Exercise sheet 6

by Robin Heinemann (group 4), Paul Rosendahl (group 4)

February 1, 2019

1 Perturbed quantum mechanical oscillator

The Hamiltonian is given by

$$h = \frac{H}{\hbar\omega} = \left(\frac{1}{2}\Pi^2 + \frac{1}{2}Q^2 + \lambda Q^4\right)$$
$$(h)_{nm} = (h_0)_{nm} + \lambda\left(Q^4\right)_{nm}$$
$$(h_0)_{nm} = \left(n + \frac{1}{2}\right)\delta_{nm}$$

Q_{nm} is given by

$$Q_{nm} = \frac{1}{\sqrt{2}}\left(\sqrt{n+1}\delta_{n,m-1} + \sqrt{n}\delta_{n,m+1}\right)$$

For Q^2 we can find

$$
\begin{aligned}
(Q^2)_{ik} &= \frac{1}{2}(\sqrt{i+1}\delta_{i,j-1} + \sqrt{i}\delta_{i,j+1})(\sqrt{j+1}\delta_{j,k-1} + \sqrt{j}\delta_{j,k+1}) \\
&= \frac{1}{2}(\sqrt{i+1}\sqrt{j+1}\delta_{i,j-1}\delta_{j,k-1} + \sqrt{i+1}\sqrt{j}\delta_{i,j-1}\delta_{j,k+1} + \sqrt{i}\sqrt{j+1}\delta_{i,j+1}\delta_{j,k-1} + \sqrt{i}\sqrt{j}\delta_{i,j+1}\delta_{j,k+1}) \\
&= \frac{1}{2}(\sqrt{(i+1)(i+2)}\delta_{i,k-2} + (2i+1)\delta_{ik} + \sqrt{i(i-1)}\delta_{i,k+2}) \\
(Q^4)_{ik} &= \frac{1}{4}(\sqrt{(i+1)(i+2)}\delta_{i,j-2} + (2i+1)\delta_{ij} + \sqrt{i(i-1)}\delta_{i,j+2}) \\
&\quad (\sqrt{(j+1)(j+2)}\delta_{j,k-2} + (2j+1)\delta_{jk} + \sqrt{j(j-1)}\delta_{j,k+2}) \\
&= \frac{1}{4}(\sqrt{(j+1)(j+2)}\sqrt{(i+1)(i+2)}\delta_{i,j-2}\delta_{j,k-2} + (2j+1)\sqrt{(i+1)(i+2)}\delta_{i,j-2}\delta_{jk} \\
&\quad + \sqrt{j(j-1)}\sqrt{(i+1)(i+2)}\delta_{i,j-2}\delta_{j,k+2} + \sqrt{(j+1)(j+2)}(2i+1)\delta_{ij}\delta_{j,k-2} \qquad + (2 \\
&\quad + \sqrt{(j+1)(j+2)}\sqrt{i(i-1)}\delta_{i,j+1}\delta_{j,k-2} + (2j+1)\sqrt{i(i-1)}\delta_{i,j+2}\delta_{jk} \\
&\quad + \sqrt{j(j-1)}\sqrt{i(i-1)}\delta_{i,j+2}\delta_{j,k+2}) \\
&= \frac{1}{4}(\sqrt{(i+1)(i+2)(i+3)(i+4)}\delta_{i,k-4} + (2i+5)\sqrt{(i+1)(i+2)}\delta_{i,k-2} \\
&\quad + (i+1)(i+2)\delta_{i,k} + \sqrt{(i+1)(i+2)}(2i+1)\delta_{i,k-2} \\
&\quad + (2i+1)^2\delta_{i,k} + \sqrt{i(i-1)}(2i+1)\delta_{i,k+2} \\
&\quad + i(i-1)\delta_{i,j} + (2i+5)\sqrt{i(i-1)}\delta_{i,k+2} \\
&\quad + \sqrt{i(i-1)(i-2)(i-3)}\delta_{i,k+4})
\end{aligned}
$$

If the matrix is of size n there are boundary terms for the diagonals, because of the finite size, which results in

$$
\begin{aligned}
&= \frac{1}{4}(\sqrt{(i+1)(i+2)(i+3)(i+4)}\delta_{i,k-4} + (2i+5)\sqrt{(i+1)(i+2)}\delta_{i,k-2} \\
&\quad + (i+1)(i+2)\delta_{i,k} + \sqrt{(i+1)(i+2)}(2i+1)\delta_{i,k-2} \\
&\quad + (2i+1)^2\delta_{i,k} + \sqrt{i(i-1)}(2i+1)\delta_{i,k+2} \\
&\quad + i(i-1)\delta_{i,j} + (2i+5)\sqrt{i(i-1)}\delta_{i,k+2} \\
&\quad + \sqrt{i(i-1)(i-2)(i-3)}\delta_{i,k+4} \\
&\quad (i+1)(i+2)(\delta_{i,0}\delta_{k,0} + \delta_{i,1},\delta_{k,1}) - i(i-1)(\delta_{i,n-1}\delta_{j,n-1} + \delta_{i,n}\delta_{j,n}))
\end{aligned}
$$

```
#include <tred2.c>
#include <tqli.c>
#include <nrutil.c>
#include <stdio.h>
#include <math.h>

```

```
double pythag(double a, double b) {
    return sqrt(a * a + b * b);
}

void matmul(double ** a, double ** b, double ** c, int n) {
    for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        c[i][j] = 0.0;
        for(int k = 1; k <= n; k++) {
        c[i][j] += a[i][k] * b[k][j];
        }
    }
    }
}

void p(double ** m, int n) {
    for(int i = 1; i <= n; i++) {
    for(int j = 1; j <= n; j++) {
        printf("%2.5lf  ", m[i][j]);
    }
    printf("\n");
    }
}

void perturbed_oscillator(int n, double l) {
    double ** m = dmatrix(1, n + 4, 1, n + 4);
    double ** m2 = dmatrix(1, n + 4, 1, n + 4);
    double ** m3 = dmatrix(1, n + 4, 1, n + 4);

    double sqrt2 = sqrt(2.0);

    for(int i = 1; i < n; i++) {
    m[i + 1][i] = sqrt(i) / sqrt2;
    m[i][i + 1] = sqrt(i) / sqrt2;
    }

    matmul(m, m, m2, n);
    matmul(m2, m2, m, n);

    for(int i = 1; i <= n; i++) {
    m[i][i] += (i - 1 + .5) / l;

    for(int j = 1; j <= n; j++) {
        m[i][j] *= l;

        m2[i][j] = m[i][j];
    }
    }

    double * d = malloc(sizeof(double) * (n + 1));
```

```
57      double * e = malloc(sizeof(double) * (n + 1));
58      d = d - 1;
59      e = e - 1;
60
61      tred2(m, n, d, e);
62      tqli(d, e, n, m);
63
64      matmul(m2, m, m3, n);
65
66      double * eps = malloc(sizeof(double) * n);
67
68      int * idx = malloc(sizeof(int) * n);
69
70      for(int i = 1; i <= n; i++) {
71      idx[i - 1] = i;
72      }
73
74      for (int i = 1; i < n; i++) {
75      int key = idx[i];
76      int j = i - 1;
77
78      while(j >= 0 && d[idx[j]] > d[key]) {
79          idx[j + 1] = idx[j];
80          j = j - 1;
81      }
82      idx[j + 1] = key;
83      }
84
85      printf("|-\n");
86      printf("| n = %d\n", n);
87      for(int i = 1; i <= 10; i++) {
88      int ix = idx[i - 1];
89      double eps = 0.0;
90
91      for(int j = 1; j <= n; j++) {
92          eps += fabs(m[j][ix] - m3[j][ix] / d[ix]);
93      }
94
95      printf("| %.15lf | %.15lf \n", d[ix], eps);
96      }
97  }
98
99  int main() {
100     printf("| $\\lambda_i$ | $\\eps$ $\n");
101     for(int i = 15; i <= 30; i += 5) {
102     perturbed_oscillator(i, .1);
103     }
104 }
```

Listing 1: rust implementation of gauss elimination for a tridiagonal matrix

Table 1: the lowest ten eigenvalues λ_i and the 1-norm of $hv - \lambda v$ where v is the eigenvector with eigenvalue λ

λ_i	ε \$
n = 15	
0.559146331717481	0.000000000000056
1.769503154752390	0.000000000000001
3.138630351927867	0.000000000000002
4.628832924839890	0.000000000000002
6.219589268571163	0.000000000000004
7.900514741235316	0.000000000000003
9.684402683933815	0.000000000000003
11.394960405389993	0.000000000000003
13.144176458068515	0.000000000000002
16.606417302164019	0.000000000000004
n = 20	
0.559146327396061	0.000000000000017
1.769502650598558	0.000000000000023
3.138624310963206	0.000000000000003
4.628881549200992	0.000000000000009
6.220291663918584	0.000000000000004
7.899835172646707	0.000000000000007
9.658345956233044	0.000000000000002
11.484717894488245	0.000000000000006
13.364075642272910	0.000000000000005
15.422401778971913	0.000000000000003
n = 25	
0.559146327187369	0.000000000000233
1.769502644004068	0.000000000000003
3.138624307573727	0.000000000000028
4.628882782968839	0.000000000000006
6.220300826989985	0.000000000000010
7.899769051837731	0.000000000000004
9.657851696340218	0.000000000000009
11.487225950553698	0.000000000000006
13.381940814696728	0.000000000000008
15.340927001633565	0.000000000000006
n = 30	
0.559146327183577	0.000000000000052
1.769502643949268	0.000000000000066
3.138624308466358	0.000000000000001
4.628882808395193	0.000000000000038
6.220300899870113	0.000000000000004
7.899767283848700	0.000000000000025
9.657840218111822	0.000000000000005
11.487312780939364	0.000000000000024
13.382455649955588	0.000000000000004
15.338741261872123	0.000000000000011

Comparing the eigenvalues for different n it is clear that the lowest one is by far the most accurate. For $n = 20$

λ_0 is accurate to about nine digits. Also ε is really small for all of the values, which shows that these values are actually eigenvalues.