Lecture 2

Takiko Sasaki *

March 16, 2023

Contents

1 Introduction

1.1 Calculation by Python

```
print("Hello")
```

is “comment out ”. Python does not recognize.

*t-sasaki@musashino-u.ac.jp

```
# summation
1 + 1
```

```
# subtraction
2 - 1
```

```
# multiplication
3 * 2
```

```
# division
4 / 3
```

```
# substitution
c = 1 + 2
print(c)
```

```
# power
c**2
```

```
# list
d = [1, 2, 3]
print(d)
```

```
# addition of elements to the list
d.append(4)
print(d)
```

1.2 Library

Normally, a library is a collection of books or is a room or place where many books are stored to be used later. Similarly, in the programming world, a library is a collection of precompiled codes that can be used later on in a program for some specific well-defined operations. https://www.geeksforgeeks.org/libraries-in-python/

NumPy is the fundamental package for scientific computing in Python.

```
import numpy as np
```

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

```
np.array(range(10))
```

```
np.array(range(2,5))
```

1.3 For loops

A for loop is used for iterating over a sequence. We consider

$$a_1 = 1, \quad a_{n+1} = 2a_n + 1.$$

Then,

$$a_8 = 15$$

```
a = 1
for n in range(7):
  a = a + 2
print(a)
```

Next, we consider the summation:

$$S_n = \sum_{k=1}^{n} a_k.$$

```
a = 1
b = 1 # summation
for n in range(7):
  a = 2*a +  1
  b = b + a
print(a)
print(b)
```

1.4 While loops

With the while loop we can execute a set of statements as long as a condition is true.

```
k = 5
while k > 0:
  k = k - 1
  print(k)
```

1.5 If, elif, else statement

Python supports the usual logical conditions from mathematics:

- Equals: a `==` b
- Not Equals: a `!=` b
- Less than: a `<` b
- Less than or equal to: a `<=` b
- Greater than: a `>` b
- Greater than or equal to: a `>=` b

An "if statement" is written by using the if keyword.

The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".

The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 4
if a==2:
  print("This is two.")
elif a<4:
  print("This is less than four.")
else:
  print("This is not less than four")
```

1.6 Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

```
def hello():
  print("hello")
```

```
hello()
```

```
# a,b : arguments
def add(a,b):
  c = a + b
  return c
```

```
add(1,2)
```

1.7 Plot

```
import matplotlib.pyplot as plt
```

```
xvec = np.linspace(-2,2,100)
# We devide [-2,2] into 100 points.
yvec = np.sin(xvec)
plt.plot(xvec,yvec)
```

2 Heat equation

We consider

$$\begin{cases} u_t = u_{xx} & (0 < x < 1,\ t > 0) \\ u(0,t) = 0, \quad u(1,t) = 0 & (t > 0) \\ u(x,0) = a(x) & (0 \le x \le 1) \end{cases} \tag{2.1}$$

- $N \in \mathbb{N},\ h = \dfrac{1}{N+1}$
- $0 = x_0 < x_1 < x_2 < \cdots < x_i = ih < x_{i+1} < \cdots < x_{N+1} = 1$
- $t_n = n\tau$
- $u_i^n \approx u(x_i, t_n)$

2.1 Explicit scheme

The following is a explicit scheme to the heat equation.

$$\begin{cases} \dfrac{u_i^{n+1} - u_i^n}{\tau} = \dfrac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{h^2} & (1 \le i \le N,\ n \ge 0), \\ u_0^n = u_{N+1}^n = 0 & (n \ge 1), \\ u_i^0 = a(x_i) & (0 \le i \le N+1) \end{cases} \tag{2.2}$$

Let $\lambda = \dfrac{\tau}{h^2}$. Then, we have

- In the case $2 \le i \le N$, we have

$$\begin{aligned} u_i^{n+1} &= u_i^n + \lambda(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \\ &= \lambda u_{i+1}^n + (1-2\lambda)u_i^n + \lambda u_{i-1}^n \end{aligned}$$

- In the case $i = 1$, we have

$$u_1^{n+1} = \lambda u_2^n + (1-2\lambda)u_1^n + \lambda \underbrace{u_0^n}_{=0} = \lambda u_2^n + (1-2\lambda)u_1^n$$

- In the case $i = N$, we have

$$u_N^{n+1} = \lambda \underbrace{u_{N+1}^n}_{=0} + (1-2\lambda)u_N^n + \lambda u_{N-1}^n$$
$$= (1-2\lambda)u_N^n + \lambda u_{N-1}^n.$$

```
# Import Library
import numpy as np
import matplotlib.pyplot as plt
```

We consider the following initial conditions:

$$f(x) = \begin{cases} y = x & (0 \le x \le \frac{1}{2}) \\ y = 1-x & (\frac{1}{2} < x \le 1) \end{cases}, \tag{2.3}$$

$$g(x) = \sin(\pi x) \quad (0 \le x \le 1). \tag{2.4}$$

```
# Definition of Initial data
def f(x):
    y = x.copy()
    for i in range(0,len(y)):
        if y[i] > 0.5:
            y[i] = 1-y[i]
    return y
```

```
def g(x):
  y = np.sin(np.pi*x)
  return y
```

```
# space interval
x = np.linspace(0,1,100)
```

```
plt.plot(x,f(x))
```

```
plt.plot(x,g(x))
```

```
# space interval and space mesh
N=100
x=np.linspace(0.0, 1.0, N+2)
h=1.0/(N+1)
```

```
# time increment
lam = 0.5
tau = lam*h**2
```

```
# for visualization
dt=0.01
skip=int(dt/tau)
```

```
# Computation time
Tmax=1.0
#number o iteration
nmax=int(Tmax/tau)
```

```
# Container for the solution
u=np.zeros([N+2,nmax])
```

```
# initial data
u[:,0]=f(x)
```

```
#plot initial data
plt.plot(x,u[:,0],color='red')
```

```
# plot initial data
plt.plot(x,u[:,0],color='red')
# solve the explicit scheme
```

```
for n in range(nmax-1):
    u[0,n+1]=0
    u[N+1,n+1]=0
    for i in range(1,N):
        u[i,n+1]=(1-2*lam)*u[i,n]+lam*(u[i-1,n]+u[i+1,n])
    if n%skip == 50:
        plt.plot(x,u[:,n+1],color='blue')
```

2.2 Discrete maximum and minimum principle (explicit scheme)

Assume that

$$(0 <)\lambda \leq \frac{1}{2}.$$

Then, the solution $\boldsymbol{u}^n = (u_i) \in \mathbb{R}^n$ of the explicit scheme satisfies

$$\begin{aligned}\max_{0\leq i\leq N+1} u_i^{n+1} &\leq \max_{0\leq i\leq N+1} u_i^n \leq \cdots \leq \max_{0\leq i\leq N+1} u_i^0 \\ &= \max_{0\leq i\leq N+1} f(x_i) \\ \min_{0\leq i\leq N+1} u_i^{n+1} &\geq \min_{0\leq i\leq N+1} u_i^n \geq \cdots \geq \min_{0\leq i\leq N+1} u_i^0 \\ &= \min_{0\leq i\leq N+1} f(x_i)\end{aligned}$$

2.3 Exercise

Plot the solution in the cases $\lambda < 0.5$, $\lambda = 0.5$ and $\lambda > 0.5$.

2.4 θ scheme

We consider there average with weight $\theta \in [0,1]$ (θ scheme)

$$\begin{cases} \dfrac{u_i^n - u_i^{n-1}}{\tau} \\ = (1-\theta)\dfrac{u_{i+1}^{n-1} - 2u_i^{n-1} + u_{i-1}^{n-1}}{h^2} \\ \qquad +\theta\dfrac{u_{i+1}^{n} - 2u_i^{n} + u_{i-1}^{n}}{h^2} \\ \qquad\qquad (1 \leq i \leq N,\ n \geq 1), \\ u_0^n = u_{N+1}^n \qquad (n \geq 1) \\ u_i^0 = a(x_i) \qquad (0 \leq i \leq N+1) \end{cases} \tag{2.5}$$

Let $\lambda = \frac{\tau}{h^2}$. Then, we have

$$\begin{aligned}&(1+2\theta\lambda)u_i^n - \theta\lambda(u_{i+1}^n + u_{i-1}^n) \\ &= (1-2(1-\theta))u_i^{n-1} + (1-\theta)\lambda(u_{i+1}^{n-1} + u_{i-1}^{n-1}).\end{aligned}$$

Hence we have

$$A\boldsymbol{u}^n = B\boldsymbol{u}^{n-1}, \quad \boldsymbol{u}^0 = \boldsymbol{a}$$

Here,

$$\boldsymbol{u}^n = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{pmatrix} \in \mathbb{R}^N, \quad \boldsymbol{a} = \begin{pmatrix} a(x_1) \\ a(x_2) \\ \vdots \\ a(x_N) \end{pmatrix} \in \mathbb{R}^N,$$

$$A = \begin{pmatrix} 1+2\theta\lambda & -\theta\lambda & & & 0 \\ -\theta\lambda & 1+2\theta\lambda & -\theta\lambda & & \\ & \ddots & \ddots & \ddots & \\ & & -\theta\lambda & 1+2\theta\lambda & -\theta\lambda \\ 0 & & & -\theta\lambda & 1+2\theta\lambda \end{pmatrix} \in \mathbb{R}^{N\times N},$$

$$B = \begin{pmatrix} 1-2(1-\theta)\lambda & (1-\theta)\lambda & & & \\ (1-\theta)\lambda & 1-2(1-\theta)\lambda & (1-\theta)\lambda & & \\ & \ddots & \ddots & \ddots & \\ & & (1-\theta)\lambda & 1-2(1-\theta)\lambda & (1-\theta)\lambda \\ 0 & & & (1-\theta)\lambda & 1-2(1-\theta)\lambda \end{pmatrix} \in \mathbb{R}^{N\times N}$$

```
## functions for visualization
def functz(x,y,u):
    z = u[x,y]
    return z
```

```
# Library
from mpl_toolkits.mplot3d import Axes3D
from numpy import linalg as LA
```

```
# theta scheme
def heat_theta(N,lam,theta,Tmax):
    # space mesh
    h = 1/(N+1)
    # time increments
    tau = lam*h**2
    # number of increments
    nmax = int(Tmax/tau)
    # for visualization
    dt = 0.01
```

```
skip = int(dt/tau)

# space  interval
x = np.linspace(0.0,1.0,N+2)
# container for solution
u = np.zeros([N+2,nmax])

# initial data
u[:,0] = g(x)
# plot initial data
plt.plot(x, u[:,0], color='red')

# bondary condition
for i in range(nmax):
    u[0,i] = 0 #x[0] で u =0
    u[N+1,i] = 0 #x[m] で u = 0

# Matrix
# zero matrix (N*N)
A = np.zeros([N,N])
for i in range(N):
    A[i,i] = 1 + 2*theta*lam
    #Substitute 1 + 2*theta*lam
    # to the diagonal components of A
    if i>=1:
    # Substitute -theta*lam
    # to the two sides
    # of the diagonal components
        A[i,i-1] = -theta*lam
        A[i-1,i] = -theta*lam

# zero matrix (N*N)
B = np.zeros([N,N])#
for i in range(N):
    B[i,i] = 1 - 2*(1-theta)*lam
    #Substitute 1 - 2*(1-theta)*lam
    #to the diagonal component of A
    if i >= 1:
    # Substitute (1-theta)*lam
    # to the two sides
    # of the diagonal component
        B[i,i-1] = (1-theta)*lam
        B[i-1,i] = (1-theta)*lam

# Solve the finite difference scheme
U = u[1:N+1,:]
err = 0
for n in range(nmax-1):
        UU = np.dot(B,U[:,n]) # B*u
        U[:,n+1] = np.linalg.solve(A,UU)
        # A^{-1}*B*u
        u[1:N+1,n+1]=U[:,n+1]
        # for the visialization
        if n%skip == 50:
          plt.plot(x,u[:,n+1],color='blue')

    # 3D visialization
    xx= list(range(N+2))
    yy = list(range(nmax))

    X,Y = np.meshgrid(xx,yy)
    Z = functz(X,Y,u)
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_wireframe(X,Y,Z)
    ax.set_xlabel('x')
    ax.set_ylabel('t')
    ax.set_zlabel('u')

    return err
```

```
heat_theta(100,1,1,1)
```

2.5 Discrete maximum and minimum principle (θ scheme)

Assume that

$$0 \le \theta \le 1, \quad 2(1-\theta)\lambda \le 1.$$

Then, the solution $\boldsymbol{u}^n = (u_i) \in \mathbb{R}^n$ of the explicit scheme satisfies

$$\max_{0\le i\le N+1} u_i^{n+1} \le \max_{0\le i\le N+1} u_i^n \le \cdots \le \max_{0\le i\le N+1} u_i^0 = \max_{0\le i\le N+1} f(x_i)$$
$$\min_{0\le i\le N+1} u_i^{n+1} \ge \min_{0\le i\le N+1} u_i^n \ge \cdots \ge \min_{0\le i\le N+1} u_i^0 = \min_{0\le i\le N+1} f(x_i)$$

2.6 Exercise

Plot the solution in the cases $\theta = 1$ and $\lambda < 0.5$, $\lambda = 0.5$, $\lambda > 0.5$.

2.7 Error estimates

The following function satisfies the heat equation.

$$u_{exact}(x,t) = \exp(-\pi^2 t)\sin(\pi x)$$

```
## exact solution
def func_sol(x,t):
    y = np.exp(-np.pi**2*t)*np.sin(np.pi*x)
    return y
```

For arbitrary $T > 0$, we set $Q = [0,1] \times [0,T]$. Assume that

$$0 \le \theta \le 1, \quad 2(1-\theta)\lambda \le 1,$$

$$\begin{cases} \partial_x^m u \in C(T) & (0 \le m \le 4), \\ \partial_t^l u \in C(T) & \begin{cases} (0 \le l \le 2) & \text{if } \theta \ne \frac{1}{2} \\ (0 \le l \le 3) & \text{if } \theta = \frac{1}{2} \end{cases} \end{cases}.$$

Then, we

$$\|\boldsymbol{e}^{(n)}\|_\infty \le \begin{cases} C_{T,\theta}(\tau + h^2) & (\theta \ne \frac{1}{2}) \\ C_{T,\frac{1}{2}}(\tau^2 + h^2) & (\theta = \frac{1}{2}) \end{cases}$$

where

$$C_{T,\theta} = \begin{cases} \frac{1}{2}\|\partial_t^2 u\|_{L^\infty(Q)} + \frac{k}{12}\|\partial_x^2 u\|_{L^\infty(Q)}, \\ \frac{5}{12}\|\partial_t^2 u\|_{L^\infty(Q)} + \frac{k}{12}\|\partial_x^2 u\|_{L^\infty(Q)} \end{cases}.$$

```
# theta scheme for the error estimates
def heat_theta_error(N,lam,theta,Tmax):
    # space mesh
    h = 1/(N+1)
    # time increments
    tau = lam*h**2
    # number of increments
    nmax = int(Tmax/tau)

    # space  interval
    x = np.linspace(0.0,1.0,N+2)
    # container for solution
    u = np.zeros([N+2,nmax])

    # initial data
    u[:,0] = g(x)

    # bondary condition
    for i in range(nmax):
        u[0,i] = 0 #x[0] ↔ u =0
        u[N+1,i] = 0 #x[m] ↔ u = 0
```

```
    # Matrix
    # zero matrix (N*N)
    A = np.zeros([N,N])
    for i in range(N):
        A[i,i] = 1 + 2*theta*lam
        #Substitute 1 + 2*theta*lam
        # to the diagonal components of A
        if i>=1:
        # Substitute -theta*lam
        # to the two sides
        # of the diagonal components
            A[i,i-1] = -theta*lam
            A[i-1,i] = -theta*lam

    # zero matrix (N*N)
    B = np.zeros([N,N])#
    for i in range(N):
        B[i,i] = 1 - 2*(1-theta)*lam
        #Substitute 1 - 2*(1-theta)*lam
        # to the diagonal component of A
        if i >= 1:
        # Substitute (1-theta)*lam
        # to the two sides
        # of the diagonal component
            B[i,i-1] = (1-theta)*lam
            B[i-1,i] = (1-theta)*lam

    # Solve the finite difference scheme
    U = u[1:N+1,:]
    err = 0
    for n in range(nmax-1):
        UU = np.dot(B,U[:,n])
        # B*u
        U[:,n+1] = np.linalg.solve(A,UU)
        # A^{-1}*B*u
        u[1:N+1,n+1]=U[:,n+1]
        errvec = u[:,n+1] - func_sol(x, (n+1)*tau)
        # Calculate the error
        err = max(LA.norm(errvec,np.inf),err)
        # l infinitiy norm of the error vector

    return err
```

We assume $E_h^* := \|\boldsymbol{e}\|_\infty = Ch^p$. Here, h and p are positive constants. Then,

$$\frac{\log E_{2h}^* - \log E_h^*}{\log(2h) - \log h} = \frac{\log \frac{E_{2h}^*}{E_h^*}}{\log \frac{2h}{h}} = \frac{\log \frac{C(2h)^p}{Ch^p}}{\log 2} = \frac{p \log \frac{2h}{h}}{\log 2} = p.$$

```
def heat_theta_error_log(lam,theta,Tmax):
    N0 = 30
    jmax = 5
    hvec = []
    Errvec = []
    nvec = []
    Hvec = []
    for j in range(1,jmax):
        N = N0*j
        err  = heat_theta_error(N,lam,theta,Tma
        h = 1/N
        hvec.append(h)
        Hvec.append(h**2)
        Errvec.append(err)
        nvec.append(N)

    plt.plot(hvec,Errvec,color='blue')
    plt.plot(hvec,Hvec,color='red')
    plt.yscale('log')
    plt.xscale('log')

    plt.grid(which='major',color='black',linest
    plt.grid(which='minor',color='black',linest
    plt.show()
    return Errvec, Hvec
```

```
heat_theta_error_log(100,1,1)
```

2.8 Fisher-KPP equation

Consider the following equation (Fisher-KPP)

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} = u(1-u) \quad 0 < x < 1, \quad t > 0,$$
$$u(0,t) = u(1,t) = 0, \quad 0 < x < 1,$$
$$u(x,0) = f(x), \quad 0 \le x \le 1.$$

The explict scheme to Fishee-KPP is following:

$$\frac{u_j^{n+1} - u_j^n}{\tau} = k\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2} + u_j^n(1-u_j^n)$$

Hence, we have

$$u_j^{n+1} = \left(1 - 2k\frac{\tau}{h^2}\right)u_j^n + k\frac{\tau}{h^2}(u_{j+1}^n + u_{j-1}^n) + \tau u_j^n(1-u_j^n).$$

We consider the following initial condition:

$$f_2(x) = x\sin^2(3\pi x)$$

```
def f2(x):
    y=x*(np.sin(3*np.pi*x))**2
    return y
```

```
## Fisher KPP u_t-ku_xx = 10*u(1-u)
def fk(N,lam,theta,Tmax,k):

    h = 1/(N+1)
    tau = lam*h**2
    dt = 0.01
    nmax = int(Tmax/tau)
    skip = int(dt/tau)

    x = np.linspace(0.0,1.0,N+2)
    u = np.zeros([N+2,nmax])

    u[:,0] = f2(x)
    plt.plot(x, u[:,0], color='red')

    for i in range(nmax):
        u[0,i] = 0
        u[N+1,i] = 0

    A = np.zeros([N,N])
    for i in range(N):
        A[i,i] = 1 + 2*theta*lam
        if i>=1:
            A[i,i-1] = -theta*lam
            A[i-1,i] = -theta*lam

    B = np.zeros([N,N])
    for i in range(N):
        B[i,i] = 1 - 2*(k-theta)*lam
        if i >= 1:
            B[i,i-1] = (k-theta)*lam
            B[i-1,i] = (k-theta)*lam

    U = u[1:N+1,:]
    for n in range(nmax-1):
        UU = np.dot(B,U[:,n])+tau*10*U[:,n]*(1-U[:,n])
        U[:,n+1] = np.linalg.solve(A,UU)
```

```
        u[1:N+1,n+1]=U[:,n+1]

        # for the visialization
        if n%skip == 50:
          plt.plot(x,u[:,n+1],color='blue',linewidth=0.1)

    #print(u[int(len(x)/2),nmax-1])

    xx= list(range(N+2))
    yy = list(range(nmax))

    X,Y = np.meshgrid(xx,yy)
    Z = functz(X,Y,u)
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.plot_wireframe(X,Y,Z)
    ax.set_xlabel('x')
    ax.set_ylabel('t')
```

```
fk(100,0.5,0,2,1)
```

```
fk(100,0.5,0,2,0.01)
```

3 Wave equation

3.1 Explicit scheme

We consider the following wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \quad (0 < x < 1), \quad t > 0, \quad \text{(Wave equation)}$$

$$u(0,t) = u(1,t) = 0, \quad (t > 0)$$

(Dirichlet boundary condition)

$$u(x,0) = f(x), \quad \partial_t u(x,0) = g(x) \quad (0 \le x \le 1)$$

(Initial condition)

We consider the following finite difference scheme to the wave equation:

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\tau^2} = c^2 \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{h^2}, \quad (3.1)$$

$$u_j^0 = f(x_j), \quad \frac{u_j^1 - u_j^{-1}}{2\tau} = g(x_j). \quad (3.2)$$

It is rewritten as

$$u_j^{n+1} = 2\left(1 - c^2\frac{\tau^2}{h^2}\right)u_j^n + c^2\frac{\tau^2}{h^2}(u_{j+1}^n + u_{j-1}^n) - u_j^{n-1}$$

$$u_j^0 = f(x_j), \quad u_j^{-1} = u_j^1 - 2\tau g(x_j),$$

Since

$$\frac{u_j^1 - 2u_j^0 + u_j^{-1}}{\tau^2} = \frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{h^2},$$

we have

$$\frac{u_j^1 - 2u_j^0 + u_j^1 - 2\tau g(x_j)}{\tau^2} = \frac{u_{j+1}^0 - 2u_j^0 + u_{j-1}^0}{h^2}$$

$$u_j^1 = \left(1 - \frac{\tau^2}{h^2}\right)u_j^0 + \frac{\tau^2}{2h^2}\left(u_{j+1}^0 + u_{j-1}^0\right) + \tau g(x_j)$$

Let $c = 1$ and $\lambda = \frac{\tau^2}{h^2}$. Moreover, we assume $g(x) = 0$.

```
# space interval and space mesh
N=50
h=1.0/(N+1)
x=np.linspace(0.0, 1.0, N+2)
```

```
# time increment
lam=1
tau=np.sqrt(lam*h*h)
Tmax=1.0
nmax=int(Tmax/tau)
```

```
# for visualization
dt = 10
skip = int(dt/tau)
skip
```

```
# container of the solution
u = np.zeros([N+2,nmax])
```

```
#initial data
u[:,0]=f2(x)
plt.plot(x,u[:,0],color='red')
for i in range(1,N):
  u[i,1]=(1 - lam)*u[i,0]+lam*(u[i-1,0]+u[i+1,0])/2
```

```
## boundary condition
for n in range(nmax-1):
    u[0,n] = 0 #x[0] で u =0
    u[N+1,n] = 0 #x[m] で u = 0
```

```
for n in range(1,nmax-1):
    for i in range(1,N):
        u[i,n+1]=2*(1 - lam)*u[i,n]+lam*(u[i-1,n]+u[i+1,n]) - u[i,n-1]
        #if n%skip == 10:
        plt.plot(x,u[:,n+1],color='blue',linewidth=0.1)
```

```
#for visualization
xx=list(range(N))
yy=list(range(nmax))

X, Y = np.meshgrid(xx,yy)

def functz(u):
    z=u[X,Y]
    return z

Z = functz(u)
fig = plt.figure()
ax = Axes3D(fig)
ax.plot_wireframe(X,Y,Z,linewidth=0.2,color='blue')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('T')
```

3.2 Exercise

Plot the solution in the cases $\lambda < 1$, $\lambda = 1$, $\lambda > 1$.

4 References

[1] M. Katsurada, http://nalab.mind.meiji.ac.jp/~mk/program/

[2] N. Saito, Introduction to Numerical Analysis for Partial Differential Equations, 2019 Summer Semester.