1c)

```python
import numpy as np


def piecewise_func(c, Ne, psi, D, N):
    """
    Input parametere:
        c    : Koeffisient vektoren, oppndd ved  lse matriselikningen Ac = b
        Ne   : Antall elementer
        psi  : En liste med basisfunksjonene som funksjoner, alts at de kan ta et argument x
        D    : Grensebetingelsen for u(1,t) = D
        N    : Antall punkter jeg vil evaluere u i hvert element
    """

    c = np.append(c, D)    # Legger til grensebetingelsen slik at vi unngr en if-test nr vi finner lsningen

    N_p1 = int(Ne/2)       # Antall P1 elementer
    N_p2 = int(Ne/2)       # Antall P2 elementer


    x = np.linspace(0, 1, Ne+1)

    # Degrees of freedom map som holder de lokale nodene for P2 elementene
    dof_map_P2 = []
    node = 0
    for e in range(N_p2):
        dof_map_P2.append([])
        for i in range(node, 3+node):
            dof_map_P2[e].append(i)
        node += 2

    # Degrees of freedom map som holder de lokale nodene for P1 elementene
    dof_map_P1 = []
    for e in range(N_p1):
        dof_map_P1.append([])
        for i in range(node, 2+node):
            dof_map_P1[e].append(i)
        node += 1


    x_total = []    # Holder den totale x-arrayen
    u = []          # Holder den totale lsningen

    counter = 0
    for index, e in enumerate(dof_map_P2):
        start = x[index]
        slutt = x[index+1]

        x_lokal = np.linspace(start, slutt, N)

        u.append(c[dof_map_P2[e][0]]*psi[dof_map_P2[e][0]](x_lokal) + c[dof_map_P2[e][1]]*psi[dof_map_P2[e][1]](x_lokal) + c[dof_map_P2[e][2]]*psi[dof_map_P2[e][2]](x_lokal))

        x_total = np.concatenate((x_total, x_lokal))
        counter = index

    for index, e in enumerate(dof_map_P1):
        start = x[index+counter]
        slutt = x[index+1+counter]

        x_lokal = np.linspace(start, slutt, 10)

        u.append(c[dof_map_P1[e][0]]*psi[dof_map_P1[e][0]](x_lokal) + c[dof_map_P1[e][1]]*psi[dof_map_P1[e][1]](x_lokal) + c[dof_map_P1[e][2]]*psi[dof_map_P1[e][2]](x_lokal))

        x_total = np.concatenate((x_total, x_lokal))


    u = np.concatenate(u, axis=0)      # Siden u er en liste med arrayer, gjr dette u til n array

    # For  unng at samme x-koordinat blir brukt to ganger, finner indeksen til de elementene i x_total som er like
    idx = np.unique(x_total, return_index=True)[0]

    X = x_total[idx]
```

```
72      U = u[idx]
73
74      return X, U
```

For å unngå if-tester inne i for-løkkene er selve utregningen delt opp i to løkker, avhengig av hvor vi er på domenet. Ettersom vi allerede vet hvor på x-aksen de ulike elementene er definert, trenger vi ikke if-tester siden vi kan finne de med indeksering, gitt at elementene er sortert i logisk og stigende rekkefølge. Det ville vært verre om vi hadde hatt en tilfeldig sortering av elementene, da det måtte ha gått med ekstra regneoperasjoner til å sortere disse.

Ettersom vi har et x-array kan vi for hvert element regne ut følgende vektorisert:

$$u(\boldsymbol{x}) = c_n\psi_n(\boldsymbol{x}) + \ldots + c_m\psi_m(\boldsymbol{x}), \tag{1}$$

for $\boldsymbol{x} \in [nh, mh]$, og det er dette som skjer i linje 49 og 60.

1d)

Trapesmetoden for numerisk integrasjon er gitt av

$$\int_a^b f(x)dx \approx h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{N-1} f(x_i)\right), \tag{2}$$

der h er steglengden og N er antall integrasjonspunkter. L_2 normen, som er definert som

$$L_2 = \sqrt{\int_\Omega (u_e(x) - u(x))^2 dx}, \tag{3}$$

blir da

$$L_2 \approx \sqrt{h\left(\frac{(u_e(a)-u(a))^2 + (u_e(b)-u(b))^2}{2} + \sum_{i=1}^{N-1}(u_e(x_i)-u(x_i))^2\right)}, \tag{4}$$

siden $f(x) = u_e(x) - u(x)$.

```
1   import numpy as np
2
3
4   def L2_norm(u_e, u, Ns):
5       """
6       Input parametere:
7           u_e : den eksakte lsningen som er en funksjon av x
8           u   : den tilnrmede lsningen i diskrete punkter
9           Ns  : antall integrasjonspunkter
10      """
11
12      h = 1/(Ns-1)
13      x = np.linspace(0, 1, Ns)
14
15
16      # Regner ut arealet av trapesene som tilhrer de indre punktene
17      inder_punkter = 0
18      for i in range(1,Ns):
19          indre_punkter += (u_e(x[i]) - u[i])**2
20
21      endepunkter = ((u_e(x[0]) - u[0])**2 + (u_e(x[-1]) - u[-1])**2)/2
22
23      return np.sqrt(h*(endepunkter + indre_punkter))
```

Antar at Ns samsvarer med antall punkter vi har brukt for å finne den numeriske løsningen i forrige oppgave.

Når det gjelder konvergensrate vet vi at siden basisfunksjonene til P2 elementene er kvadratiske polynomer, har de en konvergensrate på 3, mens basisfunksjonene for P1 elementene har en rate på 2. Dette er basert på et a priori estimat av løsningen, som sier at feilen konvergerer som h^{s+1} der s er polynomgraden til basisfunksjonene. Derfor er det naturlig å anta at konvergensraten til L_2 normen vil ligge et sted mellom 2 og 3.

For å finne koeffisientene i likningssystemet $Ac = b$, utfører man en matrise invertering av A. Dette betyr at A må være ikke-singulær. Et problem som kan oppstå mtp. denne betingelsen er hvis noen av elementene i A er veldig små. Dette kan gi numeriske feil, som kan føre til at koeffisientene ikke blir helt korrekte.

2f)

I den første metoden så diskretiserer vi før vi lineariserer. Denne metoden, så vidt jeg kan finne, kalles på norsk for ettersleps metoden [1]. Siden vi Taylor utvikler om u^-, og kun beholder de lineære leddene vil vi i teorien ha tregere konvergens enn når vi gjennomfører såkalt kvasi-linearisering [1]. Dette er den andre metoden vi brukte, der vi først lineariserer og så diskretiserer. Siden de begge er basert på Newtons metode, som kan vises at har kvadratisk konvergens. Ettersom ettersleps metoden har tregere konvergens enn kvasi-lineariseringen, antar jeg derfor at denne har nærmere kvadratisk konvergens.

En flervariabel versjon av ettersleps metoden medfører at man må regne ut Jacobi determinanten. Dette kan være en kostbar operasjon, som kan bidra til tregere konvergens.

2g)

Siden vi har diskrete punkter, så kan vi tilnærme dobbelt integralet med to uavhengige summer, som løper over de ulike x og t verdiene. Ettersom den eksakte løsning er definert overalt, mens den approksimerte er begrenset til diskrete punkter tilnærmer vi L_2 normen med følgende dobbel sum

$$\int_0^T \int_0^1 (u_e(x,t) - u(x,t))^2 dxdt \approx \Delta x \Delta t \sum_{i=0}^{N_x} \sum_{j=0}^{N_t} f(x_i, t_j), \tag{5}$$

der $f(x,t) = (u_e(x,t) - u(x,t))^2$, $\Delta x = 1/N_x$ og $\Delta t = T/N_t$. Der vi vet at for $j = 0$ så har vi intital betingelsen. Fra oppgave 2b) vet vi at den diskrete 1D diffusjonslikningen har en trunkeringsfeil som går som $\mathcal{O}(\Delta x^2, \Delta t)$, ergo antar vi at konvergensraten er lineær, ettersom leddet Δt dominerer Δx^2.

Referanser

[1] Jan Aarseth. Numeriske beregninger. http://folk.ntnu.no/leifh/teaching/tkt4140/NumeriskeBeregninger.pdf, 2014.