Project 2: 差分格式稳定性及数值效应比较实验

纪经纬 5141209098

June 5, 2018

1 问题的提出

用不同的差分格式求解对流方程

$$\begin{cases} u_t + a u_x = & 0 \\ u(0,x) = & f(x) = \begin{cases} 1, & x \le 0 \\ 0, & x > 0 \end{cases} \end{cases} \tag{1.1}$$

参数选取：$a = 1, 2, 4, h = 0.1, \tau = 0.08, \lambda = \frac{\tau}{h} = 0.8$，得到$t = 4$时的数值结果。

利用特征线易知方程的解为$u(x,t) = f(x - at)$。那么$t = 4$时，

$$u(x,4) = \begin{cases} 0, & x > 4a \\ 1, & x \le 4a \end{cases} \tag{1.2}$$

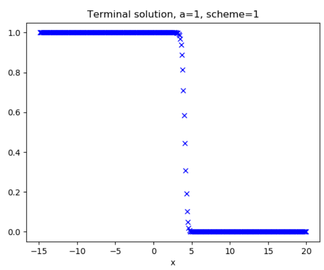
2 不同差分格式的数值计算与分析

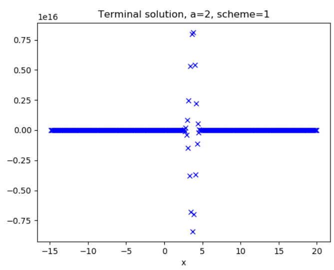
本节简要列出差分格式，并附代码。考虑到本次实验用到的都是显式格式，使用循环计算即可，所以使用Python进行数值求解。在此列出$t = 4$时解的情况，其他图示列在附录中。

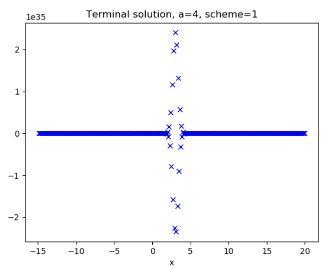
2.1 迎风格式Upwind scheme

$$u_j^{n+1} = u_j^n - a\lambda(u_j^n - u_{j-1}^n) \tag{2.1}$$

该格式的稳定性条件为$a\lambda \le 1$。



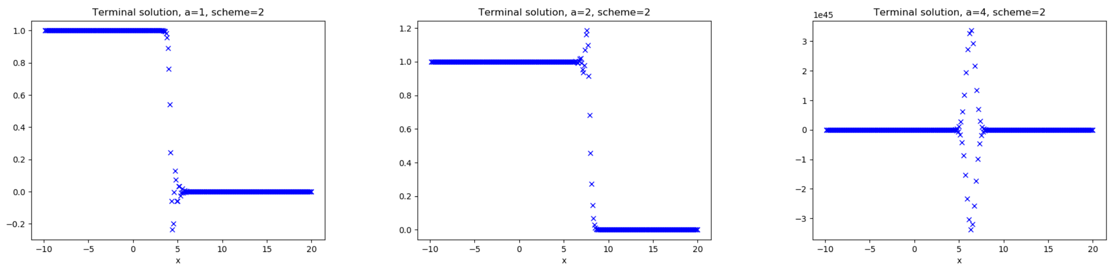




2.2 Beam-Warming格式

$$u_j^{n+1} = u_j^n - a\lambda(u_j^n - u_{j-1}^n) - \frac{a\lambda}{2}(1 - a\lambda)(u_j^n - 2u_{j-1}^n + u_{j-2}^n) \tag{2.2}$$

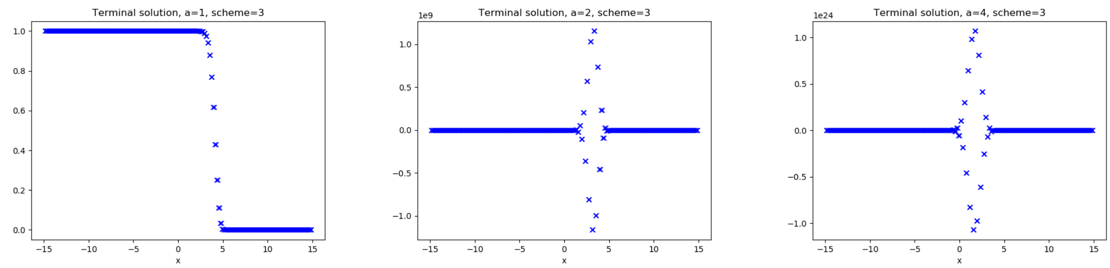
该格式的稳定性条件为$a\lambda \le 2$。



2.3 Lax-Friedrichs格式

$$u_j^{n+1} = \frac{1}{2}(u_{j+1}^n + u_{j-1}^n) - \frac{1}{2}a\lambda(u_{j+1}^n - u_{j-1}^n) \tag{2.3}$$

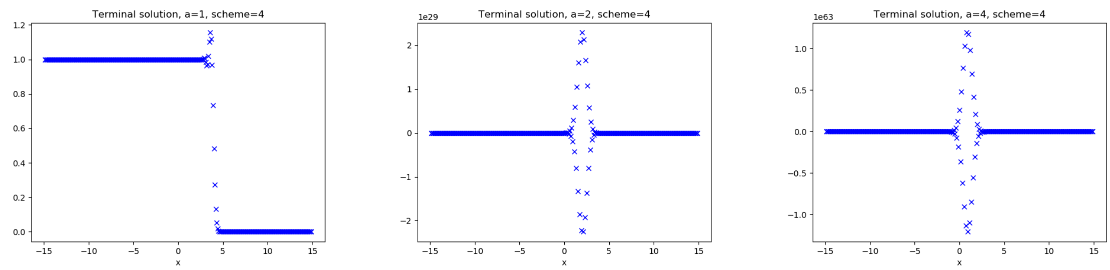
该格式的稳定性条件为$a\lambda \leq 1$。



2.4 Lax-Wendroff格式

$$u_j^{n+1} = u_j^n - \frac{1}{2}a\lambda(u_{j+1}^n - u_{j-1}^n) + \frac{1}{2}a^2\lambda^2(u_{j+1}^n - 2u_j^n + u_{j-1}^n) \tag{2.4}$$

该格式的稳定性条件为$a\lambda \leq 1$。



2.5 数值分析

从误差图中我们可以看到，稳定性的情况符合预期。Beam-Warming格式稳定性条件相比另外几个格式更好一些。

在解稳定的情况下，向间断点靠拢，Beam-Warming和Lax-Wendroff格式的误差会上下震荡，另外两个则只会将解“抹掉”；在间断点处，相对误差从两边向其递增。

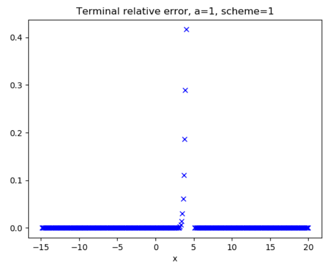
在解不稳定的情况下，受间断点影响，附近的一段段区间误差会急剧上升。并且，从相对误差的数量级大小来看，这4个格式的不稳定程度的升序：Lax-Friedrichs, Upwind, Beam-Warming, Lax-Wendroff。

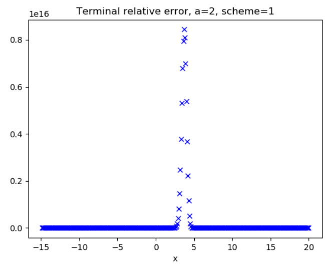
从全局解的图示中可以清晰地看到，不稳定的情况下初始的微小误差会进行爆炸性累积；“稳定性”在数学上的严格定义在这里有非常直观的体现。

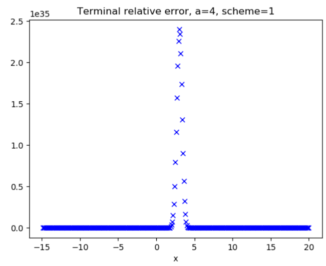
3 附录

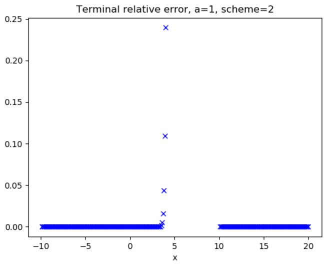
3.1 更多图示

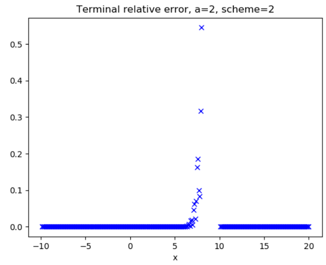
在这里，附上更多相关图示用以说明。包括终值相对误差、全局解、全局相对误差。这里使用的衡量逐点误差的指标相对误差是$|\frac{u_j^n - u(x_j,t_n)}{u(x_j,t_n)}|$。

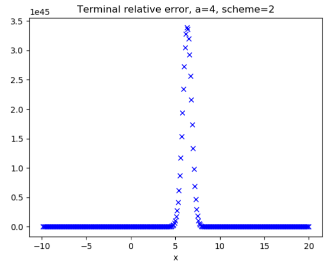


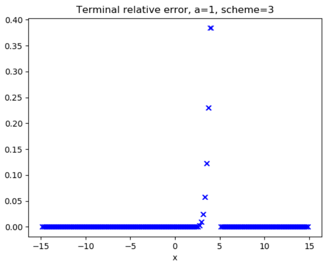


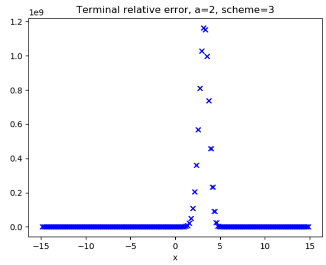


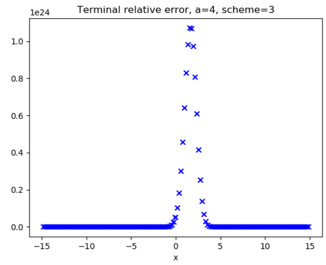


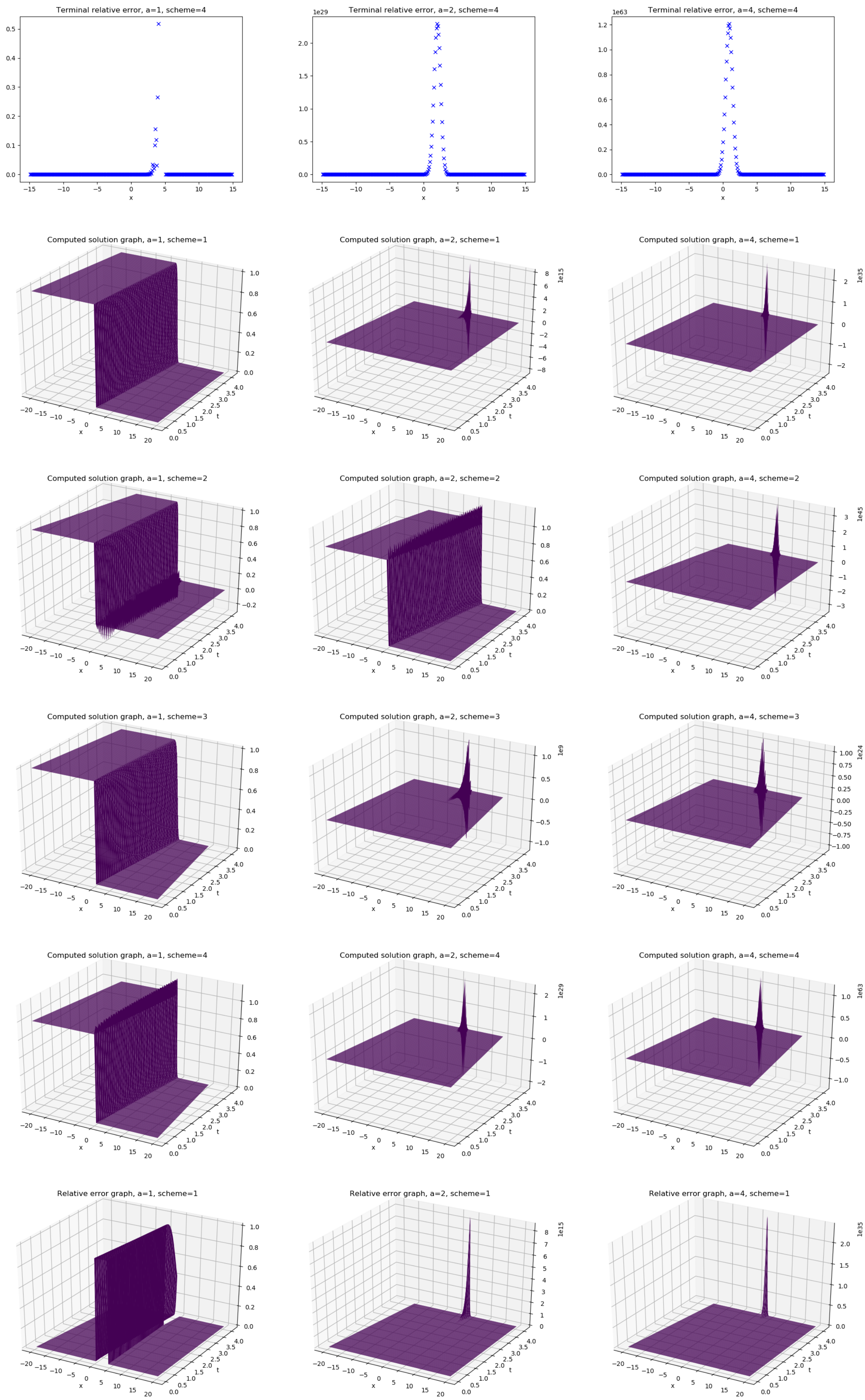


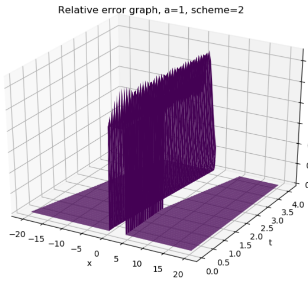


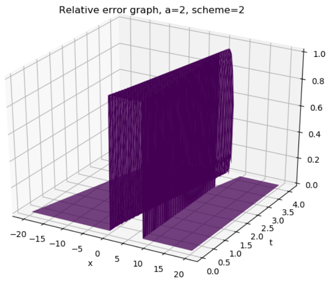


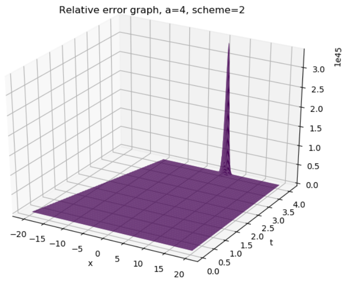


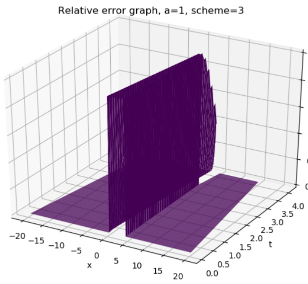


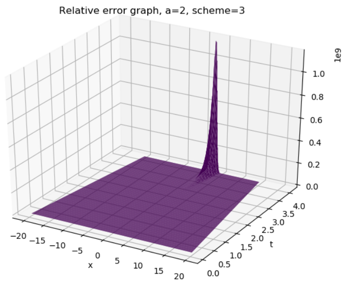

Terminal relative error, a=1, scheme=4
Terminal relative error, a=2, scheme=4
Terminal relative error, a=4, scheme=4
Computed solution graph, a=1, scheme=1
Computed solution graph, a=2, scheme=1
Computed solution graph, a=4, scheme=1
Computed solution graph, a=1, scheme=2
Computed solution graph, a=2, scheme=2
Computed solution graph, a=4, scheme=2
Computed solution graph, a=1, scheme=3
Computed solution graph, a=2, scheme=3
Computed solution graph, a=4, scheme=3
Computed solution graph, a=1, scheme=4
Computed solution graph, a=2, scheme=4
Computed solution graph, a=4, scheme=4
Relative error graph, a=1, scheme=1
Relative error graph, a=2, scheme=1
Relative error graph, a=4, scheme=1

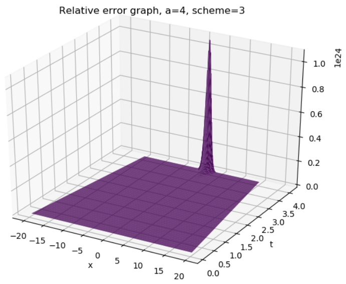

Relative error graph, a=1, scheme=2
x
t

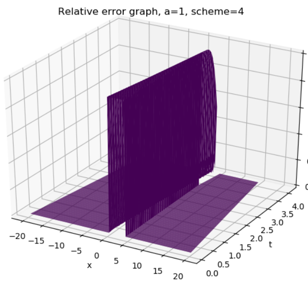

Relative error graph, a=2, scheme=2
x
t

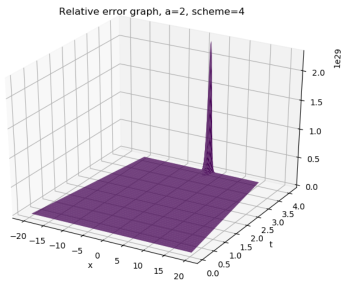

Relative error graph, a=4, scheme=2
1e45
x
t

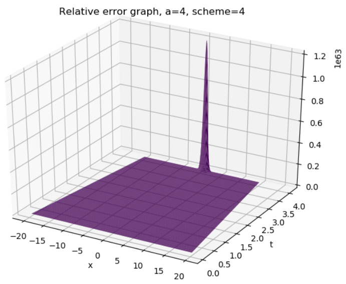

Relative error graph, a=1, scheme=3
x
t


Relative error graph, a=2, scheme=3
1e9
x
t


Relative error graph, a=4, scheme=3
1e24
x
t


Relative error graph, a=1, scheme=4
x
t


Relative error graph, a=2, scheme=4
1e29
x
t


Relative error graph, a=4, scheme=4
1e63
x
t

3.2 源代码

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

np.set_printoptions(threshold=np.inf)

class ConvectionEquation:
    '''
    x range: [ -X, X ]
    t range: [0, T]
    '''

    def __init__(self, a, h, tau, T, X, scheme):
        self.a = a
        self.h = h
        self.tau = tau
        self.lambda_ = tau / h
        self.T = T
        self.X = X
        self.scheme = scheme

        self.N = int(self.T / self.tau)
        self.J = int(self.X / self.h)
        # t_n: t_0=0, t_1 = tau, .... t_N = tau * self.N
        # x_j: x_{-J} = -J * h, ..., x_0 = 0, .... , x_J = J * h

        self.u_matrix = np.zeros([self.N+1, 1+2*self.J])

    def solve(self):
        self.set_initial_condition()
        self.solve_engine(scheme)
        self.error_analysis()
        self.plot_u_matrix()
        self.plot_terminal()


    def initial_condition_function(self, j):
        # x_j = j * h
        if j < 0 or j == 0:
            return 1
        else:
            return 0

    def set_initial_condition(self):
        n = 0
        for j in range(-self.J, self.J+1):
            self.set_u_matrix_elements(j, n, self.initial_condition_function(j))

    def set_u_matrix_elements(self, j, n, u):
        self.u_matrix[self.N-n][j+self.J] = u

    def get_u_matrix_elements(self, j, n):
        return self.u_matrix[self.N-n][j+self.J]

    def set_matrix_elements(self, M, j, n, element):
        M[self.N-n][j+self.J] = element

    def get_matrix_elements(self, M, j, n):
        return M[self.N-n][j+self.J]
```

```
    def print_u_matrix(self):
        print(self.u_matrix)
        # for x in range(self.N+1):
        # line_string = ""
        # for y in range(1+2*self.J):
        # line_string += str(self.u_matrix[x][y])
        # print(line_string)

    def plot_terminal(self):
        terminal_series = self.u_matrix[0]
        terminal_error_series = self.error_matrix[0]

        fig = plt.figure()
        plt.plot(np.arange(-self.X, self.X+self.h, self.h), terminal_series)
        plt.title("Terminal solution, a=%s, scheme=%s"%(str(self.a), str(self.scheme)))
        plt.xlabel("x")
        # plt.show()
        plt.savefig("figures\\Terminalsolutiona=%sscheme=%s"%(str(self.a), str(self.scheme)))

        fig = plt.figure()
        plt.plot(np.arange(-self.X, self.X + self.h, self.h), terminal_error_series)
        plt.title("Terminal relative error, a=%s, scheme=%s" % (str(self.a), str(self.scheme)))
        plt.xlabel("x")
        # plt.show()
        plt.savefig("figures\\Terminalrelativeerrora=%sscheme=%s" % (str(self.a), str(self.scheme)))

    def real_solution(self, j):
        if j * self.h > 4 * self.a:
            return 0
        else:
            return 1

    def error_analysis(self):
        self.error_matrix = np.zeros([self.N + 1, 1 + 2 * self.J])
        for n in range(0, self.N+1, 1):
            for j in range(- self.J, self.J+1):
                u = self.get_matrix_elements(M=self.u_matrix, j=j, n=n)
                real_u = self.real_solution(j=j)
                if u-real_u == 0:
                    error = 0
                else:
                    error = abs((u-real_u)/real_u)
                self.set_matrix_elements(M=self.error_matrix, j=j, n=n, element=error)

        X = np.arange(-self.X, self.X + self.h, self.h)
        T = np.arange(self.T + self.tau, 0, -self.tau)
        X, T = np.meshgrid(X, T)

        fig = plt.figure()
        ax = Axes3D(fig)
        ax.plot_surface(X, T, self.error_matrix, rstride=1, cstride=1, cmap=cm.viridis)
        ax.set_title("Relative error graph, a=%s, scheme=%s" % (str(self.a), str(self.scheme)))
        ax.set_xlabel("x")
        ax.set_ylabel("t")
        # plt.show()
        plt.savefig("figures\\Relativeerrorgrapha=%sscheme=%s" % (str(self.a), str(self.scheme)))

    def plot_u_matrix(self):
        X = np.arange(-self.X, self.X+self.h, self.h)
        T = np.arange(self.T+self.tau, 0, -self.tau)
        X, T = np.meshgrid(X, T)
```

```
        fig = plt.figure()
        ax = Axes3D(fig)
        ax.plot_surface(X, T, self.u_matrix, rstride=1, cstride=1, cmap=cm.viridis)
        ax.set_title("Computed solution graph, a=%s, scheme=%s"%(str(self.a), str(self.scheme)))
        ax.set_xlabel("x")
        ax.set_ylabel("t")
        # plt.show()
        plt.savefig("figures\\Computedsolutiongrapha=%sscheme=%s"%(str(self.a), str(self.scheme)))

    def solve_engine(self, scheme):
        '''
        scheme:
        1: u_{j}^{n+1} = u_j^n - a \lambda (u_j^n - u_{j-1}^n)
        2: u_{j}^{n+1} = u_j^n - a \lambda (u_j^n - u_{j-1}^n) - \frac{a \lambda }{2} (1- a \lambda )(
            u_j^n - 2 u_{j-1}^n + u^{n}_{j-2} )
        3: u_{j}^{n+1 } = \frac{1}{2} ( u_{j+1}^n + u_{j-1}^n ) - \frac{1}{2} a \lambda (u_{j+1}^n - u_
            {j-1}^n )
        4: u_{j}^{n+1 } = u_j^n - \frac{1}{2} a \lambda (u_{j+1}^n - u_{j-1}^n ) + \frac{1}{2} a^2 \
            lambda^2 (u_{j+1}^n - 2 u_j^n + u_{j-1}^n )

        '''
        for n in range(1, self.N+1, 1):
            for j in range(- self.J, self.J+1):
                # compute u(n,j)
                if scheme == 1:
                    if j-1 <= -self.J or np.isnan(self.get_u_matrix_elements(n=n-1, j=j)) or np.isnan(
                        self.get_u_matrix_elements(n=n-1, j=j-1)):
                        self.set_u_matrix_elements(n=n, j=j, u=np.nan)
                    else:
                        # u_j^n: self.get_u_matrix_elements(n=n-1, j=j)
                        u = self.get_u_matrix_elements(n=n-1, j=j) - self.a * self.lambda_ * ( self.
                            get_u_matrix_elements(n=n-1, j=j) - self.get_u_matrix_elements(n=n-1, j=j-1)
                             )
                        self.set_u_matrix_elements(n=n, j=j, u=u)

                if scheme == 2:
                    if j-2 <= -self.J or np.isnan(self.get_u_matrix_elements(n=n-1, j=j)) or np.isnan(
                        self.get_u_matrix_elements(n=n-1, j=j-1)) or np.isnan(self.get_u_matrix_elements
                        (n=n-1, j=j-2)):
                        self.set_u_matrix_elements(n=n, j=j, u=np.nan)
                    else:
                        u = self.get_u_matrix_elements(n=n-1, j=j) - self.a * self.lambda_ * ( self.
                            get_u_matrix_elements(n=n-1, j=j) - self.get_u_matrix_elements(n=n-1, j=j-1)
                             ) \
                        - self.a*self.lambda_ / 2 * (1-self.a*self.lambda_) * (self.
                            get_u_matrix_elements(n=n-1, j=j) - 2 *self.get_u_matrix_elements(n=n-1, j=j
                            -1) + self.get_u_matrix_elements(n=n-1, j=j-2))
                        self.set_u_matrix_elements(n=n, j=j, u=u)

                if scheme == 3:
                    if j-1 <= -self.J or j+1 >= self.J or np.isnan(self.get_u_matrix_elements(n=n-1, j=j
                        -1)) or np.isnan(self.get_u_matrix_elements(n=n-1, j=j+1)):
                        self.set_u_matrix_elements(n=n, j=j, u=np.nan)
                    else:
                        u = 0.5 * (self.get_u_matrix_elements(n=n-1, j=j+1) + self.get_u_matrix_elements
                            (n=n-1, j=j-1) ) - 0.5*self.a*self.lambda_*(self.get_u_matrix_elements(n=n
                            -1, j=j+1) - self.get_u_matrix_elements(n=n-1, j=j-1) )
                        self.set_u_matrix_elements(n=n, j=j, u=u)

                if scheme == 4:
                    if j-1 <= -self.J or j+1 >= self.J or np.isnan(self.get_u_matrix_elements(n=n-1, j=j
                        -1)) or np.isnan(self.get_u_matrix_elements(n=n-1, j=j+1)) or np.isnan(self.
                        get_u_matrix_elements(n=n-1, j=j)):
```

```python
                        self.set_u_matrix_elements(n=n, j=j, u=np.nan)
                    else:
                        u = self.get_u_matrix_elements(n=n-1, j=j) - 0.5*self.a*self.lambda_*(self.
                            get_u_matrix_elements(n=n-1, j=j+1) - self.get_u_matrix_elements(n=n-1, j=j
                            -1)) \
                        + 0.5 * self.a*self.a*self.lambda_*self.lambda_*( self.get_u_matrix_elements(n=n
                            -1, j=j+1) - 2 *self.get_u_matrix_elements(n=n-1, j=j) + self.
                            get_u_matrix_elements(n=n-1, j=j-1) )
                        self.set_u_matrix_elements(n=n, j=j, u=u)

if __name__ == "__main__":
    a_list = [1, 2, 4]
    h = 0.1
    tau = 0.08
    T = 4.0
    X = 20.0
    for scheme in [1, 2, 3, 4]:
        for a in a_list:
            pdeEngine = ConvectionEquation(a=a, h=h, tau=tau, T=T, X=X, scheme=scheme)
            pdeEngine.solve()
```