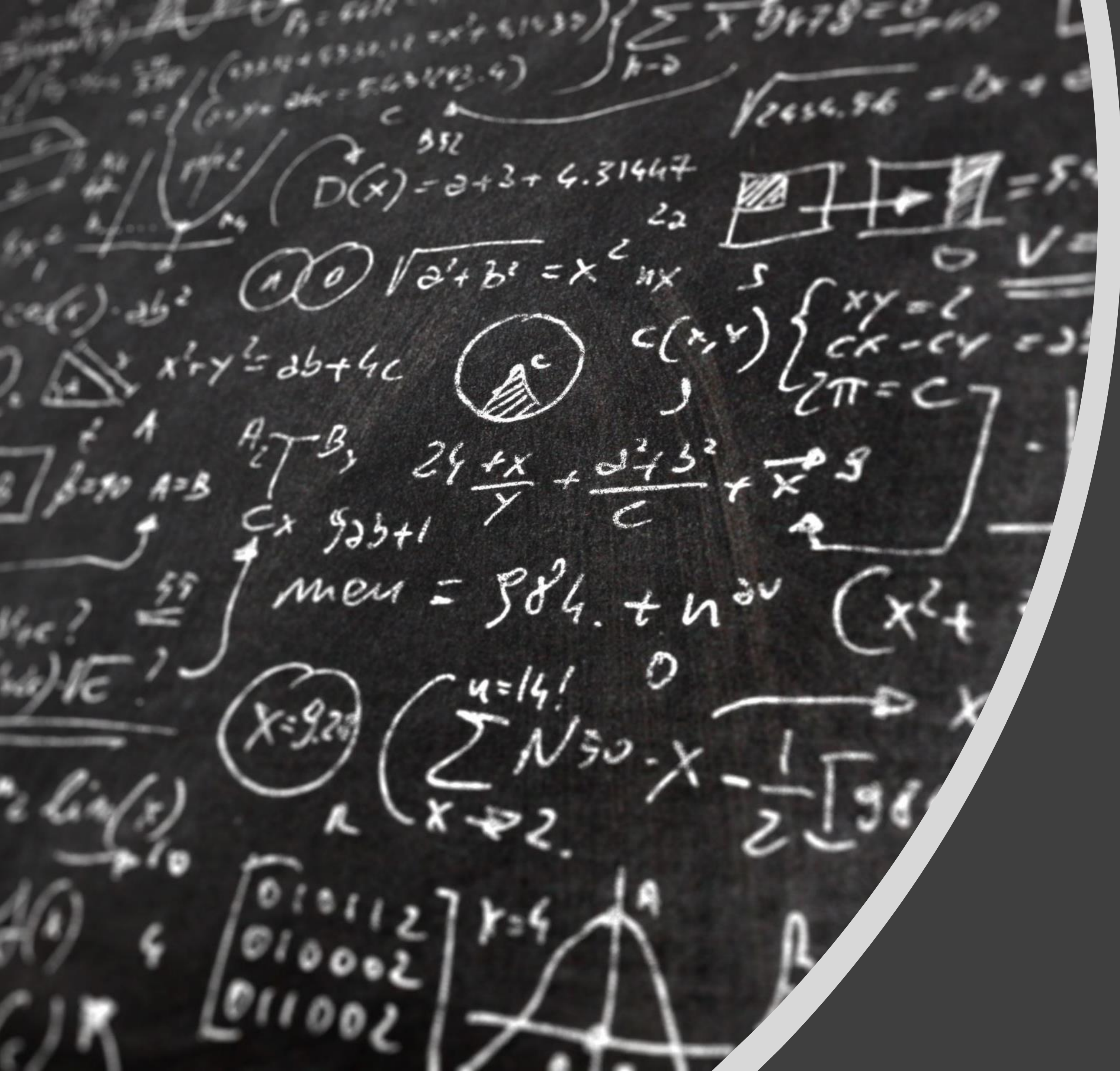
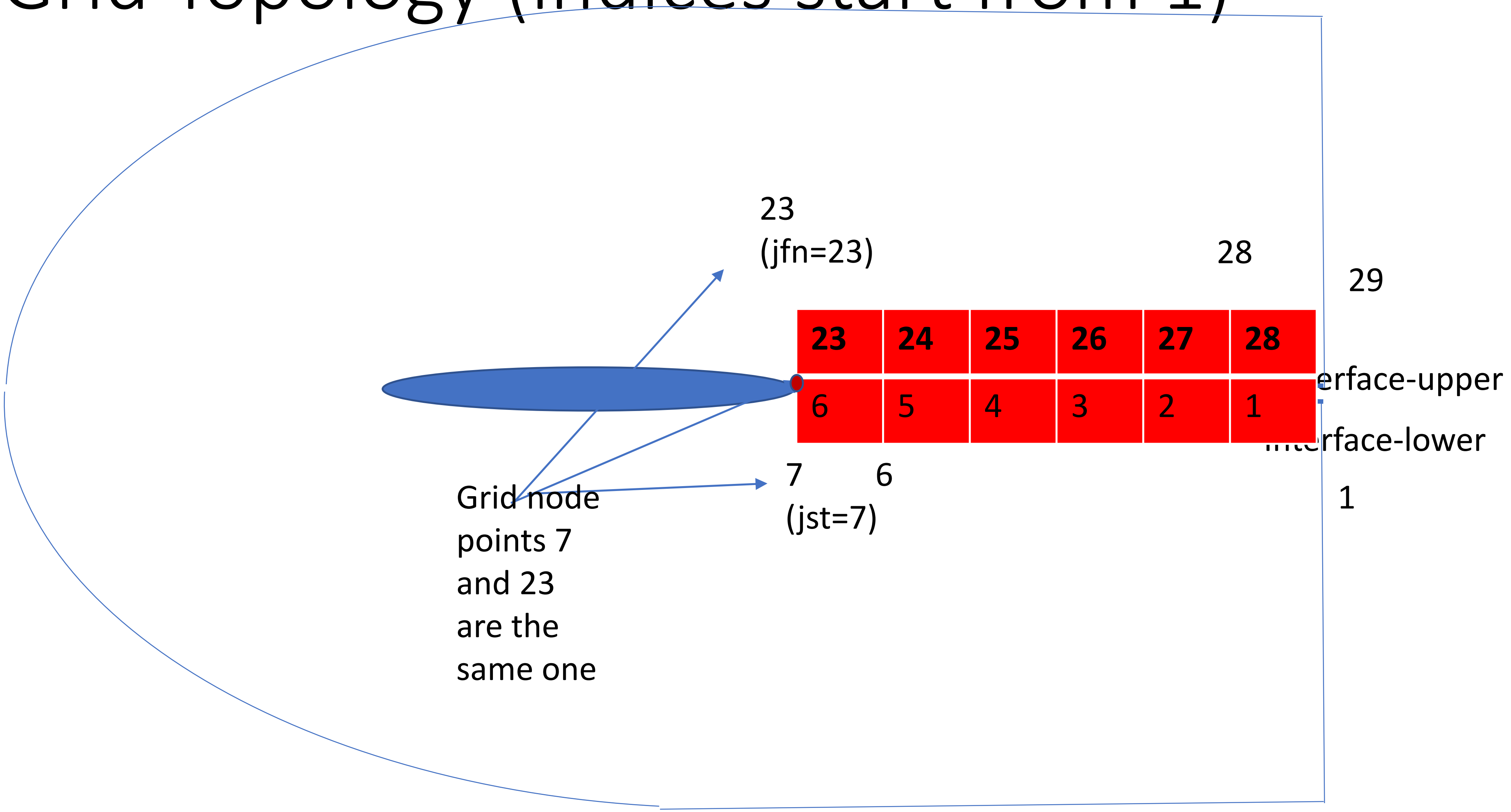


Instruction

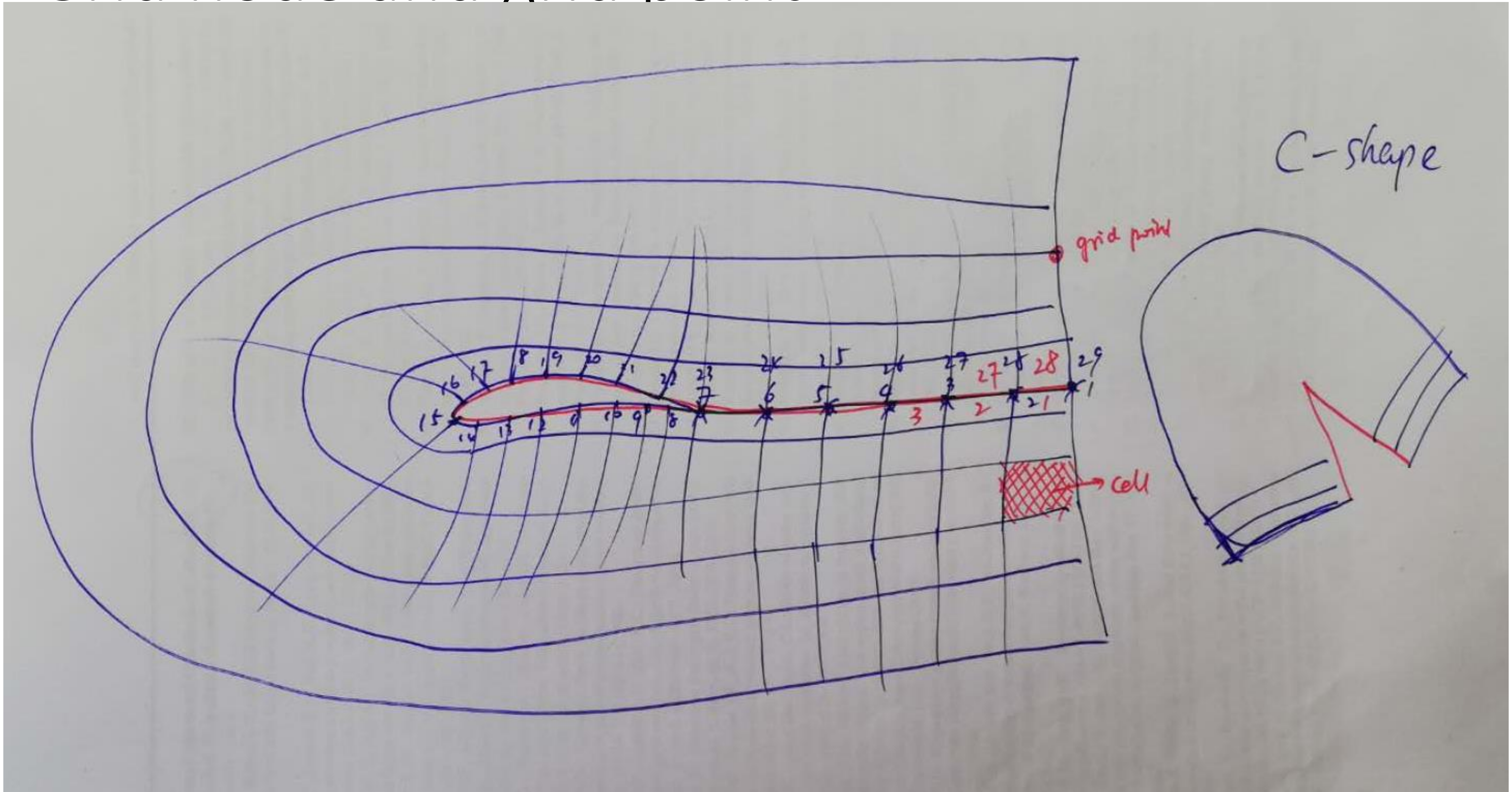
Liwei Chen

2023-02-10

Grid Topology (indices start from 1)



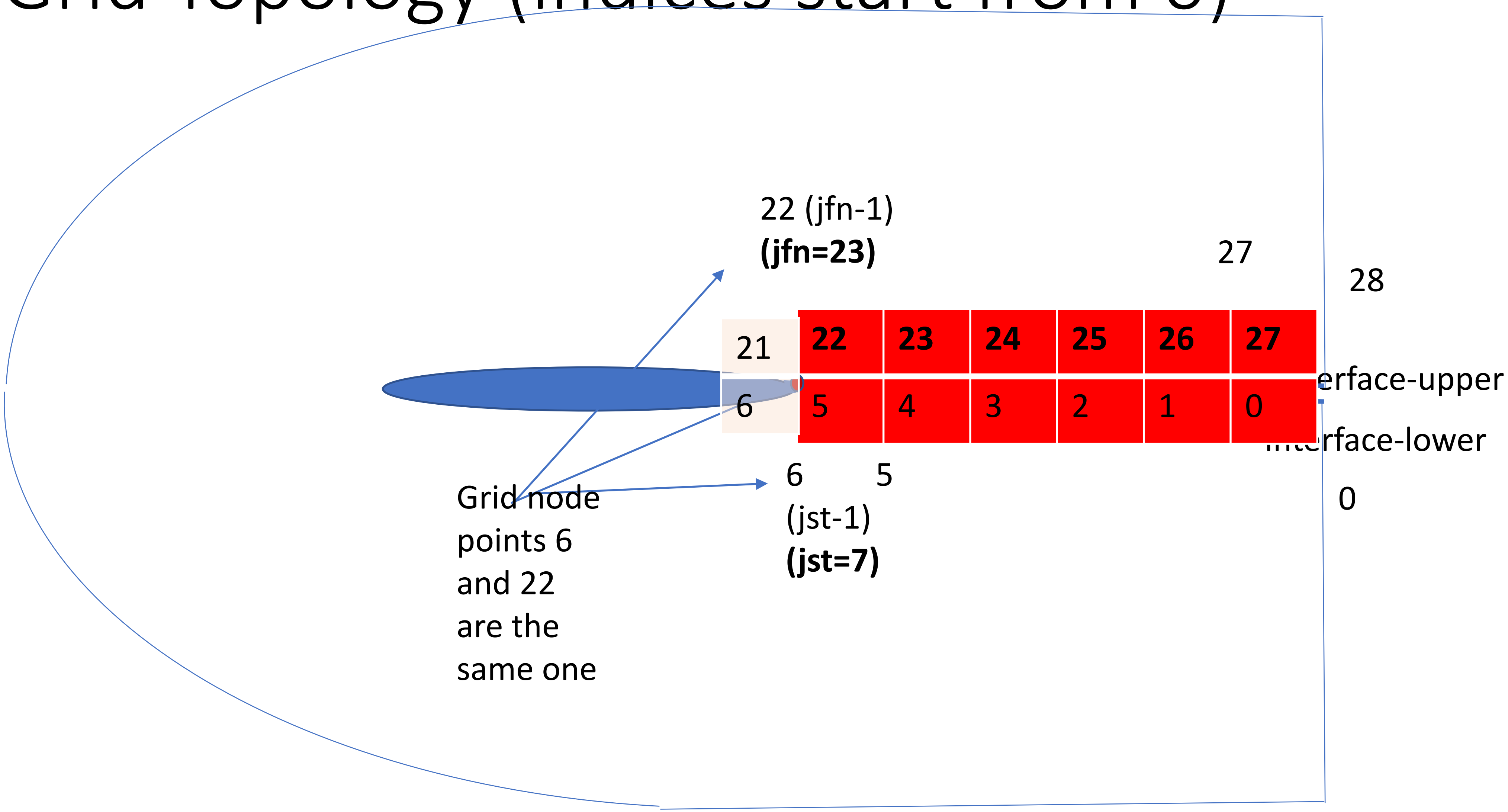
Grid node and grid point



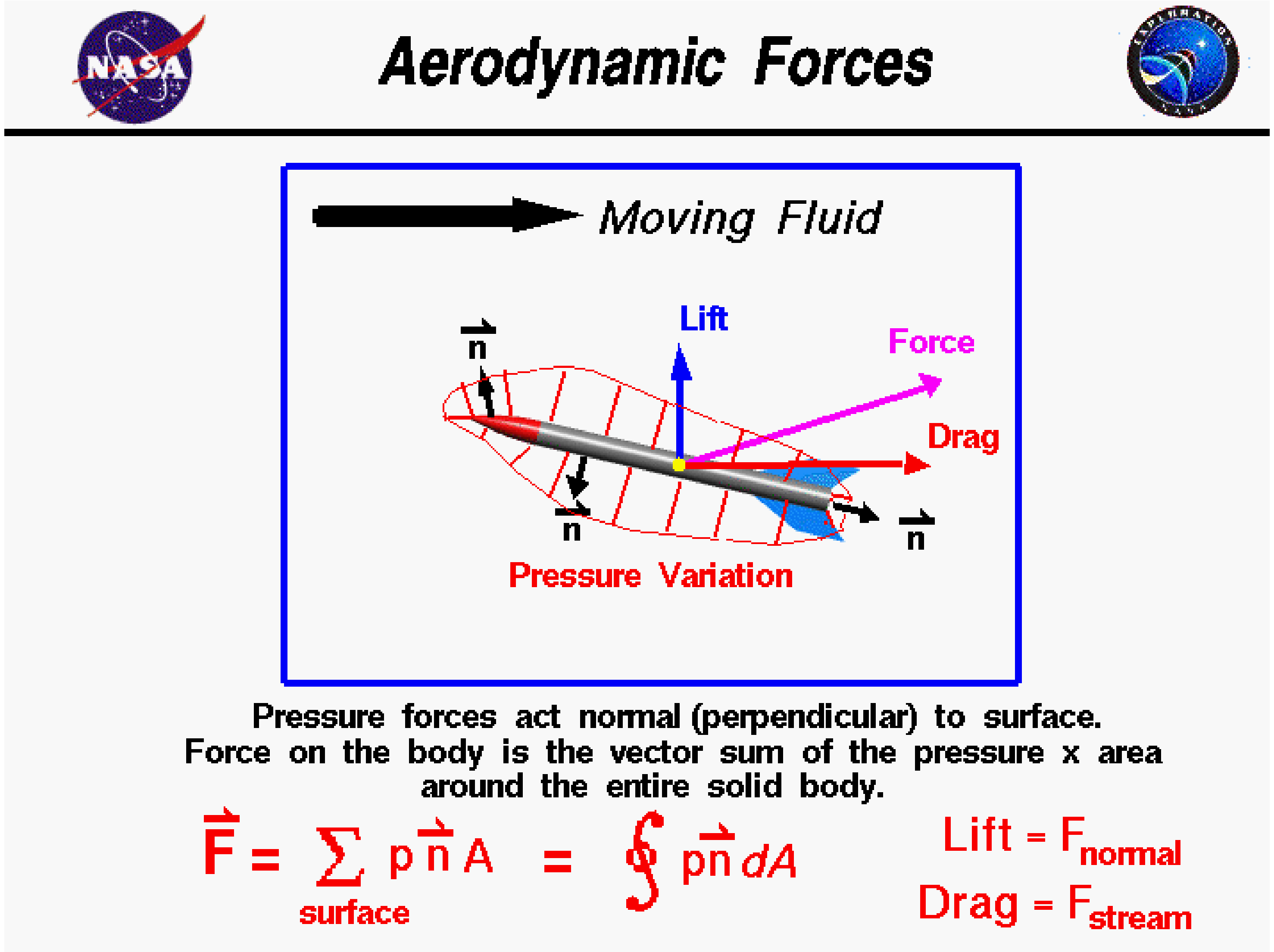
Some notes

- Grid node file is under the folder: train_mesh_point (129 x 129 grid points)
- Grid cell file in the folder: train_mesh (128 x 128 cells)
- All the flowfield variables are defined on grid cell: 128 x 128. Files are under the folder "train"
- At the moment, I only provide you three airfoils

Grid Topology (indices start from 0)



How to calculate Force-X and Force-Y



Calculate "Cl, Cd" w/ Differentiable Functions

Liwei Chen

2023-02-22

Differentiable Metrics

- Implemented and tested the calculation of Cl and Cd using differentiable functions.
- Here we provide some example codes for the caluclation of metrics using differentiable operators (in pytorch).
- We are currently developing it for complex cases, such as „differentiable solver + NN“, learned simulator for unsteady aerodynamics with mesh deformations (we will publish it in the near future).

Example Codes

tfi_torch.py

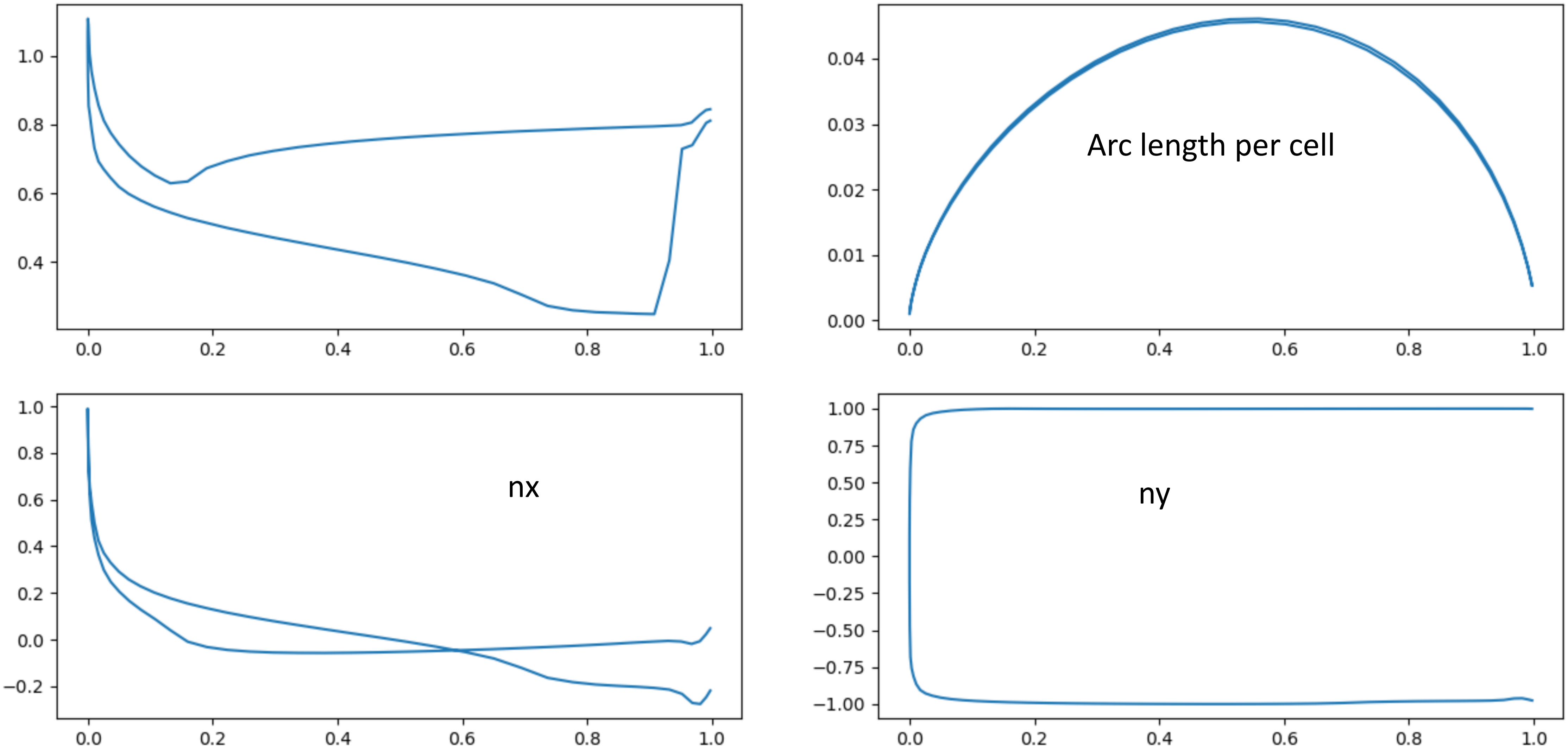
readnpz_and_mesh--Torch.py

global_variables.py

- readnpz_and_mesh--Torch.py

```
(base) liwei@PC05:~/Codes/coord-trans-encoding_inv/BASIC_simulations_inv$ python readnpz_and_mesh--Torch.py
Enter the file name: ah63k127_80_125_1000.npz
ah63k127
0.8 1.25 1.0
Successfully read grid file train_mesh_point/ah63k127.p3d
torch.Size([1, 1, 128, 129])
jst-2 tensor(0.0053, device='cuda:0')
jfn-1 tensor(0.0053, device='cuda:0')
wake_lower_jfn - 1 tensor(3.5774, device='cuda:0')
wake_upper_jfn - 2 tensor(3.5774, device='cuda:0')
jst-1 tensor(0.0053, device='cuda:0')
jfn-2 tensor(0.0053, device='cuda:0')
jst-1 1.0 0.0
jfn-1 1.0 0.0
(128, 128) (128, 128)
torch.Size([128, 128])
tensor([0.9476], device='cuda:0') tensor([0.1344], device='cuda:0') tensor([2.0387], device='cuda:0')
```

Cl, Cd from the ref. CFD: 0.9476330E+00 0.1343771E+00



1.0
0.8
0.6
0.4
0.0
0.2
0.4
0.6
0.8
1.0
0.04
0.03
0.02
0.01
0.00
Arc length per cell
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
nx
0.0
0.2
0.4
0.6
0.8
1.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
ny
0.0
0.2
0.4
0.6
0.8
1.0

tfi_torch.py

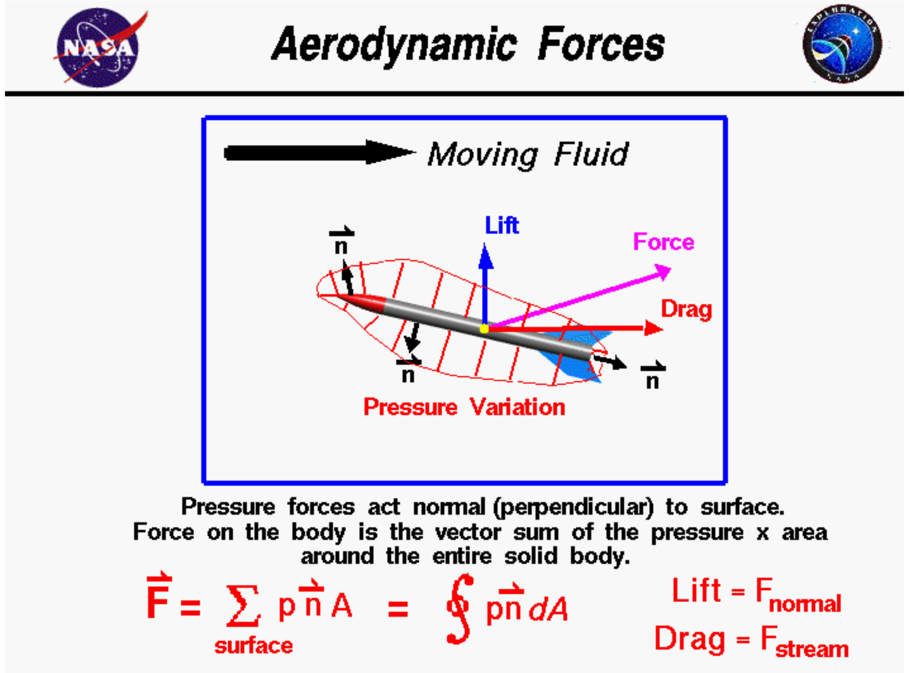
readnpz_and_mesh--Torch.py

global_variables.py

```python
#imax, jmax, kmax, xc, yc, threed = read_grid("train_mesh/"+basename+".p3d")

_, _, _, xp, yp, threed = read_grid("train_mesh_point/"+basename+".p3d")
xc, yc =  convert_center(xp, yp)


x_current, y_current = torch.from_numpy(xp).cuda(), torch.from_numpy(yp).cuda()
x_current = x_current.view(batch_size,1,idim,jdim).type(torch.cuda.DoubleTensor)
y_current = y_current.view(batch_size,1,idim,jdim).type(torch.cuda.DoubleTensor)

# now calculate the coordinate transformation metrics according to the paper Comput and Fluids
# ideally this should be done with "cell-point-type" grid files.
#dx_i, dy_i, ds_i, dx_j, dy_j, ds_j, area = calcMetrics(xp, yp)
dx_i, dy_i, ds_i, dx_j, dy_j, ds_j = calculateMetrics(x_current, y_current)
#pressure_center = calculateCellCenter(dynamics[:,istep,3], x_current, y_current)

# dx_i: the x-component of the unit normal-wise vector of i-face
# dy_i: the y-component of the unit normal-wise vector of i-face
# ds_i: the arc length of the cell along-i (circumferential)
# note: the face norm vec of i-face points into the airfoil (into the object)

# dx_j: the x-component of the unit normal-wise vector of j-face
# dy_j: the y-component of the unit normal-wise vector of j-face
# ds_j: the arc length of the cell along-j (normal)
# note: the face norm vec of j-face points clock-wise direction


x, y = xc, yc

# data[0] - xmach # in our current case,
# data[1] - aoa   # these three numbers are the
# data[2] - re    # same in all the samples.
# data[3] - rho ... density
# data[4] - rho*u ... density times X-velocity
# data[5] - rho*v ... density times Y-velocity
# data[6] - rho*E ... density times total energy (it is complex, but below you will see how I calculate pressure)

print(ds_i.shape)
```

```python
def calculateMetrics(x, y):
    '''
    Input x, y
    Outputs:
    mesh edge-i unit normal vector x-component: dx_i
    mesh edge-i unit normal vector y-component: dy_i
    mesh edge-i arc length:                     ds_i
    mesh edge-j unit normal vector x-component: dx_j
    mesh edge-j unit normal vector y-component: dy_j
    mesh edge-j arc length:                     ds_j
    '''
    #kernel_dx = torch.Tensor( [[-1,  0,  1],
    #                          [-2,  0,  2],
    #                          [-1,  0,  1]] ) * (1/8)
    #kernel_dy = torch.Tensor( [[-1, -2, -1],
    #                          [ 0,  0,  0],
    #                          [ 1,  2,  1]] ) * (1/8)
    bs = x.shape[0]
    kernel_i = torch.Tensor( [[-1.0], [1.0]]).cuda().type(torch.cuda.DoubleTensor)  # 2x1
    kernel_j = torch.Tensor( [[-1.0, 1.0]]  ).cuda().type(torch.cuda.DoubleTensor)    # 1x2
    kernel_i  = kernel_i.view((1,1,2,1))
    kernel_j  = kernel_j.view((1,1,1,2))
    # the first dim: # of input channels
    # the second dim: # of output channels
    # the thrid and fourth dims: kernel size
    #print(kernel_i, kernel_j)

    # calculate tangential direction

    ti1 = F.conv2d(x, kernel_i, padding=0)  # dx/di (or dx/dxi)
    ti2 = F.conv2d(y, kernel_i, padding=0)  # dy/di (or dy/dxi)
    #print(ti1.shape, ti2.shape)
    #print(ti1, ti2)
    ds_i = ((ti1)**2 + (ti2)**2)**0.5
    ## normalize
    #dx_i = torch.div(ti1, ds_i)
    #dy_i = torch.div(ti2, ds_i)
    # rotate -90deg to get normal, i.e. *(0 - 1*j)
    #dx_i = torch.div(-ti2, ds_i)
    #dy_i = torch.div( ti1, ds_i)
    dx_i =  ti2/ds_i
    dy_i = -ti1/ds_i

    #print(dx_i.shape, dy_i.shape)

    # calculate tangential direction
    tj1 = F.conv2d(x, kernel_j, padding=0)
    tj2 = F.conv2d(y, kernel_j, padding=0)
    ds_j = ((tj1)**2 + (tj2)**2)**0.5
    ## normalize
    #dx_j = torch.div(tj1, ds_j)
    #dy_j = torch.div(tj2, ds_j)
    # rotate -90deg to get normal, i.e. *(0 - 1*j)
    #dx_j = torch.div( tj2, ds_j)
    #dy_j = torch.div(-tj1, ds_j)
    dx_j =  tj2/ds_j
    dy_j = -tj1/ds_j
```

How to calculate Force-X and Force-Y



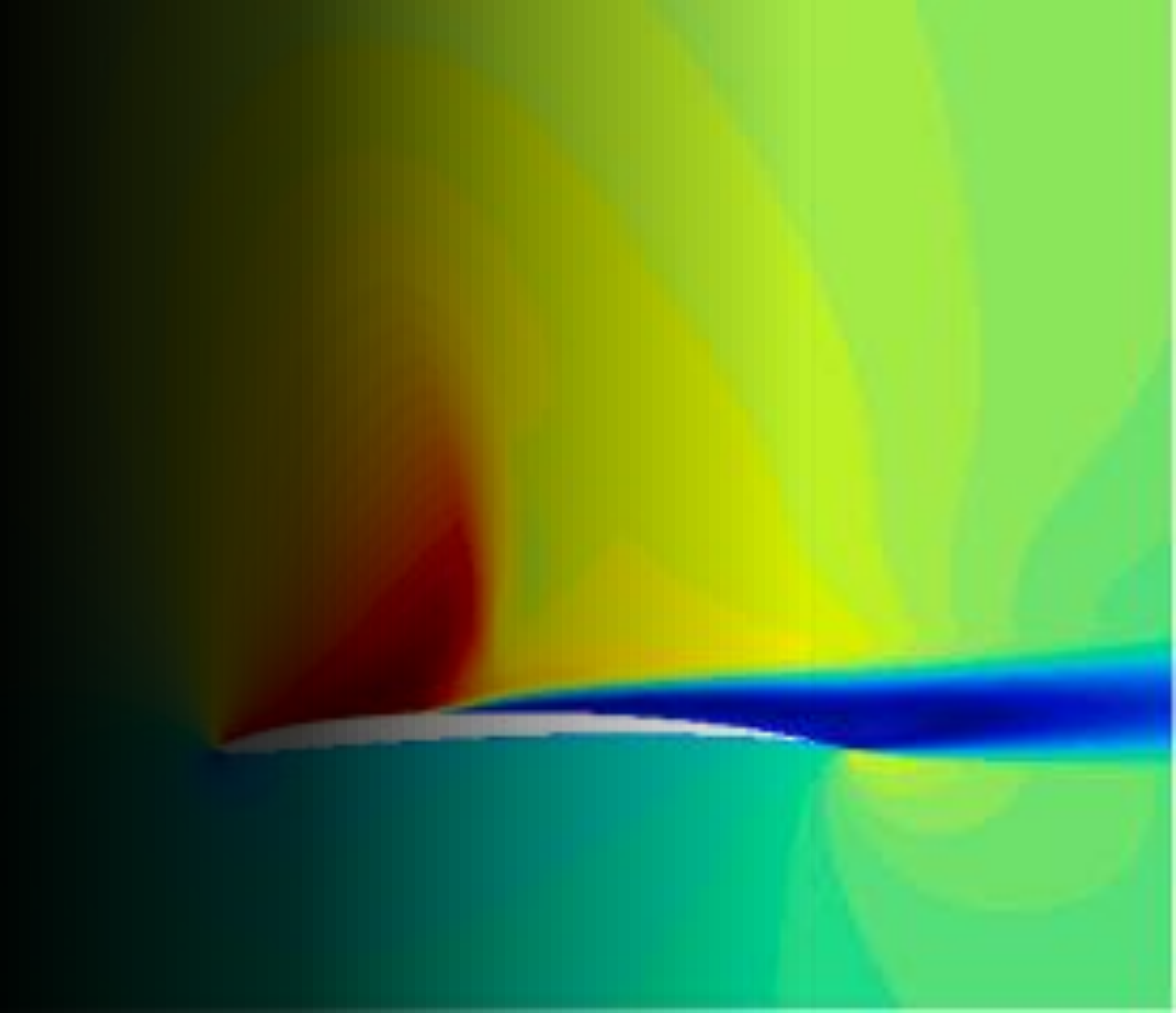

```
angle_of_attack = 1.25 #degree
angle_of_attack = angle_of_attack / 180. * np.pi
cosDir = np.cos(angle_of_attack)
sinDir = np.sin(angle_of_attack)


pressure_center = torch.from_numpy(p).cuda().type(torch.cuda.DoubleTensor)
#pressure_farfield_boundary = torch.from_numpy(p[:,-1]).cuda().type(torch.cuda.DoubleTensor)
#pressure_farfield_boundary = pressure_farfield_boundary.view(imax,1)
#print(pressure_farfield_boundary.shape)
#pressure_center = torch.cat((pressure_center, pressure_farfield_boundary), dim=1) # just make sure it has the same shape as ds and dx
print(pressure_center.shape)

xc_cuda = torch.from_numpy(xc).cuda().type(torch.cuda.DoubleTensor)
#yc_cuda = torch.from_numpy(yc).cuda().type(torch.cuda.DoubleTensor)
pressure_surface = torch.masked_select( pressure_center, cellcenterMask ).view(1,-1)
xc_surface       = torch.masked_select( xc_cuda,         cellcenterMask ).view(1,-1)
arcLength_surface = torch.masked_select( ds_i,          segmentMask ).view(1,-1)
nx_surface = torch.masked_select( dx_i,          segmentMask ).view(1,-1)
ny_surface = torch.masked_select( dy_i,          segmentMask ).view(1,-1)

#print("inner_x:", inner_x.shape) # inner_x: torch.Size([5, 1, 256, 129])
#print("segmentMask:", segmentMask.shape) # segmentMask: torch.Size([5, 256, 129])
#print("mode_1a:",mode_1a.shape)
#print("test mask:",torch.masked_select(inner_x,segmentMask).view(batch_size,-1).shape)
#f_pred_1 =     torch.sum(  torch.masked_select(inner_x,segmentMask).view(batch_size,-1), dim=1 ) #*mode_1a
#f_pred_2 =     torch.sum(  torch.masked_select(inner_y,segmentMask).view(batch_size,-1), dim=1 ) #*mode_2a
#print(f_pred_1, f_pred_2)

#plt.figure()
fig, axs = plt.subplots(2, 2)
axs[0,0].plot(xc_surface[0].cpu().detach().numpy(),  pressure_surface[0].cpu().detach().numpy())
axs[0,1].plot(xc_surface[0].cpu().detach().numpy(), arcLength_surface[0].cpu().detach().numpy())
axs[1,0].plot(xc_surface[0].cpu().detach().numpy(), nx_surface[0].cpu().detach().numpy())
axs[1,1].plot(xc_surface[0].cpu().detach().numpy(), ny_surface[0].cpu().detach().numpy())
plt.show()



dfx = (1.4*pressure_surface-1)*arcLength_surface*nx_surface/(0.5*0.8**2*1.4)
dfy = (1.4*pressure_surface-1)*arcLength_surface*ny_surface/(0.5*0.8**2*1.4)

fx =     torch.sum( dfx, dim=1 )
fy =     torch.sum( dfy, dim=1 )
# for sanity check, calculate the wet surface, theoretically it should 2.0
wet_surface =     torch.sum( arcLength_surface, dim=1 )

Cd = fx*cosDir + fy*sinDir
Cl =-fx*sinDir + fy*cosDir


print(Cl, Cd, wet_surface)
```

Project fx, fy onto the fluid-moving direction, so we need angle of attack.

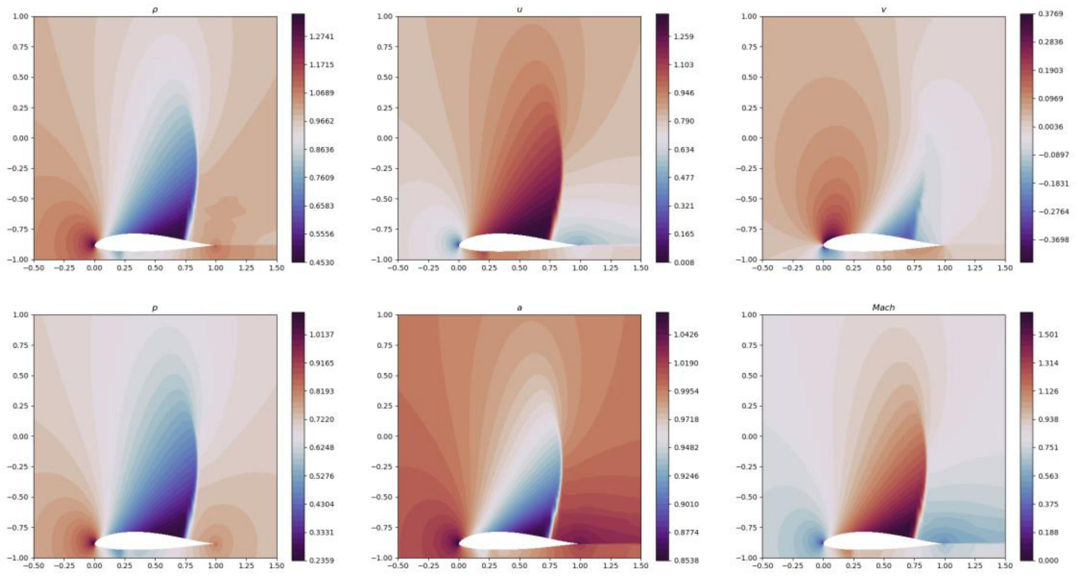
In Progress: Surrogate Model

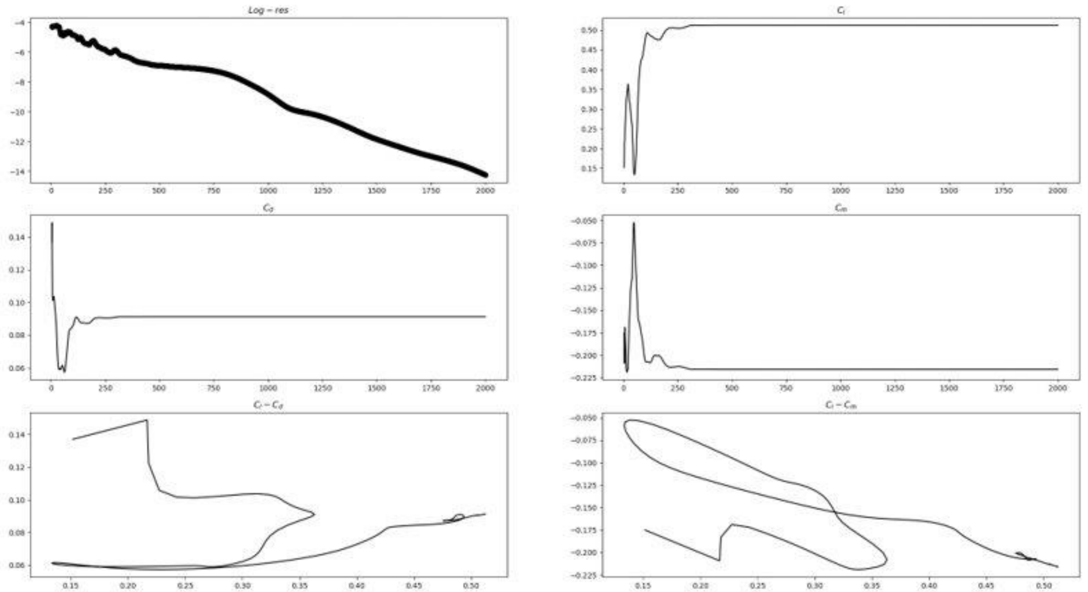
- https://github.com/tum-pbs/coord-trans-encoding

- Now we directly calculate metrics information by python script. No need Fortran solver anymore.



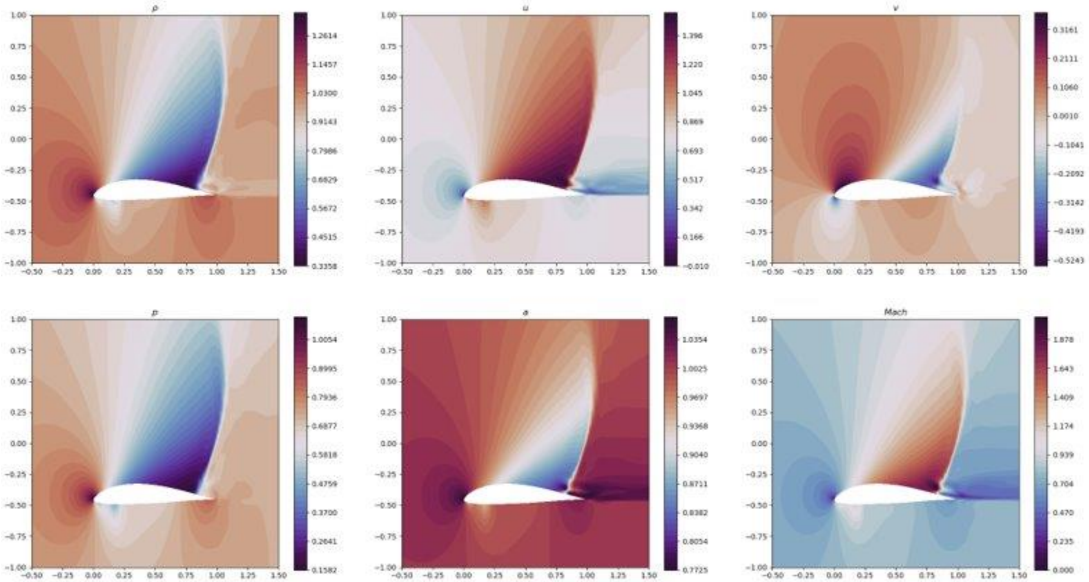
0.283, 0.087, -0.062





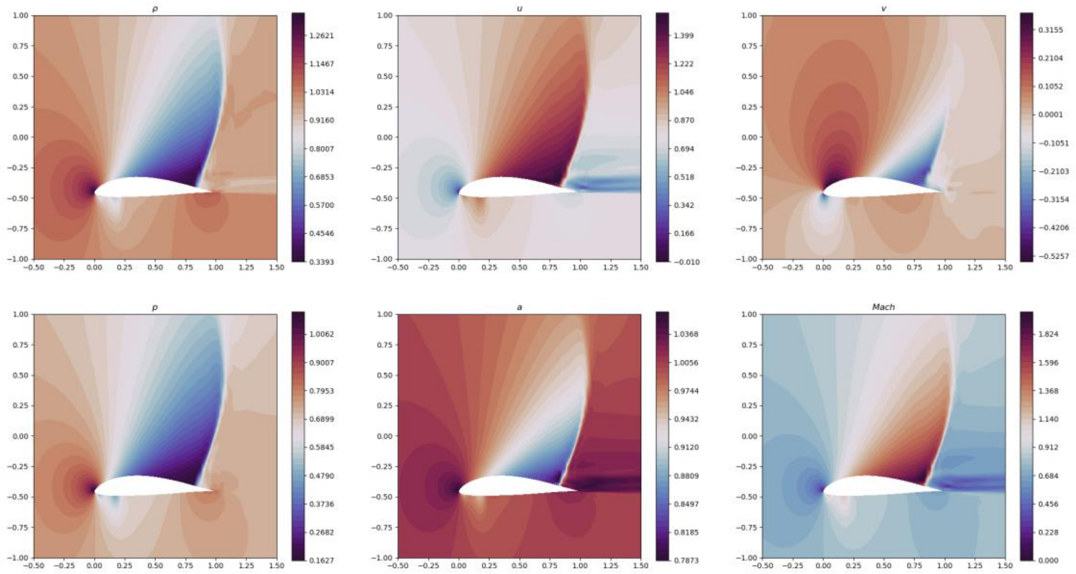
Cl =0.5123548E+00
Cd=0.9131187E-01
Cm=-0.2159179E+00

0.639, 0.043, -0.127



Cl = 0.8308828E+00
Cd = 0.1734693E+00
Cm = -0.3984813E+00

0.692, 0.162, -0.283



Cl = 0.8255088E+00
Cd = 0.1721221E+00
Cm = -0.3907791E+00

256 points

512 points

768 points

1024 points