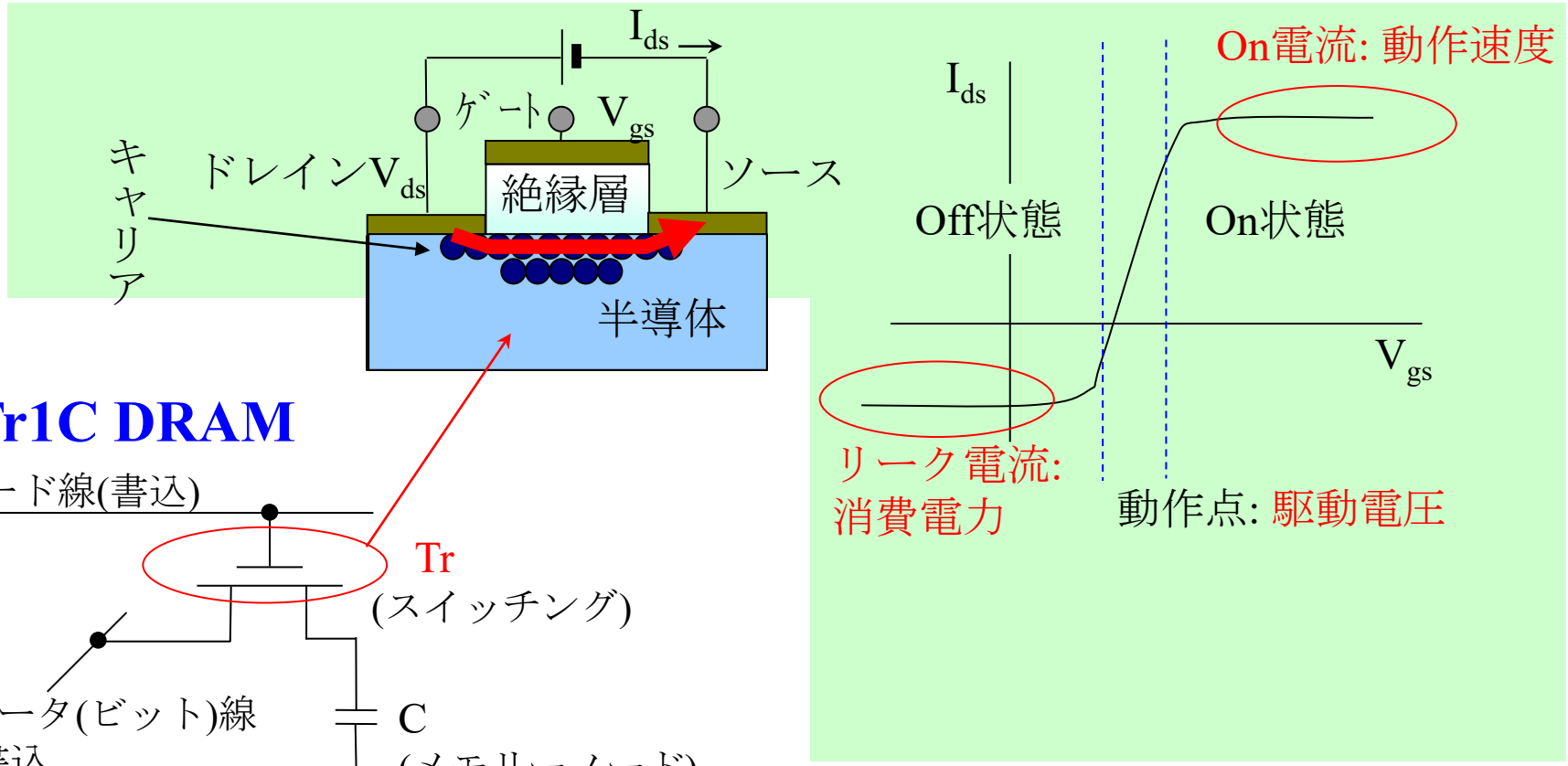
問題

1. 厚さ 100 nmの a-SiO_2 の単位面積当たり静電容量 C_{OX} を求めよ。
   a-SiO_2 の 比誘電率は $\varepsilon_r = 11.9$ とする。

2. TransferCurve.csvのデータから、飽和移動度を求めよ
   電極幅 W = 300 μm, L = 50 μm とする。
   飽和移動度を求める際の Vg、Vd は各自で選ぶこと。
   その値を選んだ理由も説明せよ。

電界効果トランジスタ(FET)の基本動作

トランジスタの基本機能
1. 増幅機能　　ゲート電圧に電流が比例する領域を利用
2. スイッチ機能　ゲート電圧による大きな電流の変調を利用



FET特性の解析: 飽和領域

$$I_{DS} = \frac{W}{L}\mu C_{OX}\left[(V_{GS} - V_{th})V_{DS} - \frac{{V_{DS}}^2}{2}\right]$$

$V_{DS} > V_p = V_{GS} - V_{th}$

$$I_{DS} = \frac{W}{2L}\mu C_{OX}(V_{GS} - V_{th})^2$$

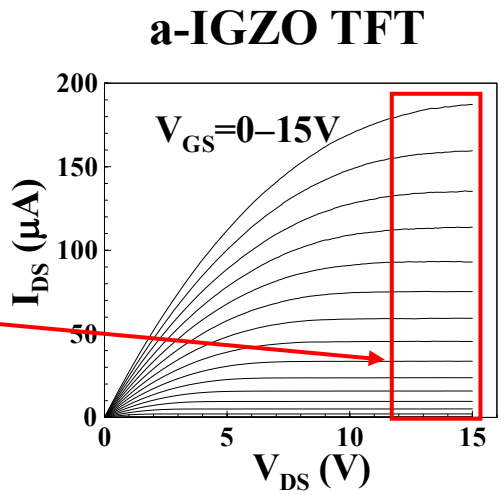
$$I_{DS}{}^{1/2} = \sqrt{\frac{W}{2L}\mu C_{OX}}(V_{GS} - V_{th})$$

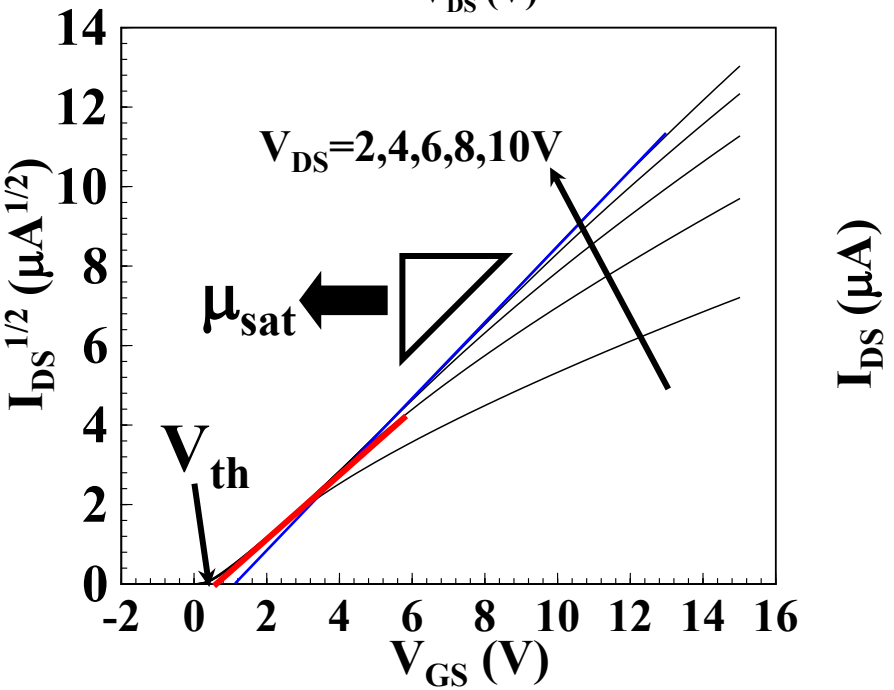
$I_{DS}{}^{1/2}$ vs. V_{GS}をプロット

V_{GS}軸切片: V_{th}

傾き: 飽和移動度

Saturation mobility, μ_{sat}





ゲート容量 C_{OX}の計算

ゲート絶縁体: アモルファス SiO_2
誘電率: $11.9\varepsilon_0$　　($\varepsilon_0 = 8.854418782\text{e-}12\ C^2N^{-1}m^{-2}$)

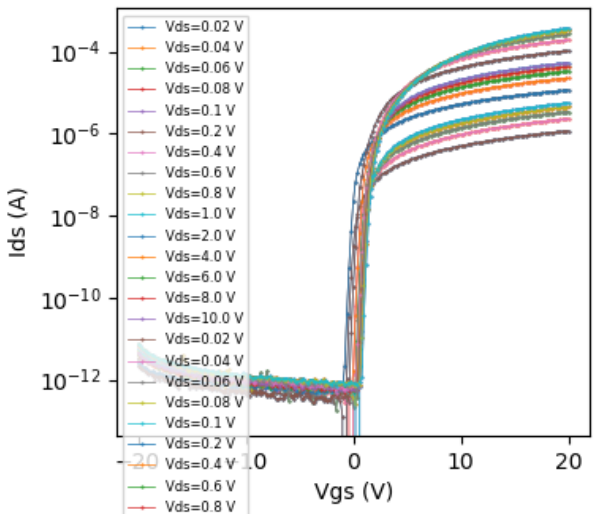
厚さ　100 nm

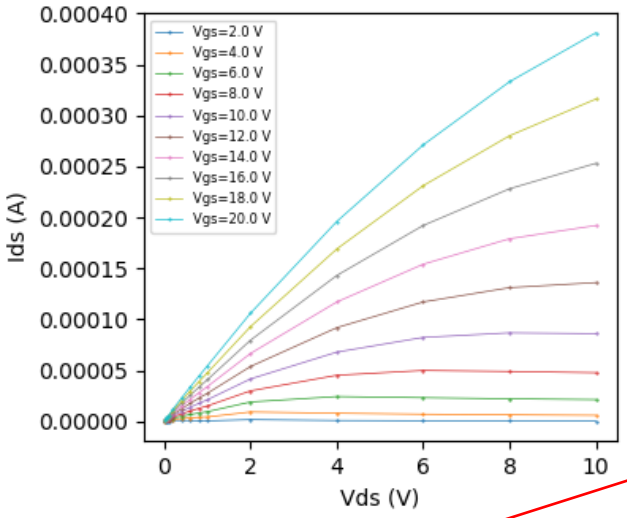
1 m^2当たり静電容量

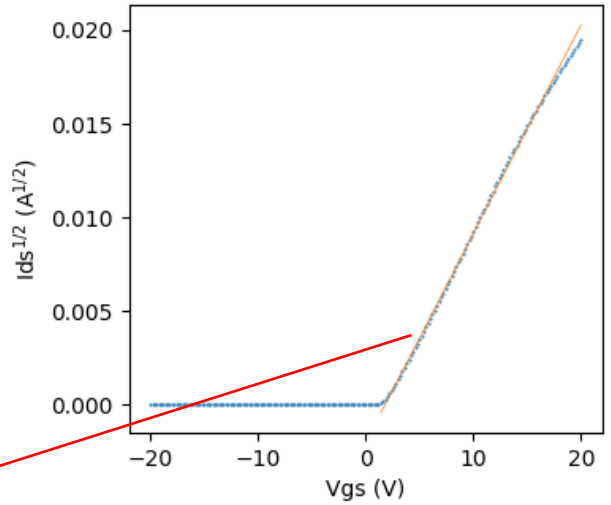
$$C_{OX} = 11.9\varepsilon_0 / 100\text{e-}9\ [m] = 0.00105\ F/m^2$$

プログラム

TFTiv.py







直線に見える 1.9 ~ 10.0 V で、numpy.polyfit() で一次多項式にフィッティング

ai = np.polyfit(xfit, yfit, 1)

y = ai[1] + ai[0]x

Vth = -ai[1] / ai[0] = 1.794 V

$\frac{dIg^{\frac{1}{2}}}{dVgs}$ = ai[0] = 0.001114 $A^{1/2}/V$

$$I_{DS}{}^{1/2} = \sqrt{\frac{W}{2L}\mu C_{OX}}(V_{GS} - V_{th})$$

$$\boldsymbol{\mu_{sat} = \frac{(dIgs^{1/2}/dVgs)^2}{\frac{W}{2L}C_{ox}} = 0.000392\ m^2/Vs = 3.92\ cm^2/Vs}$$

プログラム (抜粋)

TFTiv.py

```
import re      # 正規表現モジュールを読み込む

#==================================
# parameters
#==================================
infile = 'TransferCurve.csv'

dg  = 100e-9 #m
erg = 11.9

W = 300.0e-6 #m
L = 50.0e-6

# Ids^(1/2)-VgsプロットをするVds
Vds0 = 10.0

# 余計な文字が含まれている文字列から、
# 浮動小数点に変換できる最初の文字列を切り出し、
# 浮動小数点に変換して返す
def pfloat(str, defval = None):
# 文字列から、浮動小数点に使える文字が連続している部分を切り出す
    m = re.search(r'([+¥-eE¥d¥.]+)', str)
# 一致した文字列を取得
    valstr = m.group()
    try:
        return float(valstr)
    except:
        return defval
```

```
def read_csv(fname):
    print("")
    with open(fname) as f:
        fin = csv.reader(f)

        labels = next(fin)
        xlabel = labels[0]

# label行が 空文字 の場合、データとしては読み込まない
        ylabels = []
        for i in range(1, len(labels)):
            if labels[i] == '':
                break
            ylabels.append(labels[i])
        ny = len(ylabels)

        x     = []
        ylist = []
        for i in range(ny):
            ylist.append([])

        for row in fin:
            x.append(pfloat(row[0]))
            for i in range(1, ny+1):
                v = pfloat(row[i])
                if v is not None:
                    ylist[i-1].append(v)
                else:
                    ylist[i-1].append(None)

    return xlabel, ylabels, x, ylist
```

プログラム (抜粋)

TFTiv.py

```
def main():
    Cox = erg * e0 / dg  #(F/m^2)
    print("")
    print("Cox = {:12.6g} [F/m^2]".format(Cox))

    xlabel, ylabels, Vgs, IdsVgs = read_csv(infile)
    nVgs = len(IdsVgs[0])
    nVds = len(ylabels)
    Vds = []
    for i in range(nVds):
        Vds.append(pfloat(ylabels[i]))
    print("")
    print("nVds=", nVds)
    print("nVgs=", nVgs)
    print("xlabel : ", xlabel)
    print("ylabels: ", ylabels)
    print("Vds: ", Vds)

# 出力特性 Ids - Vds をプロットするためのデータを作る
    IdsVds = np.empty([nVgs, nVds])
    for ig in range(nVgs):
        for id in range(nVds):
            IdsVds[ig][id] = IdsVgs[id][ig]

# sqrt(Ids) - Vgsプロットをする Vds0 のデータ番号 iVds を探す
# Vds[i] が小さい方から順に、Vds0 <= Vds[i] となる i を探すが、
# 浮動小数点誤差があることを考慮し、Vds0 - 1.0e-3 <= Vds[i] とする
    for i in range(nVds):
        if Vds0 - 1.0e-3 <= Vds[i]:
            iVds = i
            break

# sqrt(Ids) データを作る
    print("")
    print("Vds used: {} V (iVds = {})".format(Vds[iVds], iVds))
    sqrtIds = []
    for ig in range(nVgs):
        sqrtIds.append(sqrt(IdsVgs[iVds][ig]))

# 最小二乗法のデータ
    xfit = []
    yfit = []
    for i in range(nVgs):
        if xfitmin <= Vgs[i] <= xfitmax:
            xfit.append(Vgs[i])
            yfit.append(sqrtIds[i])
    print("")
    print("Least squares fitting:")
    print("Vgs range: {} - {} V".format(xfitmin, xfitmax))
    print("Vgs=", xfit)
    print("Igs^(1/2)=", yfit)
    ai = np.polyfit(xfit, yfit, 1)
# y = ai[1] + ai[0]x
    Vth  = -ai[1] / ai[0]
    grad = ai[0]
    mu = grad * grad / (W * Cox / 2.0 / L)
    print("Vth = {:6.4g} V".format(Vth))
    print("dIgs^1/2/dVgs = {:12.4g} A^(1/2)/V".format(grad))
    print("mu_sat = {} m^2/Vs = {} cm^2/Vs".format(mu, 1.0e4 * mu))
```

プログラム (抜粋)

TFTiv.py

```
# 最小二乗の結果を確認する直線
# フィッティング範囲より広くプロットする
    xcal = []
    ycal = []
    xx = xfitmin - 0.5
    xcal.append(xx)
    ycal.append(ai[1] + ai[0] * xx)
    xx = max(Vgs)
    xcal.append(xx)
    ycal.append(ai[1] + ai[0] * xx)

# 伝達特性  Ids-Vgs のグラフ
    for id in range(nVds):
        ax1.plot(Vgs, IdsVgs[id],  linewidth = 0.5, marker = 'o',
markersize = 0.5,
                 label = 'Vds={} V'.format(Vds[id]))
    for id in range(nVds):
        ax1.plot(Vgs, IdsVgs[id],  linewidth = 0.5, marker = 'o',
markersize = 0.5,
                 label = 'Vds={} V'.format(Vds[id]))
    ax1.set_xlabel('Vgs (V)')
    ax1.set_ylabel("Ids (A)")
    ax1.set_yscale('log')
# 凡例のアンカーを左上に設定
     ax1.legend(loc = 'upper left', fontsize = legend_fontsize)
```

```
# 出力特性  Ids-Vds のグラフ
# 20点の Vgs を選び、そのうち、線形でIdsが見えるデータのみ表示する
    nskip = int(nVgs / 20.0 + 1.0e-6)
    for ig in range(0, nVgs, nskip):
# 線形プロットで見えないほどIdsが小さいデータは表示しない
        if max(IdsVds[ig]) < 1.0e-7:
            continue
        ax2.plot(Vds, IdsVds[ig],  linewidth = 0.5, marker = 'o', markersize
= 0.5,
                 label = 'Vgs={} V'.format(Vgs[ig]))
    ax2.set_xlabel('Vds (V)')
    ax2.set_ylabel("Ids (A)")
    ax2.legend(loc = 'upper left', fontsize = legend_fontsize)

# Ids^(1/2) - Vgs グラフ
    ax3.plot(Vgs, sqrtIds,  linestyle = 'none', marker = 'o', markersize =
0.5)
    ax3.plot(xcal, ycal,   linestyle = '-', linewidth = 0.5)
    ax3.set_xlabel('Vgs (V)')
    ax3.set_ylabel("Ids$^{1/2}$ (A$^{1/2}$)")

    plt.tight_layout()

    plt.pause(0.1)
    print("Press ENTER to exit>>", end = '')
    input()
```