MAT-INF3360
Mandatory Exercises 1

Jonas van den Brink

j.v.d.brink@fys.uio.no

February 7, 2014

Exercise 2.30

In this exercise we will show the orthogonality relation between the discrete eigenfunctions of the operator L_h, which is defined as the negative of a 2. order central difference, meaning we have:

$$L_h v(x_j) = \frac{-v(x_{j+1}) + 2v(x_j) - v(x_{j-1})}{h^2}.$$

So we are studying the functions $v \in D_{h,0}$ that satisfy the eigenvalue problem

$$L_h v = \mu v.$$

From the text we know that the eigenstates are:

$$v_k(x_j) = \sin(k\pi x_j) \text{ for } k = 1, 2, \ldots, n.$$

And we will show the orthogonality

$$\langle v_n, v_m \rangle_h = \delta_{n,m}/2,$$

where $\delta_{n,m}$ denotes the Kronecker-delta.

a)

We will show that the sum

$$S_k = \sum_{j=0}^{n} \cos(k\pi x_j),$$

is $S_k = 0$ for k even, and $S_k = 1$ if k is odd. For $k = 0$ the sum is trivial and obviously equal to $n + 1$.

We start by writing the cosine term into a linear combination of expoentials with a purely imaginary argument and splitting the sum into parts

$$\sum_{j=0}^{n} \cos(k\pi x_j) = \frac{1}{2}\bigg(\sum_{j=0}^{n} \exp(ik\pi x_j) + \sum_{j=0}^{n} \exp(-ik\pi x_j)\bigg).$$

by introducing the step size h through $x_j = h \cdot j$, we can rewrite these sums as finite geometric sums, to which the value is known[1]. We have

$$\sum_{j=0}^{n} \exp(\pm ik\pi x_j) = \sum_{j=0}^{n} [\exp(\pm ik\pi h)]^j = \frac{1 - [\exp(\pm ik\pi h)]^{n+1}}{1 - \exp(\pm ik\pi h)}.$$

So our sum can be written

$$S_k = \frac{1}{2}\left(\frac{1 - [\exp(ik\pi h)]^{n+1}}{1 - \exp(ik\pi h)} + \frac{1 - [\exp(-ik\pi h)]^{n+1}}{1 - \exp(-ik\pi h)}\right).$$

Now we use the fact that

$$[\exp(\pm ik\pi h)]^{n+1} = \exp(\pm ik\pi h(n+1)) = \exp(\pm ik\pi) = (-1)^k.$$

Due to the fact that $h(n+1) = 1$.

We now expand the latter fraction by $\exp(ik\pi h)$ and add the two fractions together to get

$$S_k = \frac{1}{2}\frac{1 - \exp(ik\pi h) - (-1)^k(1 - \exp(ik\pi h))}{1 - \exp(ik\pi h)} = \frac{1 - (-1)^k}{2}.$$

And from this expression we readily see that

$$S_k = \begin{cases} 0 & \text{for } k \text{ even} \\ 1 & \text{for } k \text{ odd.} \end{cases}$$

b)

We now turn to the inner-product, and will first show the orthogonality property of the discrete eigenfunctions v_k. Remebering that the inner product in the space $D_{h,0}$ is defined

$$\langle u, v \rangle_h = h \sum_{j=1}^{n} u_j v_j,$$

we can write the inner product as

$$\langle v_k, v_m \rangle_h = h \sum_{j=0}^{n} \sin(k\pi x_j)\sin(m\pi x_j).$$

Note that we let j run from 0 and not 1, this is ok as $v_k(x_0) = 0$ for all k anyway, so it doesn't affect the sum.

Now, using the trigonometric identity

$$\sin(x)\sin(y) = \frac{1}{2}\big(\cos(x-y) - \cos(x+y)\big),$$

we can rewrite this sum as

$$\langle v_k, v_m \rangle_h = \frac{h}{2}\left(\sum_{j=0}^{n} \cos((k-m)\pi x_j) - \sum_{j=0}^{n} \cos((k+m)\pi x_j)\right),$$

[1] See for example *Rottman* page 112.

which using the results of exercise a) can be written as

$$\langle v_k, v_m \rangle_h = \frac{h}{2}\Big(S_{(k-m)} - S_{(k+m)} \Big).$$

We now see that for any integer values of k and m, both the difference $k-m$ and the sum $k+m$ will either be odd or even, and so the two sums cancel out, and we have

$$\langle v_k, v_m \rangle_h = 0 \text{ when } k \neq m.$$

When $k = m$, the first sum becomes $S_0 = n+1$, and so this result does *not* apply to that case.

c)

We now look at the case

$$\langle v_k, v_k \rangle_h.$$

Using the same steps as in the previous exercise we can show that this is equal to

$$\langle v_k, v_k \rangle_h = \frac{h}{2}\Big(S_0 - S_{2k} \Big).$$

From exercise a), we know that $S_{2k} = 0$ as $2k$ is even for all non-zero integers k, and we also know that $S_0 = n+1 = 1/h$. Using this we have

$$\langle v_k, v_k \rangle_h = \frac{1}{2}.$$

And we have thus shown that

$$\langle v_k, v_m \rangle_h = \delta_{k,m}/2 \text{ q.e.d.}$$

Project 2.1

We will be studying the problem

$$-u''(x) = f(x), \qquad x \in (0,1), \qquad u(0) = u(1) = 0.$$

Which has solutions on the form

$$u(x) = x\int_0^1 (1-y)f(y)\,\mathrm{d}y - \int_0^x (x-y)f(y)\,\mathrm{d}y.$$

We will be looking at numerical schemes that result from using numerical approximations to the integrations in this form.

(a) The Trapezoidal Rule

The trapezoidal rule is an intutive approximation to a definite integral. It is most easily understood from a geometric perspective. The definite integral

$$I = \int_a^b F(x)\,\mathrm{d}x,$$

is the area under the curve $F(x)$ between the limits a and b. The trapezoidal rule approximates this area as many small trapezoids, from which the area is easily calculated. See figure 1 and 2 for illustrations.

Using more trapezoids should give a better approximation, but will also require more samplings. Using $n+1$ trapezoids, means using $n+2$ mesh points and thus $n+2$ samplings of the function $F(x)$.

Looking at figure 2, it is easy to see that the area of a single trapezoid is

$$a_i = (x_{i+1} - x_i)\frac{F(x_{i+1}) + F(x_i)}{2}.$$

Here, the width of the trapezoid is given as $x_{i+1} - x_i$, which is the general case. In this project however, we will assume uniformely spaced mesh points, meaning we use

$$x_i = a + ih, \qquad h = x_{i+1} - x_i = \frac{b-a}{n+1}.$$

Summing over all the trapezoids, we see that all the internal mesh points are added twice, while the endpoints are added just once, we thus have

$$I = \int_a^b F(x)\,\mathrm{d}x \approx h\biggl(\frac{F(a)+F(b)}{2} + \sum_{i=1}^{n} F(x_i)\biggr).$$

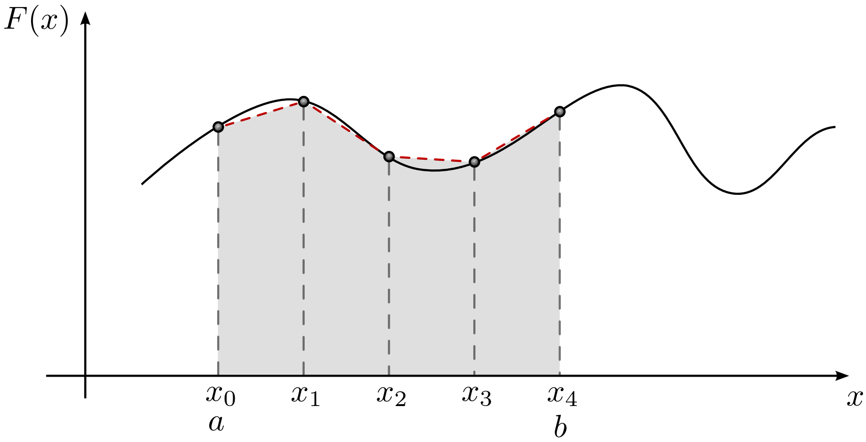

Figure 1: Illustration of using the trapezoidal rule to approximate an integral using 4 trapezoids, meaning we have $n = 3$ internal mesh points

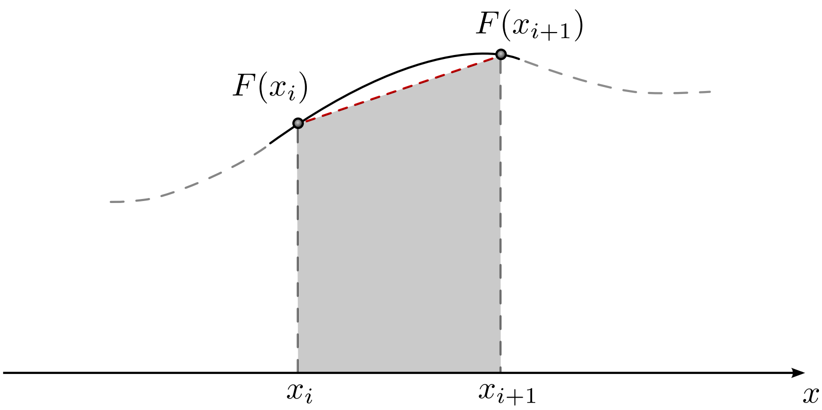

Figure 2: A single trapezoid element.

(b) Writing a Trapezoidal Integration Function

We will now implement a procedure that for a given a, b, $F(x)$ and n computes the approximation given by the trapezoidal rule.

For simplicity, we write the code in Python:

```
def trapez(F,a,b,n):
    h = (b-a)/(n+1)

    s = F(a)/2. + F(b)/2.
    for i in range(n):
        x += h
        s += F(x)

    return s*h
```

Note that this function is not very efficient. For practical use, it should either be vectorized using numpy or implemented in some other language, like for example C++.

(c)

We now set $F(x) = x^5$ and $G(x) = \sqrt{|x - \frac{1}{2}|}$ and will compute the integrals

$$I_1 = \int_0^1 F(x)\,\mathrm{d}x \qquad \text{and} \qquad I_2 = \int_0^1 G(x)\,\mathrm{d}x,$$

using the trapezoidal rule. We first compute them analytically, so that we have an exact solution to compare our results to.

The first integral is trivial

$$I_1 = \int_0^1 x^5\,\mathrm{d}x = \frac{1}{6}.$$

The second integral can be solved by realizing it is symetric about $x = 1/2$, and substituting $u = x - 1/2$:

$$I_2 = \int_0^1 \sqrt{|x - \frac{1}{2}|}\,\mathrm{d}x = 2\int_0^{\frac{1}{2}} \sqrt{u}\,\mathrm{d}u = \frac{\sqrt{2}}{3}.$$

Using the function `trapez(F,a,b,n)` defined above, we calculate numerical approximations to I_1 and I_2 using $n = 10, 20, 40, 80, 160$ internal mesh points.

To estimate the rate of convergence for the approximations of these integrals, we assume that the error is proportional to some order of the step length h:

$$e_h = ch^r,$$

where c is a proportionality constant. To find r, we compare for two different values of h

$$r = \frac{\ln(e_{h_1}/e_{h_2})}{\ln(h_1/h_2)}.$$

The results are shown in table 1.

n	Rel. error I_1	Conv. rate I_2	Rel. error I_2	Conv. rate I_2
10	$2.06 \cdot 10^{-2}$		$9.15 \cdot 10^{-3}$	
20	$5.67 \cdot 10^{-3}$	1.9981	$3.25 \cdot 10^{-3}$	1.5995
40	$1.49 \cdot 10^{-3}$	1.9995	$1.13 \cdot 10^{-3}$	1.5760
80	$3.81 \cdot 10^{-4}$	1.9999	$3.92 \cdot 10^{-4}$	1.5567
160	$9.64 \cdot 10^{-5}$	2.0000	$1.36 \cdot 10^{-4}$	1.5417

Table 1: Results of using the `trapez` function to evalute the integrals of $F(x)$ and $G(x)$ on the unit interval.

(d)

We now define the functions

$$\alpha(x) = \int_0^x f(y)\,\mathrm{d}y \qquad \text{and} \qquad \beta(x) = \int_0^x y f(y)\,\mathrm{d}y.$$

And will use these to redwrite the general solution to our boundary value problem

$$u(x) = x\int_0^1 (1-y)f(y)\,\mathrm{d}y - \int_0^x (x-y)f(y)\,\mathrm{d}y.$$

Divding the integrals makes this an easy task

$$u(x) = x\Bigg(\underbrace{\int_0^1 f(y)\,\mathrm{d}y}_{\alpha(1)} - \underbrace{\int_0^1 yf(y)\,\mathrm{d}y}_{\beta(1)}\Bigg) + \underbrace{\int_0^x yf(y)\,\mathrm{d}y}_{\beta(x)} - x\underbrace{\int_0^x f(y)\,\mathrm{d}y}_{\alpha(x)}.$$

So we see that we can write the solution as

$$u(x) = x\big(\alpha(1) - \beta(1)\big) + \beta(x) - x\alpha(x) \quad \text{q.e.d.}$$

(e)

We will now find approximations to $\alpha(x_i)$ and $\beta(x_i)$. Remember that

$$\alpha(x_i) = \int_0^{x_i} f(y)\,\mathrm{d}y, \quad \beta(x_i) = \int_0^{x_i} yf(y)\,\mathrm{d}y \quad \text{where } x_i = ih.$$

We will also be using the shorthand $f(x_i) = f_i$. We start by realizing that if the value of $\alpha(x_i)$ is known, calculating the value of $\alpha(x_{i+1})$ can be simplified in the following manner:

$$\alpha(x_{i+1}) = \int_0^{x_{i+1}} f(y)\,\mathrm{d}y = \int_0^{x_i} f(y)\,\mathrm{d}y + \int_{x_i}^{x_{i+1}} f(y)\,\mathrm{d}y = \alpha(x_i) + \int_{x_i}^{x_{i+1}} f(y)\,\mathrm{d}y.$$

The integral from x_i to x_{i+1} can then be approximated with the trapezoidal rule, using one internal mesh point—as this mesh point will lie between x_i and x_{i+1} it is a kind of *virtual* mesh point. We then have

$$\int_{x_i}^{x_{i+1}} f(y)\,\mathrm{d}y \approx \frac{h}{4}\big(f_i + 2f_{i+1/2} + f_{i+1}\big).$$

For simplicity, we let α_i denote the approximation to $\alpha(x_i)$ and *not* the exact. We then have

$$\alpha_0 = 0, \qquad \alpha_{i+1} = \alpha_i + \frac{h}{4}\big(f_i + 2f_{i+1/2} + f_{i+1}\big).$$

For $\beta(x_i)$, we do the exact same steps, and find

$$\beta_0 = 0, \qquad \beta_{i+1} = \beta_i + \frac{h}{4}\big(x_i f_i + 2x_{i+1/2}f_{i+1/2} + x_{i+1}f_{i+1}\big).$$

(f)

As $\alpha(x_{n+1}) = \alpha(1) \approx \alpha_{n+1}$, we can define an approximation to the solution of the boundary value problem u as:

$$u_i = x_i(\alpha_{n+1} - \beta_{n+1}) + \beta_i - x_i\alpha_i.$$

Note that as previously, u_i is the discrete approximation in the point x_i, and not the exact solution evaluated in x_i. From our Dirichlet boundary conditions we also know that $u_0 = u_{n+1} = 0$.

We will now implement this approximation on a computer. There is however one problem, in some cases, we might not be able to evaluate the source-term f in the virtual point $x_{i+1/2}$, which is not a mesh point. In some cases, we will therefore have to approximate this value, using a mean

$$f_{i+1/2} \approx (f_i + f_{i+1})/2.$$

This is effectively the same as using the trapezoidal rule with no internal mesh points for the integral

$$\int_{x_i}^{x_{i+1}} f(y)\,\mathrm{d}y.$$

g)

$$\beta_{i+1} = \beta_i + \frac{h}{2}\big(x_i f_i + 2x_{i+1/2} f_{i+1/2} + x_{i+1} f_{i+1}\big)$$

$$\beta_i - x_i \alpha$$

```
from pylab import *


a[n+1] = a[i] + h/2*(f[i] + 2*f[i+0.5] + f[i+1])
b[n+1] = b[i] + h/2()

def boundary_value_problem_solver(f, n):
    u = zeros(n+2)
    a = zeros(n+2)
    b = zeros(n+2)
    x = linspace(0,1,n+2)
    h = x[1]-x[0]

    if callable(f):
        for i in range(n+1):
            a[i+1] = a[i] + h/2*(f(x[i] + 2*f(x[i]+h/2.) +
                f(x[i+1])
```

We now want to test our solver, and will do so for the following source terms

$$\begin{aligned}
f(x) &= 1,\\
f(x) &= x,\\
f(x) &= x^2,\\
f(x) &= e^x,\\
f(x) &= \cos(ax) \text{ for } a \in \mathbb{R}.
\end{aligned}$$

All of these are solvable, with known exact solutions, giving us a perfect chance to compare our numerical solution.