STÆ401G Stærðfræðigreining IV
Heimaverkefni

Andri Freyr Viðarsson, Katrín Agla Tómasdóttir,
Kristófer Sigurðsson, Valgeir Einarsson

Valentina Giangreco M Puletti

25. apríl 2020

1 Mismunaaðferð og varmaleiðnijafna

Í þessum hluta á að finna nálgunarlausn fyrir varmaleiðnijöfnuna með almennu jaðar- og upphafsskilyrðunum í einni rúmvídd með því að nota mismunaaðferð á rétthyrningi.
Verkefnið má setja fram með

$$\begin{cases} \delta_t u = \kappa \delta_x^2 u, & 0 < x < L, t > 0, \\ u(0,t) = \psi(t), u(L,t) = \phi(t), & t > 0, \\ u(x,0) = f(x), & 0 \le x \le L, \end{cases} \tag{1.1}$$

þar sem κ er jákvæður fasti og f, ψ, ϕ eru föll.

Í mismunaaðferð er búið til net af $P = (M+1)(N+1)$ hnútpunktum.
Fyrir x-ás er eftirfarandi skipting notuð

$$0 = x_0 < x_1 < ... < x_N = L,$$

þannig að $x_i = ih$, $i = 0, ..., N$ og $h = \frac{h}{L}$.
Fyrir t-ás er eftirfarandi skipting notuð

$$0 = t_0 < t_1 < ... < t_M = T,$$

þannig að $t_\alpha = \alpha\tau$, $\alpha = 0, ..., M$ og $\tau = \frac{T}{M}$.
Gildi fallsins u í hnútpunktum er táknað með jaðarskilyrðunum

$$u_{i,\alpha} \coloneqq u(x_i, t_\alpha) = u(ih, \alpha\tau),$$

og upphafskilyrðin eru táknuð með

$$\begin{aligned} u_{0,\alpha} &\coloneqq u(0, \alpha\tau) = \psi(\alpha\tau) \coloneqq \psi, \\ u_{N,\alpha} &\coloneqq u(Nh, \alpha\tau) = \phi(\alpha\tau) \coloneqq \phi_\alpha, \\ u_{i,0} &\coloneqq u(ih, 0) = f(ih) \coloneqq f_i, \end{aligned}$$

hvert um sig.
Hlutafleiður m.t.t. t eru nálgaðar með „forward diffrence” mismunakvótum og hlutafleiður m.t.t. x eru nálgaðar með „central diffrence” mismunakvótum. Nálgunargildi á $u_{i,\alpha}$, $u_{0,\alpha}$, $u_{N,\alpha}$ og $u_{i,0}$ er hægt að tákna með $c_{i,\alpha}$, $c_{0,\alpha}$, $c_{N,\alpha}$ og $c_{i,0}$, hvert um sig.
Nálgunarverkefnið má því setja fram með

$$\begin{cases} c_{i,\alpha+1} =_{i+1,\alpha} +(1-2\sigma)c_{i,\alpha} +_{i-1,\alpha}, & 0 \le \alpha \le M-1, 1 \le i \le N-1, \\ c_{0,\alpha} = \psi_\alpha, c_{N,\alpha} = \phi_\alpha, & \alpha = 1, ..., M, \\ c_{i,0} = f_i, & 0 \le i \le N, \end{cases} \tag{1.2}$$

þar sem $\sigma \coloneqq \frac{\tau\kappa}{h^2}$

Uppsetning jöfnuhneppisins með $P \times P$ fylkinu A og $P \times 1$ vigrinum b er þannig að farið er eftir hverri línu fyrir sig í A og b, og viðeigandi dálkar í A er fylltir út frá 1.2 með því að nota eftirfarandi vörpun

$$\sigma : (i, \alpha) \longrightarrow d = \sigma(i, \alpha) = i + \alpha(N + 1),$$

sem varpar punkti (x_i, t_α) á tvívíða netinu í tölu. Hægt er að skipta uppsetningunni í tvo hluta:

1. Innri punktar
   Nálgunarjafna 1.2 fyrir varmaleiðnijöfnuna er notuð.

2. Gefin gildi í einstaka hnútpunktum
   Þetta á við um jaðarskilyrði og upphafskilyrði. Ef einstök gildi eru þekkt í punktum u_d þá gildir að einu stökin í A í línu d sem eru ekki núll eru $a_{d,d} = 1$ og hægri hlið jöfnuhneppisins er þannig að $b_d = u_d$.

Forrit sem leysir ofangreint verkefni má sjá hér að neðan.

I.

```python
# function to represent point on grid with one index u
def d1_rep(j, k, N):
    i = j + (k)*(N+1)
    return i

def heatwave(L, T, N, M, sigma, f, phi, psi):
    to_2d = {} # for grid presantation of values
    for alpha in range(M+1):
        for i in range(N+1):
            index = d1_rep(i, alpha, N)
            to_2d[index] = (alpha, i)

    x_grid = np.linspace(0, L, N+1)
    t_grid = np.linspace(0, T, M+1)
    h = L/N
    tau = T/M
    # construct matrix
    P = (M+1)*(N+1)
    A = np.zeros((P, P))
    b = np.zeros((P,1))
    row_index = 0 # used to insert equations into matrix

    # eq(16)
    # inner points
    for alpha in range(M):
        for i in range(1, N):
```

```
        A[row_index, d1_rep(i, alpha, N)] = (1-2*sigma)
        A[row_index, d1_rep(i, alpha+1, N)]= -1
        A[row_index, d1_rep(i+1, alpha, N)]= sigma
        A[row_index, d1_rep(i-1, alpha, N)] = sigma
        b[row_index] = 0
        row_index +=1 #next equation will appear in the following row of the matrix

# boundary conditions
for alpha in range (1, M+1):
    index_1 = d1_rep(0, alpha, N)
    A[row_index, index_1] = 1
    b[row_index] =psi
    row_index+=1
    index_2 = d1_rep(N, alpha, N)
    A[row_index, index_2] = 1
    b[row_index] = phi
    row_index+=1

# inital condition
for i in range(0, N+1):
    x = x_grid[i]
    index = d1_rep(i, 0, N)
    A[row_index, index] = 1
    b[row_index] = f(x)
    row_index+=1

# create and calculate with sparse matrix
S = csc_matrix(A)
c_out = spsolve(S, b)
c_mat = np.zeros((M+1, N+1)) # (P x 1) vector

# here the grid values are unflattened
for i, c in enumerate(c_out):
    index = to_2d[i]
    c_mat[index[0], index[1]] = c
hw = c_mat
return hw
```

II.

Prófunarkeyrsla. Sett er $\sigma = 1/4$, $L = 1$, $T = 1/100$, $N = 4$, $M = 5$ og

$$
\begin{aligned}
f(x) &= 100, & 0 \le x \le 1,\\
\psi(t) = \phi(t) &= 0, & 0 < t \le 1/100.
\end{aligned}
$$

```
L, T, N, M , sigma, phi, psi = 1, 1/100, 4, 5, 1/4, 0, 0
f2 = lambda x : 100
hw2 = heatwave(L, T, N, M, sigma, f2, phi, psi)
print(hw2)
```

```
[[100.      100.      100.      100.      100.     ]
 [  0.      100.      100.      100.        0.     ]
 [  0.       75.      100.       75.        0.     ]
 [  0.       62.5      87.5      62.5       0.     ]
 [  0.       53.125    75.       53.125     0.     ]
 [  0.       45.3125   64.0625   45.3125    0.     ]]
```

Mynd 1: Útkoma úr keyrslu

III.

Forrit er prófað á móti beinni lausn. Sett er $\sigma = 1/4$, $L = 1$, $T = 1/100$, $N = 10$, $M = 100$ og

$$
\begin{aligned}
f(x) &= x(L - x), & 0 \le x \le L,\\
\psi(t) = \phi(t) &= 0, & 0 < t \le T.
\end{aligned}
$$

Nálgunarlausn er borin saman við beina lausn sem er gefin með

$$u_a(x,t) = \frac{8L^2}{\pi^3} \sum_{n=1}^{\infty} \frac{1}{(2n+1)^3} e^{-\kappa(2n+1)^2\omega^2 t} \sin(2n+1)$$

og sett í töflu.

t \ x	$x = .2$	$x = .4$	$x = .6$	$x = .8$	$x = 1$
$t = \tau$	0.155	0.235	0.235	0.155	$3.4612 \cdot 10^{-17}$
	0.155	0.235	0.235	0.155	0.
$t = 2\tau$	0.15012	0.23	0.23	0.15012	$3.2763 \cdot 10^{-17}$
	0.15	0.23	0.23	0.15	0.
$t = 5\tau$	0.13705	0.21505	0.21505	0.13705	$2.9146 \cdot 10^{-17}$
	0.1368	0.215	0.215	0.1368	0.

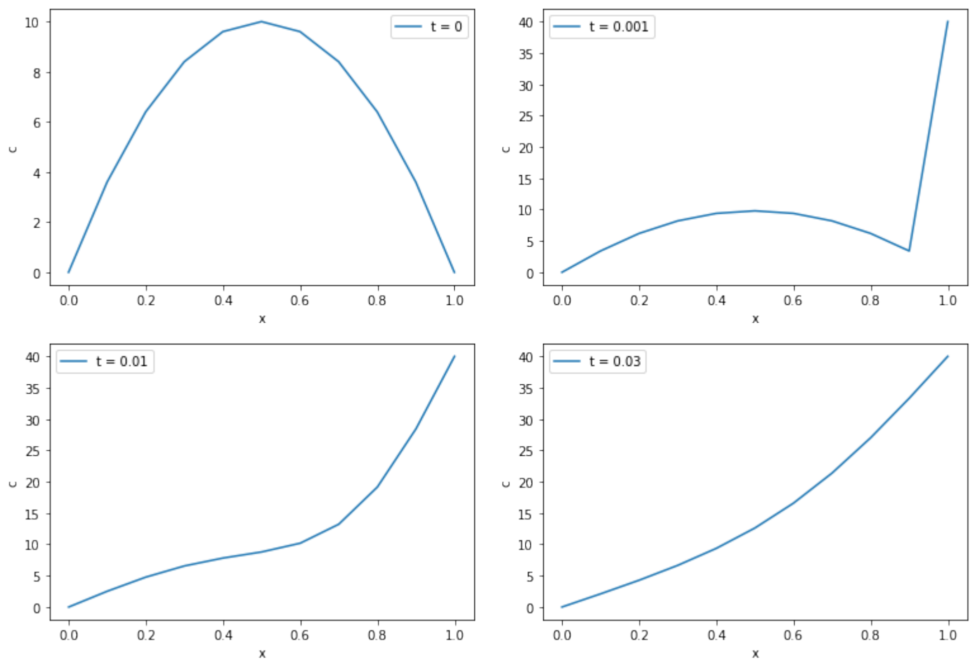
Tafla 1: Samanburður á nálgunargildum reiknuð með HW og gildum reiknuð út frá jöfnu 21

IV.

Forrit prófað með því að setja $\sigma = 1/4$, $L = 1$, $T = 1/10$, $N = 10$, $M = 100$ og

$$\begin{aligned} f(x) &= 40x(L-x), && 0 \le x \le L, \\ \psi(t) &= 0, && 0 < t \le T, \\ \phi(t) &= 40, && 0 < t \le T. \end{aligned}$$

Lausnir eru teiknaðar þegar $t = 0, 0.001, 0.01, 0.03$ og túlkaðar.



Mynd 2: Gröf af tölulegri lausn þegar $t = 0, 0.001, 0.01, 0.03$

Hægt er að ímynda sér stöng í einni rúmvídd með lengd L. Fallið gæti lýst t.d. hitastigi eða fjölda einda (samanber jafna Ficks) í stöng sem er hituð eða „diffused" í $x = L$ samkvæmt jaðarskilyrðunum ψ og ϕ. Í umræðunnni að neðan er gert ráð fyrir hitastigsdrefingu en ekki dreifingu einda.

Þegar $t = 0$ lýsir grafið upphafsástandi stangarinnar. Hitastig er mest í miðju stangarinnar en ekkert í endapunktum hennar.

Þegar $t = 0.001$ hafa jaðarskilyrðin komið inn en enn sjást áhrif upphafsskilyrðis greinilega í „bungunni" sem sést ennþá.

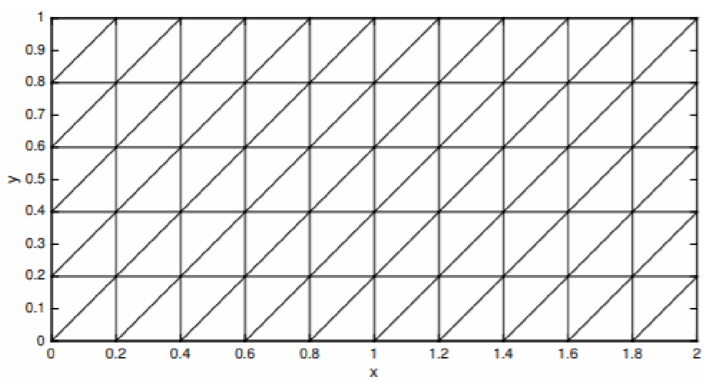
Þegar $t = 0.01$ og $t = 0.03$ sést að hitinn er hægt og rólega að streyma inn við $x = L$ á meðan hann tapast á öðrum stöðum í stönginni. Dreifing hitans virðist vera að dreifast línulega þegar líður á. Til staðfestingar var teiknað graf þegar $t = 0.1$ og þá sást að hitinn var línulega dreifður í gegnum stöngina, segja má að stöngin hafi því náð stöðugu ástandi.

2 Helmotz jafna og bútaaðferð

Í þessum hluta á að finna nálgunarlausn á Helmholtz jöfnu með Neumann og Dirichlet jaðarskilyrðum í tveimur rúmvíddum með því að nota bútaaðferð á þríhyrningsneti á svæðinu $D = \{(x, y) \in \mathbb{R}^2 : 0 < x < L_1, 0 < y < L_2\}$.
Verkefnið má setja fram með

$$\begin{cases} -\Delta u(x,y) - \lambda^2 u(x,y) = 0, & (x,y) \in D, \\ u(x,0) = w(x), u(x,L_2) = v(x), & 0 \le x \le L_1, \\ \delta_x u(0,y) = \delta_x(L,y) = 0, & 0 < y < L_2, \end{cases} \tag{2.1}$$

þar sem Δ er Laplace-virkinn, λ^2 er jákvæður fasti, og w og v eru föll.
Í bútaaðferð á þríhyrningsneti mynda rétthyrndir þríhyrningar með jafnlangar skammhliðar, h, netið. Dæmi um þríhyrningsnet má sjá á mynd 3 þegar $L_1 = 2$, $L_2 = 1$ og fjöldir þríhyrninga eftir x-ás er 10 og 5 eftir y-ásnum, skilgreint sem N og M hvort um sig fyrir þetta verkefni.



Mynd 3: Þríhyrningsnet þegar $L_1 = 2$, $L_2 = 1$, N=10 og M=5

Fjöldi punkta á þríhyrningsnetinu er $P = (M+1)(N+1)$, fylkið A er af stærð $P \times P$ og vigurinn b er af stærð $P \times 1$.
Athuga skal að almennt jaðargildisverkefni í $\mathbb{R}^2$ einfaldast mikið fyrir Helmholtz jöfnu vegna þess að föllin $f(x, y) = 0$ og $q(x, y) = -\lambda^2$ eru fastaföll.
Fyrir þetta verkefni skiptist jaðarinn δD í $\delta D_1 = \{(x, y) \in \mathbb{R}^2 : y = 0, y = L_2\}$ og $\delta D_2 = \{(x, y) \in \mathbb{R}^2 : x = 0, x = L_1\}$.
Númering punktanna á þríhyrningsnetinu fæst með vörpuninni

$$\sigma : (j, p) \longrightarrow \beta = \alpha = \sigma(j, p) = j + p(N + 1).$$

Allir punktarnir eru tölusettir með β og punktar á þríhyrningnum sem hafa β sem hornpunkt eru tölusettir með α. Framhaldið fer síðan eftir því hvort punktarnir eru innri punktar, á δD_1 eða á δD_2. Þá gefa kaflar 6.5.4, 6.5.5 og 6.5.6 í nótum að hægt er að skipta verkefninu í þrjá hluta:

1. Innri punktar
   Höfum $\beta = \sigma(j,p)$. Samtals sex þríhyrningar sem hafa þetta tiltekna β sem hornpunkt, einu fylkjastökin sem eru ekki núll í línu númer β í jöfnuhneppinu eru

$$
\begin{aligned}
a_{\beta,\beta} &= 4 + \frac{h^2}{3}q \\
a_{\beta,\alpha} &= \frac{h^2}{9}q, && \alpha = \sigma(j+1,p-1), \alpha = \sigma(j-1,p+1) \\
a_{\beta,\alpha} &= -1 + \frac{h^2}{9}q, && \alpha = \sigma(j,p-1), \alpha = \sigma(j,p+1) \\
a_{\beta,\alpha} &= -1 + \frac{h^2}{9}q, && \alpha = \sigma(j-1,p), \alpha = \sigma(j+1,p) \\
b_\beta &= h^2 f
\end{aligned}
$$

2. Punktar á δD_2
   Höfum $\beta = \sigma(j,p)$. Hér eru þrír þríhyrningar sem hafa þetta tiltekna β sem hornpunkt, einu stökin sem eru ekki núll í línu númer β í jöfnuhneppinu eru

$$
\begin{aligned}
a_{\beta,\beta} &= 2 + \frac{h^2}{6}q \\
a_{\beta,\alpha} &= -\frac{1}{2} + \frac{h^2}{18}q, && \alpha = \sigma(j-1,p), \alpha = \sigma(j+1,p) \\
a_{\beta,\alpha} &= -1 + \frac{h^2}{9}q, && \alpha = \sigma(j,p+1) \\
a_{\beta,\alpha} &= \frac{h^2}{9}q, && \alpha = \sigma(j-1,p+1) \\
b_\beta &= \frac{h^2}{2}f
\end{aligned}
$$

3. Punktar á δD_1
   Höfum $\beta = \sigma(j,p)$. Einu stökin sem eru ekki núll í línu númer β í jöfnuhneppinu eru

$$
\begin{aligned}
a_{\beta,\beta} &= 1 \\
b_\beta &= w(x), && \text{ef } y = 0 \\
b_\beta &= v(x), && \text{ef } y = L_2
\end{aligned}
$$

Forrit sem leysir ofangreint verkefni má sjá hér að neðan.

I.

```
# returns 1d representation of 2d grid
def flat_rep(j, p, N):
    alpha = j + (p)*(N+1)
    return alpha

def helmotzeq(L1, L2, h, lambda_, v, w):
```

```python
add = h/10 # used to include L1/L2 in grid
x_grid = np.arange(0, L1+add ,h)
y_grid = np.arange(0, L2+add ,h)
N = int(L1/h)
M = int(L2/h)
P = (M+1)*(N+1)
A = np.zeros((P, P))
b = np.zeros((P,1))

# make map from 1d rep to 2d xy, get xy coordinates
xy_map = {}
for x in range(N+1):
    for y in range(M+1):
        point = flat_rep(x, y, N)
        xy_map[point] = (x, y)

# categorize points(inner or not)
adj_inner = 6 # number of triangles connected to inner point
inner  = [] # list of inner point indices
D2 = [] # list of points on vertical boundary (deltaD2)
D1 = [] # list of points on horizontal boundary
D2_left = []
D2_right = []
D1_bottom = []
D1_top = []

for beta in xy_map.keys():
    x, y = xy_map[beta]
    if y == 0 or y == M:
        D1.append(beta)
        if y == 0:
            D1_bottom.append(beta)
        else:
            D1_top.append(beta)
    elif x == 0 or x == N:
        D2.append(beta)
        if x == 0:
            D2_left.append(beta)
        else:
            D2_right.append(beta)
    else:
        inner.append(beta)
```

```python
# start calculations(chapter 6.5.4 and 6.5.6 edbook)
# in this case p(x,y) = 1, q(x, y) = -lambda^2, f(x,y) = 0

# start with inner points
for beta in inner:
    j, p = xy_map[beta]
    a_bb = (2/h**2)*(2*h**2)+(h**2/18)*(-6*lambda_**2)
    A[beta, beta] = a_bb

    # 1.
    a1 = flat_rep(j+1, p-1, N)
    a2 = flat_rep(j-1, p+1, N)
    a_ba = (h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    # 2.
    a1 = flat_rep(j, p-1, N)
    a2 = flat_rep(j, p+1, N)
    a_ba = -1+(h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    # 3.
    a1 = flat_rep(j-1, p, N)
    a2 = flat_rep(j+1, p, N)
    a_ba = -1+(h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    b[beta] = 0

# boundary points
# D2
for beta in D2:
    j, p = xy_map[beta]
    # 2a.
    # a_bb, nuna er i_a = i_b og er í allar áttir, 3 þríhyrningar með bb sem hornpunkt
    # þetta tilfelli er alveg eins og áður(innri punktar)
    a_bb = 2 + ((h**2)/18)*(-3*lambda_**2)
    A[beta, beta] = a_bb
```

```
    b[beta] = 0
    if beta in D2_left:
        # 2b.
        a1, a2 = flat_rep(j, p-1, N), flat_rep(j, p+1, N)

        a_ba = (-1/2)+(h**2/18)*(-lambda_**2)
        A[beta, a1] = a_ba
        A[beta, a2] = a_ba

        # 2c.
        a = flat_rep(j+1, p, N)
        a_ba = -1+ (h**2/18)*(-2*lambda_**2)
        A[beta, a] = a_ba

        # 2d.
        a = flat_rep(j+1, p-1, N)
        a_ba = (h**2/18)*(-2*lambda_**2)
        A[beta,a] = a_ba

    else:
        # 2b.
        a1, a2 = flat_rep(j, p+1, N), flat_rep(j, p-1, N)
        a_ba = (-1/2)+(h**2/18)*(-lambda_**2)
        A[beta, a1] = a_ba
        A[beta, a2] = a_ba

        # 2c.
        a = flat_rep(j-1, p, N)
        a_ba = -1+ (h**2/18)*(-2*lambda_**2)
        A[beta, a] = a_ba

        # 2d.
        a = flat_rep(j-1, p+1, N)
        a_ba = (h**2/18)*(-2*lambda_**2)
        A[beta,a]= a_ba

# D1
for beta in D1:
    j, p = xy_map[beta]
    A[beta, beta] = 1
    x_j = x_grid[j]
    y_j = y_grid[p]
```

```
        if y_j == 0.0:
            b[beta] = w(x)
        else:
            b[beta] = v(x)

    # solve system
    c_out = np.linalg.solve(A, b)
    c_mat = np.zeros((M+1, N+1)) # (p x 1) vector

    # here the grid values are unflattened
    to_2d = {} # for grid presantation of values
    for alpha in range(M+1):
        for i in range(N+1):
            index = flat_rep(i, alpha, N)
            to_2d[index] = (alpha, i)

    for i, c in enumerate(c_out):
        index = to_2d[i]
        c_mat[index[0], index[1]] = c
    return c_mat
```

II.

Prófunarkeyrsla. Sett er $\lambda = \frac{1}{100}$, $L_1 = L_2 = 1$, $h = \frac{1}{4}$ og

$$
\begin{aligned}
w(x) &= 1, \quad 0 \le x \le 1, \\
v(x) &= 0, \quad 0 \le x \le 1
\end{aligned}
$$

```
lambda_, L1, L2, h = 1/100, 1, 1, 1/4
w = lambda x : 1
v = lambda x : 0
hz = helmotzeq(L1, L2, h, lambda_, v, w)
print(hz)
```

```
[[1.         1.         1.         1.         1.        ]
 [0.75000564 0.75000553 0.75000547 0.7500054  0.7500053 ]
 [0.50000647 0.50000634 0.50000625 0.50000616 0.50000603]
 [0.25000408 0.25000397 0.25000391 0.25000384 0.25000373]
 [0.         0.         0.         0.         0.        ]]
```

Mynd 4: Útkoma úr keyrslu

III.

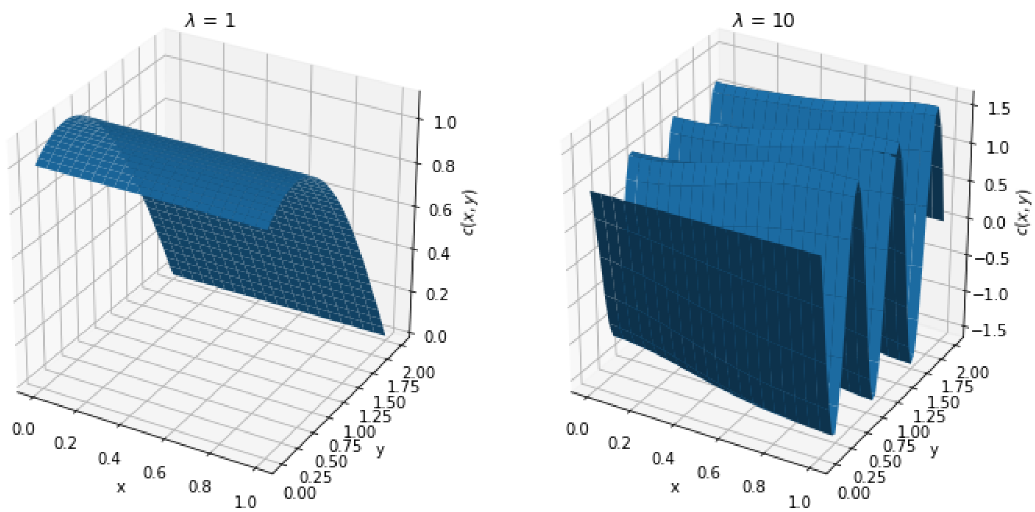
Forrit prófað á móti beinni lausn. Sett er $L_1 = 1$, $L_2 = 2$, $h = \frac{1}{20}$ og

$$w(x) = 1, \quad 0 \leq x \leq 1,$$
$$v(x) = 0, \quad 0 \leq x \leq 1.$$

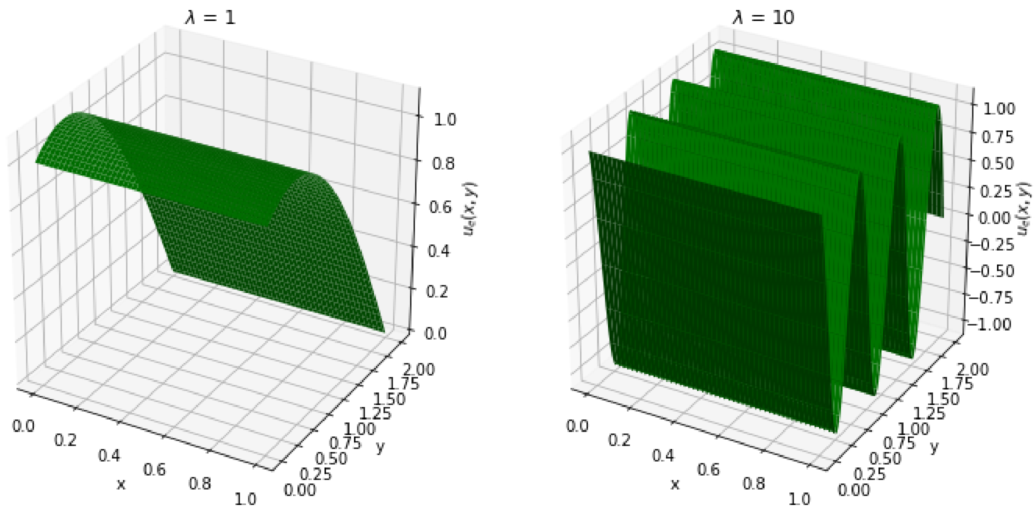
Nálgunarlausnin er borin saman við beina lausn sem er gefin með

$$u_e(x, y) = \frac{\sin \lambda (L_2 - y)}{\sin \lambda L_2}$$

fyrir $\lambda = 1$ og 10, og þrívíð mynd af þeim er teiknuð.



Mynd 5: Tölulegar lausnir



Mynd 6: Analýtískar lausnir

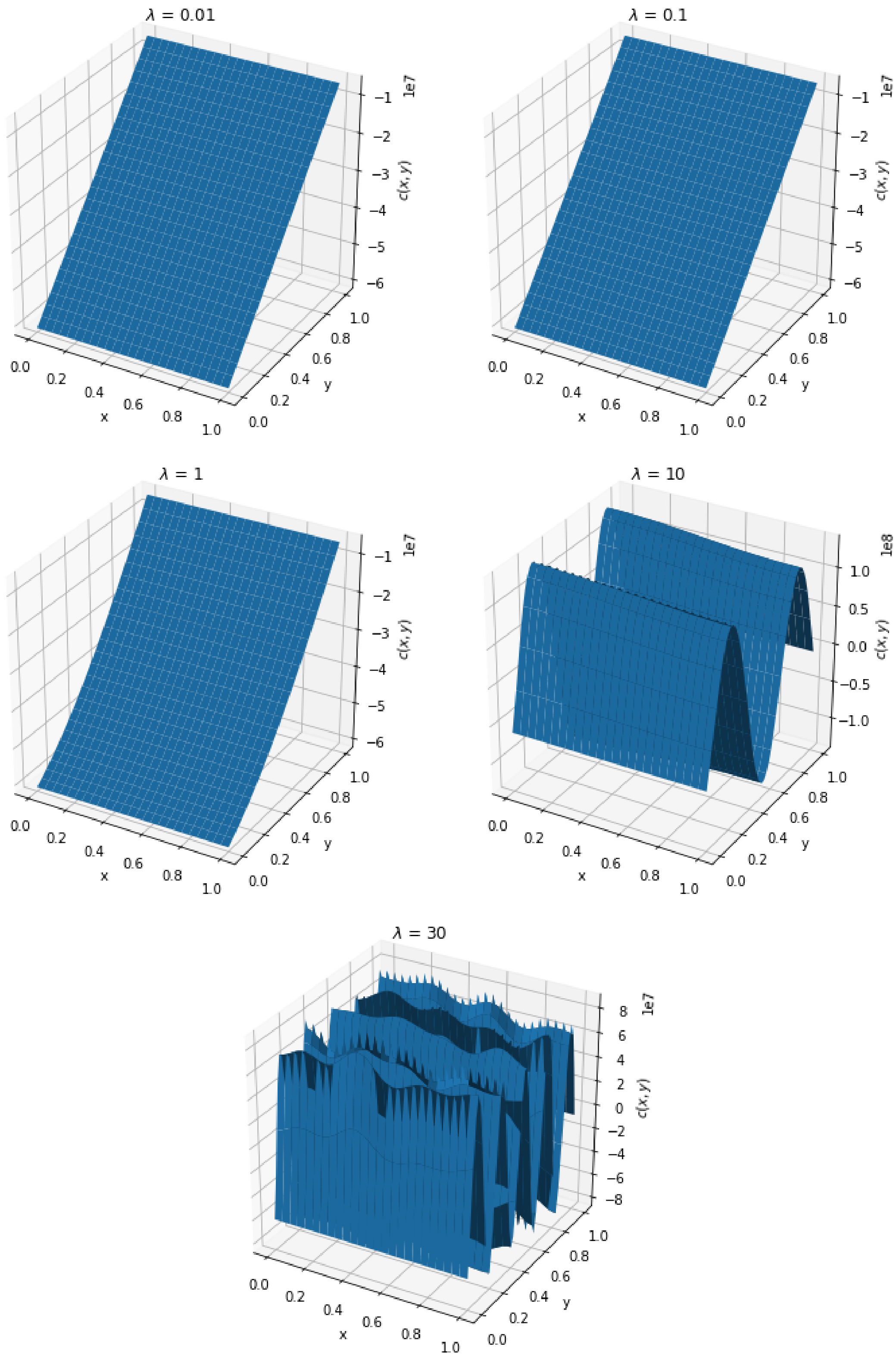
Á myndunum sést að fyrir stærri gildi á λ þá stækkar skekkjan.

IV.

Sett er $L_1 = L_2 = 1$, $h = \frac{1}{50}$ og

$$w(x) = -\frac{u_0 x}{L_1}(\frac{x}{L_1} - 1)^2(1 + \frac{x}{L_1}), \quad 0 \leq x \leq L_1,$$
$$v(x) = \frac{u_1 x}{L_1}(1 - \frac{x}{L_1})(1 + \frac{x}{L_1})^2, \qquad 0 \leq x \leq L_1,$$

þar sem $u_0 = 10$ og $u_1 = 1$. Lausnir eru teiknaðar þegar $\lambda = \frac{1}{100}, \frac{1}{10}, 1, 10, 30$.



Mynd 7: Tölulegar lausnir

Viðauki Python kóði

Hér kemur fyrir allur forritskóði sem notaður er í verkefninu

Hluti 1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import spsolve

#I)
# function to represent point on grid with one index u
def d1_rep(j, k, N):
    i = j + (k)*(N+1)
    return i

def heatwave(L, T, N, M, sigma, f, phi, psi):
    to_2d = {} # for grid presantation of values
    for alpha in range(M+1):
        for i in range(N+1):
            index = d1_rep(i, alpha, N)
            to_2d[index] = (alpha, i)

    x_grid = np.linspace(0, L, N+1)
    t_grid = np.linspace(0, T, M+1)
    h = L/N
    tau = T/M
    # construct matrix
    P = (M+1)*(N+1)
    A = np.zeros((P, P))
    b = np.zeros((P,1))
    row_index = 0 # used to insert equations into matrix

    # eq(16)
    # inner points
    for alpha in range(M):
        for i in range(1, N):
            A[row_index, d1_rep(i, alpha, N)] = (1-2*sigma)
            A[row_index, d1_rep(i, alpha+1, N)]= -1
            A[row_index, d1_rep(i+1, alpha, N)]= sigma
            A[row_index, d1_rep(i-1, alpha, N)] = sigma
            b[row_index] = 0
```

```
            row_index +=1 #next equation will appear in the following row of the matrix

    # boundary conditions
    for alpha in range (1, M+1):
        index_1 = d1_rep(0, alpha, N)
        A[row_index, index_1] = 1
        b[row_index] =psi
        row_index+=1
        index_2 = d1_rep(N, alpha, N)
        A[row_index, index_2] = 1
        b[row_index] = phi
        row_index+=1

    # inital condition
    for i in range(0, N+1):
        x = x_grid[i]
        index = d1_rep(i, 0, N)
        A[row_index, index] = 1
        b[row_index] = f(x)
        row_index+=1

    # create and calculate with sparse matrix
    S = csc_matrix(A)
    c_out = spsolve(S, b)
    c_mat = np.zeros((M+1, N+1)) # (P x 1) vector

    # here the grid values are unflattened
    for i, c in enumerate(c_out):
        index = to_2d[i]
        c_mat[index[0], index[1]] = c
    hw = c_mat
    return hw

#II)
L, T, N, M , sigma, phi, psi = 1, 1/100, 4, 5, 1/4, 0, 0
f2 = lambda x : 100
hw2 = heatwave(L, T, N, M, sigma, f2, phi, psi)
print('Numerical solution:\n',hw2)

#III)
sigma, L, T, N, M, phi, psi = 1/4, 1, 1/100, 10, 100, 0, 0
f3 = lambda x : x*(L-x)
```

```python
h = L/N
tau = T/M
kappa = (sigma*h**2)/tau
hw3 = heatwave(L, T, N, M, sigma, f3, phi, psi)

# based on (20)
def corr_sol(x, t, kappa, L):
    w = np.pi/L
    u_xt = 0 # init
    for n in range(0, 21):
        u_xt += ((1)/(2*n+1)**3)*np.exp(-kappa*(2*n+1)**2*w**2*t)*np.sin((2*n+1)*w*x)
    u_xt *= (8*L**2)/(np.pi**3)
    return u_xt

# prepare table for comparision of numerical and analytic solution for the given values
t_list = [tau, 2*tau, 5*tau]
x_list = [0.2, 0.4, 0.6, 0.8, 1]
x_grid = np.linspace(0, L, N+1)
t_grid = np.linspace(0, T, M+1)

# table/matrix with analytic values
U_mat = np.zeros((3, 5))
for i, x in enumerate(x_list):
    for j, t in enumerate(t_list):
        u_a  = corr_sol(x, t, kappa, L)
        U_mat[j, i] = u_a

# next we will find the corresponding indices in hw for the table
x_inds = []
t_inds = []
x_grid = np.around(x_grid, decimals = 10)
t_grid = np.around(t_grid, decimals = 10)

for x in x_list:
    x_inds.append(int(np.where(x_grid == x)[0]))
for t in t_list:
    t_inds.append(int(np.where(t_grid == t)[0]))

# table/matrix with numerical values
table = np.zeros((3, 5))
num_sol = hw3
```

```
for i ,x in enumerate(x_inds):
    for j, t in enumerate(t_inds):
        table[j, i] = num_sol[t, x]

print('Numerical solution:\n',table)
print()
print('Analytic solution:\n',U_mat)

#IV)
sigma, L, T, N, M, phi, psi = 1/4, 1, 1/10, 10, 100, 40, 0
f4 = lambda x : 40*x*(L-x)
hw4 = heatwave(L, T, N, M, sigma, f4, phi, psi)

# prepare to graph numerical solution for t = 0 , 0.001 , 0.01 , 0.03
t_list = [0, 0.001, 0.01, 0.03, 0.1]
t_grid = np.linspace(0, T, M+1)
t_grid = np.around(t_grid, decimals = 10)

t_inds = []
for t in t_list:
    t_inds.append(int(np.where(t_grid == t)[0]))

x_grid = np.linspace(0, L, N+1)

for j, i in enumerate(t_inds):
    plt.plot(x_grid, hw4[i,:],label = f't = {t_list[j]}')
    plt.xlabel('x')
    plt.ylabel('c')
    plt.legend()
    plt.show()
```

Hluti 2

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d

# I)
# returns 1d representation of 2d grid
def flat_rep(j, p, N):
    alpha = j + (p)*(N+1)
    return alpha
```

```python
def helmotzeq(L1, L2, h, lambda_, v, w):
    add = h/10 # used to include L1/L2 in grid
    x_grid = np.arange(0, L1+add ,h)
    y_grid = np.arange(0, L2+add ,h)
    N = int(L1/h)
    M = int(L2/h)
    P = (M+1)*(N+1)
    A = np.zeros((P, P))
    b = np.zeros((P,1))

    # make map from 1d rep to 2d xy, get xy coordinates
    xy_map = {}
    for x in range(N+1):
        for y in range(M+1):
            point = flat_rep(x, y, N)
            xy_map[point] = (x, y)

    # categorize points(inner or not)
    adj_inner = 6 # number of triangles connected to inner point
    inner  = [] # list of inner point indices
    D2 = [] # list of points on vertical boundary (deltaD2)
    D1 = [] # list of points on horizontal boundary
    D2_left = []
    D2_right = []
    D1_bottom = []
    D1_top = []

    for beta in xy_map.keys():
        x, y = xy_map[beta]
        if y == 0 or y == M:
            D1.append(beta)
            if y == 0:
                D1_bottom.append(beta)
            else:
                D1_top.append(beta)
        elif x == 0 or x == N:
            D2.append(beta)
            if x == 0:
                D2_left.append(beta)
            else:
                D2_right.append(beta)
        else:
```

```
        inner.append(beta)

# start calculations(chapter 6.5.4 and 6.5.6 edbook)
# in this case p(x,y) = 1, q(x, y) = -lambda^2, f(x,y) = 0

# start with inner points
for beta in inner:
    j, p = xy_map[beta]
    a_bb = (2/h**2)*(2*h**2)+(h**2/18)*(-6*lambda_**2)
    A[beta, beta] = a_bb

    # 1.
    a1 = flat_rep(j+1, p-1, N)
    a2 = flat_rep(j-1, p+1, N)
    a_ba = (h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    # 2.
    a1 = flat_rep(j, p-1, N)
    a2 = flat_rep(j, p+1, N)
    a_ba = -1+(h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    # 3.
    a1 = flat_rep(j-1, p, N)
    a2 = flat_rep(j+1, p, N)
    a_ba = -1+(h**2/18)*(-2*lambda_**2)
    A[beta, a1] = a_ba
    A[beta, a2] = a_ba

    b[beta] = 0

# boundary points
# D2
for beta in D2:
    j, p = xy_map[beta]
    # 2a.
    # a_bb, nuna er i_a = i_b og er í allar áttir, 3 þríhyrningar með bb sem hornpunkt
    # þetta tilfelli er alveg eins og áður(innri punktar)
    a_bb = 2 + ((h**2)/18)*(-3*lambda_**2)
```

```
        A[beta, beta] = a_bb
        b[beta] = 0
        if beta in D2_left:
            # 2b.
            a1, a2 = flat_rep(j, p-1, N), flat_rep(j, p+1, N)

            a_ba = (-1/2)+(h**2/18)*(-lambda_**2)
            A[beta, a1] = a_ba
            A[beta, a2] = a_ba

            # 2c.
            a = flat_rep(j+1, p, N)
            a_ba = -1+ (h**2/18)*(-2*lambda_**2)
            A[beta, a] = a_ba

            # 2d.
            a = flat_rep(j+1, p-1, N)
            a_ba = (h**2/18)*(-2*lambda_**2)
            A[beta,a] = a_ba

        else:
            # 2b.
            a1, a2 = flat_rep(j, p+1, N), flat_rep(j, p-1, N)
            a_ba = (-1/2)+(h**2/18)*(-lambda_**2)
            A[beta, a1] = a_ba
            A[beta, a2] = a_ba

            # 2c.
            a = flat_rep(j-1, p, N)
            a_ba = -1+ (h**2/18)*(-2*lambda_**2)
            A[beta, a] = a_ba

            # 2d.
            a = flat_rep(j-1, p+1, N)
            a_ba = (h**2/18)*(-2*lambda_**2)
            A[beta,a]= a_ba

    # D1
    for beta in D1:
        j, p = xy_map[beta]
        A[beta, beta] = 1
        x_j = x_grid[j]
```

```
        y_j = y_grid[p]
        if y_j == 0.0:
            b[beta] = w(x)
        else:
            b[beta] = v(x)

    # solve system
    c_out = np.linalg.solve(A, b)
    c_mat = np.zeros((M+1, N+1)) # (p x 1) vector

    # here the grid values are unflattened
    to_2d = {} # for grid presantation of values
    for alpha in range(M+1):
        for i in range(N+1):
            index = flat_rep(i, alpha, N)
            to_2d[index] = (alpha, i)

    for i, c in enumerate(c_out):
        index = to_2d[i]
        c_mat[index[0], index[1]] = c
    return c_mat

#II)
lambda_, L1, L2, h = 1/100, 1, 1, 1/4
w = lambda x : 1
v = lambda x : 0
hz = helmotzeq(L1, L2, h, lambda_, v, w)
print(hz)

#III)
lambda_list = [1, 10]
L1, L2, h = 1, 2, 1/20
w = lambda x : 1
v = lambda x : 0

# plot numerical solution
print('Numerical soulutions:\n')
for i in range(len(lambda_list)):
    num_sol3 = helmotzeq(L1, L2, h, lambda_list[i], v, w)
    fig = plt.figure(figsize = (6,6))
    ax = plt.axes(projection="3d")
    x_num = np.arange(0, L1+h/2 ,h)
```

```
    y_num = np.arange(0, L2+h/2 ,h)
    X_num, Y_num = np.meshgrid(x_num, y_num)
    Z_num = num_sol3
    ax.plot_surface(X_num, Y_num, Z_num)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('$c(x,y)$')
    ax.set_title(f'$\lambda$ = {lambda_list[i]}')
    plt.show()

def corr_sol(x,y,lambda_):
    return (np.sin(lambda_*(L2-y)))/(np.sin(lambda_*L2))

# plot correct solution
print('Correct soulutions:\n')
for i in range(len(lambda_list)):
    fig = plt.figure(figsize = (6,6))
    ax = plt.axes(projection="3d")
    x_corr = np.linspace(0, L1, 100)
    y_corr = np.linspace(0, L2, 100)
    X_corr, Y_corr = np.meshgrid(x_corr, y_corr)
    Z_corr = corr_sol(X_corr, Y_corr,lambda_list[i])
    ax.plot_surface(X_corr, Y_corr, Z_corr, color='green')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('$u_e(x,y)$')
    ax.set_title(f'$\lambda$ = {lambda_list[i]}')
    plt.show()

#IV)
lambda_list = [1/100, 1/10, 1, 10, 30]
L1, L2, h, u0, u1 = 1, 1, 1/50, 10, 1
w = lambda x : (-u0*x)/L1*(x/L1-1)**2*(1+x/L1)
v = lambda x : (u1*x)/L1*(1-x/L1)*(1+x/L1)**2

# plot numerical solution for lambda = ...
for i in range(len(lambda_list)):
    num_sol4 = helmotzeq(L1, L2, h, lambda_list[i], v, w)
    fig = plt.figure(figsize = (6,6))
    ax = plt.axes(projection="3d")
    x_num = np.arange(0, L1+h/2 ,h)
    y_num = np.arange(0, L2+h/2 ,h)
```

```python
X_num, Y_num = np.meshgrid(x_num, y_num)
Z_num = num_sol4
ax.plot_surface(X_num, Y_num, Z_num)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('$c(x,y)$')
ax.set_title(f'$\lambda$ = {lambda_list[i]}')
plt.show()
```