PGE 382 - Numerical Methods in Petroleum and Geosystems Engineering

Renato Poli - rep2656

CP6 - Mar, 30th

a) Case 1

In [1]:
```
from math import pi, sin, cos
import numpy as np
np.set_printoptions(threshold=10000, linewidth=10000)

from numpy import exp, linspace, vectorize
import matplotlib.pyplot as plt

plt.style.use('paper.mplstyle')

# index i
Ni = 25
dx = 1/Ni
Ni += 1

# index N, t
Nt=10
dt = 1/Nt
Nt = Nt + 1

# Position and time arrays
X = np.linspace(0,1,Ni)
T = np.linspace(0,1,Nt)

# Indexed by xi
Uni = np.zeros( [ Nt, Ni ] )
Vni = np.zeros( [ Nt, Ni ] )

Uidx = np.arange(0,2*Ni,2)
Vidx = np.arange(1,2*Ni,2)

# IC
Uni[0, :] = ( X + 1 ) / 2
Vni[0, :] = pi
# BC
Uni[:, 0] = 1/2
Vni[:, 0] = pi

# Free/known DOFs
Uf = np.arange( 1, Ni )
Vf = np.arange( 1, Ni )
K_ix = np.ix_(Uf,Uf)
F_ix = np.ix_(Uf)

K_bcix = np.ix_(Uf,[0])
F_bcix = np.ix_([0])

#
# Solver
#
U = Uni[0,:]
V = Vni[0,:]
for n in np.arange(1,Nt) :
    print(f"Solving timestep {n} ...")

    # Eqn 1 : u_t = v^2 u_xx + 2 v v_x u_x - u v + u^2 + 10
    K = np.zeros([ Ni, Ni ] )
    F = np.zeros(Ni)
    for i in np.arange(1,Ni-1) :
        # - u_t
        K[ i, i ] += 1/dt
        F[ i ]    += 1/dt * U[i]
        # v^2 u_xx
        K[ i, i-1 ] += ( -V[i]**2/dx/dx )
        K[ i, i ]   += ( -V[i]**2/dx/dx ) * ( -2 )
        K[ i, i+1 ] += ( -V[i]**2/dx/dx )
        # 2 v v_x u_x
        K[ i, i-1 ] += ( -2*V[i]*( V[i]-V[i-1] ) ) / dx / dx * ( -1 )
        K[ i, i ]   += ( -2*V[i]*( V[i]-V[i-1] ) ) / dx / dx
        # - uv
        K[ i, i ]   += ( V[i] )
        # u^2
        K[ i, i ]   += ( - U[i] )
        # 10
        F[i] = ( 10 )

    # Derivative BC at Ni (note:arrays are indexed starting from ZERO. Hence, Ni is the note past the last)
    l = Ni-1  # index of the last element
    K[l,l]    += 1/dx
    K[l,l-2]  += (-1/dx)
    F[l-1]       += 1/2 - sin(U[l]*V[l])
    # Apply BC @ X=0
    F[F_ix] -= K[K_bcix] @ U[F_bcix]
    # Solve and save the results ( the index remove the known at x=0 )
    Uni[n,F_ix] = np.linalg.solve( K[K_ix], F[F_ix] )
    U = Uni[n,:]

    # Eqn 2 : v_t = u^2 v_xx + 2 u u_x v_x + u_xx + uv - v^2
    K = np.zeros([ Ni, Ni ] )
    F = np.zeros(Ni)
    for i in np.arange(1,Ni-1) :
        # - v_t
        K[ i, i ] += 1/dt
        F[ i ]    += 1/dt * V[i]
        # u^2 v_xx
        K[ i, i-1 ] += ( -U[i]**2/dx/dx )
        K[ i, i ]   += ( -U[i]**2/dx/dx ) * ( -2 )
        K[ i, i+1 ] += ( -U[i]**2/dx/dx )
        # 2 u u_x v_x
        K[ i, i-1 ] += ( -2*U[i]*( U[i]-U[i-1] ) ) / dx / dx * ( -1 )
        K[ i, i ]   += ( -2*U[i]*( U[i]-U[i-1] ) ) / dx / dx
        # u_xx
        F[ i ]   += ( U[i-1] - 2 * U[i] + U[i+1] ) /dx/dx
        # u v
        K[ i, i ]   += ( - U[i] )
        # -v^2
        K[ i, i ]   += ( V[i] )

    # Derivative BC at Ni (note:arrays are indexed starting from ZERO. Hence, Ni is the note past the last)
    l = Ni-1  # index of the last element
    K[l,l]    += 1/dx
    K[l,l-1]  += (-1/dx)
    F[l]       += 1 + cos(U[l]*V[l])
    # Apply BC @ X=0
    F[F_ix] -= K[K_bcix] @ V[F_bcix]

    # Solve and save the results ( the index remove the known at x=0 )
    Vni[n,F_ix] = np.linalg.solve( K[K_ix], F[F_ix] )
    V = Vni[n,:]
```

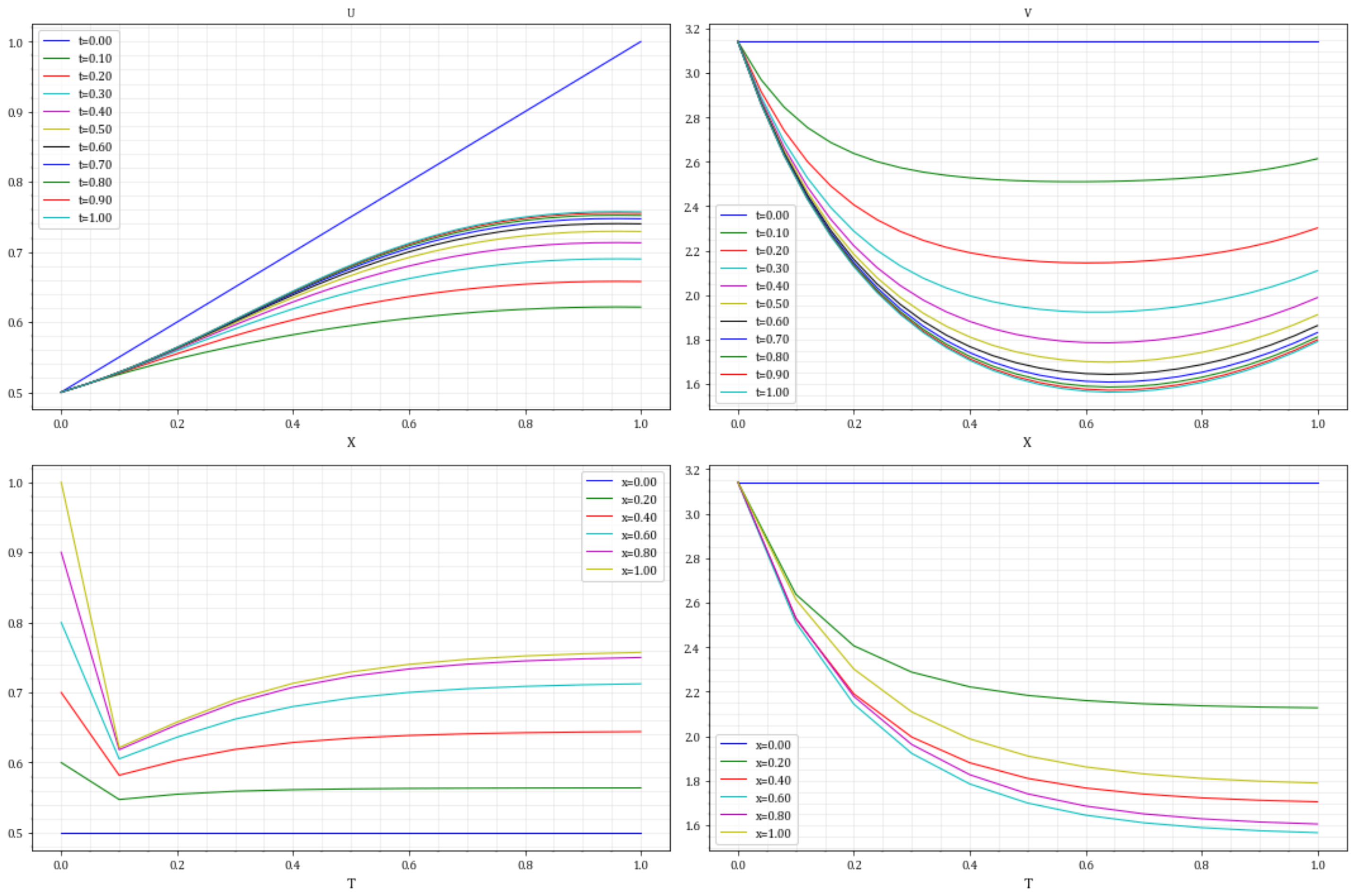
```
Solving timestep 1 ...
Solving timestep 2 ...
Solving timestep 3 ...
Solving timestep 4 ...
Solving timestep 5 ...
Solving timestep 6 ...
Solving timestep 7 ...
Solving timestep 8 ...
Solving timestep 9 ...
Solving timestep 10 ...
```

In [2]:

```
# Cleanup arrays - remove dummy boundaries
import matplotlib.pyplot as plt
fig, [[ax1,ax2],[ax3,ax4]] = plt.subplots( 2, 2, figsize=(15,10) )
for n in np.arange(0,Nt) :
    ax1.plot( X, Uni[n,:], label=f"t={T[n]:.2f}" )
    ax2.plot( X, Vni[n,:], label=f"t={T[n]:.2f}" )

for i in np.arange(0,Ni,5) :
    ax3.plot( T, Uni[:,i], label=f"x={X[i]:.2f}" )
    ax4.plot( T, Vni[:,i], label=f"x={X[i]:.2f}" )
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()

ax1.set_title("U")
ax2.set_title("V")
ax1.set_xlabel("X")
ax2.set_xlabel("X")
ax3.set_xlabel("T")
ax4.set_xlabel("T")
fig.tight_layout()
```



CASE 2

```
In [1]:
from math import pi, sin, cos
import numpy as np
np.set_printoptions(threshold=10000, linewidth=10000)

from numpy import exp, linspace, vectorize
import matplotlib.pyplot as plt

plt.style.use('paper.mplstyle')

# index i
Ni = 25
dx = 1/Ni
Ni += 1

# index N, t
Nt=10
dt = 1/Nt
Nt = Nt + 1

# Position and time arrays
X = np.linspace(0,1,Ni)
T = np.linspace(0,1,Nt)

# Indexed by xi
Uni = np.zeros( [ Nt, Ni ] )
Vni = np.zeros( [ Nt, Ni ] )

# Indices for the global linear system
F1_idx = np.arange(0,2*Ni,2)
F2_idx = np.arange(1,2*Ni,2)
U_idx = np.arange(0,2*Ni,2)
V_idx = np.arange(1,2*Ni,2)

# K matrix organization - foreach submatrix
K = np.zeros( [ 2*Ni, 2*Ni ] )
F = np.zeros( 2*Ni )
J1u_ix = np.ix_(F1_idx,U_idx)
J2u_ix = np.ix_(F2_idx,U_idx)
J1v_ix = np.ix_(F1_idx,V_idx)
J2v_ix = np.ix_(F2_idx,V_idx)
F1_ix = np.ix_(F1_idx)
F2_ix = np.ix_(F2_idx)
U_ix = np.ix_(U_idx)
V_ix = np.ix_(V_idx)

# Select free dofs (U0 and V0 are known)
Kff_ix = np.ix_( np.arange(2, 2*Ni), np.arange(2, 2*Ni) )
Ff_ix  = np.ix_( np.arange(2, 2*Ni) )
```

```python
# IC
Uni[0, :] = ( X + 1 ) / 2
Vni[0, :] = pi
# BC
Uni[:, 0] = 1/2
Vni[:, 0] = pi

# Free and Prescribed DOFs
Uf = np.arange( 1, Ni )
Vf = np.arange( 1, Ni )
K_ix = np.ix_(Uf,Uf)
F_ix = np.ix_(Uf)

K_bcix = np.ix_(Uf,[0])
F_bcix = np.ix_([0])

# Solver
for n in np.arange(1,Nt) :
    print(f"Solving timestep {n} ...")

    # Solution from the previous TS
    Un = Uni[n-1,:]
    Vn = Vni[n-1,:]
    Uk = Un.copy()
    Vk = Vn.copy()

    # Newton loops
    k = 0
    err = 999
    while(1) :
        # Jacobians and functions
        J1u = np.zeros([ Ni, Ni ] )
        J1v = np.zeros([ Ni, Ni ] )
        J2u = np.zeros([ Ni, Ni ] )
        J2v = np.zeros([ Ni, Ni ] )
        F1 = np.zeros(Ni)
        F2 = np.zeros(Ni)

        # Assembly
        for i in np.arange(1,Ni-1) :
            # Shortcuts
            h2 = dx*dx ; h=dx
            u = Uk[i]; v = Vk[i]
            u0 = Uk[i-1]; u1 = Uk[i+1] ; v0 = Vk[i-1]; v1 = Vk[i+1]
            du = u - u0   ; dv = v-v0 ; d2u = u1-2*u+u0 ; d2v = v1-2*v+v0
            v2 = v**2 ; u2=u**2

            F1[i] += (Un[i]-u) / dt
            F1[i] += v2 / h2 * d2u
            F1[i] += 2*v/h2 * dv * du
            F1[i] += -u*v + u2 + 10

            J1u[i,i]   += -2*v2/h2 + 2*v*dv/h2 - v + 2*u - 1/dt
            J1u[i,i-1] += v2/h2    - 2*v*dv/h2
            J1u[i,i+1] += v2/h2

            J1v[i,i]   += 2*v*d2u/h2 + 4*du*v/h2 - 2*v0/h2*du - u
            J1v[i,i-1] += -2*v/h2*du

            F2[i] += (Vn[i]-v)/dt
            F2[i] += u2/h2 * d2v
            F2[i] += 2*u*du*dv/h2
            F2[i] += d2u/h2
            F2[i] += u*v
            F2[i] += -v2

            J2u[i,i]   += 2*u*d2v/h2   +   4*u*dv/h2  -2*u0*dv/h2 -   2/h2  +  v
            J2u[i,i-1] += -2*u*dv/h2   +   1/h2
            J2u[i,i+1] += 1/h2

            J2v[i,i]   +=  -1/dt - 2*u2/h2   +   2*u*du/h2   +   u   +   -2*v
            J2v[i,i-1] +=  u2/h2   - 2*u*du/h2
            J2v[i,i+1] += u2/h2

        # Boundary conditions
        # u_x + sin(uv) = 1/2   @ x=1
        u = Uk[-1] ; v = Vk[-1] ; uv = u*v ;
        J1u[-1,-1] += 1/dx + v * cos(uv)
        J1u[-1,-2] += -1/dx
        J1v[-1,-1] += u * cos( uv )
        F1[-1] = ( u - Uk[-2] )/dx + sin(uv) - 1/2
        # v_x - cos(uv) = 1     @ x=1
        J2v[-1,-1] += 1/dx + u * sin( uv )
        J2v[-1,-2] += -1/dx
        J2u[-1,-1] += v * sin( uv )
        F2[-1] = ( v - Vk[-2] )/dx - cos(uv) - 1

        # Assemble global linear system
        K = np.zeros( [ 2*Ni, 2*Ni ] )
        F = np.zeros( 2*Ni )
        K[J1u_ix] = J1u
        K[J2u_ix] = J2u
        K[J1v_ix] = J1v
        K[J2v_ix] = J2v
        F[F1_ix] = F1
        F[F2_ix] = F2

        # Only free dofs - as we have forced the initial to the right value, the delta is zero (homogeneous)
        Kff = K[Kff_ix]
        Ff  = F[Ff_ix]

        # Solve for the unknowns, update vectors
        df = np.linalg.solve( Kff, -Ff )
        dUV = np.zeros_like(F);
        dUV[Ff_ix] = df
        Uk += dUV[U_ix]
        Vk += dUV[V_ix]

        err = np.linalg.norm(dUV)
        print(f"   Newton iteration #{k} ... (err={err:.3e})")

        # break conditions
        if k > 50 : break   # max iterations ?
        if err < 1e-15 : break # min error ?

        # Close newton iteration
        k += 1

    # Left the newton loop, update solution
    Uni[n,:] = Uk
    Vni[n,:] = Vk
```

```
Solving timestep 1 ...
   Newton iteration #0 ... (err=2.312e+00)
   Newton iteration #1 ... (err=1.774e-01)
   Newton iteration #2 ... (err=4.174e-02)
   Newton iteration #3 ... (err=1.023e-03)
   Newton iteration #4 ... (err=6.728e-07)
   Newton iteration #5 ... (err=3.933e-13)
   Newton iteration #6 ... (err=6.403e-16)
Solving timestep 2 ...
   Newton iteration #0 ... (err=1.577e+00)
   Newton iteration #1 ... (err=3.525e-01)
   Newton iteration #2 ... (err=1.002e-01)
   Newton iteration #3 ... (err=1.881e-03)
   Newton iteration #4 ... (err=3.850e-06)
   Newton iteration #5 ... (err=6.941e-12)
   Newton iteration #6 ... (err=9.375e-16)
Solving timestep 3 ...
   Newton iteration #0 ... (err=1.068e+00)
   Newton iteration #1 ... (err=6.089e-02)
   Newton iteration #2 ... (err=4.929e-03)
   Newton iteration #3 ... (err=3.500e-05)
   Newton iteration #4 ... (err=1.550e-09)
   Newton iteration #5 ... (err=2.531e-15)
   Newton iteration #6 ... (err=2.001e-15)
   Newton iteration #7 ... (err=8.901e-16)
Solving timestep 4 ...
   Newton iteration #0 ... (err=8.453e-01)
   Newton iteration #1 ... (err=2.664e-01)
   Newton iteration #2 ... (err=1.075e-01)
   Newton iteration #3 ... (err=1.441e-02)
   Newton iteration #4 ... (err=2.047e-04)
   Newton iteration #5 ... (err=3.770e-08)
   Newton iteration #6 ... (err=6.902e-15)
   Newton iteration #7 ... (err=6.371e-15)
   Newton iteration #8 ... (err=3.263e-15)
   Newton iteration #9 ... (err=3.206e-15)
   Newton iteration #10 ... (err=6.703e-15)
   Newton iteration #11 ... (err=3.700e-15)
   Newton iteration #12 ... (err=6.794e-16)
Solving timestep 5 ...
   Newton iteration #0 ... (err=5.418e-01)
   Newton iteration #1 ... (err=5.495e-01)
   Newton iteration #2 ... (err=3.773e-01)
   Newton iteration #3 ... (err=1.879e+00)
   Newton iteration #4 ... (err=1.265e+00)
   Newton iteration #5 ... (err=2.103e+00)
   Newton iteration #6 ... (err=5.319e-01)
   Newton iteration #7 ... (err=5.479e-01)
   Newton iteration #8 ... (err=3.815e-01)
   Newton iteration #9 ... (err=8.842e-01)
   Newton iteration #10 ... (err=4.189e-01)
   Newton iteration #11 ... (err=3.754e-01)
   Newton iteration #12 ... (err=5.410e+00)
   Newton iteration #13 ... (err=4.441e+00)
   Newton iteration #14 ... (err=2.551e+00)
   Newton iteration #15 ... (err=1.449e+00)
   Newton iteration #16 ... (err=6.786e-02)
   Newton iteration #17 ... (err=5.265e-03)
   Newton iteration #18 ... (err=5.309e-06)
   Newton iteration #19 ... (err=2.552e-12)
   Newton iteration #20 ... (err=7.469e-16)
Solving timestep 6 ...
   Newton iteration #0 ... (err=1.420e+00)
   Newton iteration #1 ... (err=3.116e-02)
   Newton iteration #2 ... (err=4.997e-04)
   Newton iteration #3 ... (err=2.096e-08)
   Newton iteration #4 ... (err=9.676e-16)
Solving timestep 7 ...
   Newton iteration #0 ... (err=6.585e-01)
   Newton iteration #1 ... (err=9.692e-03)
   Newton iteration #2 ... (err=6.493e-05)
   Newton iteration #3 ... (err=6.389e-10)
   Newton iteration #4 ... (err=7.206e-16)
Solving timestep 8 ...
   Newton iteration #0 ... (err=3.118e-01)
   Newton iteration #1 ... (err=2.281e-03)
   Newton iteration #2 ... (err=2.914e-06)
   Newton iteration #3 ... (err=1.287e-12)
   Newton iteration #4 ... (err=6.957e-16)
Solving timestep 9 ...
   Newton iteration #0 ... (err=1.479e-01)
   Newton iteration #1 ... (err=4.989e-04)
   Newton iteration #2 ... (err=1.313e-07)
   Newton iteration #3 ... (err=3.085e-15)
   Newton iteration #4 ... (err=8.203e-16)
Solving timestep 10 ...
   Newton iteration #0 ... (err=7.042e-02)
   Newton iteration #1 ... (err=1.101e-04)
   Newton iteration #2 ... (err=6.283e-09)
   Newton iteration #3 ... (err=6.627e-16)
```

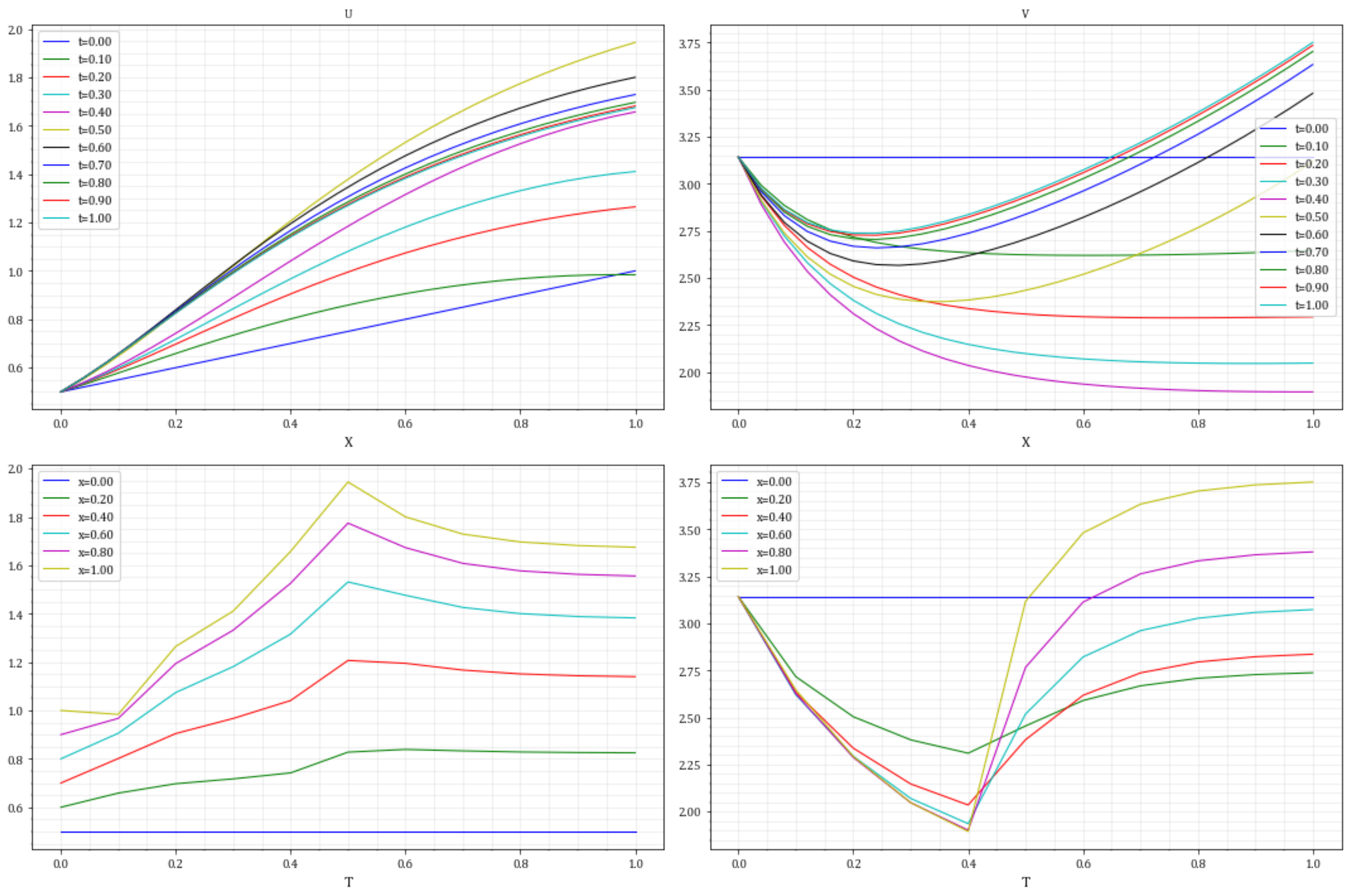
In [3]:

```python
# Cleanup arrays - remove dummy boundaries
import matplotlib.pyplot as plt
fig, [[ax1,ax2],[ax3,ax4]] = plt.subplots( 2, 2, figsize=(15,10) )
```

```python
for n in np.arange(0,Nt) :
    ax1.plot( X, Uni[n,:], label=f"t={T[n]:.2f}" )
    ax2.plot( X, Vni[n,:], label=f"t={T[n]:.2f}" )

for i in np.arange(0,Ni,5) :
    ax3.plot( T, Uni[:,i], label=f"x={X[i]:.2f}" )
    ax4.plot( T, Vni[:,i], label=f"x={X[i]:.2f}" )
ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()

ax1.set_title("U")
ax2.set_title("V")
ax1.set_xlabel("X")
ax2.set_xlabel("X")
ax3.set_xlabel("T")
ax4.set_xlabel("T")
fig.tight_layout()
```



In [4]:
```python
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

plt.imshow(Kff!=0, interpolation='none')
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x1fcc31891f0>
```