tag{10}$$

They are implemented in the following code

```
function my_gaussian(x; α=1.0)
    return exp(-α*x^2)
end

function d2_my_gaussian(x; α=1.0)
    return (-2*α + 4*α^2 * x^2) * exp(-α*x^2)
end
```



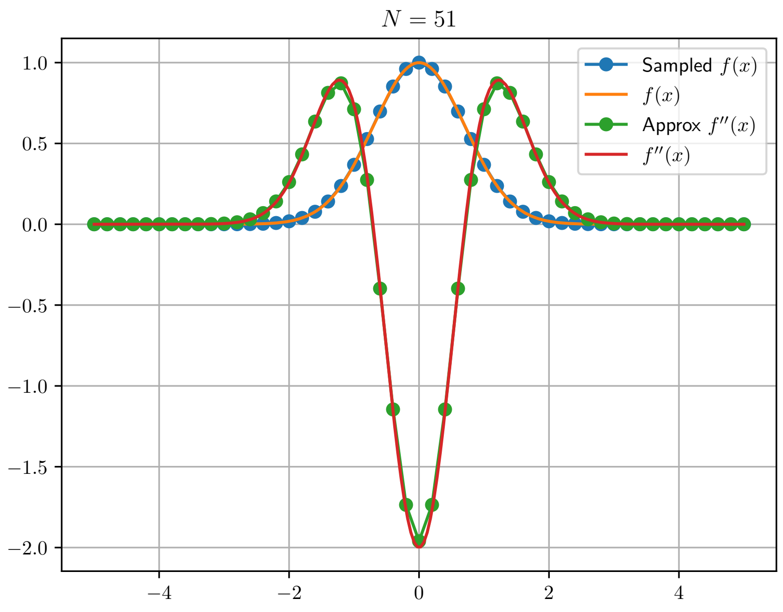


Figure 1: Finite difference approximation to a Gaussian function and its second derivative

Harmonic potential

We will start with a simple potential with known exact solution, namely the harmonic potential:

$$V(x) = \frac{1}{2}\omega^2 x^2 \tag{11}$$

The Hamiltonian in finite difference representation:

$$\mathbb{H} = -\frac{1}{2}\mathbb{D}^{(2)} + \mathbb{V} \tag{12}$$

where $\mathbb{V}$ is a diagonal matrix whose elements are:

$$\mathbb{V}_{ij} = V(x_i)\delta_{ij} \tag{13}$$

Code to solve harmonic oscillator:

```
using Printf
using LinearAlgebra
```

```julia
using LaTeXStrings

import PyPlot
const plt = PyPlot
plt.rc("text", usetex=true)

include("init_FD1d_grid.jl")
include("build_D2_matrix_3pt.jl")

function pot_harmonic( x; ω=1.0 )
    return 0.5 * ω^2 * x^2
end

function main()
    # Initialize the grid points
    xmin = -5.0
    xmax =  5.0
    N = 51
    x, h = init_FD1d_grid(xmin, xmax, N)
    # Build 2nd derivative matrix
    D2 = build_D2_matrix_3pt(N, h)
    # Potential
    Vpot = pot_harmonic.(x)
    # Hamiltonian
    Ham = -0.5*D2 + diagm( 0 => Vpot )
    # Solve the eigenproblem
    evals, evecs = eigen( Ham )
    # We will show the 5 lowest eigenvalues
    Nstates = 5
    @printf("Eigenvalues\n")
    for i in 1:Nstates
        @printf("%5d %18.10f\n", i, evals[i])
    end

    # normalize the first three eigenstates
    for i in 1:3
        ss = dot(evecs[:,i], evecs[:,i])*h
        evecs[:,i] = evecs[:,i]/sqrt(ss)
    end

    # Plot up to 3rd eigenstate
    plot_title = "N="*string(N)
    plt.plot(x, evecs[:,1], label="1st eigenstate", marker="o")
    plt.plot(x, evecs[:,2], label="2nd eigenstate", marker="o")
    plt.plot(x, evecs[:,3], label="3rd eigenstate", marker="o")
    plt.legend()
    plt.tight_layout()
    plt.savefig("IMG_main_harmonic_01_"*string(N)*".pdf")
end

main()
```

Compare with analytical solution.

Plot of eigenfunctions:

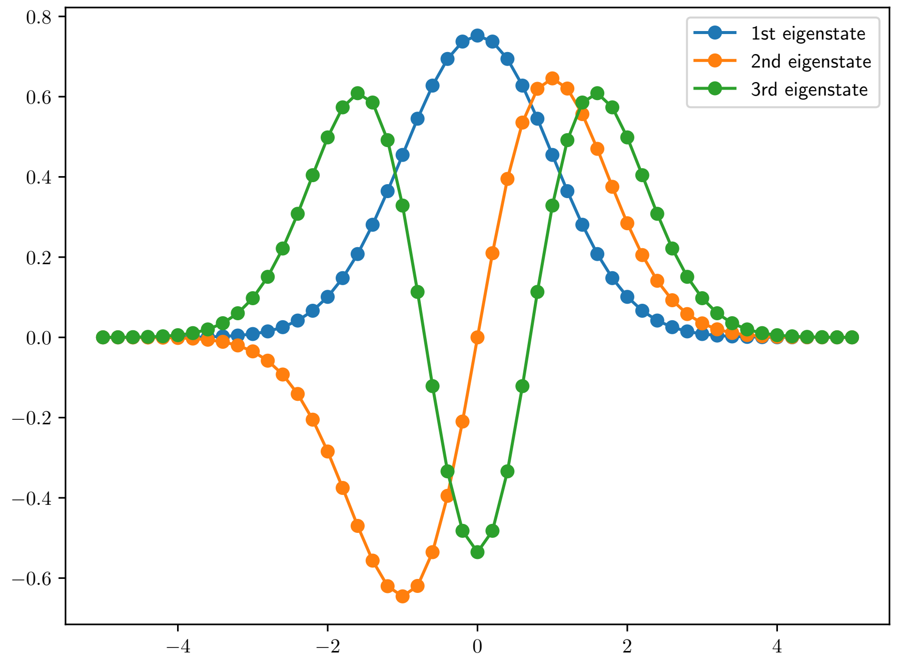


Figure 2: Eigenstates of harmonic oscillator

2.1 Higher order finite difference

An alternative to using more grid points To obtain higher accuracy
Implementing higher order finite difference.

A Alternative solutions

```
function my_func()
    println("OK ...")
end
```