# ADL4P: Exercise 6 Automatic Differentiation

In [49]:
```
%%capture
import numpy as np
from google.colab import drive
import matplotlib.pyplot as plt
!pip install nbconvert;
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc;
!apt-get install -y texlive-xetex texlive-fonts-recommended texlive-generic-recommended
drive.mount('/content/drive')
```

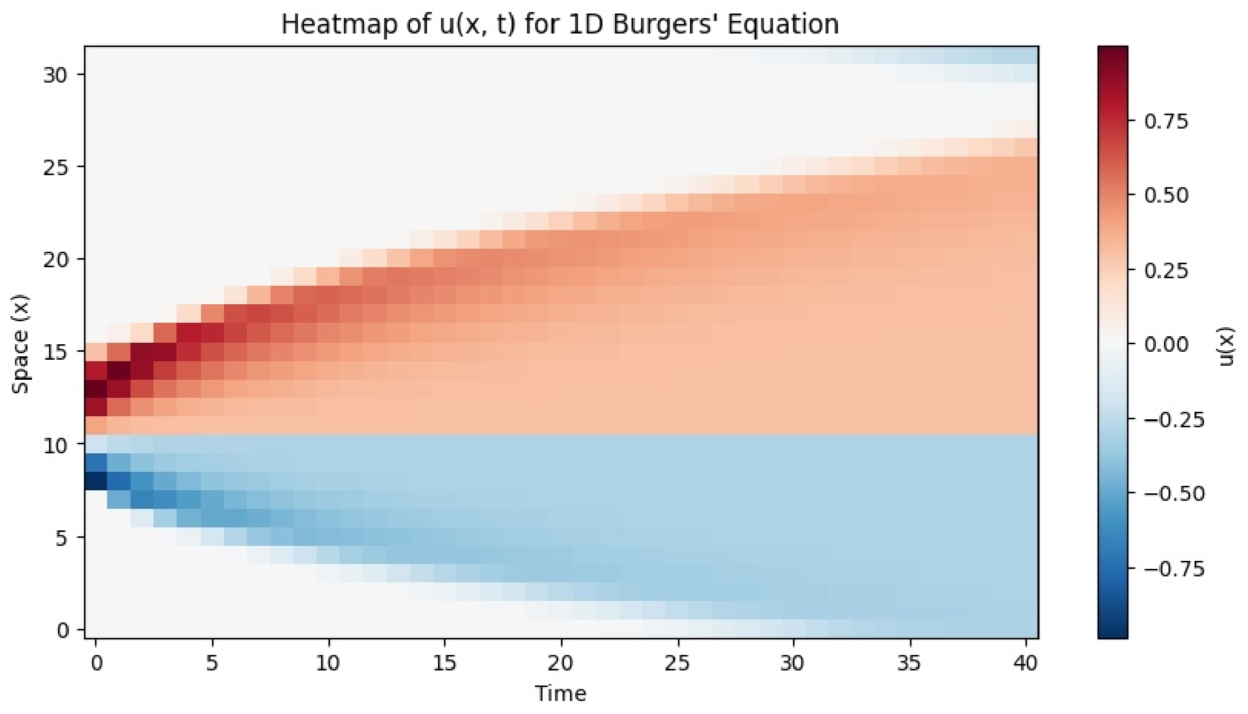
## (1) Numerical solver for Burger’s equation

In [50]:
```
# Domain and initial conditions
Lx = 2*np.pi
Nx = 32
x = np.linspace(0, Lx, Nx)
dx = Lx/(Nx-1)
dt = Lx/(Nx-1)
steps = 40
u0 = np.array([np.sin(3*x[i]) if x[i] > np.pi/2 and x[i] < np.pi else 0 for i in range(Nx)])
```

In [51]:
```
def burger_advection(u, dt, dx):

    u_new = np.zeros_like(u)

    for i in range(Nx):

        # Periodic boundary indices
        ip1 = (i + 1) % Nx
        im1 = (i - 1) % Nx

        # Local average
        u_bar = 0.5 * (u[ip1] + u[im1])

        # Compute spatial derivative based on upwinding
        if u_bar < 0:
            dudx = ((u[ip1]**2 - u[i]**2) / dx)
        else:
            dudx = ((u[i]**2 - u[im1]**2) / dx)

        # Explicit Euler update
        u_new[i] = u[i] - 0.5 * dt * dudx

    return u_new
```

In [52]:
```
# Store data and advec until step 15
data = u0
u=u0
for i in range(steps):
  u = burger_advection(u, dt, dx)
  data = np.vstack((data,u))
```

In [53]:
```
# Plot the results on a heatmap
plt.figure(figsize=(10, 5))
plt.imshow(data.T, aspect='auto', origin='lower', cmap='RdBu_r')
plt.colorbar(label='u(x)')
plt.xlabel('Time')
plt.ylabel('Space (x)')
plt.title("Heatmap of u(x, t) for 1D Burgers' Equation")
plt.show()
```



## (2) Backpropagation

(a): The analitycal solutions of the three partial derivatives are:

- $\frac{\partial u_i^{n+1}}{\partial u_i^n} = 1 - \frac{\Delta t}{2} \cdot \frac{-2u_i^n}{\Delta x} = 1 - u_i^n \frac{\Delta t}{\Delta x} \cdot sgn(\bar{u}_i)$

- $\frac{\partial u_i^{n+1}}{\partial u_{i-1}^n} = \begin{cases} 0 & \text{if } \bar{u}_i < 0 \\ -\frac{\Delta t}{2} \cdot \frac{-2u_{i-1}^n}{\Delta x} = u_{i-1}^n \frac{\Delta t}{\Delta x} & \text{otherwise} \end{cases}$

- $\frac{\partial u_i^{n+1}}{\partial u_{i+1}^n} = \begin{cases} -\frac{\Delta t}{2} \cdot \frac{-2u_{i+1}^n}{\Delta x} = u_{i+1}^n \frac{\Delta t}{\Delta x} & \text{if } \bar{u}_i < 0 \\ 0 \text{ otherwise} \end{cases}$

the Loss function is $||\cdot||_2$ so:

- $L(x^{(n)}) = \sum_{i=1}^N x_i^{(n)2} \Rightarrow \frac{\partial L(x^{(n)})}{\partial x^{(n)}} = \left[2x_1^{(n)}, \ldots, 2x_N^{(n)}\right]$
- $\frac{\partial L(u^{n+1})}{\partial u^n} = \sum_{i,j=1}^N \frac{\partial L(u^{n+1})}{\partial u_i^{n+1}} \frac{\partial u_i^{n+1}}{\partial u_j^n} = \left[2u_1^{(n+1)}, \ldots, 2u_N^{(n+1)}\right] \cdot$

$$\begin{bmatrix} 2\left(1 - u_1^n \frac{\Delta t}{\Delta x} \cdot \text{sgn}(\bar{u}_1)\right) & c_1^{\text{sup}} \cdot u_2^n \frac{\Delta t}{\Delta x} & 0 & \cdots & 0 \\ c_1^{\text{sub}} \cdot u_1^n \frac{\Delta t}{\Delta x} & 2\left(1 - u_2^n \frac{\Delta t}{\Delta x} \cdot \text{sgn}(\bar{u}_2)\right) & c_2^{\text{sup}} \cdot u_3^n \frac{\Delta t}{\Delta x} & \ddots & \vdots \\ 0 & c_2^{\text{sub}} \cdot u_2^n \frac{\Delta t}{\Delta x} & 2\left(1 - u_3^n \frac{\Delta t}{\Delta x} \cdot \text{sgn}(\bar{u}_3)\right) & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{N-1}^{\text{sup}} \cdot u_N^n \frac{\Delta t}{\Delta x} \\ 0 & \cdots & 0 & c_{N-1}^{\text{sub}} \cdot u_{N-1}^n \frac{\Delta t}{\Delta x} & 2\left(1 - u_N^n \frac{\Delta t}{\Delta x} \cdot \text{sgn}(\bar{u}_N)\right) \end{bmatrix}$$

where $c_i^{\text{sup}} = \begin{cases} 1 & \text{if } \bar{u}_i < 0 \\ 0 & \text{otherwise} \end{cases}$ $\quad c_i^{\text{sub}} = \begin{cases} 1 & \text{if } \bar{u}_i > 0 \\ 0 & \text{otherwise} \end{cases}$

```
In [54]: def backprop(u, dt, dx):
    Nx = len(u)  # Define Nx from u
    u_forward  = np.roll(u, -1)  # u[i+1], wraps around
    u_backward = np.roll(u, 1)   # u[i-1], wraps around
    ubar = 0.5 * (u_backward + u_forward)

    # Initialize Jacobian
    Jac = np.zeros((Nx, Nx))

    # Main diagonal
    dui_dui = 2 * (1 - u * dt / dx * np.sign(ubar))
    Jac += np.diag(dui_dui)

    # Subdiagonal (i-1)
    mask_sub = ubar[1:Nx] >= 0
    dui_duim1 = np.zeros(Nx - 1)
    dui_duim1[mask_sub] = (dt / dx) * u[0:Nx - 1][mask_sub]
    Jac += np.diag(dui_duim1, k=-1)

    # Superdiagonal (i+1)
    mask_sup = ubar[0:Nx - 1] < 0
    dui_duip1 = np.zeros(Nx - 1)
    dui_duip1[mask_sup] = (dt / dx) * u[1:Nx][mask_sup]
    Jac += np.diag(dui_duip1, k=1)

    return Jac
```

In [55]:
```
def loss(du):
  grad = 2*du
  return grad, np.linalg.norm(du)**2
```

## (3) Reconstructing inital conditions

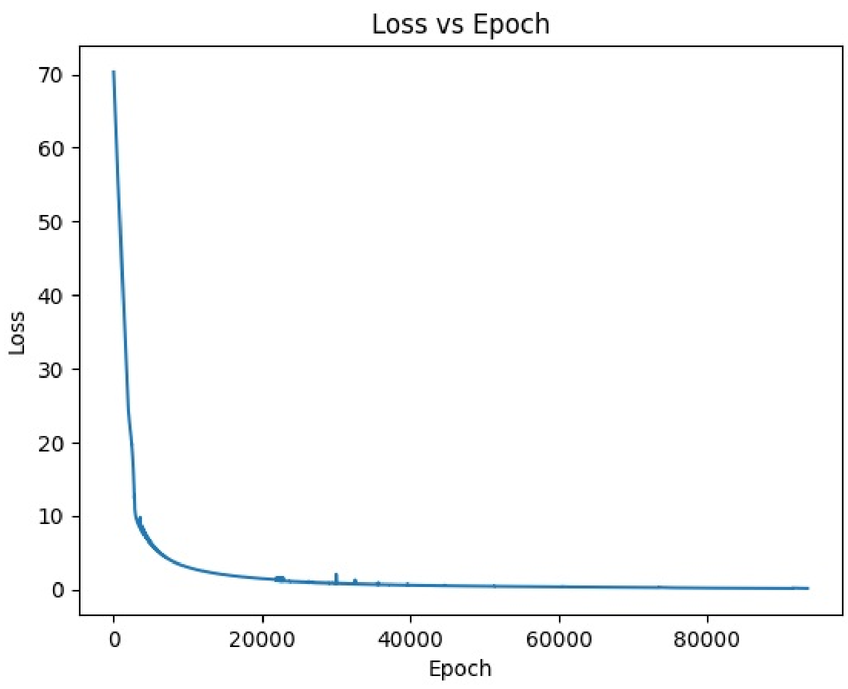
In [56]:
```
# Initial conditions
Lx = np.pi*2
Nx = 32
x = np.linspace(0, Lx, Nx)
dx = Lx/(Nx-1)
dt = .1*dx
uf = np.load('/content/drive/MyDrive/Fisica/ADL4P/burgers_target_state.npy')
steps = 15
# Hyperparameters
initial_lr = 0.00000001
epochs = 200000
u0 = np.array([np.sin(3*x[i]) if x[i] > np.pi/2 and x[i] < np.pi else 0 for i in range(Nx)])
```

In [57]:
```
def optimizer(u0, u, epochs, steps, initial_lr, patience=100,patience_es=2000, decay_factor=0.5):
    loss_list = []
    u0 = u0.astype(float)  # ensure float type
    # Define parameters to controll gradient descend
    lr = initial_lr
    prev_loss = float('inf')
    best_loss = float('inf')
    best_u0 = u0.copy()
    bad_epochs = 0
    # store gradients
    J_total = np.eye(Nx)
    Jac = np.empty((Nx,Nx))

    for k in range(epochs):
        u = u0.copy()
        for _ in range(steps):
            # forward pass

            u = burger_advection(u, dt, dx)
            # save grad for chain rule
            Jac = backprop(u, dt, dx)  # ∂u^{n+1}/∂u^n
            J_total = J_total * Jac.T # Hadamard Multiplication

        # Compute loss and gradient
        du = uf - u

        grad, loss_val = loss(du)
        grad = grad @ J_total.T
        J_total = np.eye(Nx)

        # Compute loss and gradient
        u0 = u0 + lr*grad

        # Save best model
        if loss_val <= best_loss:
            best_loss = loss_val
            best_u0 = u0.copy()
            bad_epochs_es = 0
        else:
            bad_epochs_es += 1

        # Learning rate decay on plateau
        if loss_val >= prev_loss:
            bad_epochs += 1
            if bad_epochs >= patience:
                lr *= decay_factor
                bad_epochs = 0
                u0 = best_u0.copy()
                print(f"⬇  Learning rate reduced to {lr:.2e} at epoch {k}")
        else:
            bad_epochs = 0

        prev_loss = loss_val

        if k % 6000 == 0:
            print(f"Epoch {k}, Loss: {loss_val:.6f}, LR: {lr:.2e}, Best Loss: {best_loss:.6}")

        # Early stopping
        if bad_epochs_es > patience_es:
            print("  Early stopping triggered.")
            print(f"Epoch {k}, Loss: {loss_val:.6f}, LR: {lr:.2e}, Best Loss: {best_loss:.6}")
            break

        loss_list.append(loss_val)

    return best_u0, loss_list
```

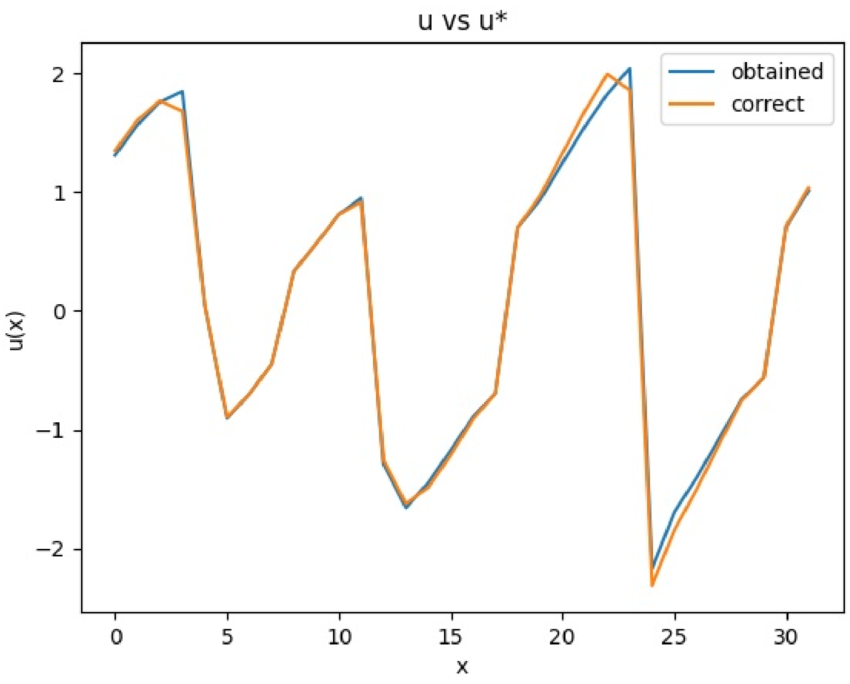
In [58]:
```
# Optimize the problem
x = np.linspace(0, 2 * np.pi, Nx)  # 32 points evenly spaced between 0 and 2π
u0 = np.sin(x) # initial guess
u = u0.copy()
loss_list = []
u0, loss_list = optimizer(u0,u,epochs,steps,initial_lr)
print(u0)
```

```
Epoch 0, Loss: 70.350171, LR: 1.00e-08, Best Loss: 70.3502
Epoch 6000, Loss: 5.157166, LR: 1.00e-08, Best Loss: 5.14146
Epoch 12000, Loss: 2.478939, LR: 1.00e-08, Best Loss: 2.47883
Epoch 18000, Loss: 1.601262, LR: 1.00e-08, Best Loss: 1.60107
Epoch 24000, Loss: 1.044140, LR: 1.00e-08, Best Loss: 1.04414
Epoch 30000, Loss: 0.793544, LR: 1.00e-08, Best Loss: 0.793544
Epoch 36000, Loss: 0.635812, LR: 1.00e-08, Best Loss: 0.635812
Epoch 42000, Loss: 0.529361, LR: 1.00e-08, Best Loss: 0.529361
Epoch 48000, Loss: 0.452507, LR: 1.00e-08, Best Loss: 0.452507
Epoch 54000, Loss: 0.392672, LR: 1.00e-08, Best Loss: 0.392672
Epoch 60000, Loss: 0.347692, LR: 1.00e-08, Best Loss: 0.343456
Epoch 66000, Loss: 0.293441, LR: 1.00e-08, Best Loss: 0.293441
Epoch 72000, Loss: 0.244757, LR: 1.00e-08, Best Loss: 0.244757
Epoch 78000, Loss: 0.196990, LR: 1.00e-08, Best Loss: 0.19699
Epoch 84000, Loss: 0.156732, LR: 1.00e-08, Best Loss: 0.156732
Epoch 90000, Loss: 0.132142, LR: 1.00e-08, Best Loss: 0.132142
  Early stopping triggered.
Epoch 93465, Loss: 0.133106, LR: 1.00e-08, Best Loss: 0.128429
[ 1.99687246  2.0448687   1.94770417  1.25952537 -1.3438376  -1.14410783
 -1.09318853 -0.58137135  0.20173466  0.89192317  1.16924632  0.95250666
  0.81060261 -1.75201043 -2.02057499 -1.98220812 -1.56302789 -0.65836997
  0.74188903  1.76143802  2.25643438  2.37244823  2.52570691  0.76349115
 -1.56223005 -2.40478596 -2.31049598 -2.07342532 -1.36252351 -0.07535751
  1.19777538  1.84956672]
```

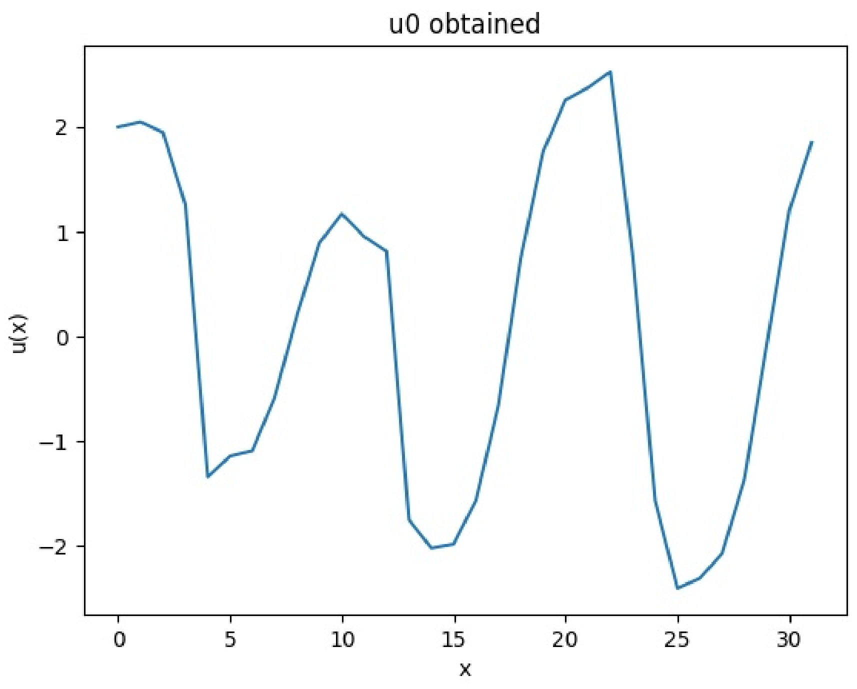
In [59]:
```
# Plot the loss values
plt.plot(loss_list)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs Epoch')
plt.show()
```



In [60]:
```
# compare correct u_final vs obtained u_f
u = u0.copy()
for i in range(steps):
  u = burger_advection(u, dt, dx)

plt.plot(u, label='obtained')
plt.plot(uf, label='correct')
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('u vs u*')
plt.legend()
plt.show()
```



In [61]:
```
# Print u0 obtained
plt.plot(u0)
plt.xlabel('x')
plt.ylabel('u(x)')
plt.title('u0 obtained')
plt.show()
```



In [62]:
```
%%capture
!jupyter nbconvert --to html /content/drive/MyDrive/Fisica/ADL4P/Exercise_6.ipynb --output /content/drive/MyDri
```