Project 2 in FYS3150

Bendik Steinsvåg Dalen, Ulrik Seip

October 3, 2018

1 ABSTRACT

In this project we have implemented Jacobian algorithm, to find eigenvectors, and their corresponding eigenvalues in tridiagonal matrices. We then used this to model a harmonic oscillator problem in three dimensions, with one and two electrons. This turned out to be a computationally heavy, but relatively accurate method.

2 INTRODUCTION

Finding eigenvectors analytically is complicated, and can be tedious, and this is why it is much more convenient to do so numerically. A common way of doing this is by the application of the Jacobian method. Essentially we rotate one matrix element at a time, always taking the one with the highest absolute value, until all but the diagonal elements are essentially zero. All transformations are also applied to an identity matrix that then turns into our eigenvectors.

3 METHOD

3.a Implementing the Jacobian algorithm

The implementation follows a standard recipe: See the documentation in section 6.a for further explanation.

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}}.$$

We can then define the angle θ so that the non-diagonal matrix elements of the transformed matrix a_{kl} become non-zero and we obtain the quadratic equation (using $\cot 2\theta = 1/2(\cot\theta - \tan\theta)$

$$t^2 + 2\tau t - 1 = 0,$$

resulting in

$$t = -\tau \pm \sqrt{1+\tau^2},$$

and c and s are easily obtained via

$$c = \frac{1}{\sqrt{1+t^2}},$$

and $s = tc$.

$$1 = 2 \tag{1}$$

$$3 = \tag{2}$$

3.b Testing the code

For testing the algorithm we have implemented two tests. One for checking if the largest element in the matrix is correctly located, and one for testing if the resulting eigenvalues are correct. The first one is more useful for development purposes, whilst the second one is essential for validating that our implementation works correctly.

3.c Quantum dots in three dimensions, one electron

Now that we had a general algorithm we used it to model a electron that moves in a three-dimensional harmonic oscillator potential. In other words, we looked for the solution of the radial part of Schroedinger's equation for one electron, which reads

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr} - \frac{l(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r). \tag{3}$$

This problem also has analytical solutions, so we can test how accurate our algorithm is.

(Some math-stuff).

The Schroedinger's equation then becomes

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda y(\rho). \tag{4}$$

Since we are working in radial coordinates we have $\rho \in [0, \infty)$. Since we can't represent infinity on a computer we have to find an aproximation, which we will come back to later. For now we define $\rho_{min} = 0$ and ρ_{max} to represent the minimum and maximum values of ρ.

Function 4 is an differential equation that can be modeled similarly to (ting). If we have n mesh points we get a step length

$$h = \frac{\rho_{max} - \rho_{min}}{n}. \tag{5}$$

The value of ρ at a point i is then

$$\rho_i = \rho_0 + ih \qquad i = 1, 2, \ldots, N. \tag{6}$$

3.d Quantum dots in three dimensions, two electrons

4 RESULTS

5 CONCLUSIONS

6 APENDICES

6.a Integration loop from rotator.jl

```
function maxKnotL(a)
    max = 0
    kl = [1,1]
    n = Int64(length(a[1,:]))
    for k = 1:n
        for l = k+1:n
            ma = abs(a[k, l])
            if (ma > max)
                max = ma
                kl = [k, l]
            end
        end
    end
    return kl[1], kl[2]                #k, l
end

function rotate(a, tol)            #den faktiske rotasjonslkken. Tar inn en matrise a og nyaktighet.
    n = Int64(length(a[1,:]))      #initiates n for later use
    r = Matrix{Float64}(I, n, n)   #initialising eigenvector matrix
    counter = 0                    #teller antall "similarity transformaitons"
    k, l = maxKnotL(a)             #finds indices of matrix element with highest value
    while abs(a[k ,l]) > tol       #this is the actual loop
        counter += 1
        if (a[k, l] != 0.0)
            #kl = maxKnotL(a)
            tau = (a[l, l] - a[k, k])/2*a[k, l] #blir ikke dette alltid null?
            if (tau > 0)
                t = -tau + sqrt(1.0+tau^2)
            else
                t = -tau - sqrt(1.0+tau^2)
            end
            c = 1/sqrt(1.0+t^2)
            s = c*t
        #end
        else
            c = 1.0
            s = 0.0
        end
```

```
        a_kk = a[k, k]
        a_ll = a[l, l]
        #doing stuff with indices k and l
        c2 = c^2
        s2 = s^2
        csakl2 = 2.0*c*s*a[k, l]
        a[k, k] = c2*a_kk - csakl2 + s2*a_ll
        a[l, l] = s2*a_kk + csakl2 + c2*a_ll
        a[k, l] = 0
        a[l, k] = 0
        #doing stuff with the remaing matrix elements
        for i = 1:n
            if ((i != k) && (i != l))
                a_ik = a[i, k]
                a_il = a[i, l]
                a[i, k] = c*a_ik - s*a_il
                a[k, i] = a[i, k]
                a[i, l] = c*a_il + s*a_ik
                a[l, i] = a[i, l]
            end
            #calculating eigenvectors
            r_ik = r[i, k]
            r_il = r[i, l]
            r[i, k] = c*r_ik - s*r_il
            r[i, l] = c*r_il + s*r_ik
        end
```

7 REFERENCES

References

[1] Computational Physics, Lecture Notes Fall 2015, Morten Hjort-Jensen p.215-220